Міністерство освіти і науки України

Донбаська державна машинобудівна академія (ДДМА)

ТЕОРІЯ АЛГОРИТМІВ І ГРАФІВ

Навчальний посібник

для здобувачів вищої освіти спеціальності 122 «Комп'ютерні науки»

Затверджено
на засіданні
вченої ради ДДМА
Протокол № 11 від 30.06.2022

Краматорськ
ДДМА
2022

УДК 004.421+519.171(075.8)
Т33

Рецензенти:

Вовна О. В., доктор технічних наук, професор, академік Академії Метрології України, завідувач кафедри електронної техніки.

Разживін О. В., канд. техн. наук, доцент, доцент кафедри автоматизації виробничих процесів Донбаської державної машинобудівної академії.

Малигіна, С. В.

Т33 Теорія алгоритмів та графів : навчальний посібник для здобувачів вищої освіти спеціальності 122 «Комп'ютерні науки» / С. В. Малигіна, І. А. Гетьман, О. В. Бережна, М. А. Держевецька. – Електрон. дані. – Краматорськ : ДДМА, 2022. – 1 електрон. опт. диск (CD-ROM); 12 см. – Назва з тит. екрана.

ISBN 978-617-7889-27-3

Посібник містить такі розділи теорії алгоритмів: класична теорія алгоритмів (машина Поста, машина Тьюринга, алгоритмічно нерозв'язні задачі), асимптотичний аналіз складності алгоритмів, складні класи та практичний порівняльний аналіз обчислювальних алгоритмів, основи теорії графів. Наведено необхідні теоретичні відомості, приклади розв'язання типових задач і задачі для самостійного розв’язання.

УДК 004.421+519.171(075.8)

ISBN 978-617-7889-27-3

ЗМІСТ

ВСТУП

Теорія алгоритмів – це наука, що вивчає загальні властивості і закономірності алгоритмів, різноманітні формальні моделі їх подання. На основі формалізації поняття алгоритму можливо порівняння алгоритмів по їх ефективності, перевірка їх еквівалентності, визначення областей використання.

Розроблені в 1930-х роках різноманітні формальні моделі алгоритмів (Пост, Тьюринг, Черч), так само як і запропоновані в 1950-х роках моделі Колмогорова і Маркова, виявилися еквівалентними в тому сенсі, що будь-який клас проблем, вирішуваних в одній моделі, можна розв'язати і в інший.

Доки математика мала справу в основному з числами й обчисленнями, а поняття алгоритму ототожнювалося з поняттям методу обчислення, необхідність у вивченні самого поняття алгоритму не виникало. Вважалося, якщо для розв’язання деякого класу задач пропонувався конкретний алгоритм, то вчені доходили згоди вважати його дійсно шуканим алгоритмом. І тільки доведення того, що не існує алгоритму розв’язання певного класу, яке є висловлюванням про всі можливі алгоритми, потребує уточнення цього важливого поняття. Саме уточнення поняття “алгоритм” і виконує така наукова дисципліна як теорія алгоритмів. Отже, теорія алгоритмів – наукова дисципліна, що вивчає поняття про алгоритм на основі алгоритмічних моделей, їх функціонування та взаємозв’язку.

Теорія алгоритмів використовуються в теорії релейноконтактних схем, в теорії автоматів, у лінгвістиці, в економічних дослідженнях, у фізіології мозку і психології.

Теорія алгоритмів дуже важлива для професійної підготовки фахівців комп’ютерних наук, адже навчальна дисципліна сприяє розвитку алгоритмічного мислення.

1 ВСТУП ДО ТЕОРІЇ АЛГОРИТМІВ. ОСНОВНІ ПОЛОЖЕННЯ Й ОЗНАЧЕННЯ ТЕОРІЇ АЛГОРИТМІВ

1.1 Історичний огляд

«***Алгоритм***» походить від латинської транслітерації «alchorismi»або «algorismi» арабського імені середньоазіатського вченого ал-Хорезмі Мухаммеда бен Муси (780 – 847).

Першим що дійшли до нас алгоритмом в його інтуїтивному розумінні – кінцевої послідовності елементарних дій, які вирішують поставлене завдання, вважається запропонований Евклідом в III столітті до нашої ери алгоритм знаходження найбільшого спільного дільника двох чисел (алгоритм Евкліда).

Відзначимо, що протягом тривалого часу, аж до початку XX століття саме слово «алгоритм» вживалося в стійкому поєднанні «алгоритм Евкліда». Для опису покрокового вирішення інших математичних задач використовувалося слово «метод».

Початковою точкою відліку сучасної теорії алгоритмів можна вважати роботу німецького математика Курта Геделя (1931 рік – теорема про НЕповноту символічних логік), в якій було показано, що деякі математичні проблеми не можуть бути вирішені алгоритмами з деякого класу. Спільність результату Геделя пов'язана з тим, чи збігається використаний ним клас алгоритмів з класом всіх (в інтуїтивному сенсі) алгоритмів. Ця робота дала поштовх до пошуку та аналізу різних формалізацій визначення поняття алгоритму.

Перші фундаментальні роботи по теорії алгоритмів були опубліковані незалежно в 1936 році роки Аланом Тьюрингом, Алоізом Черчем і Емілем Постом. Запропоновані ними машина Тьюринга, машина Посту і лямбда-обчислення Черча були еквівалентними формалізмами алгоритму. Сформульовані ними тези (Посту і Черча-Тьюринга) постулювали еквівалентність запропонованих ними формальних систем і інтуїтивного поняття алгоритму. Важливим розвитком цих робіт стало формулювання і доказу-будівництві алгоритмічно нерозв'язних проблем.

У 1950-ті роки суттєвий внесок у теорію алгоритмів внесли роботи Колмогорова і Маркова.

1. 2 Формалізація поняття алгоритму

Поняття алгоритму є одним з основних понять сучасної математики. Ще на початкових етапах її розвитку (Стародавній Єгипет, Вавилон, Греція) у математиці стали формуватися різноманітні обчислювальні процеси. За їх допомогою при розв'язуванні задач шукані величини обчислювалися послідовно з висхідних величин за певними правилами та інструкціями. З часом такі правила (інструкції) одержали назву алгоритмів.

Термін алгоритм походить з імені середньовічного вченого-математика Аль-Хорезмі, котрий ще в IX ст. (825 р.) дав правила виконання чотирьох операцій арифметичних дій в десятковій системі числення. Перелік виконуваних арифметичних дій був названий алгоризм.

З 1747 р. замість слова алгоризм стали вживати алгорисмус, смисл якого полягав в комбінуванні чотирьох операцій арифметичного обчислення – додавання, віднімання, множення та ділення.

Через століття для створеного Бебіджем комп'ютера, Адою Лавлейс було написано першу програму, що є першим технічним втіленням алгоритму.

До 1950 р. алгорисмус став алгорифмом. Смисл алгорифму найчастіше пов'язувався з алгорифмами Евкліда – процесами знаходження найменшої спільної міри (двох відрізків, найменшого спільного дільника двох многочленів тощо). Аж до 30-х років XX століття поняття алгоритму мало методологічне значення. Його означували евристично.

Означення 1. Скінченна сукупність коректно сформульованих правил розв'язування задачі з класу (множини) задач називається алгоритмом.

Таке означення алгоритму не є строгим, математичним, оскільки у ньому не означено понять клас задач, правило (їх розв'язування). Протягом тривалого часу математики задовольнялися цим означенням, теорії алгоритмів не існувало.

Становище суттєво змінилося, коли технічні засоби математичних обчислень стали настільки потужними та зручними у практичному користуванні, що розв'язок задачі в аналітичному вигляді (у вигляді формули) виявився не потрібним.

Багато теоретичних і практичних завдань вимагають вказати ***алгоритм – такий набір інструкцій, виконуючи які, ми за кінцеве число кроків вирішимо поставлену задачу.***

У 20-х роках XX-го століття проблема означення алгоритму стала однією з основних, фундаментальних математичних проблем. Її вирішення було одержано у двох формах всередині 30-х років 20-го століття у роботах таких відо-

мих математиків як Гільберт, Гедель, Черч, Кліні, Пост, Т'юрінґ (рис. 1.1) на базі означень понять класів:

а) арифметичних функцій (які назвали рекурсивними);

б) процесу.

Рисунок 1.1 – Засновники теорії алгоритмів

Означення 2. Скінченна сукупність операцій, якій притаманні властивості.

а**) дискретності** – виконання чергової операції починається у певний момент часу, моменти часу не обов'язково еквідистантні;

б) **детермінованості** – перед виконанням першої операції відомі початкові дані, у будь-який момент часу точно відомі наступні операції (прогнозованість) та результат попередньої операції;

в) **впорядкованості** – встановлена черговість операції (інакше – відношення порядку на їх множині);

г) **масовості** – алгоритм розв'язує задачу, що належить до класу задач;

д**) елементарності** – завжди можна вказати елементарну (виконувану апаратно, не програмовану) операцію – називатимемо алгоритмом.

Таким чином, алгоритм виникає при вирішенні проблеми (інформування, управління). Для цього будується математична модель проблеми і на її базі ставиться задача(і) та будуються методи її розв'язування.

Означення А. Н. Колмогорова. Алгоритм – це будь-яка система обчислень, що виконуються за строго певними правилами, яка після деякої кількості кроків завідомо приводить до розв'язання поставленої задачі.

Означення А. А. Маркова. Алгоритм – це точне розпорядження, що визначає обчислювальний процес, який йде від варійованих початкових даних до шуканого результату.

1.3 Підходи до визначення алгоритму

1. ***Алгоритм як обчислювальна функція*** – тобто алгоритм завжди обчислює деяку функцію. Необхідно знати, який набір операцій обчислює задачу

2. ***Алгоритмічні моделі на основі детермінованих пристроїв – абстрактні машини Поста і Тьюринга (1936)***. Дані алгоритмічні моделі суттєво вплинули на розуміння логічної природи майбутніх ЕОМ. З точки зору англійського математика А. Тьюринга алгоритмічний процес – це робота деякої уявної обчислювальної машини **– *машини Тьюринга***. Він запропонував ототожнити інтуїтивне поняття «Алгоритм» з точним поняттям «Машина Тьюринга».

Формалізація А. Черча і С. Кліні (1930–1936): суворе визначення алгоритму, запропоноване Черчем і Кліні, засноване на понятті частково-рекурсивної функції.

Вони запропонували ототожнити інтуїтивне поняття «Алгоритм» із суворим математичним поняттям «частково рекурсивна функція».

3. ***Комбінаторні моделі –*** перетворення слів у довільному алфавіті за допомогою операцій підстановки, тобто заміни однієї частини слова на іншу. Основною теоретичною моделлю даного типу є нормальні алгоритми Маркова.

Замість інтуїтивного поняття «Алгоритм» А. А. Марков вводить суворе поняття ***«нормальний алгорифм***».

Саме на основі цих моделей були реалізовані різні напрямки в програмуванні. Так, мікропрограмування базується на ідеї машини Тьюринга, структурне програмування запозичило свої конструкції з теорії рекурсивних функцій, а мови символьного оброблення інформації (ПРОЛОГ, РЕФАЛ) беруть початок від нормальних алгоритмів Маркова.

1.4 Способи запису алгоритмів

Найбільш поширеними є такі форми подання алгоритмів:

– словесна (запис алгоритму на природній мові);

– покрокова;

– графічна (зображення алгоритму з використанням графічних символів);

– псевдокод (напівформалізований опис алгоритмів, що включає в себе як елементи мови програмування, так і фрази природної мови, загальноприйняті математичні позначення та ін.);

– програмна (тексти на мовах програмування).

Словесне представлення алгоритмів

Приклад 1.1

Словесний опис алгоритму знаходження всіх простих чисел, які не перебільшують заданого числа N (решето Ератосфена (рис. 1.2)).

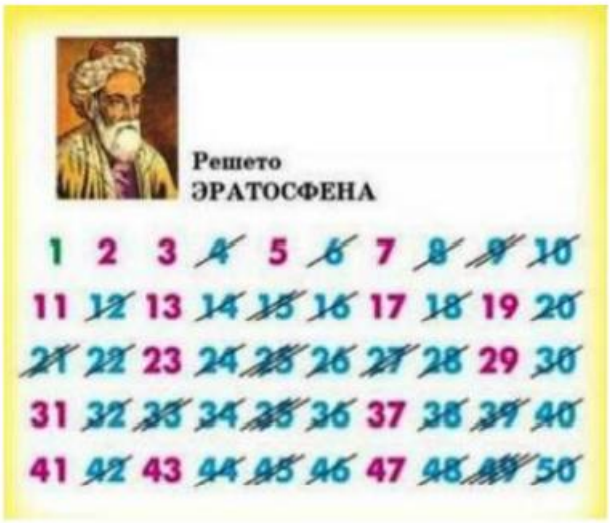


Рисунок 1.2 – Решето Ератосфена

1. Виписати підряд всі цілі числа від 2 до N.

2. Нехай змінна P початково дорівнює 2 (першому простому числу).

3. Викреслити в списку числа від 2P до N (тобто кратні P: 2P, 3P, 4P ….).

4. Знайти перше незакреслене число в списку, більше за P і присвоїти значенню змінної P це число.

5. Повторювати кроки 3 і 4 поки це можливо.

Словесний спосіб не має поширення з наступних причин: – такі описи строго не формалізуються; – страждають на багатослівність записів; – припускають неоднозначність тлумачення окремих приписів.

Покрокове представлення алгоритмів

Покроковий опис алгоритму може бути представлений як послідовність кроків. На кожному кроці виконується певна дія (оператор присвоювання) або здійснюється перевірка деякої умови (умовний перехід).

Приклад 1.2.

Знайти суму S перших n елементів одновимірного числового масиву X, скориставшись формулою:

$$S = \sum_{i=1}^{n} x_i$$

Крок 1. $S = 0$;

Крок 2. $I = 1$;

Крок 3. $S = S + x_i$;

Крок 4. $I = I + 1$;

Крок 5. Якщо $I \le N$, то перейти на крок 3;

Крок 6. Кінець

Представлення алгоритмів за допомогою псевдокоду

Псевдокод – опис структури алгоритму на природній, частково формалізованій мові, який дозволяє виявити основні етапи виконання задачі, перед точним його записом на мові програмування. У псевдокоді використовуються деякі формальні конструкції і загальноприйнята математична символіка. Строгих синтаксичних правил для запису псевдокоду не існує. Це полегшує запис алгоритму при проєктуванні і дозволяє описати алгоритм, використовуючи будь-який набір команд. Однак в псевдокоді зазвичай використовуються деякі конструкції, властиві формальним мовам, що полегшує перехід від псевдокоду до запису алгоритму на мові програмування. Одна з переваг використання псевдокоду полягає в тому, що він дозволяє спочатку розробити тільки основу програми, а всі деталі розробити потім.

Крім того, псевдокод допомагає зрозуміти, як побудувати процедуру вирішення задачі на тій чи іншій мові програмування. Псевдокод – це система позначень і правил, призначена для однакового запису алгоритмів. Він займає

проміжне місце між природною і формальною мовами. З одного боку, він наближений до звичайної природної мови, тому алгоритми можуть на ньому записуватися і читатися як звичайний текст. З іншого боку, в псевдокоді використовуються деякі формальні конструкції і математична символіка, що наближає запис алгоритму до загальноприйнятого математичного запису.

Способи графічного представлення алгоритмів

Представлення алгоритмів за допомогою блок-схем

Графічні способи представлення алгоритмів є більш компактними і наочними порівняно зі словесним. При графічному поданні алгоритм зображується у вигляді послідовності пов'язаних між собою функціональних блоків, кожен з яких відповідає виконанню однієї або декількох дій. Таке графічне представлення називається схемою алгоритму або блок-схемою.

Переваги та недоліки блок-схем

Блок-схеми графічно та наочно відображають логіку програми за допомогою стандартних геометричних фігур і сполучних ліній. Вони є інтуїтивно зрозумілим методом представлення послідовності дій. Разом з тим, блок-схеми мають цілий ряд недоліків.

Блок-схеми не піддаються формалізації, тому їх не можна використовувати для безпосереднього введення в машину.

Блок-схеми не узгоджуються зі структурним програмуванням, оскільки в значній мірі орієнтовані на використання команди goto.

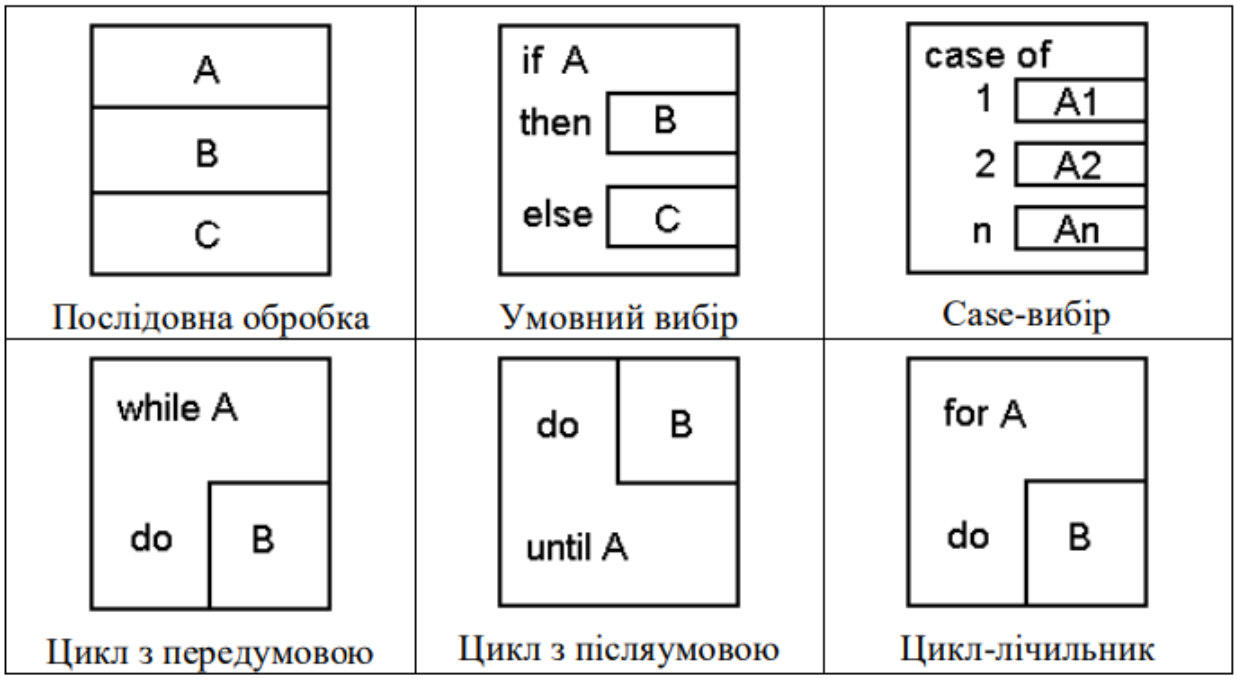
Блок-схеми приховують особливості програм, створених за правилами структурного програмування.

Flow-форми

Flow-форми і діаграми Нассі-Шнейдермана відносяться до візуальних мов специфікації обчислювальних процесів. Вони базуються на основних ідеях структурного програмування і дозволяють визначати потоки управління за допомогою спеціальних ієрархічно організованих схем.

Кожен символ FLOW-форми має вигляд прямокутника (табл.1.1) і може бути вписаний в будь-який внутрішній прямокутник будь-якого іншого символу. Символи позначаються за допомогою виразів на природній мові або з використанням математичної нотації.

Таблиця 1.1 – Основні символи Flow-форм



Структурограми Нассі–Шнейдермана

Структурограма (схема Нассі–Шнейдермана) – це схема, що ілюструє структуру передачі управління всередині модуля за допомогою вкладених один в один блоків. Спосіб подання програми за допомогою схем Нассі–Шнейдермана дозволяє зображати передачу управління від блоку до блоку не за допомогою явної вказівки ліній переходу, а за допомогою вкладених структур.

Дії в структурограмі розташовуються одна під одною. Це дозволяє наочно відслідковувати оброблення даних в алгоритмах. Кожен блок має форму прямокутника і може бути вписаний в будь-який внутрішній прямокутник будь-якого іншого блоку. Запис всередині блоку проводиться на природній мові або за допомогою математичних позначень (див. рис. 1.3). Основні символи структурограм і їх зв'язок з аналогічними конструкціями псевдокоду показані в таблиці нижче.

Переваги N – S – діаграм:

– наочність;

– відсутність з'єднувальних ліній зі стрілками, що допомагає уникнути випадкових помилок;

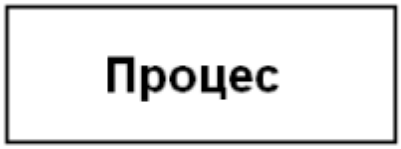
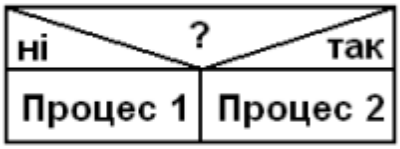
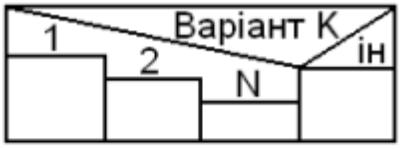
– компактність, навіть відносно довгий алгоритм на мові N-S-діаграм нескладно розмістити на одній сторінці;

– простота використання;

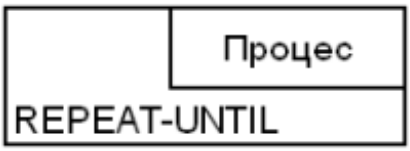
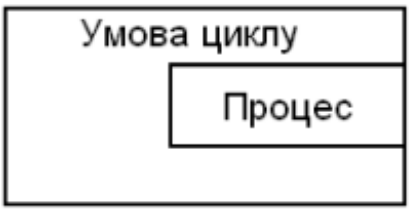
– підтримуються автоматичною кодогенерацією;

– дозволяють здійснювати декомпозицію обчислювальних процесів. Основний недолік N – S – діаграм: – труднощі модифікації при зміні деталей.

Таблиця 1.2 – Основні символи структурограм і їх зв'язок з аналогічними конструкціями псевдокоду

Структурограма		Псевдокод
	Обробка	Будь-яка дія
	Слідування	Група дій, що утворює блок
	Неповне розгалуження	**якщо** *умова* **то** *дії* **все**
	Повне розгалуження	**якщо** *умова* **то** *дія 1* **інакше** *дія 2* **все**
	Вибір	**вибір** **при** *умова 1*: *дії 1* **при** *умова 2*: *дії 2* ..... **при** *умова N*:

Продовження таблиці 1.2

Структурограма		Псевдокод
		дії N **все**
	Цикл з передумовою	**нц поки** *умова* *тіло цикла* *(послідовність дій)* **кц**
	Цикл з післяумовою	**нц для** *i* **від** *i1* **до** *i2* *тіло цикла* *(послідовність дій)* **Кц**
	Цикл-лічильник	**нц для** *i* **від** *i1* **до** *i2* *тіло цикла* *(послідовність дій)* **Кц**

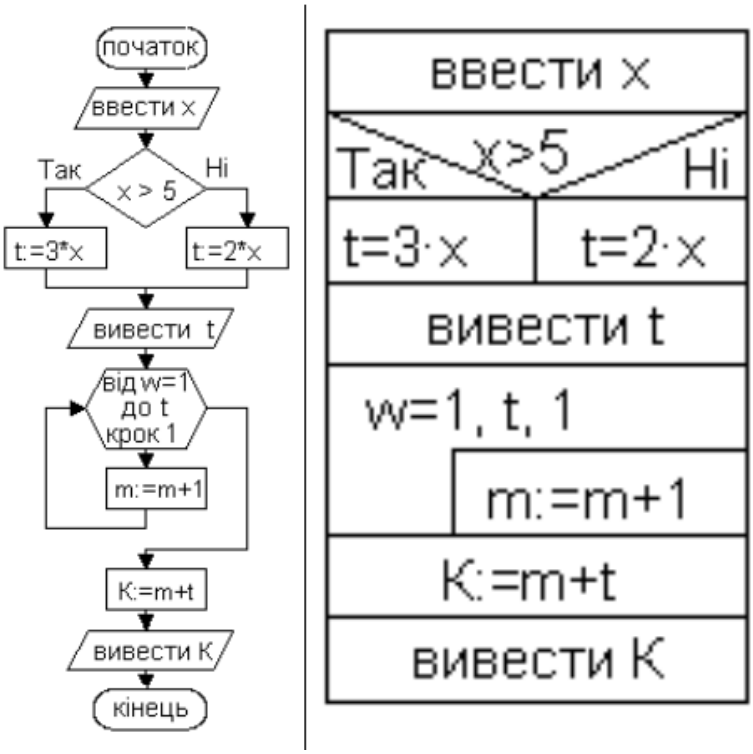


Рисунок 1.3 – Блок-схема і структурограма Нассі–Шнейдермана для обчислення математичного виразу з розгалуженням

Приклад 1.3

Знаходження найбільшого спільного дільника двох чисел за допомогою алгоритму Евкліда.

(Найбільшим спільним дільником (НСД) двох цілих чисел називається таке найбільше по модулю число, яке ділить ці два числа.)

Словесний опис алгоритму.

Є два ненульових натуральних числа. Більше з них (або будь-яке, якщо числа рівні) замінюється різницею цих чисел. Цей процес повторюється до тих пір, поки не залишиться одне нульове число. Це число і буде найбільшим спільним дільником двох натуральних чисел.

Покроковий опис алгоритму.

1. Задати два числа.

2. Якщо числа рівні, то взяти будь-яке з них як відповідь і зупинитися, інакше продовжити виконання алгоритму.

3. Визначити більше з чисел.

4. Замінити більше з чисел різницею більшого і меншого з чисел.

5. Повторити алгоритм з кроку 2.

Опис алгоритму Евкліда за допомогою блок-схеми (рис. 1.4).

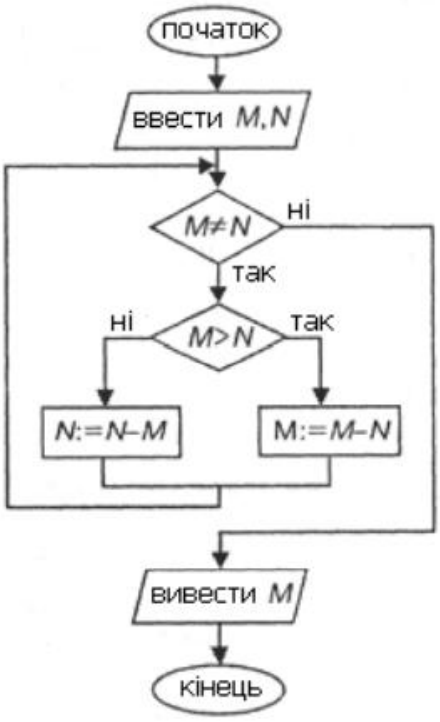


Рисунок 1.4 – Блок-схема алгоритму Евкліда

Опис алгоритму Евкліда за допомогою псевдокоду (рис. 1.5).

```
Алг Алгоритм Евкліда
Поч Вивод ('Визначення НБС двох чисел за допомогою
алгоритму Евкліда')
    Запит на введення даних ('введіть перше число
A='; A)
    Ввести A
    Запит на введення даних ('введіть друге число
B='; B)
    Ввести B
        циклпоки A ≠ B
            Якщо A > B
                То A = A – B
                    Інакше B = B – A
                Кінякщо
            Кінцикл
        Вивести A
    Кін
```

Рисунок 1.5 – Псевдокод алгоритму Евкліда

Опис алгоритму Евкліда за допомогою структурограми Нассі–Шнейдермана (рис. 1.6)

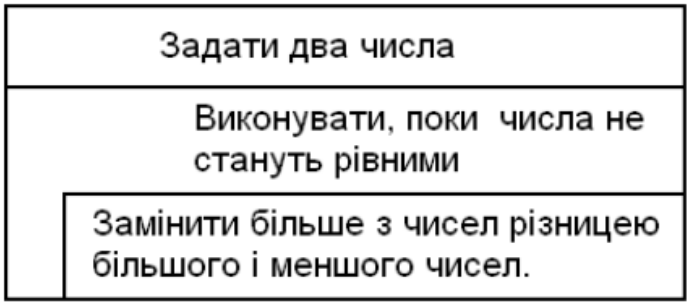


Рисунок 1.6 – Структурограма Нассі–Шнейдермана алгоритму Евкліда

Подання алгоритму графом

Безпосередньо за функціональною блок-схемою будується граф цифрового кола (рис. 1.7).

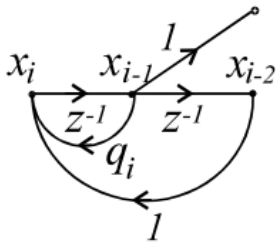


Рисунок 1. 7 – Подання алгоритму у вигляді графу (цифрового кола)

Формально граф алгоритму подається множинами вузлів *v*, направлених гілок *h* з "вагою" (деколи їх називають вершинами та ребрами) і предикатними відношеннями між ними. Для подання цих відношень використовуються матриці.

Матрицею інциденцій $I = [V \times H]$ описують відношення між вузлами та гілками. Її елементами є 0 чи 1, залежно до відношення між відповідним вузлом та гілкою.

Матрицею суміжності $R = [V \times V]$ описують відношення між вузлами. Її елементами r_{ij} є числа, якими подають число гілок між відповідними вузлами.

Між цими матрицями існує зв’язок. Використовуючи матриці інциденцій і суміжності, інші властивості графу обчислюють характеристики графу чи його частин — шляхів, дерев тощо.

Наприклад, знаючи час виконання операцій, які відображені вузлами та гілками графу досліджуваного алгоритму, та використавши поняття шляху можна побудувати алгоритм обчислення часу роботи досліджуваного алгоритму; автоматичним підрахунком числа вузлів, гілок можна встановити складність

алгоритму (надаючи певним операціям відповідну "вагу"). Матриці *R* та *I* використовують при цьому для подання алгоритму відповідною машинною мовою програмування.

На базі графів алгоритму проводяться також тотожні перетвори його структури. При цьому модифікуються матриці *I* та *R*. Ця модифікація є комбінаторною з використанням тотожних перетворів матриць та відношення порядку на множині вузлів.

Систематичне вивчення алгоритмів і різних моделей обчислень привело до створення ряду прикладних дисциплін, розвитку засобів обчислювальної техніки та сучасних комунікацій. Розвиток теорії алгоритмів в 30-і роки XX століття, коли ніяких комп'ютерів ще не було, стало стимулом для появи в 40-х роках перших комп'ютерів.

2 АЛГОРИТМІЧНІ МОДЕЛІ – АБСТРАКТНІ МАШИНИ ПОСТА Й ТЬЮРІНГА. КОМБІНАТОРНІ МОДЕЛІ – НОРМАЛЬНІ АЛГОРИФМИ МАРКОВА

2.1 Машина Поста

Однією з фундаментальних статей, результати якої лежать в основі сучасної теорії алгоритмів є стаття Еміля Поста (Emil Post), «Фінітні комбінаторні процеси, формулювання 1», опублікована в 1936 році в вересневому номері «Журналу символічної логіки».

Пост розглядає загальну проблему, що складається з безлічі конкретних проблем, при цьому розв'язок загальної проблеми це такий розв'язок, який доставляє відповідь для кожної конкретної проблеми.

Наприклад, розв'язок рівняння $3 * x + 9 = 0$ – це одна з конкретних проблем, а розв'язок рівняння $a * x + b = 0$ – це загальна проблема, тим самим алгоритм (сам термін «алгоритм» не використовується Постом) повинен бути універсальним, тобто може бути поєднаний із загальною проблемою.

Основні поняття алгоритмічного формалізму Посту – це простір символів (*мова L*) в якому задається конкретна проблема і виходить відповідь, і набір інструкцій, тобто операцій в просторі символів, які задають як самі операції, так і порядок виконання інструкцій.

Постівскій простір символів – це нескінченна стрічка осередків (ящиків). Кожен ящик або осередок можуть бути позначені або не позначені.

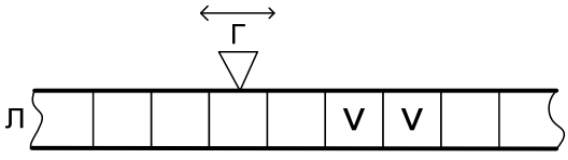


Рисунок 2.1 – Конструкція машини Посту (основний фрагмент) – Л – стрічка, V – мітка, Г – головка

Конкретна проблема задається «зовнішньою силою» (термін Посту) позначкою кінцевого кількості осередків, при цьому, очевидно, що будь-яка конфігурація починається і закінчується поміченим ящиком. Після застосування до конкретної проблеми деякого набору інструкцій розв'язання представляється

так само у вигляді набору помічених і непомічених ящиків, яке розпізнається тієї ж зовнішньої силою (рис. 2.2, 2.3)

Варіанти закінчення виконання програми на машині Поста:

• Команда "стоп" – коректна зупинка. Виникає в результаті виконання правильно написаного алгоритму.

• Виконання неприпустимою команди – нерезультативна зупинка. Випадки, коли головка повинна записати мітку там, де вона вже є, або стерти мітку там, де її немає, є аварійними (неприпустимими).

• Догляд в нескінченність, зациклення. Машина Поста в результаті роботи алгоритму може взагалі не зупинитися (ніколи не дійти до команди «стоп» і ніколи не завершитися аварійною ситуацією).

→	Шаг вправо
←	Шаг влево
V	Записать отметку
X	Стереть отметку
? a; b	Просмотреть ячейку; если в ячейке находится 0, то перейти на команду с номером *a*, иначе на команду с номером *b*
!	Останов

Рисунок 2.2 – Основні команди машини Поста

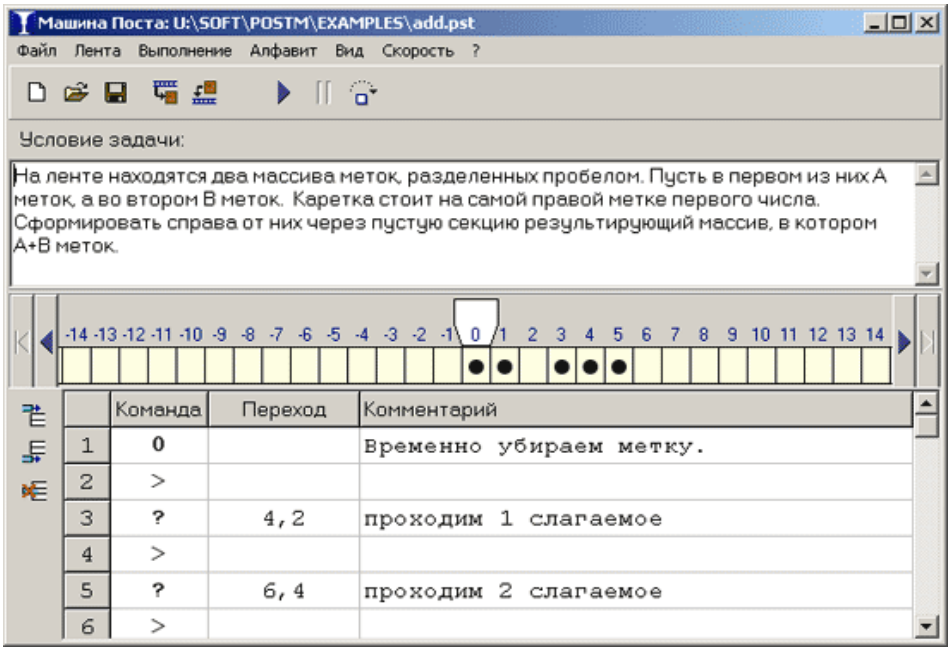


Рисунок 2.3 – Тренажер машини Поста

Елементарні дії (команди) машина Поста простіше команд машини Тьюрингу. Тому програми для машини Поста мають більше число команд, ніж аналогічні програми для машини Тьюрингу.

2.2 Машини Тьюринга

Машини, введені Постом і Тьюрингом, мають не дуже істотну відмінність і зазвичай називаються машинами Тьюринга.

Алан Тьюринг (Turing) у 1936 році опублікував у працях Лондонського математичного товариства статтю «Про обчислюваних числах в додатку до проблеми дозволу», яка нарівні з роботами Посту і Черча лежить в основі сучасної теорії алгоритмів.

Машина Тьюринга – це уявний обчислювальний пристрій, що має такі складові частини. Він має стрічку, розбиту на осередки, і каретку, розташовану в кожен конкретний момент роботи машини над деяким осередком стрічки (рис. 2.4)

У кожній машині Тьюринга є дві частини:

1) необмежена в обидві сторони стрічка, розділена на осередки;

2) автомат (головка для зчитування / запису, яка керована програмою).

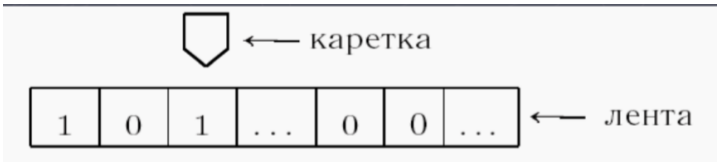


Рисунок 2.4 – Конструкція машини Тьюринга

Кожна клітинка містить рівно один із символів 0 або 1. Стрічка представляється кінцевою, але доповнюється в будь-який момент осередками зліва і справа для запису нових символів 0 або 1 (див. рис. 2.5).

Машина Тьюринга має програму. Це кінцева послідовність інструкцій *q1, q2 , ..., qn*. Кожна з яких є рядком із 5 компонентів: **(*i, a, x, y, z*)**,

де *i* – номер інструкції;

a = 0, або *a* = 1;

x = 0, або *x* = 1;

y = *L* або *y* = *R*;

z – номер інструкції.

Машина Тьюринга – це строга математична побудова, математичний апарат (аналогічний, наприклад, апарат диференціальних рівнянь), створений для вирішення певних завдань. Цей математичний апарат був названий "машиною" з тієї причини, що за описом його складових частин і функціонуванню він схожий на обчислювальну машину.

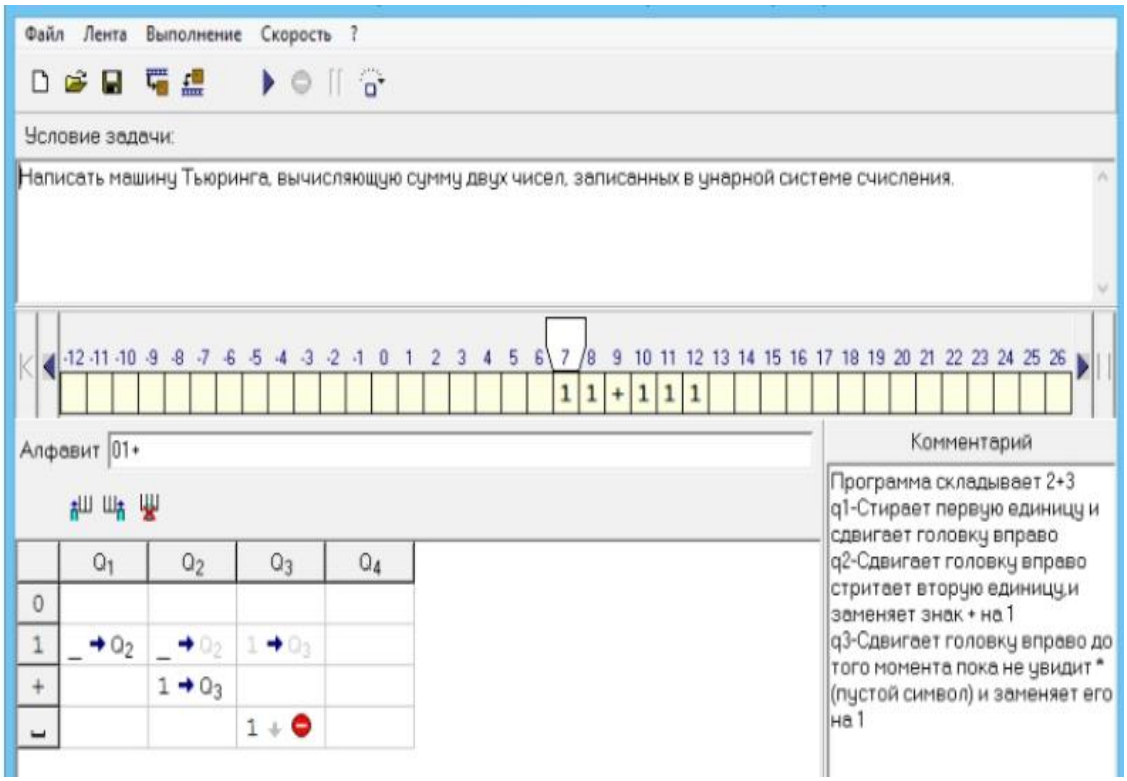


Рисунок 2.5 – Тренажер машини Тьюринга

Принципова відмінність машини Тьюринга від обчислювальних машин полягає в тому, що її пристрій являє собою нескінченну стрічку: у реальних обчислювальних машин пристрій може бути як завгодно великим, але обов'язково кінцевим. Машину Тьюринга можна реалізувати саме через нескінченності її стрічки. У цьому сенсі вона потужніша будь-якої обчислювальної машини.

Алан Тьюринг висловив припущення, що ***будь-який алгоритм в інтуїтивному сенсі цього слова може бути представлений еквівалентної машиною Тьюрінга.***

Це припущення відомо як ***теза Черча-Тьюринга***. Кожен комп'ютер може моделювати машину Тьюринга (операції перезапису елементів, порівняння і переходу до іншої сусідньої осередку з урахуванням зміни стану машини). Отже, він може моделювати алгоритми в будь-якому формалізмі, і з цієї тези випливає, що всі комп'ютери (незалежно від потужності, архітектури тощо) еквівалентні з точки зору принципової можливості вирішення алгоритмічних задач.

2.3 Нормальні алгорифми Маркова (НАМ)

Третій варіант формалізації поняття алгоритму запропоновано російським математиком А. А. Марковим. У цьому визначенні вважається, що алгоритмічний процес – це процес переробки слів деякого алфавіту.

Для формалізації поняття алгоритму російський математик А. А. Марков запропонував використовувати асоціативні обчислення.

Розглянемо деякі поняття асоціативного обчислення. Нехай є алфавіт (кінцевий набір різних символів). Складові його символів будемо називати буквами. Будь-яка кінцева послідовність літер алфавіту (лінійний їх ряд) називається словом в цьому алфавіті.

Розглянемо два слова *N* і *M* в деякому алфавіті *A*. Якщо *N* є частиною *M*, то кажуть, що *N* входить в *M*.

Наприклад, в алфавіті *A = {a, b, c}* є слова *N = ab*, *M = bcb*, *K = abcbcbab*, Замінивши в слові *K* слово *N* на *M*, отримаємо *bcbcbcbab* або *abcbcbbcb*, і, навпаки, замінивши *M* на *N*, отримаємо *aabcbab* або *abcabab.*

Підстановка *ab – bcb* недопустима до слова *bacb*, так як ні *ab*, ні *bcb* не входять в це слово. До отриманих за допомогою допустимих підстановок слів можна знову застосувати допустимі підстановки і т. д.

Сукупність усіх слів в даному алфавіті разом з системою допустимих підстановок називають ***асоціативним обчисленням***. Щоб задати асоціативне обчислення, досить задати алфавіт і систему підстановок.

Слова *P1* і *P2* в деякому асоціативному обчисленні називаються ***суміжними***, якщо одне з них може бути перетворено в інше одноразовим застосуванням допустимої підстановки.

Послідовність слів *P, P1, P2, ..., M* називається ***дедуктивним ланцюжком***, що веде від слова *P* до слова *M*, якщо кожне з двох слів, які стоять поруч цього ланцюжка – суміжне.

Слова *P* і *M* називають ***еквівалентними***, якщо існує ланцюжок від *P* до *M* і назад.

Такий набір приписів разом з алфавітом *A* і набором підстановок *B* визначають нормальний алгоритм. Процес зупиняється тільки в двох випадках: 1) коли відповідну підстановку не знайдено; 2) коли застосована остання підстановка з їх набору. Різні нормальні алгоритми відрізняються один від одного алфавітами і системами підстановок.

Нормальний алгоритм Маркова можна розглядати як універсальну форму завдання будь-якого алгоритму. Універсальність нормальних алгоритмів декларується принципом нормалізації: для будь-якого алгоритму в довільному кінцевому алфавіті *A* можна побудувати еквівалентний йому нормальний алгоритм над алфавітом *A*.

Алгоритмом в алфавіті A називається зрозуміле точне розпорядження, що визначає процес над словами з *A* і допускає будь-яке слово в якості вихідного.

Алгоритм в алфавіті *A* задається в вигляді системи допустимих підстановок, доповненої точним розпорядженням про те, в якому порядку потрібно застосовувати допустимі підстановки і коли настає зупинка.

Так, застосовуючи систему підстановок *B* з розглянутого прикладу до слів *babaac* і *bcacabc* отримуємо:

babaac → *bbcaaac* → зупинка

bcacabc → *bcacbcac* → *bcacccac* → *bcacabc* → нескінченний процес (зупинки немає), так як ми отримали вихідне слово.

Для кожного асоціативного обчислення існує завдання: для будь-яких двох слів визначити, чи є вони еквівалентними чи ні.

Приклад 2.1. Алфавіт підстановки

{a, b, c, d, e} ac -ca, abac - abace ad - da; eca - ae bc - cb; eda - be bd - db; edb - be

Слова a**bcde** і **acbde** – суміжні (підстановка **bc - cb**). Слова **abcde** і **cadbe** – еквівалентні.

Нижче перераховані способи композиції (що дозволяють будувати нові алгоритми з уже відомих) нормальних алгоритмів.

I. Суперпозиція алгоритмів. При суперпозиції двох алгоритмів ***A*** і ***B*** вихідне слово першого алгоритму розглядається як вхідне слово другого алгоритму ***B***. Результат суперпозиції ***C*** може бути представлений у вигляді ***C (p) = B (A (p)).***

II. Об'єднання алгоритмів. Об'єднанням алгоритмів ***A*** і ***B*** в одному і тому ж алфавіті називається алгоритм ***C*** в тому ж алфавіті, що перетворює будь-яке слово ***p***, що міститься в перетині областей визначення алгоритмів ***A*** і ***B***, в записані поруч слова ***A (p)*** *і* ***B (p)***.

III. Розгалуження алгоритмів. Розгалуження алгоритмів є композицією ***D*** трьох алгоритмів ***A***, ***B*** і ***C***, причому область визначення алгоритму ***D*** є перетином областей визначення всіх трьох алгоритмів ***A***, ***B*** і ***C***, а для будь-якого слова ***p*** з цього перетину ***D (p) = A (p),*** якщо ***C (p) = e, D (p) = B (p***), якщо ***C (p) = e***, де ***e*** – порожній рядок.

IV. Ітерація алгоритмів. Ітерація (повторення) являє собою таку композицію ***C*** двох алгоритмів ***A*** і ***B***, що для будь-якого вхідного слова ***p*** відповідне слово ***C(p)*** виходить в результаті послідовного багаторазового застосування алгоритму ***A*** до тих пір, поки не вийде слово, що перетворюється алгоритмом ***B.***

Нормальні алгоритми Маркова є не тільки засобом теоретичних побудов, а й основою спеціалізованої мови програмування, що застосовується як мова символьних перетворень при розробці систем штучного інтелекту.

Існує строгий доказ того, що за можливостями перетворення нормальні алгоритми Маркова еквівалентні машинам Тьюрінга**.**

3 АЛГОРИТМИ Й ОБЧИСЛЮВАЛЬНІ ФУНКЦІЇ. РЕКУРСИВНІ ФУНКЦІЇ

3.1 Основні означення

Період до початку XX століття можна вважати етапом накопичування інформації про алгоритми. Інтуїтивне розуміння алгоритму було достатнім, щоб вважати ту чи іншу процедуру розв'язання задач алгоритмом. Проте одночасно нагромаджувались проблеми, пошук алгоритму вирішення яких ні до чого не приводив. Та для того щоб довести, що алгоритм існує, достатньо його пред'явити, а ось для доведення його відсутності необхідне вже строге визначення шуканого об'єкта. Цим питанням і потрібно було зайнятися.

Оскільки в математиці поняття алгоритму тісно пов'язане з поняттям функції, то історично першою формалізацією алгоритму став клас обчислювальних функцій (К. Гьодель, А. Черч, 1935–1936 рр.). Основна ідея в тому, що довільний алгоритм можна звести до обчислення значення деякої числової функції, тобто з кожним алгоритмом можна зв'язати функцію, яку він обчислює. При цьому виникає ряд запитань: чи для будь-якої функції існує обчислювальний її алгоритм; для яких функцій алгоритми існують і як описати такі алгоритмічні функції? Пошук відповіді на ці питання й привів до створення теорії рекурсивних функцій.

При цьому необхідно зазначити, що в теорії обчислювальних функцій визначають множину натуральних чисел $N=\{0,1,2,3,\ldots\}$ і розглядають тільки числові функції, розуміючи під ними функції k змінних (k-місні функції), аргументи і значення яких належать N. Тобто, об'єкти з областю визначення $D_f \subseteq N^k$ (k – ціле додатне) та з областю значень $R_f \subseteq N$ будемо називати k-місними частковими функціями. Термін "часткова" нагадує, що функція визначена на підмножині N^k(звичайно, може статися, що $D_f = N^k$, в такому разі функція стає всюди визначеною).

Числову функцію $f: N^k \to N$ називають ***обчислювальною,*** якщо існує алгоритм, за допомогою якого можна обчислити значення функції для будь-якого набору значень аргументів із області визначення функції.

Надалі використовуватиме ідею К. Гьоделя та С. Кліні (1936 р.), за якою всі обчислювальні функції можна одержати із множини базисних функцій та алгебраїчних операцій. Самі операції прийнято називати операторами.

Розглянемо клас числових функцій, що використовуються як базис для побудови обчислювальних функцій:

O(x) = 0 – нуль-функція (можна задати і ***n –*** місну нуль – функцію $O^n(x_1, x_2, \ldots, x_n) = 0$).

S(x) = x + 1 – функція слідування або наступності (але не додавання одиниці).

$I^n_m(x_1, x_2, x_3, .., x_m, \ldots x_n) = x_m$ – функція проєкції або введення фіктивних змінних або вибору аргументу.

Як операторів, застосування яких до базисних функцій призводить до утворення нових функцій оберемо такі три оператори:

1) оператор суперпозиції;

2) оператор примітивної рекурсії;

3) оператор мінімізації або найменшого кореня.

3.2 Оператор суперпозиції

Операція суперпозиції полягає у підстановці одних арифметичних функцій замість аргументів інших функцій.

Нехай задана ***m***-місна функція $F(x_1, x_2, x_3, .., x_m)$ та ***m***-на кількість ***n***-місних функцій $f_1(x_1, x_2, x_3, .., x_n)$, $f_2(x_1, x_2, x_3, .., x_n)$, $f_3(x_1, x_2, x_3, .., x_n), \ldots, f_m(x_1, x_2, x_3, .., x_n)$. Тоді говорять, що ***n***-місна функція $\varphi(x_1, x_2, x_3, .., x_n)$ утворилася в результаті підстановки у функцію ***F*** замість її аргументів *m* функцій $f_1, f_2, f_3, \ldots, f_m$. Така підстановка називається ***суперпозицією*** S^n_m. Тоді:

$$S^n_m(F, f_1, f_2, \ldots, f_m) = F\,(f_1(x_1, x_2, .., x_n)\,, f_2(x_1, x_2, .., x_n), \ldots, f_m(x_1, x_2, .., x_n)) = \varphi\,(x_1, x_2, .., x_n). \quad (3.1)$$

Приклад 3.1. Здійснити суперпозицію нуль-функції та функції слідування, тобто знайти $S^2(S(x); O(x))$.

Розв'язання

Для визначення результату операції суперпозиції потрібно функцію ***O(x) = 0*** підставити в ***S(x) = x+1*** замість значення аргументу. Отримаємо

$$S^2(S(x);O(x)) = S(O(x)) = O(x) + 1 = 0+1 = 1.$$

3.3 Оператор примітивної рекурсії

Оператор примітивної рекурсії ($\boldsymbol{R^n}$) дозволяє будувати $\boldsymbol{n+1}$-місну арифметичну функцію $\boldsymbol{f(x_1, x_2, x_3,.., x_n, y)}$ з двох заданих функцій, одна з яких є $\boldsymbol{n}$-місна $\boldsymbol{\varphi(x_1,x_2,..,x_n)}$, а інша - $\boldsymbol{n+2}$-місна функція $\boldsymbol{\psi(x_1,x_2,..,x_n,y,z)}$ за такою схемою**:** $\boldsymbol{f(x_1, x_2, x_3,.., x_n,0) = \varphi\ (x_1,x_2,..,x_n);}$

$$f(x1, x2, x3,.., xn, y + 1) = \psi(x1, x2,.., xn, y, f(x1, x2, x3,.., xn, y)). \quad (3.2)$$

Таким чином, $\boldsymbol{f(x_1, x_2, x_3,.., x_n, y) = R^n(\varphi,\psi).}$

Для розуміння операції примітивної рекурсії необхідно зазначити, що будь-яку функцію від меншої кількості аргументів можна розглядати як функцію від більшої кількості аргументів. Зокрема, функції – константи ($\boldsymbol{n=0}$) є одномісними і відповідно**:** $\boldsymbol{f(x)=a;\ f(x, y + 1) = \psi(x,y, f(y))}$, де $\boldsymbol{a}$ – константа.

Схеми примітивної рекурсії визначають функцію $\boldsymbol{f}$ тільки через інші функції $\boldsymbol{\varphi}$ та $\boldsymbol{\psi}$, а й через значення $\boldsymbol{f}$ у попередніх точках – значення $\boldsymbol{f}$ у точці $\boldsymbol{(y + 1)}$ залежить від значення $\boldsymbol{f}$ у точці ($\boldsymbol{y}$).

Приклад 3.2. Знайти значення функції $f(3,2)$, якщо вона задана співвідношеннями

$f(x, 0) = 0;\ f(x, y+1) = f(x, y) + x.$

Розв'язання. У даному разі за схемою (3.2) маємо : $f(x, 0) = \varphi(x) = 0$,

$\psi(x,y,z) = y + z$. Виходячи з того, що $f(x, 0) = \varphi(x) = 0$ при будь-якому x, тоді й $f(2, 0) = 0$. Обчислюючи послідовно, одержимо:

$f(2, 1) = f(2 ,0) + 2 = 0+ 2 = 2;$

$f(2, 2) = f(2, 1) + 2 = 2+ 2 = 4;$

$f(3, 2) = f(2, 2) + 2 = 4+ 2 = 6$, що є остаточною відповіддю.

Нескладно довести, що в даному прикладі $f(x,y)=x \bullet y$.

3.4 Оператор мінімізації

Розглянемо обчислювальну $\mathbf{n+1}$-місну функцію $\boldsymbol{f(x_1, x_2, x_3,.., x_n, x_{n+1})}$. Зафіксуємо значення $\boldsymbol{x_1, x_2, x_3,.., x_n}$ її перших n аргументів та розглянемо рівняння $\boldsymbol{f(x_1, x_2, x_3,.., x_n, y) = 0}$, тобто знайдемо значення, y при якому функція дорівнює нулю. Більш складною буде задача відшукати найменше з усіх значень y, при якому $\boldsymbol{f(x_1, x_2, x_3,.., x_n, y) = 0}$. Оскільки результат знаходження залежить від $\boldsymbol{x_1, x_2, \ldots, x_n}$, то й найменше значення $\boldsymbol{y}$ є також їх функцією. Введемо позначення

$$\varphi\ (x_1, x_2, \ldots, x_n) = \mu_y\ [\ f(x_1, x_2, \ldots, x_n, y) = 0], \qquad (3.3)$$

де ***y*** таке, що $\boldsymbol{f(x_1, x_2, \ldots, x_n, y) = 0}$, а $\boldsymbol{\mu_y}$ – оператор мінімізації.

Для знаходження функції $\varphi(x_1, x_2, \ldots, x_n)$ можна запропонувати таку процедуру.

1. Обчислюємо $\boldsymbol{f(x_1, \ldots x_n, 0)}$; якщо її значення буде нуль, то $\boldsymbol{\varphi(x_1, \ldots x_n) = 0}$. Якщо $\boldsymbol{f(x_1, \ldots x_n, 0) \neq 0}$, то переходимо до наступного кроку.

2. Обчислюємо $\boldsymbol{f(x_1, \ldots x_n, 1)}$; якщо її значення буде нуль, то $\boldsymbol{\varphi(x_1, \ldots x_n) = 1}$. Якщо $\boldsymbol{f(x_1, \ldots x_n, 0) \neq 0}$, то переходимо до наступного кроку.

І так далі, поки не знайдемо перше значення ***y***, при якому $\boldsymbol{f(x_1, \ldots x_n, y) = 0}$.

Якщо визначиться, що для всіх ***y*** $\boldsymbol{f(x_1, \ldots x_n, y) \neq 0}$, то функція $\boldsymbol{\varphi(x_1, \ldots x_n)}$ вважається невизначеною.

Приклад 3.3. Розглянемо функцію $\boldsymbol{\varphi(x,y) = x - y}$, яку можна отримати за допомогою μ-оператора $\boldsymbol{d(x,y) = \mu_z[y + z = x] = \mu_z[S((I^3_2\ (x, y, z),\ I^3_3\ (x, y, z),)] = I^3_1\ (x, y, z)}$ та обчислимо $\boldsymbol{f(7,2)}$.

Розв'язання. Задамо ***y*** значення 2 та встановимо змінній *z* послідовно значення 0,1,2.., кожного разу обчислюючи суму $\boldsymbol{y + z}$. Як тільки при якомусь першому в заданому порядку ***z*** сума дорівнюватиме 7, то відповідне значення візьмемо за значення $\boldsymbol{d(7, 2)}$. Проведемо обчислення:

z=0 2+0=2 < > 7

z=1 2+1=3 < > 7

z=2 2+2=4 < > 7

z=3 2+3=5 < > 7

z=4 2+4=6 < > 7

z=5 2+5=7=7

Таким чином, $d(7, 2) = 5$.

3.5 Рекурсивні функції

По суті один і той же метод, стосовно до різних областей носить різні назви – це ***індукція***, ***рекурсія*** і ***рекурентні співвідношення*** – відмінності стосуються особливостей використання.

Під ***індукцією*** розуміється метод доведення тверджень, який будується на базі індукції при $\boldsymbol{n = 0,1}$, потім твердження покладається правильним при $\boldsymbol{n = n*b}$ і проводиться доказ для $\boldsymbol{n + 1}$.

Під ***рекурсією*** розуміється метод визначення функції через її попередні і раніше певні значення, а так само спосіб організації обчислень, при якому функція викликає сама себе з іншим аргументом.

Термін ***рекурентні співвідношення*** пов'язаний з американським науковим стилем і визначає математичне завдання функції за допомогою рекурсії.

Основним завданням дослідження рекурсивно заданих функцій є отримання ***f(n)*** в явній або як ще кажуть «замкнутої» формі, тобто у вигляді аналітично заданої функції. У зв'язку з цим розглянемо ряд прикладів:

Приклади рекурсивного завдання функцій

1. $\begin{cases} f(0)=0 \\ f(n)= f(n-1)+1 \end{cases}$

Ясно, що $f(n)=n$.

2. $\begin{cases} f(0)=1 \\ f(n)= n*f(n-1) \end{cases}$

Послідовна підстановка дає $f(n)=1*2*3*...*n = n!$

$n! \approx (2\pi n)^{1/2} *(n^n)/(e^n)$ – формула Стирлінгу наближеного обчислення факторіала для великих ***n***.

3. $\begin{cases} f(0)=1 \\ f(1)=1 \\ f(n)= f(n-1)+ f(n-2), \quad n\geq 2 \end{cases}$

Ця рекурсивна функція визначає числа Фібоначчі: 1 1 2 3 5 8 13, які досить часто виникають при аналізі різних завдань, у тому числі й при аналізі алгоритмів. Відзначимо, що асимптотично $f(n)\approx [1,618^n]$ [9].

4. $\begin{cases} f(0)=1 \\ f(n)= f(n-1)+ f(n-2)+...+1 = \sum f(i)+1 \end{cases}$

Для отримання функції в явному вигляді розглянемо її послідовні значення: $f(0)=1$, $f(1)=2$, $f(2)=4$, $f(3)=8$, що дозволяє припустити, що $f(n)=2^n$, точний доказ виконується по індукції.

5. $\begin{cases} f(0)=1 \\ f(n)= 2*f(n-1) \end{cases}$

Одна і та ж функція може мати різні рекурсивні визначення – $f(n) = 2^n$, як і в прикладі 4, що перевіряється елементарної підстановкою.

6. $\begin{cases} f(0)=1 \\ f(1)=2 \\ f(n)= f(n-1)*f(n-2) \end{cases}$

У цьому випадку ми можемо отримати розв'язок у замкнутій формі, зіставивши значення функції, що відповідають ступені двійки:

$$f(2) = 2 = 2^1$$

$$f(3) = 4 = 2^2$$

$$f(4) = 8 = 2^3$$

$$f(5) = 32 = 2^5$$

$$f(6) = 256 = 2^8$$

Позначивши через $\boldsymbol{F_n}$ **- n** -е число Фібоначчі, маємо: $f(n) = 2^{Fn}$.

3.6 Гіпотеза Черча та примітивно-рекурсивні функції

Функцію називають ***примітивно-рекурсивною,*** якщо вона утворена з найпростіших функцій за допомогою скінченного числа застосувань операторів суперпозиції $\mathbf{S^n_m}$ та примітивної рекурсії $\mathbf{R^n}$.

Функцію називають ***частково-рекурсивною,*** якщо вона утворена з найпростіших функцій за допомогою скінченного числа застосувань операторів суперпозиції $\boldsymbol{S^n_m}$, примітивної рекурсії $\boldsymbol{R^n}$ та мінімізації $\boldsymbol{\mu y}$.

З цих означень та зауваження щодо зберігання операторами властивості обчислювальності функцій випливає, що кожна стандартно задана частково-рекурсивна функція є обчислювальною за певною процедурою, яка відповідає інтуїтивному уявленню алгоритму, а з іншого боку – які б досі не будувалися класи точно визначених алгоритмів, завжди з'ясовувалося, що числові функції, які обчислювалися за алгоритмами цих класів, були частково-рекурсивними. Тому загальноприйнятою є така наукова гіпотеза (теза Черча).

Гіпотеза (теза) Черча. ***Клас алгоритмічно обчислювальних часткових числових функцій збігається з класом усіх частково-рекурсивних функцій.***

До формулювання цієї тези входить інтуїтивне поняття обчислювальності, тому його не можна ні спростувати, ні довести. Це факт, на користь якого свідчить багаторічна математична практика.

Зрозуміло, що будь-яка примітивно рекурсивна функція є частково рекурсивною, оскільки за визначенням оператори для побудови частково рекурсивних функцій містять у собі оператори для побудови примітивно рекурсивних

функцій. Також зрозуміло, що примітивно рекурсивна функція визначена скрізь і тому є загальнорекурсивною функцією (у примітивно рекурсивної функції немає приводу «*зависати*», тому що при її побудові використовуються оператори, що визначають скрізь визначені функції).

Досить складно довести існування і навести приклад загальнорекурсивної функції, що не є примітивно рекурсивною.

Функція Аккермана – приклад такої обчислювальної функції, яка не є примітивно рекурсивною. Вона набуває два невід'ємних цілих числа як параметри і повертає натуральне число, позначається *A(m;n).*

Ця функція зростає дуже швидко, наприклад, число A (4; 4) настільки велике, що кількість цифр у порядку цього числа у багато разів перевершує кількість атомів в спостережуваної частини Всесвіту. Функція Аккермана визначається рекурсивно для невід'ємних цілих чисел m та n таким чином:

$$A(m,n) = \begin{cases} n+1, & m = 0 \\ A(m-1,1) & m > 0, n = 0; \\ A(m-1), A(m,n-1)), & m > 0, n > 0 \end{cases}$$

За означенням примітивно-рекурсивної функції, неважко знайти процедуру, що породжує всі примітивно-рекурсивні функції.

Надамо цьому означенню більш формального індуктивного вигляду.

1. Функції $O(x) = 0$, $S(x) = x+1$, $I^n_m(x_1, x_2, \ldots, x_m, \ldots, x_n) = x$ для всіх натуральних n, m є примітивно-рекурсивними.

2. Якщо $g_1(x_1,\ldots,x_n), \ldots, g_m(x_1,\ldots,x_n), h(x_1,\ldots,x_m)$ – примітивно-рекурсивні функції, то $S^n_m(h,g_1,\ldots,g_m)$ – примітивно-рекурсивні функції для будь-яких натуральних n,m.

3. Якщо $g(x_1,\ldots,x_n)$, та $h(x_1,\ldots,x_n,y,z)$ – примітивно-рекурсивні функції, то $R^n(g,h)$ – примітивно-рекурсивна функція.

4. Інших примітивно-рекурсивних функцій немає.

Приклад 3.4. Навести приклади арифметичних функцій, що є примітивно-рекурсивними.

Розв'язання

Функція додавання двох натуральних чисел $Sum(a,b) = a+b$.

Ця функція може бути розглянута як примітивно-рекурсивна функція двох змінних, одержувана внаслідок застосування оператора примітивної рекурсії до функцій I^1_1 та F, другу з яких одержимо підстановкою функції проєкції I^3_3 у функцію слідування S. Покажемо це математично:

$$Sum\ (x,0) = I_1^1\ (x),$$

$$Sum(x,y+1) = x+y+1=Sum(x,y)+1 = S(Sum(x,y)),$$

$$Sum(x, y) = R^1(I_1^1\ (x), F(x, y, z)),\ де\ F(x, y, z) = S(I_3^3\ (x, y, z))=z+1.$$

Функція множення двох натуральних чисел ***Mul(a,b) = a•b.***

Ця функція може бути розглянута як примітивно-рекурсивна функція двох змінних, одержувана внаслідок застосування оператора примітивної рекурсії до функцій ***O*** та ***G***, другу з яких одержимо підстановкою функції ції I_3^1 та I_3^3 у функцію додавання ***Sum***. Покажемо це математично.

$$Mul(x, 0) = O(x),$$

$$Mul(x,y+1) = x *$$

$$*y+x = G(x, y, Mul(x, y)),$$

$$G(x, y, z) = Sum(I^1{}_3(x, y, z), I^3{}_3(x, y, z)),$$

$$Mul(x, y) = R^1(O(x), G(x, y, z).$$

3.7 Рекурсивна реалізація алгоритмів

Більшість сучасних мов високого рівня підтримують механізм рекурсивного виклику, коли функція, як елемент структури мови програмування, яка повертає обчислене значення за своїм ім’ям , може викликати сама себе з іншим аргументом. Ця можливість дозволяє безпосередньо реалізовувати обчислення рекурсивно певних функцій. Відзначимо, що в силу тези Черча-Тьюринга апарат рекурсивних функцій Черча рівносильний машині Тьюринга, і, отже, будь-який рекурсивний алгоритм може бути реалізований ітераційно.

Розглянемо приклад рекурсивної функції, що обчислює факторіал:

F(n);

If n=0 or n=1 (перевірка можливості прямого обчислення)

Then

F ← 1

Else

*F ← n*F(n-1);* (рекурсивний виклик функції)

Return (F);

End;

Можна помітити, що деяка гілка дерева рекурсивних викликів обривається при досягненні такого значення переданого параметра, при якому функція може бути обчислена безпосередньо. Таким чином, рекурсія еквівалентна конструкції циклу, в якому кожен прохід є виконання рекурсивної функції з заданим параметром.

Розглянемо приклад для функції обчислення факторіала (рис. 3.1).

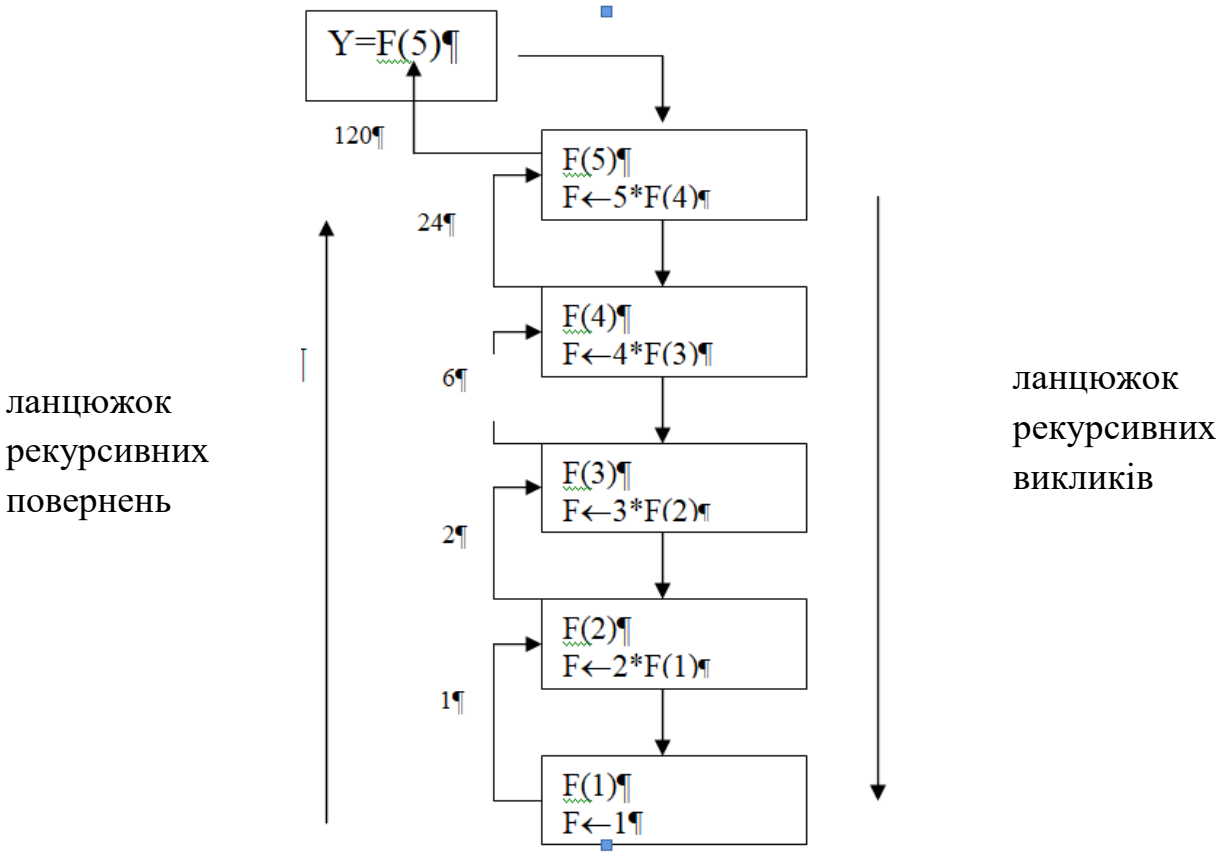


Рисунок 3.1 – Дерево рекурсії при обчисленні факторіала – F(5)

Графічне представлення породжується даними алгоритмом ланцюжка рекурсивних викликів називається *деревом рекурсивних викликів*. Дерево рекурсивних викликів може мати і більш складну структуру, якщо на кожному виклику породжується кілька звернень – фрагмент дерева рекурсій для чисел Фібоначчі представлений на рис. 3.2.

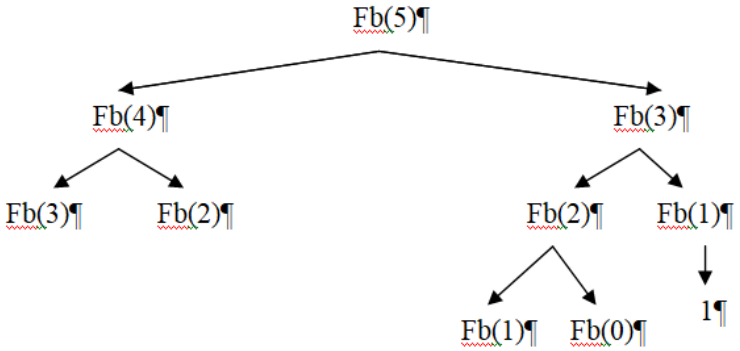


Рисунок 3.2 – Фрагмент дерева рекурсії при обчисленні чисел Фібоначчі – F(5)

Аналіз трудомісткості рекурсивних реалізацій алгоритмів, очевидно, пов'язаний як з кількістю операцій, які виконуються при одному виконанні функції, так і з кількістю таких викликів. Більш детальний розгляд призводить до необхідності врахування витрат як на організацію виклику функції і передачі параметрів, так і на повернення обчислених значень і передачу управління в точку виклику.

3.8 Функції, обчислюванні за Тьюрингом. Теза Тьюринга

Функція ***f(x1, x2,…xn) , n>=1***, називається ***частковою числовою функцією***, якщо аргументи набувають значень з підмножин множини натуральних чисел і сама вона набуває значень в підмножині множини натуральних чисел. Таким чином, часткова числова функція – це відображення : $f : N^n \to N$.

$$D(f) = \{(x_1, x_2, \ldots x_n) : f(x_1, x_2, \ldots, x_n) - \textit{визначено}\} \subseteq N^n;$$

Область визначення часткової числової функції

множина значень

$$E(f) = \{y \in N : \exists x_1, x_2, \ldots x_n \, f(x_1, x_2, \ldots x_n) = y\} \subseteq N.$$

Часткова числова функція ***f(x1, x2,…xn)*** *називається* ***обчислюваною****, якщо існує алгоритм (у розумінні інтуїтивного означення), який дозволяє обчислювати її значення для тих наборів аргументів, для яких вона визначена і який продовжується нескінченно, якщо функція f для даного набору значень аргументів не визначена.*

4 АЛГОРИТМІЧНО НЕРОЗВ'ЯЗНІ ЗАДАЧІ. ОСНОВИ АНАЛІЗУ АЛГОРИТМІВ. АНАЛІЗ ТРУДОМІСТКОСТІ АЛГОРИТМІВ. СКЛАДНІСТЬ АЛГОРИТМУ. КЛАСИ *P* ТА *NP*

4.1 Алгоритмічно нерозв'язні задачі

У теорії обчислювальності існує проблема зависання (обчислювальності): дано опис алгоритму та початкові вхідні дані, необхідно визначити, чи може алгоритм з цими вхідними даними завершитися коли-небудь.

Альтернативою зупинки є ситуація, коли робота алгоритму не припиняється (він зависає).

Машина Тьюринга не може бути використана для вирішення проблеми зависання будь-якого алгоритму. Тому подібна задача відноситься до алгоритмічно нерозв‘язних задач.

Алгоритмічна нерозв‘язність – важлива властивість деяких класів коректно поставлених задач, які допускають використання алгоритмів, яка полягає в тому, що задачі кожного із цих класів в принципі не мають якого-небудь загального універсального алгоритму вирішення, що об‘єднує весь клас.

Незважаючи на повну однотипність умов та вимог для подібних задач не існує єдиного алгоритму вирішення.

Однак це не означає, що подібні задачі не можуть бути вирішені в принципі, просто для їх вирішення необхідно використовувати різні підходи, нерідко творчі і нетрадиційні.

Проблема відсутності загального методу вирішення задачі

Ця проблема визначає певний клас алгоритмічно нерозв‘язних задач, приклади:

1. ***Розподіл дев‘яток у записі числа π:***

Визначимо функцію f(n) = i, де n – кількість цифр ―9‖ підряд у десятковому запису числа π, а i – номер найбільш лівої дев‘ятки із n тих, що йдуть підряд, наприклад: π = 3,141592…; f(1) = 5 Задача полягає у визначенні f(n) для довільного n. Оскільки число π є ірраціональним і трансцендентним, то ми не знаємо жодної інформації про розподіл дев'яток (так само як і будь-яких інших цифр) в десятковому записі числа. Обчислення f (n) пов'язане з обчисленням наступних цифр у розкладі π, до тих пір, поки ми не виявимо n дев'яток поспіль, однак у нас немає загального методу обчислення f (n), тому для деяких n обчислення можуть тривати нескінченно – ми навіть не знаємо в принципі (за природою числа π) чи існує розв’язок для всіх n.

2. ***Обчислення досконалих чисел***

Досконалі числа – це числа, які дорівнюють сумі своїх дільників, наприклад: 28 = 1 + 2 + 4 + 7 + 14. Визначимо функцію *S (n) = n-e* за рахунком досконале число і поставимо задачу обчислення *S (n)* за довільно заданим *n*. Немає загального методу обрахунку досконалих чисел, невідомо навіть, чи множина досконалих чисел обрахована, чи ні, тому алгоритм має перебирати усі числа підряд, перевіряючи їх досконалість. Відсутність загального методу вирішення не дозволяє відповісти на питання про проблему зупинки: якщо ми перевірили *M* чисел під час пошуку *n*-го досконалого числа, чи може це означати, що його взагалі не існує?

3. ***Десята проблема Гільберта***

Нехай задано многочлен *n*-го ступеня з цілими коефіцієнтами – *P*, чи існує алгоритм, який визначає, чи існує розв‘язок рівняння *P = 0* у цілих числах? Математиками доведено, що такого алгоритму не існує, тобто загальний метод визначення цілих коренів рівняння *P* = 0 по цілочисельним коефіцієнтам многочлену відсутній.

4. Проблема зупинки для машин Тьюринга (тобто проблема визначення результативності алгоритмів).

Для заданої машини Тьюринга і вихідних даних вказати алгоритм, який визначав би, чи зупиниться машина Тьюринга при цих даних чи ні.

Для окремих класів машин Тьюринга проблема зупинки є розв’язуваною. Наприклад, машина *M* з зовнішнім алфавітом , внутрішньою пам'яттю і програмою , з будь-якого початкового стану буде працювати нескінченно, заповнюючи одиницями всю стрічку вправо від початкової точки.

У 1936 р. американець Алонзо Черч розробив теорію ***лямбда-числення,*** а також підтвердив існування ***нерозв'язних завдань.*** У тому ж році англієць Алан Тьюринг запропонував алгоритм здійснення ***ефективної вичислімості,*** сполучаються з концептом абстрактної математичної машини. Подібно Геделю і Черчу, Тьюринг підтвердив наявність не тільки вирішуваних, але й ***нерозв'язних функцій.***

Всяка функція, обчислювана на машині Тьюринга, частково-рекурсивна.

4.2 Аналіз алгоритмів на складність

Час роботи будь-якого алгоритму залежить від кількості вхідних даних. Так при роботі алгоритму сортування час залежить від розміру вхідного масиву. У загальному вивчають залежність часу роботи алгоритму від розміру входу. В одних задачах розмір входу це кількість елементів на вході (задача сортування, перетворення Фур‘є). В інших випадках розмір входу це загальна кількість бітів, що необхідні для представлення всіх вхідних даних. Часом роботи алгоритму називається число елементарних кроків, які він виконує.

Вважається, що один рядок псевдокоду вимагає не більше ніж фіксованого числа операцій. Розрізняють також виклик функції, на який іде фіксована кількість операцій та виконання функції, яке може бути довгим.

Так для алгоритму пошуку максимального за значенням числа в масиві розміром n необхідно виконати n кроків. Тоді час роботи такого алгоритму позначимо **T(n)**. Для кожного алгоритму проводиться оцінка кожної операції, скільки разів ця операція виконується. Усі оцінки додаються і виводиться залежність часу роботи алгоритму від розміру входу.

Часова складність алгоритму вирішує строкову задачу за час O (T (n))

T (n) – час, необхідний алгоритму для виконання бітового рядка, довжиною n.

Якщо алгоритм призначений для оброблення рівних за об‘ємом даних, тривалість його роботи щораз залишається незмінною й складність алгоритму є постійною.

Час роботи алгоритму оброблення яких-небудь масивів даних залежить від розмірів цих масивів. Час роботи алгоритму, що виконує тільки операції читання й занесення даних в оперативну пам'ять, визначається формулою ***an+b***, де ***a –*** час, необхідний для читання й занесення в пам‘ять однієї інформаційної одиниці, і ***b –*** час, затрачуваний на виконання допоміжних функцій. Оскільки ця формула виражає лінійну залежність від ***n***, складність відповідного алгоритму називають лінійною.

Якщо алгоритм передбачає виконання двох незалежних циклів, перший від ***1*** до ***k***, і другий від ***1*** до ***t***, то загальна кількість кроків буде ***k + t***, а час роботи ***T(k + t)***.

Якщо алгоритм передбачає виконання двох вкладених циклів, від ***1*** до ***k*** і від ***1*** до ***t***, то загальна кількість кроків буде ***k * t***, а час роботи ***T(k * t)***. Це є лінійна функція. І вона є найбільш сприятливою для оцінки швидкодії алгоритму.

Чим більша степінь складності алгоритму в залежності від розміру входу (n^2, n^3, n^n…) тим ефективність алгоритму нижча.

Час роботи алгоритму також залежить від подачі вхідних даних. Так для алгоритму сортування «бульбашкою» суттєво як розміщені елементи вхідного масиву. Якщо вони частково відсортовані, то вихід за прапорцем відбудеться швидше. Якщо елементи стоять в різнобій, то алгоритм відпрацює всі цикли повністю.

Складність можна оцінити як стосовно задачі, так і стосовно алгоритму. Оцінками часу роботи алгоритму є максимальний, мінімальний і середній час його виконання. Залежно від використовуваних евристик ці оцінки можуть збігатися або не збігатися.

Складність задачі повинна оцінюватися за реалізаціями правильних алгоритмів. Вона, являє собою верхню межу для часу роботи алгоритму. Але часто можна знайти також і нижню межу. Очевидно, що для виконання якого-небудь виду оброблення n елементів потрібен час, принаймні пропорційний n.

Час роботи в гіршому випадку є більш цікавим для дослідження ефективності алгоритму, оскільки:

1) це гарантує, що виконання алгоритму закінчиться за деякий час, незалежно від розміру входу;

2) на практиці «погані» входи трапляються частіше;

3) час роботи в середньому є достатньо близьким до часу роботи в гіршому випадку.

Розглянемо три позначення асимптотики росту часу роботи алгоритму.

Q-позначення (тета)

Наприклад, якщо час роботи $T(n)$ деякого алгоритму становить залежність n^2 від розміру входу, то знайдуться такі константи $c_1, c_2 > 0$ і таке число n_0 що $c_1 n^2 \le T(n) \le c_2 n^2$ при всіх $n \ge 0$. Взагалі, якщо $g(n)$ – деяка функція, то запис $f(n) = Q(g(n))$ *(тета)* означає, що знайдуться такі $c_1, c_2 > 0$ і таке n_0, що $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ для всіх $n > n_0$. Тобто складність алгоритму геометрично розміщена в межах умовної смуги між функціями $c_1 g(n)$ і $c_2 g(n)$ (рис. 4.1, а).

O-позначення (о)

Запис $f(n) = Q(g(n))$ включає дві оцінки: верхню і нижню. Їх можна розділити. Говорять, що $f(n) = O(g(n))$, якщо знайдеться така константа $c > 0$ і таке число n_0, що $0 < f(n) \le cg(n)$ для всіх $n \ge n_0$ – верхня оцінка, тобто складність

алгоритму не перевищує деякої функції ***cg(n)*** і геометрично знаходиться нижче її графіку (рис. 4.1, б).

Ω-позначення (омега)

Говорять, що ***f(n) = Ω(g(n)),*** якщо знайдеться така константа ***c > 0*** і таке число n_0, що ***0 ≤ cg(n) ≤ f(n)*** для всіх ***n ≥*** n_0 – нижня оцінка, тобто складність алгоритму не менша деякої функції ***cg(n)*** і геометрично знаходиться вище її графіку (рис. 4.1, в).

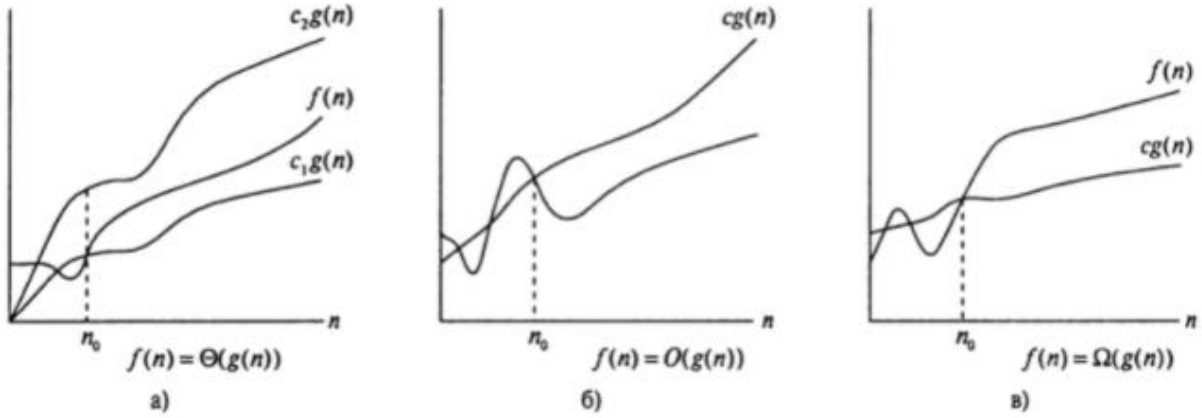


Рисунок 4.1 – Визначення оцінок складності:
а – для f(n) = Θ (g(n)); б – для f(n) = O(g(n)); в – для f(n) = Ω(g(n))

Введені позначення володіють властивостями транзитивності, рефлективності та симетричності.

Транзитивність:

f(n) = Q (g(n)) і g(n) = Q (h(n)) то f(n) = Q (h(n))

f(n) = O(g(n)) і g(n) = O (h(n)) то f(n) = O (h(n))

f(n) = Ω (g(n)) і g(n) = Ω (h(n)) то f(n) = Ω (h(n))

Рефлексивність:

f(n) = Q (f(n)), f(n) = O(f(n)),

f(n) = Ω (f(n)) .

Симетричність:

f(n) = Q (g(n)) якщо і тільки якщо ***g(n) = Q (f(n)).***

Наприклад, алгоритм зчитування й занесення в пам‘ять множини даних має оцінку ***O(n).*** Алгоритм двійкового пошуку в таблиці з упорядкованими елементами оцінюється як ***O (log$_2$ n).*** Просте обмінне сортування має оцінку ***O(n^2)***, а множення матриць – ***O(n^3).*** Коли говорять, що складність сортування рівна ***O(n^2),*** під цим мається на увазі, що сортування інформації буде виконано в найкращому випадку за час, пропорційний ***n^2*** .

Складність розв'язку задачі можна зменшити, якщо знайти більш вигідний метод (як, наприклад, при заміні лінійного пошуку двійковим) або використовувати оптимізуючі евристики. У таблиці 4.1 показана залежність різних видів складності алгоритмів від зростання ***n***. Логарифмічна залежність більш прийнятна, ніж лінійна, а лінійна залежність переважніша за поліноміальну й експонентну.

Таблиця 4.1 – Складність алгоритму

N	O ($\log_2 n$)	O(n)	O ($n \log_2 n$)	O(n^2)	O(2^n)	O (n!)	O (n^n)
1	0	1	0	1	2	1	1
5	2,3219	5	11,6069	25	32	120	3,125
10	3,3219	10	33,2193	100	1,024	$3,6*10^6$	10^{10}
20	4,3219	20	86,4386	400	10^6	$2,4*10^{18}$	10^{26}
30	4,9069	30	147,2067	900	10^9	$2,6*10^{32}$	$2*10^{44}$
100	6,6439	100	664,3856	10000	10^{30}	10^{51}	10^{200}

4.3 Класи складності

У теорії алгоритмів *класами складності називаються множини обчислювальних задач, приблизно однакових за складністю обчислення.* Говорячи більш вузько, класи складності – це множини предикатів (функцій, які отримують на вхід слово і повертають відповідь 0 або 1), що використовують для обчислення ресурси приблизно однакової кількості.

У рамках класичної теорії здійснюється класифікація завдань за класами складності (***P***-складні, ***NP***-складні, експоненціально складні та ін.).

Кожен клас складності (у вузькому сенсі) визначається як множина предикатів, що мають деякі властивості. Типове визначення класу складності має такий вигляд.

Класом складності X називається множина предикатів P(x), що обчислюються на машинах Тьюринга і використовуються для обчислення O (f (n)) ресурсу, де n – довжина слова x.

Як ресурси зазвичай береться час обчислення (кількість робочих тактів машини Тьюринга) або робоча зона (кількість використаних осередків на рядку під час роботи). Мови, які розпізнаються предикатами з деякого класу (тобто множини слів, на яких предикат повертає 1), також називаються мовами, що належать до того самого класу. Класи прийнято позначати прописними літерами. Доповнення до класу *C* (тобто клас мов, доповнення яких належать *C*) позначається *co-C*.

Для кожного класу існує категорія задач, які є «найскладнішими». Це означає, що будь-яка задача з класу зводиться до такої задачі, і до того ж сама задача лежить у класі. Такі задачі називають повними задачами для даного класу. Найбільш відомими є ***NP***-повні задачі. Повні задачі – хороший інструмент для доведення рівності класів. Достатньо для однієї такої задачі подати алгоритм, що розв'язує її і належить більш маленькому класу, і рівність буде доведено.

Розглянемо найбільш відомі класи складності задач.

1. **Клас P** *(задачі з поліноміальною складністю).*

Формальне визначення. Алгоритм ототожнюється з детермінованою машиною Тюрінга, яка обчислює відповідь при даному на вхідний рядок слові із вхідного алфавіту ∑.

Час роботи алгоритму $T_M(x)$ при фіксованому вхідному слові x визначається кількістю робочих тактів машини Тюрінга від початку до зупинки машини.

Складністю функції $f: \sum^* \to \sum^*$, що обчислюється деякою машиною Тюрінга, називається функція $C: N \to N$, що залежить від довжини вхідного слова і дорівнює максимуму часу роботи машини за всіма вхідними словами фіксованої довжини:

$$C_M(n) = \max_{x:|x|=n} T_M(x). \qquad (4.1)$$

Означення. *Якщо для функції f існує машина Тюрінга* ***M*** *така, що*

$C_M(n) < n^c$ *для деякого числа* ***c*** і *досить великих* ***n****, то кажуть, що вона належить класу* ***P****, або* ***поліноміальна за часом****.*

Тобто, задача називається поліноміальною (належить до класу ***P***), якщо існують константа ***k*** та алгоритм, що розв'язує задачу з $F_a(n)=O(n^k)$, де *n* – довжина входу алгоритму в бітах $n = |D|$.

Іноді під класом P мають на увазі більш вузький клас функцій, а саме клас предикатів (функцій $f: \sum^* \to \{0, 1\}$).

У такому разі мовою L, що розпізнає даний предикат, називається множина слів, на яких предикат дорівнює 1. Мовами класу P називаються мови, для яких існують предикати, що їх розпізнають, класу P. Очевидно, що якщо мови L_1 і L_2 належать класові P, то і їх об'єднання, перетин та доповнення також належать класові P.

Задачі класу P – це, інтуїтивно, задачі, розв'язувані за реальний час. Відзначимо такі переваги алгоритмів із цього класу:

- для більшості задач із класу P константа k менше 6;
- клас P інваріантний за моделлю обчислень (для широкого класу моделей);
- клас P має властивість природної замкненості (сума або добуток поліномів є поліном).

Таким чином, задачі класу P є уточненням визначення «практично розв'язної» задачі. Прикладами завдань із класу P є цілочислове додавання, множення, ділення, одержання залишку від ділення, множення матриць, з'ясування зв'язності графів і деякі інші.

2. **Клас NP** (*поліноміальної перевірки*).

Клас NP складається із задач, розв'язання яких можна перевірити протягом поліноміального часу. Мається на увазі, що якщо ми одержимо якимось чином розв'язок деякої задачі цього класу, то протягом часу, який буде поліноміально залежати від розміру вхідних даних задачі , можна перевірити коректність такого розв'язку.

Означення. $\forall D \in D_A$, $|D|=n$ *поставимо у відповідність сертифікат* $S \in S_A$, *такий, що* $|S| = O(n^l)$, *та алгоритм* $A_S = A_S(D, S)$, *такий, що видає «1», якщо розв'язок правильний, і «0» , якщо розв'язок неправильний. Тоді задача належить до класу* NP, *якщо* $F(A_S) = O(n^m)$.

Еквівалентне визначення можна отримати, використовуючи поняття недетермінованої машини Тьюринга (тобто такої машини Тьюринга, у програми якої можуть існувати різні рядки з однаковою лівою частиною).

Означення. *Мова* L *належить до класу* NP *(недетермінованих поліноміальних), якщо вона розпізнається недетермінованою машиною Тюрінга з поліноміальною часовою складністю* $T(n)$.

Отже, основна відмінність класів P і NP та, що до класу P належить задачі, які можуть бути розв'язані за час, що поліноміально залежить від об'єму початкових даних, за допомогою детермінованої обчислювальної машини (наприклад, машини Тьюринга), а до класу NP – задачі, які можуть бути розв'язані за поліноміально виражений час за допомогою недетермінованої обчислювальної машини, тобто машини, наступний стан якої не завжди однозначно визначаєть-

ся попередніми. Роботу такої машини можна представити як процес, що розгалужується на кожній неоднозначності: задача вважається розв'язаною, якщо хоча б одна гілка процесу прийшла до відповіді. Оскільки кожна детермінована машина Тьюринга може розглядатись як недетермінована, але без вибору варіантів кроків, то клас ***P*** міститься в класі ***NP (P⊆NP***).

Оскільки клас ***P*** міститься в класі ***NP***, приналежність того або іншого завдання до класу ***NP*** часто відображає наше поточне уявлення про способи розв'язання даної задачі й носить неостаточний характер.

3. **Клас NPC** (*NP – повні задачі*)

Неформально задача належить класу ***NPC,*** якщо вона належить класу ***NP*** і є такою самою «складною», як і будь-яка задача із класу ***NP. NP***-повні задачі складають підмножину ***NP***-задач і відрізняються тією властивістю, що всі ***NP***-задачі можуть бути тим або іншим способом зведені до них. З цього виходить, що якщо для ***NP***-повної задачі буде знайдений ***P***-розв'язок, то ***P***-розв'язок буде знайдений для всіх задач класу ***NP***.

Поняття ***NP*** – повноти було введене на початку 1970-х років і ґрунтується на понятті зведення однієї задачі до іншої.

Зведення може бути представлене у такий спосіб: якщо ми маємо задачу 1 та алгоритм, що розв'язує цю задачу, тобто видає правильну відповідь для всіх конкретних проблем, що становлять задачу, а для задачі 2 алгоритм розв'язання невідомий, але якщо ми можемо переформулювати (звести) задачу 2 у термінах задачі 1, то ми розв'язуємо задачу 2.

Таким чином, якщо задача 1 задана множиною конкретних проблем D_{A1}, а задача 2 – множиною D_{A2}, й існує функція f_s(алгоритм), що зводить конкретну постановку задачі 2 (d_2) до конкретної постановки задачі 1(d_1):

$fs\ (d_{(2)} \in D_{A2}) = d_{(1)} \in D_{A1}$, то задача 2 зводиться до задачі 1.

Якщо при цьому $F(f_s) = O(n^k)$, тобто алгоритм зведення належить класу ***P***, то говорять, що задача 1 поліноміально зводиться до задачі 2.

Означення. *Задача належить до класу* ***NPC,*** *якщо виконуються такі дві умови: по-перше, задача повинна належати до класу* ***NP (L є NP),*** *і, по-друге, до неї поліноміально повинні зводитися всі задачі із класу NP (Lx=< p, для кожного* ***Lx є NP).***

Виконання цього означення показане на рис. 4.2.

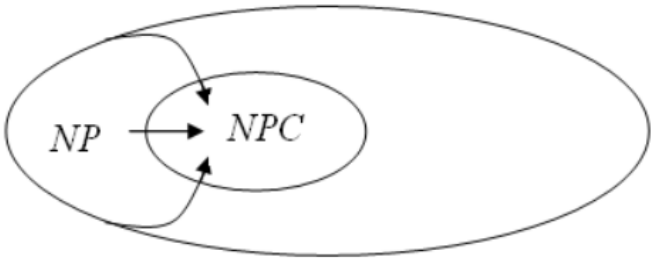


*Рисунок 4.2 – Зведення і клас **NPC***

Теорема 4.1. *Якщо існує задача, що належить до класу **NPC**, для якої існує поліноміальний алгоритм розв'язання **(F = O(n^k))**, то клас **P** збігається із класом **NP**, тобто **P=NP**.*

Схема доведення буде складатися зі зведення будь-якої задачі з ***N***P до даної задачі із класу ***NPC*** з поліноміальною трудомісткістю й розв'язання цієї задачі за поліноміальний час (за умовою теореми).

У цей час доведено існування сотень ***NPC*** задач, але для жодної з них поки не вдалося знайти поліноміального алгоритму розв'язання. У цей час дослідники припускають таке співвідношення класів, що наведене на рис. 4.3, де ***P ≠NP*** і задачі із класу ***NPC*** не можуть бути розв'язані (сьогодні) з поліноміальною трудомісткістю.

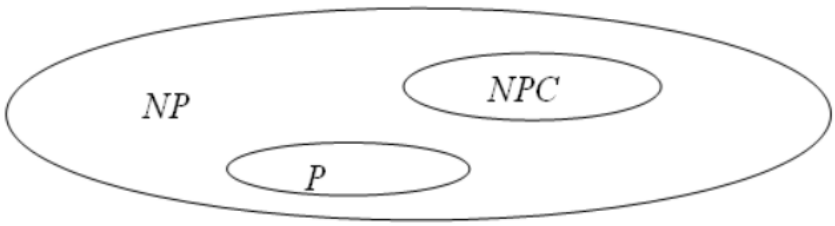


*Рисунок 4.3 – Співвідношення класів **P, NP, NPC***

Прикладами ***NP***-повних задач є, крім задачі про кон'юнктивну форму, задача комівояжера, розфарбування графа, задача про гамільтонів цикл у графі й інші.

4. Інші класи складності

Дослідження складності алгоритмів дозволили по-новому поглянути на розв'язання багатьох класичних математичних задач і знайти для ряду таких задач (множення многочленів і матриць, розв'язання лінійних систем рівнянь та ін.), розв'язання, що вимагають менше ресурсів, ніж традиційні.

Прості класи складності визначаються такими факторами:

- *типом обчислювальної задачі:* найбільш часто використовуються задачі прийняття рішень.
- *моделлю обчислень:* найбільш поширеною моделлю обчислень є детермінована машина Тьюринга, але багато класів складності визначають на основі недетермінованої машини Тьюринга, логічних схем, квантової машини Тьюринга, монотонних схем і т.д.
- *ресурсами, які мають межі:* вказується як "поліноміальний час", "логарифмічний простір", "стала глибина" тощо.

5 АНАЛІЗ АЛГОРИТМІВ ПОШУКУ, ПЕРЕБОРУ, СОРТУВАННЯ

5.1 Найважливіші не обчислювальні алгоритми (пошук і сортування)

Одними з найважливіших процедур оброблення структурованої інформації є сортування і пошук. ***Сортуванням*** називають процес перегрупування заданої послідовності (кортежу) об'єктів в деякому певному порядку. Певний порядок (наприклад, впорядкування в алфавітному порядку, за зростанням або спаданням кількісних характеристик, за класами, типами тощо) в послідовності об'єктів необхідний для зручності роботи з цими об'єктами. Зокрема, однією з цілей сортування є полегшення подальшого пошуку елементів в відсортованому безлічі. Під ***пошуком*** мається на увазі процес знаходження в заданій множині об'єкта, що володіє властивостями або якостями задається апріорі еталона (або шаблону).

Існують різні алгоритми сортування даних (рис. 5.1). І зрозуміло, що не існує універсальна, найкращого в усіх відношеннях алгоритму сортування.

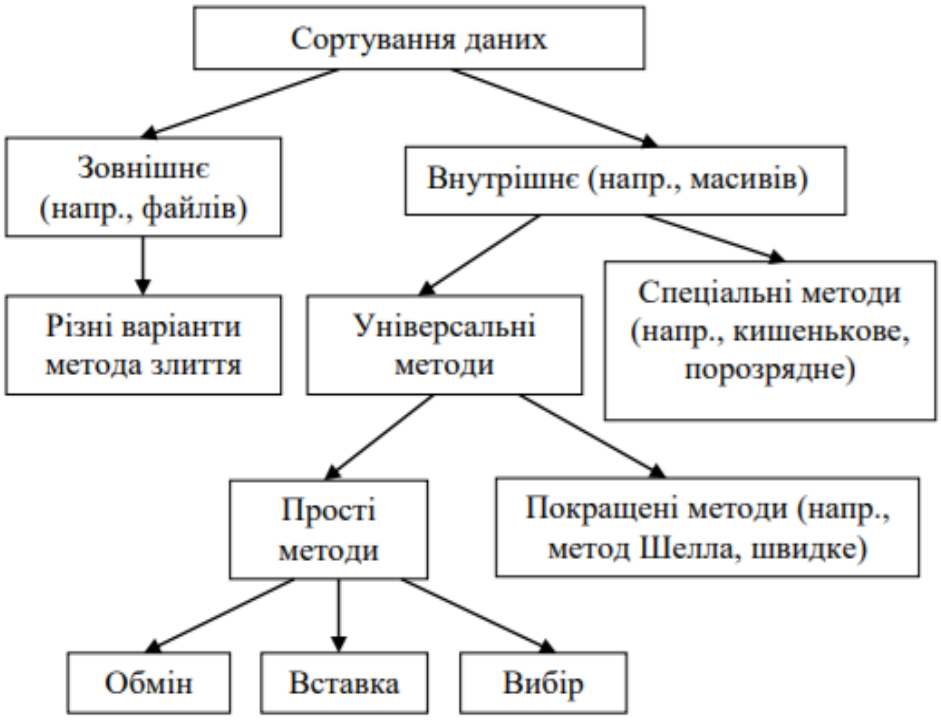


Рисунок 5.1 – Класифікація алгоритмів сортування

Практично кожен алгоритм сортування можна розбити на три основних етапи:

I. Порівняння (впорядкованість пари елементів).

II. Перестановка (зміна місць пари елементів).

III. Перестановка (власне алгоритм сортування).

Алгоритми сортування мають велике практичне застосування в процесі оброблення, збереження великих обсягів інформації. Незважаючи на значну кількість алгоритмів сортування – універсального алгоритму, здатного для вирішення більшої кількості завдань не існує. Вибір алгоритму сортування для конкретного завдання є досить складний процес, який залежить від різних параметрів.

До основних параметрів алгоритмів сортування належить: час сортування, необхідна пам'ять, стійкість, природність поведінки

Час сортування – основний параметр, що характеризує швидкодію алгоритму.

Ефективність алгоритму залежить від безлічі факторів, серед яких можна виділити основні:

- числа елементів, які сортуються;
- ступеня початкового сортування (діапазону і розподілу значень елементів, які сортуються);
- необхідність виключення або додавання елементів;
- доступу до елементів, які сортуються (прямого або послідовного).

Принциповим для вибору методу сортування є останній фактор.

Сортування вставкою

Сортування вставкою – це метод який майже настільки ж простий, що й сортування вибором, але набагато більш гнучкий. Більшість людей при сортуванні колоди гральних карт, використовують метод, схожий на алгоритм сортування вставкою. Суть алгоритму: беремо один елемент і вставляємо його в потрібне місце серед тих, що ми вже обробили (тим самим залишаючи їх відсортованими).

Алгоритм:

1) зліва направо проходимо масив, порівнюючи сусідні елементи, поки не знайдемо елемент, що розташований не в порядку сортування;

2) обмінюємо цей елемент з елементами розташованими ліворуч від нього, поки він не займе потрібну позицію;

3) повторюємо перші дві дії, поки масив не буде відсортовано (рис. 5.2).

```
Процедура СортуванняВставкою(масив a, цілочисельна N)
 Початок
    Змінні цілочисельні i, j, t
    для i від 2 до N
    початок
        t = a[i]
        j = i-1
        поки j > 0 та t < a[j]
            початок
                a[j+1] = a[j]
                j = j-1

                        кінець
                    a[j+1] = t
                кінець
            Кінець
```

Рисунок 5.2 – Псевдокод процедури сортування вставкою

Бульбашкове сортування (сортування простими обмінами)

Алгоритм працює таким чином – у масиві порівнюються два сусідні елементи. Якщо один з елементів, не відповідає критерію сортування (є більшим, або ж, навпаки, меншим за свого сусіда), то ці два елементи міняються місцями. Прохід списком продовжується до тих пір, доки дані не будуть відсортованими.

Алгоритм (рис. 5.3) отримав свою назву від того, що процес сортування за ним нагадує поведінку бульбашок повітря у резервуарі з водою. Оскільки для роботи з елементами масиву він використовує лише порівняння, це сортування на основі порівнянь.

Бульбашкове сортування є не найефективнішою, особливо для послідовностей, у яких «спливаючі» елементи знаходяться в крайній правій стороні. У поліпшеною (швидкої) бульбашкового сортування пропонується проводити перестановки на великі відстані, причому рухатися з двох сторін. Ідея алгоритму полягає в порівнянні елементів, з яких один береться зліва (i = 1), інший-праворуч (j = n). Якщо a [i] <= a [j], то встановлюють j = j - 1 і проводять наступне порівняння. Далі зменшують j до тих пір, поки a [i]> a [j]. В іншому випадку міняємо їх місцями і встановлюємо i = i + 1. Збільшення і продовжуємо до тих пір, поки не отримаємо a [i]> a [j].

```
Процедура БульбашковеСортування(масив a,
Цілочисельна N)
 Початок
     Змінні цілочисельні i, j, t
     для i від 1 до N
         для j від 1  до N-1
           якщо a[j]>a[j+1] тоді
                початок
                     t = a[j]
                     a[j] = a[j+1]
                     a[j+1] = t
                кінець
 Кінець
```

Рисунок 5.3 – Псевдокод процедури сортування простими обмінами

Сортування вибором

Сортування вибором (рис. 5.4) – можливо найпростіший у реалізації алгоритм сортування. Як і в більшості інших подібних алгоритмів, в основі лежить операція порівняння. Порівнюючи кожен елемент із кожним, і у разі потреби роблячи обмін, метод наводить послідовність до необхідного впорядкованого виду.

```
Процедура СортуванняВибором(масив a, цілочисельна N)
 Початок
     змінні цілочисельні i, j, min, t
     для i від 1 до N-1 //N-розмір масиву
     початок
          min = i
          //цикл знаходження мінімального елементу
          для j від i + 1 до N
               якщо a[j] < a[min] тоді
                   min = j


              t = a[min] //заміна елементів
              a[min] =a[i]
              a[i] = t
       кінець
 Кінець
```

Рисунок 5.4 – Псевдокод процедури сортування вибором

Сортування вибором проста в реалізації, і в деяких ситуаціях варто віддати перевагу найбільш складним і досконалим методам. Але в більшості випадків даний алгоритм поступається в ефективності останнім, так як у гіршому, кращому та середньому випадку їй знадобиться $O(n^2)$ часу.

Сортування Шелла

1959 року американський вчений Дональд Шелл опублікував алгоритм сортування, який згодом отримав його ім'я – «Сортування Шелла». Цей алгоритм може розглядатися як узагальнення бульбашкового сортування, і сортування вставками.

Ідея методу (рис. 5.5) полягає у порівняння розділених на групи елементів послідовності, що знаходяться одна від одної на деякій відстані. Спочатку ця відстань дорівнює ***d*** або ***N/2***, де ***N*** – загальна кількість елементів. На першому кроці кожна група включає два елементи розташованих один від одного на відстані ***N/2***; вони порівнюються між собою, і, у разі потреби, змінюються місцями. На наступних кроках також відбуваються перевірка та обмін, але відстань ***d*** скорочується на ***d/2***, і кількість груп відповідно зменшується. Поступово відстань між елементами зменшується, і на ***d = 1*** прохід масивом відбувається востаннє.

```
Shell_Sort(A,N)
 begin
   for k = 1 to m do
       for i = d[k]+1 to N do
        begin
            a = A[i]
            j = i
            while  j-d[k] >= 1 i A[j-d[k]] > a do
             begin
                A[j] = A[j-d[k]]
                j = j - d[k]
                A[j] = a
             end
        end
 end
```

Рисунок 5.5 – Псевдокод процедури сортування Шелла

5.2 Алгоритми пошуку. Послідовний пошук елемента. Бінарний пошук елемента. Послідовний пошук рядка

Пошук – знаходження будь-якої конкретної інформації у великому обсязі раніше зібраних даних. Дані діляться на записи, і кожний запис має хоча б один ключ. Ключ використовується для того, щоб відрізнити один запис від іншого. Метою пошуку є знаходження всіх записів, що підходять до заданого ключа пошуку.

Пошук елемента в масиві

Для знаходження інформації в неупорядкованому масиві потрібен послідовний пошук, що починається з першого елемента й закінчується при виявленні необхідних даних або при досягненні кінця масиву. Цей метод підходить для пошуку неупорядкованої інформації, але також можна використовувати його й на відсортованих даних. Однак якщо дані вже відсортовані, можна застосувати двійковий пошук, що знаходить дані швидше.

Послідовний пошук

Послідовний пошук дуже легко запрограмувати. Наведений нижче псевдокод ілюструє пошук у масиві символів відомої довжини, поки не буде знайдений елемент із заданим ключем

Двійковий пошук

Якщо дані, у яких здійснюється пошук, відсортовані, для знаходження елемента можна застосовувати метод, що набагато перевершує попередній – двійковий пошук.

У двійкового пошуку є й інші назви: дихотомічний пошук, логарифмічний пошук, пошук розподілом навпіл. У ньому застосовується метод половинного розподілу.

Спочатку перевіримо середній елемент. Якщо він більший, ніж шуканий ключ, перевіримо середній елемент першої половини, у протилежному випадку – середній елемент другої половини.

Будемо повторювати цю процедуру доти, поки шуканий елемент не буде знайдений або поки не залишиться чергового елемента.

У двійковому пошуку кількість порівнянь у найгіршому разі дорівнює $\log^2 n$.

У середньому випадку кількість порівнянь значно нижче, а в кращому – дорівнює 1. Двійковий пошук суттєво швидший за лінійний, відносно простий в реалізації і загальновживаний.

Проте, в реальних програмах трапляються випадки помилкового використання лінійного пошуку в упорядкованих даних, що призводять до значного зменшення швидкодії.

5.3 Алгоритми роботи з хеш-таблицями

Хеш-таблиця – це звичайний масив з незвичайною адресацією, що задається хеш-функцією.

Хеш-функція – функція, що трансформує ключ у деякий індекс у таблиці. Ситуація, коли два або більше ключів асоціюються з однією й тією ж коміркою називається колізією при хешуванні. Слід зазначити, що гарною хеш-функцією є така функція, що мінімізує колізії й розподіляє записи рівномірно по всій таблиці. Хеш-таблиці часто застосовуються в базах даних, і, особливо, у мовних процесорах типу компіляторів і асемблерів, де вони вдало обслуговують таблиці ідентифікаторів. У таких додатках, таблиця – найкраща структура даних.

Метод відкритого хешування (інша назва: хешування ланцюжками, рис. 5.6). У випадку хешування ланцюжками, елементи з однаковими індексами об'єднуються в зв'язаний список.

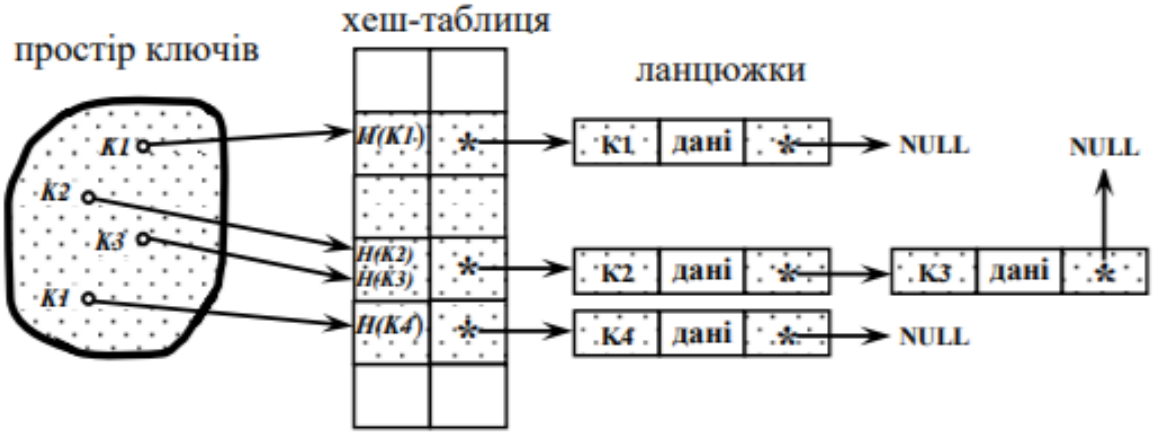


Рисунок 5.6 – Метод відкритого хешування

Хеш-таблиця являє собою узагальнення звичайного масиву. Можливість прямої індексації елементів звичайного масиву забезпечує доступ до довільної позиції в масиві за час *O(1)*. Пряма індексація можлива, якщо ми можемо виді-

лити масив розміру, достатнього для того, щоб для кожного можливого значення ключа існувала окрема комірка.

Закриті хеш-таблиці (рис. 5.7) особливо ефективні, коли максимальні розміри вхідного набору даних вже відомі.

У випадку методу закритого хешування всі елементи зберігаються безпосередньо в хеш-таблиці, без використання зв'язаних списків.

На відміну від хешування з ланцюжками, при використанні методу відкритої адресації може виникнути ситуація, коли хеш-таблиця виявиться повністю заповненою, так що буде неможливо додавати в неї нові елементи. Так що при виникненні такої ситуації рішенням може бути динамічне збільшення розміру хеш-таблиці, з одночасною її перебудовою.

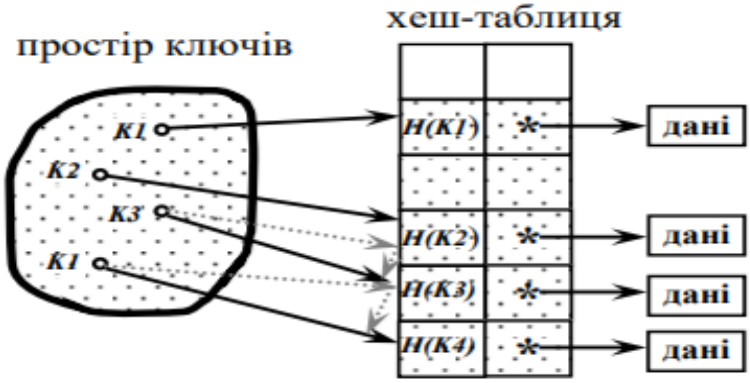


Рисунок 5.7 – Закриті хеш-таблиці

5.4 Інші види алгоритмів

Пошук і сортування є класичними завданнями теорії оброблення даних, вирішують ці завдання за допомогою безлічі різних алгоритмів.

Лінійний пошук. Процедура полягає в простому послідовному перегляді всіх елементів масиву і порівнянні їх з еталоном X.

Пошук розподілом навпіл. У більшості випадків процедура пошуку застосовується до упорядкованих даних (телефонний довідник, бібліотечні каталоги та ін.). У подібних ситуаціях ефективним алгоритмом є пошук розподілом навпіл. У цьому методі порівняння еталона X здійснюється з елементом, розташованим у середині масиву і в залежності від результату порівняння (більше або менше) подальший пошук проводиться в лівій або в правій половині масиву.

Важливою умовою сортування масиву великого обсягу є економне використання доступної пам'яті. У прямих методах сортування здійснюється прин-

цип перестановки елементів «на тому ж місці». Нижче розглянемо три групи угруповань: за допомогою включення, пошуку і обміну.

Сортування за допомогою включення. Елементи подумки діляться на вже «готову послідовність» і неправильно розташовану послідовність. Тепер на кожному кроці, починаючи з i = 2, з неправильно розташованої послідовності витягується черговий елемент і перекладається в готову послідовність на потрібне місце.

Сортування за допомогою прямого вибору. Алгоритм прямого вибору є одним з поширених в силу своєї простоти. Спочатку визначають мінімальний елемент серед усіх елементів масиву, потім його міняють місцями з першим. Далі процес повторюється з тією лише різницею, що мінімальний шукається з другого і змінюється з другим і т. д.

Сортування за допомогою обмінів. Характерною рисою алгоритмів сортування за допомогою обміну є обмін місцями двох елементів масиву після їх порівняння між собою. У так званій «бульбашкового сортування» проводять кілька проходів по масиву, в кожному з яких повторюється одна і та ж процедура: порівняння двох послідовно розташованих елементів і їх обмін місцями в порядку меншини (старшинства) Подібна процедура зрушує найменші елементи до лівого кінця масиву.

6 АЛГОРИТМИ ПОШУКУ НАЙКОРОТШОГО ШЛЯХУ

У теорії графів, задача про найкоротший шлях полягає в знаходженні такого шляху між двома вершинами (або вузлами) графу, що сума ваг ребер з яких він складається мінімальна. Прикладом може бути знаходження найкоротшого шляху між двома пунктами на дорожній мапі; в цьому випадку, вершинами є пункти, а ребрами – відтинки дороги із вагами, рівними часу, необхідному для подолання цього відрізку.

Формально, маємо зважений граф (це набір вершин V і ребер E з дійсно-значимою функцією ваги $f : E \to R$), і заданим елементом v з V, знайти шлях P з v до v' з V такий, що

$$\sum_{p \in P} f(p)$$

найменша серед усіх шляхів, що зв'язують v з v' .

Така задача іноді згадується як Задача про найкоротший шлях між парою вершин, щоб відрізнити від наступних узагальнень:

- задача про найкоротші шляхи з одного входу, тут ми маємо знайти найкоротші шляхи між вхідною вершиною v та всіма іншими вершинами графа;
- задача про найкоротші шляхи з одним виходом, тут ми маємо знайти найкоротші шляхи з усіх вершин графа до однієї вихідної v. Може бути зведена до задачі з одним входом шляхом зміни на зворотні ребер графа;
- задача про найкоротші шляхи для всіх пар, тут ми маємо знайти найкоротші шляхи між кожною парою вершин v, v' в графі.

Ці узагальнення мають значно дієвіше алгоритми ніж спрощений підхід із запуском алгоритму пошуку найкоротшого шляху між всіма значимими парами вершин.

Найважливіші алгоритми для розв'язання цієї задачі:

- алгоритм Дейкстри розв'язує задачі з однією парою, одним входом і одним виходом;
- алгоритм Беллмана-Форда розв'язує задачі з одним входом, якщо ваги ребер можуть бути від'ємні;
- алгоритм пошуку A* розв'язує задачу для однієї пари із використанням евристики в спробі пришвидшити пошук;
- алгоритм Флойда-Воршелла розв'язує задачу для всіх пар;
- алгоритм Джонсона розв'язує задачу для всіх пар, і може бути швидшим за алгоритм Флойда-Воршелла на розріджених графах;
- теорія збурень знаходить (в найгіршому випадку) локально найкоротший шлях.

6.1 Алгоритм Дейкстри

Алгоритм Дейкстри – алгоритм на графах, відкритий Дейкстрою. Знаходить найкоротший шлях від однієї вершини графа до всіх інших вершин. Класичний алгоритм Дейкстри працює тільки для графів без циклів від'ємної довжини.

Алгоритм Дейкстри – задача про найкоротший шлях полягає в знаходженні найкоротшого шляху від заданої початкової вершини a до заданої вершини z.

Наступні дві задачі – безпосередні узагальнення сформульованої задачі про найкоротший шлях:

1. Для заданої початкової вершини a знайти найкоротші шляхи від a до всіх інших вершин.

2. Знайти найкоротші шляхи між усіма парами вершин.

Виявляється що майже всі методи розв'язання задачі про найкоротший шлях від заданої початкової вершини ***a*** до заданої вершини ***z*** також дають змогу знайти й найкоротші шляхи від вершини ***a*** до всіх інших вершин графа.

Отже, за їх допомогою можна розв'язати задачу 1 із невеликими додатковими обчислювальними витратами.

З іншого боку, задачу 2 можна розв'язати або n разів застосувавши алгоритм задачі 1 із різними початковими вершинами, або один раз застосувавши спеціальний алгоритм.

Найефективніший алгоритм визначення довжини найкоротшого шляху від фіксованої вершини до будь-якої іншої запропонував 1959 р. датський математик Е. Дейкстра. Цей алгоритм застосований лише тоді, коли вага кожного ребра (дуги) додатна.

Опишемо докладно цей алгоритм для орієнтованого графа.

Нехай ***G = (V, E)*** – зважений орієнтований граф, ***w(vi, vj)*** – вага дуги ***(vi, vj).*** Почавши з вершини ***a***, знаходимо віддаль від a до кожної із суміжних із нею вершин. Вибираємо вершину, віддаль від якої до вершини a найменша; нехай це буде вершина ***v**.*** Далі знаходимо віддалі від вершини a до кожної вершини суміжної з ***v**** вздовж шляху, який проходить через вершину ***v**.*** Якщо для якоїсь із таких вершин ця віддаль менша від поточної, то замінюємо нею поточну віддаль. Знову вибираємо вершину, найближчу до ***a*** та не вибрану раніше; повторюємо процес.

Описаний процес зручно виконувати за допомогою присвоювання вершинам міток. Є мітки двох типів: тимчасові та постійні. Вершини з постійними мітками групуються у множину ***M***, яку називають множиною позначених вер-

шин. Решта вершин має тимчасові мітки, і множину таких вершин позначимо як ***T, T = V \ M***.

Позначатимемо мітку (тимчасову чи постійну) вершини ***v*** як ***l(v).*** Значення постійної мітки ***l(v)*** дорівнює довжині найкоротшого шляху від вершини ***a*** до вершини ***v***, тимчасової – довжині найкоротшого шляху, який проходить лише через вершини з постійними мітками.

Алгоритм працює покроково – на кожному кроці він «відвідує» одну вершину і намагається зменшувати мітки.

Робота алгоритму завершується, коли всі вершини відвідані.

Ініціалізація

Метка самої вершини a покладається рівною 0, мітки інших вершин – нескінченності.

Це відображає те, що відстані від ***a*** до інших вершин поки невідомі. Усі вершини графа позначаються як невідвідані.

Крок алгоритму

Якщо все вершини відвідані, алгоритм завершується.

В іншому випадку, з ще не відвіданих вершин вибирається вершина ***u***, що має мінімальну позначку.

Ми розглядаємо різні маршрути, в яких ***u*** є передостаннім пунктом. Вершини, в які ведуть ребра з ***u***, назвемо *сусідами* цієї вершини. Для кожного сусіда вершини ***u***, крім позначених відвідані, розглянемо нову довжину шляху, що дорівнює сумі значень поточної мітки ***u*** і довжини ребра, що з'єднує ***u*** з цим сусідом.

Якщо отримане значення довжини менше значення мітки сусіда, замінимо значення мітки отриманим значенням довжини. Розглянувши всіх сусідів, позначимо вершину ***u*** як відвідану і повторимо крок алгоритм.

6.2 Алгоритм Флойда–Уоршелла

Алгоритм Флойда-Уоршелла є алгоритмом на графах, розроблений в 1962 Робертом Флойдом і Стівеном Уоршеллом. Він служить знаходження найкоротших шляхів між усіма парами вершин графа.

Алгоритм Флойда-Уоршелла зазвичай надає тільки довжини шляхів між усіма парами вершин. За допомогою простих модифікацій можна створити ме-

тод відновлення фактичного шляху між будь-якими двома вершинами кінцевої точки.

Замість цього дерево найкоротших шляхів може бути обчислено для кожного вузла за час, використовуючи пам'ять для зберігання кожного дерева, що дозволяє нам ефективно реконструювати шлях з будь-яких двох пов'язаних вершин.

Алгоритм Флойда-Уоршелла є ефективним для розрахунку всіх найкоротших шляхів в щільних графах, коли має місце велика кількість пар ребер між парами вершин. У разі розріджених графів з ребрами невід'ємної ваги кращим вибором вважається використання алгоритму Дейкстри для кожного можливого вузла.

Метод Флойда безпосередньо ґрунтується на тому факті, що у графі з позитивними вагами ребер будь-який неелементарний (що містить більше 1 ребра) найкоротший шлях складається з інших найкоротших шляхів.

Цей алгоритм є більш загальним у порівнянні з алгоритмом Дейкстри, оскільки він знаходить найкоротші шляхи між будь-якими двома вершинами графа.

В алгоритмі Флойда використовується матриця розміром ***nxn***, в якій обчислюються довжини найкоротших шляхів. Елемент ***A[i, j]*** дорівнює відстані від вершини і до вершини j, яка має кінцеве значення, якщо існує ребро ***(i,j***), і дорівнює нескінченності в іншому випадку.

Алгоритм Флойда

Основна ідея алгоритму. Нехай є три вершини ***i, j, k*** та задані відстані між ними. Якщо виконується нерівність ***A[i,k]+A[k,j]<A[i,j***], то доцільно замінити шлях ***i->j*** шляхом ***i->k->j***. Така заміна виконується систематично у процесі виконання цього алгоритму.

7 ЙМОВІРНІСНІ АЛГОРИТМИ; ГЕНЕРАТОРИ ВИПАДКОВИХ ЧИСЕЛ

RANDOMIZED ALGORITHM – це алгоритм, який використовує елемент випадковості як частину своєї логіки. Алгоритм зазвичай використовує рівномірно випадкові біти як допоміжний вхід для спрямування своєї поведінки в надії досягнення хорошої швидкодії в *середньому* серед усіх можливих виборів випадкових бітів.

Випадковість – це фактор, який визначає результат експерименту з множини можливих результатів, відомих заздалегідь. Тобто ми вважаємо, що експеримент має випадковий результат, якщо його кінцевий результат не може бути визначений апріорі з множини можливих варіантів.

Початок якісної теорії імовірнісних алгоритмів було покладено в 1956 році, коли вперше було встановлено, що за допомогою імовірнісних алгоритмів можна обчислити в точності ті ж функції, що і за допомогою звичайних, детермінованих алгоритмів.

У 1974 році було показано, що можна побудувати таку мову L і функцію $t(x)$, що для будь-якого $\varepsilon>0$ існує імовірнісна машина Тьюрінга, що розпізнає L з ймовірністю 1- ε за час $t(x)$ і якщо $t'(x)$- час роботи детермінованою машини Тьюрінга, яка розпізнає L, то для нескінченної кількості x виконується $t'(x)>t(x)$

7.1 Методи Монте-Карло (ММК)

Методи Монте-Карло (ММК) – група чисельних методів для вивчення випадкових процесів.

Сутність методу полягає в наступному: процес описується математичною моделлю з використанням генератора випадкових величин, модель багаторазово обраховується, на основі отриманих даних обчислюються імовірнісні характеристики даного процесу.

Наприклад, щоб дізнатися методом Монте-Карло, яке в середньому буде відстань між двома випадковими точками в колі, потрібно взяти координати великого числа випадкових пар точок в межах заданої окружності, для кожної пари обчислити відстань, а потім для них порахувати середнє арифметичне.

Методами Монте-Карло називають будь-яку статистичну процедуру, яка використовує статистичну вибірку. Застосуємо цю процедуру, для знаходження **розв'язку задачі комівояжера**. Для цього припустимо, що у задачі ко-

мівояжер повинен проїхати ***n*** міст, і побудувати свій маршрут таким чином, щоб побувати у кожному місті по одному разу і повернутися у початкове. Відмітимо, що, при цьому, витрати на переміщення між містами повинні бути мінімальними.

Задача комівояжера є класичною задачею дискретної оптимізації і має численні додатки: транспортні, логістичні задачі, завдання з'єднання пунктів лінією електропередач та інше.

Отже, для розв'язання задачі комівояжера методом Монте-Карло, використовується датчик випадкових чисел. ***стандартна функція Random, яка дозволяє програмно формувати випадкові числа.***

Методи використовуються для вирішення завдань в різних областях фізики, хімії, математики, економіки, оптимізації, теорії управління та ін.

Очевидно, перше завдання, пов'язане з проблемою комівояжера, було поставлено і вирішено Леонардом Ейлером у 1757 р.

У цьому випадку йдеться не про пошук найкоротшого шляху, а про факт встановлення можливості існування такого шляху, що проходить через усі вершини графа один і лише один раз. Для довільного графа відповідну проблему сформулював Вільям Гамільтон у 1859 р. Він вигадав головоломку «Подорож навколо світу», пов'язану з обходом вершин додекаедра (вище, праворуч) і навіть випустив її у продаж.

Наведемо невелику класифікацію варіантів завдання:

Симетрична проблема комівояжера (TSP = traveling salesman problem) у якій відстані задані між будь-якими двома містами і матриця відстаней симетрична: ***Dji = Dij.***

Асиметрична проблема комівояжера (ATSP) допускає несиметричність матриці ***Dji ≠ Dij.*** У ще більш загальному випадку, шляхи між деякими містами можуть бути відсутніми (== мати нескінченну довжину). може бути кілька).

Пошук циклу Гамільтона (HCP = namiltonian cycle problem) – виявлення в довільному графі замкнутих шляхів, що проходять через кожну вершину в точності один раз.

Завдання комівояжера відноситься до класу ***NP***-важких і не відомий алгоритм, який дозволить гарантовано її вирішити за поліноміальний за кількістю міст ***N*** час ***T ~ Nk***, ***k*** = const. Тим не менш, для "оглядної" кількості міст існує безліч способів вирішення:

1. **Точні методи**

Ті, які знаходять деякий розв'язок, а й під час закінчення своєї роботи доводять, що цей розв'язок – найкращий. Зазначимо такі:

Повний перебір перестановок N-1 чисел (стартове місто фіксований). Практично некорисний при N > 15.

Спрямований пошук з поверненнями – перебір варіантів "навколо" деякого розв'язку з відсіканням шляхів, що мають довжину більшу, ніж кращий до поточного моменту шлях.

Метод гілок і кордонів – найбільш ефективний із відомих метод відсікання "неперспективних" вузлів, за рахунок аналізу матриці відстаней. При пошуку оптимального розв'язку будується бінарне дерево (у кожному вузлі породжуються 2 гілки: комівояжер йде в деяке місто або не йде в нього).

Лінійне програмування застосовується для мінімізації (з обмеженнями) лінійної форми d x, де x { {1}} – бінарний вектор розмірності N(N-1)/2, компоненти якого x_i рівні 1 або 0, залежно від того, входить і-е ребро в дорогу чи ні. Вектор d (тієї ж розмірності) дорівнює довжинам ребер.

2. **Евристичні методи**

Зазвичай, значно швидші від точних, але вони не гарантують оптимальності знайденого розв'язку. Результат їх комбінації може далі використовуватися як перше наближення для подальшого покращення, наприклад, за допомогою пошуку з поверненням:

Жадібний алгоритм при виборі чергового міста бере найближче не відвідане місто.

Метод шнурка – геометрична варіація жадібного алгоритму, у якій міста охоплюються замкнутим контуром. Він поступово розтягується, намагаючись пройти через усі міста, мінімальним образом збільшивши свою довжину.

Ковзний перебір переставляє місцями міста з невеликої частини шляху. Потім таке "вікно перебору" ковзає вздовж усього шляху. Метод має різні варіації та виявляється ефективним способом покращення розв'язку, знайденого попередніми двома евристичними методами.

3. **Імовірнісні методи**

Фактично ніколи не зупиняються, здійснюючи випадкові зміни шляху в очікуванні отримання більш короткого. З цього класу методів відзначимо:

Метод відпалу, в якому відбуваються перестановки міст з поступово "згасаючою" інтенсивністю. У цьому постійно зберігається найкращий знайдений розв'язок.

Генетичний алгоритм – більш "просунутий " варіант, у якому створюється багато різних шляхів. Вони постійно "мутують" і "схрещуються" один з одним, обмінюючись окремими ділянками.

7.2 Розв’язання задачі комівояжера методом Монте-Карло

Методами Монте-Карло називають будь-яку статистичну процедуру, яка використовує статистичну вибірку. Застосуємо дану процедуру, для знаходження **розв'язку задачі комівояжера**.

Для цього припустимо, що у задачі комівояжер повинен проїхати ***n*** міст, і побудувати свій маршрут таким чином, щоб побувати у кожному місті по одному разу і повернутися у початкове. Відмітимо, що, при цьому, витрати на переміщення між містами повинні бути мінімальними.

Отже, для **розв’язання задачі комівояжера методом Монте-Карло**, використовується датчик випадкових чисел.

Тобто, на першому кроці, серед міст з номерами ***2,3,4,..,n*** (місто з номером один є початковим, тому його вибирати не потрібно) випадковим чином вибираємо по одному місту і запам'ятовуємо їх послідовність.

Відмітимо, що в подальшому дана послідовність вважатиметься за оптимальний маршрут. Для цього маршруту розраховуємо функцію мети і також запам'ятовуємо її.

Після цього, процедуру повторюємо. Якщо функція мети не змінилась, або має гірше значення, то результат не враховують. Якщо функція мети має краще значення, то нові, кращі, результати запам'ятовуються, а старі викреслюють.

Відмітимо, що з допомогою ЕОМ ця процедура дозволяє за короткий термін оглянути велику кількість маршрутів і обрати серед них якщо не найкращий, то принаймі не найгірший маршрут.

Завдання комівояжера може бути сформульована як задача на графі в наступній постановці: побудувати граф G (X, A), вершини якого відповідають містам в зоні комівояжера, а дуги відображають комунікації, що з'єднують пари міст.

Нехай довжина a (x, y) ≥0 кожної дуги (x, y) ε A дорівнює відстані, вартості або часу. Контур, що включає кожну вершину графа G хоча б один раз, називається маршрутом комівояжера. Контур, що включає кожну вершину графа G рівно один раз, називається Гамільтона контуром (по імені ірландського математика Вільяма Роуана Гамільтона, який у 1859 р. першим почав вивчення цих завдань).

Загальним завданням комівояжера називається задача пошуку маршруту найменшої загальної довжини.

Завданням комівояжера називається задача пошуку Гамільтона контуру найменшою загальної довжини. Контур комівояжера, що має найменшу дов-

жину, називається оптимальним маршрутом комівояжера і є оптимальним розв’язком загальної задачі комівояжера. Гамильтонов контур найменшої довжини називається оптимальним гамільтоновим контуром і є оптимальним розв’язком задачі комівояжера.

Оптимальний маршрут комівояжера не обов'язково є гамільтоновим контуром.

7.3 Жадібний алгоритм

Жа́дібний алгори́тм – простий і прямолінійний евристичний алгоритм, який набуває найкращого розв’язку, виходячи з наявних на кожному етапі даних, не зважаючи на можливі наслідки, сподіваючись урешті-решт отримати оптимальний розв'язок. Легкий в реалізації і часто дуже ефективний за часом виконання. Багато задач не можуть бути розв'язані за його допомогою.

Наприклад, використання жадібної стратегії для задачі комівояжера породжує такий алгоритм: «На кожному етапі вибирати найближче з невідвіданих міст».

Жадібні алгоритми можна характеризувати як «короткозорі» і «невідновлювані». Вони ідеальні лише для задач з «оптимальною підструктурою». Попри це, жадібні алгоритми найкраще підходять для простих задач. Для багатьох інших задач жадібні алгоритми зазнають невдачі у продукуванні оптимального розв'язку, і можуть навіть видати найгірший з можливих розв'язків. Один з прикладів – алгоритм найближчого сусіднього міста, згаданий вище

Базується на п'яти принципах:

1. Набір можливих варіантів, з яких робиться вибір.
2. Функція вибору, за допомогою якої знаходиться найкращий варіант, який буде додано до розв'язку.
3. Функція придатності, яка визначає придатність отриманого набору.
4. Функція цілі, передає значення розв'язку або частковому розв'язку.
5. Функція розв'язку, яка вказує на те, що кінцевий розв'язок знайдено.

Принцип жадібного вибору

До оптимізаційної задачі можна застосувати принцип жадібного вибору, якщо послідовність локально оптимальних виборів дає глобально оптимальний розв'язок. В типовому випадку доведення оптимальності здійснюється за такою схемою: спочатку доводиться, що жадібний вибір на першому етапі не унеможливлює шляху до оптимального розв'язку: для будь-якого розв'язку є інший, узгоджений із жадібним і не гірший від першого. Далі доводиться, що підзадача,

яка виникла після жадібного вибору на першому етапі, аналогічна початковій, і міркування закінчується за індукцією. Інакше кажучи, за жадібного алгоритму ніколи не переглядаються попередні вибори для здійснення наступного, на відміну від динамічного програмування.

Приклад 7.1. На конференції, щоб відвести більше часу на неформальне спілкування, різні секції рознесли по різних аудиторіях. Вчений із надзвичайно широкими інтересами бажає відвідати декілька доповідей у різних секціях. Відомі початок ***s(i)*** і кінець ***f(i)*** кожної доповіді. Визначити, яку найбільшу кількість доповідей можна відвідати.

Наведемо жадібний алгоритм (рис. 7.1), який розв'язує цю задачу. При цьому вважатимемо, що заявки впорядковано за зростанням часу закінчення. Якщо це не так, то можна відсортувати їх за час ***O(nlog n)***; заявки з однаковим часом закінчення розташовуємо в довільному порядку.

```
Activity-Selector(s,f)
1. n ← length[s]
2. A ← {1}
3. j ← 1
4. for i ← 2 to n
   • do if s_i ≥ f_j
     • then A <- A ∪ {i}
       • j ← i
5. return A
```

Рисунок 7.1 – Жадібний алгоритм, який розв'язує задачу

На вхід алгоритму подаються масиви початків і закінчень доповідей. Множина ***A*** складається з номерів вибраних заявок, а ***j*** – номер наступної заявки. Жадібний алгоритм шукає заявку, що починається не раніше від закінчення ***j***-ї, потім знайдену заявку включає в ***A***, а ***j*** присвоює її номер.

Алгоритм працює за ***O(nlog n),*** тобто за складність сортування, оскільки вибірка має меншу складність ***O(n).*** На кожному кроці вибирається найкращий розв'язок. Покажемо, що результатом буде оптимальний розв'язок.

Доведення. Зауважимо, що всі заявки відсортовано за неспаданням часу завершення. Заявка номер 1 однозначно входить до оптимального розв'язку, (якщо ні, замінимо найпершу заявку в оптимумі нею, гірше від цього не стане). Відкинувши всі заявки, що суперечать першій, отримаємо початкову задачу з меншою кількістю заявок. Розмірковуючи індуктивно, приходимо до оптимального розв'язку.

8 КРИПТОАЛГОРИТМИ

8.1 Класифікація криптографічних алгоритмів

Криптографічний алгоритм, або шифр – це математична формула, що описує процеси шифрування і розшифрування. Щоб зашифрувати відкритий текст, криптоалгоритм працює в сполученні з ключем – словом, числом або фразою (рис. 8.1).

Рисунок 8.1 – Основна схема класифікації криптоалгоритмів

Класифікація криптографічних алгоритмів

1. Тайнопис. Відправник і одержувач роблять над повідомленням перетворення, відомі лише їм двом. Стороннім особам невідомий сам алгоритм шифрування.

2. Криптографія з ключем. Алгоритм впливу на передані дані відомий усім стороннім особам, але він залежить від деякого параметра – "ключа", яким володіють лише відправник й одержувач.

- **Симетричні криптоалгоритми**. Для зашифровки і розшифровки повідомлення використовується один і той же блок інформації (ключ).
- **Асиметричні криптоалгоритми.** Алгоритм такий, що для зашифровки повідомлення використовується один ("відкритий") ключ, а для розшифровки – інший ("закритий").

У залежності від кількості ключів, які застосовуються у конкретному алгоритмі:

- **безключові КА** – не використовують в обчисленнях ніяких ключів;
- **одноключові КА** – працюють з одним додатковим ключовим параметром (якимсь таємним ключем);
- **двухключові КА** – на різних стадіях роботи в них застосовуються два ключових параметри: секретний та відкритий ключі.

У залежності від характеру впливів, що виробляються над даними, алгоритми підрозділяються на:

- **перестановочні** – блоки інформації (байти, біти, більші одиниці) не змінюються самі по собі, але змінюється їх порядок проходження, що робить інформацію недоступною сторонньому спостерігачеві;
- **підстановочні** – самі блоки інформації змінюються за законами криптоалгоритму. Переважна більшість сучасних алгоритмів належить цій групі.

Залежно від розміру блоку інформації криптоалгоритми поділяються на:

- **потокові шифри** – одиницею кодування є один біт. Результат кодування не залежить від минулого раніше вхідного потоку. Схема застосовується в системах передачі потоків інформації, тобто в тих випадках, коли передача інформації починається і закінчується в довільні моменти часу і може випадково перериватися. Найбільш поширеними представниками потокових шифрів являються скремблери;
- **блочні шифри** – одиницею кодування є блок з декількох байтів (зараз 4-32). Результат кодування залежить від усіх вихідних байтів цього блоку. Схема застосовується при пакетній передачі інформації та кодування файлів

8.2 Симетричні криптоалгоритми

Симетричні криптоалгоритми – спосіб шифрування, в якому для шифрування і дешифрування застосовується один і той же криптографічний ключ. До винаходу схеми асиметричного шифрування єдиним існуючим способом було симетричне шифрування. Ключ алгоритму повинен зберігатися в секреті обома сторонами.

Симетричні криптоалгоритми виконують перетворення невеликого (1 біт або 32-128 біт) блоку даних у залежності від ключа таким чином, що прочитати оригінал повідомлення можна тільки знаючи цей секретний ключ.

Симетричні криптоалгоритми діляться на:

– скремблери (*програмні або апаратні реалізації алгоритму, що дозволяють шифрувати побітно безперервні потоки інформації*);

– блокові шифри.(*роблять перетворення блоку вхідної інформації фіксованої довжини і одержують результуючий блок того ж обсягу, але недоступний для прочитання стороннім особам, що не володіють ключем*.)

Схему роботи блочного шифру можна описати функціями:

$$Z = EnCrypt (X, Key) \text{ і } X = DeCrypt (Z, Key)$$

$$Z = E (X, K) \text{ і } X = D (Z, K) \text{ (скорочено)}$$

Ключ Key є параметром блокового криптоалгоритму і являє собою деякий блок двійковій інформації фіксованого розміру. Вихідний (X) і зашифрований (Z) блоки даних також мають фіксовану розрядність, рівну між собою, але необов'язково рівну довжині ключа.

8.3 Асиметричні алгоритми шифрування

Асиметричні криптосистеми – ефективні системи криптографічного захисту даних, які також називають криптосистемами з відкритим ключем. У таких системах для зашифровування даних використовується один ключ, а для розшифровування – інший ключ (звідси і назва – асиметричні). Перший ключ є відкритим і може бути опублікованим для використання усіма користувачами системи, які шифрують дані. Розшифровування даних за допомогою відкритого ключа неможливе. Для розшифровування даних отримувач зашифрованої інформації використовує другий ключ, який є секретним. Зрозуміло, що ключ розшифровування не може бути визначеним із ключа зашифровування.

8.4 Алгоритми RSA-шифрування

RSA (абревіатура від прізвищ Rivest, Shamir та Adleman) – криптографічний алгоритм з відкритим ключем, що базується на обчислювальній складності задачі факторизації великих цілих чисел.

RSA став першим алгоритмом такого типу, придатним і для шифрування, і для цифрового підпису. Алгоритм застосовується до великої кількості криптографічних застосунків.

Алгоритм RSA складається з 4 етапів: генерації ключів, шифрування, розшифрування та розповсюдження ключів

Як виявилося, теорія асиметричного шифрування дозволяє дуже красиво вирішувати ще одну проблему інформаційної безпеки – перевірку справжності автора повідомлення. Для вирішення цієї проблеми за допомогою симетричної криптографії була розроблена дуже трудомістка і складна схема. У той же час за допомогою, наприклад, того ж алгоритму RSA створити алгоритм перевірки дійсності автора та незмінності повідомлення надзвичайно просто.

Але асиметрична криптографія знайшла витончений спосіб дуже значного зниження ризику подібної атаки. Якщо замислитися, то неправильно говорити, що між Вами і Вашим співрозмовником немає гарантованої лінії зв'язку. Наприклад, можна надіслати цей відкритий ключ, підписавши повідомлення своїм електронним підписом.

На сьогоднішній день не існує єдиної мережі розповсюдження відкритих ключів, і справа, як це часто буває, полягає у війні стандартів. Розвиваються кілька незалежних систем, але жодна з них не отримала значної переваги над іншими, який називається "світовим стандартом".

Надійність RSA

Факторизація N призведе до розкриття алгоритму RSA. Не було доведено, що ***немає поліноміального за витратами часу розв'язання задачі знаходження простих факторів великого числа*** (тобто можливо, що в майбутньому буде знайдений алгоритм, який факторизує N досить швидко для розтину RSA).

Однак, незважаючи на постійне просування в алгоритмах факторизації, жоден з них не відповідає критерію поліноміальний по часу, що зробило б проблему RSA такою, яку можливо розв'язати.

9 ЕЛЕМЕНТИ ТЕОРІЇ ГРАФІВ. АНАЛІЗ ОРІЄНТОВАНИХ ГРАФІВ

Об'єктом, який покладено в основу методів планування, є зв'язний орієнтований граф без контурів.

Визначимося з поняттями, які мають відношення до неорієнтованих графів і аналогічним (близьким) поняттям з теорії орієнтованих графів (наведені нижче поняття визначено в дисципліні «Дискретна математика»).

1. Неорієнтовані графи: вершина, ребро, маршрут, ланцюг, простий ланцюг, цикл.

2 Орієнтовані графи (орграфи) – аналог з дискретної математики «бінарне відношення»: вершина, дуга (петля), шлях, контур.

Поняття «вершина» $V = \{v_1, v_2, ..., v_n\}$ – скінчена множина елементів, яка графічно представляється у вигляді точок, в 1 і 2 збігаються.

Поняття «дуга» $E = \{e_1, e_2, ..., e_m\}$ – скінчена множина векторів (v_i, v_j) або впорядкованих пар з множини V, графічно представляється у вигляді стрілок (в представленні 1 – це ребро або неупорядкована пара $< v_i, v_j >$.

Поняття «шлях» – така послідовність дуг, в якій кожні дві сусідні дуги мають загальну вершину (графічно – це ланцюг дуг, який пов'язує (з урахуванням напрямку) початкову і кінцеву вершини шляху. Якщо ці вершини збігаються, то такий замкнений шлях називають в орграфі контуром. Довжина шляху або контуру дорівнює числу дуг, які в нього входять (якщо орграф зважений (кожній дузі приписано деяке позитивне ціле число), то довжина шляху дорівнює сумі ваг дуг, які в нього входять).

Зв’язність орграфа: орграф називають зв'язним, якщо між будь-якою парою його вершин існує хоча б один ланцюг, який їх зв’язує (послідовність ребер без урахування їх орієнтації); орграфом називають сильно зв'язним, якщо між будь-якою парою його вершин існує хоча б один шлях, що їх зв'язує.

Надалі нас будуть цікавити як моделі проєкту тільки зв'язні орграфи без контурів. Частково властивості такого орграфа визначаються в поняттях властивостей бінарних відношень (орграф як бінарне відношення антисиметричний і антирефлексивний). Крім цього, в такому графі встановлюється відношення «строгий порядок» між шарами вершин орграфа (що таке шари вершин, шари дуг і алгоритми розбиття відповідних множин на шари, буде розглянуто нижче).

9.1 Розбиття вершин (дуг) зв'язного орграфа без контурів на шари

Необхідно множину вершин орграфа розбити на такі неперетинні підмножини (шари) і так їх упорядкувати, щоб:

- усі елементи шару не мали предків в своєму і наступних після нього шарах (або не мали нащадків в своєму і попередніх шарах); графічно це означає, що дуги, які закінчуються в вершинах, що належать одному шару, повинні обов'язково починатися в попередніх шарах (або дуги, які починаються в вершинах, що належать одному шару, повинні обов'язково закінчуватися в наступних шарах);

- порядок вершин всередині одного шару довільний (вершини одного шару не з'єднані дугами).

Примітки:

1. Елементи першого (0-го) шару не мають предків (немає дуг, які закінчуються в вершинах цього шару).

2. Елементи останнього (*n*-го) шару не мають нащадків (немає дуг, які починаються в вершинах цього шару).

Аналогічні визначення можна дати при розбитті дуг на шари (дуги розбивають на шари за принципом «починаються в вершинах одного шару» або «закінчуються в вершинах одного шару»).

Процедуру розбиття множини вершин (дуг) на шари можна виконати, використовуючи різні алгоритми. Одні зручні для графічного виконання процедури вручну, інші – для реалізації в вигляді програми.

9.2 Графічний спосіб

1. На орграфі знаходять вершини без предків (ці вершини не мають вхідних дуг); такі вершини утворюють перший (0-й) шар.

2. Видаляють (закреслюють) ці вершини і дуги, які виходять із них. В орграфі, що залишився, з'являються нові вершини, які відповідають п. 1; такі вершини утворюють другий (1-й) шар.

3. Виконувати п. 2, отримуючи нові шари до тих пір, поки залишаться вершини (вершина) взагалі без дуг. Ці вершини утворюють останній шар.

Примітки:

1. Описаний алгоритм розбиває вершини на шари за принципом: «всі елементи поточного шару не мають нащадків в своєму і попередніх шарах».

2. Якщо потрібно розбити вершини на шари за принципом: «всі елементи поточного шару не мають предків в своєму і наступних шарах», то виконують ту ж послідовність дій, але починають з вершин без нащадків (вони утворюють останній шар) і потім видаляють ці вершини і дуги, що входять в них, формуючи передостанній шар і т. д. до першого шару. Сформовані за таким принципом шари в загальному випадку можуть відрізнятися від шарів, сформованих за описаним алгоритмом (хоча і не обов'язково).

Розглянемо приклад використання графічного способу для розбиття вершин на шари (рис. 9.1).

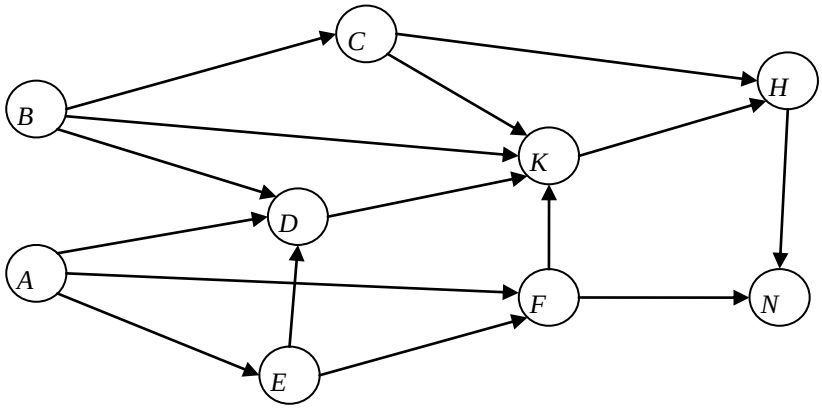


Рисунок 9.1 – Орграф без контурів

Знаходимо вершини за п. 1 (*A*, *B*) – утворюють перший шар і переходимо до п. 2. Після його виконання результат представлено на рис. 9.2

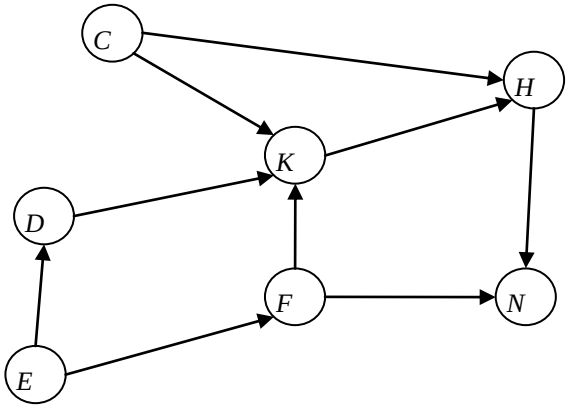


Рисунок 9.2 – Орграф після виконання п.п. 2.2

Знову знаходимо вершини за п. 1 (*C*, *E*) – утворюють другий шар і переходимо до п. 2. Після його виконання результат представлено на рис. 9.3.

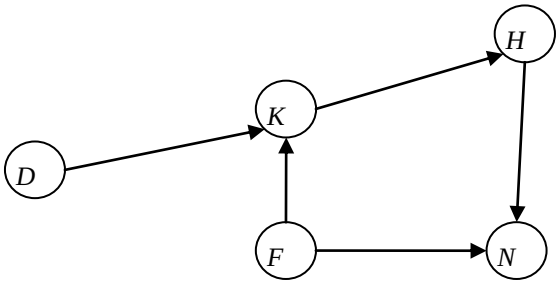


Рисунок 9.3 – Орграф після виконання п.п. 2.2 другий раз

Знову знаходимо вершини за п. 1 (*D*, *F*) – утворюють третій шар і переходимо до п. 2. Після його виконання результат представлений на рис. 9.4.

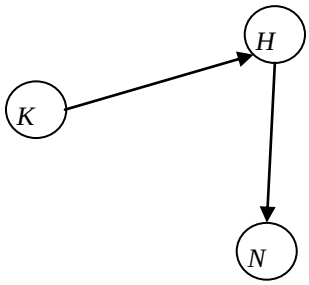


Рисунок 9.4 – Орграф після виконання п.п. 2.2 третій раз

Дії повторюємо ще два рази, після чого залишиться тільки вершина N, яка утворює останній (шостий) шар. Результат буде виглядати так

(*A*, *B*), (*C*, *E*), (*D*, *F*), (*K*), (*H*), (*N*). (9.1)

Дужками в попередньому записі об'єднані вершини одного шару. Наведено варіант лінійного упорядкування вершин орграфа. Інші варіанти можна отримати, змінюючи порядок проходження вершин в будь-якому з шарів. Для розбиття дуг на шари досить скористатися результатом (9.1), замінивши вершини в кожному шарі, назвами дуг (в наведеному прикладі дуги не мали позначень), які виходять з цих вершин.

9.3 Аналітичний спосіб

1. Будують транспоновану матрицю суміжності орграфа (кожна дуга (*V*i, *V*j) орграфа представлена в цій матриці 1 на перетині – стовпець-початок дуги, строка- кінець дуги).

2. Визначають вектор V_0 як суму вектор-стовпців матриці суміжності:

$V_0 = V_A + V_B + V_C + ... + V_N = (0,0,1,3,1,2,4,2,2)^T$; (вершини нульового шару *A*, *B*).

Інші вектор-стовпці визначають так:

$V_1 = V_0 – (V_A + V_B) = (x,x,0,1,0,1,3,2,2)^T$; (Вершини першого шару *C*, *E*).

$V_2 = V_1 – (V_C + V_E) = (x,x,x,0,x,0,2,1,2)^T$; (Вершини другого шару *D*, *F*) і т. д.

Результати розрахунків представлені в табл. 9.1. Розбивка вершин на шари збігається з (1).

Таблиця 9.1 – Транспонована матриця суміжності орграфа з процедурою визначення шарів

$V_j \backslash V_i$	*A*	*B*	*C*	*D*	*E*	*F*	*K*	*H*	*N*	$\boldsymbol{V_0}$	$\boldsymbol{V_1}$	$\boldsymbol{V_2}$	$\boldsymbol{V_3}$	$\boldsymbol{V_4}$	$\boldsymbol{V_5}$
A										0	*x*	*x*	*x*	*x*	*x*
B										0	*x*	*x*	*x*	*x*	*x*
C		1								1	0	*x*	*x*	*x*	*x*
D	1	1			1					3	1	0	*x*	*x*	*x*
E	1									1	0	*x*	*x*	*x*	*x*
F	1				1					2	1	0	*x*	*x*	*x*
K		1	1	1		1				4	3	2	0	*x*	*x*
H			1				1			2	2	1	1	0	*x*
N						1		1		2	2	2	1	1	0
							номер шару			0	1	2	3	4	5
							вершини шару			*A,B*	*C,E*	*D,F*	*K*	*H*	*N*

На рис. 9.5 представлено орграф, який показано на рис. 9.1, з компонуванням вершин по шарам. Проєкт, який представлено у вигляді орграфа з навантаженими ребрами, під дугами має на увазі деякі роботи (вага дуги – час її виконання), а під вершинами – події (початок відповідних робіт для вихідних дуг і закінчення робіт для вхідних дуг). Послідовність виконання робіт зафіксована

в структурі орграфа. У літературі такий орграф називають мережею проєкту. Представлення проєкту у вигляді, як це зображено на рис. 9.5 дозволяє додавати нові роботи, використовуючи нечіткі описи їх місця в проєкті по відношенню до інших робіт, а також виконувати дії, пов'язані з оптимізацією мережі.

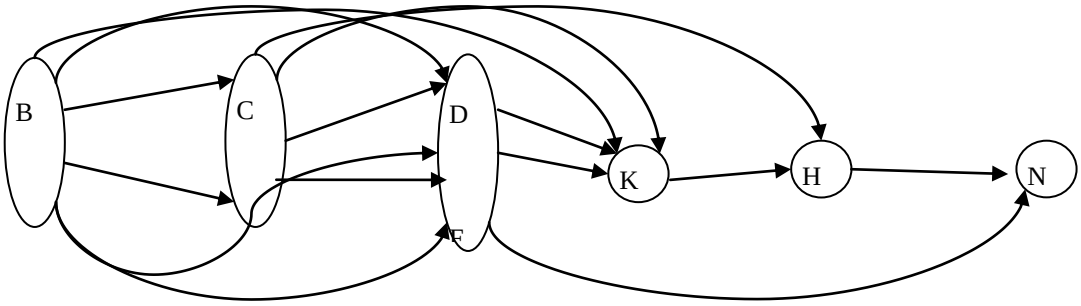


Рисунок 9.5 – Орграф без контурів в новому представленні

10 ТРАНСПОРТНІ МЕРЕЖІ

Транспортна мережа (ТМ, рис. 10.1) – це зважений ациклічний орієнтований граф, з двома особливими вершинами в якому:

- кожній дузі e_i приписано ціле додатне число $C(e_i)$ – пропускна здатність дуги;

- у цьому графі особливими є дві вершини (S та t), при цьому нема дуг, що входять у S і нема дуг, що виходять із t. S називають джерелом, а t – стоком.

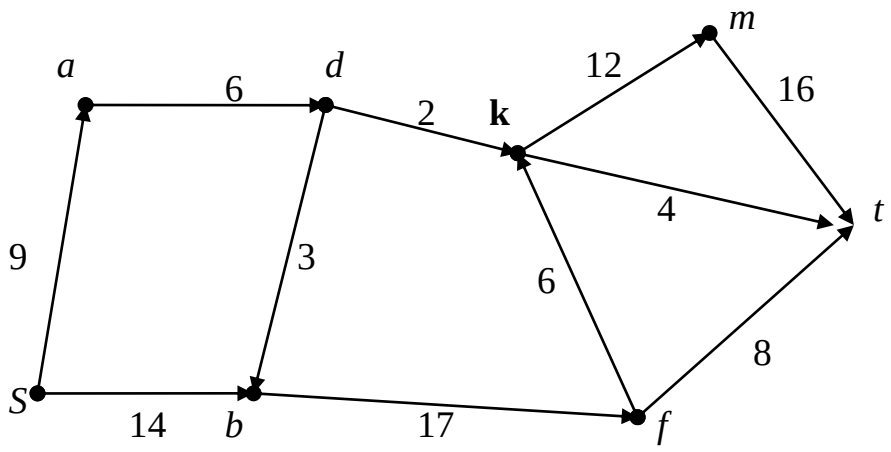


Рисунок 10.1 – Приклад ТМ

ТМ можна давати різні інтерпретації. Наприклад: у вершині S є необмежене джерело деякого однорідного продукту, який потрібно доставити в вершину t через проміжні (внутрішні) вершини. Максимальна пропускна здатність дуг обмежена значеннями $C(e_i)$.

Задача. Організувати переміщення продукту по мережі таким чином, щоб, не перевищуючи пропускних спроможностей дуг, переміщати з S в t максимальну кількість продукту в одиницю часу.

Основне поняття – потік в транспортній мережі – Φ.

Позначимо для кожної вершини V_i через E^+_{Vi} множину всіх дуг, що входять в цю вершину, а через E^-_{Vi} – множину всіх дуг, що виходять з вершини V_i.

Для вершин S і t маємо: $E^+_S = E^-_t = \varnothing = \{\}$ (порожньо).

Функція φ, що визначена на дугах мережі і приймає значення цілих невід'ємних чисел, називається потоком в мережі, якщо виконуються наступні три умови:

1. $\varphi(e_i) \geq 0$, $e_i \in E$ (де E – множина дуг мережі);

2. $\sum_{e_i \in E^+_{\upsilon i}} \varphi(e_j) - \sum_{e_i \in E^-_{\upsilon i}} \varphi(e_j) = 0$ ($\upsilon_i \in V$ и $\upsilon_i \neq S, t$); (10.1)

3. $\varphi(e_i) \leq C(e_i)$.

Умови 1 і 3 визначають функцію як невід'ємне число, яке не перевищує пропускної здатності дуги, а умова 2 – рівняння збереження кількості продукту в кожній внутрішній вершині мережі (скільки прийшло в вершину, стільки і пішло; накопичення немає). Якщо виконати підсумовування по всіх вершинам мережі, то в результаті отримаємо:

$$\sum_{e_i \in E^+_t} \varphi(e_j) - \sum_{e_i \in E^-_S} \varphi(e_j) = 0. \quad (10.2)$$

Для будь-якого потоку кількість продукту, що виходить з джерела в одиницю часу дорівнює кількості продукту, що прибуває в стік. Цю величину і будемо називати величиною потоку в ТМ – Φ

$$\Phi = \sum_{e_i \in E^+_t} \varphi(e_j) = \sum_{e_i \in E^-_S} \varphi(e_j). \quad (10.3)$$

Основна задача: для заданої транспортної мережі знайти потік, що має найбільшу можливу величину (максимальний потік Φ_{max}). Розв'язання такого завдання залежить від конфігурації мережі (виду графа) і максимальної пропускної спроможності кожної дуги.

Для її вирішення розглянемо поняття «розріз мережі». Припустимо $A \in V$ (деяку підмножину вершин мережі, яка володіє властивістю, що $S \in A$, а $t \in \overline{A}$) ($\overline{A} = V \backslash A$).

Розглянемо підмножину всіх дуг, що починаються в A, а закінчуються в $\overline{A}$ (прямі дуги):

$$E_{(A,\overline{A})} = \{(x, y) \mid x \in A, y \in \overline{A}\}. \quad (10.4)$$

Сума пропускних спроможностей прямих дуг, що потрапили в розріз, називають пропускною спроможністю розрізу.

$$C(A, \overline{A}) = \sum_{e_i \in E^+_{(A,\overline{A})}} C(e_j), \quad (10.5)$$

Лемма. Величина потоку в транспортній мережі не може перевищувати пропускної здатності будь-якого розрізу.

$$\Phi \leq C(A, \overline{A}). \quad (10.6)$$

Теорема Форда – Фалкерсона. Максимальна величина потоку в транспортній мережі дорівнює найменшій пропускній здатності всіх можливих розрізів ТМ.

$$\Phi_{\max} = \min_{A} C(A, \overline{A}). \qquad (10.7)$$

10.1 Визначення максимального потоку в заданій ТМ

За леммою для будь-якого потоку і будь-якого розрізу виконується нерівність (10.6), отже:

$$\Phi = \sum_{e_i \in E^{+}_{(A,\overline{A})}} \varphi(e_i) - \sum_{e_j \in E^{-}_{(A,\overline{A})}} \varphi(e_j) \le \sum_{e_i \in E^{+}_{(A,\overline{A})}} \varphi(e_i)$$

$$\le \sum_{e_i \in E^{+}_{(A,\overline{A})}} C(e_i) = C(A, \overline{A}) \qquad (10.8)$$

З співвідношення (10.7) видно, що величина потоку Φ збігається з пропускною спроможністю розрізу $C(A, \overline{A})$ тоді і тільки тоді, коли виконуються дві умови:

а) $\varphi(e_i) = C(e_i)$ для всіх дуг $e_i \in E^{+}_{(A,\overline{A})}$; (10.9)

б) $\varphi(e_j) = 0$ для всіх дуг $e_j \in E^{-}_{(A,\overline{A})}$.

Алгоритм пошуку $\Phi_{\max}$ спрямований на побудову такого розрізу, для якого виконується умова (10.8) При цьому або вдається побудувати такий розріз, або вдається з'ясувати, як розглянутий потік можна збільшити. Збільшення потоку виконується за допомогою збільшувальних ланцюгів, які пов'язують джерело і стік (ланцюг-послідовність дуг, в якій дві сусідніх дуги інцідентні одній вершині і при цьому дуги в ланцюзі не повторюються). Ланцюг можна також представити як послідовність вершин ТМ, в якій будь-які сусідні вершини пов'язані дугою (без урахування її напрямку).

Перша вершина в ланцюзі завжди збігається з джерелом *S*, остання – завжди зі стоком *t*.

Прикладом ланцюга може служити така послідовність вершин : *S, b, d, k, f, t*. Якщо перехід від будь-якої вершини до сусідньої справа вершині збігається

з напрямком дуги, то така дуга називається – пряма дуга, а якщо навпаки – зворотна дуга (приклади прямих дуг (*S*, *b*); (*d*, *k*); (*f*, *t*); приклади зворотних дуг (*b*, *d*); (*k*, *f*)).

Нехай в ТМ задано деякий потік (поруч з пропускною здатністю дуги в круглих дужках вказується потік; наприклад для дуги (*S*, *b*) - 14 (3)). Потік вказують для кожної дуги ТМ, але при цьому обов'язково повинні виконуватися обмеження (10.1) для потоку.

Найбільш простий спосіб – задати у всіх дугах мережі потік рівний 0.

Ланцюг з *S* в *t* називають збільшувальним, якщо для кожної прямої дуги ланцюга виконується нерівність

$$\varphi(e_j) < C(e_i), \tag{10.10}$$

а для кожної зворотної дуги нерівність

$$\varphi(e_j) > 0. \tag{10.11}$$

Припустимо, що для деякого потоку φ в ТМ вдалося знайти збільшувальний ланцюг. Тоді збільшуючи потік на 1 на прямих дугах, зменшуючи на 1 на зворотних дугах і залишаючи незмінним на дугах, що не входять в ланцюг, можна отримати новий потік, величина якого на 1 більша.

10.2 Загальний алгоритм побудови максимального потоку Фmax у ТМ

Мета алгоритму – після чергового збільшення потоку побудувати збільшувальний ланцюг в ТМ (якщо це можливо) і після цього отримати новий, більший у порівнянні з попереднім потік або показати, що побудова збільшувального ланцюга неможлива, й отриманий на попередньому кроці потік є максимальним.

10.3 Пошук збільшувального ланцюга

На кожному кроці цього процесу будь-яка вершина ТМ знаходиться в одному з трьох станів:

а) не позначена;

б) позначена, але не переглянута (наприклад *);

в) позначена і переглянута (наприклад [*]).

Задамо який-небудь (краще нульовий) потік в ТМ.

Крок 1. Позначити вершину *S* (джерело) (вершина *S* в стані б), решта в стані а)). Перейти до кроку 2.

Крок 2. Вибрати будь-яку вершину в стані б) і виявити всі сусідні з нею вершини (вершини ТМ, з якими позначена вершина пов'язана дугами, які входять і виходять з неї). Позначка передається в сусідню вершину, якщо вона знаходиться в стані а) і для прямої дуги виконується умова (9.10), а для зворотного дуги умова (9.11). Якщо не виконується хоча б одна з умов, то сусідня вершина свого стану не міняє. Після виконання всіх дій для обраної вершини відповідно до кроку 2 ця вершина переходить в стан в) (позначена і переглянута). Перейти до кроку 3.

Крок 3. Вибрати будь-яку вершину в стані б) і повторювати крок 2 до тих пір, поки або вершина *t* отримає позначку (перейде в стан б), після цього перейти до кроку 4), або чергове повторення кроку 2 стане неможливим – перейти до кроку 5.

Крок 4. Побудувати збільшувальний ланцюг, починаючи з вершини *t* і рухаючись до вершини *S* тільки по вершинах, які знаходяться в стані в). Змінювати потік в дугах ланцюга циклічно з кроком 1 (в прямих – збільшувати, в зворотних – зменшувати) до тих пір, поки ланцюг перестане бути збільшувальним (порушаться умови (10.10) або (10.11) на одній або декількох дугах – нерівності перетворяться в рівності). Зафіксувати нові потоки в дугах на ТМ. Перейти до кроку 1.

Крок 5. Закінчити процес. Зафіксувати потік в кожній дузі і визначити $\Phi_{\max}$ як потік через будь-який розріз ТМ.

На рисунку 10.2 наведено результат роботи викладеного алгоритму. $\Phi_{\max} = 16$.

Обговорення результатів розрахунку. Припустимо у нас є можливість відкоригувати пропускну здатність тільки однієї дуги. Видно, що слабкою ланкою в цій ТМ є дуга *dk*. Якщо збільшити її пропускну спроможність на 4 од., то $\Phi_{\max} = 20$.

Можна також проаналізувати завантаженість дуг з метою спрощення ТМ. Якщо потік в дузі дорівнює 0 (*db*), то таку дугу можна видалити з ТМ. Крім цього, потік деякої дуги (*kt*) можна перерозподілити між недовантаженими дугами, а саму дугу видалити. Можна також видалити вершини *b*, *d*. Після цього ТМ набере вигляду, як це показано на рис. 10.4.

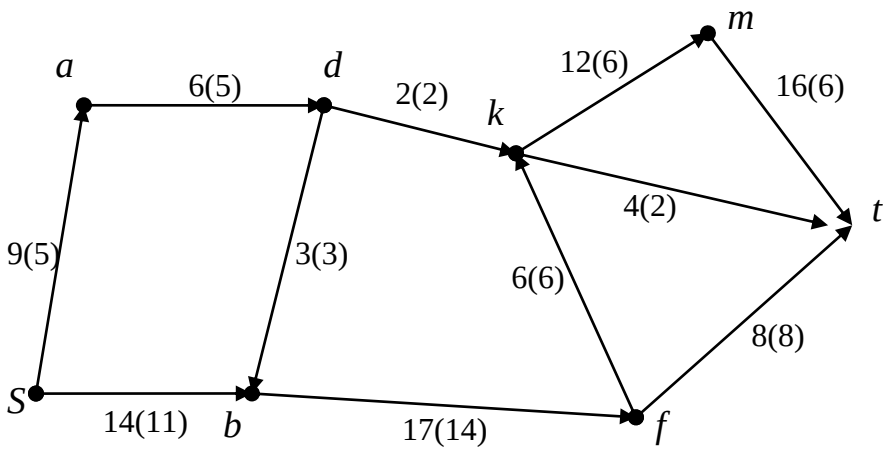


Рисунок 10.2 – Результат роботи алгоритму

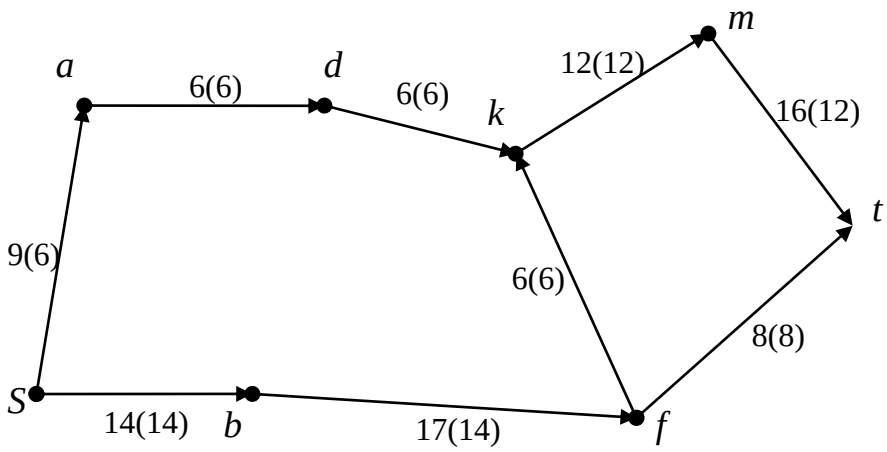


Рисунок 10.3 – Результат корегування ТМ

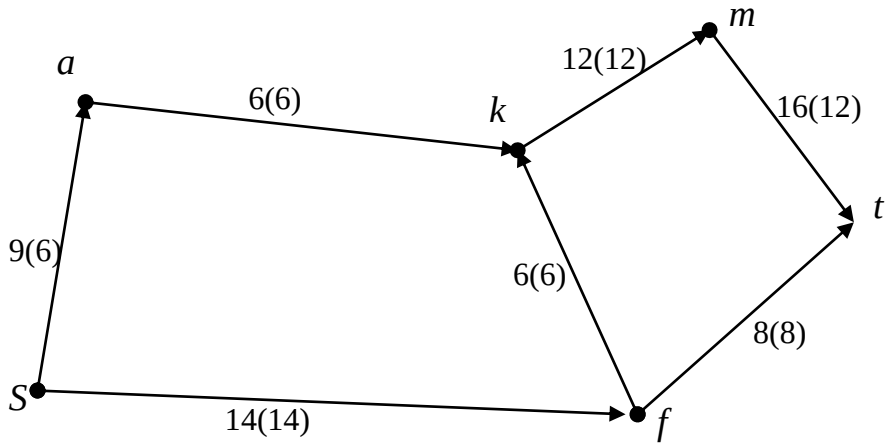


Рисунок 10.4 – Результат кінцевого спрощення ТМ

11 МЕРЕЖЕВА МОДЕЛЬ ПРОЄКТУ ТИПУ І ТА ЇЇ АНАЛІЗ

Розглянемо мережу проєкту типу І «подія-операція» (рис. 11.1), тобто вершини мережі – події (початок одних робіт, дуги яких виходять з вершини і одночасно закінчення інших робіт, дуги яких входять в вершину), дуги, що з'єднують вершини – операції (іншими словами роботи).

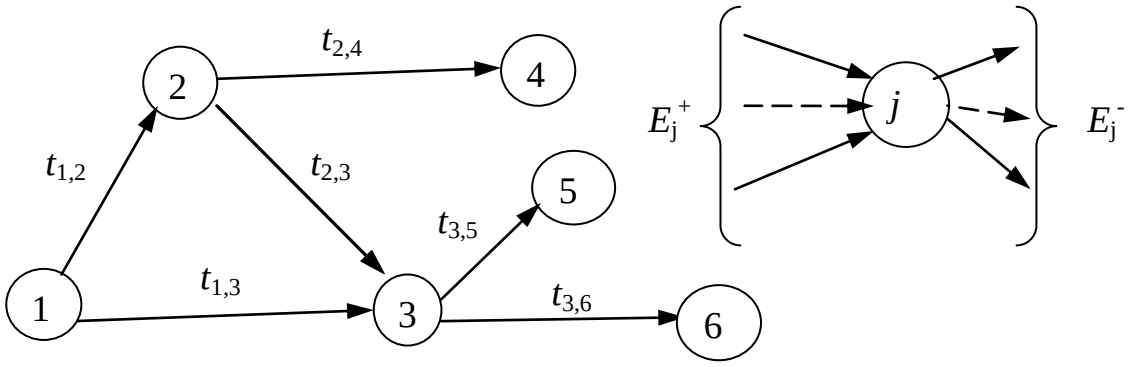


Рисунок 11.1 – Фрагменти мережі з позначеннями

Кожній дузі мережі ставлять у відповідність додатне число t_{ij} - час виконання P_{ij} роботи (номінальне, планове, очікуване), тобто час, який пройде між початком у вершині i і закінченням в вершині j роботи P_{ij}. Надалі буде показано, що кожна вершина мережі має складну структуру в сенсі розподілу подій за часом, які мають до неї відношення. Позначимо E_j^+ – множину робіт, які закінчуються в вершині j; E_j^- – множину робіт, які починаються в вершині j.

Для побудови графіка виконання кожної з робіт з прив'язкою до певної часової шкали, необхідно кожній вершині i мережі приписати деяке число t_i, який відповідає часу, який минув від початку проєкту та в вершині i закінчені всі роботи, з нею пов'язані (дуги, які входять). У цей момент часу (або пізніше) можна починати будь-яку з робіт, які пов'язані з цією вершиною (вихідні дуги). Найбільш пізній час, в який можна почати роботи, пов'язані з вершиною i не впливаючи на загальний час виконання проєкту, позначимо через t_i^*.

11.1 Алгоритм проставляння відміток часу t_i для вершин

1. Знайти вершину мережі, яка має тільки вихідні дуги – початок проєкту (якщо таких вершин кілька – додати ще одну вершину, зв'язати її вихідними дугами з цими вершинами і кожній дузі приписати вагу рівну 0) і приписати їй

значення 0 (початок відліку часу виконання проєкту). Виконати аналогічні дії, якщо в мережі є не одна, а кілька вершин, що мають тільки вхідні дуги (додати вершину і зв'язати її з цими вершинами дугами, що входять, вага яких дорівнює 0, часову позначку ця вершина (закінчення проєкту) отримає в результаті виконання цього алгоритму).

2. Перейти від вершини *j* до вершини *i* суміжній з тою, що отримала позначку (безпосередньо пов'язаною дугою) і обчислити позначку часу (t_i) для цієї вершини за формулою

$$t_i = \max_{E_i^+} \{t_j + t_{ji}\}. \qquad (11.1)$$

3. Перейти до наступної вершини, вибираючи її так, щоб пов'язані з нею дуги, які входять, вже мали часові позначки, і виконати п.2.

4. Закінчити процес проставляння часових відміток, коли всі вершини мережі отримають позначки.

Примітка: часова відмітка t_i, яку отримала вершина мережі, пов'язана із закінченням проєкту і буде мінімальним часом, протягом якого можна реалізувати проєкт (довжиною критичного шляху).

Розглянемо приклад реалізації алгоритму на фрагменті мережі (рис. 11.2).

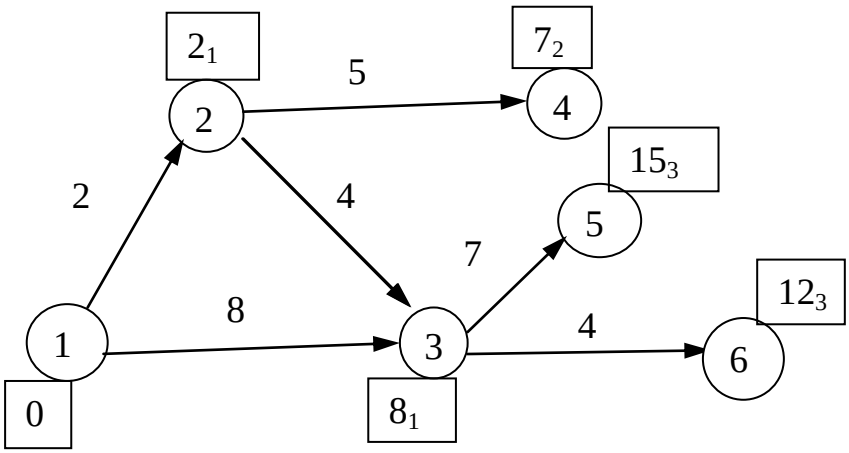


Рисунок 11.2 – Приклад проставляння часових t_i відміток у вершинах

Розрахунок для вершини 3 (як правило, такі розрахунки виконують усно)

$$t_3 = \max_{E_j^+} \{(t_1 + t_{13}); (t_2 + t_{23})\} = \max_{E_j^+} \{(0+8); (2+4)\} = 8. \qquad (11.1)$$

Результати обчислень записують поряд з кожною вершиною мережі (в прямокутнику), додаючи індекс, який вказує вершину *j*, що доставила максимум у формулі (11.1). Ця інформація знадобиться для отримання критичного шляху в мережі проєкту.

Таким чином, часова відмітка вершини вказує час, який минув з початку виконання проєкту і при цьому закінчилися всі роботи, закінчення яких пов'язане з цією вершиною (тільки з цього моменту можна починати роботи, початок яких пов'язаний з цією вершиною). Розглянемо приклад пошуку критичного шляху для мережевої моделі проєкту типу I.

Приклад. Нехай задана мережа типу I «подія-операція» (рис. 11.3). Знайти в цій мережі критичний шлях і визначити його довжину.

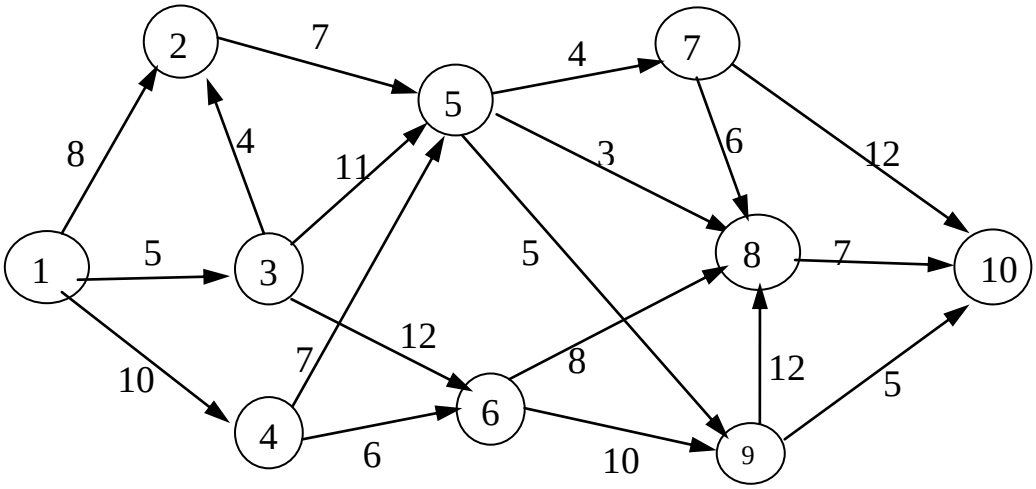


Рисунок 11.3 – Мережа проєкту «подія-операція»

Після реалізації алгоритму кожна вершина отримала часову позначку t_i (число в прямокутнику). Відмітка 46, яку отримала кінцева вершина 10, означає, що довжина критичного шляху (мінімально можливий час виконання всього проєкту) становить 46 одиниць (годин, днів, тижнів або інших одиниць часу).

Послідовність робіт, яка складає критичний шлях в мережі можна відновити, рухаючись від вершини 10 до вершини 1 у відповідності з індексами часових відміток 10-8-9-6-3-1 (рис. 11.4). Іншими словами, послідовність робіт, які визначають час виконання проєкту в 46 од. часу буде така

$$P_{1,3} - P_{3,6} - P_{6,9} - P_{9,8} - P_{8,10}, \qquad (11.3)$$

Перераховані роботи є критичними (керівнику проєкту необхідно стежити за чітким виконанням планових термінів цих робіт, а також початком чергової роботи в критичному ланцюгу відразу ж після закінчення попередньої). Будь-яке збільшення тривалості роботи або декількох робіт з ланцюжка (11.3), наприклад, на 6 од. часу в сумі, відразу збільшують час виконання всього проєкту на стільки

ж. Запізнення з початком чергової роботи з критичного шляху призводить до того ж результату. Критичних шляхів у мережі може бути кілька.

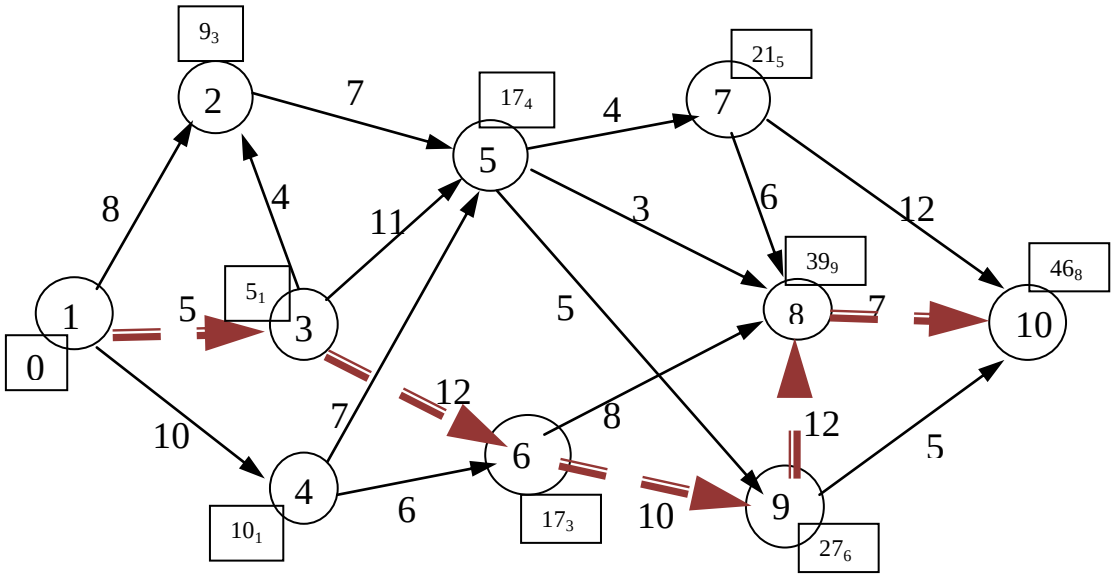


Рисунок 11.4 – Приклад проставлення часових відміток t_i у вершинах мережі

11.2 Алгоритм проставляння відміток часу t_i^* для вершин

Для визначення t_i^* необхідний аналіз всієї мережі. Для вершин 1,3, 6,9,8,10, які знаходяться на критичному шляху $t_i^* = t_i$, для інших вершин (2, 4, 5, 7) $t_i^* > t_i$, тобто утворюється деякий інтервал свободи i-ої події $[t_i; t_i^*]$ (рис. 11.5):

$$t_i^* = \min_{E_i^-} \{t_k^* - t_{ik}\}. \qquad (11.4)$$

Для вершини 7

$$t_7^* = \min_{E_7^-} \{t_{10}^* - t_{7,10}; t_8^* - t_{7,8}\} = \min_{E_7^-} \{46 - 12; 39 - 6\} = 33.$$

Для вершини 5

$$t_5^* = \min_{E_5^-} \{t_8^* - t_{5,8}; t_9^* - t_{5,9}; t_7^* - t_{5,7}\} = \min_{E_5^-} \{39 - 3; 27 - 5; 33 - 4\} = 22.$$

Для вершини 2

$$t_2^* = \min_{E_2^-} \{t_5^* - t_{2,5}\} = 22 - 7 = 15.$$

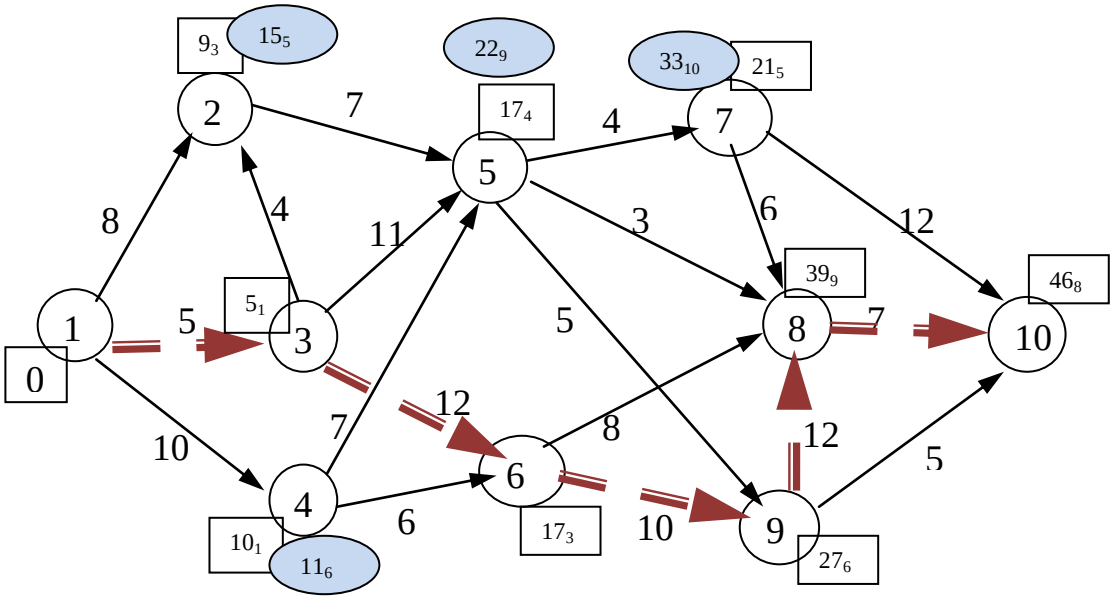


Рисунок 11.5 – Приклад проставляння часових відміток t_i^*

11.3 Часова оцінка операцій і подій

Для часової оцінки операцій (дуги мережі проєкту) будемо використовувати наступні параметри:

– t_{ij} – номінальний (плановий) час, який відводиться на виконання роботи (операції) P_{ij} – на мережі вказується, як вага дуги;

– MS_{ij} – вільний резерв часу для виконання операції (роботи) P_{ij};

$$MS_{ij} = t_j - t_i - t_{ij} = t_j - (t_i + t_{ij}), \qquad (11.5)$$

для операцій, які входять до критичного шляху $MS_{ij} = 0$;

– MP_{ij} – повний резерв часу для виконання операції (роботи) P_{ij};

$$MP_{ij} = t_j^* - t_i - t_{ij} = t_j^* - (t_i + t_{ij}); \qquad (11.6)$$

– MN_{ij} – незалежний резерв часу для виконання операції (роботи) P_{ij};

$$MN_{ij} = \max\{0, (t_j - t_i^* - t_{ij})\}; \qquad (11.7)$$

При необхідності, вказують ці параметри на мережі поруч з вагою дуги в круглих дужках через кому в наведеній вище послідовності: t_{ij} (MS_{ij}, MP_{ij}, MN_{ij}).

Для часової оцінки подій (вершини мережі проєкту) будемо використовувати наступні параметри: t_i, t_i^* і резерв події

$$M_j = t_j^* - t_j. \quad (11.8)$$

Усі роботи, які закінчуються в вершині *i*, повинні закінчитися не пізніше t_i; всі роботи, які починаються в вершині *i*, повинні початися не раніше t_i.

Всі резерви часу для аналізу зручно звести в таблицю (табл. 11.1). Мережу можна перетворити, передаючи частину резерву події-операції і навпаки.

$MP_{ij} = t_j^* - (t_i + t_{ij}) = (7) - (5)$

$MS_{ij} = t_j - (t_i + t_{ij}) = (7(\text{зн. в дужках})) - (5)$,

$MN_{ij} = \max\{0, (t_j - (t_i^* + t_{ij}))\} = \max\{0, (7(\text{зн. в дужках}) - (4(\text{зн.в дужках}) + (3)))\}$

Таблиця 11.1 – Результати аналізу мережевої моделі проєкту у табличному вигляді

Робота P_{ij}	Кількість попер. робіт	Тривалість роботи t_{ij}	Строки виконання робіт				Резерви часу			
			ранні		пізні		робіт P_{ij}			подій
			початок $t_i(t_i^*)$	закінч. $t_i + t_{ij}$	початок $t_j^* - t_{ij}$	закінч. $t_j^*(t_j)$	MP_{ij}	MS_{ij}	MN_{ij}	M_j
1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*
(1,2)	0	8	0	8	7	15(9)	7	1	1	6
(1,3)	0	5	0	5	0	5	***0***	***0***	0	0
(1,4)	0	10	0	10	1	11(10)	1	0	0	1
(3,2)	1	4	5	9	11	15(9)	6	0	0	6
(3,5)	1	11	5	16	11	22(17)	6	1	1	5
(2,5)	2	7	9(15)	16	15	22(17)	6	1	0	5
(3,6)	1	12	5	17	5	17	***0***	***0***	0	0
(4,5)	1	7	10(11)	17	15	22(17)	5	0	0	5
(4,6)	1	6	10(11)	16	11	17	1	1	0	0
(5,7)	3	4	17(22)	21	29	33(21)	12	0	0	12

Продовження таблиці 11.1

1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*
(5,8)	3	3	17(22)	20	36	39	19	19	14	0
(5,9)	3	5	17(22)	22	22	27	5	5	0	0
(6,8)	2	8	17	25	31	39	14	14	14	0
(6,9)	2	10	17	27	17	27	***0***	***0***	0	0
(7,8)	1	6	21(33)	27	33	39	12	12	0	0
(7,10)	1	12	21(33)	33	34	46	13	13	1	0
(8,10)	4	7	39	46	39	46	***0***	***0***	0	0
(9,8)	2	12	27	39	27	39	***0***	***0***	0	0
(9,10)	3	5	27	32	41	46	14	14	14	0

12 МЕРЕЖЕВА МОДЕЛЬ ПРОЄКТУ ТИПУ II ТА ЇЇ АНАЛІЗ

12.1 Оптимізація часу реалізації проєкту при обмежених ресурсах

На етапі попереднього аналізу проєкту важливим параметром є час його реалізації. До цього на мережевий моделі проєкту (в представленні події – вершини, дуги – роботи), кожній дузі ставилося у відповідність її вага (час виконання роботи в одиницях часу). Передбачалося за замовчуванням, що до моменту початку кожної роботи є всі необхідні для її виконання матеріали та ресурси, які дозволять виконати цю роботу за вказаний час (робітники, механізми, транспорт і т. п. – так звані відновлювальні ресурси; після закінчення поточної роботи такі ресурси можуть бути використані для виконання наступних робіт). Мережеву модель, яка враховує наявність відновлювальних ресурсів будемо називати моделлю типу II.

У такій моделі вершини-роботи, а дугами задають порядок виконання робіт (відношення передування) Використовують цю модель в реальних проєктах в яких накладені певні обмеження з міркувань економії або фізичних обмежень (певну роботу може виконувати тільки один механізм або людина з обмеженою продуктивністю). Такі обмеження істотно впливають на час реалізації проєкту.

Доступні ресурси для реалізації проєкту представлені у вигляді вектора $Z = (z_1; z_2; z_3; ..., z_m)$ розмірності m з елементами z_j ($j = \overline{1,m}$).

Ресурси для реалізації кожної роботи представлені у вигляді векторів $A_i = (a_{i1}; a_{i2}; a_{i3}; ..., a_{im})$ з елементами a_{ij} ($i = \overline{1,n}$; $j = \overline{1,m}$), де n – число робіт (кількість вершин мережевої моделі проєкту); m – кількість різновидів відновлювальних ресурсів, необхідних для реалізації проєкту.

Умова можливої реалізації проєкту наступна: $z_j \geq a_{ij}$ ($i = \overline{1,n}$; $j = \overline{1,m}$) (будь-яку роботу можна починати, якщо кожного з необхідних для її виконання відновлювальних ресурсів достатньо на момент її початку).

Постановка задачі оптимізації в загальному вигляді така: варіюючи порядком виконання робіт з урахуванням їх послідовності, заданої мережевою моделлю, з урахуванням обмежених відновлювальних ресурсів, отримати мінімально можливий час реалізації проєкту.

Задача оптимізації в такій постановці належить до класу NP-повних задач (кількість можливих варіантів, для яких потрібно визначити час виконання проєкту, з подальшим вибором мінімального, експоненціально залежить від кількості вершин мережевої моделі і кількості різновидів ресурсів).

У загальному вигляді таку задачу розв'язати неможливо. Для отримання розв'язання в окремих випадках (розв'язання конкретних задач) використовують метод гілок і меж (на кожному кроці розв'язання оцінюється його перспективність у порівнянні із вже знайденим розв'язком, що дозволяє відсікати неперспективні розв'язки до отримання конкретного результату, збільшуючи при цьому швидкість перебору варіантів).

12.2 Алгоритм методу гілок і меж

1. Обчислити нижню оцінку цільової функції задачі (найменший час реалізації проєкту).

а) для виконання робіт задіяні ресурси одного різновиду і для виконання кожної з робіт потрібен лише один ресурс: $Z = (z_1)$ та $A_i = (1)$ ($i = \overline{1,n}$):

$$T_н = \max_{i=\overline{1,n}}\{S_{кр}; \frac{\sum_1^n t_i}{z_1}\}, \qquad (12.1)$$

де $S_{кр}$ – величина критичного шляху в мережевий моделі проєкту;

t_i – час виконання роботи A_i.

б) для виконання робіт задіяні ресурси одного різновиду і для виконання кожної з робіт потрібно одна або кілька одиниць ресурсу: $Z = (z_1)$ $A_i = (a_{i1})$ ($i = \overline{1,n}$) і $z_1 \geq a_{i1}$.

$$T_н = \max_{i=\overline{1,n}}\{S_{кр}; \frac{\sum_1^n a_{i1} t_i}{z_1}\}. \qquad (12.2)$$

в) загальний випадок (відновлювальні ресурси m різновидів і для виконання кожної з робіт A_i потрібно нуль, одна або кілька одиниць кожного ресурсу): $Z = (z_1; z_2; z_3; ..., z_m)$.

$$A_i = (a_{i1}; a_{i2}; a_{i3}; ..., a_{im}) \text{ і } z_j \geq a_{ij} \ (i = \overline{1,n}; j = \overline{1,m});$$

$$T_н = \max_{i=\overline{1,n}}\{S_{кр}; \max_{j=\overline{1,m}} \frac{\sum_1^n a_{ij} t_i}{z_j}\}. \qquad (12.3)$$

Суть нижньої оцінки полягає в тому, що час виконання проєкту з одного боку не може бути менше величини критичного шляху (всі роботи, розташовані на цьому шляху виконуються послідовно). З іншого боку тривалість реалізації

проєкту не може бути меншою сумарної тривалості безперервної роботи кожного з відновлювальних ресурсів при виконанні усіх робіт.

2. Обчислити верхню оцінку, отримавши будь-який допустимий план реалізації проєкту (побудувати діаграму Ганта – отриманий при побудові час реалізації проєкту і буде на цьому кроці верхньої оцінкою):

2а) якщо $T_н = T_в$, то розв'язок отриманий і оптимальним буде побудований план;

2б) якщо $T_н < T_в$, то йти до кроку 3.

3. Побудувати черговий допустимий план реалізації проєкту, в процесі побудови якого обчислювати нижню поточну оцінку для моментів часу (τ_i), в які змінюється порядок виконання робіт в порівнянні з вже реалізованими проєктами (повністю побудовані діаграми Ганта на кроках 2 або 3) за такою формулою:

$$T_н^{пот}(\tau) = \tau + \max_{i=\overline{1,n},\, k=\overline{1,l}}\{t_i^{\tau}; S_k^{\tau}; \max_{j=\overline{1,m}} \frac{\sum_1^n a_{ij} t_i^{\tau}}{b_j}\}, \qquad (12.4)$$

де l – кількість шляхів в мережевій моделі від початку до закінчення проєкту;

τ – момент часу (від початку виконання) завершення будь-якої з робіт проєкту з метою перевірки перспективності продовження побудови;

t_i^{τ} – тривалість i–ої роботи з урахуванням її часткового або повного виконання до моменту часу τ (в останньому випадку $t_i^{\tau} = 0$);

S_k^{τ} – довжина залишку шляху S_k з моменту часу τ з урахуванням повного або часткового виконання робіт, що входять в цей шлях:

3а) якщо виявилося, що $T_н^{пот}(\tau) \geq T_в$, то побудову поточного плану припиняють через його неперспективність і переходять до початку кроку 3 (тобто, обирають новий варіант плану виконання проєкту);

3б) якщо виявилося, що $T_н^{пот}(\tau) < T_в$, то вибирають чергову роботу для виконання, виділяють необхідні їй ресурси і в момент часу τ^1 ($\tau^1 > \tau$) – коли закінчилася одна або кілька робіт, звільнилися ресурси і можна починати наступну роботу (роботи), обчислюють $T_н^{тек}(\tau^1)$. В процесі побудови або буде отримано новий план реалізації проєкту з часом виконання $T' < T_в$. Тоді встановлюють нове значення $T_в = T'$ і переходять до 2а (тобто переходять до чергового варіанта реалізації проєкту). Оптимальний розв'язок буде знайдено або коли $T_в$ (чергове) стане рівним $T_н$, або коли аналіз усіх допустимих варіантів побудови плану реалізації буде вичерпано. В якості оптимального в тому і в іншому випадку вибирають план реалізації проєкту, що відповідає останньому значенню верхньої оцінки часу виконання: $T^* = T_в^{(ост)}$. Описаний алгоритм представлений схематично (рис. 12.1).

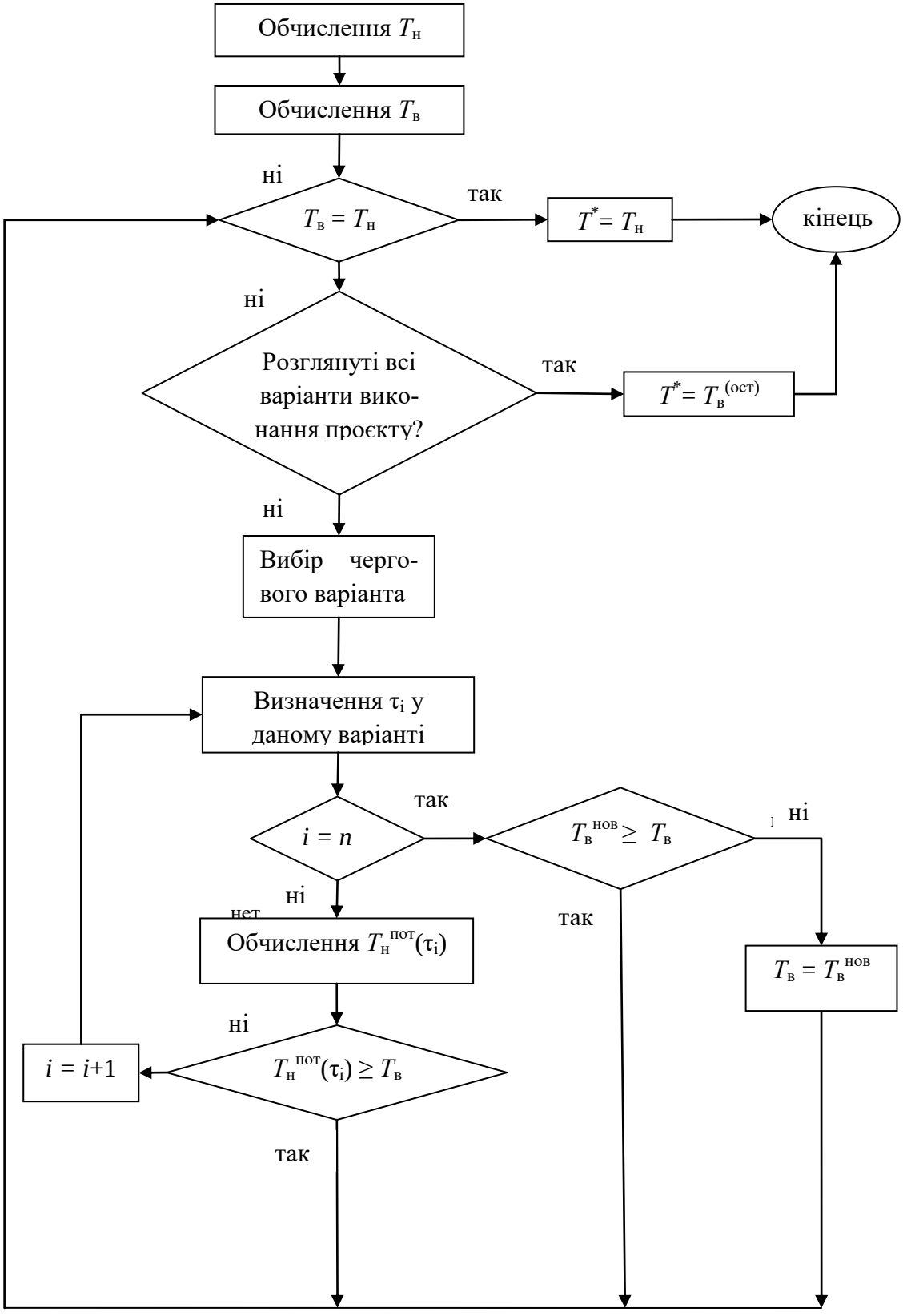


Рисунок 12.1 – Блок-схема алгоритму оптимізації часу виконання проєкту при обмежених відновлювальних ресурсах

12.3 Приклад 1

Для заданої мережевої моделі проєкту типу II з обмеженням відновлювальних ресурсів (рис. 12.2) отримати мінімальний можливий час виконання проєкту і календарний план завантаження ресурсів у вигляді діаграми Ганта. Для виконання кожної роботи потрібен тільки один ресурс $A_i = (1)$ ($i = \overline{1,n}$); усього відновлювального ресурсу: Z = (2).

Розв'язання

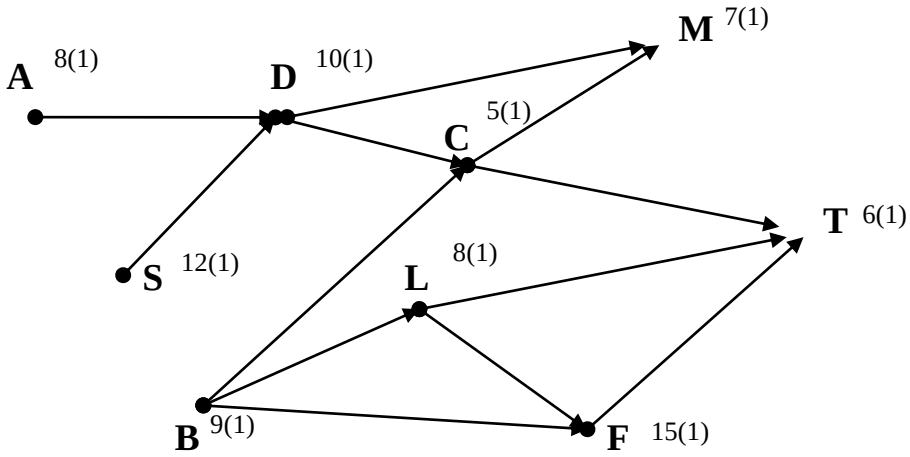


Рисунок 12.2 – Мережева модель проєкту

1. Приведемо мережеву модель проєкту до стандартного виду (одна вершина без дуг, що входять, і одна вершина без вихідних дуг) і по можливості спростимо цю модель (видалимо надлишкові дуги).

Дуги BF, LT, DM надлишкові оскільки умова передування для робіт F, T, M виконується через інші вершини (рис. 12.3).

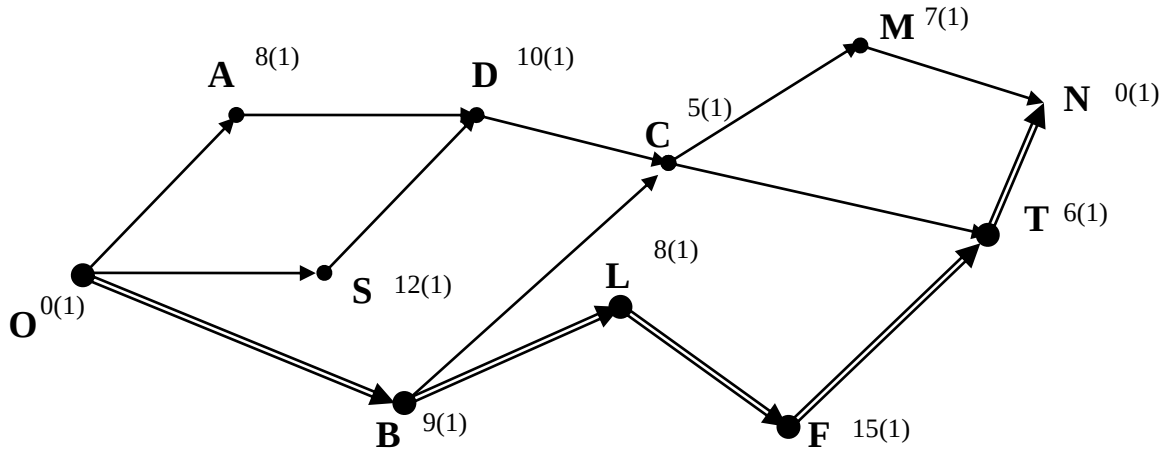


Рисунок 12.3 – Мережева модель проєкту після перетворення і спрощення

2. Визначимо $S_{кр}$ в мережевій моделі(знаходимо усі шляхи, що зв'язують початок О і кінець N проєкту, визначаємо їх величину (довжину) і виберемо максимальну):

S_1(O-A-D-C-M-N) = 8 + 10 + 5 + 7 = 30;

S_2(O-A-D-C-T-N) = 8 + 10 + 5 + 6 = 29;

S_3(O-S-D-C-M-N) = 12+10+5+7 = 34;

S_4(O-S-D-C-T-N) = 12+10+5+6 = 33;

S_5(O-B-C-M-N) = 9+5+7 = 21;

S_6(O-B-C-T-N) = 9+5+6 = 20;

S_7(O-B-L-F-T-N) = 9+8+15+6 = 38 = $S_{кр}$ (на рис. 12.3 виділений подвійною лінією).

3. Обчислимо нижню оцінку часу виконання проєкту за формулою (12.1):

$$T_{н} = \max_{i=\overline{1,n}}\{S_{кр}; \frac{\sum_1^n t_i}{b_1}\}; \qquad (12.1)$$

де $S_{кр}$ – довжина критичного шляху в мережевій моделі проєкту;

t_i – час виконання роботи A_i.

$\sum_1^n t_i$ = 8+10+5+7+12+6+9+8+15 = 80;

$$T_{\text{н}} = \max_{\text{i}=\overline{1,\text{n}}}\{38; \frac{80}{2}\} = 40.$$

4. Обчислимо верхню оцінку, побудувавши який-небудь допустимий план реалізації проєкту у вигляді діаграми Ганта (отриманий при побудові час реалізації проєкту і буде на цьому кроці верхньою оцінкою $T_{\text{в}}$, рис. 12.4):

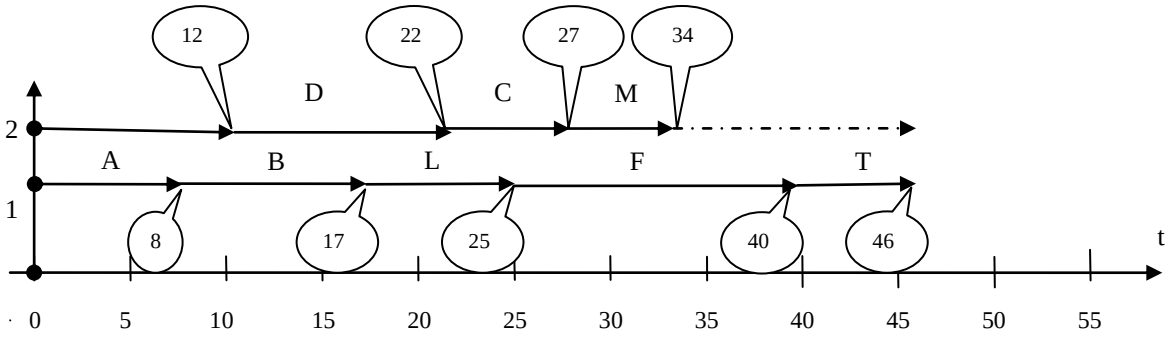


Рисунок 12.4 – Побудова діаграми Ганта для визначення $T_в$

Діаграму Ганта будуємо так: по горизонтальній осі відкладаємо час, по вертикальній – кожен ресурс (у нашому випадку два ресурси одного різновиду). Визначаємо ті роботи, з яких можна починати виконання проєкту, виділяємо для їх виконання ресурси і зображуємо на діаграмі у вигляді стрілок, довжина яких в масштабі відповідає їх тривалості. Відлік часу виконання проєкту розпочинають з нуля. Винесення на діаграмі (див. рис. 12.4) означають моменти закінчення (початку) відповідних робіт. Порядок виконання робіт визначається спрощеною мережевою моделлю проєкту (див. рис. 12.3). Незавантаженість ресурсу на діаграмі відображається штрихпунктирною стрілкою (ресурс 2 на ділянці 34–46). У результаті побудови отриманий варіант реалізації проєкту, для якого час виконання $T_{\text{в}} = 46$. Відповідно до алгоритму випадок 2б) ($T_{\text{н}} < T_{\text{в}}$). Переходимо до п. 3.

5. Будуємо черговий допустимий план реалізації проєкту, в процесі побудови якого обчислюємо нижню поточну оцінку для моментів часу(τ_{i}), в які міняється порядок виконання робіт в порівнянні із вже реалізованими проєктами (повністю побудовані діаграми Ганта на кроках 2 і 3) за формулою (12.2):

$$\text{T}_{н}^{пот}(\tau) = \tau + \max_{i=\overline{1,n},\, k=\overline{1,l}}\{\text{t}_{\text{i}}^{\tau}; \text{S}_{k}^{\tau}; \max_{j=\overline{1,m}} \frac{\sum_{1}^{n} \text{t}_{i}^{\tau}}{\text{b}_{1}}\}; \qquad (12.2)$$

де l – кількість шляхів в мережевій моделі від початку до закінчення проєкту;

τ – момент часу (від початку виконання) завершення якої-небудь роботи проєкту;

t_i^{τ} – тривалість i– ої роботи після моменту часу τ з урахуванням її часткового або повного виконання до моменту часу τ (у останньому випадку $t_i^{\tau} = 0$);

S_k^{τ} – довжина залишку шляху S_k до моменту часу τ з урахуванням повного або часткового виконання робіт, що входять у цей шлях:

Побудову нової діаграми Ганта розпочнемо з виконання робіт A і B, які закінчуються в моменти часу 8 і 12.

Нехай $\tau = 8$; визначимо $T_н^{пот}(8)$ за формулою (4) (виконана робота A і 8 з 12 роботи B)

$$T_н^{пот}(8) = 8 + \max_{\overline{i=1,n},\ \overline{k=1,l}}\{1;22,21,34,33,13,12,30;\frac{80-16}{2}\} = 8 + 34 = 42.$$

S_1 (O-A-D-C-M-N) = 8+10+5+7 = 30 (стало 22);

S_2 (O-A-D-C-T-N) = 8+10+5+6 = 29 (стало 21);

S_3 (O-S-D-C-M-N) = 12+10+5+7 = 34 (не змінилось);

S_4 (O-S-D-C-T-N) = 12+10+5+6 = 33(не змінилось);

S_5 (O-B-C-M-N) = 9+5+7 = 21(стало 13);

S_6 (O-B-C-T-N) = 9+5+6 = 20(стало 12);

S_7 (O-B-L-F-T-N) = 9+8+15+6 = 38(стало 30).

Відповідно до алгоритму – випадок 3б): виявилось, що $T_н^{пот}(8) = 42 < 46$; вибираємо роботу S для продовження побудови нового плану, виділяємо необхідні їй ресурси і у момент часу $\tau = 9$) – закінчилися роботи A і B, а робота S тривалістю 12 виконана на 1 од. Визначимо $T_н^{пот}(9)$.

S_1 (O-A-D-C-M-N) = 8+10+5+7 = 30 (стало 22);

S_2 (O-A-D-C-T-N) = 8+10+5+6 = 29 (стало 21);

S_3 (O-S-D-C-M-N) = 12+10+5+7 = 34 (стало 33);

S_4 (O-S-D-C-T-N) = 12+10+5+6 = 33(стало 32);

S_5 (O-B-C-M-N) = 9+5+7 = 21(стало 12);

S_6 (O-B-C-T-N) = 9+5+6 = 20(стало 11);

S_7 (O-B-L-F-T-N) = 9+8+15+6 = 38(стало 29).

$$T_н^{пот}(9) = 9 + \max_{i=\overline{1,n},\ k=\overline{1,l}}\{1;\ 22,21,33,32,13,11,29;\ \frac{80-18}{2}\} = 9 + 33 = 42.$$

Відповідно до алгоритму – випадок 3б): виявилось, що $T_н^{пот}(9) = 42 < 46$; обираємо роботу L для продовження побудови нового плану, виділяємо необхідні їй ресурси і у момент часу τ = 17) – закінчилися роботи A, B і L, а робота S тривалістю 12 виконана на 9 од. Визначимо $T_н^{пот}(17)$.

S_1 (O-A-D-C-M-N) = 8+10+5+7 = 30 (стало 22);

S_2 (O-A-D-C-T-N) = 8+10+5+6 = 29 (стало 21);

S_3 (O-S-D-C-M-N) = 12+10+5+7 = 34 (стало 25);

S_4 (O-S-D-C-T-N) = 12+10+5+6 = 33(стало 24);

S_5 (O-B-C-M-N) = 9+5+7 = 21(стало 12);

S_6 (O-B-C-T-N) = 9+5+6 = 20(стало 11);

S_7 (O-B-L-F-T-N) = 9+8+15+6 = 38(стало 21).

$$T_н^{пот}(17) = 17 + \max_{i=\overline{1,n},\ k=\overline{1,l}}\{1;\ 22,21,25,24,12,11,21;\ \frac{80-34}{2}\} = 17 + 25 = 42.$$

Відповідно до алгоритму – випадок 3б): виявилось, що $T_н^{пот}(17) = 42 < 46$; продовжуємо побудову нового плану, обчислюючи $T_н^{пот}$ для моментів часу τ = 20, 30, 35, які підтверджують перспективність побудови. Результат остаточної побудови представлений на рис. 12.5.

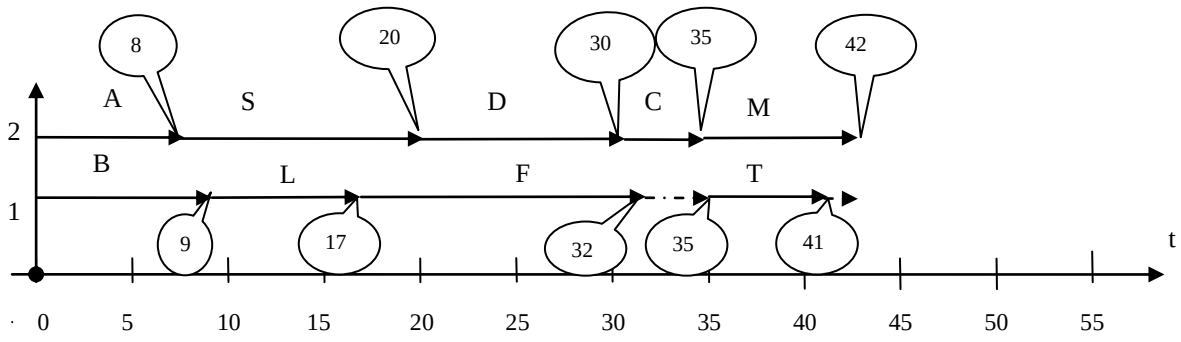


Рисунок 12.5 – Побудова діаграми Ганта для поліпшеного плану

Отримали $T_{\text{н}}^{\text{нов}}$ = 42. Ресурс 2 повністю завантажений усі 42 од. часу, а ресурс 1 не завантажений в проміжку 32…35 (3 од. часу) і 41…42 (1 од. часу), що в сумі складає 4 од. часу. Формально треба продовжувати пошук кращого варіанта, намагаючись будувати нові варіанти плану реалізації проєкту. Для практичних цілей припиняють пошук нових варіантів, якщо невідповідність складає менше 5…8 % ($\delta = \frac{T_{\text{н}}^{\text{нов}} - T_{\text{н}}}{T_{\text{н}}} \cdot 100\%$). У нашому випадку ((42–40)/40) · 100 % = 5 %.

12.4 Приклад 2

Для заданої мережевої моделі проєкту типу II з обмеженням відновлюваних ресурсів отримати мінімальний можливий час виконання проєкту і календарний план завантаження ресурсів у вигляді діаграми Ганта (рис. 12.6). Для виконання кожної роботи потрібно один або кілька одиниць одного ресурсу Ai = (a_{i1}) ($i = \overline{1,n}$); запас ресурсу: Z = (3).

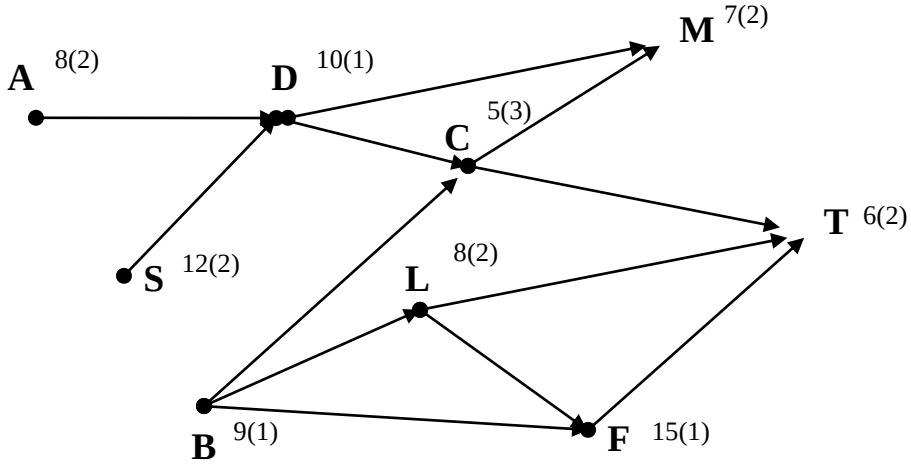


Рисунок 12.6 – Мережева модель проєкту

Розв'язання

1. Приведемо мережеву модель проєкту до стандартного виду (одна вершина без вхідних дуг і одна вершина без вихідних дуг) і по можливості спростимо цю модель (видалимо надлишкові дуги).

Дуги BF, LT, DM надлишкові оскільки умова передування для робіт F, T, M виконується через інші вершини.

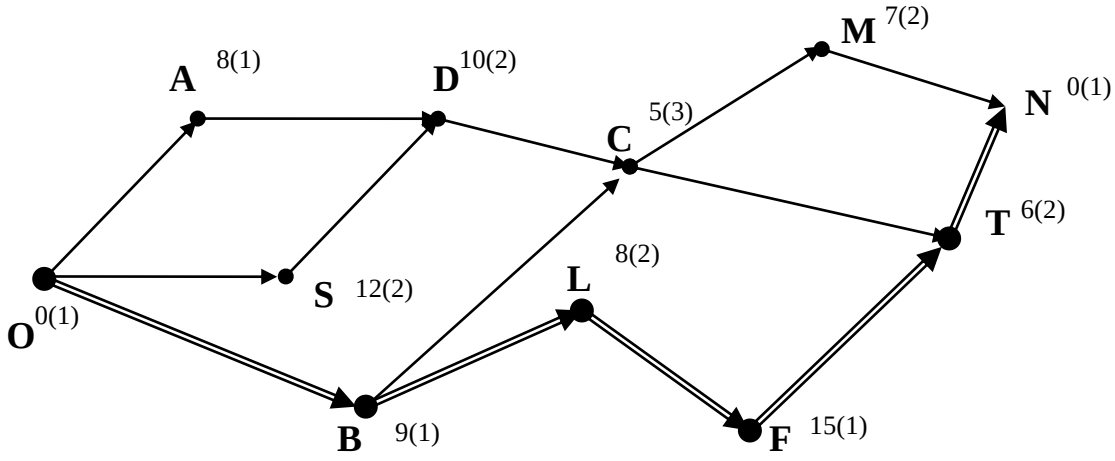


Рисунок 12.7 – Мережева модель проєкту після стандартизації і спрощення

2. Визначимо $S_{кр}$ у мережевій моделі (знаходимо усі шляхи, що зв'язують початок O і кінець N проєкту, визначаємо їх довжину й вибираємо максимальну, табл. 12.1).

Таблиця 12.1 – Залишки шляхів в різні моменти часу τ_i

S_k	Шлях по вершинах	Довжина шляху	Довжина залишку шляху до моменту часу τ_i					
			9	12	20	30	35	40
S_1	(O-A-D-C-M-N)	30	30	30	22	12	12	7
S_2	(O-A-D-C-T-N)	29	29	29	21	11	11	6
S_3	(O-S-D-C-M-N)	34	25	22	22	12	12	7
S_4	(O-S-D-C-T-N)	33	24	21	21	11	11	6
S_5	(O-B-C-M-N)	21	12	12	12	12	12	7
S_6	(O-B-C-T-N)	20	11	11	11	11	11	6
S_7	(O-B-L-F-T-N)	38	29	29	21	11	6	6
максимум			30	30	22	12	12	7

$S_{кр} = S_7 = 9 + 8 + 15 + 6 = 38 =$ (на рис. 12.7 виділений подвійною лінією).

3. Обчислимо нижню оцінку часу виконання проєкту $T_н$ за формулою (12.3):

$$T_н = \max_{i=\overline{1,n}}\{S_{кр}; \frac{\sum_1^n a_{i1} t_i}{z_1}\}; \qquad (12.3)$$

де $S_{кр}$ – довжина критичного шляху в мережевій моделі проєкту;

t_i – час виконання роботи A_i.

$$\sum_1^n a_{i1} t_i = 8\cdot1+10\cdot2+5\cdot3+7\cdot2+12\cdot2+6\cdot2+9\cdot1+8\cdot2+15\cdot1 = 133;$$

$T_н = \max_{i=\overline{1,n}}\{38; \frac{133}{3}\} = 45$ (ділення 133 на 3 виконують без залишку: результат – найближче більше ціле число 133/3 = 44,33 → 45).

Обчислимо верхню оцінку, побудувавши який-небудь допустимий план реалізації проєкту у вигляді діаграми Ганта (отриманий при побудові час реалізації проєкту і буде на цьому кроці верхньою оцінкою $T_в$,, рис. 12.8).

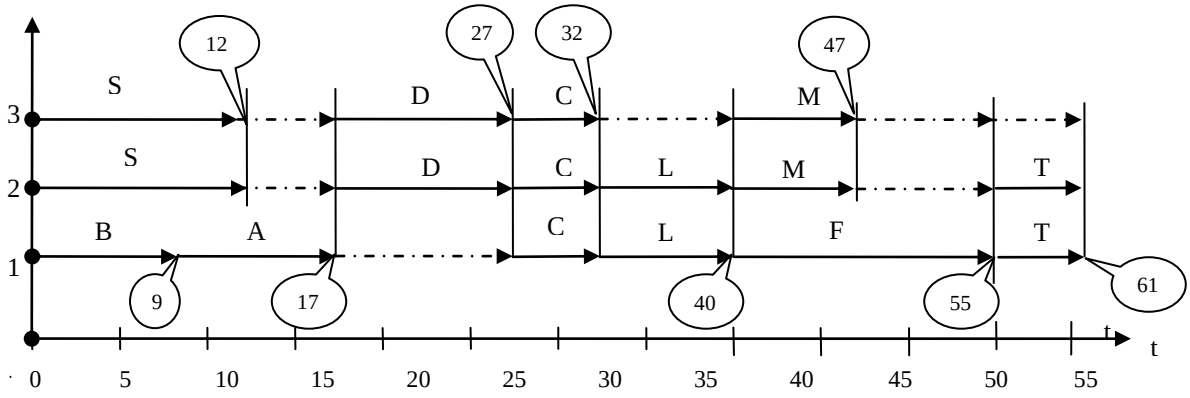


Рисунок 12.8 – Побудова діаграми Ганта для визначення $T_в$

$T_н(45) < T_в(61)$ переходимо до п. 2б алгоритму (виконати п. 3).

5. Будуємо черговий допустимий план реалізації проєкту, в процесі побудови якого обчислюємо нижню поточну оцінку для моментів часу(τ_i), в які міняється порядок виконання робіт в порівнянні із вже реалізованими проєктами (повністю побудовані діаграми Ганта на кроках 2 і 3, рис. 12.9) за такою формулою (12.4):

$$T_н^{пот}(\tau) = \tau + \max_{i=\overline{1,n},\, k=\overline{1,l}}\{t_i^\tau; S_k^\tau; \frac{\sum_1^n a_{i1} t_i^\tau}{b_1}\}; \qquad (12.4)$$

де l – кількість шляхів в мережевій моделі від початку до закінчення проєкту;

τ – момент часу (від початку виконання) завершення якої-небудь роботи проєкту;

t_i^{τ} – тривалість залишку і-ої роботи після моменту часу τ (якщо робота виконана до моменту часу τ повністю, то для цієї роботи $t_i^{\tau} = 0$);

S_k^{τ} – довжина залишку шляху S_k в момент часу τ з урахуванням повного або часткового виконання робіт, що входять в цей шлях (для різних τ представлені в табл. 12.1):

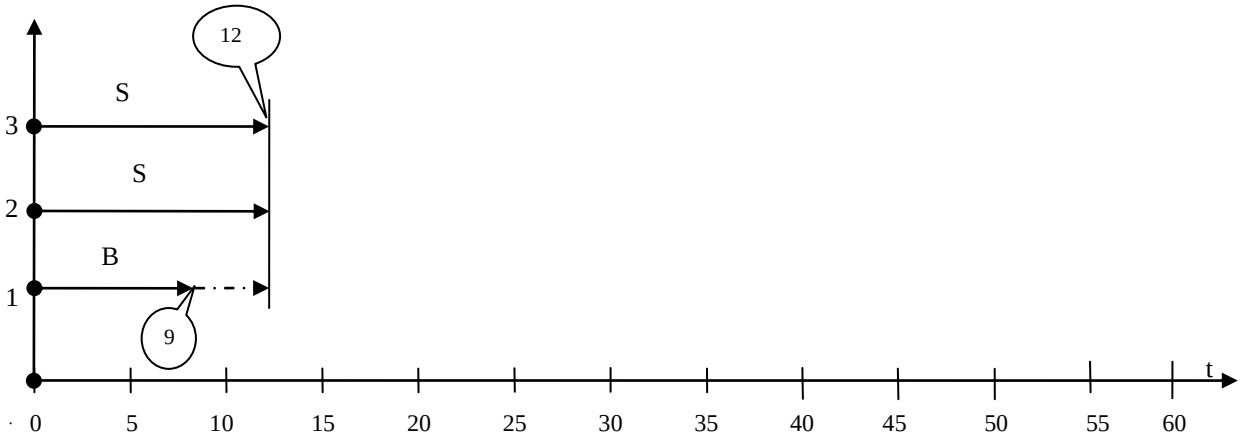


Рисунок 12.9 – Фрагмент побудови діаграми Ганта з оцінкою $T_н^{пот}(\tau)$

$$T_н^{пот}(9) = 9 + \max_{i=\overline{1,n},\ k=\overline{1,l}}\{3;\ 30;\ 29;\ 25;\ 24;\ 12;\ 11;\ 29;\ \frac{133-27}{3}\} = 9 + 36 = 45.$$

$T_н^{пот}(9)\ (45) < T_в(61)$ (продовжуємо будувати новий план реалізації проєкту).

$$T_н^{пот}(12) = 12 + \max_{i=\overline{1,n},\ k=\overline{1,l}}\{3;\ 30;\ 29;\ 22;\ 21;\ 12;\ 11;\ 29;\ \frac{133-24-9}{3}\} = 12 + 34 = 46.$$

$T_н^{пот}(12)\ (46) < T_в(61)$ (продовжуємо будувати новий план реалізації проєкту (рис. 12.10).

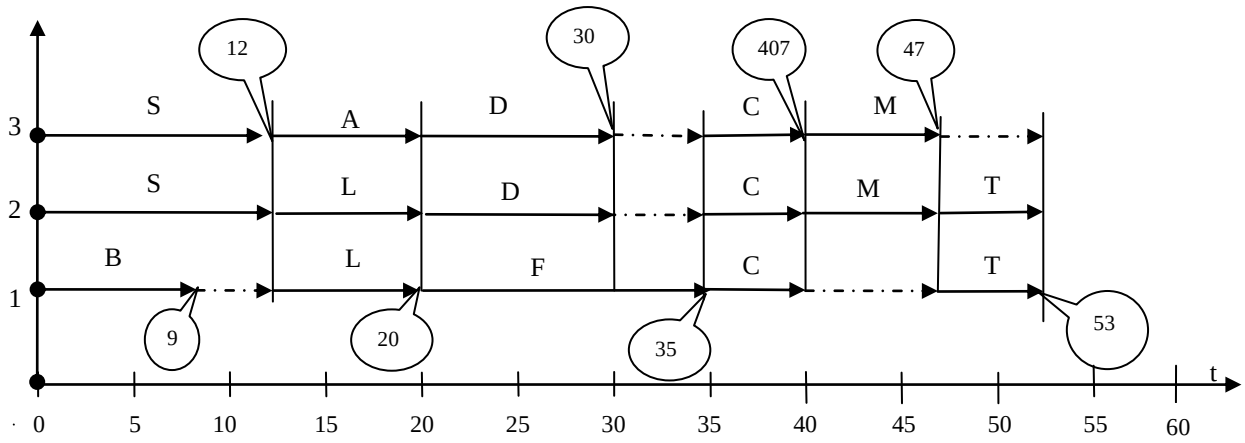


Рисунок 12.10 – Закінчення побудова діаграми Ганта з оцінкою $\mathrm{T}_{н}^{пот}(\tau)$

Починаючи з моменту часу 12 виконуємо роботи A і L і визначаємо $\mathrm{T}_{н}^{пот}(20)$.

$$\mathrm{T}_{н}^{пот}(20) = 20 + \max_{i=\overline{1,n},\, k=\overline{1,l}}\{0;\, 22;\, 21;\, 22;\, 21;\, 12;\, 11;\, 21;\, \frac{133-40-17}{3}\} = 20 + 26 = 46.$$

$\mathrm{T}_{н}^{пот}(12)\ (46) < \mathrm{T}_{в}(61)$ (продовжуємо будувати новий план реалізації проєкту).

Починаючи з моменту часу 20 і до 30 виконуємо роботи D і F (частково) і визначаємо $\mathrm{T}_{н}^{пот}(30)$.

$$\mathrm{T}_{н}^{пот}(30) = 30 + \max_{i=\overline{1,n},\, k=\overline{1,l}}\{5;\, 12;\, 11;\, 12;\, 11;\, 12;\, 11;\, 11;\, \frac{133-57-30}{3}\} = 30 + 16 = 46.$$

$\mathrm{T}_{н}^{пот}(30)\ (46) < \mathrm{T}_{в}(61)$ (продовжуємо будувати новий план реалізації проєкту).

Починаючи з моменту часу 30 і до 35 закінчуємо роботу F і визначаємо $\mathrm{T}_{н}^{пот}(35)$.

$$\mathrm{T}_{н}^{пот}(35) = 35 + \max_{i=\overline{1,n},\, k=\overline{1,l}}\{0;\, 12;\, 11;\, 12;\, 11;\, 12;\, 11;\, 6;\, \frac{133-87-5}{3}\} = 35 + 14 = 49.$$

$\mathrm{T}_{н}^{пот}(35)\ (49) < \mathrm{T}_{в}(61)$ (продовжуємо будувати новий план реалізації проєкту).

Починаючи з моменту часу 35 і до 53 виконуємо роботи C, M і T. Отримали нову діаграму Ганта (варіант реалізації проєкту), для якої $T_{\text{в}}^{\text{нов}} = 53$. Відповідно до алгоритму встановлюємо нове значення $T_{\text{в}} = 53$ і так $T_{\text{н}}(45) < T_{\text{в}}(53)$ переходимо до п. 2б алгоритму (виконувати п.3 до перебору усіх варіантів реалізації проєкту). Аналіз останньої діаграми Ганта (див. рис. 4.10) показує неможливість її поліпшення.

12.5 Перетворення мережевих моделей проєктів типу I на мережеві моделі типу II

Алгоритм перетворення

1. Позначити кожну дугу моделі I буквами (A, B, C.) – назви робіт.

2. Скласти таблицю відношення передування R (для кожної вершини моделі I заповнити рядок таблиці, рис. 12.11):

Назва вершини	Закінчені роботи у вершині	Розпочаті роботи у вершині
1	2	3

Рисунок 12.11 – Таблиця відношення передування R

3. Представити інформацію таблиці п. 2 у вигляді відношення передування R ={елементи відношення.

4. Зобразити відношення у вигляді орієнтованого графа, доповнити інформацією про тривалість робіт, необхідних для виконання кожної роботи, ресурсах і привести орієнтований граф, як мережеву модель проєкту, до стандартного виду (одна вершина без дуг, що входять, і одна вершина без вихідних дуг).

Примітка. У мережевих моделях проєктів зручно додавати (видаляти) роботи. Якщо проєкт великий і з'ясувалося після його початку, що в моделі забули представити одну з робіт, то з'ясовують, які роботи передують цій роботі й які роботи не можна починати до її закінчення і просто додають забуту роботу, виконуючи п.4.

Нижче наведений приклад реалізації алгоритму з додаванням роботи і розв’язанням оптимізаційної задачі з обмеженням ресурсів для загального випадку.

12.6 Приклад 3

Перетворити задану на рис. 12.12 мережеву модель проєкту типу I в мережеву модель проєкту типу II (тип II: вершини – роботи; дуги – послідовність виконання робіт), додати роботу Y, яка починається після закінчення роботи C,

а після її виконання починається робота D і отримати оптимальний план завантаження ресурсів для випадку, коли видів ресурсу для виконання будь-якої роботи два - $A_i = (a_{i1}; a_{i2})$, а загальний об'єм цих ресурсів $Z = (z_1; z_2;) = (2, 3)$ і визначити оптимальний (мінімально можливий) час виконання проєкту (табл. 12.2) і календарний план завантаження ресурсів у вигляді діаграми Ганта.

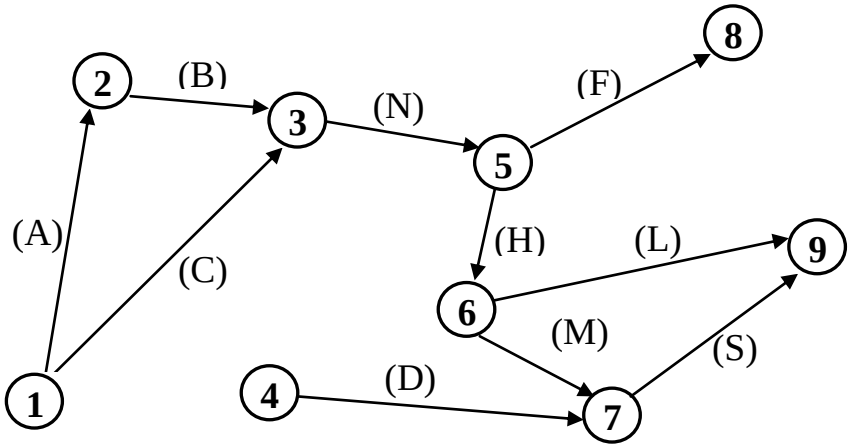


Рисунок 12.12 – Мережева модель проєкту типу I

Таблиця 12. 2 – Таблиця ресурсів

Робота	A	B	C	D	F		N	H	M	L	S	Z
Ресурс	12(1,1)	8(1,2)	6(0,2)	8(1,0)	15(1,1)		4(2,1)	6(1,1)	7(1,1)	5(2,2)	8(1,1)	6(1,2)

Розв'язання

1. Перетворимо мережеву модель проєкту типу I в тип II.

1.1. Позначимо кожну дугу моделі типу I буквами(назви робіт) – на рис. 12.12 ця дія виконана.

1.2. Складемо таблицю (табл. 12.3) для побудови відношення передування R.

1.3. За інформацією таблиці п. 1.2 побудуємо відношення R.

R ={AB, BN, CN, NF, NH, HL, HM, DS, MS }

1.4. Представимо відношення у вигляді орієнтованого графа, приведемо до стандартного виду (одна вершина без дуг, що входять, і одна вершина без вихідних дуг, рис. 12.13).

Таблиця 12. 3 – Таблиця для побудови відношення передування

Назва вершини	Закінчені роботи у вершині	Розпочаті роботи у вершині
1	–	A,C
2	A	B
3	B,C	N
4	–	D
5	N	F,H
6	H	L,M
7	D,M	S
8	F	–
9	L,S	–

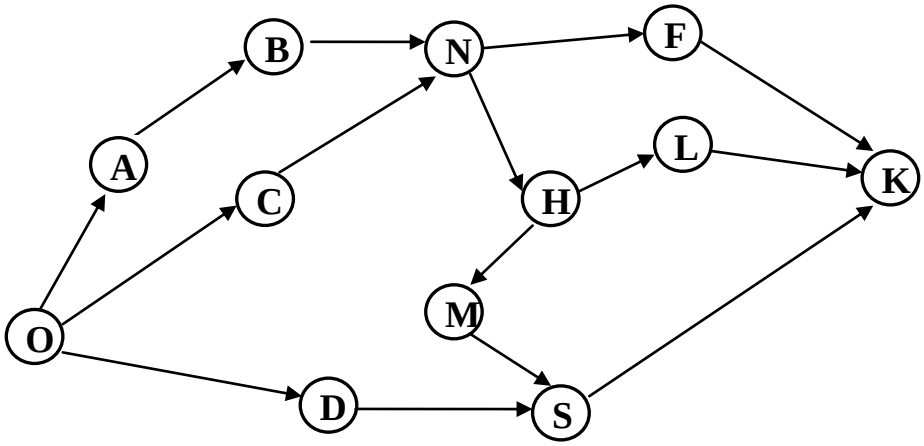


Рисунок 12.13 – Мережева модель проєкту типу II

2. Додамо відповідно до опису роботу Y, видалимо надлишкові дуги (OD) і доповнимо мережеву модель інформацією про тривалість робіт і необхідні для їх виконання ресурси (рис. 12.14).

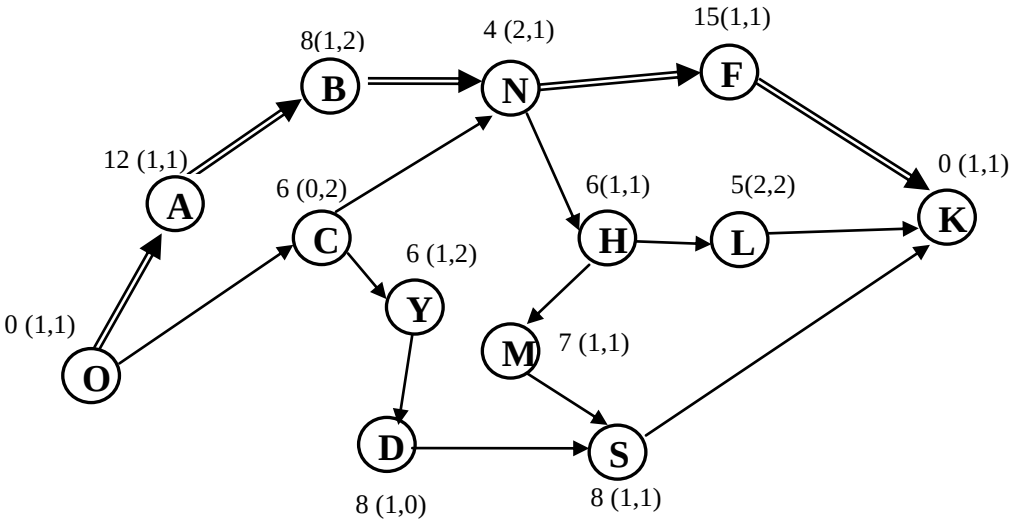


Рисунок 12.14 – Мережева модель проєкту типу II з додаванням роботи Y

3. Визначимо $S_{кр}$ в мережевій моделі (знаходимо усі шляхи, що зв'язують початок О і кінець К проєкту, визначаємо їхню довжину і вибираємо максимальну, табл. 12.3).

Таблиця 12.3

S_k	Шлях по вершинах	Довжина шляху	Довжина залишку шляху до моменту часу τ_i					
			12	20	24	37		
S_1	(O-A-B-N-F-K)	39	27	19	15	2		
S_2	(O-A-B-N-H-L-K)	35	23	15	11	5		
S_3	(O-A-B-N-H-M-S-K)	45	33	25	21	8		
S_4	(O-C-N-F-K)	25	19	19	15	2		
S_5	(O-C-N-H-L-K)	21	15	15	11	5		
S_6	(O-C-N-H-M-S-K)	31	25	25	21	8		
S_7	(O-C-Y-D-S-K)	28	16	8	8	8		
максимум			27	25	21	8		

$S_{кр} = S_3 = 12 + 8 + 4 + 6 + 7 + 8 = 45$ (на рис. 12.14 виділений подвійною лінією).

4. Обчислимо нижню оцінку часу виконання проєкту ($T_н$) за формулою:

$$T_н = \max_{i=\overline{1,n}}\{S_{кр}; \max_{j=\overline{1,2}} \frac{\sum_1^n a_{ij} t_i}{z_j}\},$$

де $S_{кр}$ – довжина критичного шляху в мережевій моделі проєкту;

t_i – час виконання роботи A_i;

a_{ij} – кількість j-го ресурсу для виконання роботи A_i.

$\sum_1^n a_{i1} t_i = 12\cdot1+8\cdot1+6\cdot1+8\cdot1+4\cdot2+6\cdot1+7\cdot1+8\cdot1+15\cdot1+ 5\cdot2 = 88;$

$\sum_1^n a_{i2} t_i = 12\cdot1+8\cdot2+6\cdot2+6\cdot2+4\cdot1+6\cdot1+7\cdot1+8\cdot1+15\cdot1+ 5\cdot2 = 102.$

$T_н = \max_{i=\overline{1,n}}\{45; \frac{88}{2}; \frac{102}{3}\} = 45$ (ділення 102 на 3 виконують без залишку: результат – найближче більше ціле число 102/3 = 34→ 34).

5. Обчислимо верхню оцінку, побудувавши будь-який допустимий план реалізації проєкту у вигляді діаграми Ганта (отриманий при побудові час реалізації проєкту і буде на цьому кроці верхньою оцінкою, рис. 12.15).

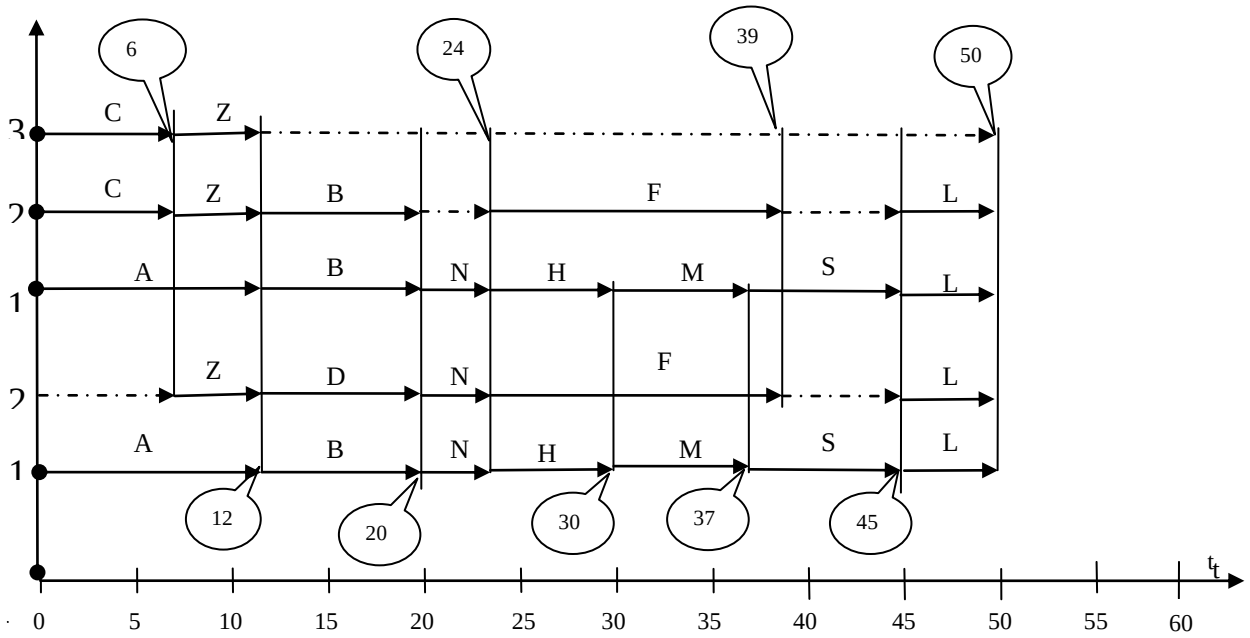


Рисунок 12.15 – Побудова діаграми Ганта для визначення $T_в$

$T_н(45) < T_в(50)$ переходимо до п. 2б алгоритму (виконати п.3).

6. Будуємо черговий допустимий план реалізації проєкту, у процесі побудови якого обчислюємо нижню поточну оцінку для моментів часу (τ_i), в які змі-

нюється порядок виконання робіт в порівнянні з вже реалізованими проєктами (повністю побудовані діаграми Ганта на кроках 2 і 3) за такою формулою (12.5), рис. 12.16:

$$T_н^{пот}(\tau) = \tau + \max_{i=\overline{1,n},\ k=\overline{1,l}}\{t_i^{\tau}; S_k^{\tau}; \max_{i=\overline{1,n},\ j=\overline{1,2}} \frac{\sum_1^n a_{ij} t_i^{\tau}}{z_j}\}; \qquad (12.5)$$

де l – кількість шляхів в мережевий моделі від початку до закінчення проєкту;

τ – момент часу (від початку виконання) завершення будь-якої роботи проєкту;

t_i^{τ} – тривалість залишку i-ої роботи після моменту часу τ (якщо робота виконана до моменту часу τ повністю, то для цієї роботи $t_i^{\tau} = 0$);

S_k^{τ} – довжина залишку шляху S_k до моменту часу τ з урахуванням повного або часткового виконання робіт, що входять в цей шлях (для різних τ представлені в табл. 12.3).

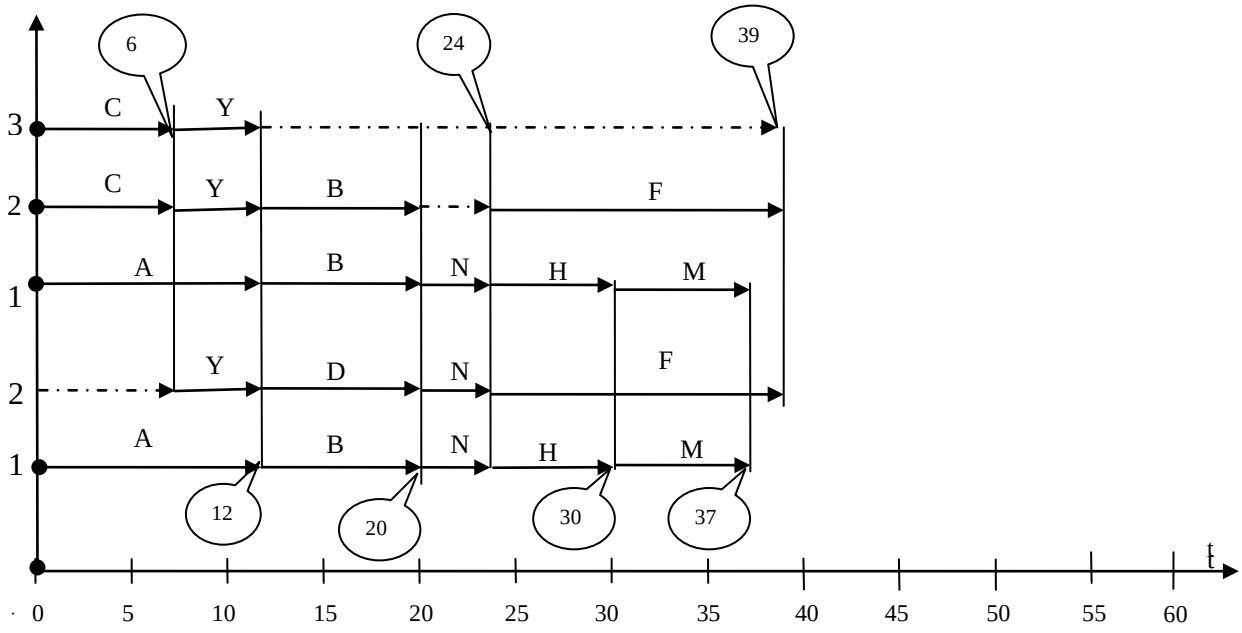


Рисунок 12.16 – Побудова покращеної діаграми Ганта

$$T_н^{пот}(12) = 12 + \max_{i=\overline{1,n},\ k=\overline{1,l}}\{0; 27; 23; 22; 19; 15; 25; 16; \frac{88-24}{2}; \frac{102-36}{3}\} = 12 + 32 = 44.$$

$T_н^{пот}(12)\ (=44) < T_в\ (=50)$ (продовжуємо будувати новий план реалізації проєкту).

$$T_н^{пот}(20) = 20 + \max_{i=\overline{1,n},\ k=\overline{1,l}}\{0; 19; 15; 14; 19; 15; 25; 8; \frac{88-40}{2}; \frac{102-60}{3}\} = 20 + 25 = 45.$$

$T_н^{пот}(20)\ (=45) < T_в (=50)$ (продовжуємо будувати новий план реалізації проєкту).

$$T_н^{пот}(24) = 24 + \max_{i=\overline{1,n},\ k=\overline{1,l}}\{0; 15; 11; 10; 15; 11; 21; 8; \frac{88-48}{2}; \frac{102-72}{3}\} = 24 + 21 = 45.$$

$T_н^{пот}(24)\ (=45) < T_в (=50)$ (продовжуємо будувати новий план реалізації проєкту).

$$T_н^{пот}(37) = 37 + \max_{i=\overline{1,n},\ k=\overline{1,l}}\{2; 5; 8; 2; 5; 8; 8; 2; \frac{88-74}{2}; \frac{102-111}{3}\} = 37 + 8 = 45.$$

Добудова (роботи S і T показує, що покращення нема) $T_в = 50$.

12.7 Перетворення мережевих моделей проєктів типу II на мережеві моделі типу I

Алгоритм перетворення

1 Перевірити мережеву модель проєкту типу II на надмірність дуг (надлишкові дуги видалити).

2 Представити мережеву модель типу II у вигляді відношення передування R = {елементи відношення} (кожна дуга мережевої моделі XY – елемент відношення R).

3 Відповідно до інформації п.2 побудувати таблицю (12.4), що складається з двох стовпців (кожен рядок таблиці відображає один елемент відношення R, причому X розташовують в першому стовпці, а Y – у другому).

Таблиця 12.4 – Таблиця даних моделі типу II

Робота (початок дуги)	Робота (кінець дуги)
1	2
X	Y

4. Модифікувати таблицю п. 3:

а) об'єднати рядки, в яких роботи збігаються в першому або другому стовпцях, вказавши незбіжні роботи у вигляді підмножин;

б) визначити роботи, які не зустрічаються в другому стовпці (не мають робіт-предків) і роботи, які не зустрічаються в першому стовпці (не мають ро-

біт-нащадків) у таблиці п. 3; додати в модифіковану таблицю два рядки – перший (у стовп. 1 – прочерк, у стовп. 2 – у вигляді підмножини перелік робіт без предків) і останній (у стовп. 2 – прочерк, у стовп. 1 – у вигляді підмножини перелік робіт без нащадків);

в) пронумерувати рядки модифікованої таблиці (12.4) за порядком, додавши стовп. 3.

Таблиця 12.4 – Модифікована таблиця даних моделі типу II

Робота (Початок дуги)	Робота (Кінець дуги)	Назва (номер) вершини
1	2	3
–	перелік робіт без предків	1
…..	…..	…..
перелік робіт без нащадків	–	n

5. Представити таблицю п. 4 у вигляді орієнтованого графа, на якому кожен рядок таблиці представлений так: показати вершину 1 і провести з неї дуги для кожної з робіт, зазначених в стовп. 2 (початок роботи – дуги); для визначення вершин, в яких закінчуються ці роботи необхідно в стовп. 1 знайти роботу з тією ж назвою і, переміщаючись по рядку до стовп. 3, встановити вершину, в якій закінчується дуга – робота; процедуру повторювати до оброблення всіх рядків.

Примітка: якщо в результаті побудови з'явилося кілька дуг-робіт з однаковою назвою, то додати ще одну вершину, з якої виходить тільки одна ця робота, а входять робота нульової тривалості, яка починаються в тих вершинах, в яких раніше починалася робота, представлена кількома дугами. Викладене в примітці представлено на рисунку 12.17 в вигляді перетворення фрагмента мережевої моделі I.

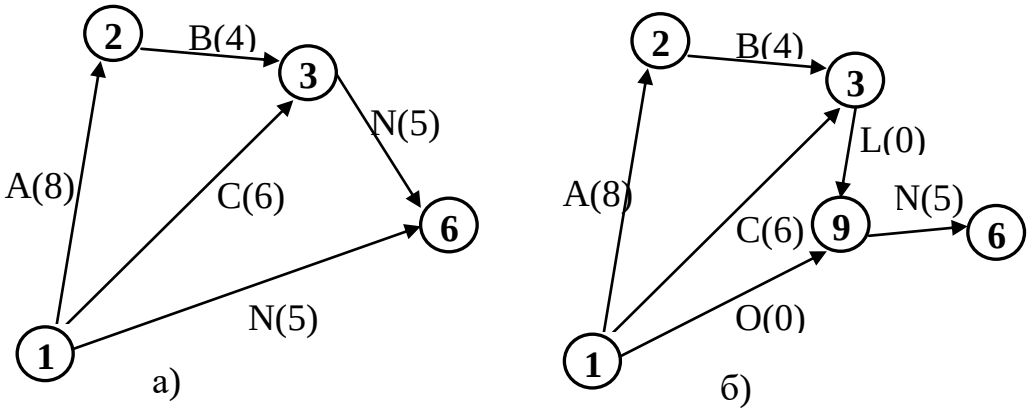


Рисунок 12.17 – Виконання дій згідно з приміткою

13 ЛАБОРАТОРНІ РОБОТИ

13.1 Лабораторна робота № 1. Алгоритми. Властивості алгоритмів. Способи запису алгоритмів

Мета роботи: отримання навичок дослідження властивостей та способів запису алгоритмів.

Завдання до виконання

1. Розробити алгоритм розв'язання задачі (варіант див. в таблиці 13.1).

2. Запропонувати алгоритм вирішення задачі. Номер варіанта задачі відповідає номеру студента в списку групи.

3. Для запропонованого алгоритму надати – словесний опис – покроковий опис – блок-схему алгоритму – опис у вигляді псевдокоду – структурограму Нассі-Шнайдермана – Flow-форму. (див. *Приклад 1.3*)

4. Показати, що для запропонованої процедури вирішення задачі виконуються всі властивості алгоритму.

Контрольні питання

1. Дати визначення поняття «алгоритм».

2. Охарактеризувати основні властивості алгоритму.

3. Перелічити основні способи представлення алгоритмів.

4. Базові структури алгоритму.

5. Охарактеризувати мовний спосіб запису алгоритмів.

6. В чому полягає сутність процесу представлення алгоритмів за допомогою блок-схем?

7. Перелічити основні символи, які використовуються при розробці блок-схем алгоритмів.

8. Охарактеризувати сутність представлення алгоритмів за допомогою псевдокоду.

9. Описати процедуру використання структурограм Нассі–Шнейдермана при запису алгоритмів.

Таблиця 13.1 –Варіанти завдань

Варіант	Задача
1	Дано 20 цілих чисел. Визначити, скільки з них приймає найбільше значення.
2	Дано натуральне k. Вивести k-у цифру послідовності 1123581321 ..., в якій виписані підряд всі числа Фібоначчі.
3	Дана послідовність з не менше ніж 2-х натуральних чисел, за якою слідує 0. Обчислити суму тих з них, порядкові номери яких - прості числа.
4	Дано натуральне k. Вивести k-у цифру послідовності 149162536 ..., в якій виписані підряд квадрати всіх натуральних чисел. (Приклад: для k=7 результат дорівнює 5, це сьома цифра означеної послідовності, а для k=11 результат дорівнює 9, це одинадцята цифра послідовності 149162536496481…)
5	Дана непорожня послідовність з натуральних чисел, за якою слідує 0. Обчислити суму тих з них, порядкові номери яких - числа Фібоначчі.
6	Дано 10 натуральних чисел. Знайти їх найбільший спільний дільник.
7	Визначити, чи є задане натуральне число досконалим, тобто рівним сумі всіх своїх (позитивних) дільників, крім самого цього числа (напр. Число 6 досконале: 6 = 1 + 2 + 3).
8	Дана непорожня послідовність ненульових чисел, за якою слідує 0. Визначити, скільки разів в цій послідовності змінюється знак (напр., в послідовності 1, -34, 8, 14, -5 знак змінюється 3 рази).
9	Дано 20 дійсних чисел. Визначити, скільки з них більше своїх "сусідів", тобто попереднього і наступного чисел.
10	Дано не менше 3-х різних натуральних чисел, за якими слідує 0. Визначити 3 найбільших числа серед них.

Продовження таблиці 13.1

Варіант	Задача
11	Визначити число, що утворюється виписуванням в зворотному порядку цифр заданого натурального числа.
12	Дана послідовність з 20 цифр. Визначити кількість цифр в найбільш довгій підпослідовності з підряд розташованих нулів. (наприклад, для послідовності 25007000805568400421 відповідь 3).
13	Дано 20 дійсних чисел. Знайти порядковий номер того з них, яке найближче до якого-небудь цілого числа.
14	Вивести в порядку зростання всі тризначні числа, в десятковому записі яких немає однакових цифр (операцію ділення не використовувати).
15	Дано 10 дійсних чисел. Обчислити різницю між максимальним і мінімальним з них.
16	Дано натуральне k. Вивести k-у цифру послідовності 12345678910111213 ..., в якій виписані підряд всі натуральні числа.
17	Дана послідовність з 20-ти цілих чисел. Визначити, з скількох негативних чисел вона починається.
18	Визначити, чи є задане натуральне число паліндромом, тобто таким, десятковий запис якого читається однаково зліва направо і зправа наліво.
19	Дано 20 дійсних чисел. Визначити, чи утворюють вони зростаючу послідовність.
20	Дано ціле n> 0 і послідовність з n дійсних чисел, серед яких є хоча б одне негативне число. Знайти величину найбільшого серед негативних чисел цієї послідовності.
21	Знайти суму цифр заданого натурального числа.
22	Дана непорожня послідовність різних натуральних чисел, за

Продовження таблиці 13.1

Варіант	Задача
	якою слідує 0. Визначити порядковий номер найменшого з них.
23	Дано 20 дійсних чисел. Обчислити різницю між максимальним і мінімальним з них.
24	Логічній змінній t присвоїти значення *true* або *false* залежно від того, можна чи ні натуральне число n представити у вигляді суми трьох повних квадратів.
25	Логічній змінній p присвоїти значення *true*, якщо ціле n (n> 1) - просте число, і значення *false* в іншому випадку.
26	Підрахувати k - кількість цифр в десятковому записі цілого не-від'ємного числа n.
27	Логічній змінній t присвоїти значення *true* або *false* залежно від того, чи є задане натуральне число k ступенем 3 чи ні.
28	Дано натуральні числа k і n. Обчислити суму цифр між k-им і n-им елементами послідовності ступенів числа 3: 1392781... за умови, що самі граничні елементи в суму не включаються. (напр., для k=2 і n=6 в суму цифр включаються 3-й, 4-й і 5-й елементи послідовності і сума дорівнює 9+2+7).
29	Вивести всі прості дільники заданого натурального числа.
30	Дано ціле n> 2. Вивести всі прості числа з діапазону [2, n].

13.2 Лабораторна робота № 2. Розроблення програм для алгорифмів Маркова

Мета роботи: розвинути аналітичне та логічне мислення, математичну інтуїцію, за допомогою розроблення алгоритмів задач для логічних машин.

Завдання до виконання

1. Розробити алгоритм вирішення задачі.
2. Запропонувати алгоритм вирішення задачі. Номер варіанта задачі відповідає номеру студента в списку групи.

3. Заповнити таблицю (табл. 13.2) вписавши алгоритм програми, коментуючи кожен рядок.

4. Для запропонованого алгоритму надати код.

Таблиця 13.2 – Алгоритм вирішення задачі

	зразок		заміна	коментар
		>		
		>		

Варіанти завдань

1. *A = {f, h, p}.* У слові *P* замінити всі пари *ph* на *f.*

2. A = {f, h, p}. У слові *P* замінити на *f* тільки першу пару *ph*, якщо така є.

3. *A = {a, b, c}.* Приписати слово *bac* зліва до слова *P*.

4. *A = {a, b, c}.* Замінити слово *P* на порожнє слово, тобто видалити з *P* все символи.

5. *A = {a, b, c}.* Замінити будь вхідний слово на слово *a*.

6. Побудувати НАМ, що не міняє вхідний слово (при будь-якому алфавіті *A*).

7. *A = {| }.* Вважаючи слово *P* записом числа в одиничної системі числення, отримати залишок від ділення цього числа на 2, тобто отримати слово з однієї палички, якщо число непарній, або пусте слово, якщо число парне.

8. *A = {| }.* Вважаючи слово *P* записом позитивного числа в одиничної системі числення, зменшити це число на *1*.

9. *A = {| }.* Вважаючи слово *P* записом числа в одиничної системі числення, збільшити це число на *2*.

10. *A = {0,1,2}.* Вважаючи слово *P* записом числа в трійчастий системі числення, отримати залишок від ділення цього числа на *2*, тобто отримати слово *1*, якщо число непарній, або слово 0, якщо число парне. (Зауваження: у парному трійковому числі має бути парна кількість цифр *1*.)

11. *A = {a, b, c}.* Визначити, чи входить символ a в слово *P*. Відповідь (вихідне слово): слово *a*, якщо входить, або пусте слово, якщо не входить.

12. *A = {a, b}.* Якщо в слово *P* входить більше символів a, ніж символів *b*, то в якості відповіді видати слово з одного символу *a*, якщо в *P* рівну кількість *a* і *b*, то в якості відповіді видати пусте слово, а інакше видати відповідь *b*.

13. *A = {0,1,2,3}.* Перетворити слово *P* так, щоб спочатку йшли всі парні цифри (*0 і 2*), а потім – всі непарні.

14. *A = {a, b, c}.* Перетворити слово *P* так, щоб спочатку йшли всі символи *a*, потім – все символи *b* і в кінці – все символи *c*.

15. *A = {a, b, c}.* Визначити, зі скількох різних символів складено слово *P*; відповідь отримати в одиничної системі числення (наприклад: *acaac* → | |).

16. *A = {a, b, c}.* У не порожнього слові *P* подвоїти перший символ, тобто приписати цей символ зліва до *P*.

17. *A = {a, b, c}.* За першим символом не порожнього слова *P* вставити символ *c*.

18. *A = {a, b, c}.* З слова *P* видалити другий символ, якщо такий є.

19. *A = {a, b, c}.* Якщо в слові *P* не менше двох символів, то переставити два перших символу.

20. *A = {0,1,2}.* Вважаючи не порожнє слово *P* записом трійкового числа, видалити з цього запису все незначущі нулі.

21. *A = {a, b, c}.* Приписати слово *abc* справа до слова *P*.

22. *A = {a, b, c}.* Видалити з не порожнього слова *P* його останній символ.

23. *A = {0,1}.* Вважаючи не порожнє слово *P* записом числа в двійковій системі, отримати двійкове число, що дорівнює збільшеному у 4 рази числу *P* (наприклад: *101 → 10100*).

24. *A = {0, 1}.* Вважаючи не порожнє слово *P* записом числа в двійковій системі, отримати двійкове число, що дорівнює неповної частці від ділення числа *P* на *2* (наприклад: *1011 → 101*).

25. *A = {a, b}.* У слові *P* усі символи *a* замінити на *b*, а усі (колишні) символи *b* – на *a*.

Контрольні питання

1. Що таке алгорифм Маркова?
2. Система підстановок алгорифм Маркова.
3. Правила виконання алгорифм Маркова.
4. Варіанти закінчення виконання програми.

13.3 Лабораторна робота № 3. Розроблення програм із використанням рекурсивних функцій

Мета роботи: познайомитися з поняттям "рекурсія" і особливостями рекурсивних процедур і функцій;. освоїти способи програмування алгоритмів з використанням рекурсії.

Завдання до виконання

Під час виконання цієї роботи слід пам'ятати, що:

1) вихідні дані повинні включати позитивні, негативні числа і нулі;

2) програма повинна працювати з будь-яким допустимим набором даних.

Варіанти завдань

1. Дано натуральне число n. Виведіть усі числа від 1 до n.

2. Дано два цілих числа A і B (кожне в окремому рядку). Виведіть усі числа від A до B включно, в порядку зростання, якщо A B, або в порядку убування в іншому випадку.

3. У теорії обчислюваності важливу роль відіграє функція Аккермана A (m, n), яка задана в такий спосіб:

Дано два цілих невід'ємних числа m і n, кожне в окремому рядку. Виведіть A (m, n).

4. Дано натуральне число N. Виведіть слово YES, якщо число N є точною ступенем двійки, або слово NO в іншому випадку.

5. Дано натуральне число N. Обчисліть суму його цифр.

6. Дано натуральне число N. Виведіть всі його цифри по одній, в зворотному порядку, розділяючи їх пробілами або новими рядками.

7. Дано натуральне число N. Виведіть всі його цифри по одній, в звичайному порядку, розділяючи їх пробілами або новими рядками.

8. Дано натуральне число n1. Перевірте, чи є воно простим. Програма повинна вивести слово YES, якщо число просте і NO, якщо число складене.

9. Дано натуральне число n1. Виведіть всі прості множники цього числа в порядку неспадання з урахуванням кратності.

10. Дано слово, що складається лише з малих латинських букв. Перевірте, чи є це слово паліндромом. Виведіть YES або NO.

11. Дана послідовність натуральних чисел (одне число в рядку), що завершується числом 0. Виведіть всі непарні числа з цієї послідовності, зберігаючи їх порядок.

12. Дана послідовність натуральних чисел (одне число в рядку), що завершується числом 0. Виведіть перше, третє, п'яте і т. д. з введених чисел. Завершальний нуль виводити не треба.

13. Дана послідовність натуральних чисел (одне число в рядку), що завершується числом 0. Визначте найбільше значення числа в цій послідовності.

14. Дано натуральні числа k і s. Визначте, скільки існує k-значних натуральних чисел, сума цифр яких дорівнює d. Запис натурального числа не може починатися з цифри 0.

15. Дано числа a і b. Визначте, скільки існує послідовностей з a нулів і b одиниць, в яких ніякі два нуля не варті поруч.

16. Дано число n, десяткова запис якого не містить нулів. Отримайте число, записане тими ж цифрами, але в протилежному порядку.

Контрольні питання

1. Що таке «рекурсія»?
2. Як оголошується рекурсивна підпрограма?
3. У чому переваги і недоліки використання рекурсії?
4. Які види рекурсій бувають і в чому їх особливість?
5. Як рекурсія пов’язана з ітераціями?
6. Що таке рекурентність обчислень?
7. Як реалізуються циклічні обчислювальні процеси у функційному програмуванні?

13.4 Лабораторна робота № 4. Розроблення програм із використанням алгоритмів перебору й сортування

Мета роботи: познайомитися з особливостями алгоритмів перебору і сортування даних.

Завдання до виконання

Освоїти способи програмування алгоритмів перебору і сортування даних .

Теоретичні відомості

Якщо в завданні необхідно заповнити масив випадковими позитивними і негативними числами таким чином, щоб всі числа по модулю були різними. Це

означає, що в масиві не може бути не тільки двох рівних чисел, але не може бути двох рівних по модулю.

Якщо масив зсувається на один крок вліво, то на місце елемента з індексом і записується той елемент, який знаходиться на місці і + 1. Тобто на місце поточного елемента записується наступний за ним. В останній осередок масиву записувати нічого. За умовою завдання туди слід записати число 0.

Таким чином, зрушення масиву на один крок вліво – це цикл від першого елемента до передостаннього включно, в тілі якого відбувається запис значення з наступної комірки в поточну комірку (arr [i] : = arr [i + 1]). У передостанню клітинку записується останній елемент. Після циклу присвоюється 0 в останній осередок масиву.

Якщо масив зсувається на один крок вправо, то його слід "перебирати" з кінця. На місце елемента і присвоюється стоїть перед ним (i-1). У звільнилася першу клітинку записується нуль.

Зрушення масиву на один крок вправо – це цикл від останнього елемента до другого включно, в тілі якого відбувається запис значення з попередньої комірки в поточну (arr [i]: = arr [i-1]). Після циклу в перший осередок масиву записується нуль.

Кількість кроків зсуву визначається зовнішнім циклом.

Варіанти завдань

1. Напишіть програму, яка виводить всі перестановки довжини N в зворотному лексикографічному порядку. Наприклад, якщо N = 3, то програма повинна вивести 321, 312, 231, 213, 132, 123.

2. Нехай дано масив A, що містить 10 цілих чисел. Відсортувати елементи масиву в низхідному порядку за допомогою методу вставки ..

3. Заданий масив, що містить 20 цілих чисел. Знайти позицію (позиції) елемента, який має мінімальне значення. Якщо таких елементів декілька, сформувати додатковий масив індексів.

4. Напишіть програму, яка виводить всі перестановки довжини N такі, що будь-які два сусідні елементи відрізняються як мінімум на два.

Наприклад, якщо N = 4, то це 1324 і 2413.

5. Є одномірний масив довжиною N = 15. Відсортувати за зростанням за допомогою методу простого обміну ті елементи масиву, які розташовуються на непарних позиціях.

6. Напишіть програму, яка виводить всі послідовності довжини N з 0 і 1 без 2-х одиниць поспіль. Наприклад, якщо N = 3, то це 000, 001, 010, 100, 101.

7. Знайти в масиві ті елементи, значення яких менше середнього арифметичного, взятого від всіх елементів масиву.

8. Напишіть програму, яка виводить всі поєднання з N по K в зворотному лексикографічному порядку. Наприклад, якщо N = 5 і K = 2, то вона повинна вивести 54, 53, 52, 51, 43, 42, 41, 32, 31, 21.

9. Є двовимірний масив розмірністю NxN, де N = 10. Відсортувати всі стовпці методом простого обміну так, щоб елементи в них розташовувалися за спаданням.

10. Напишіть програму, яка виводить всі розкладання заданого числа N на складові. Наприклад, якщо N = 4, то це 1 + 1 + 1 + 1, 1 + 1 + 2, 1 + 3, 2 + 2 і 4.

11. Всі негативні елементи масиву замінити нулями. Видати повідомлення про кількість таких замін.

12. Є одномірний масив довжиною N=15. Відсортувати за спаданням методом вибору ті елементи масиву, які є парними числами.

13. Двадцяти змінним масиву присвоїти одне і те ж значення, що дорівнює найбільшому з них.

14. Напишіть програму, яка виводить всі перестановки довжини N такі, що будь-які два сусідні елементи відрізняються як мінімум на три.

15. Напишіть програму, яка виводить всі послідовності довжини N з 0 і 1 без 2-х нулів підряд.

16. Яких чисел в масиві більше: позитивних чи негативних? (Масив може містити нульові значення).

17. Знайти кількість одиниць в масиві.

18. Нехай дано масив A, що містить 10 цілих чисел. Відсортувати елементи масиву в висхідному порядку за допомогою методу вставки.

19. Дан масив з 10 елементів. Перші 4 впорядкувати по зростанню, останні 4 по спадаючій.

20. Стиснути масив, видаливши з нього всі елементи, величина яких знаходиться в інтервалі [a, b]. Вивільнені в кінці масиву елементи заповнити нулями.

21. заданий масив із 15 цілих чисел на відрізку [-5; 5]. Впорядкувати масив, видаливши повторювані елементи.

22. В одновимірному масиві, що складається з n дійсних елементів: Перетворити масив таким чином, щоб спочатку розташовувалися всі елементи, що відрізняються від максимального не більше ніж на 20%, а потім – всі інші.

23. В одновимірному масиві, що складається з n дійсних елементів: Впорядкувати по зростанню окремо елементи, що стоять на парних місцях, і елементи, що стоять на непарних місцях.

24. В одновимірному масиві, що складається з n дійсних елементів: Замінити всі негативні елементи масиву їх квадратами і впорядкувати елементи масиву по зростанню.

25. В одновимірному масиві, що складається з n дійсних елементів: Перетворити масив таким чином, щоб спочатку розташовувалися всі елементи, ціла частина яких не перевищує 1, а потім – всі інші.

26. В одновимірному. масиві, що складається з n дійсних елементів: Впорядкувати елементи масиву по спадаючій модулів елементів.

27. Створити одновимірний масив a з 20 цілих чисел. Створити одновимірний масив b, що складається з зворотного масиву a. Вивести на екран масив b. Відсортувати отриманий масив в порядку зростання і вивести результат на екран.

28. Створити одновимірний масив a з 10 цілих чисел. Перетворити масив таким чином, щоб спочатку розташовувалися всі негативні елементи, а потім – всі позитивні. Вивести на екран створений масив. Відсортувати отриманий масив в порядку зростання і вивести його на екран.

29. Створити одновимірний масив, який складається з чисел Фіббоначі, і вивести його на екран. Відсортувати отриманий масив в порядку спадання і вивести результат на екран.

Контрольні питання

1. Що таке сортування даних?
2. Які алгоритми сортування відомі?
3. Які алгоритми сортування даних вважаються більш ефективними та чому?
4. У чому полягає алгоритм сортування вибором?
5. У чому полягає алгоритм сортування вставками?
6. У чому полягає алгоритм сортування обміном?
7. У чому полягає алгоритм сортування Шелла?
8. У чому полягає алгоритм сортування злиттям?
9. У чому полягає алгоритм сортування підрахунком?
10. Які характеристики використовуються для порівняння алгоритмів сортування?
11. Яким чином можна поліпшити ефективність алгоритму сортування обміном?

13.5 Лабораторна робота № 5. Складання алгоритму пошуку найкоротшого шляху. Алгоритм Дейкстри

Мета і задачі: навчитися реалізовувати алгоритм Дейкстри для пошуку найкоротшого шляху від однієї вершини графа до інших.

Теоретичні відомості і методичні вказівки

Алгоритм Дейкстри – алгоритм на графах, відкритий Дейкстрою. Знаходить найкоротший шлях від однієї вершини графа до всіх інших вершин. Класичний алгоритм Дейкстри працює тільки для графів без циклів від'ємної довжини.

Наглядне використання алгоритму:

1. Дана мережа автомобільних доріг, що з'єднують міста Львівської області. Знайти найкоротшу відстань від Львова до кожного міста області, якщо рухатись можна тільки по дорогах.
2. Дана карта велосипедних доріжок Латвії та Білорусі. Знайти мінімальну відстань, яку треба проїхати, щоб дістатися від Риги до Бобруйська.
3. Є план міста з нанесеними на нього місцями розміщення пожежних частин. Знайти найближчу до кожного дому пожежну станцію.

Розглянемо приклад

Дано неорієнтований зв'язний граф **G(V, U)** (рис. 13.1). Знайти відстань від вершини **a** до всіх інших вершин **V**.

Розв'язання (рис. 13.1–13.12)

Зберігатимемо поточну мінімальну відстань до всіх вершин **V** (від даної вершини **a**) і на кожному кроці алгоритму намагатимемося зменшити цю відстань. Спочатку встановимо відстані до всіх вершин рівними нескінченості, а до вершини **a** – нулю.

Хай потрібно знайти відстані від 1-ої вершини до всіх інших. Кружечками позначені вершини, лініями – шляхи між ними («дуги»). Над дугами позначена їх «ціна» – довжина шляху. Надписом над кружечком позначена поточна найкоротша відстань до вершини.

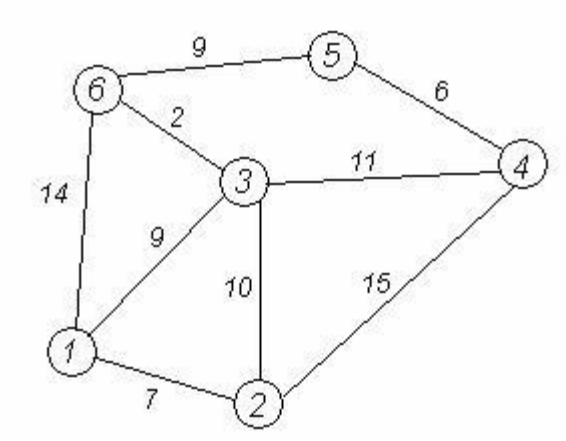


Рисунок 13.1 – Неорієнтований зв'язний граф ***G(V, U)***

Крок 1

Ініціалізація. Відстань до всіх вершин графа **V** = ∞. Відстань до **a = 0**. Жодна вершина графа ще не опрацьована.

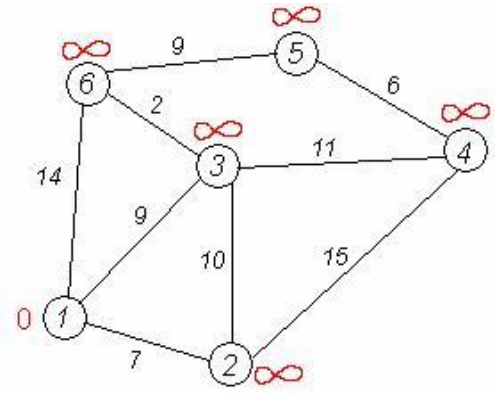


Рисунок 13.2 – Ілюстрація. Крок 1

Крок 2

Знаходимо таку вершину (із ще не оброблених), поточна найкоротша відстань до якої мінімальна. У нашому випадку це вершина 1. Обходимо всіх її сусідів і, якщо шлях в сусідню вершину через 1 менший за поточний мінімальний шлях у цю сусідню вершину, то запам'ятовуємо цей новий, коротший шлях як поточний найкоротший шлях до сусіда.

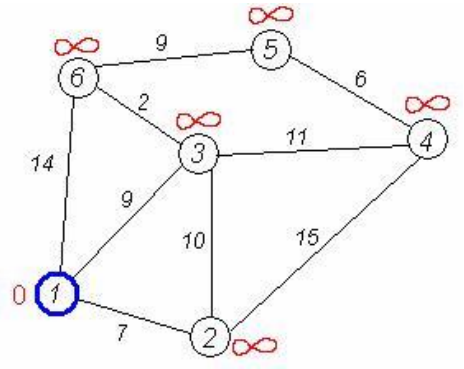


Рисунок 13.3 – Ілюстрація. Крок 2

Крок 3

Перший за порядком сусід 1-ї вершини – 2-га вершина. Шлях до неї через 1-шу вершину дорівнює найкоротшій відстані до 1-ї вершини + довжина дуги між 1-ю та 2-ю вершиною, тобто 0 + 7 = 7. Це менше поточного найкоротшого шляху до 2-ї вершини, тому найкоротший шлях до 2-ї вершини дорівнює 7.

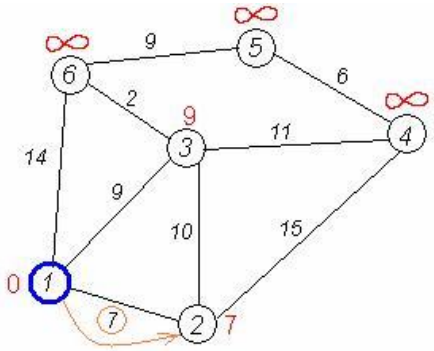


Рисунок 13.4 – Ілюстрація. Крок 3

Кроки 4, 5

Аналогічну операцію проробляємо з двома іншими сусідами 1-ї вершини – 3-ю та 6-ю.

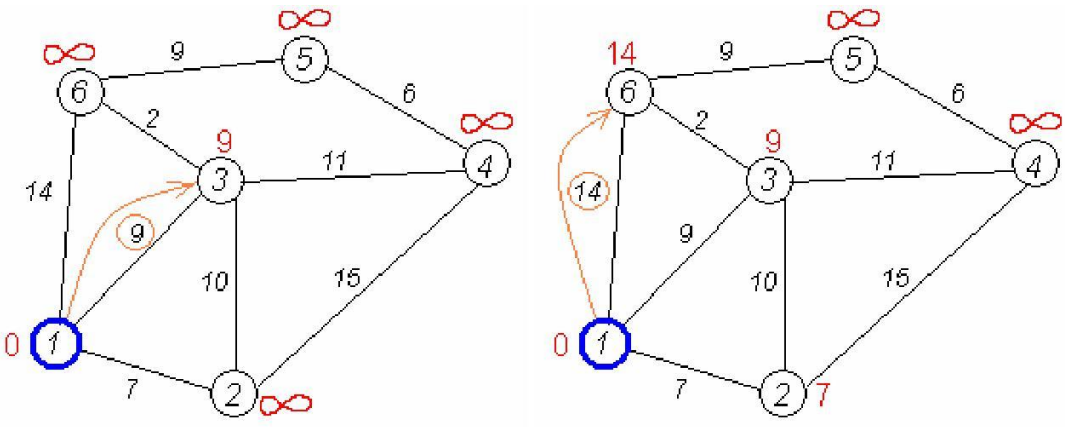


Рисунок 13.5 – Ілюстрація. Кроки 4, 5

Крок 6

Усі сусіди вершини 1 перевірені. Поточна мінімальна відстань до вершини 1 вважається остаточною і обговоренню не підлягає (те, що це дійсно так, вперше довів Дейкстра).

Тому викреслимо її з графа, щоб відмітити цей факт.

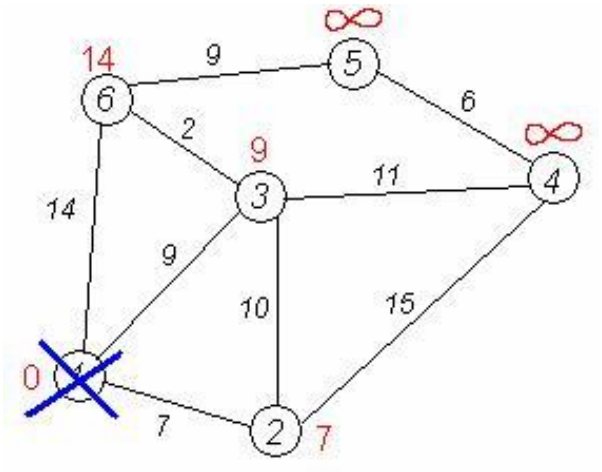


Рисунок 13.6 – Ілюстрація. Крок 6

Крок 7

Практично відбувається повернення до кроку 2. Знову знаходимо «найближчу» необроблену (невикреслену) вершину.

Це вершина 2 з поточною найкоротшою відстанню до неї = 7. І знову намагаємося зменшити відстань до всіх сусідів 2-ої вершини, намагаючись пройти в них через 2-у. Сусідами 2-ої вершини є 1, 3, 4.

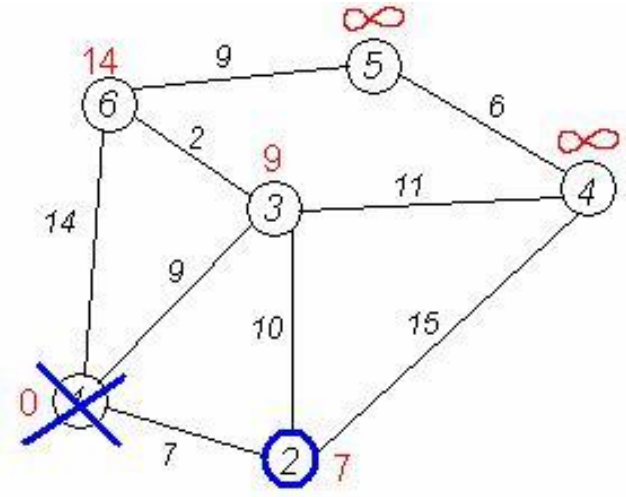


Рисунок 13.7 – Ілюстрація. Крок 7

Крок 8

Перший (за порядком) сусід вершини № 2 – 1-а вершина. Але вона вже оброблена (або викреслена – див. крок 6). Тому з 1-ою вершиною нічого не робимо.

Крок 8 (з іншими вхідними даними)

Інший сусід вершини 2 — вершина 4. Якщо йти в неї через 2-у, то шлях буде = найкоротша відстань до 2-ої + відстань між 2-ою і 4-ою вершинами = 7 + 15 = 22. Оскільки 22 < ∞, встановлюємо відстань до вершини № 4 рівним 22.

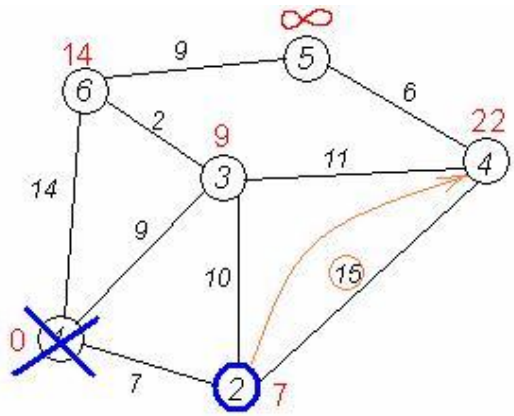


Рисунок 13.8 – Ілюстрація. Крок 8

Крок 9

Ще один сусід вершини 2 – вершина 3. Якщо йти в неї через 2-у, то шлях буде = 7 + 10 = 17. Але 17 більше за відстань, що вже запам'ятали раніше до вершини № 3 і дорівнює 9, тому поточну відстань до 3-ої вершини не міняємо.

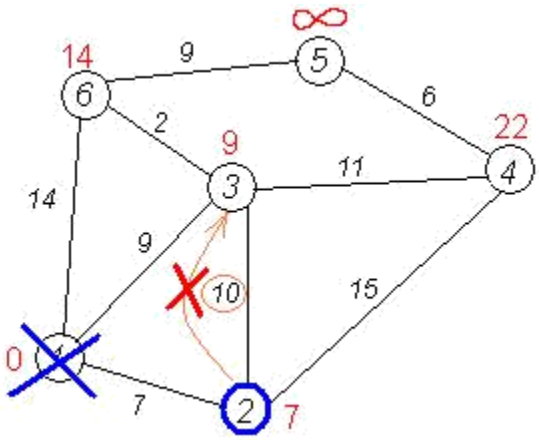


Рисунок 13.9 – Ілюстрація. Крок 9

Крок 10

Усі сусіди вершини 2 переглянуті, заморожуємо відстань до неї і викреслюємо її з графа.

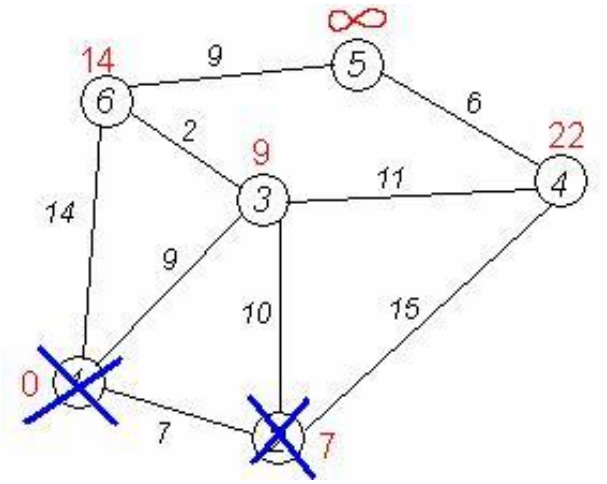


Рисунок 13.10 – Ілюстрація. Крок 10

Кроки 11–15

За вже «відпрацьованій» схемі повторюємо кроки 2–6. Тепер «найближчою» виявляється вершина № 3. Після її «оброблення» отримаємо такі результати:

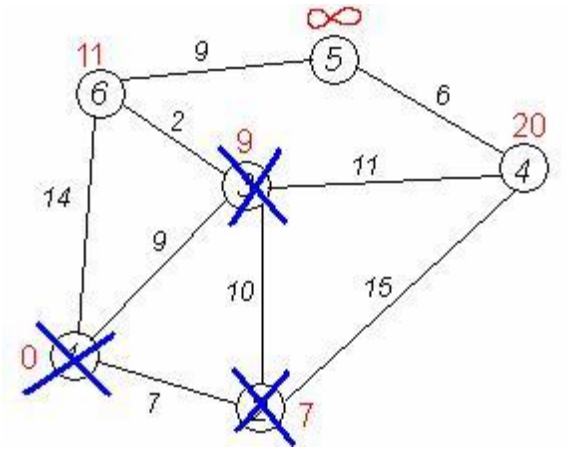


Рисунок 13.11 – Ілюстрація. Кроки 11–15

Наступні кроки

Проробляємо те саме з вершинами, що залишилися (№ за порядком: 6, 4 і 5):

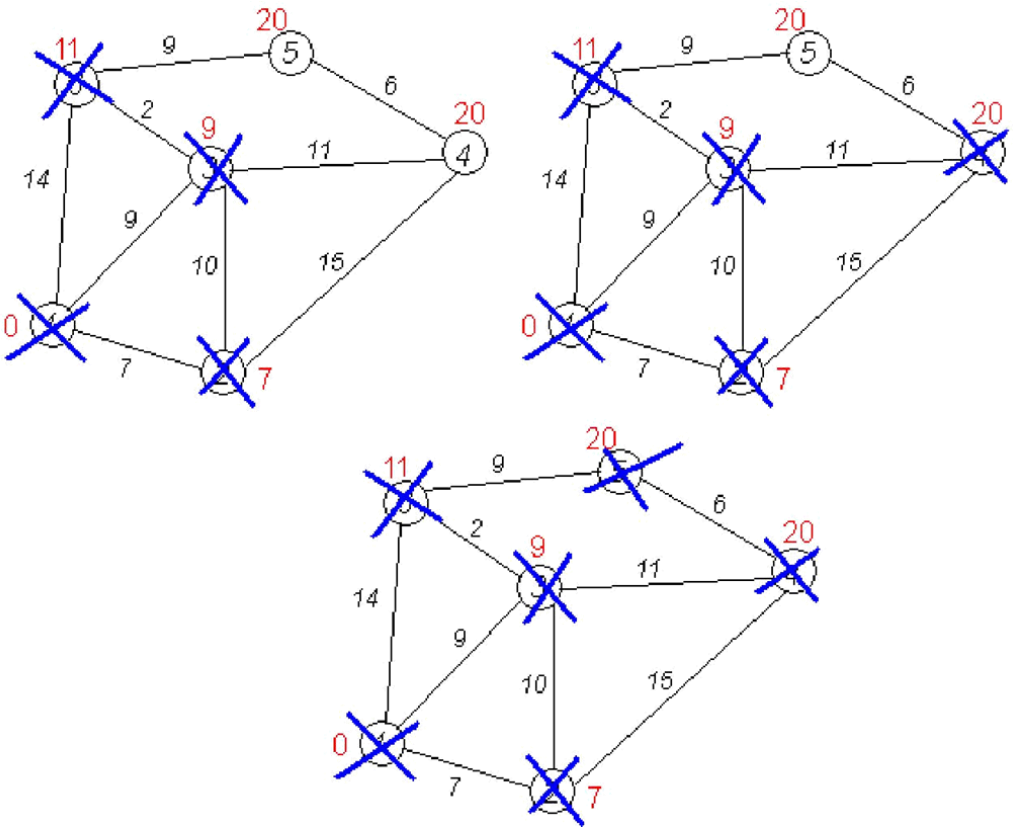


Рисунок 13.12 – Ілюстрація. Наступні кроки

Завершення виконання алгоритму

Алгоритм закінчує роботу, коли викреслені всі вершини. Результат його роботи видно на останньому малюнку: найкоротший шлях від 1-ої вершини до 2-ої становить 7, до 3-ої – 9, до 4-ої – 20, до 5-ої – 20, до 6-ої – 11 умовних одиниць.

Порядок виконання і звітування

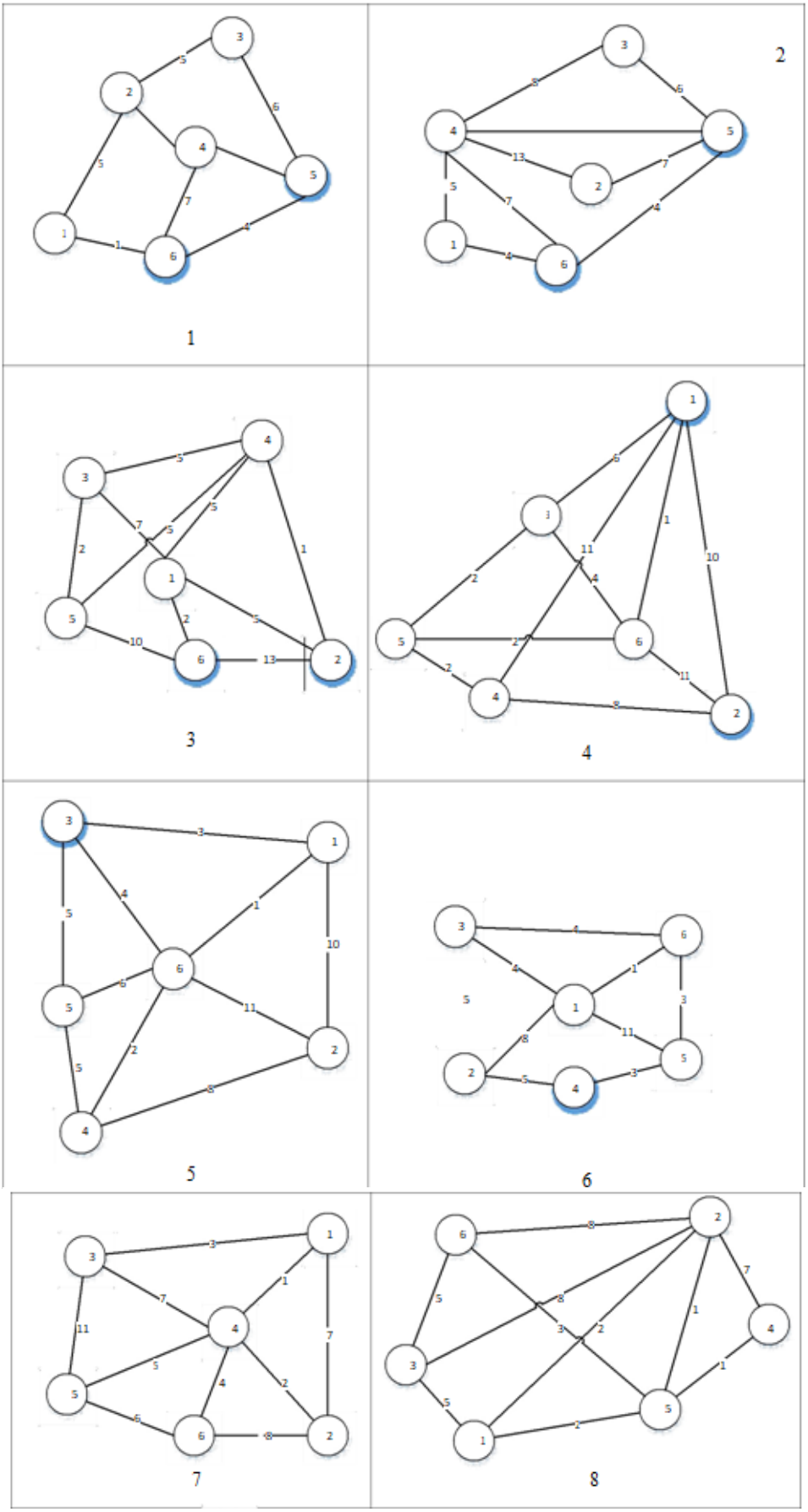
1. Написати програму, яка реалізує алгоритм Дейкстри

2. Відкомпілювати та відлагодити програму.
3. Протестувати роботу програми використовуючи різні вхідні дані
4. Відповісти на контрольні запитання.
5. Зробити висновки.

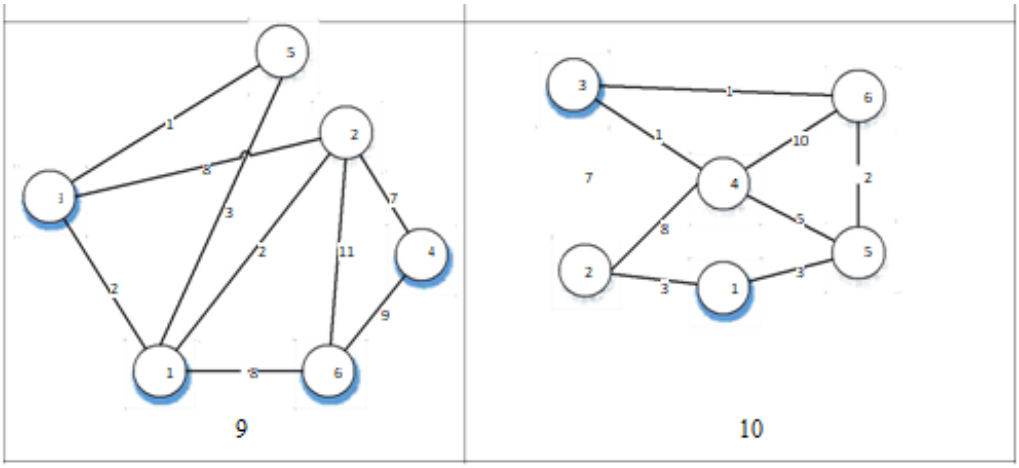
6. Звіт по лабораторній роботі має складатися з титульної сторінки, мети , лістингів програм, висновків по роботі.

Варіанти завдань (табл. 13.3)

Таблиця 13.3 – Варіанти завдань



Продовження таблиці 13.3



Контрольні питання

1. Що таке граф?
2. Які є види графів?
3. Як можуть задаватися графи?
4. Що таке вершина у графі?
5. Що таке ребро у графі?
6. Які алгоритми для пошуку мінімальної відстані у графі вам відомі?

13.6 Лабораторна робота № 6. Транспортні мережі

Мета і задачі: побудова максимального потоку в транспортній мережі. Спрощення (оптимізація) транспортної мережі.

Завдання

1. Вивчити теоретичний матеріал.

2. Вивчити алгоритм побудови максимального потоку в транспортній мережі.

3. Для індивідуального варіанта реалізувати алгоритм вручну (отримати максимальний потік в мережі), зобразивши кожен збільшувальний ланцюг і результати перерахунку потоку.

4. Виконати перевірку отриманого результату.

5. Провести аналіз отриманого результату і запропонувати рекомендації щодо збільшення максимального потоку (допускається коригування в бік збільшення пропускної здатності тільки однієї дуги).

6. Виконати спрощення (оптимізацію) транспортної мережі, отриманої в п. 4.

Методичні вказівки

1. Вивчити теоретичний матеріал по транспортних мережах.

2. При реалізації алгоритму задати початковий потік в транспортній мережі рівним 0.

3. При побудові збільшувальних ланцюгів стежити за виконанням обмежуючих умов для кожної дуги.

4. При перевірці результату потік в будь-якому розрізі транспортної мережі повинен дорівнювати максимальному.

5. Вибір варіанта: студент вибирає № варіанта завдання (рис. 13.13–13.22), визначивши значення t, де t = N/10 – залишок від ділення без остачі числа N (порядковий номер студента у основному списку групи).

Варіанти транспортних мереж

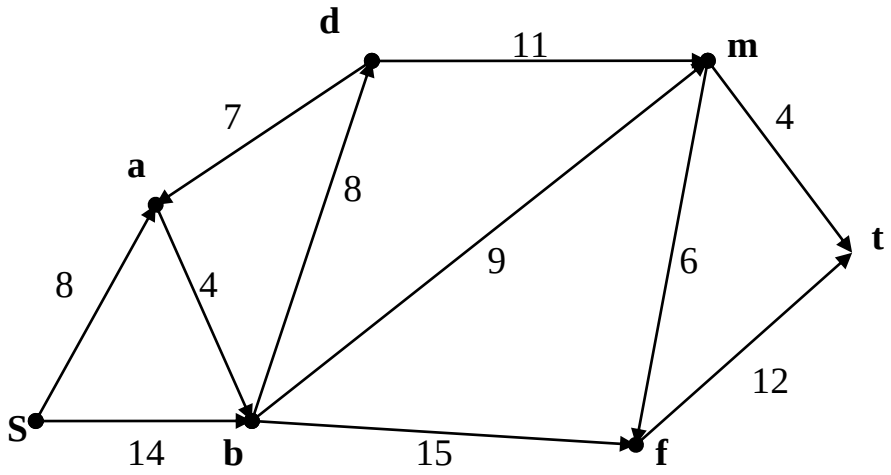


Рисунок 13.13 – Транспортна мережа варіанта 0

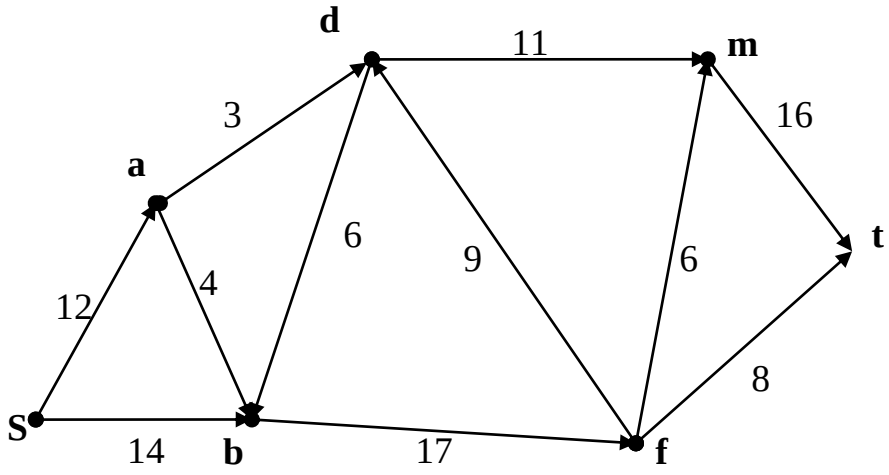


Рисунок 13.14 – Транспортна мережа варіанта 1

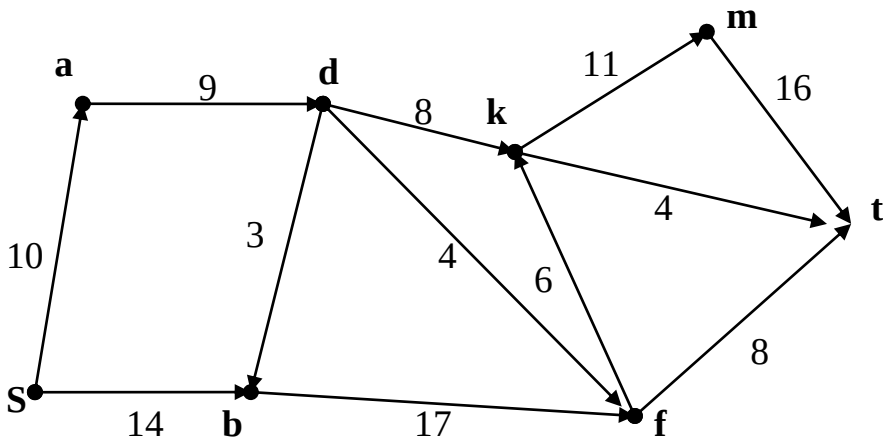


Рисунок 13.15 – Транспортна мережа варіанта 2

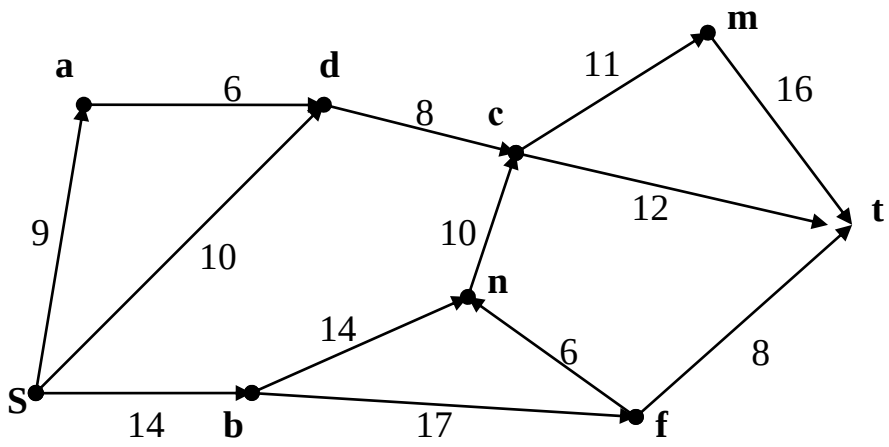


Рисунок 13.16 – Транспортна мережа варіанта 3

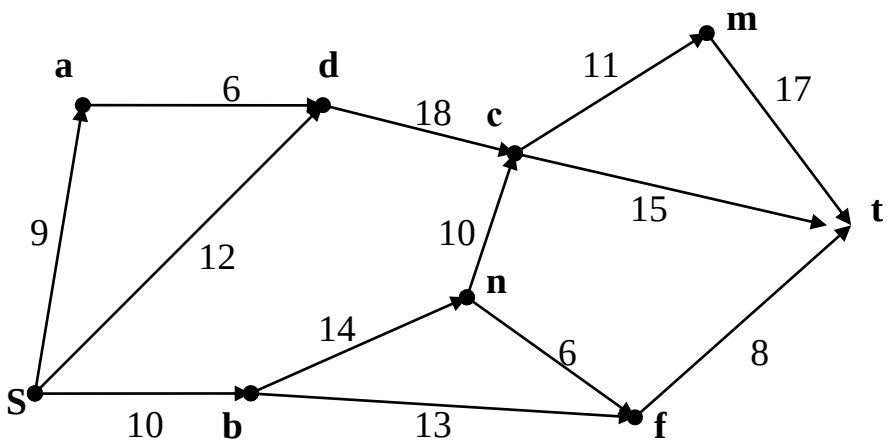


Рисунок 13.17 – Транспортна мережа варіанта 4

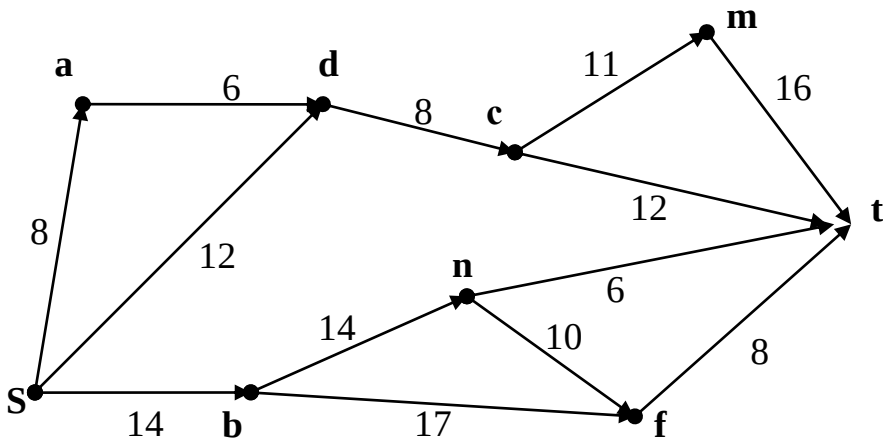


Рисунок 13.18 – Транспортна мережа варіанта 5

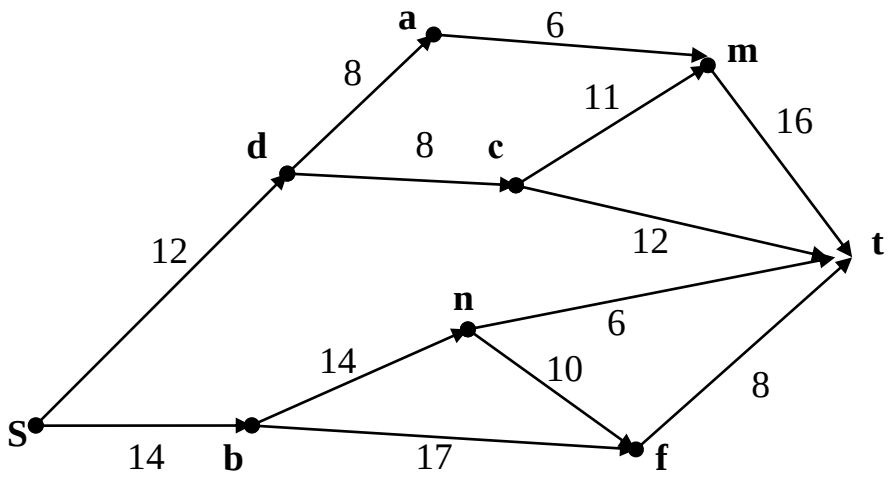


Рисунок 13.19 – Транспортна мережа варіанта 6

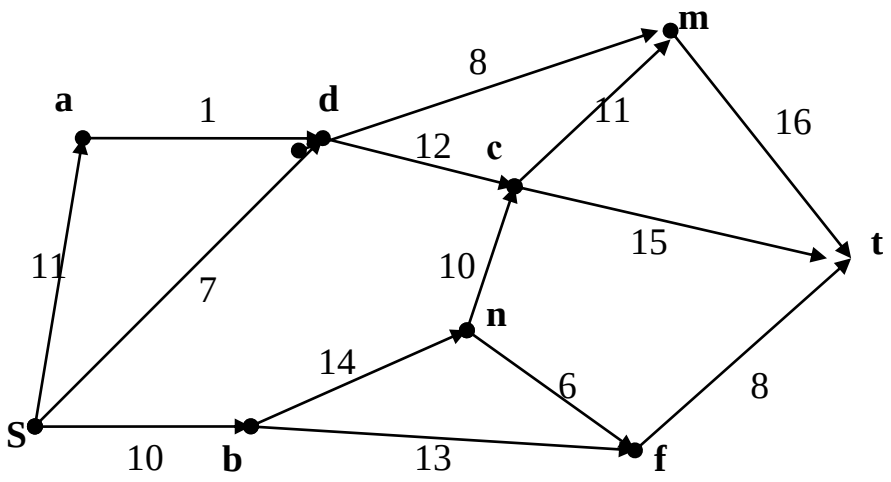


Рисунок 13.20 – Транспортна мережа к варіанта 7

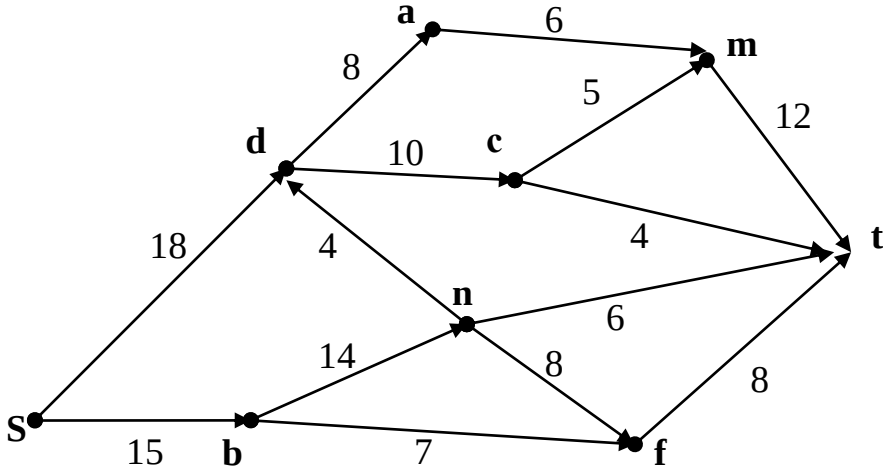


Рисунок 13.21 – Транспортна мережа варіанта 8

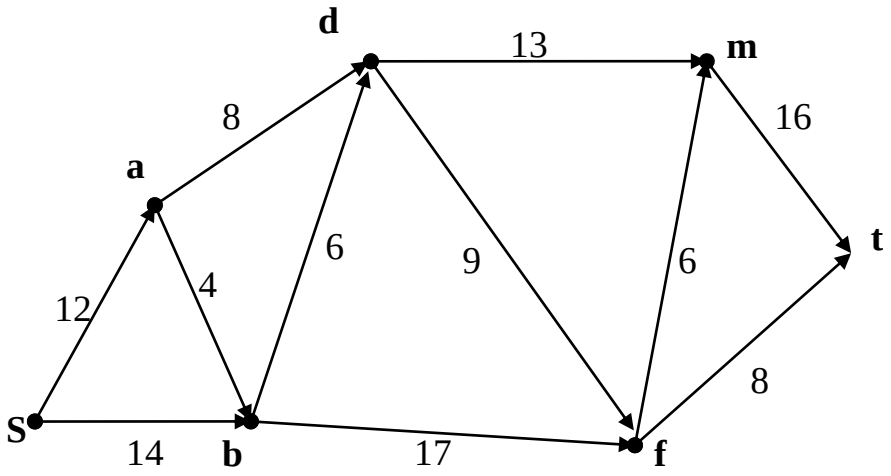


Рисунок 13.22 – Транспортна мережа варіанта 9

Контрольні питання

1. Транспортна мережа і її математична модель. Постановка оптимізаційної задачі на транспортній мережі.

2. Пропускна здатність дуги й потік у дузі ТМ. Поняття потоку в ТМ; умови для потоку.

3. Розріз ТМ. Прямі та зворотні дуги. Пропускна здатність розрізу. Теорема Форда – Фалкерсона.

4. Збільшувальний ланцюг в ТМ. Алгоритм побудови збільшувального ланцюга. Збільшення потоку в ТМ з використанням збільшувальних ланцюгів.

13.7 Лабораторна робота № 7. Мережева модель проєкту типу I

Мета і задачі: Навчитися визначати критичний шлях (шляхи) в мережевий моделі проєкту, заданого навантаженим орієнтованим графом без контурів типу I «подія-операція». Визначення резервів операцій і подій. Якісний аналіз проєкту.

Завдання

1. Ознайомитися із теоретичними відомостями за темою роботи.

2. Вивчити основні поняття для дослідження мережевої моделі проєкту типу I «подія-операція».

3. Вивчити структуру подій (вершин мережі) і їх часову прив'язку до початку виконання проєкту.

4. Для індивідуального варіанта виконати пошук критичного шляху в мережі проєкту, визначити мінімально можливий час його реалізації (виконання).

5. Виконати розрахунки резервів різного виду для подій і робіт що не лежать на критичному шляху; оцінити. Результати розрахунків представити в табличному вигляді.

6. Розробити рекомендації щодо оптимізації мережевої моделі проєкту.

Методичні вказівки

1. Вивчити теоретичний матеріал до лабораторної роботи.

2. При необхідності, привести мережеву модель проєкту до стандартного виду (по одній вершині тільки з вихідними (початок проєкту) і тільки з вхідними (закінчення проєкту) дугами).

3. При виконанні п. 3 розписати процедуру отримання вершинами часових відміток для 3-х вершин (для інших – виконати усно).

4. Вибір варіанта: студент вибирає № варіанта завдання (рис. 13.23–13.31), визначив значення t, де $t = N/9$ – залишок від ділення без остачі числа N (порядковий номер у основному списку групи).

Контрольні питання

1. Мережеві моделі проєкту типу I. Графічне подання; інтерпретація.

2. Постановка оптимізаційної задачі на мережевий моделі проєкту типу I.

3. Алгоритм проставляння для вершин мережевої моделі проєкту I відміток часу t_i та t_i^*.

4. Визначення мінімально можливого часу виконання проєкту.

5. Визначення ранніх і пізніх термінів виконання робіт мережевої моделі I.

6. Алгоритм перетворення мережевих моделей проєктів типу I в тип II.

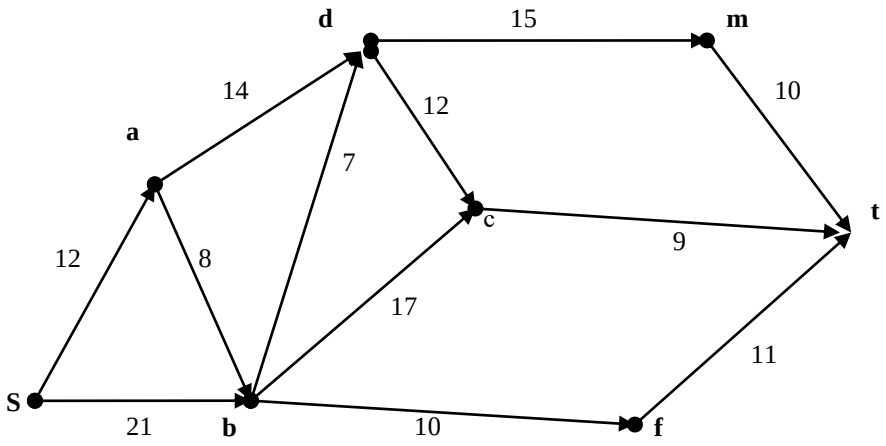


Рисунок 13. 23 – Мережева модель варіанта 0

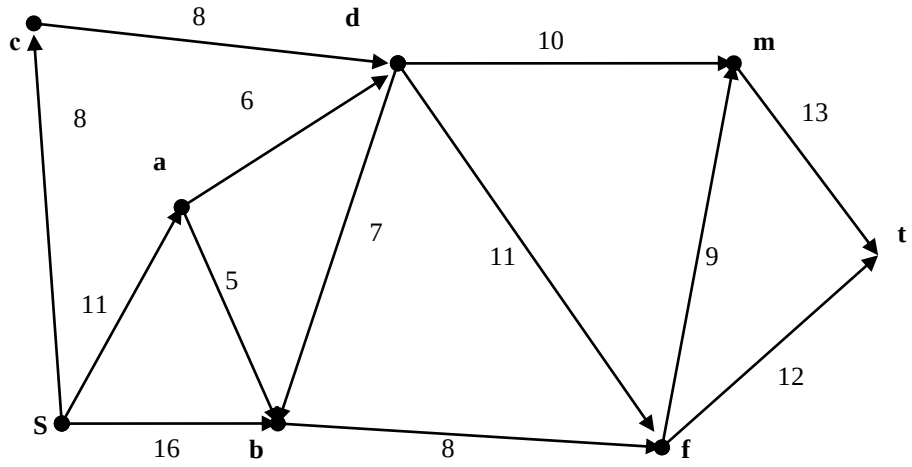


Рисунок 13.24 – Мережева модель варіанта 1

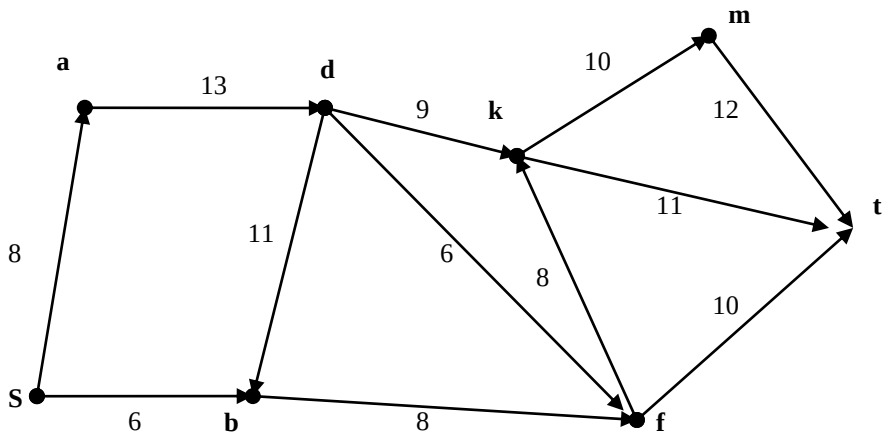


Рисунок 13.25 – Мережева модель варіанта 2

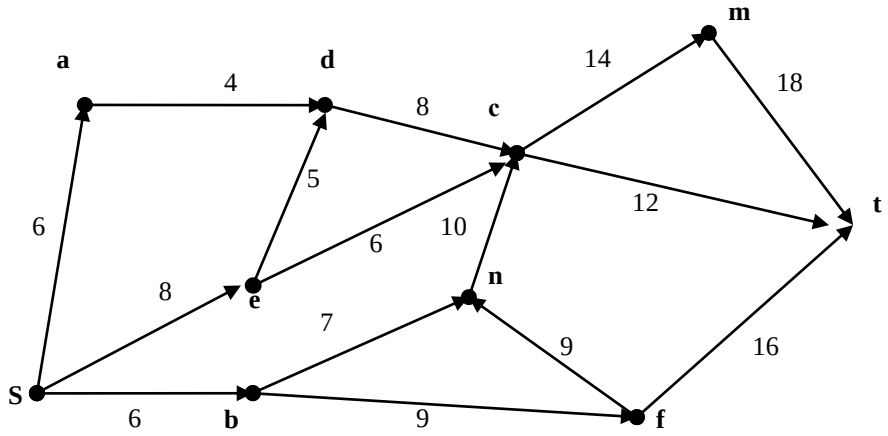


Рисунок 13.26 – Мережева модель варіанта 3

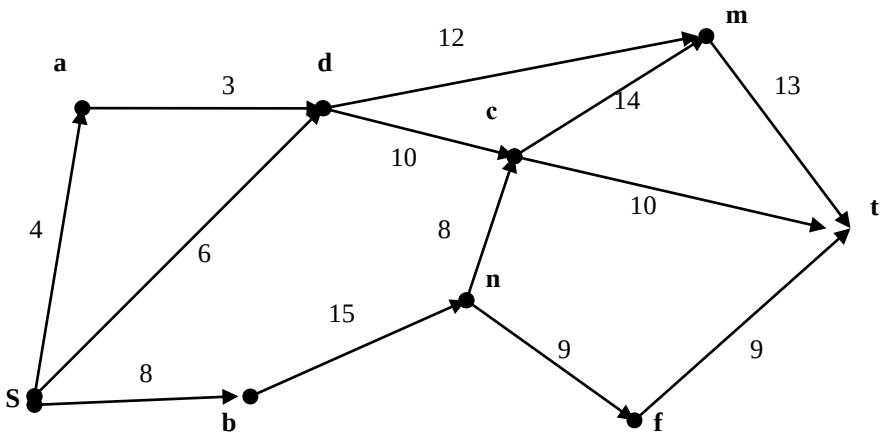


Рисунок 13.27 – Мережева модель варіанта 4

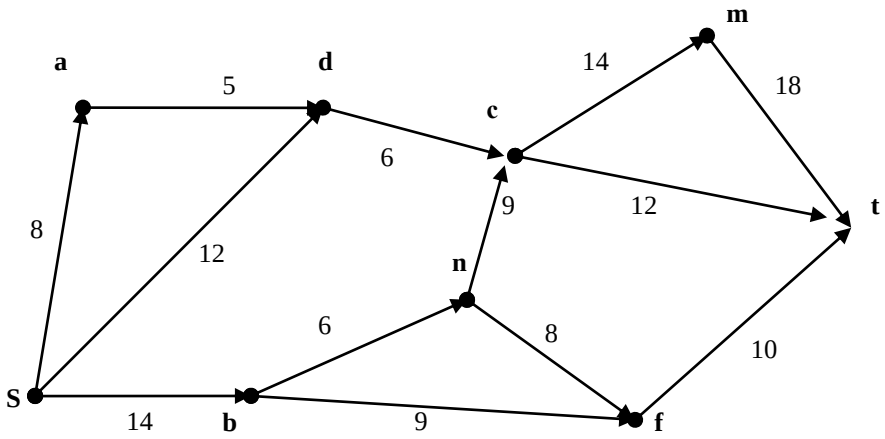


Рисунок 13.28 – Мережева модель варіанта 5

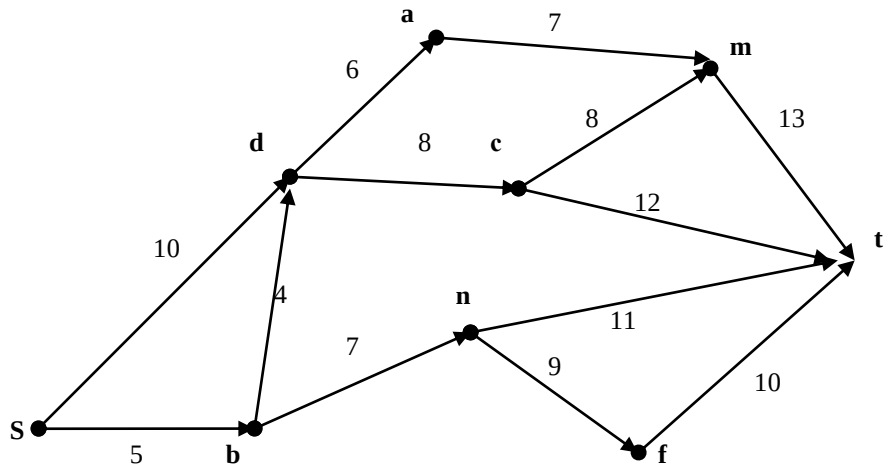


Рисунок 13.29 – Мережева модель варіанта 6

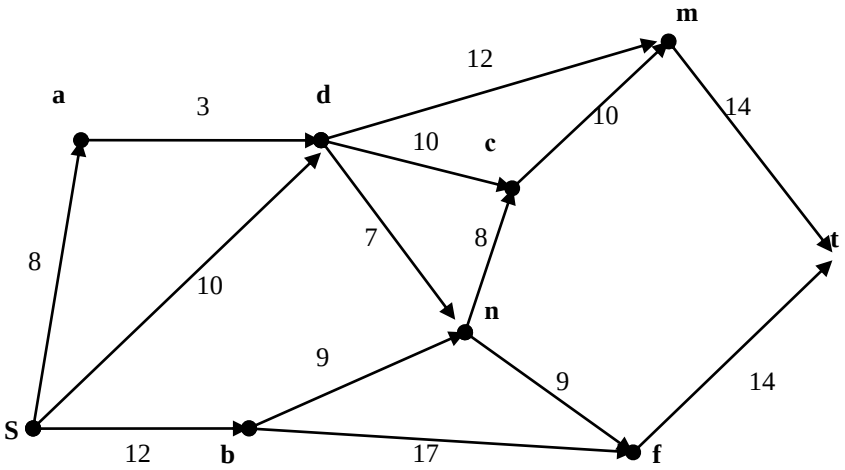


Рисунок 13.30 – Мережева модель варіанта 7

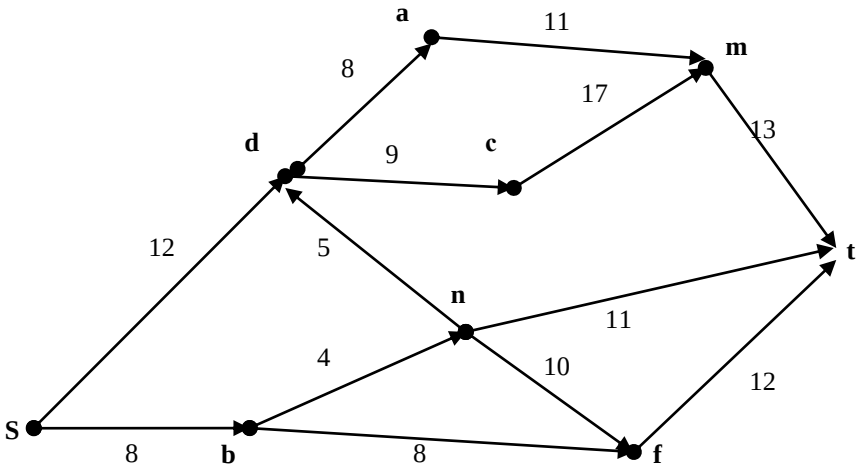


Рисунок 13.31 – Мережева модель варіанта 8

13.8 Лабораторна робота № 8. Оптимізація часу виконання проєкту при обмеженні на відновлювальні ресурси

Мета і задачі: навчитися визначати план оптимального завантаження ресурсів, що не складуються в мережевий моделі проєкту, заданого навантаженим орієнтованим графом без контурів типу II «вершини-роботи; дуги – послідовність виконання робіт».

Завдання

1. Вивчити основні поняття для дослідження мережевої моделі проєкту типу II «вершини-роботи; дуги – послідовність виконання робіт ».

2. Вивчити алгоритм отримання плану оптимального завантаження відновлювальних ресурсів при виконанні проєкту.

3. Для індивідуального варіанта отримати оптимальний план завантаження ресурсів для випадку, коли вид ресурсу для виконання будь-якої роботи один – $A_i = (a_{1i})$, а загальний обсяг цього ресурсу $Z = (z_1)$) і визначити оптимальне (мінімально можливий) час виконання проєкту.

4. Розробити рекомендації щодо оптимізації мережевої моделі проєкту.

Методичні вказівки

1. Вивчити теоретичний матеріал до лабораторної роботи.

2. При необхідності, перетворити мережеву модель проєкту до стандартного вигляду (по одній вершині тільки з вихідними (початок проєкту) і тільки вхідними (закінчення проєкту) дугами).

3. При побудові чергового допустимого плану завантаження ресурсів застосовувати метод гілок і меж.

4. Вибір варіанта: студент вибирає № варіанта завдання (рис. 13.32−13.36, таблиця 13.4), визначив значення t, де $t = N/10$ – залишок від ділення без остачі числа N (порядковий номер у основному списку групи).

Таблиця 13.4 - Варіанти завдання

№ варіанта	№ рисунку	z_1	Вершини для яких $A_i = (2)$	% запасу ресурсів
0	6.29	2	c,d	2
1	6.30	3	b,f,d	2
2	6.31	2	b,d	3
3	6.32	3	c,b,d	3
4	6.33	2	a,d	2
5	6.29	3	c,b,a	3
6	6.30	2	a,d	2
7	6.31	3	c,f,d	3
8	6.32	2	b,a	2
9	6.33	3	c,b,d	3

Примітка:

1. Можлива зміна структури орграфа, якщо кількість шляхів перевищує 8.

2. Можлива зміна тривалості робіт для забезпечення % запасу ресурсів.

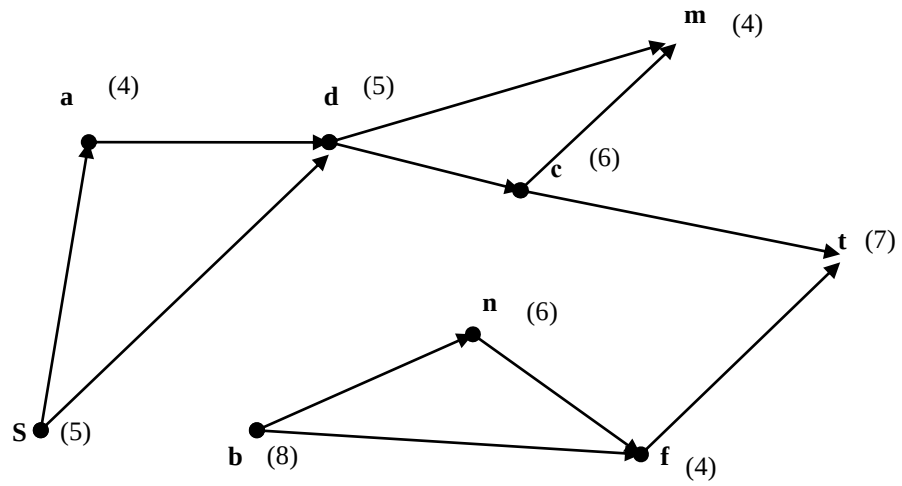


Рисунок 13.32 – Мережева модель проєктуII варіантів 0, 5

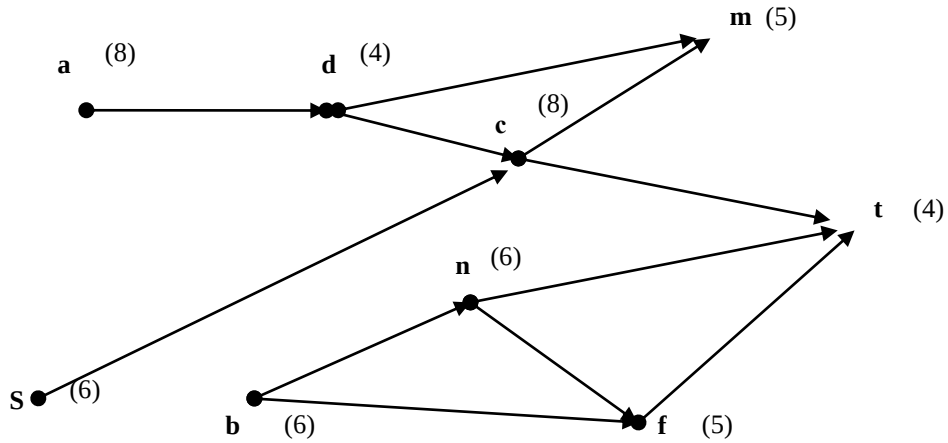


Рисунок 13.33 – Мережева модель проєкту II варіантів 1, 6

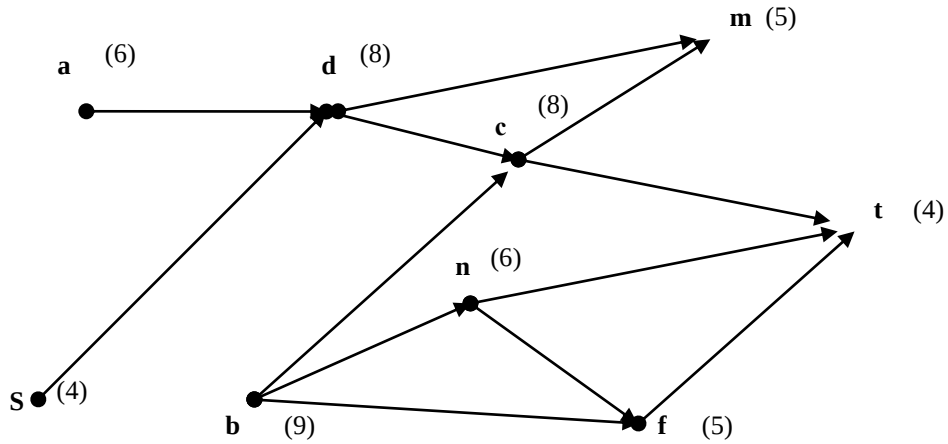


Рисунок 13.34 – Мережева модель проєкту II варіантів 2, 7

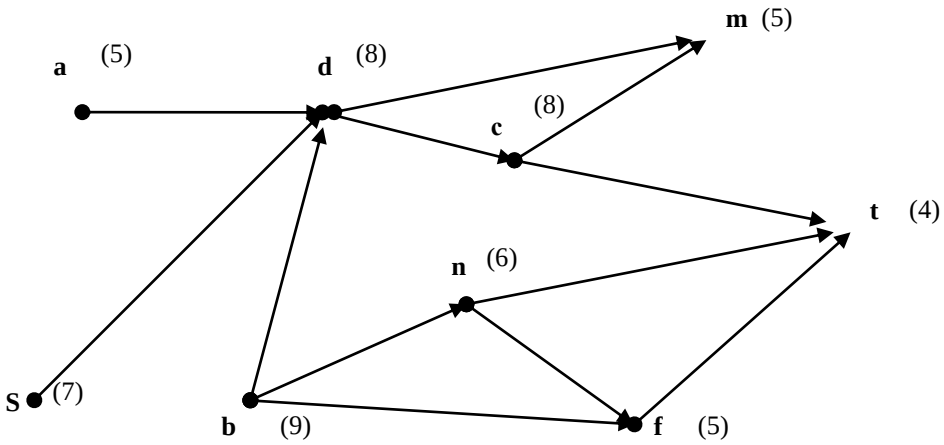


Рисунок 13.35 – Мережева модель проєкту

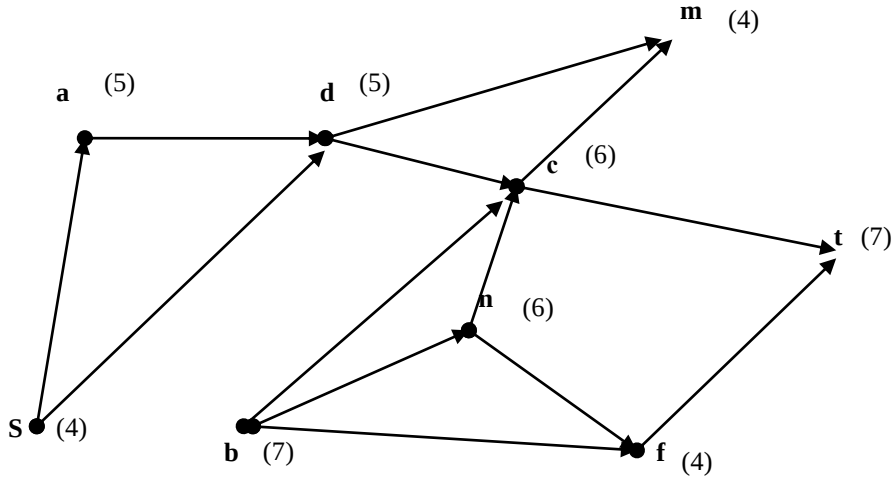


Рисунок 13.36 – Мережева модель проєкту II варіантів 3,8

Контрольні питання

1. Мережева модель проєкту типу II. Відновлювальні ресурси в мережевих моделях проєктів. Умови реалізації проєкту.

2. Постановка оптимізаційної задачі на мережевих моделях проєктів типу II.

3. Обчислення критичного шляху в мережевих моделях проєктів типу II.

4. Визначення нижньої оцінки часу виконання проєкту типу II.

5. Визначення верхньої оцінки часу виконання проєкту типу II (побудова діаграми Ганта).

6. Визначення нижньої оцінки часу виконання проєкту в процесі побудови чергового варіанта діаграми Ганта для фіксованого моменту часу.

7. Алгоритм перетворення мережевих моделей проєктів типу II в тип I.

ЛІТЕРАТУРА

1. Акимов О. Е. Дискретная математика: логика, группы, графы / Акимов О. Е. – 2-е изд., дополн. – М. : Лаборатория Базовых Знаний, 2001. – 376 с. : ил.

2. Бородкіна І. Л. Теорія алгоритмів : посібник для студентів вищих навчальних закладів / І. Л. Бородкіна, Г. О. Бородкін ; М-во освіти і науки України, Національний університет біоресурсів та природокористування України. – Київ, 2018. – 213 с.

3. Вагнер Г. Основы исследования операций: Т2, пер.с англ. – М. : Изд. «Мир», 1972.– 484 с. : ил.

4. Дискретна математика : підручник / Бардачов Ю. М. та ін. – К. : Вища шк., 2002. – 287 с. : іл.

5. Дискретная математика : учеб. для вузов / Белоусов А. И. и др. – М. : Изд-во МГТУ им. Н. Э. Баумана,2001. – 744 с.

6. Ершов С. С. Элементы теории алгоритмов : учебное пособие / С. С. Ершов. – Челябинск : Издательский центр ЮУрГУ, 2009. – 64 с.

7. Конюховский П. В. Математические методы исследования операций в экономике / Конюховский П. В. – СПб : Питер, 2000. – 208 с. : ил.

8. Мао В. Современная криптография: теория и практика : пер. с англ. / Мао В. – М. : Изд. дом «Вильямс», 2005. –768 с. : ил.

9. Новиков Ф. А. Дискретная математика для программистов / Новиков Ф. А. – СПб : Питер, 2000. – 304 с. : ил.

10. Ульянов М. В. Математическая логика и теория алгоритмов. Часть 2 : Теория алгоритмов / Ульянов М. В., Шептунов М. В. – М. : МГАПИ, 2003. – 80 с.

11. Яворський Б. І. Теорія алгоритмів : конспект лекцій / Яворський Б. І. – Тернопіль : ТДТУ імені Івана Пулюя, 2000. – 36 с.

Навчальне видання

МАЛИГІНА Світлана Валеріївна,

ГЕТЬМАН Ірина Анатоліївна,

БЕРЕЖНА Олена Валеріївна,

ДЕРЖЕВЕЦЬКА Марина Анатоліївна

ТЕОРІЯ АЛГОРИТМІВ І ГРАФІВ

Навчальний посібник

для здобувачів вищої освіти
спеціальності 122 «Комп'ютерні науки»

Редагування, комп'ютерне верстання Я. О. Бершацька

/2022. Формат 60 × 84/16. Ум. друк. арк. 8,43.
Обл.-вид. арк. 6,58. Тираж 50 пр. Зам. № 7

Видавець і виготівник
Донбаська державна машинобудівна академія
84313, м. Краматорськ, вул. Академічна, 72.
Свідоцтво суб'єкта видавничої справи
ДК № 1633 від 24.12.03