ДРОГОБИЦЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА
Кафедра фізики та інформаційних систем

Ольга ГАРБИЧ-МОШОРА

Методичні вказівки для виконання лабораторних робіт з курсу “ТЕОРІЯ АЛГОРИТМІВ”

122 "Комп'ютерні науки"

Дрогобич – 2024

УДК 510.5(072)
Г20

Рекомендовано до друку вченою радою Дрогобицького державного педагогічного університету імені Івана Франка
(протокол № 11 від 31. 10. 2024 р.)

Рецензенти:

Василь ЛИТВИН, доктор технічних наук, професор, завідувач кафедри іформаційних систем та мереж Національного університету «Львівська політехніка»;

Оксана СІКОРА , кандидат технічних наук, доцент кафедри фізики та інформаційних систем Дрогобицького державного педагогічного університету імені Івана Франка.

Відповідальний за випуск –

Роман ЛЕШКО, кандидат фізико-математичних наук, доцент кафедри фізики та інформаційних систем Дрогобицького державного педагогічного університету імені Івана Франка.

Гарбич-Мошора Ольга

Навчально-методичний посібник з курсу «Теорія алгоритмів» для студентів спеціальності 122 «Комп'ютерні науки» Дрогобич : Дрогобицький державний педагогічний університет імені Івана Франка, 2024. 76 с.

У навчально-методичному посібнику запропоновані методичні вказівки до виконання лабораторних робіт з дисципліни "Теорія алгоритмів", спрямовані на формування навичок та вмінь у галузі розробки й аналізу алгоритмів. Мета посібника – ознайомити студентів із сучасними методами створення, аналізу й оптимізації алгоритмів, а також надати їм систематизовану інформацію і практичні рекомендації для успішного виконання лабораторних робіт з цієї дисципліни.

Навчально-методичний посібник розроблено відповідно до програми навчальної дисципліни “Теорія алгоритмів” для підготовки фахівців напряму підготовки 12 Інформаційні технології, спеціальності 122 Комп’ютерні науки, затвердженої вченою радою Дрогобицького державного педагогічного університету імені Івана Франка.

ЗМІСТ

ВСТУП

У сучасному світі, де інформаційні технології проникають у всі сфери життя, розуміння алгоритмів стає необхідним для успішної роботи у галузі комп'ютерних наук. Алгоритми є основою програмних рішень, а їхня ефективність визначає продуктивність програм, які використовуються щодня. Цей навчально-методичний посібник призначений для студентів спеціальності 122 "Комп'ютерні науки" і спрямований на закріплення знань, отриманих на лекціях з курсу "Теорія алгоритмів". Він охоплює базові та складні алгоритми, необхідні для ефективного вирішення задач у галузі програмування.

Метою курсу є ознайомлення студентів з основними поняттями та техніками створення, аналізу і оптимізації алгоритмів. У ході навчання студенти навчаться ідентифікувати різні типи алгоритмів, оцінювати їхню складність та вибирати ефективні методи для конкретних задач.

Ключові теми курсу включають:

✓ Алгоритми сортування – базові та ефективні методи упорядкування даних.

✓ Жадібні алгоритми – алгоритми, що вибирають локально оптимальне рішення на кожному етапі.

✓ Рекурсивні методи – підходи, де задача розбивається на підзадачі, які розв’язуються рекурсивно.

✓ Алгоритми динамічного програмування – складні алгоритми для розв’язання задач оптимізації.

✓ Алгоритми на графах – застосування графів у розв'язанні задач маршрутизації та аналізу мереж.

✓ Алгоритми шифрування – шифри Цезаря, Віженера та інші методи криптографічного захисту, які використовуються для безпечного передавання даних. Вивчення алгоритмів шифрування дає змогу студентам зрозуміти основи захисту інформації та їх застосування у реальних умовах.

Курс допоможе студентам розвивати критичне мислення, навички аналізу алгоритмів та їхнього практичного застосування. Особливий акцент зроблено на оцінці алгоритмів – вміння аналізувати їхню ефективність та порівнювати різні підходи. Це важливо як для навчання, так і для майбутньої професійної діяльності. У навчально-методичному посібнику передбачено роботу над проєктами, що імітують реальні задачі. Використання таких інструментів, як JavaScript, C#, PHP, Python та Java, допоможе студентам отримати практичний досвід у розробці програмних рішень, включаючи веб-додатки та мобільні платформи..

Курс "Теорія алгоритмів" є фундаментом для поглибленого вивчення таких складних тем, як штучний інтелект, машинне навчання та обробка даних. Цей навчально-методичний посібник допоможе студентам оволодіти базовими знаннями та зацікавити їх у подальшому вивченні комп'ютерних наук.

Розв’язування лабораторних завдань розвиває вміння критично мислити, навички розв’язування реальних задач за допомогою алгоритмів та готує студентів до застосування здобутих знань у професійній сфері. У сучасному світі, де інформаційні технології проникають у всі сфери життя, розуміння алгоритмів є необхідним для успішної роботи у галузі комп'ютерних наук.

Лабораторна робота № 1

АЛГОРИТМ ЕВКЛІДА

Мета роботи:

- Ознайомитися з алгоритмом Евкліда для знаходження найбільшого спільного дільника (НСД) двох чисел.
- Розвинути навички написання програмного коду та аналізу алгоритмів.

Основні теоретичні відомості

Алгоритм Евкліда – один з найстаріших і найвідоміших алгоритмів в історії математики. Він був розроблений у Стародавній Греції для знаходження найбільшого спільного дільника (НСД) двох чисел. Оригінальний алгоритм базувався на ідеї "взаємного вирахування", тобто постійного віднімання меншого числа від більшого до моменту, коли одне з них стане рівним нулю. Ця техніка застосовувалась як до натуральних чисел, так і до геометричних величин (довжин відрізків).

Історія та еволюція алгоритму

З плином часу, починаючи з XIX століття, алгоритм було розширено для роботи з іншими типами чисел, включаючи цілі та раціональні числа. На зміну початковій техніці взаємного віднімання прийшло використання операції знаходження залишку від ділення, що зробило алгоритм більш ефективним. Незважаючи на це нововведення, суть алгоритму залишилася незмінною.

Сьогодні алгоритм Евкліда використовується не тільки для знаходження НСД, але й у багатьох галузях математики та комп'ютерної науки. Він є основою для багатьох інших алгоритмів, включаючи алгоритми для розв'язування діофантових рівнянь, криптографії та обчислювальної геометрії.

Цей алгоритм демонструє, як древні математичні ідеї можуть еволюціонувати й залишатися корисними тисячоліттями після свого відкриття.

Принцип роботи алгоритму Евкліда

Принцип роботи алгоритму Евкліда: Алгоритм Евкліда ґрунтується на такому принципі: НСД двох чисел *a* і *b* можна знайти, якщо замінити *a* на *b* і *b* на залишок від ділення *a* на *b* (тобто *a* mod *b*), доки *b* не стане рівним нулю. На цьому етапі *a* і буде найбільшим спільним дільником.

Формально:

$$\text{НСД}(a,b) = \begin{cases} a & \text{якщо } b = 0 \\ \text{НСД}(b, a \mod b) & \text{інше} \end{cases}$$

Остання відмінна від нуля остача *rk*, на яку націло ділиться остача *rk-1*, буде найбільшим спільним дільником чисел *a* і *b*.

Часова складність: Часова складність алгоритму Евкліда оцінюється як *O (log (min (a, b)))*.

Практична частина

Ітеративна реалізація

```
using System;

class Program
{
    static int GCDIterative(int a, int b)
    {
        while (b != 0)
        {
            int temp = b;
            b = a % b;
            a = temp;
        }
        return a;
    }

    static void Main(string[] args)
    {
        Console.WriteLine("Введіть два числа для знаходження НСД:");
        int a = int.Parse(Console.ReadLine());
        int b = int.Parse(Console.ReadLine());

        Console.WriteLine($"Ітеративний НСД: {GCDIterative(a, b)}");
    }
}
```

Рекурсивна реалізація

```
using System;

class Program
{
    static int GCDRecursive(int a, int b)
    {
        if (b == 0)
            return a;
        return GCDRecursive(b, a % b);
    }

    static void Main(string[] args)
    {
        Console.WriteLine("Введіть два числа для знаходження НСД:");
        int a = int.Parse(Console.ReadLine());
        int b = int.Parse(Console.ReadLine());

        Console.WriteLine($"Рекурсивний НСД: {GCDRecursive(a, b)}");
    }
}
```

Лінійне діофантове рівняння

Означення. Рівняння виду $ax + by = c$ називається *лінійне діофантове рівняння* з двома невідомими, якщо a, b, c – цілі числа, $a \neq 0, b \neq 0, c \neq 0$.

Ці рівняння названі на честь грецького математика Діофанта, який жив у III столітті н. е. У своїй книзі «Арифметика» він розв'язав 189 задач з цілими числами, для яких навів один або декілька розв'язків.

Етапи розв'язання діофантових рівнянь

1. **Визначення наявності розв'язків**: Перше питання, яке треба вирішити – чи має рівняння $ax+by=c$ хоча б один розв'язок у цілих числах. Для цього необхідно перевірити, чи ділиться число c на найбільший спільний дільник чисел a і b. Якщо $НСД(a,b)$ ділить c, то рівняння має принаймні один розв'язок.

2. **Скінченність чи нескінченність множини розв'язків**: Якщо рівняння має принаймні один розв'язок, можна визначити, чи є множина розв'язків скінченною чи нескінченною. Лінійне діофантове рівняння має нескінченну кількість розв'язків, якщо існує хоча б один базовий розв'язок. Всі інші

розв'язки утворюються шляхом додавання кратних певного параметра до базового розв'язку.

3. **Знаходження всіх цілих розв'язків**: Щоб знайти всі розв'язки рівняння, можна скористатися алгоритмом Евкліда для знаходження найбільшого спільного дільника (НСД) чисел *a* і *b*. Якщо НСД ділить *c*, знаходиться один базовий розв'язок, а інші розв'язки записуються у вигляді:

$$x = x_0 + t \cdot \frac{b}{d}, \quad y = y_0 - t \cdot \frac{a}{d}$$

де *d=НСД(a,b), x_0 і y_0* – базовий розв'язок, а *t* – параметр, що приймає будь-які цілі значення.

Приклад: Розв’язати рівняння 3x + 5y = 7 в цілих числах.

Розв’язання:

1. Перевіримо умову розв’язності: коефіцієнти рівняння *a =3, b =5, c =7, НСД(3, 5) = 1*, отже маємо ціле число, якщо *7: НСД(3, 5),* тому дане рівняння має множину розв’язків в цілих числах.

2. Знайдемо спочатку який-небудь конкретний розв’язок: Тут використаємо таку ідею, до речі, часто допомагає і при розв’язанні інших завдань.

Спочатку знайдемо одну пару цілих чисел *(m; n),* яка є рівняння розв’язком іншого, легшого рівняння: *3x + 5y = 1*, тоді матимемо правильну рівність: *3m + 5n = 1,* а для того, щоб знайти один розв’язок *(xo, yo)* для рівняння *3x + 5y = 7*, треба буде помножити правильну рівність *3m + 5n = 1* на *7*. Продемонструємо цю ідею на практиці. Оскільки легко встановити, що *3m + 5n = 3·2 + 5·(-1) = 1*, то *3x + 5y = 3·(2·7) + 5·(-7·1) = 1·7* і, отже, $x_o = 14, y_o = 7$ – це розв’язок даного рівняння.

3. Отже, маємо дві рівності:

3x + 5y = 7,

$3\,x_o + 5\,y_o = 7$.

Віднімемо одне рівняння з іншого, позначимо $x\text{-}x_o$ і $y\text{-}y_o$ через *pig*, і отримаємо *3a + 5b = 0*. Звідси ми бачимо, що *b* ділиться на *3*, а *a* – на *5*.

Покладемо $p = 5k$, тоді $g = 3k$ – тут очевидно, що k – може бути будь-яким цілим числом. Отже, ми отримуємо набір розв'язків:

$x - x_o = 5k$; $x = 14 + 5k$, де k – ціле число.

y - yo = -3k; y = - 7 -3, де k – ціле число.

Інших розв'язків, звичайно, немає.

Відповідь: $(14 + 5k; -7 - 3k)$, де k – довільне ціле число.

ЗАВДАННЯ

1. Знайдіть найменше спільне кратне (НСК) за формулою $НСК(a,b) = \frac{a * b}{НСД(a,b)}$, де a = 9453; b = 4384.

2. Дано два цілі числа (66;88). З'ясуйте, чи вони взаємно прості.

3. Дано натуральні числа ***a*** і ***b***, що позначають чисельник та знаменник простого дробу. Скоротіть дріб, тобто знайдіть такі натуральні числа ***p*** та ***q***, що не мають спільних дільників, що $\frac{a}{b} = \frac{p}{q}$.

Пояснення: за алгоритмом Евкліда знаходимо НСД(a, b) та ділимо чисельник та знаменник на це число.

4. Знайдіть суму двох простих дробів. Тобто дано натуральні числа ***a, b, c, d***. Потрібно знайти два взаємно простих числа ***p*** і ***q***, таких що $\frac{a}{b} + \frac{c}{d} = \frac{p}{q}$.

Пояснення: скласти два дроби. Чисельник дорівнює ***a*d+b*c***. Знаменник дорівнює ***b*d***. Потім за алгоритмом Евкліда знаходимо НСД чисельника та знаменника і ділимо чисельник та знаменник на це число.

5. Ви плануєте організувати вечірку, на якій буде 3 види закусок (A, B, C). У вас є 24 порції A, 36 порцій B та 30 порцій C. Яку максимальну кількість

гостей ви можете запросити, якщо кожен отримає по 1 порції кожного виду закусок?

6. У вас є 72 книги з різних жанрів, які ви хочете розділити між 8 друзями так, щоб кожен отримав однакову кількість книг. Скільки книг буде в кожного друга?

7. Ви маєте 54 порції піци і 36 порцій салату. Як ви можете розподілити їх між групами, щоб кожна група отримала однакову кількість порцій піци і салату?

8. У вас є 48 аркушів кольорового паперу та 36 фломастерів. Скільки наборів для малювання ви можете створити, щоб у кожному наборі було однакову кількість аркушів паперу та фломастерів?

9. У вас є 30 медалей та 18 дипломів для учасників змагань. Яка максимальна кількість учасників може отримати як медаль, так і диплом?

10. Дано числа ***M, N, R***. Знайдіть в інтервалі **[M, N]** усі числа взаємно прості з ***R***.

11. Знайти всі правильні прості дроби, що не скорочуються, знаменники яких не більше 7 (дріб задається двома натуральними числами – чисельником та знаменником). **Відповідь**: 1/2, 1/3, 2/3, 1/4, 3/4, 1/5, 2/5, 3/5, 4/5, 1/6, 5/6, 1/7, 2/7, 3/7, 4/7, 5/7, 6/7.

12. Три пароплави заходять в Очаків після кожного рейсу і зразу ж виходять у наступний рейс. Перший пароплав здійснює свій рейс за 120 год., другий за 150 год., а третій за 90 год. Пароплави вийшли в море одночасно. Через скільки діб всі вони найраніше зустрінуться в Очакові?

13. Скоротити дроби: $а)\frac{299}{1449};\quad б)\frac{4914}{21385};\quad в)\frac{6494}{7021}.$

14. Знайдіть усі пари взаємно простих чисел в інтервалі **[M, N]**.

15. Яку найбільшу кількість однакових подарунків можна зробити з 36 помаранчів, 42 груш та 54 яблук так, щоб використати всі фрукти і кожен подарунок містив однакову кількість фруктів.

16. Школа № 1 м. Дрогобич отримала гуманітарну допомогу. Ця допомога складалася з 1686 кокосів, 2810 штук плодів манго та 3372 штук

плодів ківі. Допомога була розділена порівну між учнями. Яка максимальна кількість учнів може навчатися у цій школі?

17. Яке найбільше число однакових подарунків можна зробити з 320 горіхів, 240 цукерок та 240 пряників при умові, що всі подарунки будуть однакові? Скільки горіхів, цукерок та пряників буде у кожному подарункові?

18. Теплохід "Симоненко" виконує свій рейс туди та назад за 8 днів, теплохід "Шевченко" – за 12 днів, а теплохід "Очаків" – за 18 днів. У суботу 30 серпня всі теплоходи пішли в рейс. Через скільки днів теплоходи знову зустрінуться в порту? В який день тижня і якого числа це станеться?

19. Відомо, що парне число N ділиться без остачі на 8, 10 та 12. На яке з чисел 60, 80 або 120 воно також ділиться?

20. Знайти найменший спільний знаменник дробів:

а) $\frac{5}{28}, \frac{4}{35}, \frac{7}{60}$; *б)* $\frac{1}{6}, \frac{4}{15}, \frac{5}{36}, \frac{5}{1440}$.

ЗАВДАННЯ

Знайти загальний розв’язок лінійного діофантового рівняння:

1. 13x + 17y = 1
2. 144x + 233y = 1;
3. 1263x +204y = 6;
4. 504x + 726y = 36;
5. $13x + 21y = 55$
6. $45x - 37y = 25$
7. $2183x - 1961y = 6327$
8. 3x + 5y = 7
9. 35x – 2004y = 11.
10. 3x+4y=60
11. 21x + 48y = 6
12. 2x+3y=30
13. 5x+2y=50
14. 7x+5y=70
15. 4x+6y=72
16. 8x+3y=45
17. 2x+5y=28
18. 10x+15y=300
19. 6x+2y=42
20. 3x+9y=81
21. 2x+8y=64
22. 7x+2y=35

Контрольні запитання

1. Що таке алгоритм Евкліда? (Визначте його основну мету.)
2. Яка проблема розв'язується за допомогою алгоритму Евкліда? (Яку математичну задачу він допомагає розв’язати?)
3. Як виглядає формула для обчислення найбільшого спільного дільника (НСД)? (Визначте, як використовуються числа в цій формулі.)
4. Які основні етапи алгоритму Евкліда? (Опишіть покроково, як виконується алгоритм.)
5. Які є варіанти реалізації алгоритму Евкліда? (Які відмінності між рекурсивною та ітеративною версією?)
6. Яка складність алгоритму Евкліда у найгіршому випадку? (Чому така складність є оптимальною для цього алгоритму?)
7. Як можна продемонструвати роботу алгоритму Евкліда на прикладі? (Наведіть конкретні числа та покажіть, як обчислюється НСД.)
8. Які альтернативні алгоритми для знаходження НСД ви знаєте? (Порівняйте їх з алгоритмом Евкліда.)
9. Які властивості НСД використовуються в алгоритмі Евкліда? (Наведіть приклади, як ці властивості допомагають у виконанні алгоритму.)
10. Які практичні застосування має алгоритм Евкліда? (В яких сферах він використовується?
11. Які помилки можуть виникнути при реалізації алгоритму Евкліда? (Як їх уникнути?)
12. Яка роль рекурсії в алгоритмі Евкліда? (Які переваги та недоліки рекурсивного підходу?)
13. Як можна оптимізувати алгоритм Евкліда? (Які методи існують для підвищення його ефективності?)

14. Що таке алгоритм Евкліда для декількох чисел? (Як він адаптується для обчислення НСД кількох чисел одночасно?)

15. Які зв’язки існують між алгоритмом Евкліда та теорією чисел? (Як його результати впливають на вивчення числа в математиці?)

Лабораторна робота № 2

ІТЕРАЦІЙНІ ТА РЕКУРСИВНІ АЛГОРИТМИ

Мета роботи:

- Ознайомитися з принципами роботи ітераційних та рекурсивних алгоритмів. Вивчити основні відмінності між ітерацією та рекурсією.
- Дослідити ефективність і оптимізацію ітераційних та рекурсивних алгоритмів. Проаналізувати складність ітераційних і рекурсивних алгоритмів.

Основні теоретичні відомості

Ітерація (від лат. *iteratio* – повторення) – це багаторазове виконання певної дії або операції в обчислювальному процесі. Ітераційні алгоритми базуються на циклах, у яких виконуються повторні обчислення з оновленими даними. ***Метою ітерації*** є поступове наближення до остаточного результату. При цьому не використовуються рекурсивні виклики (виклики функції самою собою).

Ітерація в алгоритмах часто базується на використанні циклів. Ось приклад псевдокоду для ітераційного алгоритму обчислення факторіала числа ***n***:

```
Функція ІтераційнийФакторіал(n)
    Результат = 1
    Для і від 1 до n
        Результат = Результат * і
    Кінець циклу
    Повернути Результат
Кінець функції
```

Опис:

- Створюється змінна Результат, яка ініціалізується значенням 1.
- Далі у циклі від 1 до n (включно) послідовно перемножаються всі числа.
- Після завершення циклу результат (факторіал) повертається як результат роботи функції.

Цей алгоритм показує, як ітераційний підхід використовує цикл для повторення дій замість рекурсивних викликів.

Рекурсія (від лат. *recursio* – повернення) – це підхід, при якому функція або метод викликає сам себе для розв'язання підзадачі. Ключовим елементом рекурсивного алгоритму є наявність **базового випадку**, який визначає, коли алгоритм припиняє викликати сам себе.

Рекурсивні алгоритми вирішують задачу шляхом розбиття її на менші підзадачі, причому функція викликає саму себе. Ось приклад псевдокоду для рекурсивного обчислення факторіала числа n:

```
Функція РекурсивнийФакторіал(n)
    Якщо n = 0 або n = 1
        Повернути 1
    Інакше
        Повернути n * РекурсивнийФакторіал(n - 1)
Кінець функції
```

Опис:

- Якщо n = 0 або n = 1, алгоритм повертає 1, оскільки факторіал цих чисел дорівнює 1.
- В іншому випадку функція викликає сама себе з аргументом n - 1, що реалізує рекурсивний підхід. Після досягнення базового випадку, значення починають обчислюватися знизу вгору.

Цей приклад демонструє, як рекурсія розв'язує задачу через поділ на підзадачі, що є меншими варіантами тієї ж задачі.

Порівняння ітерації та рекурсії

Простота коду: Рекурсивні алгоритми часто виглядають більш компактно та легко читаються, оскільки описують задачу природно через підзадачі. Ітераційні алгоритми, як правило, використовують явні цикли і можуть бути більш громіздкими.

Використання пам'яті: Ітераційні алгоритми зазвичай використовують менше пам'яті, оскільки не вимагають збереження проміжних станів функцій у стосі викликів, як це відбувається в рекурсії.

Продуктивність: У деяких випадках рекурсивні алгоритми можуть бути менш ефективними через накладні витрати на рекурсивні виклики, особливо якщо відсутні оптимізації, такі як хвостова рекурсія.

Базові випадки: Важливим елементом рекурсії є базовий випадок – умова, яка припиняє подальші виклики. Відсутність базового випадку може призвести до безкінечної рекурсії.

ЗАВДАННЯ

1. Надрукувати всі спадні послідовності довжини k, елементами яких є натуральні числа від 1 до n: Скласти програму, яка виводить усі можливі спадні послідовності чисел довжини k, де кожен елемент належить до множини натуральних чисел від 1 до n. Використати рекурсію або ітерацію для генерації цих послідовностей.

Ускладнення: Заборонити повторення однакових елементів у послідовності та додати можливість виводу лише тих послідовностей, що містять парні числа.

2. Вивести всі можливі перестановки чисел від 1 до n. Напишіть програму, яка виводить усі можливі перестановки чисел від 1 до n. Можна використовувати рекурсію для генерування перестановок або циклічні перестановки з використанням ітерації.

Ускладнення: Додати умову, що кожна перестановка не повинна містити сусідні однакові числа, якщо n не є унікальним.

3. Знайти всі числа від 1 до n, які є паліндромами. Напишіть програму, яка перевіряє, чи є число паліндромом, і виводить усі числа від 1 до n, що є паліндромами. Можна використовувати ітерацію або рекурсію для перевірки кожного числа.

P.s. (Паліндром – це слово, фраза, число, яка читається однаково в обох напрямках, тобто зліва направо і справа наліво. Наприклад: 121, 12321)

Ускладнення: Додати перевірку на простоту для паліндромів і вивести лише ті, що є простими.

4. Знайти всі числа від 1 до n, які є досконалими. Напишіть програму, яка перевіряє, чи є число досконалим (сума його дільників дорівнює самому числу), і виводить усі такі числа від 1 до n.

Ускладнення: Вивести також всі досконалі числа, які мають непарну кількість дільників.

5. Знайти всі підмножини множини чисел від 1 до n. Напишіть рекурсивну програму для знаходження всіх підмножин множини чисел від 1 до n.

Ускладнення: Вивести лише ті підмножини, сума елементів яких є парним числом.

6. Надрукувати всі можливі комбінації чисел від 1 до n, які в сумі дають k. Напишіть програму для знаходження всіх можливих комбінацій чисел від 1 до n, які в сумі дають k. Реалізуйте її рекурсивно або ітераційно.

Ускладнення: Заборонити використання однакових чисел у комбінації та вивести лише ті комбінації, в яких сума елементів парна.

7. Знайти всі пари чисел (a, b), такі що їх добуток дорівнює n. Напишіть програму, яка знаходить усі пари чисел (a, b), для яких добуток a * b = n.

Ускладнення: Вивести тільки ті пари, де a та b є простими числами.

8. Дана послідовність натуральних чисел (одне число в рядку), що завершується числом 0. Виведіть всі непарні числа з цієї послідовності, зберігаючи їх порядок. (*Вказівка. Не можна використовувати глобальні змінні і передавати будь-які параметри в рекурсивну функцію. Функція отримує дані, зчитуючи їх з клавіатури, не повертає значення, а відразу ж виводить результат на екран. Основна програма повинна складатися тільки з виклику цієї функції.)*

Ускладнення: Вивести не лише непарні числа, а й їх кількість, а також обчислити та вивести суму цих непарних чисел.

9. Скласти програму знаходження n-го члена геометричної прогресії.

Ускладнення: Додати можливість обчислення n-го члена геометричної прогресії для різних значень a_1 і r, запитуючи їх у користувача, а також реалізувати перевірку на коректність введених значень (наприклад, перевірити, що r>0).

10. Скласти програму знаходження суми членів арифметичної прогресії.

Ускладнення: Додати можливість знаходження суми всіх членів арифметичної прогресії у заданому діапазоні, а також реалізувати обчислення суми з використанням рекурсії.

11. Дано натуральне число n > 1. Виведіть всі прості множники цього числа в порядку зростання з урахуванням кратності. Алгоритм повинен мати складність O (logn).

Ускладнення: Знайти не лише прості множники числа, а й кількість кратностей кожного з них, а також вивести результати у вигляді таблиці з множником і його кратністю.

12. Перевірити, чи можна представити число *n* як суму двох квадратів. Напишіть програму, яка перевіряє, чи можна представити число n як суму квадратів двох натуральних чисел.

Ускладнення: Додати можливість представлення n як суми трьох або більше квадратів.

13. Надрукувати всі можливі способи розкладу числа n на суму цілих додатних чисел. Напишіть програму, яка виводить усі можливі розклади числа n на додатні частини.

Ускладнення: Вивести тільки розклади, в яких усі доданки є простими числами.

14. Дано натуральне число N. Вивести всі цифри числа по одній у зворотному порядку, розділяючи їх пробілами чи нулями.

Ускладнення: Замість простого виведення цифр у зворотному порядку, реалізуйте функцію, яка виведе не лише цифри, а і їхню кількість, а також суму всіх цифр. Додайте можливість виведення цифр у вигляді рядка, де кожна цифра відокремлена комою.

15. Дано число n, десятковий запис якого не містить нулів. Отримайте число, записане тими ж цифрами, але в протилежному порядку. *(Вказівка. Функція повинна повертати ціле число, яке є результатом роботи програми, виводити число по одній цифрі неможна).*

Ускладнення: Додати можливість перевіряти, чи є отримане число парним чи непарним, і виводити відповідне повідомлення. Також реалізуйте перевірку, чи входить число до заданого діапазону (наприклад, від 1 до 10000) та виводьте повідомлення, якщо число виходить за межі.

16. Дано натуральне число n> 1. Перевірте, чи є воно простим, на екран вивести слово ТАК, якщо число просте і НІ, якщо число складене. Алгоритм повинен мати складність O (logn).

(Вказівка. Зрозуміло, що завдання само собою нерекурсивне, тому що перевірка числа n на простоту ніяк не зводиться до перевірки на простоту менших чисел. Тому потрібно зробити ще один параметр рекурсії: дільник числа, і саме за цим параметром і робити рекурсію.)

Ускладнення: Реалізувати функцію, яка, якщо число не є простим, виведе всі його прості множники. Також додати можливість перевірки кількох чисел одночасно, виводячи результати для кожного з них.

17. Дана монотонна послідовність, в якій кожне натуральне число k зустрічається рівно k раз: 1, 2, 2, 3, 3, 3, 4, 4, 4, 4, ... За даним натуральному n виведіть перші n членів цієї послідовності. Спробуйте обійтися тільки одним циклом *for*.

Ускладнення: Додати можливість виведення не лише перших n членів послідовності, а й підрахунку їх суми. Зробити так, щоб програма могла виводити члени послідовності до заданого значення, наприклад, до максимальної цифри, що входить у послідовність.

18. Є деяка сума грошей S і набір монет з номіналами a1, ..., an. Монета кожного номіналу є в єдиному екземплярі. Необхідно знайти всі можливі способи розміняти суму S за допомогою цих монет.

Ускладнення: Реалізувати програму, яка не лише знайде всі способи розмінювання, але й підрахує їх кількість. Додати можливість вводити кілька наборів монет і обчислювати для кожного з них способи розмінювання заданої суми, а також виводити найменшу та найбільшу комбінацію за значенням.

19. Надрукувати всі зростаючі послідовності довжини k, елементами яких є натуральні числа від *1* до *n*. (Передбачається, що *k* не перевищує *n* – інакше таких послідовностей не існує.).

Ускладнення: Додати можливість генерувати не лише зростаючі, а й спадаючі послідовності довжини k з тих же елементів. Реалізуйте функцію, яка дозволяє генерувати всі можливі комбінації довжини k, а не лише ті, що є строго зростаючими. Додайте можливість виведення усіх зростаючих та спадних послідовностей до заданого значення, наприклад, обмеження по сумі елементів.

20. Складіть програму, яка вибирає з даного масиву чисел ті і тільки ті його елементи, які задовольняють обом властивостям (функцію перевірки простоти числа і функцію перевірки числа на розкладність в суму виду *1+2x*).

Ускладнення: Замість простого вибору, реалізуйте функцію, яка не лише знаходить елементи, що задовольняють обом властивостям, а й підраховує кількість таких елементів та їхню суму. Додайте можливість передавати масив з негативними і позитивними числами та реалізуйте фільтрацію, яка виключає негативні числа на етапі перевірки. Реалізуйте виведення елементів у порядку спадання або зростання.

21. Дано натуральне чотирицифрове число N. Виведіть всі його цифри по одній, в звичайному порядку, розділяючи їх пробілами або новими рядками.

Ускладнення. Додати можливість не лише виводити цифри в звичайному порядку, а й в зворотному, а також підраховувати їхню суму та добуток. Реалізуйте функцію, яка перевіряє, чи є число N паліндромом, і виводить відповідне повідомлення. Зробіть так, щоб програма могла обробляти числа довільної довжини (не лише чотиризначні) і коректно виводила результати для всіх цифр.

Контрольні запитання

1. Що таке рекурсія, і як вона відрізняється від ітерації?
2. Які основні елементи рекурсивного алгоритму?
3. Як працює рекурсивний виклик функції у програмі? Наведіть приклад.
4. Які переваги та недоліки використання рекурсії у програмуванні?
5. Що таке рекурсивне повернення, і чому воно важливе для коректної роботи рекурсивних функцій?
6. Які можуть виникнути проблеми при використанні рекурсії, і як їх можна уникнути?
7. Наведіть приклад рекурсивної функції для обчислення факторіала числа.
8. Яка роль базового випадку у рекурсивній функції? Чому його наявність є критично важливою?
9. Як перевірити, чи є заданий масив симетричним, використовуючи рекурсію?
10. Як реалізувати рекурсивну функцію для виведення всіх непарних чисел з послідовності, що завершується нулем?
11. Що таке ітерація, і як вона використовується в алгоритмах?
12. Які основні конструкції ітерації використовуються в програмуванні?
13. Які переваги та недоліки ітерації в порівнянні з рекурсією?
14. Як можна обчислити n-тий член арифметичної прогресії за допомогою ітерації?
15. Пряма і непряма рекурсія. Наведіть приклади

Лабораторна робота № 3

АЛГОРИТМИ СОРТУВАННЯ

Мета роботи:

- Ознайомитися з різними алгоритмами сортування, дослідити принципи роботи цих алгоритмів, зокрема їх ефективність та оптимізацію в різних ситуаціях.
- Розвинути навички написання програмного коду та аналізу алгоритмів.

Основні теоретичні відомості

Алгоритми сортування – це алгоритми, які впорядковують елементи масиву або списку відповідно до певного порядку (зазвичай зростання або спадання). Сортування є важливою операцією в багатьох задачах, пов'язаних із пошуком, впорядкуванням і аналізом даних.

Вибір алгоритму сортування

- *Для малих масивів* підійдуть прості алгоритми, такі як сортування вставками або сортування вибором. Ці алгоритми мають меншу складність реалізації та ефективні для невеликих обсягів даних.
- *Для великих масивів* варто використовувати швидке сортування або сортування злиттям, оскільки вони мають кращу часову складність і дозволяють швидше обробляти великі набори даних.
- *Якщо потрібно стабільне сортування* (збереження порядку рівних елементів), найкраще використовувати сортування злиттям. Це важливо в задачах, де значення елементів можуть повторюватися і порядок має значення.
- *Сортування Шелла* є зручним вибором для масивів середнього розміру, коли необхідно щось швидше за звичайне сортування вставками, але менш складне у реалізації, ніж злиття. Воно підвищує продуктивність шляхом зменшення інтервалів між елементами, які порівнюються.

•*Пірамідальне сортування* ідеально підходить для великих масивів та випадків, коли важлива стабільна швидкість сортування без втрати продуктивності. Воно забезпечує ефективність сортування з часовою складністю **O(n log n)** у найгірших випадках і добре працює навіть на частково відсортованих масивах.

Ці рекомендації допоможуть вибрати оптимальний алгоритм для конкретних задач, залежно від розміру масиву та вимог до стабільності сортування.

Деякі з найпоширеніших алгоритмів сортування:

1. Сортування вставками (Insertion Sort)

Алгоритм сортування вставками працює за принципом поступового вставляння кожного елемента на його правильну позицію в частково відсортованому масиві.

Алгоритм

•Розглядається кожен елемент масиву.

•Кожен новий елемент вставляється на відповідну позицію у відсортовану частину масиву.

Псевдокод:

```
for i = 1 to n:
    current = array[i]
    j = i - 1
    while j >= 0 and array[j] > current:
        array[j + 1] = array[j]
        j = j - 1
    array[j + 1] = current
```

Часова складність: $O(n^2)$ у найгіршому випадку, $O(n)$ у найкращому (якщо масив вже відсортований).

2. Сортування вибором (Selection Sort)

Алгоритм сортування вибором знаходить найменший елемент у масиві і міняє його місцями з першим елементом, потім шукає найменший елемент серед інших і повторює процес.

Алгоритм

- Знаходиться найменший елемент в масиві і міняється місцями з першим елементом.
- Повторюється для наступної частини масиву.

Псевдокод:

```
for i = 0 to n-1:
    min_index = i
    for j = i+1 to n:
        if array[j] < array[min_index]:
            min_index = j
    swap(array[i], array[min_index])
```

Часова складність: $O(n^2)$ у всіх випадках.

3. Сортування бульбашкою (Bubble Sort)

Сортування бульбашкою передбачає порівняння сусідніх елементів і обмін їх місцями, якщо вони стоять не в правильному порядку. Це повторюється доти, доки масив не буде відсортований.

Алгоритм

- Порівнюються сусідні елементи і, якщо вони стоять неправильно, міняються місцями.
- Процес повторюється для всіх елементів.

Псевдокод:

```
for i = 0 to n-1:
    for j = 0 to n-i-1:
        if array[j] > array[j+1]:
            swap(array[j], array[j+1])
```

Часова складність: $O(n^2)$ у найгіршому випадку, $O(n)$ у найкращому.

4. Сортування Шелла (Shell Sort)

Алгоритм Шелла – це удосконалена версія сортування вставками. Вона зменшує кількість переміщень елементів за рахунок сортування з використанням інтервалів. Алгоритм Шелла починає з великих інтервалів між елементами і поступово зменшує їх до 1.

Алгоритм

- Масив ділиться на кілька підмасивів з елементами, які розміщені на певній відстані (інтервалі).
- Кожен підмасив сортується вставками.
- Інтервал зменшується і процес повторюється, доки інтервал не дорівнює 1.

Псевдокод:

```
shellSort(array):
    n = length(array)
    gap = n // 2  // початковий інтервал

    while gap > 0:
        for i = gap to n:
            temp = array[i]
            j = i
            while j >= gap and array[j - gap] > temp:
                array[j] = array[j - gap]
                j = j - gap
            array[j] = temp
        gap = gap // 2
```

Часова складність: У середньому випадку O(n^(3/2)), у найгіршому випадку $O(n^2)$.

5. Сортування пірамідою (Heap Sort)

Алгоритм пірамідального сортування (або Heap Sort) ґрунтується на використанні структури даних, що називається **піраміда** (heap). Алгоритм будує максимальну піраміду і потім витягує з неї елементи один за одним, переміщаючи їх в кінцевий відсортований масив.

Алгоритм

1. Створюється максимальна піраміда з масиву.
2. Найбільший елемент (корінь піраміди) переноситься у кінець масиву.
3. Відновлюється піраміда для решти масиву (без останнього елемента).
4. Повторюються кроки 2–3 доти, доки всі елементи не будуть відсортовані.

Часова складність: O(n log n) у всіх випадках (найкращий, середній, найгірший).

ЗАВДАННЯ

У контрольному прикладі забезпечити сортування потрібних елементів в не відсортованих масивах 3 способами сортування, вказавши кількість проходів алгоритму, додавши таймер часу виконання сортування який буде виводитися на екран, графічне відображення масиву за допомогою бібліотеки matplotlib одним із трьох один із методів.

1. Сортування балів студентів

Напишіть програму, яка отримує масив балів студентів і сортує його за зростанням.

Ускладнення: Виведіть також індекси студентів після сортування, щоб зберегти інформацію про те, хто які бали отримав. Додатково порівняйте швидкість роботи обраного алгоритму для різних розмірів масивів.

2. Сортування продуктів за ціною

Напишіть програму для інтернет-магазину, яка отримує список продуктів і сортує їх за ціною від найменшої до найбільшої.

Ускладнення: Додайте можливість сортувати продукти також за іншими параметрами (наприклад, вагою або кількістю), щоб порівняти різницю у швидкості та складності виконання різних методів на практиці.

3. Сортування музичних треків за тривалістю

Напишіть програму, яка сортує музичні треки за тривалістю, використовуючи будь-який метод сортування. Треки повинні бути впорядковані за зростанням або спаданням тривалості, залежно від вибору користувача.

Ускладнення: Порівняйте ефективність обраного алгоритму на різних кількостях треків. Додайте тест для стабільності сортування (якщо треки однакові за тривалістю, чи зберігається їх початковий порядок?).

4. Сортування списку завдань за тривалістю

Напишіть програму, яка сортує список завдань (із зазначенням часу на виконання кожного) за тривалістю за допомогою будь-якого алгоритму сортування.

Ускладнення: Додайте функцію пошуку мінімального числа завдань, які потрібно виконати для досягнення заданої кількості часу, щоб порівняти ефективність різних методів сортування в цьому контексті.

5. Сортування авіарейсів за часом відправлення

Напишіть програму, яка сортує список авіарейсів за часом відправлення за допомогою будь-якого методу сортування. Реалізуйте можливість сортувати як у прямому, так і в зворотному порядку.

Ускладнення: Додайте можливість сортувати за іншими параметрами, наприклад, за тривалістю рейсу чи відстанню. Порівняйте результати для різних методів сортування.

6. Сортування файлів за розміром

Напишіть програму, яка сортує список файлів у директорії за розміром.

Ускладнення: Порівняйте ефективність обраного алгоритму для сортування великої кількості файлів та невеликої кількості великих файлів. Виведіть також середній час виконання кожного алгоритму для різних наборів даних.

7. Сортування результатів спортивних змагань

Напишіть програму для сортування результатів учасників спортивного змагання за часом проходження дистанції. Результати повинні бути відсортовані від найшвидшого до найповільнішого.

Ускладнення: Якщо два учасники показали однаковий час, додайте можливість сортувати їх за іншими параметрами (наприклад, за віком або алфавітно). Порівняйте ефективність різних методів сортування на великих і малих масивах результатів.

8. Сортування книг за кількістю сторінок

Напишіть програму, яка сортує список книг за кількістю сторінок.

Ускладнення: Порівняйте різні методи сортування для списку книг з однаковою кількістю сторінок. Виведіть 5 книг з найбільшою кількістю сторінок після сортування та порівняйте продуктивність різних алгоритмів.

9. Сортування товарів за кількістю продажів

Напишіть програму, яка сортує товари за кількістю продажів.

Ускладнення: Додайте фільтр для сортування товарів, які мають більше 50 продажів. Порівняйте, як різні алгоритми сортування впливають на швидкість сортування великих масивів товарів.

10. Сортування дат у календарі подій

Напишіть програму, яка отримує список дат подій і сортує їх у хронологічному порядку.

Ускладнення: Якщо події відбуваються в один і той самий день, додайте можливість сортувати їх за назвою події. Порівняйте, який алгоритм є найефективнішим при сортуванні великої кількості дат.

11. Сортування масиву B і робота з його елементами

Задано не відсортований масив B, що містить 20 дійсних чисел. Знайдіть суму елементів масиву, що лежать на парних позиціях (0, 2, 4 тощо). Відсортуйте масив у порядку зростання та спадання, знайдіть суму елементів для кожного сформованого масиву. Порівняйте ці три суми та відсортуйте їх у порядку зростання.

*Ускладнення***:** Реалізуйте функцію, яка б виводила всі парні і непарні елементи масиву в окремі рядки. Додайте можливість вводити масив з клавіатури або генерувати випадкові числа.

12. Сортування масиву та обробка його частин

Задано не відсортований масив, що містить 45 дійсних чисел у межах від 0 до 70. Відсортуйте його у порядку зростання. Виберіть випадкове парне число з масиву та знайдіть його позицію. Елементи ліворуч від цієї позиції відсортуйте за зростанням, праворуч – за спаданням. Обчисліть суму лівих елементів масиву та середнє арифметичне правих елементів.

Ускладнення: Додайте перевірку на те, чи є парне число у масиві. Якщо немає, виведіть відповідне повідомлення.

13. Сортування двовимірного масиву

Задано двовимірний масив дійсних чисел розмірністю [15, 15]. Відсортуйте стовпці за спаданням елементів у першому рядку. Обчисліть суму елементів, розташованих на діагоналях отриманої матриці. Відсортуйте

елементи периметру матриці за зростанням та обчисліть їхнє середнє арифметичне. Виведіть на екран початкову та покроково отриману матриці.

Ускладнення: Реалізуйте алгоритм, який виведе матрицю в зручному для читання вигляді. Додайте можливість введення значень матриці з клавіатури або генерації випадкових значень.

14. Сортування та робота з парними та непарними елементами

Задано не відсортований масив B, що містить 100 дійсних чисел (випадково або з клавіатури). Відсортуйте масив у порядку зростання та спадання. Знайдіть суму всіх парних і непарних елементів для кожного відсортованого масиву. Обчисліть, у скільки разів сума парних елементів є меншою (або більшою) за суму непарних.

Ускладнення: Додайте можливість виводити на екран всі парні та непарні числа з масиву. Реалізуйте функцію для візуалізації результатів у вигляді гістограми.

15. Сортування та заміна елементів масиву

Задано не відсортований масив B, що містить 80 дійсних чисел (випадково або з клавіатури). Відсортуйте масив у порядку зростання та замініть усі парні числа на число 5, якщо їх кількість більша за 10; на -9, якщо кількість більша за 25; в інших випадках замініть на 0. Відсортуйте масив у порядку спадання та замініть усі непарні числа на 4, якщо їх кількість більша за 10; на 0, якщо більша за 15; в інших випадках – на 10.

Ускладнення: Визначте і виведіть кількість парних і непарних елементів масиву перед і після заміни. Додайте можливість динамічно вводити значення масиву.

16. Робота з лінійною таблицею

Дано лінійну таблицю з 95 дійсних чисел: Відсортуйте масив у порядку зростання та замініть від’ємні елементи їхніми квадратами. Знайдіть середнє

арифметичне елементів нового масиву. Відсортуйте масив у порядку спадання та замініть ірраціональні числа на 0, раціональні на 2. Обчисліть середнє арифметичне нового масиву.

Ускладнення: Реалізуйте метод для перевірки, чи є число раціональним або ірраціональним. Додайте можливість візуалізувати результати обробки у вигляді таблиці.

17. Сортування та робота з матрицею

Для матриці matr (9, 10) відсортуйте кожен рядок у порядку зростання, а кожен стовпець – у порядку спадання. Обчисліть суми елементів кожного рядка та стовпця та запишіть їх в одновимірний масив. Відсортуйте цей масив за спаданням та зростанням, знайдіть мінімальне та максимальне значення.

Ускладнення: Додайте можливість обчислення середнього арифметичного для кожного рядка та стовпця. Реалізуйте виведення матриці в зручному форматі.

18. Об'єднання двох масивів і сортування

З двох впорядкованих масивів a_i (n = 8) та b_j (m = 12) сформуйте третій, також впорядкований масив. Обчисліть суму елементів, що лежать на непарних позиціях у всіх трьох масивах. Відсортуйте ці три суми у порядку зростання. У створеному масиві відсортуйте рядки за спаданням, а стовпці – за зростанням.

Ускладнення: Реалізуйте можливість виводу елементів трьох масивів у вигляді таблиці. Додайте функцію для визначення кількості спільних елементів між трьома масивами.

19. Сортування відомості успішності студентів

Використовуючи динамічну пам'ять, обробіть відомість успішності студентів вашої групи з п'яти дисциплін («Програмування», «Комп'ютерна графіка», «Web», «Організація баз даних», «Операційні системи»). Виставте рейтинг для кожного предмету, підрахуйте середній бал групи та кількість

відмінників. Сформуйте спільний рейтинг з п’яти дисциплін та виведіть гістограму.

Ускладнення: Додайте можливість вводити оцінки з клавіатури та зберігати їх у файлі. Реалізуйте графічне відображення результатів за допомогою бібліотек для візуалізації даних.

20. Сортування та обробка двох масивів

З двох невпорядкованих масивів A (15) і B (17) сформуйте третій, упорядкований масив C, що містить елементи масиву A, які повторюються та є також у масиві B. Відсортуйте масив A у порядку спадання та знайдіть середнє арифметичне його елементів. Відсортуйте масив B у порядку зростання, знайдіть мінімум та максимум.

Ускладнення: Реалізуйте функцію для виведення повторюваних елементів та їх кількості. Додайте можливість введення масивів A та B з клавіатури або генерації випадкових значень. Знайдіть та виведіть суми елементів масивів A, B та C, а також їх середні арифметичні значення. Реалізуйте перевірку на порожні масиви та виведіть відповідні повідомлення. Додайте можливість зберігати масиви в файл і завантажувати їх для подальшої обробки.

21. Визначення добутків сусідніх чисел

Задано не відсортований масив C, що містить 50 дійсних чисел. Відсортувати масив у порядку спадання. Визначити суму добутків всіх пар сусідніх чисел. Відсортувати масив у порядку зростання та визначити середнє арифметичне елементів. Вивести на екран парні числа в одному рядку, а непарні числа в другому рядку відсортованого масиву.

Ускладнення: Замість простого сортування реалізуйте метод швидкого сортування. Додати функцію, яка б обчислювала добутки сусідніх чисел для масиву з випадковими значеннями.

22. Сортування та обчислення в матриці

Дано матрицю r розміром [15,15]. Відсортувати елементи з 1 по 7 рядок у порядку зростання, а з 8 по 15 рядок – у порядку спадання. Знайти суми елементів кожного рядка, визначити номер рядка, де сума буде максимальною, та записати їх в одновимірний масив, відсортувавши в порядку зростання. Обчислити середнє арифметичне елементів, розташованих на діагоналях отриманої матриці. Вивести на екран початковий і покроково отриманий масиви у вигляді матриці.

Ускладнення: Реалізуйте метод сортування злиттям для рядків матриці. Додайте можливість вводити значення матриці з клавіатури або генерувати випадкові значення.

23. Формування та сортування нових масивів

З двох невпорядкованих одновимірних масивів K(12) і N(13) сформуйте одновимірний масив розміром K+N. Спочатку впорядкуйте масиви по зростанню, а потім утворіть масив у зворотному порядку. Також створіть масив розміром K+N, впорядкований у спадання, спочатку впорядкувавши масиви по спаданню. Знайти мінімальне та максимальне числа двох створених масивів.

Ускладнення: Додайте можливість вибору способу сортування (вставками, Шелла, пірамідальне). Визначте, скільки разів елементи масиву, сформованого з K і N, зустрічаються в обох масивах.

24. Упорядкування елементів масиву

Задано не відсортований одномірний числовий масив A (90), який містить додатні і від’ємні числа. Скласти алгоритм, що упорядковує елементи масиву, які стоять на непарних місцях, у зростаючому порядку, а на парних – у порядку спадання. Знайти мінімальне та максимальне числа в отриманих масивах.

Ускладнення: Визначте кількість позитивних і негативних чисел у масиві, а також замініть їх на нулі відповідно до певних умов. Реалізуйте можливість повторного сортування масиву після модифікацій.

25. Сортування матриці та обчислення

Дано двовимірний масив дійсних чисел розмірністю [10,10]. Провести сортування рядків за спаданням. Переставити рядки так: перший з останнім, другий з передостаннім і так далі. В утвореному масиві провести сортування стовпців за спаданням. Знайти суми кожного зі стовпців, відсортувати дані суми за зростанням, знайти мінімальне, максимальне та середнє арифметичне.

Ускладнення: Реалізуйте динамічне виділення пам'яті для матриці. Додайте можливість обирати метод сортування для рядків і стовпців.

Контрольні запитання

1. Що таке сортування масиву? (Визначте, чому сортування є важливим у програмуванні.)
2. Які основні алгоритми сортування ви знаєте? (Перерахуйте принаймні три з них і надайте короткий опис кожного.)
3. Як працює бульбашкове сортування? (Опишіть, які дії виконує цей алгоритм на кожному кроці.)
4. Які основні переваги та недоліки бульбашкового сортування? (Порівняйте з іншими методами сортування.)
5. Що таке швидке сортування і в чому полягає його суть? (Яка його середня складність?)
6. Опишіть алгоритм злиття. Як він реалізується в коді? (Які є основні етапи цього алгоритму?)
7. Як ви можете виміряти час виконання алгоритму сортування? (Які інструменти або методи для цього можна використовувати?)

8. Що таке стабільні та нестабільні алгоритми сортування? (Наведіть приклади кожного з них.)

9. Яка складність алгоритму сортування злиттям у найгіршому випадку? (Поясніть, чому це так.)

10. Як можна візуалізувати процес сортування? (Які методи або інструменти для цього існують?)

11. Які особливості сортування масиву з дійсними числами в порівнянні з цілими? (Які додаткові нюанси можуть виникати?)

12. Що таке рекурсія і як вона застосовується в алгоритмах сортування? (Наведіть приклади використання рекурсії в швидкому сортуванні.)

13. Яким чином можна оптимізувати бульбашкове сортування? (Які техніки покращують його ефективність?)

14. Які алгоритми сортування є найбільш ефективними для великих обсягів даних? (Поясніть, чому ці алгоритми є кращими.)

15. Які практичні застосування алгоритмів сортування ви можете назвати? (Наведіть приклади з реального життя або програмного забезпечення.)

Лабораторна робота № 4
АЛГОРИТМ ДЕЙКСТРИ

Мета роботи:

- Ознайомтеся з основами алгоритму Дейкстри, дослідіть його принципи роботи, а також механізми вибору найкоротшого шляху на кожному етапі.
- Розвинути навички написання програмного коду для реалізації алгоритму Дейкстри на практиці та його застосування до завдань на графах, а також проведення аналізу отриманих результатів для реальних завдань.

Основні теоретичні відомості

Алгоритм Дейкстри знаходить найкоротшу відстань від однієї вершини графу до всіх решти, працює лише для графів у яких ребра не мають від'ємної ваги. Алгоритм широко застосовується в програмуванні і технологіях, наприклад, його використовують протоколи маршрутизації OSPF (спершу відкрити найкоротший шлях) та IS-IS (проміжна система до проміжної системи).

Граф називається ***зваженим***, коли всі його ребра мають вагу. ***Вагою*** ребра часто називають його довжину. Найбільш розповсюдженими способами зображення графів є:

*Списки суміжних вершин***:** для кожної вершини зберігаються сусідні вершини разом з вагами ребер, що їх з'єднують.

*Матриця суміжності***:** двовимірний масив, у якому елемент *(i,j)* представляє вагу ребра між вершинами *i* та *j*.

Для обходу графів існують два основні алгоритми:

Пошук у глибину (DFS): алгоритм, який досліджує максимально можливу глибину графа перед тим, як виконати назад.

Пошук по ширині (BFS): алгоритм, який досліджує всі сусідні вершини одного рівня перед переходом до наступного.

Принцип роботи алгоритму Дейкстри:

1. *Ініціалізація:* На початковому алгоритмі всім вершинам графа призначаються початкові значення відстаней, причому для початкової вершини відстань дорівнює нулю, а для решти – нескінченності.

2. *Вибір найкоротшого шляху:* На кожному етапі алгоритм обирає вершину з мінімальною поточною відстанню з множини ще не оброблених вершин. Ця вершина додається до множини вершин, для яких мінімальна відстань вже знайдена.

3. *Оновлення відстаней:* Для кожної сусідньої вершини вибраної вершини оновлюється її поточна мінімальна відстань, якщо шлях через щойно обрану вершину є коротшим.

4. Повторення: Процес вибору вершини та оновлення відстаней повторюється, доки всі вершини не будуть оброблені або не буде знайдено найкоротше.

ЗАВДАННЯ

Графи повинні мати не менше ніж 6 вершин. У звіт відповідно намалювати граф і матрицю суміжності. Програма має відображати реальні міста чи вулиці написані латиницею (кирилицею). На екран вивести довжини усіх можливих маршрутів. Показати графічно як відбувається перехід між вершинами.

1. Дана мережа автомобільних доріг, що з'єднують міста Львівської області. Можливо знайти найкоротшу відстань від Львова до таких міст, як Дрогобич, Самбір, Старий Самбір та Трускавець. Відстань між містами не обов'язково дорівнює відстань у зворотному напрямку. Відомо, що з Дрогобича є дороги до Самбора та Старого Самбора.

Ускладнення:

1. Додайте більше міст: до зазначених додайте ще два додаткові міста. Це збільшить кількість можливих маршрутів і ускладнить вибір оптимального шляху.

2. Обмеження на типи доріг: Врахуйте, що інші типи автомобільних доріг (швидкісні траси та місцеві дороги), які можуть впливати на відстань та час подорожі. Важливо зважати на цей фактор при пошуку найкоротшого шляху.

3. Візуалізація результатів: Спробуйте візуалізувати отримані результати у вигляді графіка або карти, щоб наочно побачити найкоротший маршрут між містами.

2. Наявна деяка кількість авіарейсів між містами світу, вартість кожного з них відома. До того ж вартість перельоту з міста А в місто В не обов'язково дорівнює вартості перельоту у зворотному напрямку. Визначити маршрут між двома заданими містами (можливо з пересадками, і не одніє, а декількома), який має мінімальну вартість.

Ускладнення:

1. Додати можливість врахування обмежень на *час польоту і прибуття* , які можуть вплинути на доступність рейсів. Наприклад, не всі рейси можуть бути доступні для пересадок через розклад або обмежений час для зміни літака.

2. Зробити так, щоб маршрут міг включати *максимальну кількість пересадок , але з обмеженням на загальний час подорожі* . Студентам необхідно вибрати оптимальний маршрут, враховуючи і пересадки, і час перебування у повітрі.

3. Включити *реальні дані про ціни на авіаквитки* , а також варіативність залежно від *дня тижня або доби*. Це дозволяє змоделювати зміну вартості авіарейсів залежно від того, коли буде створений переліт.

3. Є план міста з нанесеними на нього місцями розміщення пожежних частин, а саме 10. Знайти найближчу до вашого дому пожежну станцію, до того

ж відстань між частинами не обов’язково дорівнює відстані у зворотному напрямку.

Ускладнення:

1. Додати до завдання *додаткові фактори*, такі як *час реагування*, які змінюються залежно від завантаженості доріг. Враховувати зміну часу на дорогах в різні періоди отримання, використовувати найкращий час для досягнення.

2. Запропонувати студентам *розробити модель*, яка враховувала б не тільки відстань до пожежної станції, але *ймовірність виникнення надзвичайних ситуацій* у різних районах міста. Це створити більш складну модель оцінки ризиків і вибору найкращої станції.

3. Врахувати *не лише найближчу*, але й *найвидодоступнішу станцію* в умовах заторів. Необхідно моделювати ситуацію з більш високими рівнями заторів і розташувати станцію, яка найбільше може дістатися до місця пожежі.

4. Дана мережа автомобільних доріг міста Дрогобича. Знайти найкоротшу відстань від ТОЗ Тандем до залізничного вокзалу, якщо рухатися можна тільки по основних вулицях міста.

Ускладнення:

1. Додати можливість вибору *альтернативних маршрутів*, які можуть бути довшими за відстанню, але *менш завантаженими.* Це дозволяє студентам порівнювати ефективність різних шляхів.

2. Запропонувати студентам *змоделювати різні сценарії подорожі*. Наприклад, врахувати *погодні умови* або наявність *будівельних робіт*, які можуть зменшити швидкість пересування або зробити деякі шляхи недоступними.

3. Додайте елементи випадковості, такі як *корки, аварії*, або інші несподівані події, що ускладнюють поршук оптимального маршруту.

5. Дано усі корпуси ДДПУ ім. І.Франка із зазначеними адресами. Знайти найкоротшу відстань від ІФМІКТ до головного корпусу, якщо рухатися можна

тільки по основних вулицях міста. До того ж відстань між корпусами не обов’язково дорівнює відстані у зворотному напрямку.

Ускладнення:

1. Додати можливість вибору альтернативних маршрутів через інші корпуси університету, що можна збільшити маршрут, але зменшити затори або скоротити час подорожі.

2. Запропонувати студентам моделювати різні сценарії поїздки, враховуючи різні погодні умови (дощ або сніг), які можуть вплинути на швидкість пересування містом.

3. Додати обмеження на тип транспорту (пішки, велосипеди або автомобілі), що впливає на вибір маршрутів і загальний час подорожі.

4. Включити можливість врахування заторів на головних вулицях міста в різні години, що може змінити оптимальний маршрут.

6. Дана карта велосипедних доріжок Тернопільської та Львівської областей. Знайти мінімальну відстань, яку треба проїхати, щоб дістатися від Львова до Тернополя з урахуванням реальних даних. До того ж відстань між містами не обов’язково дорівнює відстані у зворотному напрямку.

Ускладнення:

1. Додати можливість врахування *різних типів велосипедних доріжок* (наприклад, гірські, рівнинні, або шосейні доріжки), які впливають на швидкість руху.

2. Врахувати обмеження на *максимальну кількість зупинок* під час маршруту, а також *час на відпочинок* .

3. Додати реальні дані про *погодні умови*, які можуть вплинути на маршрут (наприклад, дощ чи вітер, які складають певні частини маршруту складнішими для проїзду).

7. Дана мережа автомобільних доріг, що з’єднують міста Львівської області. Знайти найкоротшу відстань від Львова до кожного районного центру області, якщо рухатись можна тільки по дорогах.

Ускладнення:

1. Додати до завдання обмеження на *час доби*, коли можна рухатися по певних дорогах (наприклад, через закриття певних ділянок дороги вночі).
2. Врахувати можливість того, що *деякі дороги можуть бути перекриті* через ремонт або інші причини, що змусити студентів змінити маршрут у реальному часі.
3. Додати такий фактор, як *кількість палива*, яке потрібно для подорожі, і зробити завдання ще більш реалістичним, змушуючи студентів планувати заправки на шляху.

8. Дана карта велосипедних доріжок Латвії та Білорусі. Знайти мінімальну відстань, яку треба проїхати, щоб дістатися від Риги до Бобруйська.

Ускладнення:

1. Включити *варіанти перетину кордону*, де потрібно виконати час на контроль, а також можливість відмовитися через політичні або адміністративні фактори.
2. Додати обмеження на *максимальний час подорожі*, змушуючи студентів вибрати найбільш оптимальні за час маршруту.
3. Врахувати зміну рельєфу між країнами, що можна вплинути на *швидкість пересування*, зокрема при наявності пагорбів або гір.

9. Дана карта автомобільних доріг Волинської області. Знайти мінімальну відстань, яку треба проїхати, щоб дістатися від Шацька до Луцька, врахувати те що маршрут повинен прокладатись через Володимиро-Волинський, Рожищенський, Ківерцівський, Любешівський райони. Дані мають бути реальними.

Ускладнення:

1. Додати обмеження на *максимальну кількість проміжних зупинок*, щоб підвищити складність вибору оптимального маршруту.

2. Включити реальні дані про *завантаженість дороги* залежно від часу отримання, зокрема корки на головних трасах, які можуть вплинути на швидкість пересування.

3. Врахувати можливі *погодні зміни*, такі як сніг або дощ, які можуть зробити певні ділянки дороги менш проїзними.

10. Маємо мережу автомобільних доріг, яка з'єднує міста Дніпровської області. Деякі з них односторонні. Знайти найкоротші шляхи від Дніпра до 8 міст області (якщо рухатися можна тільки по дорогах, згідно із GPS).

*Ускладнення**:***

1. Додайте *односторонні дороги* на деяких ділянках, змушуючи вибирати тільки дозволені напрямки руху та враховувати їх при плануванні маршруту.

2. Включити можливість *відключення деяких доріг через аварії або інші непередбачені об'єкти*, що змусить студентів змінити свій маршрут на ходу.

3. Додати *різні типи транспортних засобів* (легкові автомобілі, вантажівки), кожен з яких може мати обмеження на певних дорогах, наприклад, через розміри або швидкість.

11. Дана карта велосипедних доріжок Тернопільської області. Знайти мінімальну відстань, яку треба проїхати, щоб дістатися від Бережан до Бродів, врахувати те що маршрут повинен прокладатись через місто Тернопіль. З врахуванням реальних даних.

Ускладнення:

1. Включити *вплив рельєфу* на швидкість пересування по певних ділянках маршруту, враховуючи гірські або рівнинні райони.

2. Додати обмеження на *кількість доступних велосипедних доріжок*, через які можна рухатися, враховуючи ремонтні роботи або погодні умови.

3. Запропонувати студентам *моделювати маршрут у різний час доби*, враховуючи зміну освітлення та стан дороги уночі.

12. Є план міста з нанесеними на нього місцями розміщення лікарень, а саме 7. Знайти найближчу до ратуші лікарню.

Ускладнення:

1. Включити обмеження на *доступність лікарень* залежно від часу (наприклад, деякі будуть тимчасово недоступні через перевантаженість або закриття на ніч).

2. Враховувати *завантаженість міських доріг* та можливість виникнення заторів у різний час отримання, що вплине на оптимальний маршрут.

3. Додати можливість *врахування пішохідних зон* або велосипедних доріжок, які можуть скоротити шлях, якщо рухатися без автомобіля.

13. Дано план пішохідних зон в Івано-Франківську, де практично відсутній автомобільний рух. Туристи бажають прогулятися цими двома мальовничими вулицями, допоможіть скласти GPS-маршрут із мінімальною затратою часу для відвідин пішохідних зон між вул. Січових Стрільців і вул. Дністровською. До того ж відстань між якими не обов'язково дорівнює відстані у зворотному напрямку.

Ускладнення:

1. Включити можливість *альтернативних маршрутів через пішохідні переходи*, які можуть бути довшими, але безпечнішими.

2. Врахувати фактори, такі як *погодні умови або наявність святкових ярмарків*, які можуть перекривати деякі частини піхохідних зон.

3. Запропонувати студентам *моделювати можливість перепони на шляху* (наприклад, натовпи туристів або ремонт дороги), що можуть вплинути на швидкість пересування.

14. Дана карта залізничного сполучення України. Знайти мінімальну відстань, яку треба проїхати, щоб дістатися від Львова до Кривого Рогу з з пересадками.

Ускладнення:

1. Включити можливість *пересадки з додатковим часом на очікування поїздів*, враховуючи затримки або розклад руху.
2. Враховувати *різні класи поїздок* (швидкісні, нічні, регіональні) та їх вплив на час подорожі та комфорт.
3. Додати обмеження на *максимальну кількість пересадок* або загальний час подорожі, що змусить студентів оптимізувати маршрут.

15. Дана карта аеропортів України. Знайти мінімальну відстань, щоб дістатися від Львова до Харкова з врахуванням реальних даних.

Ускладнення:

1. Врахувати *виліт та прибуття рейсів* у різний час доби, а також можливість *нічних пересадок* , що може вплинути на тривалість подорожі.
2. Додати можливість *вибору між прямими рейсами та рейсами з пересадками*, де студенти повинні вибрати оптимальний варіант за розрахунками часу та ціни.
3. Включити до завдання реальні дані про *ціни на авіаквитки*, які змінюються залежно від дня тижня та часу бронювання.

16. Дана мережа залізничних доріг Польщі. Знайти найкоротший шлях від Варшави до Кракова.

Ускладнення:

1. Включити можливість *вибору різних маршрутів з пересадками*, враховуючи різницю в розкладах поїздів і типах залізничних колій (швидкісні або звичайні).
2. Врахувати *затримки поїздів*, що можуть вплинути на вчасність пересадки.

3. Запропонувати студентам змоделювати *оптимальний маршрут* при різних обмеженнях, таких як обмежений бюджет або бажання скоротити час подорожі.

17. Дана карта автомобільних доріг між містами Київської області. Знайти найкоротший маршрут від Києва до Обухова.

Ускладнення:

1. Додати можливість *вибору між основними та другорядними дорогами* , де другорядні можуть бути менш завантаженими, але повільнішими.

2. Враховувати реальні дані про *ремонт доріг або обмеження руху*, які можуть змінити доступні маршрути.

3. Запропонувати студентам змоделювати *різні сценарії подорожі*, такі як поїздка в робочий час або у вихідні, що вплинули на швидкість пересування.

18. Є план міста Київ з нанесеними основними транспортними вузлами (метро, автобуси). Знайти найкоротший шлях від Майдану Незалежності до станції метро Академмістечко.

Ускладнення:

1. Враховувати можливість використання різних видів транспорту, таких як *метро, автобуси або таксі*, що впливає на час подорожі та вартість.

2. Додати обмеження на *час пересадки* між транспортними засобами, враховуючи затримки на пересадках або очікування наступного транспорту.

3. Запропонувати студентам *врахувати трафік та завантаженість метро* в різний час доби.

19. Дана мережа водних шляхів між островами Хорватії. Знайти найкоротший маршрут від міста Спліт до острова Хвар, враховуючи можливість пересадки на інші острови.

Ускладнення:

1. Додати можливість *вибору типу транспорту* (паром, яхта) з високою швидкістю та вартістю.

2. Врахувати *погодні умови*, які можуть змінити час подорожі або доступність певних маршрутів.

3. Запропонувати студентам *моделювати маршрути в різні сезони*, враховуючи зміни частоти руху паромів або можливості штормів.

20. Дана схема автобусного сполучення між містами Італії. Знайти мінімальну вартість поїздки від Риму до Венеції з пересадками.

Ускладнення:

1. Включити можливість *вибору різних класів автобусів* (експрес або звичайний), де швидкість пересування та вартість різняться.

2. Врахувати *обмеження на час очікування пересадки*, щоб уникнути надто довгих зупинок.

3. Додати можливість *вибору більших екологічних маршрутів* із меншим викидом CO2, що може вплинути на вибір транспорту.

21. Є мережа залізничного транспорту Великої Британії. Знайти найкоротший шлях від Лондона до Единбурга.

Ускладнення:

1. Врахувати *різницю у вартості квитків* залежно від часу придбання (передчасне бронювання або в день поїздки).

2. Додати можливість *врахування погодних умов***,** таких як снігопад або злива, які можуть вплинути на розклад руху поїздів.

3. Запропонувати студентам *змоделювати сценарій*, де вони можуть подорожчати лише в певні дні тижня, через що вибір маршруту обмежений.

22. Є карта річкових шляхів України. Знайти найкоротший маршрут між двома містами по річці Дніпро.

Ускладнення:

1. Додати можливість *вибору типу суден* (швидкі катери або звичайні річкові пароплави), враховуючи швидкість і комфорт.

2. Врахувати *сезонні зміни рівня води*, що можуть вплинути на доступність певних річкової інфраструктури.

3. Запропонувати студентам *врахувати обмеження на кількість зупинок*, щоб оптимізувати маршрут за часом або вартістю.

Контрольні питання

1. Що розуміють під терміном „граф ”?
2. Які вершини графа називаються суміжними?
3. Що розуміють під поняттям матриця інцидентності?
4. Що розуміють під поняттям матриця суміжності?
5. Для яких графі працює класичний алгоритм Дейкстри?
6. Який граф називають зваженим?
7. Що таке орієнтований графований?
8. Чим відрізняється інший орієнтований граф від неорієнтованого?
9. Яка різниця між простим та мультиграфом?
10. Що таке шлях і цикл у графі?
11. Які алгоритми використовують для обходу графів?
12. Що таке мінімальне кістякове дерево?
13. Як працює алгоритм для знаходження найкоротших шляхів?
14. Яка різниця між деревом і графом?
15. Як різниться ступінь вершини в графі?

Лабораторна робота № 5
АЛГОРИТМ ФЛОЙДА-ВОРШЕЛЛА

Мета роботи:

- Ознайомитися з основами алгоритмів динамічного програмування, дослідити принципи роботи алгоритму Флойда-Воршелла та його застосування для знаходження найкоротших шляхів між усіма парами вершин у зваженому графі
- Провести аналіз часової та просторової складності алгоритму Флойда-Воршелла, порівнюючи його з іншими алгоритмами для розв'язання задачі пошуку найкоротших шляхів, такими як алгоритм Дейкстри або Беллмана-Форда.
- Розвинути навички написання програмного коду та аналізу алгоритмів.

Основні теоретичні відомості

Алгоритм Флойда-Воршелла – це динамічний алгоритм для знаходження найкоротших шляхів між усіма парами вершин у зваженому графі. Алгоритм працює для графів, у яких ребра можуть мати від'ємні ваги, але він не підходить для графів із негативними циклами (цикли, сума ваг яких є від'ємною).

Основна ідея

Алгоритм поступово поліпшує оцінки найкоротших шляхів між вершинами, припускаючи, що ці шляхи можуть проходити через певну підмножину вершин. Він проходить через всі можливі вершини як потенційні проміжні точки і на кожному кроці перевіряє, чи можна поліпшти поточний найкоротший шлях.

Пояснення алгоритму

Алгоритм використовує матрицю відстаней, де елемент *d[i][j]* представляє найкоротшу відстань від вершини *i* до вершини *j*. Спочатку значення у цій матриці – це ваги ребер між вершинами, а якщо немає прямого ребра, то значенням є нескінченність.

На кожному етапі алгоритм перевіряє, чи можна скоротити шлях між вершинами *i* і *j*, якщо пройти через якусь іншу вершину *k*. Якщо це можливо, то шлях через вершину *k* стає новим найкоротшим шляхом між *i* і *j*.

```
for k = 1 to n:
    for i = 1 to n:
        for j = 1 to n:
            if d[i][j] > d[i][k] + d[k][j]:
                d[i][j] = d[i][k] + d[k][j]
```

Опис кроків

1. **Ініціалізація:** Спочатку для кожної пари вершин записуємо відстань між ними в матрицю. Якщо вершини не з'єднані, записуємо нескінченність.
2. **Ітерації:** На кожному кроці додаємо вершину *k* як потенційний проміжний пункт на шляху між усіма парами вершин і перевіряємо, чи поліпшить це поточний найкоротший шлях.
3. **Оновлення:** Якщо через вершину *k* знаходиться коротший шлях між вершинами *i* і *j*, оновлюємо значення в матриці.

Часова складність

Алгоритм Флойда-Воршелла має складність $\mathbf{O(n^3)}$, де *n* – кількість вершин у графі. Це робить його придатним для невеликих графів або графів з повним зв'язком, але неефективним для великих графів.

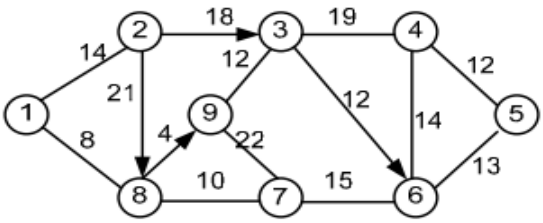
Переваги

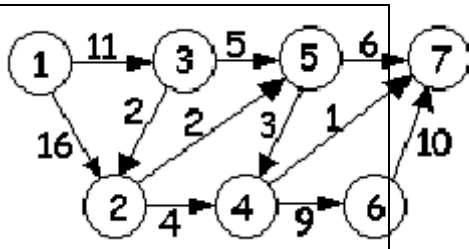
- Може працювати з графами, де є від'ємні ваги.
- Знаходить найкоротші шляхи між усіма парами вершин.

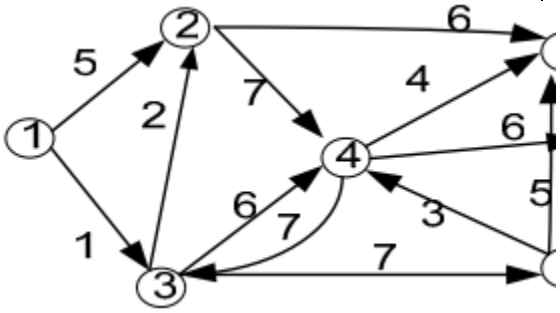
Недоліки

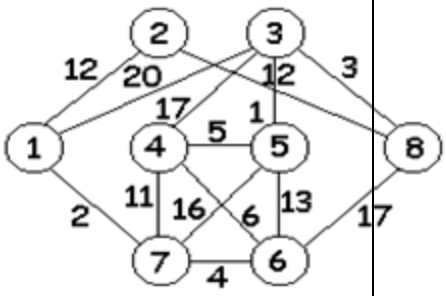
- Не працює для графів з негативними циклами (цикли, де сума ваг є від'ємною).
- Має високу часову складність для великих графів.

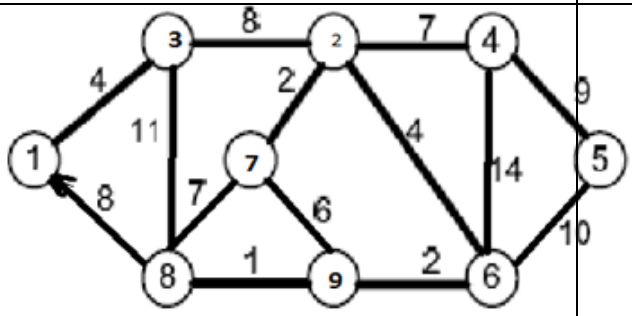
ЗАВДАННЯ

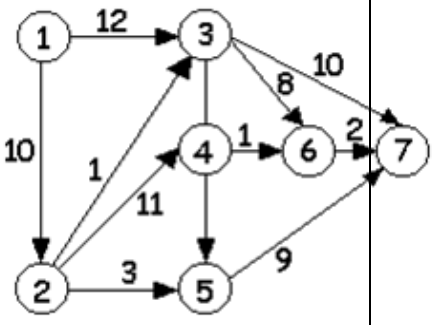

1
2
3
4
5
6
7
8
9
14
21
8
18
12
19
4
12
12
14
22
10
15
13

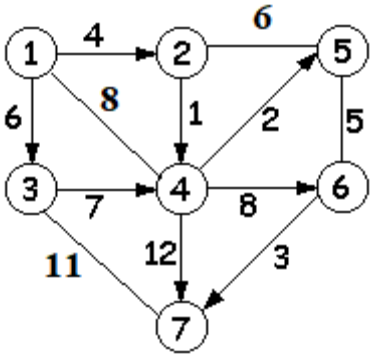

1
11
3
5
5
6
7
16
2
2
2
3
1
10
4
4
9
6

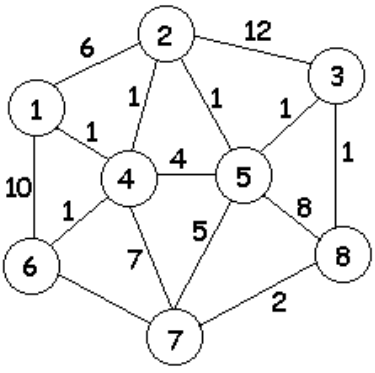

1
2
3
4
5
6
7
2
1
6
4
6
5
3
6
7
7

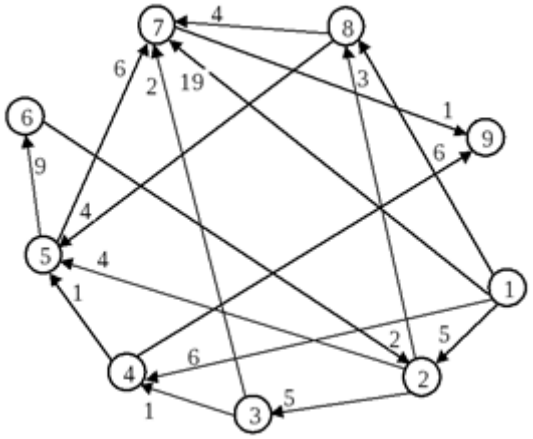
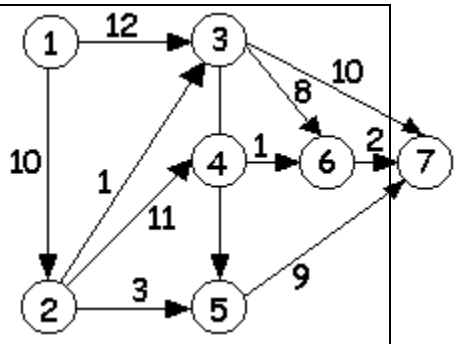
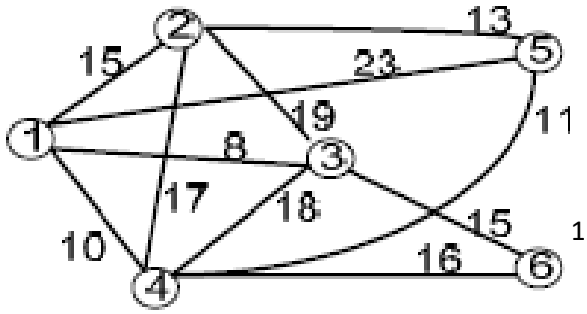
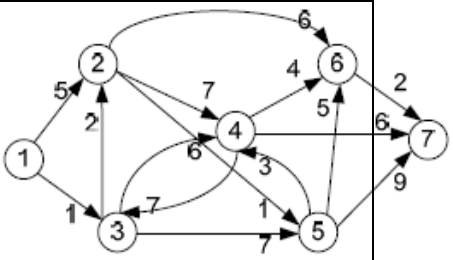
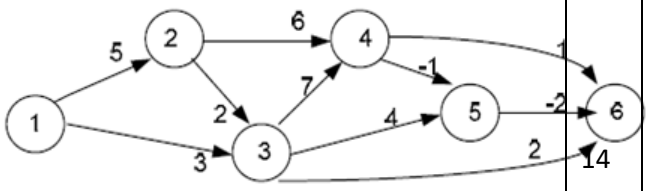
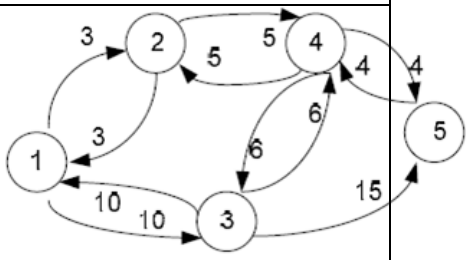
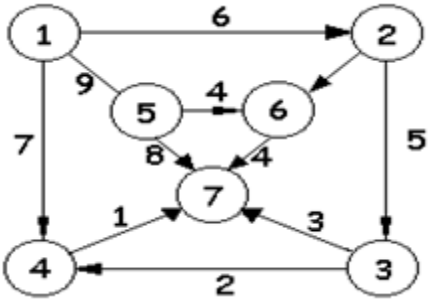
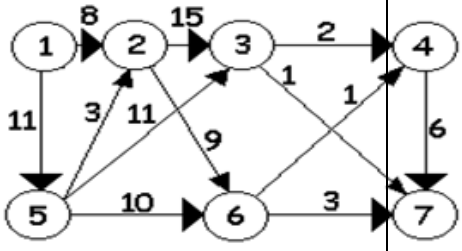
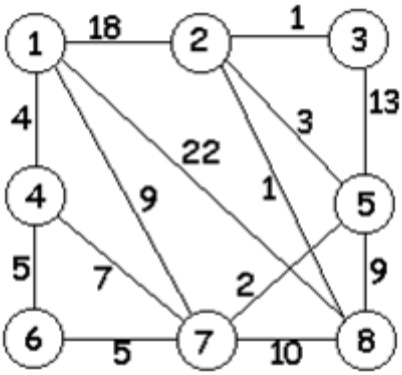
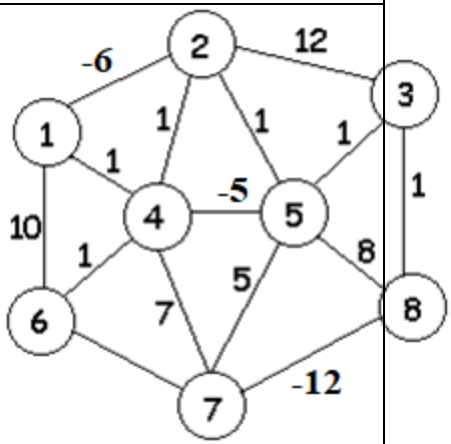

1
2
3
4
5
8
12
20
12
3
17
1
5
2
11
16
6
13
17
7
6
4

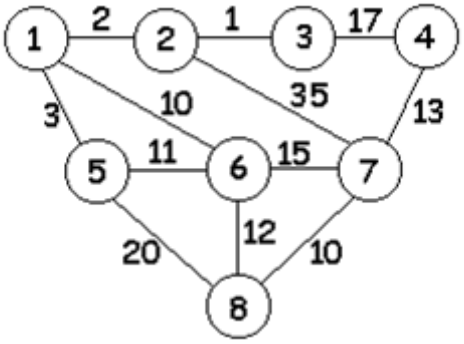
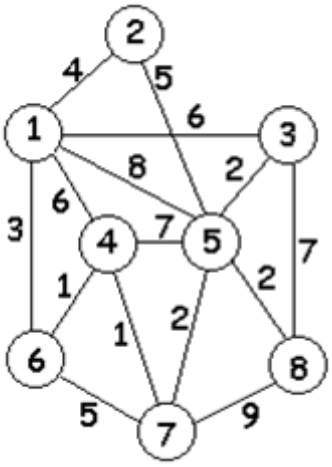
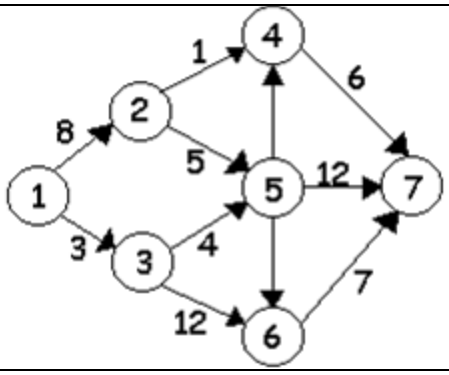
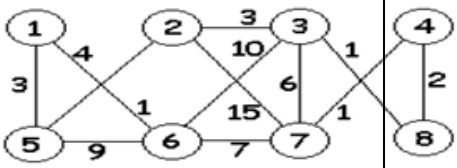
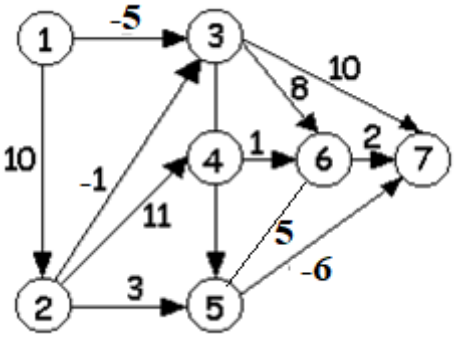
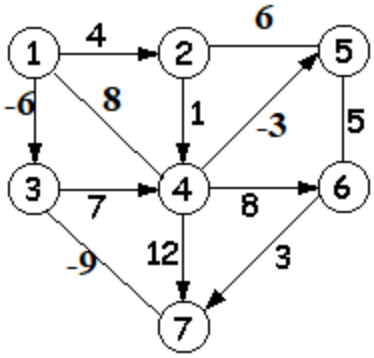
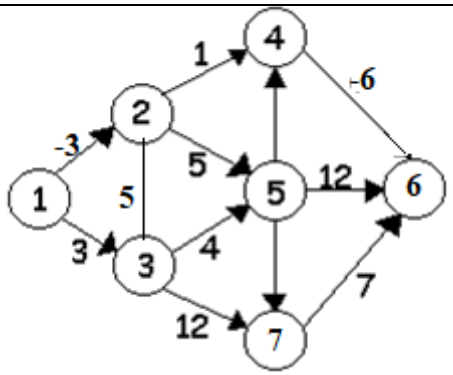
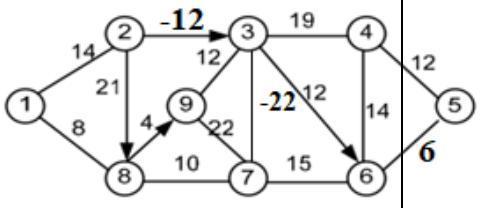

1
2
3
4
5
6
7
8
9
8
7
4
2
4
9
11
8
7
6
14
10
1
2


1
2
3
4
5
6
7
12
10
8
10
1
1
11
2
3
9


1
2
3
4
5
6
7
4
6
8
6
1
8
1
2
7
5
11
12
3


1
2
3
4
5
6
7
8
6
12
1
1
1
1
1
10
4
4
1
8
5
6
7
2

		0	
1		12	
13		14	
15		16	
7		8	

19		0	
21		22	
3		4	
5		6	

Контрольні питання

1. Які типи графів підходять для використання алгоритму Флойда-Воршелла?
2. Як алгоритм Флойда-Воршелла знаходить найкоротші шляхи між усіма парами вершин?
3. Яка основна ідея динамічного програмування, що використовується в алгоритмі Флойда-Воршелла?
4. Як відрізняється алгоритм Флойда-Воршелла від алгоритму Дейкстри?
5. Чому алгоритм Флойда-Воршелла не підходить для графів із негативними циклами?
6. Яка часова складність алгоритму Флойда-Воршелла?
7. Яким чином алгоритм Флойда-Воршелла оновлює відстані між вершинами?
8. Які структури даних використовуються для реалізації алгоритму Флойда-Воршелла?
9. Як можна використати алгоритм Флойда-Воршелла для знаходження транзитивного замикання графа?
10. Чи можна застосовувати алгоритм Флойда-Воршелла для орієнтованих графів? Якщо так, як?
11. Як виявити наявність негативних циклів за допомогою алгоритму Флойда-Воршелла?
12. Як виглядає матриця суміжності для графа, який використовується в алгоритмі Флойда-Воршелла?
13. У яких практичних задачах, крім пошуку найкоротших шляхів, можна використовувати алгоритм Флойда-Воршелла?
14. Як алгоритм Флойда-Воршелла можна оптимізувати для великих графів?

Лабораторна робота № 6

ЖАДІБНІ АЛГОРИТМИ

Мета роботи:

- Ознайомитися з основами жадібних алгоритмів, дослідити їхні принципи роботи та механізми прийняття рішень на кожному етапі
- Провести аналіз часової та просторової складності жадібних алгоритмів, порівнюючи їх з іншими підходами до розв'язання аналогічних задач.
- Розвинути навички написання програмного коду та аналізу алгоритмів.

Основні теоретичні відомості

Жадібні алгоритми – це клас алгоритмів, які вибирають найкраще можливе рішення на кожному кроці, зважаючи тільки на поточну ситуацію, не намагаючись оптимізувати майбутні результати. Основна ідея полягає у тому, щоб на кожному етапі обирати найбільш вигідний варіант, сподіваючись, що локальний оптимум призведе до глобального оптимуму.

Основні характеристики жадібних алгоритмів

Локальний вибір: На кожному етапі алгоритм вибирає найкращий варіант, базуючись на поточній інформації. Наприклад, при розв'язанні задачі про рюкзак, алгоритм може обирати найбільш вартісні предмети, поки це не стає неможливим.

Оптимальність: Для певних задач жадібний підхід призводить до оптимального рішення (наприклад, алгоритм Хаффмана для стиснення даних), але в багатьох випадках він може бути неоптимальним. Жадібний алгоритм не завжди забезпечує глобально оптимальний результат, навіть якщо вибір локального оптимуму виглядає розумним.

Простота: Жадібні алгоритми зазвичай легко реалізуються і розуміються, оскільки вони слідують простому принципу – обирати те, що здається найкращим на цей момент.

Переваги жадібних алгоритмів

Швидкість: Жадібні алгоритми часто виконуються швидше, ніж інші підходи, оскільки не вимагають перебору всіх можливих варіантів. Це робить їх дуже корисними в практичних застосуваннях.

Легкість реалізації: Ці алгоритми зазвичай менш складні в реалізації, що зменшує ризик помилок під час програмування та тестування.

Недоліки жадібних алгоритмів

Не завжди оптимальні: Жадібний підхід не гарантує оптимального рішення для всіх задач. Наприклад, у задачі про рюкзак з цілими предметами жадібний алгоритм може дати підоптимальний результат.

Вибір локального оптимуму: Оскільки алгоритм приймає рішення, не враховуючи наслідки для подальших кроків, це може призвести до вибору рішень, які виявляться поганими в довгостроковій перспективі.

Жадібні алгоритми є потужним інструментом для розв'язання різноманітних задач, і їх застосування вимагає оцінки конкретної ситуації. Важливо розуміти, що жадібний підхід не є універсальним і може не завжди приводити до оптимального рішення, але він є ефективним і простим способом розв'язання багатьох задач у реальному житті.

ЗАВДАННЯ

1. **Розподіл шоколадок:** Задано n дітей і m шоколадок різної вартості. Розподіліть шоколадки так, щоб максимальна вартість, отримана дитиною, була мінімальною.

Ускладнення: Визначте оптимальне рішення за допомогою алгоритму бінарного пошуку.

2. **Задача про активності:** Задано список активностей з початком та закінченням. Виберіть максимальну кількість активностей, що не перетинаються.

Ускладнення: Врахуйте, що деякі активності можуть мати різні пріоритети.

3. **Створення кодів Хаффмана:** Задано набір символів і їх частоти. Сформуйте коди для символів, використовуючи алгоритм Хаффмана, і виведіть отримані коди.

Ускладнення: Реалізуйте декодування отриманих кодів, щоб перевірити їхню правильність.

4. **Задача про розрізання стрижня:** Задано стрижень довжини n та масив цін для різних довжин. Знайдіть максимальний прибуток, якщо стрижень може бути розрізаний на кілька частин.

Ускладнення: Додайте обмеження на максимальну кількість розрізів.

5. **Оптимізація маршрутів:** У вас є кілька вантажівок, які потрібно доставити в різні пункти призначення. Знайдіть оптимальний маршрут, щоб зменшити загальну витрату пального.

Ускладнення: Врахуйте затори на маршрутах і змініть пріоритети.

6. **Купівля акцій:** Задано дні та ціни акцій. Сформулюйте стратегію купівлі та продажу акцій, щоб максимізувати прибуток, купуючи лише один раз.

Ускладнення: Додайте ймовірність зміни цін, щоб визначити ризики.

7. **Задача про рюкзак:** Задано масив предметів з вагою та вартістю. Реалізуйте жадібний алгоритм для максимізації вартості предметів, які можна вмістити в рюкзак з обмеженою вагою.

Ускладнення: Додайте обмеження на кількість кожного предмета, що можна взяти.

8. **Мінімальна кількість монет:** Вам потрібно видати решту певної суми. Задано номінали монет. Знайдіть мінімальну кількість монет для видачі решти.

Ускладнення: Включіть випадкову затримку для видачі кожної монети, що може вплинути на загальний час.

9. **Сортування завдань за пріоритетом:** Задано список завдань з різними термінами виконання та важливістю. Визначте порядок виконання завдань для максимізації загальної важливості.

Ускладнення: Включіть фактор часу на виконання кожного завдання.

10. **Групування студентів за успішністю:** Задано оцінки студентів. Використайте жадібний підхід для групування студентів у команди так, щоб різниця середніх оцінок у командах була мінімальною.

Ускладнення: Додайте обмеження на максимальну кількість студентів у команді.

11. **Задача про мінімальне остовне дерево:** Задано граф, в якому потрібно знайти мінімальне остовне дерево, використовуючи алгоритм Крускала або Прима.

Ускладнення: Додайте випадкові важливі ребра, що можуть змінювати структуру графа.

12. **Оптимізація покупки:** Задано список товарів з цінами та знижками. Визначте, як купити максимальну кількість товарів, щоб витрати були мінімальними.

Ускладнення: Включіть обмеження на бюджет та максимальну кількість товарів.

13. **Кодування символів:** Задано рядок символів. Використовуючи жадібний алгоритм, створіть оптимальні коди для символів, щоб зменшити загальний обсяг даних.

Ускладнення: Реалізуйте алгоритм для автоматичного виявлення частот символів.

14. **Доставлення вантажів:** Задано різні вантажі з вагою та термінами доставки. Сформуйте план доставки, використовуючи жадібний підхід для мінімізації затримок.

Ускладнення: Додайте обмеження на максимальну кількість вантажів, які можуть бути доставлені одночасно.

15. **Оптимізація складу:** Задано різні товари на складі з обмеженнями на обсяг. Визначте оптимальний розподіл товарів, щоб максимізувати прибуток.

Ускладнення: Включіть витрати на зберігання товарів.

16. **Задача про туристичні маршрути:** Задано кілька туристичних об'єктів з вартістю візиту. Сформуйте оптимальний маршрут для відвідування об'єктів з максимальною сумарною вартістю.

Ускладнення: Включіть обмеження на час відвідування кожного об'єкта.

17. **Задача про вибір сімейного бюджету:** Задано список витрат з їхніми пріоритетами. Визначте, які витрати слід покрити з обмеженим бюджетом, щоб максимізувати загальну задоволеність.

Ускладнення: Врахуйте непередбачені витрати та можливі знижки.

18. **Оптимізація виробництва:** Задано різні процеси виробництва з їхніми витратами та прибутком. Визначте, які процеси слід запустити для максимізації прибутку з урахуванням обмежень на ресурси.

Ускладнення: Включіть ризики, пов'язані з кожним процесом.

19. **Задача про вибір варіантів подорожі:** Задано різні варіанти подорожі з їхніми витратами та тривалістю. Визначте оптимальний маршрут з максимальним комфортом та мінімальними витратами.

Ускладнення: Включіть відгуки інших мандрівників про кожен варіант.

20. **Оптимізація розподілу енергії:** Задано кілька джерел енергії з різними витратами на виробництво та споживання. Визначте, як оптимально розподілити енергію між споживачами, щоб зменшити загальні витрати.

Ускладнення: Включіть непередбачувані зміни в споживанні енергії.

21**. Задача про планування транспорту**: Є кілька транспортних засобів з різною швидкістю і витратами на пальне. Потрібно оптимізувати вибір транспорту для перевезення товарів, щоб мінімізувати загальні витрати і час доставки.

Ускладнення: Включіть умови погоди, які можуть вплинути на швидкість або витрати на паливо.

22. Оптимізація будівельних ресурсів: Задано кілька проєктів, для яких необхідні будівельні матеріали. Мета – розподілити ресурси між проєктами так, щоб мінімізувати час будівництва і витрати.

Ускладнення: Деякі проєкти мають бути завершені раніше за інші, і матеріали можуть бути недоступними в певний час.

23. **Задача про пакування рюкзака**: Є набір предметів, кожен з яких має свою вартість і вагу. Завдання – вибрати предмети таким чином, щоб максимізувати загальну вартість предметів у рюкзаку, не перевищуючи його допустиму вагу.

Ускладнення: Деякі предмети можуть бути взаємозалежними, тобто якщо вибраний один предмет, то інший вибирати не можна.

24. **Задача про розподіл медичних ресурсів**: Є кілька лікарень і обмежена кількість медичних препаратів або обладнання. Потрібно оптимізувати розподіл ресурсів, щоб максимізувати кількість вилікуваних пацієнтів.

Ускладнення: Врахуйте різні ступені потреби в ресурсах для пацієнтів з різними рівнями тяжкості хвороб.

25. **Задача про інвестування**: Є кілька інвестиційних проєктів з різними рівнями прибутковості та ризиками. Завдання – вибрати, в які проекти вкласти гроші, щоб максимізувати загальну прибутковість за умов обмеженого бюджету.

Ускладнення: Включіть змінні ринкові умови, які можуть вплинути на прибутковість інвестицій.

26. **Оптимізація робочого графіка**: Є кілька співробітників, кожен з яких має свої обмеження по часу роботи і різну продуктивність. Завдання – скласти графік роботи, щоб максимізувати продуктивність команди.

Ускладнення: Врахуйте додаткові обмеження, як-от різні зміни або збої в роботі через технічні проблеми.

Контрольні запитання

1. Що таке жадібний алгоритм і як він працює?
2. Які ключові властивості повинна мати задача, щоб до неї можна було застосувати жадібний алгоритм?
3. Чим жадібні алгоритми відрізняються від динамічного програмування?
4. Поясніть на прикладі, як працює жадібний алгоритм для задачі про здачу решти.

5. Як працює жадібний алгоритм Краскала для знаходження мінімального остовного дерева (MST)?

6. У чому полягає жадібний алгоритм для задачі про рюкзак з дробовими вагами?

7. Які ситуації можуть призвести до неефективних рішень жадібних алгоритмів?

8. Чи гарантує жадібний алгоритм оптимальний результат у задачі про інтервальне планування?

9. Як можна застосувати жадібний підхід до задачі про покриття множини?

10. Наведіть приклади реальних застосувань жадібних алгоритмів у повсякденному житті чи технічних системах.

11. Як жадібний алгоритм розв'язує задачу про оптимальний вибір активностей?

12. Чому жадібний алгоритм не завжди є оптимальним для задачі про рюкзак із цілими вагами ?

13. Як працює жадібний алгоритм для задачі Хаффмана і яка його мета?

14. Які критерії необхідно перевірити для обґрунтування коректності жадібного алгоритму?

15. Чому важливо доводити оптимальність жадібних алгоритмів, і які методи для цього існують?

Лабораторна робота № 7

КРИПТОАЛГОРИТМИ

Мета роботи:

- Ознайомитися з типами криптографічних алгоритмів (симетричні, асиметричні, хеш-функції) та їх застосуванням.
- Дослідити базові концепції шифрування та розшифрування даних.
- Розвинути навички написання програмного коду та аналізу алгоритмів.

Основні теоретичні відомості

Криптоалгоритми це алгоритми, що використовуються в криптографії для забезпечення безпеки даних, шляхом їх шифрування та розшифрування. Криптоалгоритми розроблені з метою захисту інформації від несанкціонованого доступу і забезпечення її цілісності та конфіденційності.

Існує кілька типів криптоалгоритмів:

1. **Симетричні крипто алгоритми** – використовують один і той самий ключ для шифрування і розшифрування. Приклад: AES (Advanced Encryption Standard), DES (Data Encryption Standard).
2. **Асиметричні криптоалгоритми** – використовують пару ключів: відкритий ключ для шифрування і закритий ключ для розшифрування. Приклад: RSA, ECC (Elliptic Curve Cryptography).
3. **Хеш-функції** – використовуються для створення унікального цифрового підпису повідомлення або даних, не маючи можливості відновити оригінальні дані. Приклад: SHA (Secure Hash Algorithm), MD5.

Під криптографією розуміються методи та засоби, що забезпечують захист інформації шляхом її шифрування, тобто перетворення повідомлень на таку форму, яка є незрозумілою для сторонніх осіб. Криптографія включає

техніки, що використовуються для забезпечення конфіденційності, цілісності, автентифікації та незаперечності повідомлень.

Криптоаналітика, зі свого боку, спрямована на вивчення та розробку методів, які дозволяють подолати криптографічний захист без знання ключа шифрування. Це включає аналіз алгоритмів шифрування для пошуку їх слабких місць.

Сукупність криптографії та криптоаналітики утворює ширшу галузь знань, що називається криптологією. Вона охоплює всі аспекти захисту інформації та аналізу методів цього захисту.

Розшифровування означає процес відновлення оригінального повідомлення за наявності ключа шифрування, тобто коли одержувач повідомлення володіє всіма необхідними засобами для його читання.

Дешифрування – це спроба відновити оригінальне повідомлення без знання ключа шифрування, що зазвичай роблять криптоаналітики або зловмисники, намагаючись перехопити і зрозуміти захищене повідомлення.

Таким чином, ті, кому призначено зашифроване повідомлення, його розшифровують, а сторонні особи або зловмисники намагаються дешифрувати це повідомлення.

Шифр Цезаря – це метод шифрування, який замінює кожну букву тексту на іншу букву, розміщену на певній кількості позицій вперед або назад в алфавіті. Кількість позицій, на які переміщується буква, називається *ключем*.

Наприклад, якщо ключ дорівнює 3, то:

Буква "А" буде замінена на "Г"

Буква "Б" буде замінена на "Д"

Буква "Я" буде замінена на "Е" (з початку алфавіту)

Основні кроки алгоритму

✓ **Вибір ключа**: Визначте значення ключа, яке визначає, на скільки позицій в алфавіті буде зсунуто кожну букву.

✓ **Шифрування**:

Для кожного символу у вхідному повідомленні:

➢ Якщо символ є літерою, замініть його на символ, який знаходиться на n позицій далі в алфавіті (де n — ключ).

➢ Якщо символ не є літерою (наприклад, пробіл або знак пунктуації), залиште його без змін.

✓ **Дешифрування**:

➢ Використовуйте той самий ключ, але зворотний напрямок: замініть символи на n позицій раніше в алфавіті.

Математична модель шифру Цезаря може бути представлена у вигляді формул, які описують процес шифрування та дешифрування. Основні елементи цієї моделі включають:

1. Алфавіт

Нехай A – це набір символів алфавіту. Для української мови алфавіт може містити 33 літери:

А={А,Б,В,Г,Ґ,Д,Е,Є,Ж,З,И,І,Ї,Й,К,Л,М,Н,О,П,Р,С,Т,У,Ф,Х,Ц,Ч,Ш,Щ,Ь,Ю,Я}

2. Ключ

Нехай k – це ключ шифрування, який визначає кількість позицій, на які символи алфавіту зсуваються. Ключ може бути цілим числом від 0 до $n-1$, де n – кількість символів у алфавіті.

3. Процес шифрування

Шифрування символу ссс з алфавіту виконується за формулою:

$$E(c)=(c+k) \mod n$$

де:

- $E(c)$ – зашифрований символ,
- c – позиція символу в алфавіті (індексація починається з 0),
- k – ключ шифрування,
- n – кількість символів в алфавіті.

4. Процес дешифрування

Дешифрування символу c' виконується за формулою:

$$D(c')=(c'-k+n) \mod n$$

де:

- $D(c')$ – розшифрований символ,
- c' — позиція зашифрованого символу в алфавіті.

Шифр Віженера – це поліалфавітний шифр, що застосовує ключ для шифрування тексту, забезпечуючи вищий рівень безпеки в порівнянні з простими шифрами, такими як шифр Цезаря. Алгоритм був розроблений французьким математиком Блезом де Віженером у 16 столітті.

1. Основні компоненти

✓ **Алфавіт**: Використовується стандартний алфавіт, наприклад, для української мови, який містить 33 літери.

✓ **Ключ**: Це слово або фраза, яка повторюється, щоб покрити всю довжину відкритого тексту. Ключ впливає на кожну букву тексту, змінюючи, як саме вона буде зашифрована.

2. Таблиця Віженера

Таблиця Віженера – це структура, що містить всі можливі зсуви алфавіту. Кожен рядок таблиці є циклічно зсунутим алфавітом, що починається з різних літер. Наприклад, якщо використовуємо латинський алфавіт, перші кілька рядків таблиці можуть виглядати так:

Таблиця для шифрування

	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я
а	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я
б	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а
в	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б
г	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в
ґ	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г
д	д	е	є	ж	з	и	і	ї	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ
е	е	є	ж	з	и	і	ї	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д
є	є	ж	з	и	і	ї	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е
ж	ж	з	и	і	ї	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е	є
з	з	и	і	ї	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е	є	ж
и	и	і	ї	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е	є	ж	з
і	і	ї	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е	є	ж	з	и
ї	ї	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е	є	ж	з	и	і
й	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї
к	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й
л	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к
м	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л
н	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м
о	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м	н
п	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м	н	о
р	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м	н	о	п
с	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м	н	о	п	р
т	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м	н	о	п	р	с
у	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м	н	о	п	р	с	т
ф	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м	н	о	п	р	с	т	у
х	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м	н	о	п	р	с	т	у	ф
ц	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м	н	о	п	р	с	т	у	ф	х
ч	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц
ш	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч
щ	щ	ь	ю	я	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш
ь	ь	ю	я	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ
ю	ю	я	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь
я	я	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю

3. Процес шифрування

Шифрування повідомлення за допомогою шифру Віженера виконується так:

1. **Підготовка тексту**: Відкритий текст очищається від пробілів та спеціальних символів.

2. **Повторення ключа**: Ключ повторюється, поки не досягне довжини відкритого тексту.

3. **Шифрування**:

✓ Для кожної букви відкритого текстує відповідна буква з ключа.

✓ Визначається позиція відкритої літери в алфавіті та позиція літери ключа.

✓ Сумуються позиції обох літер, а результат береться за модулем кількості літер в алфавіті.

✓ Отримана позиція у таблиці Віженера визначає зашифровану букву.

ЗАВДАННЯ ПО ШИФРУ ЦЕЗАРЯ

1. **Змішаний алфавіт**: Зашифруйте текст "Криптографія" з ключем 3, використовуючи змішаний алфавіт (наприклад, починаючи з "Г, Д, Е, Є" замість "А, Б, В").
2. **Декодування з підказками**: Дайте текст "Шифрування" з ключем, але надайте 3 підказки про можливі ключі (наприклад, "менше 5", "парне число", "менше 10").
3. **Шифр з пропусками**: Розшифруйте текст "Кр-Ипт-граф!" з ключем 2, враховуючи, що пропуски в тексті не змінюються.
4. **Використання чисел**: Зашифруйте текст "2023 рік" з ключем 4, ігноруючи цифри, та поясніть, чому вони залишаються незмінними.
5. **Декодування на основі частотності**: Аналізуйте шифрований текст "Лепрекон" і визначте ключ за частотною таблицею української мови.
6. **Дешифрування з помилками**: Дайте зашифрований текст "Виноград" з помилковими символами (наприклад, "Ви@но!град") і визначте ключ.
7. **Алгоритм для всіх символів**: Розширте шифр Цезаря, щоб він міг обробляти не лише букви, а й символи, пропускаючи їх при шифруванні.
8. **Шифрування з динамічним ключем**: Реалізуйте алгоритм, де ключ змінюється на кожному кроці, наприклад, перша буква +1, друга +2 і так далі.
9. **Випадкові вставки**: Шифруйте текст "Загадка" з ключем 3, вставляючи випадкові символи через кожну букву (наприклад, "З#а@г$а%д&к*а").
10. **Генерація ключа**: Створіть генератор випадкових ключів для шифрування тексту "Таємниця" та поясніть, як ключ впливає на результат.
11. **Атака на шифр**: Дайте декілька шифрованих фраз з однаковим ключем і спробуйте відновити оригінальні фрази без відомого ключа.

12. **Порівняння з іншими шифрами**: Зашифруйте один і той же текст за допомогою шифру Цезаря та шифру Віженера, порівняйте результати та обговоріть переваги та недоліки.

13. **Планування атак**: Складіть план атаки на шифр, вказуючи можливі способи дешифрування тексту, зашифрованого з ключем 1-25.

14. **Розв'язання рівнянь**: Використовуйте шифр Цезаря для зашифрування математичного виразу, наприклад, "x+ 2 = 5", та дайте його дешифрувати.

15. **Групове шифрування**: Зашифруйте 5 різних слів з різними ключами, а потім створіть алгоритм для їх дешифрування одночасно.

16. **Текстове повідомлення**: Напишіть текстове повідомлення з 50 символів, зашифруйте його за допомогою шифру Цезаря, а потім поясніть, як обрати правильний ключ для дешифрування.

17. **Шифр у грі**: Створіть ігровий сценарій, в якому персонаж повинен знайти шифроване повідомлення за допомогою шифру Цезаря.

18. **Шифрування в контексті**: Дайте короткий уривок з твору (наприклад, "Кобзаря") та зашифруйте його, обравши відповідний ключ, та поясніть, чому вибрали саме цей ключ.

19. **Зміна напрямку шифрування**: Зашифруйте текст "Математика" з ключем 5, але змініть напрямок шифрування на зворотний.

20. **Аналіз шифру**: Проаналізуйте зміни у частоті букв в тексті до і після шифрування "Дослідження" з ключем 3.

21. **Відновлення тексту**: Відновіть текст з зашифрованого повідомлення "Ядро", знаючи, що ключ 6.

22. **Кодування кирилиці**: Розширте алгоритм, щоб він міг зашифровувати текст, написаний кирилицею, з ключем 7.

23. **Кодування з одним пробілом**: Напишіть текст з пропусками, зашифруйте його, але зберігайте лише один пробіл між словами.

24. **Тест на аномалії**: Створіть текст з аномаліями, такими як повторювані символи, та зашифруйте його, пояснюючи можливі проблеми при дешифруванні.

25. **Кодування віршів**: Напишіть вірш і зашифруйте його, використовуючи ключ 4, а потім розшифруйте, повертаючись до оригіналу.

26. **Кодування з використанням формул**: Включіть формули у текст (наприклад, "E=mc^2") та зашифруйте з ключем 2.

ЗАВДАННЯ НА ШИФРУВАННЯ ТА ДЕШИФРУВАННЯ ВИКОРИСТАННЯМ ШИФРУ ВІЖЕНЕРА

1. Зашифруйте текст "HELLOWORLD" з ключем "KEY".
✓ Розшифруйте текст "QWERTYUIOP" з ключем "PASSWORD".
2. Зашифруйте текст "SECURITY" з ключем "GUARD".
✓ Розшифруйте текст "MELONFARM" з ключем "FRUIT".
3. Зашифруйте текст "PYTHON" з ключем "CODE".
✓ Розшифруйте текст "CIPHER" з ключем "LOCK".
4. Зашифруйте текст "COMPUTER" з ключем "SCIENCE".
✓ Розшифруйте текст "PROGRAMMING" з ключем "TECHNOLOGY".
5. Зашифруйте текст "DATABASE" з ключем "STORAGE".
✓ Розшифруйте текст "INFORMATION" з ключем "DATA".
6. Зашифруйте текст "ALGORITHM" з ключем "ANALYZE".
✓ Розшифруйте текст "SOFTWARE" з ключем "APPLICATION".
7. Зашифруйте текст "NETWORK" з ключем "CONNECT".
✓ Розшифруйте текст "SYSTEMS" з ключем "COMPUTE".
8. Зашифруйте текст "APPLICATION" з ключем "SOFTWARE".
✓ Розшифруйте текст "ENCRYPTION" з ключем "SECURE".
9. Зашифруйте текст "BLOCKCHAIN" з ключем "CHAIN".
✓ Розшифруйте текст "DECIPHER" з ключем "ENCRYPT".
10. Зашифруйте текст "MACHINELEARNING" з ключем "AI".
✓ Розшифруйте текст "BIGDATA" з ключем "ANALYTICS".
11. Зашифруйте текст "DATAANALYSIS" з ключем "ANALYZE".

✓ Розшифруйте текст "RESEARCH" з ключем "FINDINGS".

12. Зашифруйте текст "DATABASEMANAGEMENT" з ключем "ADMIN".

✓ Розшифруйте текст "PLATFORM" з ключем "SERVICE".

13. Зашифруйте текст "WEBDEVELOPMENT" з ключем "CODEBASE".

✓ Розшифруйте текст "HOSTING" з ключем "SERVER".

14. Зашифруйте текст "CLOUDCOMPUTING" з ключем "INTERNET".

✓ Розшифруйте текст "SUPPLIER" з ключем "PROVIDER".

15. Зашифруйте текст "VIRTUALREALITY" з ключем "3DTECH".

✓ Розшифруйте текст "GAMING" з ключем "FUN".

16. Зашифруйте текст "SECURECOMMUNICATION" з ключем "ENCRYPT".

✓ Розшифруйте текст "PROTECTION" з ключем "SAFETY".

17. Зашифруйте текст "INFORMATIONSECURITY" з ключем "RISK".

✓ Розшифруйте текст "ANALYSIS" з ключем "STUDY".

18. Зашифруйте текст "E-COMMERCE" з ключем "SHOP".

✓ Розшифруйте текст "PAYMENT" з ключем "CASH".

19. Зашифруйте текст "DIGITALMARKETING" з ключем "MEDIA".

✓ Розшифруйте текст "BRANDING" з ключем "PROMOTE".

20. Зашифруйте текст "USEREXPERIENCE" з ключем "UXUI".

✓ Розшифруйте текст "FEEDBACK" з ключем "SURVEY".

21. Зашифруйте текст "MOBILEAPPLICATION" з ключем "ANDROID".

✓ Розшифруйте текст "DEVELOPER" з ключем "PROGRAMMER".

22. Зашифруйте текст "MACHINELEARNING" з ключем "AI".

✓ Розшифруйте текст "MODEL" з ключем "TRAIN".

23. Зашифруйте текст "QUANTUMCOMPUTING" з ключем "PHYSICS".

✓ Розшифруйте текст "EXPERIMENT" з ключем "LAB".

24. Зашифруйте текст "DATAENGINEERING" з ключем "TECH".

✓ Розшифруйте текст "PROCESSING" з ключем "ANALYZE".

25. Зашифруйте текст "CROSSPLATFORM" з ключем "DEVICE".

✓ Розшифруйте текст "APPLICATIONS" з ключем "TOOLS".

26. Зашифруйте текст "GAMEDEVELOPMENT" з ключем "PLAY".

✓ Розшифруйте текст "LEVELUP" з ключем "SKILL"

Контрольні запитання

1. Що таке шифрування? Опишіть основну мету шифрування даних.
2. Які основні типи шифрування існують? Назвіть і коротко опишіть симетричне та асиметричне шифрування.
3. Що таке шифр Віженера? Опишіть принцип роботи шифру Віженера.
4. Як обчислюється ключ у шифрі Віженера?
5. Які дії потрібно виконати для розширення ключа до довжини відкритого тексту?
6. Як виконується шифрування в шифрі Віженера? Опишіть процес шифрування, використовуючи позиції букв в алфавіті.
7. Що таке поліальтернативне шифрування? Які переваги і недоліки поліальтернативного шифрування порівняно з простим?
8. Яка роль ключа в процесі шифрування? Чому важливо зберігати ключ у секреті?
9. Які методи дешифрування існують для шифру Віженера? Опишіть процес дешифрування, використовуючи ключ.
10. Які проблеми можуть виникнути при використанні шифру Віженера? Які вразливості має цей метод шифрування?
11. Що таке частотний аналіз? Як частотний аналіз може використовуватися для злому шифрів?
12. Які способи забезпечення безпеки ключів? Назвіть декілька методів захисту ключів шифрування.
13. Які алгоритми симетричного шифрування ви знаєте? Назвіть і коротко опишіть кілька популярних алгоритмів симетричного шифрування.

14. Які методи аутентифікації даних існують? Опишіть роль аутентифікації у забезпеченні безпеки даних.

15. Які сучасні застосування шифрування? Назвіть приклади, де шифрування використовується в повсякденному житті.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

Основна

1. Богданов В. Основи алгоритмізації та програмування : посібник. Київ : , 2010. 136 с.
2. Клакович Л.М., Левицька С.М., Костів О. М. Теорія алгоритмів. Львів, : Вид-во Львів ун-ту, 2008. 154 с.
3. Лісовик Л.П., Шкільняк С.С. Основи теорії алгоритмів. Київ : Інтелектуальні системи, 2013. 194 с
4. Прийма С.М. Теорія алгоритмів : навчальний посібник. Мелітополь : ФОП Однорог Т.В., 2018. 116 с.
5. Саволюк А. П. Основи алгоритмізації та програмування : збірник завдань. Київ : «Основа», 2011. 208 с.
6. Техніка обчислень і алгоритмізація / І. Ф. Следзінський, А.М. Ломакович, Ю.С. Рамський та ін. Київ : Вища шк., 2001. 199 с.
7. Яворський Б.І. Теорія алгоритмів : конспект лекцій. Тернопіль : ТДТУ імені Івана Пулюя, 2015. 132 с.

Додаткова

1. Aho A. Construction and analysis of computational algorithms / A. Aho, D. Hopcroft, D. Ulman. M. : Mir, 1979. 536 p.
2. Virt N. Algorithms and data structures / N. Virt; Pers with Eng.. M. : Mir, 1989. 360 p.
3. Knut D.E. Sortirovka i poisk : учебн. help / D E. Whip; trans. with English 2nd ed. M. : Publishing House "Williams", 2000. 832 p.
4. Техніка обчислень і алгоритмізація / І.Ф. Следзінський, А.М. Ломакович, Ю.С. Рамський та ін. Київ : Вища шк., 2001. 297 с.

Електронне навчально-методичне видання

Ольга ГАРБИЧ-МОШОРА

Методичні вказівки для виконання лабораторних робіт з курсу “ТЕОРІЯ АЛГОРИТМІВ”

122 "Комп'ютерні науки"

Дрогобицький державний педагогічний університет
імені Івана Франка

Редактор
Ірина Невмержицька
Технічний редактор
ІринаАртимко

Здано до набору 12.08.2024 р. Формат 60х90/16. Гарнітура Times. Ум. друк. арк. 4.65 Зам. 104.

Дрогобицький державний педагогічний університет імені Івана Франка. (Свідоцтво про внесення суб’єкта видавничої справи до державного реєстру видавців, виготівників та розповсюджувачів видавничої продукції ДК № 5140 від 01.07.2016 р.). 82100, Дрогобич, вул. Івана Франка, 24, к. 203