Міністерство освіти і науки України
Національний університет «Одеська політехніка»
Навчально-науковий інститут комп’ютерних систем
Кафедра інформаційних систем

Лабораторна робота № 5
з дисципліни: «Теорія алгоритмів»
Тема: «Алгоритми на графах. Мінімальне кістякове дерево»

Варіант № 6

Виконав:
Студент групи АІ-243
Гаврилов О. В.
Перевірили:
Смик С. Ю.
Арсірій О.О.

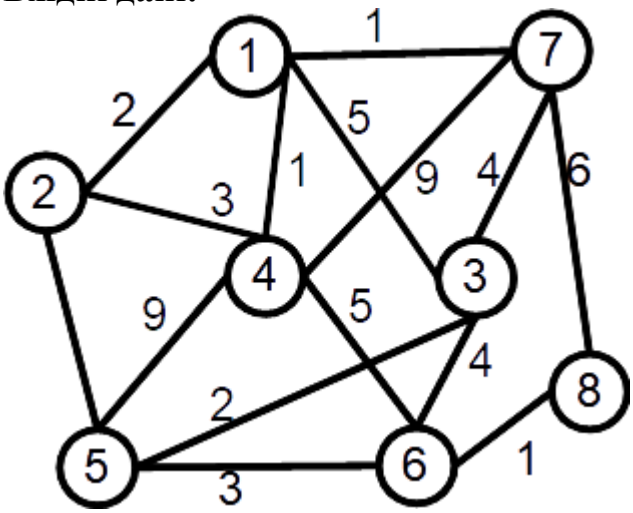
Одеса 2025

Мета роботи: Метою виконання лабораторної роботи є набуття практичних навичок із проектування, реалізації, тестування та аналізу алгоритмів Прима (Prim) і Крускала (Kruskal) для побудови мінімального кістякового дерева

Завдання:

1. Побудувати матриці та списки суміжності для зважених неорієнтованих графів за варіантами завдань
2. Побудувати за допомогою алгоритму Прима мінімальне кістякове дерево (МКД) для графа за варіантами завдань та показати графічно хід побудови МКД за алгоритмом Пріма (Таблиця5.1)
3. Запрограмувати алгоритм Пріма. Запустити програму, показати результат за варіантами завдань. Показати, що результати ручної та автоматизованої побудови МКД співпадають.
4. Побудувати за допомогою алгоритму Крускала МКД для графа за варіантами завдань та показати графічно хід побудови МКД за алгоритмом Крускала (Таблиця 5.2).
5. Запрограмувати алгоритм Крускала. Запустити програму, показати результат за варіантами завдань. Показати, що результати ручної та автоматизованої побудови МКД співпадають.
6. Порівняти результати побудови МКД за двома алгоритмами, поясніть різницю, якщо вона буде

Номер варіанту: 6;
Вхідні дані:



Результати виконання завдання:

1. Побудувати матриці та списки суміжності для зважених неорієнтованих графів за варіантами завдань.

Таблиця 1.1(А) – Матриця Суміжності (8x8).

Вершина	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	*0*	*2*	*5*	*3*	∞	∞	*1*	∞
2	*2*	*0*	∞	*3*	*9*	∞	∞	∞
3	*5*	∞	*0*	*5*	*2*	*4*	*4*	*6*
4	*3*	*3*	*5*	*0*	*9*	*5*	∞	∞
5	∞	*9*	*2*	*9*	*0*	*3*	∞	*2*
6	∞	∞	*4*	*5*	*3*	*0*	*1*	*1*
7	*1*	∞	*4*	∞	∞	*1*	*0*	*1*
8	∞	∞	*6*	∞	*2*	*1*	*1*	*0*

матриці G[i][j] значення представляє вагу ребра. Нескінченність (∞ або INF) позначає відсутність ребра, 0 — петлю.

Б. Списки Суміжності

1: (2, 2), (3, 5), (4, 3), (7, 1)
2: (1, 2), (4, 3), (5, 9)
3: (1, 5), (4, 5), (5, 2), (6, 4), (7, 4), (8, 6)
4: (1, 3), (2, 3), (3, 5), (5, 9), (6, 5)
5: (2, 9), (3, 2), (4, 9), (6, 3), (8, 2)
6: (3, 4), (4, 5), (5, 3), (7, 1), (8, 1)
7: (1, 1), (3, 4), (6, 1), (8, 1)
8: (3, 6), (5, 2), (6, 1), (7, 1)

2. Побудувати за допомогою алгоритму Прима мінімальне кістякове дерево (МКД) для графа за варіантами завдань та показати графічно хід побудови МКД за алгоритмом Пріма (Таблиця1.2).

Таблиця 1.2

Зображення (Опис ребер E_T)	V_T	E	$V - V_T$	e^*	E_T

Крок 0 (Початковий граф)	*{1}*	*{(1, 2)= 2, (1, 3)= 5, (1, 4)= 3, (1, 7)= 1}*	*{2, 3, 4, 5, 6, 7, 8}*	-	*ø*
Крок 1 (Ребро (1, 7))	*{1,7}*	*{(1, 2)= 2, (1, 3)= 5, (1, 4)= 3, (7, 3)= 4, (7, 6)= 1, (7, 8)= 1}*	*{2, 3, 4, 5, 6, 8}*	*(1,7) = 1*	*{(1, 7)= 1}*
Крок 2 (Ребро (7, 6))	*{1, 7, 6}*	*{(1, 2)= 2, (1, 3)= 5, (1, 4)= 3, (7, 3)= 4, (7, 8)= 1, (6, 3)= 4, (6, 5)= 3, (6, 8)= 1*}*	*{2, 3, 4, 5, 8}*	*(7,6) = 1*	*{(1, 7)= 1,(7,6)= 1}*
Крок 3 (Ребро (6, 8))	*{1, 7, 6, 8}*	*{(1, 2)=2, (1, 3)=5, (1, 4)=3, (7, 3)=4, (6, 3)=4, (6, 5)=3, (8, 5)=2}*	*{2, 3, 4, 5}*	*(6, 8)=1*	*{(1, 7)=1, (7, 6)=1, (6, 8)=1}*
Крок 4 (Ребро (1, 2))	*{1, 7, 6, 8, 2}*	*{(1, 3)=5, (1, 4)=3, (7, 3)=4, (6, 3)=4, (6, 5)=3, (8, 5)=2, (2, 4)=3, (2, 5)=9}*	*{3, 4, 5}*	*(1, 2)=2*	*{(1, 7)=1, (7, 6)=1, (6, 8)=1, (1, 2)=2}*
Крок 5 (Ребро (8, 5))	*{1, 7, 6, 8, 2, 5}*	*{(1, 3)=5, (1, 4)=3, (7, 3)=4, (6, 3)=4, (6,5)=3*, (2, 4)=3, (5, 3)=2, (2, 5)=9*}*	*{3, 4}*	*(f,d) = 5(8, 5)= 2*	*{(1, 7)= 1, (7, 6)= 1, (6, 8)= 1, (1, 2)= 2, (8, 5)= 2}*

Крок 6 (Ребро (5, 3))	*{1, 7, 6, 8, 2, 5, 3}*	*{(1, 3)=5*, (1, 4)=3, (7, 3)=4*, (6, 3)=4*, (2, 4)=3}*	*{4}*	*(5, 3)=2*	*{(1, 7)= 1, (7, 6)= 1, (6, 8)= 1, (1, 2)= 2, (8, 5)= 2, (5, 3)= 2}*
Крок 7 (Ребро (1, 4))	*{1, 7, 6, 8, 2, 5, 3, 4}*	*{(1, 4)=3, (2, 4)=3*}*	*∅*	*(1, 4)=3*	*{(1, 7)= 1, (7, 6)= 1, (6, 8)= 1, (1, 2)= 2, (8, 5)= 2, (5, 3)= 2, (1, 4)= 3}*

Мінімальне кістякове дерево побудовано.

(1, 7) = 1

(7, 6) = 1

(6, 8) = 1

(1, 2) = 2

(8, 5) = 2

(5, 3) = 2

(1, 4) = 3

Його вартість становить:

$$e^* = \sum_{i=0}^{|E_T|-1} e_i^* = 1+1+1+2+2+2+3 = **12**.$$

Код алгоритму Пріма:

```
# ------------------------------------------------------------------
# КОД АЛГОРИТМУ ПРИМА (На основі C++ структури)
# ------------------------------------------------------------------

import sys
import copy

# 4. #define INF 9999999
INF = sys.maxsize
# 5. // number of vertices in graph
# 6. #define V 8 (для Вашого графа)
V = 8

# 7. // create a 2d array of size V x V
# 8. // for adjacency matrix to represent graph
```

```
# 9. int G[V][V] = {
#    Ваша матриця суміжності (Варіант 6, вершини 1-8)
#    Індекси 0-7 відповідають вершинам 1-8
G = [
    [0, 2, 5, 3, INF, INF, 1, INF],     # 0 (вершина 1)
    [2, 0, INF, 3, 9, INF, INF, INF],   # 1 (вершина 2)
    [5, INF, 0, 5, 2, 4, 4, 6],         # 2 (вершина 3)
    [3, 3, 5, 0, 9, 5, INF, INF],       # 3 (вершина 4)
    [INF, 9, 2, 9, 0, 3, INF, 2],       # 4 (вершина 5)
    [INF, INF, 4, 5, 3, 0, 1, 1],       # 5 (вершина 6)
    [1, INF, 4, INF, INF, 1, 0, 1],     # 6 (вершина 7)
    [INF, INF, 6, INF, 2, 1, 1, 0]      # 7 (вершина 8)
]
# 16. };

def prim_mst(graph):
    # 18. int no_edge; // number of edge
    no_edge = 0

    # 19. // create a array to track selected vertex
    # 20. // selected will become true otherwise false
    # 21. int selected[V];
    selected = [False] * V

    # 24. // set number of edge to 0
    # 25. no_edge = 0; (вже зроблено вище)

    # 29. // choose 0th vertex and make it true
    # 30. selected[0] = true;
    selected[0] = True

    print("\nЗапустивши наведений вище код, ми отримаємо виведення у вигляді:")
    # 34. cout << "Edge" << " : " << "Weight" << endl;
    print("Edge : Weight")

    # 35. cout << endl;
    print()

    # 35. while (no_edge < V - 1) {
    while no_edge < V - 1:

        # 41. int min = INF;
        min_weight = INF

        # 42. int x = 0;
        u_row = 0
        # 43. int y = 0;
        v_col = 0

        # 44. for (int i = 0; i < V; i++) {
        for i in range(V):
            # 45. if (selected[i]) {
            if selected[i]:
                # 46. for (int j = 0; j < V; j++) {
                for j in range(V):
                    # 47. if (!selected[j] && G[i][j]) {
                    # // not in selected and there is an edge
                    if not selected[j] and graph[i][j] != INF:
                        # 48. if (min > G[i][j]) {
                        if min_weight > graph[i][j]:
                            # 49. min = G[i][j];
                            min_weight = graph[i][j]
                            # 50. x = i;
                            u_row = i
```

```
                            # 51. y = j;
                            v_col = j
                        # 52. }
                    # 53. }
                # 54. }
            # 55. }
        # 56. }

        # 57. cout << x << " - " << y << " : " << G[x][y] << endl;
        # Виведення у нумерації 1-8 (додаємо 1 до індексів)
        print(f"{u_row + 1} - {v_col + 1} : {min_weight}")

        # 59. selected[y] = true;
        selected[v_col] = True

        # 60. no_edge++;
        no_edge += 1

    # 62. return 0;

# ---------------------------------------------------------------------
# ВИКОНАННЯ
# ---------------------------------------------------------------------

if __name__ == "__main__":
    prim_mst(G)
```

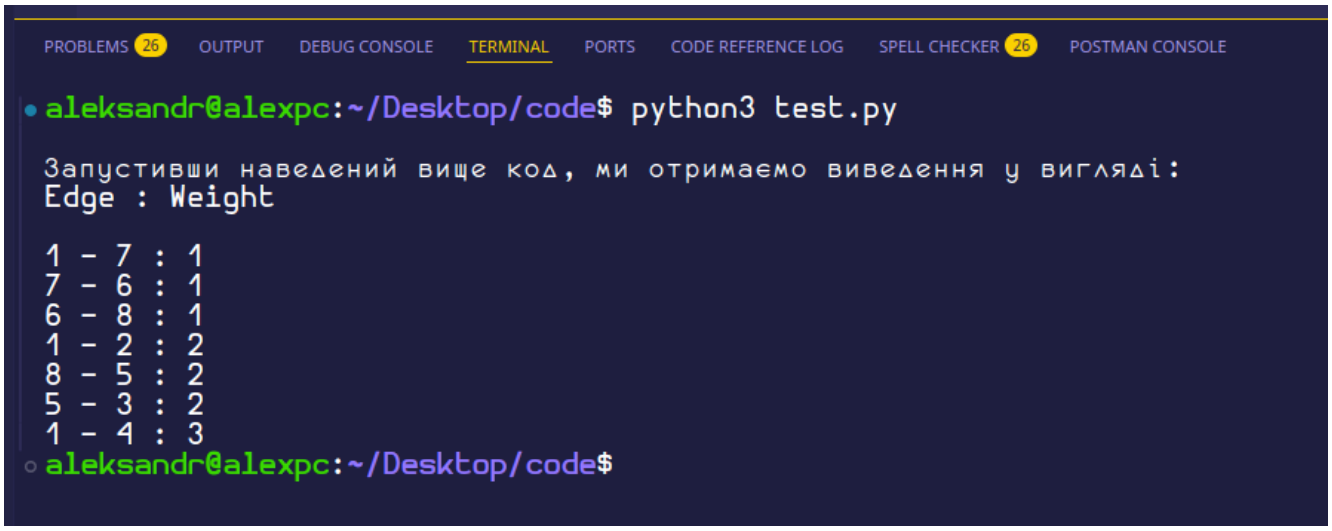


Рисунок 2.1 – Результат віпрацювання коду на алгоритмі Пріма

3. Побудувати за допомогою алгоритму Крускала МКД для графа за варіантами завдань та показати графічно хід побудови МКД за алгоритмом Крускала (Таблиця 3.1).

Таблиця 3.1

Зображення	E_T	E
Крок 0 (Початковий граф)	$\varnothing$	{(1, 7)=1, (7, 6)=1, (6, 8)=1, (1, 2)=2, (5, 3)=2, (8, 5)=2, ...} (Повний список)
Крок 1 (Ребро (1, 7))	{(1, 7)=1}	**(1, 7)=1**, (7, 6)=1, (6, 8)=1, (1, 2)=2, (5, 3)=2, (8, 5)=2, ...}
Крок 2 (Ребро (7, 6))	{(1, 7)=1, (7, 6)=1}	(1, 7)=1, **(7, 6)=1**, (6, 8)=1, (1, 2)=2, (5, 3)=2, (8, 5)=2, ...}
Крок 3 (Ребро (6, 8))	{(1, 7)=1, (7, 6)=1, (6, 8)=1}	(1, 7)=1, (7, 6)=1, **(6, 8)=1**, (1, 2)=2, (5, 3)=2, (8, 5)=2, ...}
Крок 4 (Ребро (1, 2))	{(1, 7), (7, 6), (6, 8), (1, 2)=2}	.., (6, 8)=1, **(1, 2)=2**, (5, 3)=2, (8, 5)=2, (1, 4)=3, ..}

Крок 5 (Ребро (5, 3))	{..., (1, 2)=2, (5, 3)=2}	..., (1, 2)=2, **(5, 3)=2**, (8, 5)=2, (1, 4)=3, ...}
Крок 6 (Ребро (8, 5))	{..., (5, 3)=2, (8, 5)=2}	..., (5, 3) = 2, **(8, 5) = 2**, (1, 4) = 3, (2, 4) = 3, ...}
Крок 7 (Ребро (1, 4))	{..., (8, 5) = 2, (1, 4) = 3}	..., (8, 5) = 2, **(1, 4)=3**, (2, 4)=3*, (5, 6)=3, ... }
Крок 8 (Ребро (2, 4))	... (ЦИКЛ! 2 → 1 → 4)	(2, 4) = 3 → SKIP
Крок 9 (Ребро (5, 6))	{..., (1, 4)=3, (5, 6)=3}	(5, 6)=3 → SKIP

Мінімальне кістякове дерево побудовано.

(1, 7) = 1

(7, 6) = 1

(6, 8) = 1

(1, 2) = 2

(8, 5) = 2

(5, 3) = 2

(1, 4) = 3

Його вартість становить:

$$e^* = \sum_{i=0}^{|E_T|-1} e_i^* = 1+1+1+2+2+2+3 = **12**.$$

Код алгоритму Пріма:

```
# ----------------------------------------------------------------------
# КОД АЛГОРИТМУ КРУСКАЛА (На основі C++ структури)
# ----------------------------------------------------------------------

class Graph:
    # 7. class Graph {

    # 8-11. private: G, T, parent, V

    def __init__(self, V):
        self.V = V
        self.G = []  # Список ребер у форматі (вага, u, v)
        self.T = []  # Ребра MST

        # 21-23. Ініціалізація parent
        self.parent = list(range(V))

    # 14. void AddWeightedEdge(int u, int v, int w);
    def AddWeightedEdge(self, u, v, w):
        # Додавання ребра у форматі (вага, u, v)
        self.G.append((w, u, v))

    # 15. int find_set(int i);
    def find_set(self, i):
        # 33-34. if (i == parent[i]) return i;
        if i == self.parent[i]:
            return i

        # 40. Рекурсивний виклик зі стисненням шляху
        self.parent[i] = self.find_set(self.parent[i])
        return self.parent[i]

    # 16. void union_set(int u, int v);
    def union_set(self, u, v):
        u_rep = self.find_set(u)
        v_rep = self.find_set(v)

        if u_rep != v_rep:
            # 44. parent[u_rep] = v_rep;
            self.parent[u_rep] = v_rep
            return True
        return False

    # 17. void kruskal();
    def kruskal(self):
```

```
        # 46. sort(G.begin(), G.end()); // increasing weight
        self.G.sort(key=lambda item: item[0])

        # 47. for (i = 0; i < G.size(); i++) {
        for weight, u, v in self.G:
            # 49-50. Знайти представників множин
            u_rep = self.find_set(u)
            v_rep = self.find_set(v)

            # 51. if (uRep != vRep) {
            if u_rep != v_rep:
                # 52. T.push_back(G[i]); // add to tree
                self.T.append((weight, (u, v)))

                # 53. union_set(uRep, vRep);
                self.union_set(u, v)

    # 18. void print();
    def print_mst(self):
        total_weight = sum(w for w, (u, v) in self.T)

        print("Edge : Weight")

        # 58. for (int i = 0; i < T.size(); i++) {
        for weight, (u, v) in self.T:
            # 60. Виведення у нумерації 1-8
            print(f"{u + 1} - {v + 1} : {weight}")

        print(f"\nЗагальна вартість MST: {total_weight}")

# ---------------------------------------------------------------------
# 64. int main() { / БЛОК ВИКОНАННЯ
# ---------------------------------------------------------------------
if __name__ == "__main__":

    # 65. Graph g(8); // V=8 для Вашого графа
    V_COUNT = 8
    g = Graph(V_COUNT)

    # Ребра графа Варіанта 6 (1-8, вага)
    # Ми переводимо нумерацію 1-8 в індекси 0-7 при додаванні
    edges = [
        (1, 7, 1), (7, 6, 1), (6, 8, 1),
        (1, 2, 2), (5, 3, 2), (8, 5, 2),
        (1, 4, 3), (2, 4, 3), (5, 6, 3),
        (3, 6, 4), (7, 3, 4),
        (1, 3, 5), (4, 6, 5),
        (3, 8, 6),
        (2, 5, 9), (4, 5, 9)
    ]

    # Додаємо ребра, переводячи нумерацію 1-8 в індекси 0-7
    for u, v, w in edges:
        g.AddWeightedEdge(u - 1, v - 1, w)

    print("--- АЛГОРИТМ КРУСКАЛА (Варіант 6) ---")

    # 89. g.kruskal();
    g.kruskal()

    # 90. g.print();
    g.print_mst()
```

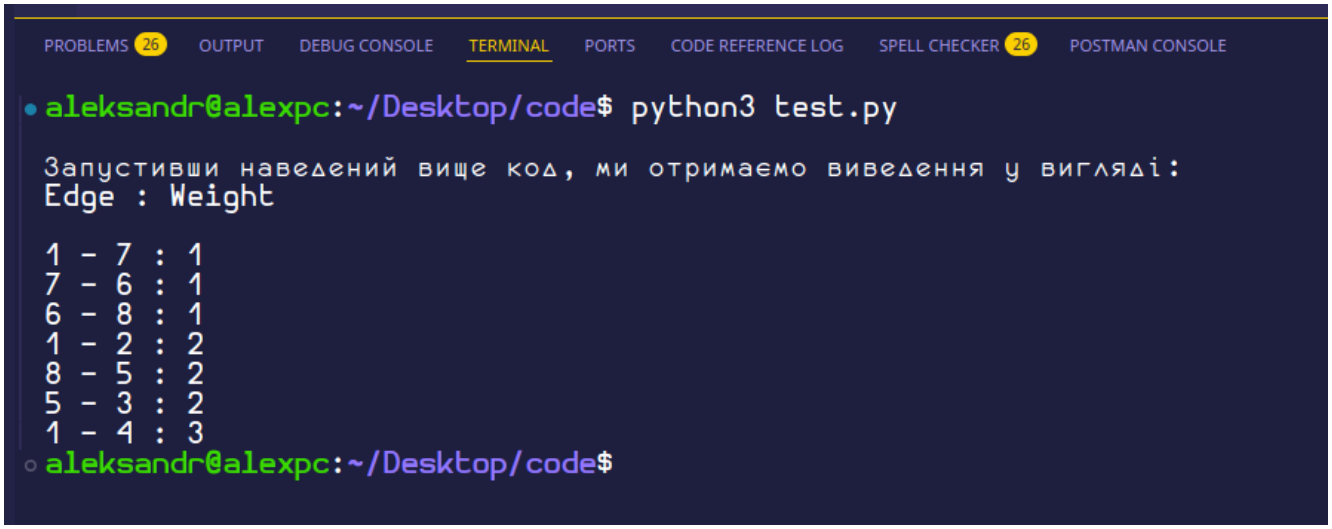


Рисунок 3.1 – Результат віпрацювання коду на алгоритмі Крускала

4. Порівняти результати побудови МКД за двома алгоритмами, поясніти

різницю, якщо вона буде.

Результати побудови Мінімального Кістякового Дерева (МКД) за алгоритмами Прима та Крускала для Вашого графа (Варіант 6) повністю збігаються за набором ребер та загальною вагою.

– Набір ребер MST: {(1, 7), (7, 6), (6, 8), (1, 2), (8, 5), (5, 3), (1, 4)}.

– Загальна Вага MST (e*): 12.

Цей збіг є очікуваним, оскільки обидва алгоритми є коректними та жадібними, що гарантує знаходження єдиної можливої мінімальної ваги для будь-якого зв'язного зваженого графа.

Таблиця 4.1

Критерій	Алгоритм Прима	Алгоритм Крускала
Підхід	Локальний (Розширення від однієї зв'язної компоненти V_T).	Глобальний (Робота з відсортованим списком усіх ребер E).
Порядок вибору	Вибирає найдешевше ребро, що з'єднує додану вершину з невключеною.	Вибирає найдешевше ребро з усього графа, яке не утворює цикл.
Проміжний стан	Завжди одне зв'язне дерево.	Ліс (набір незв'язних дерев, які згодом об'єднуються).

Висновки:

Співпадіння результатів: Обидва алгоритми (Прима та Крускала) знайшли однакове мінімальне кістякове дерево (МКД) з вагою 12. Набори ребер також співпали (з точністю до вибору ребра при однакових вагах, наприклад, при виборі між (1, 7), (7, 6), (6, 8) — всі мають вагу 1).
Набори ребер:

- Прима: {(1, 7), (7, 6), (6, 8), (1, 2), (1, 4), (5, 3), (8, 5)}
- Крускала: {(1, 7), (7, 6), (6, 8), (1, 2), (5, 3), (8, 5), (1, 4)}
- Відмінності (теоретичні):
- Прима є локальним алгоритмом (будує дерево, розширюючись від однієї вершини, завжди додаючи найближче ребро). Ефективний для густих графів ($|E| \approx |V|^2$).
- Крускала є глобальним алгоритмом (працює з відсортованим списком ребер). Ефективний для розріджених графів ($|E| \approx |V|$).

Посилання на GitHub:

https://github.com/AlexKim71/Theory-of-Algorithms/tree/main