МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ”ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Кафедра «Систем Автоматизованого Проектування»

Звіт
Лабораторна робота № 1
з курсу *“* Дискретні моделі*”*
на тему: *«* АЛГОРИТМ ПОБУДОВИ ДЕРЕВ»

Виконав:
Ст.гр. КН-409
Єлечко Олег
Прийняв:
Кривий Р.З.

Львів 2023

Мета роботи: Вивчення алгоритмів рішення задач побудови остових дерев. Побудова максимального і мінімального покриваючого дерева.

Теоретичні відомості:

Графом G називають скінчену множину V з нерефлексивним симетричним відношенням R на V. Визначим E як множину симетричних пар в R. Кожний елемент V називають вершиною. Кожний елемент E називають ребром, а E множиною ребер G.
Граф називається зв’язним, якщо в ньому для будь-якої пари вершин знайдеться ланцюг, який їх з’єднує, тобто, якщо по ребрах (дугах) можна попасти з будь-якої вершини в іншу.
Цикл - це ланцюг, в якого початкова і кінцева точки співпадають.
Дерево - це зв’язний граф без циклів.
Покриваючим деревом графа називається любе дерево, що утворене сукупністю його ребер(дуг), які включають всі вершини графа.
Лісом називається будь-яка сукупність дуг (ребер) інцидентних до вершин, які не містять циклів. Таким чином, ліс складається з одного або більше дерев.
Остовним деревом графа називається будь-яке дерево, яке утворене сукупністю дуг, які включають всі вершини графа.
Будь-який зв’язний граф має остовне дерево. Коренем орієнтованого дерева (прадерева) називається його вершина, в яку не входить жодна з дуг. Орієнтований ліс визначається як звичайний, тільки складається не з простих дерев, а орієнтованих.
Остовним орієнтованим деревом називається орієнтоване дерево, яке одночасно є і остовим деревом. Остовним орієнтованим лісом називається орієнтований ліс, який включає всі вершини відповідного графа.
Вага дерева - це сума ваг його ребер. Поставимо у відповідність кожній дузі (x,y) графа G вагу a(x,y). Вага орієнтованого лісу (або орієнтованого дерева) визначається як сума ваг дуг, що входять в даний ліс (дерево).
Максимальним орієнтованим лісом графа G називається орієнтований ліс графа G з максимально можливою вагою. Максимальним орієнтованим деревом графа G називається орієнтоване дерево графа G з максимально можливою вагою.
Мінімальні орієнтовані ліс і дерево визначаються аналогічним чином.
Кущ(букет) - зв’язний фрагмент графа

ПОШУК ОСТОВОГО ДЕРЕВА.

Алгоритм Борувки.

Це алгоритм знаходження мінімального остового дерева в графі. Вперше був опублікований в
1926 році Отакаром Борувкой, як метод знаходження оптимальної електричної мережі в Моравії. Робота алгоритму складається з декількох ітерацій, кожна з яких полягає в послідовному додаванні ребер до остового лісу графа, до тих пір, поки ліс не перетвориться на дерево, тобто, ліс, що складається з однієї компоненти зв'язності.

Алгоритм Крускала.

Найбільш відомий алгоритм Крускала (Joseph Kruskal) був придуманий автором у 1956 році. Основна стратегія цього алгоритму така: ребра упорядковуються за вагою; на кожному кроці до споруджуваного остового дерева додається найлегше ребро, яке з'єднує вершини з різних компонент. Таким чином на кожному кроці побудована множина складається з однієї або більше нетривіальних компонент, кожна з яких є підграфом деякого Національний університет «Львівська політехніка». Кафедра САП Інструкція до лабораторної роботи №1 з курсу
"Дискретні моделі в САПР" мінімального кістяка. Час роботи алгоритму Крускала становить O (ElogE) при використанні для зберігання компонент зв'язності системи непересічних множин з об'єднанням за рангом і стиском шляхів (найшвидший відомий метод). Більша частина часу йде на сортування ребер.

Прима.

Цей алгоритм названий на честь американського математика Роберта Прима (Robert Prim), який відкрив цей алгоритм у 1957 р. Втім, ще в 1930 р. цей алгоритм був відкритий чеським математиком Войтеком Ярніком (Vojtěch Jarník). Крім того, Едгар Дейкстра (Edsger Dijkstra) в 1959 р. також винайшов цей алгоритм, незалежно від них. Даний зважений неорієнтований граф з вершинами і ребрами. Потрібно знайти таке піддерево цього графа, яке б з'єднувало всі його вершини, і при цьому мало найбільшу можливу вагою (тобто сумою ваг ребер). Таке піддерево називається максимальним остовим деревом. У природному постановці ця задача звучить наступним чином: є міст, і для кожної пари відома вартість з'єднання їх дорогою (або відомо, що з'єднати їх не можна). Потрібно з'єднати всі міста так, щоб можна було доїхати з будь-якого міста в інший, а при цьому вартість прокладання доріг була б максимальною

ЛАБОРАТОРНЕ ЗАВДАННЯ

1. Отримати у викладача індивідуальне завдання.
2. Підготувати програму для вирішення виданого завдання.
3. Запустити на покрокове виконання програму побудови мінімального покриваючого дерева і максимального покриваючого дерева.
4. Зафіксувати результати роботи.
5. Оформити і захистити звіт.

   Github-link: https://github.com/OlehYelechko/Labs_DM/tree/main/Lab1

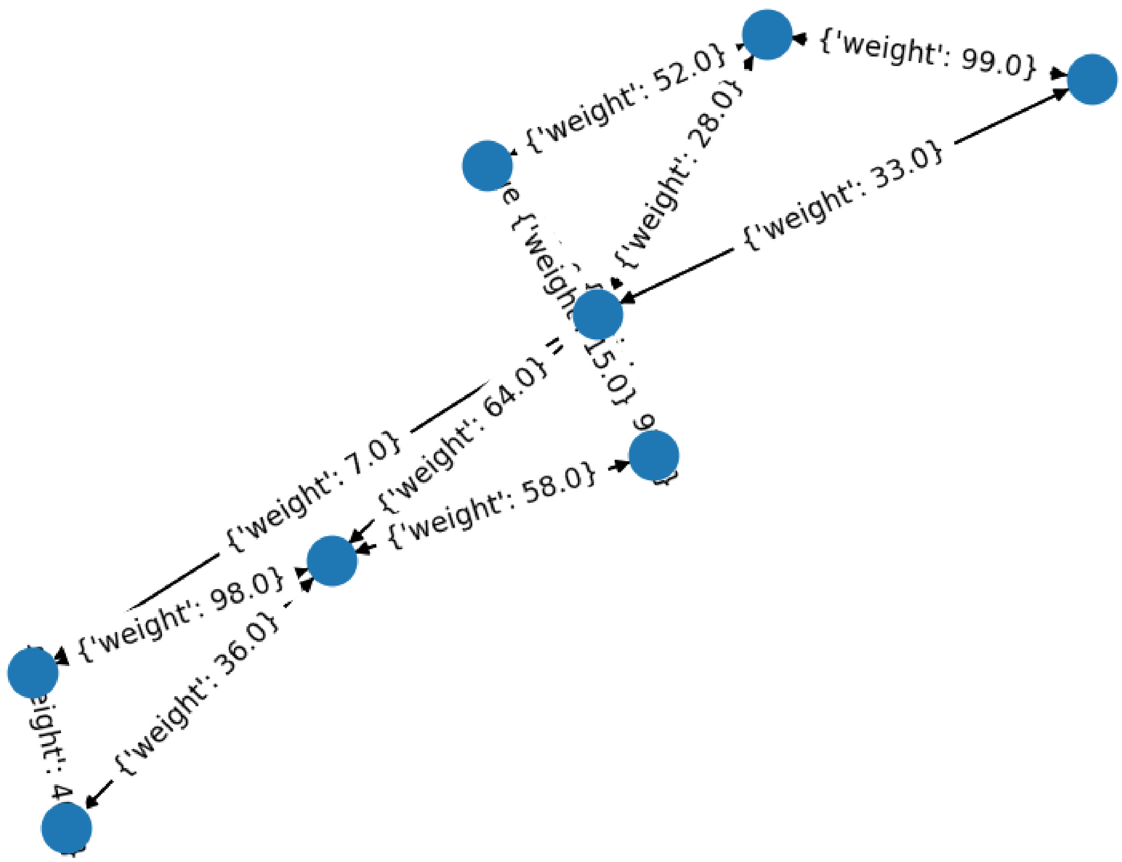
Код алгортиму Прима.

```python
# Імпортуємо необхідні бібліотеки
import sys
import matplotlib.pyplot as plt
import networkx as nx
import numpy as np
# Граф на основі матриці суміжності

path1 = "l1_1.txt"
path2 = "l1_2.txt"
path3 = "l1_3.txt"
```

```
with open(path1) as f:
    lines = (line for line in f if not line.startswith('#'))
    graph = np.loadtxt(lines, skiprows=1)

G = nx.from_numpy_matrix(np.matrix(graph), create_using=nx.DiGraph)
layout = nx.spring_layout(G)
nx.draw(G, layout)
nx.draw_networkx_edge_labels(G, pos=layout)
plt.show()



print(graph)
# Визначаємо кількість вершин у графі
V = len(graph)

class Graph:

    def __init__(self, graph, vertices):
        self.V = vertices
        self.graph = graph

    # Функція для знаходження вершини з мінімальною вагою ребра
    def minKey(self, key, mstSet):

        # Ініціалізуємо мінімальне значення
        min = sys.maxsize

        for v in range(self.V):
            if key[v] < min and mstSet[v] == False:
                min = key[v]
                min_index = v

        return min_index

    # Функція для знаходження вершини з максимального вагою ребра
    def maxKey(self, key, mstSet):

        # Ініціалізуємо мінімальне значення
        max = -sys.maxsize

        for v in range(self.V):
            if key[v] > max and not mstSet[v]:
                max = key[v]
                max_index = v
        return max_index

    # Функція для виведення мінімального кісткового дерева
```

```python
    def primMST(self, phonk=0):

        # Зберігаємо ключі та батьківські вершини
        if not phonk:
            key = [sys.maxsize] * self.V
        else:
            key = [-sys.maxsize] * self.V
        parent = [None] * self.V  # Зберігаємо конструкцію MST
        key[0] = 0  # Забезпечуємо першій вершині ключ зі значенням 0
        mstSet = [False] * self.V

        parent[0] = -1  # Перший вузол є кореневим в MST

        for cout in range(self.V):

            # Вибираємо вершину з мінімальним або максимальним ключем
            if phonk:
                u = self.maxKey(key, mstSet)
            else:
                u = self.minKey(key, mstSet)

            # Додаємо вершину до MST Set
            mstSet[u] = True
            if not phonk:
                # Оновлюємо значення ключів та батьківських вершин сусідніх вершин
                for v in range(self.V):

                    # graph[u][v] непорожнє, mstSet[v] дорівнює False та ключ v більший за вагу ребра (u,v)
                    if self.graph[u][v] > 0 and mstSet[v] == False and key[v] > self.graph[u][v]:
                        key[v] = self.graph[u][v]
                        parent[v] = u
            else:
                # Оновлюємо значення ключів та батьківських вершин сусідніх вершин
                for v in range(self.V):

                    # graph[u][v] непорожнє, mstSet[v] дорівнює False та ключ v більший за вагу ребра (u,v)
                    if self.graph[u][v] > 0 and not mstSet[v] and key[v] < self.graph[u][v]:
                        key[v] = self.graph[u][v]
                        parent[v] = u
        # Виводимо збудоване MST
        print("Edge \tWeight")
        for i in range(1, self.V):
            print(parent[i], "-", i, "\t", self.graph[i][parent[i]])

graph = Graph(graph, V)
print("Мінімальне:")
graph.primMST()
print("Максимальне:")
graph.primMST(phonk=1)
```

Результат виконання:



```
[[ 0.  0.  7.  0.  0.  0. 46. 98.]
 [ 0.  0. 33.  0.  0. 99.  0.  0.]
 [ 7. 33.  0. 99. 92. 28.  0. 64.]
 [ 0.  0. 99.  0. 15. 52.  0.  0.]
 [ 0.  0. 92. 15.  0.  0.  0. 58.]
 [ 0. 99. 28. 52.  0.  0.  0.  0.]
 [46.  0.  0.  0.  0.  0.  0. 36.]
 [98.  0. 64.  0. 58.  0. 36.  0.]]
Мінімальне:
Edge    Weight
2 - 1    33.0
0 - 2    7.0
5 - 3    52.0
3 - 4    15.0
2 - 5    28.0
0 - 6    46.0
6 - 7    36.0
Максимальне:
Edge    Weight
5 - 1    99.0
7 - 2    64.0
2 - 3    99.0
2 - 4    92.0
3 - 5    52.0
0 - 6    46.0
0 - 7    98.0
```

Висновок: на цій лабораторній роботі було реалізовано пошук максимального покриваючого дерева методом Прими.