Львівський національний університет імені Івана Франка
Факультет електроніки та комп’ютерних технологій

Звіт про виконання лабораторної роботи №3

Алгоритм Краскала

Виконав:
Студент групи ФЕП-22
Серафим Д.В.

Львів-2022

Хід роботи:

Частина 1.

Побудова каркасу мінімальної ваги за алгоритмом Краскала

1. Створити нову бібліотеку DSU (файли DSU.h, DSU.cpp) в якій реалізувати основні функції для роботи з системою DSU: set_make(v), set_find(v) та set_union(v1, v2), для операцій створення, пошуку та об'єднання, відповідно.

2. Створити новий проект Lab_2_1, до якого підключити бібліотеку DSU.

3. В цьому проекті описати структуру для задавання графа списком ребер. Структура повинна містити поля, що задають вершини ребра та його вагу. Задати масив таких структур для опису графа.

4. Реалізувати функцію sort(…) для сортування списку ребер графа за зростанням ваг (алгоритм сортування задає викладач) та додати її у проект.

5. Згідно описаної вище методики запрограмувати реалізацію алгоритму Краскала.

6. Отримати від викладача варіант завдання (графу) і виконати для нього побудову мінімального каркасу. Продемонструвати результат викладачеві.

```
from collections import defaultdict
import sys
from random import randint

decor = "=" + "-" * 20 + "="


class Graph:
    def __init__(self, vertices):
        self.vertices = vertices
        self.graph = []

    def set_make(self, u_inside, v_inside, w_inside):  # Добавити вершину
        self.graph.append([u_inside, v_inside, w_inside])

    def init_vertices(self):
        print(decor)
        how_start = input("Введіть яким чином хочете заповнити вершини:\n1 -
Вручну\n0 - Автоматично\n=")
        print(decor)
        if int(how_start) == 0:
            self.set_make(2, 4, 4)
            self.set_make(3, 2, 3)
            self.set_make(3, 4, 3)
            self.set_make(0, 2, 4)
            self.set_make(1, 2, 2)
            self.set_make(4, 3, 3)
            self.set_make(5, 2, 2)
            self.set_make(5, 4, 3)
            self.set_make(1, 0, 4)
            self.set_make(2, 0, 4)
            self.set_make(2, 1, 2)
```

```
            self.set_make(2, 3, 3)
            self.set_make(4, 2, 4)
            self.set_make(0, 1, 4)
            self.set_make(2, 5, 2)
        elif int(how_start) == 1:
            print("Якщо хочеш зупинити процес введи в консоль STOP")
            num = 0
            x_inside = ""
            y_inside = ""
            weight_inside = ""
            while str(x_inside) != "STOP" and str(y_inside) != "STOP" and
str(weight_inside) != "STOP":

                num += 1
                x_inside = input(f"x {num} = ")
                y_inside = input(f"y {num} = ")
                weight_inside = input(f"weight {num} = ")
                print(decor)

                if str(x_inside) != "STOP" and str(y_inside) != "STOP" and
str(weight_inside) != "STOP":

                    self.set_make(int(x_inside), int(y_inside),
int(weight_inside))

        else:
            print("Введено дані котрі не подялгають умові!")

    def set_find(self, parent_element, y_or_x_inside):  # Знайти
        if parent_element[y_or_x_inside] == y_or_x_inside:
            return y_or_x_inside
        return self.set_find(parent_element, parent_element[y_or_x_inside])

    def set_union(self, parent_element, rank, x_inside, y_inside):  # Обєднати
        y_root = self.set_find(parent_element, y_inside)
        x_root = self.set_find(parent_element, x_inside)
        if rank[x_root] < rank[y_root]:
            parent_element[x_root] = y_root
        elif rank[x_root] > rank[y_root]:
            parent_element[y_root] = x_root

        else:
            parent_element[y_root] = x_root
            rank[x_root] += 1

    def print_tree(self, parent_element):
        print("Edge \tWeight")
        for i in range(1, self.vertices):
            print(f"{parent_element[i]} - {i}
{self.graph[i][parent_element[i]]}")

    def kruskal_algo(self):  # Алгоритм Краскала (реалізація)
        result = []
        i = 0
        e = 0
        # print(self.graph)
        self.graph = sorted(self.graph, key=lambda item: item[2])
        # print(self.graph)
        parent_element = []

        for node in range(self.vertices):
            parent_element.append(node)
        # print(parent_element)
        while e < self.vertices - 1:
            u_inside, v_inside, w_inside = self.graph[i]
            i += 1
```

```
            x = self.set_find(parent_element, u_inside)
            y = self.set_find(parent_element, v_inside)

            if x != y:
                e = e + 1
                result.append([u_inside, v_inside, w_inside])

        min_cost = 0
        print("Ребра в побудованому каркасі")
        for u_inside, v_inside, weight in result:
            min_cost += weight
            print(f"{u_inside} -- {v_inside} == {weight}")
        print("Мінімальна сума ваг ребер", min_cost)


g = Graph(12)
g.init_vertices()
g.kruskal_algo()
```

```
=--------------------=
Введіть яким чином хочете заповнити вершини:
1 - Вручну
0 - Автоматично
=0
=--------------------=
Ребра в побудованому каркасі
1 -- 2 == 2
5 -- 2 == 2
2 -- 1 == 2
2 -- 5 == 2
3 -- 2 == 3
3 -- 4 == 3
4 -- 3 == 3
5 -- 4 == 3
2 -- 3 == 3
2 -- 4 == 4
0 -- 2 == 4
Мінімальна сума ваг ребер 31

Process finished with exit code 0
```

Частина 2. Побудова каркасу мінімальної ваги за алгоритмом Прима

1. Створити новий проект Lab_2_2, в якому реалізувати описаний вище алгоритм Прима:

1.1.Створити та ініціалізувати масиви min_e, sel_e та visited.

1.2.Для кожної незанесеної в каркас вершини реалізувати пошук

вершини, занесеної в каркас, з якою вона з’єднана ребром мінімальної ваги: оновлення масивів sel_e та min_e. Серед оновлених значень min_e знайти мінімальне значення і відповідне ребро (та вершину) занести у каркас. Продовжувати описану процедуру поки всі вершини графа не будуть занесені у каркас.

2. Отримати від викладача варіант завдання (графу) та початкової вершини і виконати для нього побудову мінімального каркасу методом

```
from random import randint

"""Decor"""
decor = "=" + "-" * 20 + "="

INF = 9999999  # кількість вершин у графі
array_size = 5  # створити двовимірний масив розміром 5x5
# for adjacency matrix to represent graph
matrix = []

print(decor)
how_start = input("""Введіть яким чином хочете заповнити двохвимірний
масив(Матрицю):
0 - Вручну\n1 - Автоматично\n2 - Автоматично з відомими даними\n=""")
print(decor)
if int(how_start) == 0:
    print("Заповни двохвимірний масив(Матрицю)")
    for i in range(array_size):
        buffer_array = []
        for j in range(array_size):
            buffer_array.append(int(input(f"Рядок{i + 1}: Елемент {j + 1}= ")))
        matrix.append(buffer_array)

elif int(how_start) == 1:
    for i in range(array_size):
        buffer_array = []
        for j in range(array_size):
            buffer_array.append(randint(0, 50))
        matrix.append(buffer_array)

elif int(how_start) == 2:
    matrix = [[0, 9, 75, 0, 0],
              [9, 0, 95, 19, 42],
              [75, 95, 0, 51, 66],
              [0, 19, 51, 0, 31],
              [0, 42, 66, 31, 0]]


else:
    print("Введено дані котрі не подялгають умові!")

# створити масив для відстеження вибраної вершини
# вибране стане істинним, інакше стане хибним
selected = [0, 0, 0, 0, 0]
# встановити кількість ребер на 0
no_edge = 0
# кількість egde у мінімальному охоплюючому дереві буде завжди менше ніж
(array_size - 1), де array_size - кількість вершин у графіку
# виберіть 0-ту вершину та зробіть її істинною
selected[0] = True
```

```
print("Вершини : Вага")
print(decor)
while (no_edge < array_size - 1):
    # Для кожної вершини множини S знайдіть усі суміжні вершини
    # обчисліть відстань від вершини, вибраної на кроці 1.
    # якщо вершина вже є в наборі S, інакше відкинути її
    # виберіть іншу вершину, найближчу до вибраної вершини на кроці 1.
    minimum = INF
    x = 0
    y = 0
    for i in range(array_size):
        if selected[i]:
            for j in range(array_size):
                if (not selected[j]) and matrix[i][j]:
                    # не в обраному і є край
                    if minimum > matrix[i][j]:
                        minimum = matrix[i][j]
                        x = i
                        y = j

    print(f"{x} ----- {y} : {matrix[x][y]}")
    selected[y] = True
    no_edge += 1
```

```
=--------------------=
Введіть яким чином хочете заповнити двохвимірний масив(Матрицю):
0 - Вручну
1 - Автоматично
2 - Автоматично з відомими даними
=1
=--------------------=
Вершини : Вага
=--------------------=
0 ----- 2 : 28
2 ----- 4 : 3
2 ----- 3 : 7
3 ----- 1 : 8

Process finished with exit code 0
```

```
=--------------------=
Введіть яким чином хочете заповнити двохвимірний масив(Матрицю):
0 - Вручну
1 - Автоматично
2 - Автоматично з відомими даними
=2
=--------------------=
Вершини : Вага
=--------------------=
0 ----- 1 : 9
1 ----- 3 : 19
3 ----- 4 : 31
3 ----- 2 : 51

Process finished with exit code 0
```

Висновок: На цій лабораторній роботі я навчився будувати каркас мінімальної ваги за алгоритмом Краскала і також будувати каркасу мінімальної ваги за алгоритмом Прима