Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"
Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

Звіт

з лабораторної роботи № 4 з дисципліни
«Алгоритми та структури даних 2. Структури даних»

«Прикладні задачі теорії графів ч.2»

Виконав(ла) *ІП-13 Лисенко Анастасія Олегівна* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив *Сопов Олексій Олександрович*
(прізвище, ім'я, по батькові)

Київ 2022

ЗМІСТ

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ 3

2 ЗАВДАННЯ 4

3 ВИКОНАННЯ 8

Псевдокод алгоритму 8

Програмна реалізація алгоритму 8

Вихідний код 8

ВИСНОВОК 10

КРИТЕРІЇ ОЦІНЮВАННЯ 11

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Мета роботи – вивчити додаткові прикладні алгоритми на графах та способи їх імплементації.

2 ЗАВДАННЯ

Згідно варіанту (таблиця 2.1), розробити та записати алгоритм задачі на графах за допомогою псевдокоду (чи іншого способу за вибором).

Виконати програмну реалізацію алгоритму на будь-якій мові програмування для довільного графа, передбачити введення розмірності графа та введення даних графа вручну чи випадковим чином.

Для самостійно обраного графа (розмірності не менше 7 вершин) розв’язати задану за варіантом задачу вручну.

Зробити узагальнений висновок з лабораторної роботи, у якому порівняти програмне та ручне розв’язання задачі.

Варіант 20

20	Задача про максимальний потік	Форда - Фалкерсона	Ортграф, матриця вагів

3 ВИКОНАННЯ

Псевдокод алгоритму

```
Функція __init__(self, graph):
   self.graph = graph
   self. ROW = len(graph)

функція searching_algo_BFS(self, s, t, parent):
        visited = [False] * (self.ROW)
        queue = []
        queue.append(s)
        visited[s] = True

        повторити поки queue:

        u = queue.pop(0)

        повторити для ind, val в enumerate(self.graph[u]):
            якщо visited[ind] == False and val > 0:
                queue.append(ind)
                visited[ind] = True
                parent[ind] = u
            все якщо
        все повторити

 повернути True якщо visited[t] або False


Функція ford_fulkerson(self, source, sink):
   parent  = [-1] * (self.ROW)
   max_flow = 0
   повторити поки self.searching_algo_BFS(source, sink, parent):
        path_flow = float("Inf")
        s = sink
        повторити поки(s != source):
            path_flow = min(path_flow, self.graph[parent[s]][s])
            s = parent[s]
        все повторити
        max_flow += path_flow
        v = sink
        повторити поки (v != source):
            u = parent[v]
            self.graph[u][v] -= path_flow
            self.graph[v][u] += path_flow
            v = parent[v]
        все повторити
   все повторити

 повернути max_flow
```

Програмна реалізація алгоритму

Вихідний код

```
from collections import defaultdict
import numpy as np
from random import *
class Graph(object):

    # Initialize the matrix
    def __init__(self, size):
        self.adjMatrix = []
        for i in range(size):
            self.adjMatrix.append([0 for i in range(size)])
        self.size = size


    def add_edge(self, v1, v2, weight):
        if v1 == v2:
            print("Same vertex %d and %d" % (v1, v2))
        self.adjMatrix[v1][v2] = weight


    def __len__(self):
        return self.size

    def print_matrix(self, n):
        for i in range(0, n):
            for j in range(0, n):
                if self.adjMatrix[i][j] == 0 and i!=j:
                    print('{:3}'.format('inf'), end = " ")
                else:
                    print('{:3}'.format(self.adjMatrix[i][j]), end=" ")
            print()



def input_amount():
    PROBABILITY = 50
    n = int(input("enter the amount of nodes in your graph:"))
    g = Graph(n)

    mode = input("choose the mode of filling up the matrix\n0 - for hand input\n1 -
for random input\nenter mode:")
    if mode == '0':
    print("enter input data in format: <1 node> <2 node> <weight>\nto end input enter
'<' in new line\n")
    for i in range(0, n*n):
        line = input()
        if (line[0]) == '<':
            break
        else:
            l = line.split()
            g.add_edge(int(l[0]), int(l[1]), int(l[2]))
    if mode == "1":
        limits_line = input("enter your limits range in one line <1 limit> <2
limit>:")
        limits = limits_line.split()
        for i in range(0, n):
            for j in range(i+1, n):
                if randrange(1, 100, 1) > PROBABILITY:
                    g.add_edge(i, j, randrange(int(limits[0]), int(limits[1]), 1))
                else:
```

```
                    g.add_edge(i, j, 0)
                if randrange(1, 100, 1) > PROBABILITY:
                    g.add_edge(j, i, randrange(int(limits[0]), int(limits[1]), 1))
                else:
                    g.add_edge(j, i, 0)
    print("matrix of weights:")
    g.print_matrix(n)

    algorithm = Algo(g.adjMatrix)
    source = 0
    sink = n-1

    print("Max Flow: %d " % algorithm.ford_fulkerson(source, sink))

class Algo:

    def __init__(self, graph):
        self.graph = graph
        self. ROW = len(graph)

    def searching_algo_BFS(self, s, t, parent):

        visited = [False] * (self.ROW)
        queue = []

        queue.append(s)
        visited[s] = True

        while queue:

            u = queue.pop(0)

            for ind, val in enumerate(self.graph[u]):
                if visited[ind] == False and val > 0:
                    queue.append(ind)
                    visited[ind] = True
                    parent[ind] = u

        return True if visited[t] else False


    def ford_fulkerson(self, source, sink):
        parent = [-1] * (self.ROW)
        max_flow = 0
        while self.searching_algo_BFS(source, sink, parent):

            path_flow = float("Inf")
            s = sink
            while(s != source):
                path_flow = min(path_flow, self.graph[parent[s]][s])
                s = parent[s]

            max_flow += path_flow

            v = sink
            while(v != source):
                u = parent[v]
                self.graph[u][v] -= path_flow
                self.graph[v][u] += path_flow
                v = parent[v]

        return max_flow
```

```
if __name__ == '__main__':
    input_amount()
```

Приклад роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми для графів на 7 і 15 вершин відповідно.

```
enter the amount of nodes in your graph:7
choose the mode of filling up the matrix
0 - for hand input
1 - for random input
enter mode:1
enter your limits range in one line <1 limit> <2 limit>:1 10
matrix of weights:
  0 inf inf   1 inf inf   4
  9   0   6 inf   6   2 inf
inf inf   0 inf inf   9 inf
inf inf inf   0   1   7 inf
inf inf inf inf   0 inf   1
inf   1 inf inf inf   0 inf
inf inf   7 inf   3 inf   0
Max Flow: 5
```

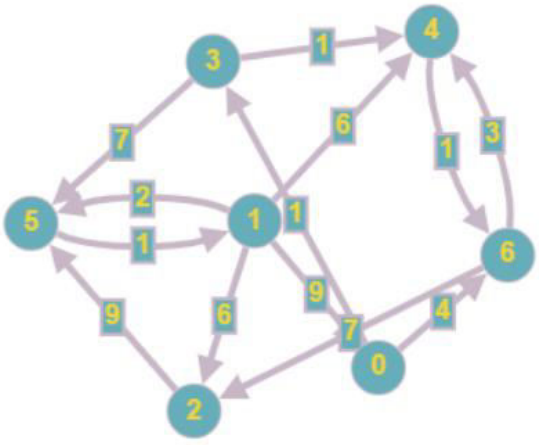


Рисунок 3.1 – Задача на 7 вершин

```
enter the amount of nodes in your graph:15
choose the mode of filling up the matrix
0 - for hand input
1 - for random input
enter mode:1
enter your limits range in one line <1 limit> <2 limit>:1 10
matrix of weights:
  0 inf inf inf inf   4 inf   4 inf inf inf   2   1 inf inf
inf   0 inf   7 inf inf   7   7 inf inf inf   4   7   2 inf
inf inf   0 inf inf   2   1 inf   2 inf inf inf   2 inf inf
inf   4 inf   0 inf inf inf   2 inf inf inf inf inf inf   3
  7 inf inf   1   0 inf inf inf inf inf inf inf inf inf inf
inf   6 inf inf   2   0   2   4   8 inf   3 inf inf inf inf
  2   9 inf inf   5 inf   0 inf   5 inf inf inf inf   2 inf
inf inf   5 inf inf inf inf   0 inf inf inf   3   7 inf inf
inf   1 inf inf inf   1 inf inf   0   3 inf inf inf inf inf
inf   1 inf   6 inf   9   7 inf inf   0 inf inf inf inf inf
inf   1 inf inf   9 inf inf inf inf inf   0 inf inf inf   8
inf inf   8   2 inf   1 inf inf inf   3   4   0 inf   7 inf
  3 inf inf inf inf inf inf inf inf inf inf inf   0 inf inf
inf inf   3 inf inf inf inf   4   6   9 inf inf inf   0 inf
inf inf   5   9 inf inf inf   9   8   9   3 inf inf   7   0
Max Flow: 10
```

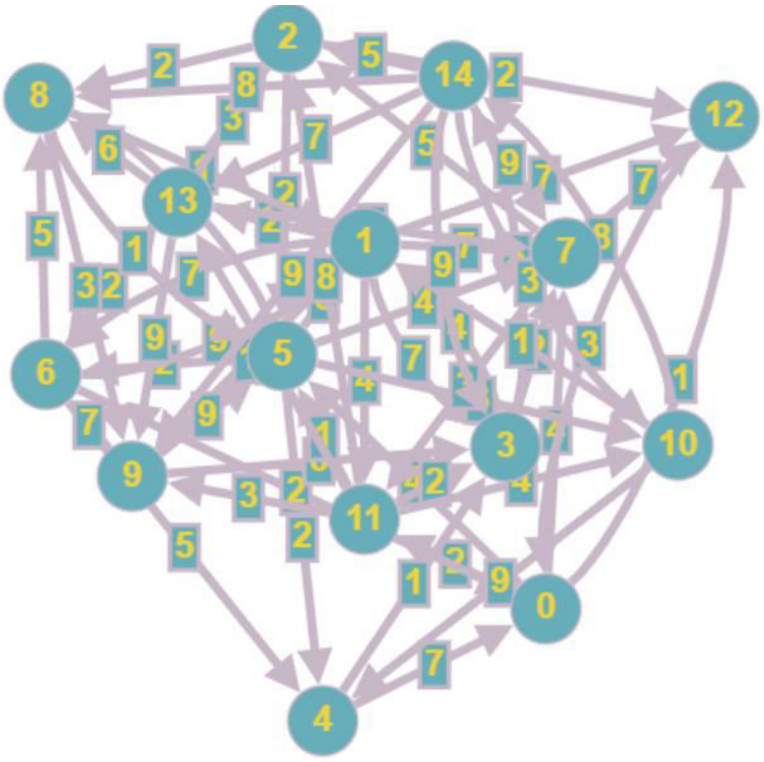
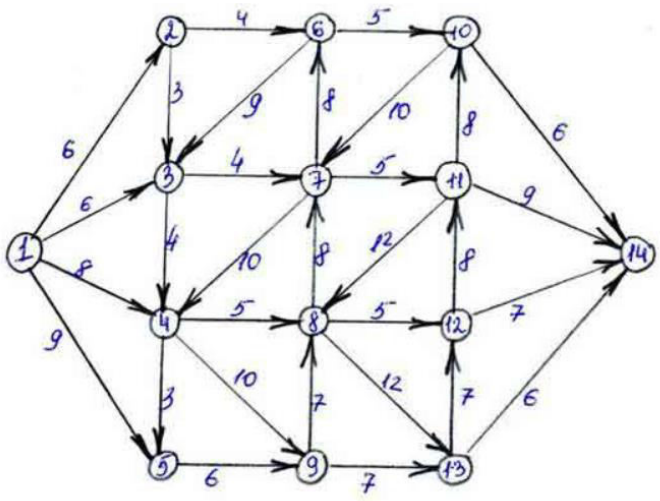


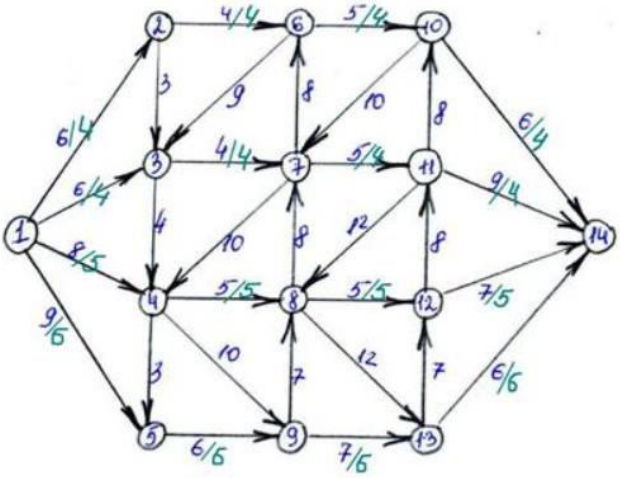
Рисунок 3.2 – Задача на 15 вершин

Розв’язання задачі вручну

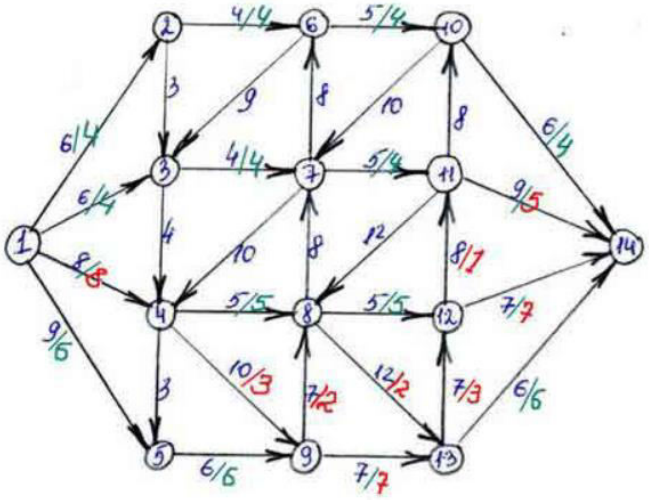
На рисунку 3.3 наведено розв’язання задачі на 14 вершин вручну.

Початкова задача:





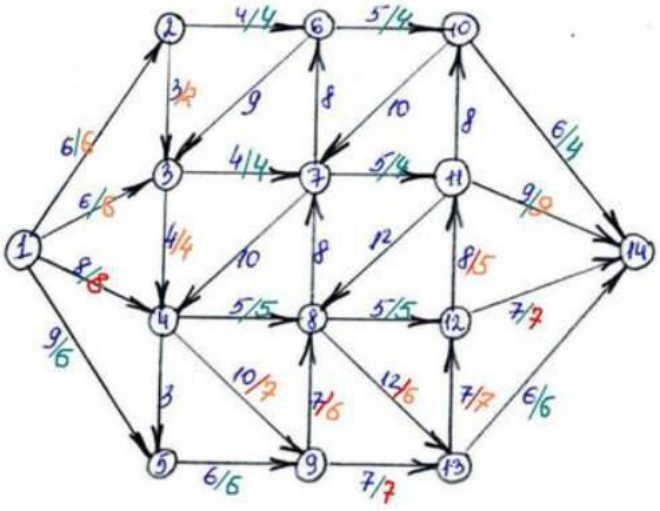
1-5-9-13-14;
1-4-8-12-14;
1-3-7-11-14;
1-2-6-10-14



1-4-9-13-12-14 (тут потік кожної дуги збільшується на одиницю);

1-4-9-8-13-12-14 (тут також на одиницю);

1-4-9-8-13-12-11-14 (і тут також на одиницю).



1-3-4-9-8-13-12-11-14 (тут потік кожної дуги збільшується на двійку);

1-2-3-4-9-8-13-12-11-14 (тут потік кожної дуги знову збільшується на двійку).

Отже, потік дорівнює: 4+9+7+6=6+6+8+6=26.

Рисунок 3.3 – Розв’язання задачі на 14 вершин вручну

ВИСНОВОК

При виконанні даної лабораторної роботи я розібрала та реалізувала алгоритм Форда – Фалкерсона (Задача про максимальний потік) за допомогою мови Python.

В цій лабораторній роботі я порівняла програмне та ручне розв’язання цієї задачі.

Та вивчила додаткові прикладні алгоритми на графах та способи їх імплементації.

КРИТЕРІЇ ОЦІНЮВАННЯ

У випадку здачі лабораторної роботи до 30.03.2022 включно максимальний бал дорівнює – 5. Після 30.03.2022 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

– псевдокод алгоритму – 10%;

– програмна реалізація алгоритму – 50%;

– розв’язання задачі вручну – 20%;

– відповідь на 3 теоретичні питання по темі роботи 15%

– висновок – 5%.

НЕ ДІЮТЬ