МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КПІ»

Кафедра інформаційних систем та технологій

Звіт

з комп'ютерного практикуму 2

«Методи розробки алгоритмів»

з дисципліни

«Теорія алгоритмів»

Бригада – 10

Варіант № 4

Перевірила:

ст. вик. Солдатова М. О.

Виконали:

Бойко Катерина,

Гоголь Софія,

Павлова Софія,

Хіврич Володимир

Київ 2022

Комп'ютерний практикум 2

Тема: Методи розробки алгоритмів.

Мета роботи: порівняння алгоритмів розв'язку задачі, побудованих різними методами.

Завдання

Постановка задачі:

Варіант 2:

Потрібно розрахувати максимальну кількість автомобілів, яка може досягнути Музею однієї вулиці, якщо від КПІ буде прямувати максимальна кількість авто. Вважати, що за годину по одній смузі шляху може проїхати до 400 автомобілів.

Мета задачі – розрахувати максимальну кількість автомобілів, що може досягнути Музею однієї вулиці, від КПІ.

Вибір алгоритму відповідно до поставленої задачі

За математичну залежність обираємо алгоритм Форда-Фалкерсона.

Принцип дії алгоритму:

- Обнулюємо всі потоки. Залишкова мережа спочатку збігається з вихідною мережею.
- У залишковій мережі знаходимо будь-який шлях із джерела в стік. Якщо такого шляху немає, зупиняємось.
- Пускаємо через знайдений шлях (він називається збільшуючим шляхом або ланцюгом, що збільшує) максимально можливий потік:
- На знайденому шляху в залишковій мережі шукаємо ребро з мінімальною пропускною здатністю C_{min}

- Для кожного ребра на знайденому шляху збільшуємо потік на C_{min} а в протилежному йому зменшуємо на C_{min} .
- Модифікуємо залишкову мережу. Для всіх ребер на знайденому шляху, а також для протилежних їм ребер обчислюємо нову пропускну здатність. Якщо вона стала ненульовою, додаємо ребро до залишкової мережі, а якщо обнулилася, перемо його.
- Повертаємось на крок 2.

Використані структури даних: масиви вершин та ребер, цілочисленний тип для збереження кількості вершин та ребер.

Вхідні та вихідні дані:

- У якості вхідних даних беремо файл, в якому знаходиться граф.
- У якості вихідних даних – максимальний потік автомобілів, що доїде до вулиці.

Модель:

Основні величини:

Назва змінної	Тип змінної	Значення змінної
numOfVertex	float	кількість вершин
numOfEdge	float	кількість ребер
sourceVertex	float	джерело
destinationVertex	float	сток
capacity[]	int	пропускна здатність

onEnd[]	int	кінець ребра
nextEdge[]	int	наступна вершина на черзі в список вершин M
edgeCount	int	кількість вершин у списку M
firstEdge[]	int	початок списку вершин M
visited[]	int	маркер відвіданої вершини

Допоміжні величини:

Назва змінної	Тип змінної	Значення змінної
MAX_E	const int	розмір для масиву пропускної здатності, кінця ребер та наступних вершин в черзі
MAX_V	const int	розмір для масиву початку списку вершин M. маркерів відвіданих вершин
read	float	змінна для зчитування байтів з файлу
finput	FILE*	показчик для ім’я файлу

Табл 1. Таблиця величин

Блок-схема

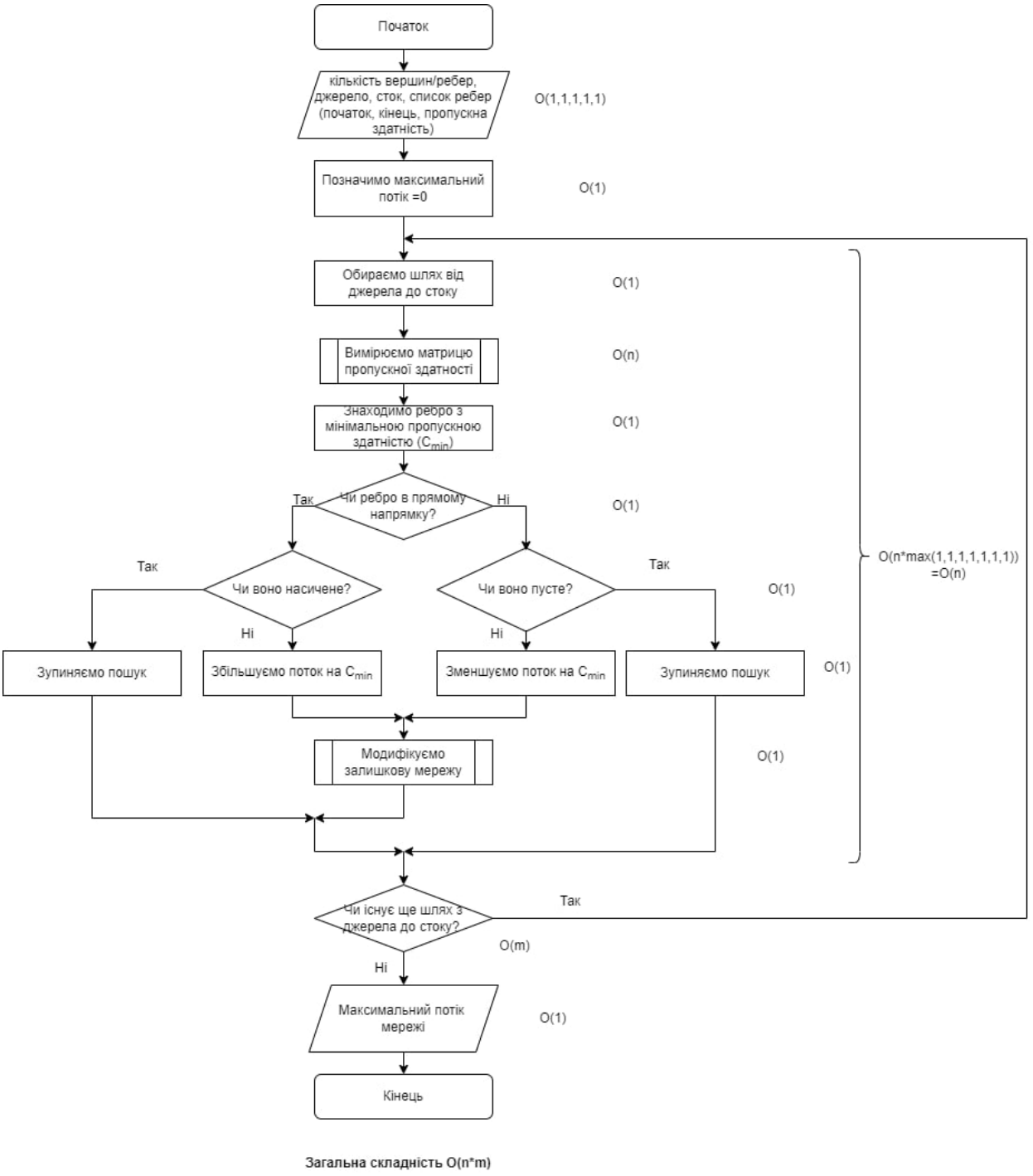


Рис. 1. Блок схема алгоритму програми з оцінкою складності

Оцінка теоретичної складності алгоритму:

На кожному кроці алгоритм збільшує потік хоча б на одну одиницю, отже він закінчить роботу не більше ніж за O(m) кроків, де m - максимальний потік графу. Кожен крок можна виконати за час n, де n - кількість ребер в графі, тоді загальний час роботи алгоритму O(n*m).

Висновки про доцільність використання алгоритму

Алгоритм Форда-Фалкерсона вирішує завдання знаходження максимального потоку в транспортній мережі. У ньому реберно-зважений граф розглядається як мережа труб, причому вага ребра (i,j) задає пропускну здатність труби. Цей алгоритм найбільше підходить для завдань, пов'язаних з розрахунком навантаження на локальну мережу або задач масового обслуговування. Отже саме тому, цей алгоритм дуже добре підійшов до нашого завдання.

Лістінг програми:

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <windows.h>
using namespace std;
#define ICHAR 80  // Довжина рядку опису системи

const int MAX_E = (int)1e6;
const int MAX_V = (int)1e3;
const int INF = (int)1e9;

float numOfVertex1, numOfEdge1;          // кількість вершин, кількість ребер
float sourceVertex1, destinationVertex1; // джерело, сток
int capacity[MAX_E];                     // пропускна здатність
int onEnd[MAX_E], nextEdge[MAX_E];       // кінець ребра, наступна вершина
на черзі в список вершин M
int edgeCount, firstEdge[MAX_V];         // кількість вершин у списку M,
початок списку вершин M
int visited[MAX_V];                      // маркер відвіданої вершини
int numOfVertex, numOfEdge;
int sourceVertex, destinationVertex;
```

```
void addEdge(int u, int v, int cap)
{
   // Пряме ребро - ребро, чий напрямок збігається з напрямком мережі
   onEnd[edgeCount] = v;                    // кінець прямого ребра = v
   nextEdge[edgeCount] = firstEdge[u];      // додаємо в початок списку для u
   firstEdge[u] = edgeCount;                // тепер початок списку = нове
ребро
   capacity[edgeCount++] = cap;             // пропускна здатність

   // Зворотнє ребро - ребро, чий напрямок протилежний напрямку мережі
   onEnd[edgeCount] = u;                    // кінець зворотнього ребра = u
   nextEdge[edgeCount] = firstEdge[v];      // додаємо в початок списку для v
   firstEdge[v] = edgeCount;                // тепер початок списку = нове
ребро
   capacity[edgeCount++] = 0;               // пропускна здатність
}

void ErorCheck(float a)
{
   //Обмеження на вхідні дані
   if (a <= 0 || (a - int(a) != 0))
   {
      cout << "\n * -------- * ПОМИЛКА * -------- * \n";
      cout << "    Введено НЕ натуральне число! \n\n";
      exit(0);
   }
}

int findFlow(int u, int flow)
{
   if (u == destinationVertex)
      return flow; // повертаємо отриманий мінімум на ланцюгу
   visited[u] = true;
   for (int edge = firstEdge[u]; edge != -1; edge = nextEdge[edge])
   {
      int to = onEnd[edge];
      if (!visited[to] && capacity[edge] > 0)
      {
         int minResult = findFlow(to, min(flow, capacity[edge])); // шукаємо потік
у ланцюгу
```

```
            if (minResult > 0)                          // якщо знашли
            {
                capacity[edge] -= minResult;        // від прямих ребер віднімаєм потік
                capacity[edge ^ 1] += minResult;   // до зворотніх ребер додаємо
                return minResult;
            }
        }
    }
    return 0; // якщо не знашли потік з цієї вершини, повернемо 0
}

int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    cout << "\t * -------- * Комп'ютерний практикум 2 * -------- *\n";
    cout << "\t\t\t Бригада 10. Варіант 2  \n";
    cout << "\t\t Реалізація алгоритму Форда-Фалкерсона\n\n";
    fill(firstEdge, firstEdge + MAX_V, -1);     // -1 означає, що ребра немає
    float read;
    char   desc[ICHAR];
    FILE* finput;
    finput = fopen("Ford-Falkerson.TXT", "r");
    if (finput == NULL)
    {
        cout << "Текстовий файл \"Ford-Falkerson.TXT\" НЕ знайдено!\n";
        return(-1);
    }
    // Відскануємо перший рядок файлу до 80 знаків
    fgets(desc, ICHAR, finput);

    // Зчитування кількості вершин
    fscanf(finput, "%f", &numOfVertex1);
    ErorCheck(numOfVertex1);
    numOfVertex = int(numOfVertex1);
    cout << "Кількість вершин = " << numOfVertex << endl;

    // Зчитування кількості ребер
    fscanf(finput, "%f", &numOfEdge1);
    ErorCheck(numOfEdge1);
```

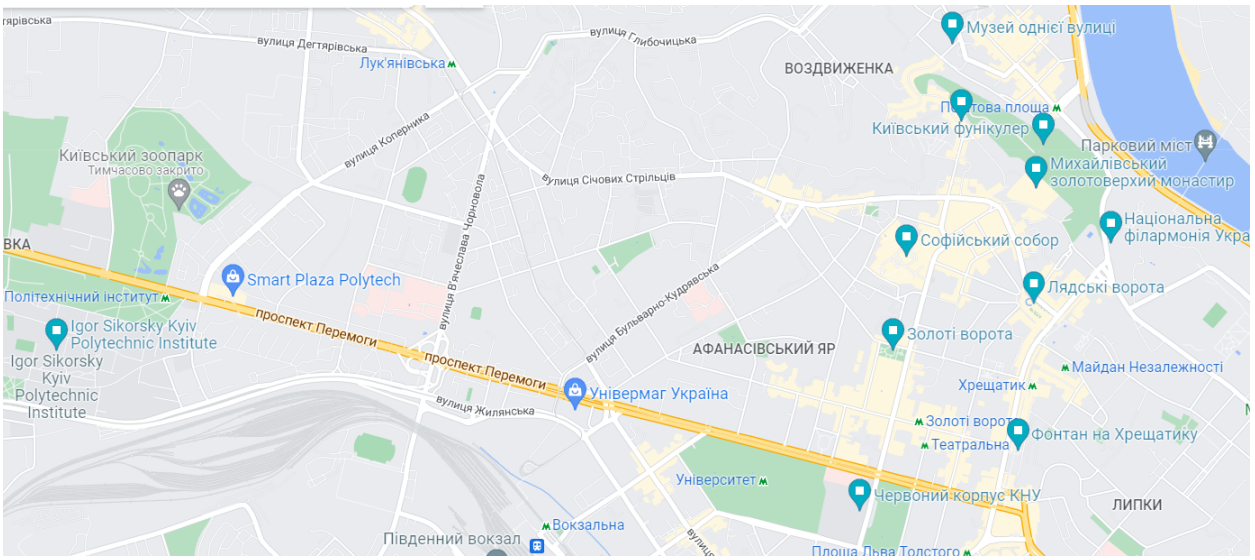
```
    numOfEdge = int(numOfEdge1);
    cout << "Кількість ребер = " << numOfEdge << endl;

    // Зчитування джерела
    fscanf(finput, "%f", &sourceVertex1);
    ErorCheck(sourceVertex1);
    sourceVertex = int(sourceVertex1);
    cout << "\nДжерело - вершина № " << sourceVertex << endl;

    // Зчитування стоку
    fscanf(finput, "%f", &destinationVertex1);
    ErorCheck(destinationVertex1);
    destinationVertex = int(destinationVertex1);
    cout << "Сток - вершина № " << destinationVertex << endl;

    cout << "\nСписок ребер (початок, кінець, пропускна здатність):\n";
    for (float i = 0, u, v, cap; i < numOfEdge; i++)
    {
        fscanf(finput, "%f%f%f", &u, &v, &cap);
        ErorCheck(u); ErorCheck(v); ErorCheck(cap);
        int u1 = int(u), v1 = int(v), cap1 = int(cap);
        addEdge(u, v, cap);
        cout << "e" << i + 1 << " = (" << u << ", " << v << ", " << cap << ")" <<
endl;
    }
    // Знаходження максимального потоку
    int maxFlow = 0;
    int iterationResult = 0;
    while ((iterationResult = findFlow(sourceVertex, INF)) > 0)
    {
        fill(visited, visited + MAX_V, false);
        maxFlow += iterationResult;
    }
    // Виводимо максимальний потік
    cout << "\nМаксимальний потік мережі = ";
    cout << maxFlow << endl;
    cout << "\nМаксимальна кількість авто, що доїде до Музею  = ";
    cout << maxFlow << endl;
    return 0;
}
```

Граф:



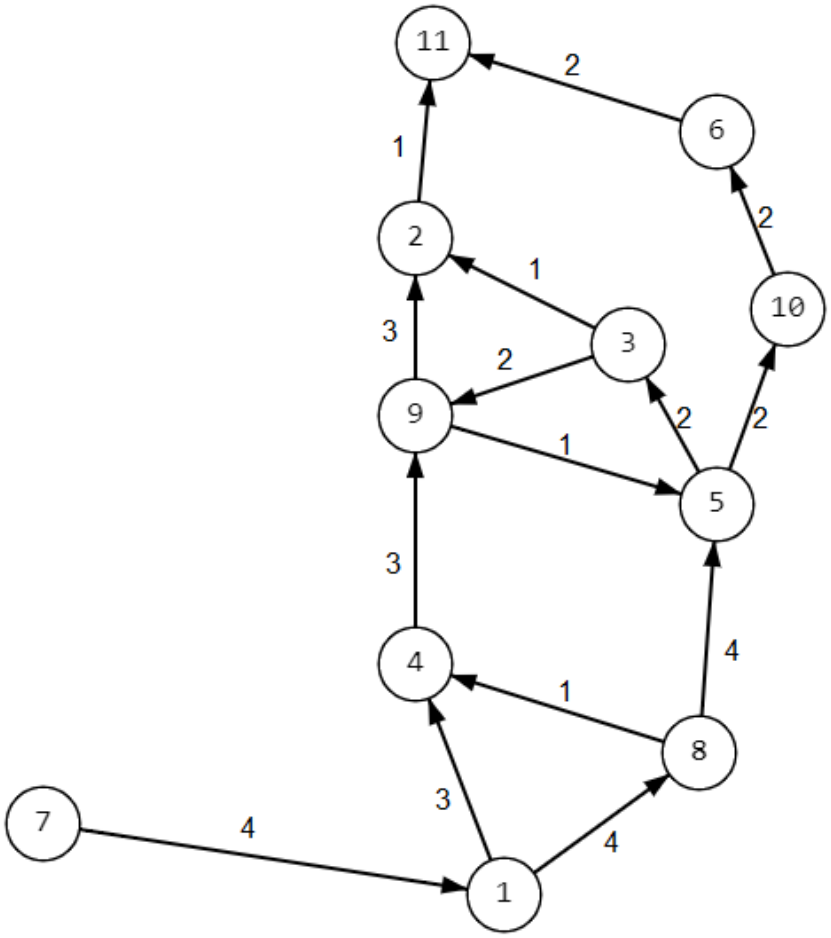


Рис. 2-3 Карта маршруту з переліком місць та граф шляху

Перевірка обчислювальної складності програми:

Діагр. 1. Діаграма часу виконання програми

Червоний пунктирний графік – теоретична складність.
Синій графік – реальна складність.

На великих значеннях > 500 000 ребер, алгоритм перестає збігатися. Проте за отриманим графіком можна зробити прогноз моди та переконатися в тому, що теоретична та обчислювальна складності співпадають.

Перевірка обчислювальної складності програми підтвердила аналітичну оцінку O(n*m).

Перевірка правильності програми:

Вхідні дані Filename	Результат	Призначення тесту
IncorrectName.asdf	Відомий	Перевірка реакції на некоректні вхідні дані (додавання повідомлень про помилки)

CorrectName.txt	Відомий	Програма працює та виконує свої вимоги коректно
CorrectName.txt	Відомий	Якщо файл існує, але сутність його не відповідає дійсності, буде виведена помилка
numOfVertex (value = 2.5)	Відомий	Перевірка реакції на некоректні вхідні дані (додавання повідомлень про помилки)
numOfVertex (value = 0)	Відомий	Перевірка реакції на некоректні вхідні дані (додавання повідомлень про помилки)
numOfVertex (value = -1)	Відомий	Перевірка реакції на некоректні вхідні дані (додавання повідомлень про помилки)
numOfEdge (value = -1)	Відомий	Перевірка реакції на некоректні вхідні дані (додавання повідомлень про помилки)
numOfEdge (value = 2.5)	Відомий	Перевірка реакції на некоректні вхідні дані (додавання повідомлень про помилки)
numOfEdge (value = 0)	Відомий	Перевірка реакції на некоректні вхідні дані (додавання повідомлень про помилки)

sourceVertex (value = -1)	Відомий	Перевірка реакції на некоректні вхідні дані (додавання повідомлень про помилки)
sourceVertex (value = 2.5)	Відомий	Перевірка реакції на некоректні вхідні дані (додавання повідомлень про помилки)
sourceVertex (value = 0)	Відомий	Перевірка реакції на некоректні вхідні дані (додавання повідомлень про помилки)

Табл. 2. Таблиця тестування програми

Скріншоти роботи програми:

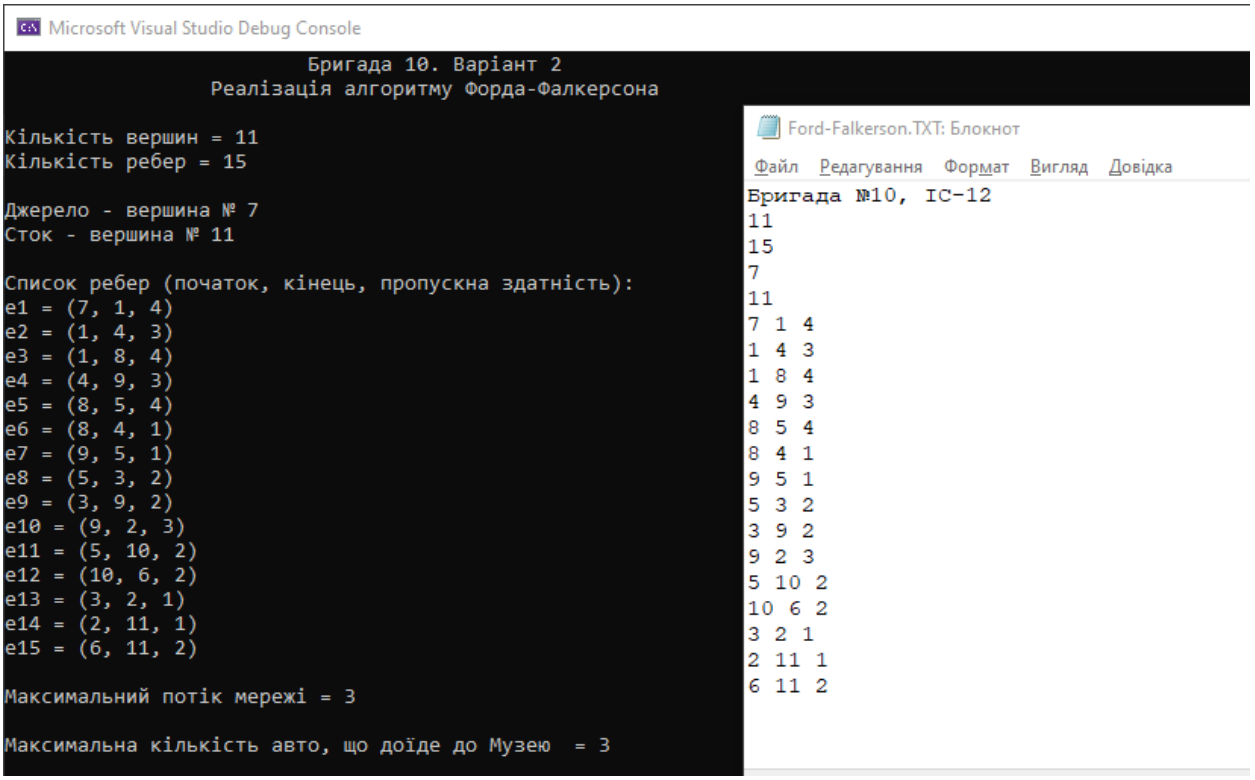


Рис. 4. Результат виконання програми

Висновок

Під час виконання лабораторної роботи ми ознайомились із різними методами обрання алгоритмів для виконання завдання. Обравши за основний алгоритм Форда-Фалкерсона ми розрахували максимальну кількість автомобілів, яка може досягнути Музею однієї вулиці від КПІ, побудувавши граф шляху та використавши програмну реалізацію. Наше дослідження було перевірено нативно, за допомогою функції перегляду вулиць Google Maps, ми повторили заданий маршрут.

Відповіді на контрольні запитання:

1. *Перерахуйте відомі вам методи розробки алгоритмів. Докладніше розкажіть про один з них.*

- метод проміжних цілей (розбиття)
- метод локального пошуку (підйому)
- метод перебору (пошуку з поверненням)

Метод проміжних цілей пов'язаний зі зведенням важкої задачі до послідовності більш простих задач. Він припускає таку декомпозицію (розбиття) завдання розміру n на дрібніші завдання, що на основі рішень цих дрібніших задач можна отримати рішення початкового завдання.

2. *Перерахуйте переваги та недоліки наступних методів розробки алгоритмів: методу часткових (проміжних) цілей, методу підйому (локального пошуку), методу відпрацювання назад.*

Метод проміжних цілей

Переваги: якщо розв'язати цілу задачу не є можливим, то для проміжних задач можна знайти такий алгоритм пошуку, що зробить розв'язання набагато простіішим.

Недоліки: використання даного методу не завжди є простим та чітким.

Метод підйому

Переваги: знаходження "хороших", але не надто оптимальних розв'язків. Легка реалізація.

Недоліки: грубість алгоритму за рахунок намагання покращити розв'язок у будь-який спосіб.

3. *Який тип алгоритмів називають «жадібними» і чому?*

Жадібний алгоритм – простий і прямолінійний евристичний алгоритм, який приймає найкраще рішення, виходячи з наявних на кожному етапі данних.

4. *Дайте характеристику евристичним алгоритмам. В яких випадках доцільно використовувати цей тип алгоритмів? Опишіть загальний підхід до побудови евристичних алгоритмів.*

Евристичний алгоритм або евристика, визначається як алгоритм з наступними властивостями:

- він зазвичай знаходить хороше, хоча не обов’язково оптимальне рішення;
- його можна швидше і простіше реалізувати, ніж будь-який відомий точний алгоритм (тобто такий, що гарантує оптимальний розв’язок).

Хоча не існує універсальної структури, якою можна описати евристичні алгоритми, переважна більшість з них базується або на методі часткових цілей або на методі підйому.

6. *Поясніть, для чого можна використовувати метод альфа-бета відсікання.*

Метод альфа-бета відсікання – алгоритм пошуку, що зменшує кількість вузлів, які необхідно оцінити в дереві пошуку мінімаксного алгоритму і при цьому дозволяє отримати ідентичний результат.

7. *Поясніть термін «структурне програмування». Для чого воно застосовується?*

Структурне програмування - це технологія створення програм, що дозволяє шляхом дотримання певних правил зменшити час розробки і кількість помилок, а також полегшити можливість модифікації програми. Структурний підхід охоплює всі стадії розробки проекту: специфікацію, проектування, власне програмування і тестування.