МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТРАНСПОРТНИЙ УНІВЕРСИТЕТ

НАДВІРНЯНСЬКИЙ КОЛЕДЖ НТУ

КУРСОВИЙ ПРОЕКТ

з дисципліни “Алгоритмізація та програмування”

на тему: «Знаходження максимального потоку

за методом Форда−Фалкерсона»

Студент групи ІТ-21

_____Пицюк В.М.

(підпис, дата)

Керівник викладач

_____ Грига В.М.

(підпис, дата)

Оцінка_____

Члени комісії _____ _____

(підпис) (ініціали та прізвище)

_____ _____

(підпис) (ініціали та прізвище)

_____ _____

(підпис) (ініціали та прізвище)

НАДВІРНА

2017

Зміст

Вступ

В наш час інформаційні технології займають одне з найважливіших місць у всіх сферах нашого життя. Комп'ютери застосовуються скрізь: в навчанні, в менеджменті, в торгівлі, на виробництві та в інших видах діяльності людини. Але функціонування будь-якого комп'ютера неможливе без необхідних програм, а отже і алгоритмів, на основі яких пишуться програми. Таким чином, різноманітні алгоритми щодня допомагають людині у різних сферах діяльності. І деякі з них відіграють дуже важливу роль в розвитку людства. Отже, питання алгоритмізації є дуже актуальними й потребують багато уваги для подальшої розробки алгоритмів та вдосконалення вже існуючих. Поряд із цим, не може залишатися осторонь, також й процес програмування, як один з фундаментальних розділів інформатики.

Однією за важливих задач, вирішення якої допомагає оптимізувати транспортування вантажів, побудову нафто-, водо- та газопроводів, проектування електромереж є задача пошуку максимального потоку мережі. Для вирішення якої часто використовується метод Форда-Фалкерсона. Алгоритм реалізації якого, за допомогою методу обходу в глибину, ми і розглянемо більш детально.

Об'єктом дослідження є потоки в транспортних мережах.

Предметом дослідження є процес знаходження максимального потоку у транспортній мережі за допомогою алгоритму Форда-Фалкерсона.

Метою курсової роботи є реалізація алгоритму Форда-Фалкерсона мовою програмування C++.

1. Аналітичний розділ

1.1. Загальні поняття

Теорія графів — розділ математики, що вивчає властивості графів. Наочно граф можна уявити як геометричну конфігурацію, яка складається з точок (вершини) сполучених лініями (ребрами). У строгому визначенні графом називається така пара множин G = (V, E), де V є підмножина будь-якої зліченної множини, а E — підмножина V × V.

Визначення графу є настільки загальним, що цим терміном можна описувати безліч подій та об'єктів повсякденного життя. Високий рівень абстракції та узагальнення дозволяє використовувати типові алгоритми теорії графів для вирішення зовнішньо несхожих задач у транспортних і комп'ютерних мережах, будівельному проектуванні, молекулярному моделюванні тощо.

Теорія графів знаходить застосування, наприклад, в геоінформаційних системах (ГІС). Існуючі або запроектовані будинки, споруди, квартали тощо розглядаються як вершини, а з'єднують їхні дороги, інженерні мережі, лінії електропередачі тощо — як ребра. Застосування різних обчислень, вироблених на такому графі, дозволяє, наприклад, знайти найкоротший об'їзний шлях або найближчий продуктовий магазин, спланувати оптимальний маршрут.

Теорія графів містить велику кількість невирішених проблем і поки не доведених гіпотез.

Формальне означення графа**:**

Нехай X = { x1,…,xn} — деяка скінченна множина (множина вершин), M2- множина всіх невпорядкованих пар елементів з X,

M2 = {(xi,xj): xi ∈ X, xj ∈ X, i ≠ j}

1. Граф G(X, W) — пара множин X, W ⊂ M2. Множина X — це множина вершин, множина W — це множина ребер. Якщо (xi,xj) ∈ W, то ми говоримо, що ребро (xi,xj) сполучає вершину xi з вершиною xj; інша термінологія — ребро (xi,xj) і вершини xi та xj інцидентні.

2. Граф G(X, W) називається повним, якщо W = M2. Якщо множина X містить n вершин, то, очевидно, число ребер повного графа дорівнює Cn2. Повний граф з n вершинами позначається Kn.
3. Граф G(X, W) називається порожнім, якщо W = ∅.
4. Вершини x_i та x_j графа G(X, W) інцидентні, якщо $(x_i,x_j) \in W$.
5. Степенем $d(x_i)$ вершини x_i графа G(X, W) називається число вершин x_j, які інцидентні вершині x_i (число відрізків, які виходять з вершини x_i).
6. Якщо $d(x_i)=1$, то вершина x_i називається *кінцевою* вершиною графа G(X, W). Якщо $d(x_i)=0$, то вершина x_i називається *ізольованою*.

В теорії графів, потокова мережа (англ. flow network) це орієнтований граф де кожне ребро має ємність, пропускну спроможність і кожне ребро отримує потік. Загальний потік на ребрі не може перевищувати ємність ребра. В дослідженні операцій орієнтований граф часто називають мережею, вершини — вузлами і ребра — дугами. Потік має задовільняти обмеженю, що загальний вхідний потік вершини дорівнює загальному вихідному, за винятком джерела, що має більший вихідний потік, або стоку, що має більший вхідний потік. Таку мережу можна використати для моделювання руху в дорожній системі, струму в електронних мікросхемах або будь-чого, що рухається через мережу вузлів.

В теорії оптимізації та теорії графів, задача про максимальний потік полягає у знаходженні такого потоку за транспортною мережею, щоб сума потоків з витоку, або, що означає те ж саме, сума потоків до стоку була максимальна.

Задача про максимальний потік є окремим випадком більш складних задач, таких, як, наприклад, задача про циркуляцію.

Після вступу США у Другу світову війну у 1941 році математик Джордж Бернард Данціг почав працювати у відділі статистичного управління Військово-повітряних сил США у Вашингтоні. З 1941 по 1946 роки він очолював підрозділ аналізу військових дій (Combat Analysis Branch), де працював над різноманітними математичними проблемами. Згодом з використанням роботи Данцига задача про максимальний потік була вперше розв'язана у ході

підготовки повітряного мосту під час Берлінського повітряного мосту, що відбувався у 1948–1949 роках.

У 1951 році Джордж Данциг вперше сформулював задачу у загальному вигляді.

У 1955 році Лестер Форд і Делберт Фалкерсон вперше побудували алгоритм, призначений для вирішення цього завдання. Їх алгоритм отримав назву алгоритм Форда — Фалкерсона.

Надалі рішення задачі багато разів поліпшувалося.

У 2010 році дослідники Джонатан Келнер (Jonathan Kelner) і Олександр Мондри (Aleksander Mądry) з MTI разом зі своїми колегами Даніелєм Спілманом з Єльського університету і Шень-Хуа Тенем з університету Південної Каліфорнії продемонстрували чергове покращення алгоритму, вперше за 10 років.

Метод Форда-Фалкерсона знаходить максимальний потік у транспортній мережі. Метод Форда-Фалкерсона - метод, який базується на трьох концепціях: залишкові мережі, шляхи що збільшуються і розрізи. Ключову роль у методі Форда-Фалкерсона грають два поняття: залишкові мережі і доповнюють шляху. Дані концепції лежать в основі важливої теореми про максимальний потік і мінімальний розріз, яка визначає значення максимального нащадка за допомогою розрізів траспортної мережі.

Метод Форда-Фалкерсона є ітеративним. Спочатку величині потоку присвоюється значення 0: $f(u, v) = 0$ при будь-яких u , $v \in V$. На кожній ітерації величина потоку збільшується за допомогою пошуку «шляху, що збільшується» (тобто деякого шляху від джерела s до стоку t, уздовж якого можна послати більший потік) і подальшого збільшення потоку. Цей процес повторюється до тих пір, поки вже неможливо буде відшукати збільшуючий шлях.

1.2. Застосування

Даний алгоритм застосовується:

- В комунікаційних і транспортних системах. Зокрема, для маршрутизації даних в Інтернеті;
- В побудові нафто-, водо- та газопроводів;
- В проектуванні електромереж;
- В економіці;
- В дискретній математиці;

1.3. Методи знаходження максимального потоку.

Алгоритм просовування предпотоку

Замість потоку оперує з передпотоком. Різниця в тому, що для будь-якої вершини u, крім джерела і стоку, сума потоків, що входять до неї для потоку повинна бути строго нульовою (умова збереження потоку), а для передпотока — невід'ємною. Ця сума називається надмірним потоком у вершину, а вершина з позитивним надмірним потоком називається переповненою . Крім того, для кожної вершини алгоритм зберігає додаткову характеристику, висоту, яка є цілим невід'ємним числом. Алгоритм використовує дві операції: просування , коли потік по ребру, що йде з більш високої в нижчу вершину, збільшується на максимально можливу величину, і підйом , коли переповнена вершина, просування з якої неможливо через недостатню висоту, підіймається. Просування можливо, коли ребро належить залишковій мережі, коли воно веде з більш високої вершини в більш низьку, і вихідна вершина переповнена. Підйом можливий, коли вершина, що піднімається, переповнена, але жодна з вершин, в котру з неї ведуть ребра залишкової мережі, не нижче за неї. Він вчиняється до висоти на 1 більшою, ніж мінімальна з висот цих вершин. Спочатку висота джерела V, по всім ребрам, що виходять з джерела, тече максимально можливий потік, а решта висоти і потоки нульові. Операція просування і підйому виконуються до тих пір, поки це можливо.

Алгоритм Едмондса-Карпа

Виконуємо алгоритм Форда — Фалкерсона, щоразу обираючи найкоротший шлях, що збільшується (знаходиться пошуком у ширину).

Алгоритм Дініца

Удосконалення алгоритму Едмондса-Карпа (але хронологічно був знайдений раніше). На кожній ітерації, використовуючи пошук у ширину, визначаємо відстані від джерела до всіх вершин у залишковій мережі. Будуємо граф, який містить лише такі ребра залишкової мережі, на яких ця відстань зростає на 1. Виключаємо з графа усі тупикові вершини з інцидентними їм ребрами, поки всі вершини стануть не тупиковими. (Тупиковою називається вершина, в яку не входить і з якої не виходить жодне ребро, крім джерела і стоку.) На отриманому графі відшукуємо найкоротший шлях, що збільшується (їм буде будь-який шлях з s в t). Виключаємо із залишкової мережі ребро з мінімальною пропускною здатністю, знову виключаємо тупикові вершини, і так поки ще існують шляхи, що збільшуються.

1.4. Опис математичної моделі метода Форда-Фалкерсона

Нехай *G(V, E)* граф, і для кожного ребра з u в v, нехай *c(u, v)* буде ємність і *f(u , v)* буде потік. Ми хочемо знайти максимальний потік від джерела s до раковини t. Після кожного кроку в алгоритмі виходить наступне:

- **Обмеженість потенціалу:** $\forall (u,v) \in E \; f(u,v) \leq c(u,v)$ Потік уздовж краю не може перевищувати свій потенціал
- **Косі симетрії:** $\forall (u,v) \in E \; f(u,v) = -f(v,u)$ Чистий потік від u до v повинен бути протилежністю чистого припливу від *v* до *u*
- **Збереження потоку:** $\forall u \in V : u \neq s \text{ and } u \neq t \Rightarrow \sum_{w \in V} f(u,w) = 0$ Тобто, якщо u не s або t. Чистий потік до вузла дорівнює нулю, для джерела, який "виробляє" потік, і раковину, яка "поглинає" потік

- **Значення (F):** $\sum_{(s,u)\in E} f(s,u) = \sum_{(v,t)\in E} f(v,t)$ Тобто, потік виходячи з s повинен бути рівним потоку, що надходить у *t*

Це означає, що потік через мережу є легальним потоком після кожного раунду в алгоритмі. Визначимо залишкову мережу $G_f(V, E_f)$, щоб бути в мережі з потужністю $c_f(u, v)=c(u, v)-f(u, v)$, і без потоку. Зверніть увагу, що може статися, що потік від v до u дозволений в залишковій мережі, хоча заборонений в вихідній мережі: якщо $f(u, v)>0$ та $c(v, u)=0$, тоді $c_f(v, u) = c(v, u)-f(v, u)=f(u, v)>0$.

2. Опис програмної частини

2.1. Опис алгоритму

Ідея алгоритму полягає в наступному. Ми вибираємо такий шлях від джерела до стоку, щоб для кожного ребра залишкова пропускна здатність була строго більше нуля. При цьому ребра на даному шляху можуть проходитися як у прямому, так і в зворотному напрямку. Вибираємо мінімальне значення серед залишкових пропускних спроможностей ребер даного шляху. Збільшуємо потік на кожному з ребер даного шляху на обране мінімальне значення. Далі шукаємо наступний аналогічний шлях. Робота алгоритму продовжується до тих пір, поки вдається знаходити дані шляхи. Відразу відзначимо, що даний алгоритм відноситься до класу недетермінованих, тобто кожен наступний крок алгоритму визначено неоднозначно. І час роботи (кількість кроків) алгоритму залежить від того, як будуть вибиратися кроки.

Алгоритм Форда-фалкерсона:

1. Прирівнюємо до нуля всі потоки. $\forall e(v_i, v_j) \in E \quad f(e) = 0$. Залишкова мережа спочатку збігається з вихідною мережею;
2. У залишкової мережі знаходимо будь-який шлях з джерела s у стік t . Дуги якого задовольняють умові $f(v_i, v_j) <= (v_i, v_j)$. Якщо такого шляху немає, то потік у мережі максимальний;
3. Пускаємо через знайдений шлях (він називається збільшувальним шляхом) максимально можливий потік;
4. На знайденому шляху в залишковій мережі шукаємо ребро з мінімальною пропускною здатністю C_{min};
5. Для кожного ребра на знайденому шляху збільшуємо потік на C_{min} , а в протилежному йому — зменшуємо на C_{min};
6. Модифікуємо залишкову мережу. Для всіх ребер на знайденому шляху, а також для протилежних їм ребер, обчислюємо нову пропускну здатність. Якщо нова пропускна здатність не дорівнює нулю, додаємо ребро до залишкової мережі, а якщо дорівнює нулю, стираємо його;
7. Повертаємося на крок 2.

Важливо те, що алгоритм не конкретизує, який саме шлях ми шукаємо на кроці 2 або як ми це робимо. З цієї причини алгоритм гарантовано сходиться тільки для цілих пропускних спроможностей, але навіть для них при великих значеннях пропускних спроможностей він може працювати дуже довго або зовсім не привести до оптимального рішення.

2.2. Опис меню користувача

Розроблена програма шукає максимальний потік за методом Форда-Фалкерсона. Також в програмі можна переглянути інформацію про курсовий проект.

На самому початку я включив в програму залежні бібліотеки через директиву препроцесора #include:

```
#include <iostream>
#include <limits.h>
#include <string.h>
#include <queue>
```

А також включаємо <windows.h>, для виклику функцій, які відповідають за кодування в ОС Windows, яке підтримує українські букви:

```
#include <windows.h>
```

І об’явимо константу V, яка відповідає за кількість вершин в графі, який буде пізніше задано:

```
#define V 6
```

Розглянемо точку входу (функцію main()):

Для використання українських букв я використав функції SetConsoleOutputCP(), яка встановлює кодування виводу на консоль і SetConsoleCP(), яка встановлює кодування для вводу, з вище включеної бібліотеки windows.h:

```
SetConsoleCP(1251);
SetConsoleOutputCP(1251);
```

Наступним є вивід слова courseword (курсова робота англ.), написаного програмою figlet, яка створює текст з великих літер з звичайного тексту. В наступному рядку вивів слово “@vovawed” вирівнюючи його так щоб кінець обох слів був в одному місці. Для цього я зробив 47 пробілів через перезавантаження конструктора класу string, який дозволяє записувати символ вказану кількість разів(перший параметр – кількість повторень, другий - символ для повтору):

```
cout << string(47, ' ') + "@vovawed" << endl << endl;
```

Результат можна побачити на фото

Рис. 1.1.

Пізніше виводиться інформація про меню програми:

```
cout << "1) Інформація про курсовий проект" << endl
<< "2) Запустити виконання програми" << endl
<< "3) Вихід" << endl;
```

Потім об’являється змінна num, типу short, яка буде зберігати відповіді на меню до програми:

```
short num;
```

За допомогою оператора switch обробляється змінна num, яка перед цим вводиться значенням з клавіатури. Оператор switch заключений в цикл do-while, з умовою num != 2 (2 – запуск програми). Switch обробляє 4 варіанти:

1. Виводить інформацію про дану курсову роботу
2. Запускає виконання
3. Вихід з програми

І якщо змінна num не в діапазоні 1-3, то виводиться повідомлення про невірний ввід і програма просить ввести число ще раз. Сам код:

```
do {
   cout << "Введіть число: ";
   cin >> num;
```

```
        switch (num) {
            case 1:
                cout << "Повідомлення про курсову" << endl << endl;
            case 2:
                break;
            case 3:
                cout << "Вихід" << endl;
                return 0;
            default:
                cout << "Не вірний ввід, спробуйте ще раз" << endl;
        }
    } while (num != 2);
```

Далі вводиться матриця, кількість вершин для якої вказана вище в костанті V (6):

```
    int graph[V][V] = {
                { 0, 16, 13, 0, 0, 0 },
                { 0, 0, 10, 12, 0, 0 },
                { 0, 4, 0, 0, 14, 0 },
                { 0, 0, 9, 0, 0, 20 },
                { 0, 0, 0, 7, 0, 4 },
                { 0, 0, 0, 0, 0, 0 }
        };
```

Потім на екран виводиться повідомлення: “Максимально можливий потік: 23”, де 23 – максимальний потік який вираховує і повертає функція fordFulkerson(), описана нижче:

```
    cout << "Максимально можливий потік: " << fordFulkerson(graph, 0, 5) << endl;
```

В самому кінці викликається функція system() з аргументом “pause”, яка виводить на екран повідомлення “Press any key to continue . . .” та чекає нажаття будь-якої клавіші:

```
    system("pause");
```

Функція fordFulkerson():

Функція fordFulkerson() повертає максимальний потік від s до t в даному графіку:

```
int fordFulkerson(int graph[V][V], int s, int t)
{
    int u, v;
    // Створення залишкового графа і заповнити залишковий граф з
    // з урахуванням потенціалу в якості вихідного графа залишкових потужностей
    // в залишковому графі
    int rGraph[V][V]; // Залишковий граф де rGraph[i][j] вказує
    // залишкову ємність ребра від i до j (якщо існує
    // ребро. Якщо rGraph[i][j] дорівнює 0, то немає)
    for (u = 0; u < V; u++)
        for (v = 0; v < V; v++)
            rGraph[u][v] = graph[u][v];

    int parent[V]; // Цей масив заповнюється пошуком в ширину і зберігає шлях
    int max_flow = 0;
    // Збільшити потік поки йде шлях від джерела до приймача
    while (bfs(rGraph, s, t, parent))
    {
        // Знайти максимальний потік через знайдений шлях
        int path_flow = INT_MAX;
        for (v = t; v != s; v = parent[v])
        {
            u = parent[v];
            path_flow = min(path_flow, rGraph[u][v]);
        }
        // Оновити залишкові можливості ребер і
        // зворотний край уздовж шляху
        for (v = t; v != s; v = parent[v])
        {
            u = parent[v];
```

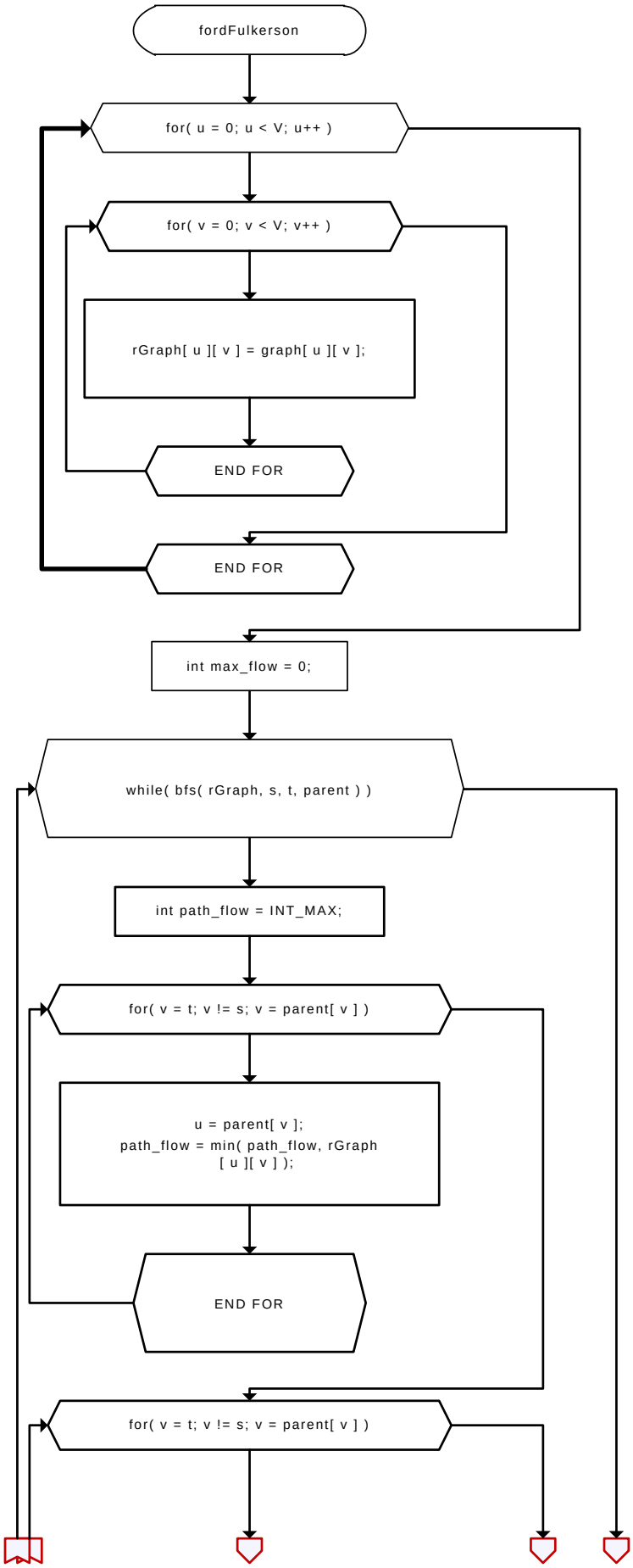
```
            rGraph[u][v] -= path_flow;
            rGraph[v][u] += path_flow;
        }

        // Додати потік шляху до максимального потоку
        max_flow += path_flow;
    }

    // Повертає максимальний потік
    return max_flow;
}
```

В цій функції використовується інша функція bfs() типу bool, яка повертає true якщо є шлях від 's' до 't' в залишковий граф. Її тіло:

```
bool bfs(int rGraph[V][V], int s, int t, int parent[])
{
    // Створити масив відвіданих і позначити всі вершини які не відвідані
    bool visited[V];
    memset(visited, 0, sizeof(visited));
    // Створити чергу, поставлене в чергу джерело вершини і мітки джерела вершини як відвідані
    queue <int> q;
    q.push(s);
    visited[s] = true;
    parent[s] = -1;
    // Пошук в ширину
    while (!q.empty())
    {
        int u = q.front();
        q.pop();
        for (int v = 0; v<V; v++)
        {
            if (!visited[v] && rGraph[u][v] > 0)
```

```
            {
                q.push(v);
                parent[v] = u;
                visited[v] = true;
            }
        }
    }
    return visited[t];
}
```

2.3. Блок-схема



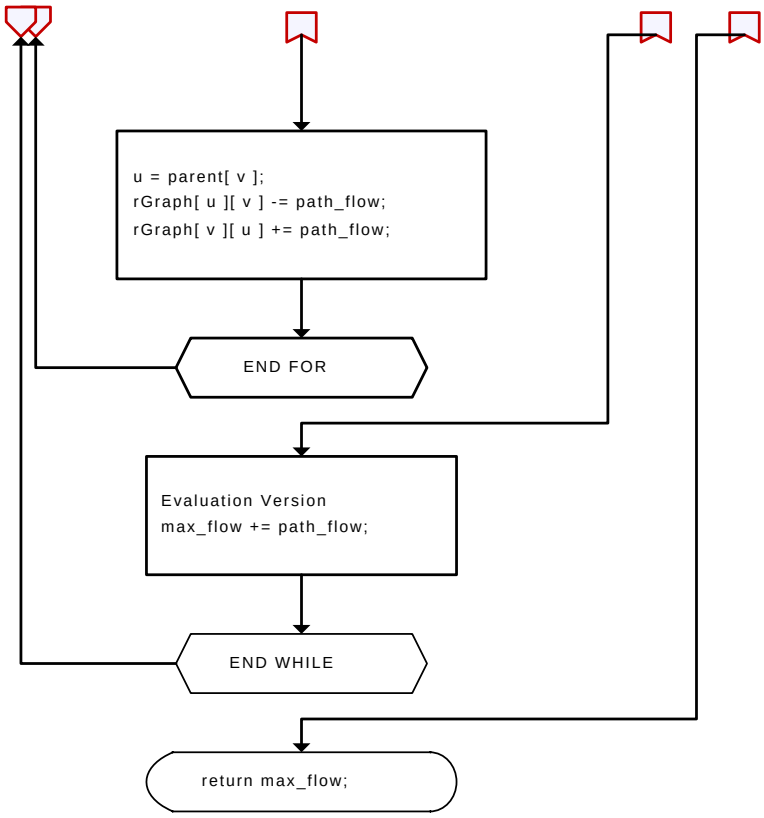


Рис. 2.1. Блок-схема функції fordFulkerson()

3. Тестування програми

3.1. Системні вимоги

Будь-яка машина, на яку є компілятор C++, як мінімум 400 кб оперативної пам’яті. Для коректної роботи на компіляторі Visual Studio і ОС Windows перед компіляцією потрібно розкоментувати деякі строки (вказано в коментарях).

3.2. Запуск програми

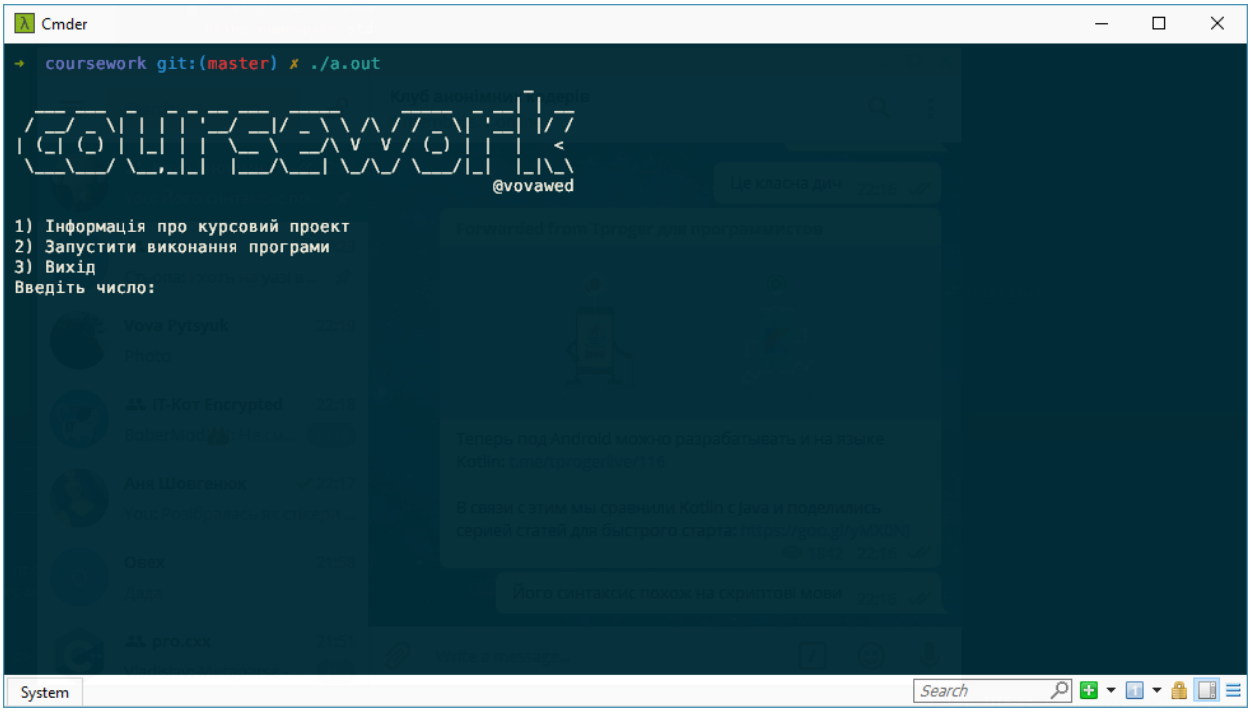


Рис. 3.1. Головне меню

Щоб запустити дану програму потрібно скомпілювати її під вашу ОС, якщо це Windows і компілятор Visual Studio C++, то ще необхідно розкоментувати деякі строки, для коректного відображення український букв. А також розкоментувати system(“pause”), щоб консоль з програмою не закрилась зразу після завершення роботи програми. Вже зібрану програму (exe файл) для 64х бітної версії Windows можна завантажити на http://bit.ly/2rzb8av.

3.3. Інструкція користувача

Після запуску програми відобразиться текст coursework та меню з трьома пунктами (Рис. 3.1.): інформація про курсовий проект, запуск програми і вихід. Потрібно вибрати 1 з цих пунктів і ввести його в програму. Програма зробить

відповідну до цього числа дію. Якщо вибрати 1, то на екран виведеться інформація про тему курсового проекту, студента, навчальний заклад. Після цього програма не закінчує роботу, а чекає ввід нової цифри.

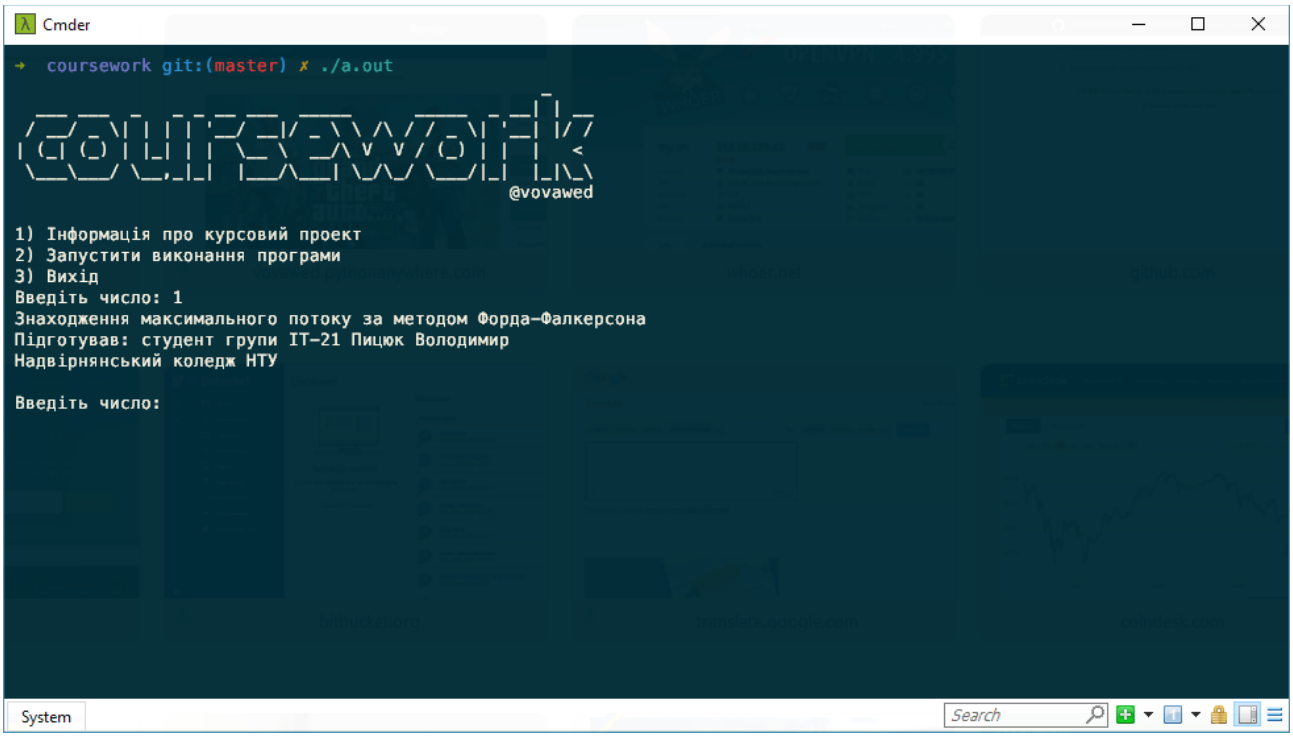


Рис. 3.2. Інформація про курсовий проект

Якщо ввести 2, то програма виведе максимальний потік графа, заздалегідь введеного в код програми, який має наступний вигляд:

```
int graph[V][V] = {
        { 0, 16, 13, 0, 0, 0 },
        { 0, 0, 10, 12, 0, 0 },
        { 0, 4, 0, 0, 14, 0 },
        { 0, 0, 9, 0, 0, 20 },
        { 0, 0, 0, 7, 0, 4 },
        { 0, 0, 0, 0, 0, 0 }
};
```

Після того як програма вивела результат обчислення максимального потоку вона виводить повідомлення “Press any key to continue . . .” та після нажаття клавіші Enter закінчує свою роботу.

Якщо ввести цифру 3 (Рис. 3.3.), то програма завершить свою роботу (без виводу повідомлення “Press any key to continue . . .”).

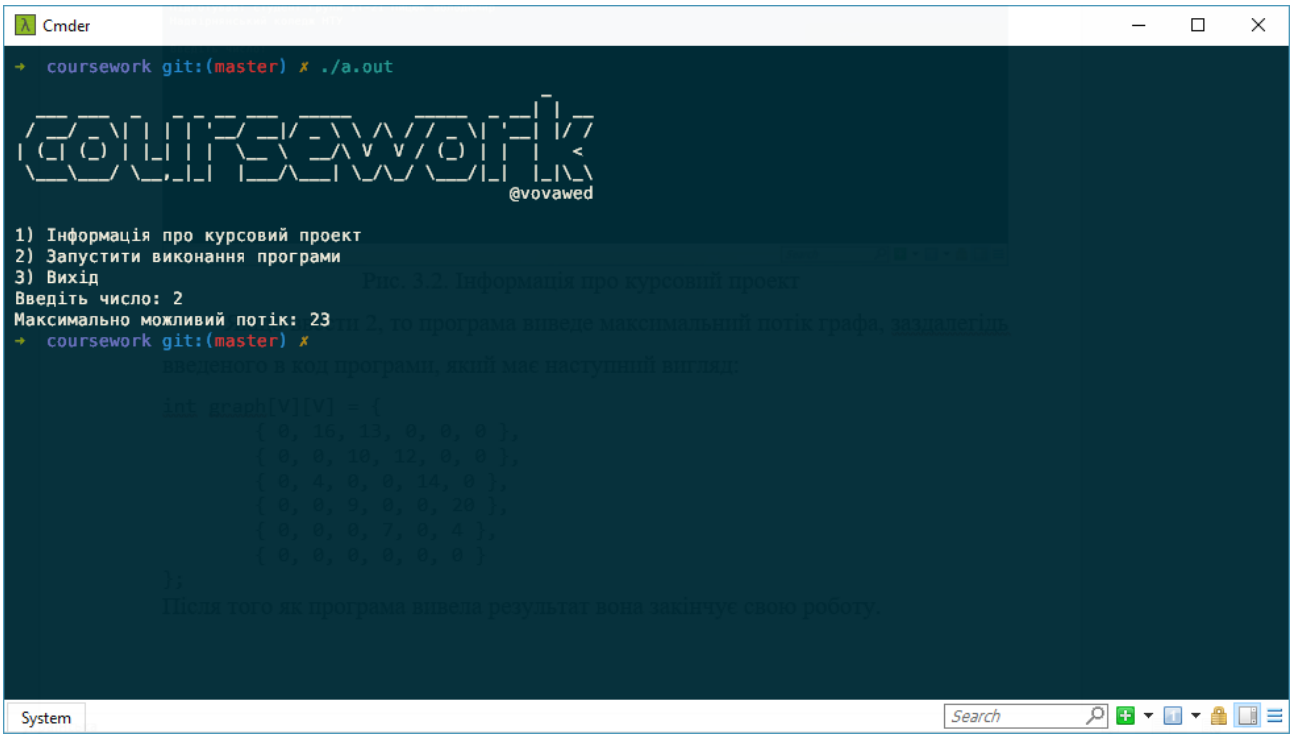


Рис. 3.3. Завершення роботи програми

Висновок

В даному курсовому проекті реалізувано алгоритм Форда-Фалкерсона, який знаходить максимальний потік у транспортній мережі, мовою програмування C++.

У першому розділі було розглянуто теоритичні ввідомості і загальну інформацію.

У другому розділі подано аглоритм пошуку максимального потоку за методом Форда-Фалкерсона і його реалізацію мовою програмування C++.

У третьому розділі запуск програми і інструкція користувача.

Даний алгоритм використовується:

- В комунікаційних і транспортних системах. Зокрема, для маршрутизації даних в Інтернеті;
- В побудові нафто-, водо- та газопроводів;
- В проектуванні електромереж;
- В економіці;
- В дискретній математиці;

Використані джерела

1. Вступ в алгоритми / Т.Кормен, Ч. Лейзерсон, Р. Рівест, К. Штайн. – Кембрідж: MIT press, 1990. – 1312 с. – (3).
2. Ліппман С. Мова програмування C++. Базовий курс / С. Ліппман, Ж. Лажойе, Б. Му., 2014. – 1120 с. – (5).
3. Ковалюк Т. В. Основи програмування / Т. В. Ковалюк. – Київ: Видавнича група BHV, 2005. – 384 с.
4. http://bit.ly/2sesmsY
5. http://bit.ly/2rgtPBQ
6. http://bit.ly/2rdS7uc
7. http://bit.ly/2rdmRLC
8. http://bit.ly/2seAQjR

ДОДАТКИ

```
#include <iostream>
#include <limits.h>
#include <string.h>  // Видалити '.h' для vs
#include <queue>
// #include <windows.h>
using namespace std;

// Кількість вершин в даному графі
#define V 6

/* Повертає true якщо є шлях від 's' до 't' в
залишковий граф. Також заповнює parent[] щоб зберегти шлях */
bool bfs(int rGraph[V][V], int s, int t, int parent[])
{
    // Створити масив відвіданих і позначити всі вершини які не відвідані
    bool visited[V];
    memset(visited, 0, sizeof(visited));

    // Створити чергу, поставлене в чергу джерело вершини і мітки джерела вершини як відвідані
    queue <int> q;
    q.push(s);
    visited[s] = true;
    parent[s] = -1;

    // Пошук в ширину
    while (!q.empty())
    {
        int u = q.front();
        q.pop();

        for (int v = 0; v<V; v++)
```

```
            {
                if (!visited[v] && rGraph[u][v] > 0)
                {
                    q.push(v);
                    parent[v] = u;
                    visited[v] = true;
                }
            }
        }

        return visited[t];
    }

    // Повертає максимальний потік від s до t в даному графіку
    int fordFulkerson(int graph[V][V], int s, int t)
    {
        int u, v;

        // Створення залишкового графа і заповнити залишковий граф з
        // з урахуванням потенціалу в якості вихідного графа залишкових
потужностей
        // в залишковому графі
        int rGraph[V][V]; // Залишковий граф де rGraph[i][j] вказує
        // залишкову ємність ребра від i до j (якщо існує
        // ребро. Якщо rGraph[i][j] дорівнює 0, то немає)
        for (u = 0; u < V; u++)
            for (v = 0; v < V; v++)
                rGraph[u][v] = graph[u][v];

        int parent[V]; // Цей масив заповнюється пошуком в ширину і
зберігає шлях

        int max_flow = 0;

        // Збільшити потік поки йде шлях від джерела до приймача
```

```
    while (bfs(rGraph, s, t, parent))
    {
        // Знайти максимальний потік через знайдений шлях
        int path_flow = INT_MAX;
        for (v = t; v != s; v = parent[v])
        {
            u = parent[v];
            path_flow = min(path_flow, rGraph[u][v]);
        }

        // Оновити залишкові можливості ребер і
        // зворотний край уздовж шляху
        for (v = t; v != s; v = parent[v])
        {
            u = parent[v];
            rGraph[u][v] -= path_flow;
            rGraph[v][u] += path_flow;
        }

        // Додати потік шляху до загального потоку
        max_flow += path_flow;
    }

    // Повертає загальний потік
    return max_flow;
}

// Запуск функцій вище
int main()
{
    // Українська мова для Visual Studio
    // SetConsoleCP(1251);
    // SetConsoleOutputCP(1251);

    cout << "                                                    _\n"
```

```
                "  ___ ___  _   _ _ __ ___  _____      _____  _ __| |
__\n"
                " / __/ _ \\| | | | '__/ __|/ _ \\ \\ /\\ / / _ \\|
'__| |/ /\n"
                "| (_| (_) | |_| | |  \\__ \\  __/\\ V  V / (_) | |  |
<\n"
                " \\___\\___/ \\__,_|_|  |___/\\___| \\_/\\_/ \\___/|_|
|_|\\_\\"
             << endl ;
        cout << string(47, ' ') + "@vovawed" << endl << endl;

        cout << "1) Інформація про курсовий проект" << endl
             << "2) Запустити виконання програми" << endl
             << "3) Вихід" << endl;

        short num;

        do {
            cout << "Введіть число: ";
            cin >> num;
            switch (num) {
                case 1:
                    cout  <<  "Знаходження  максимального  потоку  за
методом Форда-Фалкерсона" << endl
                         << "Підготував: Пицюк Володимир" << endl
                         << "Надвірнянський коледж НТУ" << endl
                         << endl;
                case 2:
                    break;
                case 3:
                    cout << "Вихід" << endl;
                    return 0;
                default:
                    cout << "Не вірний ввід, спробуйте ще раз" << endl;
            }
```

```
        } while (num != 2);

        // Матриця
        int graph[V][V] = {
                { 0, 16, 13, 0, 0, 0 },
                { 0, 0, 10, 12, 0, 0 },
                { 0, 4, 0, 0, 14, 0 },
                { 0, 0, 9, 0, 0, 20 },
                { 0, 0, 0, 7, 0, 4 },
                { 0, 0, 0, 0, 0, 0 }
        };

        cout << "Максимально можливий потік: " << fordFulkerson(graph,
0, 5) << endl;

        // system("pause"); Для Visual Studio

        return 0;
    }
```