МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Кафедра прикладної математики та фундаментальних наук

Звіт

до індивідуальної роботи №1

з дисципліни «Паралельні обчислення та засоби їх реалізації»

**Виконав:**

студент групи КНІК-11

Мельніков В. В.

**Прийняв:**

Яджак М.С.

Львів 2022

**Завдання**

На підставі наведеного вище алгоритму для виявлення ярусів ЯПФ написати програму та реалізувати її на ПК (Індивідуальне завдання № 1).

**Теоретичні відомості**

**Будь-який граф** можна подати **матрицею суміжності**. Будь-яку **квадратну матрицю**, що складається із нулів та одиничок, можна розглядати, як **матрицю суміжності** деякого **графа** і за нею будувати відповідний граф. Інформацію про граф з допомогою матриці суміжності можна зберігати без зображення графа, зокрема і в пам’яті ПК.

Наведемо алгоритм (послідовність кроків) з виявлення у ГА вершин, які можуть виконуватися паралельно, тобто можуть знаходитись в одному ярусі ЯПФ.

**Крок 1:** номери вхідних вершин занести в масив S1.

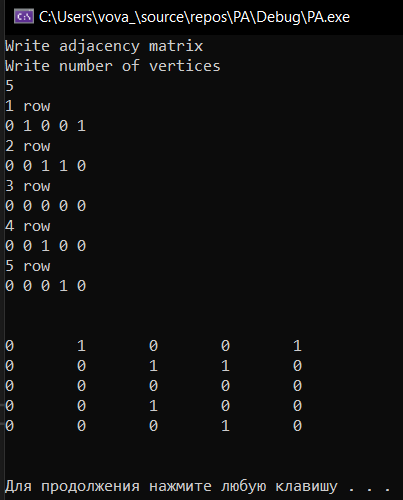
**Крок 2:** знайти вершини, залежні (за ребрами) від вершин, які входять лише в масив S1; занести знайдені вершини в деякий масив S2.

**Крок 3:** якщо масив S2 містить вершини, то додаємо ці вершини в масив S1 і переходимо на крок 2, інакше – закінчуємо роботу.

На **кроці 1** початкові дані (n вершин ГА та його матриця суміжності) для роботи задаються **вручну. Крок 2** – найбільш трудоємкий і вимагає перегляду матриці суміжності для кожного елементу масиву S1. На **кроці 3** здійснюється цикл по ярусах до тих пір, поки вони будуть виявлені. **Результатами** цієї процедури є **кількість ярусів** та **номери вершин**, розташованих у кожному з них.

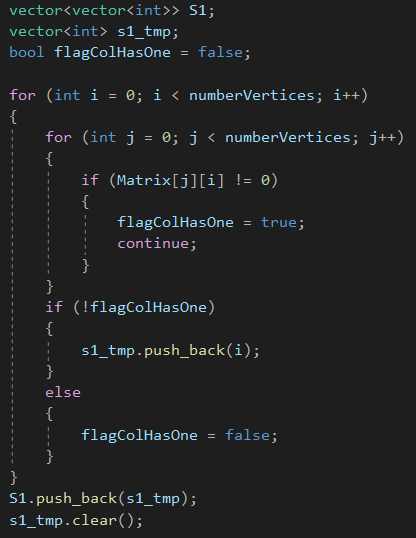
# Хід роботи

1. Створив відповідний код для вводу розмірності матриці суміжності і власне наступного її вводу, а також виводу для користувача



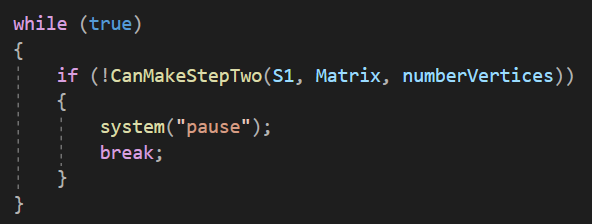
*Рис. 1. Приклад введення матриці суміжності та її подальший вивід*

1. Відповідно до алгоритму наведеного в теоретичних відомостях реалізував **крок 1**, а саме внесення вхідних вершин в масив через такий код:

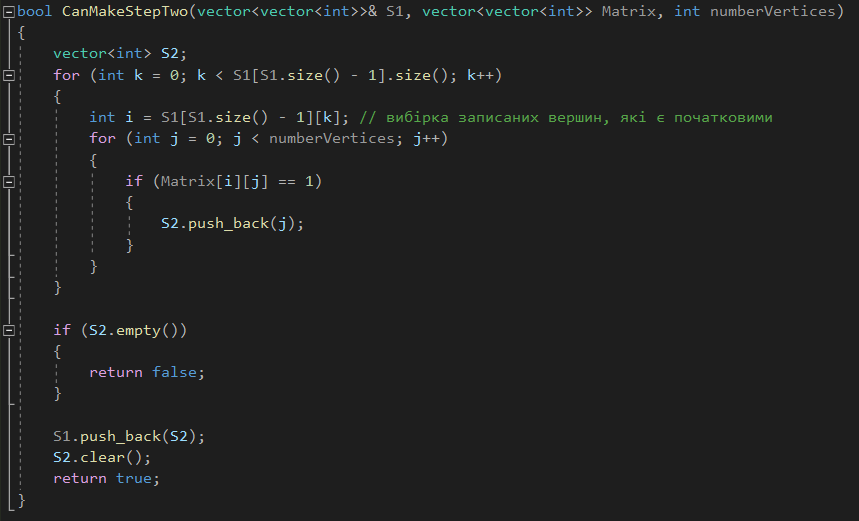


*Рис. 2. Код який реалізує внесення початкових вершин в масив*

1. Наступним кроком створив цикл, який виконує **крок 2**, а саме знаходження вершини, залежних (за ребрами) від вершин, які входять лише в масив S1; занесення знайдених вершини в деякий масив S2.

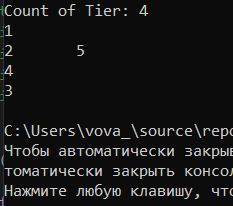


*Рис. 3. Код циклічного виклику функції, що реалізує крок 2*



*Рис. 4. Код кроку 2*

1. Дальше, якщо умова **кроку 3**, а саме, якщо масив S2 містить вершини, то додаємо ці вершини в масив S1 і переходимо на **крок 2**, інакше – закінчуємо роботу, то виводимо яруси і відповідні вершини на кожному з них



*Рис. 5. Вивід кількості ярусів і вершин на кожному з них*

**Додаток 1**

#include <iostream>

#include <vector>

#include <string>

**using** **namespace** std;

vector <**int**> StrToArrInt(string str);

**bool** **IsBadInMatrix**(vector<vector<**int**>> checkedMatrix, **int** numberVert);

**bool** **CanMakeStepTwo**(vector<vector<**int**>>& S1, vector<vector<**int**>> Matrix, **int** numberVertices);

**void** **PrintResult**(vector<vector<**int**>> S1);

**int** **main**()

{

**int** numberVertices = **4**;

vector<vector<**int**>> Matrix;

**while** (true) //змінити на false, якщо треба використати шаблону матрицю

{

cout << "Write adjacency matrix**\n**";

cout << "Write number of vertices**\n**";

cin >> numberVertices;

cin.ignore();

**if** (!(numberVertices > **0** && numberVertices < **1000**))

{

cout << "Incorrect data entered**\n**";

system("pause");

system("cls");

**continue**;

}

**for** (**int** i = **0**; i < numberVertices; i++)

{

string matrixRow;

cout << **int**(i + **1**);

cout << " row" << endl;

getline(cin, matrixRow);

Matrix.push\_back(StrToArrInt(matrixRow));

}

**if** (IsBadInMatrix(Matrix, numberVertices)) {

system("pause");

system("cls");

**continue**;

}

**break**;

}

/\*Matrix.push\_back(vector<int> {0, 1, 0, 0, 1 });

Matrix.push\_back(vector<int> {0, 0, 1, 1, 0 });

Matrix.push\_back(vector<int> {0, 0, 0, 0, 0 });

Matrix.push\_back(vector<int> {0, 0, 1, 0, 0 });

Matrix.push\_back(vector<int> {0, 0, 0, 1, 0 });\*/

cout << "**\n\n**";

**for** (**int** i = **0**; i < numberVertices; i++)

{

**for** (**int** j = **0**; j < numberVertices; j++)

{

cout << Matrix[i][j] << "**\t**";

}

cout << endl;

}

cout << "**\n\n**";

vector<vector<**int**>> S1;

vector<**int**> s1\_tmp;

**bool** flagColHasOne = false;

**for** (**int** i = **0**; i < numberVertices; i++)

{

**for** (**int** j = **0**; j < numberVertices; j++)

{

**if** (Matrix[j][i] != **0**)

{

flagColHasOne = true;

**continue**;

}

}

**if** (!flagColHasOne)

{

s1\_tmp.push\_back(i);

}

**else**

{

flagColHasOne = false;

}

}

S1.push\_back(s1\_tmp);

s1\_tmp.clear();

**while** (true)

{

**if** (!CanMakeStepTwo(S1, Matrix, numberVertices))

{

system("pause");

**break**;

}

}

//PrintResult(S1);

DeleteDuplicates(S1, numberVertices);

PrintResult(S1);

}

vector <**int**> StrToArrInt(string str)

{

string strtemp = "";

vector<**int**> numberInRow;

**for** (**int** i = **0**; i < str.size()+**1**; i++)

{

**if** (str[i] != ' ' && str[i] != '\0')

{

strtemp += str[i];

}

**else**

{

**if** (strtemp.empty()) **continue**;

numberInRow.push\_back(stoi(strtemp));

strtemp = "";

}

}

**return** numberInRow;

}

**bool** IsBadInMatrix(vector<vector<**int**>> checkedMatrix, **int** numberVert)

{

**for** (**int** i = **0**; i < checkedMatrix.size(); i++)

{

**if** (checkedMatrix.size() != checkedMatrix[i].size() || checkedMatrix.size() != numberVert)

{

cout << "Error in Dimensionality Of Matrix**\n**";

**return** true;

}

**for** (**int** j = **0**; j < checkedMatrix[i].size(); j++)

{

**if** (!(checkedMatrix[i][j] == **0** || checkedMatrix[i][j] == **1**))

{

cout << "Error, can be only 1 or 0 numbers**\n**";

**return** true;

}

}

}

**return** false;

}

**bool** CanMakeStepTwo(vector<vector<**int**>> &S1, vector<vector<**int**>> Matrix, **int** numberVertices)

{

vector<**int**> S2;

**for** (**int** k = **0**; k < S1[S1.size() - **1**].size(); k++)

{

**int** i = S1[S1.size() - **1**][k]; // вибірка записаних вершин, які є початковими

**for** (**int** j = **0**; j < numberVertices; j++)

{

**if** (Matrix[i][j] == **1**)

{

S2.push\_back(j);

}

}

}

**if**(S2.empty())

{

**return** false;

}

S1.push\_back(S2);

S2.clear();

**return** true;

}

**void** PrintResult(vector<vector<**int**>> S1)

{

cout << "**\n**Count of Tier: " << S1.size() << endl;

**for** (**int** i = **0**; i < S1.size(); i++)

{

**for** (**int** j = **0**; j < S1[i].size(); j++)

{

cout << (S1[i][j] + **1**) << "**\t**";

}

cout << endl;

}

}