**Міністерство освіти і науки України**

**Національний університет "Львівська політехніка"**

**Кафедра ЕОМ**

****

**Звіт з лабораторної роботи №3**

**з дисципліни “Паралельні та розподілені обчислення ”**

**Варіант 1**

**Виконав: студент .гр. КІ-33**

**Барсієнко В.О.**

**Прийняв: асистент**

**Козак Н.Б.**

**Львів 2020 р.**

**Мета:** Дослідити можливості розв’язання різноманітних задач за допомогою паралельних алгоритмів. Навчитися виділяти незалежні гілки обчислень та виконувати їх паралельно.

**Завдання:**

Графом називається сукупність точок (вузлів), деякі з яких з’єднані між собою направленими ребрами. Граф, що складається з *n* вузлів можна описати двома матрицями порядку *n*: матрицею з’єднань та матрицею зв’язків. Елемент матриці з’єднань  якщо граф містить ребро направлене від вузла i до вузла j та  в іншому випадку. Елемент матриці зв’язків  якщо з вузла *i* можна попасти у вузол *j*, рухаючись по ребрах і  в іншому випадку.

Завдання: ввести кількість вузлів деякого графу. Задавши довільним чином матрицю з’єднань(в інтерактивному режимі або випадковим чином), побудувати матрицю зв’язків для цього графу. Передбачити графічне відображення такого графу та числовий вивід обох матриць.

**Код програми:**

#include <iostream>

#include <vector>

#include <ctime>

#include <list>

using namespace std;

//при перевірці чи можна попасти з точки А в точку А, не залежно від наявності петлі - відповідь ТАК

class Graph{

    int V;

    list<int>\* adj;

public:

    Graph(int V);

    void addEdge(int v, int w);

    bool isReachable(int s, int d);

};

Graph::Graph(int V){

    this->V = V;

    adj = new list<int>[V];

}

void Graph::addEdge(int v, int w){

    adj[v].push\_back(w);

}

bool Graph::isReachable(int s, int d){

    if (s == d) { return true; }

    bool\* visited = new bool[V];

    for (int i = 0; i < V; i++) { visited[i] = false; }

    list<int> queue;

    visited[s] = true;

    queue.push\_back(s);

    list<int>::iterator i;

    while (!queue.empty()){

        s = queue.front();

        queue.pop\_front();

        for (i = adj[s].begin(); i != adj[s].end(); ++i){

            if (\*i == d) { return true; }

            if (!visited[\*i]) {

                visited[\*i] = true;

                queue.push\_back(\*i);

            }

        }

    }

    return false;

}

void matrixOut(int n, vector<vector<int>> &A, vector<vector<int>> &B) {

int i, j;

cout << endl << "Matrix A" << endl;

for (i = 0; i < n; i++) {

for (j = 0; j < n; j++) {

cout << A[i][j] << "\t";

}

cout << endl;

}

cout << endl << "Matrix B:" << endl;

for (i = 0; i < n; i++) {

for (j = 0; j < n; j++) {

cout << B[i][j] << "\t";

}

cout << endl;

}

cout << endl;

}

void randGraph(int n, vector<vector<int>> &A, Graph g) {

for (int i = 0; i < n; i++) { for (int j = 0; j < n; j++) { if (A[i][j] == 1) { g.addEdge(j, i); } } }

}

void checkReachabillity(int n, vector<vector<int>> &B, Graph g) {

for (int i = 0; i < n; i++) { for (int j = 0; j < n; j++) { if (g.isReachable(j, i)) { B[i][j] = 1; } } }

}

int main() {

srand(time(NULL));

int i, j, n, choice;

cout << "Enter vertices amount(n): "; cin >> n;

Graph g(n);

vector<vector<int>> A(n, vector<int>(n)), B(n, vector<int>(n));

fail:

cout << "Fill matrix of edges(1 - manually, 2 - randomly): "; cin >> choice;

if (choice == 1) { for (i = 0; i < n; i++) { for (j = 0; j < n; j++) { cout << "A[" << i << "][" << j << "]= "; cin >> A[i][j]; } } cout << endl; }

else if (choice == 2) { for (i = 0; i < n; i++) { for (j = 0; j < n; j++) { A[i][j] = rand() % 2; } } }

else { cout << "Wrong choice, please try again!" << endl << endl; goto fail; }

randGraph(n, A, g);

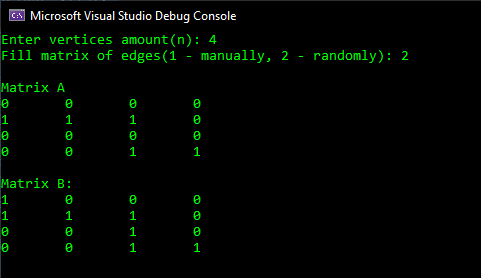
checkReachabillity(n, B, g);

matrixOut(n, A, B);

return 0;

}

**Результат виконання програми:**



**Висновок:**

На цій лабораторній роботі я дослідив можливості розв’язання різноманітних задач за допомогою паралельних алгоритмів та навчився виділяти незалежні гілки обчислень та виконувати їх паралельно.