Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра ЕОМ



**Звіт**

з лабораторної роботи №3

з дисципліни: “ Паралельні та розподілені обчислення”

на тему:

“ **МОЖЛИВОСТІ ВИКОРИСТАННЯ ПАРАЛЕЛЬНИХ АЛГОРИТМІВ.**”

Виконав: ст. гр. КІ-34

Гриценко В.В.

Прийняв: викл.

Козак Н.Б.

Львів

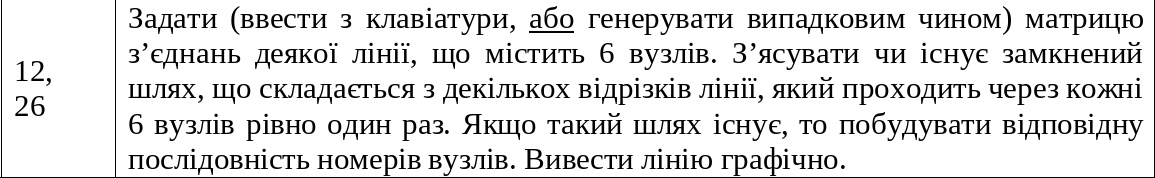
2020

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

**Завдання:**

3,

17

Рис. 1 Завдання до лабораторної роботи. Варіант №3

**Аналіз**

Маємо:

- матрицю з“єднань X[6][n] (6 вузлів, n ребер).

З“ясувати:

- чи існує замкнений шлях, що проходить через усі 6 вузлів один раз?

Алгоритм вирішення задачі:

1. Отримання вхідної матриці зʼєднань, при заданих вручну к-сті відрізків для лінії.
2. Перетворення матриці зʼєднань в матрицю вигляду:

1| 2, 3, 6

2| 1, . . .

. . .

х| . . .

де «х|» - індекс вузла, а «| . . . » - масив вузлів зʼєднаних вузлом «х»

1. Рекурсивний алгоритм проходження графа, для поданої матриці (С++):

* nodes — матриця з п.2
* index — вершина з якої починається алгоритм (у прикладній програмі обрано 0)
* result — результуючий масив індексів вузлів у правильній послідовності.

bool getCycleList(

vector<vector<int>>& nodes,

int index,

vector<int>& result)

{

// if we already have this node in result

if(find(result.begin(), result.end(), index) != result.end())

return false;

result.push\_back(index);

// There is no cycle if node have <2 connects

if(nodes[index].size() < 2)

return false;

// END:

// size should be NODES\_COUNT

// index should have connection to the 0th element

if(result.size() == NODES\_COUNT &&

find(nodes[index].begin(), nodes[index].end(), 0) != result.end()

)

{

return true;

}

for(int el : nodes[index])

if(getCycleList(nodes, el, result))

return true;

result.pop\_back();

return false;

}

**Текст програми:**

#include <iostream>

#include <iomanip>

#include <sstream>

#include <vector>

#include <algorithm>

using namespace std;

const int NODES\_COUNT = 6;

typedef vector<vector<bool>> con\_matrix;

typedef vector<vector<int>> nodes\_map; // vectors with sublines

bool getCycleList(

vector<vector<int>>& nodes,

int index,

vector<int>& result)

{

// if we already have this node in result

if(find(result.begin(), result.end(), index) != result.end())

return false;

result.push\_back(index);

// There is no cycle if node have <2 connects

if(nodes[index].size() < 2)

return false;

// END:

// size should be NODES\_COUNT

// index should have connection to the 0th element

if(result.size() == NODES\_COUNT &&

find(nodes[index].begin(), nodes[index].end(), 0) != result.end()

)

{

return true;

}

for(int el : nodes[index])

if(getCycleList(nodes, el, result))

return true;

result.pop\_back();

return false;

}

int main()

{

int n;

cout << "Enter please count of the line segments (branches in connection matrix): " << endl;

cin >> n;

con\_matrix X(NODES\_COUNT, vector<bool>(n, 0));

// Random line (connection matrix) generating with

// NODES\_COUNT nodes and n segments

srand(time(nullptr));

int node1, node2;

bool notReady;

for (int i = 0; i < n; ++i) {

do

{

notReady = false;

node1 = rand() % NODES\_COUNT;

node2 = rand() % NODES\_COUNT;

if(node1 == node2)

node2 = (node1 + 1) % 6;

// Check if segment already exists

for (int j = 0; j < i; ++j)

if (X[node1][j] && X[node2][j])

notReady = true;

} while(notReady);

X[node1][i] = true;

X[node2][i] = true;

}

cout << "We got the connection matrix: " << endl << endl;

cout << " ";

for (int i = 0; i < n; ++i)

cout << setw(3) << i + 1;

cout << endl << endl;

for(int i = 0; i < NODES\_COUNT; ++i)

{

cout << i + 1 << " ";

for(int j = 0; j < n; ++j)

cout << setw(3) << X[i][j];

cout << endl;

}

// matrix with index - node, int elem - another node connected to index-node

vector<vector<int>> nodes(NODES\_COUNT);

int temp;

for(int j = 0; j < n; ++j)

{

temp = -1;

for(int i = 0; i < NODES\_COUNT; ++i)

{

if(X[i][j] == 1 && temp == -1)

temp = i;

else if (X[i][j] == 1 && temp != -1) {

nodes[i].push\_back(temp);

nodes[temp].push\_back(i);

}

}

}

// CALCULATE CYCLE LIST OF NODES

vector<int> result;

if(getCycleList(nodes, 0, result))

{

cout << endl << "Ahaha, we got a cycle list!!!" << endl

<< "Our cycle: ";

for(int el : result)

cout << el + 1 << " ";

cout << result[0] + 1 << endl << endl;

}

else

cout << endl << "Our line doesn't have a cycle through all the nodes." << endl << endl;

return(0);

}

**Результат виконання програми:**

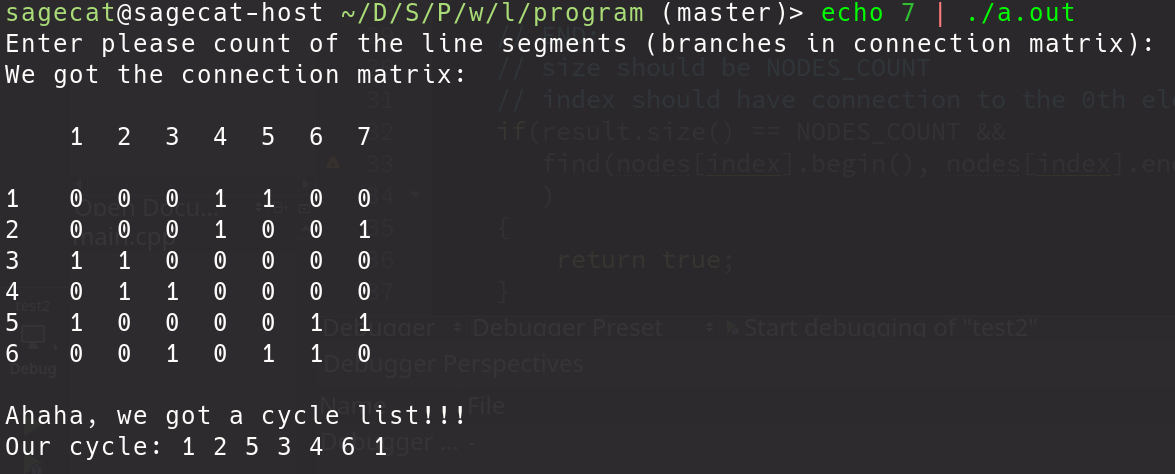


Рис 2. Результат роботи програми

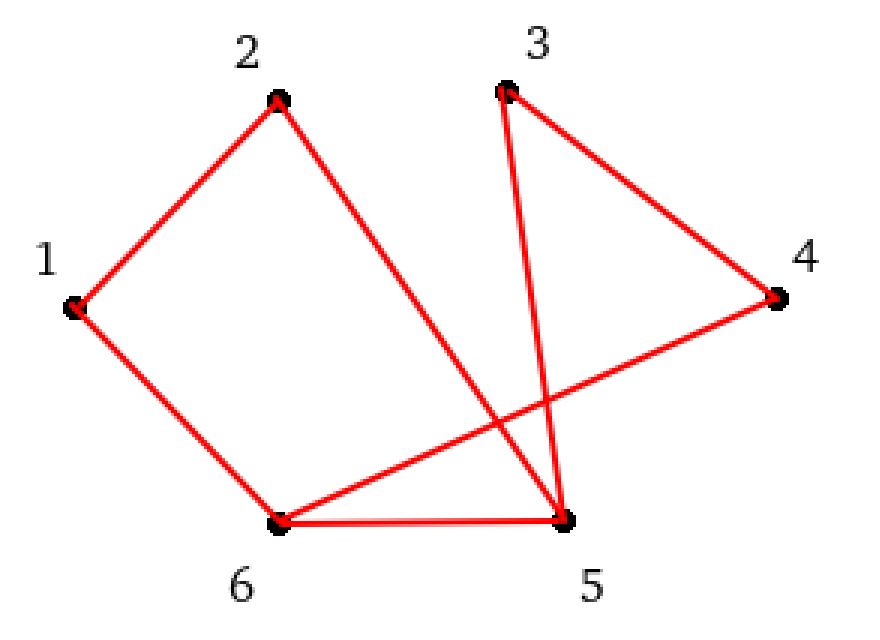
****

Рис 3. Графічне представлення

**Висновки:** під час виконання даної лабораторної роботи я дослідив можливості розв’язання різноманітних задач за допомогою паралельних алгоритмів та навчився виділяти незалежні гілки обчислень та виконувати їх паралельно.