**Міністерство освіти і науки України**

**Національний університет «Львівська політехніка»**

**Кафедра ЕОМ**



Звіт

до лабораторної роботи № 3

з дисципліни «Паралельні та розподілені обчислення»

на тему: «Можливості Використання Паралельних Алгоритмів»

Виконав:

ст.гр. КІ-34

Мороз О.Ю.

Прийняв:

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:**

1. Проаналізувати завдання і виділити в ньому незалежні гілки обчислень(якщо це можливо). Використати метод розбиття задачі на підзадачі та виділення незалежних подій.
2. Реалізувати кожну гілку окремо.
3. Об’єднати всі частини для вирішення поставленої задачі і переконатися у правильності реалізації.
4. Оцінити обчислювальні та часові затрати створеної програми (тобто як зростає час виконання при збільшенні розмірності задачі)

Варіант №2:

|  |  |
| --- | --- |
| 2,  16 | Лінія називається унікурсальною, якщо її можна накреслити не відриваючи перо від паперу та не проходячи два рази одне і теж ребро. (Зауважимо, що лінія є унікурсальною лише тоді, коли кількість тих вузлів з яких виходить непарна кількість ребер не більше двох). Лінію, що містить *n* вузлів можна задати квадратною матрицею з’єднань,(порядку *n*), в якій елемент  якщо вузол *i* з’єднаний з вузлом *j* ребром, що не містить інших вузлів.  Завдання: ввести кількість вузлів деякої лінії. Задавши довільним чином матрицю з’єднань(в інтерактивному режимі або випадковим чином), визначити чи є така ліні унікурсальною і якщо є, то отримати послідовність номерів вузлів, які будуть пройдені для викреслювання лінії. |

**Алгоритм вирішення задачі:**

* Оберіть будь-яку стартову вершину *v* і слідуйте ланцюгом ребер починаючи з цієї вершини допоки не повернетесь в *v*. Неможливо застрягнути в будь-якій вершині окрім *v*, бо парна степінь кожної вершини гарантує, що коли ланцюг досягає іншої вершини *w*, то мусить існувати невикористане ребро з *w*. Ланцюг сформований каим чином замкнений, тобто цикл, але може не покривати всіх ребер початкового графа.
* Допоки існує вершина *u*, яка не належить до поточного ланцюга, але має інцидентні ребра не в ланцюзі, почніть інший ланцюг з *u*, слідуючи невикористаними ребрами допоки не повернетесь в *u* і приднайте новий цикл до вже наявного.

Використання такої структури як двобічно зв'язаний список уможливлює виконання кожної операції за константний час (знаходження невикористаних ребер для кожної вершини, знаходження нової стартової вершини і поєднання двох циклів зі спільною вершиною), таким чином весь алгоритм потребує лінійний час, {\displaystyle O(|E|)}.

**Код програми:**

#include <iostream>

#include <cstdlib>

struct Node

{

int inf;

Node \*next;

};

//============================Stack==============================

void push(Node \*&st, int dat)

{ // Завантаження числа в стек

Node \*el = new Node;

el->inf = dat;

el->next = st;

st = el;

}

int pop(Node \*&st)

{ // Вилучення з стеку

int value = st->inf;

Node \*temp = st;

st = st->next;

delete temp;

return value;

}

int peek(Node \*st)

{ // Отримання числа без його вилучення

return st->inf;

}

//==============================================================

Node \*\*graph; // Масив списків суміжности

const int vertex = 1; // Перша вершина

void add(Node\*& list, int data)

{ //Створення суміжної вершини

if (!list) { list = new Node; list->inf = data; list->next = 0; return; }

Node \*temp = list;

while (temp->next)temp = temp->next;

Node \*elem = new Node;

elem->inf = data;

elem->next = NULL;

temp->next = elem;

}

void del(Node\* &l, int key)

{ // Видалення вершини key з списку

if (l->inf == key) { Node \*tmp = l; l = l->next; delete tmp; }

else

{

Node \*tmp = l;

while (tmp)

{

if (tmp->next) // є наступна вершина

if (tmp->next->inf == key)

{ // і вона шукана

Node \*tmp2 = tmp->next;

tmp->next = tmp->next->next;

delete tmp2;

}

tmp = tmp->next;

}

}

}

int eiler(Node \*\*gr, int num)

{ // Визначення чи граф эйлерів

int count;

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

{ //проходимо всі вершини

count = 0;

Node \*tmp = gr[i];

while (tmp)

{ // рахуємо степінь

count++;

tmp = tmp->next;

}

if (count % 2 == 1)return 0; // степінь непарна

}

return 1; // всі степені парні

}

void eiler\_path(Node \*\*gr)

{ //Побудова циклу

Node \*S = NULL;// Стек для пройдених вершин

int v = vertex;// 1 вершина (довільна)

int u;

push(S, v); //зберігаємо її в стек

while (S)

{ //до тих пір поки стек не пустий

v = peek(S); // поточна вершина

if (!gr[v]) { // якщо немає інцидентних ребер

v = pop(S);

std::cout << v + 1 << " "; //виводимо вершину (у нас відлік від 1, тому +1)

}

else

{

u = gr[v]->inf; push(S, u); //проходимо в наступну вершину

del(gr[v], u); del(gr[u], v); //видаляемо пройдене ребро

}

}

}

int main()

{

setlocale(LC\_ALL, "Ukr");

system("CLS");

std::cout << "Кiлькiсть вершин: ";

int n;

std::cin >> n;

int zn;// Поточне значення

graph = new Node\*[n];

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

graph[i] = NULL;

for (int i = 0; i<n; i++) // заповнюємо масив списків

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

{

std::cin >> zn;

if (zn) add(graph[i], j);

}

std::cout << "\n\nРЕЗУЛЬТАТ ";

if (eiler(graph, n))

eiler\_path(graph);

else

std::cout << "Граф не є эйлерiв.";

std::cout << std::endl;

std::cin.get();

std::cin.get();

return(0);

}

**Результати виконання програми:**

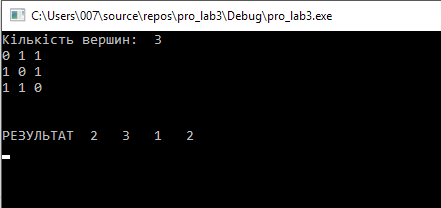


Рис.1. Результат виконання програми

**Висновок:** Дослідив можливості розв’язання різноманітних задач за допомогою паралельних алгоритмів. Навчився виділяти незалежні гілки обчислень та виконувати їх паралельно.