**ЛАБОРАТОРНА РОБОТА #4**

З дисципліни «Системне програмне забезпечення»

На тему «Дослідження принципів проектування та роботи динамічного та статичного планування»

|  |  |
| --- | --- |
| **Виконав**  Студент групи КВ-21  Комарницький О.Б. | **Перевірив**  Проф. Симоненко В.П. |

|  |  |  |
| --- | --- | --- |
| **Оцінка** | **Дата** | **Підпис** |
|  |  |  |

**Варіант #9**

**Завдання:**

Написати програму, що реалізує один із алгоритмів динамічного або статичного плування, відповідно до варіанту.

|  |  |
| --- | --- |
| **Варіант #** | **Алгоритм** |
| 9 | Визначити часову складність алгоритму виділення всіх критичних вершин для заданого зваженого ациклічного графа (число вершин не менше 30). |

**Теоретичні відомості і опис алгоритму**

Необхідно знайти критичні вершини графа. Критичні вершини – всі вершини, що входять в критичний шлях. Тому опишемо алгоритм пошуку критичного шляху.

Шлях максимальної довжини є сенс шукати тільки в графах, де немає контурів. В графі з контурами рішення однозначне і дорівнює нескінченності, бо існує шлях, який нескінченну кількість разів проходить по цьому контуру. Одним із завдань, де використовується пошук максимального шляху, є задача аналізу мережного графіка. Дамо необхідні визначення.

Семантика завдання. Зіставимо кожній дузі роботу. Вага дуги - час виконання роботи.

Вершині зіставимо подію, що полягає в тому, що всі роботи (вхідні в неї дуги), виконані, і можна починати всі роботи, що відповідають вихідним дугам. Граф з таким трактуванням називається мережевим графіком.

У мережевому графіку максимальному шляху (критичному), буде відповідати мінімальний час, необхідний для виконання всіх робіт. Для скорочення загального часу можна розглядати, наприклад, питання про автоматизацію робіт на критичному шляху з метою скорочення часу їх виконання.

Для вирішення задачі пошуку максимального шляху можна скористатися алгоритмом Дейкстри, змінивши його відповідним чином.

Однак є додаткова умова на граф задач: в ньому немає контурів.

Додаткові умови можуть ускладнити алгоритм, але можна використовувати ці умови і для скорочення алгоритму, що ми і зробимо.

1. Привласнимо всіх вершин вага l (i) = 0.

2. Послідовно для вершин 1, 2, 3, ... проведемо перерахунок ваг за формулою l (i) = max (l (i), l (j) + Cji), де максимум береться по всіх вершин j, з яких є дуги у вершину i. Це можна зробити, оскільки до часу перерахунку вершини i ваги всіх вершин j обчислені раніше, бо i> j.

Скорочення обчислювальних витрат в порівнянні з алгоритмом Дейкстри пов'язано з тим, що відпадає необхідність на кожному кроці визначати чергову вершину з мінімальною вагою для продовження розрахунків.

3. Як і в алгоритмі Дейкстри, виділимо зворотним ходом шуканий шлях.

**Визначення тимчасової складності алгоритму**

Складність алгоритму: O(n2)

**Час виконання**

**Розмірність матриці**

**Висновок:**

На графіку помітний квадратичний приріст часу виконання задачі, як це визначено у складності алгоритму. Неоднорідність отриманих значень пояснюється пошуком першої критичної вершини із кінця та кількістю критичних вершин у графі: чим більше критичних вершин знайдено, тим менше відбудеться зайвих перевірок, адже при знаходженні критичної вершини пошук в даній вершині завершується.

**Лістинг основної частини програми мовою С++**

***Файл Interface.h***

#include <iostream>

struct Setting

{

const int EDGES\_INTENSITY = 50;

const int MAX\_WEIGHT = 20;

};

void init(unsigned int dim);

void calc();

void output\_matr();

void print\_res();

***Файл Interface.cpp***

#include "Interface.h"

#include <vector>

using namespace std;

vector<vector<unsigned int>> Matrix;

vector<unsigned int> Vertices;

vector<unsigned int> Result;

Setting setting;

void init(unsigned int dim)

{

Matrix = vector<vector<unsigned int>>(dim);

for (unsigned int i = 0; i < dim; i++)

{

Matrix[i] = vector<unsigned int>(dim);

for (unsigned int j = i + 1; j < dim; j++)

{

if (rand() % 100 <= setting.EDGES\_INTENSITY)

Matrix[i][j] = (unsigned) (rand() % setting.MAX\_WEIGHT + 1);

else

Matrix[i][j] = 0;

}

}

Vertices = vector<unsigned int>(dim);

}

void calc()

{

// Make null vertices values

for (unsigned int i = 0; i < Vertices.size(); i++)

Vertices[i] = 0;

// Calculate new vertices values

for (unsigned int i = 0; i < Matrix.size(); i++)

{

unsigned int max = 0;

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

if (Matrix[j][i] && Vertices[j] + Matrix[j][i] > max)

max = Vertices[j] + Matrix[j][i];

Vertices[i] = max;

}

/\* Find critical way \*/

// Find start vertice

int start\_point = -1;

for (unsigned long i = Vertices.size() - 1; i >= 0; i--)

{

int j = i - 1;

while(j >= 0 && Vertices[i] != Vertices[j] + Matrix[j][i] | !Matrix[j][i])

j--;

if (Vertices[i] == Vertices[j] + Matrix[j][i])

{

start\_point = i;

break;

}

}

if (!start\_point) return;

Result.push\_back(start\_point);

while(start\_point > 0)

{

int j = start\_point - 1;

while (j >= 0 && Vertices[start\_point] != Vertices[j] + Matrix[j][start\_point] | !Matrix[j][start\_point])

j--;

start\_point = j;

if (start\_point >= 0)

Result.push\_back(start\_point);

}

}

***Файл Main.cpp***

#include "Interface.h"

using namespace std;

int main(int argc, char \* argv[])

{

unsigned int dim;

if (argc > 1)

dim = (unsigned int)stoi(argv[1]);

else

{

cout << "Dim-> " << endl;

cin >> dim;

}

init(dim);

time\_t time = clock();

calc();

cout << "Time taken: " << double(clock() - time) / CLOCKS\_PER\_SEC << endl;

return 0;

}