Министерство образования и науки Российской Федерации

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | | | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования | | | | | |
| «Алтайский государственный технический университет им. И.И. Ползунова» | | | | | |
| Факультет информационных технологий  Кафедра "Прикладная математика" | | | | | |
|  | | | Отчет защищен с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | |
|  | | | Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А.В. Астахова  Подпись | | |
|  | | | «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2017 г. | | |
|  | | |  | | |
| **ОТЧЕТ**  **ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2** | | | | | |
|  | | | | | |
| по дисциплине «ОСНОВЫ ЭКОНОМИКИ ПРОГРАММНОЙ ИНЖЕНЕРИИ» | | | | | |
| **ЛР 09.03.04.9.001 О** | | | | | |
| Студент группы ПИ-31 И.О.Ильиных | | | | | |
|  |  |  | | подпись | и.о.,фамилия |
| Преподаватель профессор А.В. Астахова | | | | | |
| должность, ученая степень подпись | | | | | и.о.,фамилия |

БАРНАУЛ 2017

Постановка задачи

Рассчитать параметры событий и параметры работ сетевого графика.

Составит программу, которая позволит: рассчитывать параметры событий начальной вершины, рассчитывать свободный и полный резерв времени работ, определять топологию и длину критических путей, обнаруживать и удалять циклы и петли в графе.

Алгоритм решения

Расчёт сетевого графика заключается в установлении ранних и поздних начал и окончаний работ, определении общих и частных резервов времени и определении критического пути.

Наиболее удобным способом расчёта сетевого графика является расчёт непосредственно на сети графика.

Для расчёта параметров сетевого графика события обозначают в виде круга, разделённого на четыре сектора. В верхнем секторе помещают порядковый номер события; в нижнем секторе –резерв наступления события. В левый сектор круга заносят значение раннего начала последующей работы, а в правый – раннее окончание предшествующей работы.

Расчёт графика начинают с заполнения левых секторов событий по направлению движения стрелок от начального (исходного) события до завершающего. Предварительно каждому событию должен быть присвоен свой номер.

Если в одно событие входит несколько работ, то в левый сектор этого события (раннее начало последующей работы) заносят наибольшее значение времени на любом из путей, ведущих к данному событию.

В завершающем событии значение максимального значения переносят из левого сектора в правый.

Далее расчёт графика осуществляют по ранним окончаниям работ и ведут его в обратном порядке: от завершающего события к исходному. В этом случае значение правого сектора события получают путём вычитания из данных правого сектора завершающего события продолжительности работы, идущей от предыдущего события.

Если из события выходит несколько работ, то в правый сектор заносят разницу, имеющую наименьшее значение.

Далее, по значениям ранних начал работ и ранних окончаний работ определяют общие и частные резервы времени.

Под общим (полным) резервом времени понимают количество рабочего времени, на которое может быть увеличена продолжительность данной работы при условии, что продолжительность наибольшего из путей, проходящих через эту работу, не превышало длины критического пути.

Частным резервом времени называют количество рабочего времени, на которое может быть увеличена продолжительность данной работы или перенесено её начало так, чтобы при этом не изменилось раннее начало последующих работ.

Резервы времени записывают над стрелками работ в виде дроби: в числителе общий резерв времени, в знаменателе – частный резерв времени.

Общий (полный) резерв времени находят путём вычитания из значения правого сектора события (раннее окончание предшествующей работы), куда эта работа входит, значение левого сектора (раннее начало последующей работы), откуда эта работа выходит, минус продолжительность самой работы.

Частный резерв времени определяют путём вычитания значения левого сектора события, куда эта работа входит, значение левого сектора события, откуда эта работа выходит и минус продолжительность самой работы.

Критический путь проходит по тем работам, у которых значения правого и левого секторов события идентичны и у которых полностью отсутствуют резервы времени (как общие, так и частные).

Реализация

#include "stdafx.h"

#include <iostream>

#include <fstream>

#include <algorithm>

#include <vector>

#include <queue>

#include <stack>

using namespace std;

struct Line

{

int i, j, t;

Line() {}

Line(int a, int b, int c) { i = a; j = b; t = c; }

};

struct Vertex

{

int i, tr, tp, r;

Vertex();

Vertex(int a) { i = a; tr = 0; tp = 999; r = 0; }

};

vector <Line> table;//массив значений

vector <bool> check;

vector <Vertex> param;

int last;

//Считывание из файла.

void readFile(string name)

{

ifstream inp(name);

if (!inp.is\_open())

{

cout << "Error! File cannot be opened!" << endl;

system("pause");

exit(0);

}

while (!inp.eof())

{

Line l;

inp >> l.i >> l.j >> l.t;

table.push\_back(l);

}

inp.close();

}

void printTable(vector <Line> t)

{

cout << "i\tj\tt" << endl;

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

{

cout << t[i].i << "\t" << t[i].j << "\t" << t[i].t << endl;

}

}

int searchBegin()

{

vector<int> num;

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

{

bool flag = false;

for (int j = 0; j < num.size(); j++)

{

if (table[i].i == num[j]) flag = true;

}

if (!flag) num.push\_back(table[i].i);

}

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

{

for (int j = 0; j < num.size(); j++)

{

if (table[i].j == num[j]) num.erase(num.begin() + j);

}

}

if (num.size() == 1)

return num[0];

int n;

cout << "More than one begining vertexes found" << endl;

cout << endl << "Creating new fake vertex" << endl;

bool flag = false;

do

{

flag = false;

cout << "Enter nuber of vertex: ";

cin >> n;

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

{

if (table[i].i == n || table[i].j == n) flag = true;

}

if (flag) cout << "Vertex already exists" << endl;

} while (flag);

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

{

table.push\_back(Line(n, num[i], 0));

}

return n;

}

int searchEnd()

{

vector<int> num;

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

{

bool flag = false;

for (int j = 0; j < num.size(); j++)

{

if (table[i].j == num[j]) flag = true;

}

if (!flag) num.push\_back(table[i].j);

}

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

{

for (int j = 0; j < num.size(); j++)

{

if (table[i].i == num[j]) num.erase(num.begin() + j);

}

}

if (num.size() == 1)

return num[0];

int n;

cout << "More than one ending vertexes found" << endl;

cout << endl << "Create new fare vertex" << endl;

bool flag = false;

do

{

flag = false;

cout << "Enter number of vertex: ";

cin >> n;

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

{

if (table[i].i == n || table[i].j == n) flag = true;

}

if (flag) cout << "Vertex already exists" << endl;

} while (flag);

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

{

table.push\_back(Line(num[i], n, 0));

}

return n;

}

bool comp(Line a, Line b)

{

return a.j < b.j;

}

vector<Line> getFinal(int n)

{

vector<Line> a;

queue <Line> q;

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

{

if (table[i].i == n) { q.push(table[i]); a.push\_back(table[i]); table.erase(table.begin() + i); i--; }

}

while (!q.empty())

{

Line temp = q.front();

q.pop();

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

{

if (temp.j == table[i].i) { q.push(table[i]); a.push\_back(table[i]); table.erase(table.begin() + i); i--; }

}

}

return a;

}

int getNum(Line a)

{

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

{

if (a.i == table[i].i && a.j == table[i].j) return i;

}

}

Line NUM;

bool checkCyc(Line a)

{

if (a.i == a.j) { return true; }

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

{

if (a.j == table[i].i)

{

if (check[i] == true) { NUM = a; return true; }

check[i] = true;

if (checkCyc(table[i])) return true;

}

}

if (a.i != -999)check[getNum(a)] = false;

return false;

}

int getNum(int a)

{

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

{

if (a == param[i].i) return i;

}

return -999;

}

void getParam()

{

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

{

int n = getNum(table[i].i);

if (n == -999) param.push\_back(Vertex(table[i].i));

}

param.push\_back(Vertex(last));

for (int i = 1; i < param.size(); i++)

{

for (int j = 0; j < table.size(); j++)

{

if (table[j].j == param[i].i &&

param[i].tr < param[getNum(table[j].i)].tr + table[j].t)

param[i].tr = param[getNum(table[j].i)].tr + table[j].t;

}

}

for (int i = 1; i < param.size(); i++)

{

for (int j = 0; j < table.size(); j++)

{

if (table[j].j == param[i].i &&

param[i].tr < param[getNum(table[j].i)].tr + table[j].t)

param[i].tr = param[getNum(table[j].i)].tr + table[j].t;

}

}

param[param.size() - 1].tp = param[param.size() - 1].tr;

for (int i = param.size() - 2; i >= 0; i--)

{

for (int j = 0; j < table.size(); j++)

{

if (table[j].i == param[i].i &&

param[i].tp > param[getNum(table[j].j)].tp - table[j].t)

{

param[i].tp = param[getNum(table[j].j)].tp - table[j].t;

}

}

}

for (int i = param.size() - 2; i >= 0; i--)

{

for (int j = 0; j < table.size(); j++)

{

if (table[j].i == param[i].i &&

param[i].tp > param[getNum(table[j].j)].tp - table[j].t)

{

param[i].tp = param[getNum(table[j].j)].tp - table[j].t;

}

}

}

cout << "i\tTp\tTn\tPi" << endl;

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

{

param[i].r = param[i].tp - param[i].tr;

cout << param[i].i << "\t" << param[i].tr << "\t" << param[i].tp << "\t" << param[i].r << endl;

}

}

void delCyc()

{

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

{

if (table[i].i == table[i].j) {

table.erase(table.begin() + i);

}

}

for (int i = 0; i < check.size(); i++) check[i] = false;

while (checkCyc(Line(-666, table[0].i, 0)))

{

for (int i = 0; i < check.size(); i++) check[i] = false;

vector<Line> temp;

temp.push\_back(NUM);

Line temp2 = NUM;

bool flag = false;

for (int j = 0; j < table.size() && !flag; j++)

{

if (table[j].i == temp2.i && table[j].j != temp2.j)

flag = true;

}

while (temp2.i != NUM.j && !flag)

{

flag = false;

for (int i = 0; i < table.size() && !flag; i++)

{

if (table[i].j == temp2.i)

{

for (int j = 0; j < table.size() && !flag; j++)

{

if (table[j].i == table[i].i && table[j].j != table[i].j)

flag = true;

}

// if(temp2.i!=NUM.j&&!flag){

temp2 = table[i]; temp.push\_back(temp2);//}

}

}

}

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

{

table.erase(table.begin() + getNum(temp[i]));

}

}

}

vector<int> getCrit(vector<int> a, int n)

{

a.push\_back(n);

if (n == param[getNum(last)].i)

{

for (int i = 0; i < a.size(); i++) cout << a[i] << " ";

cout << endl;

}

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

{

if (table[i].i == n && param[getNum(table[i].j)].r == 0)

getCrit(a, table[i].j);

}

a.erase(a.begin() + (a.size() - 1));

return a;

}

void getRes()

{

cout << "i;j\tPп\tPс\t" << endl;

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

{

int r = param[getNum(table[i].j)].tp -

param[getNum(table[i].i)].tr - table[i].t;

int r2 = param[getNum(table[i].j)].tr -

param[getNum(table[i].i)].tp - table[i].t;

cout << table[i].i << ";" << table[i].j << "\t" << r << "\t" << r2 << endl;

}

}

int main()

{

setlocale(LC\_ALL, "");

readFile("in.txt");

cout << "Data:" << endl;

printTable(table);

cout << endl;

system("pause");

int beg = searchBegin();

int en = searchEnd();

last = en;

vector<Line> fin = getFinal(beg);

cout << "Sorted data:" << endl;

printTable(fin);

system("pause");

table = fin;

cout << endl;

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

check.push\_back(false);

if (checkCyc(Line(-999, table[0].i, 0)))

{

cout << "Cycles found. Deleting cycles." << endl;

delCyc();

cout << "After deleting cycles:" << endl;

printTable(table);

}

else cout << "Cycles not found" << endl;

system("pause");

cout << endl;

cout << "Calculating parameters:" << endl;

getParam();

cout << endl;

system("pause");

cout << "Free and full reserves:" << endl;

getRes();

cout << endl;

system("pause");

cout << "Crit ways:" << endl;

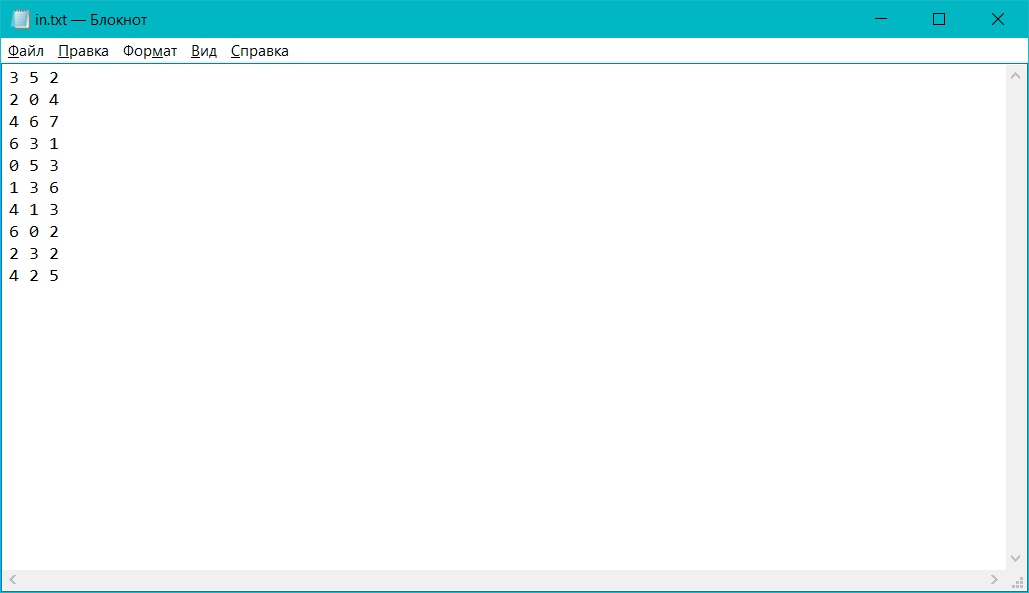
getCrit(vector<int>(), param[0].i);

cout << "Length: " << param[param.size() - 1].tp << endl;

system("pause");

}

Тестовый пример

Входные данные 

Результат

