МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики»

Факультет информационных технологий и программирования

Кафедра информационных систем

Лабораторная работа №1

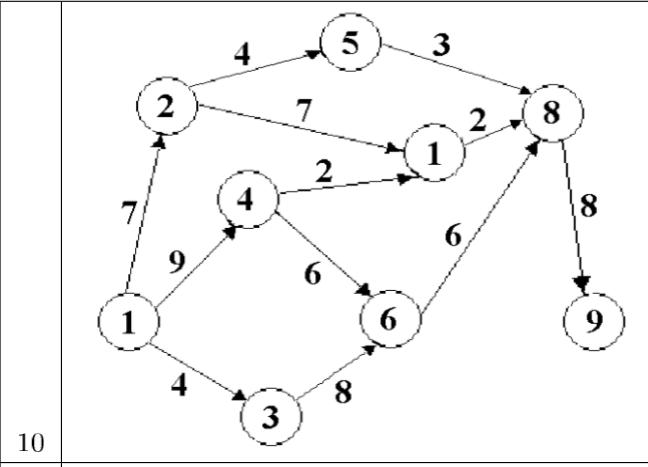
Методы одномерного поиска экстремума

Выполнил студент группы №3307:   
Егоров Николай Валерьевич

Проверила:

Сайферт Ирина Викторовна

САнкт-Петербург  
 2018



Классическая задача динамического программирования, когда целиковая задача разбивается на ряд подзадач, проще. В данном случае будем разбивать вершины следующим образом: {данная}, {выходящие},{входящие}. Тогда для каждой вершины справедливо следующее выражение для минимального пути – минимум среди множества вариантов входящих вершин. Сформируем критерий оптимальности Беллмана: каково бы ни было начальное состояние на k-м шаге и выбранное на этом шаге минимум , последующие управления должны быть оптимальными по отношению к состоянию, получающемуся в результате решения, принятого на шаге k.  
Для решения данной задачи можно воспользоваться алгоритмом Беллмана-Форда для графа в общем случае, но при условии, что граф не имеет циклов. Для каждой из вершин будет производиться релаксация по числу ребер, и, как итог, получается результат со сложностью

Результат нам даст расстояние от 1й вершины до 9й: 21.

Однако же данное время можно оптимизировать, расположив вершины графа в топологическом порядке в зависимости от начального состояния, что дает следующий порядок вершин (при данном графе):

1 3 4 6 2 7 8 9

И на каждой итерации, те для каждой вершины, проводится релаксация по инцидентным ребрам из ранее просмотренных вершин, таким образом на каждом шаге все ребра не пересматривается, и в конце алгоритм обойдем V вершин и E ребер. Те .

Результаты

Topological order solution

Shortest path costs: 21

Bellman-Ford way

Shortest path costs: 21

#include <iostream>

#include <vector>

#include <algorithm>

#include <stack>

#include <list>

using namespace std;

struct edge {

int a, b, cost;

};

class listNode{

int v;

int weight;

public:

listNode(int \_v, int \_w) { v = \_v; weight = \_w; }

int getV() { return v; }

int getWeight() { return weight; }

};

class linearGraph {

int V\_number;

list<listNode> \*graph;

void topSort(int start, vector<bool> &visited, stack<int> &Order);

public:

linearGraph(int V) {

this->V\_number = V;

graph = new list<listNode>[V];

}

void addEdge(int u, int v, int weight);

void newShortPath(int start );

};

int main() {

int numb\_e=0, was,n, m, v;

vector<edge> e;

stack<int> Order;

const int INF = 1000000000;

freopen("in.txt", "r",stdin);

cin >> n >> m >> v;

was = n;

e.emplace\_back(edge{ n, m, v });

++numb\_e;

do {

cin >> n >> m >> v;

if (n != was) {

++numb\_e;

was = n;

}

e.emplace\_back(edge{ n, m, v });

} while (cin);

vector<int> d(numb\_e+2, INF);

//new way O(V+E)

linearGraph gr(numb\_e+2);

for (int i = 0; i < e.size()-1; i++) {

gr.addEdge(e[i].a, e[i].b, e[i].cost);

}

cout << "Topological order solution\nShortest path costs: ";

gr.newShortPath(1);

d[1] = 0;

cout << "\n\nBellman-Ford way\nShortest path costs: ";

for (int i = 0; i<numb\_e -1 ; ++i)

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

if (d[e[j].a] < INF)

d[e[j].b] = min(d[e[j].b], d[e[j].a] + e[j].cost);

}

cout << d[numb\_e+1]<<endl;

/\*

for (auto i : d) {

cout << i << " \n";

}

\*/

}

void linearGraph::topSort(int start, vector<bool> &visited, stack<int> &Order) {

visited[start] = true;

list<listNode>::iterator i;

for (i = graph[start].begin(); i != graph[start].end(); ++i)

{

listNode node = \*i;

if (!visited[node.getV()])

topSort(node.getV(), visited, Order);

}

Order.push(start);

}

void linearGraph::addEdge(int u, int v, int weight)

{

listNode node(v, weight);

graph[u].push\_back(node);

}

void linearGraph::newShortPath(int start)

{

const int INF = INT\_MAX;

stack<int> Order;

vector<bool> visited(V\_number +1, false);

vector<int> dist(V\_number + 1, INF);

dist[start] = 0;

for (int i = 0; i < V\_number; i++)

if (!visited[i]) {

topSort(i, visited, Order);

}

while (Order.empty() == false)

{

int u = Order.top();

Order.pop();

list<listNode>::iterator i;

if (dist[u] != INF)

{

for (i = graph[u].begin(); i != graph[u].end(); ++i)

if (dist[i->getV()] > dist[u] + i->getWeight())

dist[i->getV()] = dist[u] + i->getWeight();

}

}

cout << dist[V\_number-1] << " ";

/\* for (auto i:dist) {

if (i < INF)

cout << i << " \n";

}

\*/

}