# ЦЕЛЬ РАБОТЫ

Изучение алгоритмов поиска кратчайших путей на графах на примере метода динамического программирования

# ПОСТАНОВКА ЗАДАЧИ

1. Составить структурную схему программы, определяющей кратчайший путь на графе от заданной начальной вершины *s* до заданной конечной вершины *t* с помощью метода динамического программирования.
2. Составить структурную схему программы, реализующей алгоритм

топологической сортировки с произвольной нумерацией вершин графа.

1. Создать программу, реализующую метод динамического программирования и алгоритм топологической сортировки вершин.
2. Исходный граф задается в виде матрицы смежности.
3. Создать программу, которая использует приведенный в данной работе

алгоритм Дейкстры для заданного графа.

Таблица 1. Исходная матрица смежности заданная вариантом задания №4.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 1 |  | 9 | 4 |  |  |  |  |  |  |
| 2 |  |  | 7 | 10 |  |  |  |  |  |
| 3 |  |  |  |  |  | 4 |  | 6 |  |
| 4 |  |  | 6 |  | 3 |  |  |  |  |
| 5 |  |  |  |  |  |  | 1 |  |  |
| 6 |  |  |  |  | 9 |  |  | 7 |  |
| 7 |  |  |  | 1 |  | 8 |  |  | 6 |
| 8 |  |  |  |  |  |  |  |  | 4 |

# D:\university\II курс\IV семестр\ОТА\lab5\2.jpg

# Рисунок 1 – Вид графа, построенного в соответствии с матрицей смежности

# СТРУКТУРНАЫЕ СХЕМЫ АЛГОРИТМОВ



Рисунок 2 – Структурная схема алгоритма функции топологической сортировки с произвольной нумерацией вершин графа



Рисунок 3 – Структурная схема алгоритма рекурсивной функции определения всех входящих и исходящих дуг вершины



Рисунок 4 – Структурная схема алгоритма функции поиска кратчайшего пути от начальной вершины s, к конечной вершине t

# ТЕКСТ ПРОГРАММЫ

1. Текст программы реализующей алгоритм Дейкстры.

#include <vector>

using namespace std;

typedef vector<size\_t> arr;

typedef vector<vector<size\_t>> matrix;

const size\_t ROOT\_AMOUNT = 9;

size\_t solve(matrix g, size\_t s, size\_t f);

size\_t sum(arr d) {

size\_t temp = 0;

for (size\_t i = 0; i < d.size(); ++i) {

temp += d[i];

}

return temp;

}

void fillMatr(matrix &rootMatr) {

rootMatr[0][1] = 9;

rootMatr[0][2] = 4;

rootMatr[1][2] = 7;

rootMatr[1][3] = 10;

rootMatr[2][5] = 4;

rootMatr[2][7] = 6;

rootMatr[3][2] = 6;

rootMatr[3][4] = 3;

rootMatr[4][6] = 1;

rootMatr[5][4] = 9;

rootMatr[5][7] = 7;

rootMatr[6][3] = 1;

rootMatr[6][5] = 8;

rootMatr[6][8] = 6;

rootMatr[7][8] = 4;

}

void fillArr(arr &d) {

for (size\_t i = 0; i < d.size(); ++i) {

d[i] = std::numeric\_limits<size\_t>::max();

}

}

size\_t main() {

matrix ROOT\_MATRIX(ROOT\_AMOUNT, arr (ROOT\_AMOUNT));

fillMatr(ROOT\_MATRIX);

arr d;

vector<bool> used(ROOT\_AMOUNT);

size\_t shortestPath = solve(ROOT\_MATRIX, 0, ROOT\_AMOUNT-1);

return 0;

}

size\_t solve(matrix g, size\_t s, size\_t f) {

// n - количество вершин

// g[n][n] - матрица смежности, g[i][j] = 0, если текущего ребра нет

// d[n] - массив ответов// s - стартовая вершина

// used - массив для пометок вершин

int i, j, // i-я итерация, j - для поиска минимальной

v, // минимальная вершина

to, // ребро из вершины v в to

len; // длины len

size\_t n = g.size();

arr d(n);

fillArr(d);

vector<bool> used(n);

d[s] = 0;

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

v = -1;

for (j = 0; j < n; j++) // поиск вершины с минимальным d[j]

if (!used[j] && (v == -1 || d[j] < d[v])){

v = j;

}

else if (!used[j] && (d[j] < d[v])) {

v = j;

}

used[v] = true; // пометка вершины

for (to = 0; to < n; to++) {

if (g[v][to]) {

len = g[v][to];

if (d[v] + len < d[to]) {

d[to] = d[v] + len;

}

}

}

}

return d[f];

}

1. Текст программы реализующей метод динамического программирования.

#include <vector>

using namespace std;

typedef vector<size\_t> arr;

typedef vector<vector<size\_t>> matrix;

const size\_t ROOT\_AMOUNT = 9;

size\_t min(size\_t a, size\_t b) {

return (a > b) ? b : a;

}

void fillMatr(matrix &rootMatr) {

rootMatr[0][1] = 9;

rootMatr[0][2] = 4;

rootMatr[1][2] = 7;

rootMatr[1][3] = 10;

rootMatr[2][5] = 4;

rootMatr[2][7] = 6;

rootMatr[3][2] = 6;

rootMatr[3][4] = 3;

rootMatr[4][6] = 1;

rootMatr[5][4] = 9;

rootMatr[5][7] = 7;

rootMatr[6][3] = 1;

rootMatr[6][5] = 8;

rootMatr[6][8] = 6;

rootMatr[7][8] = 4;

}

void fillArr(arr &d) {

for (size\_t i = 0; i < d.size(); ++i) {

d[i] = std::numeric\_limits<size\_t>::max();

}

}

size\_t dfs(const size\_t v, const matrix g, vector<bool> &used, arr &top, size\_t &l);

size\_t topSort(const matrix g, vector<bool> &used, arr &top);

size\_t solve(const matrix &g, arr &top, arr &d, const size\_t s, const size\_t fRoot);

size\_t main() {

matrix ROOT\_MATRIX(ROOT\_AMOUNT, arr(ROOT\_AMOUNT));

fillMatr(ROOT\_MATRIX);

arr top(ROOT\_AMOUNT);

arr d(ROOT\_AMOUNT);

vector<bool> used(ROOT\_AMOUNT);

topSort(ROOT\_MATRIX, used, top);

size\_t weight = solve(ROOT\_MATRIX, top, d, 0, ROOT\_AMOUNT-1);

return 0;

}

size\_t dfs(const size\_t v,const matrix g, vector<bool> &used, arr &top, size\_t &l) {

size\_t n = g.size();

if (used[v])

return 0;

used[v] = true;

for (size\_t to = 0; to < n; to++)

if (g[v][to])

dfs(to, g, used, top, l);

top[l++] = v; // добавление вершины v в отсортированный список

}

size\_t topSort(const matrix g, vector<bool> &used, arr &top) {

size\_t n = g.size();

size\_t l = 0; // номер добавляемой вершины в отсортированный список

for (size\_t i = 0; i < n; i++)

dfs(i,g,used,top,l); // запустить пробежку из всех вершин

reverse((top.begin()), (top.begin() + l)); // развернуть массив

return 0;

}

size\_t solve(const matrix &g, arr &top, arr &d, size\_t s, const size\_t fRoot) {

size\_t i, j;

size\_t n = g.size();

fillArr(d);

d[s] = 0;

for (i = 1; i < n; i++)

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

if (g[top[j]][top[i]])

d[top[i]] = min(d[top[i]], d[top[j]] + g[top[j]][top[i]]);

return d[fRoot];

}

# РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ

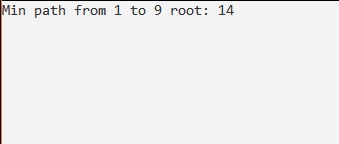


Рисунок 5 – Результат выполнения программы реализующей метод динамического программирования для поиска кратчайшего пути в графе

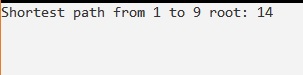


Рисунок 6 – Результат выполнения программы реализующей алгоритм Дейкстры для поиска кратчайшего пути в графе

# ВЫВОДЫ

В ходе лабораторной работы были исследованы два алгоритма для поиска кратчайших путей в ориентированных графах: алгоритм Дейкстры и метод динамического программирования. Так же для обеспечения работы алгоритма реализующего метод динамического программирования был рассмотрен алгоритм топологической сортировки вершин графа. По результатам сравнения программ можно сделать вывод, что более быстрым алгоритмом для поиска кратчайшего пути в графах является алгоритм Дейкстры, но на данном наборе исходных данных его преимущество незначительно.