# ЛАБОРАТОРНАЯ РАБОТА №5

# СРАВНИТЕЛЬНОЕ ИССЛЕДОВАНИЕ АЛГОРИТМОВ ПОИСКА

# КРАТЧАЙШИХ ПУТЕЙ НА ГРАФАХ

Цель работы

Исследование фундаментальных алгоритмов поиска кратчайших путей на графах на примере метода динамического программирования. Получение навыков практического применения алгоритмов на графах.

Задания

1. Написать программу, реализующую метод динамического программирования и алгоритм топологической сортировки вершин. Исходный граф (Рисунок 1) задаётся в виде матрицы смежности. Для определения вершин, входящих во множество Г–1(xi), использовать j-й столбец матрицы смежности;

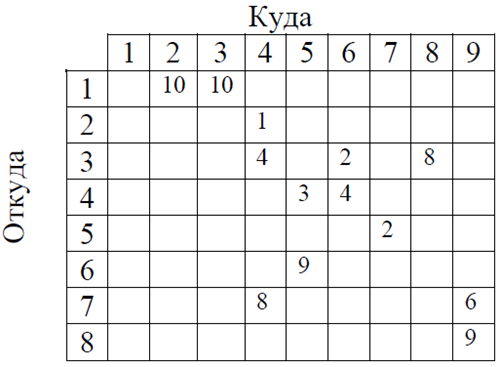


Рисунок 1 – Граф (Вариант 12)

1. Создать программу, которая использует алгоритм Дейкстры для заданного графа;
2. Сравнить время выполнения двух алгоритмов;

Текст программы

Программа с алгоритмом Дейкстры:

#include <iostream>

#include <ctime>

#include <windows.h>

using namespace std;

const int INF = 1000000000; // бесконечность

const int n = 9; // количество вершин

int d[n] = { 0 }; // массив ответов

bool used[n] = { 0 }; // массив для пометок

int g[n][n] = { // матрица смежности

{0, 10, 10, 0, 0, 0, 0, 0, 0}, // 1

{0, 0, 0, 1, 0, 0, 0, 0, 0}, // 2

{0, 0, 0, 4, 0, 2, 0, 8, 0}, // 3

{0, 0, 0, 0, 3, 4, 0, 0, 0}, // 4

{0, 0, 0, 0, 0, 0, 2, 0, 0}, // 5

{0, 0, 0, 0, 9, 0, 0, 0, 0}, // 6

{0, 0, 0, 0, 0, 0, 0, 0, 6}, // 7

{0, 0, 0, 0, 0, 0, 0, 0, 9}, // 8

{0, 0, 0, 0, 0, 0, 0, 0, 0} // 9

};

// Алгоритм Дейкстры

void solve(int s)

{

for (int i = 0; i < n; i++) // предзаполнение d[]

d[i] = INF;

d[s] = 0;

for (int i = 0; i < n; i++) // поиск минимальных расстояний

{

int v = -1; // непомеченная вершина с мин. d[v]

for (int j = 0; j < n; j++) // поиск v в массиве d[]

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

v = j;

used[v] = true; // пометка вершины

for (int to = 0; to < n; to++) // релаксации

if (g[v][to])

{

int len = g[v][to]; // длина дуги (v,to)

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

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

}

}

return;

}

// Поиск минимального пути от вершины s до f

int main()

{

int s; // начальная вершина

cout << "Enter s - ";

cin >> s;

int f; // конечная вершина

cout << "Enter f - ";

cin >> f;

LARGE\_INTEGER freq, start\_time, end\_time; // подсчёт времени

QueryPerformanceFrequency(&freq);

QueryPerformanceCounter(&start\_time);

solve(s - 1);

QueryPerformanceCounter(&end\_time);

long long el\_time\_ns = (end\_time.QuadPart - start\_time.QuadPart) \* 1000000000LL / freq.QuadPart;

cout << "Length = " << d[f - 1] << endl;

cout << "Time - " << el\_time\_ns << " ns";

return 0;

}

Программа с методом динамического программирования и топологической сортировкой:

#include <iostream>

#include <ctime>

#include <windows.h>

using namespace std;

const int INF = 1000000000; // бесконечность

const int n = 9; // количество вершин

int d[n] = { 0 }; // массив ответов

bool used[n] = { 0 }; // массив для пометок

int top[100] = { 0 }; // топологический список

int l; // l-я вершина для добавления

int g[n][n] = { // матрица смежности

{0, 10, 10, 0, 0, 0, 0, 0, 0}, // 1

{0, 0, 0, 1, 0, 0, 0, 0, 0}, // 2

{0, 0, 0, 4, 0, 2, 0, 8, 0}, // 3

{0, 0, 0, 0, 3, 4, 0, 0, 0}, // 4

{0, 0, 0, 0, 0, 0, 2, 0, 0}, // 5

{0, 0, 0, 0, 9, 0, 0, 0, 0}, // 6

{0, 0, 0, 0, 0, 0, 0, 0, 6}, // 7

{0, 0, 0, 0, 0, 0, 0, 0, 9}, // 8

{0, 0, 0, 0, 0, 0, 0, 0, 0} // 9

};

// Пробежка вершин графа

void dfs(int v)

{

if (used[v]) return;

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

if (g[v][to])

dfs(to);

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

used[v] = true;

return;

}

// Разворот массива

void reverse(int arr[], int len)

{

for (int i = 0; i <= len / 2; i++)

{

int c = arr[i];

arr[i] = arr[len - 1 - i];

arr[len - 1 - i] = c;

}

}

// Топологическая сортировка

void topSort()

{

l = 0; // номер добавляемой вершины в отсортированный список

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

dfs(i); // пробежка всех вершин

reverse(top, n); // разворот массива

return;

}

// Метод динамического программирования

void solve(int s)

{

for (int i = 0; i < n; i++) // предзаполнение d[]

d[i] = INF;

d[s] = 0;

for (int i = s + 1; i < n; i++) // обход вершин

for (int j = 0; j < i; j++) // обход вершин, входящих в xi

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

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

return;

}

// Поиск минимального пути от вершины s до f

int main()

{

int s; // начальная вершина

cout << "Enter s - ";

cin >> s;

int f; // конечная вершина

cout << "Enter f - ";

cin >> f;

LARGE\_INTEGER freq, start\_time, end\_time; // подсчёт времени

QueryPerformanceFrequency(&freq);

QueryPerformanceCounter(&start\_time);

topSort();

solve(s - 1);

QueryPerformanceCounter(&end\_time);

long long el\_time\_ns = (end\_time.QuadPart - start\_time.QuadPart) \* 1000000000LL / freq.QuadPart;

cout << "Length = " << d[f - 1] << endl;

cout << "Time - " << el\_time\_ns << " ns";

return 0;

}

Ход работы

Был проанализирован указанный в задании граф. Граф оказался с неправильной нумерацией (дуги 6-5, 7-4), циклическим на вершинах 4-5-7, поэтому для второго алгоритма из графа была убрана вершина 7-4 (Рисунок 2). Аналитически был найден кратчайший путь от первой до девятой вершины – 22.

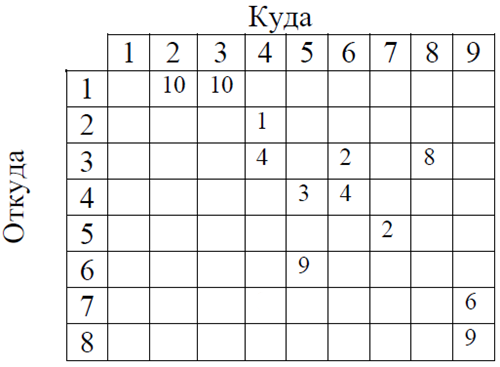


Рисунок 2 – Ациклический граф

Была запущена программа, решающая задачу о длине кратчайшего пути с помощью алгоритма Дейкстры (Рисунок 3). Ею был верно определён кратчайший путь между 1-9, на подсчёты ушло 2400 наносекунд.

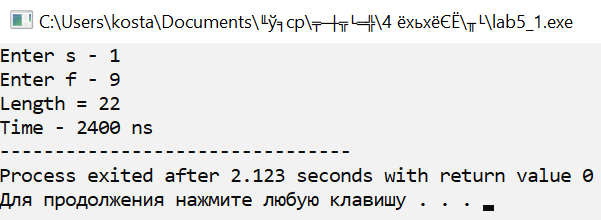


Рисунок 3 – Действие алгоритма Дейкстры

Затем была запущена программа, решающая ту же задачу методом динамического программирования (Рисунок 4). Путь также был определён верно, однако времени ей потребовалось больше – 4800 наносекунд.

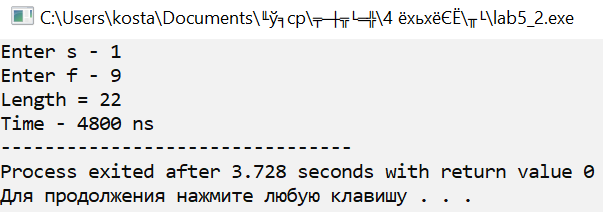


Рисунок 4 – Действие метода динамического программирования

По идее, метод динамического программирования должен быть более эффективнее, однако поскольку работа идёт с небольшим графом, имеющим неправильную нумерацию, вторая программа тратит на решение задачи заметно больше времени из-за алгоритма топологической сортировки.

Функция топологической сортировки во второй программе была помещена перед началом отсчёта времени. В результате на чистый метод динамического программирования ушло гораздо меньше времени – 1200 наносекунд (Рисунок 5).

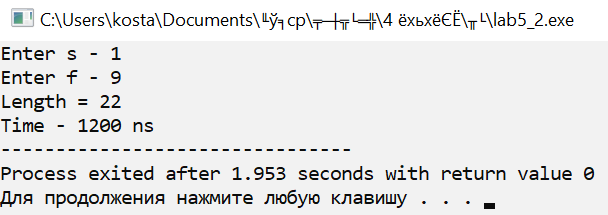


Рисунок 5 – Результат повторного запуска

Вывод

В ходе работы были исследованы фундаментальные алгоритмы поиска кратчайших путей на графах, а именно алгоритм Дейкстры и метод динамического программирования. Алгоритмы были применены на практике для решения задачи с графом.

При исследовании быстродействия указанных алгоритмов было определено, что при работе с правильно пронумерованными ациклическими графами метод динамического программирования эффективнее алгоритма Дейкстры, однако при обратных условиях необходимость топологической сортировки делает его гораздо менее эффективным при работе с небольшими графами.