МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Вятский государственный университет»

(ФГБОУ ВО «ВятГУ»)

Факультет автоматики и вычислительной техники

Кафедра ЭВМ

Отчёт

Лабораторная работа № 5-6 по дисциплине

«Исследование операций»

Выполнил студент группы ИВТб-2301-04-00 / Жеребцов К. А.

Проверил преподаватель / Коржавина А. С.

Киров 2022

1. Цель работы: закрепить на практике знания о способах решения задач динамического программирования и получить навыки их программной реализации.
2. Задание: написать программы, одна из который находит кратчайший путь в графе, а другая решает задачу вычисления произведения матриц.
3. Описание алгоритмов:

Поиск кратчайшего пути в графе

**Инициализация.** Пусть вершиной, из которой мы будем считать маршруты, будет 0. Расстояние до самой себя у этой вершины логично равно нулю. Остальные мы пока не знаем, поэтому отметим символом бесконечности.

Расстояние от 0 до 0 помечаем равным нулю, а саму вершину — посещенной.

**Первый шаг алгоритма.** Мы выбираем еще не посещенную вершину с самой маленькой меткой относительно исходной — то есть такую, которая находится ближе всех. На первом шаге это одна из соседних вершин — та, которая соединена с исходной самым «маленьким» ребром.

Для графа, который мы рассматриваем, это точка 2. Мы выбираем ее, «переходим» в нее и смотрим уже на ее соседей.

**Дальнейшие шаги алгоритма.** Для выбранной точки нужно осмотреть соседей и записать длину пути до них с учетом пройденного пути. А потом выбрать ближнюю точку. Но есть нюанс: нужно учитывать точки, которые мы уже использовали в прошлый раз. Если они дают более «выгодный» путь, лучше воспользоваться ими.

Например, на выбранном графе есть точка 1. В нее можно перейти из точки 2, где мы находимся. Но этот путь будет длиннее, чем при переходе напрямую из точки 0, а ведь она для нас исходная. Поэтому «короткий путь» для точки 1 — это маршрут 0–1. Отмечаем вершину посещенной.

Шаги повторяются, пока на графе есть непосещенные точки. Если вершину не посетили, она не участвует в расчетах. Если после ее «открытия» появился новый, более короткий путь к какой-либо точке, то минимальное расстояние для нее перезаписывается.

**Конец алгоритма.** Когда непосещенные вершины заканчиваются, алгоритм прекращает работу. Результат его действия — список кратчайших маршрутов до каждой точки из исходной. Для каждого маршрута указана его длина.

Задача вычисления перемножения матриц

Ниже приведен рекурсивный алгоритм для нахождения минимальной стоимости:

* Возьмите последовательность матриц и разделите ее на две подпоследовательности.
* Найдите минимальную стоимость умножения каждой подпоследовательности.
* Сложите эти затраты вместе и добавьте цену умножения двух матриц результатов.
* Сделайте это для каждой возможной позиции, в которой последовательность матриц может быть разделена, и возьмите минимум по всем из них.

1. Листинг первой программы:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#define SIZE 5

int main()

{

int a[SIZE][SIZE];

int d[SIZE];

int v[SIZE];

int temp, minindex, min;

int begin\_index = 0;

system("chcp 1251");

system("cls");

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

{

a[i][i] = 0;

for (int j = i + 1; j < SIZE; j++) {

printf("Введите расстояние между вершинами %d - %d: ", i + 1, j + 1);

scanf("%d", &temp);

a[i][j] = temp;

a[j][i] = temp;

}

}

printf("\nТаблица:\n");

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

{

for (int j = 0; j < SIZE; j++)

printf("%5d ", a[i][j]);

printf("\n");

}

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

{

d[i] = 10000;

v[i] = 1;

}

d[begin\_index] = 0;

// алгоритм

do {

minindex = 10000;

min = 10000;

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

{

if ((v[i] == 1) && (d[i] < min))

{

min = d[i];

minindex = i;

}

}

if (minindex != 10000)

{

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

{

if (a[minindex][i] > 0)

{

temp = min + a[minindex][i];

if (temp < d[i])

{

d[i] = temp;

}

}

}

v[minindex] = 0;

}

} while (minindex < 10000);

int ver[SIZE];

int end = 4;

ver[0] = end + 1;

int k = 1;

int weight = d[end];

while (end != begin\_index)

{

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

if (a[end][i] != 0)

{

int temp = weight - a[end][i];

if (temp == d[i])

{

weight = temp;

end = i;

ver[k] = i + 1;

k++;

}

}

}

printf("\nВывод кратчайшего пути\n");

for (int i = k - 1; i >= 0; i--)

printf("%3d ", ver[i]);

getchar(); getchar();

return 0;

}

1. Экранные формы первой программы:

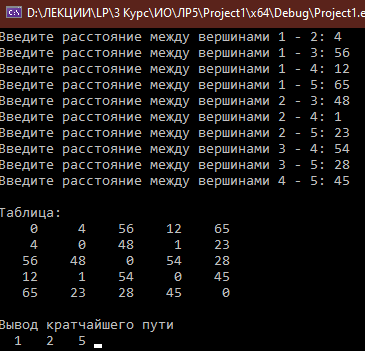


Рисунок 1 – Успешное выполнение программы.

6. Листинг второй программы:

#include <iostream>

#include <vector>

#include <climits>

using namespace std;

int matrixChainMultiplication(vector<int> const& dims, int i, int j, vector<vector<int>> & lookup, vector<vector<int>> & s)

{

// базовый вариант: одна матрица

if (j <= i + 1)

{

cout << "!!!\n\n";

return 0;

}

// сохраняет минимальное количество скалярных умножений (т.е. стоимость)

// необходимо вычислить матрицу M[i+1] … M[j] = M[i…j]

int min = INT\_MAX;

if (lookup[i][j] == 0)

{

for (int k = i + 1; k <= j - 1; k++)

{

// повторить для M[i+1]…M[k], чтобы получить матрицу i × k

int cost = matrixChainMultiplication(dims, i, k, lookup, s);

// повторить для M[k+1]…M[j], чтобы получить матрицу k × j

cost += matrixChainMultiplication(dims, k, j, lookup, s);

// стоимость умножения двух матриц i × k и k × j

cost += dims[i] \* dims[k] \* dims[j];

cout << i << "\t" << k << "\t" << j << "\n";

cout << dims[i] << "\t" << dims[k] << "\t" << dims[j] << "\n";

cout << cost << "\n";

if (cost < min) {

min = cost;

}

s[i][j] = k;

}

lookup[i][j] = min;

for (int i = 0; i < dims.size() + 1; i++)

{

for (int j = 0; j < dims.size() + 1; j++)

{

cout << lookup[i][j] << " |\t";

}

cout << "\n";

}

cout << "\n\n";

for (int i = 0; i < dims.size() + 1; i++)

{

for (int j = 0; j < dims.size() + 1; j++)

{

cout << s[i][j] << " |\t";

}

cout << "\n";

}

}

cout << lookup[i][j] << "\n";

// вернуть минимальную стоимость для умножения M[j+1]…M[j]

cout << i << "\t" << j << "\n";

cout << "----------------------------------------------------\n";

return lookup[i][j];

}

// Задача умножения цепочки матриц

int main()

{

vector<int> dims = { 30, 15, 5, 60 };

int n = dims.size();

// таблица поиска для хранения решения уже вычисленных подзадач

vector<vector<int>> lookup(n + 1, vector<int>(n + 1));

vector<vector<int>> s(n + 1, vector<int>(n + 1));

cout << "The minimum cost is " << matrixChainMultiplication(dims, 0, n - 1, lookup, s) << "\n";

cout << n << "\n";

return 0;

}

7. Экранные формы второй программы:

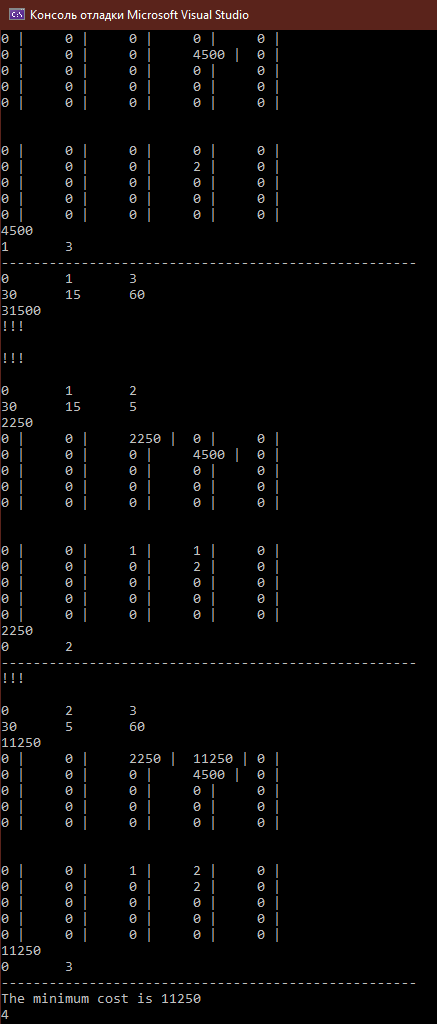


Рисунок 2 – Успешное выполнение программы.

1. Вывод:

В ходе выполнения лабораторной работы были закреплены на практике знания о методах поиска кратчайшего пути в графе и нахождения оптимальной последовательности перемножения матриц. Также получены навыки их программной реализации.