ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет Информационных технологий
Кафедра СМАРТ-технологии

направление подготовки
09.03.02 «Информационные системы и технологии»

ЛАБОРАТОРНАЯ РАБОТА № 4

Дисциплина: Теория принятия решений

Тема: ”Разработка программного обеспечения для принятия решения с использованием метода динамического программирования”

Выполнил: студент группы 211-723

Сергеев С. О.

27.11.2022
(Дата) (Подпись)

Проверил: д.т.н. профессор Истомина Т. В.
(Фамилия И.О., степень, звание) **(Оценка)**

_____ _____
(Дата) (Подпись)

Замечания:

_____

_____

_____

Москва

2022

1. Задача

Написать программу, реализующую Алгоритм Флойда — Уоршелла.

Алгоритм Флойда — Уоршелла находит кратчайшие пути между всеми вершинами взвешенного ориентированного графа.

2. Цель

Целью данной работы разработка программы, реализующей алгоритм Флойда методом динамического программирования.

Для решения данной задачи используются методы теории графов.

3. Теоретический материал

Граф - математическая абстракция реальной системы объектов любой природы, обладающих парными связями. Граф как математический объект есть совокупность двух множеств - множества самих объектов, называемого множеством вершин и множеством их парных связей, называемой множеством рёбер. Элемент множества рёбер есть пара элементов множества вершин.

Задача о кратчайшем пути - задача поиска самого короткого пути (цепи) между двумя точками (вершинами) на графе, в которой минимизируется сумма весов ребер, составляющих путь.

Существуют различные постановки задачи о кратчайшем пути:

- Задача о кратчайшем пути в заданный пункт назначения. Требуется найти кратчайший путь в заданную вершину назначения t, который начинается в каждой из вершин графа (кроме t). Поменяв направление каждого принадлежащего графу ребра, эту задачу можно свести к задаче о единой исходной вершине (в которой осуществляется поиск кратчайшего пути из заданной вершины во все остальные).
- Задача о кратчайшем пути между заданной парой вершин. Требуется найти кратчайший путь из заданной вершины u в заданную вершину v.
- Задача о кратчайшем пути между всеми парами вершин. Требуется найти кратчайший путь из каждой вершины u в каждую вершину v. Эту задачу тоже можно решить с помощью алгоритма, предназначенного для решения задачи об одной исходной вершине, однако обычно она решается быстрее.

Алгоритм Флойда — Уоршелла находит кратчайшие пути между всеми вершинами взвешенного ориентированного графа.

Алгоритм Флойда — Уоршелла – алгоритм для нахождения кратчайших расстояний между всеми вершинами взвешенного графа без циклов с отрицательными весами с использованием метода динамического программирования

Динамическое программирование в теории управления и теории вычислительных систем - способ решения сложных задач путём разбиения их на более простые подзадачи.

4. Ход решения

Имеется взвешенный, ориентированный граф, изображенный на рисунке 1.

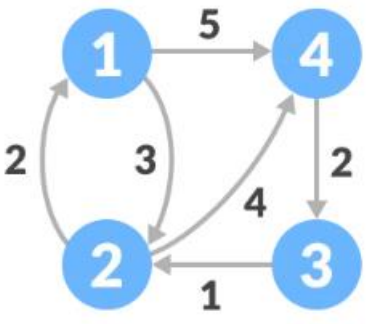

Рисунок 1 - Исходный граф

Шаг 1. Создадим матрицу A^0 размерности n*n, где n - количество вершин. Строки и столбцы индексируются как i и j соответственно. Каждая ячейка A^0 [i][j] заполнена расстоянием от i-й вершины до j-й вершины. Если пути от i-й вершины к j-й вершине нет, ячейка остается бесконечной. Полученная матрица изображена на рисунке 2.

$$A^0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & \infty & 5 \\ 2 & 0 & \infty & 4 \\ \infty & 1 & 0 & \infty \\ \infty & \infty & 2 & 0 \end{bmatrix} \end{array}$$

Рисунок 2 - Матрица A^0

Шаг 2. Теперь создадим матрицу A^1, используя матрицу A^0. Элементы в первом столбце и первой строке оставим без изменений. Остальные ячейки заполняются следующим образом.

Пусть k будет промежуточной вершиной на кратчайшем пути от источника до пункта назначения. На этом шаге k - первая вершина. A [i] [j] заполняется (A [i] [k] + A [k] [j]), если (A [i] [j]> A [i] [k] + A [k] [j]).

То есть, если прямое расстояние от источника до пункта назначения больше, чем путь через вершину k, то ячейка заполняется значением A [i] [k] + A [k] [j]. На этом шаге k является вершиной 1. Мы вычисляем расстояние от исходной вершины до конечной вершины через эту вершину k.

Полученные значения заносятся в матрицу A^1(рисунок 3).

$$A^1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & \infty & 5 \\ 2 & 0 & & \\ \infty & & 0 & \\ \infty & & & 0 \end{bmatrix} \end{array} \rightarrow \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & \infty & 5 \\ 2 & 0 & 9 & 4 \\ \infty & 1 & 0 & 8 \\ \infty & \infty & 2 & 0 \end{bmatrix} \end{array}$$

Рисунок 3 - Матрица A^1

Шаг 3. Точно так же матрица A^2(рисунок 4), изображенная на рисунке 4, создается с помощью A^1. Элементы во втором столбце и второй строке оставлены как есть. На этом шаге k - вторая вершина (т.е. вершина 2). Остальные шаги такие же, как в шаге 2.

$$A^2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & & \\ 2 & 0 & 9 & 4 \\ & 1 & 0 & \\ & \infty & & 0 \end{bmatrix} \end{array} \rightarrow \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & 9 & 5 \\ 2 & 0 & 9 & 4 \\ 3 & 1 & 0 & 5 \\ \infty & \infty & 2 & 0 \end{bmatrix} \end{array}$$

Рисунок 4 - Матрица A^2

Шаг 4. Таким же образом создаются матрицы A^3 (Рисунок 5) и A^4 (Рисунок 6).

Рисунок 5 - Матрица A^3

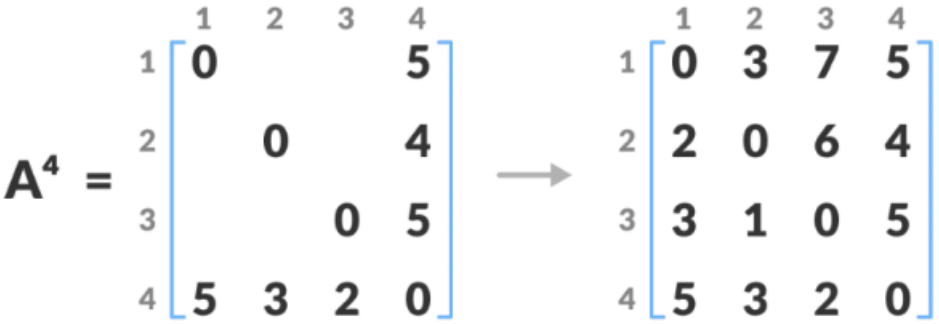

Рисунок 6 - Матрица A^4

Матрица A^4 является матрицей кратчайших путей на графе A.

Листинг программы:

```
#include <iostream>
#include <iomanip>
#include <vector>
using namespace std;
const int inf = 999;

class Matrix
{
private:
	vector<vector<int>> graph;
	int countOfVertices;
public:
	Matrix(int _countOfVertices)
	{
		vector<vector<int>> _graph(_countOfVertices, vector<int>(_countOfVertices, 0));
		graph = _graph;
		countOfVertices = _countOfVertices;
	}
	void AddToMatrix(int _i, int _j, int _value)
	{
		graph[_i][_j] = _value;
	}

	int GetFromMatrix(int _i, int _j)
	{
		return graph[_i][_j];
	}

	void setCountOFVertices(int _count)
	{
		countOfVertices = _count;
	}

	void getMatrix()
```

```
	{
		for (int i = 0; i < countOfVertices; i++)
		{
			for (int j = 0; j < countOfVertices; j++)
			{
				cout << graph[i][j] << " ";
			}
			cout << endl;
		}
	}

	int getCountOFVertices()
	{
		return countOfVertices;
	}

	Matrix floydWarshall(Matrix _startMatrix)
	{
		Matrix resultMat(5);
		resultMat = _startMatrix;
		for (int k = 0; k < countOfVertices; k++)
			for (int i = 0; i < countOfVertices; i++)
				for (int j = 0; j < countOfVertices; j++)
				{
					if (resultMat.GetFromMatrix(i, j) >
(resultMat.GetFromMatrix(i, k) + resultMat.GetFromMatrix(k, j)) && (resultMat.GetFromMatrix(k,
j) != inf) && (resultMat.GetFromMatrix(i, k) != inf))
					{
						int value = resultMat.GetFromMatrix(i, k) +
resultMat.GetFromMatrix(k, j);
						resultMat.AddToMatrix(i, j, value);
					}
				}
		return resultMat;
	}
};

void createMatrix(int _countOfVertices, Matrix &_mat)
{
	int value = 0;
	for (int i = 0; i < _countOfVertices; i++)
	{
		for (int j = 0; j < _countOfVertices; j++)
		{
			if (i == j)
				_mat.AddToMatrix(i, j, 0);
			else
			{
				cout << "Введите длину пути из " << i + 1 << " в " << j + 1 << ": ";
				cin >> value;
				if (value == 0)
					_mat.AddToMatrix(i, j, inf);
				else
					_mat.AddToMatrix(i, j, value);
			}
		}
	}
}

void printMatrix(Matrix _mat)
{
	for (int i = 0; i < _mat.getCountOFVertices(); i++)
	{
		for (int j = 0; j < _mat.getCountOFVertices(); j++)
		{
			if (_mat.GetFromMatrix(i, j) == inf)
				cout << setfill(' ') << setw(5) << "INF";
			else
				cout << setfill(' ') << setw(5) << _mat.GetFromMatrix(i, j);
```

```
                }
                cout << endl;
        }
}

int main()
{
        setlocale(LC_ALL, "Russian");
        int NumV = 0;
        bool _bool = true;
        char _char;
        do
        {
                cout << "Введите количество вершин: ";
                cin >> NumV;
                Matrix startMat(NumV);
                createMatrix(NumV, startMat);
                printMatrix(startMat);
                cout << "Матрица кротчайших путей: " << endl;
                printMatrix(startMat.floydWarshall(startMat));
                cout << "Ввести еще матрицу? (y/n): ";
                cin >> _char;
                cout << endl;
                if (_char != 'y')
                        _bool = false;
        } while (_bool);

        return 0;
}
```

```
D:\МПУ\2 курс\3 семестр\тпр\4\Debug\тпр4.exe
Введите количество вершин: 5
Введите длину пути из 1 в 2: 0
Введите длину пути из 1 в 3: 0
Введите длину пути из 1 в 4: 21
Введите длину пути из 1 в 5: 31
Введите длину пути из 2 в 1: 2
Введите длину пути из 2 в 3: 25
Введите длину пути из 2 в 4: 9
Введите длину пути из 2 в 5: 0
Введите длину пути из 3 в 1: 0
Введите длину пути из 3 в 2: 24
Введите длину пути из 3 в 4: 17
Введите длину пути из 3 в 5: 32
Введите длину пути из 4 в 1: 7
Введите длину пути из 4 в 2: 0
Введите длину пути из 4 в 3: 25
Введите длину пути из 4 в 5: 34
Введите длину пути из 5 в 1: 11
Введите длину пути из 5 в 2: 0
Введите длину пути из 5 в 3: 30
Введите длину пути из 5 в 4: 0
    0  INF  INF   21   31
    2    0   25    9  INF
  INF   24    0   17   32
    7  INF   25    0   34
   11  INF   30  INF    0
Матрица кротчайших путей:
    0   70   46   21   31
    2    0   25    9   33
   24   24    0   17   32
    7   49   25    0   34
   11   54   30   32    0
Ввести еще матрицу? (y/n):
```

Рисунок 7 - Результат работы программы

Ожидаемый результат работы программы на рисунке 8.

Матрица кратчайших путей в заданном графе				
0	70	46	21	31
2	0	25	9	33
24	24	0	17	32
7	49	25	0	34
11	54	30	32	0

Рисунок 8 - Ожидаемый результат

5. Вывод

Сравнив результат своей программы с ожидаемым результатом, могу убедится, что результат выполнения программы полностью совпадает с ожидаемым. Это говорит о корректности работы программы.