МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра АПУ

ОТЧЕТ

по лабораторной работе № 5

по дисциплине «Алгоритмы и структуры данных»

Тема: Нахождение кратчайшего пути в графе

Студентка гр. 1361	_______________	Горбунова Д. А.
Студентка гр. 1361	_______________	Токарева У. В.
Преподаватель	_______________	Беляев А. В.

Санкт-Петербург

2022

Цель работы: ознакомление с вариантами реализации алгоритмов на графах на примере задачи поиска кратчайшего пути в неориентированном графе.

Теоретическая часть

Пусть дан взвешенный связный неориентированный граф. Кратчайшим путем из одной вершины графа в другую будет называться путь, имеющий минимальную сумму весов ребер, входящих в него.

Задача поиска кратчайшего пути может быть сформулирована в виде:

- поиска кратчайшего пути между двумя конкретными вершинами
- поиска кратчайших путей от заданной вершины до всех остальных вершин графа
- поиска кратчайших путей между всеми вершинами графа попарно.

Существует несколько алгоритмов решения задачи. В данной работе будут рассмотрены:

- Алгоритм Беллмана-Форда (Ричард Беллман, Лестер Форд, 1956-1958 гг.)
- Алгоритм Дейкстры (Эдсгер Дейкстра, 1959 г.)

Алгоритм Беллмана-Форда

Алгоритм использует метод динамического программирования и формирует решение в виде квадратной матрицы, количество строк и столбцов которой равно количеству вершин графа. Ячейка на пересечении строки “m” и столбца “n” после окончания расчета содержит длину кратчайшего путь от заданной вершины до вершины “m”, при условии, что он (путь) содержит не более “n” ребер (считая номера столбцов с “0”).

Матрица заполняется по столбцам слева направо. Начальное заполнение содержит нулевой столбец, где для строки заданной (исходной) вершины установлено значение “0”, а для всех остальных строк – значение “∞” (на практике используется достаточно большая по величине константа).

На каждой итерации цикла заполняется один столбец по следующему алгоритму:

1) в заполняемый столбец копируются значения из предыдущего (соседнего слева) столбца (в качестве базовых значений)

2) перебираются все ребра графа и, если данное ребро позволяет улучшить (уменьшить) текущее значение в ячейке, соответствующей одному из концов данного ребра, то значение в ней заменяется на улучшенное

Параллельно матрице длин кратчайших путей, описанной выше, можно вести матрицу маршрутов, в которую в момент обновления значения в матрице длин записывается номер вершины, из которой “пришел” улучшенный путь.

Алгоритм Дейкстры

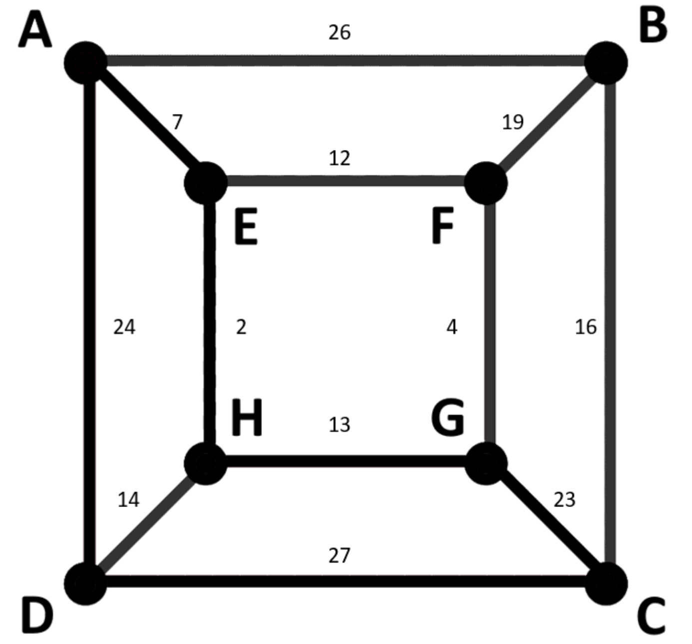
Алгоритм последовательно анализирует (“обрабатывает”) все вершины графа, начиная от заданной (исходной) следующим образом.

Изначально всем вершинам, кроме исходной, присваивается оценка длины кратчайшего пути, равная “∞”, (исходной вершине присваивается оценка “0”). Все вершины считаются “необработанными”.

В каждой итерации цикла среди необработанных вершин выбирается одна, имеющая наименьшую на текущий момент оценку кратчайшего пути от заданной (исходной). Анализируются все ребра, исходящие от нее в сторону необработанных вершин, и если какое-либо из ребер улучшает (уменьшает) текущую оценку, то эта оценка обновляется.

ИДЗ

Горбунова Дарья
Вариант 5



Алгоритм Форда-Беллмана

Изначальный вид матрицы:

A	0	0	0	0	–
B	∞	∞	∞	∞	
C	∞	∞	∞	∞	
D	∞	∞	∞	∞	
E	∞	∞	∞	∞	
F	∞	∞	∞	∞	
G	∞	∞	∞	∞	
H	∞	∞	∞	∞	

Добавляем значения ребер, соединенных с A, указывая путь до вершины

A	0	0	0	0	–
B	26,A	∞	∞	∞	A-B
C	∞	∞	∞	∞	
D	24,A	∞	∞	∞	A-D
E	7,A	∞	∞	∞	A-E
F	∞	∞	∞	∞	
G	∞	∞	∞	∞	
H	∞	∞	∞	∞	

Переписываем пути длины 1, параллельно добавляя пути длины 2. Если вес нового пути меньше существующего, заменяем его.

A	0	0	0	0	–
B	26,A	26,A	∞	∞	A-B
C	∞	42,B	∞	∞	A-B-C
D	24,A	24,A	∞	∞	A-D
E	7,A	7,A	∞	∞	A-E

F	∞	19,E	∞	∞	A-E-F
G	∞	23,F	∞	∞	A-E-F-G
H	∞	9,E	∞	∞	A-E-H

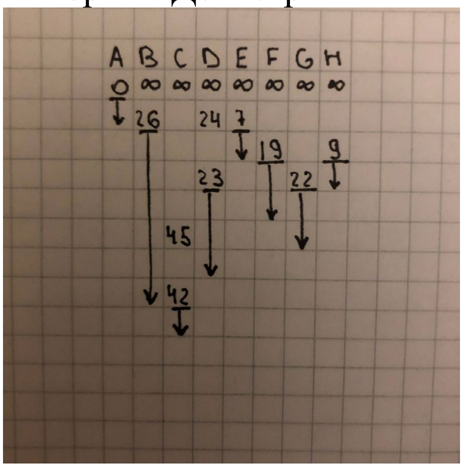
Ищем новые пути, вес которых меньше веса существующего пути. Если изменений нет, завершаем алгоритм.

A	0	0	0	0	–
B	26,A	26,A	26,A	∞	A-B
C	∞	42,B	42,B	∞	A-B-C
D	24,A	24,A	23,H	∞	A-E-H-D
E	7,A	7,A	7,A	∞	A-E
F	∞	19,E	19,E	∞	A-E-F
G	∞	23,F	22,H	∞	A-E-H-G
H	∞	9,E	9,E	∞	A-E-H

Ищем новые пути, вес которых меньше веса существующего пути. Если изменений нет, завершаем алгоритм.

A	0	0	0	0	–
B	26,A	26,A	26,A	26,A	A-B
C	∞	42,B	42,B	42,B	A-B-C
D	24,A	24,A	23,H	23,H	A-E-H-D
E	7,A	7,A	7,A	7,A	A-E
F	∞	19,E	19,E	19,E	A-E-F
G	∞	23,F	22,H	22,H	A-E-H-G
H	∞	9,E	9,E	9,E	A-E-H

Алгоритм Дейкстры



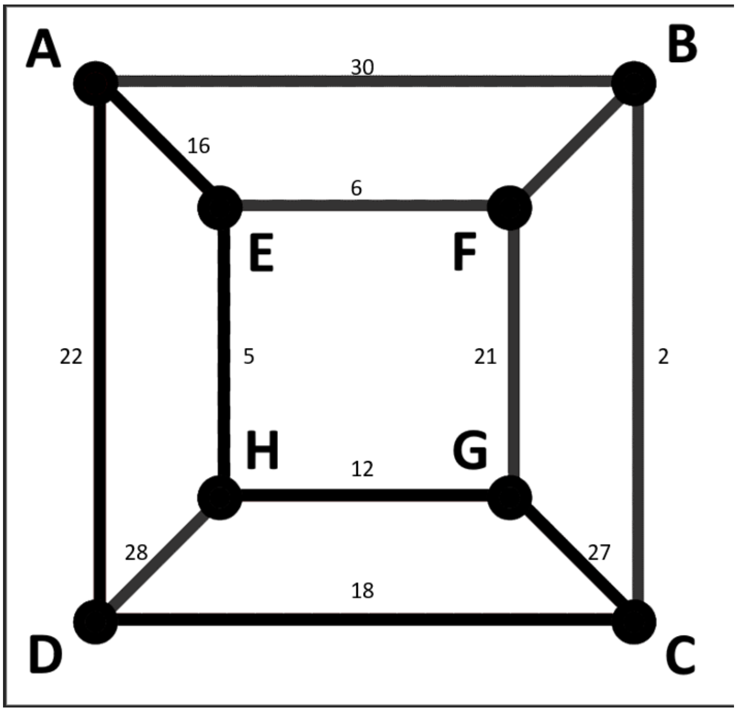
A	0	0	0	0	0	0	0	-
B	∞	26,A					26,A	AB
C	∞					45,G	42,B	ABC
D	∞	24,A		23,H			23,H	AEHD
E	∞	7,A					7,A	AE
F	∞	∞	19,E				19,E	AEF
G	∞	∞	∞	22,H			22,H	AEHG
H	∞	∞	9,E				9,E	AEH

Описание алгоритма:

В таблице на каждой иттерации (каждый столбец) указан минимальный путь до вершины и та вершина, из которой мы пришли. Предпоследний столбец окончательный итог минимальных весов, к которым мы пришли в ходе выполнения алгоритма.

Вывод: В результате нахождения кратчайших путей оба метода дали одинаковый результат, следовательно алгоритмы работают корректно.

Токарева Ульяна
Вариант 17



Алгоритм Форда-Беллмана

Изначальный вид матрицы:

A	0	0	0	0	0	–
B	∞	∞	∞	∞	∞	
C	∞	∞	∞	∞	∞	
D	∞	∞	∞	∞	∞	
E	∞	∞	∞	∞	∞	
F	∞	∞	∞	∞	∞	
G	∞	∞	∞	∞	∞	
H	∞	∞	∞	∞	∞	

Добавляем значения ребер, соединенных с А, указывая путь до вершины

A	0	0	0	0	0	–
B	∞	30,A	∞	∞	∞	A - B
C	∞	∞	∞	∞	∞	
D	∞	22,A	∞	∞	∞	A - D

E	∞	16,A	∞	∞	∞	A-E
F	∞	∞	∞	∞	∞	
G	∞	∞	∞	∞	∞	
H	∞	∞	∞	∞	∞	

Переписываем пути длины 1, параллельно добавляя пути длины 2. Если вес нового пути меньше существующего, заменяем его.

A	0	0	0	0	0	–
B	∞	30,A	30,A	∞	∞	A-B
C	∞	∞	32,B	∞	∞	A-B-C
D	∞	22,A	22,A	∞	∞	A-D
E	∞	16,A	16,A	∞	∞	A-E
F	∞	∞	22,E	∞	∞	A-E-F
G	∞	∞	43,F	∞	∞	A-E-F-G
H	∞	∞	21,E	∞	∞	A-E-H

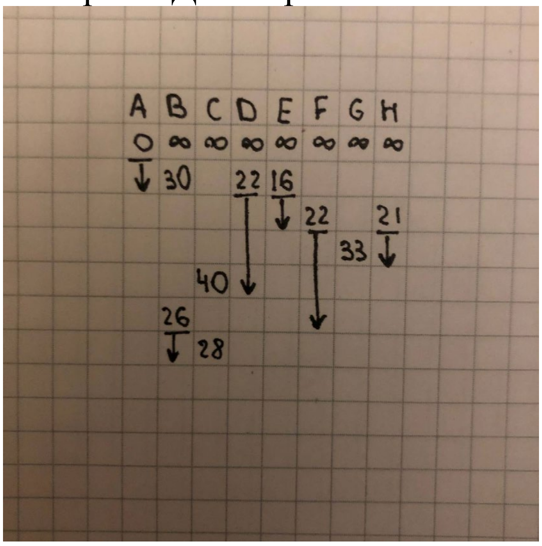
Ищем новые пути, вес которых меньше веса существующего пути.

A	0	0	0	0	0	–
B	∞	30,A	30,A	26,F	∞	A-E-F-B
C	∞	∞	32,B	28,B	∞	A-E-F-B-C
D	∞	22,A	22,A	22,A	∞	A-D
E	∞	16,A	16,A	16,A	∞	A-E
F	∞	∞	22,E	22,E	∞	A-E-F
G	∞	∞	43,F	33,H	∞	A-E-H-G
H	∞	∞	21,E	21,E	∞	A-E-H

Ищем новые пути, вес которых меньше веса существующего пути. Если изменений нет, завершаем алгоритм.

A	0	0	0	0	0	–
B	∞	30,A	30,A	26,F	26,F	A-E-F-B
C	∞	∞	32,B	28,B	28,B	A-E-F-B-C
D	∞	22,A	22,A	22,A	22,A	A-D
E	∞	16,A	16,A	16,A	16,A	A-E
F	∞	∞	22,E	22,E	22,E	A-E-F
G	∞	∞	43,F	33,H	33,H	A-E-H-G
H	∞	∞	21,E	21,E	21,E	A-E-H

Алгоритм Дейкстры



A	0	0	0	0	0	0	0	-
B	∞	30,A				26,F	26,F	AEFB
C	∞	∞	∞	∞	40,D		28,B	AEFBA
D	∞	22,A					22,A	AD
E	∞	16,A					16,A	AE
F	∞	∞	22,E				22,E	AEF
G	∞	∞	∞	33,H			33,H	AEHG
H	∞	∞	21,E				21,E	AEH

Описание алгоритма:

В таблице на каждой иттерации (каждый столбец) указан минимальный путь до вершины и та вершина, из которой мы пришли. Предпоследний столбец окончательный итог минимальных весов, к которым мы пришли в ходе выполнения алгоритма.

Вывод: В результате нахождения кратчайших путей оба метода дали одинаковый результат, следовательно алгоритмы работают корректно.

Исходный код программы Algoritm_Ford_Bellman

```
#include <iostream>

#include <time.h>

#include <vector>

#include <algorithm>


using namespace std;

struct Edge{

    int vertex_begin;

    int vertex_end;

    int weight;};

struct Way{

    int vertex;

    int weigth_edge;};

void Print_Edges (int ** graph, int flag){

    cout << "Edges \t Weight \t Number"<<  endl;

    int count = 1;

    for (int v1=0; v1 < flag; v1++)

        for (int v2= v1 + 1; v2 < flag; v2++)

            if (graph[v1][v2] > 0)

                cout << v1 << " -- " << v2 << ": \t" << graph[v1][v2] << " \t" << count++<< endl;

    return;}
```

```
int Generator_graph(int **graph, int flag){
    int Curr_time = time(NULL);
    if (Curr_time == -1 ){
        cout << "Error time!"<< endl;
        return 1;
    }
    srand (Curr_time);
    int count_edges=0;
    for (int v1=0; v1 < flag-1; v1++)
        for (int v2= v1 + 1; v2 < flag; v2++){
            if ((v1 != v2 && graph[v1][v2] == 0 ) && rand() % 100 < 34 ) {
                graph[v1][v2] = graph[v2][v1]= rand() % 50 + 1 ;
            }
            else graph[v1][v2] = graph[v2][v1] = -1;}
    int critical_weight=200;
    for (int v1=0; v1<flag-1; v1++){
        int v2=v1+1;
        if (graph[v1][v2] == -1){
            graph[v1][v2] = graph[v2][v1]= critical_weight;
            count_edges++;}}
    return count_edges;
}
```

```
void Add_Struct_Edges (int ** graph, Edge* edges, int count, int flag ) {
    int k = 0;
    for (int v1 = 0; v1 < flag; v1++)
        for (int v2 = v1 + 1; v2 < flag; v2++)
            if (graph[v1][v2] > 0) {
                edges[k].weight = graph[v1][v2];
                edges[k].vertex_begin = v1;
                edges[k++].vertex_end = v2;
            }}
void Way_graph(Way** way, int ver_i,int flag){
    int edge;
    printf("Way: \t%d", ver_i);
    for (int i=flag -1; i> 0 ; i--)
        if (way[ver_i][i].vertex!=-1)
            cout << "("<<way[ver_i][i].weigth_edge<<") "<< way[ver_i][i].vertex;
    cout << endl;
    return;}
void Alg_Ford_Bellman(int * start_vertex, int ed, Edge * graph, Way ** MyWay, int flag){
    int min_way [flag][flag];
    for (int v1=0;v1<flag;v1++)
        for (int v2=0; v2<flag;v2++){
            if (*start_vertex == v1) min_way[v1][v2] = 0;
```

```
            else min_way[v1][v2] =  5000;
            if (v2!=0) MyWay[v1][v2].vertex = -1;
            else MyWay[v1][0].vertex =v1;
            MyWay[v1][v2].weigth_edge = 0; }
    for (int v1=0;v1<flag;v1++) {
        for (int v2 = 0; v2 < flag; v2++)
            min_way[v2][v1] = min_way[v2][v1 - 1];
        for (int v2 = 0; v2 < flag; v2++)
            for (int k = 0; k < ed; k++)
                if (graph[k].vertex_begin == v2 || graph[k].vertex_end == v2) {
                    if (min_way[graph[k].vertex_end][v1] > min_way[graph[k].vertex_begin][v1 - 1] + graph[k].weight) {
                        min_way[graph[k].vertex_end][v1] = min_way[graph[k].vertex_begin][v1 - 1] + graph[k].weight;
                        MyWay[graph[k].vertex_end][v1].weigth_edge = graph[k].weight;
                        MyWay[graph[k].vertex_end][v1].vertex = graph[k].vertex_begin;}
                    else if (min_way[graph[k].vertex_begin][v1] > min_way[graph[k].vertex_end][v1 - 1] +
graph[k].weight){
                        min_way[graph[k].vertex_begin][v1] = min_way[graph[k].vertex_end][v1 - 1] + graph[k].weight;
                        MyWay[graph[k].vertex_begin][v1].weigth_edge = graph[k].weight;
                        MyWay[graph[k].vertex_begin][v1].vertex = graph[k].vertex_end;}}}
    for (int i=0; i<flag;i++){
        for (int j=0; j<flag; j++)
            if (min_way[i][j] >0 )
```

```
            cout << min_way[i][j]<<", ";
        cout << endl;}
    cout << endl;
    return;}
int main(){
    int n;
    cout << "Enter the number of verties:";
    cin >> n;
    int** graph = new int*[n];
    for(int v1 = 0; v1<n; v1++){
        graph[v1] = new int[n]; }
    for (int v1=0; v1<n; v1++)
        for (int v2=0; v2<n; v2++)
            graph[v1][v2]=0;
    int count_edges;
    count_edges=Generator_graph(graph, n);
    Edge* currEdge = new Edge [count_edges];
    Print_Edges(graph,n);
    Add_Struct_Edges(graph, currEdge, count_edges, n);
    int start_vertex;
    cout <<"Enter the start vertex:";
    cin >> start_vertex;
```

```
Way ** MyWay = new Way * [n];
for(int i = 0; i<n; i++)
    MyWay[i] = new Way [n];
Alg_Ford_Bellman(&start_vertex, count_edges, currEdge, MyWay, n);
for (int i=0; i<n;i++)
    if (i!=start_vertex)
        Way_graph(MyWay, i, n);
delete(graph);
delete (currEdge);
delete (MyWay);
return 0;}
```

Результат работы программы.

```
Enter the number of verties:15
Edges          Weight          Number
0 -- 1:          200     1
0 -- 3:          16      2
0 -- 9:          33      3
0 -- 11:         39      4
1 -- 2:          200     5
1 -- 5:          25      6
1 -- 6:          50      7
1 -- 7:          22      8
1 -- 10:         26      9
1 -- 11:         15      10
1 -- 13:         30      11
1 -- 14:         49      12
2 -- 3:          200     13
2 -- 4:          27      14
2 -- 7:          10      15
2 -- 8:          35      16
2 -- 9:          5       17
2 -- 13:         31      18
3 -- 4:          200     19
3 -- 5:          41      20
3 -- 6:          48      21
3 -- 8:          5       22
3 -- 9:          21      23
3 -- 10:         24      24
4 -- 5:          17      25
```

```
4 -- 11:         18      26
4 -- 13:         1       27
5 -- 6:          13      28
5 -- 9:          8       29
5 -- 10:         18      30
6 -- 7:          200     31
6 -- 8:          30      32
6 -- 9:          7       33
6 -- 11:         13      34
6 -- 12:         20      35
6 -- 14:         26      36
7 -- 8:          37      37
7 -- 14:         45      38
8 -- 9:          200     39
8 -- 12:         6       40
9 -- 10:         50      41
9 -- 13:         13      42
10 -- 11:        200     43
10 -- 14:        24      44
11 -- 12:        50      45
11 -- 13:        25      46
11 -- 14:        9       47
12 -- 13:        200     48
13 -- 14:        200     49
```

Рисунок 1 – Вывод созданного графа

```
Enter the start vertex:5

50, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28,
5000, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12,
16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16,
5000, 30, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21,
13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,

200, 67, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22,
30, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21,
7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,
5000, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31,
13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20,
5000, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20,
26, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22,
```

Рисунок 2 – Вывод матрицы длин кратчайших путей.

```
Way:    0
Way:    1(15) 11
Way:    2(5) 9
Way:    3
Way:    4(1) 13(17) 5
Way:    6
Way:    7(10) 2(37) 8
Way:    8(5) 3
Way:    9
Way:    10(18) 5
Way:    11
Way:    12
Way:    13(13) 9
Way:    14(9) 11
```

Рисунок 3 – Вывод списка восстановленных кратчайших путей до каждой вершины, кроме исходной.

ВЫВОД

В ходе выполнения данной лабораторной работы был изучен алгоритм Форда-Беллмана для нахождения кратчайшего пути в графе. Программа, выполняющая построение графа, была дополнена алгоритмом Форда-Беллмана.