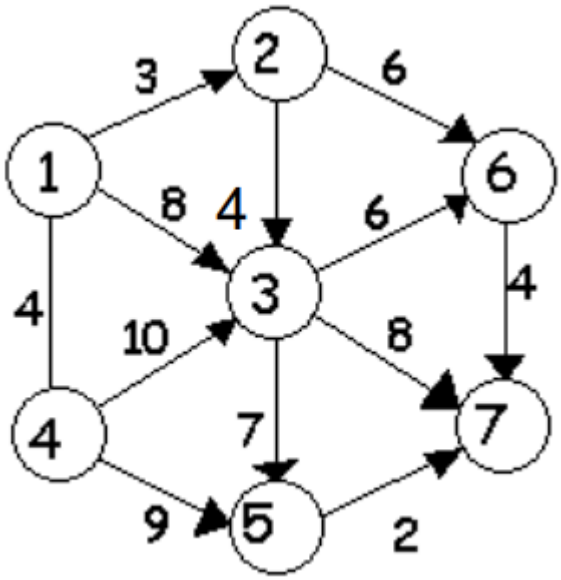
Отчёт по лабораторной работе №3

Выполнил: Трусов Михаил Павлович, ПИН-22

Задание 1. Найти кратчайший путь на графе между двумя вершинами методом динамического программирования (вариант 13)



Результат выполнения программы:

```
Кратчайшие пути от вершины 1: {1=0, 2=3, 3=7, 4=4, 5=13, 6=9, 7=13}
Кратчайший путь от вершины 1 до вершины 7: [1, 2, 6, 7]
```

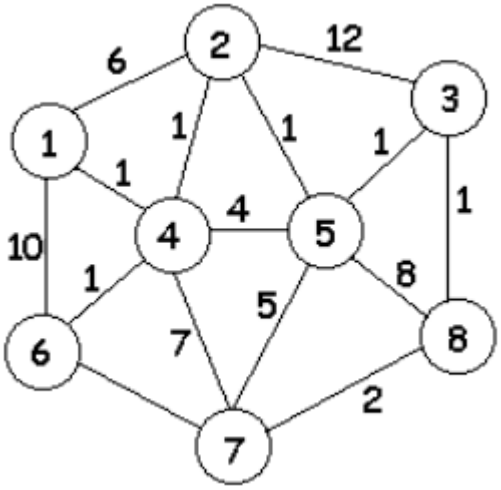
Программный код:

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.Map;

class Edge implements Comparable<Edge> {
    int start;
    int end;
    int weight;

    public Edge(int start, int end, int weight) {
        this.start = start;
        this.end = end;
        this.weight = weight;
    }

    @Override
    public int compareTo(Edge edge) {
        return Integer.compare(weight, edge.weight);
    }
```

```
}

class Graph {
    Edge[] edges;
    int nodes;


    public Graph(Edge[] edges, int nodes) {
        this.edges = new Edge[edges.length];
        System.arraycopy(edges, 0, this.edges, 0, edges.length);
        this.nodes = nodes;
    }

    public boolean ifCanCalculateV(Map<Integer, Integer> found, int
v) {
        ArrayList<Integer> needing = new ArrayList<>();
        for (int i = 0; i < edges.length; i++) {
            if (edges[i].end == v) {
                needing.add(edges[i].start);
            }
        }

        for (int i = 0; i < needing.size(); i++) {
            if (!found.containsKey(needing.get(i))) {
                return false;
            }
        }
        return true;
    }

    public int[] calculatev(Map<Integer, Integer> found, int v) {
        int minweight = Integer.MAX_VALUE;
        int from = 0;
        for (int i = 0; i < edges.length; i++) {
            if (edges[i].end == v && edges[i].weight +
found.get(edges[i].start) < minweight) {
                minweight = edges[i].weight +
found.get(edges[i].start);
                from = edges[i].start;
            }
        }
        return new int[]{minweight, from};
    }


    public void printWay(Map<Integer, Integer> fromwhere, int
startv, int finishv) {
        LinkedList<Integer> way = new LinkedList<>();
        int temp = finishv;
        way.addFirst(temp);
        while (temp != startv) {
            temp = fromwhere.get(temp);
            way.addFirst(temp);
        }
```

```
        System.out.println("Кратчайший путь от вершины " + startv +
" до вершины " + finishv + ": " + way);
    }

    public ArrayList<Integer> dynamicalProgramming(int startv, int
finishv) {
        Map<Integer, Integer> fromwhere = new HashMap<>();
        Map<Integer, Integer> found = new HashMap<>();
        found.put(startv, 0);
        while (found.size() < nodes) {
            for (int i = 1; i <= nodes; i++) {
                if (!found.containsKey(i)) {
                    if (ifCanCalculateV(found, i)) {
                        int[] temp = calculatev(found, i);
                        found.put(i, temp[0]);
                        fromwhere.put(i, temp[1]);
                    }
                }
            }
        }
        System.out.println("Кратчайшие пути от вершины " + startv +
": " + found);
        printWay(fromwhere, startv, finishv);
        return new ArrayList<>();
    }

}

public class Main {
    public static void main(String[] args) {
        Edge edge1 = new Edge(1, 2, 3);
        Edge edge2 = new Edge(1, 3, 8);
        Edge edge3 = new Edge(1, 4, 4);
        Edge edge4 = new Edge(2, 3, 4);
        Edge edge5 = new Edge(2, 6, 6);
        Edge edge6 = new Edge(3, 6, 6);
        Edge edge7 = new Edge(3, 7, 8);
        Edge edge8 = new Edge(3, 5, 7);
        Edge edge9 = new Edge(4, 3, 10);
        Edge edge10 = new Edge(4, 5, 9);
        Edge edge11 = new Edge(5, 7, 2);
        Edge edge12 = new Edge(6, 7, 4);

        Graph graph = new Graph(new Edge[]{edge1, edge2, edge3,
edge4, edge5, edge6, edge7, edge8, edge9, edge10, edge11, edge12},
7);
        graph.dynamicalProgramming(1, 7);


    }
}
```

Задание 2. Найти кратчайший путь на графе между тремя парами вершин методом Дейкстры. (вариант 13)



Результат выполнения программы:

```
Длина путей от вершины 1 до вершин: {1=0, 2=2, 3=4, 4=1, 5=3, 6=2, 7=3, 8=5}
Путь от вершины 1 до вершины 7: [1, 4, 6, 7]
Длина этого пути: 3

Длина путей от вершины 6 до вершин: {1=2, 2=2, 3=4, 4=1, 5=3, 6=0, 7=1, 8=3}
Путь от вершины 6 до вершины 3: [6, 4, 2, 5, 3]
Длина этого пути: 4

Длина путей от вершины 4 до вершин: {1=1, 2=1, 3=3, 4=0, 5=2, 6=1, 7=2, 8=4}
Путь от вершины 4 до вершины 5: [4, 2, 5]
Длина этого пути: 2
```

Программный код:

```java
import java.util.*;

class Edge implements Comparable<Edge> {
    int start;
    int end;
    int weight;

    public Edge(int start, int end, int weight) {
        this.start = start;
        this.end = end;
        this.weight = weight;
    }

    @Override
    public int compareTo(Edge edge) {
        return Integer.compare(weight, edge.weight);
    }
}
```

```
class Graph {
    Edge[] edges;
    int nodes;


    public Graph(Edge[] edges, int nodes) {
        this.edges = new Edge[edges.length];
        System.arraycopy(edges, 0, this.edges, 0, edges.length);
        this.nodes = nodes;
    }

    // вспомогательный метод, возвращающий все смежные ребра графу takenV
    public ArrayList<Edge> getNewEdges(Map<Integer, Integer> ways) {
        ArrayList<Integer> takenV = new ArrayList<>();
        ArrayList<Integer> notTakenV = new ArrayList<>();

        for (int i = 1; i <= nodes; i++) {
            if (ways.containsKey(i)) {
                takenV.add(i);
            } else {
                notTakenV.add(i);
            }
        }

        ArrayList<Edge> neighbours = new ArrayList<>();
        for (int i = 0; i < edges.length; i++) {
            if ((takenV.contains(edges[i].start) &&
notTakenV.contains(edges[i].end)) || (takenV.contains(edges[i].end) &&
notTakenV.contains(edges[i].start))) {
                neighbours.add(edges[i]);
            }
        }
        return neighbours;
    }

    public void dijkstra(int vstart, int vfinish) {
        Map<Integer, Integer> ways = new HashMap<>();
        Map<Integer, Integer> finalway = new HashMap<>();
        LinkedList<Integer> wayorder = new LinkedList<>();
        ways.put(vstart, 0);


        while (ways.size() < nodes) {
            ArrayList<Edge> neighbours = getNewEdges(ways);
            int minweight = Integer.MAX_VALUE;
            Edge minEdge = new Edge(0, 0, 0);
            for (int i = 0; i < neighbours.size(); i++) {
                int tempstart;
                if (ways.containsKey(neighbours.get(i).start)) {
                    tempstart = neighbours.get(i).start;
                } else {
                    tempstart = neighbours.get(i).end;
                }
                if (ways.get(tempstart) + neighbours.get(i).weight <
minweight) {
                    minweight = ways.get(tempstart) +
neighbours.get(i).weight;
                    minEdge = neighbours.get(i);
                }

            }
            if (ways.containsKey(minEdge.start)) {
                ways.put(minEdge.end, minweight);
                finalway.put(minEdge.end, minEdge.start);
```

```
            } else {
                ways.put(minEdge.start, minweight);
                finalway.put(minEdge.start, minEdge.end);
            }
        }


        int temp = vfinish;
        wayorder.addFirst(temp);
        while (temp != vstart) {
            temp = finalway.get(temp);
            wayorder.addFirst(temp);
        }

        System.out.println("Длина путей от вершины " + vstart + " до вершин:
" + ways);
        System.out.println("Путь от вершины " + vstart + " до вершины " +
vfinish + ": " + wayorder);
        System.out.println("Длина этого пути: " + ways.get(vfinish) + "\n");
    }
}

public class Main {
    public static void main(String[] args) {
        Edge edge1 = new Edge(1, 2, 6);
        Edge edge2 = new Edge(1, 4, 1);
        Edge edge3 = new Edge(1, 6, 10);
        Edge edge4 = new Edge(2, 4, 1);
        Edge edge5 = new Edge(2, 5, 1);
        Edge edge6 = new Edge(2, 3, 12);
        Edge edge7 = new Edge(3, 5, 1);
        Edge edge8 = new Edge(3, 8, 1);
        Edge edge9 = new Edge(4, 5, 4);
        Edge edge10 = new Edge(4, 6, 1);
        Edge edge11 = new Edge(4, 7, 7);
        Edge edge12 = new Edge(5, 7, 5);
        Edge edge13 = new Edge(5, 8, 8);
        Edge edge14 = new Edge(6, 7, 1);
        Edge edge15 = new Edge(7, 8, 2);


        Graph graph = new Graph(new Edge[]{edge1, edge2, edge3, edge4, edge5,
edge6, edge7, edge8, edge9, edge10, edge11, edge12, edge13, edge14, edge15},
8);
        graph.dijkstra(1, 7);
        graph.dijkstra(6, 3);
        graph.dijkstra(4, 5);
    }
}
```