# Lab: Graphs –Shortest Path and MST

Please submit your solutions (source code) to all below-described problems in [Judge](https://judge.softuni.org/Contests/3464).

## Shortest Path in Unweighted Graph

You will be given a graph from the console your task is to find the shortest path and print it back on the console. The first line will be the number of Nodes - **N** the second one the number of Edges - **E**, then on each **E** line theedge in form **{destination} – {source}.** On the last two lines, you will read the start node and the end node.

Print on the first line the **length** of the shortest path and the second the **path** **itself,** see the examples below.

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| 8  10  1 2  1 4  2 3  4 5  5 8  5 6  5 7  5 8  6 7  7 8  1  7 | Shortest path length is: 3  1 4 5 7 |
| 11  11  1 5  1 4  5 7  7 8  8 2  2 3  3 4  4 1  6 2  9 10  11 9  6  3 | Shortest path length is: 2  6 2 3 |

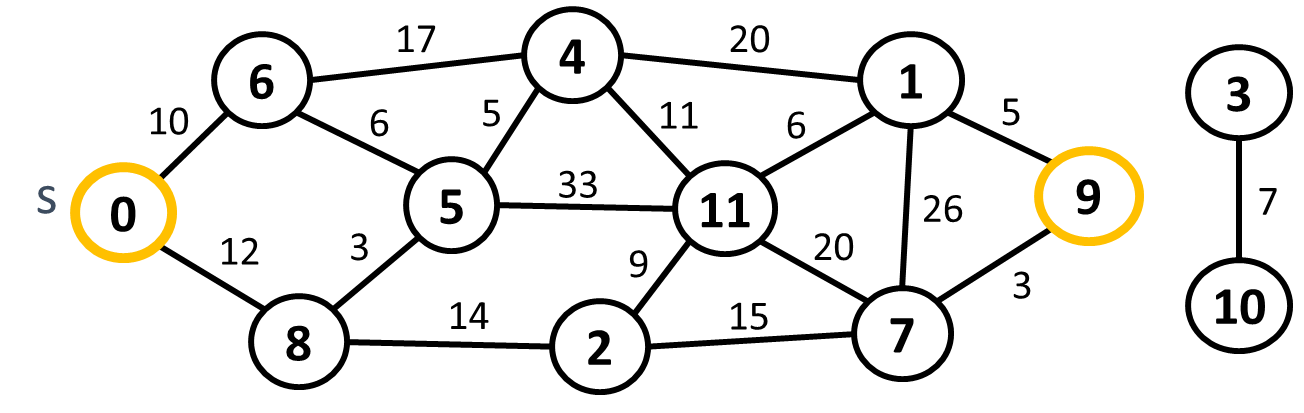
## Dijkstra's Algorithm

Finding the **shortest path between two nodes** in an unweighted graph is done by applying simple BFS. When we’re working with weighted graphs though, things get more complicated. **Dijkstra’s algorithm** is one of the most famous ones that solve this task.

A classical application of the shortest path algorithm might be to find the shortest path between two towns on a map holding towns connected with roads where each road holds the distance between two towns.

In this problem, we’ll try to implement the optimized version of Dijkstra’s algorithm using a priority queue.

Example: Find the shortest path between **node 0** and **node 9** in the following weighted undirected graph:



### Input

* On the first, you will receive an integer – e – number of edges.
* On the next e lines, you will receive an edge in the following format: "{start}, {end}, {weight}".
* On the next line, you will receive a **start node**.
* On the last line, you will receive an **end node**.

### Output

* Print the cost of the shortest path.
* Print all nodes that form the shortest path.
* If the end node can’t be reached from the start node, then you have to print: "There is no such path."

### Example

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Graph** |
| 18  0, 6, 10  0, 8, 12  6, 4, 17  6, 5, 6  8, 5, 3  5, 4, 5  5, 11, 33  8, 2, 14  2, 11, 9  2, 7, 15  4, 1, 20  4, 11, 11  11, 1, 6  11, 7, 20  1, 9, 5  1, 7, 26  7, 9, 3  3, 10, 7  0  9 | 42  0 8 5 4 11 1 9 |  |
| 18  0, 6, 10  0, 8, 12  6, 4, 17  6, 5, 6  8, 5, 3  5, 4, 5  5, 11, 33  8, 2, 14  2, 11, 9  2, 7, 15  4, 1, 20  4, 11, 11  11, 1, 6  11, 7, 20  1, 9, 5  1, 7, 26  7, 9, 3  3, 10, 7  0  10 | There is no such path. |  |

## Bellman-Ford

Find the **shortest** **path in a graph** and print it as a sequence **from S vertex to D vertex**.

Implement the Bellman-Ford algorithm.

### Input

The input comes from the console.

* On the first line, you will receive an integer – n – the number of **nodes**.
* On the next line, you will receive an integer – e – the number of edges.
* On the next e lines, you will receive an edge in the following format:"{from} {to} {weight}".
* On the last two lines, you will receive source and destination nodes.

### Output

* Print "Negative Cycle Detected"if detect a negative cycle.
* Otherwise, print the **shortest path** separated by a space and the total **distance**.

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| 6  8  1 2 8  1 3 10  2 4 1  3 6 2  4 3 -4  4 6 -1  6 5 -2  5 3 1  1  6 | 1 2 4 3 6  7 |
| 4  4  1 2 1  2 3 -1  3 4 -1  4 1 -1  1  4 | Negative Cycle Detected |

## Kruskal's Algorithm

If we have a weighted undirected graph, we can extract a sub-graph where all nodes (vertices) of the original graph are connected by edges without any cycles. This is referred to as a **spanning tree**. A **minimum spanning tree (MST)** is the spanning tree with the smallest weight (several MST could exist in some graphs).

For example, a cable operator wants to connect its customers to a **cable network**. The homes of the customers are connected by streets (edges) with different lengths (weights). To find out how to connect all homes to its network most efficiently (least distance covered) you need to find the **MST**.

One simple algorithm to find the MST of a given graph is [**Kruskal’s algorithm**](https://en.wikipedia.org/wiki/Kruskal%27s_algorithm).

### Input

* On the first line you will receive e – an integer – number of edges that you have to read.
* On the next e lines, you will receive an edge in the following format: "{firstNode}, {seconNode}, {weight}".

### Example

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Details** |
| 11  0, 3, 9  0, 5, 4  0, 8, 5  1, 4, 8  1, 7, 7  2, 6, 12  3, 5, 2  3, 6, 8  3, 8, 20  4, 7, 10  6, 8, 7 | 3 - 5  0 - 5  0 - 8  1 - 7  6 - 8  1 - 4  2 - 6 | **Graph:**    **MST:** |

## Prim's Algorithm

If we have a weighted undirected graph, we can extract a sub-graph where all nodes (vertices) of the original graph are connected by edges without any cycles. This is referred to as a spanning tree. A minimum spanning tree (MST) is the spanning tree with the smallest weight (several MST could exist in some graphs).

For example, a cable operator wants to connect its customers to a cable network. The homes of the customers are connected by streets (edges) with different lengths (weights). To find out how to connect all homes to its network most efficiently (least distance covered) you need to find the MST.

One simple algorithm to find the MST of a given graph is Prim’s algorithm.

### Input

* On the first line, you will receive e – an integer – number of edges that you have to read.
* On the next e lines, you will receive an edge in the following format: "{firstNode}, {seconNode}, {weight}".

### Output

* Print all edges part of the minimum spanning tree in the following format: "{first} - {second}".
* The order of the output **doesn’t matter**.

### Example

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Details** |
| 11  0, 2, 5  2, 4, 7  4, 5, 12  0, 1, 4  1, 3, 2  0, 3, 9  2, 3, 20  3, 4, 8  6, 7, 8  6, 8, 10  7, 8, 7 | 0 - 1  1 - 3  0 - 2  2 - 4  4 - 5  6 - 7  7 - 8 | **Graph:**    **MST:** |