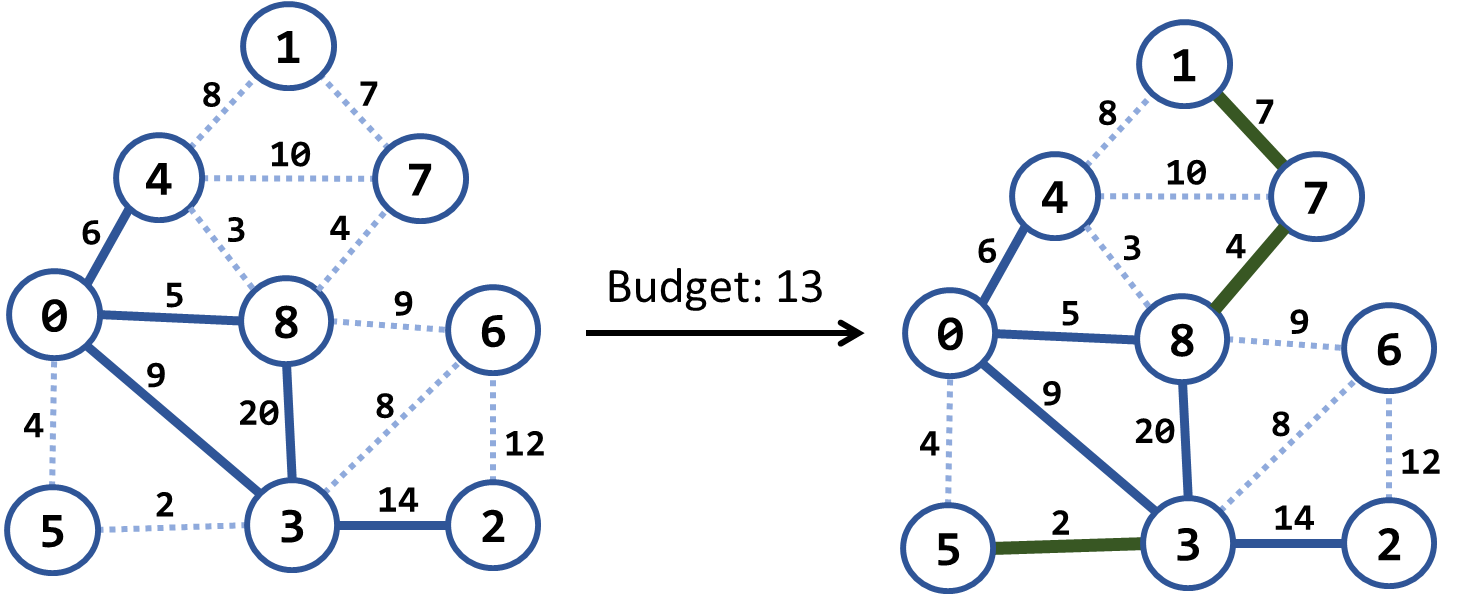
**Exercise: Graphs Bellman Ford, Longest Path in DAG, Dijkstra and MST**

This document defines the lab for the ["Algorithms – Advanced (Java)" course @ Software University](https://softuni.bg/trainings/3812/algorithms-advanced-with-java-september-2022). Please submit your solutions (source code) to all below-described problems in [Judge](https://judge.softuni.bg/Contests/2494/Graphs-Bellman-Ford-Longest-Path-in-DAG-Exercise).

# Cable Network

A cable networking company plans to extend its existing **cable network** by connecting as many customers as possible within a **fixed budget limit**. The company has calculated the **cost** of building some prospective connections. You are given the existing cable network (a set of **customers** and **connections** between them) along with the **estimated connection costs** between some pairs of customers and prospective customers. A customer can only be connected to the network via a direct connection with an already connected customer. Example:



In the above example, we have an existing cable network (the solid blue lines), the estimated costs for connecting some of the customers (dotted blue lines), and a budget limit of 20. Within this budget, the company can connect 3 new customers by the following new connections (solid green lines): {3 → 5}, {8 → 7} and {7 → 1}. The total cost for those new connections will be 2 + 4 + 7 = 13, which fits in the budget limit of 20. No more customers can be connected within this budget limit. Note that each edge, at the time of its addition to the network, connects a new customer with an existing one.

### Examples

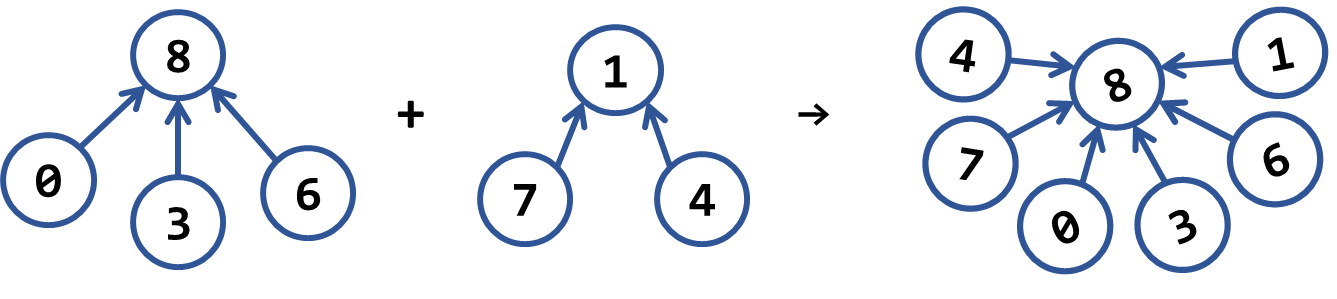
|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Picture (Before)** | **Output** | **Picture (After)** |
| Budget: 20  Nodes: 9  Edges: 15  1 4 8  4 0 6 connected  1 7 7  4 7 10  4 8 3  7 8 4  0 8 5 connected  8 6 9  8 3 20 connected  0 5 4  0 3 9 connected  6 3 8  6 2 12  5 3 2  3 2 14 connected |  | Budget used: 13 |  |
| Budget: 7  Nodes: 4  Edges: 5  0 1 9  0 3 4 connected  3 1 6  3 2 11 connected  1 2 5 |  | Budget used: 5 |  |
| Budget: 20  Nodes: 8  Edges: 16  0 1 4  0 2 5  0 3 1 connected  1 2 8  1 3 2  2 3 3  2 4 16  2 5 9  3 4 7  3 5 14  4 5 12  4 6 22  4 7 9  5 6 6  5 7 18  6 7 15 |  | Budget used: 12 |  |

### Hints

Modify Prims's algorithm. Until the budget is spent, connect the smallest possible edge from the connected node to the non-connected node.

# Modified Kruskal Algorithm

Implement Kruskal's algorithm by keeping the **disjoint sets** in a **forest** where each node holds a **parent + children**. Thus, when two sets need to be merged, the result can be easily optimized to have two levels only: root and leaves. When two **trees are merged**, all nodes from the second (its root + root's children) should be attached to the first tree's root node:

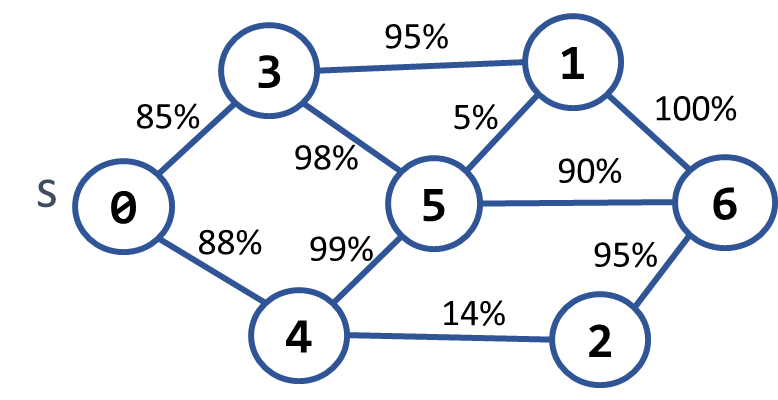
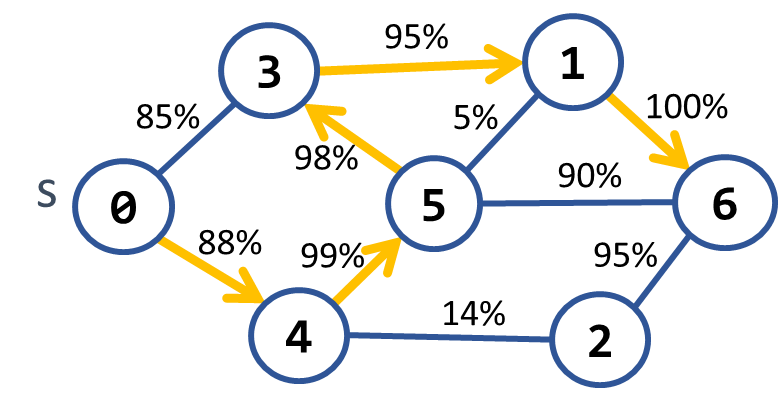


### Examples

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Picture (Graph)** | **Output** | **Picture (MST)** |
| Nodes: 4  Edges: 5  0 1 9  0 3 4  3 1 6  3 2 11  1 2 5 |  | Minimum spanning forest weight: 15  (0 3) -> 4  (1 2) -> 5  (1 3) -> 6 |  |
| Nodes: 9  Edges: 15  1 4 8  4 0 6  1 7 7  4 7 10  4 8 3  7 8 4  0 8 5  8 6 9  8 3 20  0 5 4  0 3 9  6 3 8  6 2 12  5 3 2  3 2 14 |  | Minimum spanning forest weight: 45  (3 5) -> 2  (4 8) -> 3  (0 5) -> 4  (8 7) -> 4  (0 8) -> 5  (1 7) -> 7  (3 6) -> 8  (6 2) -> 12 |  |
| Nodes: 8  Edges: 16  0 1 4  0 2 5  0 3 1  1 2 8  1 3 2  2 3 3  2 4 16  2 5 9  3 4 7  3 5 14  4 5 12  4 6 22  4 7 9  5 6 6  5 7 18  6 7 15 |  | Minimum spanning forest weight: 37  (0 3) -> 1  (1 3) -> 2  (2 3) -> 3  (5 6) -> 6  (3 4) -> 7  (2 5) -> 9  (4 7) -> 9 |  |

# Most Reliable Path

We have a set of **towns,** and some of them are connected by **direct paths**. Each path has a coefficient of reliability (in percentage): the chance to pass without incidents. Your goal is to compute the **most reliable path** between two given nodes. Assume all percentages will be integer numbers and round the result to the second digit after the decimal separator. For example, let's consider the graph below:

The **most reliable path** **between 0 and 6** is shown on the right: 0 **→** 4 **→** 5 **→** 3 **→** 1 **→** 6. Its cost = 88% \* 99% \* 98% \* 95% \* 100% = **81.11%**. The table below shows the optimal reliability coefficients for all paths starting from node 0:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **v** | **0** | **1** | **2** | **3** | **4** | **5** | **6** |
| **reliability[s → d]** | 100% | 81.11% | 77.05% | 85.38% | 88% | 87.12% | 81.11% |

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Picture** |
| Nodes: 7  Path: 0 – 6  Edges: 10  0 3 85  0 4 88  3 1 95  3 5 98  4 5 99  4 2 14  5 1 5  5 6 90  1 6 100  2 6 95 | Most reliable path reliability: 81.11%  0 -> 4 -> 5 -> 3 -> 1 -> 6 |  |
| Nodes: 4  Path 0 – 1  Edges: 4  0 1 94  0 2 97  2 3 99  1 3 98 | Most reliable path reliability: 94.11%  0 -> 2 -> 3 -> 1 |  |

### Hints

Modify Dijkstra's algorithm.

# Cheap Town Tour

You now live in a new country, and you want to start a tour and **visit every town** in the country, and since you are new in the country, your finances are **minimalized**, so you want your trip to be as **cheap** as possible. You will be given the **amount** of the **cities** on the first line, then the amount of the **roads** (**n**), and on the next **n** lines, you will receive which tows the road connects and the cost to use it.

Assume you can **start from any** town, and your target is to **visit every one** of them at the **minimum** cost.

### Input

* On the **first line,** you will be given the **number of** the **towns**
* On the **second** **line**, you will be given the **amount of streets** (**n**)
* On the **next n** **lines** you will be given a connection in the format: **"{first} -> {second} -> {cost}"**

### Output

* Print the **total cost** of the road you have chosen in the format: **"Total cost: {totalCost}"**

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comment** |
| 4  5  0 - 1 - 10  0 - 2 - 6  0 - 3 - 5  1 - 3 - 15  2 - 3 - 4 | Total cost: 19 | The cheapest way to visit all the towns is using the roads with cost: 4 + 5 + 10 = 19 |

# Undefined

You will be given a **graph** from the console. Your **task** is to find the **shortest** **path** and print it as a sequence **from S source vertex to D destination vertex** and then on the **second** line, the **weight** of that path, be careful there might be negative cycles if there are printed **"Undefined".**

**Input**

* The input comes from the console. First is an integer, the number of **nodes**, then the number of **edges**. After that each **edge** on a new line in the following format **"{source} {destination} {weight}".** Then you will read **two** integers on a **separate** **line,** the **source,** and **destination** nodes.

**Output**

* Print on a single line the **path** **found** **separated** by **spaces and** on the second line the **weight** of that path, or if there is no path, message **"Negative Cycle Detected".**

**Example**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5  8  1 2 -1  1 3 4  2 3 3  2 4 2  2 5 2  4 2 1  4 3 5  5 4 -3  1  4 | 1 2 5 4  -2 |
| 5  8  1 2 -1  1 3 4  2 3 3  2 4 2  2 5 2  4 2 -1  4 3 5  5 4 -3  1  4 | Undefined |

# Big Trip

You will be given a trip description from the console. Your **task** is to find the **longest** **trip possible and** printhowmuch **time** itwillcost. After that, print **the path itself** on a single line separated by spaces.

**Input**

* The input comes from the console. First is an integer, the number of **nodes**, then the number of **edges**. After that each **edge** on a new line in the following format **"{source} {destination} {weight}".** Then you will read **two** integers on a **separate** **line,** the **source,** and **destination** nodes.

**Output**

* Print on the first line the **weight** of that path.
* On the second line the path.

**Example**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 6  10  1 2 3  1 5 5  2 4 4  2 3 7  2 6 2  3 4 -1  3 6 2  5 3 6  5 2 2  4 6 -2  1  3 | 14  1 5 2 3 |
| 6  10  1 2 3  1 5 5  2 4 4  2 3 -7  2 6 2  3 4 -1  3 6 2  5 3 6  5 2 2  4 6 -2  1  3 | 11  1 5 3 |