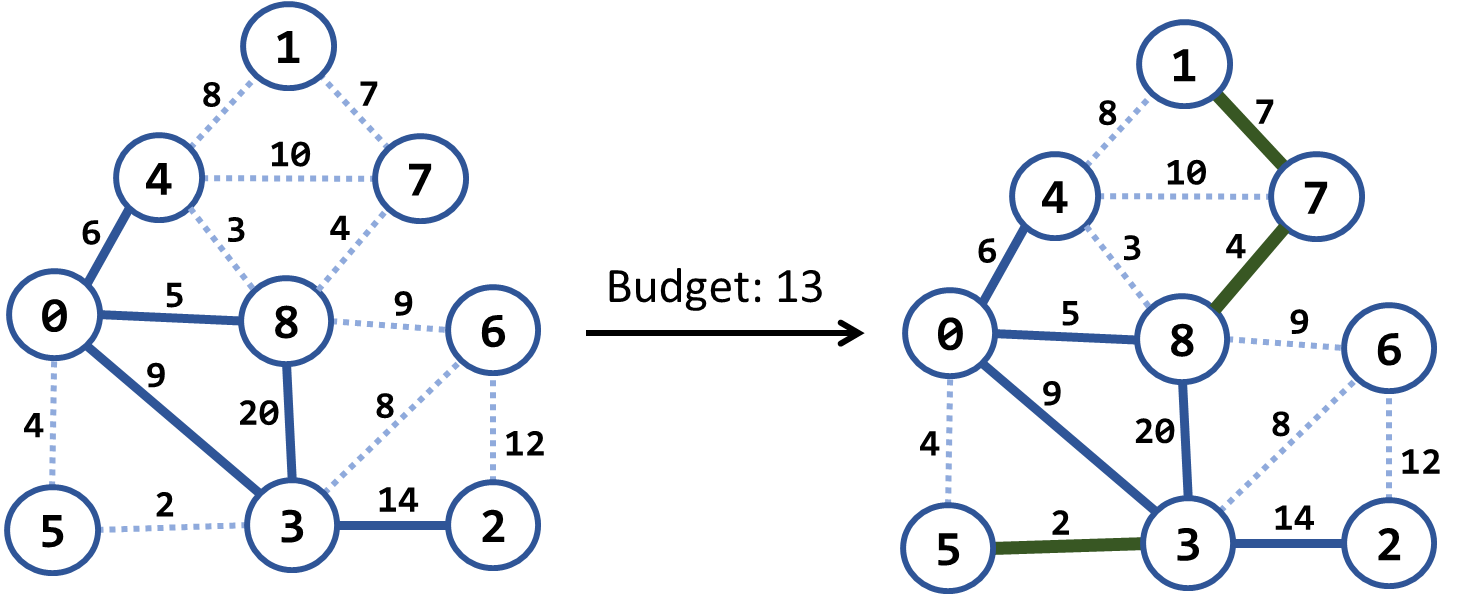
# Homework: Advanced Graph Algorithms - Part I

This document defines the **homework assignments** for the ["Algortihms" course @ Software University](https://softuni.bg/opencourses/algorithms).

## 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 these 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

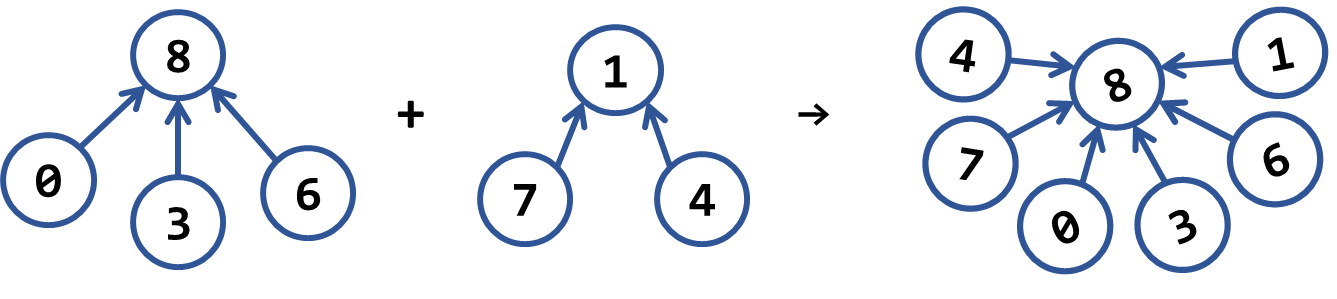
|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Picture (Before)** | **Output** | **Picture (After)** |
| Budget: 20  Nodes: 9  Edges: 15  1 4 8  4 0 6 connected  1 7 7  4 7 10  4 8 3  7 8 4  0 8 5 connected  8 6 9  8 3 20 connected  0 5 4  0 3 9 connected  6 3 8  6 2 12  5 3 2  3 2 14 connected |  | 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 connected node to 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:

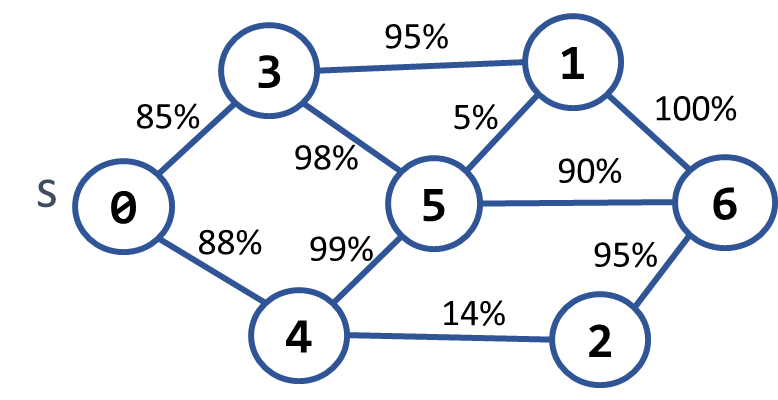
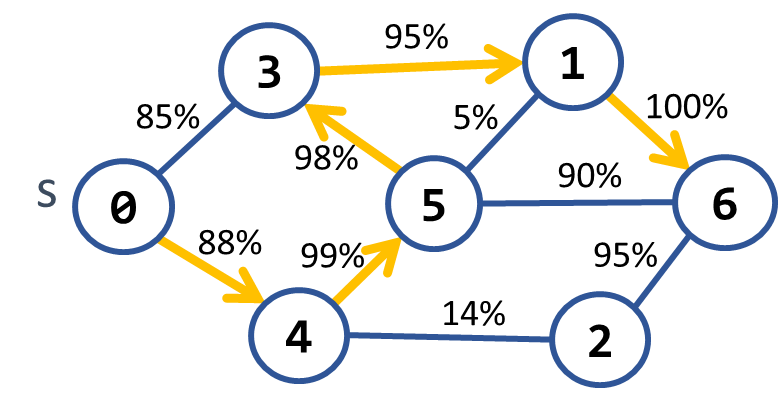


### Examples

|  |  |  |  |
| --- | --- | --- | --- |
| **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.