# CS2040 Tutorial 11 Suggested Solution

Week 13, starting 7 Nov 2022

### *Q1 Networks*

There are **V** machines (e.g. x-ray, MRI, …) in Lion University, and you have information about the cost of linking **E** possible pairs of machines up into the same network (unlike real life, assume for simplicity that the cost of such linking is only dependent on the 2 machines in the pair and not on other machines in the same network).

You would like to link all **V** machines up into ONE network with cheapest total cost of links, but you are not sure whether this can be done. For example, machine **A** may not be compatible to be linked up with machine **B**, so (**A**, **B**) will not be in **E**.

To make things worse, you have a positive integer budget **B** which you cannot exceed. Still, you want to prioritize forming as few networks as possible, and secondarily, having the cheapest total cost of links to form those networks.

Design an algorithm to **efficiently** find and output the number of networks you will end up with (**V** networks in the worst case, 1 network in the best case), as well as the total cost of building such a network. What is the time complexity of your algorithm?

### *Answer*

The cheapest total cost of links if there is unlimited budget **B** will involve linking up all **V** machines using **V**-1 links. These machines and chosen links form a minimum spanning tree of the graph of (**V** machines, **E** links). However, if the budget is lowered, then some of these links cannot be made, starting from the most costly link downwards, resulting in a minimum spanning forest.

Instead of finding the entire minimum spanning tree and then removing edges, we can run Kruskal's algorithm. Maintain the number of unions (edges picked) and sum of edge weights picked so far. After it is confirmed that an edge to be picked does not form a cycle (as two vertices are already in same set), if this edge weight would cause the sum of edge weights to exceed the budget, then we can terminate the algorithm without picking this edge.

The number of networks that will remain is **V** - number of unions and the total cost of building the network is the sum of edge weights picked. The time complexity of the algorithm is O(**E** log **E** + **V**) in the worst case.

### *Q2 Minimax Path*

Try solving Tutorial 11 Q1e using a solution that involves a minimum-spanning-tree algorithm in O(**RC**log(**RC**)) time.

### *Answer*

While a shortest path spanning tree minimizes the sum of edge weights along the path, a minimum spanning tree minimizes the max edge weight along the path[1], among all possible paths from a source to any reachable destination. Intuitively, this is because lowest possible edge weights are taken as long as they don't form any cycle.

Minimum Spanning Tree is not defined on a directed graph, but we can still find the minimax path from one vertex to another, if a path exists, using Prim's algorithm.

Run Prim's algorithm to obtain the MST and build an adjacency list from the MST. DFS from Mario on the MST (not on the original graph) to find Princess, this being the minimax path. Keep track of the maximum edge weight while recursing, and when princess if found, return the maximum edge weight on that path.

~~Run Kruskal's algorithm, and the moment Mario and Princess are in the same set after two sets are united (picked an edge that does not form a cycle), that edge has the highest edge weight along the minimax path.~~ Kruskal's algorithm will not work the directed edges selected may not form a path that takes us from Mario to Princess.

[1] Can formally prove using cycle property https://en.wikipedia.org/wiki/Minimum_spanning_tree#Cycle_property

### *Question 3 (Online Discussion) – BambooBear*

You are working in BambooBear, which has *bases* in **K** cities and delivers bamboo to **V**-**K** different cities without *bases*. Transporting bamboo across city **A** to city **B** will incur some positive integer *overhead*, and the *overhead* from city **A** to city **B** is the same as the *overhead* from **B** to **A**. There is no *overhead* incurred for delivering bamboo within the same city.

Each of the **K** bases should be *responsible* for delivering bamboo to some set of cities. Some may only be *responsible* over its own city, while others are *responsible* for two or more cities – they deliver to one or more other cities without *bases*.

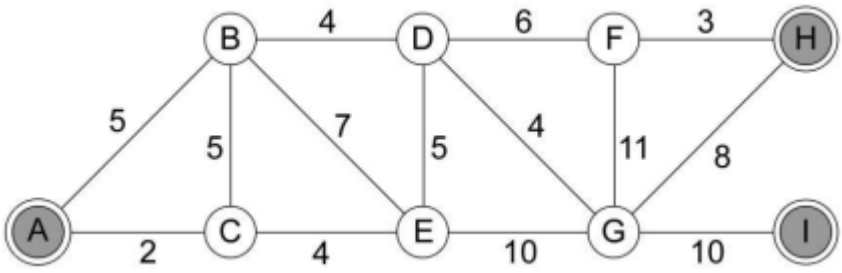
For each *base*, the *expense* for that *base* would be the sum of the *overheads* between roads that need to be travelled on among cities that the *base* is *responsible* for. The goal is to **minimize the total *expense*** of all *bases*.

You are given **V**, the ids of the **K** cities with *bases*, as well as **E** pieces of information about the *overhead* from one city to another. It is guaranteed that the 2 cities in one piece of information will be distinct among the **E** pieces of information.

Design an algorithm that **efficiently** finds and outputs, for each *base*, the *expense* of that *base*, as well as the number of cities it is *responsible* for, in O(**E** log **V**) time.

In the example below, there are:

- 9 *cities* A..H
- 3 *bases* A, H, I
- 14 pieces of information about *overheads* between cities




To minimize the total *expense*:

- only 6 out of the 14 possible roads should be utilized
- city A should be *responsible* for 6 cities, *expense* being 19
- city H should be *responsible* for 2 cities, *expense* being 3
- city I should be *responsible* for only 1 city (itself), 0 *expense*
- total *expense* is 22