CS2040: Data Structures and Algorithms

Tutorial Problems for Week 11: Graph Traversal 2 and MST

For: 31 October 2024, Tutorial 9

Solution: Secret! Shhhh... This is the solutions sheet.

Problem 1. Skyscrapers

There are n skyscrapers in a city, numbered from 1 to n. You would like to order the skyscrapers by height, from the tallest to the shortest. Unfortunately, you do not know the exact heights of the skyscrapers.

Problem 1.a. Suppose that you have m pieces of information about the skyscrapers. Each piece of information tells you that skyscraper x is taller than skyscraper y for some pair of skyscrapers x and y. **Describe the most efficient algorithm you can think of to output any one possible ordering of the buildings by height which is consistent with the m pieces of information given. What is the running time of your algorithm?**

Solution: We can model the graph by representing the n skyscrapers as vertices, and the m pieces of information as directed edges. If the piece of information tells you that skyscraper x is taller than skyscraper y, then we represent that as a directed edge from x to y. Specifically, the graph will be a Directed Acyclic Graph (DAG), as the graph cannot have cycles (the heights cannot have a circular relation). Getting a possible ordering of heights consistent with the information given is thus equivalent to finding a topological ordering of the vertices. We can run the Topological Sort algorithm which has time complexity $O(n + m)$.

Problem 1.b. Suppose that you have m pieces of information about the skyscrapers. Each piece of information tells you one of the following regarding two skyscapers x and y:

- x is taller than y
- x has the same height as y

Describe the most efficient algorithm you can think of to output any one possible ordering of the buildings by height. What is the running time of your algorithm?

Solution:

Method 1. Since there can be skyscrapers with the same height, we cannot directly apply the Topological Sort algorithm in 4a since we have two different types of relations. However, using a UFDS we can "combine" the skyscrapers with the same height as one single vertex by reading in all the "equality" relations and and doing a unionSet operation on each pair of equal skyscrapers to create our combined "vertex". Thus, each "vertex" will represent one or more skyscrapers with the same height. This allows us to again obtain a DAG, which we can now apply the Topological Sort algorithm with $O(n + m)$ time complexity.

Method 2. If x and y are the same height use a bi-directed edge to represent this relationship (i.e store y as neighbor of x and x as neighbor of y in adjacency list of the graph), thus all vertices which are of the same height must form a SCC (here we consider case of 2 vertices linked by a bi-directed edge as forming a cycle). Now if each of the SCC is viewed as a vertex, you can see that the graph is now a DAG and going through the vertices of the SCCs in topological ordering of the SCCs will result in a valid ordering of the vertices. Thus we can simply run Kosaraju's algorithm and the order of the vertices as they are being visited (in the final step of the algorithm) is a valid ordering.

Problem 2. True or False?

For each of the following, determine if the statement is True or False, justifying your answer with appropriate explanation.

a) The MST is always a connected, undirected graph.

b) The MST will always have $V - 1$ edges.

c) For a graph with unique edge weights, the edge with the largest weight in any cycle of the graph can be included in the MST.

d) For a graph with two disjoint sets of vertices A and B (vertices in A are not in B and vice versa), and another vertex x not inside both the sets, the combined MST of $A \cup x$ and $B \cup x$ is a MST of the original graph.

Solution:

a) True. The MST is a spanning tree, and by definition a spanning tree will connect all vertices in the graph.

b) True. The MST is a spanning tree, and does not have a cycle. It cannot have more than $V - 1$ edges. Since all vertices must be connected, it cannot have less than $V - 1$ edges.

c) False. The edge with the largest weight in any cycle of the graph will always be excluded for unique edge weights.

d) False. This is not true in all cases. An example is seen in the diagram below. If $A = \{0, 1\}$, $B = \{2, 3\}$ and some vertex X. The edge (1, 2) should be included in the MST, but the stated method will give a non-minimal spanning tree that does not include (1, 2).

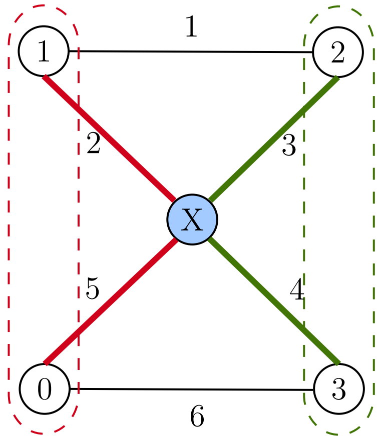


Figure 1: Wrong MST

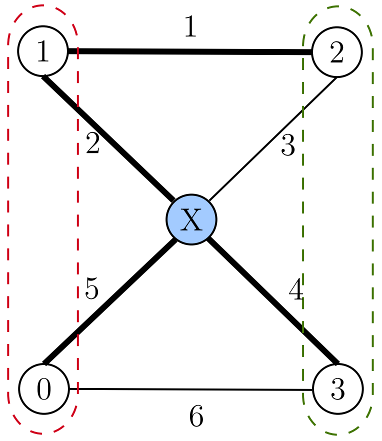


Figure 2: Correct MST

(Alternatively, we can consider the cut property. If we consider the two sets of vertices $A \cup x$ and B, and their cut-set, the smallest weighted edge in the cut-set that connects $A \cup x$ and B must be inside the MST. This may not necessarily be an edge connecting x to B if there is an edge that directly connects A to B with a lower weight.)

Problem 3. Lets add some new stuff!

Given a graph G with V vertices and E edges, and the **unique** Minimum Spanning Tree (MST) of G (i.e G has only 1 MST), give an algorithm (including the time complexity) to update the MST if

Problem 3.a. A new edge *(A, B)* is to be inserted into G.

Solution: First, we need to realize that we do not need to insert *(A,B)* into the original graph G and re-run Prim's or Kruskal's. Although obviously correct, this runs in $O((E+1)\log V) = O(E\log V)$. We can do a little bit better by only inserting the edge *(A,B)* into the MST itself and then re-run Prim's or Kruskal's on this MST + 1 edge. As an MST is a tree, it only has $E = V - 1$ edges, so this strategy runs in $O((V-1+1)\log V) = O(V\log V)$. Is this the best that we can do?

Do more observation. Before insertion of this edge *(A,B)* into the MST, there is only a path from vertex A to vertex B in the MST. After the insertion of edge *(A,B)*, we have a cycle which consists of the edges of the path from A and B and the new edge *(A,B)*. In order to update the MST, we only need to find the largest edge in this cycle and remove it. This will ensure that the resultant graph will remains the MST for the new graph $G \cup (A, B)$.

In order to do so, simply run DFS from A in the original MST to find the unique path from A to B. This allows us to find the edge X with the largest weight in the path from A to B (this can be done as part of the recursion or by first constructing the path). If the weight of *(A,B)* is larger than X, then then the MST does not change. Otherwise, we remove X and insert *(A,B)* instead into the MST to have a better MST.

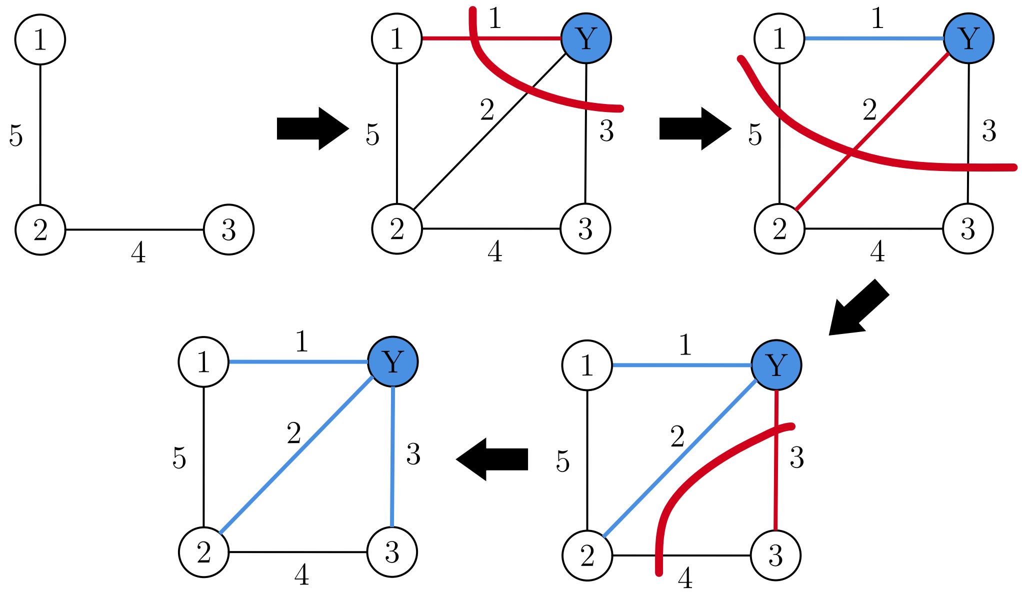
This algorithm runs in $O(V)$ since DFS is $O(V)$ on a tree and removing X and inserting *(A,B)* can also be done in $O(V)$ (e.g. we use a (sorted) EdgeList, we can find X with an $O(V)$ loop and we can insert *(A,B)* in the correct position also in $O(V)$).

Problem 3.b. A new vertex Y, along with a set of edges connecting Y to the rest of the graph, is inserted to G.

Solution: An inefficient solution would be to take the original graph G with the new vertex Y and its edges and re-run Prim's or Kruskal's which will take $O(E \log V)$ time. However, we can observe that in the resulting MST, the edges can only consist of edges from the MST of the original graph and the new edges connected to vertex Y. If any other edge can be found in the resulting MST, then the MST of the original graph would not be unique. Thus, we run Prim's or Kruskal's on the original MST together with vertex Y and its edges.

We note that when adding a new vertex Y, there will be at most V edges that connect Y to the original V vertices in the graph. Thus, there are a total of $V + 1$ vertices and $V - 1 + V = 2V - 1$ edges. The time complexity is thus $O(V \log V)$.

One might be tempted to think that we will always only take 1 edge from the edges connecting to Y and all $V - 1$ edges from the original MST to form the updated MST. However, this claim is not true. An example is shown in the figure below.



The first graph in the diagram (top left) shows the original MST. Subsequent graphs show the repeated application of the cut property after the addition of vertex Y. The red curve is the current cut of the graph we are considering, and the red edge is the edge in the cut set with the lowest cost, that is to be added into the MST. Edges in blue refer to previously chosen edges. The resulting updated MST does not have a single edge from the original MST, and all the edges are from those connected to Y.

Problem 4. Clearly Has MST Flavor

An ambitious cable company has obtained a contract to wire up the government offices in the city with high speed fiber optics to create a high speed intra-net linking up all the different governmental departments. In the beginning they were confident that the minimum cost of connecting all the offices will be within budget. However, they later found they made a miscalculation, and the minimum cost is in fact too costly. In desperation, they decided to group the government offices into K groups and link up the offices in each group, but not offices between groups to save on the cost. This effectively creates k intra-nets instead of one big intra-net.

Given V government offices, the cost of linking E pairs of government offices, and a budget b, help the company design a program which will tell them what is the smallest value of k (so as to minimize the number of intra-nets). The program also needs to output the government offices in each of the k groups and how they should be linked such that the total cost of all selected links is within budget b.

The program should model the problem as a graph. It should also run in $O(E \log V)$time. Where E and V are the number of vertices (government offices) and edges (possible links between those government offices), respectively.

Solution: Run Kruskal's which sorts E edges from cheapest to most expensive (Prim's algorithm is not suitable to solve this problem), keep track of sum of the weights of the edges picked so far. Once an edge that is going to be added will cause the sum to exceed the given budget b, we stop the algorithm. Then report k = the number of disjoint sets currently in the UFDS. We can do an $O(V)$ pass to trace the `p[i]` array of the UFDS to list down members of each groups. How the offices are linked is simply listing down all the edges that are picked. As we don't do anything other than potentially stopping Kruskal's earlier, this is at worse $O(E \log V)$, same as the standard Kruskal's algorithm.

Problem 5. The MST problem is not obvious

Please look at the CS2010 AY2011/2012 S1 Final Exam paper in the '2011-12-S1-final.pdf' file that came in the same zip file as the tutorial questions and solve the problem titled: **Vehicle Monitoring System**.

Solution: Simply run Kruskal's but sort edges in non-increasing (downwards) order to obtain the Maximum Spanning Tree (MaxST). Edges not taken by Kruskal's for the MaxST will be the edges where we have to place the cameras, and take the sum of their weights. The time complexity is simply $O(E \log V)$, the same as Kruskal's time complexity. The solution is very short but it is not easy to arrive at such solution. Your tutor will go through the "thinking process" on how to arrive at such solution during the actual tutorial.

You may want to try solving `https://uva.onlinejudge.org/external/12/1234.pdf` using this newfound knowledge.

Problem 6. Is this MST (ICPC Indonesia 2013 question)

Indonesia is the worlds largest archipelago with approximately 17,000 islands scattered for more than 5,000 km from Sabang (west most) in Sumatra island to Merauke (east most) in Papua island. Thus, providing electricity in all cities and towns across all islands is a challenging problem for the government.

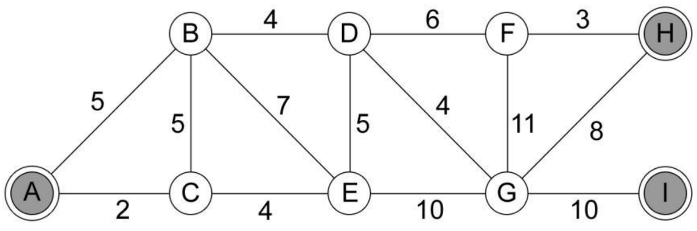


Figure 3: An example of power plants, sites and cables

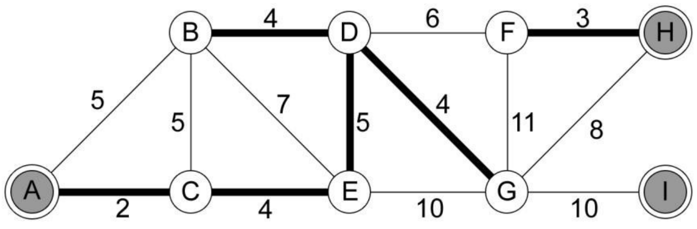


Figure 4: An example of the edges chosen to minimise total cost of cables to build

Power plants, cities, towns and all other important sites can be represented as a graph where each node represents a site and each edge which connects two different sites represents a cable transferring electricity between the two sites in both directions. You may assume that all sites are connected through some cables and for each pair of sites there is at most one cable connecting them. There is of course a cost to maintain each cable and some of them probably have a very high cost to maintain.

The government has a plan to calculate the minimum total cost to maintain only necessary cables such that all sites are connected to at least one power plant, except for the power plants (which are already connected to themselves). A site is considered connected to a power plant if and only if there is a path which consists of only maintained cables from the site to a power plant.

Consider the following example. There are 9 sites and 3 of them are power plants (A, H and I). The connectivity and the cost of each cable are shown in Figure **??**. The minimum total cost to maintain the cables in this example is 22 as shown in Figure **??**.

Given an undirected graph (which you may assume is stored in an adjacency list) which represents the connectivity between all sites, determine the minimum total cost needed to maintain cables such that all sites except power plants are connected to at least one power plant.

Solution: This is multi-source MST. Simply run Prim's algo but during initialization enqueue the priority queue with not just one vertex as source but all vertices representing power plants as source vertices. In this way, each power plant vertex will grow an MST to a subset of the vertices in the graph (those cities and towns using that power plant), and all the MSTs will cover all vertices in the graph. These MSTs will be disjoint since vertices already added to some MST will not be added to another MST (checking the taken array during Prim's).

Another way to solve it is to create a special node s which has a edge of weight 0 to every vertex representing power plants. Then run Prim's by initializing the priority queue with s. This ensures that the MST created will include the edges of weight 0 to each power plant. Removing those edges will give you the MSTs associated with each power plant.