Mahitha Valluru

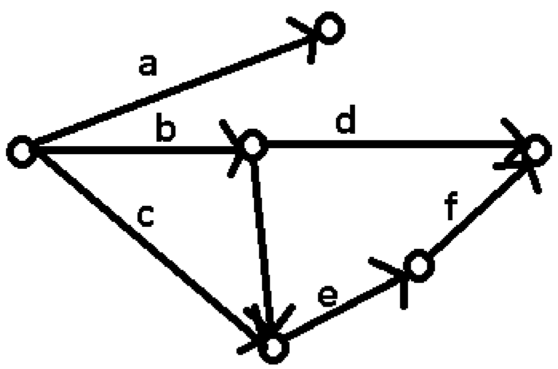
CS4800 – Algorithms and Data

Professor Or Beit Aharon

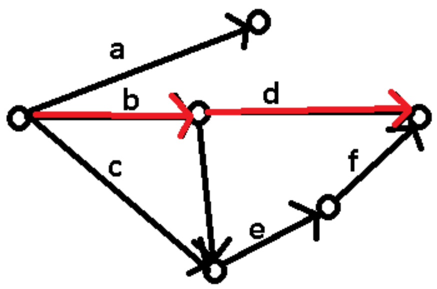
Homework 5

1.
    a. Create a new weighted graph G' that is identical to G. For all the edge weights of G', negate their values. Then, use Kruskal's or Prim's algorithm on G' to find the minimal spanning tree of G'.
    Because the edge weights of G are negated in G', this means that the edges found in the minimal spanning tree found in G' are the same edges for the maximum spanning tree of G.
    To prove Prim's algorithm, we will make a claim that for any $k \in \{1,2,\dots,n = |V|$, the tree T_k is contained in a minimal spanning tree of G.
    For k = 1, there is only one vertex which is clearly in some MST. By our induction hypothesis there exists a MST $T = (V, F)$ such that $T_k \subseteq T$. If $e_k \in F$ then $T_{k+1} \subseteq T$.
    Assume $e \notin F$. It follows that the graph with edges $F \cup \{e_k\}$ has a cycle C. Let $f \in (C - e_k)$be such that $f \in E(V_k, V - V_k)$. From the definition of e_k it follows that the weight of e_k is less than the weight of f. Set $T' = (V, (F - \{f\}) \cup \{e_k\})$. It follows that T' is a tree and the weight of T' equals the weight of T subtracted by the weight of f plus the weight of e_k. This is less than the weight of T. Therefore, we see that T' is a minimal spanning tree and $T_k \subseteq T'$.
    b. Use the algorithm described in part a above for a maximum spanning tree. This can be done by creating a duplicate graph G' and negate all of the edges' values. Then use Kruskal's or Prim's algorithm to find the minimum spanning tree of G. The same edges that were chosen for G' are the edges of the maximum spanning tree of G.
    Then, we will need to find the edges F. We can think about this intuitively. A spanning tree must not contain cycles at all. That means that a maximum spanning tree avoided certain edges so as to not create a cycle. It also happens to be that those that were avoided are of minimum weight. This fits the description of F. To answer the question, we will need to choose all the edges $F \subseteq E$ that are not included in the Maximum Spanning Tree.

2. Let's say we have a graph that looks like this:



The letters a, b, c, d, e, f are the probabilities of the message being transferred correctly along each edge. This is what the best probability path given this graph:



Now, what Dijkstra's algorithm does is it adds all the probabilities together, which looks like this: $b + d$. However, what we need is $b * d$.
How do we go about doing this?

Let's look at a certain property of logarithms:

$$(\log_{10} a + \log_{10} b) = \log_{10} ab$$

By using this property, we can use it to "convert" the value we got to the value we want. This is quite a process, so bear with me while I try to explain it.

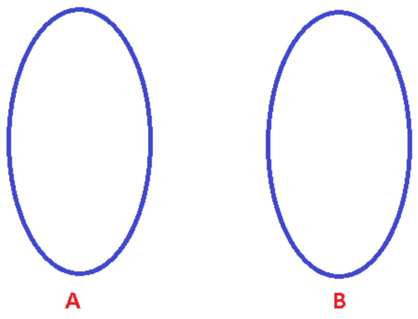
$$\log_{10}(b + d) = \log_{10}(b * d)$$

We have successfully multiplied b and d, but we have taken the log of that value and we need it so that it is a probability (between 0 and 1).
We can say that $\log_{10}(b * d) = x$.
By the property of logs, x is currently a number that is not between 0 and 1. We can see this by looking at the property of logs: $\lim_{n \to 0}(\log_{10} k) = \infty$ and $\lim_{n \to 1}(\log_{10} k) = 0$.
Thus, we can say that x is a number that is much too high.

Let's simplify this equation even more: $(b * d) = 10^x$. However, we know that this is not a decimal still.

What we can do, however, to make this a probability, is to negate the value of x so that the equation turns to: $(b * d) = 10^{-x}$, which then equals $(b * d) = \frac{1}{10^x}$. This will successfully give us the probability of the best path.

3. This is how the visualization looks like:

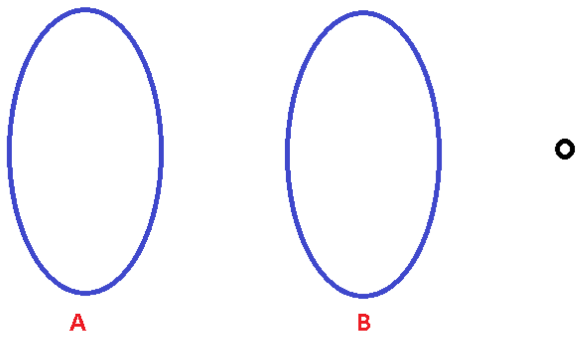


Where the sections A and B have some number of vertices in them.

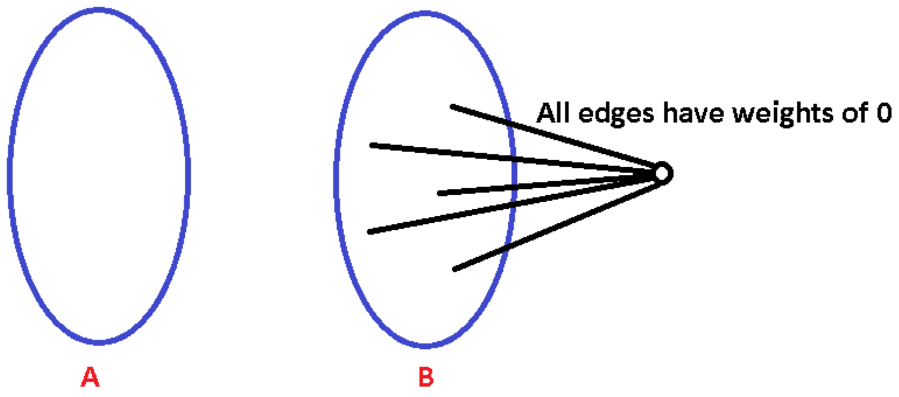
Now, what would I do if B had only one vertex in it? My solution would be to start at the vertex in group B and use Dijkstra's algorithm to find all the vertices in group A. This is just one case though.

But what if we could make this possible?

It *is* possible. I can create a vertex that is outside of groups A and B, like below:



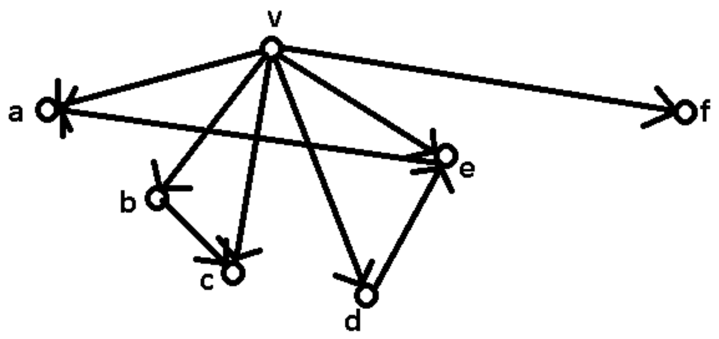
With this vertex, I will connect it to all the vertices in group B so that these created edges have weights of 0, like so:



Now, I can use Dijkstra's algorithm to find the shortest path, except I will have to reverse the path and start at the newly-formed vertex and go to group B and finally to group A.
Because the newly-created edges have weights of 0, it will not matter what path the algorithm takes, and when displaying the shortest paths, you can delete the new vertex and all its edges connecting to it.

4. Depth-First Search is an algorithm that will go through each edge twice, however there are cases where it goes through an edge more than two times. To prevent this, we can modify the DFS algorithm: If an edge has been visited twice already but it's the edges that its connected to have not, then create an edge that goes from the current vertex to another vertex so that there is a connected to the unvisited edge. That way, the unvisited edge and the created edge will be visited twice through the DFS algorithm.

5. Take for example a subset of the graph G like this:



Where v is the root.
By definition, this means that all of the vertices are connected to v.
If you start DFS at v, then it will take you to all vertices in the graph G. Thus, the set R_G is in the DFS-forest component.
But like we said, the graph we showed above is just a subset of graph G. If you start at another vertex that is not the root v shown above, that means that the vertex that DFS

has started at is another root and by definition will be connected to all of the vertices in the graph G. If the DFS run gets to vertex v, then it will to all of the vertices v is a root of. Thus, the DFS run will get to all of the components in R_G and it will be the same as the DFS-forest component.