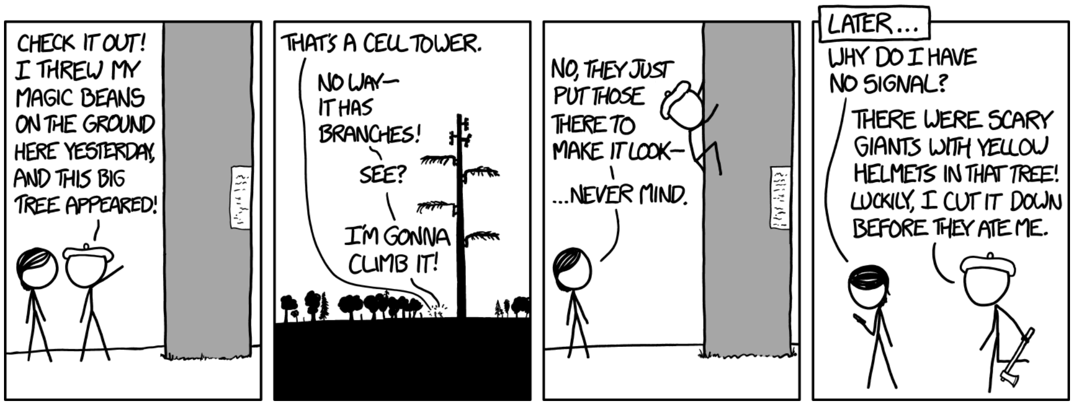
CS140 - Assignment 9

Due: Sunday, November 10th at 11:59pm


CHECK IT OUT! I THREW MY MAGIC BEANS ON THE GROUND HERE YESTERDAY, AND THIS BIG TREE APPEARED!
THAT'S A CELL TOWER.
NO WAY— IT HAS BRANCHES!
SEE?
I'M GONNA CLIMB IT!
NO, THEY JUST PUT THOSE THERE TO MAKE IT LOOK—
...NEVER MIND.
LATER...
WHY DO I HAVE NO SIGNAL?
THERE WERE SCARY GIANTS WITH YELLOW HELMETS IN THAT TREE! LUCKILY, I CUT IT DOWN BEFORE THEY ATE ME.

`https://xkcd.com/1569/`

Note: If not specified in the problem, you may assume whatever graph representation makes your algorithm more efficient (adjacency list or adjacency matrix). **State which one you are using.**

1. [**10 points**] Induction on Trees

   Use induction to prove that the number of degree-2 nodes in a non-empty binary tree is 1 less than the number of leaves. Recall that the degree of a vertex in a tree is the number of children that it has.

2. [**5 points**] Someone suggests to you the following algorithm for finding the shortest path from node s to node t in a directed graph with some negative edges: add a large constant to each edge weight so that all the weights become positive, then run Dijkstra's algorithm starting at node s, and return the shorted path found to node t.

   Is this a valid method? Argue that it works correctly or give a counterexample.

3. [**10 points**] Can we do better than Bellman-Ford?

   Assume that you are given a graph and a source vertex. You are told that the graph has the following property: for every vertex v, the weights of the edges along the shortest path from s to v are strictly increasing. Describe an efficient algorithm to solve the single source shortest path problem for such a graph. Explain why your algorithm is correct and give the runtime.

   You may assume that there are no negative weight cycles, though there may be negative edge weights.

4. **[10 points]** Suggesting interesting routes

   Let G be an undirected graph with n vertices (representing major attractions) and weighted edges (representing roads and their lengths). The *boringness* of a path between vertices u and v in G is defined to be the maximum weight among all edges on that path. The *cob (coefficient of boringness)* of a pair (u, v) is the minimum boringness over all paths from u to v.

   The weights (lengths of the roads between the vertices) are specified by a symmetric $n \times n$ adjacency matrix where the element in row i and column j specifies the weight of the edge between vertices i and j and is ∞ if no such edge exists.

   (a) Describe a modification to the Floyd-Warshall Algorithm to compute the cobs of all pairs of vertices in the graph and very briefly explain why it is correct.

   (b) What is the running time of your algorithm to compute the cobs? Explain briefly.

5. **[13 points]** Borůvka's Algorithm

   The first algorithm for computing minimum spanning trees was published by the Czech mathematician Otakar Barůvka in 1926 and was used for laying out electrical networks. It goes as follows:

```
A = {};  \\ Comment:  A is a subset of a minimum spanning tree

Consider the n  vertices in the graph as n connected components;
while A contains fewer than n-1 edges {
   for each connected component C {
      Find the least weight edge (u,v) with one vertex in C and
        one vertex not in C;
        Indicate that edge (u,v) is "chosen" but do not add it yet to A;
   }
   Add all "chosen" edges to A;
   Compute the new connected components;
}
return A \\ Comment:  This is intended to be a MST!
```

   Notice that if C_1 and C_2 are two different connected components before we begin the for loop, then inside the for loop the algorithm will choose the least weight edge coming out of component C_1 and also the least weight edge coming out of C_2. The edge chosen by C_1 might join C_1 and C_2 into a new connected component, but this new connected component will not be discovered until the for loop has ended! In other words, both C_1 and C_2 will each get an opportunity to choose the least weight edges coming out of their components.

   (a) **[2 points]** Give a counterexample that shows that Borůvka's Algorithm doesn't work!! (You might find it useful to use the fact that some edges in the graph may have the same weight.) Show your counterexample graph and explain carefully why Borůvka's Algorithm would not compute a minimum spanning tree in this case.

(b) **[5 points]** Now assume that no two edges in the graph have the same weight. Such a graph has exactly one minimum spanning tree (you may just use this fact here, although you can think about how to prove it!) Under this assumption, prove that Borůvka's Algorithm is correct.

(c) **[3 points]** Why doesn't your proof from part (b) work if some edges in the graph have the same weights?

(d) **[3 points]** How could Borůvka's Algorithm be modified slightly to work in the most general case that edge weights are not necessarily distinct? Explain briefly why this modification preserves the correctness of the algorithm.