Due date: **Friday, December 6** 11:59 PM You will need to submit this as a PDF via GradeScope.

Please review the syllabus and course reference for the expectations of assignments in this class. Remember that problem sets are not online treasure hunts. You are welcome to discuss matters with classmates, but remember the Kenny Loggins rule.

0. Please take a few minutes to fill out course evaluations. After all, I am grading you this quarter, it's only fair you get to grade me also. I read all evaluations and I take them seriously.

1. ESCAPE! I am trapped on an island with velociraptors. I am currently at a safe house $S$ and I want to get to a boat $T$ that will allow me to get safely to land. The island has many paths that I can travel along, represented as a directed graph $G = (V, E)$. Unfortunately, each edge $e$ in the graph has some probability $p(e)$ that a traveler on this edge will be eaten by a raptor. We assume these probabilities are independent of one another. If we select some path $P$ from $S$ to $T$ on which to travel, I will arrive safely with probability $\Pi_{e \in P}(1 - p(e))$. I would like to select a path $P$ which maximizes the probability of arriving safely.
   Fortunately, someone has already implemented and tested Dijkstra's Algorithm, and we'd like to re-use that code. Unfortunately, we can't access the source code, so we can't just modify the algorithm to multiply edge weights instead of adding them. Instead, let's create a new set of edge weights $w_e$ such that, if the original graph and these edge weights (instead of the given probabilities) are passed to the implemented Dijkstra's algortihm, the path it returns solves the problem here.
   Give a linear-time algorithm that, given the graph and set of probabilities, will compute a set of edge weights that does so. Explain briefly why the output of Dijkstra's Algorithm, when provided with your modified edge weights, will solve the raptor problem on the original graph. You will receive zero credit on this problem if you attempt an algorithm "from scratch" instead of modifying the input and calling Dijkstra's Algorithm.
   *Tempting as it might be, please do not feed your teacher to hungry dinosaurs.*

2. Here's a problem that occurs in automatic program analysis. For a set of variables $x_1, \ldots, x_n$, you are given some equality constraints, of the form $x_i = x_j$ and some disequality constraints, of the form $x_i \neq x_j$. Is it possible to satisfy all of them?

   **Example**: $x_1 = x_2, x_2 = x_3, x_3 = x_4, x_1 \neq x_4$ cannot be satisfied.

   Give an efficient algorithm that takes as input $m$ constraints across $n$ variables and determines if they can or cannot be satisfied. For full credit, your running time must be $\mathcal{O}(m \log n)$ ; there is a solution that is faster than $\Theta(m \log n)$, but you do not need to come up with it for full credit.

   *Hint: Read the constraints twice.*

3. *You may want to wait until after lecture on December 3 to work on this problem.*

   The Steiner Tree problem is as follows. Given an undirected graph $G = (V, E)$ with nonnegative edge costs and whose vertices are partitioned into two sets, $R$ and $S$, find a tree $T \subseteq G$ such that for every $v \in R$, $v \in T$ with total cost at most $C$. That is, the tree that contains every vertex in $R$ (and possibly some in $S$) with a total edge cost of at most $C$.

   This is similar to a Minimum Spanning Tree with some vertices not being required to be part of the output graph.

   (a) The difficulty in the Steiner tree problem is determining which set $S' \subseteq S$ should be included in $T$. Give a polynomial time algorithm to find the Steiner tree of minimum cost for the case where the optimal $S'$ is also provided.

   (b) Now we want to give a polynomial time 2-approximation to the problem of finding a minimum-cost Steiner tree from a complete graph (a simple graph with $n$ vertices and all $\binom{n}{2}$ possible edges). We are *not* given the optimal $S' \subseteq S$. You may assume that the edge costs obey the triangle inequality if you would like to do so, although that isn't strictly necessary to obtain the correct answer.

   The solution is to compute a minimum spanning tree on $R$, treating $S' = \emptyset$ (which might not be true in the optimal solution). Show that this has cost at most twice the optimal.

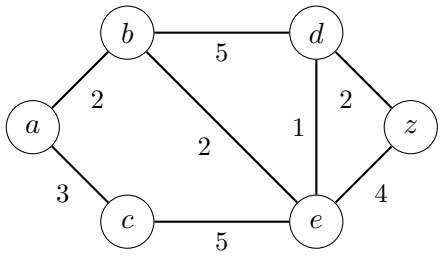
## Not Collected Questions

One good thing to do is to reinforce your understanding of the algorithms from lecture before moving to expand on them. The first two sections of these not-collected questions should help with that.

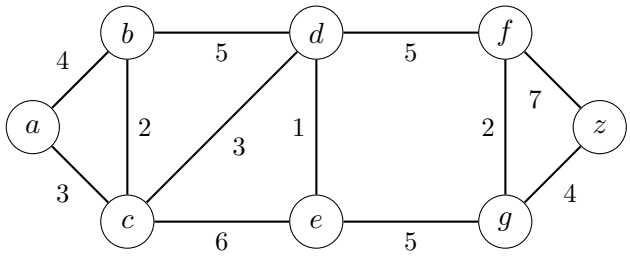
### Dijkstra's (SSSP)

Use Dijkstra's Algorithm to find a single-source shortest path tree starting at $a$ in the following graphs:

1.



2.

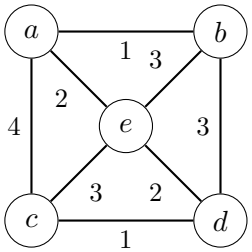



This exercise can be repeated with different starting points.

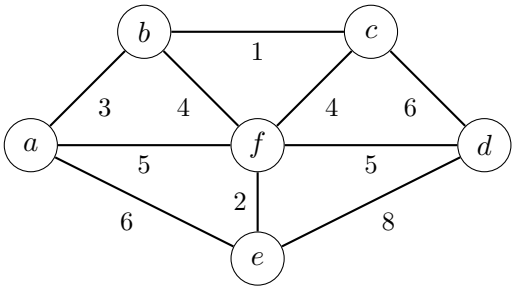
### Minimum Spanning Trees

Use your favorite MST algorithm to find MSTs for the following graphs.

1.




2.




Then, repeat the above exercise with an MST algorithm that isn't your favorite.

A more typical set of practice problems from the textbook will be added later.