## problem 1 *Prim's MST Algorithm*

Consider Prim's algorithm for finding a minimal spanning tree for a graph $G = (V, E)$; Prim's is the one that works much like Dijkstra's. Although we learned it in the graphs section, it is a greedy algorithm. For this problem, assume the graph is connected and undirected.

1. State the greedy choice made at each step of the algorithm.

   **Solution:**

   Pick the next unknown edge with the lowest cost edge from a known node.

2. Prove that Prim's algorithm has *optimal substructure*. Hint: this is a proof by contradiction, using an exchange argument (the proof of the greedy choice for interval scheduling used an exchange argument of $i_1$ and $o_1$).

   **Solution:**

   This is a proof by contradiction. Assume that a MST does *not* have optimal substructure. This MST, by definition, has minimal overall weight. We will assume this MST has branches at some point (if there are no branches, then the graph is just a line, which is a trivial case). If any of those branches was not optimal, then we could chose a different set of edges from the original graph $G$ for that branch that has a lower overall branch weight. We could then use this lower branch weight in the overall MST, which would lower the overall cost of the MST. However, this contradicts the assumption that the MST solution is minimal.

3. Prove that Prim's algorithm's greedy choice produces an optimal solution. Hint: this is a proof by induction, and uses a proof by contradiction to show the inductive step.

   **Solution:**

   This is a proof by induction. We will refer to $T_i$, which is a tree with $i$ vertices.

   **Base case:** A graph with one vertex: $T_1$. That node is part of a minimal spanning tree.

   **Inductive hypothesis:** Assume all trees $T_0, \ldots, T_k$ are optimal.

   **Inductive step:** Prove that $T_{k+1}$ is also optimal.

   Prim's algorithm will choose the edge $e = (v, u)$ of minimal weight, where $v$ is a known node and $u$ is an unknown node. Assume, for a proof by contradiction, that edge $e$ is *not* in any minimal spanning tree. We know that edge $e$ has minimal cost $c_e$ of the available edges to choose from, as that is how Prim's selects the next edge to add; note that there may be

multiple available edges to choose from of that minimal cost. Since $e$ is not in our graph, an optimal MST generation algorithm would have chosen some other edge $e' = (v', u')$. We know that $c_e \leq c_{e'}$: since Prim's chooses a least-cost edge, and it chose edge $e$, then $c'_e$ cannot be less than $c_e$. If $c_e = c_{e'}$, then Prim's could have chosen either, and this is an ordering issue as to which one was chosen first. Thus, we know that $c_e < c_{e'}$.

Since the graph is connected, and a tree, there must be some *other* path from the start node $s$ to $u$ that does not use edge $e = (v, u)$. This implies that there must be a cycle: the path from $s$ to $v$ to $u$ (with potentially other nodes in the path as well), which uses edge $e = (v, u)$. The "other" path is from $s$ to $u$, but does not use $e = (v, u)$. This path uses edge $e' = (v', u')$, and is from $s$ to $v'$ to $u'$ to $u$ (again, with potentially other nodes in the path as well). The cycle, then, includes vertices $(s, v, u, u', v', s)$, although there may be other vertices in the cycle as well.

The other MST generation algorithm chose $e'$ over $e$, even though $c_e < c_{e'}$. In the cycle $(s, v, u, u', v', s)$, edge $e'$ was removed, when edge $e$ had lower cost, which means that the choice of using $e'$ will NOT lead to a minimal tree.

## PROBLEM 2 *Matrix Maximization*

You have a matrix $M$, of size $r \times c$, of integers (positive, negative, or zero). The goal is to make the total sum of the matrix values as high as possible. However, your only operation is multiplying *all* the elements of an entire row by -1, or multiplying *all* the elements of an entire entire column by -1.

1. Describe a greedy algorithm that makes the total sum as high as possible and argue briefly why it works.

   **Solution:**

   Greedily flip any row or column that has a negative sum until there are no more rows. Works because total sum is monotonically increasing so we must hit the maximum possible at some point.

2. What is the runtime of your algorithm? Why?

   **Solution:**

   **FIXME**

   $\Theta(r * c)$. You have to consider each of the $r$ rows. For each row, you have to sum up the values in that row, and there are $c$ such values. This takes $r * c$ time. Similarly for the columns, which will take $c * r$ time.

## PROBLEM 3 *Weighted Interval Scheduling*

In class we saw the interval scheduling problem, where the goal is to select as many intervals as possible. Consider the *weighted* version: each interval has a weight (or reward, profit, etc.). Given a series of intervals with weights, the goal in this version of the problem is to find the maximum profit (sum of the selected intervals' weights) from the intervals provided. As with the unweighted version, you cannot have two intervals that overlap in the solution.

Show that the greedy algorithm for the unweighted version presented in class – choosing the next interval based on the earliest finish time – is not optimal by providing a counter-example.

**Solution:**

Consider three intervals sorted by finish time: $i_1, i_2, i_3$. Assume that $i_1$ and $i_3$ do not overlap, but $i_2$ overlaps with the other two. The algorithm presented in class would choose $i_1$ and $i_3$, as

$i_1$ is chosen first (it has the earliest finish time), since it overlaps with $i_2$, then $i_2$ is removed from consideration, and then $i_3$ – which does not overlap with $i_1$ – is chosen.

If the weight of $i_2$ is greater than the sum of the weights of $i_1$ and $i_3$ (formally: $w_{i_2} > w_{i_1} + w_{i_3}$), then the algorithm presented in class will not chose the correct solution for the weighted version of this problem.