- The instructions are the same as in Homework-0 and 1.

There are 5 questions for a total of 100 points.

---

1. Counterexamples are effective in ruling out certain algorithmic ideas. In this problem, we will see a few such cases.

   (a) (5 points) Recall the following event scheduling problem discussed in class (lecture 15):

   > You have a conference to plan with $n$ events and you have an unlimited supply of rooms. Design an algorithm to assign events to rooms in such a way as to minimize the number of rooms.

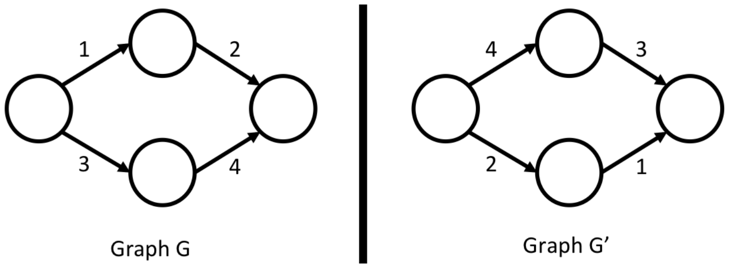
   The following algorithm was suggested during class discussion.

```
ReduceToSingleRoom($E_1, ..., E_n$)
  - $U \leftarrow \{E_1, ..., E_n\}$; $i \leftarrow 1$
  - While $U$ is not empty:
      - Use Earliest Finish Time greedy algorithm on events in set $U$
        to schedule a subset $T \subseteq U$ of events in room $i$
      - $i \leftarrow i + 1$; $U \leftarrow U \setminus T$
```

   Show that the above algorithm does not always return an optimal solution.

   (b) (5 points) A longest simple path from a node $s$ to $t$ in a weighted, directed graph is a simple path from $s$ to $t$ such that the sum of weights of edges in the path is maximised. Here is an idea for finding a longest path from a given node $s$ to $t$ in any weighted, directed graph $G = (V, E)$:

   > Let the weight of the edge $e \in E$ be denoted by $w(e)$ and let $w_{max}$ be the weight of the maximum weight edge in $G$. Let $G'$ be a graph that has the same vertices and edges as $G$ but for every edge $e \in E$, the weight of the edge is $(w_{max} + 1 - w(e))$. (*For example, consider the graph $G$ below and its corresponding graph $G'$.*)



   > Run Dijkstra's algorithm on $G'$ with starting vertex $s$ and return the shortest path from $s$ to $t$.

   Show that the above algorithm does not necessarily output the longest simple path.

(c) (5 points) Recall that a *Spanning Tree* of a given connected, weighted, undirected graph $G = (V, E)$ is a graph $G' = (V, E')$ with $E' \subseteq E$ such that $G'$ is a tree. The cost of a spanning tree is defined to be the sum of the weight of its edges. A *Minimum Spanning Tree (MST)* of a given connected, weighted, undirected graph is a spanning tree with minimum cost. The following idea was suggested for finding an MST for a given graph in the class.

> Dijkstra's algorithm gives a shortest path tree rooted at a starting node $s$. Note that a shortest path tree is also a spanning tree. So, simply use Dijkstra's algorithm and return the shortest path tree.

Show that the above algorithm does not necessarily output an MST. In other words, a shortest path tree may not necessarily be an MST. (*For this question, you may consider only graphs with positive edge weights.*)

2. (20 points) You are given a directed graph $G = (V, E)$ where nodes represent cities and directed edges represent road connections. There is a positive weight $w(e) > 0$ associated with every directed edge $e \in E$ that denotes the cost of travelling along that road (this could be toll, gas cost etc.). There is also a positive weight $c(v) > 0$ associated with every node $v \in V$ that denotes the cost of visiting the city $v$ (this could be food, lodging etc.). You are planning a trip from a city $s \in V$ to a city $t \in V$ and you want to find a route that will cost you the least amount of money. Note that this is a path that starts with $s$ and ends at $t$ and the sum of the weight of edges plus the sum of the weight of *intermediate nodes* (i.e., nodes except $s$ and $t$) in the path is minimised.

   Design an algorithm for this problem. Give running time analysis and proof of correctness.

3. (*Example for modify-the-solution*) You are staying in another city for $n$ days. Unfortunately, not every hotel is available every day. You are given a list of hotels, $h_1, ..h_k$ and a $k \times n$ array of booleans $Avail[i, j]$ that tells you whether hotel $i$ is available on the day $j$ of your trip. You wish to pick which hotel to stay in each day of your trip, to minimize the number of times you have to switch hotels. You can assume that at least one hotel is available every day.

   *For example, say $n = 7$ and there are three hotels, $A, B, C$. Hotel A is available days 1-3 and days 6-7, Hotel B is available days 1-5, and Hotel C is available days $3 - 7$. Then one solution is to stay at Hotel B days 1-5, then switch to Hotel C for days 6-7. (We could equally well switch to Hotel A; either way the optimal is 1 switch).*

   Here is a greedy strategy that finds the schedule with the fewest switches.

   > **Greedy Strategy:** Stay at the hotel available on the first day with the most consecutive days of availability starting at that day. Stay there until it becomes unavailable, and then repeat with the remaining days.

   We will show that the above greedy strategy gives an optimal solution using modify-the-solution. For this, we will first need to prove the following exchange lemma.

   Exchange lemma: *Suppose that the greedy algorithm stays at hotel $g$ on the first day and stays there until day $I$. Let $OStays$ be any scheduling of available hotel rooms. Then there exists a schedule of available hotel rooms assignment $OStays'$ that stays at hotel $g$ until day $I$, and switches hotel rooms no more than the number of switches for $OStays$.*

   proof: Let $OStays$ be as above. We ask you to complete the proof of the exchange lemma below.

   (a) (1 point) Define $OStays'$.

   (b) (2 points) $OStays'$ is a valid solution because... (Justify why in $OStays'$, we never try to stay at an unavailable hotel.)

(c) (2 points) The number of switches in $OStays'$ is less than or equal to that in $OStays$ because... (justify).

We will now use the above exchange lemma to argue that the greedy algorithm outputs an optimal solution for any input instance. We will show this using mathematical induction on the input size (i.e., number of days). The base case for the argument is trivial since, for $n = 1$, there is no switch in the greedy algorithm which is optimal.

(a) (5 points) Show the inductive step of the argument.

Having proved the correctness, we now need to give an efficient implementation of the greedy strategy and give time analysis.

(a) (8 points) Give pseudocode of an efficient algorithm implementing the above strategy and give a time analysis for your algorithm.

4. (*Example of greedy-stays-ahead*) You are going cross country, and want to visit different national parks on your way. The parks on your trip are, in order, $Park_1, Park_2, ..., Park_k$. You have $1 \leq n \leq k$ days for your trip, and you would like to visit a different park each day. You don't want to do unnecessary driving so you will only visit parks in order from the smallest number to the largest.

   Your input is a $k \times n$ array of booleans $Avail[i, j]$ that tells you whether park $i$ is available on the day $j$ of your trip. You wish to pick which park to stay in each day of your trip. Your strategy should either find a way to visit $n$ different parks, or tell you that this is impossible.

   *For example, say $n = 3$ and there are four parks, $1, 2, 3, 4$ (i.e., $k = 4$). Park 1 is available all three days. Park 2 is available on day 3 only. Park 3 is available on days 2 and 3. Park 4 is available all three days. Then we could visit Park 1 the first day, skip Park 2, visit Park 3 the second day, and visit Park 4 the last day.*

   Here is a greedy strategy that finds a schedule visiting $n$ parks if one exists:

   > **Greedy Strategy:** On the first day, visit the first park available on the first day. Each other day, visit the first park available on that day which is after your current park. If there is no park available to pick up on a day, say that the trip is impossible.

   We will show the optimality of the greedy strategy using the greedy-stays-ahead method. Suppose that the greedy algorithm stays at parks number $g_1, ..g_n$, in order, and $OStays$ is another solution staying at parks number $p_1, ..p_n$. We claim that at all days $i$, $g_i \leq p_i$. We prove this by induction on $i$.

   (a) (10 points) Prove the above claim using induction on $i$. Show base case and induction step.

   (b) (5 points) Use the claim to argue that the greedy algorithm outputs a valid travel plan in case such a plan exists and outputs "impossible" otherwise.

   (c) (7 points) Give an efficient algorithm implementing the above strategy and give a time analysis for your algorithm.

5. (25 points) You are a party organiser and you are asked to organise a party for a company. The CEO of the company wants this party to be fun for everyone who is invited and has asked you to invite the maximum number of employees of the company with the constraint that for no two employees who are invited, one is the immediate boss of the other. The company has a typical hierarchical tree structure, where every employee except the CEO has exactly one immediate boss.

   Design an algorithm for this problem. You are given as input an integer array $B[1...n]$, where $B[i]$ is the immediate boss of the $i^{th}$ employee of the company. The CEO is employee number 1 and $B[1] = 0$. The output of your algorithm is a subset $S \subseteq \{1, ..., n\}$ of invited employees. Give running time analysis and proof of correctness.