- The instructions are the same as in Homework-0, 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):

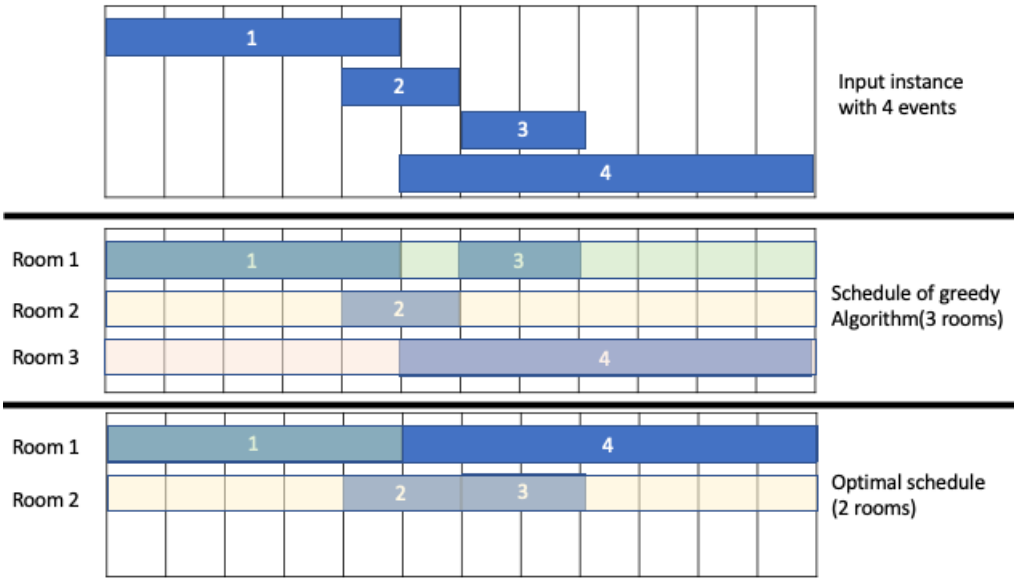
   > You have a conference to plan with n events and you have an unlimited supply of rooms. Design an algorithm to assign events to rooms in such a way as to minimize the number of rooms.

   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
     - $i \leftarrow i + 1$; $U \leftarrow U \setminus T$

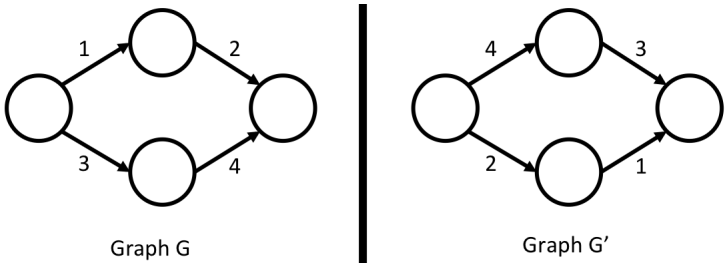
   Show that the above algorithm does not always return an optimal solution.

   Solution: Here is a counterexample.



(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 $G = (V, E)$:

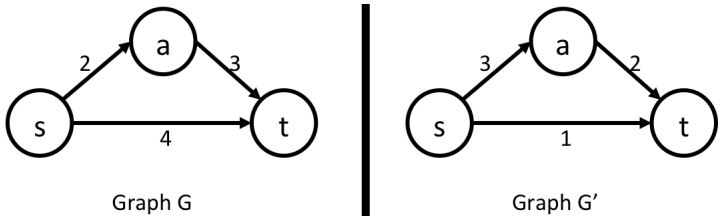
> Let the weight of the edge $e \in E$ be denoted by $w(e)$ and let w_{max} be the weight of the maximum weight edge in G. Let G' be a graph that has the same vertices and edges as G but for every edge $e \in E$, the weight of the edge is $(w_{max} + 1 - w(e))$. (*For example, consider the graph G below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

Solution: Consider the counterexample given below.



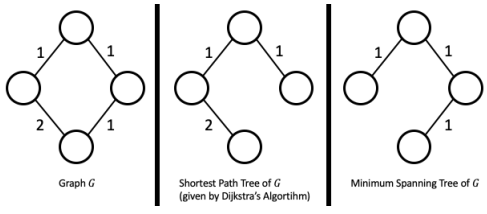
The shortest path from s to t in G' is $s \to t$. However, this is not a longest simple path from s to t in G.

(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 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 a MST. In other words, a shortest path tree may not necessarily be a MST. (*For this question, you may consider only graphs with positive edge weights.*)

Solution: Here is a counterexample.



2. (*Example for "greedy stays ahead"*) Suppose you are placing sensors on a one-dimensional road. You have identified n possible locations for sensors, at distances $d_1 \leq d_2 \leq ... \leq d_n$ from the start of the road, with $0 \leq d_1 \leq M$ and $d_{i+1} - d_i \leq M$. You must place a sensor within M of the start of the road, and place each sensor after that within M of the previous one. The last sensor must be within M of d_n. Given that, you want to minimize the number of sensors used. The following greedy algorithm, which places each sensor as far as possible from the previous one, will return a list $d_{i_1} \leq d_{i_2} \leq ... \leq d_{i_k}$ of locations where sensors can be placed.

```
GreedySensorMin(d_1...d_n, M)
  - Initialize an empty list
  - Initialize I = 1, PreviousSensor = 0.
  - While (I < n):
      - While (I < n and d_{I+1} ≤ PreviousSensor + M) I++
      - If (I < n) Append d_I to list; PreviousSensor = d_I; I++.
  - if list is empty, append d_n to list
  - return(list)
```

In using the "greedy stays ahead" proof technique to show that this is optimal, we would compare the greedy solution $d_{g_1}, ..d_{g_k}$ to another solution, $d_{j_1}, ..., d_{j_{k'}}$. We will show that the greedy solution "stays ahead" of the other solution at each step in the following sense:
Claim: For all $t \geq 1, g_t \geq j_t$.

(a) (5 points) Prove the above claim using induction on the step t. Show base case and induction step.

Solution: We prove using induction.
Base case: Since the greedy algorithm picks that last location along the road that is within distance M from 0, we have that $g_1 \geq d_1$.
Inductive step: We assume that $g_k \geq d_k$ for all $1 \leq k < t$ for an arbitrary $t > 1$. We will show that $g_t \geq d_t$. Assume for the sake of contradiction that $g_t < d_t$. Then we have $d_t - d_{t-1} \leq M$ and since $g_{t-1} \geq d_{t-1}$, this implies that $d_t - g_{t-1} \leq M$. In such a case, the greedy algorithm will not pick g_t but pick d_t since it picks the last feasible location for the sensor.

(b) (3 points) Use the claim to argue that $k' \geq k$. (Note that this completes the proof of optimality of the greedy algorithm since it shows that greedy algorithm places at most as many sensors as any other solution.)

Solution: Assume for the sake of contradiction that $k' < k$. From the previous part, we know that $g_{k'} \geq d_{k'}$. Since $d_{j_1}, d_{j_2}, ..., d_{j_{k'}}$ is a valid solution, we have that $d_n - d_{j_{k'}} \leq M$. This further implies that $d_n - d_{g_{k'}} \leq M$. However, in this case, the greedy algorithm will terminate. It will not place any more sensors. This is a contradiction. So, we get that $k' \geq k$.

(c) (2 points) In big-O notation, how much time does the algorithm as written take? Write a brief explanation.

Solution: The algorithm runs in $O(n)$ time. The algorithm makes a pass over the list of sensors. So, the time is proportional to the size of the list.

3. (*Example for "modify the solution"*) You have n cell phone customers who live along a straight highway, at distances $D[1] < D[2] < ... < D[n]$ from the starting point. You need to have a cell tower within Δ distance of each customer, and want to minimize the number of cell towers.

   (*For example, consider* $\Delta = 3$ *and there are* 3 *customers (i.e.,* $n = 3$*) with* $D[1] = 3, D[2] = 7, D[3] = 10$. In this case, you can set up two cell towers, one at 6 and one at 10.)

   Here is a greedy strategy for this problem.

   > **Greedy strategy**: Set up a tower at distance d which is at the farthest edge of the connectivity range for the customer who is closest to the starting point. That is, $d = \Delta + D[1]$. Note that all customers who are within Δ distance of this tower at d, are covered by this tower. Then recursively set up towers for the remaining customers (who are not covered by the first tower).

   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: *Let G denote the greedy solution and let g_1 be the location of the first cell phone tower set up by the greedy algorithm. Let OS denote any solution that does not have cell phone tower at g_1. Then there exists a solution OS' that has a cell phone tower set up at g_1 and OS' has the same number of towers as OS.*

   Proof. Let $OS = \{o_1, ..., o_k\}$. That is, the locations of the cell phone towers as per solution OS is $o_1 < o_2 < ... < o_k$. We ask you to complete the proof of the exchange lemma below.

   (a) (2 points) Define OS'.

   > **Solution:** $OS' = (OS - \{o_1\}) \cup \{g_1\}$.

   (b) (4 points) OS' is a valid solution because ... (*justify why OS' provides coverage to all customers.*)

   > **Solution:** OS' is a valid solution because since $o_1 \leq g_1$, all the customers who are covered by o_1 are also covered by g_1.

   (c) (3 points) The number of cell phone towers in OS' is at most the number of cell phone towers in OS because... (*justify*)

   > **Solution:** Since OS' is obtained from OS by throwing out o_1 and including g_1.

   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 customers). The base case for the argument is trivial since for $n = 1$, the greedy algorithm opens a single tower which is indeed optimal.

   (a) (6 points) Show the inductive step of the argument.

   > **Solution:** We will assume that the greedy solution is optimal for any input of size k for $1 \leq k < n$. We will show that the greedy solution is optimal for any input of with n customers. We will do this using the following sequence of steps.
   >
   > - Let I be any problem instance with n customers.
   > - Let g_1 denote the first tower set up in the greedy solution $GS(I)$.
   > - Let C be the set of customers in I that are covered by the tower at g_1. Then let $I' = I \setminus C$.
   > - Then, we can write $GS(I) = \{g_1\} \cup GS(I')$.

- Let OS be any solution for instance I.
- Then, by the exchange lemma, there is another solution OS' that sets up a tower at g_1 and $|OS'| \leq |OS|$.
- This means that we can write $OS' = \{g_1\} \cup SomeSolution(I')$.
- Now the following sequence of inequalities completes the inductive step:

$$\begin{aligned} |GS(I)| &= 1 + |GS(I')| \\ &\leq 1 + |SomeSolution(I')| \quad \text{(Using induction hypothesis)} \\ &= |OS'| \\ &\leq |OS| \quad \text{(using the exchange lemma).} \end{aligned}$$

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

(a) (10 points) Give an efficient algorithm implementing the above strategy, and give a time analysis for your algorithm.

Solution: Here is the pseudocode for the algorithm.

```
PlaceTowers(D[1...n], Δ)
  - T = D[1] + Δ; I = 1; Append T to list
  - while (I ≤ n)
      - while (D[I] ≤ T + Δ)I++
      - if (I ≤ n) T = D[I] + Δ; Append T to list
  - return(list)
```

The algorithm makes a pass over the list D. So, the running time is $O(n)$.

4. (25 points) You are a conference organiser and you are asked to organise a conference for a company. The capacity of the conference room is limited and hence you would want to minimise the number of people invited to the conference. To make the conference useful for the entire company, you need to make sure that if an employee is not invited, then every employee who is an immediate subordinate of this employee gets the invitation (*if an employee is invited, then you may or may not invite a subordinate*). 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 $B[1] = 1$. The output of your algorithm is a subset $S \subseteq \{1, ..., n\}$ of invited employees. Give running time analysis and proof of correctness.

Solution: Given an undirected graph $G = (V, E)$, a vertex cover of the graph is a subset $S \subseteq V$ of vertices such that for every edge $(u, v) \in E$ at least one of u or v is in S. The given problem is the problem of finding the smallest vertex cover of a given tree. This problem is closely related to the maximum independent set problem defined next. Given an undirected graph $G = (V, E)$, an independent set of the graph is a subset $I \subseteq V$ of vertices such that for no two nodes $u, v \in I$ we have $(u, v) \in E$. The maximum independent set problem is problem of finding an independent set of largest cardinality. The connection between the maximum independent set problem and minimum vertex cover is given in the next lemma.

Lemma: For any graph $G = (V, E)$, S is a vertex cover of G if and only if $V \setminus S$ is an independent set of G.

Proof. Let S be a vertex cover of G. Assume for the sake of contradiction that $V \setminus S$ is not an independent set. This means that there are two vertices u, v in $V \setminus S$ such that $(u, v) \in E$. However, this means that S cannot be a vertex cover because for edge (u, v) neither u nor v is in S.

In the reverse direction, let I be an independent set of G. Assume for the sake of contradiction that $V \setminus I$ is not a vertex cover of G. In this case, there is an edge (u, v) such that both u and v is in I but then this contradicts with the fact that I is an independent set. □

The above lemma suggests that we can as well design an algorithm for finding the maximum independent set I and then return $V \setminus I$. We will design a greedy algorithm for finding the largest independent set of any given forest (a set of trees). Here is the greedy strategy that we will use for this problem:

> **Greedy algorithm**: Pick a node v of degree ≤ 1 from the forest, remove this node and its neighbors, and recurse on the remaining forest (obtained by removing node v and its neighbor in case one exists).

Let us prove that the above algorithm indeed outputs a largest independent set of any given forest. We show this using an exchange argument. Let us first obtain the exchange lemma.

Lemma: *For any input forest $G = (V, E)$, let $g \in V$ be the first vertex picked by the greedy algorithm. Let I be any independent set of G that does not contain g. Then there is another independent set I' that contains g and $|I'| \geq |I|$.*

Proof. Note that g is a vertex in G with degree 0 or 1. If g is a degree 0 vertex, then $I' = I \cup \{g\}$ is also an independent set of larger size. If g is of degree 1, then there are two cases to consider:

- The neighbor u of g is in I: In this case, $I' = I - \{u\} \cup \{g\}$ is also an independent set of same size as I.

- The neighbor u of g is not in I: In this case, $I' = I \cup \{g\}$ is also an independent set of larger size.

This completes the proof of the exchange lemma. □

We now prove using induction that the greedy algorithm outputs a largest independent set for any given forest. We consider the following proposition:

$P(n)$: The greedy algorithm produces a largest independent set for any forest with n nodes.

We will show that $P(n)$ holds for all n using induction.

Base case: $P(1)$ holds since for any graph with 1 node, the greedy algorithm outputs one node which is optimal.

Induction Hypothesis: Assume that $P(1), P(2), ..., P(n-1)$ hold for an arbitrary $n > 1$.

Induction step: Here, we will show that $P(n)$ holds. For any forest $G = (V, E)$ with $|V| = n$,

- Let g be the first vertex picked by the greedy algorithm.
- Let G' be the forest obtained by deleting vertex g and its neighbor (if there exists one). Note that G' is a forest that has less than n nodes.
- Let $GS(.)$ denote the greedy solution for an input graph.
- Then, $GS(G) = \{g\} \cup GS(G')$.
- Let I be any independent set of forest G. Then from the exchange lemma, there exists an independent set I' such that $I' = \{g\} \cup IS(G')$ where $IS(G')$ denotes some solution for forest G' and $|I'| \geq |I|$.
- Now we can conclude the argument using the following sequence of inequalities:

$$\begin{aligned} |GS(G)| &= 1 + |GS(G')| \\ &\geq 1 + |IS(G')| \quad \text{(using induction hypothesis)} \\ &\geq |I'| \quad \text{(using the definition of } I') \\ &\geq |I| \quad \text{(using the exchange lemma)} \end{aligned}$$

This implies that the greedy solution is as good as any other solution which means that the greedy algorithm outputs a largest independent set for G. This completes the inductive argument and proof of optimality of our greedy algorithm.

Let us now focus on the implementation of the algorithm. Note that what we are given is not the tree in adjacency list representation but the parent of every node in the array B. However, we can obtain the adjacency list using the following pseudocode:

```
ConvertToTree(B, n)
  - Initialize an adjacency list L with n nodes and no edges
  - For i = 2 to n:
      - Append B[i] to the linked list for node i in L
      - Append i to the linked list for node B[i] in L
  - return(L)
```

The running time of the above procedure is clearly $O(n)$. We can use the above as a subroutine in our algorithm.

`GreedyAlgorithm`(B, n)
- $L \leftarrow$ `ConvertToTree`(B, n)
- Go over adjacency list L to find the degree $D[1...n]$ of all nodes
- Initialise a list S of all degree 0 and degree 1 nodes
- While there is at least one node that is not marked deleted:
  - Let v be any node in S that is not marked deleted
  - Remove v from S; $I \leftarrow I \cup \{v\}$
  - Mark v as deleted
  - If $(D[v] = 1)$
    - Let u be the only node in the linked list of v that has not been marked deleted
    - Mark u as deleted
    - For every neighbor w of u that has not been marked deleted:
      - $D[w] \leftarrow D[w] - 1$
      - If $(D[w] \leq 1)$ append w to list S
- return$(V \setminus I)$

Running time: The first step takes $O(n)$ time. Going over the adjacency list and initialising the list takes $O(n)$ time. Within the while loop the time spent per node before it is marked deleted is some constant times the degree of the node. So, the time in the while loop is $O(|V| + |E|)$ where V and E are the vertex and edge sets respectively. Since the given graph is a tree, we have $|E| = O(|V|) = O(n)$. So, the overall running time is $O(n)$.

5. (25 points) A town has n residents labelled $1, ..., n$. In the midst of a virus outbreak, the town authorities realise that hand sanitiser has become an essential commodity. They know that every resident in this town requires at least T integer units of hand sanitiser. However, at least $\lceil \frac{n}{2} \rceil$ residents do not have enough sanitiser. On the other hand, there may be residents who may have at least T units. With very few new supplies coming in for the next few weeks they want to implement some kind of sharing strategy. At the same time, they do not want too many people to get in close proximity to implement sharing. So, they come up with the following idea:

   Try to *pair up* residents (based on the amount of sanitiser they possess) such that:

   1. A resident is either unpaired or paired with exactly one other resident.
   2. Residents in a pair together should possess at least $2T$ units of sanitiser.
   3. The number of unpaired residents with less than T units of sanitzer is minimised.

   Once such a pairing is obtained, the unpaired residents with less than T units of sanitiser can be managed separately. The town authorities have conducted a survey and they know the amount of sanitiser every resident possesses. You are asked to design an algorithm for this problem. You are given as input integer n, integer T, and integer array $P[1...n]$ where $P[i]$ is the number of units of sanitiser that resident i possesses. You may assume that $0 \leq P[1] \leq P[2] \leq ... \leq P[n]$. Your algorithm should output a pairing as a list of tuples $(i_1, j_1), (i_2, j_2), ..., (i_k, j_k)$ of maximum size such that (i) For all $t = 1, ..., k$, $P[i_t] + P[j_t] \geq 2T$ and (ii) $i_1, ..., i_k, j_1, ..., j_k$ are distinct. Give proof of correctness of your algorithm and discuss running time.

Solution: Here is the greedy strategy that we will use for this problem:

> Greedy strategy: If $P[1] < T$, then try to pair up with $P[n]$. If $P[1] + P[n] \geq 2T$, then pair up residents 1 and n and recurse on residents $2, ..., n-1$, otherwise, leave resident 1 unpaired and recurse on $2, ..., n$.

We will prove the optimality of the greedy strategy using an exchange argument. For this, we first show the following exchange lemma:

Exchange lemma: Let g_1 be the first pair of residents picked by the greedy strategy. Note that $g_1 = (d, n)$ for some $d \geq 1$. Let OS denote any other solution. Then there is another solution OS' that includes the pair g_1 and the number of unpaired residents that have less than T units of the sanitizer in OS' (denoted by $|OS'|$) is at most those in OS.

Proof. Let s be the largest index resident who is paired as per OS and let it be paired with r. So, $(r, s) \in OS$. Given this, note that $r \geq d$ since otherwise d is not the first resident who can be paired with n to have at least $2T$ units. Consider $OS' = (OS - \{(r, s)\}) \cup \{g_1\}$. This is also a valid solution. Furthermore, the number of unpaired residents with less than T units does not decrease on doing this exchange (since r is replaced with d). □

We will show that greedy algorithm returns an optimal solution using induction on the number of residents n. The base case is trivial since for $n = 1$ and $n = 2$, the greedy algorithm indeed returns an optimal solution. We give the inductive argument in the following sequence of steps:

- Let I be any problem instance with n residents.
- Let $g_1 = (d, n)$ denote the first pair picked in the greedy solution $GS(I)$.
- Let $I' = I \setminus (\{1, ..., d-1\} \cup \{n\})$.
- Then, we can write $GS(I) = \{g_1\} \cup GS(I')$.

- Let OS be any solution for instance I.
- Then, by the exchange lemma, there is another solution OS' that includes the pair g_1 and $|OS'| \leq |OS|$.
- This means that we can write $OS' = \{g_1\} \cup SomeSolution(I')$.
- Now the following sequence of inequalities completes the inductive step:

$$\begin{aligned} |GS(I)| &= (d-1) + |GS(I')| \\ &\leq (d-1) + |SomeSolution(I')| \qquad \text{(Using induction hypothesis)} \\ &= |OS'| \\ &\leq |OS| \qquad \text{(using the exchange lemma).} \end{aligned}$$

Here is the pseudocode for the implementation of the greedy strategy.

```
GreedyPair(P[1...n], T)
  - i = 1; j = n; Initialize an empty list
  - while (i < j and P[i] < T)
      - while (P[i] + P[j] < 2T) i++
            - if (i < j and P[i] < T)
                  - Append (i, j) to the list
                  - i++; j- -
  - return list
```

The algorithm maintains pointers i and j with starting state $i = 1, j = n$ and in every step the difference between these pointers decreases by 1. The algorithm terminates when these points cross each other. So, the running time is proportional to the size of the instance which is $O(n)$.