- This is for preparatory material on Shortest Paths and Greedy algorithms for the minor exam.

There are 4 questions for a total of 100 points.

---

1. (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 traveling along that road (this could be tolls, gas costs, 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 minimized.

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

2. 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, in order 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 is the optimal 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 on that day. Stay there until it becomes unavailable, and then repeat with the remaining days.

   (a) Fill in the steps of the proof of the following modify-the-solution lemma:

   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.

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

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

   iii. (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 a pseudocode of an efficient algorithm implementing the above strategy and give a time analysis for your algorithm.

```
GreedyHotels(Avail[1..k, 1..n])
  - Day = 0.
  - While Day < n do:
    - MaxEnd = Day
    - MaxHotel = 0
    - FOR J = 1 TO k do:
      - ThisEnd = Day
      - While ThisEnd < n and Avail[J, ThisEnd + 1] do: ThisEnd++
      - IF ThisEnd > MaxEnd THEN do: MaxEnd = ThisEnd, MaxHotel = J.
    - FOR d = Day + 1 to MaxEnd do: Stay[d] = MaxHotel
    - Day = MaxEnd
  - Return Stay[1..n]
```

3. 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 smallest number to largest.

   Your input is a $k \times n$ array of booleans $Avail[i, j]$ that tells you whether park i is available on 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 day 3 only. Park 3 is available 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.

   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.

```
GreedyParks(Avail[1..k, 1..n])
  - Day = 1, Park = 1.
  - While Day ≤ n and Park ≤ k do:
    - While (Avail[Park, Day] == False and Park ≤ k) do: Park++;
    - IF Park > k THEN return "Impossible" ELSE Stay[Day] = Park, Day++, Park++
  - IF Days > n return Stays[1..n] ELSE return "Impossible".
```

4. (40 points) (**Monotone Matchings**) Let G be a bipartite graph, with $L = \{u_1, ..u_l\}$ the set of nodes on the left, $R = \{v_1, ..v_r\}$ the set of nodes on the right, E the set of edges, each with one endpoint in L and the other in R, and $m = |E|$ the number of edges.

   A *matching* in G is a set of edges $M \subseteq E$ so that no two edges in M share an endpoint (neither the one in L nor the one in R). A matching M is *monotone* if for every two edges (u_{i_1}, v_{j_1}) and (u_{i_2}, v_{j_2}) in M, if $i_1 < i_2$ then $j_1 < j_2$. One could draw all the edges in the matching without crossing if the nodes are put in order on the two sides.

   The problem is, given a bipartite graph G, find the largest monotone matching in G.

   Assume $l \leq r$. Then a monotone matching M is *perfect* if it has size l, i.e., $|M| = l$.

   Below is a greedy strategy for the largest monotone matching problem.

   > **Candidate Strategy A**: For each $i = 1$ to l, if u_i has at least one undeleted neighbor v_j, match it to the unmatched neighbor with smallest value of j. Then delete u_i and $v_1, ...v_j$, and repeat.

   (a) Give a counter-example where the greedy strategy fails to produce an optimal solution. (*Hint: Since you will show that the algorithm works when the maximum monotone matching is perfect, your example shouldn't have a perfect monotone matching.*)

   (b) Illustrate the above strategy on the following graph with a perfect matching: $L = \{u_1, u_2, u_3\}$,$R = \{v_1, v_2, v_3, v_4, v_5\}$, and $E = \{(u_1, v_2), (u_1, v_3), (u_1, v_4), (u_2, v_1), (u_2, v_2), (u_2, v_3), (u_2, v_5)(u_3, v_1), (u_3, v_5)\}$.

   (c) Prove that, if G has a perfect matching, then Candidate Strategy A finds one.

   > *Hint: Let strategy A match u_i with v_{j_i} (unless it can't be matched). Let OPT be a perfect matching that matches each u_i with v_{k_i}. (Note: all left nodes will be matched by OPT, since it is perfect.) Use one of the following two methods.*
   >
   > - *Exchange method: Prove by induction on T that there is a left-perfect matching OPT_T that matches each u_i with v_{j_i} for $1 \leq i \leq T$.*
   > - *Greedy-stays ahead: Prove by induction on T that v_{j_T} exists and that $j_T \leq k_T$.*

   (d) Describe an efficient algorithm that carries out the strategy. Your description should specify which data structures you use, and any pre-processing steps. Assume the graph is given in adjacency list format. Give a time analysis, in terms of l, r and m.