Problem Set: Greedy Algorihtms

Problem 1. Recall the interval scheduling problem. Given a set of requests $r_1, r_2, \cdots, r_n$, the start and finish time of request r_i is denoted by s_i and f_i. Requests are compatible if they do not overlap in time. In order to find the largest compatible subset of requests, we proved in class that scheduling the requests by *earliest finishing time* is optimal.

1. Suppose that instead of always selecting the first request to finish, we instead select the last request to start that is compatible with all previously selected requests, i.e. requests are scheduled by *latest starting time*. Describe how this approach is a greedy algorithm, and prove that it yields an optimal solution.

2. Suppose that each request r_i is assigned a value v_i. The objective is no longer to maximize the number of requests processed, but instead to *maximize the total value of the requests processed*, i.e. we want to choose a set A of compatible requests such that $\sum_{v_i \in A} v_i$ is maximum. This is called the *weighted interval scheduling problem.* Is the earliest finishing time greedy algorithm an optimal solution to this problem? Justify your answer.

Problem 2. An optimal greedy algorithm for job scheduling is the so-called *STCF: Shortest-Time-to-Completion-First*. Given n jobs, where job j_i has starting time s_i and processing time p_i, at each point, the available job with the shortest remaining processing time is scheduled. Write pesudo-code to implement this in $O(n \log n)$. Briefly explain the runtime of the algorithm. [*Hint:* You can use a priority queue to solve this.]

Problem 3. Suppose a job j_i is given by two numbers d_i and p_i, where d_i is the deadline and p_i is the penalty. The length of each job is equal to 1 minute. All n such jobs are to be scheduled, but only one job can run at any given time. If job j_i does not complete before its deadline d_i, its penalty p_i should be paid. Design a greedy algorithm to find a schedule which minimizes the sum of penalties and prove its optimality.

Problem 4. Jane has bought a new house and wants to have a house-warming party. She is deciding whom to invite from the n people she knows. She has made up a list of all pairs of these people who know each other. Assuming that all invitees come to the party, she wants to invite as many people as possible such that each person should have at least five other people whom they know and at least five other people whom they do not know. Give an efficient algorithm that takes as input the list of n people and the list of all pairs who know each other and outputs a largest subset of these n people which satisfies the constraints. Briefly argue about the optimatily of your algorithm, i.e. it indeed selects the subset with the largest possible number of invitees.

Problem 5. A group of friends want to hike a canyon trail. They collectively that for safety reasons, they do not want to hike in the dark after nightfall. On a map, they have identified a series of *stopping points* for camping along the route. We make the following assumptions:

- The group can hike d km per day irrespective of area, weather conditions etc.
- The distance between two adjacent stopping points is *at most* d km.
- The first and last stops are at most d km from the start and end of the trail respectively.
- The group has complete information of the distances between stopping points on their map.

The group wants to minimize the number of camping stops they make. Devise a greedy algorithm for determining the set of stopping points the group should camp at such that the number of stops is minimum.
Hint: The group should hike as much as possible during the day.

Problem 6. A thief enters a store and sees a set I of n items, $I = \{a_1, a_2, \cdots, a_n\}$. Each item has an associated weight w_i and value v_i. Ideally, the thief would like to steal everything in order to gain maximum benefit. However, there is only so much he can carry. The thief has a knapsack with capacity C. The thief now has to determine which items to steal so that their total weight does not exceed C and their total value is maximum.

1. Suppose the items are such that a fraction of an item can be taken, i.e. the thief can take some part of an item (for example, $0.4w_i$ of a_i) and leave the remaining part. This is called the *Fractional Knapsack Problem*. Let $A \subset I$ be the subset of items that the thief steals. Devise an $O(n \log n)$ greedy algorithm to find the subset A such that $\sum_{a_i \in A} w_i < C$ and $\sum_{a_i \in A} v_i$ is maximum.

2. Suppose items can be taken as a whole, i.e. the thief can only take or leave an item; he can not take a fraction of an item. This is called the *Binary Knapsack Problem.* Does your greedy algorithm for the fractional version of the problem (previous question) still find an optimal solution? Justify your answer.

Problem 7. Coins of a set of denominations (values) are available to a cashier who needs to provide change for a given amount V, such that the number of coins returned in exchange for V is minimum. We assume an infinite supply of each denomination.

1. Given the denomination set $\{1, 5, 10\}$, devise a greedy algorithm to find the minimum number of coins, that can be exchanged for V.

2. Is your greedy algorithm optimal for *any* given set of denominations? Justify.

Problem 8. What is an optimal Huffman code for the following set of frequencies, based on the first 8 Fibonacci numbers? $a : 1 \quad b : 1 \quad c : 2 \quad d : 3 \quad e : 5 \quad f : 8 \quad g : 13 \quad h : 21$
Can you generalize your answer to find the optimal code when the frequencies are the first n Fibonacci numbers?

Problem 9. Consider the Huffman encoding scheme. Prove that:

1. If some character occurs with frequency more than $\frac{2}{5}$, then there is guaranteed to be a code of length 1.

2. If all characters occur with frequency less than $\frac{1}{3}$, then there is guaranteed to be no code of length 1.

Problem 10. Suppose a newly developed 'ternary' hard disk can store values 0, 1, or 2 (instead of just 0 or 1). To take advantage of this new technology, provide a modified Huffman algorithm for compressing sequences of characters from an alphabet of size n, where the characters occur with known frequencies $f_1, f_2, \cdots, f_n$. Your algorithm should be lossless and optimal, i.e. it should encode each character with a variable-length code over the values $0, 1, 2$ such that no code is a prefix of another code and should obtain the maximum possible compression. Prove (informally) the correctness of your algorithm.