Design and Analysis of Algorithms (CS345A)

Practice Sheet on Greedy Algorithms

1. (**A generic method to design and analyse greedy algorithms**)

   For the synchronizing a circuit to minimize the total delay as well as the problem of job scheduling to minimize the maximum lateness, design and analyse the algorithms using the generic method we discussed in the class. You should also try out using this generic method for other problems in this assignment.

2. (**A job scheduling problem**)

   There is a set of n jobs. Each job has a start time and finish time. There is a single processor and it can execute only one job at a time. Design an algorithm to compute the largest subset $R \subseteq S$ of jobs that can be executed on the processor. It can be observed that no two jobs in R are allowed to overlap. You may assume for simplicity that no two jobs have same start or finish times.

   Hint: This problem is rephrasing of interval scheduling problem which you might have studied in ESO207.

3. (**Placing mobile towers**)

   There is a long and straight road. There are n houses located along the road. You need to place mobile towers so that each house has at least one tower within distance c from it. Design an efficient algorithm to compute the minimum number of towers to be placed to achieve this objective. You must give a formal proof of correctness for the algorithm you design.

4. (**Scheduling jobs on PCs and a supercomputer**)

   There are n jobs. There are n PCs and one supercomputer. Each job consists of two stages: first it needs to be preprocessed on the supercomputer, and then it needs to be finished on one of the PCs. Let us say that job J_i needs p_i seconds of time on the supercomputer, followed by f_i seconds of time on a PC. Since there are n PCs available, the finishing of the jobs can be performed in parallel - all jobs can be processed at the same time on their respective PCs. However, the supercomputer can only work on a single job at a time. So we need to find out the order in which to feed the jobs to the supercomputer. Our aim is to minimize the *completion* time of the schedule which is defined as the earliest time at which all jobs will have finished processing on the PCs. Design a greedy algorithm that finds the order in which the jobs should be scheduled on the supercomputer so that completion time achieved is as small as possible.

   Sort in descending order according to the finish time on PC.

   Hint: What would have been the completion time if p_i, the processing time of J_i on Supercomputer, had been zero ? In particular, which job would have determined the completion time in this situation ? Observe that the processing of jobs on supercomputer just delays their executions on PC. So, in order to minimize the completion time, which job should be scheduled first?

5. (**Scheduling to maximize happiness**)

A photocopy shop has a single large machine. Each morning the shop receives a set of jobs from customers. The shopkeeper wants to do the jobs on the single photocopying machine in an order that keeps their customers happiest. Customer i's job will take t_i time to complete. Given a schedule (ordering of the jobs), let C_i denote the finishing time of job i. For example, if job i is the first to be done, we would have $C_j = t_j$; and if job j is done right after job i, we would have $C_j = C_i + t_j$. Each customer has a given weight w_i that represents his or her importance to the business. The happiness of customer i is expected to be dependent on the finishing time of i's job. So the company decides that they want to order the jobs to minimize the weighted sum of the completion time, $\sum_{i=1}^{n} w_i C_i$.

Design an efficient algorithm to solve this problem. That is, you are given a set of n jobs: job i has a processing time t_i and a weight w_i. You want to order the jobs so as to minimize the weighted sum of the completion time, $\sum_{i=1}^{n} w_i C_i$.

check the order of ti/wi. sort them in non decreasing order, that is your required order..

Hint: Suppose there are only 2 jobs. Try to determine the order in which they should be executed to minimize the weighted sum of the completion time. Once you are able to find an order for a pair of jobs, try to generalize along similar lines to a problem we discussed in a lecture.

6. (**A variant of interval scheduling problem**)

You have a processor that can operate 24 hours a day, every day. People submit requests to run daily jobs on the processor. Each such job comes with a start time and an end time; if the job is accepted to run on the processor, it must run continuously, every day, for the period between its start and end times. Note that certain jobs can begin before midnight and end after midnight; this makes for a type of situation different from what we see in the interval scheduling problem.

Given a list of n such jobs, your goal is to accept as many jobs as possible (regardless of their length), subject to to constraint that the processor can run at most one job at any given point of time. Provide an algorithm to do this with a running time polynomial in n. You may assume for simplicity that no two jobs have same start or end times.

Example: Consider four jobs : (6PM, 6AM), (9PM, 4AM), (3AM, 2PM), (1PM, 7 PM).

The optimal solution would be to pick the two jobs (9PM, 4AM)and (1PM, 7 PM), which can be scheduled without overlapping.

Hint: The problem can be viewed as an interval scheduling problem on a circle instead of a line. The main hurdle is that there is no specific "origin". However, you can solve this problem by invoking the algorithm for the scheduling problem on a line "multiple" times.

Check the MST only at t1 and t2. Divide the edges into 4 types:
a. both included before implies both included after
b. none included before implies none included after
c. upper included before implies lower included after
d. lower included before
d1. upper included after(coz lower was not eliminating upper one)
d2. lower included after(coz lower was eliminating upper one)
only in d2, the weight of MST will change. let's say the slope of all these lines only decrease with time. thus one only needs to check at the start and end of the time

7. (**Time varying MST**)

   Given an undirected weighted graph $G = (V, E)$, where each edge $e \in E$ has two parameters a_e and b_e. The weight of edge e at time t is equal to $a_e t + b_e$. Your aim is to monitor the minimum spanning tree of the graph during a time interval $[t_1, t_2]$. For this purpose, you wish to compute the minimum spanning tree which has the least weight in the interval $[t_1, t_2]$. Design an $O(m \log n)$ time algorithm for this problem.

   Hint: Look at the desired time complexity. It provides a powerful hint. How does the weight of any spanning tree vary in this time interval ? Make use of the answer to this question to solve the problem.

8. (**Maximum capacity path**)

   Given a directed graph $G = (V, E)$ where each edge e has a capacity $c(e) > 0$. There are a source vertex $s \in V$ and a sink vertex $t \in V$. For any path P from s to t, we define its capacity as the capacity of the the least capacity edge lying on this path. Design an efficient algorithm to compute the path of maximum capacity from s to t in G.

   Hint: Get inspiration from the Dijkstra's algorithm we discussed in the class and start from scratch.

9. (**Shortest paths in graphs with integer weights**)

   Let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \rightarrow \{0, 1, ..., W - 1\}$ for a given positive integer W. How will you modify the Dijkstra's algorithm to compute shortest paths from a given source vertex s in $O(|W|n + m)$ time ? How will you improve the time to $O((n + m) \log W)$?

   Hint: How many distinct values can there be for $\{L(v) \mid v$ is a neighbor of $S_i\}$, where S_i is the set of i nearest vertices from the source vertex s ?

10. (**Special shortest paths**)

    Given a directed graph on n vertices and m edges, where each edge has been assigned a positive weight. There are two parameters that characterize *lightness* of a path P: the first parameter is its *length* $\ell(P)$ which is defined as the number of edges of P. The second parameter is its *weight* $w(P)$ which is defined as the sum of weights of all its edges. A path P_1 is said to be *lighter* than path P_2 if either $\ell(P_1) < \ell(P_2)$ or $\ell(P_1) = \ell(P_2)$ and $w(P_1) < w(P_2)$. A path P from u to v is said to be the lightest path from u to v if no other path from u to v is lighter than P. Design an $O(m+n)$ time algorithm that for a source s and a destination d, compute the lightest path from s to d.

    Hint: The time complexity is a big hint. Which is the well-known graph algorithm that takes so less time ?

Find the BFS order traversal of all nodes starting from the root node. then traverse all the nodes in the found order.
At each node, take the distance of that node as the min of weight of paths from the nodes lying in the previous level which have edges to that node.
2*O(n+m)

An adventurous drive in Thar desert

(Note: This problem is just for those students who love algorithms. This problem will never be asked in any exam/quiz of this course.)

Your friend X has recently purchased a THUNDERBIRD motorcycle from Royal Enfield in India. He is very excited and wants to drive it from some town s to another town d in Thar desert (think of Sahara desert if you are more adventurous). He is provided with the complete road map of the desert - the roads, their lengths and junctions (where two or more roads meet). The mobike has a very natural limitation - it can drive for c kilometers with full fuel tank. Since the destination is very far, X must plan his route so that he can refill the fuel tank along the way. Note that the shortest route may not necessarily be the feasible route. You friend is bit confused and scared - what if he gets lost in Thar desert due to improper planning. Your friend is very proud of you since you are a student of IITK. He also knows that you have done a course on algorithms. He approaches you with full faith that you will help him find the shortest feasible route from s to d if it exists. Model the problem in terms of a weighted, undirected graph in which roads are edges, junctions are vertices, and some junctions have fuel stations, and design an efficient algorithm to find shortest feasible route from s to d and inform if no route is feasible. Assume that both s and d are also at junctions and the source s has a fuel station. Your algorithm should take time of the order of $O(mn \log n)$ where m is the number of edges and n is the number of vertices. You should not exploit the planarity of the input graph while designing your algorithm. In other words, your algorithm should work for any arbitrary undirected graph with positive edge weights and under the fuel constraint mentioned above. Also note that the shortest feasible route need not be a simple path. For example, consider the following situation : There are five towns : s, a, b, e, d where fuel stations are available at s, b, e only. The connecting roads are : (s, a) of length 200 km, (s, d) of length 300 km, (a, b) of length 40 km, (a, d) of length 150 km, (b, e) of length 200 km, (e, d) of length 200 km; and motor bike can travel 250 km with full fuel tank. For this road network, shortest feasible route from s to d is is $s \to a \to b \to a \to d$, and has length 430 km.