EE360C: Algorithms
University of Texas at Austin
Dr. David Soloveichik

Homework #4
Due: October 18, 2018 (*in-class quiz*)

Homework #4

You do not need to turn in these problems. The goal is to be ready for the in-class quiz that will cover the same or similar problems, and to prepare you for the exams.

Problem 1: Discipline of Children

Your job is to arrange n rambunctious children in a straight line, facing front. You are given a list of m statements of the form "i hates j". If i hates j, then you do not want to put i somewhere behind j because then i is capable of throwing something at j. Give an algorithm that orders the line, or says that such an order does not exist, in $O(m+n)$ time.

Solution

Create a graph G in which each child is a vertex and every statement of the form "i hates j" results in a directed edge from i to j. Topologically sort the vertices. Use a modified topological sort that returns "false" if the graph has a cycle, which you'll identify if at some point in the execution you can't find a node with no incoming edges.

The order of the output vertices (children) is the reverse order to put them in line; you can reverse the order in $O(n)$ time (or you could have made edges from j to i instead of from i to j). Creating the graph takes $O(n+m)$ time; running topological sort takes $O(V+E) = O(m+n)$ time.

Problem 2: Course room assignment

You have n lectures that need to be allocated rooms. Each lecture i has its own start time s_i and finish time f_i. Your goal is to assign lectures to rooms using as few rooms as possible. Of course, if two lectures overlap in time, they cannot share a room.

(a) Consider the following greedy algorithm skeleton:

```
Sort lectures in some way (to be determined)
d := 0   ← number of allocated classrooms

for j = 1 to n {
   if (lecture j is compatible with some classroom k)
      schedule lecture j in classroom k
   else
      allocate a new classroom d + 1
      schedule lecture j in classroom d + 1
      d := d + 1
}
```

You remember that you learned about interval scheduling in class, and for that problem you needed to sort by finish times. Show by counter-example that sorting by finish times doesn't always work here.

(b) Prove that sorting by start time always gives the optimal solution.

Solution

Suppose the algorithm ends with d classrooms used. We want to show that d is the minimum number of classrooms needed.

Consider the step when the algorithm increases d to its final value and let j be the lecture under consideration. At this point lecture j conflicts with one (or more) lectures(s) in each of the $d-1$ classrooms currently used. But all these lectures have start time no later than s_j since the lectures were processed in order of their start time. Since they conflict with j, they each have finishing time after s_j. Hence they all conflict with each other as well (consider time s_j). Hence we really do need d machines.

(c) Show that your algorithm is $O(n \log n)$.

Solution

(This algorithm is analyzed in the textbook in Chapter 4 starting page 122.) For each classroom k, maintain the finish time of the last lecture added. Keep the classrooms in a priority queue. Then the loops happens n times, and each iteration takes $O(\log n)$ time.

Problem 3: Dijkstra's algorithm with negative weights

Edge weights in weighted graphs don't have to represent lengths. Imagine you are trying to model business scenarios and edge weights represent costs and benefits of different paths through the "space of business choices". In this case it makes sense to have weights that are both positive (spend money) and negative (earn money). A "shortest path" represents a set of business choices with lowest total cost. (Assume that there is no cycle of negative total weight. If there were a cycle of negative total weight, then going around this cycle arbitrarily many times, you could get the total weight of a path to be as negative as you like, and then shortest paths are not well defined.)

Unfortunately, Dijkstra's algorithm doesn't work if some weights can be negative. Explain why Dijkstra's algorithm could give incorrect shortest paths if some weights are negative. Where does the proof of correctness we did in class fail? (Later in the class, we'll learn about a shortest path algorithm that *can* handle negative weights.)

Solution

In Dijkstra's correctness proof by induction, it is shown that all nodes removed from Q(i.e. the "pool" of nodes for which we haven't found the shortest path from start node s yet) have $v.d = \delta(s, v)$. The final step in this proof is arguing that for the node v that was last relaxed on node u and was extracted in step $k+1$, it holds that $v.d \leq y.d \leq \delta(s, x) + w(x, y)$, and therefore a path from s to v through node y would be longer than $v.d$. However, when negative weights are allowed, it is possible that $w(y, v)$ has a negative value α s.t. $v.d > y.d + w(y, v)$, i.e. the path from s to v through node y could be longer.

Problem 4: Phone Base Stations

Consider a long, quiet country road with houses scattered sparsely along it. (We can picture the road as a long line segment, with an eastern endpoint and a western endpoint.) Further, let's suppose that despite the bucolic setting, the residents of all these houses are avid cell phone users. You want to place cell phone base stations at certain points along the road, so that every house is within 4 miles of one of the base stations. Give a greedy efficient algorithm that achieves this goal,

using as few base stations as possible. Prove that the greedy choice that your algorithm makes is the optimal choice.

Solution

Start at the western end of the road and begin moving east until the first moment when there's a house h exactly four miles to the west. We place a base station at this point (if we went any further east without placing a base station, we wouldn't cover h). We then delete all of the houses covered by this base station and iterate on the remaining houses.

For any point on the road, define its *position* to be the number of miles it is from the western end. We place the first base station at the easternmost point (i.e., largest position) s_1 with the property that all houses between 0 and s_1 will be covered by s_1. In general, having placed $\{s_1, \ldots, s_i\}$, we place base station $i+1$ at the largest position s_{i+1} with the property that all houses between s_i and s_{i+1} will be covered by s_i and s_{i+1}.

Let $S = \{s_1, \ldots, s_k\}$ denote the full set of base station positions that our greedy algorithm places, and let $T = \{t_1, \ldots, t_m\}$ denote the set of base station positions in an optimal solution, sorted in increasing order (i.e., from west to east). We must show that $k = m$.

We do this by showing a sense in which our greedy solution S "stays ahead" of the optimal solution T. Specifically, we claim that $s_i \geq t_i$ for each i, and prove this by induction. The claim is true for $i = 1$ since we go as far as possible to the east before placing the first base station. Assume now that it is true for some value $i \geq 1$; this means that our algorithm's first i centers $\{s_1, \ldots, s_i\}$ cover all of the houses covered by the first i centers $\{t_1, \ldots, t_i\}$. As a result, if we add t_{i+1} to $\{s_1, \ldots, s_i\}$, we will not leave any house between s_i and t_{i+1} uncovered. by the $(i+1)^{st}$ step of the greedy algorithm choose s_{i+1} to be *as large as possible* subject to the condition of covering all houses between s_i and s_{i+1}, so we have $s_{i+1} \geq t_{i+1}$. This proves the claim by induction.

Finally if $k > m$, then $\{s_1, \ldots, s_m\}$ fails to cover all houses. But $s_m \geq t_m$, and so $\{t_1, \ldots, t_m\}$ also fails to cover all houses—a contradiction to it being the optimal solution.

Problem 5: Fractional Knapsack Problem

A thief is given the choice of n objects to steal, but only has one knapsack with a capacity of taking M weight. Each object i has weight w_i, and profit p_i.

(a) First, suppose that the objects are divisible (e.g., the thief is in a cheese shop and the items are rolls of cheese that can be cut). For each object i, if a fraction x_i, $0 \leq x_i \leq 1$ (taking $x_i = 1$ would be taking the entire object) is placed in the knapsack, then the profit earned is $p_i x_i$. Come up with an efficient greedy algorithm for maximizing the profit of the thief. Prove the correctness and running time.

Solution

Efficient Greedy Algorithm. If all objects fit in the knapsack, take them all. Otherwise: $\forall i$, set $x_i = 0$. Index the objects as $1, ..., n$ s.t. for any two objects with indices i and j respectively, $i < j \iff p_i/w_i \geq p_j/w_j$, i.e. the items are sorted in decreasing order of the ratio p_i/w_i. Let k be the maximum non-negative integer s.t. $\sum_{i=1}^{k} w_i \leq M$. Then, $\forall i \in \{1, 2, .., k\}$, set $x_i = 1$. Finally, if there is still space in the knapsack, i.e. if $M - \sum_{i=1}^{k} w_i = \alpha > 0$, set $x_{k+1} = a/w_{k+1}$ (i.e. fill the remaining space with the largest possible quantity of object $k + 1$).

Correctness. The key argument for the algorithm's correctness is that an optimal solution would have to take as much quantity as possible of object 1 (i.e. the object i with the highest p_i/w_i ratio). We can prove this by contradiction: Assume that $\exists O = \{x_1 = o_1, ..., x_n = o_n\}$ s.t. O is an optimal solution with value $P(O)$ and O doesn't take the greatest amount of object 1 possible, i.e. $o_1 < min\{1, M/w_i\}$. There are two possible cases for solution O. Either the knapsack is full or it isn't. If it isn't, we can increase the amount of item i that we take, and thus have a solution of value greater than that of O. Since we assumed that O is optimal, this is a contradiction. If the knapsack is full, $\exists j \neq 1$ s.t. $o_j > 0$. By removing ϵ weight of item j, s.t. $\epsilon \in (0, min\{w_1 - o_1, w_j * o_j\}]$, and adding ϵ weight of item 1 in its place, we get a solution that has value $P(O) - \epsilon * p_j/w_j + \epsilon * p_2/w_2 = P(O) + \epsilon * (p_2/w_2 - p_j/w_j) > P(O)$. Since we assumed that O is optimal, this again is a contradiction.

Applying this argument recursively, we get that our greedy algorithm outputs an optimal solution.

Running Time. Sorting takes $O(nlogn)$ time. Iterating through our list of objects to figure out how many of them can be added to the solution takes $O(n)$ time. Total $O(nlogn)$.

(b) Now, suppose the items cannot be taken fractionally. In other words, the thief can either take an entire item or leave it behind. Suppose we also know the following: the order of these items when sorted by increasing weight is the same as their order when sorted by decreasing value. Give a greedy algorithm to find an optimal solution to this variant of the knapsack problem. Prove the correctness and running time.

Solution

Very similar to part (a).

Efficient Greedy Algorithm. If all objects fit in the knapsack, take them all. Otherwise: $\forall i$, set $x_i = 0$. Index the objects as $1, ..., n$ s.t. for any two objects with indices i and j respectively, $i < j \iff p_i \geq p_j$. Let k be the maximum non-negative integer s.t. $\sum_{i=1}^{k} w_i \leq M$. Then, $\forall i \in \{1, 2, .., k\}$, set $x_i = 1$.

Correctness. Proof by contradiction. Assume there is an optimal solution O that does not include the first k objects. Let $i \in 1, ..., k$ be an object that is not included in O. There either is an item $j > i$ in O, or then isn't. If there isn't, we can add i to the knapsack and get a higher value than that of O. This is a contradiction. If there is such an item j, we can remove j and add i in its place, since $w_j \geq w_i$, and get a solution with a higher value than that of O, since $v_j \leq v_i$. Again, this is a contradiction to our initial assumption. Hence, the optimal solution must include the first k objects, so our algorithm is correct.

Running Time. Sorting takes $O(nlogn)$ time. Iterating through our list of objects to figure out how many of them can be added to the solution takes $O(n)$ time. Total $O(nlogn)$.