Problem Set #6

You should try to solve these problems by yourself. I recommend that you start early and get help in office hours if needed. If you find it helpful to discuss problems with other students, go for it. **You do not need to turn these problems in. The goal is to be ready for the in class quiz that will cover the same or similar problems.**

Problem 1: Counting Shortest Paths

A number of art museums around the country have been featuring work by an artist named Mark Lombardi (1951-2000), consisting of a set of intricately rendered graphs. Building on a great deal of research, these graphs encode the relationships among people involved in major political scandals over the past several decades: the nodes correspond to participants, and each edge indicates some type of relationship between a pair of participants. And so, if you peer closely enough at the drawings, you can trace out ominous looking paths from a high-ranking U.S. government official to a former business partner to a bank in Switzerland to a shadowy arms dealer.

Such pictures form striking examples of *social networks*, which have nodes representing people and organizations and edges representing relationships of various kinds. And the short paths that abound in these networks have attracted considerable attention recently, as people ponder what they mean. In the case of Mark Lombardi's graphs, they hint at the short set of steps that can carry you from the reputable to the disreputable.

Of course, a single, spurious short path between nodes v and w in such a network may be more coincidental than anything else; a large number of short paths between v and w can be much more convincing. So in addition to the problem of computing a single shortest v-w path in a graph G, social networks researchers have looked at the problem of determining the *number* of shortest v-w paths.

This turns out to be a problem that can be solved efficiently. Suppose we are given an undirected graph $G = (V, E)$, and we identify two nodes v and w in G. Give an algorithm that computes the number of shortest v-w paths in G. (The algorithm should not list all the paths; just the number suffices.) The running time of your algorithm should be $O(m+n)$ for a graph with n nodes and m edges.

Solution

We will solve the more general problem of computing the number of shortest paths from v to *every* other node.

We perform BFS from v, obtaining a set of layers $L_0, L_1, L_2, \ldots$, where $L_0 = \{v\}$. By the definition of BFS, a path from v to a node x is a shortest v-x path if and only if the layer numbers of the nodes on the path increase by exactly one in each step.

We use this observation to comptue the number of shortest paths from v to each other node x. Let $S(x)$ denote this number for a node x. For each node x in L_1, we have $S(x) = 1$, since the only shortest-path consists of the single edge from v to x. Now consider a node y in layer L_j for $j > 1$. The shortest v-y paths all have the following form: they are a shortest path to some node x in layer L_{j-1}, and then they take one more step to get to y. Thus $S(y)$ is the sum of $S(x)$ over all nodes x in layer L_{j-1} with an edge to y.

After performing BFS, we can compute all these values in order of the layers; the time spent to compute a given $S(y)$ is at most the degree of y (wince at most this many terms figure into the sum from the previous paragraph). Since we have seen that the sum of degrees in a graph is $O(m)$, this gives an overall running time of $O(m + n)$.

Problem 2: Greedy Choice

You are consulting for a trucking company that does a large amount of business shipping packages between New York and Boston. The volume is high enough that they have to send several trucks each day between the two locations. Trucks have a fixed limit W on the maximum amount of weight they are allowed to carry. Boxes arrive at the New York station one by one, and each package i has a weight w_i. The trucking station is quite small, so at most one truck can be in the station at any time. Company policy requires that boxes are shipped in the order they arrive; otherwise a customer might get upset. At the moment, the company is using a simple greedy algorithm for packing: they pack boxes in the order they arrive, and whenever the next box does not fit, they send the truck on its way.

But they wonder if they might be using too many trucks, and they want your opinion on whether the situation can be improved. Here is how they are thinking. Maybe one could decrease the number of trucks needed by sometimes sending off a truck that was less full, and in this way allow the next few trucks to be better packed.

Prove that, for a given set of boxes with specified weights, the greedy algorithm currently in use actually minimizes the number of trucks that are needed.

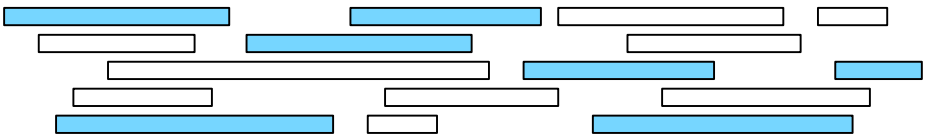
Solution

We prove that the greedy algorithm uses the fewest possible trucks by showing that it "stays ahead" of any other solution. If the greedy algorithm fits boxes $b_1, b_2, \ldots b_j$ into the first k trucks, and the other solution first $b_1, \ldots b_i$ into the first k trucks, then $i \leq j$. We will prove this by induction on k. The base case is easy; $k = 1$ and we only require a single truck to send all of the packages. Now, assuming the above holds for $k - 1$: the greedy algorithm fits j' boxes into the first $k - 1$ trucks and the other algorithm fits i' boxes into the first $k-1$ trucks; $i' \leq j'$. Now for the k^{th} truch, the alternate solution puts in boxes $b_{i'+1}, \ldots, b_i$. Thus since $j' \geq i'$, the greedy algorithm is able at least to fit all the boxes $b_{j'+1}, \ldots, b_i$, if not more into the k^{th} truck.

Problem 3: Intervals

Let X be a set of n intervals on the real line. A subset of intervals $Y \subseteq X$ is called a *tiling path* if the intervals in Y cover the intervals in X, that is, any real value that is contained in some interval in X is also contained in some interval in Y. The *size* of a tiling cover is just the number of intervals.

Describe an algorithm to compute the smallest tiling path of X. Assume that your input consists of two arrays $X_L[1..n]$ and $X_R[1..n]$, representing the left and right endpoints of the intervals in X. Argue that your algorithm is correct (hint: remember that an argument of correctness for any greedy algorithm has two components). The figure below shows an example of a tiling path, though a non-optimal one.



A set of intervals. The seven shaded intervals form a tiling path.

Solution

Sort both X_L and X_R so that the jobs with the earliest start times are first. Include the first interval (the one with the earliest start time). It has to be in your tiling. If there are multiple intervals with the same start time (and they're all the earliest start time), choose the one that has the lastest finish time (after all, it gets you the most bang for your buck). Then remove everything that is overlapped completely by the interval you just selected. (There's no use including any of these since they don't add anything. Now recurse. After the first subproblem, you want to look at all of the intervals that include the finish time of the interval you just included. Of these intervals, you want to pick the one of them that has the latest finish time. That is, say the greedy choice in the previous step was to include interval i. After removing all of the intervals that are completely overlapped by interval i, the next greedy choice is an interval that starts on or before $X_R[i]$ and finishes the latest of all such intervals.

The problem exhibits optimal substructure. After making the greedy choice and removing all of the intervals that are completely overlapped by the greedily chosen interval, we're left with a smaller problem (the original set X minus some intervals) and a smaller piece of the line to cover (specifically, we're left with the portion of the line that is NOT covered by the greedily chosen interval).

Suppose that the greedy choice was not good. That is, suppose that you hand me an optimal tiling path that does not contain the earliest starting interval (and of the earliest starting intervals the one with the latest start time). I can take your tiling path and remove the interval that has the earliest starting time in it and replace it with the greedy choice. Either my interval starts before yours (in which case, your solution wasn't correct, since you didn't cover the entire line) or they start at the same time. In the latter case, I've only covered *more* of the line since my selected interval has a later finishing time than yours (since I selected the earliest starting one with the latest finishing time). Either that or our intervals were exactly the same.

Problem 4: Greedy Scheduling

Consider a situation where you have to find available classrooms for n different lectures. Of course, you must avoid scheduling two or more overlapping lecture in the same room. Each lecture i begins at s_i and ends at t_i.

(a) Find an algorithm that assigns the smallest number of rooms possible. (Hint: Look at slides 10-12 in from Lecture 8)

Solution

The greedy choice we will make to solve this problem is to schedule the earliest compatible classes first. That is to say, we will take the earliest class that has not been scheduled, assign it to a new classroom. Then we will take the earliest class that doesn't overlap with this class, and assign it to the same classroom. We will repeat this until no more classes can be assigned to this room, then we will create a new room, and repeat the process.
We will sort the list of classes, c by starting time before running the following algorithm on it.

GreedyRooms(c)

```
1  d ← 0 (number of allocated classrooms)
2  for j = 1 to n
3        do if class j is compatible with a classroom, k
4              then schedule j in classroom k
5                   update the finish time of classroom k
6              else allocate a new classroom d + 1
7                   schedule class j in classroom d + 1
8                   d ← d + 1
```

Each classroom k maintains the finish time of the last job added.
Classrooms are kept in a priority queue.
The running time of this algorithm is $O(n \log(n))$

(b) Show that your algorithm is optimal.

Solution

The problem exhibits optimal substructure. If we remove a subset of classes that were scheduled in any classroom d, the smallest number of classrooms we could assign our new set of classes to is $d - 1$.
In our algorithm, we start a new classroom d because we need to schedule a job j that is incompatible with all other classrooms $1...d - 1$. Because we chose to schedule greedily by start time, the incompatibilities in classrooms $1...d - 1$ are caused by lectures with start times $\leq s_j$. Thus, there are d lectures that overlap, and we would need $\geq d$ classrooms to schedule all of them.