Exercises: interval scheduling [Last updated March 3]

Questions

1. Consider the greedy algorithm (#3, see lecture) for interval scheduling which picks the compatible interval that finishes as soon as possible. We saw in class that this gives an optimal solution for the problem of finding the maximum number of intervals. Does it also find an optimal solution for the total duration of the intervals?

2. Given a set of N intervals, how many possible subsets are there, ignoring whether the intervals are compatible or not ?

3. Given a set of N intervals, give an efficient algorithm to check if they are *all* compatible.

4. As in the lecture, let p[n] be the index of the last interval to finish before interval n starts. Suppose the set of intervals is such that p[n] = n-2 for all n > 2. Draw an example of this situation. Show that the call tree for a recursive solution the weighted interval scheduling problem corresponds to the call tree for the recursive solution for computing the nth Fibonacci number.

5. [Updated Feb 23] Consider a set of N intervals defined by starting and finishing times as in the lecture. Define a graph with N vertices, one for each interval, such that there is an edge from vertex i to vertex j when interval i finishes before j begins. (See bottom left corner of p. 3 of the slides.) Suppose that interval i has a value vi, for example, the duration of the interval. We can assign this weight to each edge that terminates at vertex i. In class there were several questions raised that about this graph, including whether could we use something like Dijkstra's algorithm to solve the weighted interval scheduling problem. So, could you?

6. [Added Feb 23] Suppose you have a directed acyclic graph with positive weights on the edges. How could you find the longest path using dynamic programming ?

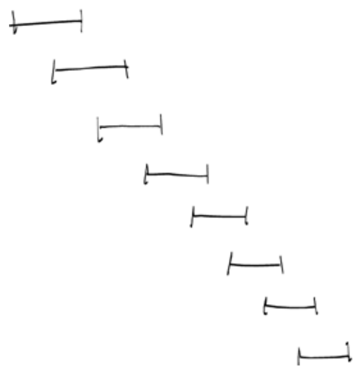
Answers

1. No. A counterexample is the intervals [1, 3], [2, 10]. The algorithm would pick the first interval, which is incompatible with the second, but the second has a longer duration.

2. 2^N. Why? Any subset of intervals defines a binary labelling of the subsets e.g. 1 if an interval is in the subset, or 0 if an interval is not in the subset. Since there are two possible labels per interval and the labellings are independent (if we ignore compatibility), then there are 2^N = 2*2*….*2 labellings in total. [Note that this is the same as asking how many ways are there to partition a set into two subsets? i.e. we label as element 0 if is in the first subset and 1 if it is in the second.]

3. One inefficient way to do it would be to take all pairs of intervals and check if each pair is compatible. This takes O(N^2), however, since there are O(N^2) pairs. A faster approach would be to being by sorting the intervals by finishing time, which takes O(N log N), and then checking that f(i) < s(i+1) for all I in 1 to N-1. That takes O(N).

4.



n	$p(n) = n-2$
1	0
2	0
3	1
4	2
5	3
6	4
7	5
8	6

Opt(n) calls Opt(n-1) and Opt(p[n]). If p[n] = n-2, then Opt(n) calls Opt(n-1) and Opt(n-2), just like Fibonacci.

5. [Updated Feb 23] There are several differences between the weighted interval scheduling problem and the problem solved by Dijkstra.

First, the graph defined in the question has a cost on all the edges. However keep in mind that a path with k edges has k+1 vertices, so we need to include the cost of an extra interval (the starting vertex of the path) in the formulation.

Second, Dijkstra's algorithm is about shortest paths (with positive weights) whereas here we are after a "longest" path. This might seem like a trivial difference, but its not. The proof that Dijkstra's algorithm finds the shortest path has very tight reasoning: it depends on the fact that taking a different path than Dijkstra never *decreases* the length of the path. But if we now want to find the longest path and we choose the next vertex in Dijkstra's algorithm to be the one that gives the locally longest path, then we can say nothing about whether that choice will lead to the longest path. It could happen that the path chosen by the new algorithm is locally the longest, but that some other local choice would eventually lead to a longer path.

Indeed there is a problem in graph theory called the longest path problem. E.g. http://en.wikipedia.org/wiki/Longest_path_problem. Although this problem does not have a fast solution in general, it does have a fast solution in the case that the graph is acyclic, which is indeed our situation here. See next question.

A final difference is that Dijkstra's algorithm finds shortest paths from a particular vertex, s. But here we do not know which interval (vertex) we want to start from. Interval 1 finishes first, but we don't necessarily want interval 1 in the solution.   In general we would need to consider all possible starting vertices for our paths.

6. [Updated March 3] Define a topological ordering on the DAG.   Let f(j) be the length of the longest path that uses vertex j and any of the vertices from {1, 2, …, j-1}.   Thus, we can write a recurrence by considering two cases: either the longest path using any of {1,…, j} doesn't contain vertex j, or it does:

   f(j) = max(f(j-1),   max_{ i, such that (i,j) is an edge} {f(i) + c(vi, vj) } }.

   Note that this is essentially the solution to the weighted interval scheduling problem that was presented in class.