CSE 347 Analysis of Algorithms January 23, 2019

Lecture 3 Summary

Instructor: Brendan Juba

1. Recall: weighted interval scheduling. Find a subset of requests (time intervals) that do not overlap, of largest total value.

2. Dynamic programming, step 1: recursive algorithm with few subproblems

   (a) Either an optimal schedule uses the last interval or it doesn't. If it does, then we toss out all intervals that conflict with it. Either way, we fill in the rest of the schedule using a solution to the subproblem given by the remaining intervals.

   (b) If the intervals were sorted by finish times, then in the second case we are tossing out a *suffix* of the list of intervals. Therefore, there are at most n distinct subproblems being solved (many, many times over).

3. Dynamic programming step 2: store solutions to subproblems

   (a) We pass a table across recursive calls storing the solutions to the corresponding subproblems, and *only make the recursive call if the corresponding entry is empty.* Therefore *every call is invoked at most once.*

   (b) Since the body of the recursive algorithm is easily implemented in $O(n)$ time, this gives an $O(n^2)$ algorithm.

   (c) Correctness follows by induction on the size of the subproblems. The tricky part: show that we can construct an optimal solution by taking an optimal solution to the subproblem and adding the last interval to it (or not).

4. Recursion is not needed – we can iteratively fill in the table

   (a) We only need to solve the subproblems in the order of increasing size: use a loop, not recursion

   (b) Actually, we can get down to $O(n \log n)$ time this way: first fill in a table consisting of just the optimal values for the subproblems, then walk "backwards," taking the ith request only if $M[i-1] < M[i]$. (Take it iff we need it to be optimal.)

5. New problem: scheduling all intervals

   (a) Now we are given a set of requests and we are asked to schedule all of them. The question is, how many resources do we need so that each can have a conflict-free schedule?

   (b) Formally: partition the requests into as few conflict-free subsets as possible.

   (c) We observe that if there are k requests that all run at the same time, we certainly need at least k distinct resources...