Homework 4

Walker Bagley

March 21, 2024

1 Exercise 15.1-2

By choosing the last activity to start in each step of the activities subproblems, we maintain a greedy algorithm. This approach works the exact same as the original solution to the activities problem, but building the solution up backwards. By picking the activity with the last start time (greedy), the algorithm does not care about the rest of the subproblem. If we find the end time t of the entire series of activities, then run the original algorithm on a new list of activities all with start and end times subtracted from t, then we will get the exact same solution once we account for the t offset.

We can prove that this solution yields an optimal solution by considering a subproblem S_k. Let A_k be the set of solutions to S_k and a_m be the activity with the last start time in S_k. Then all activities in A_k must not overlap. Lets consider the last activity a_i in A_k. Then if $a_i = a_m$, the solution contains the subproblem activity with the last start time. Otherwise, we replace a_i with a_m since a_m starts later than a_i and no activities in A_k overlap. Then we are done since A_k includes a_m.

2 Exercise 15.1-4

Create two arrays use and free which keep track of the lecture halls L_i currently in use and previously used but currently available. Then, sort the activities by start time and iterate through them. On each iteration, remove any lecture halls from use that have concluded their events and append them to free. Then, if free is empty, use a new lecture hall and append it to free. Otherwise, pop from the free list and append that lecture hall to the use array. After all activities that have been scheduled, the algorithm is finished and the combined length of use and free gives the fewest number of lecture halls.

3 Exercise 15.2-6

First calculate the value per pound of each item, that is $V_i = \frac{v_i}{w_i}$. Since sorting the items takes at best $O(n \lg n)$ time, we must improve it. We take the mean value per pound of the items and split them into two arrays, one with items with greater than the mean and one with the other lower value items, which can be done in linear time. If we can fit all the items in the high array into the knapsack, we take all of them and recurse on the low array with an adjusted weight for the remaining weight in the knapsack. If we cannot fit all the items in high, then we recurse on the high array until we can. Since each recursive step can be done in linear time which reduces each step, this algorithm completes in $O(n)$ time.