Fundamental Algorithm Techniques

Problem Set #4

Yestay Aikyn

Problem 1

For conflict-free course allocation as in the course:

Consider set of activities:

$$S = \{a_1, a_2, \dots, a_n\},$$

each with starting and ending times s_i, f_i. Assume also that S is sorted such that:

$$f_1 \leq f_2 \leq \cdots \leq f_n$$

A set of compatible activities $\hat{S}_{ap}$ means that each activity of $\hat{S}_{ap}$ starts after the last activity ended and ends before the next activity starts.

Dynamic Approach:

Consider the recurrence below for the cost $c[a,p] = c[a,k] + c[k,p] + 1$ between the course a_a and a_p:

$$c[a,p] = \begin{cases} 0 & \text{if } \hat{S}_{ap} = \emptyset, \\ \max\{c[a,k] + c[k,p] + 1 \mid a_k \in \hat{S}_{ap}\} & \text{if } \hat{S}_{ap} \neq \emptyset, \end{cases}$$

1. **Structuring the problem with $\hat{S}_{ap}$ actually characterises optimal substructures? Can therefore dynamical computing can be used?**
   Yes, because the cost between any two activities depends only on the best chain of compatible activities in between. This shows optimal substructure — the best path from a to p goes through some k where both sub-paths are also optimal. So dynamic programming works here.

2. **Explain/draw how that can be used for dynamical programming, top down with recurrence: RecuSelect(s, f, k, n), starts (s, f, 0, n), return list a_i's.**
   We define `RecuSelect(s, f, i, n)` to return the best sequence from activity i to n. Base case: if $i > n$, return empty. Else, try every $j > i$ where $s_j \geq f_i$, take the one giving max value (1 + recurse(j, n)), and build the list backward.

3. **dynamical programming with tabulation.**
   Make a 2D table $dp[i][j]$ = max number of activities from i to j. Fill diagonally: for length $l = 1$ to n, for $i = 1$ to $n - l + 1$, $j = i + l - 1$, then

   $$dp[i][j] = \max\{1 + dp[k][j] \mid i \leq k \leq j, s_k \geq f_i\}$$

   or 0 if no such k. Also keep track of choices to reconstruct the list.

Greedy Approach:

I. **What is the greedy choice for the activity-selection problem?**
Pick the activity with the earliest finish time that starts after the last selected one. Repeat.

II. **Write the pseudocode for the greedy approach GreedySchedule(s, f, n).**

```
GreedySchedule(s, f, n):
    selected = []
    last_end = -infinity
    for i = 1 to n:
        if s[i] >= last_end:
            add a_i to selected
            last_end = f[i]
    return selected
```

III. **Prove that your GreedySchedule is optimal using (hard)** *Proof by induction:*
Let G be the greedy solution, O any optimal. Base: $n = 1$, both pick a_1. Assume true for $n - 1$. Let a_k be first in G, a_m first in O. Since $f_k \leq f_m$ (earliest finish), we can replace a_m with a_k in O and still be valid. Now both start with same or better, and rest is smaller instance — by induction, greedy gets at least as many.

1 2

[1] choose the activity in S with the earliest finish time and bottom up, like in the course...:-)
[2] See course VI.