ICS 311, Fall 2020, Problem Set 07, Topics 12 & 13

Firstname Lastname <uhumail@hawaii.edu> Section 1

Due by midnight Tuesday 10/27. This is a 40 point homework. The extra 10 points are extra credit opportunities (the denominator of 1000 points for the semester remains the same), but the last problem is more difficult: take it as far as you can.

#2 Constructing and extracting the solution for Longest Path (10 pts)

In the following you use the posted solution to `LongestPathValueMemoized` from class last week.

(a) (3 pts) Rewrite `LongestPathValueMemoized` to `LongestPathMemoized` that takes an additional parameter **`next[1..|V|]`**, and records the next vertex in the path from any given vertex u in `next[u]`. Assume that all entries of `next` are initialized to 1 by the caller.

(b) (3 pts) Once that is done, write a procedure that recovers (e.g., prints) the path from s to t by tracing through `next`.

(c) (2 pts) What is the asymptotic runtime of your total solution to the Longest Path problem in terms of |V| and |E|? Include all steps (initializing arrays, `LongestPathMemoized`, and printing the solution). *Hint:* Count in aggregate across all calls rather than trying to figure out how many times the loop runs on a given Adj[u].

(d) (2 pts) What is the asymptotic use of space of your solution in terms of |V| and |E|?

#2 LCS by Suffix (15 pts)

In this problem you will redo the derivation of the LCS dynamic programming algorithm to be a variation that works on the suffixes rather than the prefixes. The intention is that revising a published derivation will help you see the approach before we ask you do one "from scratch" on your own.

In Topics 12 Lecture Notes Dynamic Programming - http://www2.hawaii.edu/~suthers/courses/ics311f20/Notes/Topic-12.html#LCS, the Longest Common Subsequence (LCS) problem is analyzed based on the notation:

Xi = prefix ⟨$x1$, ... , xi⟩

Yi = prefix ⟨$y1$, ... , yi⟩

But the optimal LCS substructure(s) can also be suffixes rather than prefixes. (Informally: It doesn't matter whether we start looking for the substructures from the beginning of the sequences X and Y of from their end.) In the suffix approach the arrows go in a more natural direction.

Below we step you through the derivation of a symmetric version of LCS based on suffixes:

Xi = suffix ⟨xi, ... , xm⟩

Yj = suffix ⟨yj, ... , yn⟩

(a) (3 pts) Reformulate ***Theorem 15.1*** for the suffix version by filling in the "___" below:

Let $Z = \langle z_1, ..., z_k \rangle$ be any LCS of $X = \langle x_1, ..., x_m \rangle$ and $Y = \langle y_1, ..., y_n \rangle$. Then

1. __________
2. (If the first characters of X and Y match, then these first characters are also the first character of the LCS Z, so we can discard the first character of all three and continue recursively on the suffix.)
3. __________
4. __________
5. (If the first characters of X and Y don't match each other, then the suffix Z must be in the substrings not involving these characters, and furthermore we can use the first character of Z to determine which one it lies in.)

(To keep this problem set from getting too long we will skip the proof, but it is an easy translation of the proof on page 392 of CLRS.)

(b) (5 pts) Now redefine the **Recursive formulation** accordingly (again, fill in the "___"):

Define $c[i, j]$ = length of LCS of X_i and Y_j . We want to find c[___, ___].

```
         { 0                  if i =___ or j = ___
c[i,j] = { c[___ , ___]       if i ___ , j ___ and x_i =y_j
         {max( ___ )          if i ___ , j ___ and x_i ≠y_j
```

Note that the original formulation started with $c[0,0]$ even though there is no x_0 or y_0. *Hint:* You do not need $c[0,0]$, but your c[] does need indices that go beyond the range of the indices in X and Y.

(c) (5 pts) Write pseudocode for LCS-LENGTH according to your **Recursive formulation**. (*Hint*: the arrows need to be different.)

(d) (2 pts) In the Notes, the longest subsequence could be only ‘printed’ with PRINT-LCS and the pseudocode needed recursion. With your ‘suffix’ method, you can do better and simpler: Write LCS(b,X,m,n) pseudocode that returns the subsequence directly. Use a vector to store the result. A vector is like an array but grows as needed.

#3 Activity Scheduling with Revenue (15 pts)

Activity scheduling problem from the greedy algorithm class: Suppose that different activities earn different amounts of revenue. In addition to their start and finish times s_i and f_i, each activity a_i has revenue r_i, and our objective is now to **maximize the total revenue:**

$$\sum_{a_i \in A} r_i$$

In class you found out that we can't use a greedy algorithm to maximize the revenue from activities, and we noted that dynamic programming will apply. Here you will develop the DP solution following the same steps as for the other problems (e.g., problem 3 above), but you are responsible for the details. Your analysis in (a) and (b) below should mirror that of section 16.1 of CLRS.

(a) (3 pts) Describe the structure of an optimal solution A_{ij} for S_{ij}, as defined in CLRS, and use a cut and paste argument to show that the problem has optimal substructure.

(b) (3 pts) Write a recursive definition of the value val[i,j] of the optimal solution for S_{ij}.

val[i,j] =

(c) (6 pts) Translate that definition into pseudocode that computes the optimal solution. *Hints:* Create fictitious activities a_0 with $f_0 = 0$ and a_{n+1} with $s_{n+1} = \infty$. Define tables val[0..n+1,0..n+1] for the values and activity[0..n+1,0..n+1], where activity[i,j] is the activity *k* that is chosen for A_{ij}. Use a bottom-up approach, filling in the tables for smallest problems first and then by increasing difference of j-i.

(d) (2 pts) Write pseudocode to print out the set of activities chosen.

(e) (1 pt) What is the asymptotic runtime of your solution including (c) and (d)?