Homework 5: Solution outlines by your TA

Problem 1 [Ex 15-2] Printing Neatly

Input:

n words whose lengths are $l_1, l_2, \ldots, l_n$.
Maximum length of the line, M.

The problem is the place the words in the same seqence so that we can *print neatly* which is determined by the left over space on each line (except the last one).

Finding the Optimal Sub-structure:

Say O is the cost of optimal arrangement of the words. In this optimal arrangement, say l_i ends a line. Let the cost of printing words from $l_1, \ldots, l_i$ be C. We claim that the C must be optimal. If there is a way of printing $l_1, \ldots, l_i$ with cost $C' < C$, then O is not optimal. Contradiction. Therefore C must be optimal.

Recurrence Relation

We define $LineCost(i,j)$ to be the cost of printing the line from word-i to word-j.

$$\begin{aligned} & = 0 && \text{if j = n (it's the last line) and the words fit in one lin} \\ LineCost(i,j) & = (M - j + i - \textstyle\sum_{k=i}^{j} l_k)^3 && \text{if the words fit in one line} \\ & = \infty && \text{otherwise} \end{aligned}$$

We define $C(j)$ to be the cost of printing all words upto l_j such that l_j ends a line in an optimal manner.
$C(0) = 0$
$C(j) = \min_{1 \le i \le j}\{C(i-1) + LineCost(i,j)$

The optimal cost of printing all the words is given by $C(n)$. which is computed from the above recurrence using memoization.

Pseudocode

```
1. C(0) = 0;
2. for j = 1 ... n
   (a) $C(j) = \min_{1 \le i \le j}\{C(i-1) + LineCost(i,j)$
       (* say this is minimum at $i = k$ *)
   (b) $L(j) = k$
   end for
   (* At this point $C(n)$ contains the optimal cost *)
3. printing the lines
   $L(n)$ consists of the 1st word on the last line.
   i.e. say $k = L(n)$ then words $l_k \ldots l_n$ goto the last line.
```

$L(k)$ consists of the 1st word on the 2nd–last line. Thus we can traverse L to find out how to print the words.

Space & Time complexities:

We maintain two lists C and L each of size $n+1$. Therefore space complexity is $O(n)$.

Running time = (Time to fill up C & L) + (time to traverse L and print).
Time to traverse L is $O(n)$. To fill up $C(j)$ we make j comparisions in the j^{th} iteration of the for–loop. Therefore the for–loop takes $O(n^2)$ time. We can also argue that there cann't be more than $M/2$ words on a single line, therefore we need to make atmost $O(M)$ comparisions in each iteration. Then the running time would be $O(nM)$.

Problem 2 [Ex 15-3] Edit Distance

Input:

Source string : $x[1 \dots m]$ and Target string : $y[1 \dots n]$.
Set of legal operations each with a cost.

Optimal Substructure:

Let O be the optimal sequence of operations to transform $x[1 \dots m]$ to $y[1 \dots n]$. Consider the subproblem of transforming $x[1 \dots i]$ to $y[1 \dots j]$. Suppose the S is the sequence of operations from O that tranfsorms $x[1 \dots i]$ to $y[1 \dots j]$ where $1 \leq i \leq m$ and $1 \leq j \leq m$. We claim that S is indeed optimal.

Recurrence: Let $D(i,j)$ be the edit distance between $x[1 \dots i]$ and $y[1 \dots j]$.

$$D(0,0) = 0$$

$$D(i,j) = \min \begin{cases} D(i-1,j-1) + cost(copy) & if x_i = y_j \\ D(i-1,j-1) + cost(replace) & if x_i \neq y_j \\ D(i-1,j) + cost(delete) & \\ D(i,j-1) + cost(insert) & \\ D(i-2,j-2) + cost(twiddle) & if [x_{i-1}x_i] = [y_i y_{i-1}] \end{cases}$$

$$D(i,n) = cost(kill) \text{ if } i < m$$

The edit distance between the two strings is given by $D(m,n)$. D is filled using already computed cells of D. In addition we will also maintain a table to keep track of what operations need to be performed.

Running times and space:

We need to fill the matrix D which is a $(m+1)x(n+1)$ matrix. We will also have an array of the same size to store the operations to be performed. Therefore the space required is $O(mn)$ For filling each cell we make a constant number of comparisions. Therefore the running time is $O(mn)$.

Problem 3 [Ex 15-7] Scheduling to Maximize Profit

Input:

n jobs $\{a_1, \ldots, a_n\}$.
Job a_i takes t_i time and has a deadline d_i before which if it is completed will fetch profit p_i.

Optimal Substructure:

First a few observations: We do not care for jobs that are scheduled to finish after their deadlines since they do not fetch any profit. There is no need to have any idle time.

We can consider the jobs in the order of their deadlines. i.e. We can have an optimal schedule of profitable jobs in the order of their deadlines. Prrof: Say we have an optimal schedule O of the profitable jobs in which there are two jobs a_i and a_j with start times s_i and s_j respectively and we have, $d_i > d_j$ but a_i scheduled before a_j, i.e. $s_i < s_j < d_j < d_i$. All jobs a_k in O scheduled after a_i and before a_j should have their deadlines before the start time of a_j. i.e. $d_k < s_j$. Since switching all the jobs a_k to before a_i will not violate any deadlines, we can do that. Further, we can also switch a_i and a_j and not violate any deadlines and still be able to finish all the jobs in the same time, therefore we can do that. Hence our claim that we can consider the jobs in their order of deadlines is correct.

Recurrence:

We consider jobs in the increasing order of their deadlines. Therefore, hence forth we assume that $d_i \leq d_j$ for $i \leq j$.

We define $P(i, T)$ to be the profit earned after considering jobs $a_1 \ldots a_i$ in the time interval $(0, T]$.

$$P(i,T) = \min \{ \begin{array}{ll} \text{P(i-1,T)} & if \text{a}_i \text{ is ignored} \\ \text{P(i-1, T-t}_i) + p_i & if \text{T} \leq d_i \end{array}$$

We also have $P(0, T) = 0$. i.e. the profit that we can get without considering any jobs is 0. And $P(i, 0) = 0$.

We need to consider times only upto $D = \max(d_i)$ because after this time we can not earn any profit. The result is available in P(n,D).

Space : The size of the memoization table is $n + 1\text{x}D + 1$.
We fill the table row wise. Filling each cell takes constant number of comparisions. Therefore the running time $= O(n \lg n) + O(nD)$.