Algorithm Design Assignment 3

Jianan Lin, 662024667, linj21@rpi.edu

Problem 6.5. There are n chars in a string $y = y_1...y_n$ without space. Our goal is to find the right sentence from this string. Here, for any consecutive substring $x = y_i...y_j$, we know its quality. We should find the largest sum of quality for our result, in which we divide the string into different words. Give an efficient algorithm that takes a string y and computes a segmentation of maximum total quality.

Solution.

First we will give the algorithm and then explain and analyze it. Here n is the number of chars and $Quality(a, b)$ is the quality of word from position a to b (including a and b) if $1 \leq a \leq b \leq n$ else 0. Notice an array starts from index 0 and ends at index $len(array) - 1$.

Algorithm 1 DP1

```
OPT = [−∞ × (n + 1)]
OPT[0] = 0
for i ∈ [1, n] do
    for j ∈ [0, i − 1] do
        OPT[i] = max( OPT[i],   OPT[j] + Quality(j + 1, i) )
    end for
end for
return OPT[n];
```

1. Subproblem: Here, OPT_i means the largest total quality in the substring $x = y_1...y_i$. If $i = 0$, then we let it be 0 since this is an empty string. If $i = 1$, then $OPT_i = Quality(1, 1)$ because we have only one choice and only when $i \geq 2$ can we have multiple choices and select the best one. When we get OPT_n, we can return this value because this is the global optimal value.

2. Recurrence: For example, we assume we have found the optimal case for substring $x = x_1...x_k$. Then we consider $x' = x_1...x_{k+1}$. For this new sentence, the last word must end with x_{k+1} and start with a char in $x_1...x_{k+1}$. For these $k + 1$ cases, we compare each case and select the one choice with largest total quality. Therefore we prove the correctness and have the following formula:

$$OPT_i = \max_{j<i} \Big(OPT_j + Quality(j + 1, i) \Big)$$

3. Complexity: Obviously the time complexity is $O(n^2)$ and the space complexity is $O(n)$.

Notice: If we require the algorithm should output the sentence with largest quality, then we have to modify the algorithm as follows. And this time the time complexity is also $O(n^2)$ but the space complexity is $O(n^2)$. (Here, "\t" means a space and $y[a : b]$ means a word starting from position a and ending in position b)

Algorithm 2 DP1-Modify

```
OPT= [−∞ × (n + 1)],   Sentences = [" " ×(n + 1)],   k = 0
OPT[0] = 0
for i ∈ [1, n] do
    for j ∈ [0, i − 1] do
```

```
    if OPT[i] < OPT[j] + Quality(j + 1, i) then
      k = j,   OPT[i] = OPT[j] + Quality(j + 1, i)
    end if
  end for
  Sentences[i] = Sentences[k] + "\t" + y[j + 1 : i]
end for
return Sentences[n] and OPT[n]
```

Problem 6.28. There are n tasks to be scheduled and each task has a deadline d_i and a processing time t_i. All the tasks should be scheduled after a starting time s. Our goal is to find a largest subset of tasks so that all the tasks in the subset can be finished before their deadlines in some scheduled.

(a). Prove that, there is an optimal solution with a largest subset J so that all the tasks in J are scheduled with increasing order of deadlines.

(b). Give an algorithm to find the optimal solution in polynomial time with variables n and $D = \max_i\{d_i\}$.

Solution.

(a). First we prove a called Lemma 1. Suppose A is a set of tasks, and we can scheduled the tasks in A with a time table S, so that all the tasks can be finished before their deadlines.

Lemma 1. If there are two consecutive tasks $i, i+1$ in the time table S with $d_i > d_{i+1}$, then exchanging their position still guarantee all the tasks being finished before their deadlines.

Proof. Since t_i is fixed for any i, we know that $t_i + t_{i+1}$ is a constant. Therefore all the finishing time f_k for $k \neq i, i+1$ do not change no matter $k < i$ or $k > i+1$.

Then we consider these two tasks. Use T_i to denote the sum of $t_1, ...t_i$, then obviously we have $T_i \leq d_i$ and $T_{i+1} \leq d_{i+1}$.

Therefore we have $d_i > d_{i+1} \geq T_{i+1}$, and $d_{i+1} \geq T_{i+1} \geq T_{i-1} + t_{i+1}$. This means that if we change the order of i and $i+1$, these two tasks will also be finished before their deadlines. So lemma 1 holds.

With lemma 1, we can use bubble sort to exchange the consecutive tasks $i, i+1$ in J with $d_i > d_{i+1}$ repeatedly and finally get a time table in increasing order of deadlines. In the same time, all the tasks in J can also be finished before their deadlines.

In this way, we prove the original proposition.

(b). First we give the algorithm and then explain and analyze it. Notice an array starts from index 0 and ends at index $len(array) - 1$.

Algorithm 3 DP2

```
Sort the tasks by increasing order of deadlines and rename them from 1 to n
OPT= [ [ 0 × (D + 1) ] × (n + 1) ],   S = [ [ [ ] × (D + 1) ] × (n + 1) ]
for i ∈ [0, n] do
  for j ∈ [1, D] do
    if j ≤ d_i and j ≥ t_i and OPT[i − 1][j] < OPT[i − 1][j − t_i] + 1 then
      OPT[i][j] = OPT[i − 1][j − t_i] + 1,   S[i][j] = S[i − 1][j − t_i] + [i]
    else
      OPT[i][j] = OPT[i − 1][j − t_i],   S[i][j] = S[i − 1][j − t_i]
    end if
  end for
end for
```

12: **return** S[n] and OPT[n]

1. Subproblem: Here, OPT[i][j] means the largest size of subset for tasks $1, ..., i$ and time j so that all the tasks in this subset can be finished before j. Obviously, when $i = 0$, the size of the subset is 0. Also, S[i][j] means the largest subset with size OPT[i][j]. When $i = 0$, $S = \emptyset$.

2. Recurrence: We consider when we can add task i to the subset S[i][j]. First, we should have $j \leq d_i$, i.e. the deadline should not be earlier than j. Then, we should have $j \geq t_i$, i.e. we should have enough time to finish this single task.

Last, we should have OPT[$i-1$][j] < OPT[$i-1$][$j-t_i$] + 1, i.e. we will have a larger subset if task i is included in it. Here we will explain it. If we add task i, then the second-to-last task should be finished before $j - t_i$. Then we will have the size of the subset be 1 + OPT[$i-1$][$j-t_i$]. If we do not add task i, then we only need to copy OPT[$i-1$][j] and S[$i-1$][j]. So we should choose the larger one of these two. In this way we prove the correctness.

3. Complexity: Obviously the time complexity is $O(Dn^2)$ and the space complexity is $O(Dn^2)$. Notice that since the size of a subset can reach n, the operation to copy a subset is $O(n)$.