

Homework 1

Lin Zejin

May 10, 2025

-
- **Collaborators:** I finish this homework by myself.
-

Problem 1. If d_j is the sequence $-1, 1, 2, 3, 4, 4$, then all of this are legal. However, first if we choose $d_1 = -1$, now none of $d_2 \sim d_5$ is available, which is what the algorithm will find, *i.e.* $S = \{1, 6\}$. However, $S = \{2, 3, 4, 5\}$ is a larger viewable set. So the algorithm is not correct.

Denote $\text{OPT}[j]$ as the size of optimal solution among d_j, d_{j+1}, \dots, d_n . Then the Bellman equation will be

$$\text{OPT}[j] = \min_{t \text{ reachable from } j} \{\text{OPT}[j+1], \text{OPT}[t] + 1\}$$

where $\text{OPT}[n] = 1$. The detail is below:

```
1: OPT[n] = 1.
2: for  $j$  from  $n-1$  to 1 do
3:    $\text{OPT}[j] = \min_{t \text{ reachable from } j} \{\text{OPT}[j+1], \text{OPT}[t] + 1\}$ .
4: end for
5: return OPT[1]
```

Problem 2. Denote $\text{OPT}[i]$ as the minimum number of rounds when the node i call all his subordinates (do not need direct subordinates)

If node i has direct subordinates j_1, \dots, j_k , WLOG we assume $\text{OPT}[j_1] \geq \text{OPT}[j_2] \geq \dots \geq \text{OPT}[j_k]$. Then easy to confirm that the Bellman equation is like

$$\text{OPT}[i] = \max_{1 \leq t \leq k} \{\text{OPT}[t] + t\}$$

So the algorithm will be The answer is $\text{OPT}(n)$ where n is the root node.

Algorithm 1 $\text{OPT}(i)$

```
1: compute  $\text{OPT}[j] = \text{OPT}(j)$  for each  $j$  direct subordinate of  $i$ 
2: sort  $\text{OPT}[j]$  decreasingly. Obtain  $\text{opt}_1, \dots, \text{opt}_k$ .
3:  $\text{OPT}[i] = \max_{1 \leq t \leq k} \{\text{OPT}[t] + t\}$ 
```

Problem 3. Denote $\text{OPT}[n][k]$ as the maximum possible return of k -shot strategy in $1 \sim n$ days.

The Bellman equation will be

$$\text{OPT}[n][k] = \min\{\text{OPT}[n-1][k], \min_{1 \leq j \leq n-1} \{\text{OPT}[j-1][k-1] + p[n] - p[j]\}\}$$

in which the first term is the case that we do not sell on the last day and the second term is the case that we sell on the last day.

So the algorithm will be

Algorithm 2 OPT

```

1: for  $j$  from 0 to  $n$ ,  $t$  from 0 to  $k$  do
2:   if  $j = 0$  or  $t = 0$  then
3:      $\text{OPT}[j, t] \leftarrow 0$ 
4:   end if
5:    $\text{OPT}[j, t] \leftarrow \min\{\text{OPT}[j-1, t], \min_{1 \leq l \leq n-1} \{\text{OPT}[l-1, t-1] + p[j] - p[l]\}\}$ 
6: end for

```

The answer is $1000 \times \text{OPT}(n, k)$ and the time complexity is $O(n^2k)$. The actual k -shot strategy can be given by tracing the choice of each state (n, k) .

Problem 4. Denote $\text{OPT}[n][H]$ as the maximum total grade, given the functions f_1, \dots, f_n and total hours H .

We can enumerate the hours spending on the last project to obtain a Bellman equation

$$\text{OPT}[n][H] = \max_{0 \leq j \leq H} \text{OPT}[n-1][H-j] + f_n(j)$$

Then there is an possible algorithm that has time complexity $O(nH)$.

Algorithm 3 OPT

```

1: for  $k$  from 0 to  $n$ ,  $t$  from 0 to  $H$  do
2:   if  $k = 0$  or  $t = 0$  then
3:      $\text{OPT}[k, t] \leftarrow 0$ 
4:   end if
5:    $\text{OPT}[k, t] \leftarrow 0$ 
6:   for  $l$  from 0 to  $t$  do
7:     if  $\text{OPT}[k, t] \leq \text{OPT}[k-1, t-j] + f_k(j)$  then
8:        $\text{OPT}[k, t] \leftarrow \text{OPT}[k-1, t-j] + f_k(j)$ 
9:        $\text{REC}[k, t] = j$ .
10:    end if
11:  end for
12: end for

```

Then $\frac{1}{n} \text{OPT}[n, H]$ is the maximum average grades and $\text{REC}[k, t]$ is the spending hours at project k at the state of (k, t) , so we can decide the hours that spend on each project by tracing it.

The time complexity is $O(H^2n)$

Problem 5. (a) Indeed, we can prove every schedule can be scheduled in increasing order of their deadlines.

If jobs $(s_i, s_i + t_i)$ is a proper schedule, and $s_1 < s_1 + t_1 \leq s_2 < s_2 + t_2 \leq \dots \leq s_k < s_k + t_k$. Assume $d_i > d_{i+1}$.

Then

$$s_i < s_i + t_i \leq s_{i+1} < s_{i+1} + t_{i+1} \leq d_{i+1} < d_i$$

Therefore

$$s_i < s_i + t_{i+1} = s_i + t_{i+1} \leq s_i + t_i + t_{i+1} < d_i$$

since $d_i > s_{i+1} + t_{i+1} \geq s_i + t_i + t_{i+1}$.

So we can exchange the order of job $i, i + 1$. By Bubble sort algorithm, we can obtain an available schedule that execute in increasing order of their deadlines.

- (b) We only need to consider the schedule that execute in increasing order of their deadlines. Denote $\text{OPT}[n][D]$ as the optimal size of jobs $1, 2, \dots, n$ and final dealines D .

WLOG, we may assume $d_1 \leq d_2 \leq \dots \leq d_n$. Each time we consider whether we choose job n or not. Since job n has the last deadlines in that state, we can arrange for it to be the last to execute. So the Bellman equation is

$$\text{OPT}[n][D] = \begin{cases} 0 & t_n > \min\{d_n, D\} \\ \max\{\text{OPT}[n-1][D], \text{OPT}[n-1][\min\{d_n, D\} - t_n] + 1\} & \text{otherwise} \end{cases}$$

So the algorithm is

Algorithm 4 OPT

```

1: for  $k$  from 0 to  $n$ ,  $t$  from 0 to  $D$  do
2:   if  $k = 0$  or  $t = 0$  or  $t_k > \min\{d_k, t\}$  then
3:      $\text{OPT}[k, t] \leftarrow 0$ 
4:   end if
5:    $\text{OPT}[k, t] \leftarrow \text{OPT}[k-1, t]$ .
6:    $\text{REC}[k, t] = 0$ .
7:   if  $\text{OPT}[k, t] < \text{OPT}[k-1][\min\{d_k, t\} - t_k] + 1$  then
8:      $\text{OPT}[k, t] \leftarrow \text{OPT}[k-1][\min\{d_k, t\} - t_k] + 1$ 
9:      $\text{REC}[k, t] = 1$ .
10:  end if
11: end for
  
```

Then $\text{OPT}[n, D]$ is the optimal solution and $\text{REC}[k, t]$ marks the choice in the state (k, t) which can be traced to form a available schedule.

Its time complexity is $O(nD)$

Problem 6. In the lecture we find its Bellman equation

$$\text{OPT}[i, j] = \min \begin{cases} \text{OPT}[i-1, j-1] + \alpha_{a_i b_j} \\ \text{OPT}[i-1, j] + \delta \\ \text{OPT}[i, j-1] + \delta \end{cases}$$

It involves all possible cases, so it suffices to mark all possible choices in the state (i, j) .

We define a function $mark[i, j]$ with

$$mark[i, j] = \begin{cases} 1 & \text{if } \begin{cases} \text{OPT}[i-1, j-1] + \alpha_{a_i b_j} \\ \text{OPT}[i-1, j] + \delta \\ \text{OPT}[i, j-1] + \delta \end{cases} \text{ has two possible choices of minimum} \\ 0 & \text{otherwise} \end{cases}$$

which can be traced in the process of DP.

So after DP in $O(nm)$, we obtain a possible minimum-cost alignment. Compared with the proposed alignment, we know whether it is optimal or not. And trace all the choices in this alignment and check whether $mark[i, j]$ is 1 or not in each state, we can find whether it is unique or not.

Problem 7. Denote $IsInter[k][i][j]$ as the possibility if the first k characters of s can be partitioned into two subsequences s_1 and s_2 so that s_1 is a repetition of x and s_2 is a repetition of y , moreover

$$\text{len}(s_1) \equiv i \pmod{\text{len}(x)}, \quad \text{len}(s_2) \equiv j \pmod{\text{len}(y)}$$

$$\text{For example, } IsInter[1][i][j] = \begin{cases} 1 & (i, j) = (1, j) \text{ and } s[1] = x[1] \\ 1 & (i, j) = (i, 1) \text{ and } s[1] = y[1] \\ 0 & \text{otherwise} \end{cases}$$

The Bellman equation is

$$IsInter[k][i][j] = \begin{cases} \max \{IsInter[k-1][i-1][j], IsInter[k-1][i][j-1]\} & \text{if } s[k] = x[i] \text{ and } s[k] = y[j] \\ IsInter[k-1][i-1][j] & \text{else if } s[k] = x[i] \\ IsInter[k-1][i][j-1] & \text{else if } s[k] = y[j] \\ 0 & \text{otherwise} \end{cases} \quad (7.1)$$

because the k -th character have two choice. In (7.1), $i-1, j-1$ is chosen to be the one in $\{1, 2, \dots, \text{len}(x)\}, \{1, 2, \dots, \text{len}(y)\}$ under the module meaning respectively.

So the algorithm can be implemented as follows The answer is $\max_{i,j} IsInter[n][i][j]$.

Since we can delete all data after two iterations, the space complexity is $O(l_x l_y)$, and the time complexity is $O(n l_x l_y)$ which is a polynomial algorithm.

Problem 8. WLOG we can normalize r_i such that $r_{min} = 1$.

(a) Noticed that

$$R(S) = \frac{\sum_{i \in S} r_i v_i}{1 + \sum_{i \in S} v_i} = \frac{\sum_{i \in S} (r_i \cdot \frac{v_i}{v_i + \frac{1}{|S|}})(v_i + \frac{1}{|S|})}{\sum_{i \in S} v_i + \frac{1}{|S|}} \in \left(\min_{i \in S, |S| \leq K} \left\{ r_i \cdot \frac{v_i}{v_i + \frac{1}{|S|}} \right\}, \max_{i \in S, |S| \leq K} \left\{ r_i \cdot \frac{v_i}{v_i + \frac{1}{|S|}} \right\} \right) \triangleq (a, b)$$

```

1:  $l_x = \text{len}(x), l_y = \text{len}(y), n = \text{len}(s)$ .
2:  $\text{IsInter}[0][i][j] \leftarrow \begin{cases} 1 & (i, j) = (l_x, l_y) \\ 0 & \text{otherwise} \end{cases}$ 
3: for  $k$  from 2 to  $n$ ,  $i$  from 1 to  $l_x$ ,  $j$  from 1 to  $l_y$  do
4:   if  $i = 1$  then
5:      $i' = l_x$ 
6:   else
7:      $i' = i - 1$ 
8:   end if
9:   if  $j = 1$  then
10:     $j' = l_y$ 
11:  else
12:     $j' = j - 1$ 
13:  end if
14:   $\text{IsInter}[k][i][j] \leftarrow 0$ 
15:  if  $s[k] = x[i]$  and  $\text{IsInter}[k-1][i'][j] = 1$  then
16:     $\text{IsInter}[k][i][j] \leftarrow 1$ .
17:  end if
18:  if  $s[k] = y[j]$  and  $\text{IsInter}[k-1][i][j'] = 1$  then
19:     $\text{IsInter}[k][i][j] \leftarrow 1$ .
20:  end if
21: end for

```

with $0 < a < b \leq r_{max} = \frac{r_{max}}{r_{min}}$.

It suffices to find the maximal possible value of $R(S)$ in (a, b) .

For each $r \in (a, b)$,

$$\begin{aligned}
\exists S, |S| \leq K, R(S) \geq r &\Leftrightarrow \sum_{i \in S} (r_i - r)v_i \geq r \\
&\Leftrightarrow \exists (r_{i_1} - r)v_{i_2} \geq (r_{i_2} - r)v_{i_2} \geq \cdots \geq (r_{i_t} - r)v_{i_t} \geq 0 \text{ such that } \sum_{k=1}^t (r_{i_k} - r)v_{i_k} \geq r \text{ and } t \leq K
\end{aligned} \tag{8.1}$$

The last circumstance only needs to sort $(r_i - r)v_i$ and calculate the first K -th sum, (or the sum of all positive value $(r_i - r)v_i$) which costs time complexity $O(N \log N)$. The function is

Algorithm 5 IsLargeThanOPT(r)

```

sort  $(r_i - r)v_i$ .
sum  $\leftarrow 0, S \leftarrow \emptyset$ 
for  $i$  from 1 to  $K$  do
  if  $(r_i - r)v_i > 0$  then
    sum  $\leftarrow$  sum +  $(r_i - r)v_i$ .
     $S \leftarrow S \cup \{i\}$ 
  end if
end for
if sum  $\geq r$  then
  return FALSE and the set  $S$ 
else
  return TRUE
end if

```

To determine $R(S^*)$, we can use bisection method:

Algorithm 6 General algorithm

```

1: Determines  $a_0 \leftarrow a, b_0 \leftarrow b, i \leftarrow 1$ .
2: while do
3:    $r_i \leftarrow \frac{a_{i-1} + b_{i-1}}{2}$ 
4:   if  $\text{IsLargeThanOPT}(r) = \text{FALSE}$  then
5:     RETURN a possible  $S$  with  $|S| \leq K$  such that  $R(S) \geq r$ .
6:      $a_i \leftarrow r_i, b_i \leftarrow b_{i-1}$ .
7:   else if  $r_i \geq R(S^*)$  then
8:      $a_i \leftarrow a_{i-1}, b_i \leftarrow r_i$ .
9:   end if
10: end while

```

After t times of WHILE iterations 2, we can find a r such that $r - R(S^*) \leq \frac{b-a}{2^t}$.

So after iterations in $O(\log \frac{b-a}{\epsilon})$, we can find a $r_i \in (a_i, b_i)$ within $(1 \pm \epsilon)$ -factor of $R(S^*)$ (as $\epsilon \rightarrow 0$)

Then $a_i \leq S(R^*)$ satisfies that $\exists S, |S| \leq K, R(S) \geq (1 - \epsilon)R(S^*)$, given by the process of Algorithm 5

Since each iteration cost $O(N \log N)$, the total time complexity will be

$$O(N \log N) \cdot O(\log \frac{b-a}{\epsilon}) = O(N \log N \log \frac{r_{max}}{r_{min}} + N \log N \log \epsilon^{-1})$$

(b,c) We still use algorithm 6 to find r . It suffices to determine the function $\text{IsLargeThanOPT}(r)$.

It is equivalent to solve (8.1), which is a knapsack problem.

(b) However, under the condition of (b), we can use a Weight-Based DP such that the time complexity of $\text{IsLargeThanOPT}(r)$ is $O(nW)$. Then the total time complexity will be

$$O(NW) \cdot O(\log \frac{b-a}{\epsilon}) = O(NW \log \frac{r_{max}}{r_{min}} + NW \log \epsilon^{-1}).$$

(c) In the lecture we find an algorithm in time complexity of $O(N^3 \epsilon^{-1})$ to find the solution S such that $R(S) \geq (1 - \frac{\epsilon}{2})R(S^*)$. So in $O(N^3 \epsilon^{-1})$, we can define a function $\text{IsLargeThanOPT}(r)$ to determine whether $r \geq (1 - \frac{\epsilon}{2})R(S^*)$ or not.

After iterations in $O(\log \frac{b-a}{\epsilon})$, we can find a $r_i \in (a_i, b_i)$ within $(1 \pm \frac{\frac{\epsilon}{2}}{1 - \frac{\epsilon}{2}})$ -factor of $(1 - \frac{\epsilon}{2})R(S^*)$ (as $\epsilon \rightarrow 0$)

Then $a_i \leq S(R^*)$ satisfies that $\exists S, |S| \leq K, R(S) \geq (1 - \epsilon)R(S^*)$, given by the process of the knapsack valued-based FPTAS

The time complexity will be

$$O(N^3 \epsilon^{-1}) \cdot O(\log \frac{b-a}{\epsilon}) = O(N^3 \epsilon^{-1} \log \frac{r_{max}}{r_{min}} + N^3 \epsilon^{-1} \log \epsilon^{-1})$$

which is a FPTAS.