CS-GY 6033: Homework #6

Due on December 6, 2023

Professor Yi-Jen Chiang

Runze Li
Nxxxxxxxx
rl50xx@nyu.edu

Tzu-Yi Chang
Nxxxxxxxx
tc39xx@nyu.edu

Jiayi Li
Nxxxxxxxx
jl156xx@nyu.edu

December 5, 2023

1. ① Sort the jobs by increasing deadline. That is, the new set of n jobs

$O(n\log n)$ $a_1, a_2, a_3, \ldots, a_n$ are ordered by deadline: $d_1 \le d_2 \le d_3 \le \cdots \le d_n$.

② Creat a 2D array dp, where dp[i][j] represents the maximum profit we can make from jobs $1, \ldots, i$ in time j.

③ Recursively compute dp[i][j] for i from 0 to n and j from 0 to T.

$O(nT)$

$$dp[i][j] = \begin{cases} 0, & \text{if } i=0 \text{ or } j=0 \\ \max(dp[i-1][j],\ dp[i-1][j-t_i]+p_i), & \text{if } j-t_i \ge 0 \text{ and } j \le d_i \\ dp[i-1][j], & \text{else} \end{cases}$$

④ Return dp[n][T], where dp[n][T] represents the maximum profit we can get from $1, \ldots, n$ jobs in time T.

Since the recursive computation of the 2D array costs $O(nT)$ time, the algorithm runs in $O(nT)$ worst-case time.

2. ① Compute $|X_i|$ (the length of X_i) for each $i=1,2,\cdots,m+1$. With the provided location of m cuts (M_i for $i=1,2,\cdots,m$),

$O(m)$ $$|X_i| = \begin{cases} M_i - M_{i-1} & \text{, if } i>1 \\ M_1 & \text{, if } i=1 \\ n-M_m & \text{, if } i=m+1 \end{cases}$$

For example, $M=[3,10]$ and $n=20$, we get $|X|=[3,7,10]$

② Create a 2D array dp, where dp[i][j] represents the minimum cost of breaking the substring $X_i X_{i+1} \cdots X_{j-1} X_j$ into pieces.

③ Define the relation $$dp[i][j] = \begin{cases} M_i & \text{, if } i=j \\ \min(dp[i][k]+dp[k+1][j]) & \text{where } k=i,i+1,\cdots,j-1, \text{ else} \end{cases}$$

④ Our goal is to determine dp[1][m+1], which represents the minimum cost of breaking the string X into m+1 pieces $X_1, X_2, \cdots, X_{m+1}$. Recursively perform the algorithm to compute dp[i][j]. For example,

$$dp[1][m+1] = \min(dp[1][1]+dp[2][m+1], dp[1][2]+dp[3][m+1], \cdots, dp[1][m]+dp[m+1][m+1])$$

The bottom-up algorithm may be as follow:

```
for i from 1 to m+1:
    dp[i][i] = |X_i|
for len from 1 to m:                          O(m^3)
    for i from 1 to m-len+1:
        j = i+len    (j ≤ m+1)
        dp[i][j] = +∞
```

```
for k from i to j-1:
    dp[i][j] = min(dp[i][j], dp[i][k] + dp[k+1][j])
```

Since there are at most m^2 possible costs to determine, and m times of comparison is performed to find the minimum value, this algorithm runs in $O(m^3)$ worst-case time.

3. ① Starting from vertex s, we perform a DFS on G and get the finish time of some vertices, forming the topological sorting order. Note that not all vertices are reachable from s, so the topological order may exclude some vertices in V.

$O(V+E)$

② Initialize the two entries, $d(v)$ and $prev(v)$ as below:

$O(V)$

$$d(v) = \begin{cases} 0, & \text{if } v = s \\ \text{undefined}, & \text{else} \end{cases} \qquad prev(v) = \begin{cases} \text{NULL}, & \text{if } v = s \\ \text{undefined}, & \text{else} \end{cases}$$

③ Iterate through the vertices in topological order. For each vertex u, update all vertices v that has an incoming edge from u

$O(V+E)$

where $d(v) = \begin{cases} d(u) + w(e) \text{ where } e = (u,v), & \text{if } d(v) = \text{undefined} \\ \max(d(v), d(u) + w(e)) \text{ where } e = (u,v), & \text{else} \end{cases}$

$$prev(v) = \begin{cases} u, & \text{if } prev(v) = \text{undefined or } d(u) + w(e) > d(v) \\ prev(v), & \text{else} \end{cases}$$

As DFS and the iteration may go through each vertex in V and each edge in E, the worst-case running time is $O(V+E)$.

4. (a) Consider all possible relationships of the two intervals $[s_1, s_2]$ and $[h_1, h_2]$ (assuming that $s_1 \le h_1$), and prove that $|s_1-h_1|+|s_2-h_2| \le |s_1-h_2|+|s_2-h_1|$.

① $s_1 \le h_1 \le s_2 \le h_2$:

$|s_1-h_1|+|s_2-h_2| \le |s_1-h_2|+|s_2-h_1|$

$\Rightarrow h_1 - \cancel{s_1} + \cancel{h_2} - s_2 \le \cancel{h_2} - \cancel{s_1} + s_2 - h_1$

$\Rightarrow h_1 \le s_2 \qquad \therefore$ True!

② $s_1 \le s_2 \le h_1 \le h_2$:

$|s_1-h_1|+|s_2-h_2| \le |s_1-h_2|+|s_2-h_1|$

$\Rightarrow \cancel{h_1} - \cancel{s_1} + \cancel{h_2} - \cancel{s_2} \le \cancel{h_2} - \cancel{s_1} + \cancel{h_1} - \cancel{s_2}$

$\Rightarrow 0 \le 0 \qquad \therefore$ True!

③ $s_1 \le h_1 \le h_2 \le s_2$:

$|s_1-h_1|+|s_2-h_2| \le |s_1-h_2|+|s_2-h_1|$

$\Rightarrow h_1 - \cancel{s_1} + \cancel{s_2} - h_2 \le h_2 - \cancel{s_1} + \cancel{s_2} - h_1$

$\Rightarrow h_1 \le h_2 \qquad \therefore$ True!

For all possible cases, we proved that there is no advantage to "cross match".

(b) ① Sort the m pairs of skis and n skiers respectively in increasing order of heights. Since $m=n$, the running time for each sorting costs $O(n\log n)$ worst-case time. $O(n\log n)$

② According to the property we proved in part (a), assigning skis to skiers in order is the best solution that we can achieve. This is because any cross matching always leads to a bigger difference of the heights, thus failing to minimize the resulting sum.

In this case where $m=n$, only the sorting on both skis and skiers matters, so the running time is $O(n\log n)$ worst-case time.

(C) ① Sort the m pairs of skis and n skiers respectively in increasing order of heights. $O(n\log n)$

② Create a 2D array dp, where $dp[i][j]$ represents the minimum sum of differences we can get with matching the first i pairs of skis and j skiers, note that every skier should get one pair of skis.

③ Compute dp recursively for i from 1 to m and j from 1 to n, where $O(mn)$

$$dp[i][j] = \begin{cases} dp[i-1][j-1] + |s_i - h_j| , & \text{if } i=j \\ \min(dp[i-1][j], dp[i-1][j-1] + |s_i - h_j|), & \text{if } i>j \\ +\infty , & \text{else } (i<j \Rightarrow \text{skis are not enough for skiers}) \end{cases}$$

④ Return $dp[m][n]$ as the minimum sum of the absolute differences of the heights for matching all skiers and their skis.

In this dynamic programming algorithm, the sorting may cost $O(n\log n)$ and $O(m\log m)$, while the computation of dp costs $O(mn)$. Since we assume that $m=\Theta(n)$, the overall algorithm runs in $O(mn)$ worst-case time.