CSE 431/531 HW4 Spring 2022

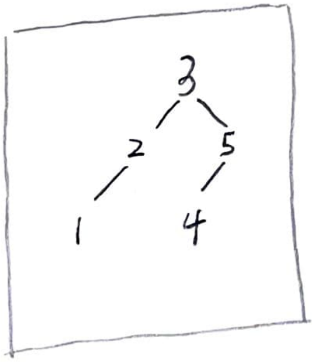
Feng Wei 50428109

Problem 1.

optimum binary search tree

opt[i, j] (i \ j)	1	2	3	4	5
1	5	35	65	95	170
2		25	55	85	155
3			15	35	90
4				10	50
5					30

π(i, j) (i \ j)	1	2	3	4	5
1	1	2	2	2	3
2		2	2	2/3	3
3			3	3	5
4				4	5
5					5



total cost $15 \times 1 + 25 \times 2 + 30 \times 2 + 5 \times 3 + 10 \times 3$

$= 15 + 50 + 60 + 15 + 30$

$= 170$

Please check Details Below :

$$opt[1,2] = \min\{0+opt[2,2],\ opt[1,1]+0\} + (f_1+f_2) = \min\{25,\ \underline{5}\} + 30 = 35 \qquad \pi(1,2) = 2$$

$$opt[2,3] = \min\{\underline{15},\ 25\} + 40 = 55 \qquad \pi(2,3) = 2$$

$$opt[3,4] = \min\{\underline{10},\ 15\} + 25 = 35 \qquad \pi(3,4) = 3$$

$$opt[4,5] = \min\{30,\ \underline{10}\} + 40 = 50 \qquad \pi(4,5) = 5$$

$$opt[1,3] = \min\{0+opt[2,3],\ opt[1,1]+opt[3,3],\ opt[1,2]+0\} + (f_1+f_2+f_3)$$

$$= \min\{55,\ \underline{5+15},\ 35\} + 45 = 65 \qquad \pi(1,3) = 2$$

$$opt[2,4] = \min\{0+opt[3,4],\ opt[2,2]+opt[4,4],\ opt[2,3]+0\} + (f_2+f_3+f_4)$$

$$= \min\{\underline{35,\ 25+10},\ 55\} + 50 = 85 \qquad \pi(2,4) = \underline{2 \text{ or } 3}$$

$$opt[3,5] = \min\{0+opt[4,5],\ opt[3,3]+opt[5,5],\ opt[3,4]+0\} + (f_3+f_4+f_5)$$

$$= \min\{50,\ 15+30,\ \underline{35}\} + 55 = 90 \qquad \pi(3,5) = 5$$

$$opt[1,4] = \min\{0+opt[2,4],\ opt[1,1]+opt[3,4],\ opt[1,2]+opt[4,4],\ opt[1,3]+0\} + (f_1+f_2+f_3+f_4)$$

$$= \min\{85,\ \underline{5+35},\ 35+10,\ 65\} + 55 = 95 \qquad \pi(1,4) = 2$$

$$opt[2,5] = \min\{0+opt[3,5],\ opt[2,2]+opt[4,5],\ opt[2,3]+opt[5,5],\ opt[2,4]+0\} + (f_2+f_3+f_4+f_5)$$

$$= \min\{90,\ \underline{25+50},\ 55+30,\ 85\} + 80 = 155 \qquad \pi(2,5) = 3$$

$$opt[1,5] = \min\{0+opt[2,5],\ opt[1,1]+opt[3,5],\ opt[1,2]+opt[4,5],\ opt[1,3]+opt[5,5],\ opt[1,4]+0\}$$

$$+ (f_1+f_2+f_3+f_4+f_5) \qquad\qquad = 170 \qquad \pi(1,5) = 3$$

$$= \min\{155,\ 5+90;\ \underline{35+50},\ 65+30,\ 95\} + 85$$

Problem 2. Weighted Interval Scheduling

Solution:

We first sort all the jobs in non-decreasing order of finishing times f_i

After that, we assume $f_1 < f_2 < \cdots f_n$. This step takes $O(n\log n)$ time (e.g. Heap Sort)

Define: P_j as the largest index $i < j$, so that job i and job j are disjoint $(s_i, f_i]$ and $(s_j, f_j) \Rightarrow f_i < s_j$

Compute P_j for job from $j=1$ to n, every P_j needs $O(\log n)$ time by using binary search. Total time $O(n\log n)$

① For every $j \in \{0,1,2 \cdots n\}$ and $i \in \{0,1,2,\cdots k\}$,

let $opt[j, i]$ be the maximum weight for $S \subseteq [n]$ with $|S| \le k$

② $\begin{cases} opt[0,0] = 0 \\ opt[0, i] = 0 \\ opt[j, 0] = 0 \end{cases}$ $opt[j,i] = \max \{ \underbrace{w_j + opt[p_j, i-1]}_{\text{Solution with job } j}, \underbrace{opt[j-1, i]}_{\text{Solution without } j} \}$

running time here is $O(1)$

③ We compute opt[j, i] for j from 1 to n and for i from 1 to K. The running time here is $O(n \cdot K)$

④ The overall running time:

Sorting: $\theta(n \log n)$ Heap Sort

Compute P_j: $O(n \log n)$ binary search for j from 1 to n

compute opt[j,i] : $O(1) \cdot O(n \cdot k)$

Overall $= O(n \log n + nk)$

Problem 3. Longest - Increasing - subsequence

Since we are only required to output the length of the longest increasing subsequence. I will convert this problem to find the longest - common - subsequence between A and sorted(A)

$A = (a_1, a_2 \cdots a_n)$ $B=$ sorted(A)

Here I use Heap Sort, so the time-complexity for sorting is $\Theta(n\log(n))$

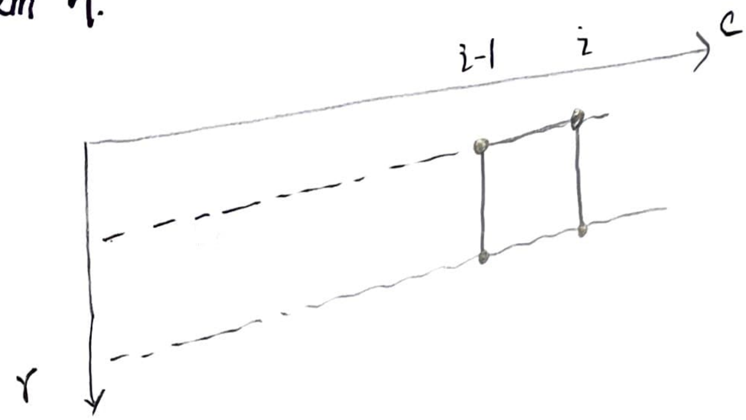
① let $opt[i,j]$, $0 \le i \le n$, $0 \le j \le n$; be the length of the LCS of A and B

② $\begin{cases} opt[0,0]=0, \\ opt[i,0]=0 \\ opt[0,j]=0 \end{cases}$ and $opt[i,j] = \begin{cases} opt[i-1,j-1]+1 & \text{if } A[i]=B[j] \\ \max\begin{cases} opt[i-1,j] \\ opt[i,j-1] \end{cases} & \text{if } A[i] \ne B[j] \end{cases}$

③ We compute $opt[i,j]$ use two for loops ① for i from 1 to n ② the inner for loops for j from 1 to n $(O(n^2))$

④ The running time for computing each $opt[i,j]$ is $O(1)$, so the DP takes $O(n^2)$. For sorting A, it takes $\Theta(n\log(n))$. So, overall time is $O(n^2)$

Problem 4.



1xn is simple

① For every $i \in (0,1,2,3 \cdots n)$, let $opt[i]$ be the optimum solution

let $optA[i]$ be the optimum solution with vertice i

let $optB[i]$ be the optimum solution without vertices i

②
$$optA[i] = optB[i-1] + W_i$$
$$optB[i] = \max \begin{cases} optA[i-1] \\ optB[i-1] \end{cases}$$
$$opt[i] = \max \begin{cases} optA[i] \\ optB[i] \end{cases}$$

$$\begin{cases} opt[0] = 0 \\ optA[0] = 0 \\ optB[0] = 0 \end{cases}$$

③ compute $opt[i]$, $optA[i]$, $optB[i]$ from 1 to n

④ Totall time $O(n)$

2×n Follow the idea of 1×n

① For every $i \in (0, 1, 2 \cdots n)$; let $opt[i]$ be the optimum solution.

let $optA[i]$ be the value of the optimum solution without vertices $(1, i)$ and $(2, i)$;

let $optB[i]$ be the value of the optimum solution with vertices $(1, i)$ but $(2, i)$;

let $optC[i]$ be the value of the optimum solution with vertices $(2, i)$ but $(1, j)$.

② $$optA[i] = \max \begin{cases} optA[i-1] \\ optB[i-1] \\ optC[i-1] \end{cases}$$

$$optB[i] = \max \left\{ \begin{array}{l} optA[i-1] \\ optC[i-1] \end{array} \right\} + W_{1,i}$$

$$optC[i] = \max \left\{ \begin{array}{l} optA[i-1] \\ optB[i-1] \end{array} \right\} + W_{2,i}$$

$$ope[i] = \max \begin{cases} opeA[i] \\ opeB[i] \\ optC[i] \end{cases}$$

3 We compare $ope[i]$, $opeA[i]$, $opeB[i]$, $opeC[i]$

From 2 to n.

4 Total time $O(n)$