

① Let S_{ij} denote the set of activities, we have to determine the activities A_{ij} which are compatible in S_{ij}

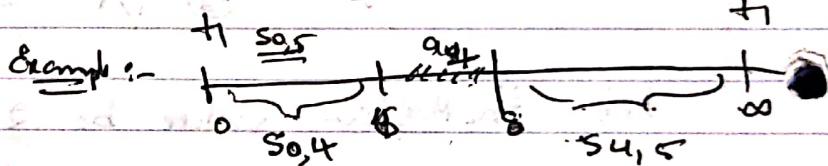
	a_0	a_1	a_2	a_3	a_4	a_5		
Example	s_i	0	2	1	3	6	∞	start time
	f_i	0	0	3	4	8	∞	end time
	Prof.	4	83	6	7	8	10	

$S_{ij} = \text{set of activities which starts after } a_i \text{ finishes and finishes before } a_j \text{ starts.}$

Select an activity i.e., $A_{[i,j]}$ which we consider to put in list of A_{ij}

$S_{i,k}, S_{k,j}$

S_{ij} is divided into three parts



Hence,

$$A_{ij} = S_{ij} \cup S_{i,k} \cup S_{k,j} \{a_k\}$$

$$A_{ij} = A_{ik} \cup A_{kj} \cup \{a_k\}$$

$$A[i,j] = \max_{i < k < j} \{ A[i,k] + A[k,j] + 1 \} \quad S_{ij} \neq \emptyset$$

$$= \emptyset \quad S_{ij} = \emptyset$$

* But given that profit is associated with each activity. Let p_k be the value associate.

Let p_k be the profit associated with each activity. On replacing 1 with p_k we have

$$A[i,j] = \max_{i < k < j} \{ A[i,k] + A[k,j] + p_k \} \quad S_{ij} \neq \emptyset$$

$$= \emptyset \quad S_{ij} = \emptyset$$

Profit-Maximizer (s, f, n)

Let $A[0 \dots n+1, 0 \dots n+1]$ and $\text{act}[0 \dots n+1, 0 \dots n+1]$
 be the tables of
 and $P[0 \dots n+1]$ be the profit table associated
 with activity

for $i = 0$ to n

{ $A[i, j] = 0$

$A[-i, i+1] = 0$

} $A[-n+1, n+1] = 0$

$A[n+1, n+1] = 0$

for $l = 2$ to $n+1$

for $i = 0$ to $n-l+1$

{ $j = i+1$

$A[i, j] = 0$

$k = j-1$

while $f[i] < f[k]$

{ $i =$

$f[i] \leq s[k]$ and $P[k] \leq s[j]$

and $A[i, k] + A[k, j] + P[k] > A[i, j]$

{

$$A[i, j] = A[i, k] + A[k, j] + P[k]$$

- $\text{act}[i, j] = k$

{

$K = k-1$

{

{

Print "The maximum compatible set with Maximum profit" is $A[0, n+1]$ "

Print - Activities (A , act , 0 , $n+1$)

3

Print - Activities (A , act , i , j)

8

If ($A[i, j] > 0$)

$$k = act[i, j]$$

Call point(k)

Print - Activities (A , act , i , k)

Print - Activities (A , act , k , j)

3

3. A problem similar to previous one is

constraint is $\leq m$

11 Ans. 3rd loop, the Time-Complexity of algorithm is $O(n^3)$

We can still optimize the Algorithm to $O(n \log n)$ by following way:

Key: If we assume that we only look at the last job when having sorted the jobs by finish time, then either we pick it or we don't.

Care! (we pick it) Then we need to remove all overlapping jobs and compute the solution for the subproblem of the jobs left.

case 2: (we don't pick it). Then we simply compute optimal solution of subproblem with $n-1$ items left.

If $\text{OPT}(j)$ is the set of all activities from 1 to j which returns max. weight

$$\text{OPT}(j) = \max \{ \text{profit}[j] + \text{OPT}(\text{PC}[j]), \text{OPT}(j-1) \}$$

\downarrow
returns set of non overlapping jobs.

$\text{PC}[j] \rightarrow$ stores index of closest non-overlapping job

```
int findMaxProfit( Job array[], int n )
```

Step 1: Sort jobs according to finish times

Step 2: Create an array to store values of subproblems profit

$$\text{maxProfit}[0] = \text{array}[0].\text{profit}$$

```
for ( i=1 to n )
```

```
{
```

Step 3: Find profit of current job

$$\text{int curProfit} = \text{array}[i].\text{profit}.$$

Step 4: get nonconflict index by using binary search

$$\text{int l} = \text{binarySearch}(\text{array}, i);$$

If ($l != -1$)

```
{
```

// include curprofit with non-overlapping Job profit

$$\text{curProfit} += \text{maxProfit}[l];$$

```
}
```

$$\text{maxProfit}[i] = \max \{ \text{curProfit}, \text{maxProfit}[i-1] \},$$

```
}
```

```
int result = maxProfit[n-1];  
return result;
```

{

Time Complexity - $O(n \log n)$

→ Sorting times - $O(n \log n)$

→ Finding non-conflicting Index $O(n \log n)$ using
binary search

→ Finding max weight $O(n)$

→ Backtracking to find optimal items $O(n)$.

2) * (a) ^{False.}
Solt: If programs are stored in increasing value of l_i .

Example :- Let $P_1 = 0.2$, $P_2 = 0.8$, $l_1 = 4$, $l_2 = 12$

Given $l_1 \neq l_2$, $T_{12} = P_1 \times l_1 + P_2 \times (l_1 + l_2)$

$$= 0.2 \times 4 + 0.8(16)$$

$$= 0.8 + 12.8$$

$$= 13.6$$

Consider $l_2 \rightarrow l_1$. $T_{21} = P_2 \times l_2 + P_1 \times (l_1 + l_2)$

$$= 0.8 \times 12 + 0.2(16)$$

$$= 9.6 + 3.2$$

$$= 12.8$$

We see that $T_{21} < T_{12}$, hence our assumption is false

Let's disprove this, Assume

$$\text{B} \quad T_{12} < T_{21}$$

$$\Rightarrow P_1 \times l_1 + P_2 \times (l_1 + l_2) < P_2 \times l_2 + P_1 \times (l_1 + l_2)$$

$$\Rightarrow P_1 \times l_1 + P_2 \times l_1 + P_2 \times l_2 < P_2 \times l_2 + P_1 \times l_1 + P_1 \times l_2$$

$$(P_2) \times l_2 < (P_1) \times l_2 \quad (\text{can't be False}).$$

This equation depends on P_1 & P_2 , hence we can't simply deduce the $T_{12} < T_{21}$ all the time.

If Programs are stored in increasing order of l_i , we can't minimize T

(b) False, If the Programs are arranged in decreasing order of P_i

Consider $P_1 = 0.4$, $P_2 = 0.3$, $l_1 = 25$, $l_2 = 5$

$$T_{12} = 0.4 \times 25 + 0.3 \times (25+5) \quad (P_1 \rightarrow P_2)$$
$$= 10 + 9 = 19$$

$$T_{21} = 0.3 \times 5 + 0.4 \times (25+5)$$
$$= 1.5 + 12 = 13.5$$

$T_{12} \geq T_{21}$ (Hence disproved)

Consider by equation If $P_2 > P_1$: $T_{12} < T_{21}$

$$P_1 \times l_1 + P_2 \times (l_1 + l_2) \leq P_2 \times l_2 + P_1 \times (l_1 + l_2)$$

$$P_1 \times l_1 + P_2 \times l_1 + P_2 \times l_2 \leq P_2 \times l_2 + P_1 \times l_1 + P_1 \times l_2$$
$$P_2 \times l_1 \leq P_1 \times l_2$$

The Equation depends on both l_1 & l_2 , hence the Time sequence is not guaranteed. And can't be proved in both cases.

Now, If the programs are stored in decreasing value of P_i we can't minimize T

Q) Yes, If then stored in decreasing value of P_i/d_i it will minimize T.

Let's prove this Consider programs $P_1, P_2, \dots, P_i, P_{i+1}, \dots, P_n$

If P_i is executed before P_{i+1} ,

$$T_1 = P_1 \times l_1 + P_2 \times l_2 + \dots + P_k (l_1 + l_2 + \dots + l_k) + P_{k+1} (l_1 + l_2 + \dots + l_{k+1}) + \dots + P_n (l_1 + \dots + l_n) \quad \text{eq } ①$$

If P_{i+1} is executed before P_i ,

$$T_2 = P_1 \times l_1 + P_2 \times l_2 + \dots + P_{k+1} (l_1 + l_2 + \dots + l_{k+1}) + P_k (l_1 + \dots + l_{k+1} + l_k) + \dots + P_n (l_1 + \dots + l_n) \quad \text{eq } ②$$

$T_2 > T_1 \Rightarrow T_2 - T_1 > 0$, when the time is minimized.

∴ If considering the decreasing order of $\frac{P_i}{d_i}$.

$$\begin{aligned} T_2 - T_1 &\Rightarrow (P_1 \times l_1 + P_2 \times l_2 + \dots + P_k (l_1 + l_2 + \dots + l_k) + P_{k+1} (l_1 + l_2 + \dots + l_k + l_{k+1}) + \dots + P_n (l_1 + \dots + l_n)) \\ &\quad - (P_1 \times l_1 + P_2 \times l_2 + \dots + P_{k+1} (l_1 + l_2 + \dots + l_{k+1}) + P_k (l_1 + \dots + l_{k+1} + l_k) + \dots + P_n (l_1 + \dots + l_n)) \end{aligned}$$

$$T_2 - T_1 = P_k \times l_{k+1} - P_{k+1} \times l_k$$

$$P_k \times l_{k+1} - P_{k+1} \times l_k > 0$$

$$\frac{P_k}{l_k} \geq \frac{P_{k+1}}{l_{k+1}} \quad (\text{Hence proved})$$

Therefore If the programs are stored in decreasing values of P_i/d_i it can guarantee the minimum retrieval time.

3) This problem may fall under NP-complete if sol:- trying to find the optimal solution, resulting in Exponential time complexity.

There can be strategies which may not provide optimal solution but feasible solution depending on the Input Instances.

1st strategy can be computing the largest independent set of graph formed by start & finish times. Put the set of jobs in one processor and remove corresponding nodes from graph. Then compute largest independent set of remaining graph so, on. until the all the jobs are assigned to some processor.

e.g. $c_1 \leq c_2 \leq c_3 \dots \leq c_n$, J_1, J_2, J_n - with start & finish times

Algorithm ProcessorAllocation:

begin

$A = \{a_1, a_2, \dots, a_n\}$ // Set of Activities

~~Random~~ P be the set of processors. $P[n]$, C be the cost of processor

do until ($A = \emptyset$)

begin

using Activity Selection Algorithm to set compatible activities (as Jobs) CA.

Assign a processor with lowest cost to

largest independent set cost = cost + $|A| * P[i]$

$A := A - CA$; $i \leftarrow i+1$

end

~~return~~ cost

end

Algorithm: Achre Selection (AoS)

begin

$n \leftarrow |A|$ where $A = \{a_1, \dots, a_n\}$

$CA \leftarrow \{a_1\}$

$i \leftarrow 1$

for $j = 2$ to n do

if $s_j \geq f_i$ then

begin

$CA \leftarrow CA \cup \{a_j\}$

$i \leftarrow j$

end

return CA

end

Strategy 2 : Can be finding the chromatic number partition of graph. As discussed in class, partition the node set of graph formed by start & finish times of activities into fewest no. of independent sets.

begin

Sort jobs by start times, $s_1 \leq s_2 \leq s_3 \dots \leq s_n$.

let P be the set of processors.

for $j=1$ to n

If (job j is compatible with some processor)
Schedule job j in any such processor.

else

Allocate a new processor p_j+1

Schedule job j in processor $p+1$

therefore $p \leftarrow p+1$

end. cost = cost + p

Return Schedule

the schedule contains minimum number of jobs processors with jobs assigned under it.

Calculate the cost by processors and number of jobs assigned to it respectively and return the cost

$$\text{Cost} = C_i * P_i + n = \text{no. of processor.}$$

```
for (i=0 ; i<n ; i++)
```

$$\text{cost} = C[i] * P[i] + \text{cost.}$$

// $C[i]$ = cost of processor

// $P[i]$ = no. of jobs in processor

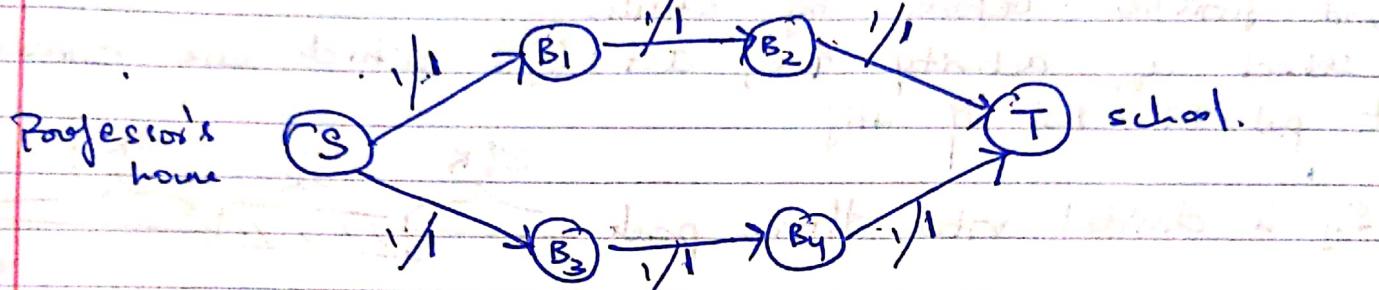
```
}
```

return cost.

Strategy 3: We can still find the minimum of SResult returned by Strategy 1 & Strategy 2, for more feasible results.

(A) Given:- the children refuse to walk together and also they don't step on same block that on same day at any time that one has stepped on. They can cross corners. Which is source & destination.

Considering the above requirements, we can have map in the following way:



Let the Max flow value be 2, for 2 children.

Consider the weight on each edge is 1 by setting the capacity of each edge to 1.

→ The Example Network flow has 2 disjoint paths satisfying the conditions and also the max flow '2' making it possible for the two children to reach their destination without any conflict.

Yet, the children can go to same school when Capacity of each edge is one & Max flow is 2.