

1.

False

Let us take the ~~match~~ preference list

$$m_1 \rightarrow \{ w_1 > w_2 \}$$

$$m_2 \rightarrow \{ w_2 > w_1 \}$$

$$w_1 \rightarrow \{ m_2 > m_1 \}$$

$$w_2 \rightarrow \{ m_1 > m_2 \}$$

In this situation, we can never get a stable matching

Let us take pair up m_1 with his most preferred w which is w_1 .

w_1 then is not paired with her highest preferred m which is m_2

Similarly if we pair (m_2, w_2) , then w_2 is not paired with m_1 .

∴ There is no way to include an m in a pair such that it can be paired with its highest preferred & we can't have w who can be paired with her highest preferred.

2.

True

Let us assume that there is a stable matching S for this instance (where m is ranked highest on preference list of w & w is ranked highest on preference list of m) \Rightarrow

the pairs (m_1, w_1) & (m_1, w_2) are taken.

Since w_1 is ranked top on m_1 's list, m_1 must choose w_1 , & since w_2 is ranked top on w_1 's list w_1 must choose m_1 . But this is unstable as m_1 & w_1 both prefer each other to their current pairs.

this is a contradiction, hence (m, w) must be in S

3. (a)

(i) There is no stable matching

- let A have 2 shows $\{a_1, a_2\}$ with rating $\{2, 4\}$.
- let D have 2 shows $\{d_1, d_2\}$ with rating $\{1, 3\}$.
- if a_1 is paired with d_1 , then D will want to switch the order of the shows in its schedule
- if a_1 is paired d_2 , then A will want to switch the order of the shows.

4. (a) Doesn't provide optimal schedule

e.g.: - $\Rightarrow A \Rightarrow$ start time = 0, end time = 10

$B, C, D, E \Rightarrow$ start time = 0, end time = 8 evenly spaced
& of equal length (2 \leftrightarrow)

the algo will choose A but optimal will choose B,C,D,E

(e) Doesn't provide optimal solution
 Using same scenario as used in (a), algo will choose A whereas optimal solⁿ chooses B,C,D,E

(c) This provides us with optimal solution
 Proof: let us take x as mentioned in the question.
 Let S be the schedule that doesn't have x , let y be the last course in S .
 $\therefore S[y] < S(x)$
 $\therefore F(i) < S(y) < S(x)$ for every i in S
 $\therefore S' = S - y + x$ is a valid schedule
 $\therefore S$ is an optimal solⁿ so is S'
 Hence proved

(d) This doesn't provide optimal solⁿ
 eg:- $\text{G class (start, end)}$
 $A(5,6)$
 $B(1,5.5) C(5.5,10)$

the optimal solⁿ chooses B,C while algo chooses A

(e) This doesn't provide optimal solⁿ
 eg:- $A(0,5) B(5,10) C(10,15) D(15,20)$
 $E(3,6) F(4,7) G(4.5,8) H(8,11) I(12,16)$
 $J(13,17) K(14,18)$
 Algo will choose H,G,I while optimal solⁿ will choose A,B,C,D

(f) This doesn't provide optimal solⁿ
 using same scenario as (d), algo chose A while optimal is B,C

(g) This doesn't give us optimal solⁿ
 using same eg from (e) but with an extra interval L(9,12). Algo \rightarrow chooses A, D, L while optimal chooses A, B, C, D

5. Algorithm:

- Sort the intervals in ~~increasing~~ accordance to L & then R.
- Select the first interval with the farthest endpoint
- Then keep a counter to keep track of the endpoint
- Then choose the interval with the farthest end point given that its starting point is less than or equal to the current ~~end~~ counter
- Repeat the steps till we get the last endpoint to cover all the intervals length
- This is a greedy algorithm, the major time taken will be done by sorting which can take $O(n \log n)$
- Let us say that our claim is wrong, consider optimal solⁿ containing the first interval,
- Let 'a' be the interval our new optimal solⁿ chooses,
- Let 'b' be the original interval chosen
- if 'a' starts at any later time than 'b' then it invalid solⁿ
- if 'a' also isn't end later than 'b' as in our algorithm
- Let us make a third optimal solⁿ that is the second optimal solⁿ but with 'a' replaced with 'b'
- This is a valid solⁿ of the same size as that of our actual optimal solⁿ

6.

A) we can see the frequencies in the following pattern

let a_1, a_2, \dots, a_n be letters sorted in increasing order of frequencies

let $f(a_i)$ denote the frequency of letter

$$a_i \quad i=2$$

$$f(a_i) > \sum_{j=1}^{i-2} f(a_j)$$

In this case, the lower frequency letters when grouped together should have combined frequency less than or equal to next letter frequency

Each step of Huffman coding reduces the no. of letters by 1 & increases the depth by 1

$$\therefore \text{total depth} = \underline{n-1}$$

B) To make it minimum $f(a_i) = \sum_{j=1}^{i-2} f(a_j) + 1$

$$\therefore f(a_i) = f(a_{i-1}) + f(a_{i-2})$$

As we have proved before, we shall now begin with frequency 1 to include the letter in Huffman coding

$$f(a_1) = 1$$

$$f(a_4) = 3$$

$$f(a_2) = 1$$

$$f(a_5) = 4$$

$$f(a_3) = 1$$

$$f(a_6) = 7$$

As we can see this follows a pattern

(we have take the equality constraint to minimize the next frequency)

$$f(a) = \left(\frac{1+\sqrt{5}}{2}\right)^{i-2} + \left(\frac{1-\sqrt{5}}{2}\right)^{i-2}$$

H/75

7. Algorithm:

- compute the cumulative sums of the array.

$$S_1 = a_1, S_2 = a_1 + a_2 = S_1 + a_2$$

$$S_3 = a_1 + a_2 + a_3 = S_2 + a_3$$

& so on this is $O(n)$ time

- now calculate sum of all possible intervals.

$$\text{sum of elements in } A[i:j] = S_j - S_{i-1}$$

- if $A[i:j]$ has the sum add it to a set of intervals $i = i, j = j$

- This will take $O(n^2)$ time throughout as no. of

combinations is $O(n^2)$

- intervals has all the intervals in A

- we use algorithm from Q5 to find the cover of all intervals in I

- any sub int belongs to atleast one interval, all the elements in A are present in overall intervals.

- computes smallest cover of intervals using Q5 to get minimum no. of the intervals

9. a, b, c, d, e, f

a, b, d, c, e, f

a, b, d, e, c, f

a, d, b, c, e, f

a, d, b, c, c, f

a, d, e, b, c, f

total 6

10.

- Pick any vertex as root
- Start a DFS, maintain DFS tree
- When a new vertex is seen, then add to the tree
- If encounters an edge from a node to an already visited node, then cycle exists
- Cycle can be returned by tracing the path from each node to the latest edge backwards in tree
- If DFS completes, then no cycle exists
- Entire procedure is dependent on DFS which takes $O(n+m)$ time
- Let us say one of the connected components is V_1, \dots, V_m ^{& is cycle}
- Let DFS visit V_1 first, & after some iteration V_2 is visited.
- While visiting all the other they will be marked visited, V_3 is visited then V_1 is adjacent but is also visited, hence cycle will be returned

11.

Assume graph is given & in-degree of each node is calculated.

- Select a node with indegree equal to 0 & not visited. If this is none & total visited no. of nodes is less than total no. of nodes then there is cycle.
- If all nodes have visited give the ordering.
- For selected node, mark it visited & put it in the ordering sequence.
- Reduce the in-degree for all the nodes that this node points to.
- Then repeat the above step.

12. ~~let us take $\{a, u_3\}$ edge in G that doesn't belong to T~~
- ~~• T is a DFS tree, let a be ancestor of b~~
 - ~~since in BFS tree, every node visited at the same level are equidistant from its parent that is 1 unit distance~~
 - ~~• As a is the ancestor b & BFS tree & DFS tree are same then a is parent of b~~
 - ~~• This is a contradiction as then $\{a, b\}$ edge is present in T~~

13. Proof (By contradiction)

Let us assume G is not connected

\therefore it has at least 2 components G_1 & G_2

\therefore every node must have degree of at least $n/2$

\therefore a node in G_1 is connected to $n/2$ nodes

\therefore there are $(n/2) + 1$ nodes in G_1 .

Similarly we can say in the case for G_2 .

\therefore the total nodes will become

$$(n/2 + 1)^2 = n + 2$$

\therefore this is a contradiction as we have only n nodes

$\therefore G$ is connected

We can similarly expand this proof for k components in that case the total no. of nodes will come as $\frac{n^k}{2} + k$ but that will contradict the no. of nodes to be n .

14.

Proof:

let us take a BFS tree T made from U with S as root

Each node t is at a length such that it is shortest path from S to t

\therefore All paths from S to t have length greater than $n/2$

$$\therefore \text{let } l = n/2 + 1$$

\therefore if each $n/2$ distance node has 2 or more nodes then total no. of nodes won't be n

\forall a node v st it lies in distance less than $n/2$

\therefore if we delete v , then all paths b/w S & t will be destroyed

Algorithm:

We will use a modified BFS Algorithm

while doing BFS, each node \rightarrow at the same distance from root node S will be put in an array

$A[i][a]$. i is distance from root node
 a is node address or node no.

After doing BFS

we find index i such that

$A[i]$ has only 1 element throughout

& that is the node we require & return that node

running time will be $O(m+n)$ as we are doing BFS on the entire U to find that particular node

TOPIC DATE

15.

Let us begin with BFS from a node v , we will keep a track of levels from v .

We know that in BFS we always get a shortest path b/w nodes.

Let us calculate the no. of shortest path from v to u .

Let A be this no. for u .
Let us take a node x at a level ' l '. The shortest $v - x$ path are of the form such that it is the shortest path to some node ' a ' in a level before ' l '.

& it takes only 1 unit to go from a to x .

∴ We can calculate the sum of all paths from a to x & to u upto u & similarly from v to u .

The time to compute ' A ' is at most the degree of the node u & sum of all degrees of nodes is $2m$.

∴ The overall running time is decided by BFS algo which is $O(m+n)$.

16.

Let us make a directed graph such that 2 vertices b_i & d_i are made for each person p_i .
Add edges b/w (b_i, d_j) if p_i was born before p_j died.

Add edges b/w (d_i, b_j) if p_i died before p_j was born.

Add edges (b_i, d_j) , (b_j, d_i) if regions of p_i & p_j overlap in same timespan.

Let us say G has a cycle.

Then each event associated with the nodes must precede the next but there no event will be first. This is incorrect

If G doesn't have a cycle, then we can create a topological sorting. It is possible w.r.t to birth & death time of all people & hence this will be consistent with the facts

17. Let us construct a directed graph G
 For each triple (C_i, C_j, t_a) in trace data
 we make nodes (C_i, t_a) & (C_j, t_a) & create
 directed edges joining in both directions
 Let us create arrays for each of the nodes & push the
 nodes into these arrays
 we create an edge between (C_i, t_a) & (C_i, t_b)
 if that triple has ever encountered before

Let us decide if a virus introduced to C_i at time a will infect C_j at time b .

Let us go through array of C_i , till we reach at some a' such that $a' \leq a$

Let us run a BFS from this node till we can reach a node (C_j, b') such that $b' \leq b$, if such a node is reached then C_j is infected by time b .

First we claim if \exists a path from (c_i, a') to (c_j, b') then c_j is infected by virus

The virus moves b/w computers c_p & c_q at time t_2 when edge from (c_p, t_1) to (c_q, t_2) is traversed in BFS

This is a possible virus transmission

Suppose there is a sequence of virus transmission such that virus leaves (c_i, t_a) & arrives (c_j, t_b) then we can start at node (c_i, t_a) & follow the edges in BFS

Each moment virus goes from c_p to c_q at t_2 we add edges from (c_2, t_1) to (c_2, t') when virus arrives at node c_j we should have added a node (c_j, t'')
 $t'' \leq t_a$

∴ There must be some sequence of edges completing the path

As we are doing a BFS & that defines the entire running time of the algorithm

$$\therefore \underline{O(m+n)}$$

8.

Steps Counter (n) $temp \leftarrow n$ $count \leftarrow 0$ while ($temp \neq 1$)if ($temp \% 2 == 0$) $temp / 2$ $count ++$

else

 $temp --$ $count ++$

return count

Let us say our number n is a power of 2
 then to reach the number we will keep on
 multiplying by 2 to reach it

it will take $O(\log n)$ time for that

Let us say our no. n is not a power of 2

$$T(n) = \min \{ T(n/2), T(n-1) \} + 1$$

if n is even then $T(n/2)$ will be smaller
 & each time a no. is even it must be divided
 by 2

if n is odd then $T(n-1)$ is defined after which
 it will divided by 2 as $n-1$ will be even
 giving us the least no. of steps

Worst Case : a no. is such that it always
 does the operation -1, /2 in this order till it
 reaches 1

Doing this in a loop it will again
 take $O(\log n)$ time

- 12.
- Let us say G is an undirected Tree T
 - DFS & BFS will then both produce tree & will contain all edges of T
 - DFS & BFS lists will be identical if same root & edges are present & will produce T
 - Let us prove by contradiction
 - let G contain cycle C
 - let the cycle C be the nodes $v_1, v_2, v_3, \dots, v_i, v_1$
 - In DFS tree, v_1, \dots, v_i will be on same path from root to leaf.
 - Let us say v_n is the node at which it arrives first
 - The other nodes will be visited at some point while visiting neighbours of v_n
 - In BFS tree cause branching while visiting v_n
 - \therefore BFS & DFS will produce same tree iff $G = T$

7.

NUM GROUPS (A[], ... n)

$S[1] = A[1]$; flag = false; if $A[1] > 0$: flag = true;
 for $i = 2$ to n :

$S[i] = S[i-1] + A[i]$

if $A[i] > 0$:

flag = true

if flag == false

return 0;

// let L, R maintain left & right endpoints of the intervals (sum of subarray is +ve)

for $i = 1$ to n :

for $j = i$ to n :

if ($S[j] - S[i] > 0$):

L.append(i)

R.append(j)

// Run algorithm from Q5 & it will compute min no. of +ve intervals that will cover all +ve nos.

Time Analysis: $\Theta(n) + \Theta(n^2) + \Theta(n \log n)$
 $= \underline{\underline{\Theta(n^2)}}$