1. As discussed in class (Kosaraju's Algo) that highest-finishing time vertex act as the source in Graph G.

as definition in question w is a source vertex as if there exist a path from any vertex to each other vertex v in graph. then this vertex should be source.

⇒ if there is a path from w to each other vertex.

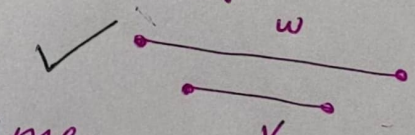
$w \longrightarrow v_1 \longrightarrow v_2 \cdots v_{n-1}$.

arranged in order of dec Finish time.

→ all the forward edge. will go to higher & these edges will ensure the reachability of ~~to every other~~ w to every other vertex.

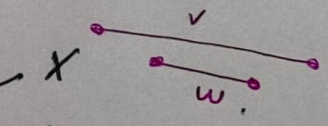
∀ vertex if path from w to v exist.

$w \rightsquigarrow v$.



⇒ w has higher finishing time.

if w has started its lifetime of v then it should finish after v if path b/w w to v exist.



(2) $C_1\ C_2\ C_3\ \ldots\ C_k$. $C_1 = 1$ $C_i | C_{i+1}$

let C_i be the largest coin & it is selected by our greedy solution of G.

$\Rightarrow$ $G_1 = C_i$ & rest problem becomes $M - C_i$

$C_1\ C_2\ C_3\ \ldots\ C_{i-1}\ \underline{C_i}\ C_{i+1}\ \ldots\ C_k$.

Let's assume there exist an optimal solution which does not select the C_i & partition problem with smaller problem (since smallest problem possible is $M - C_i$ as $C_{i+1} > M$) then it's

select C_i $\Leftarrow$ G_1 G_2 G_3 $\ldots$ G_k

O_1 $\underbrace{O_2\ O_3\ \ldots\ O_n}_{M - C_i}$.

Now assume that optimal solution does not select C_i $\Rightarrow$ it needs to select multiple coins from $[C_1, \ldots, C_{i-1}]$ which makes optimal solution worse.

hence optimal solution will be bounded to choose C_i too at first step.

(3)

n workers. worker $i^{\circ} \rightarrow S_i$

$T \Rightarrow$ given.

Objective function $\#(S_i + S_j > T)$

greedy criteria

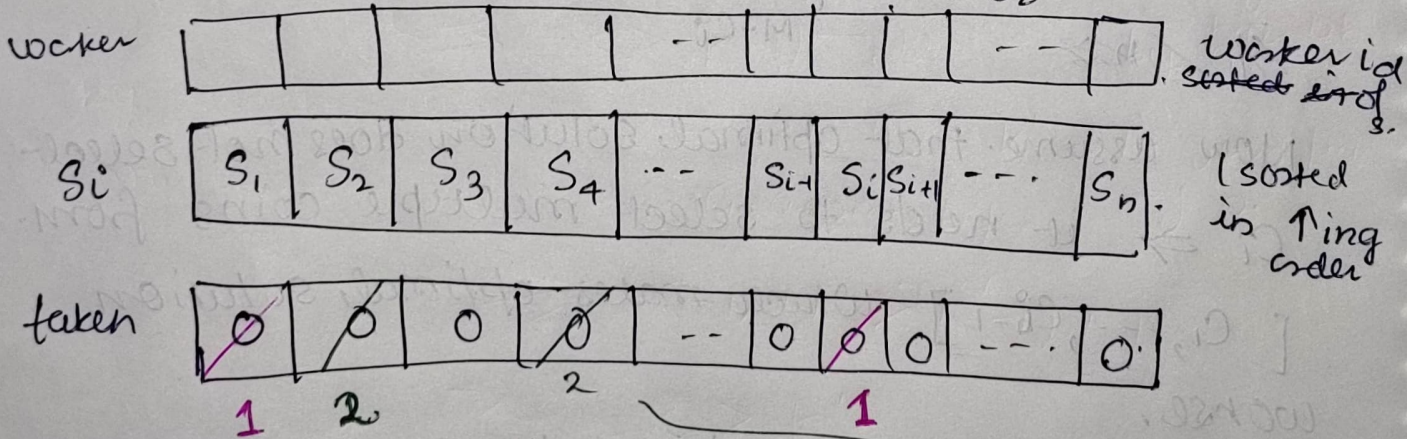
~~+) sort array S_i~~

1) sort workers with ~~value of~~ increasing order of S_i

2) design an array of size n with boolean value 0 initialized (this array is taken.

3) select the first worker (worker with min skill value) and pair it to ~~vertex~~ worker i such that $S_1 + S_{w(i-1)} < T$ & $S_1 + S_{w(i)} \geq T$

4) mark above two ~~vertices~~ workers in taken array as & team number.



$S_1 + S_{i-1} < T$

$S_1 + S_i \geq T$

$S_2 + S_3 < T$

$S_2 + S_4 \geq T$

Algorithm:

1) sort the skill array S_i
2) permute worker array as S_i
3) initialize taken array.

```
for i in Si (sorted)
    team_id = 1
    for j = 0 to n.
        if S[j] + S[i] >= T & & w[j] == 0
        then.
            taken[j] = taken[i] = team_id
        team_id = team_id + 1
return higest team_id.
```

i) an optimal Algo agrees to greedy with 1st step.

for S_1 ~~let's~~ greedy select S_i such that

$$S_1 + S_i \geq T \quad \& \quad S_1 + S_{i-1} < T$$

Assume: optimal Algorithm does not select S_i for S_1 instead selected S_j for S_i & S_1 to S_k

S_1 S_k
S_i S_j } pair selected by optimal.

$S_1 + S_k \geq T$ & $S_i + S_j \geq T$

as we know by greedy that $S_1 + S_i \geq T$

$\Rightarrow S_j \geq S_i$

selecting S_j ~~for S_1~~ instead of S_i will make the available skillset more deficit in skill and it may cause problem in optimal solution hence optimal solution should agree on 1st step.

ii) Let greedy soln is G & optimal soln is O

G : G_1 G_2 G_3 -- G_k

O : O_1 O_2 O_3 -- $O_{k'}$.

Agrees acc to (i).

acc to induction hypothesis on ~~problem~~ number of pair obtained by greedy & optimal (G & O).

Let's assume IH is true for $n-1$ steps.

(removing the 1st pair).

k : # good pair obtained by G.

~~$k \geq (k-1) \geq (k'-1)$~~ — ①

by adding the 1st pair in ① k' = # good pair by O.

on which both O & G agreed before

~~$k+1$~~ $(k-1)+1 \geq (k'-1)+1$

$\Rightarrow k > k'$

(proved for ~~next~~ step to

(4)

Approach :- by edmund's algo to find minimum arborescence.

⇓

- Select the ~~incoing~~ one edge per vertex that has min^m weight from its neighbour → done in O(n).
- n edges will be selected and there will be atmost ~~one~~ two cycle. (remove the highest-weight edge from the cycle.) → found in O(n)
- for all edges that are not selected (10 edges) add them one by one to spanning tree and remove the highest weight edge. + 1 edge

Algorithm.

1. for each vertex v : ~~select min~~ select min^m weight unvisited edge. & make graph G' from these. mark them visited.
2. run DFS to G' to find out two back edges and remove them to G' & mark them unvisited.
3. now we have (10 + 2) unvisited edges.

O(12) → for all unvisited edges
add them to G' & mark visited.
remove the highest weight edge in cycle formed in G' (don't mark that edge unvisited again).

justify why it is correct.

each vertex V chooses the ~~value~~ min edge that is ~~not~~ connected to it.

⇒ total n edges are selected.

⇒ almost 2 edges are back edge.

⇒ total 10+2 edges are left that could be part of minimum spanT.

⇒ all 12 edges are added to one by one and if they want to be in MST ⇒ if an edge which has weight greater than added edge we can safely remove that.

⇒ after checking all 12 edges we have the MST which is correct