Problem set-3

Name
Malla Sailesh
Rollnumber:
2021101106

A1)

(a)

Ans ① - ⑥ - ⑧ - ⑦ - ①

Here, weight 8 node is the highest weight.

So s has ⑧

Now we are left with nodes

① ①

Both of them are also included in s.

total weight : 8 + 1 + 1 = 10

But optimal solution is by choosing nodes with weights 6, 7

Ans 6 + 7 = 13.

Maximum sum which can be obtained.

(b)

Ans ⑧ - ① - ① - ⑨

node=1 node2 node3 node4

According to the algorithm : ⑧ ① (or) ⑨ ①

sum:=9 sum:=10

Chosen based on the maximum weight.

But actually choosing ⑧ ⑨ is the optimal way where we are getting 17 as sum

© we take vector<int> dp(n);
vector<int> weight(n);

we initialize, dp[0] = weight[0]

↳ It contains weights of all nodes.

to calculate dp[i]

↳ If there are i nodes then we need to find max value of sum independent.

so we may include the ith element or we may not include it

If we include → dp[i] = dp[i-2] + weight[i];

If we dont include = dp[i] = dp[i-1]

dp[i] = max(~~dp[i]~~, dp[i-1], dp[i-2] + weight[i])

Now we run a for loop for i = 1, ... n-1

```
for(int i=1; i<n; ++i){
if(i!=1)dp[i] = max(dp[i-1], dp[i-2] + weight[i]);
else dp[i] = max(weight[i], dp[i-1]);
}
```

Ans to the problem is dp[n-1]

↑ i.e dp[i-2] = 0 when i==1;

Thus time complexity = O(n)
space complexity = O(n).

correctness :- Let say dp2 gives the optimal solution.

Proof by induction :-

base case :- dp[0] = dp2[0] = weight[0]

because as only 1 node is there we will obviously take that node to maximize the score.

induction step

Now consider arbitary value of i

suppose the statement i.e. dp formula is correct $\forall\ i' < i$

$$dp2[i] = \max(dp2[i], dp2[i-2] + weight[i])$$

$$= \max(dp[i-1], dp[i-2] + weight[i])$$

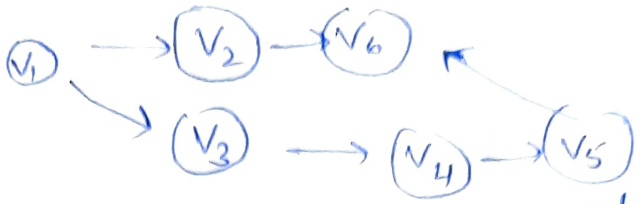
$\because$ $dp2[i'] = dp[i'] \ \forall i' < i$

$\Rightarrow dp[i]$

Thus proved $dp[n-1]$ gives the optimal solution.

A2)

a)



according to the given algorithm.

(V_1, V_2) edge is chosen.

$w = V_2$ } then $V_2 \to V_6$ is an edge
$L = 1$ $L = 2$

as there is no edge from V_6

we get (2) as the answer

But the longest length is 4 ($V_1 \to V_3 \to V_4 \to V_5 \to V_6$)

so the given algorithm is incorrect.

(b)

initialise an array : vector<int> dp(n+1); (initialised to zero by default)

stores the edges as adjacency list.

vector<int> edges[n];

idea is to find the max distance from 1 to i i = 2 to n.

longest path from 1 to 1 is 0

so dp[1] = 0.

Because we edges to node i from only lower indices : ~~there~~

(1) there is no incoming edge for node 1.

```
for i = 2; i <= N; i++
    for j = 1; j < i; j++
        if (j, i) is an edge
            dp[i] = max(dp[i], dp[j] + 1);
```

the ans is dp[N].

the ans will always be having a path from every node to N because ~~in~~ (b) have atleast 1 edge leaving it.

Note
dp[i] → finds longest path from 1 to i

time complexity = $O(n^2)$

space complexity = $O(n)$.

Correctness : proof by induction :

let dp2 gives us the correct answer.

Base step : dp2[1] = 0 = dp[1].

because there is no path 1 to 1 having length > 0

induction step

Now consider arbitary values of i
suppose the statement we said i ∈ U, algo. is correct for all values of i' < i

then for j = 1 ; j < i ; j++
if (j, i) is an edge dp2[i] = max(dp[dp2[i], dp2[j] + 1)
= max(dp[i], dp[j] + 1)
⇒ dp[i]

hence proved the above algo is correct.

A3). Let us take two vector<int> dp(n+1, 0)
vector<int> S(n+1);
↑
this stores the sequence of values

dp[i] → min cost required to take supply for i weeks.

If on the i^{th} week,

contract taken by A : $r \times s_i + dp[i-1]$
$= dp[i]$
cost for taking $i-1$ weeks
$+ r \cdot s_i$

contract taken by B : $dp[i-4] + 4C$
because contract taken for 4 consecutive weeks.

Thus, $dp[i] = \max(dp[i-1] + r \times s_i, dp[i-4] + 4C)$.

if ($n > 0$) $dp[1] = s_1$
if ($n > 1$) $dp[2] = dp[1] + r \cdot s_2$
if ($n > 2$) $dp[3] = dp[2] + r \cdot s_3$
for (int $i = 4$; $i <= n$; $i++$)
$dp[i] = \max(dp[i-1] + r \cdot s_i, dp[i-4] + 4C)$;

dp[n] gives us the answer.

correctness

Proof by induction.

Let dp2 gives us the optimal solution.

base case :

$dp2[1] = dp[1] = \gamma \cdot s_1$

$dp2[2] = dp[2] = \gamma \cdot s_2 + dp[1]$

$dp2[3] = dp[3] = \gamma \cdot s_3 + dp[2]$

because as there are less than 4 weeks. So assuming B don't agree to contract if $n < 4$.

induction step :

Now consider arbitary value of i suppose the statement we said i.e the ~~values~~ algo is correct $\forall$ values $i' < i$

$dp2[i] = \max(dp2[i-1] + \gamma \cdot s_i, dp2[i-4] + 4c)$

$= \max(dp[i-1] + \gamma \cdot s_i, dp[i-4] + 4c)$

$= dp[i]$

Thus proved.

A4) Initialise. vector<int> dp(n+1, 0).
vector<int> c(n+1)
↳ store the value of placement of copy of file at the server i

dp[i] = min cost taken for 1 to i servers.

dp[0] = 0. ~~if~~ no copy of file at any server except n.

$$dp[i] = c_i + \min_{0 \le k < i} \left(dp[k] + \text{value} \right)$$

c_i ↑ because we should anyway keep it at last server

dp[k] ↑ if server is at k.

value = $i-(k+1) + i-(k+2) + \dots + i-(i)$.

because the last copy of file before i^{th} server is at k^{th} server ($1 < k < i$) so $(k+1)^{th}$, $(k+2)^{th}$ ... i^{th} servers will take file from i^{th} server

Thus.

$$dp[i] = c_i + \min_{0 \le k < i} \left(dp[k] + [i-(k+1) + i-(k+2) + \dots + i-(i)] \right)$$

$$= c_i + \min_{0 \le k < i} \left(dp[k] + (0+1+\dots+(i-k-1)) \right)$$

$$\boxed{dp[i] = c_i + \min_{0 \le k < i} \left(dp[k] + \frac{(i-k)(i-k-1)}{2} \right)}$$

Thus.

```
for(int i = 1; i <= n; i++)
{
    int temp = dp[0] + i*(i-1)/2;
    for(int k = 0; k < i; k++)
        temp = min(temp, dp[k] + (i-k)*(i-k-1)/2);
    dp[i] = temp + c[i];
}
```

Time complexity = $O(n^2)$

Space complexity = $O(n)$.

Correctness : proof by induction.
in dp2 gives the optimal solution

Base case step : $dp2[0] = dp[0] = 0$.
at no server copy of file kept
no cost = 0.

induction step :- Now consider arbitary value of i
suppose the statement we said i.e
the algo is correct $\forall$ values $i < i$

$$dp2[i] = c_i + \min_{0 \le k < i} \left(dp2[k] + \frac{(i-k)(i-k-1)}{2} \right)$$

$$= c_i + \min_{0 \le k < i} \left(dp[k] + \frac{(i-k)(i-k-1)}{2} \right)$$

$$= dp[i]$$

Thus proved.

A5) (a)

first let us consider an edge (u, v). where u, v are vertices in G.

$d(u)$ - min. distance from u to t (cost).

$d(v)$ - min cost from v to t

possible $\{ d(u) \leq d(v) + cost_e(u \to v)$ ← edge

⇓

not possible $\{ d(u) > d(v) + cost_e(u \to v)$

because if $v \to t$ path has min cost of $d(v)$ then $u \to t$ can atmost have $d(v) + cost_e(u \to v)$ cost if it choose to go from $u \to v$ then $v \to t$.

If there exist a shortest path with cost less than $d(v) + cost_e(u \to v)$ then also its fine.

Thus $\boxed{d(u) \leq d(v) + cost_e(u \to v)}$ ✓

so ∀ edge $e \in F$ if you found

$d(u) > d(v) + cost_e(u \to v)$ then the claim is wrong.

Now remove all the edges in G except those which satisfies $d(u) = d(v) + cost_e(u \to v)$ to get Graph G_1. (say).

could be done using DFS after that.

Now check. if every node has path to t in G_1, if not claim is incorrect. or else correct

and running explanation :-

Let $d(v)$ be the min-cost path from v to t in G_1. Suppose $d'(v)$ be the min cost path from v to t in G.

It more edges (i.e have extra edge for G than G_1)

$\hookrightarrow d'(v) \leq d(v)$

because all edges of G_1 are in G and also extra edges are there in G so. it may have min cost path. so $d'(v) \leq d(v)$ — (1)

Let (y,t) be the edge in G.

$d(y) = d(t) + cost_e(y \to t)$

$\Rightarrow cost_e(y \to t)$.

Let (y,t) be the edge in G.

$d(y) = d(t) + cost_e(y \to t)$.

thus (y,t) is in G' also

$d(y) = d'(y)$

Let us consider a node s such that (s, y) is an edge in .

from (1)

$$d'(s) \leq d(s).$$

~~$d(y) + c_e(s \to y)$~~.

~~$d(s)$~~ $d'(s) = d'(y) + c_e(s \to y)$

$= d(y) + c_e(s \to y)$

$= d(s)$.

thus by $\forall$ node in V $d'(v) = d(v)$

Hence shown.

Time complexity Dijkstra $= O(|E| \log |V|)$

$= O(n \log n)$

(4)

referred Kleinberg and Tardos

So we can simply use dijkstra to find the minimum from L' to all other nodes.

But the problem is Dijkstra works only for positive edges.

Here it was said that edges may have cost weights negative

So using the hint

$$c_e' = c_e - d(v) + d(w).$$

if (v, w) is edge.

c_e - initial cost of edge (v, w)

c_e' - ~~dijkstra~~ modified cost of edge (v, w).

$d(v) \leq d(w) + c_e$.

from previous result in (a)

$\Rightarrow \quad d(w) - d(v) + c_e \geq 0$

$c_e' \geq 0$

clearly new edge costs are positive thus now we can run Dykstra on it.

Thm shortest path wrt c_e' gives also shortest path wrt c_e

For proof let us consider a path P

$c(p)$: cost of path $= \sum_{e \in p} cost_e$

$c'(p)$: modified cost $= c(p) - d(s) + d(t')$

$d(s)$: first edge, $d(t')$: last edge

$cost_{e_1} - d(s) + d(s_1) + cost_{e_2} - d(s_1) + d(s_2) + \ldots + cost_{e_n} - d(s_n) + d(t')$

Thus the dijkstra gives $c'(p)$ $\forall$ nodes in v to t'. We $d(s), d(t') \; \forall \; s \in V$

Thus we can easily find $c(p)$ i.e cost of path from any node to t'.