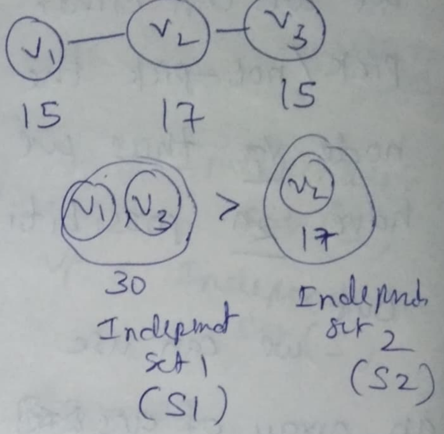
AAD - Assignment-3

ProblemSet-3

NAGA MANOHAR
2021101128

Q1

(a)



The algorithm fails when the sum of neighbouring weights is greater than the weight of ~~the~~ heaviest node

let v_1, v_2, v_3 be nodes with weights

$w(v_1) = 15$

$w(v_2) = 17$

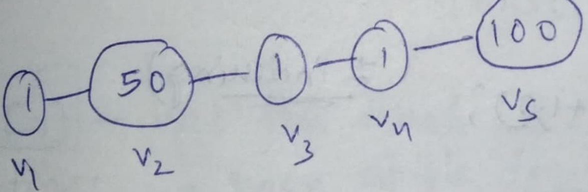
$w(v_3) = 15$

according to the algorithm ⇒ Max-weight $(S_2) = 17$

but actual max-weight possible $= 30$ (S_1)

(b)

again consider the path $G(V, E)$ (v_1, v_2, v_3, v_4, v_5 - are nodes)



the values inside nodes are weights assigned to them.

with the algorithm we get

$$ans = \max(w(\{v_1, v_3, v_5\}), w(\{v_2, v_4\}))$$

$$= \max(1+1+100, 50+1)$$

$$= 102$$

But consider the set $\{v_2, v_5\}$ ⇒ $w(\{v_2, v_5\})$

$ans' = 50 + 100$

$= 150$

$150 > 100$

$\boxed{ans' > ans}$

Hence, (a), (b) both fail at some point

(c) let path be $G(V,E)$ - with nodes $\{v_1, v_2, v_3 \ldots v_n\}$

$\Rightarrow$

$v_1 - v_2 - v_3 - \cdots - v_n$

$G(V,E)$

let the weight of v_i be w_i

Observations:-

To get a maximum weight independent set we can either pick/not-pick the node v_i thus we have 2^n possibilities.

let Max-total-weight of independent set be called ans[n] $(v_1, v_2 \ldots v_n)$

But we can use an array of ans

and ans[i] = stores the answer for $(v_1, v_2 \ldots v_i)$

(1) first we calculated the max-total-weight = ans[n]

(2) Then modify the algorithm slightly $\rightarrow$ to also store the elements of independent set that contribute to ~~answer~~ ans[i]

(1) initialize

(1 indexing)

```
vector<int> ans(n+1,0);
ans[0] = 0;
vector<int> w(n+1,0);  → weight array
                          w[i] ⇒ ith node
ans[1] = w[1];
for(i=2; i<=n; i++)

    ans[i] = max(ans[i-1], ans[i-2]+w[i]);
```

Here

ans[i] = ~~ans~~ max (ans[i-1], ans[i-2] + w[i])

$\downarrow$ don't pick $\underline{v_i}$

$\downarrow$ pick $\underline{v_i}$ $\Rightarrow$ v_{i-1} is not considered

(2) Now

add a $\underline{\text{vist (n+1, 0)}}$ ~~array~~

vector (C++) $\Rightarrow$ vist[i] = 0 $\Rightarrow$ $v_i \notin$ Independent set (S)

vist[i] = 1 $\Rightarrow$ $v_i \in$ Independent set (S)

and

Now

we make change in for_loop as

```
for (i = 2 ; i <= n ; i++) {
    if ((ans[i-2] + w[i]) > ans[i-1]) {
        vist[i] = 1 ;      (include vi)
        vist[i-1] = 0 ;    (exclude vi-1
                            if it's picked
                            previously )
    }
    ans[i] = max(ans[i-1], ans[i-2] + w[i]) ;
}
```

$\boxed{\text{ans[n]}}$ – has the final Answer

Runtime: – we have single $\underline{\text{for loop}}$

$\Rightarrow$ $O(\underline{n})$

v_{i-1} v_i — v_{i+1}

Here we don't considers neighbour's on b/s for because we consider that G is only till $\boxed{v_i}$ (Assume)

ans[i] and as v_n doesn't have node after it (v_{n+1})

we break and get final answer

Correctness :

Proof by Induction : $ans[i] = max(ans[i-1], ans[i-2] + w[i]);$

Given, that we already calculated final values of $ans[i-1]$, $ans[i-2]$ in the recursion calls

① for $i=1$ — Base Case :

~~$ans[2] = max(ans[1]$~~

as $(v_1) \Rightarrow$ for one node max-weight $\rightarrow w_1$

$ans[1] = w[1];$

for $i=2 \Rightarrow$ $ans[2] = max(ans[1], ans[0] + w[i])$ and $ans[0] = 0$

$(v_1) - (v_2)$ here pick v_1 / v_2

$\Rightarrow ans[2] = max(w_1, w_2) = max(ans(1), w_2)$

$\Rightarrow$ is correct.

② Assume that the statement is true for $n \leq k$ $(k \geq 2)$

$\Rightarrow ans[k] = max(ans[k-1], ans[k-2] + w[k])$ is true for $n = k+1$

③ Now we prove that the statement

$\Rightarrow ans[k+1]$ — not picked $\Rightarrow ans[k]$; picked $\Rightarrow ans[k-1] + w[k+1]$

$ans[k+1] = max(ans[k], ans[k-1] + w[k+1])$

Now by induction also

$ans[k+1] = max(ans[k+1-1], ans[k+1-2] + w[k+1])$

$= max(ans[k], ans[k-1] + w[k+1])$

Hence, it is true for $\boxed{n = k+1}$

Hence the st is true $\forall n \in N$

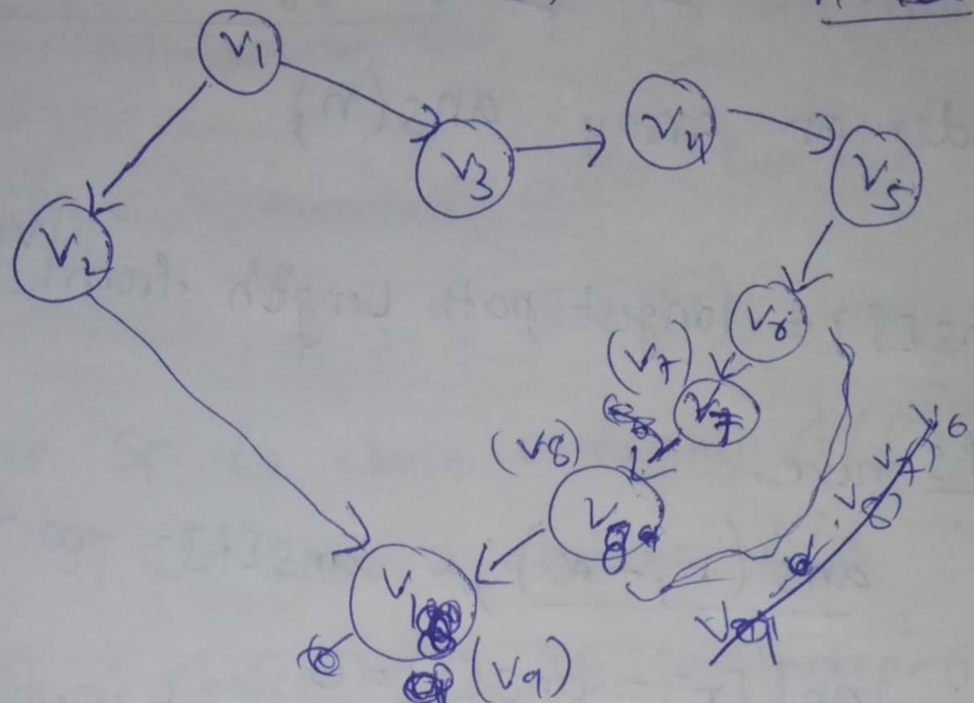
$\therefore$ The st is true

Hence, the Algorithm is correct

Q2

(a) consider the Graph G(V, E)

n = ~~100~~ nodes



Now the Algorithm gives L = longest Path length from v_1 to v_n

as

$L = 2 \Rightarrow \{v_1 \rightarrow v_2 \rightarrow v_{100}\}$

but actual longest path

$\Rightarrow L' = 7 \Rightarrow \{v_1 \rightarrow v_3 \rightarrow v_4 \rightarrow v_5 \rightarrow v_6 \rightarrow v_7 \rightarrow v_8 \rightarrow v_9\}$

Hence $\boxed{L' > L}$

$7 > 2$

The Algorithm fails in this case.

(b) let Graph $G(V,E)$ and nodes be $v_1, v_2, \dots v_n$

s/t ~~directed~~ edge (v_i, v_j) exists iff $i<j$ $(v_i \to v_j)$

Now

Consider an array ans(n)

s/t

ans[i] = longest path length from $\boxed{v_i \text{ to } v_n}$

we use dfs here.

initialise ans(n, -∞); ans[i] = -∞ ∀i

then ans[n] = ans[0] = 0 → not used.

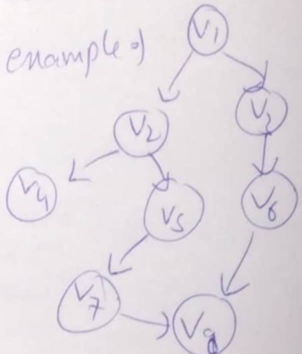
Now start a dfs from v_1 as follows

dfs on v_1 ⇒ $\boxed{\text{dfs(1)}}$

```
dfs (~~int~~ at) → index ⇒
   visited[at] = 1; tempans = -∞;
   for adjacent nodes i of at {     (vi of v_at)
       if (!visited[i]) {
           dfs(i)
       }
       tempans = max(tempans, ans[i] + 1);
   }
   ans[at] = tempans;
```

Here the dfs call goes to the complete depth and when the nodes till the level just below are updated then only we update our tempans

example)



we update dfs(1) temp ans for $\boxed{at = 1}$ only when dfs(2) & dfs(3) return

thus we already got updated ans[2], ans[3] values.

then

ans[1] = max (ans[2] + 1, ans(3) + 1)) is correct!

and for nodes like v_4 ans[u] = $-\infty$ thus

it doesn't affect our answer

but ~~for~~ when we reach v_n ans[n] = 0 | when we ~~with~~ have edge (i,n)

(i → n) edge ans[i] = $-\infty$

then: ans[i] will be updated to 1

ans<i] = max(0+1, $-\infty$)

= ~~max~~ 1

Runtime: O(V+E) — (∵ dfs)

V: vertices, E: edges

correctness:

① we first see that there are No-Cycles in the Graph G(V,E)

as (given) ⇒ ∃ an edge $e = (v_i, v_j)$ iff $i < j$ $(v_i \to v_j)$

Now consider we start making a path from v_i and form edges $v_i \to v_j$ $(i<j)$ and then $v_j \to v_k$ $(j<k)$

and so on ---

we observe that there are No-Edges going back to there previous nodes as.

$i, j, k \ldots \Rightarrow i < j < k < \ldots < n$

thus v_k can never go to any node before it, similarly ∀ nodes.

Hence, we never get a cycle

(2)

Proof by Induction :-

first thing is that for all neighbouring nodes of V_{at} say v_i's we first find ans[i]'s by recursively calling dfs(i)

thus when we evaluate

(tempans = ans[at])

ans[at] = max(ans[at], ans[i]+1);

S.t — we would have already got correct value of ans[i].

Now

(1) Base case :- $\boxed{n=1}$

$\Rightarrow$ ans[1] = 0 ($\because$ No adjacent nodes
No edges from $v_1 \to v_1$)

$\boxed{n=2}$ ans[2] = max(ans[1], ans[1]+1);
$\qquad$ 0 $\qquad$ 0

ans[2] = 1 (only $\exists$ an edge (v_1, v_2)).
$(v_1 \to v_2)$

(2) Assume that S.t is true for $n \geq k$ ($\because$ we consider $(v_k \to v_n)$)

$\Rightarrow$ ans[k] = max(ans[k], ans[i]+1); $\forall$ neighbouring nodes v_i of v_k
$i > k$

(3) Now for n = k+1

ans[k+1] = max(ans[k+1], ans[j]+1); $\forall$ neighbouring nodes $j > k \to v_j$ of v_{k+1}
— (1)

By algo :- ans[k-1] — ans[k-1] (or) ans[j]+1 $\boxed{j > k-1}$

$\Rightarrow$ ans[k-1] = max(ans[k-1], ans[j]+1);
— (2)

From (1), (2) $\Rightarrow$ S.t. is true $\forall n \in N$ $n \geq 1$

Hence the Algorithm is Correct

(Q3) 1-indexing

let $ans(n)$ = Array that store minimum cost's for all indices i.

s/t $ans[i]$ = minimum cost for the $\{S_1, S_2 \ldots S_i\}$

for a given S_i to chose company A/B

=) we have 2 choices A/B

if (A) is chosen $\Rightarrow$ we need not check if there is any block of 4 taken before

that is whether ~~S[i-3], S[i~~ $S_{i-3}, S_{i-2}, S_{i-1}$

belong to a block of 4/not

thus we get $\boxed{ans[i-1] + r \cdot s_i \text{ (for A)}}$ — ①

ans[i]

if (B) is chosen.

$\Rightarrow$ $S_i \in B$

$\Rightarrow$ there should ~~be~~ be 3 $S_j \in B$ contiguous with S_i

~~Here~~ Here we consider only till S_i

$\Rightarrow$ the S_j are $S_{i-3}, S_{i-2}, S_{i-1}$ should $\in B$

Now

consider a case -

for ans[i-1], $S_{i-3}, S_{i-2} \in B$ and $S_{i-1} \in A$

then ~~to get~~

$\Rightarrow$ ans[i] = ans[i-1] $- x.S_{i-1}$ *

$-2^i c + 4^i c$

(S_{i-3}, S_{i-2})

for this we have to use an array vist(n)

and vist[i] = 0 $\Rightarrow S_i \in A$ (it dynamically gets

vist [i] = 1 $\Rightarrow S_i \in B$ updated with i)

$ans[i] = ans[i-1] + 4 \cdot c$

so, if (vist[j] == 0) then $ans[i] \mathrel{-}= r.s_j$;

else $\quad ans[i] \mathrel{-}= c$;

$\boxed{j \in \{i-3, i-2, i-1\}}$

update vist[j] values.

instead we can directly consider $ans[i-4]$ and add $4 \cdot c$

$$ans[i] = \boxed{ans[i-4] + 4 \cdot c} \quad -(2)$$

$\boxed{ans[0] = 0}$

Now from (1), (2) we have

for $i=1, i=2, i=3$
$ans[i] = ans[i-1] + r.s_i$
($\because$ can't chose B - Not 4 continuous)

$$ans[i] = \max(\underbrace{ans[i-1] + r.s_i}_{\text{chose A}}, \underbrace{ans[i-4] + 4 \cdot c}_{\text{chose B.}})$$

(if taken update) (vist[i] = 0) $\quad$ (vist[i] = 1)

return ans[n] }

Run Time: each ans[i] is calculated in $O(1)$

thus $1-n \Rightarrow O(n)$

Correctness: Proof by Induction:

(1) Base case:

$n=1 \Rightarrow ans[1] = \underset{0}{ans[0]} + r.s_1 = r.s_1$

$n=2 \Rightarrow ans[2] = ans[1] + r.s_2 = r.s_1 + r.s_2$

$n=3 \Rightarrow ans[3] = ans[2] + r.s_3 = r.s_1 + r.s_2 + r.s_3$

Since there is no contingnous 4 weeks for $n \leq 3$ we can only take company A

St $\Rightarrow ans[i] = \max(ans[i-1] + r.s_i, ans[i-4] + 4.c)$

(2) Assume S.t is true $\forall\, n \le k$

$\Rightarrow$ $ans[k] = \max(ans[k-1] + r.S_k,\ ans[k-4] + 4.c);$

(3) Now we prove S.t for $n = k+1$

$ans[k+1] = \max(ans[k+1-1] + r.S_{k+1},\ ans[k+1-4] + 4.c);$

$= \max(ans[k] + r.S_{k+1},\ ans[k-3] + 4.c)]$ — (1)

Now by algorithm

$ans[k+1]$ — chose A $\Rightarrow$ $ans[k] + r.S_{k+1}$

— chose B $\Rightarrow$ $ans[k-3] + 4.c$

thus $ans[k+1] = \min(ans[k] + r.S_{k+1},\ ans[k-3] + 4.c)$

(∵ (1))

$\Rightarrow$ S.t is true for $n = k+1$

Hence S.t is true $\forall\, n \in N,\ n \ge 1$

Hence, the Algorithm is correct

(Q4)

Given, n servers

$S_1, S_2, S_3 \ldots S_n$ & $C_i \rightarrow S_i$

: cost's for servers

$C_1, C_2, C_3 \ldots C_n$

let $ans(n+1)$ be an array s/t

1-indexing

$ans[i]$ = Required minimum cost for the servers range $(1, i)$

Now we evaluate

$$ans[i] = \min_{1 \le j < i} \left\{ ans[j] + \text{sum of distances for } \{S_{j+1} \ldots S_i\} \right\} + C_i$$

$$= \min_{1 \le j < i} \left\{ ans[j] + \underline{0+1+2+\ldots+(i-j-1)} \right\} + C_i$$

$$= \min_{1 \le j < i} \left\{ ans[j] + {}^{i-j}C_2 \right\} + C_i$$

no. of combinations like for ${}^1C_2 = 0$, ${}^2C_2 = 2! = 2$

Base Condition: $ans[0] = 0$

Run Time: $i-1$ iterations for each i in range $(1, n)$

$\Rightarrow O(n^2)$

Correctness:

Proof by Induction:

s.t $\Rightarrow ans[i] = \min_{1 \le j < i} \{ ans[j] + {}^{i-j}C_2 \} + C_i$

① Base case:

$n=1 \Rightarrow ans[1] = \min_{1 \le 1} \{ans[0] + {}^{1}C_2\} + c_1 = ans[0] + c_1$

$ans[1] = c_1$

As there is is only 1 server.

Now
② Assume st is true for $n \le k$

$\Rightarrow ans[k] = \min_{1 \le j < k} \{ans[j] + {}^{k-j}C_2\} + C_k$

③ Now we prove that the st is true for $n = k+1$

$\Rightarrow ans[k+1] = \min_{1 \le j < k+1} \{ans[j] + {}^{k+1-j}C_2\} + C_{k+1}$ — ①

By algorithm

$ans[k+1] \Rightarrow$ min cost from any $j < k+1$ + all distances $j+1$ till $k+1$ and cost of $(k+1)^{th}$ server. — ②

$ans[k+1] = \min_{1 \le j < k+1} \{ans[j] + {}^{k+1-j}C_2\} + C_{k+1}$

①, ② $\Rightarrow$ s.t is true for $n = k+1$

Hence the s.t is true $\forall$ $n \in N$ $n \ge 1$

Hence the Algorithm is correct

NOTE: I have referred to a question that is similar to this problem - online.

Q5

(a)

<u>Algorithm</u>: Given Graph $G(V, E)$ sink t

nodes $v \in V$

Here we have 2 cases:

① Algorithm rejects G.

② Algorithm Accepts G.

①

<u>case 1</u>: if $\exists$ a $\underline{w}$ s/t edge(v, w)

and $\underline{d(v) > d(w) + c_e}$ c_e = cost of edge (v, w)

So, now if $d(w)$ is correct (min-cost) then

definitely $\exists$ a path from $\underline{v}$ to $\underline{t}$ via w

and as $d(v) > d(w) + c_e \Rightarrow d(v)$ is ~~incorrect~~ wrong

<u>case 2</u>: if $d(v) = d(w) + c_e$ then make

a new graph H s/t H has only edges $e(v, w)$ from

$G(V, E)$ satisfying this condition $\boxed{d(v) = d(w) + c_e}$

Now we apply a DFS on all nodes $(\forall v \in V)$

if we have path from (v) to t

then continue

else reject.

~~Accept.~~

Accept - if we get no reject instances.

As.

W.K.T if $d(v)$ are correct then

all edges in shortest path from v to t. $\in H$

Hence, we can always reach t from v if $d(v)$ is correct in H.

(2)

Let $d'(v)$ be actual distance from v to t in G $\forall$ nodes v.

and we should prove that $d'(v) = d(v)$ $\forall v$

W.K.T $d'(v) \leq d(v)$

Now consider $d'(v) < d(v)$ and let there be a actual shortest path P from v to t s/t

s/t v, x, y, t is there and x is the last node holding the condition $d'(x) < d(x)$ and also

as x is the last node from $v \to x$ thus y will have $d'(y) = d(y)$ ($\because$ p is real shortest path)

Now as x, y are in P

$\Rightarrow d'(x) = d'(y) + c_e(x,y)$

$\Rightarrow d(x) > d'(x) = d'(y) + c_e = d(y) + c_e$

~~$d(x) = d(x)$~~

This is a contradiction as the algorithm would reject by (1) ($d(v) > d(w) + c_e$)

Thus, $d'(v) = d(v)$ $\forall v$

Hence, proved

consistent

(b)

We can use dijkstra in this question but we modify negative edge weights so that there is no actual effect but dijkstra can be applied.

Now consider for the cost of an edge $e(v,w)$

$c'_e = c_e - d(v) + d(w)$

we see that the new costs are ≥ 0

since if $c'_e < 0$ then $d(v) \nleq d(w) + c_e$

which isn't true

Now consider a path p from an node x to t'

then actual cost of $P \Rightarrow c(p) = \sum_{e \in p} c_e$

and new costs $\Rightarrow c'(p) = \sum_{e \in p} c'_e = \sum_{e \in p} c_e - d(v) + d(w)$

but in the path ~~for~~ considering node x, t' we have

$c'(p) = c(p) - d(x) + d(t')$ (all the other nodes get cancelled out)

$-d(v_1) + \cancel{d(v_2)}$
$\cancel{-d(v_2)} + d(v_3)$

thus

$c'(p) = c(p) - d(x) + d(t')$

$\Rightarrow$ ~~c(p)~~ c'

$\rightarrow c'(p) = c(p) + \underbrace{(d'(t) - d(x))}_{constant}$ — ①

→ set of min cost paths from x to t' under $\underline{c'}$ is the same as set under $\underline{c}$

→ ① holds true given x to t' under c'

Hence,

① calculate modified costs ∀ ~~nodes~~ edges (c_e)

② Apply dijkstra's Algorithm

③ add $d'(t) + d(x)$

Thus overall time complexity will not exceed

Runtime : $O(m \log n)$

NOTE: I took help of Kleinberg Tardos soln guide to solve this problem