~~PFO~~

Name - Varun Gupta
Roll No - 2023101108
Course - AAD

Problem Set - 2

A1) $G=(V,E)$ with a MST T and shortest Path $\pi(s,t)$ (between two vertices $s,t \in V$)

(a) Weight of Edge is multiplied by a fixed const $c>0$

Let T be previous MST with total edge weight be w.
T' be the new MST with total edge weight be w', before multiplying c to edge weight. Assume T & T' are different. $T \neq T'$

After Multiplying, as T' is MST

$\therefore cw' < cw$ ——— (1)

Before Multiplying T is MST

$\therefore w < w'$ ——— (2)

Multiplying both side by c ($c>0$ so the quality will not change)

$\Rightarrow cw < cw'$ ——— (3)

Seeing (1) & (3) they contraclict each other which contradicts the assumption that $T \neq T'$

$\therefore T=T'$. ~~Hence Proof~~ Hence MST will Remain Same.

for Shortest Path ($\pi(s,t)$):-

On multiplying with c, ~~e~~ $w \longrightarrow cw$ but relative weight between edges does not change. Like if $w > w'$ before multiplying $cw > cw'$ still remain same after Multiplying.

∴, Shortest Path ($\pi(s,t)$) remains same.

(b) Weight of each edge is incremented by a fixed constant $c > 0$

Effect on MST:

Logically speaking, if T was earlier MST. After adding c, T is still spanning tree because it would still be Connected and acyclic. Now let's talk for Minimum Spanning Tree. let T' be new MST (≠ T) with total edge wt's w' before adding. Let there are n vertices so spanning tree will contain all edges (i.e n-1). so total increment wld be c(n-1) for all spanning Trees.

Before Adding,

$$\because w < w' \quad -① \quad [T \leftarrow MST]$$

After Adding,

$$w' + c(n-1) < w + c(n-1) \quad [T' \leftarrow MST]$$

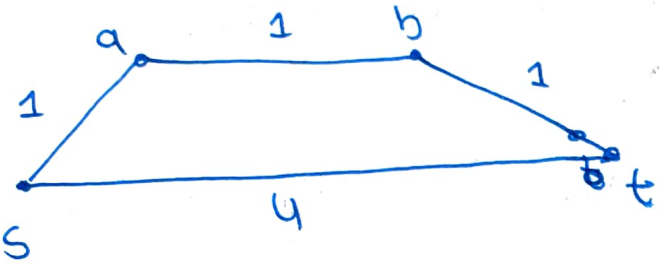
$$w' < w \quad -②$$

① & ② contradicts each other,
which contradicts our assumption that $T \neq T'$.
$\therefore T = T' \Rightarrow$ MST will remain same.

Effect on Shortest Path :

Let's disprove with the help of a counter example.

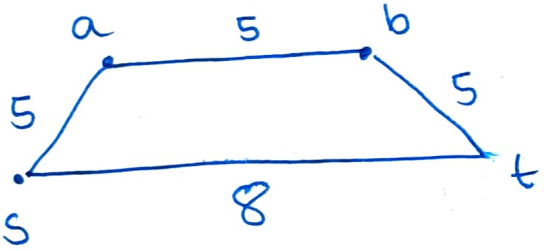
Let Graph, G be



Shortest Path $\pi(s,t)$ be
$s \xrightarrow{1} a \xrightarrow{1} b \xrightarrow{1} t]$
$\pi = 3$.

After incrementing each edge by 4.

New Graph G',



Shortest Path :
$s \longrightarrow t]$
$\pi = 8$

$\therefore$ Shortest Path will change

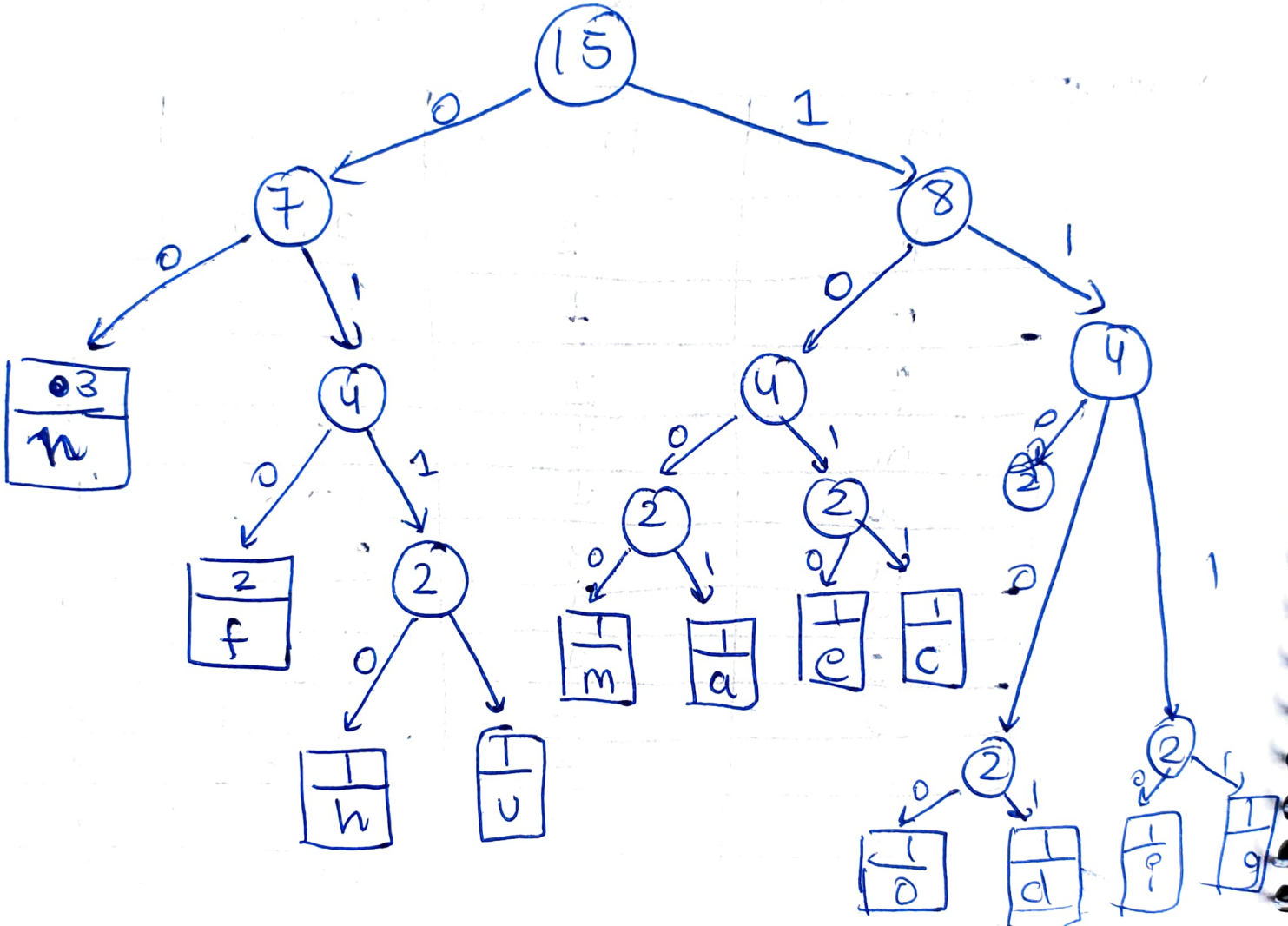
Ans2)

(a) Huffman coding for "huffman encoding"

Frequency

alphabet	frequeng	Encoding	Cost
h	1	0110	4
u	1	0111	4
f	2	010	6
m	1	1000	4
a	1	1001	4
n	3	00	6
e	1	1010	4
c	1	1011	4
o	1	1101	4
d	1	1100	4
i	1	1110	4
g	1	1111	4

huffman tree:



message be :-

0 1100111 010010 1000 1001 00 1010001011 1100
1101 1110001111

Length = number of bits = 52 bits

(b) Average bit length = $\frac{\text{Total number of bits}}{\text{Total no of characters}}$

Avg Bit length = $\frac{52}{15}$ = 3.467 bits

Ans 3) Given: N houses. Wi be cost to build a well at house i_2 and it costs $C_{i,j}$ to build pipe in between houses i & j.

To Find: Find algorithm to find min. ~~time~~ amount of money needed to supply every house with water.

Soln:

MST looks very similar to the problem we need to give connection to every house by ensuring minimum cost. But all edges can't be same because there are of two type $C_{i,j}$ & W_i for i (may be self loop). To solve this let's introduce one more vertex which is connected to vertex i with edge weight as W_i for all $i \in N$.

Now Let special vertex be N^*.

$e(i\,N^*)$ depicts that we are building a well on i^{th} house with cost W_i. To find least pipes cost we connect every i, j with edge of weight $C_{i,j}$.

$$\therefore \quad e(i,j) = C_{ij} \quad \forall\ i,j \in N$$
$$e(i,N^*) = W_i \quad \forall\ i \in N$$

Graph edges.

Now we can use any optimal algorithm for finding MST of this Graph. Total cost of MST would be minimum amount of money needed to supply every house with water.

Now we have a Graph $G(V, E)$ with $N+1$ vertices. All edges are already defined above with their weights.
We will make a Adjacency List for this.

To find MST any optimal algo like Prim's and Kruskal's can be used

Proof that MST is the Required Solution :

→ First Formally define our Graph $G(V, E)$.

$V = \{1, 2, ---N, N^*\}$

Each house rep^n by Node from 1 to N.

Each edge from special node to house represent well is built on that house with cost w_i & each edge b/w house rep^n pipes with cost C_{ij}.

Claim: Any spanning tree is a valid solution.

Any spanning tree including node N^* ensures that every house is either directly connected to node N^* (implying we built a well) or indirectly connected to node N^* via a some another house (means it recieves water through pipe)

Claim: MST is the most optimal solution (least cost)

Q For any valid config such that water is supplied to all houses it can reduce to Spanning Tree. Let MST be T & Required Ans is T' (Span. Tree) ($T \neq T'$)

As T' is with minimum total cost i.e

$(\text{Total cost})_{T'} < (\text{Total cost})_{T}$

But by defⁿ of MST, T' should be MST

∴ T = T' & MST is the solution.

Let's use Prims Algorithm for finding MST

(Assume N* = 0)

```
MinCostToSupply(N, wells, pipes):
    G ← new List[N+1]
    edgeUsed ← 0        Cost ← 0
    For i from 1 to N:
    | G.add((0, i, wells[i]))
    for each (u, v, cost) in pipes:
    | G.add((u, v, cost))
    inMST ← BoolArray[N+1]
    minHeap ← new PriorityQueue
    minHeap.push((0, 0, 0))
    while minHeap not empty & edgeUsed < N:
      (cost', u, v) ← minHeap.pop()
      If inMST[v] then
          continue
      inMST[v] ← True
      Cost += cost'
      edgeUsed ← +1
```

```
for each edge (v, neighbor, edgeCost) in graph
    Connected to v do:
        If not inMST[neighbor]:
            minHeap.push((edgeCost, v, Neighbor))
return Cost
```

Time Complexity = $O(|E| \log(|V|))$

As Replied by TA's gam not prooving correctness of Prim's Algorithm but can be easily done with the help of CUT Property

Ans 4) This is a normal example of Bellmanford algorithm

In cases where we get negative edge weights we use Bellmanford algorithm to find shortest & cycle detection.

Graph $\longrightarrow$ G(V, E) also $n = |V|$ & $e = |E|$

Pseudocode :-

```
for every vertex u in V:
        d[u] ⟵ ∞
d[s] ⟵ 0

for i from 1 to n-1:
    For every edge e ∈ E:
        if (d[e.source] != ∞ & d[e.source]
            + we < d[e.destination]):
```

then

$$d[e.dest] = d[e.source] + we$$

detecting Negative Cycle

for every edge $e \in E$:

if ($d[e.source] != \infty$ & $d[e.source] + we < d[e.dest]$)

then:

$d[e.destination] \longleftarrow -\infty$

Time Complexity : $O(V \times E)$

It is less efficient than Dijikstra Algo for graphs with non-negative weights.

Let's try to optimise it :-

See Dijikstra is efficient but it works for non-negative edge weights. Here all edge weights are negative we can replace $we \longrightarrow -we$. But still issue of negative cycles is there which can't be detected by Dijikstra. We will use the concept of TopoSort. for any vertex in Negative Cycle or reachable from negative cycle we can't make its indegree to 0.

Algo :-

Graph G' ← G // Copying Graph

indeg[v] ← Put no of incoming edge to v.

DFS(s, visited)

for every vertex, v ∈ V but v ∉ visited:

    d[v] ← ∞

Set S ← new Set{ }

for every u ∈ V:

  if indeg[u] = 0 then:

      S ← add(u)

while S is not Empty:

  for u ∈ S:

    S ← remove(u)

    for all e that is edge between (u, v):

      E ← remove(e)

      indeg[v] ← indeg[v] - 1

      if indeg[v] = 0 then:

        S ← add(v)

for v ∈ V and v ∉ S:

    d[v] ← -∞

// Run Dijikstra (Standard algo like in Book).

Dijikstra(G', S)

// Find other distances.

Approach-

1) First, we will find the indegree of all vertices as the no of edges incident on a vertex.
2) By Running DFS now from Strat node we can find unreachable vertex and mark their distance as ∞.
3) Now we will do Topo Sort :-
   * Store all vertices with indegree 0 in visited set. (S)
   * for each vertex (u) in this visited set (S), we will remove its edge with any vertex (v) in V-S & reduce its indegree. After this we remove this vertex, u.
   * We will continue this untill Visitedset (S) does not become empty.
4) After this all vertex which are not in the Visited set, are marked with distance as $-\infty$.
5) These are the vertices which are in negative cycle or reachable from Negative Cycles.
6) After that we can restore the edges of all the vertices in the Visited set. And on this new leftover graph we can run Dijkstra algorithm to compute their minimum distances.

Runtime Analysis : $O(E \log V)$

TopoSort & DFS - $O(E+V)$

Dijikstra - $O(E \log V)$

Overall - $O(E+V+E \log V)$

$= O(E \log V)$

which is way better than $O(E \cdot V)$ of Bellman ford.