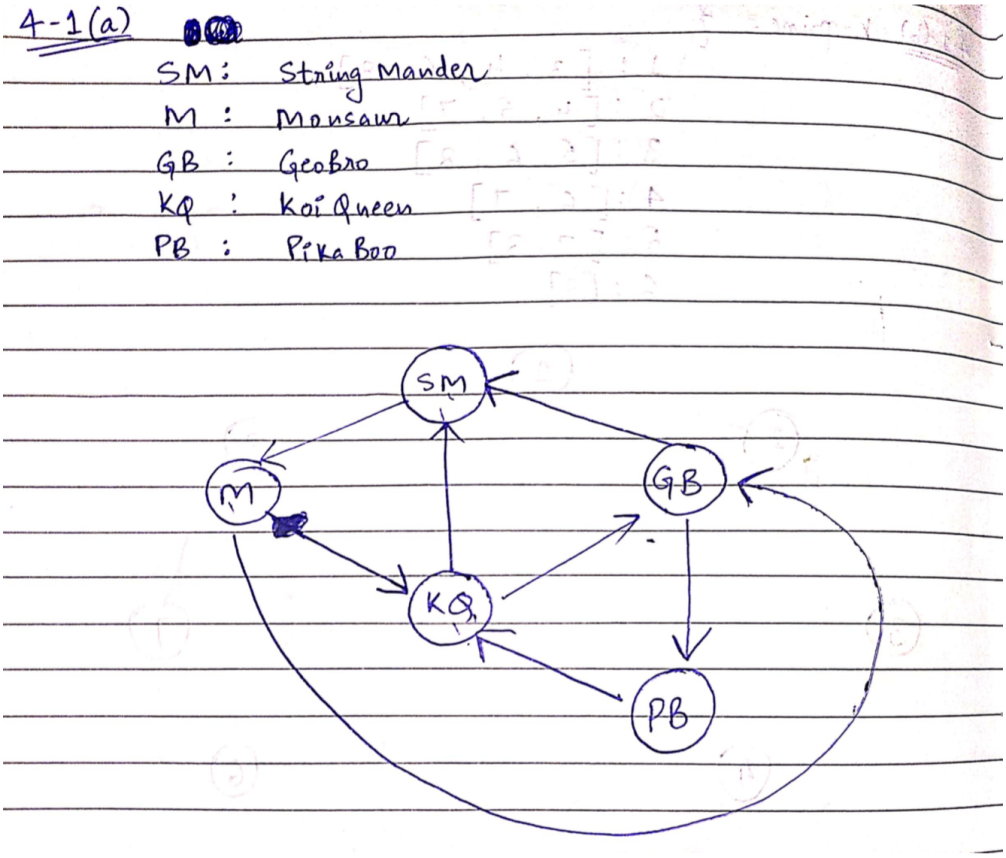
Problem Set 4

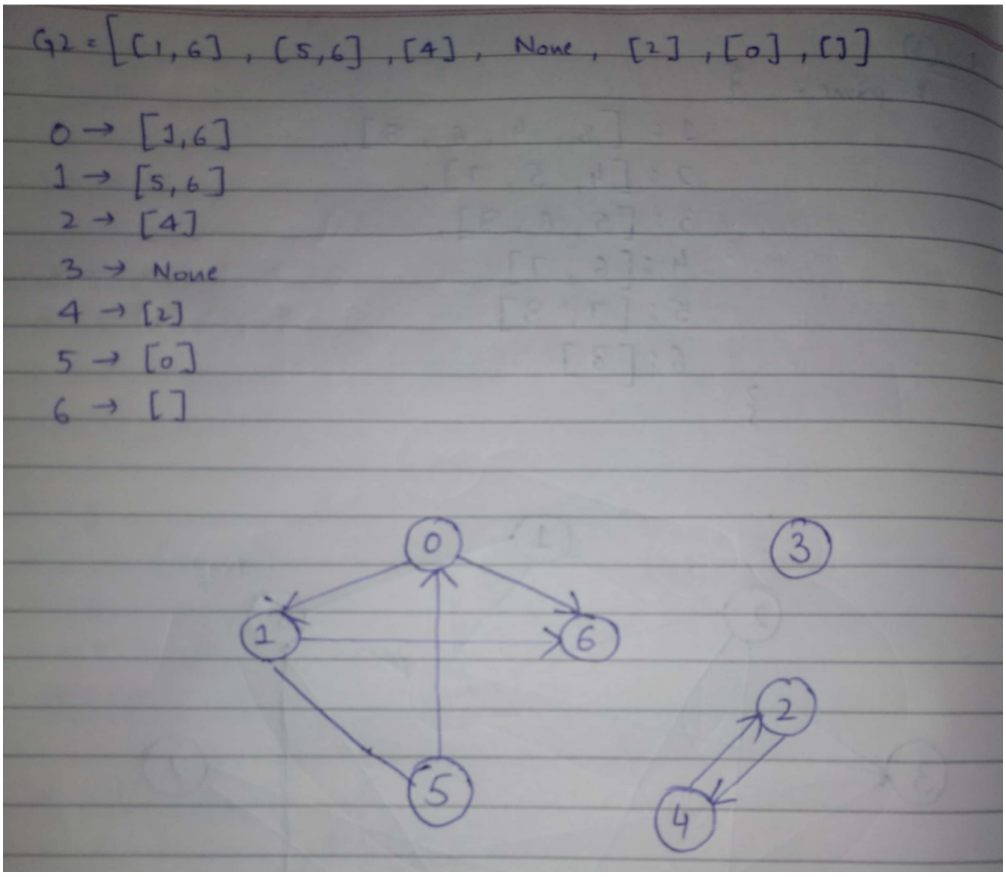
Name: Neeraj Pandey

Collaborators: Name1, Name2

Problem 4-1.

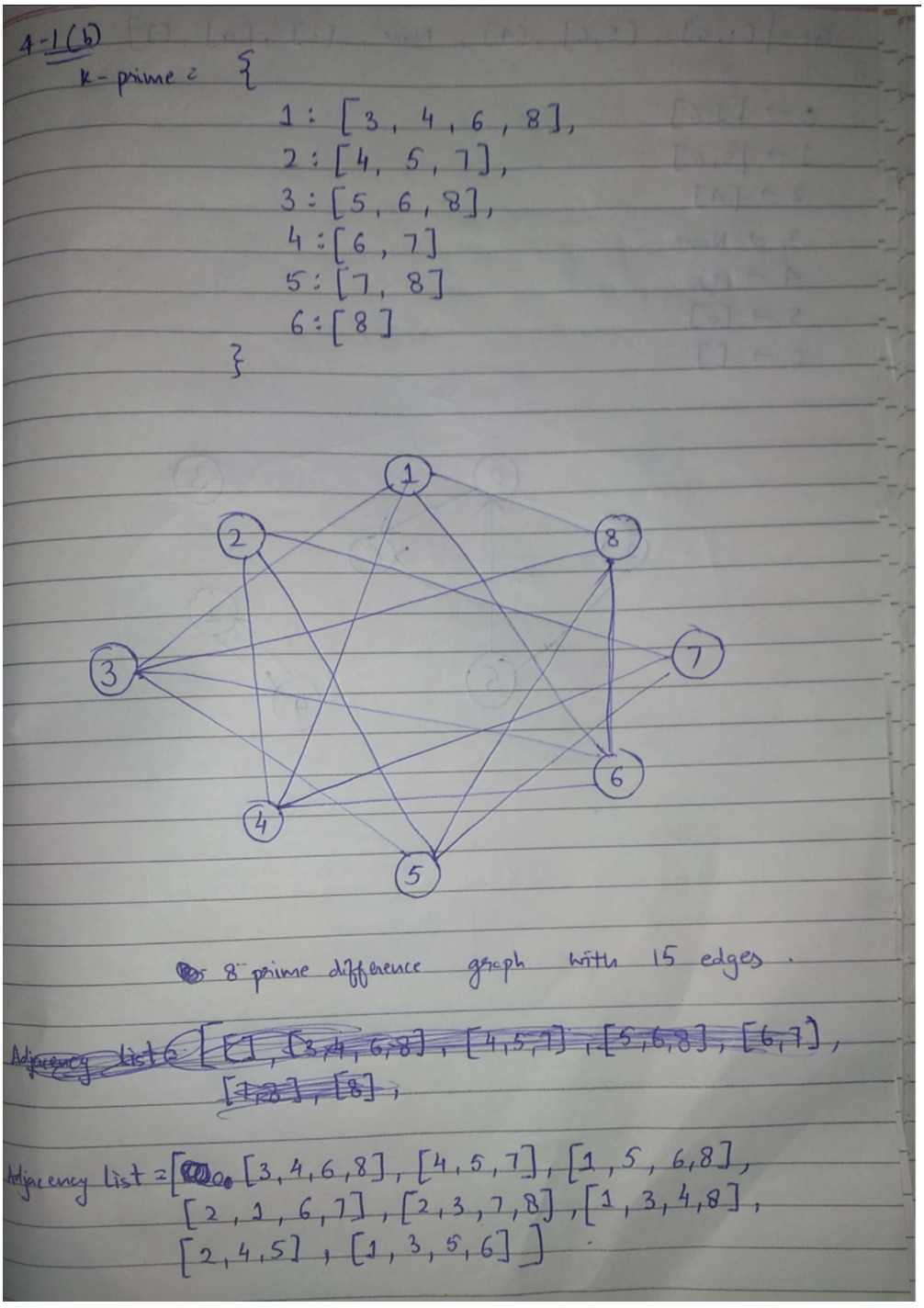
(a) Solution



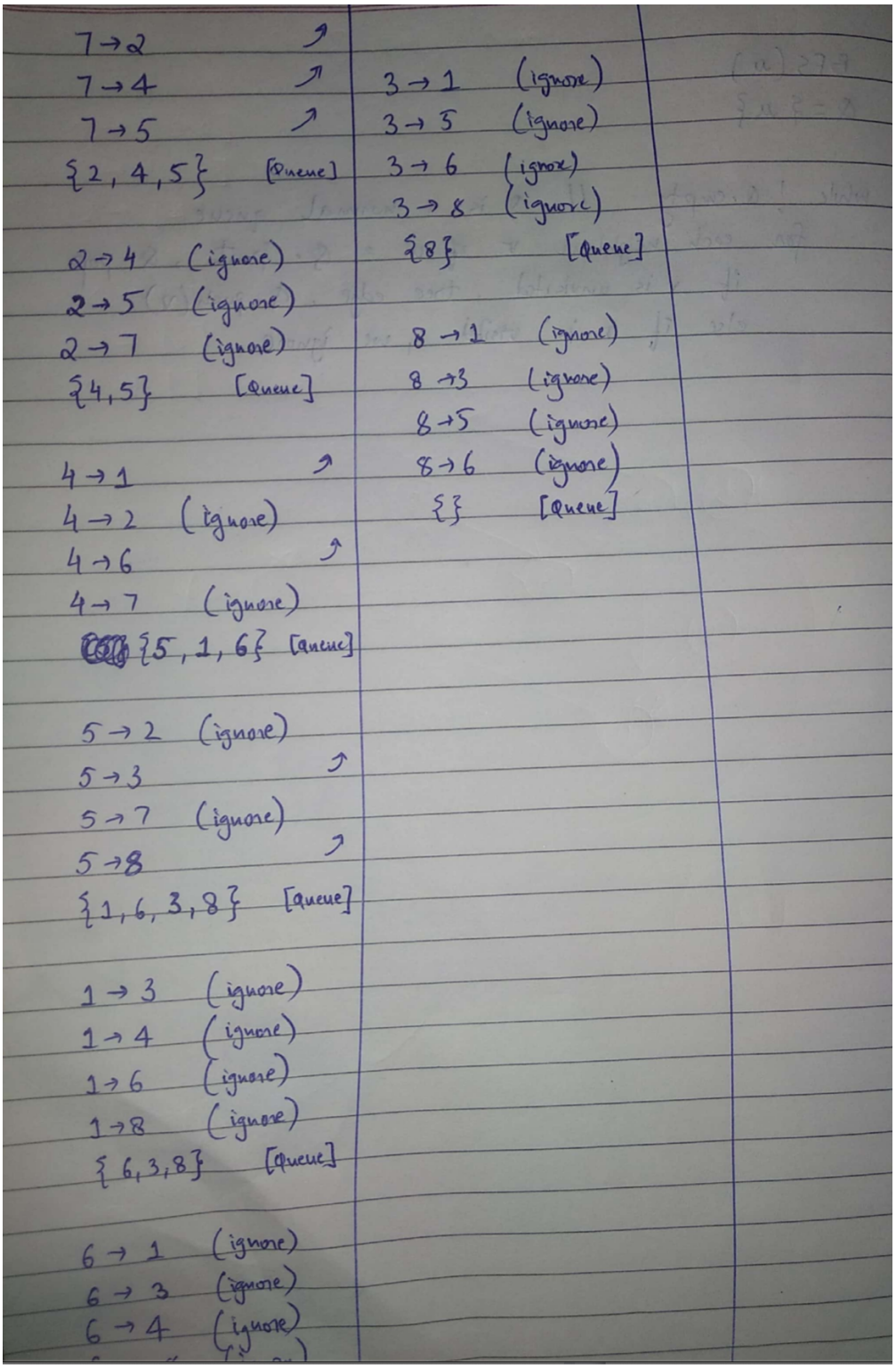


(b) Adjacency List:

$$[[3, 4, 6, 8], [4, 5, 7], [1, 5, 6, 8], [2, 1, 6, 7], [2, 3, 7, 8], [1, 3, 4, 8], [2, 4, 5], [1, 3, 5, 6]]$$



(c) Performing BFS on the above: In the trees, the child nodes are visited from left to right.

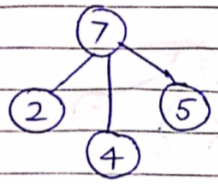

7→2 ↗
7→4 ↗
7→5 ↗
{2, 4, 5} [Queue]
2→4 (ignore)
2→5 (ignore)
2→7 (ignore)
{4, 5} [Queue]
4→1 ↗
4→2 (ignore)
4→6 ↗
4→7 (ignore)
{5, 1, 6} [Queue]
5→2 (ignore)
5→3 ↗
5→7 (ignore)
5→8 ↗
{1, 6, 3, 8} [Queue]
1→3 (ignore)
1→4 (ignore)
1→6 (ignore)
1→8 (ignore)
{6, 3, 8} [Queue]
6→1 (ignore)
6→3 (ignore)
6→4 (ignore)
3→1 (ignore)
3→5 (ignore)
3→6 (ignore)
3→8 (ignore)
{8} [Queue]
8→1 (ignore)
8→3 (ignore)
8→5 (ignore)
8→6 (ignore)
{} [Queue]

Q = ~~invalid~~ Queue with initial values as 7

⇒ Q = [7]

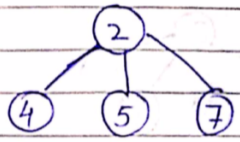
classmate Date Page

Step 1.



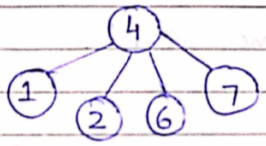
Q = [2, 4, 5]

Step 2.



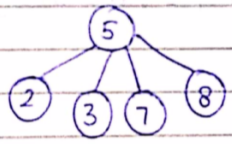
Q = [4, 5]

Step 3.



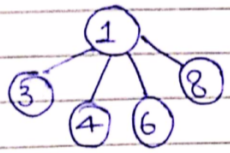
Q = [5, 1, 6]

Step 4



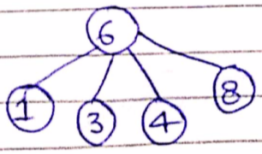
Q = [1, 6, 3, 8]

Step 5.

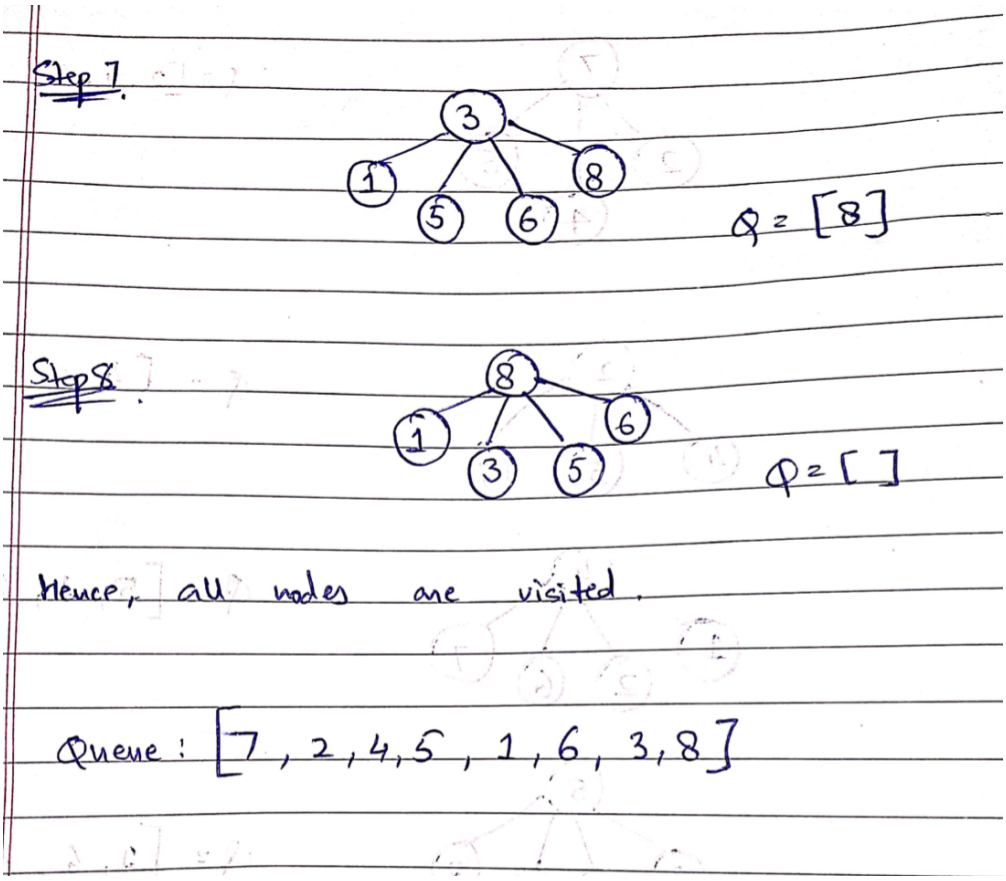


Q = [6, 3, 8]

Step 6



Q = [3, 8]


Step 7.
3
1
8
5
6
Q = [8]
Step 8.
8
1
6
3
5
Q = [ ]
Hence, all nodes are visited.
Queue : [7, 2, 4, 5, 1, 6, 3, 8]

(d)

Problem 4-2.

(a) For finding the diameter of a graph G = (V,E) in the time complexity of O($|V||E|$), we can do this by either using DFS or BFS. Here, we will be using BFS to compute the diamater. To do so, to all the nodes, then perform BFS from each node and calculate the maximum distances from a given node(u) to the other nodes that are reachable from the node (u). This can be done by creating a 2-D array of distances from each vertex to the other and compute the eccentricity. The maximum distance calculated among those will be the diameter of the graph (we can also say, going through all the eccentricities . Each time BFS is used takes the time complexity of O(V) but we have to do this for each vertex, that will be O(E). In total, this will take the time complexity of O($|V||E|$) in worst case.

(b) From each vertex on the undirected graph we travel to every other vertex across all the edges of which minimum distance between them is recorded. which gives us the eccentricity of each vertex.
We take the maximum of these eccentricities which gives us the diameter.
The time-complexity of this algorithm is O(V*E)

(c) We have to prove that the sum of (the edges in diameter) - (the edges in the graph) is lesser than the diameter itself
Thus we have to prove D/2 $\leq$ e(v) $\leq$ D for all v $\in$ V where D = max(e(u)| u $\in$ V)

$\therefore$ e(v) $\leq$ D for all v $\in$ V.

Now we have to prove that

$$D/2 \leq e(v) \text{for all v} \in V \text{for all v} \in V_{\text{for all v} \in V_{\text{for all v} \in V}}$$

This can be proved by contradiction method as follows:

If some v_1 exists such that D/2 $\geq$ e(v_1), then for all dist(v_1, u) $\leq$ D/2 for all u $\in$ V

If v_2, v_3 are the vertices at the endpoints of the diameter, then (v_2, v_3) = D

But,

$$(v_2, v_3) \leq (v_2, v_1) + (v_1, v_3)$$

,

$$(v_2, v_1) \leq D/2 \text{ and } (v_1, v_3) < D/2$$

So

$$(v_2, v_1) + (v_1, v_3) < D/2 + D/2$$

$$(v_2, v_3) < D$$

But the above one is a contradiction
e(u) $\leq$ D(G) as e(u) is the shortest path thus 2e(u) $\leq$ 2D(G) Minimum number of edges required is n-1 and so the maximum number of edges in a simple graph is $n*(n-1)/2$ where n=number of vertices
So if $E = n - 1$ then the graph is a straight line and $|E|$ = $n - 1$ = $diameter D(G)$.

(d)

Problem 4-3.

(a) For Weff to be approachable to by every user in LinkedOUT network, Weff has to connect to at least one node of every disconnected sub graphs, i.e- If becomes friend(acquaint) with anyone from a sub group, he can approach all of the people from that sequence. This can be done using a BFS or a DFS algorithm. To perform this , first create an adjacency matrix of all the nodes and create a counter element with initial value as 0. Now, choose any unlabeled node as u_0, mark it explored and perform BFS to find out all the nodes that are reachable from that node and mark them explored too. Once, all the nodes reachable to the initial node u_0 are explored, then increment the counter element and start over by picking another unlabeled node which is not reachable to any of the previous nodes which were marked explored (which means this node belongs to another disconnected graph). Now, again perform the BFS implementation on all the nodes reachable to this unexplored node. Eventually, all the nodes in the graph will be explored and we will have a final counter element value which will show the total number of disconnected graphs. Therfore, this counter element value is the minimum number of users with whom Weff needs to become acquainted in order to be approachable by every user on LinkedOUT. Time complexity to perform this operation will be O(V*E) for V vertices and E edges or O(m*n) for the matrix of size m x n.

(b)

(c)

(d)

Problem 4-4.

(a) We can perform BFS here. To do so, take the initial node that is Boston; c_1 and traverse through all the destinations(nodes) that are reachable to that node, unless all nodes are explored. While doing this, keep a count of the distance travelled each route, and the minimum distance at the end will cost minimum number of dollars as each route costs $\$1$. Here, we know that initial and final vertex, and that each route costs $\$1$, so we don't have to initialize the distance and predecessor for each vertex. So, we just examine the edges incident on a vertex when we visit it. And here the graph is a dense graph as there are only a few known vertices as destinations but there can be many routes (edges), since it is totally connected. So, the number of edges will dominate the recursive calls and therefore time complexity will be O(n) for n edges ($n >> V(vertices)$).

(b) Here, Tim has to pay full price for the routes. To perform this, we first take take all the nodes (the destinations around the country) and make a edge list of all the adjusts. In that lists, for each pair u, v, we will relax the pairs. Here relaxation increases the accuracy of the distance of the given pair of vertices. It works by continuously shortening the calculated distance between the vertices that distance with other known distances. Also, initialize the first vertex (here, the Boston node) with 0 distance and all the other node as infinity. The relaxation equation works in the following way:

$$if(Distance[u] + weight[u, v] < Distance[v]$$

$$Distance[v] = Distance[u] + weight[u, v]$$

Here weight is the price of a specific route. We need to relax the pair of vertices (n-1) times. Now, use the relaxation equation and update the distance of each vertex and repeat it for (n-1) times. At the end, each node will have a shortest distance (here, lowest price of the route as K). In best case, the time complexity for the given algorithm will be O(Kn) as it take (n-1) relaxations, where n is the number of bus routes and k is the price.

(c) Here, cost conditions are different now. Tim has to pay student discount when going to the state but has to pay full price when he has to leave. So, undirect edges get replaced with 2 directed edges of different cost (say Boston to Miami, and Miami and Boston). Now, from Boston to Miami student discount applies and from Miami to Boston full price ticket applies. To compute this, we will use BFS while going from Boston to Miami and Bellman Ford Algorithm to come back from Miami to Boston.

For Boston to Miami, we do BFS and take the initial node that is Boston; c_1 and traverse through all the destinations(nodes) that are reachable to that node, unless all nodes are explored. While doing this, keep a count of the distance travelled each route, and the minimum distance at the end will cost minimum number of dollars as

each route costs $\$1$. Here, we know that initial and final vertex, and that each route costs $\$1$, so we don't have to initialize the distance and predecessor for each vertex. So, we just examine the edges incident on a vertex when we visit it. And here the graph is a dense graph as there are only a few known vertices as destinations but there can be many routes (edges), since it is totally connected. So, the number of edges will dominate the recursive calls and therefore time complexity will be O(n) for n edges ($n >> V(vertices)$).

From Miami to Boston, we do Bellman Ford algorith. To perform this, we first take take all the nodes (the destinations around the country) and make a edge list of all the adjusts. In that lists, for each pair u, v, we will relax the pairs. Here relaxation increases the accuracy of the distance of the given pair of vertices. It works by continuously shortening the calculated distance between the vertices that distance with other known distances. Also, initialize the first vertex (here, the Boston node) with 0 distance and all the other node as infinity. The relaxation equation works in the following way:

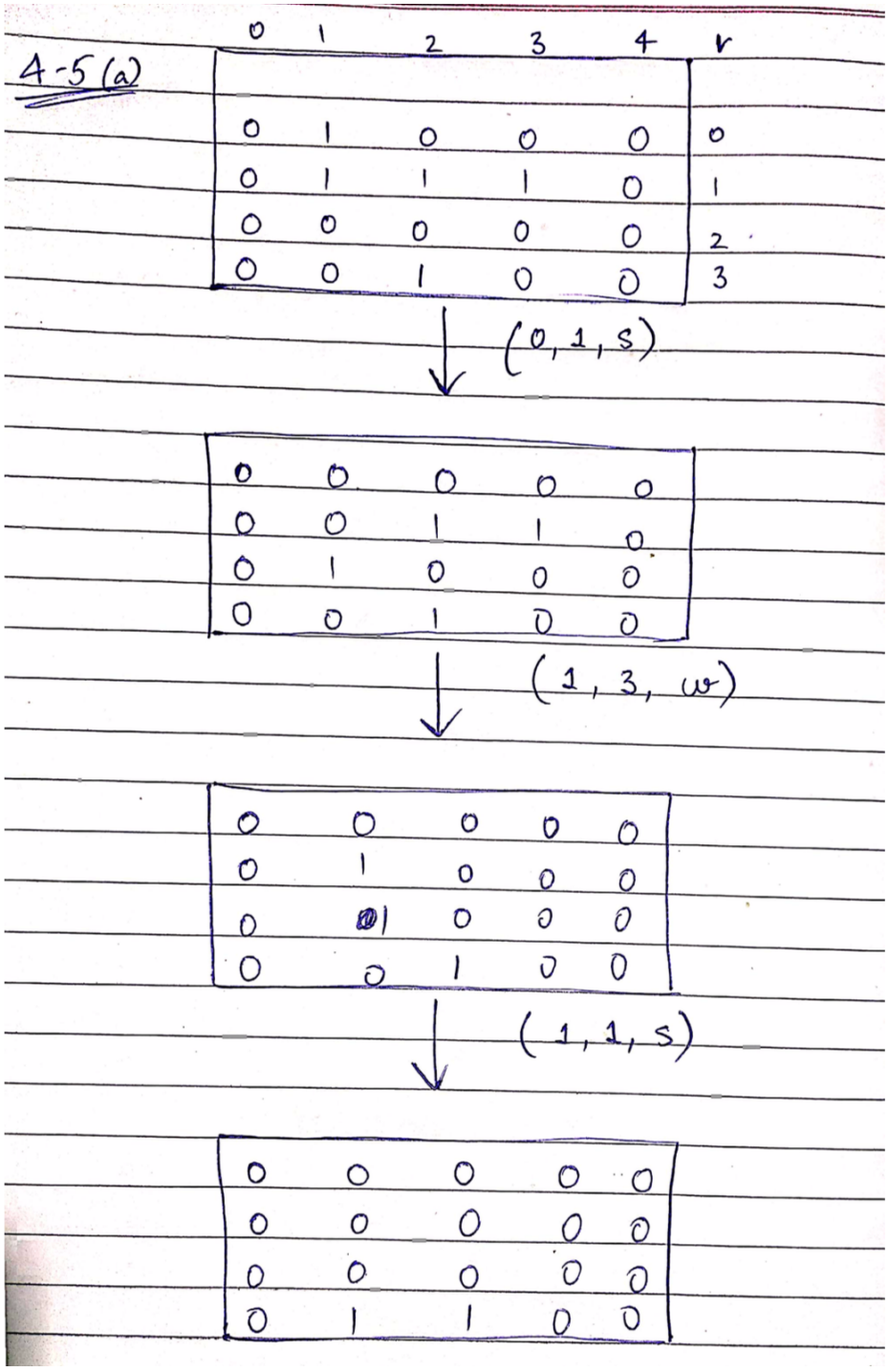
$$if(Distance[u] + weight[u, v] < Distance[v]$$
$$Distance[v] = Distance[u] + weight[u, v]$$

Here weight is the price of a specific route. We need to relax the pair of vertices (n-1) times. Now, use the relaxation equation and update the distance of each vertex and repeat it for (n-1) times. At the end, each node will have a shortest distance (here, lowest price of the route as K). In best case, the time complexity for the given algorithm will be O(Kn) as it take (n-1) relaxations, where n is the number of bus routes and k is the price.

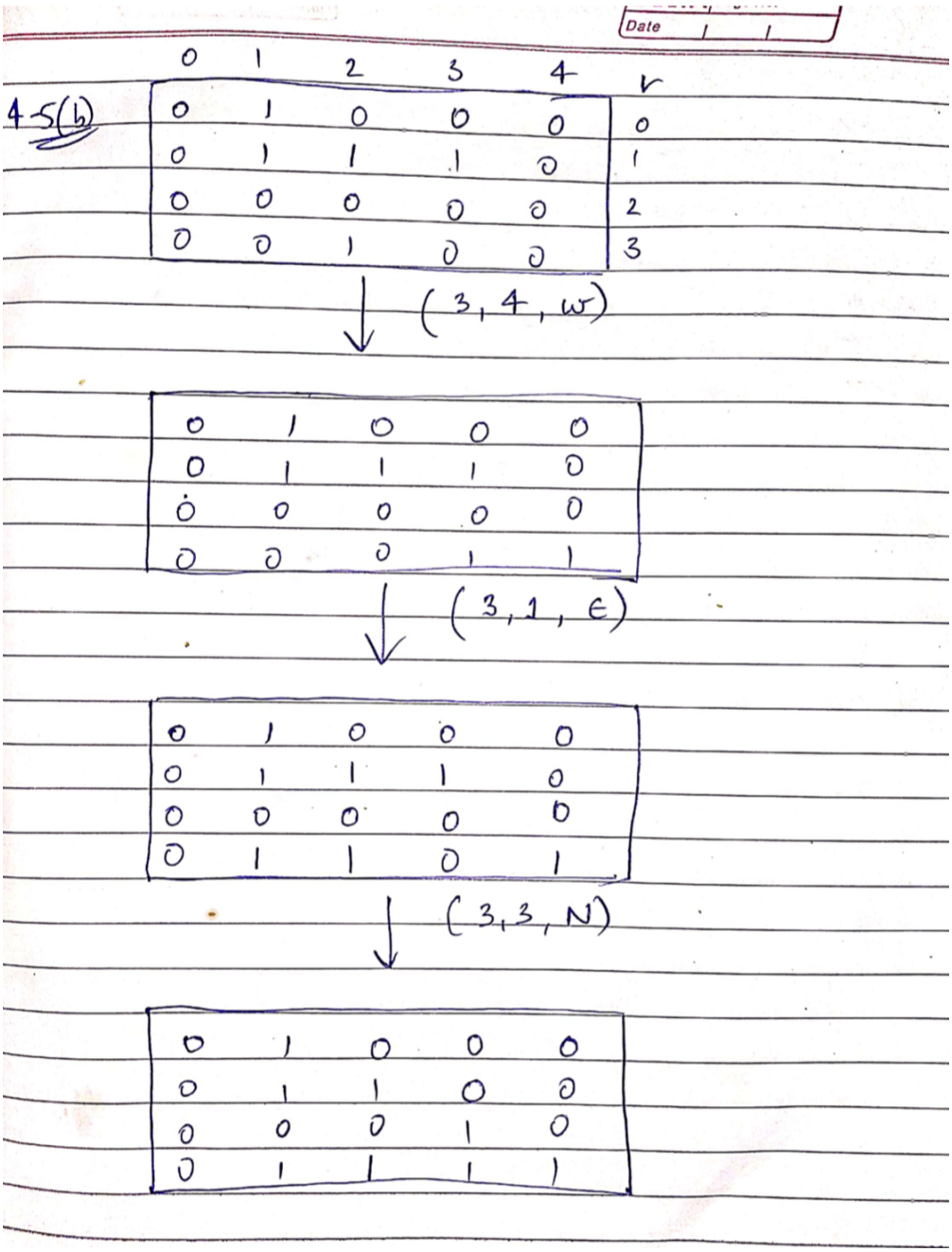
Finally, the total time complexity will be O(n) + O(kn). As $O(Kn) > O(n)$, O(n) will be rejected, and final complexity will be $O(kn)$.

Problem 4-5.

(a) Solution:



(b) Solution:


Date
4.5(b)
0 1 2 3 4 r
0 1 0 0 0 0
0 1 1 1 0 1
0 0 0 0 0 2
0 0 1 0 0 3
(3, 4, w)
0 1 0 0 0
0 1 1 1 0
0 0 0 0 0
0 0 0 1 1
(3, 1, ϵ)
0 1 0 0 0
0 1 1 1 0
0 0 0 0 0
0 1 1 0 1
(3, 3, N)
0 1 0 0 0
0 1 1 0 0
0 0 0 1 0
0 1 1 1 1

(d)

(e)