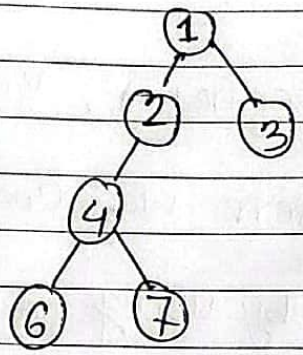
Summer - 2022

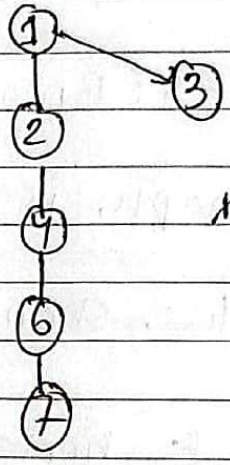
1. a)

BFS → 1,2,4,6 | 1247 | 13

DFS → 12467, 13.

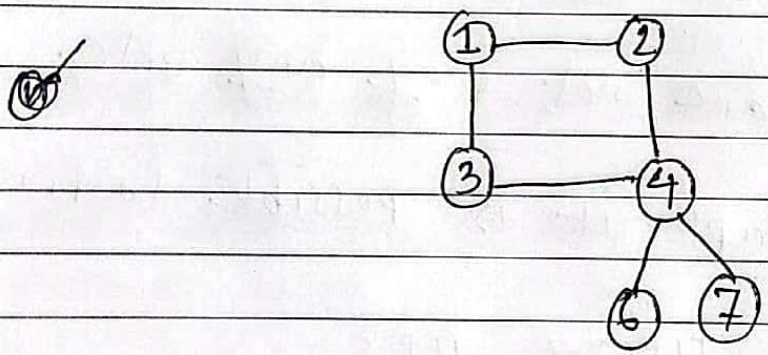


BFS



DFS.

Graph will be:



There can be multiple graph based on this information. Here is one of them.

b) If a graph is unweighted then it is possible to find shortest path by using BFS (Breadth First search), But if the graph is weighted then we can not find the shortest path by using BFS.

But here in this graph though it is a weighted graph. But all weight are equal. so for find shortest path the weight are not play any role. so for this graph it is possible to find shortest path by using BFS.

[DFS can be used But BFS is commonly used, unweighted এ DFS ব্যবহার হয়। But efficient না। BFS বেশি]

b) Here is the pseudocode for union sets operation and find set operations. using path compression and union by rank heuristics.

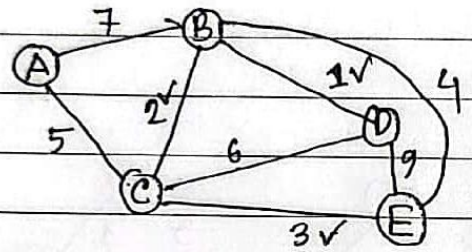
for union set operation:

```
function union (x, y)
    rootx = find(x)
    roty = find(y)
    if (rootx == roty)
        return
    if [rank(rootx) < rank(roty)]
        parent[rootx) == roty.
    else
        parent [root y] == rootx.
    if (rank[rootx) == rank[rooty))
        rank (rootx == rank.[rootx) +1
```

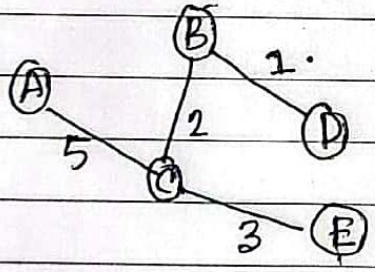
```
function find(x)
    if(parent[x] != x)
        parent[x] = find(parent[x])
    return parent[x]
```

3. a) we know that kruskal's algorithm's has timcomplexity is O(nlogn). and Bubble sort time complexity is O(n²) that are told in the question.
So, The runtime of kruskal's algorithm assuming edge are sorted using Bubble sort would be O(n²)

b)

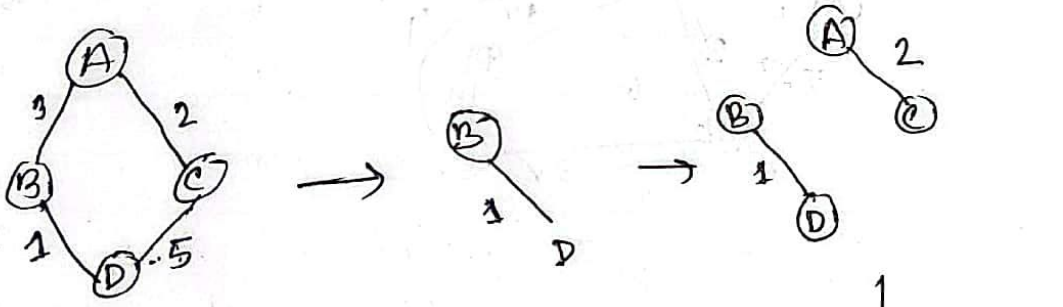


9>6>1
9>7>6>4>2
9>7>6>5>4>3
9>7>6>5>4 → it was select. Select (cycle)



"Minimum spanning Tree using Prim's Algorithm"

c) Kruskal's algorithm is a greedy algorithm that finds the MST (Minimum Spanning Trees) From a graph. It works by starting with each node its own tree, then select the edge with the lowest cost and connect so on. This process repeats & untill all node are connected froming a sigle tree. so, at first it makes ~~Fro~~ Forest of tree.



"Graph"

"MST using Kruskal's"

2. ⓒ

④ Find_set(3) = 1)

compare with parent of 3 is enequal on equal.
if it is enequal then find the parent of the 3's parent. find(2) = 1,

find(1) = 1 , return 1.

so, findset(3) = 1

findset(6) = 1.

find(6) = 3,

find(3) = 2

find(2) = 1

find(1) = 1 return 1

findset(5) = 1,

find(5) = 3

find(3) = 2

find(2) = 1

find(1) = 1 return 1

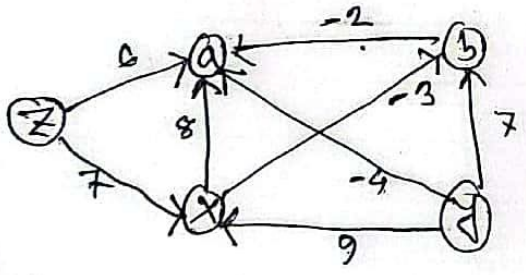
find(3) = 1.

4.

a) A graph contains vertices { A, B, C, D, E, F, G } and shortest path from A to B is A → E → C → D → F → G → B, using this information it is possible to find the shortest path from E to F.

Because, E → C → D → F is included for finding A to B shortest path. So, it A to B is the shortest path. So E to F also the shortest path.

b) Here for finding shortest path we need to use Bellman Ford algorithm. z is source vertex.



i	z	a	b	x	y
z	0	∞	∞	∞	∞
a	0	6	∞	7	∞
b x	0	6	∞	∞	∞
b					
y					

List

z	a	x	b	y
0	∞	∞	∞	∞
0	6	7	∞	∞
0	6	7	∞	∞
0	6	7	9	∞
0	2	7	4	∞
0	2	7	4	2

2nd:

	z	a	x	b	y
z	0	2	7	4	∞
a	0	2	7	4	∞
x	0	2	7	4	∞
b	0	2	7	4	∞
y	0	2	7	4	∞
	0	2	7	4	∞

here, y is unreachable from z.

The Shortest path tree satisfies the optimal substructure Property because, if we have a shortest path from the source vertex v and we also have a shortest path from v to another vertex u. then the concatenation of these two paths will also be the shortest path from the source vertex to vertex u. This means that the optimal solution for a sub problem can be obtained by combining the optimal solution of its sub-supproblems. thus the shortest path tree satisfy the optimal substructure property.

5. a) Benefits of a good hash function is Time-complexity O(1).
A good hash function allows for fast and afficient lookup, insertion and deletion of data in a hash table.

b) A linked list is the best data structure for implementing chaining Hash Table as it offers best time complexity, for two operation, insertion, search and deletion.
Insertion in to a linked list time complexity is O(1), then searching is O(n) and Deletion from a linked list is O(n). and for Binary search Tree two time complexity for a Insertion, Search and Deletion is O(logn), o(logn), O(logn).
But in generally. Linked list is good

for this operation because it is faster.

But if we consider only time complexity of Insertion, Search and Deletion.

then Binary Search tree would be better solution.

$$O(\log n) < O(n)$$

6. a) In string matching Some times it can be happen that, numerical value matched but the string does not match. And this is spurious hit. To reduce spurious hit. we use Rolling hash function.

Text = a b c a a b b c a = 1, b = 2 .... j = 10

string = b b c = $2 \times 10^2 + 2 \times 10^1 + 3 \times 10^0$

$= 200 + 20 + 3$

$= 223.$

and it used a Prime number for mod.

suppose $p = 13.$

$223 \% 13 = 2$

and for next window calculation it used,

$\{[$Prevs window value $- 10^2 \times$ (prev character value)$] \times 10 +$ nex character value$\} \% 4.$

using this function we calculate and find the substring.

b) Text = fcdabdea

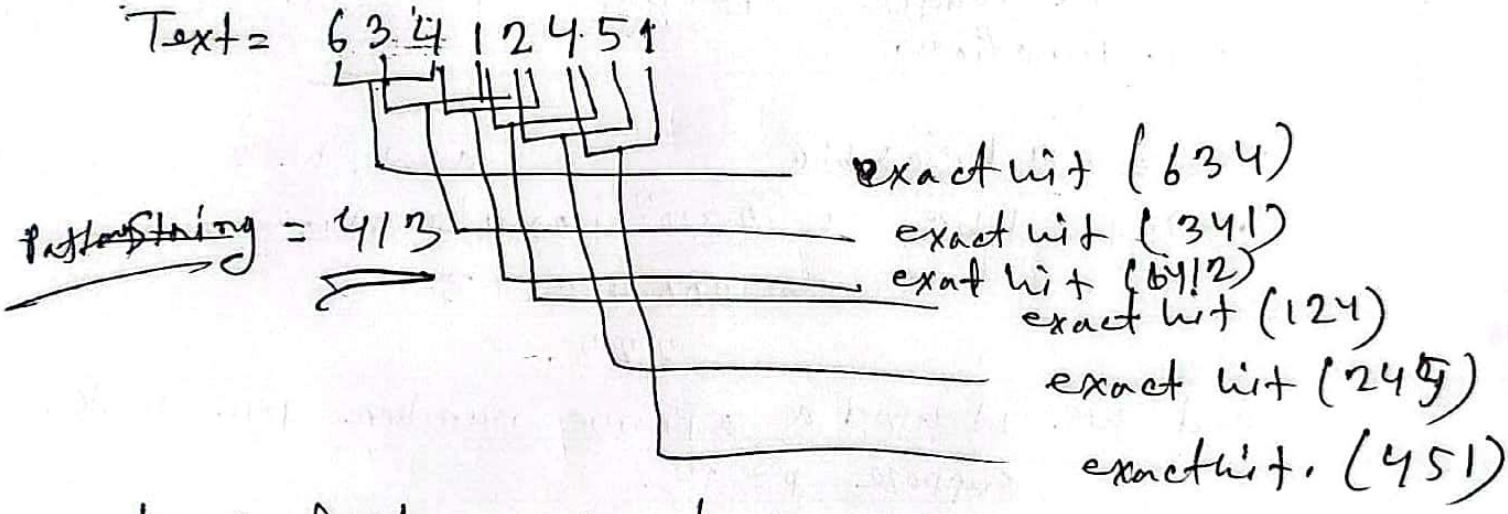
assume that P = 13.

Pattern = dac.

= 413

and using next window function is :

((Previous window value − 10^2 × Prev char value) × 10 + next char value) Mode 13



here in the text string we can not find the Pattern.

So Pattern does not exists in the string.

7. a)

i) NP-Complete: A Problem is NP-Complete if it is in the class NP (a problem that can be solved in polynomial time by a non deterministic algorithm) and every other problem in NP can be reduced to it in polynomial time.

ii) P: A Problem is that can be solved in polynomial time by a deterministic ~~algo~~ algorithm that is polynomial Problem.

iii) NP-hand: A Problem is NP-hand if it is at least as hand as the handest Problems in NP. In other words, if an NP-hand problem can be solved in problem in polynomial time, then all problems in NP can also be solved in polynomial time.