16/03/2024

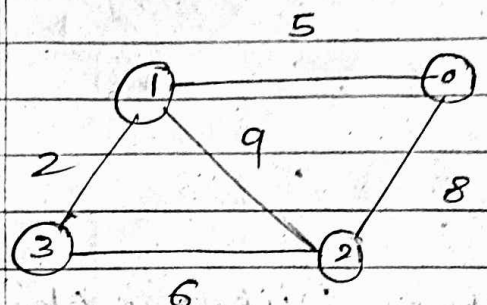
Day 19 OF DSA.

Tasks:-

Check Box — Graphs.

- [x] Shortest Path Algorithms) Dijkstra's Algo. Bellman-Ford Algo
- [x] Minimum Spanning Tree)- Prim's Algo & Kruskal's Algo
- [x] Implement Dijkstra's algorithm For Finding shortest paths.
- [x] Implement Prim's & Kruskal's algorithms For finding minimum spanning tree

Dijkstra's shortest Path :-

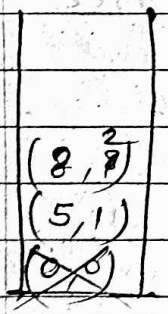


adj list :-

0 → [1,5], [2,8]
1 → [0,5], [2,9], [3,2]
2 → [0,8], [1,9] [3,6]
3 → [1,2], [2,6]

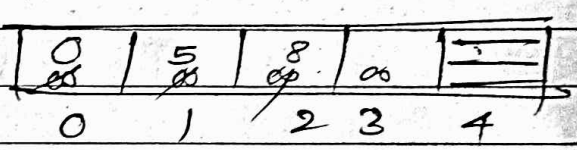
0 → 0 = 0
0 → 1 = 5
0 → 2 = 8
0 → 3 = 7

* Approach :-



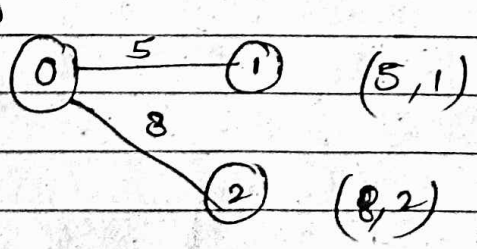
← min

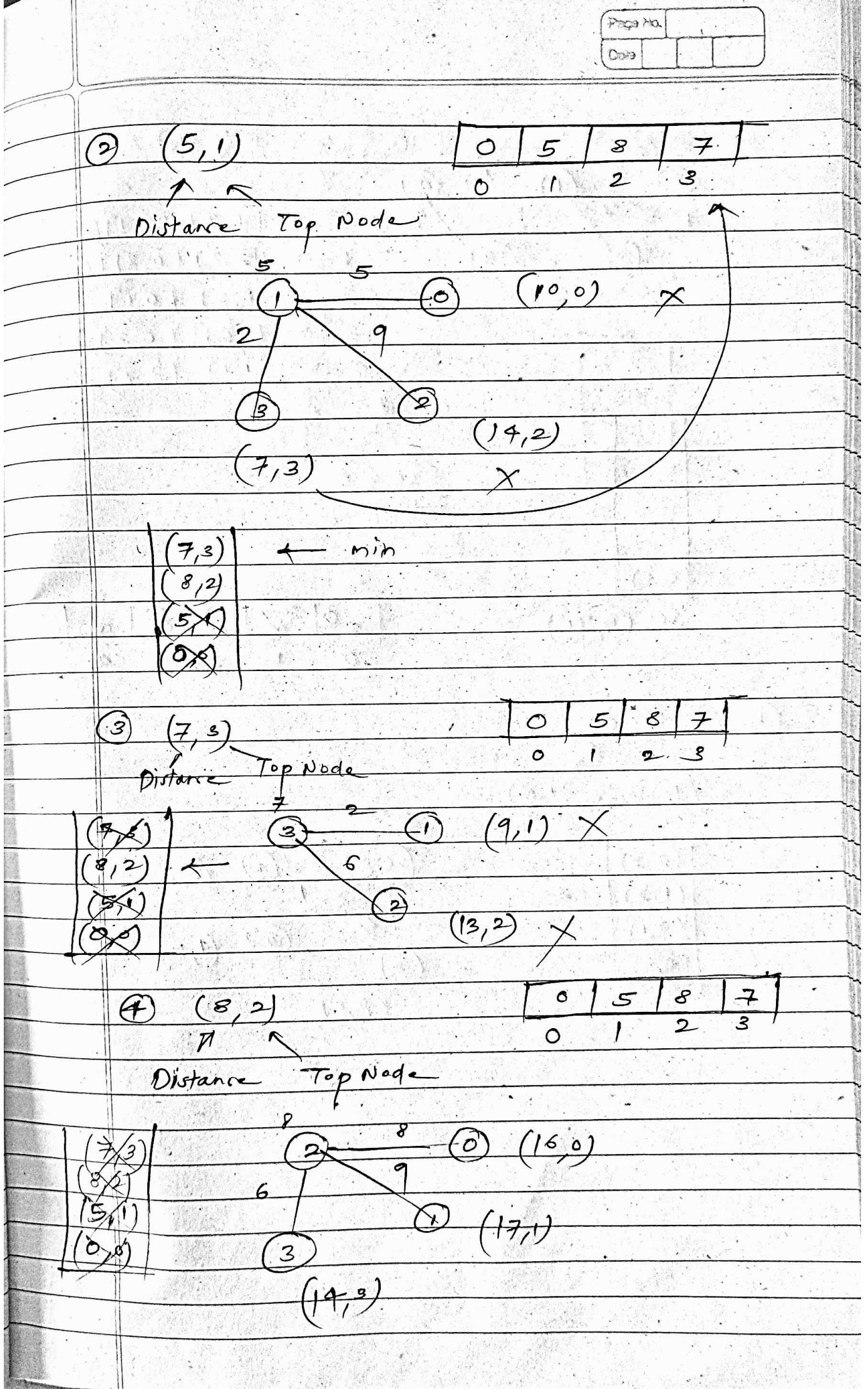
set (d, t.N)

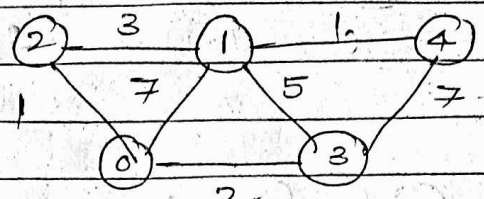


(1) (0, 0)

distance Top Node

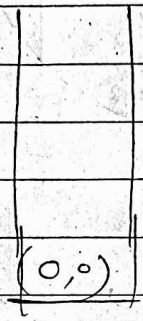



Page No.
Date
② (5,1)
Distance
Top Node
0 5 8 7
0 1 2 3
5
5
2
9
(10,0) X
(14,2)
(7,3) X
(7,3) ← min
(8,2)
(5,1)
(0,0)
③ (7,3)
Distance
Top Node
0 5 8 7
0 1 2 3
7
2
6
(9,1) X
(13,2) X
(8,2) ←
④ (8,2)
Distance
Top Node
0 5 8 7
0 1 2 3
8
8
6
9
(16,0)
(17,1)
(14,3)



adj list.

0 → [1,7], [2,1], [3,2]
1 → [2,3], [0,7], [3,5], [4,1]
2 → [1,3], [0,1]
3 → [1,5], [0,2], [4,7]
4 → [1,1], [3,7]



set (D, TN)

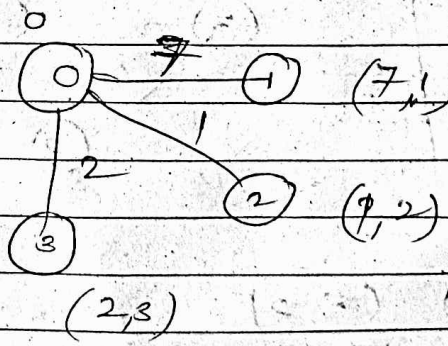
src = 0

∞	0	7	∞	∞	2	∞
0		1		2	3	4

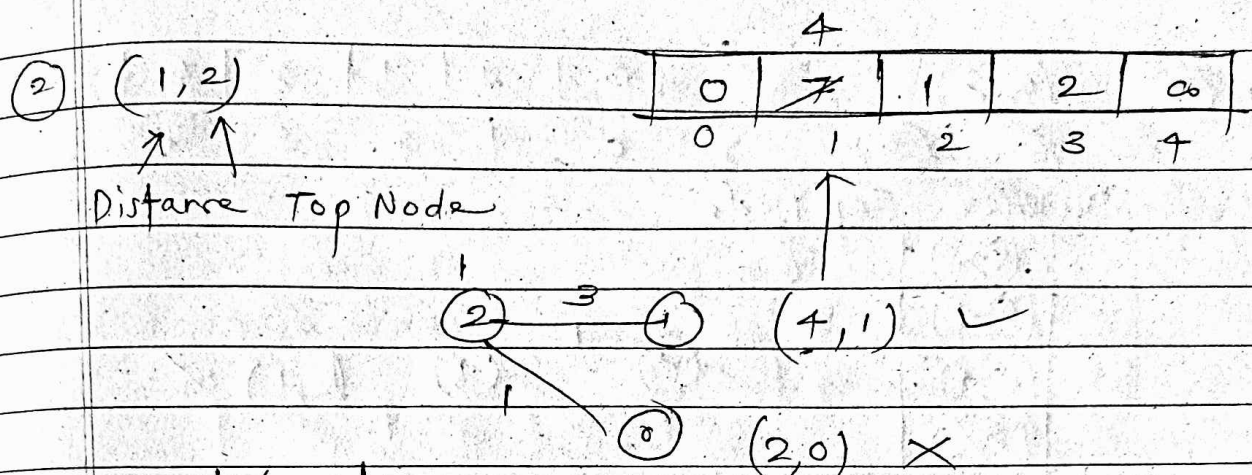
(1) (0,0)

(Distance T.N)

(2,3)
(1,2)
(7,1)
~~(0,0)~~



Page No.
Date

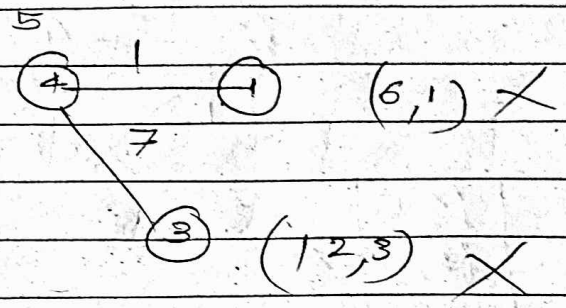

(2) (1,2)
Distance Top Node
4
0 | ∞ | 1 | 2 | ∞
0 1 2 3 4
1
2
3
1
1
(4,1) ✓
(2,0) ×
(4,1)
(2,3) ← min
(1,2)
(7,1)
(0,0)
(3) (2,3)
Distance Top Node
0 | 4 | 1 | 2 | 9
0 1 2 3 4
(0,4)
(7,1)
(2,3)
(1,2)
(7,1)
(0,0)
2
5
2
(9,4) ✓
(7,0) ×
(7,1) ×
(4) (4,1)
Distance Top Node
5
0 | 4 | 1 | 2 | 9
0 1 2 3 4
(5,4)
(9,4)
(4,1)
(2,0)
(1,2)
(7,1)
(0,0)
4
3
7
5
1
(7,2) ×
(5,4) ✓
(11,0) ×
(9,3) ×

(5) (5,4)

Distance Top Node

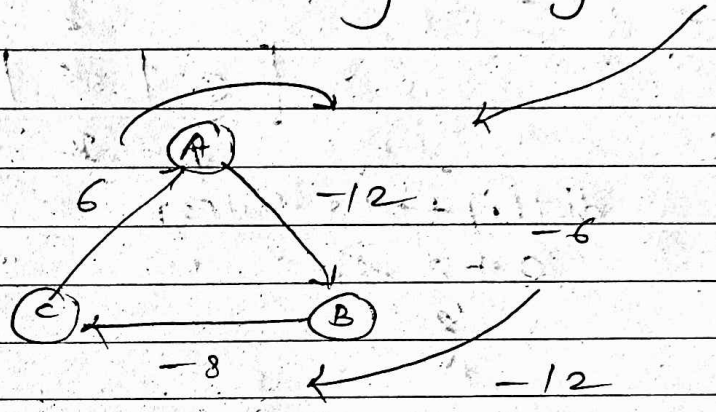
0	4	1	2	5
0	1	2	3	4

(5,4)
(9,4)
(4,1)
(2,3)
(1,2)
(3,1)
(0,0)

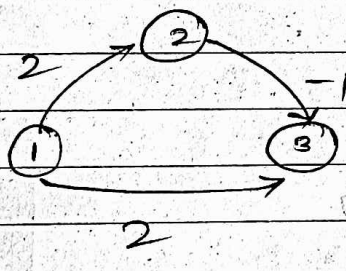


* Bellman Ford Algorithm :-

- Dijkstra's Algo can't handle -ve weights
- But Bellman Ford can
- Note - It can form negative cycle like



& Again & Again.



Approach :-

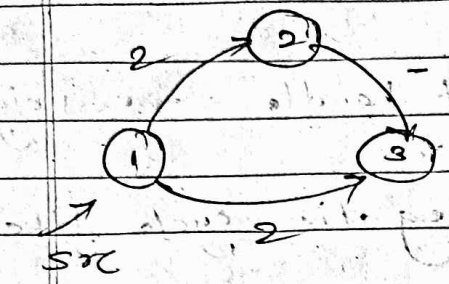
1) (N-1) times.
   ↓ we do this
   IF (dist[u] + wt < dist[v])

   dist[v] = distu + wt

2). For checking -ve cycles.
   we use same Formula
   & if any distance get updated
   then it is -ve cycle

   IF distance not getting update
   then return

   return dist[dest];



0	∞	∞
1	2	3

(1) $dist[1] + wt < dist[2]$

$0 + 2 < \infty$

2 ✓

$dist[2] + wt < dist[3]$

$2 - 1 < \infty$

$1 < \infty$ ✓

$dist[1] + wt < dist[2]$

$0 + 2 < 1$

$2 < 1$ ✗

(2) Check cycle

	u	v	wt
$dist[1] + wt < dist[2]$	1	2	(2)
$0 + 2 < 2$ ✗	2	3	(-1)
$dist[2] + wt < dist[3]$	1	3	(2)

$2 + -1 < 1$

$1 < 1$ ✗

No -ve cycle

$dist[1] + wt < dist[3]$

$0 + 2 < 1$

$2 < 1$ ✗

Page No.
Date

Disjoint set 1.

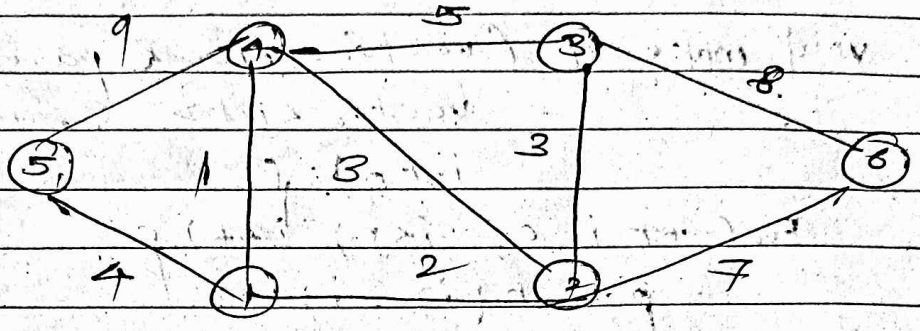
(1)
```
void makeset (vector<int> &parent,
              vector<int> &rank,
              int n) {
    For(int i=0; i<n; i++) {
        parent[i] = i;
        rank[i] = 0;
    }
}
```

(2)
```
int FindParent(vector<int> &parent, int node) {
    if( parent[node] == node) {
        return node;
    }
    return parent[node] = FindParent(parent,
                                     parent[node]);
}
```

(3)
```
void unionset( int u, int v, vector<int> &parent)
{
    u = FindParent(parent, u);
    v = FindParent(parent, v);
    if ( rank[u] < rank[v])
    { parent[u] = v; }
    else if( rank[v] < rank[u]) {
        parent[v] = u; }
    else
    {
        parent[v] = u;
        rank[u]++;
    }
}
```

Page No.
Date

Kruskal's Algo:-



Sorted,

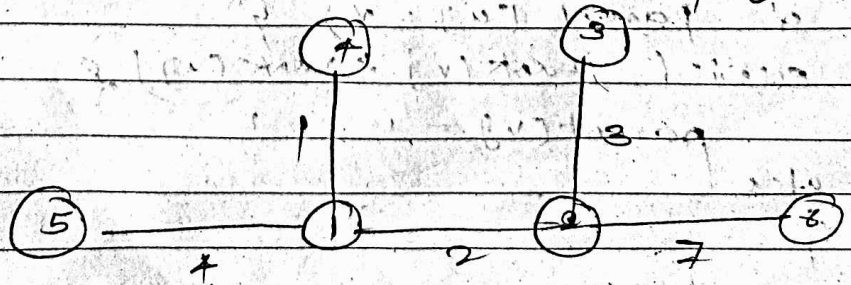
1	1	4
2	1	2
3	2	4
3	2	3
4	1	5
5	3	4
7	2	6
8	3	6
9	4	5

wt	u	v

Rule:-

(1) If u & v diff ↳ merge / union

(2) If u & v same ↳ Ignore



```
int minimumSpanningTree(vector<vector<int>
                        & edges, int n)
{
    sort(edges.begin(), edges.end(), cmp);

    vector<int> parent(n);
    vector<int> rank(n);

    int minWeight = 0;

    For(int i=0; i<edges.size(); i++)
    {
        int u = Find Parent(parent, edges[i][0]);
        int v = Find Parent(parent, edges[i][1]);
        int wt = edges[i][2];

        if(u != v){
            minWeight += wt;
            unionSet(u,v, parent, rank);
        }

    } return minWeight;
}
```

Page No.
Date

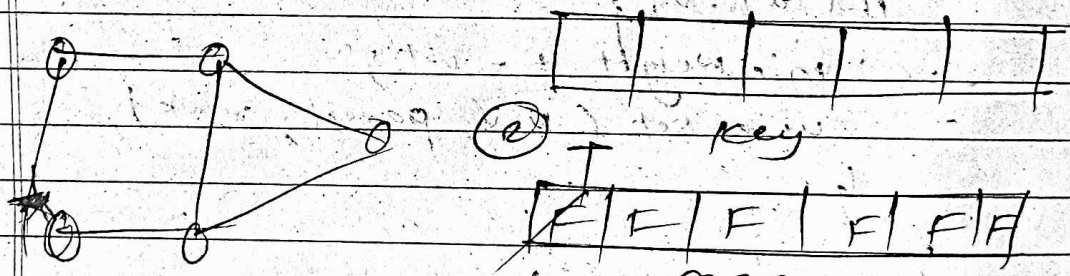
* Prim's Algorithm :-

* Steps :-

key [0] = 0
parent [0] = -1

(1) u → 0 Find out min from key
(2) mst[u] → true — make sure chek true in mst.
(3) adjacent element of u
& compare it's weight with key Array

(1) Find out min



(2) key

F	F	F	F	F	F

mST

(3) Find out Adjacent,

int eg. → | 0 | -1 | 0 | -1 | 0 |

Parent

& mark parent for it accordingly