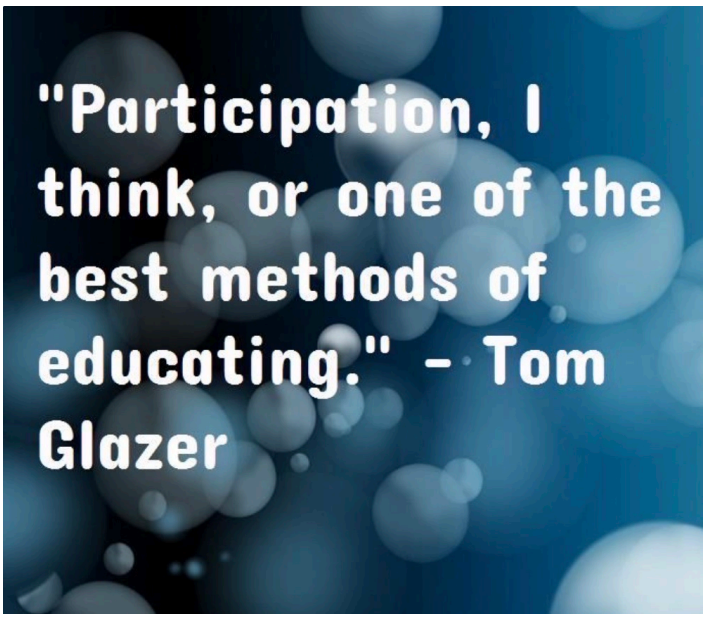
Graph 3



Good Evening

Today's content

01. Minimum spanning Tree
    → Prim's Algo
02. Djikstra Algorithm
03. shortest distance in maze

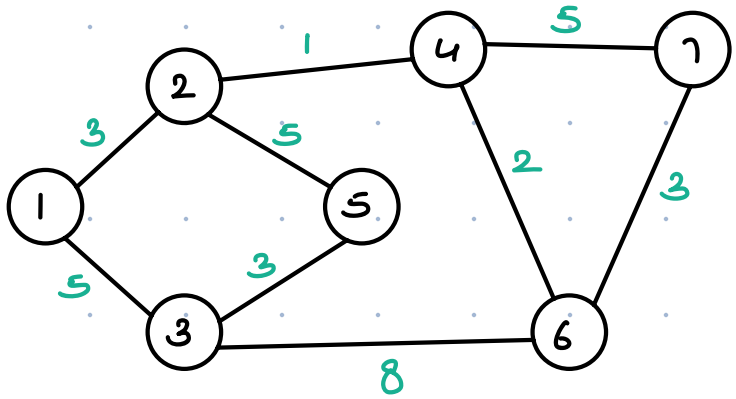
DSA 4.2
↓
Kruskal Algo
Floyd warshall Algo
Bellman's ford Algo
* DSU (Disjoint set union)

Flipkart

Q Given N no. of centres & their possible connections that can be made with particular cost. Find the minimum cost of constructing roads b/w centres such that it is possible to travel from one centre to any other centre via roads.

Eg: N = 7 E = 9

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



* To achieve lowest cost:
→ Minimum no. of roads / edges
→ N nodes ⇒ N-1 edges
Tree

Σ wt of selected edges as minimum

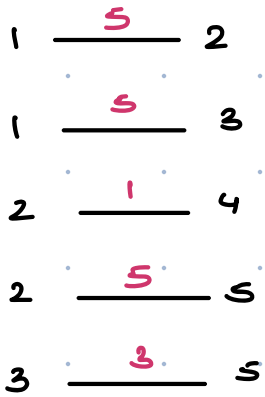
Minimum spanning Tree

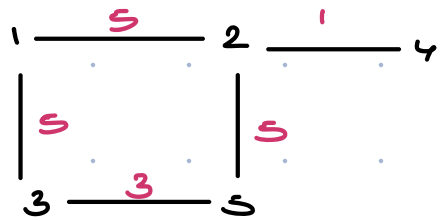
Can only be created for connected graph

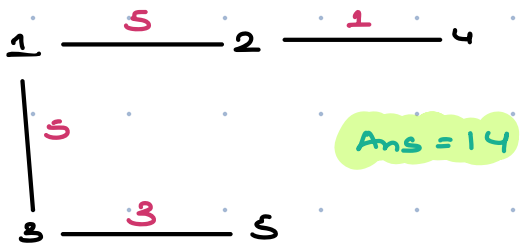
Minimum Spanning Tree

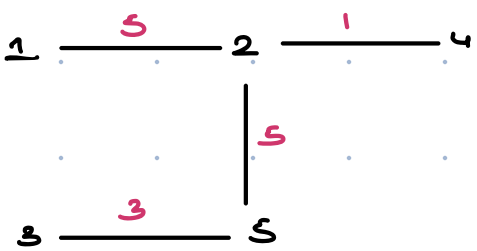
The minimum spanning tree has all the properties of a spanning tree with an added constraint of having the minimum possible weights among all possible spanning trees.

N = 5

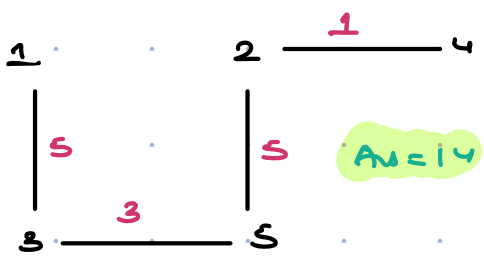


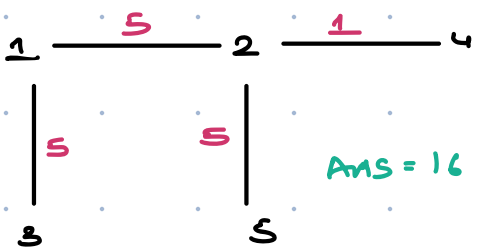






Ans = 14





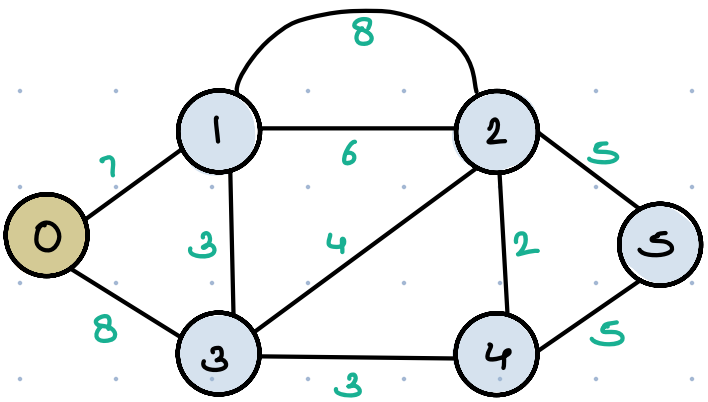
If all edge wts are unique → only one MST

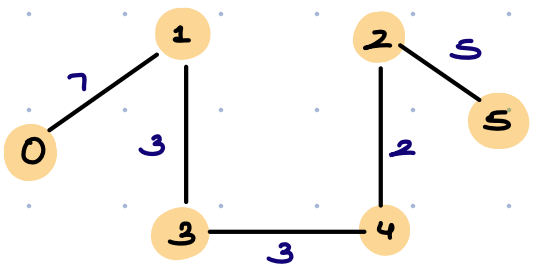
If multiple edges have same weight → more than one ans.

* Prims Algo

→ Start with any node as the root node & iterate on its nbr having minimum weight edge.

→ Visited array to make sure we are travelling only on unvisited nbrs.

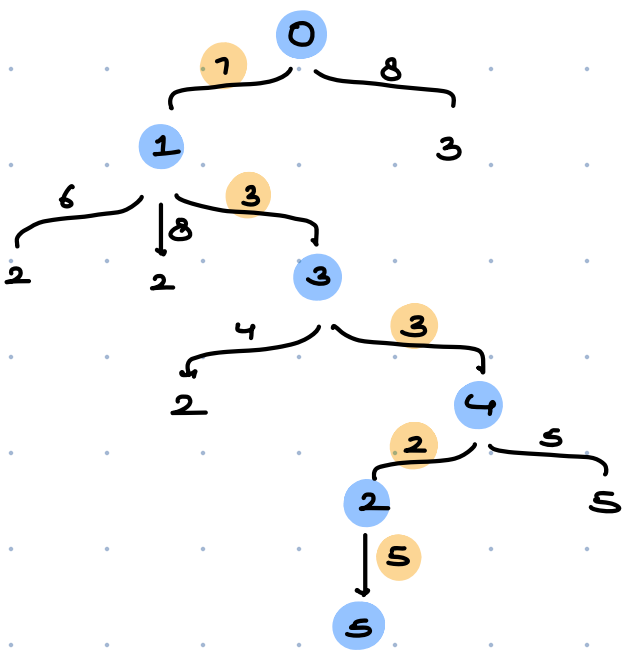


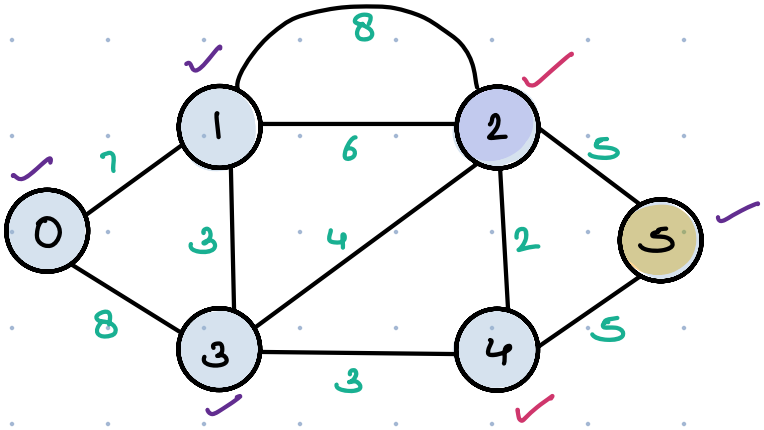


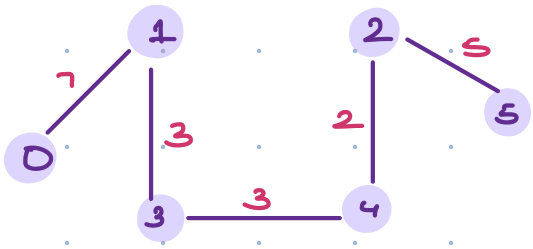
Ans = 7+3+3+2+5

= 20

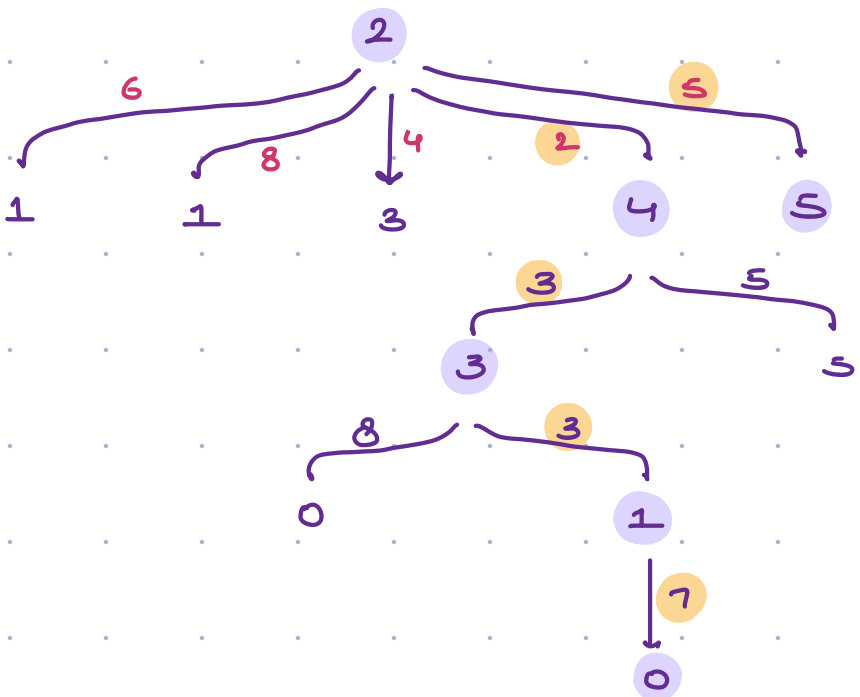
vis[ ] = { T T T T T T }
0 1 2 3 4 5





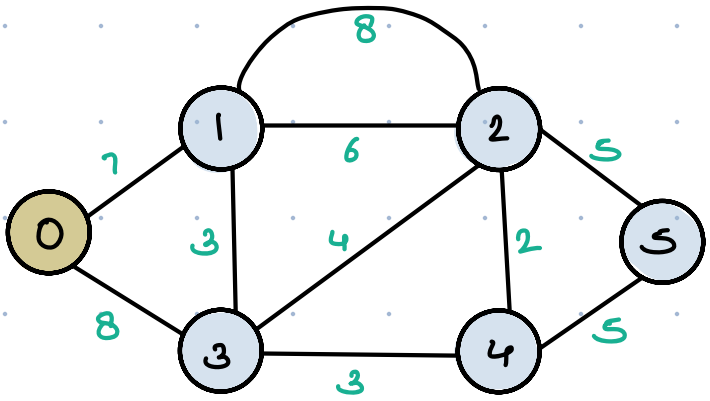


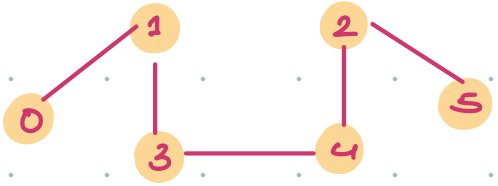
Ans = 20




```
class pair {
    int v;
    int wt;

    pair (int vt, int weight) {
        v = vt;
        wt = weight;
    }
}
```

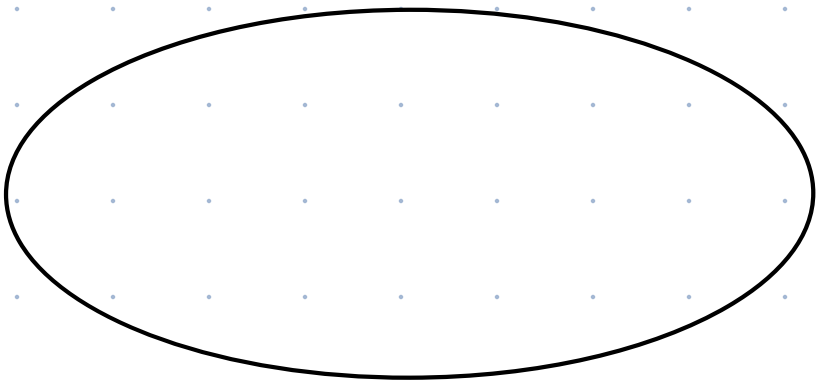
```
PriorityQueue<pair> pq = new PriorityQueue<>(new minwt());

pq.add(new pair(0, 0));   // v, wt

while (pq.size() > 0){
    pair rem = pq.remove();
    int vertex = rem.v;
    int weight = rem.wt;
    if (vis[vertex] == true) continue;
    else {
        vis[vertex] = true;
        ans = ans + weight;
        for (pair nbr : graph[vertex]){
            if (vis[nbr.v] == false){
                pq.add(new pair(nbr.v, nbr.wt));
            }
        }
    }
}
public class minwt implements Comparator<pair>{
    public int compare(pair a, pair b){
        if (a.wt < b.wt) return -1;
        else if (a.wt > b.wt) return 1;
        else return 0;
    }
}
```





vis[ ] = { T T T T T T }
0 1 2 3 4 5

ans = 0+0+3+3+2+5+7
= 20



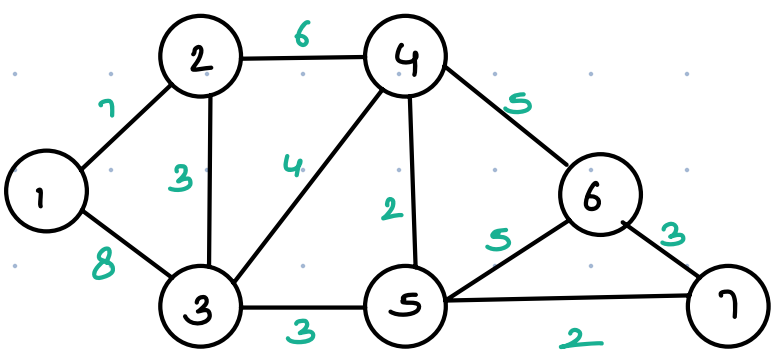
P9

10:07 pm → 10:17 pm

Single source shortest path Algorithm

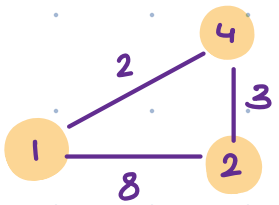
Djikstra Algorithm → All edge weight should be positive

There are n cities in a country, you are living in city 1. Find minimum distance to reach every other city fom city 1.



Dist[] = { X 0 7 8 12 11 16 13 }
index: 0 1 2 3 4 5 6 7

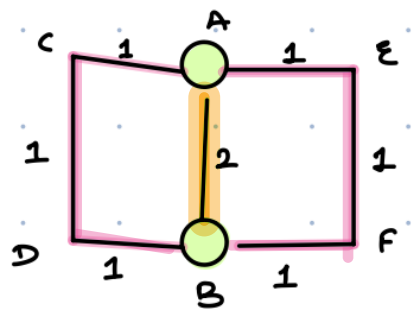
Dist[i] = minimum distance to reach i^{th} city from source city



shortest distance to reach 4 from 1 = 2 ∴ min edge at from 1

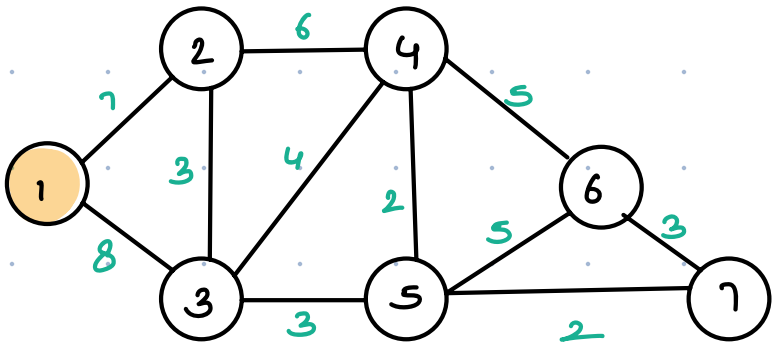
shortest distance to reach 2 from 1 = 5

* Direct edge b/w two nodes doesn't ensure the shortest path



Djikstra → Consider the path with minimum aggregated weight = 2

MST → Focus towards connecting all nodes with minimum weight

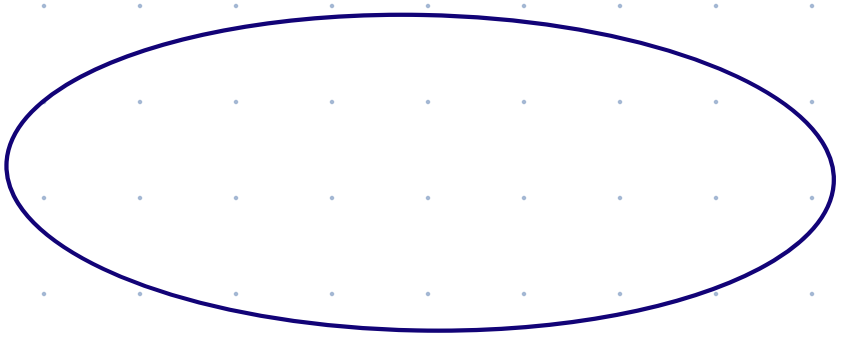


Dist [ ] = { ∞ ~~∞~~ ~~∞~~ ~~∞~~ ~~∞~~ ~~∞~~ ~~∞~~ ~~∞~~ }

0 7 8 12 11 16 13

0 1 2 3 4 5 6 7

src = 1



Arrays.fill(dist, ∞);

```
PriorityQueue<pair> pq = new PriorityQueue<>(new minwt());

pq.add(new pair(src, 0));
 dist[src] = 0
while (pq.size() > 0){
    pair rem = pq.remove();
    int vertex = rem.v;
    int wsf    = rem.wt;

    if (dis[vertex] != ∞ ) continue;
    else {
        dis[vertex] = wsf;
        for (pair nbr : graph[vertex]){
            if (dis[nbr.v] == ∞ ){
                pq.add(new pair(nbr.v, wsf + nbr.wt);
            }
        }
    }
}
public class minwt implements Comparator<pair>{
    public int compare(pair a, pair b){
        if (a.wt < b.wt) return -1;
        else if (a.wt > b.wt) return 1;
        else return 0;
    }
}
```

Purpose

1. **Prim's Algorithm**
   - **Problem Solved:** Finds the **Minimum Spanning Tree (MST)** of a graph, which connects all vertices with the minimum total edge weight and no cycles.
   - **Use Case:** Network design problems (e.g., laying cables, road networks).
2. **Dijkstra's Algorithm**
   - **Problem Solved:** Finds the **shortest path** from a source vertex to all other vertices in a graph.
   - **Use Case:** Navigation systems (e.g., GPS), packet routing.

Input Graph

1. **Prim's Algorithm:**
   - Works for **undirected graphs** with **non-negative weights**.
   - Requires a connected graph to find a valid MST.
2. **Dijkstra's Algorithm:**
   - Works for **directed or undirected graphs** with **non-negative weights**.
   - Does not require the graph to be connected.

Priority Metric

1. **Prim's Algorithm:**
   - Focuses on **edge weights** to grow the MST.
   - Prioritizes the smallest edge weight that connects to a new vertex.
2. **Dijkstra's Algorithm:**
   - Focuses on **total distance** (path weight) from the source vertex.
   - Prioritizes the vertex with the smallest cumulative path cost.