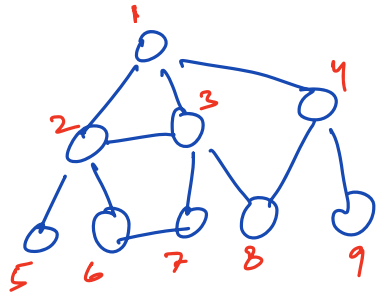
# Graphs 2 : BFS & MST

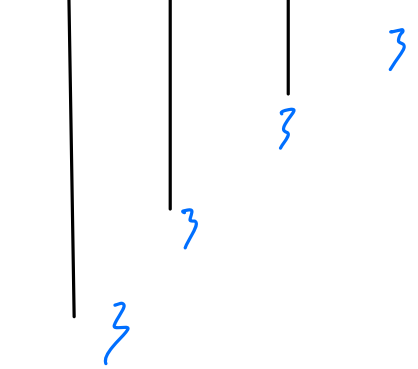
Breadth first search (BFS) → level order traversal

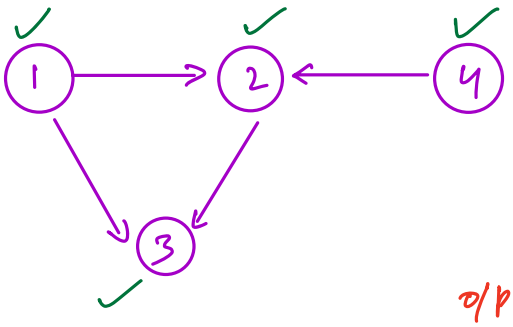



```
bool visit[n]
fi, visit[i] = false

for( i=0 to n-1) {
    if( !visit[i])
        bfs(i)
}

void bfs (int u) {
    Queue<int> q;
    q.enqueue(u);
    visit[u] = true;
    while( !q.isEmpty()) {
        x = q.dequeue()
        print(x)
        for( int v : graph[x]) {
            if( !visit[v]) {
                q.enqueue(v);
                visit[v] = true;
```

TC = O(V+E)

SC = O(V)



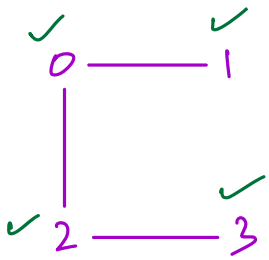



bfs(1)

bfs(4)

o/p → 1 , 2, 3, 4

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 2 | 1 | 0 | 0 | 1 |
| 3 | 0 | 0 | 1 | 0 |

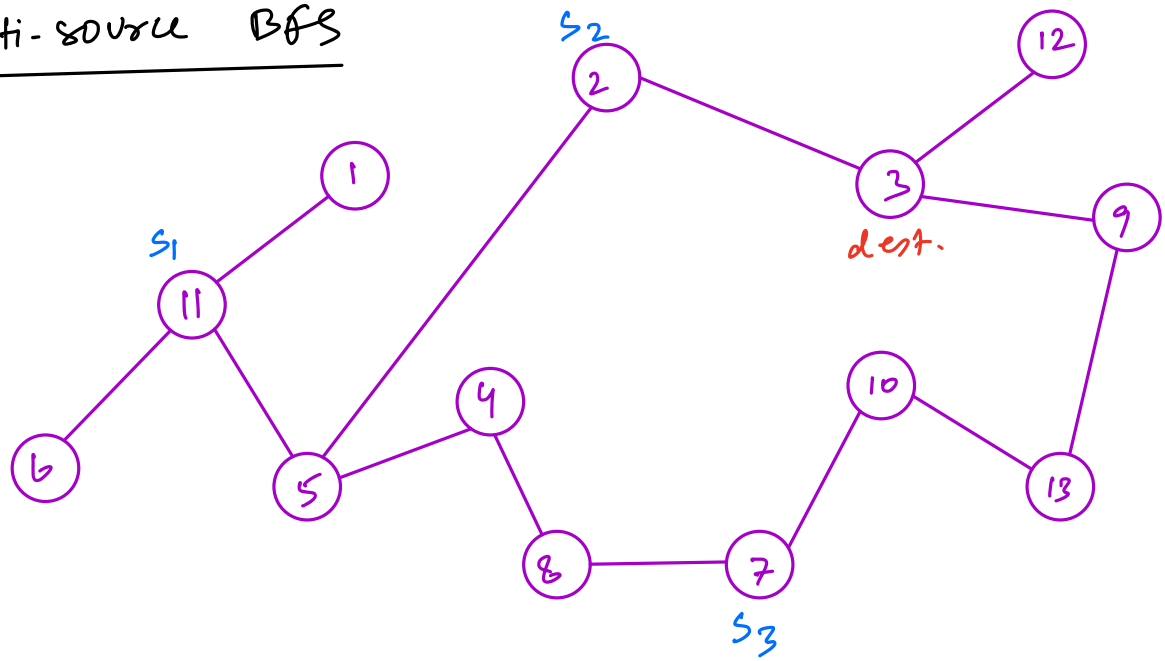



bfs(0)

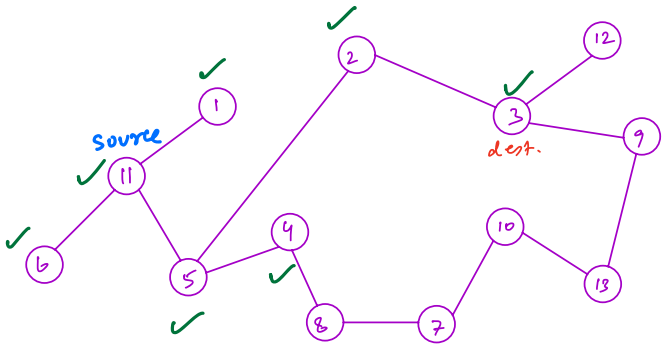
o/p → 0, 1, 2, 3

o/p → 0, 2, 1, 3

# Multi-source BFS

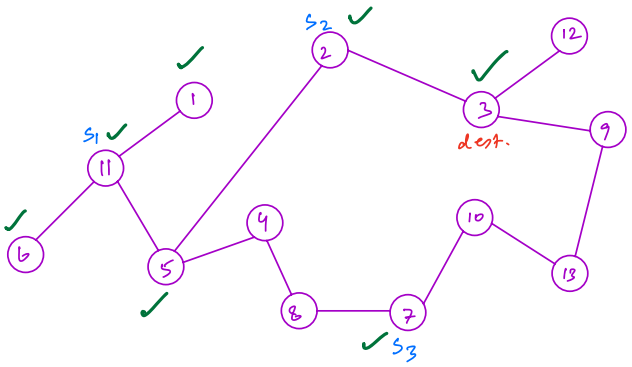



BFS always gives shortest path from source to destination in unweighted graph.




~~(11,0)~~ ~~(1,1)~~ ~~(6,1)~~ ~~(5,1)~~ ~~(2,2)~~ (4,2) (3,3)

BFS always gives shortest path from multiple sources to destination in unweighted graph.




~~(11,0)~~ ~~(2,0)~~ (7,0) (6,1) (1,1) (5,1) (3,1)

TC = O(V+E)

SC = O(V)

## Rotten Oranges

Given a matrix, there are 3 values

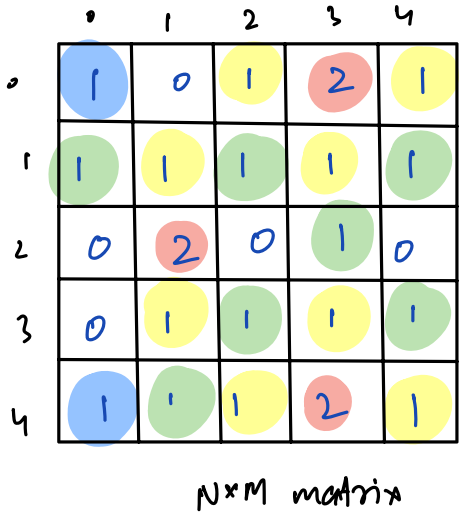
0 → empty cell

1 → fresh orange

2 → rotten orange

Find min. time to rot all oranges or return -1 if not possible.

A rotten orange, rots all of its neighbours in 1 unit of time.



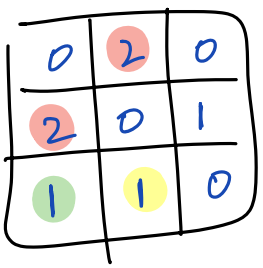

N×M matrix

after 1 unit

after 2 units

after 3 units → all are rotten

ans = 3




ans = -1

total nodes in graph = N×M

total edges in graph = 4×N×M

Code

V = N×M

E = 4×N×M

```
// input: A[n][m]

fresh = 0;
Queue<int[]> q;

for(i=0 to n-1){
   for(j=0 to m-1){
      if(A[i][j] == 2)
         q.enqueue({i,j,0});
      else if(A[i][j] == 1)
         fresh++
}
```

```
}
if (fresh == 0)  // no fresh orange
    return 0

row[] = {-1, 0, 1, 0}
col[] = {0, 1, 0, -1}

while (! q.isEmpty()) {
    cell = q.dequeue()
    i = cell[0]
    j = cell[1]
    time = cell[2]
    for (k = 0 to 3) {
        x = i + row[k]
        y = j + col[k]
        if (x >= 0 && x < n && y >= 0 && y < m) {
            if (A[x][y] == 1) {
                A[x][y] = 2;
                q.enqueue ({x, y, time+1});
                fresh--
                if (fresh == 0)
                    return time+1
            }
```

```
    }
  }
}
return -1;  // since fresh > 0
```

$TC = O(V + E)$

$= O(N \times M)$

$SC = O(V)$

$= O(N \times M)$

Question

Given N distribution centers & cost of constructing roads b/w multiple pair of centers. Find the min. cost of construction required such that all centers are connected.

N=7

1 —3— 2

1 —5— 3

2 —1— 4

2 —5— 5

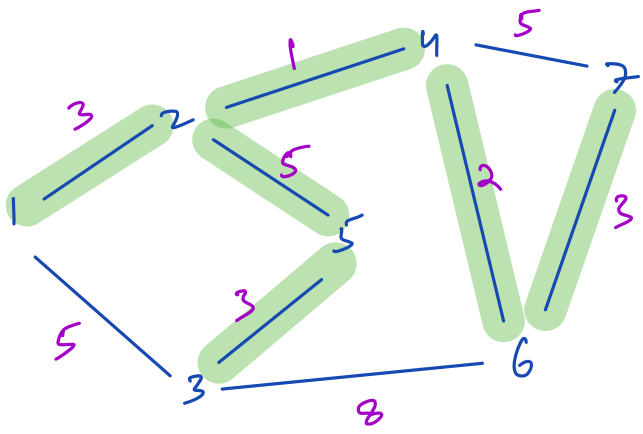
3 —3— 5

4 —2— 6

3 —8— 6

4 —5— 7

6 —3— 7




ans: 3+3+5+1+2+3

= 17

Idea:

1. Aim for fewer roads

minimizing roads with reduce cost.

2. Opt for tree structure

Tree needs minimal edges to connect all nodes. [N-1 edges]

N = 5

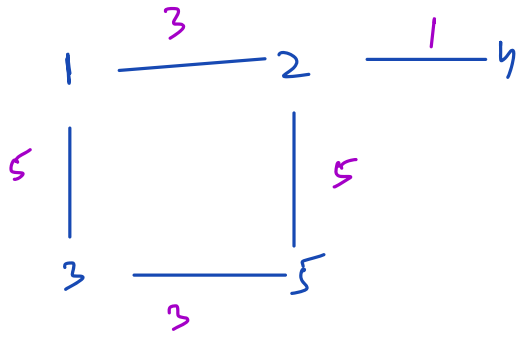
1 —3— 2

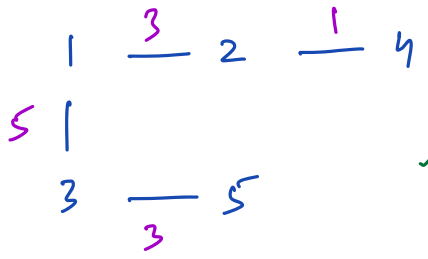
1 —5— 3

2 —1— 4

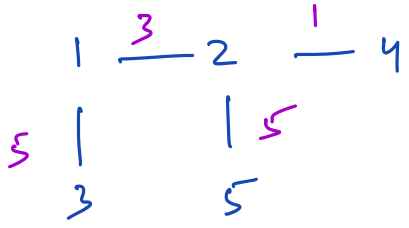
2 —5— 5

3 —3— 5




sum = 12




sum = 14

**Spanning Tree:** Tree which spans (covers) all the vertices with minimum no. of edges.

**Minimum Spanning Tree (MST)**

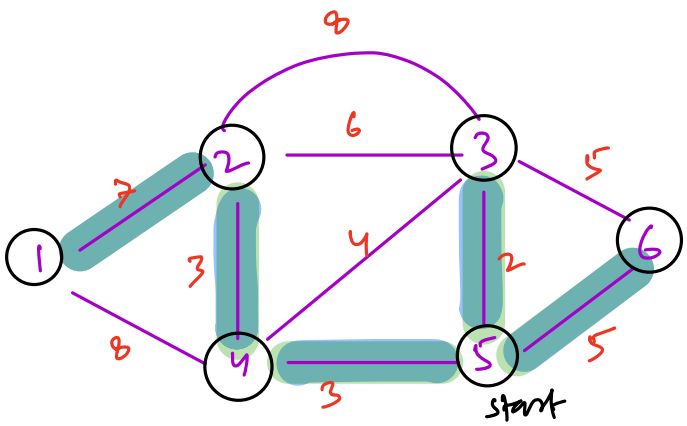
Spanning tree with minimum weight.

Note: If all edge weights are unique, there's only one MST.

Algo to find MST → Prim's Algo ✓

Kruskal's Algo ← DSA 4.2

## Prim's Algo

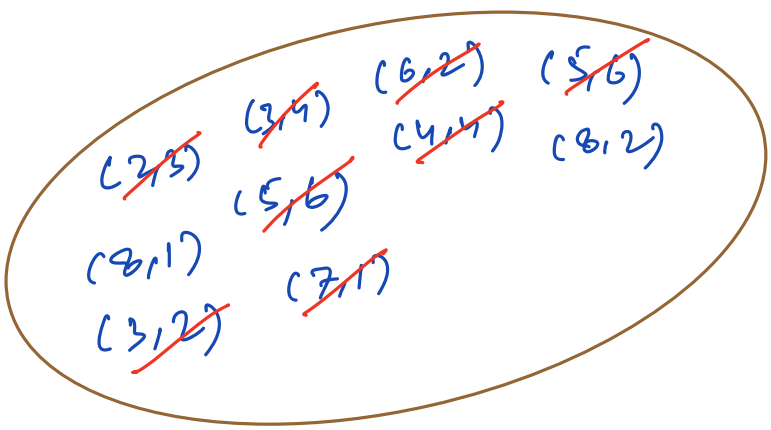



| T | T | T | T | T | T |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 |

visit array

ans = 7 + 3 + 3 + 2 + 5

= 20




Heap {edge wt, connecting node}

heap will have edges

Code

```
// input: list<pair<>> graph[N]

bool visit[N]
∀i, visit(i) = false
sum = 0
Heap<pair<>> h;
visit[0] = true
for ( pair p : graph[0] ) {
        h.insert ( {p.wt, p.v} );

while ( ! h.isEmpty() ) {
    pair p = h.getMin();
    if ( visit[p.v] == true ) {
        continue;
    }
    visit[p.v] = true;
    sum += p.wt;
    for ( pair x : graph[p.v] ) {
        if ( ! visit[x.v] ) {
            h.insert ( {x.wt, x.v} );
        }
    }
}
```

}

$TC = O(E \log E)$

$SC = O(V+E)$