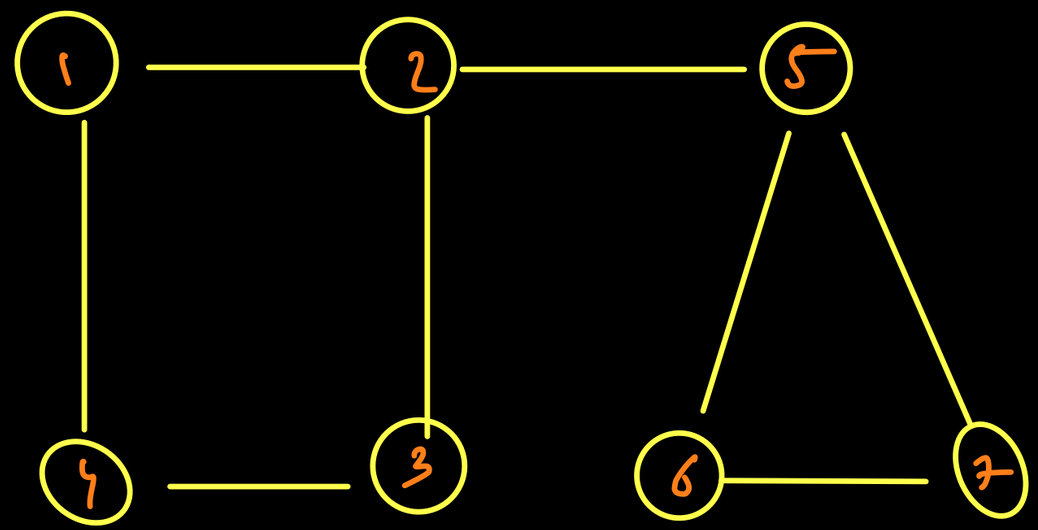
- adjacency list
- adjacency matrix
- adjacency map
- incidence matrix
- Edge list

Graph Traversals

We are given the adj List representation of the graph given below



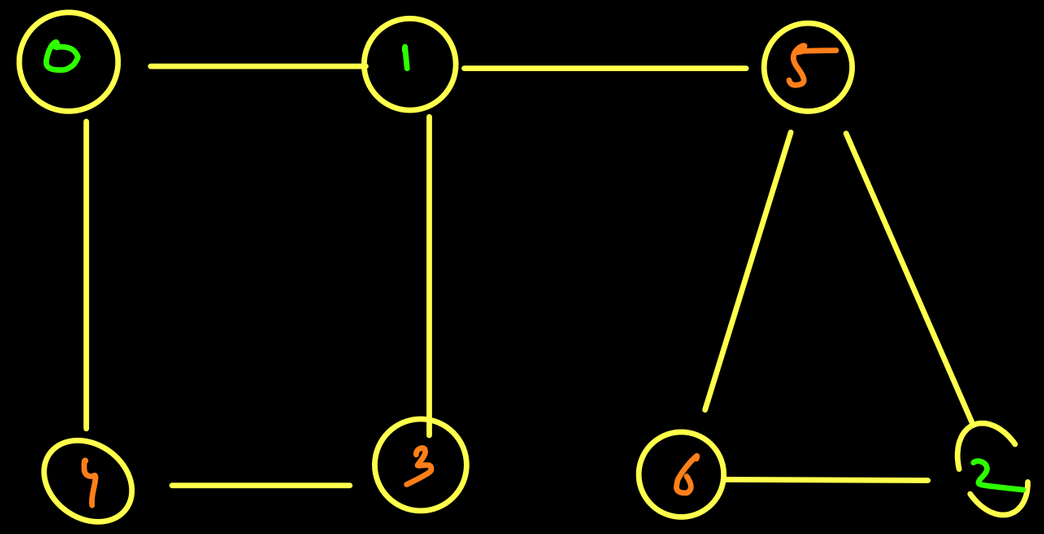
To read any graph we have 2 major techniques

- Depth first traversal / Search
- Breadth first traversal / search

Depth first traversal (Recursive)

Let's take a motivation problem →

1) Given a graph calculate all paths between 2 vertices

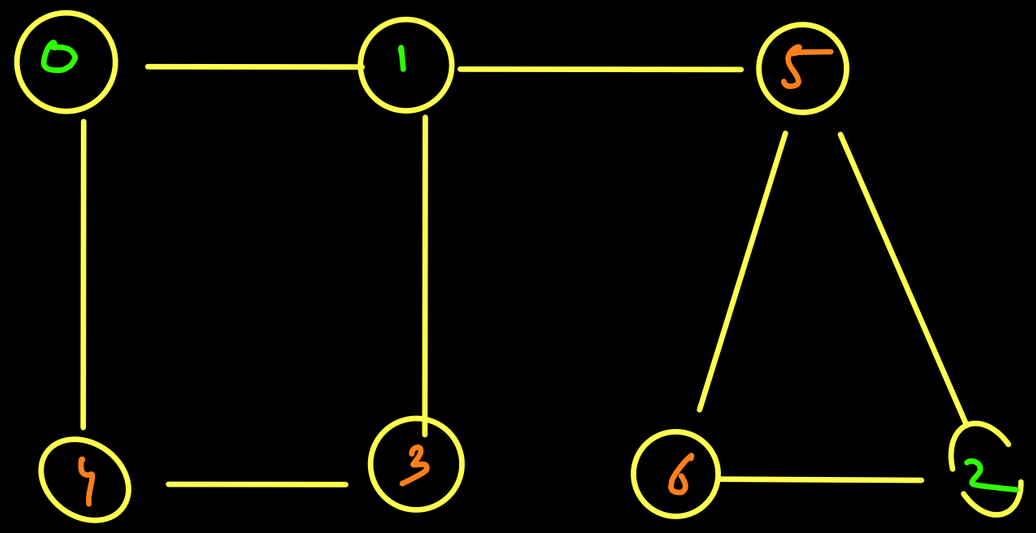


or

2) Given a graph check whether there is a path between any 2 vertices or not ??

Hint → Recursively

vertex 0 – 6

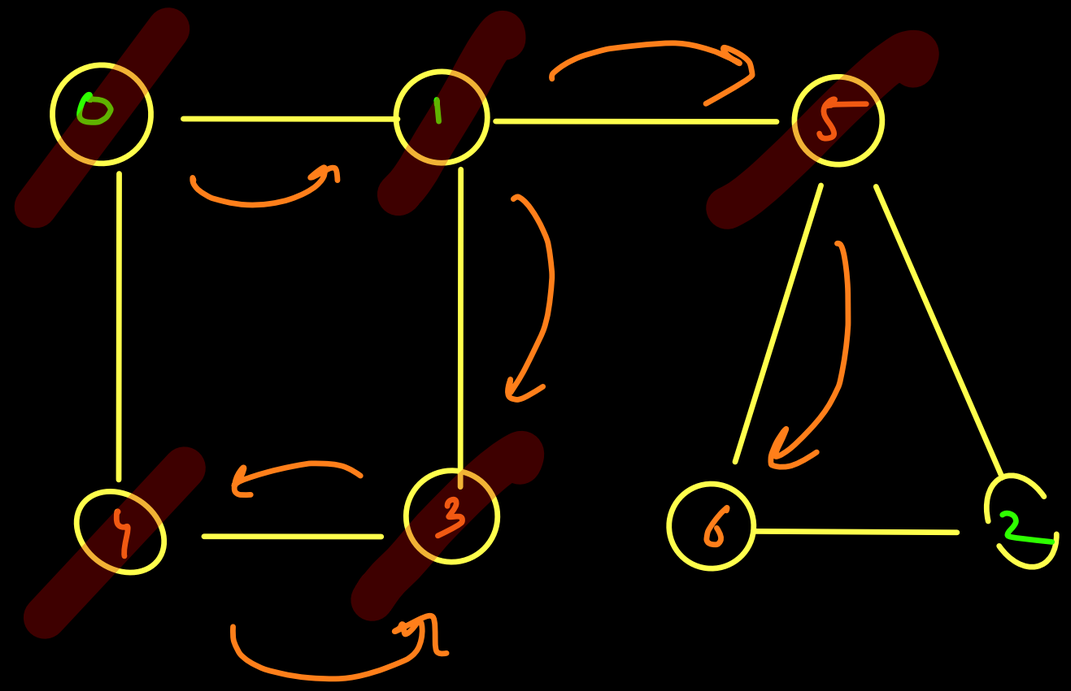


Simplest sol^n for any path problem would be if src & dest are neighbours.

if there is a path from neighbours to dest then there will be a path from src to dest via the neighbour

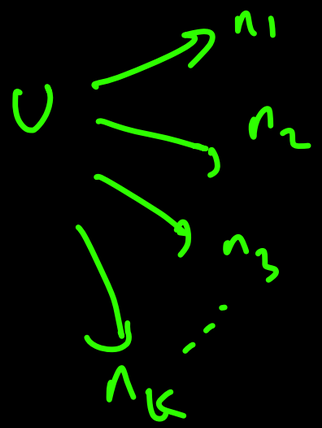
$0 \rightarrow 1 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6$

$dfs(0) \rightarrow dfs(1) \rightarrow dfs(5) \rightarrow dfs(6)$



$f(u, v)$ = ?

whether there is a path from u to v or not

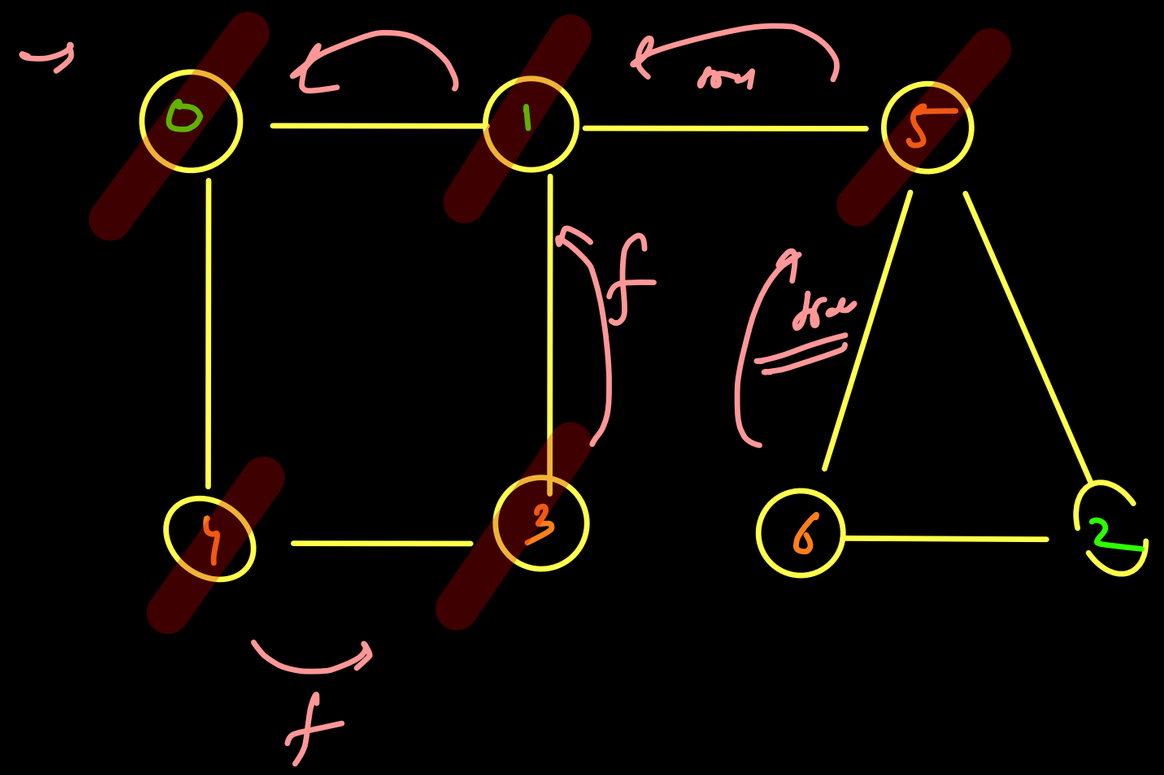


$$f(u, v) = \begin{cases} f(n_1, v) \\ \text{or} \\ f(n_2, v) \\ \text{or} \\ f(n_3, v) \\ \vdots \\ f(n_K, v) \end{cases}$$

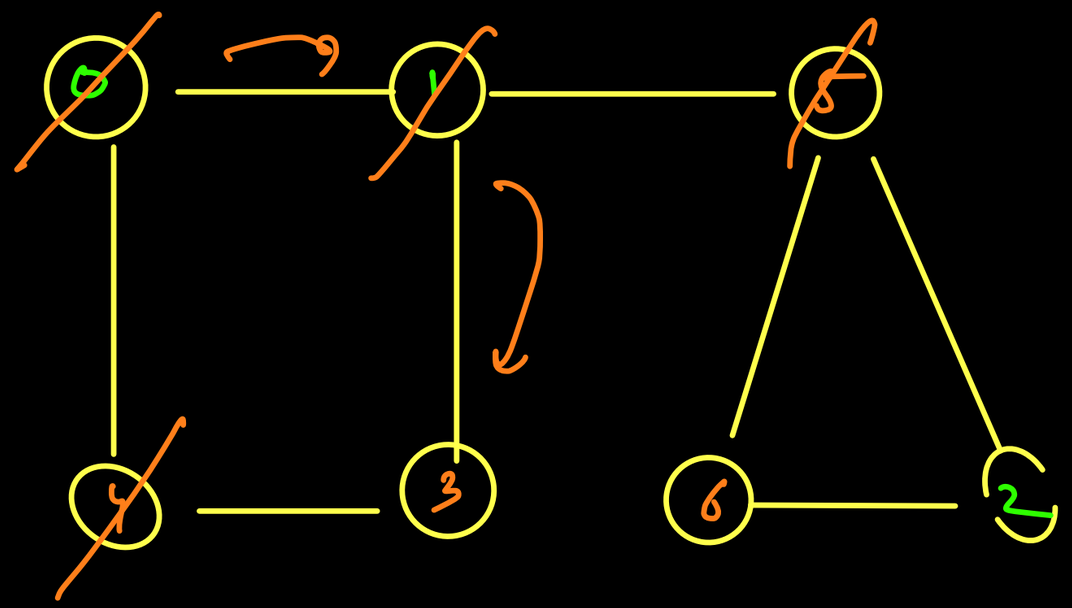
$n_1\ n_2\ n_3\ \ldots\ n_K$ are immediate neighbors of u and all these neighbors are unvisited

visited → 0, 1, 3, 4, 5,
↳ set
↳ array

0 → 1 →



```cpp
bool dfs(int curr, int end) {
    if(curr == end) return true;
    visited.insert(curr); // mark visited
    for(auto neighbour: graph[curr]) {
        if(not visited.count(neighbour)) {
            bool result = dfs(neighbour, end);
            if(result) return true;
        }
    }
    return false;
}
```



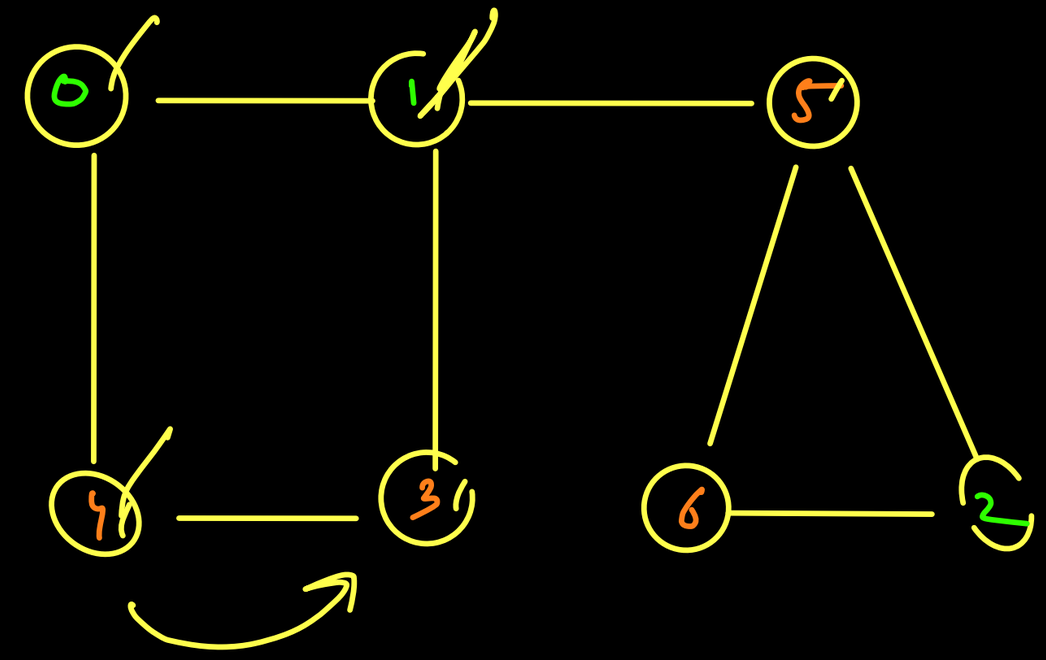
$O(V+E)$ T.C

dfs(0,6) ... dfs(0,6)

(0, 1, 3, 4, 5

Result = [[ ], [ ]]

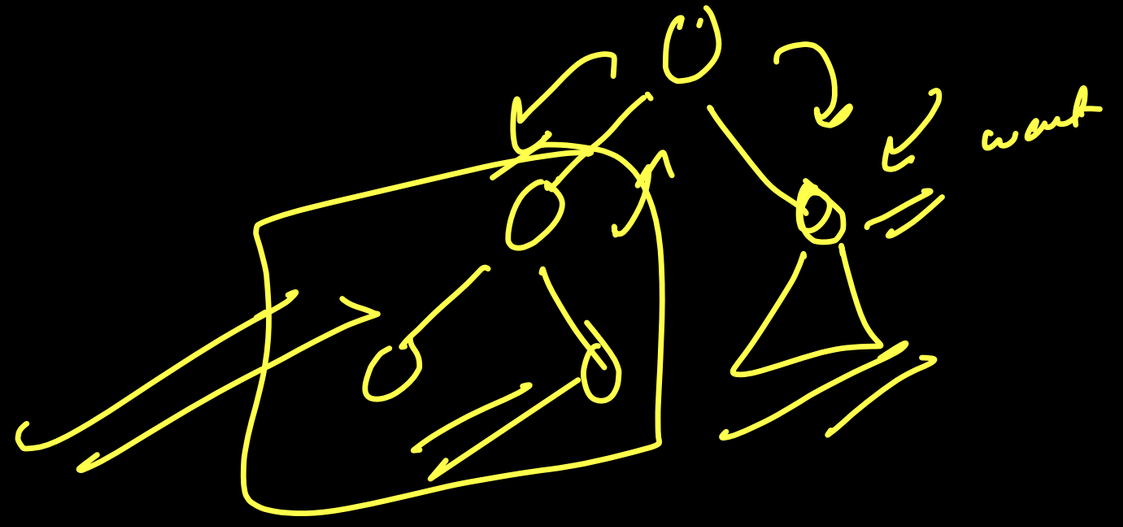
path



< 0, 4, 3, 1, 5 >

dfs(0,6) → dfs(4,6) → dfs(3,6) → dfs(1,6) →

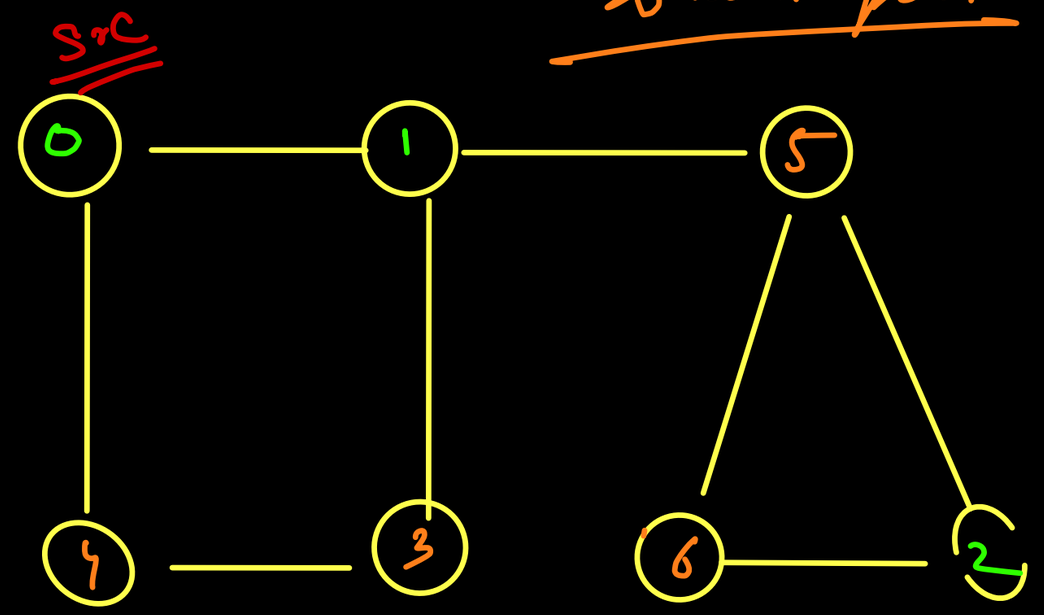
dfs(5,6)



→ Breadth first Search → unweighted graph

Shortest path

In BfS, we travel the immediate neighbours first together.



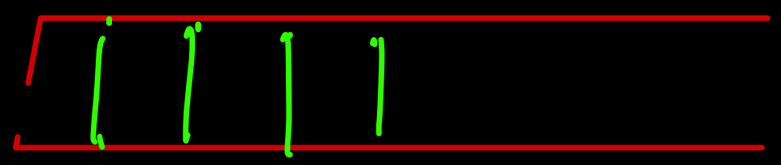
0, 1, 4, 3, 5, 6, 2

Queue

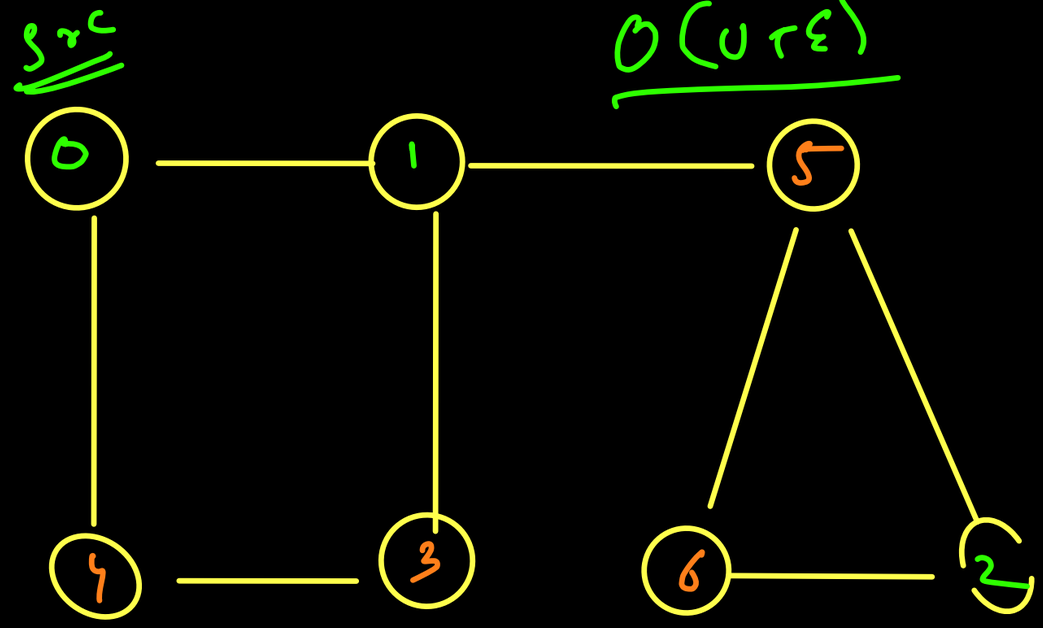
process immediate neighbours

vis = {0, 1, 4, 3, 5, 6, 2}

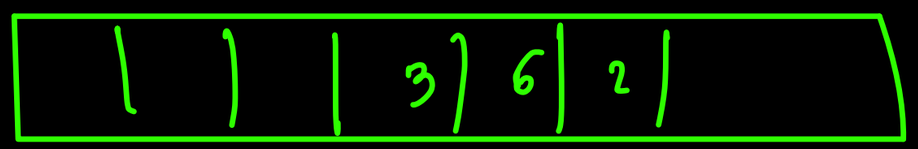
0 1 4
3 5 6 2

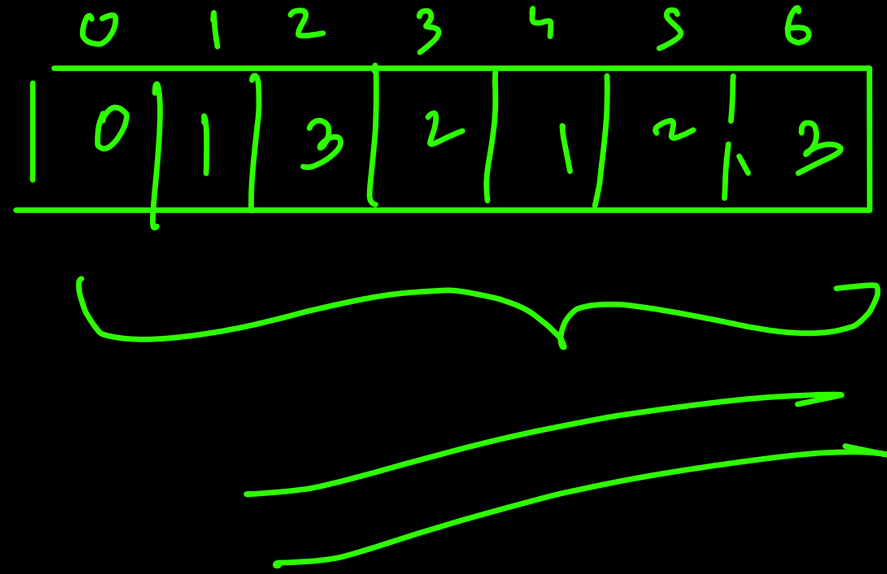


0 1 4 vis → [0, 1, 2
s, s
s]



queue





dist → shortest path