① CRUD operation on graph.

1. **CREATE** → how do you create a graph ?

2 ways : a) adjacency matrix   b) adjacency list .

→ to

$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 1 & 1 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 & 1 & 1 \\ 4 & 0 & 0 & 1 & 0 & 1 \\ 5 & 0 & 0 & 1 & 1 & 0 \end{array}$$

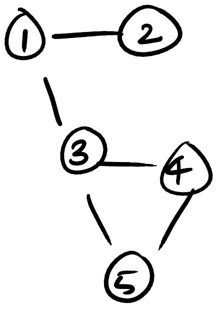
↑ from

$O(n^2)$ space

$g[u][v] = 1$

$g[v][u] = 1$

dense



graph

$n = 5$

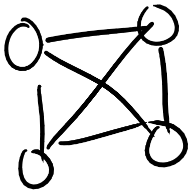
1 | 2 | 3
2 | 1
3 | 1 | 4 | 5
4 | 3 | 5
5 | 3 | 4

$O(e)$ space

sparse

g[u].push_back(v)

g[v].push_back(u)



$\frac{n(n-1)}{2} \sim {}^{n}C_2$

n nodes   min no. of edges to have all nodes connected. **n-1**

max no of edges n^2 **${}^{n}C_2$**

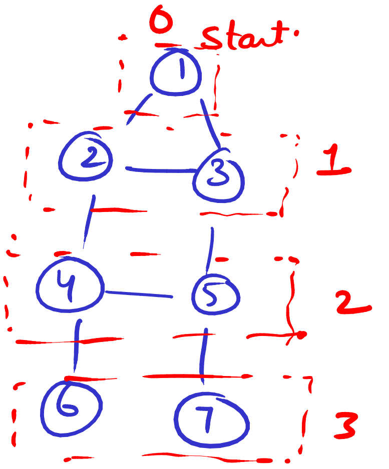
2 **READ**   Recall trees → DFT
BFT

graph
e
$O(n)$ ——— $O(n^2)$
sparse   dense

DFT → Pre
↳ in
↳ post. } graphs ✗   ∵ no left & no right

BFT → level order } graphs ✓   ∵ **level** ?

min
# steps you need to take to reach a particular node from start.



① → ② → ③

5

BFT : Breadth first traversal → Trees

BFS : Breadth first search → Graphs.

Tree

BFS (G , start) :

```
{ visited [n] = {false}
  level [n] = {-1}
  parent [n] = {-1}

  step = 1

  // process the start node
  q.push (start);
  visited [start ) = true
  level [start ] = step-1
  parent [start] = -1   → *

  while (! q.empty()):
      int u = q.top() ; q.pop( );
      cout << u << " " ;
      for all neighbours 'v' from g[u] :
  if
  ! visited [v] :  → q.push (v)
                     visited [v] = true
                     level [v] = step
                     parent [v] = u
      step++
```

queue<int> q ;

BFT

```
x { if root == null :
        return

→ queue <Node> q ✓
  q.push (root) ✓
  while ( !q.empty()):
      int u = q.top()
          q.pop()

      cout << u <<" "

      if u->left :
          q.push
          (u ->left)
      if u->right :
          q.push (u->right)
```

if the left & right exist, push in the queue. ⇐

Shortest path algo.

levels → min steps

'a' start 'b' end

how many steps ?

Do BFS (g, a) level [b] .

node = destn.

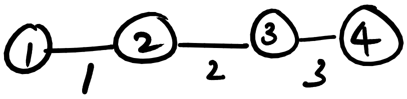
path : while (node != -1)
    path.push (node)
    if parent [node] != -1
        node = parent [node]

node

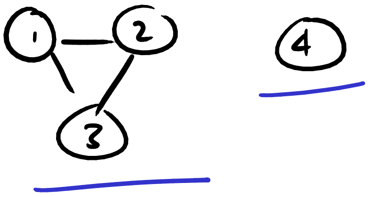
n , e edges

2. DFS. Why DFS if I know BFS ? Disconnection.

n nodes → min n-1 edges to make it connected.



components.



2 functions :

DFS — call DFSVisit = #components.

DFS Visit — finishes the traversal of a component

graph?

global OR passed as 3rd param.

```
DFS(g)
  n = g.size()
  visited[n] = {false}

  for int i = 0, i<n, i++:
      if !visited[i]
          DFSVisit(g, i)
```

connected?
↓
DFS ~ DFSVis

```
DFSVisit(graph g, start u)
    visited[u] = true

    cout << u << " ";

    for neighbour v of g[u]:
        if !visited[v]:
            DFSVisit(g,v)
```

BFS : Iterative

```
queue q;
q.push(start); ← i

while (!q.empty()):
    { traverse the component
      with i as start node.
```

q.push(start of other component)

??

⇒ done !!