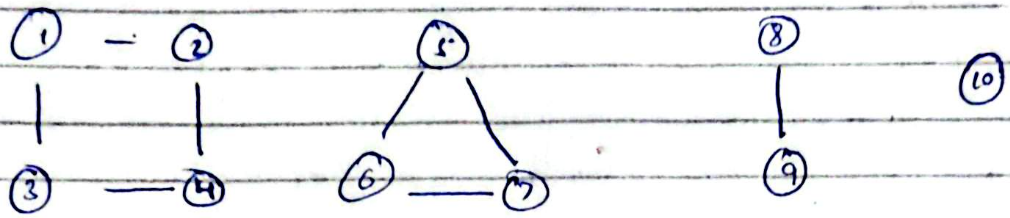
Graphs:

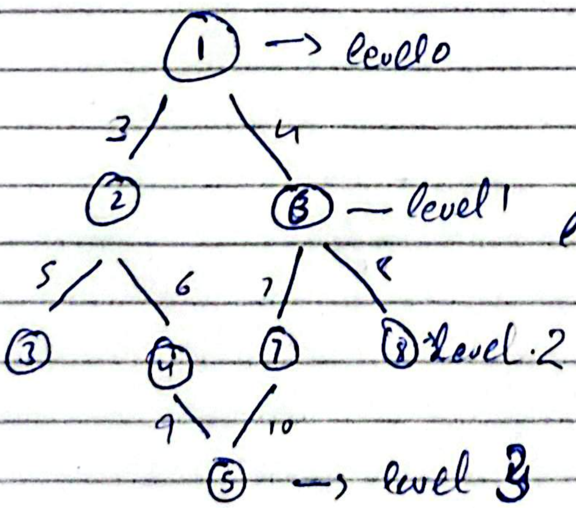
What are connected Components:



This is one graph divided into 4 components.

Traversal Algorithms:

BFS: breath first search technique.



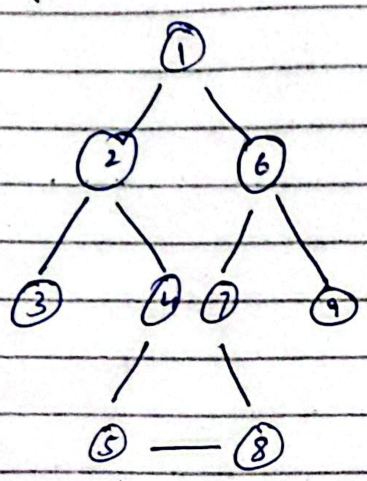
N=8

level wise traversal:

lets suppose starting node is = 1

we take queue data structure.
and a set() : visisted.

What ever the starting node is add in
the queue and set()



the adjoncency list
is

0 ⇒ [ ]
1 → [2, 6]
2 → [1, 3, 4]
3 → [2]
4 → [2, 5]
5 → [4, 8]
6 → [1, 7, 9]
7 → [6, 8]
8 → [5, 7]
9 → [6]

we go to 1 and see who are
neighbors of 1

q = [1]
v = [1]

neighbors for 1 so
q = [2, 6]
v = [1, 2, 6]

nei for 2
q = [6, 8, 3, 4]
v = [1, 2, 3,

q = [1]
v = [1]

i = 0

q = [2, 6]

v = [1, 2, 6]

i = 1

q = [6, 3, 4]

v = [1, 2, 6]

i = 2

q = [3, 4, 6, 7]

v = [1, 2, 6, 3]

i = 3

q = [4, 6, 7, 5]

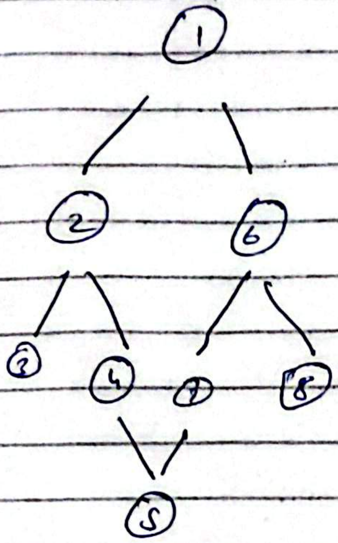
v = [1, 2, 6, 3, 4]

. . . . . .

q = [4, 6, 7, 9, 5, 8]

v = all as the level ends.

as the queue is appended as that visited becomes

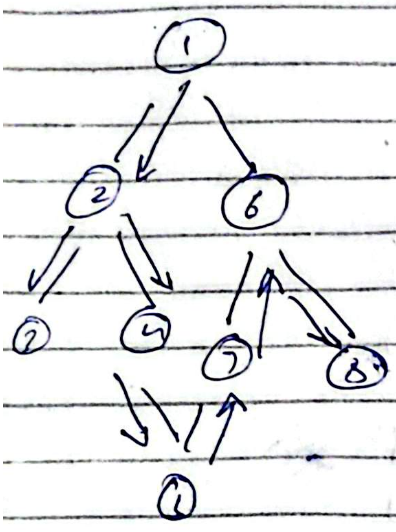


adj list

1 → [2,6]
2 → [1,3, 4]
6 → [1,7, 8]
3 → [2]
4 → [2, 5]
7 → [6, 5]
8 → [6]
5 → [4,7]

BFS = [[1], [2,6], [3,4,7,8], [5]]

```
while q
      level = []        -> to store levels
      for  i  in  range(q)  -> for every
                                 level
new     <-  n = q.pop()
level       for  nei   in   graph[n]
                 if  not in  visited
                     q.add(nei)
                     v.add(v)
      res.add(level)
```

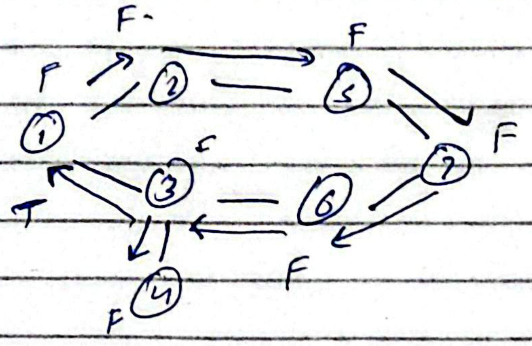


DFS
from 6 it will not
go to 1 cuz it
is in visted

so DFS = 1, 2, 3, 4, 5, 7, 6, 8

Cycles in the graph:

Undirected graphs:



adj list =

1 -> [2, 3]
2 -> [1, 5]
3 -> [1, 4, 6]
4 -> [3]
5 -> [2, 7]
6 -> [3, 7]
7 -> [5, 6]

visited = [1 1 1 1 1 1 1]
(indices: 1 2 3 4 5 6 7)

DFS(1, -1) -> parent for first node = -1

DFS(2, 1) (3, 1)

↓

DFS(5, 2)

↓

DFS(7, 5)

↓

DFS(6, 7)

↓

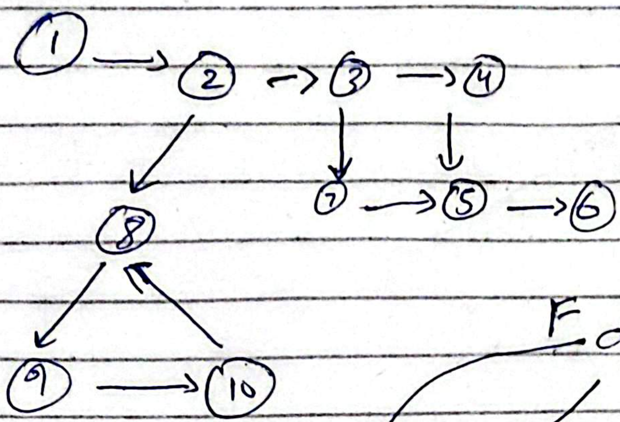
DFS(3, 6) ⟶ DFS (1, 3) = True because 1 != parent but in visited.

↓

DFS (4, 1) ⟶ False now Visited = [1, 2, 3, 4, 5, 6, 7] all of nodes.

So if a node != parent but in visited then its a cycle.

DIRECTED GRAPHS



dfs(1) T — path [1], v [1]

F dfs(2) T — path [1, 2], v [1, 2]

dfs(8) T — path [1, 2, 8], v [1, 2, 8]

dfs(9) T — path [1, 2, 8, 9], v [1, 2, 8, 9]

dfs(10) T — path [1, 2, 8, 9, 10], v [1, 2, 8, 9, 10]

dfs(8) T

8 in visited
and in path
cycle detected

dfs(3) F — path [1, 2, 3], v = [1, 2, 3]

F

path=[1,2,3,7] dfs(5) (7) — v=[1,2,3,7]

dfs(4) — p=[1, 2, 3, 4], v=[1, 2, 3, 4]

F F

dfs(5) F

dfs(6) F — p = [1, 2, 3, 4, 5, 6], v = [1, 2, 3, 4, 5, 6]

p dfs(5) — p=[1, 2, 3, 7, 5], v=[1, 2, 3, 7, 5]

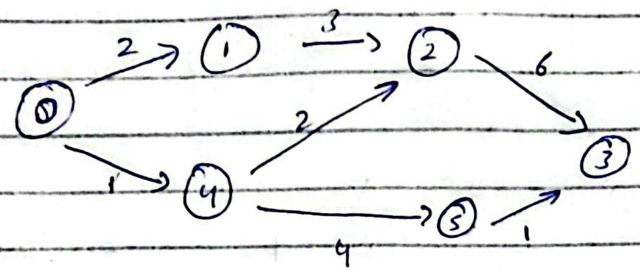
F

dfs(6) — p = [1, 2, 3, 7, 5, 6], v = [1, 2, 3, 7, 5, 6]

F

Now path.pop

SHORTES PATH in directed Acyclic Graph



dist = [0, 2, 3, 6, 1, 5]

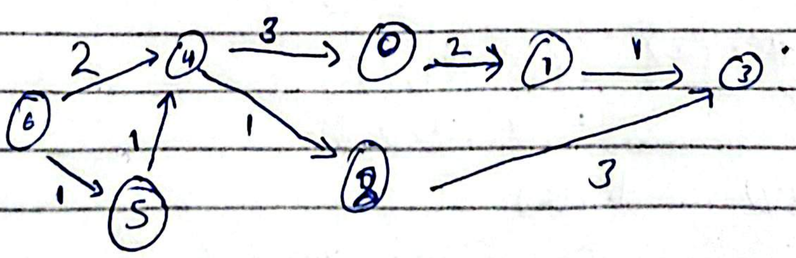
0	1	2	3	4	5

for 0 src

Algorithm:

(1) first do a toposort for the graph

Graph:



adj list:

$\quad\;\; v \;\; w$

0 -> (1, 2)

1 -> (3, 1)

2 -> (3, 3)

3 -> None

4 -> (0, 3) (2, 1)

5 -> (4, 1)

6 -> (4, 2) (5, 3)

i = 0

dfs(0)

V = [0]

dfs(1) V=[0,1]

topo.stack
[3,1,0]

dfs(3) v=[0,1,3]

Now i = 1 X in visited j = 2 Not in visited

dfs (2)

topo.stack = [3,1,0,2]

dfs(3)

Now i = 3 X i = 4

dfs(4)

topo stack = [3,1,0,2,4]
v = [0,1,2,3,4]

dfs(0) x dfs(2) x in visited

dfs(5)

topo stack = [3,1,0,2,4] 5
visited = [0,1,2,3,4,5]

dfs(4) x

dfs(6) all nei in visited

topo stack
= [3,1,0,2,4,5,6]
v = [0,1,2,3,4,5,6]

topo stack

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

each is coming before in stack.

Step

2

Take the nodes out of stack and relax the edges.

Now as we know stack is LIFO

So first one is 6

node = 6

dist = 0 because we know it is last but first of every node.

from 6 i can go to

node = 6 dist = 0

+3 +2

node = 5, 3 node = 4, 2

$$\text{dist} = \begin{array}{ccccccc} & & & & 2 & 3 & 0 \\ 0 & 1 & 2 & 3 & 4 & 5 & 6 \end{array}$$

edge weights are 2, 3

Now

node 5 dist = 3

1

node 4. dist = 3+1

x

cus $6 \to 4 = 2$

node 4   dist[4] = 2

3   1

node 0   node 2

dist = [5, inf, 3, inf, 2, 3, 0]
0 1 2 3 4 5 0

node 2   dist = 3

node 3   3+3 = 6

dist = [5, inf, 3, 6, 2, 3, 0]

node 0   dis=5

1

node = 1   5+1 = 6   dist = [5, 6, 3, 6, 2, 0]

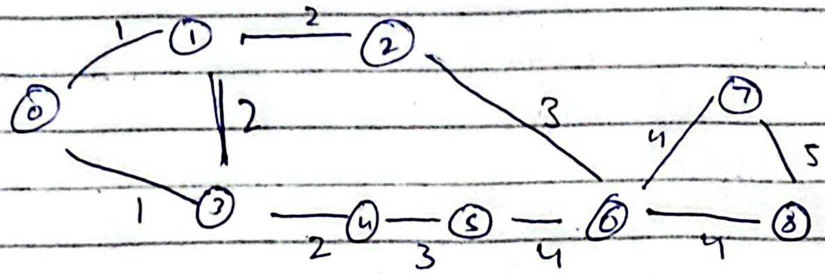
node 1   dist = 6

node 3   6+1 = 7 > 6   X nochange

node = 3   So dist = [5, 6, 3, 6, 2, 0]

no neighbors

SHORTES DISTANCE IN UNDIRECTED Graph.



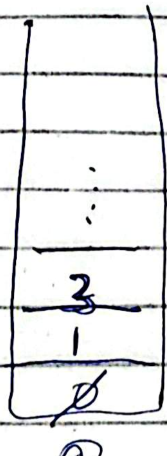
adj list=

0 → 1, 3
1 → 0, 2, 3
2 → 1, 6
3 → 0, 4
4 → 3, 5
5 → 4, 6
6 → 2, 5, 7, 8
7 → 6, 8
8 → 6, 7

node 0 dist=0

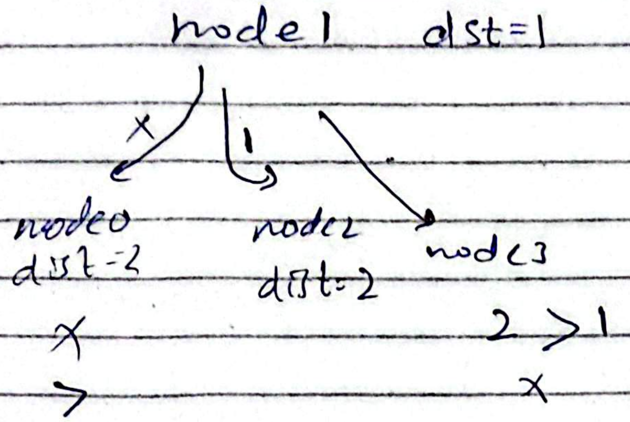
1 1

node 1 node 3



Q

dist = [0, inf....]



dist=[0, 1, i, 1, i, i, i ...]
0 1 2 3

dist=[0, 1, 2, 1 ....]

in the end

dist=[0, 1, 2, 1, 3, 3, 4, 4]
0 1 2 3 4 5 6 7