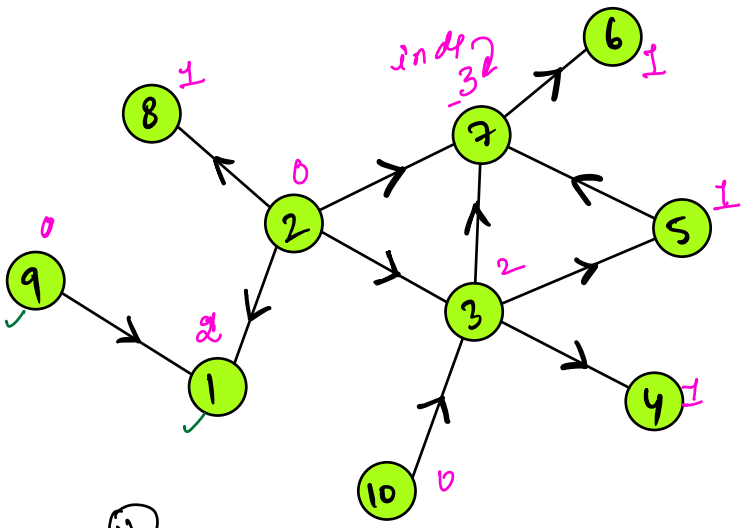
directed graph

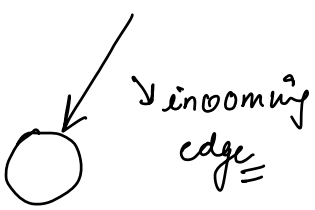
recursion dp

prerequisite



9 2 8 1 10 3 4 5 7 6

{ topological sorting }



degree :- no of edges connected to the node.

incoming degree → no of incoming edges

outgoing degree ⇓ no of outgoing edges

every incoming edge tells you ≡ count of dependencies

indegree

1	2	3	4	5	6	7	8	9	10
2	0	2	1	1	1	3	1	0	0
1	0	1	0	0	0	2	1	0	

2 9 10 8 1 3 5 4 7 6

dynamic array / array / queue / ⋮

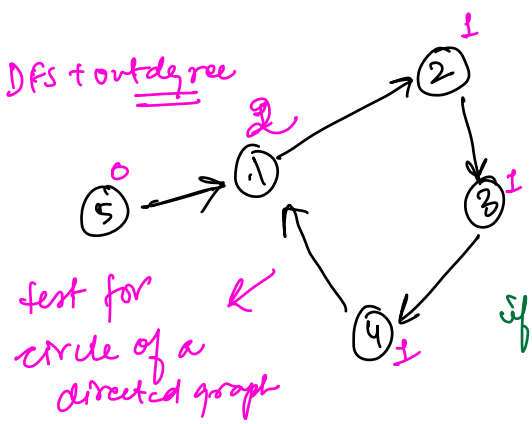
```
indegree[n+1]  // initially → 0
int m;         // no of edges
input(n)

for(int i=0; i<m; i++)
{
    int u,v;
    input(u,v);   // (u → v)
    indegree[v]++;
}
```

const. graph

T.C: n+m
S.C: O(n+m)

```
queue<int> q;
// push every element with indegree 0.
for(int i=1; i<=n; i++)
{
    if(indegree[i]==0)
        q.push(i);
}
int c=0
while(!q.empty())
{
    int node = q.front();
    print(node); c++;
    for(int i=0 → graph(node).size)
    {
        int v = graph[node][i];
        indegree[v]--;
        if(indegree[v]==0)
            q.push(v);
    }
}
if(c != n) return -1
```

DFS + outdegree



test for circle of a directed graph

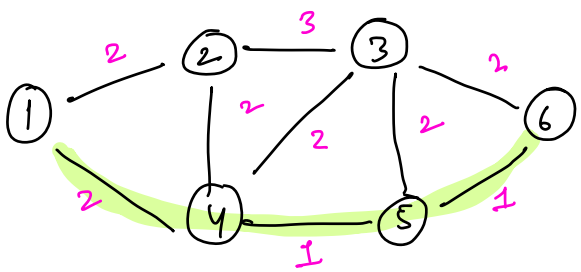
#2 graph → weighted graph

1 ≤ weight of an edge ≤ 3

source → find shortest distance to all nodes from this source

BFS

sum of weights in the path

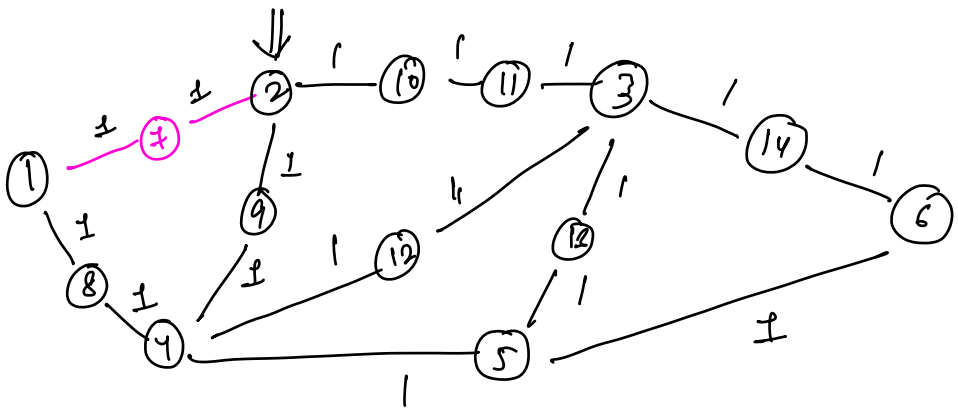


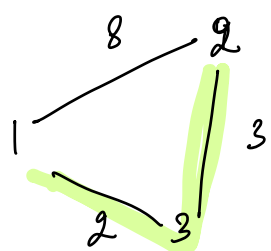
1–6 : 4

unweighted ✓

BFS

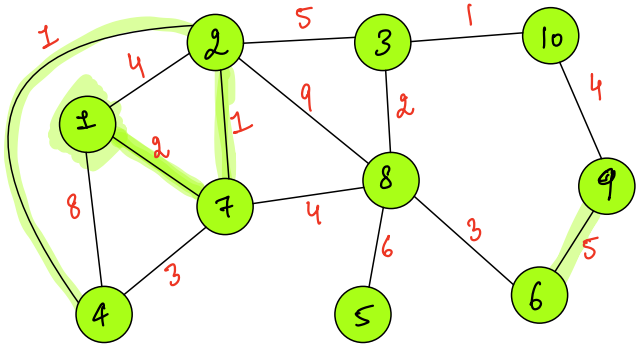
BFS ??





just multiply by x if x is the weight of all edges

Q. +ve weights - edge find shortest path to all other nodes from a given source

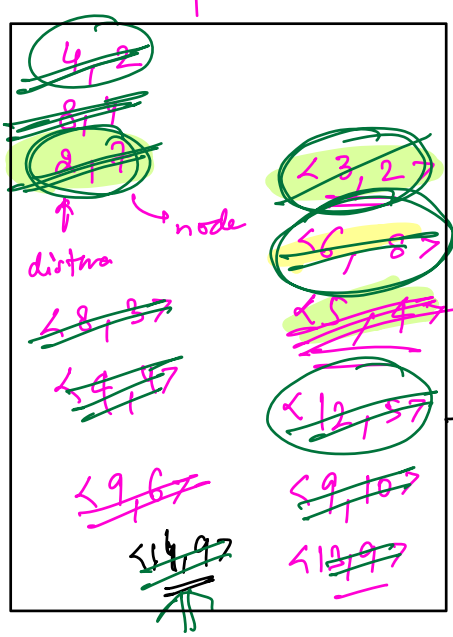


dijkstra's

1	2,4	4,8	7,2		
2	1,4	3,5	7,1	8,9	
3	2,5	8,2	10,1		
4	1,8	2,1	7,3		
5	8,6				
6	8,3	9,5			
7	1,2	2,1	4,3	8,4	
8	2,9	3,2	5,6	6,3	7,4
9	6,5	10,4			
10	3,1	9,4			

1	2	3	4	5	6	7	8	9	10
0	∞	∞	∞	∞	∞	∞	∞	∞	∞
	4 3	8	8 4	12	9	2	6	14 13	9

source = 1
dist[source] = 0



dist[2] > dist[7] + 1
2 + 1

dist[8] > dist[7] + 4
2 + 4 = 6

min-heap → mh;

dist[n+1] =

// initialize → INT_MAX (∞)

// source

dist[source] = 0;

mh.insert({0, source});

while(!mh.empty()) size()

{

{d, node} = extractmin();

if(d == dist[node])

{

for(i=0; i< graph[node].size(); i++)

{

{v, weight} = graph[node][i];

if(dist[v] > dist[node] + weight)

{

dist[v] = dist[node] + weight;

mh.insert({dist[v], v});

}

}

}

}

T.C: $O(E * \log E)$

treeset

ordered_set

- Topological
- BFS — 1-3
- BFS — equal weight
- Dijkstra