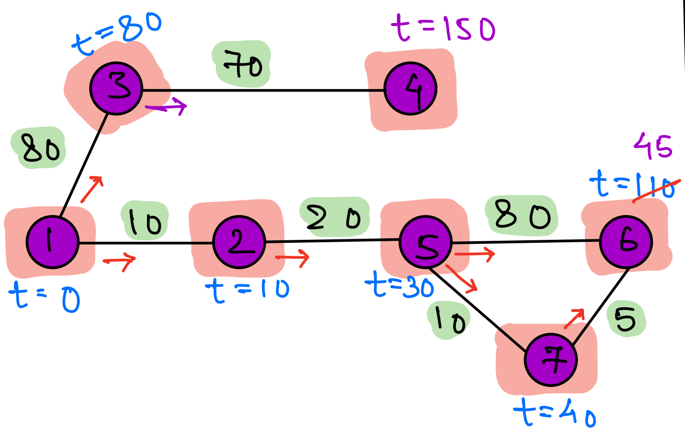
#



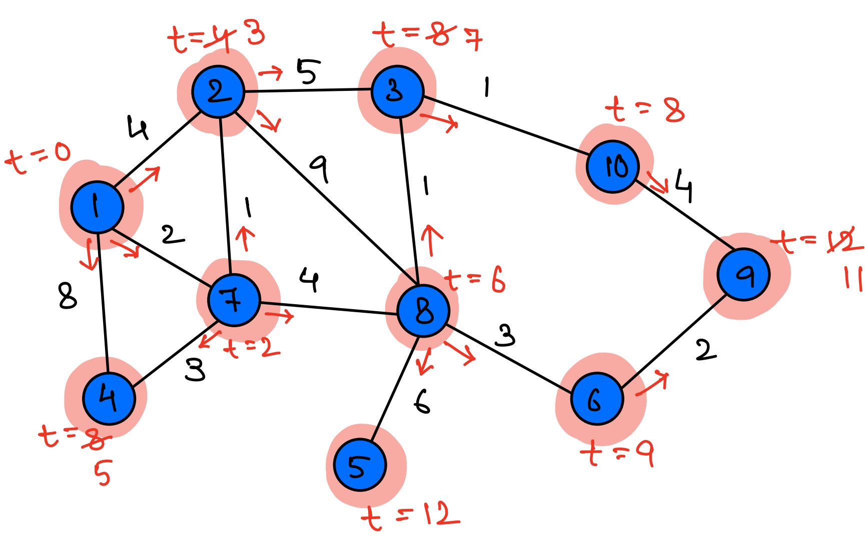
ans = 150

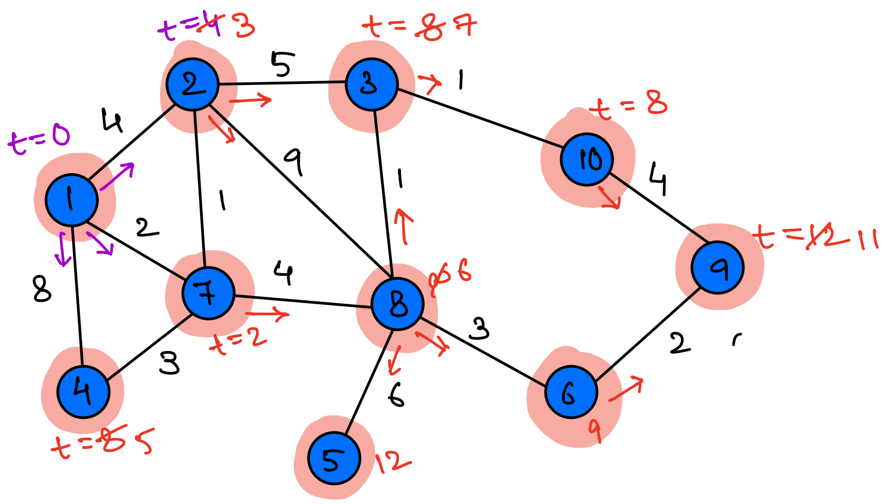
1) Each node is a Petrol bunker.
2) Edge indicates connection b/w 2 bunkers & length is also given.
3) Initially bunker 1 will blast
4) Petrol burns at 1Km/min
5) Calculate the time at which all the bunkers will be blasted.

Algorithm :- Dijkstra's

1) Node with min time will blast first

2) After a node is blasted, update the time of it neighbours node.





	0	1	2	3	4	5	6	7	8	9	10
time[11]:	x	0	~~∞~~	~~∞~~	~~∞~~	~~∞~~	~~∞~~	~~∞~~	~~∞~~	~~∞~~	~~∞~~
			~~4~~ 3	~~8~~ 7	~~8~~ 5	12	9	2	6	~~12~~ 11	8

Min Heap < pair < int , int > >
(time, node)

~~{0,1}~~
~~{4,2}~~ x
~~{8,4}~~ x
~~{2,7}~~
~~{3,2}~~
~~{5,4}~~ x
~~{6,8}~~
~~{8,3}~~ x
~~{7,3}~~
~~{9,6}~~
~~{12,5}~~
~~{8,10}~~
~~{12,9}~~
~~{11,9}~~

* 1) find no with min time (Min Heap)

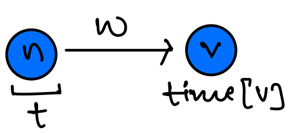
2) Iterate on neighbours of the blasted node & update their blast time.

3) If the node is already blasted skip it.

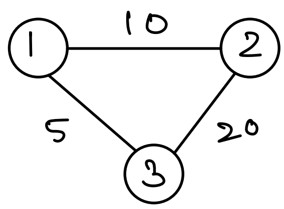
Time in Heap > Time in arr[]

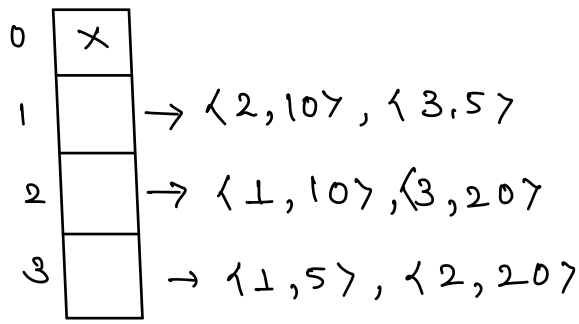
N → Nodes, E → Edges.

```
int blastTime ( list<pair<int int> > g[], N, src, ) {
                                             dest
    int time[N+1] = {∞};   time[src] = 0;
    MinHeap< pair< int, int > > mh;
                   time  node
    mh.insert( {0, src} );

    while( mh.size() > 0 ){

        Pair<int, int> u = mh.getMin();
        mh.deleteMin();
        t = u.first , n = u.second;
        if( t > time[n] ){
            // Node 'n' is already burnt.
            Continue;
        }
        // Blast node (n) & update the neigbours.
        for( i=0; i< g[n].size(); i++ ){
            Pair<int, int> ele = g[n][i];
            V = ele.first;
            w = ele.second;
            if( t + w < time[V] ){
                time[V] = t + w;
                mh.insert( {time[V], v} );
            }
        }
    }
    return time[dest];
}
```



if (t+w < time[v])
time[v] = t+w





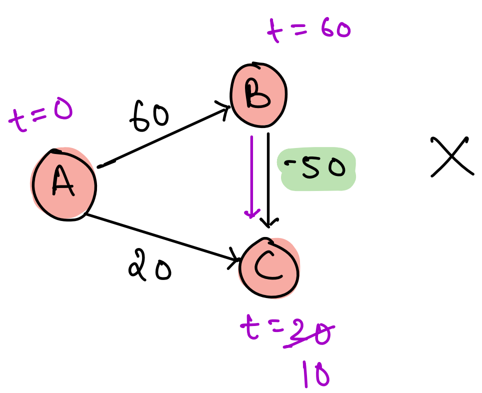
TC : $O(E * (2 \log E + 1)) \Rightarrow O(E \log E)$

for every edge ⇒ getMin() ⇒ $O(1)$
insert() ⇒ $O(\log E)$
deleteMin() ⇒ $O(\log E)$

SC : $O(E)$

⇒ Dijkstra's Shortest Path Algorithm

→ BFS :- Shortest path in unweighted graph
Dijkstra's :- Shortest path in weighted graph



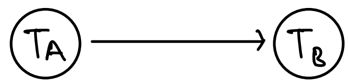
Dijkstra's

⇒ Weighted Graph

⇒ Weights should be +ve.

{ i) Bellman Ford
ii) Floyd's Algo } Weight < 0

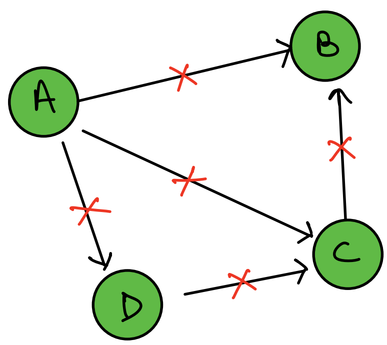
Topological Sorting



⇒ T_B depends on T_A

⇒ first complete T_A & the complete T_B.

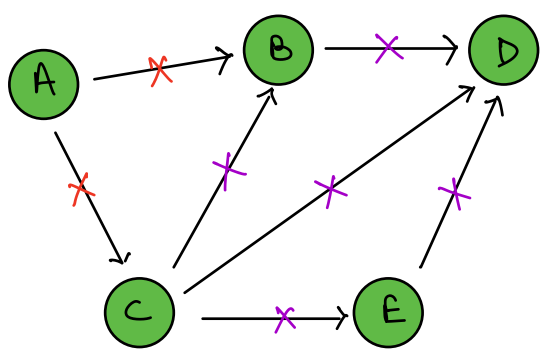
Ex



A, D, C, B

Incoming Edge = Dependency

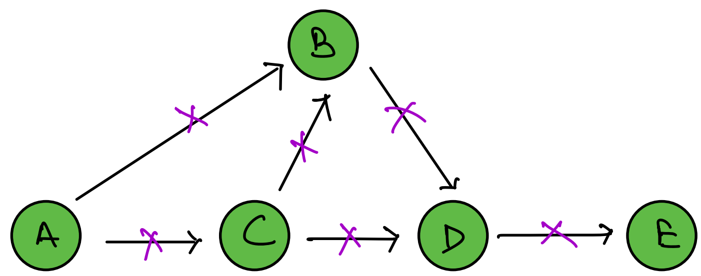
Ex



A, C, B, E, D

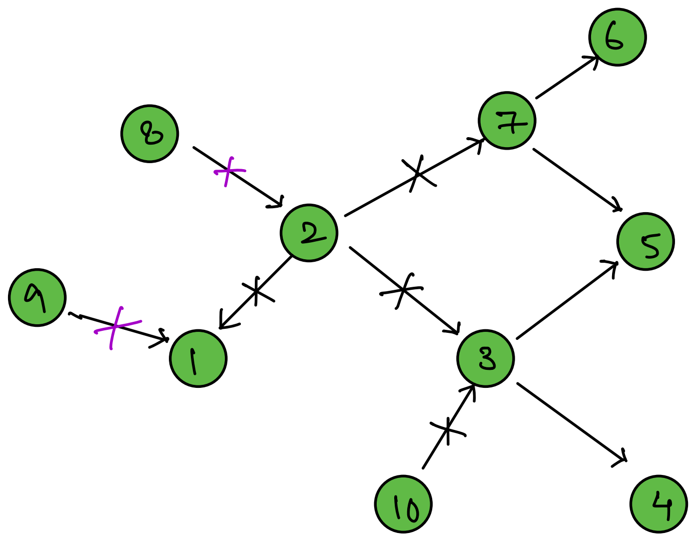
	A	B	C	D	E
dependency / ⇒ Incoming Edges	0	~~2~~ ~~1~~ 0	~~1~~ 0	~~3~~ ~~2~~ ~~1~~ 0	~~1~~ 0

Ex



A, C, B, D, E

Ex



0	x
1	→ { }
2	→ 7, 3, 1
3	→ 5, 4
4	→ { }
5	→ { }
6	→ { }
7	→ 5, 6
8	→ 2
9	→ 1
10	→ 3

int inc[11] :

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

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

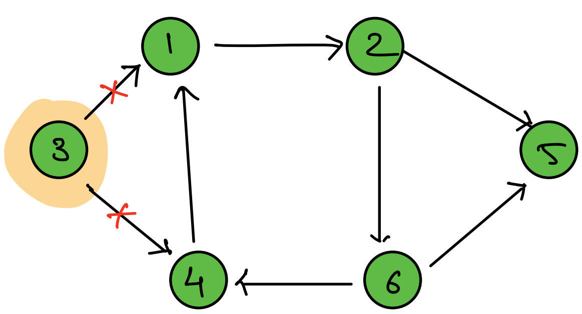
Queue. 8, 9, 10, 2, 1, 7, 3, 6, 4, 5

1) Get the count of dependencies for all nodes
2) If dependency count for any node = 0 :
   → Add in the queue
   → resolve the dependencies of it's neighbours.
   ↳ Adj. List

0	X
1	→ {}
2	→ 7, 3, 1
3	→ 5, 4
4	→ {}
5	→ {}
6	→ {}
7	→ 5, 6
8	→ 2
9	→ 1
10	→ 3

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

Ex



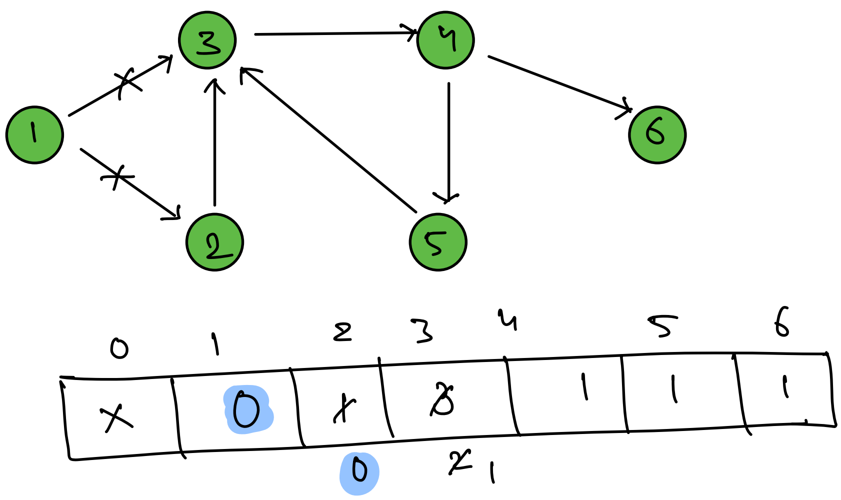
3,

0	1	2	3	4	5	6
X	~~2~~	1	0	~~2~~	2	1
	1			1		

→ If we are not able to resolve all the dependencies in a Directed graph
⇒ **Cycle**

⇒ Cycle detection in directed graph.

⇒ If dependency array is containing a non zero value, then there's a cycle in the directed graph.



Q ~~1~~, ~~2~~

⇒ Cycle

—— * ——

Topological Sorting Code

```
void topological( list<int> g[], int N){
    int in[N+1];
    for(i=1; i<=N; i++){          => O(E)
        for(j=0; j<g[i].size(); j++){
            int v = g[i][j];
            in[v]++;
        }
    }
    queue<int> q;   // insert all nodes with 0
                    // dependency.
    for(i=1; i<=N; i++){          => O(N)
        if(in[i]==0) q.insert(i);
    }
    while(q.size()>0){            => O(E)
        int u = q.front();
        print(u);
        q.dequeue();
        for(i=0; i<g[u].size(); i++){
            v = g[u][i]
            in[v]--;
            if(in[v]==0) q.insert(v);
        }
    }
}
```

TC: $O(N+E+E) = O(2E+N)$

$= O(E)$

SC: $O(N+E)$