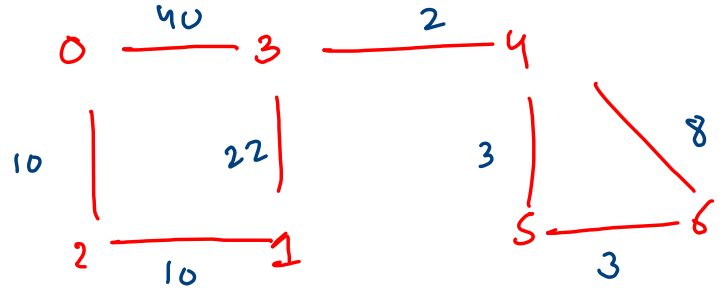
MST -> minimum spanning tree

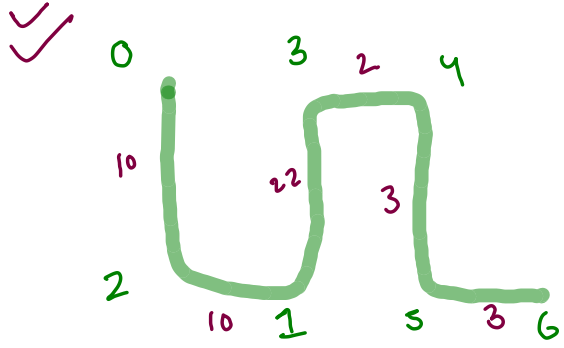
↳ connected, acyclic, visit all vertices



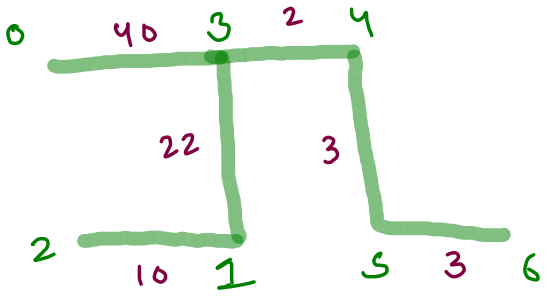
(i) Tree -> connected & acyclic graph.

(ii) span -> to have all the vertices

(iii) minimum (wt)

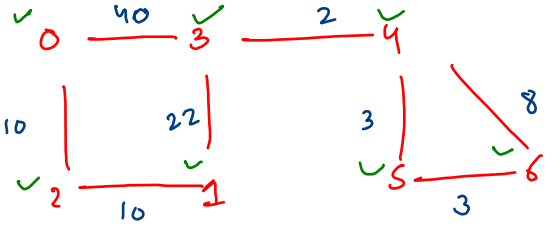


,



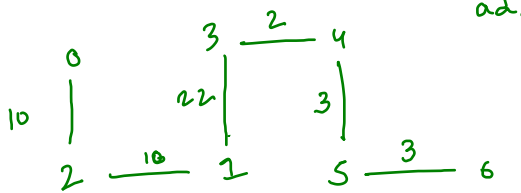
= 10 + 10 + 22 + 8 = 50

Prim's

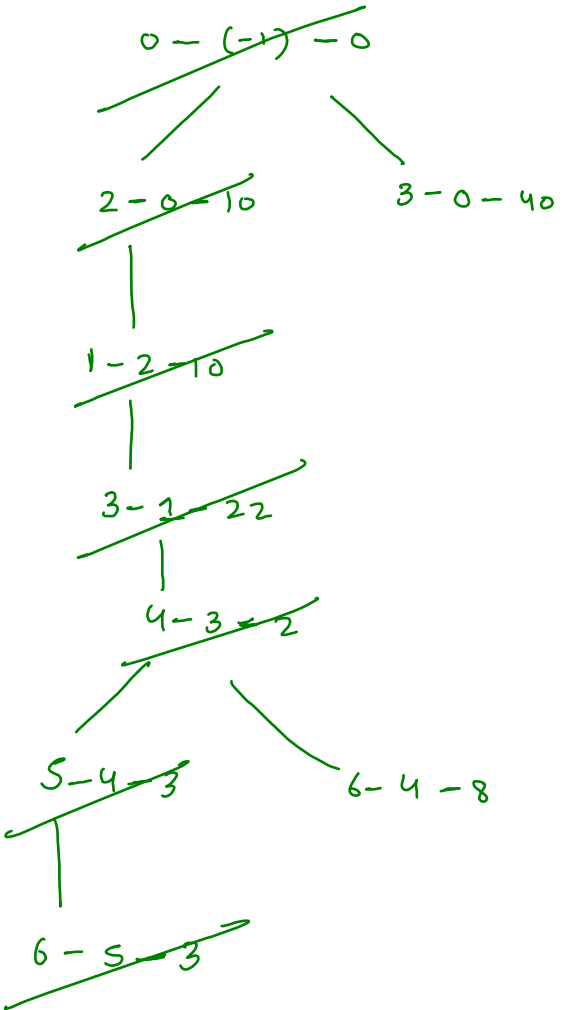


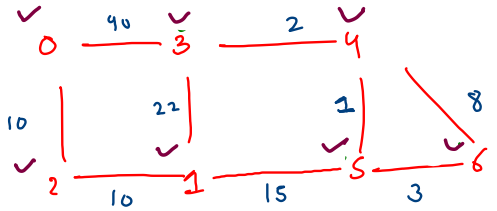
remove
mark*
work
add nby*

MST

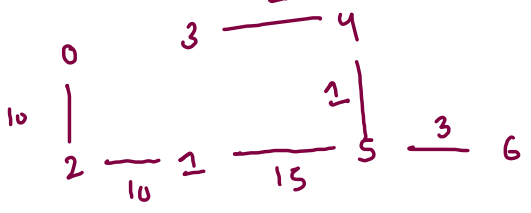


vtx - aq. vtx - wt

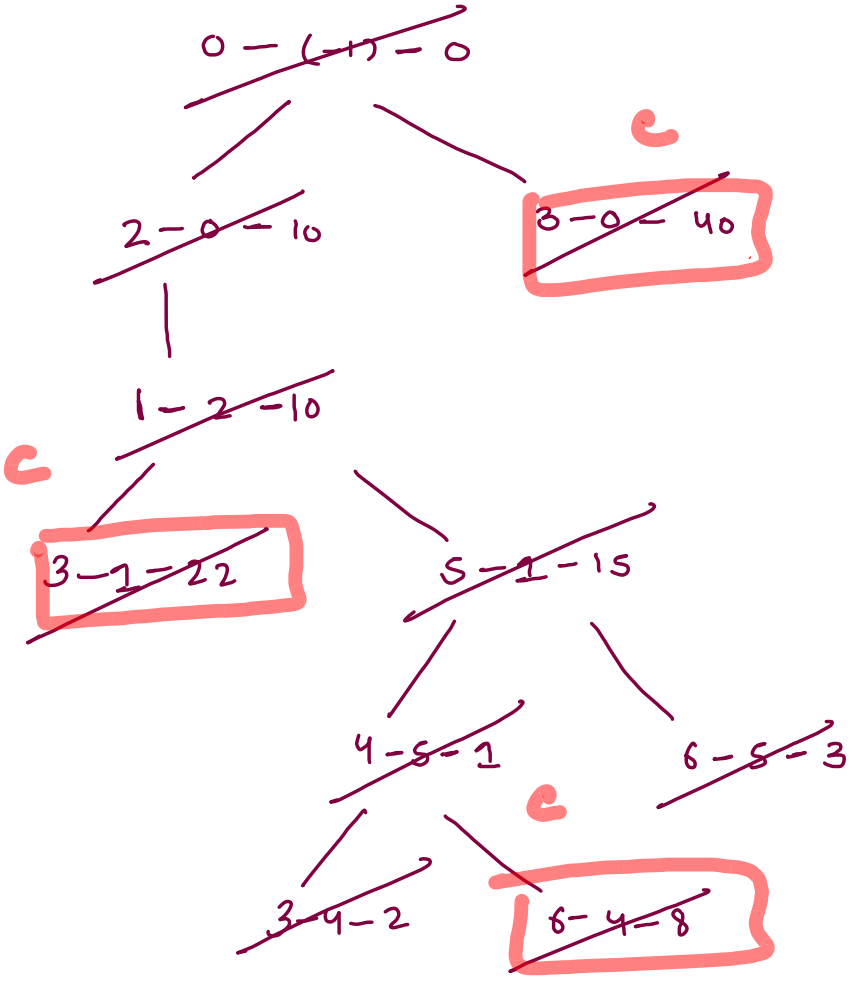


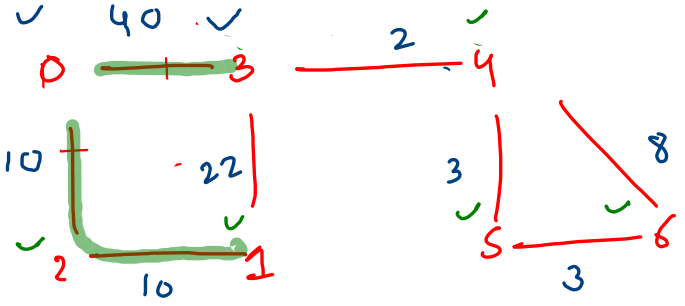


MST




```
while(pq.size() > 0) {
    //remove
    Pair rem = pq.remove();

    //mark*
    if(vis[rem.vtx] == true) {
        continue;
    }
    vis[rem.vtx] = true;

    //work
    if(rem.aqv != -1)
    System.out.println("[" + rem.vtx  + "-" + rem.aqv + "@" + rem.wt + "]");

    //add unvisited nbr
    for(Edge edge : graph[rem.vtx]) {
        int nbr = edge.nbr;

        if(vis[nbr] == false) {
            pq.add(new Pair(nbr,rem.vtx,edge.wt));
        }
    }
}
```





0 via 0 @ 0

2 via 02 @ 10

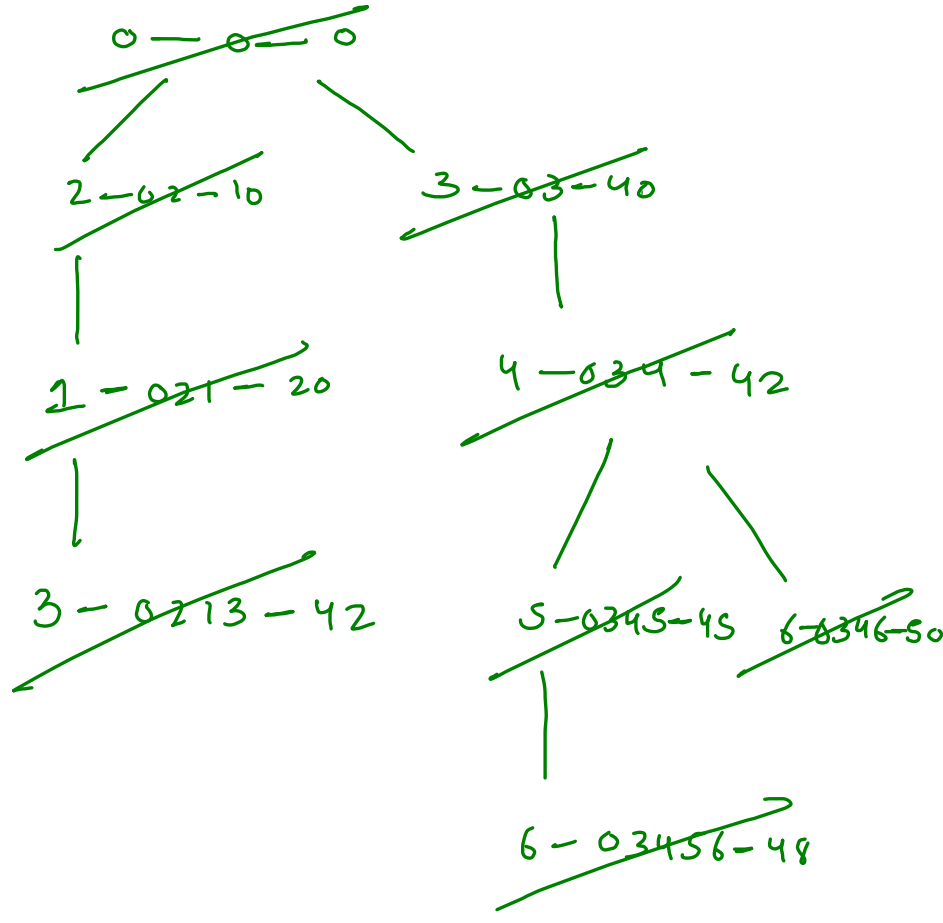
1 via 021 @ 20

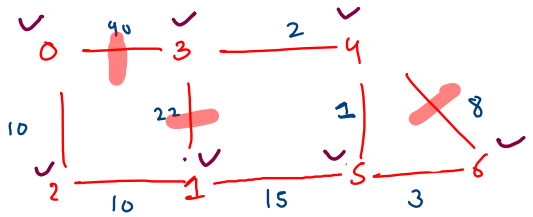
3 via 03 @ 40

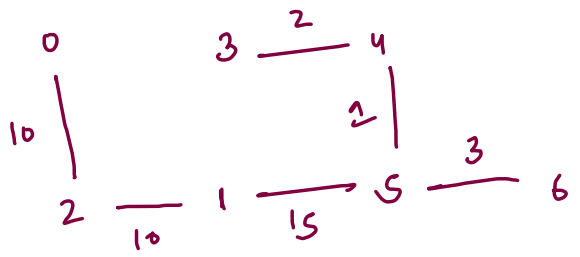
4 via 034 @ 42

5 via 0345 @ 45

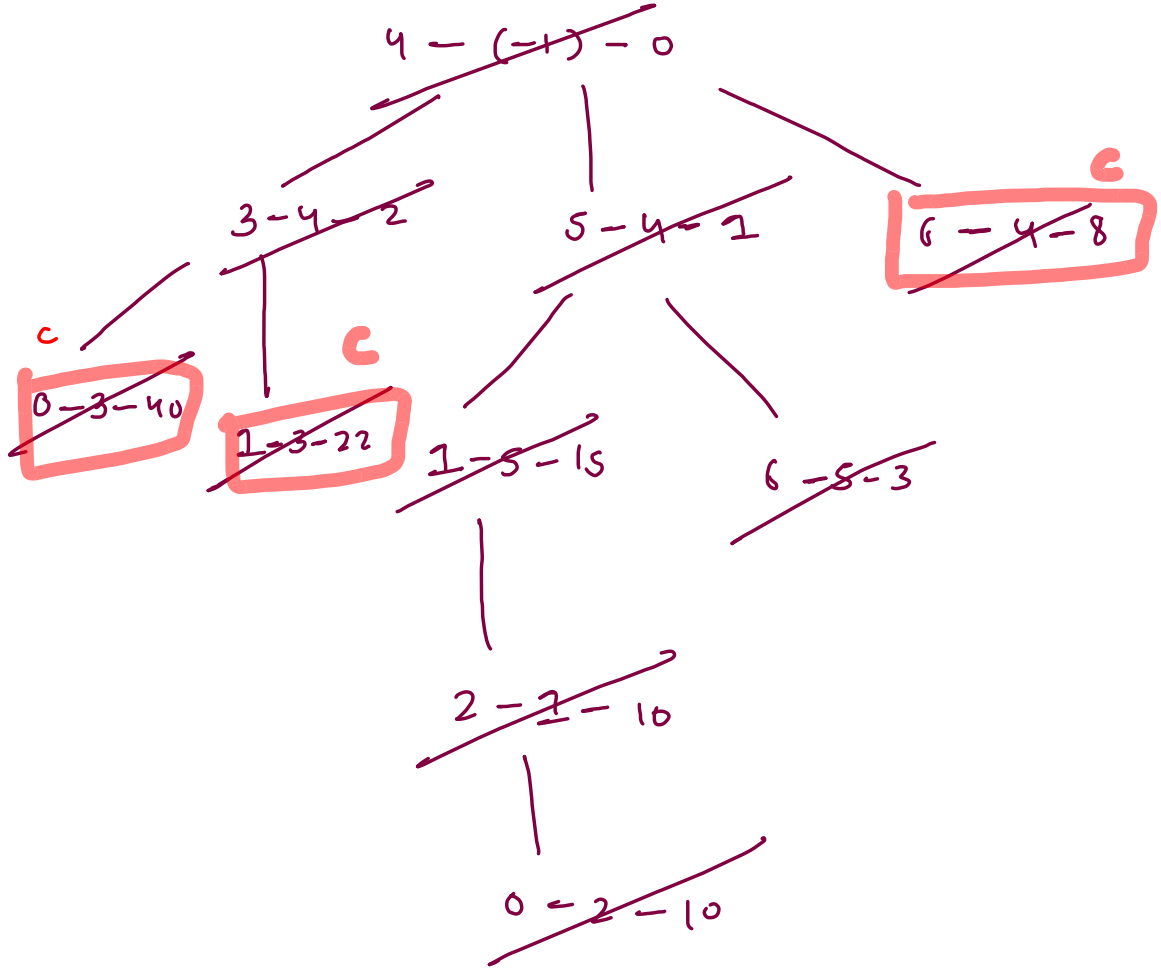
6 via 03456 @ 48







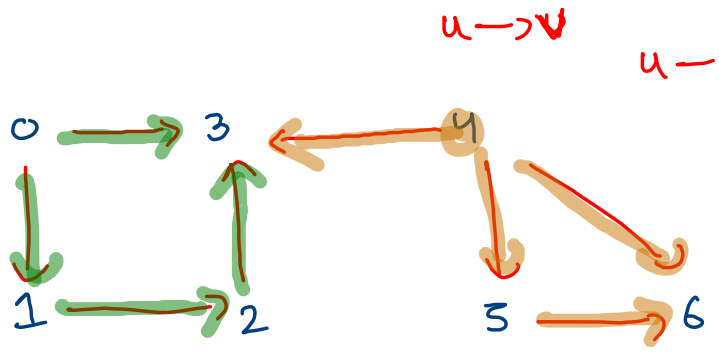
dijkstra	prim
(i) single src to all dest min cost path.	(i) min spanning tree.
(ii) paths matters (wsf)	(ii) edge matters (wt)



topological sort : DAG (directed acyclic graph)

u → v

u is dependent on v.



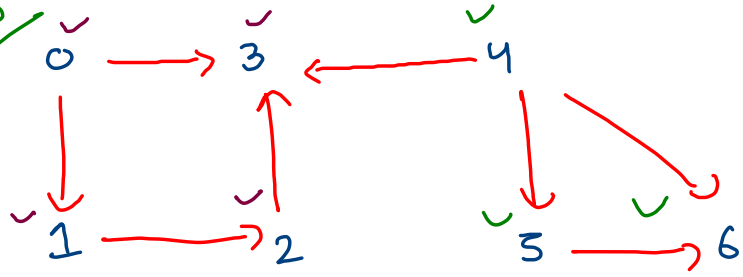
4 5 6 0 1 2 3

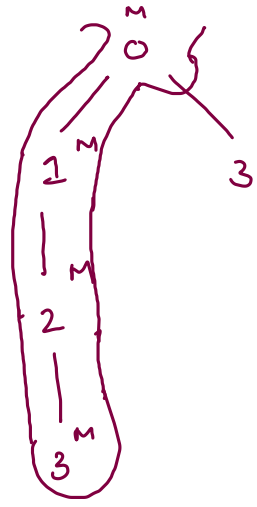
3 2 1 0 6 5 4

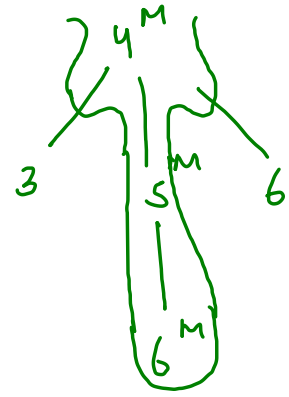
topological sort : permutation of vertices such that for every u → v edge u should come v.

order of work should be reverse of topological sort.

Algo



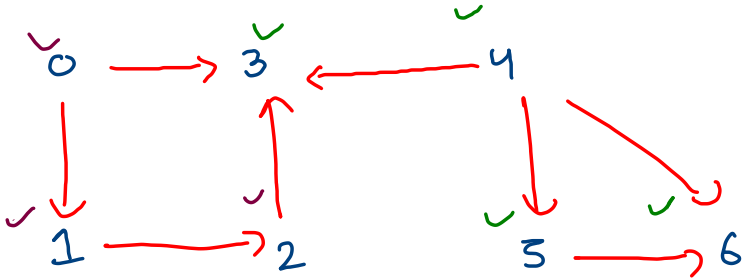


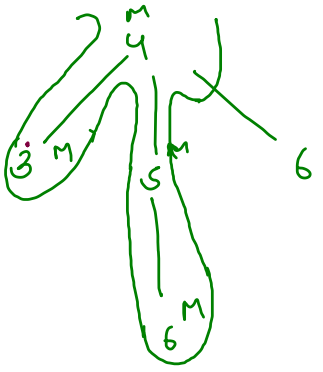


0 -) travel

4 -) travel

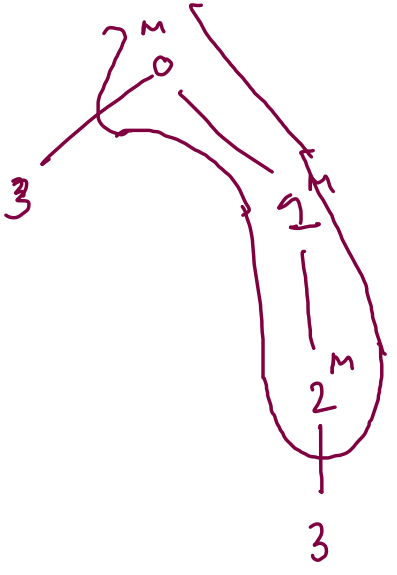
4
5
6
0
1
2
3

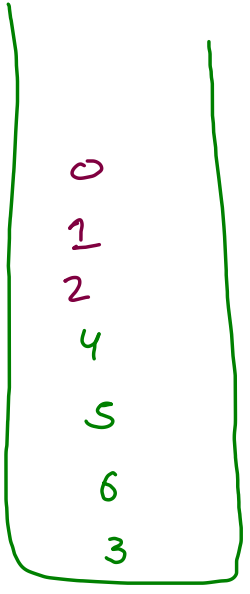


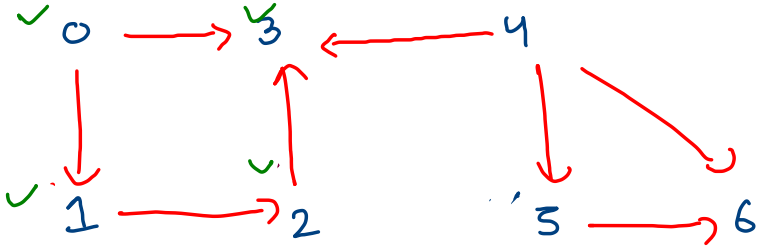


travel (4)

travel (0)







post → order of compilation

src → X

```
public static void travel(ArrayList<Edge>[]graph,int src,boolean[]vis,Stack<Integer>st)
    vis[src] = true;

    for(Edge edge : graph[src]) {
        int nbr = edge.nbr;

        if(vis[nbr] == false) {
            travel(graph,nbr,vis,st);
        }
    }

    st.push(src);
}
```

5
4

0
1
2
3
4

```
public static void topologicalSort(ArrayList<Edge>[]graph) {
    boolean[]vis = new boolean[graph.length];
    Stack<Integer>st = new Stack<>();

    for(int src = 0; src < graph.length;src++) {
        if(vis[src] == false) {
            travel(graph,src,vis,st);
        }
    }

    //print topological sort
    while(st.size() > 0 ){
        System.out.println(st.pop());
    }
}
```

(i) Creation

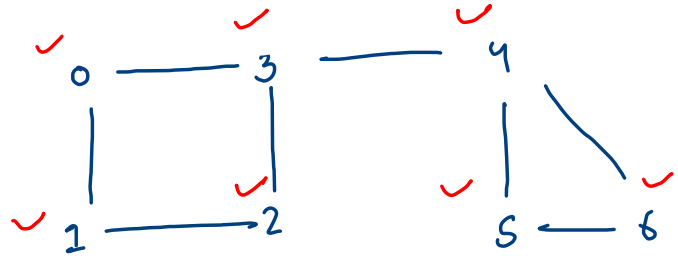
(ii) DFS , BFS

(iii) DFS —> hasPath, allPath, topological sort, gcc

(iv) BFS —> cyclic, bipartite, spread infection

(v). PQ —> Dijkstra, prim

iterative DFS



- 2 via 2
- 3 via 23
- 4 via 234
- 6 via 2346
- 5 via 23465
- 0 via 230
- 1 via 2301

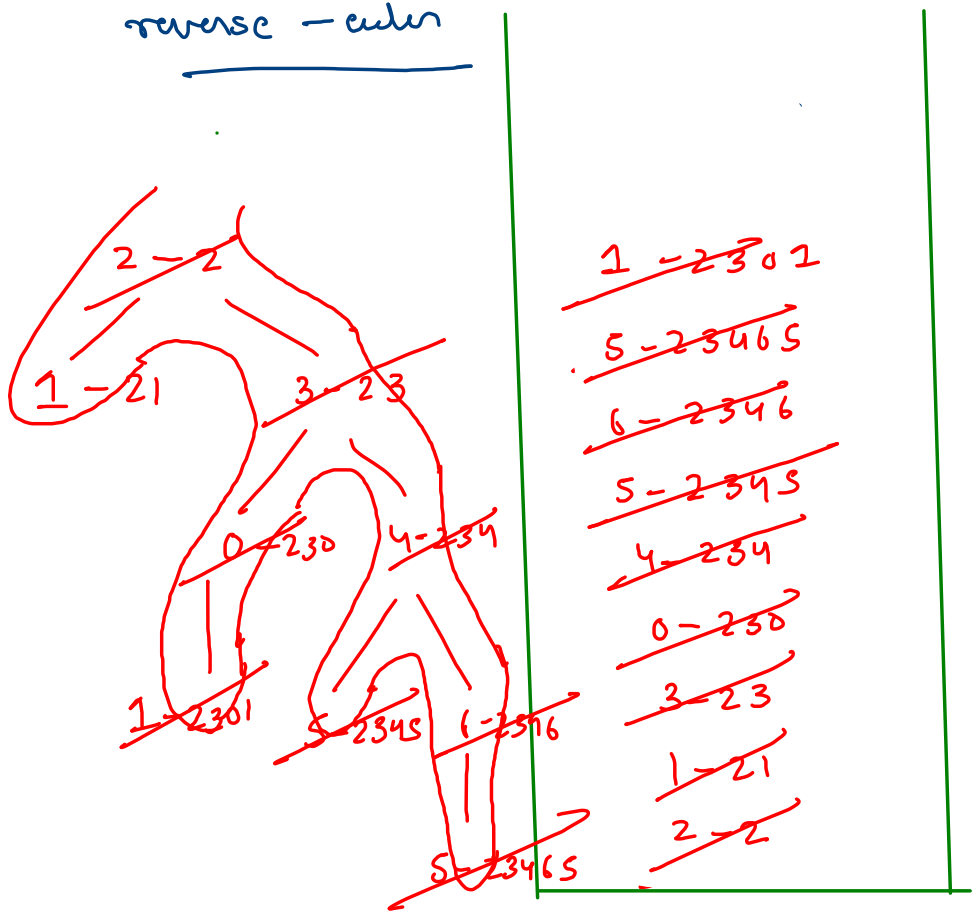
remove

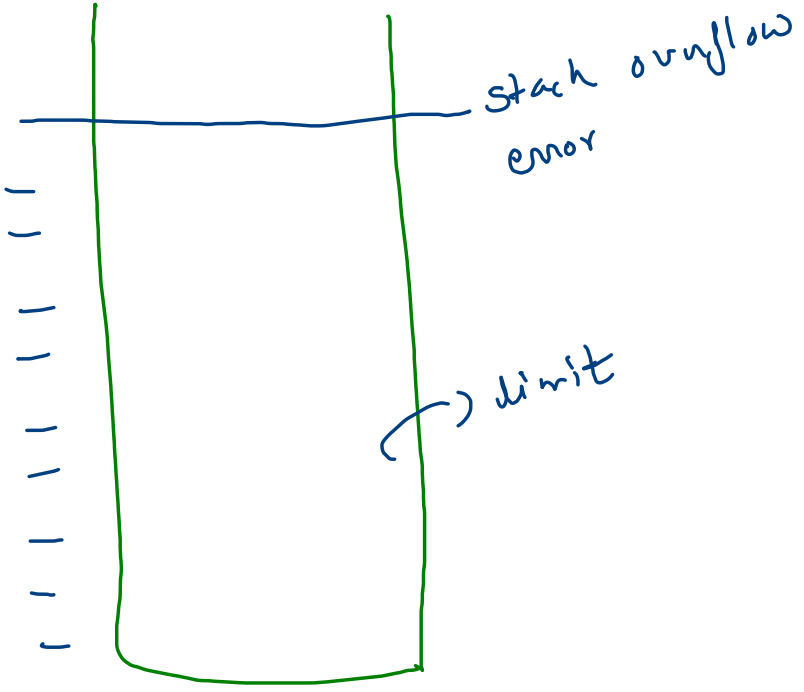
mark *

work

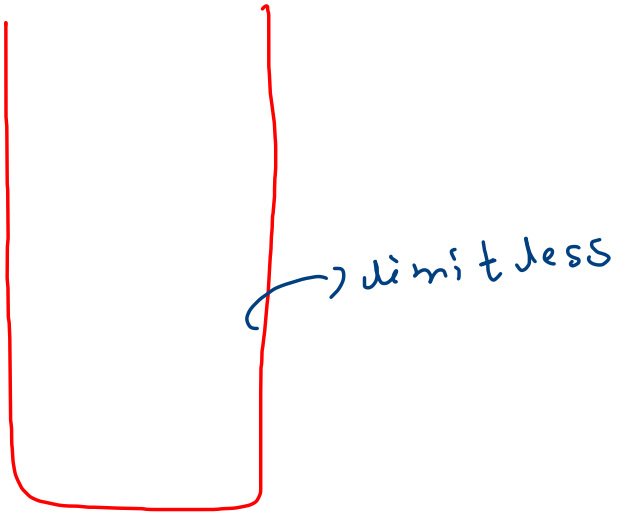
add nbr

reverse - euler





call stack



Stack