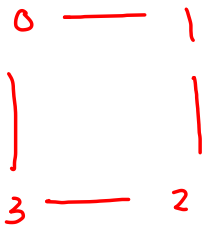
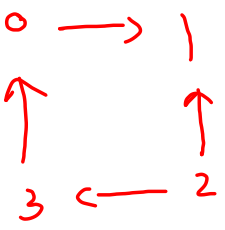
Kosaraju Algorithm : Strongly connected comp

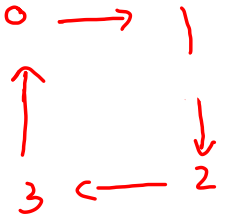
directed comp



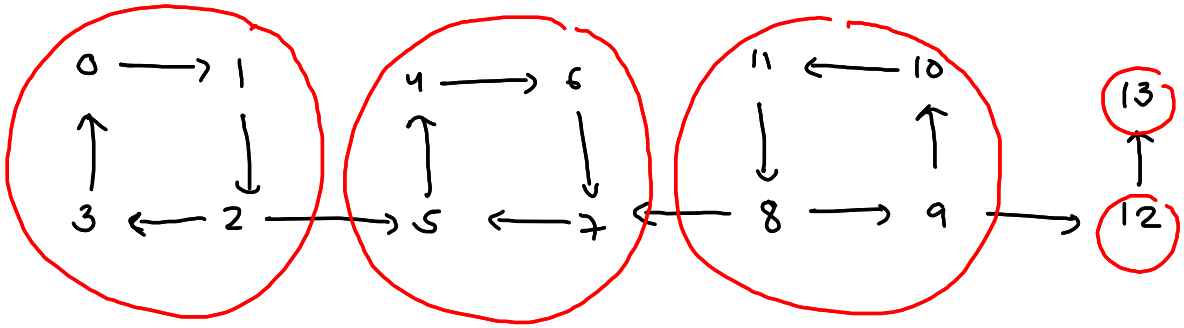
Connected comp



not a Strongly conn. comp



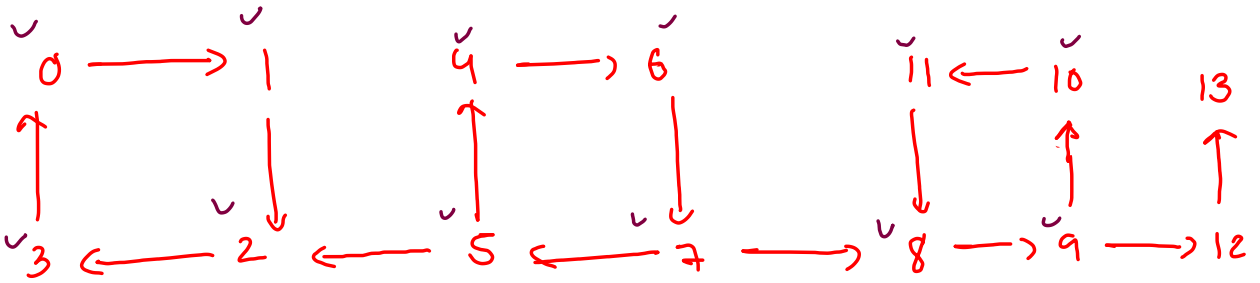
Strongly connected comp.



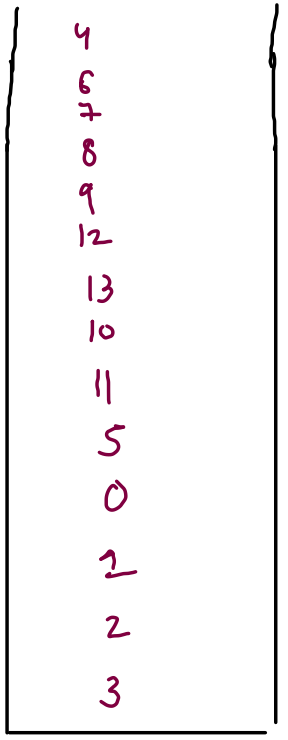
5 SCC

Strongly connected comps-

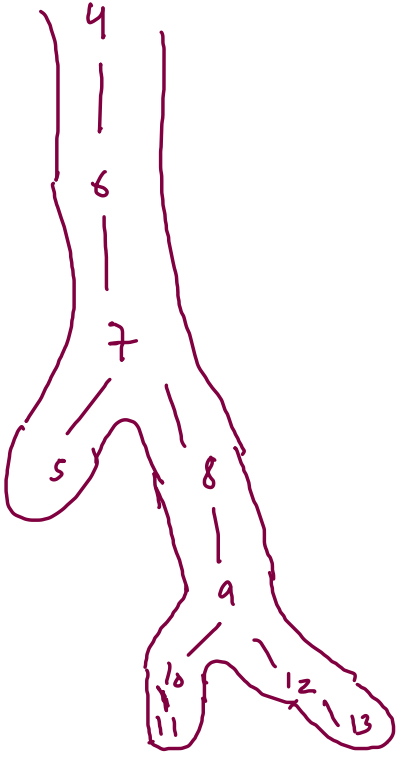
no of cycles != no. of SCC

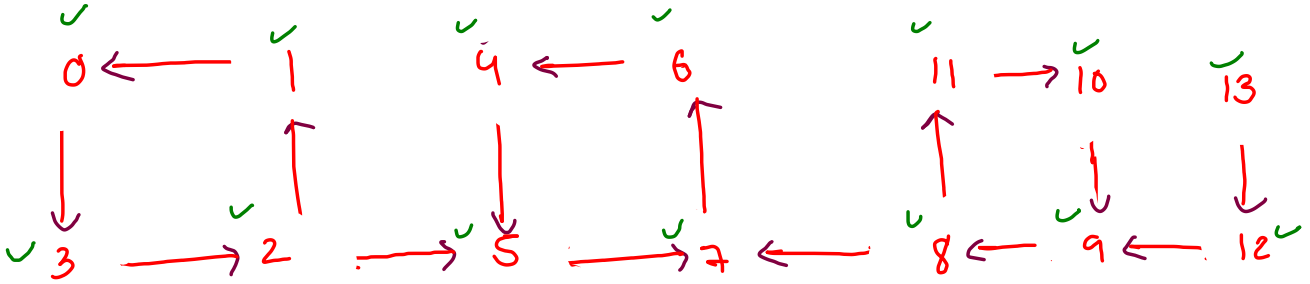

0
1
4
6
11
10
13
3
2
5
7
8
9
12

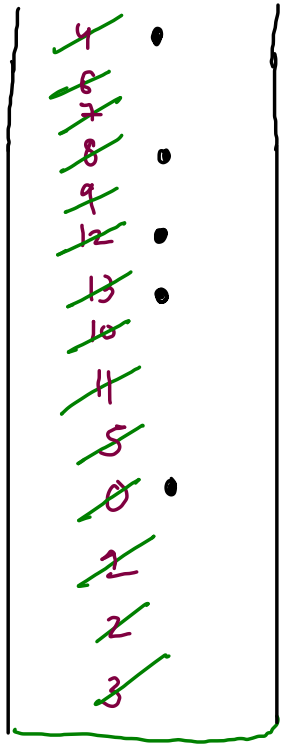
① · DFS

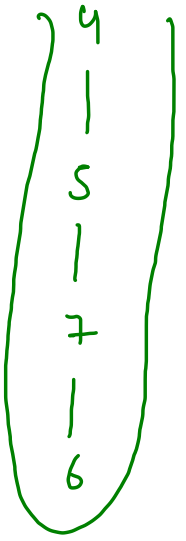

4
6
7
8
9
12
13
10
11
5
0
1
2
3

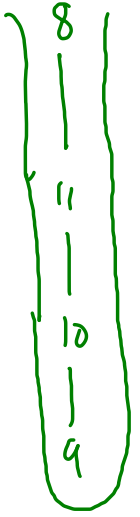

0
1
2
3


4
6
7
5
8
9
10
11
12
13


0
1
4
6
11
10
13
3
2
5
7
8
9
12

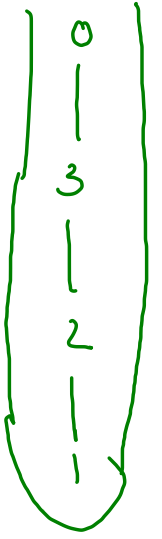

4
6
7
8
9
12
13
10
11
5
0
1
2
3

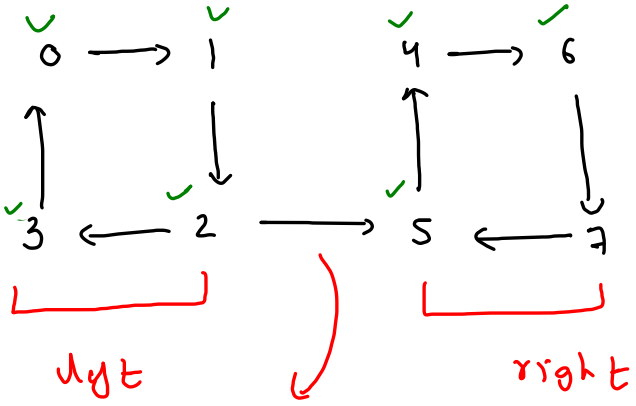

4
5
7
6


8
11
10
9

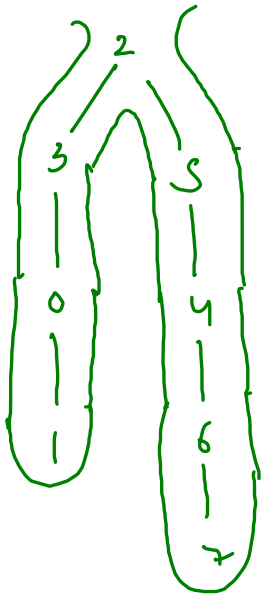
12

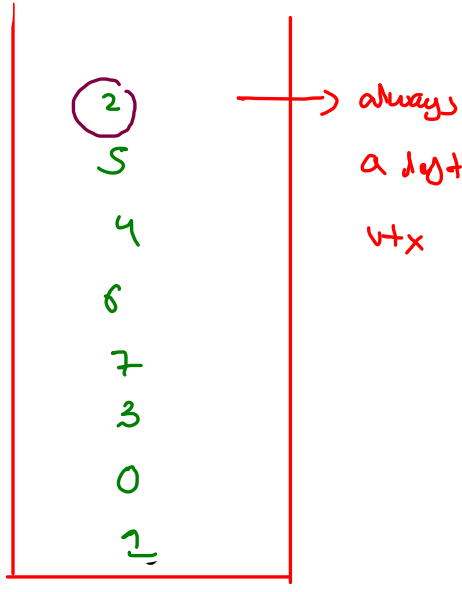
13


0
3
2
1



why ?

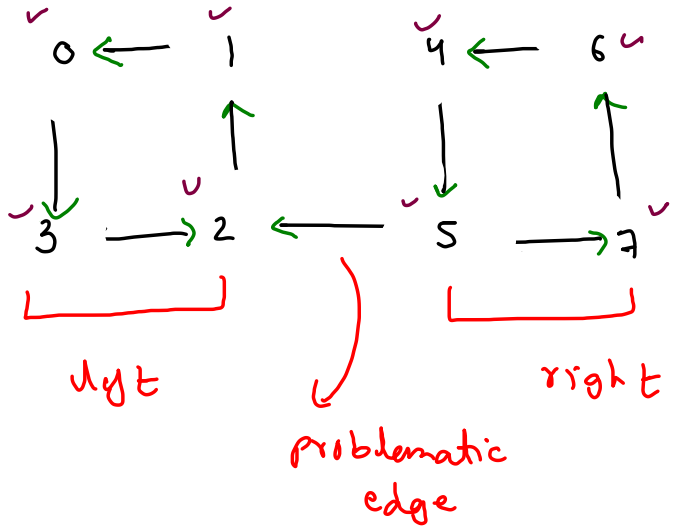


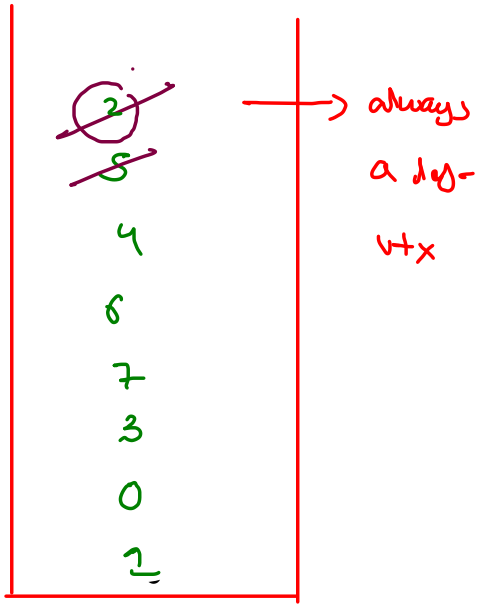


① · performs complete dfs and add vertices in stack in postorder.

② · reverse this graph.

③ · now travel Dfs using stack (no. of dfs == scc)

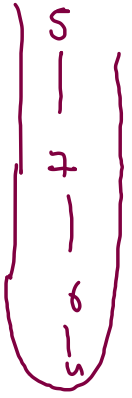

0
1
4
6
3
2
5
7
left
right
problematic edge

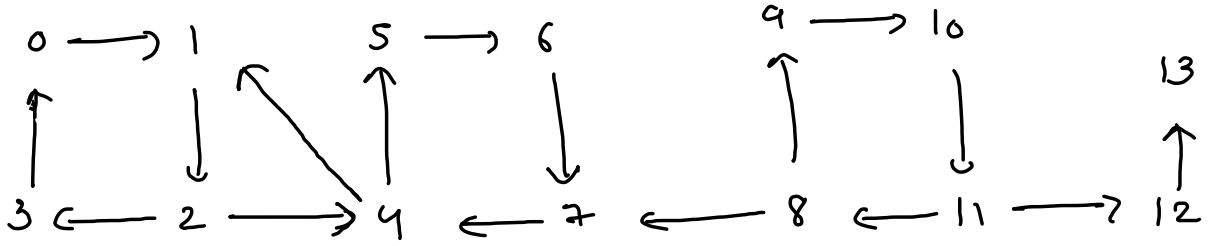

2
5
4
6
7
3
0
1
always a dp-vtx

2 scc

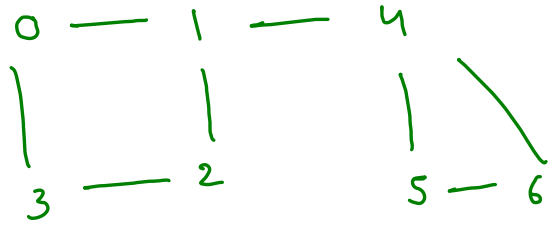
2


2
1
0
1
3


5
7
6
4


0
1
5
6
9
10
13
3
2
4
7
8
11
12

Articulation Point



1 & 4 are

articulation points.

brute force :

remove each vtx → no. of connected comps

discovery time &

low time

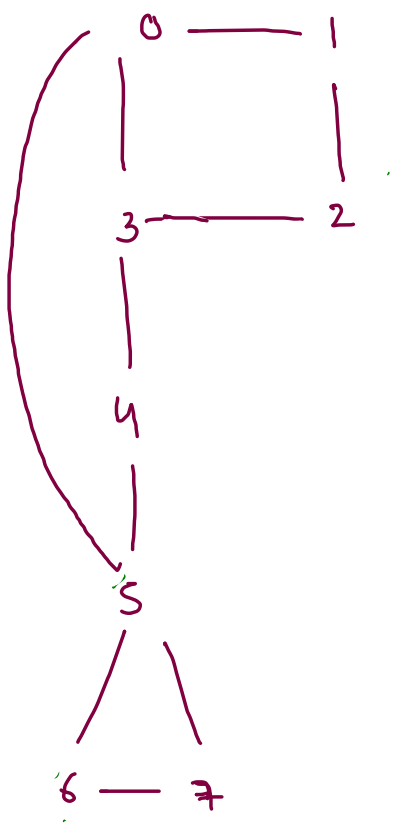
5 is ap

4 is ap

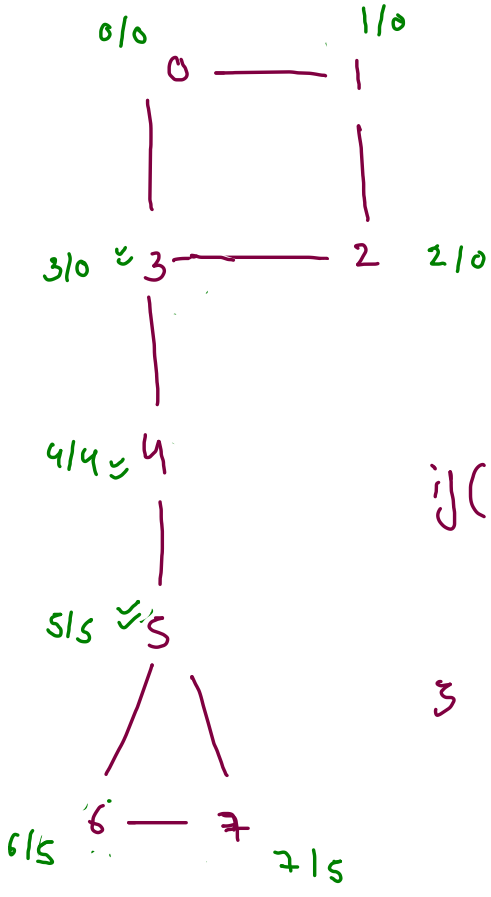
3 is ap

discovery time : the no. at which you are travelled in dfs.

low time : except your parents, the lowest vertex you know (discovered earlier)




```
if (nbr == par) {
    // ignore
}
else if (nbr == unvisited) {
    ①. go to nbr.
    ②. low[src] = Math.min(low[nbr], low[src])
}
else if (nbr == visited) {
    low[src] = Math.min(low[src], disc[nbr])
}
```

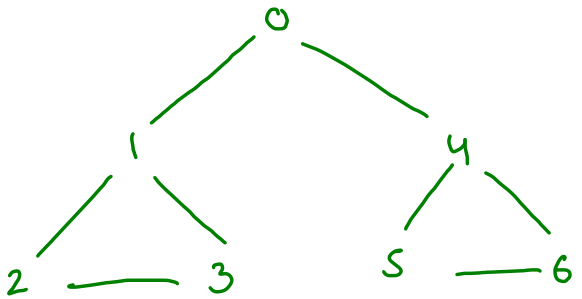


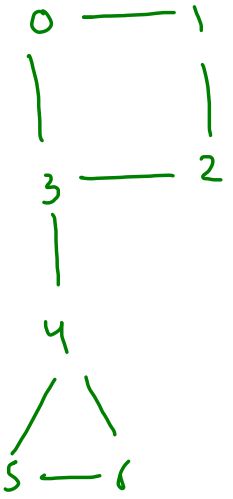
src -> manage differently

```
if (disc[src] <= low[nbr]) {
    src is an ap;
}
```

```
if (nbr == par) {
    // ignore
}
else if (nbr == unvisited) {
    ①. go to nbr.
    ②. low[src] = math.min(low[nbr], low[src])
}
else if (nbr == visited) {
    low[src] = math.min(low[src], disc[nbr])
}
```

0 is ap





0 is not ap