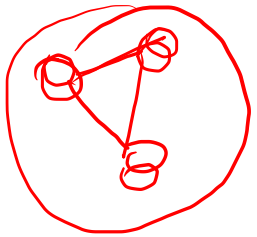
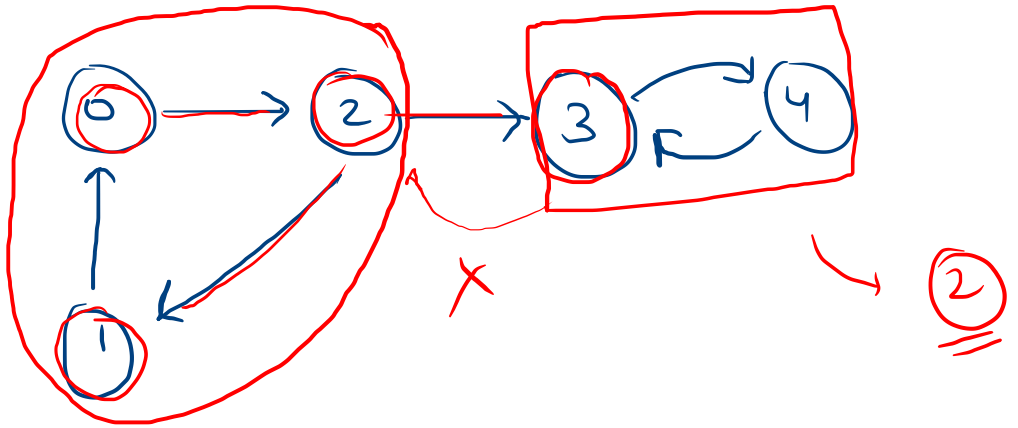


Kosaraju's Algo

[Strongly Connected Components]

if a set of vertex form connection such that every vertex is reachable from each other

I/p :

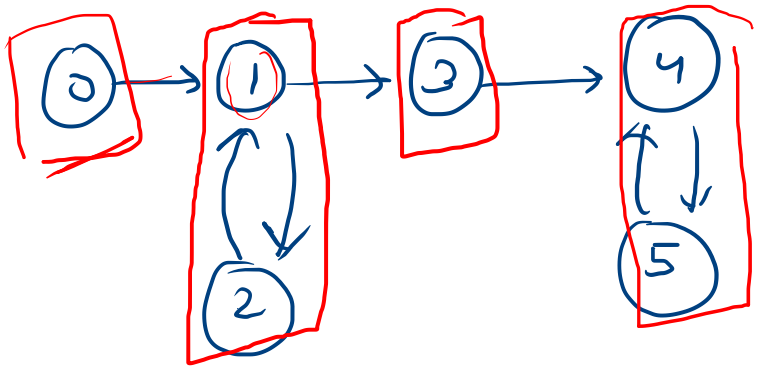


o/p :

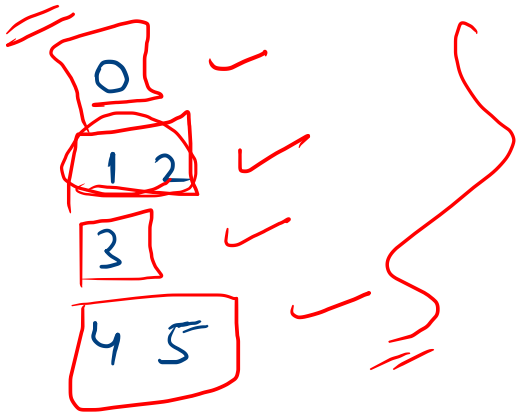
0 1 2 → 1

3 4 → 1

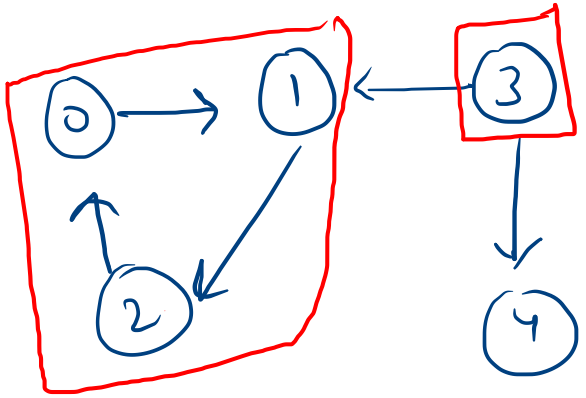
I/p :



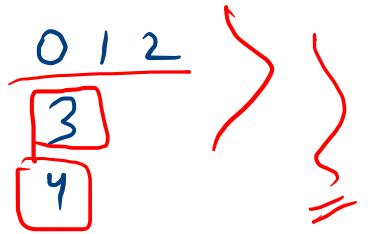
o/p :



I/p :



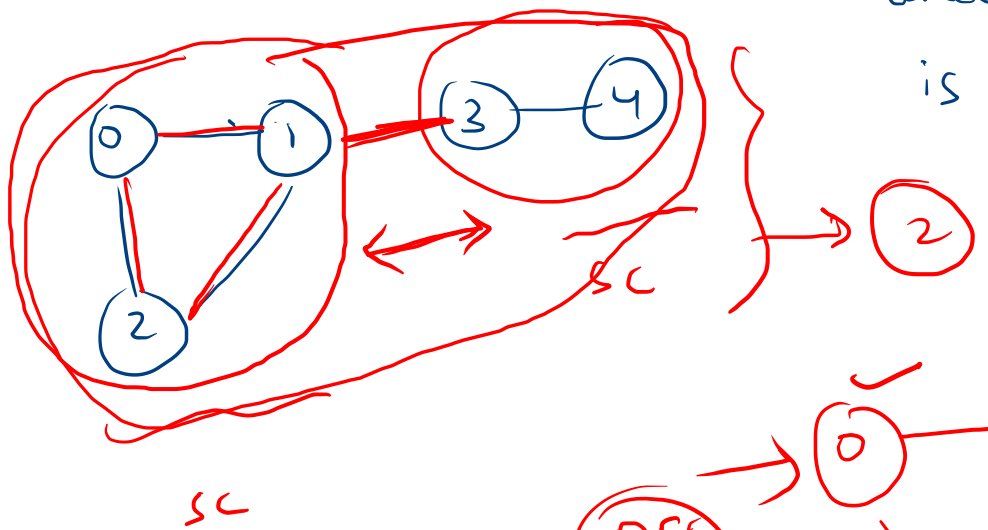
o/p :

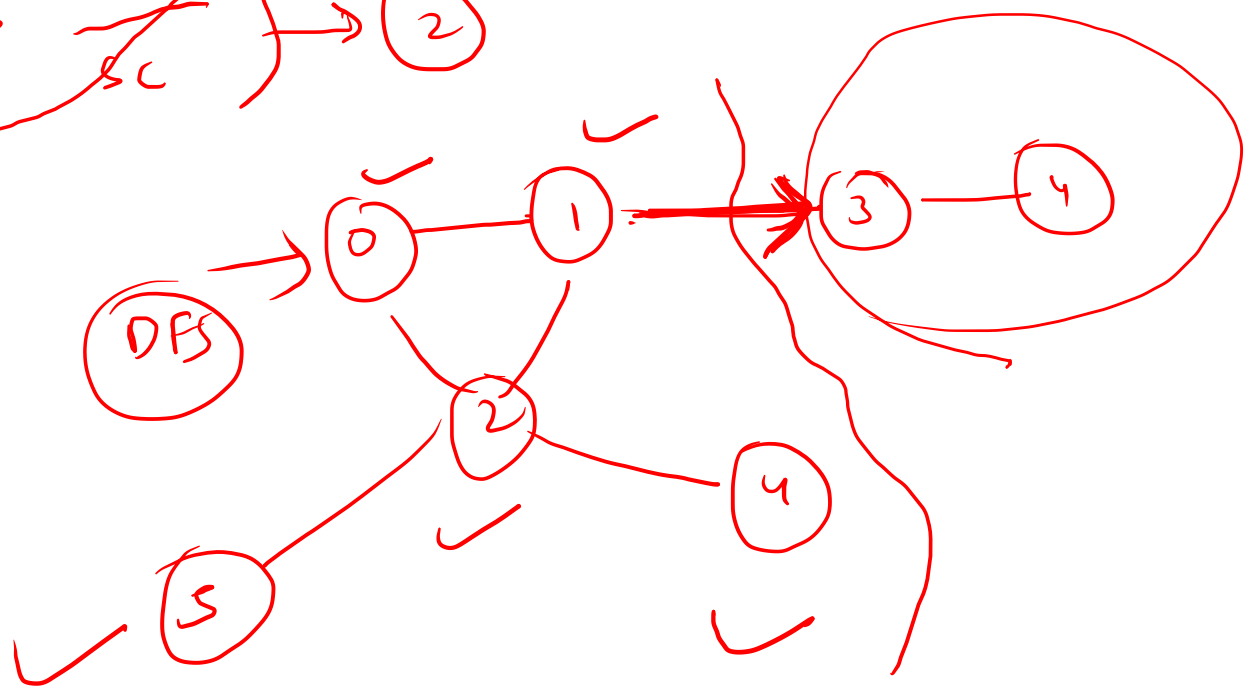


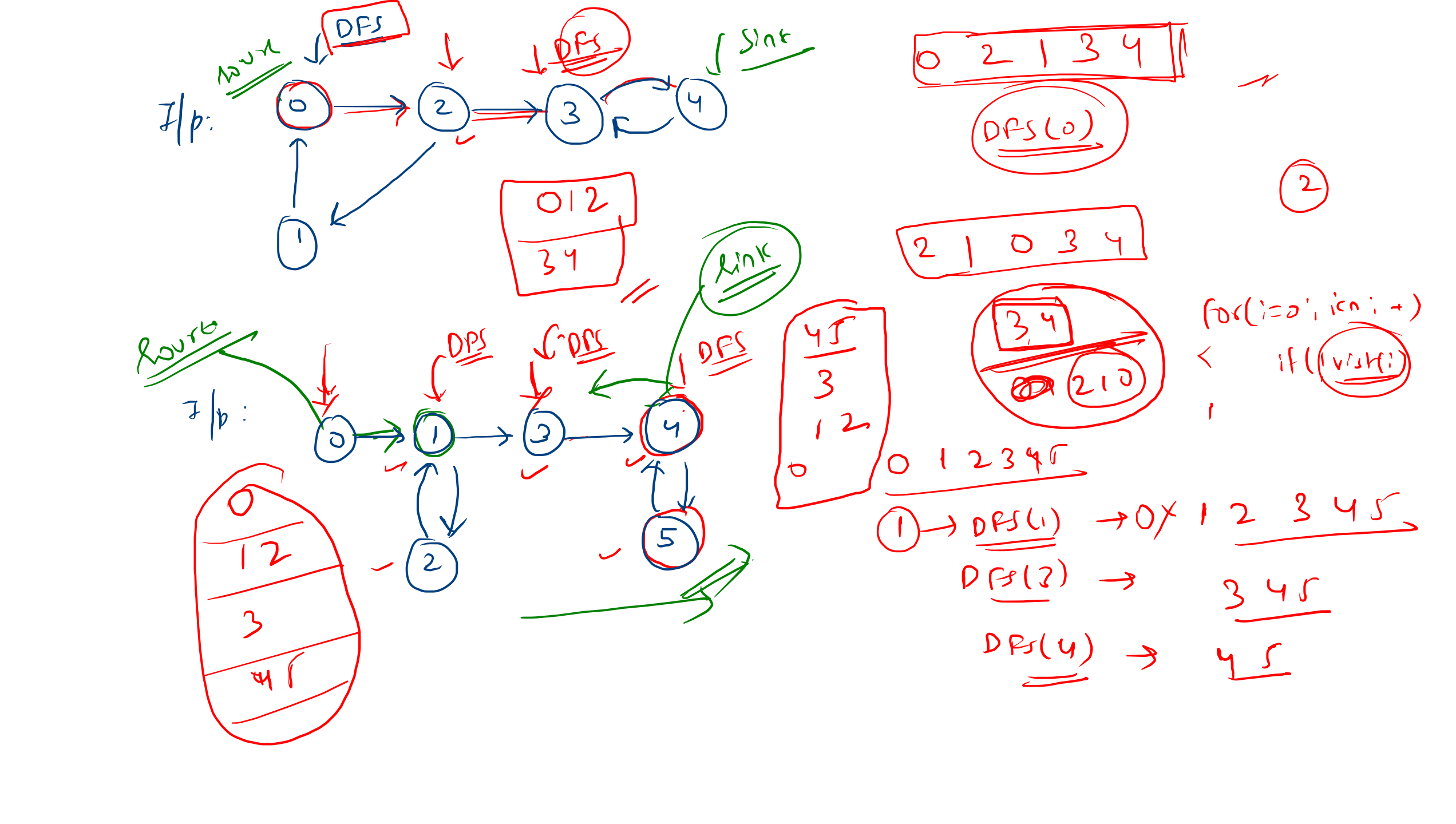
:> if Graph is **undirected**

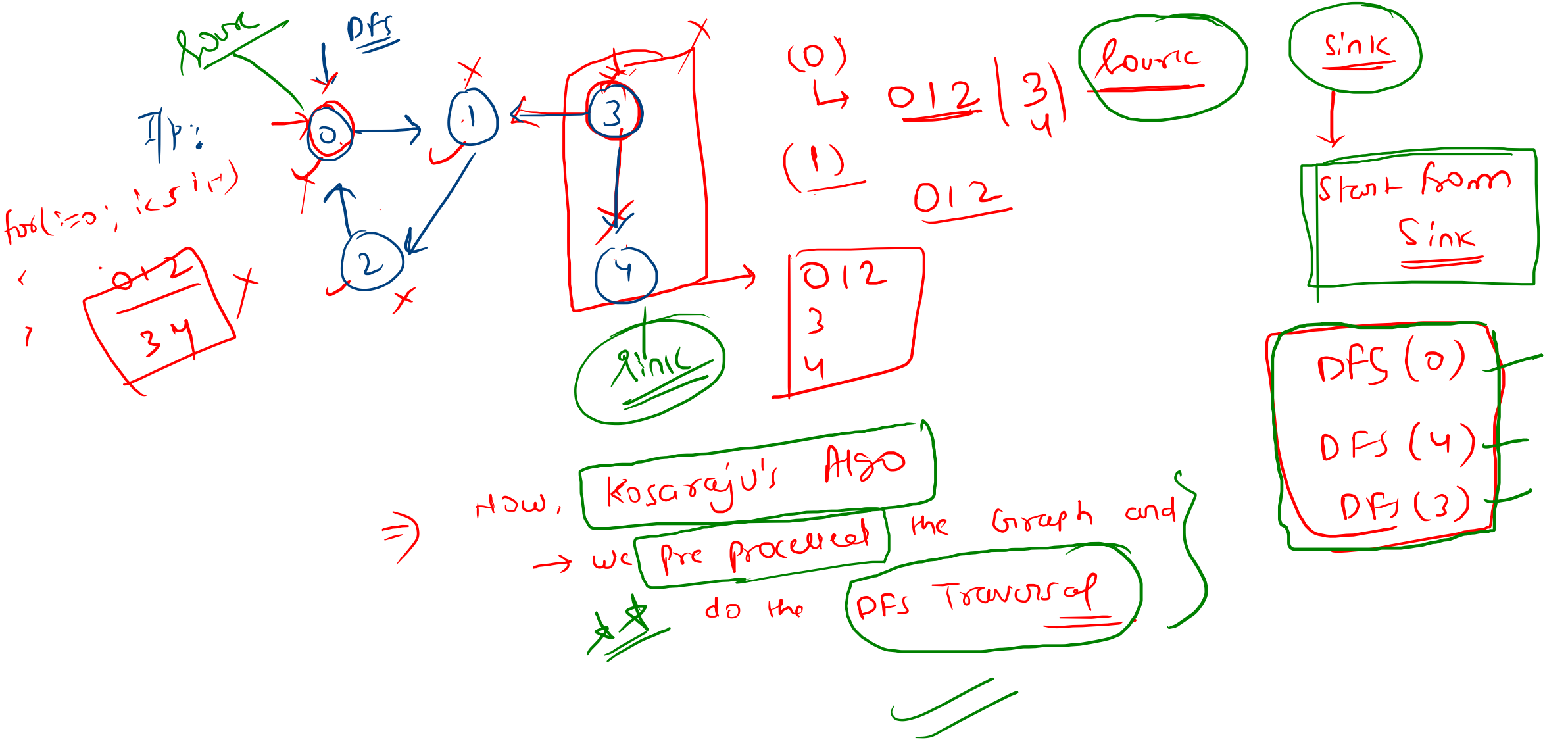
↳ DFS / BFS → we start the traversal, then whatever the **vertex is connected** is become a **part of connected component**.

→ BFS / DFS





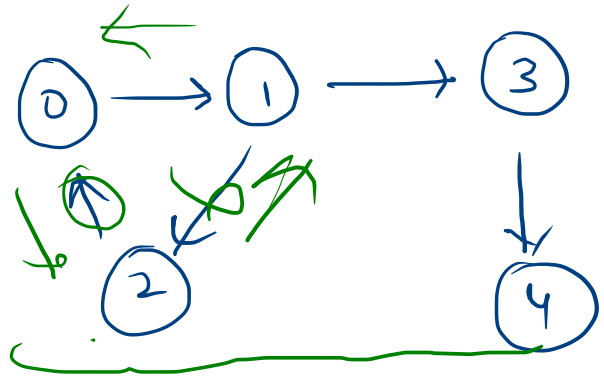

DFS
DFS
DFS
Sink
I/p:
0
2
3
4
1
0 2 1 3 4
DFS(0)
0 1 2
3 4
2
2 1 0 3 4
3 4
2 1 0
for(i=0; i<n; i++)
if(!visited[i])
Source
I/p:
DFS
DFS
DFS
Sink
0
1
3
4
2
5
4 5
3
1 2
0
0 1 2 3 4 5
1 → DFS(1) → 0 X 1 2 3 4 5
DFS(3) → 3 4 5
DFS(4) → 4 5
0
1 2
3
4 5


Source
DFS
I/P:
for(i=0; i<5; i++)
{
0+2
3 4
}
0
1
2
3
4
Sink
0 1 2
3
4
(0)
↳ 0 1 2 | 3 | 4
(1)
0 1 2
Source
Sink
Start from Sink
DFS (0)
DFS (4)
DFS (3)
How,
Kosaraju's Algo
⇒
→ we Pre processed the Graph and
do the DFS Traversal

Algo

1. Order the vertices in decreasing order of finish time in DFS → order
2. Reverse all edges
3. DO DFS of Reverse Graph in the order obtained in step 1.

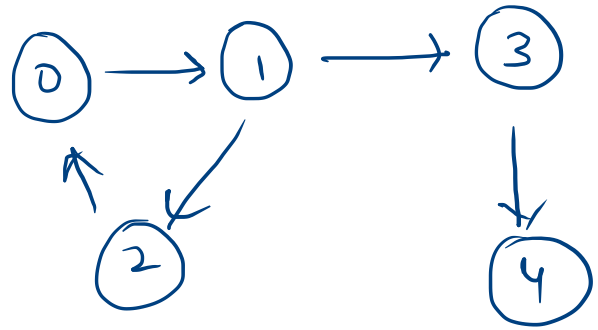
(For every vertex, print all reachable vertices as one strongly Connected Component)



Algo

1. Order the vertices in decreasing order of **Finish time** in DFS
2. Reverse all edges
3. DO DFs of Reverse Graph in the order obtained in step 1.

(For every vertex, print all reachable vertices as one strongly Connected Component



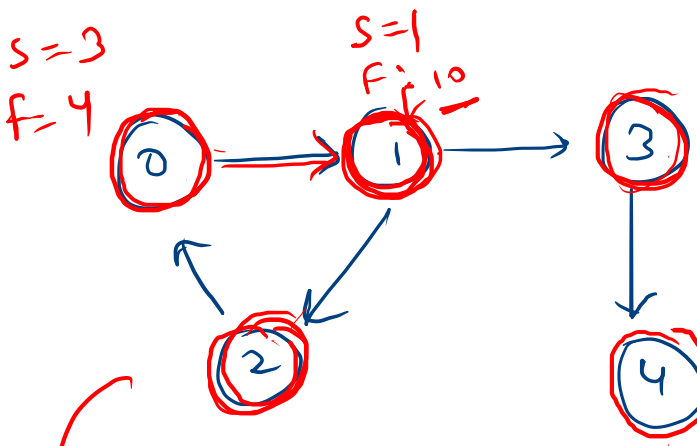
DfsRec(V)

Step 1 :

You make a recursive call --> start time

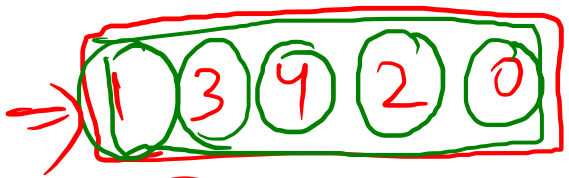
once all adjacent are processed we make a Finished call → Finish time

Step 1 : understand finish time



→ decreas finish

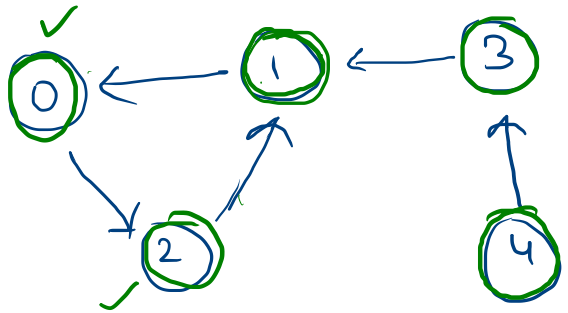
order



Orders

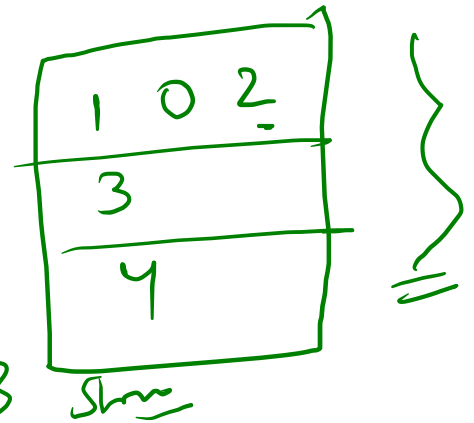
Reverse edge

step 2 :



step3 : DFS

DFS(1)
DFS(3)
DFS(4)



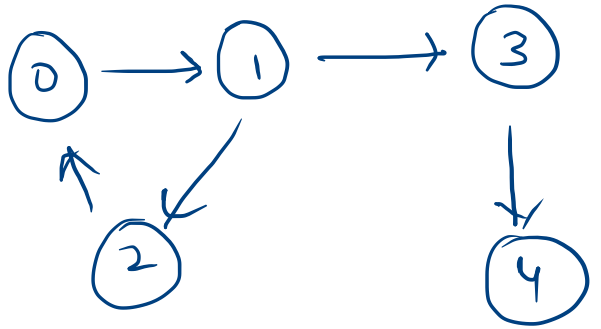
Implementation of code

Algo

1. Order the vertices in decreasing order of finish time in DFS
2. Reverse all edges (reverse edge)
3. DO DFS of Reverse Graph in the order obtained in step 1. (normal DFS call)

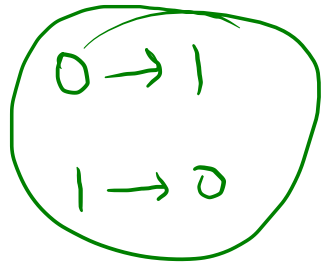
(For every vertex, print all reachable vertices as one strongly Connected Component)

DFS



1 3 4 20

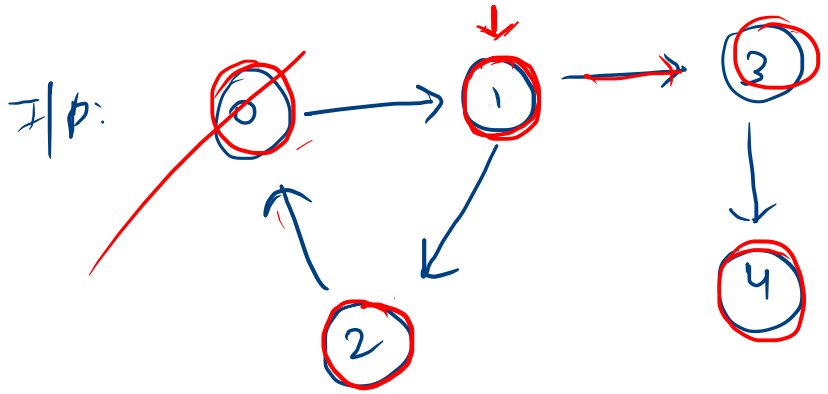
make transpos

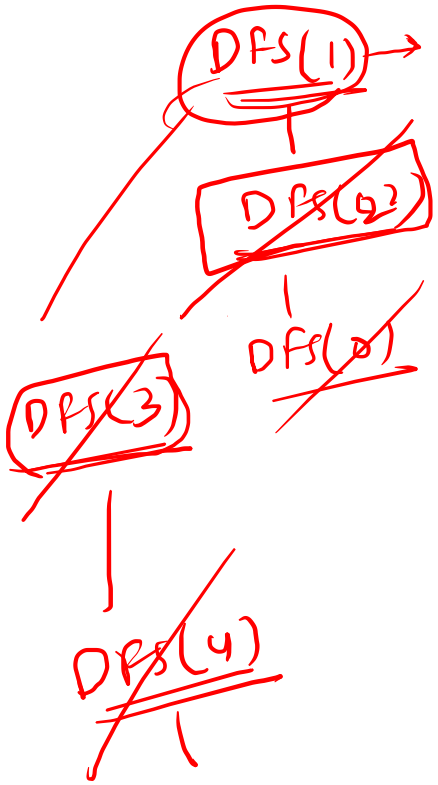


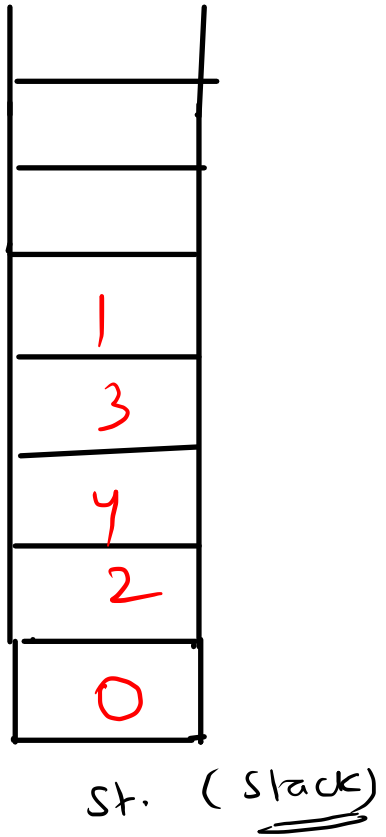
Implementation of step 1:

1. Create an empty stack, st
2. For every vertex u, do the following
   if(u is not visited)
      DFSRec(u, st)
3. while(st is not empty)
   pop an item and add to result

```
DFSRec( u, st)
start a) Mark u is visited
     b) for every adjacent v
          if( v is not visited )
               DFSRec( v, st)
finish c) st.push( u )
```



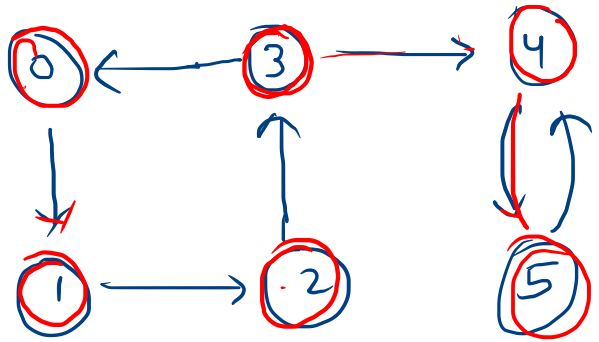




St. (Stack)

Orders: 1 3 4 2 0

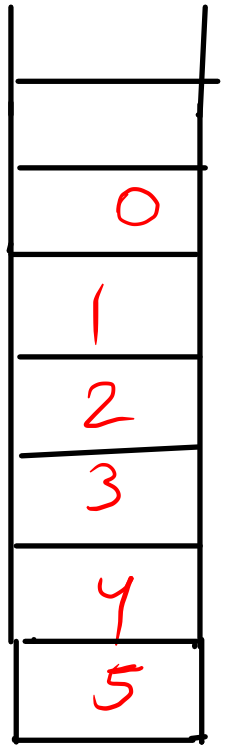
I/p:



order

0 1 2 3
4 5

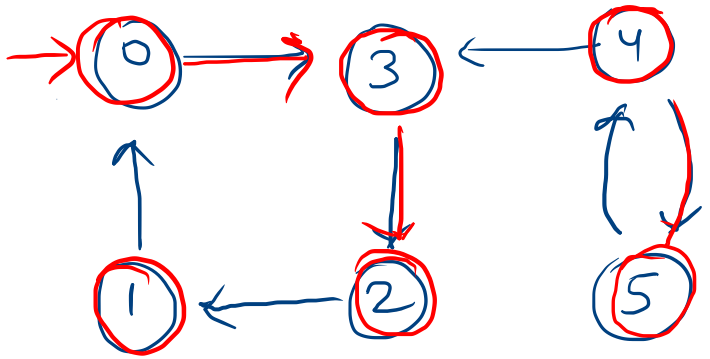
DFS(0)
DFS(1)
DFS(2)
DFS(3)
DFS(4)
DFS(5)



St. (stack)

order: 0 1 2 3 4 5

I/p:



DFS from 0:

DFS from 4:

0 3 2 1
4 5

order:

0 1 2 3 4 5

X X X ↑ X