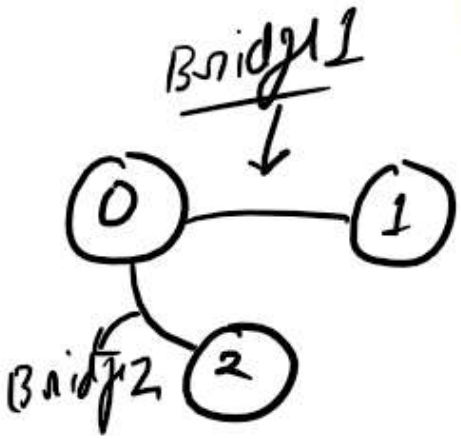
TARJANS ALGORITHM

Find Bridge in GRAPH

Leetcode - 1192



LinkedIn@manojofficialmj

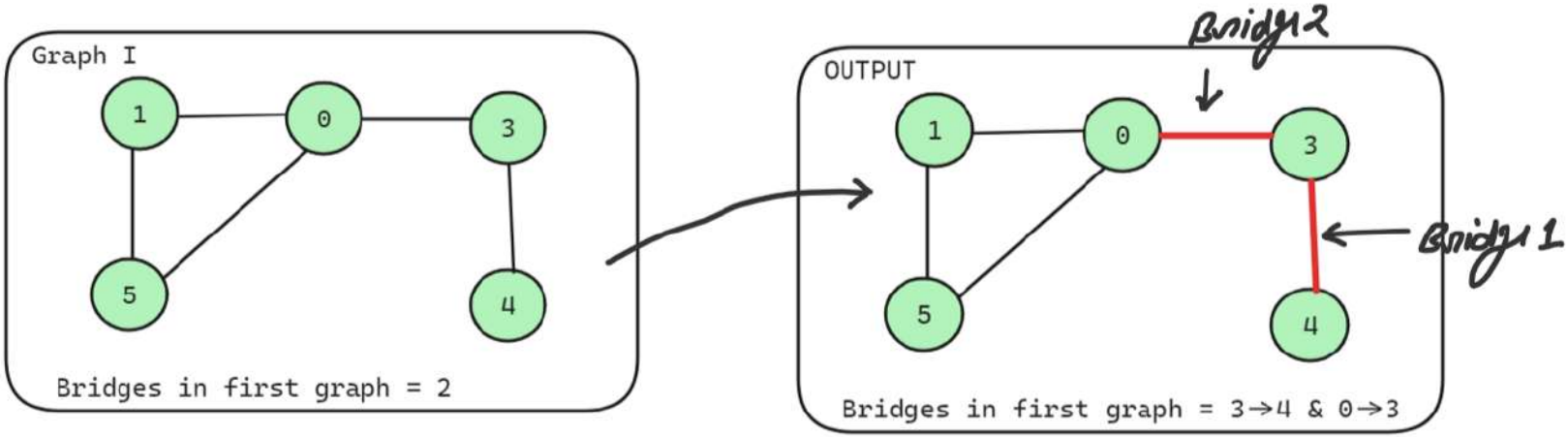
📂 5. TARJANS Algorithm

🚀 Find Bridges in Graph (Leetcode-1192)

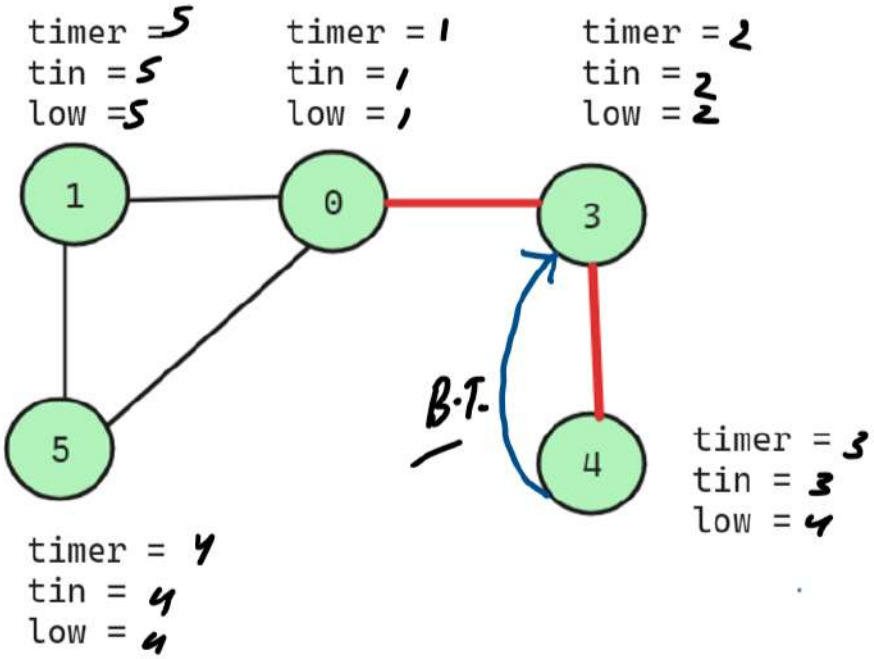
Why use Tarjans Algorithm:
Used for an undirected Graph, The task is to find the Bridges in Graph.

What is Bridge in Graph:
In an undirected connected graph, a bridge is an edge removal that increases the number of disconnected components.

😊 Bridge ek aise edge hai jisko remove karane par disconnected component increase ho jate hai.



Logic



What is timer and tin & Low ?

- **timer:** this is a initial time for a node jnha par hum khade hue hai
- **tin:** this is a current time jab hum uss node par pahunch chuke hai
- **low:** this is a minimum time jab hum uss node par pahunch chuke hai

NODE → Times → Current Time (tin)
NODE → Times → Lowest Time (low)

NODE → (A) Low updation low = min(low NBR, low SRC)
NODE → (B) Check Bridge
(B) → low of NBR > tin of SRC → Found Bridge

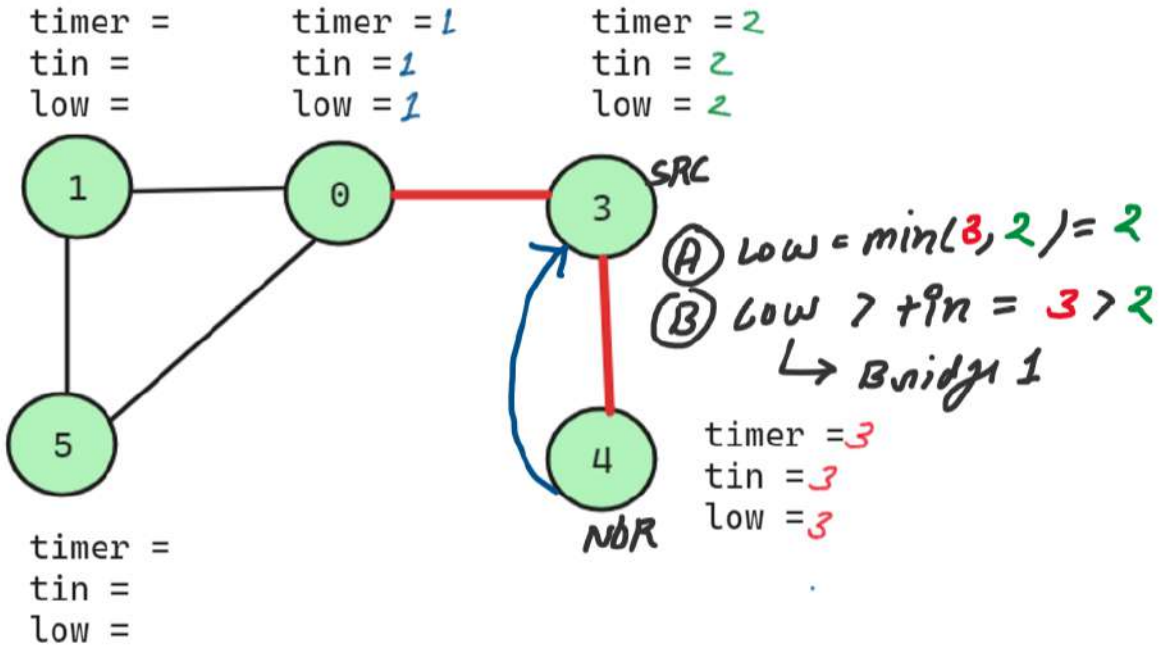
YEH SAB KAB KARNA HAI
→ DURING BACK TRACKING

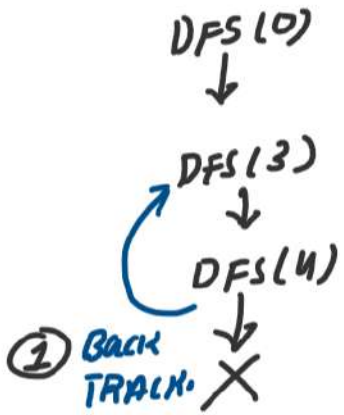
```cpp
// 5. Tarjans Algorithm (Bridges in Graph) (Leetcode-1192)

class Solution {
public:

    void dfs(int src, int parent, unordered_map<int,list<int> >& adjList, int &timer,
    vector<vector<int>>& ans,vector<int>& tin,vector<int>& low,unordered_map<int,bool>& visited ) {
        ...
    }

    vector<vector<int>> criticalConnections(int n, vector<vector<int>>& connections)
    {
        // Create the adjList
        unordered_map<int,list<int> > adjList;
        for(auto vec: connections)
        {
            int u = vec[0];
            int v = vec[1];
            // Undirected graph hai to u->v & v->u
            adjList[u].push_back(v);
            adjList[v].push_back(u);
        }

        // Main logic to get the bridge
        int timer = 1;
        vector<vector<int>> ans;
        vector<int> low(n,0); // (Node, Low time)
        vector<int> tin(n,0); // (Node, Current time)
        int src = 0;
        int parent = -1;
        unordered_map<int,bool> visited;
        dfs(src, parent, adjList,timer,ans, tin, low,visited);
        return ans;
    }
};
```

```cpp
void dfs(int src, int parent, unordered_map<int,list<int> >& adjList, int &timer,
vector<vector<int>>& ans,vector<int>& tin,vector<int>& low,unordered_map<int,bool>& visited ) {
    // Initial state
    visited[src] = true;
    tin[src] = timer;
    low[src] = timer;
    timer++;

    // Go for nbr of each src
    for(auto nbr: adjList[src]) {
        if(nbr == parent) {
            // Ignore this
            continue;
        }
        if(!visited[nbr]) {
            // Step 1: Pahle DFS call karlo jitne bhi src ke nbr hai un sabhi par
            dfs(nbr, src, adjList, timer, ans, tin, low, visited);
            // Step 2: Jab koi bhi nbr na ho to back tracking karlo
            // A: low update
            low[src] = min(low[src], low[nbr]);
            // B: bridge testing
            if(low[nbr] > tin[src]) {
                vector<int> temp;
                temp.push_back(src);
                temp.push_back(nbr);
                ans.push_back(temp);
            }
        }
        else {
            // Step 2: Ek aur raasta milgya hai iska mtlb mujhe back tracking karni hai
            // A: low update
            low[src] = min(low[src], low[nbr]);
            // B: no need of bridge testing kyunki me already yeh kam kar chuka hoonga
            // jav pahli bar nbr visite huaa hoga
        }
    }
}
```

DRY RUN B-C-1

DFS call start from (0) to (N-1)

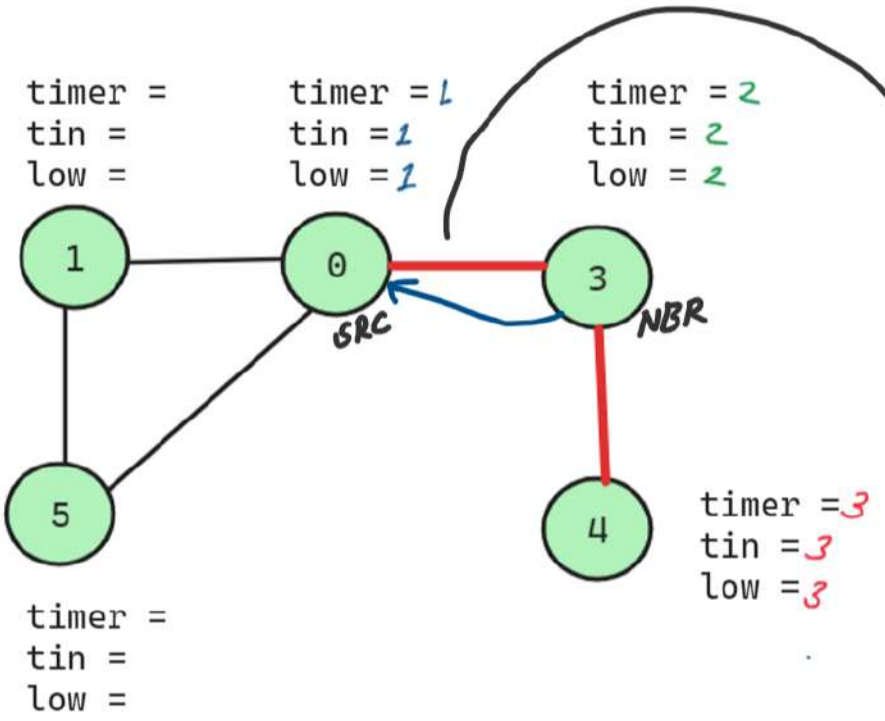




Bridge 1 => 3 → 4

DRY RUN B-C-2

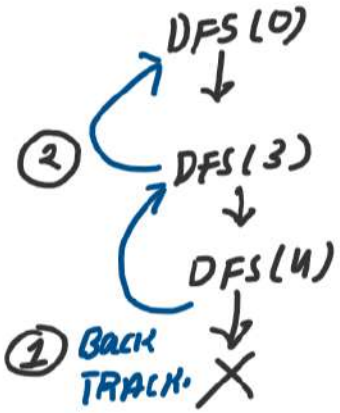
DFS call start from (0) to (N-1)



(A) low = min(2, 1) = 1

(B) low > tin = 2 > 1

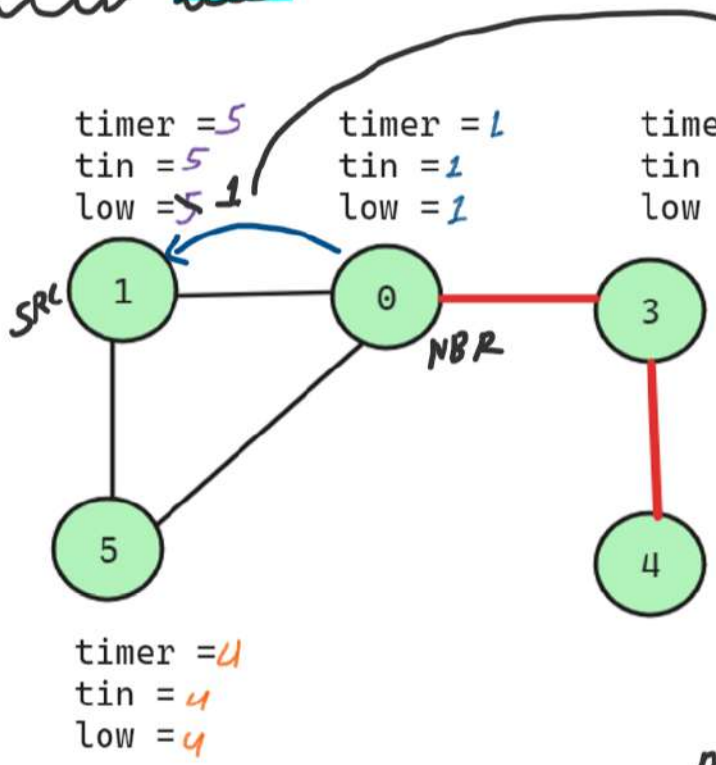
↳ Bridge = 2



Bridge 1 =>	3 → 4
Bridge 2 =>	0 → 3

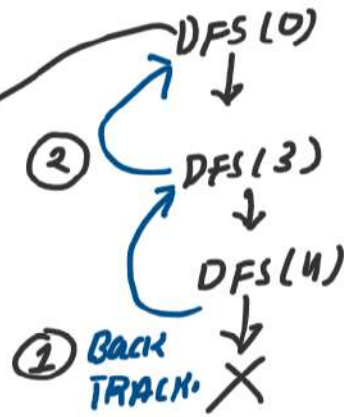
DRY RUN B-C-3

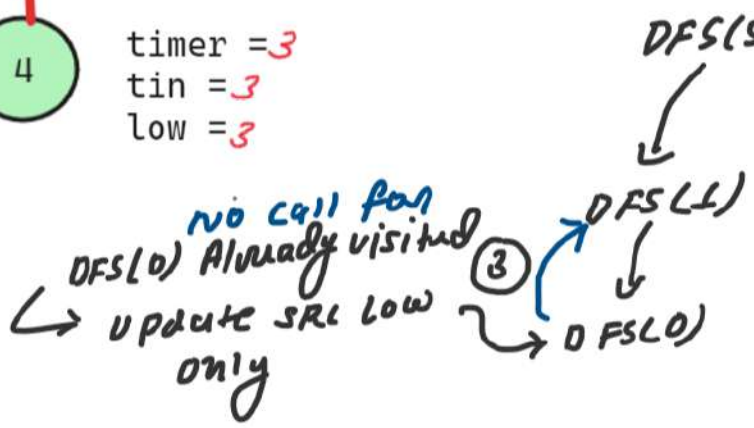
DFS call start from (0) to (N-1)



(A) low = min(1, 5) = 1

(B) low > tin = 1 > 5 X

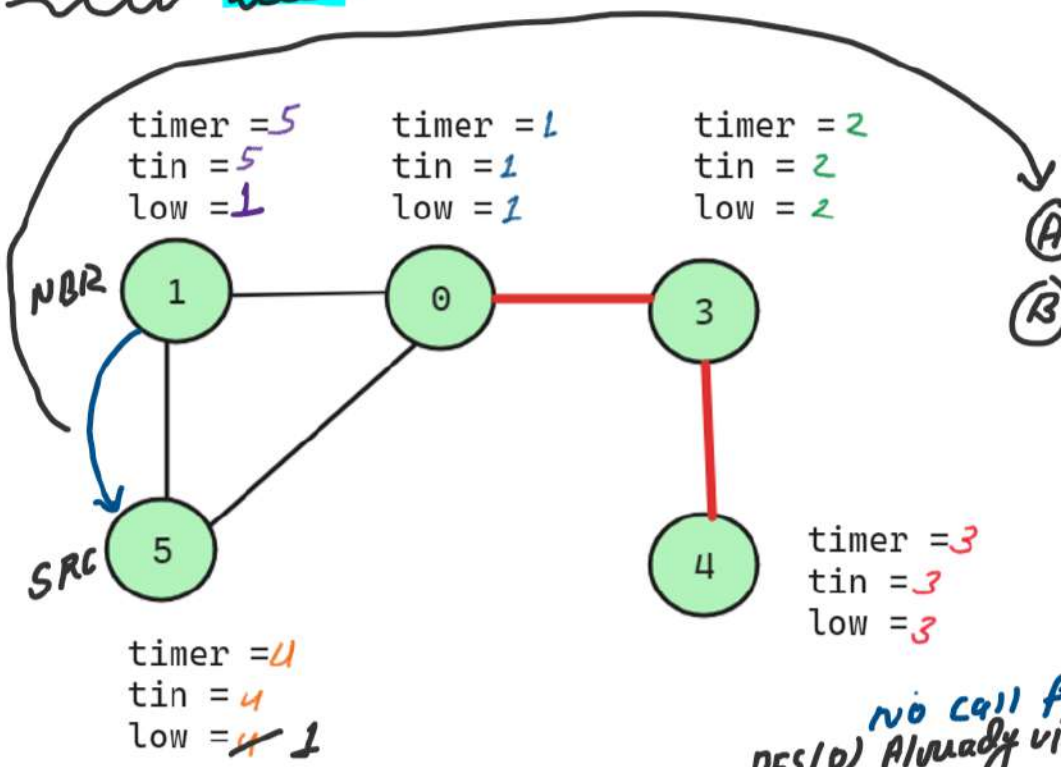




Bridge1 =>	3 → 4
Bridge2 =>	0 → 3

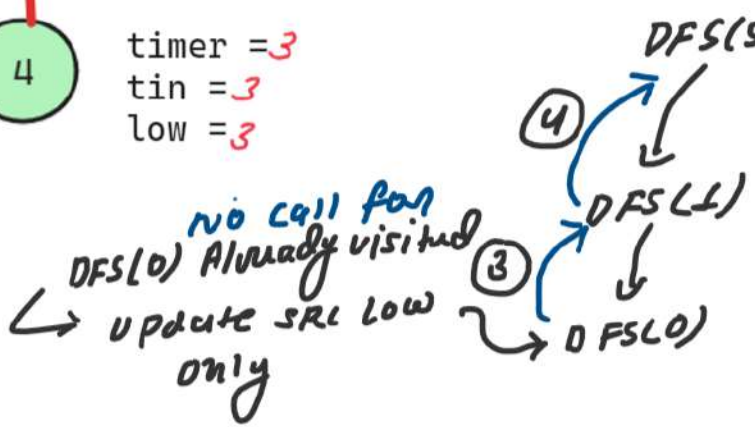
DRY RUN B-C-4

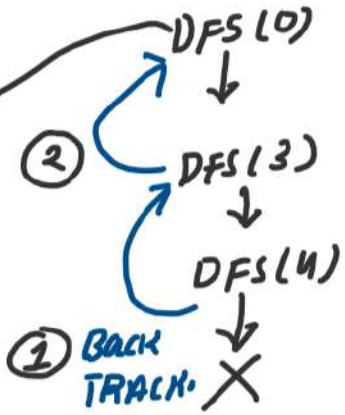
DFS call start from (0) to (N-1)



(A) low = min(1, 4) = 1

(B) low > tin = 1 > 4 X

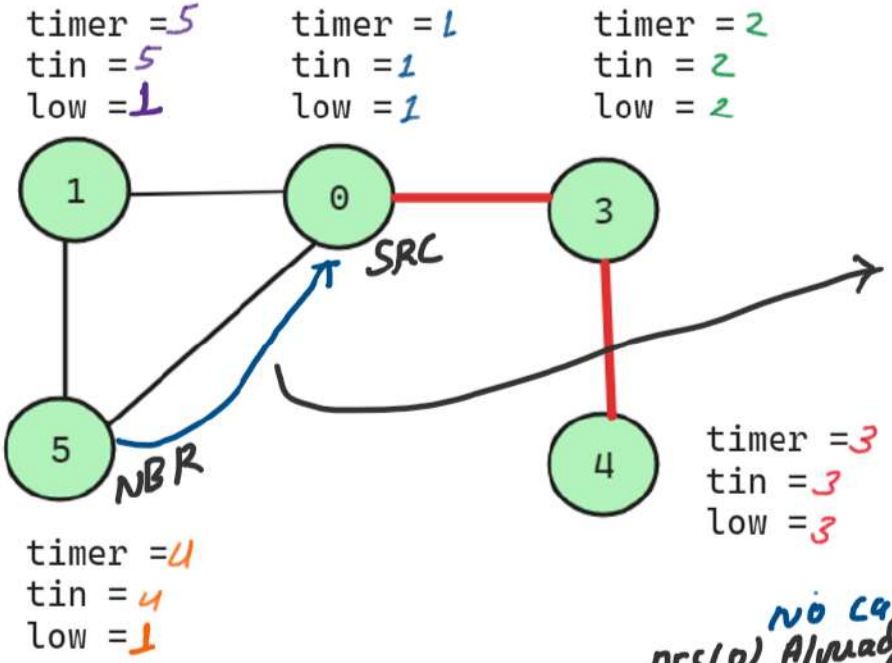




Bridge 1 =>	3 → 4
Bridge 2 =>	0 → 3

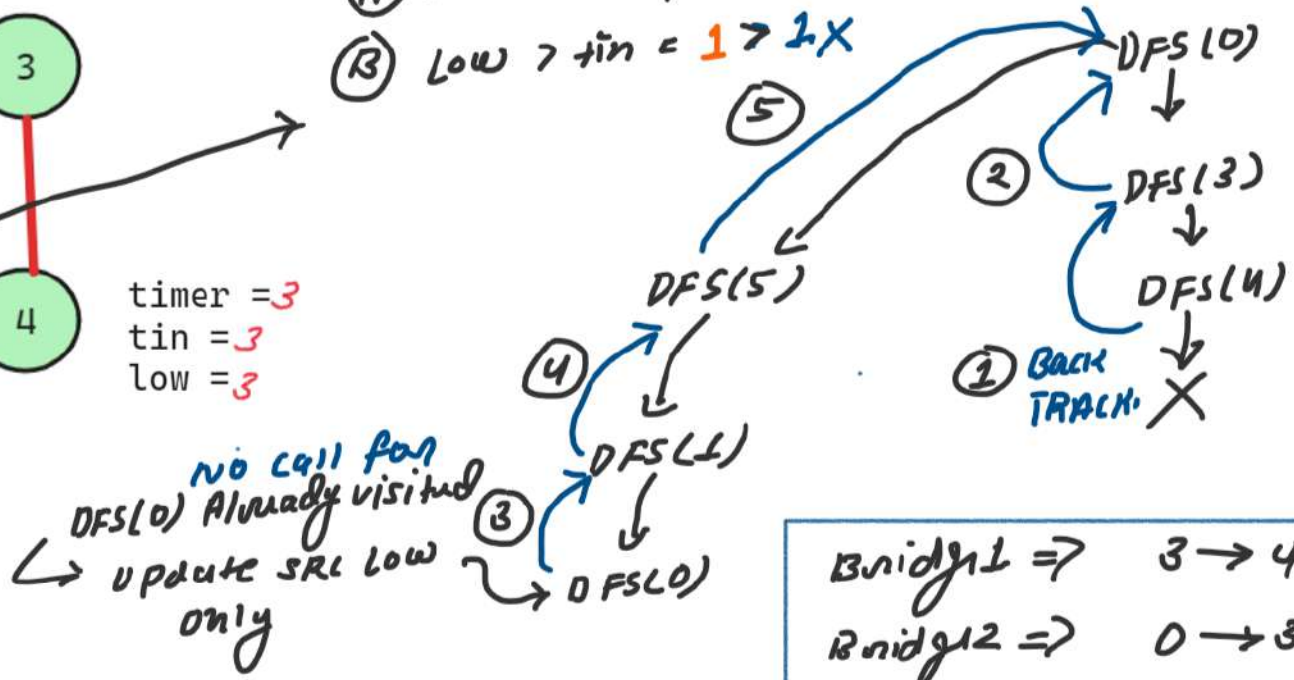
DRY RUN B-C-S

DFS call start from (0) to (N-1)



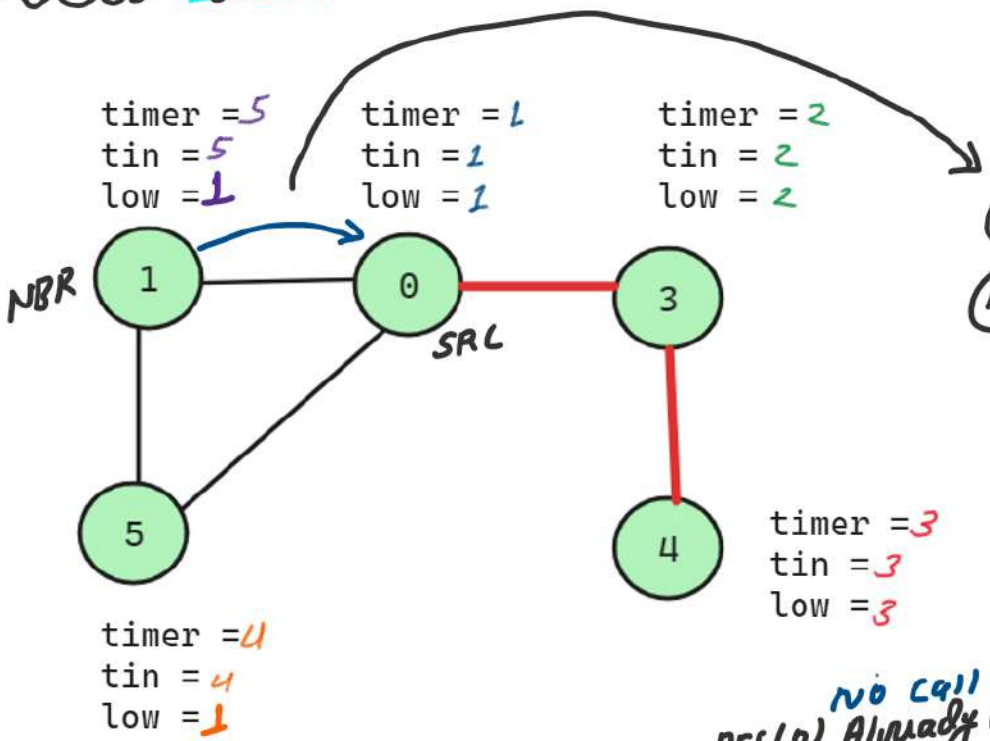
(A) low = min(1, 1) = 1

(B) low > tin = 1 > 1 X



Bridge1 =>	3 → 4
Bridge2 =>	0 → 3

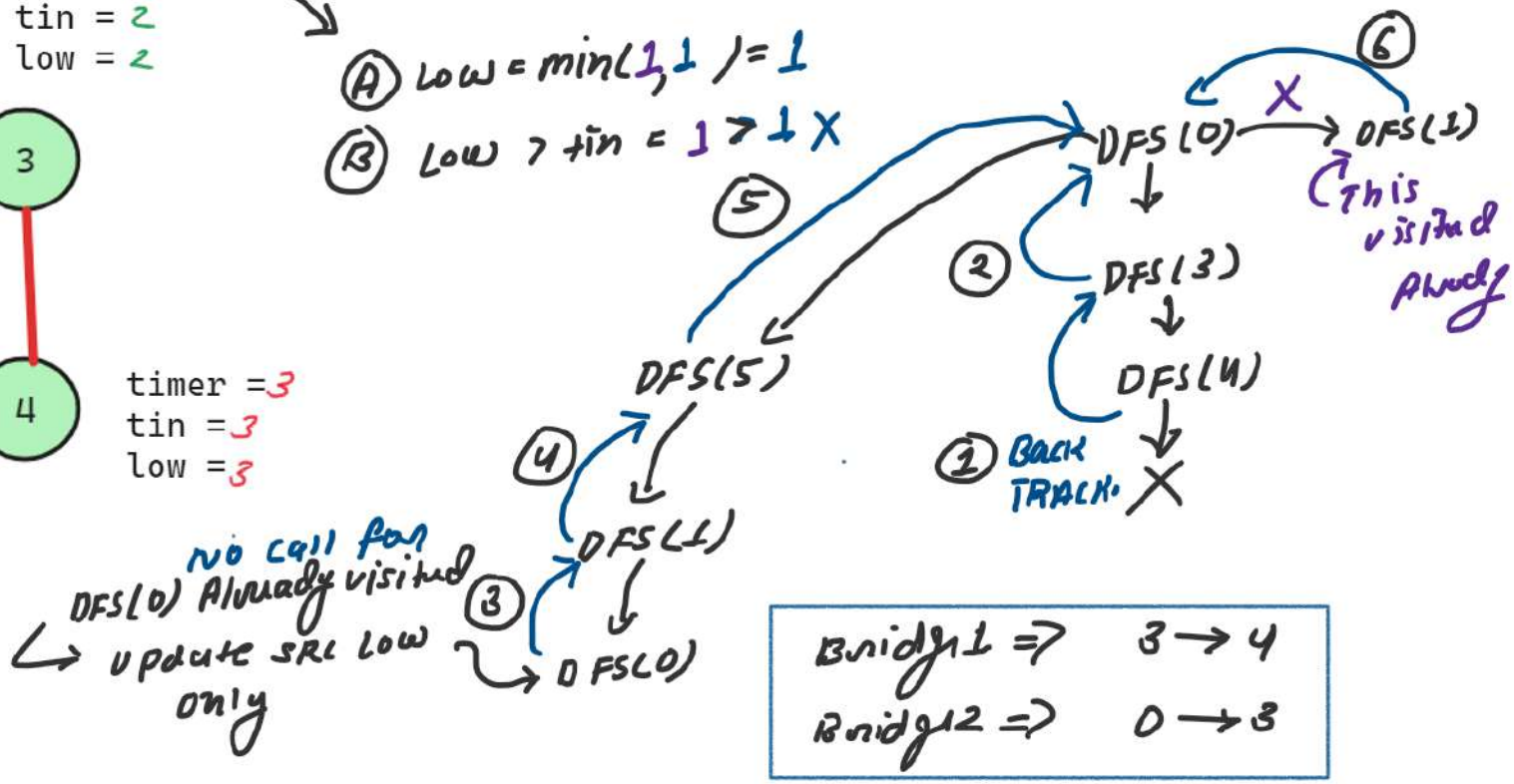
DRY RUN B-C-G



DFS call start from (0) to (N-1)

(A) low = min(1, 1) = 1

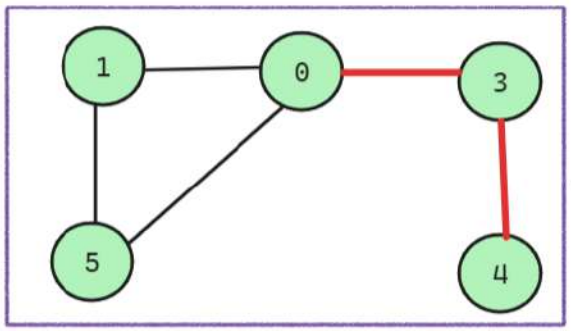
(B) low > tin = 1 > 1 X



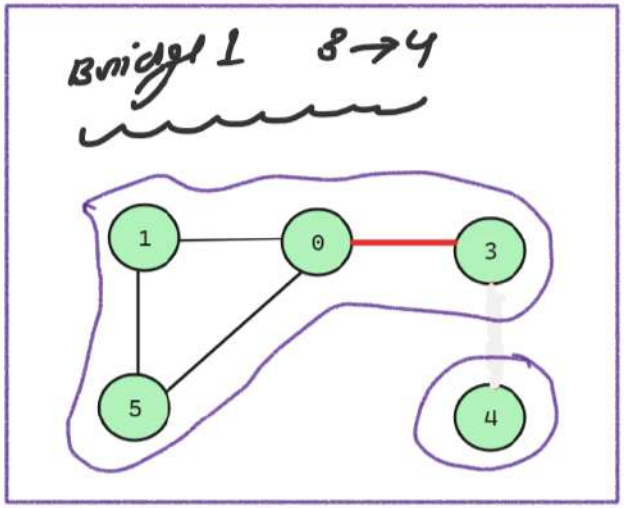
Bridge 1 => 3 → 4

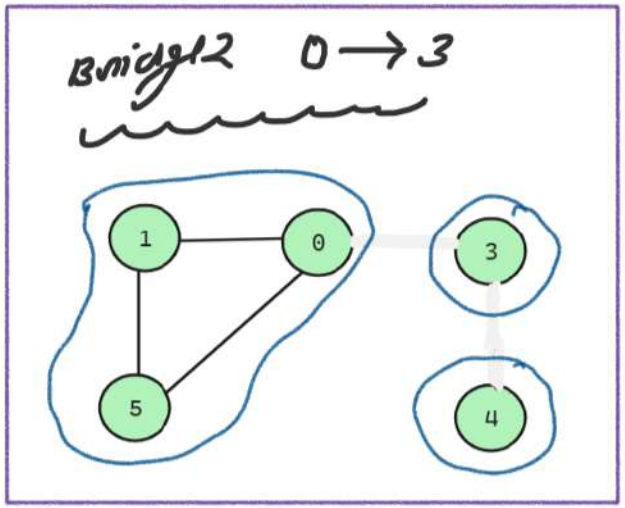
Bridge 2 => 0 → 3

Final Output



GRAPH





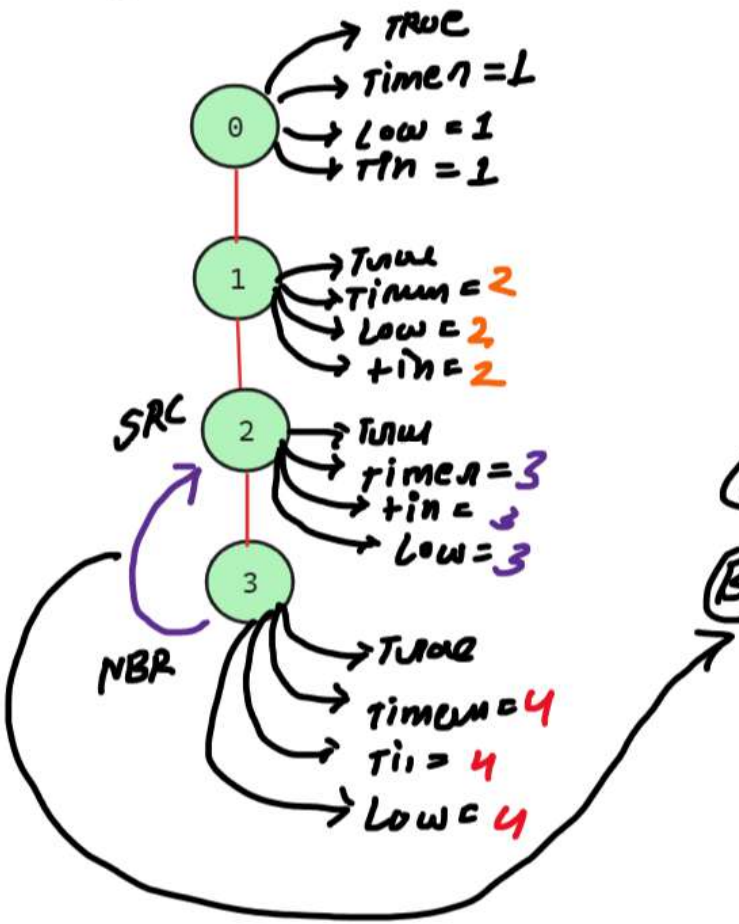
T.C. = ?

S.C. = ?

When we remove the Edge

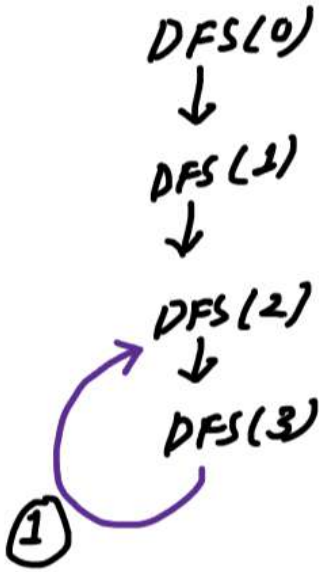
↳ Increase the No. of disconnected components in Graph due to Bridge

Example-2 B-C-1



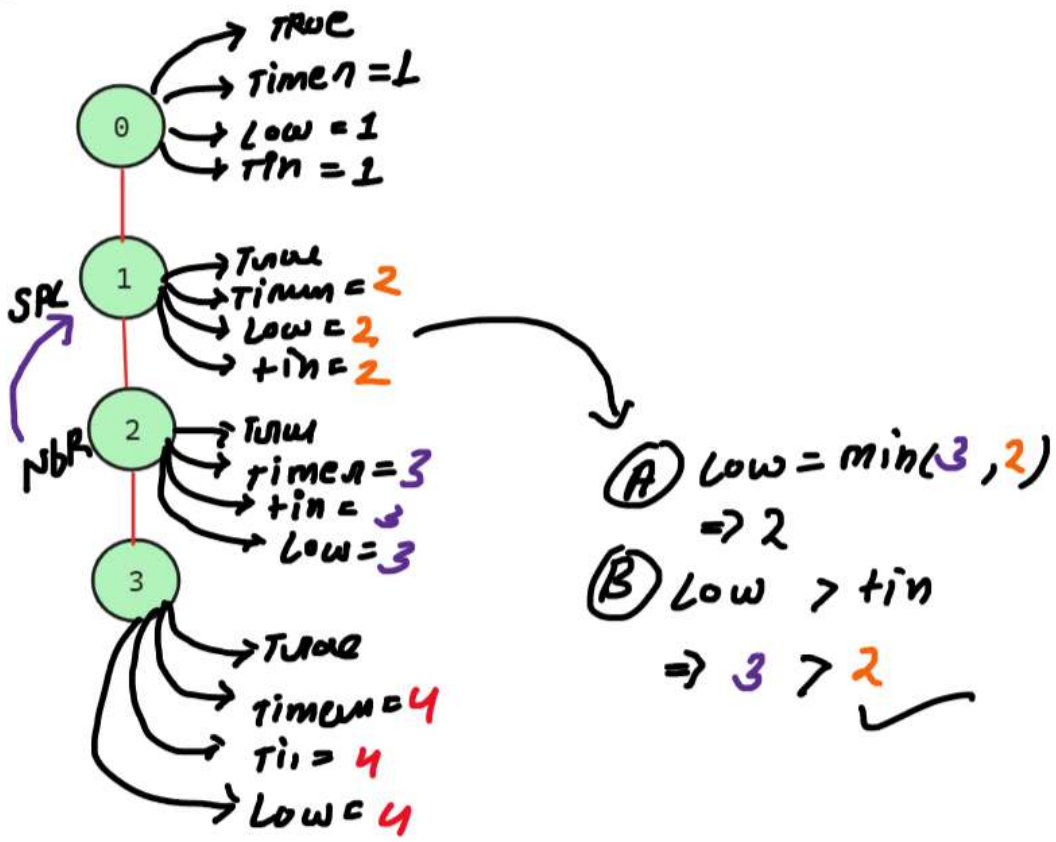
(A) low = min(4, 3)
=> 3

(B) low > fin
=> 4 > 3



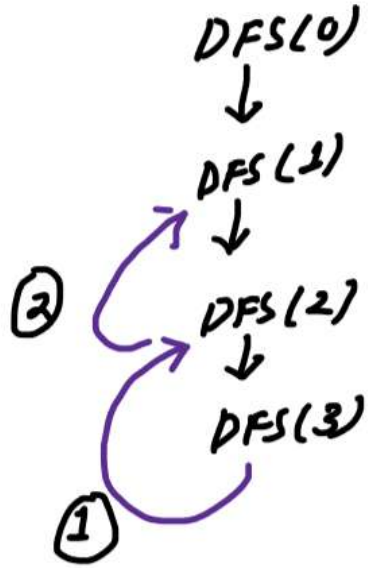
Bridges 1 2 → 3

Example-2 B.C.2



(A) low = min(3, 2)
=> 2

(B) Low > tin
=> 3 > 2



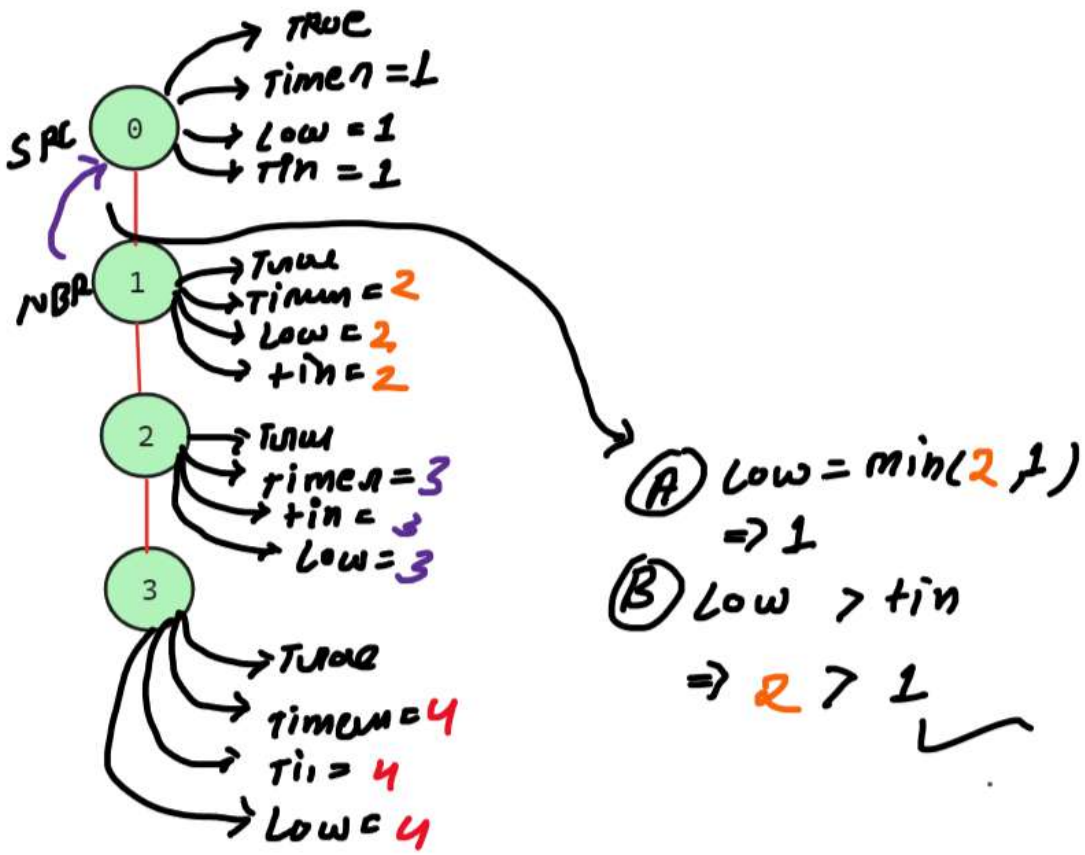
Bridges

1 2 → 3

2 1 → 2

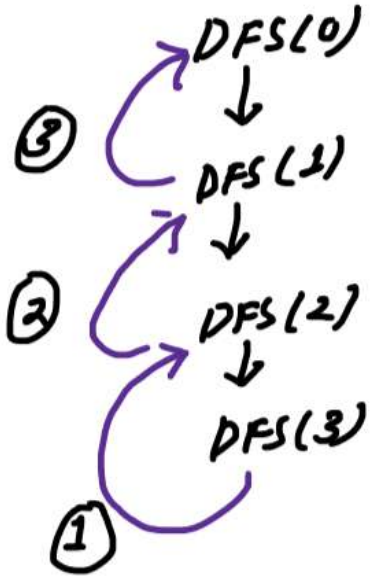
Example-2

B.C.3



(A) Low = min(2, 1)
=> 1

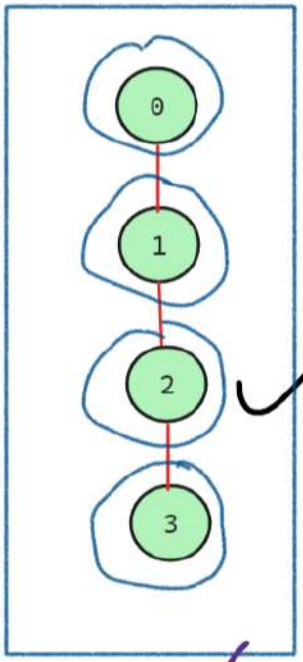
(B) Low > tin
=> 2 > 1



Bridges

1	2 → 3
2	1 → 2
3	0 → 1

Example-2



Output

Bridges

1 → 2 → 3

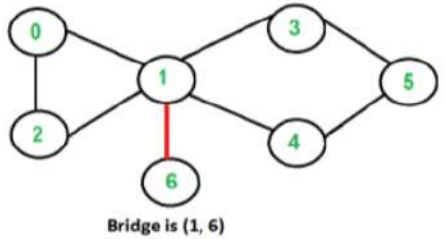
2 → 1 → 2

3 → 0 → 1

4 components

3 Bridges

Example-3




```
Bridges in third graph
1 6
```

TRY TO DRY ON THIS Ex-3 FOR Bottom understanding