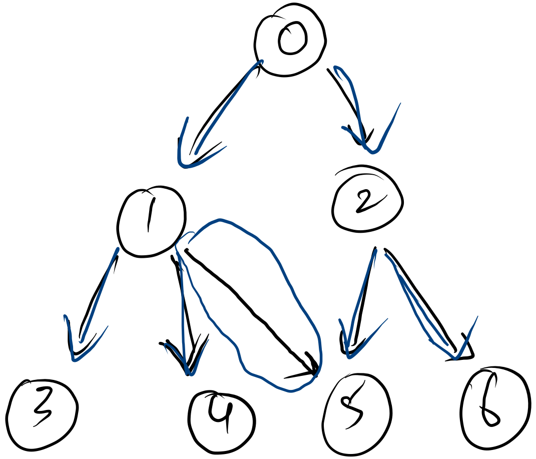
Reduendent Connection

Extra Edge

Tree → Graph



✓ 0, 1

✓ 0, 2

✓ 1, 3

✓ 1, 4

✓ 2, 5

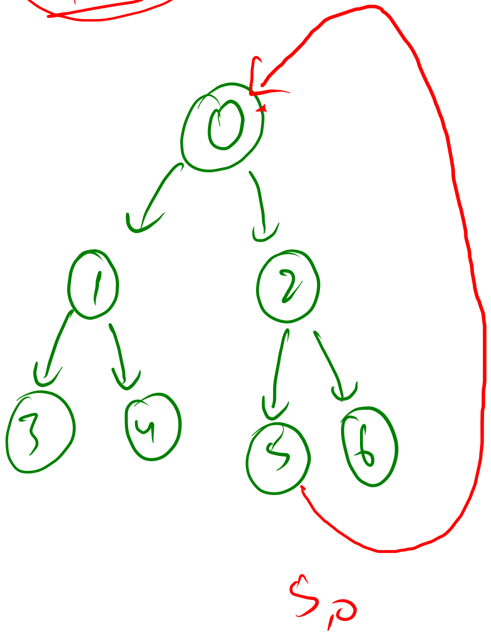
✓ 2, 6

1, 5 → Ans

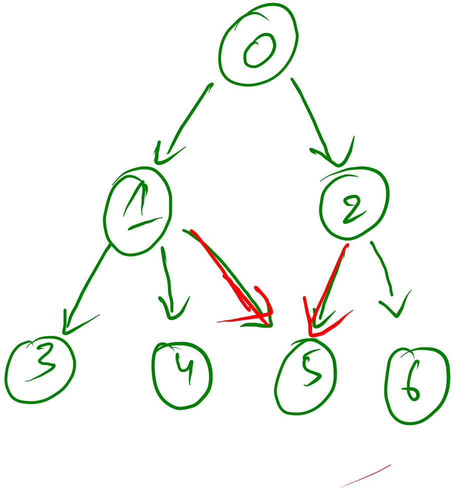
P. Stmt => Find Extra Edge

3 Kind

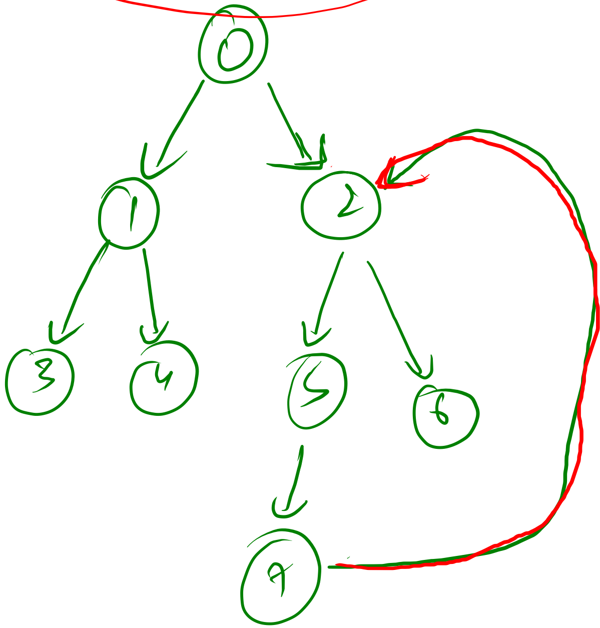
Cycle



2-indegree vtx



cycle + 2-indegree vtx



Cycle

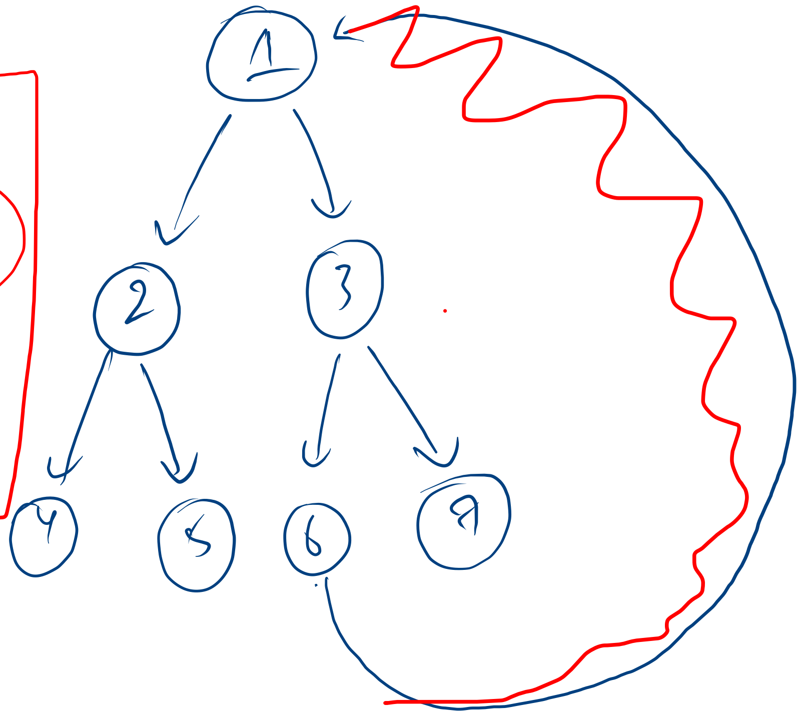
indegree == 1

+

closed structure — DSU

+

1 Extra edge



1	2	3	4	5	6	7
~~∅~~	~~∅~~	~~∅~~	~~∅~~	~~∅~~	~~∅~~	~~∅~~
1	1	1	1	1	1	1

(u,v) : u → v

2, 5 ✓

2, 4 ✓

6, 1 ✓

1, 3 ✓

3, 6 ✓ → RE

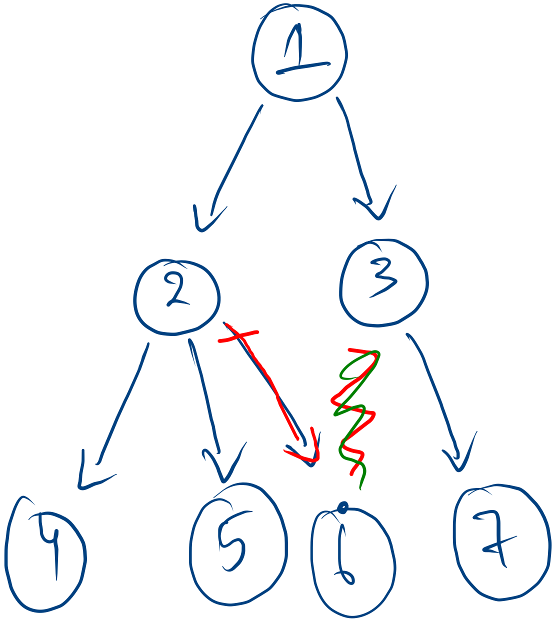
closed structure

1, 2

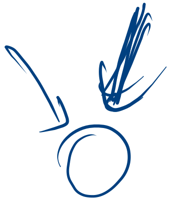
3, 7

2- In degree

1	2	3	4	5	6	7
0	~~0~~	~~0~~	~~0~~	0	~~0~~	0
	1	1	1		1	

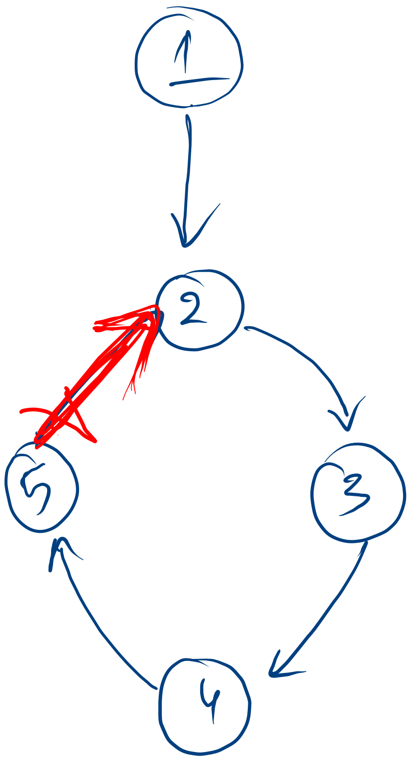


- 1, 2 ✓
- 1, 3 ✓
- 2, 4 ✓
- 2, 6 ✓
- 3, 6 * → RE
- 2, 5
- 3, 7



2-indegree + Cycle

1	2	3	4	5
0	~~0~~	~~0~~	~~0~~	~~0~~
	1	1	1	1



2, 3 ✓

3, 4 ✓

4, 5 ✓

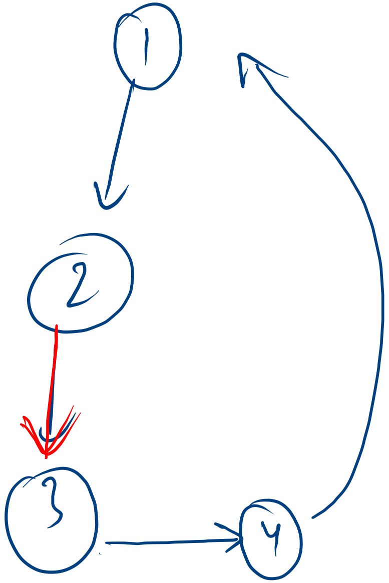
5, 2 ✓ → RE

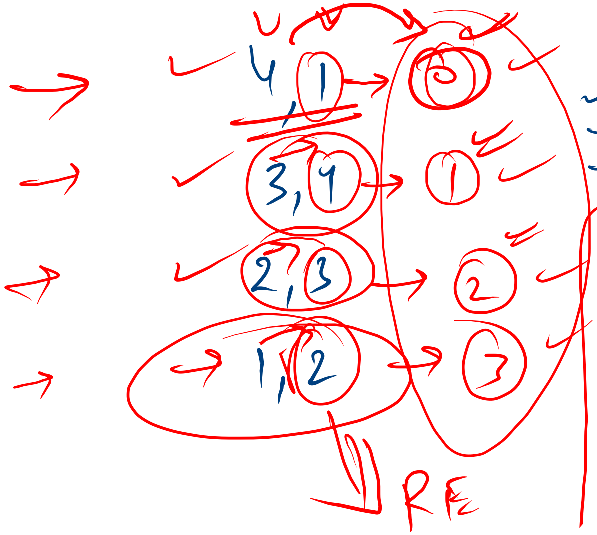
~~1, 2~~ ✓

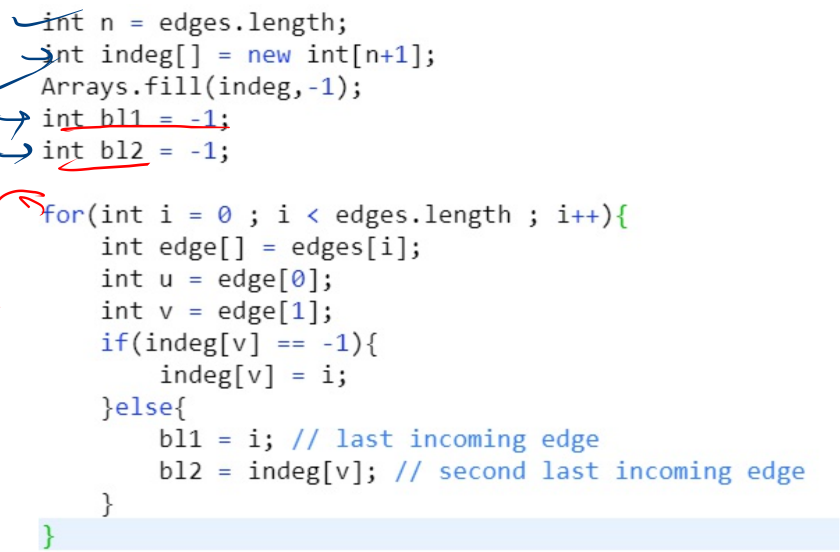
closed structure → D.S.U.

bl1 → -1

bl2 → -1

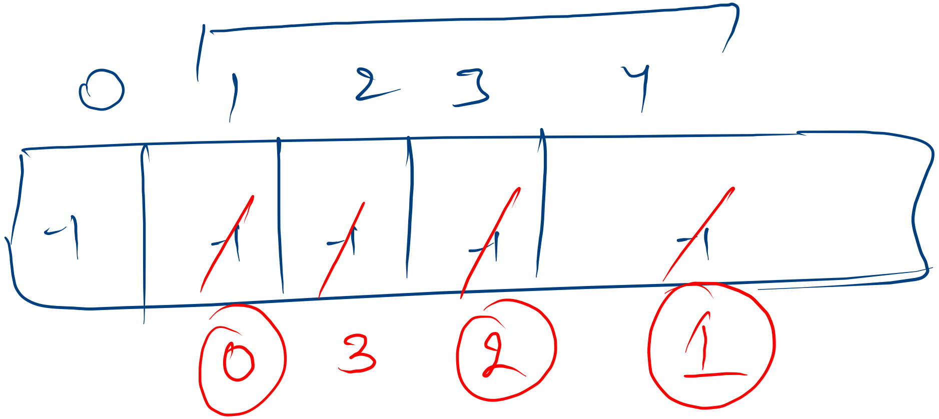


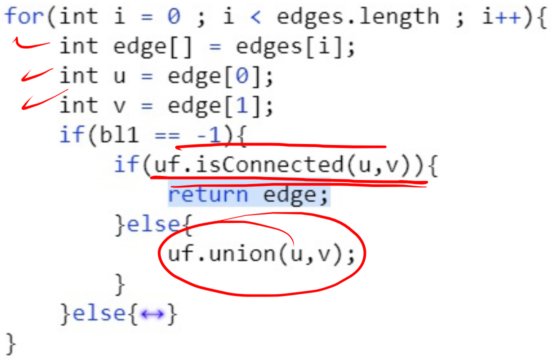




```
int n = edges.length;
int indeg[] = new int[n+1];
Arrays.fill(indeg,-1);
int bl1 = -1;
int bl2 = -1;

for(int i = 0 ; i < edges.length ; i++){
    int edge[] = edges[i];
    int u = edge[0];
    int v = edge[1];
    if(indeg[v] == -1){
        indeg[v] = i;
    }else{
        bl1 = i; // last incoming edge
        bl2 = indeg[v]; // second last incoming edge
    }
}
```



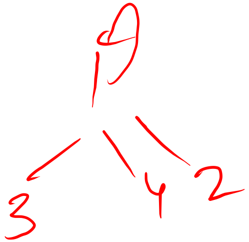


```
for(int i = 0 ; i < edges.length ; i++){
    int edge[] = edges[i];
    int u = edge[0];
    int v = edge[1];
    if(bl1 == -1){
        if(uf.isConnected(u,v)){
            return edge;
        }else{
            uf.union(u,v);
        }
    }else{↔}
}
```

bl1 .

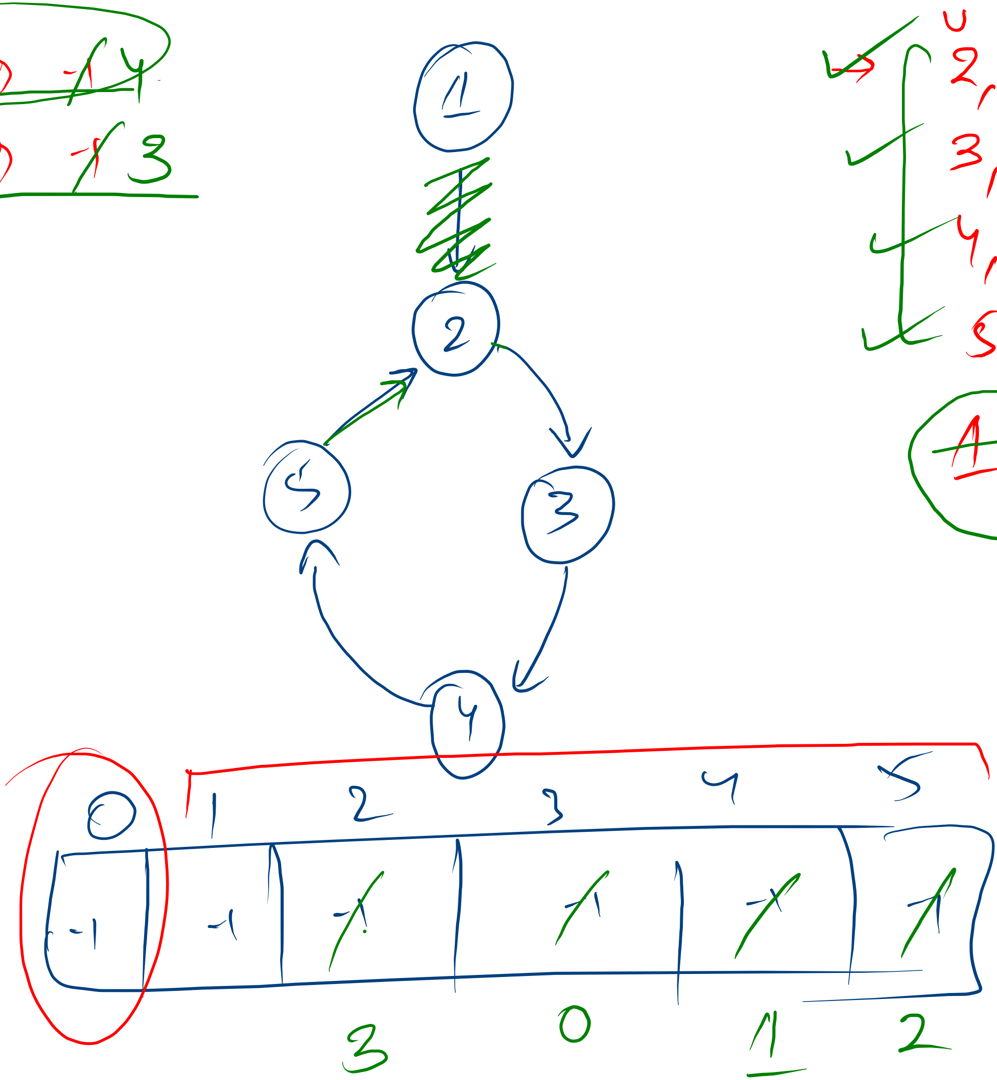
bl2

indegree=2

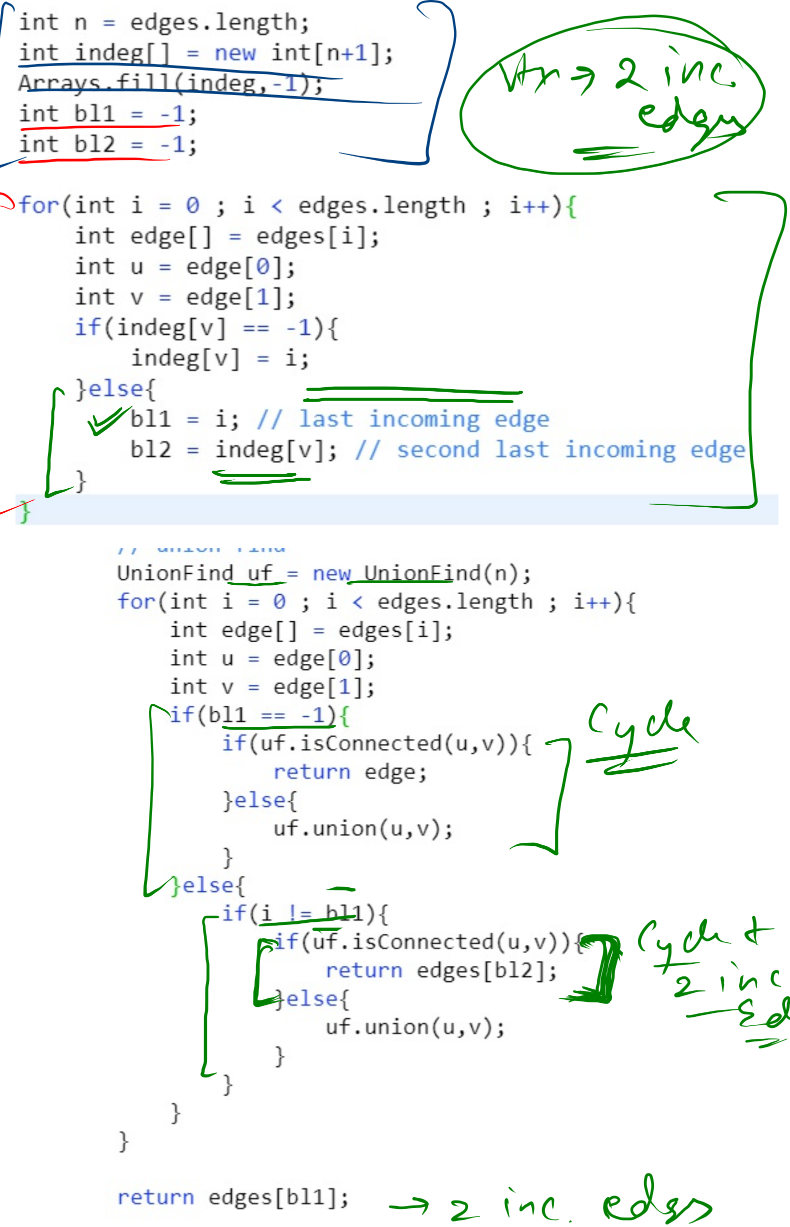


bl₁ → ~~-1~~ 4

bl₂ → ~~-1~~ 3



u v		i
2, 3	→	0
3, 4	→	1
4, 5	→	2
5, 2	→	3
~~1, 2~~	→	4



```
int n = edges.length;
int indeg[] = new int[n+1];
Arrays.fill(indeg,-1);
int bl1 = -1;
int bl2 = -1;
```

Vtx → 2 inc edges

```
for(int i = 0 ; i < edges.length ; i++){
    int edge[] = edges[i];
    int u = edge[0];
    int v = edge[1];
    if(indeg[v] == -1){
        indeg[v] = i;
    }else{
        bl1 = i; // last incoming edge
        bl2 = indeg[v]; // second last incoming edge
    }
}
```

```
// union find
UnionFind uf = new UnionFind(n);
for(int i = 0 ; i < edges.length ; i++){
    int edge[] = edges[i];
    int u = edge[0];
    int v = edge[1];
    if(bl1 == -1){
        if(uf.isConnected(u,v)){
            return edge;
        }else{
            uf.union(u,v);
        }
    }else{
        if(i != bl1){
            if(uf.isConnected(u,v)){
                return edges[bl2];
            }else{
                uf.union(u,v);
            }
        }
    }
}
```

Cycle

Cycle + 2 inc. Edges

```
return edges[bl1];
```

→ 2 inc. edges