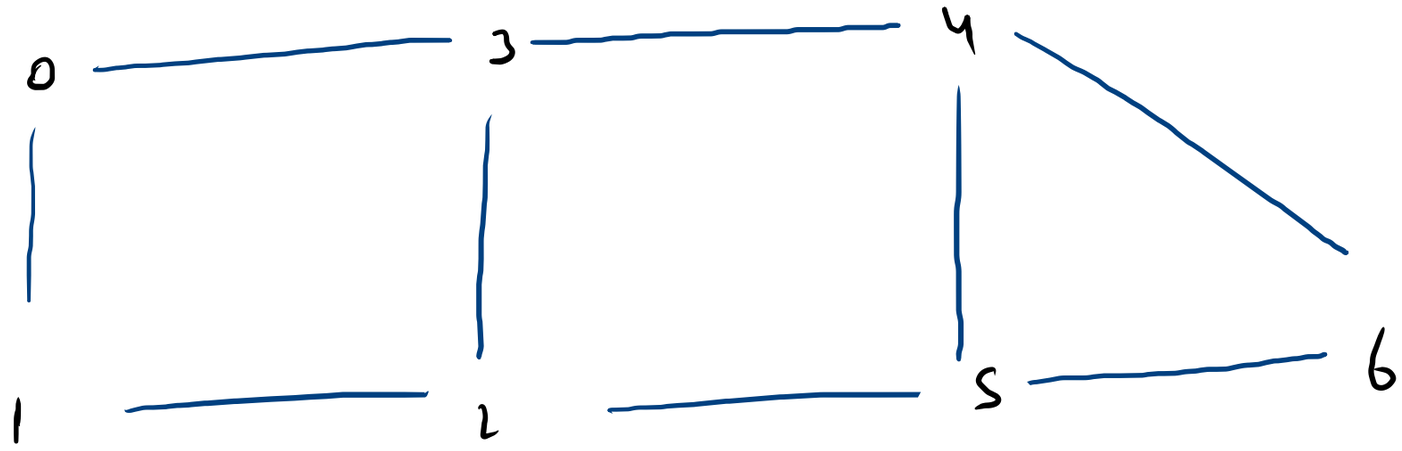
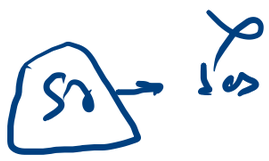
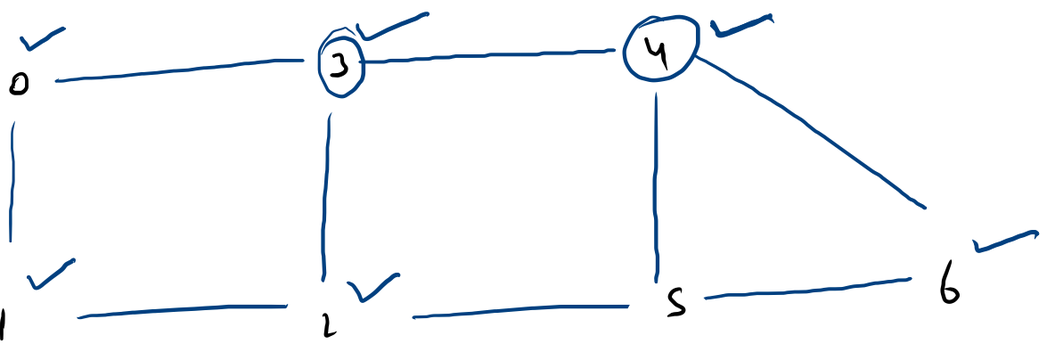


path | cycle

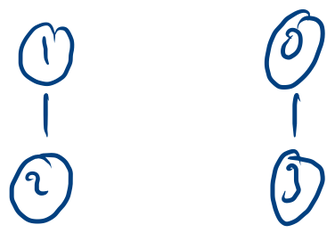
0 1 2 3 4 6 5.
0 1 2 3 4 5 6.
0 3 4 6 5 2 1*
0 1 2 5 6 4 3*

all nodes exactly once

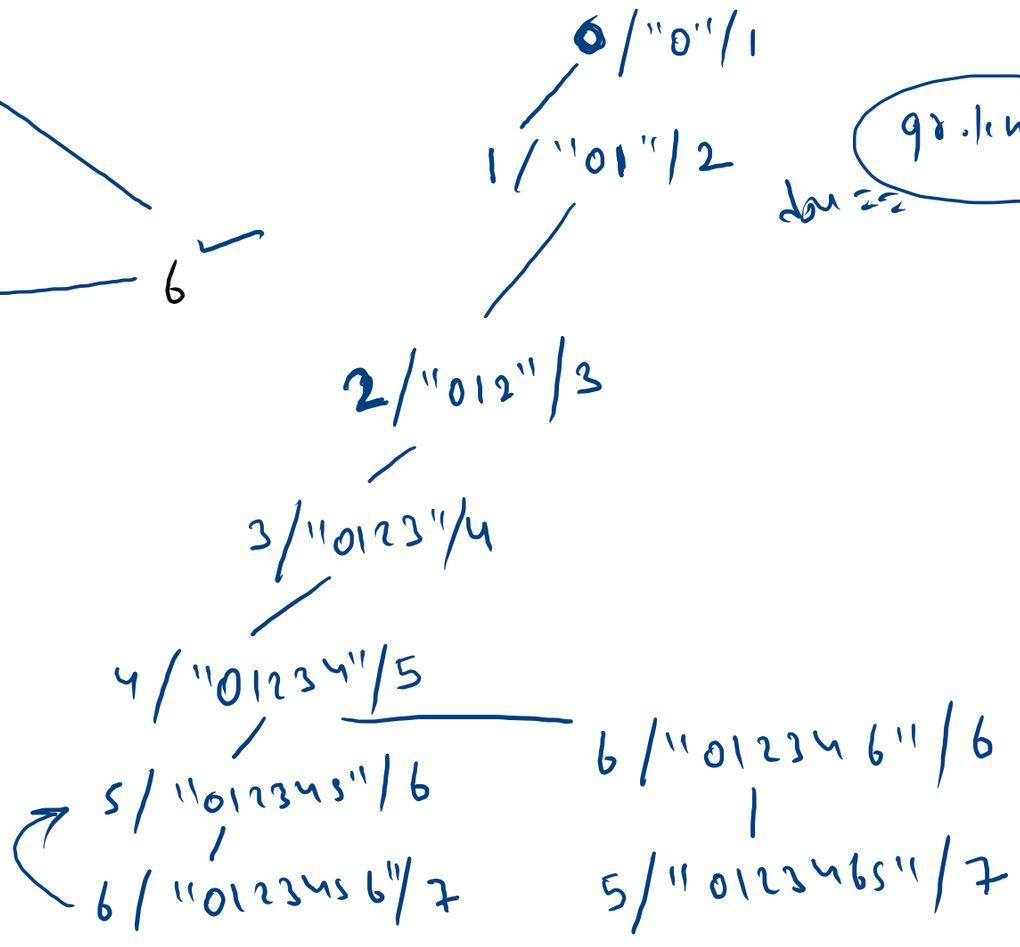



0
3
4
1
2
5
6

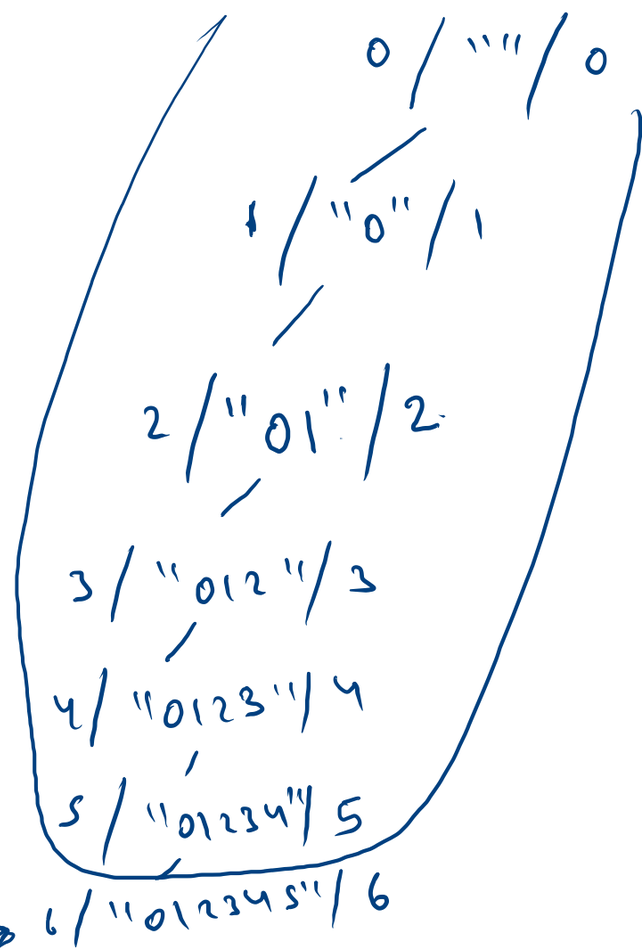
0 1 2 3 4 5 6


1
0
2
3

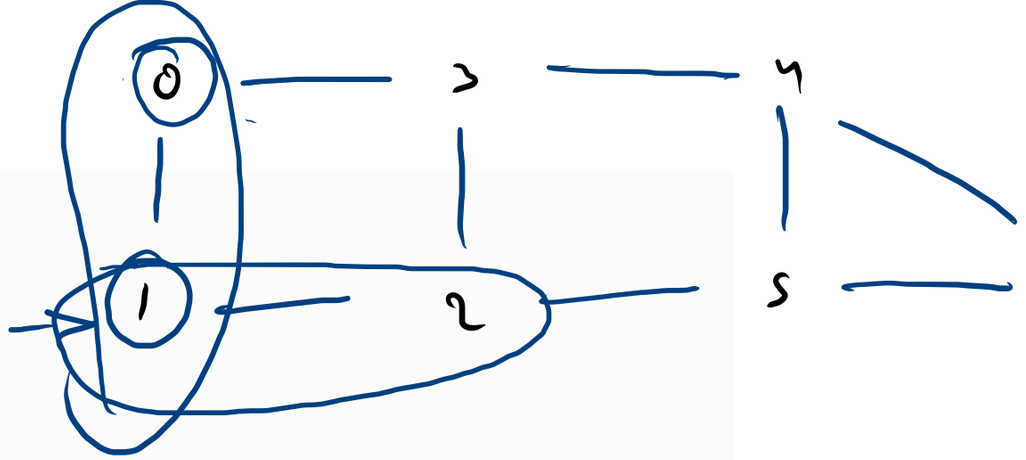
sx = 6


0/"0"/1
1/"01"/2
2/"012"/3
3/"0123"/4
4/"01234"/5
5/"012345"/6
6/"0123456"/7
6/"01234"/6
5/"012345"/7

qx.len-1


0/""/0
1/"0"/1
2/"01"/2
3/"012"/3
4/"0123"/4
5/"01234"/5
6/"012345"/6

```java
if(done == graph.length){
    boolean cycle = false;

    for(Edge e: graph[src]){
        if(e.nbr == osrc){
            cycle = true;
            break;
        }
    }
    if(cycle)
    System.out.println(path+"*");
    else System.out.println(path+".");
    return;
}

visited[src] = true;
for(Edge edge: graph[src]){
    if(visited[edge.nbr] == false){
        printHPC(graph, edge.nbr, visited, path+edge.nbr, done+1, osrc);
    }
}
visited[src] = false;
```



edge { src → nbr : 1 → 2 } 0

{ 1 → 0 }

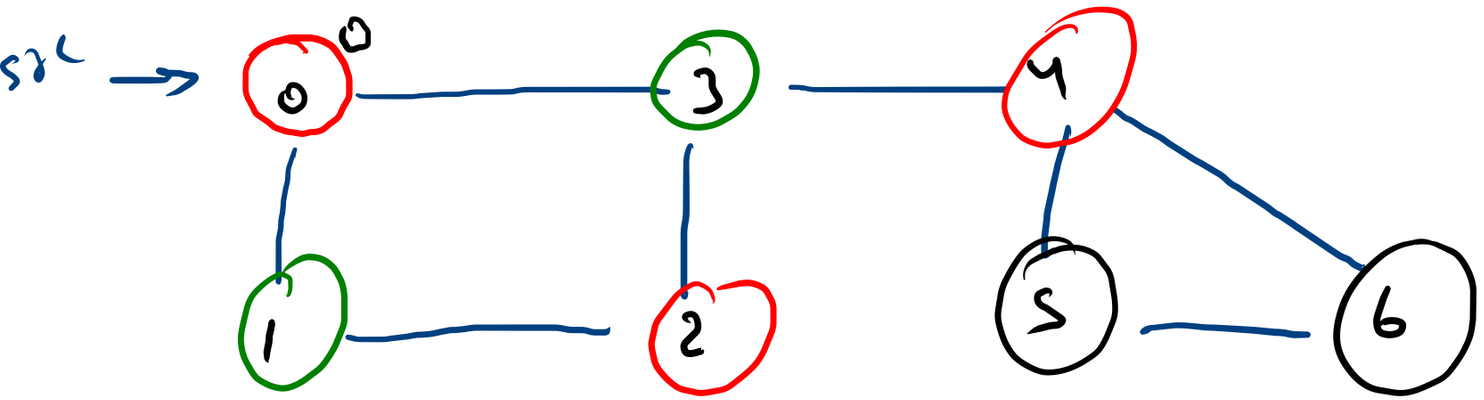
src / "0" / 1 / 0

/

1 / "01" / 2 / 0

src 6 / "0123456" / 7 / 0 osrc

1 / "0346521" / 7 / 0



0 → 4

⇒ 0 3 4

0 1 2 3 4

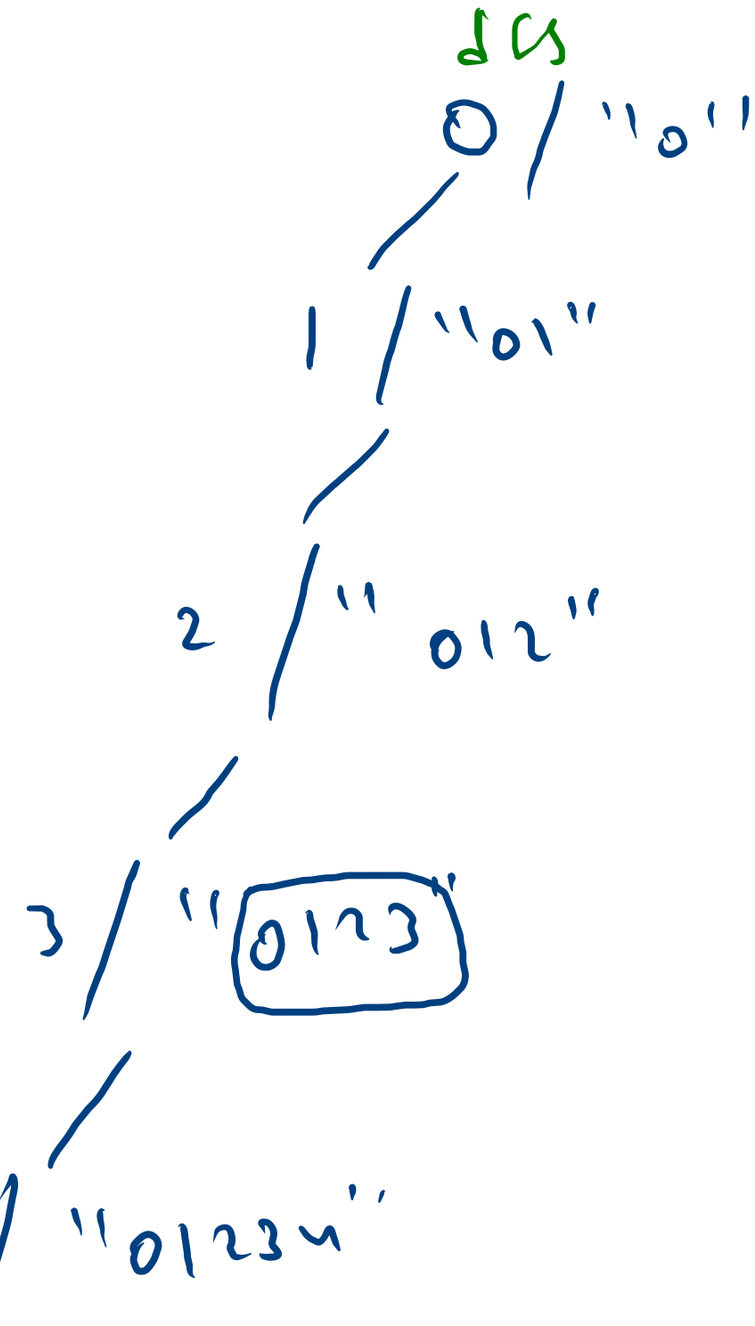
0 → 3

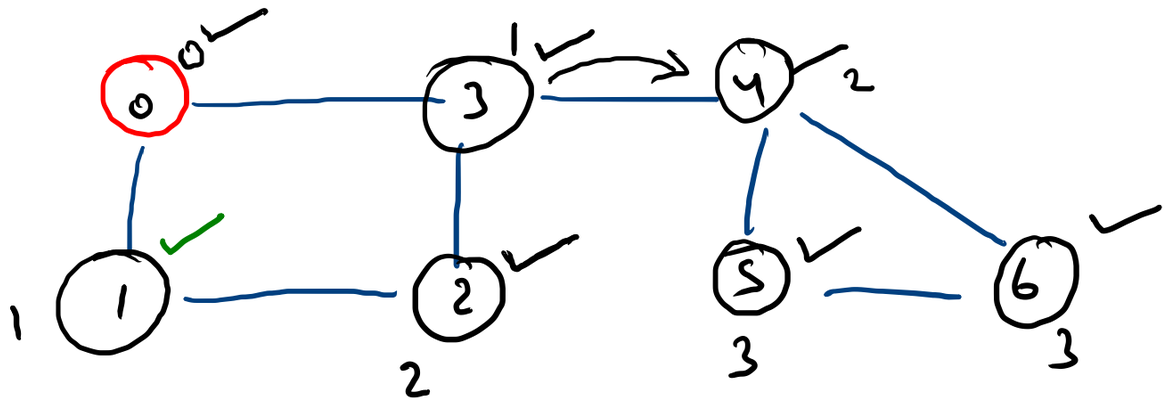
0 1 2 3

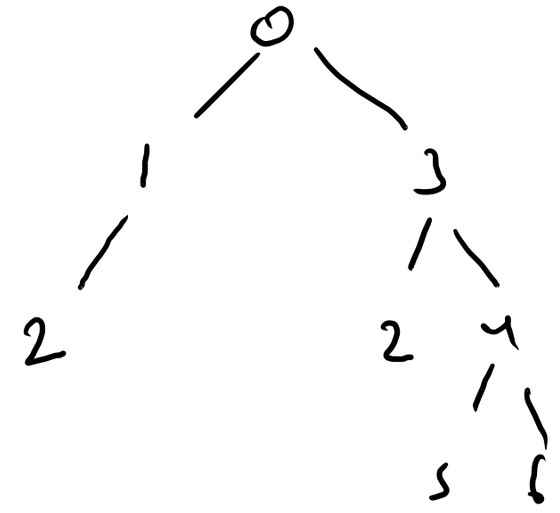
0 3

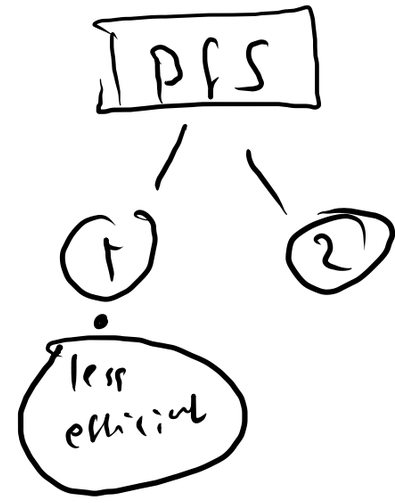
shortest path

0 3 4









r m p child

already done

0 @ 0

1 @ 01

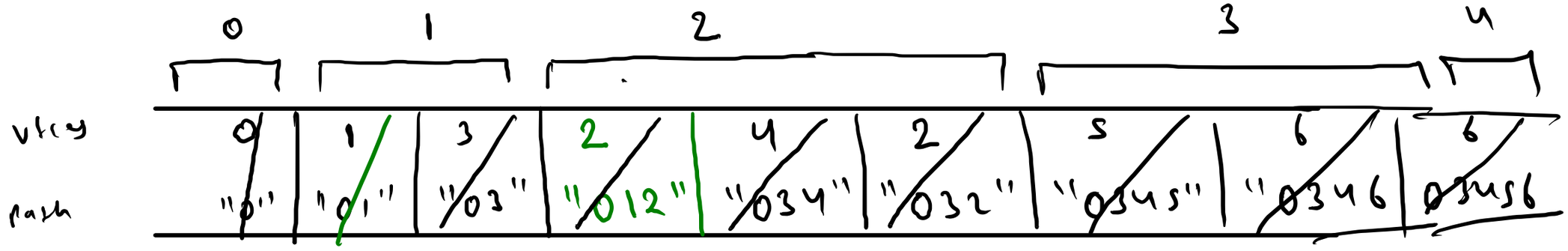
3 @ 03

2 @ 012

4 @ 034

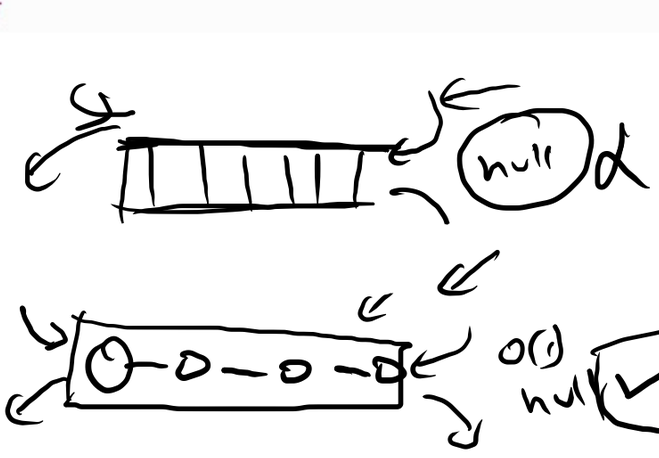
5 @ 0345

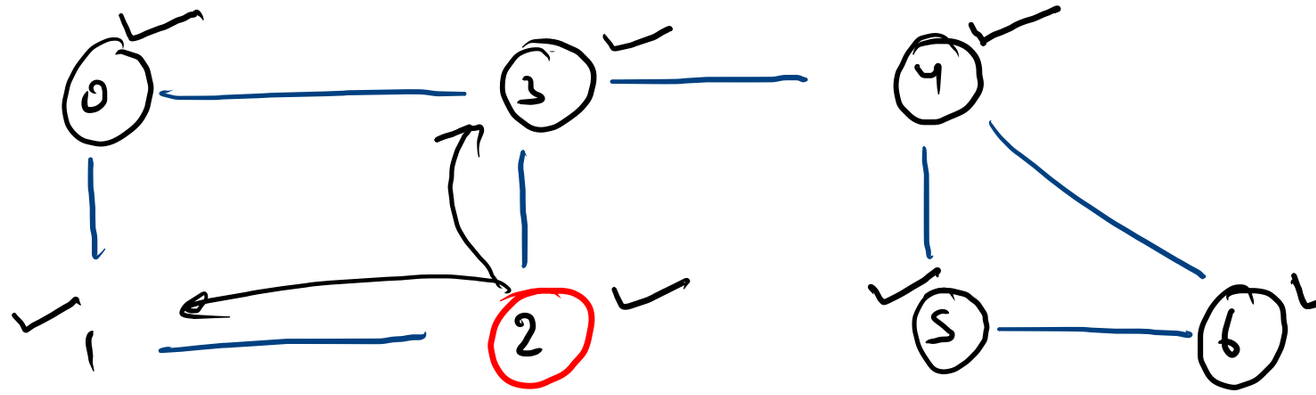
6 @ 0346

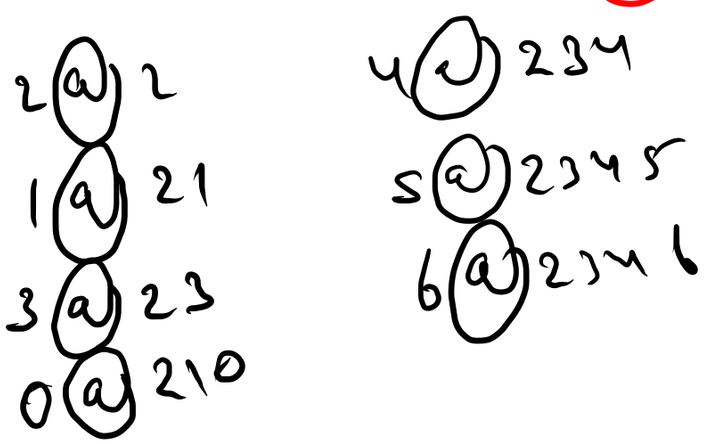


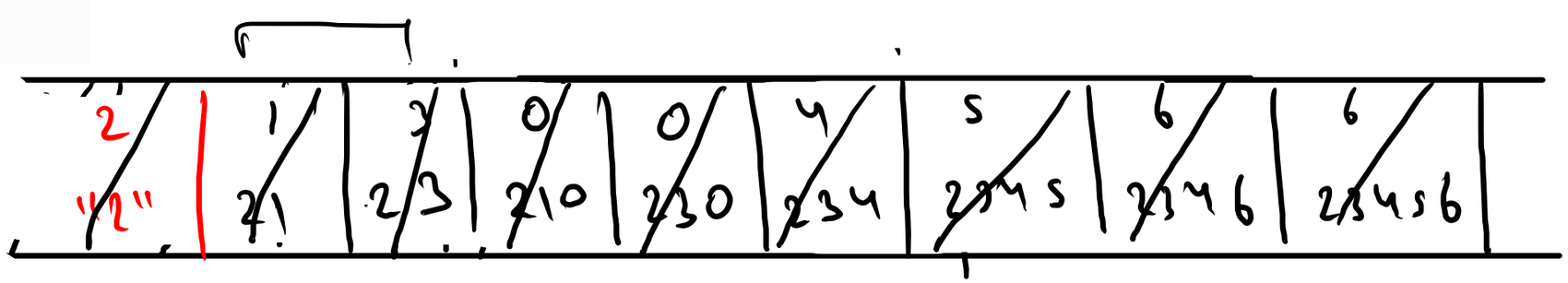
γ m p child

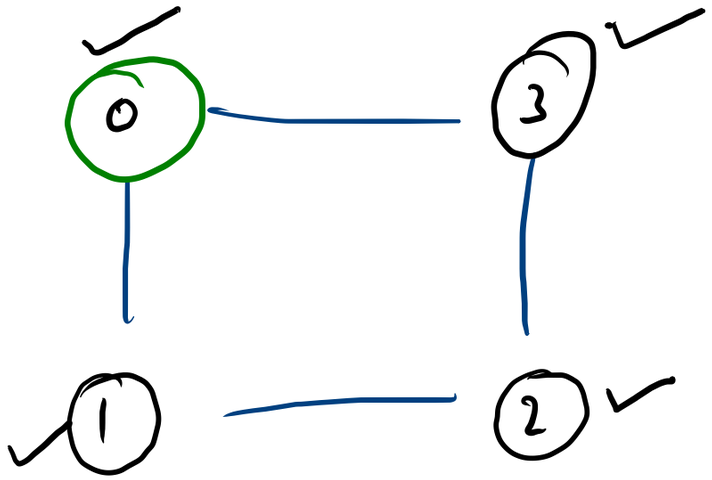
```
Queue<Pair> q = new LinkedList<>();// add remove size

q.add(new Pair(src, ""+src));
boolean visited[] = new boolean[vtces];

while(q.size() > 0){
    Pair p = q.remove();
    if(visited[p.des] == true){
        continue;
    }else{
        visited[p.des] = true;
    }
    System.out.println(p.des+"@"+p.path);
    for(Edge e : graph[p.des]){
        if(visited[e.nbr] == false)
        q.add(new Pair(e.nbr, p.path+e.nbr));
    }
}
```



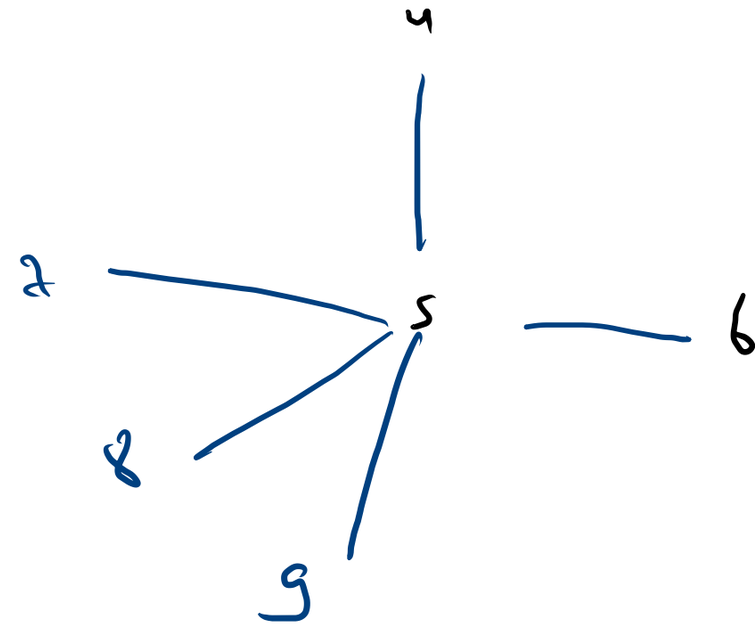


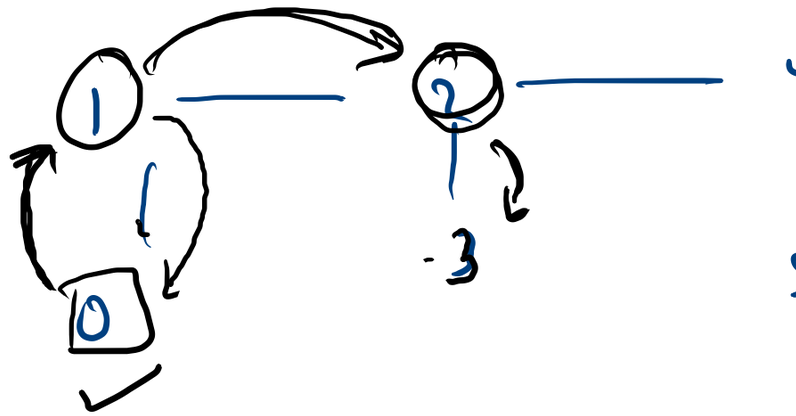


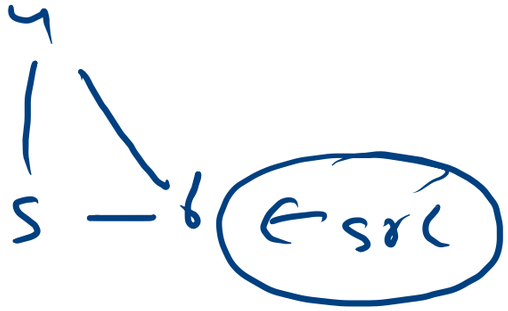


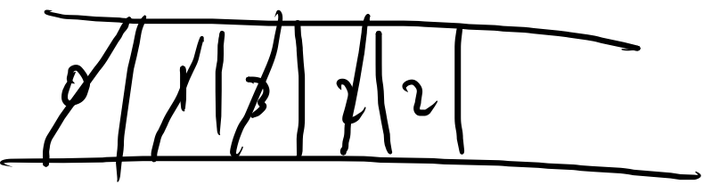


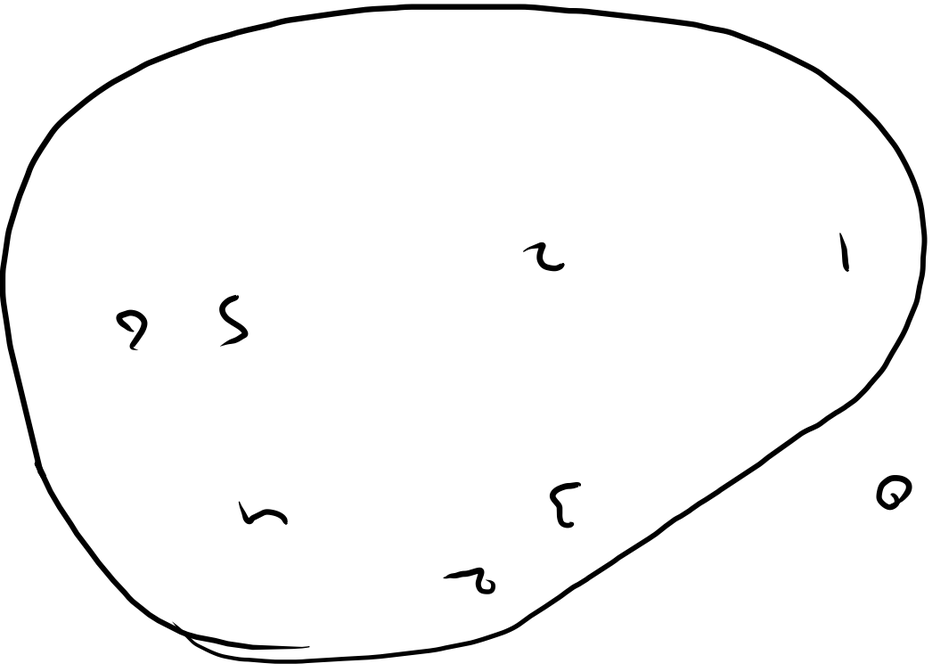
0 to 0



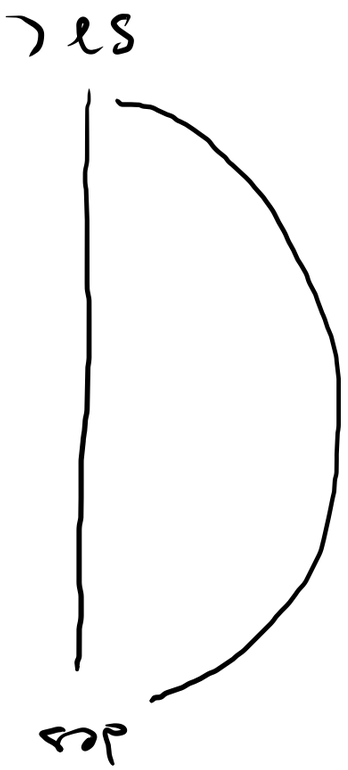


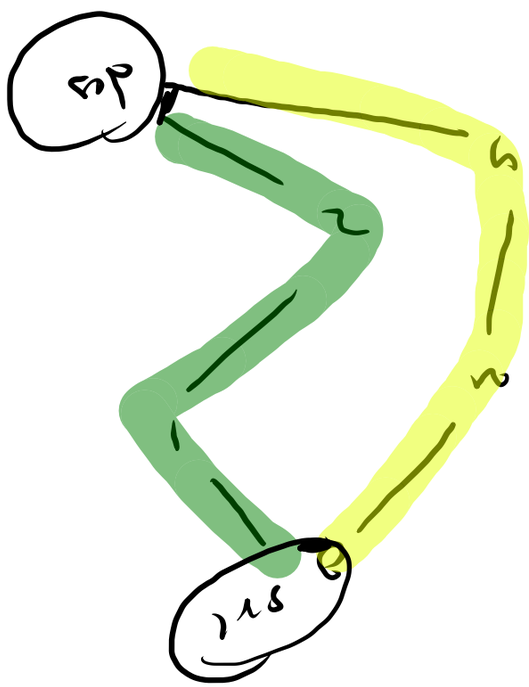


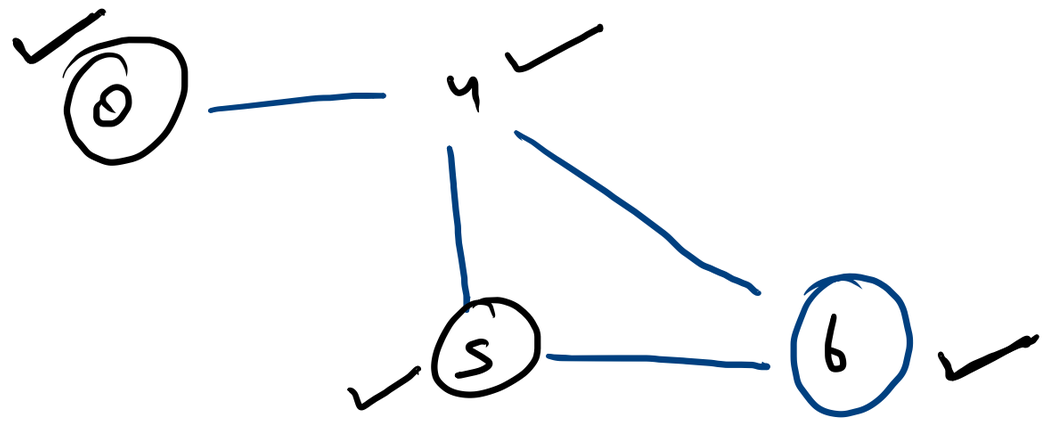




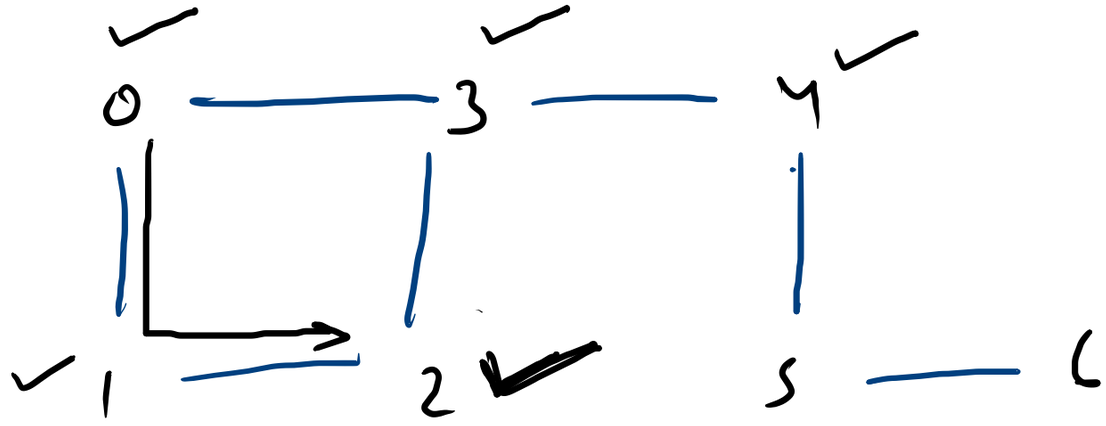
1 path
2 path

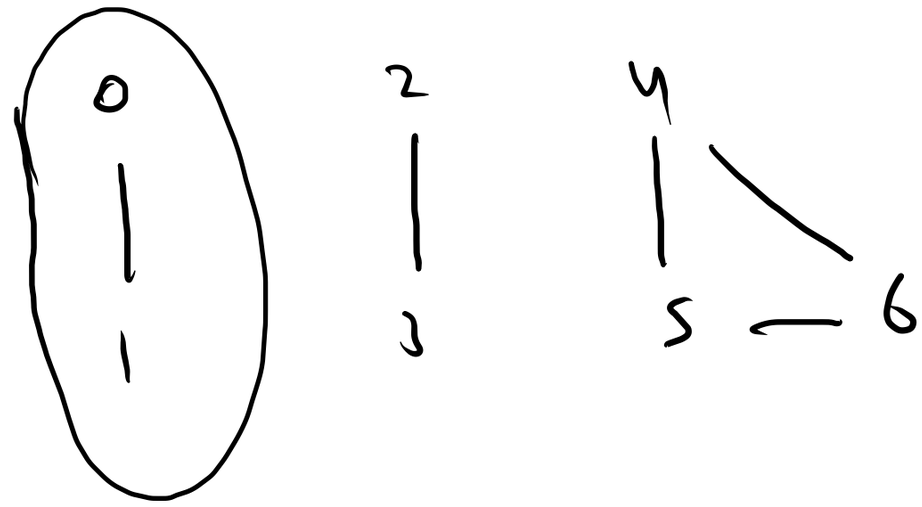






child



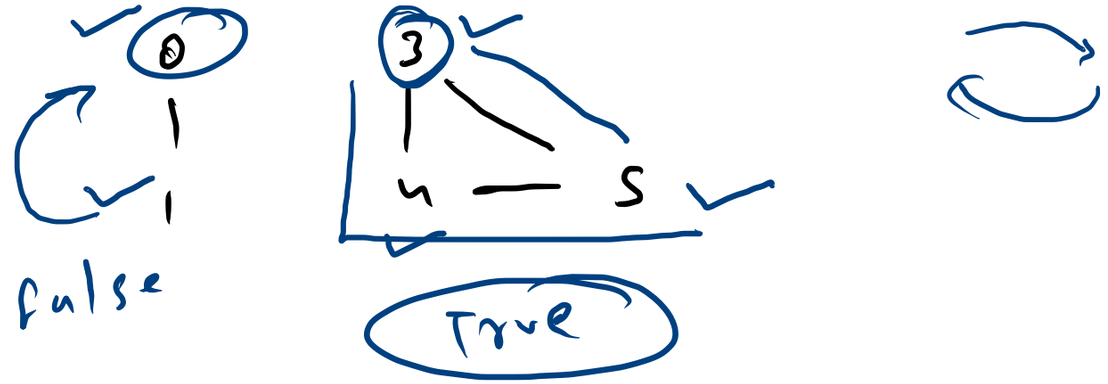


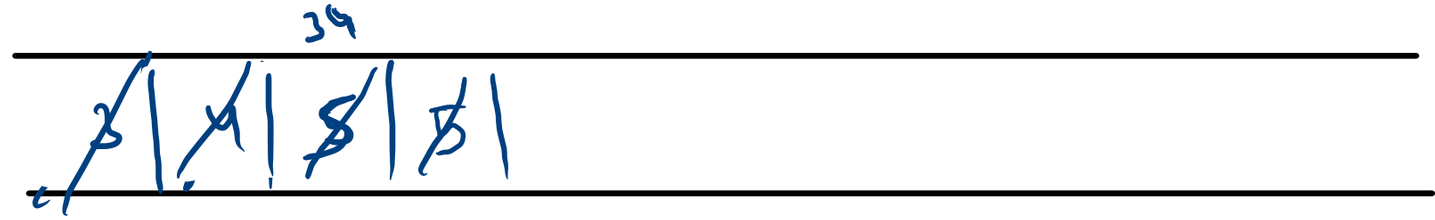
γ mark child

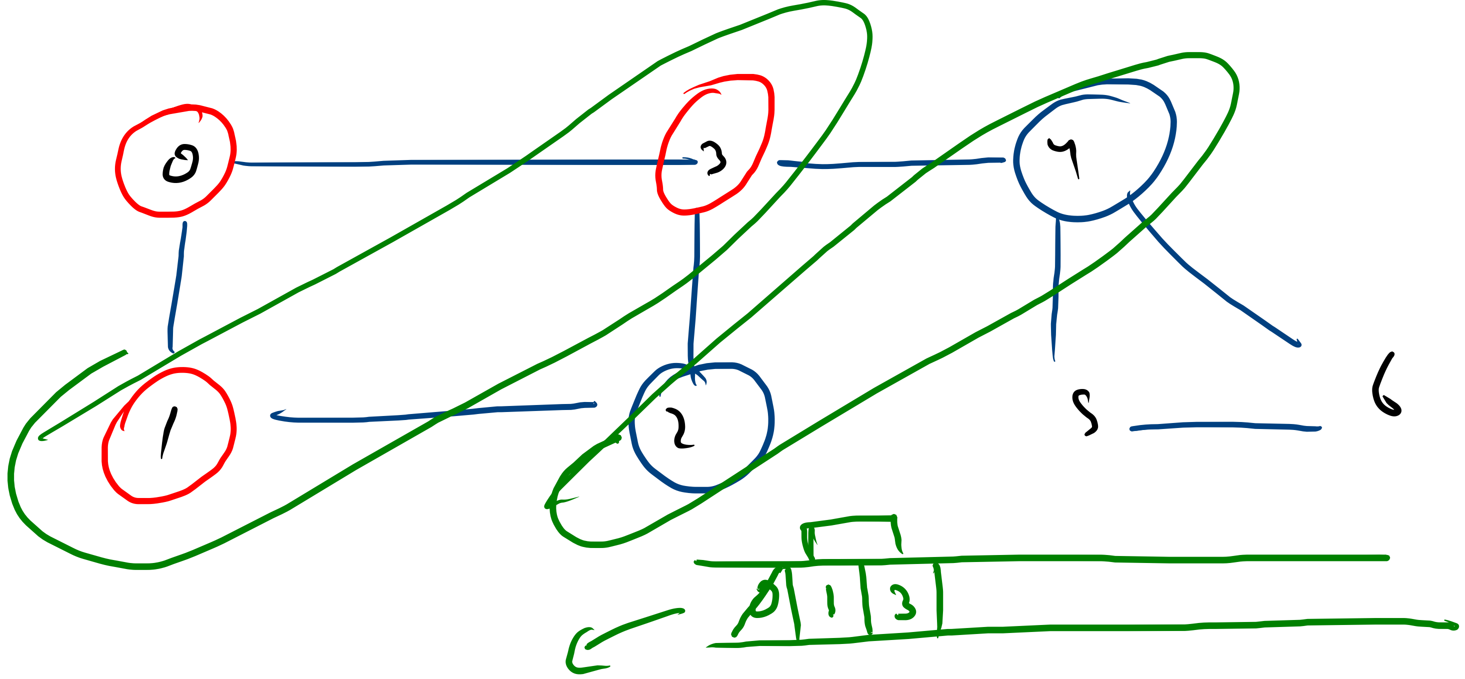
already marked cycle

∅	1	2	3	4	5	5

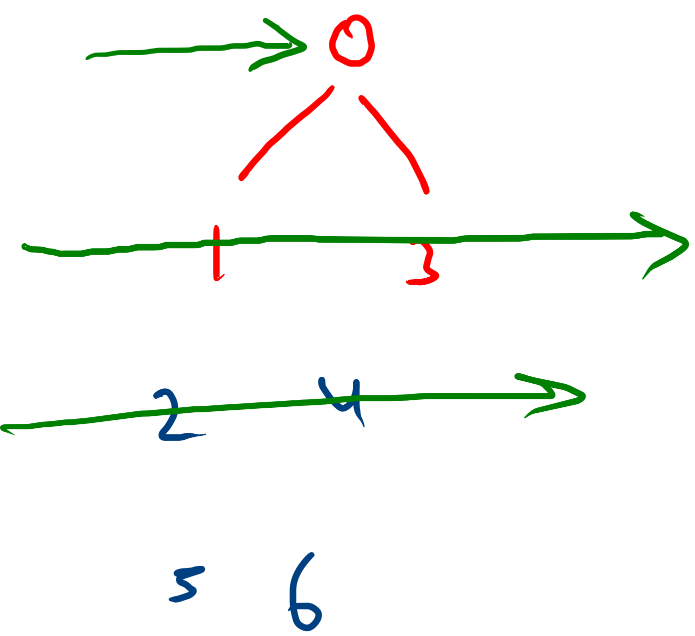
```java
Queue<Integer> q = new ArrayDeque<>();// ad

q.add(src);

while(q.size() > 0){
    Integer p = q.remove();
    if(visited[p] == true){
        return true;
    }else{
        visited[p] = true;
    }
    // System.out.println(p.des+"@"+p.path)
    for(Edge e : graph[p]){
        if(visited[e.nbr] == false)
        q.add(e.nbr);
    }
}

    return false;
}
```

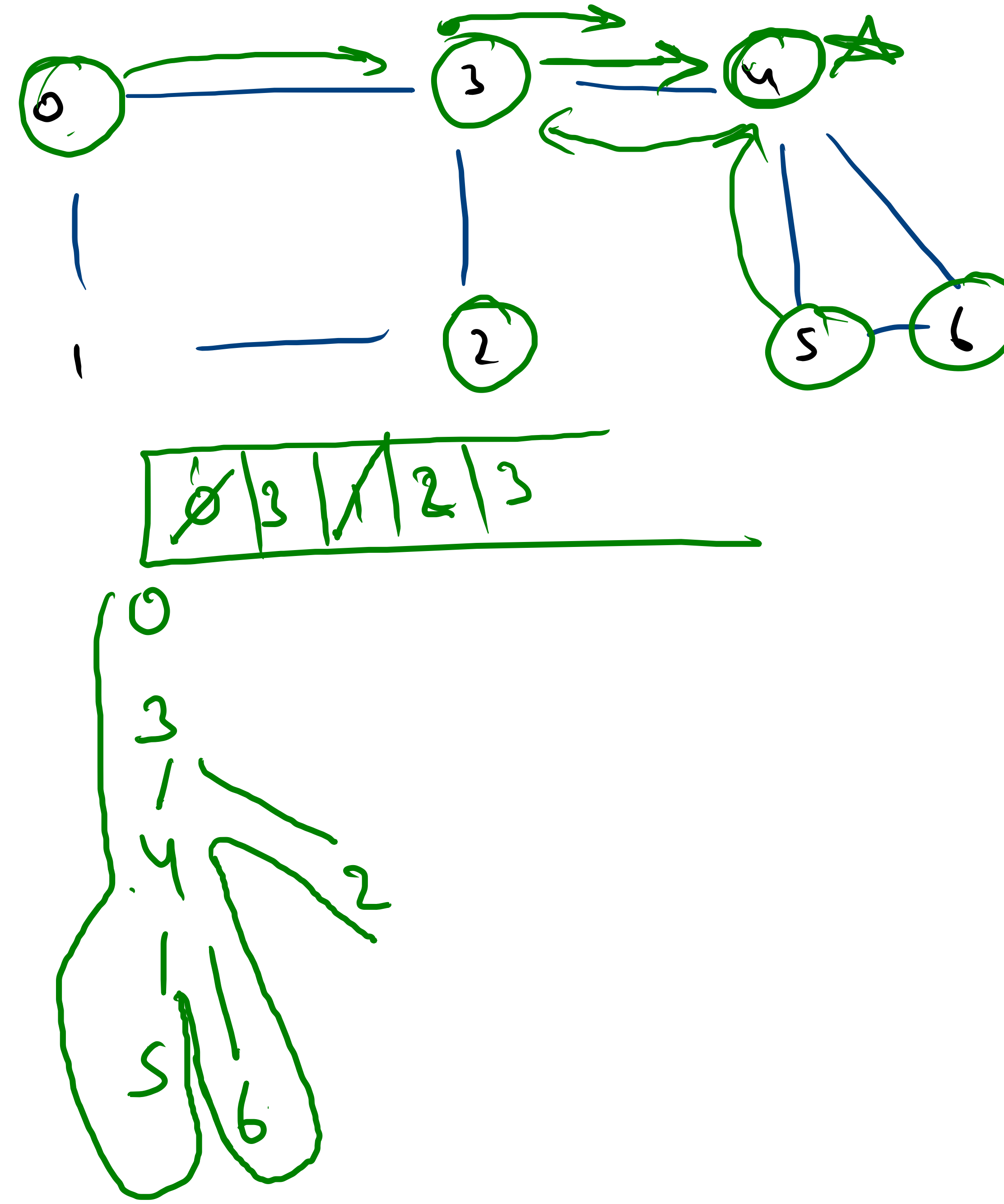


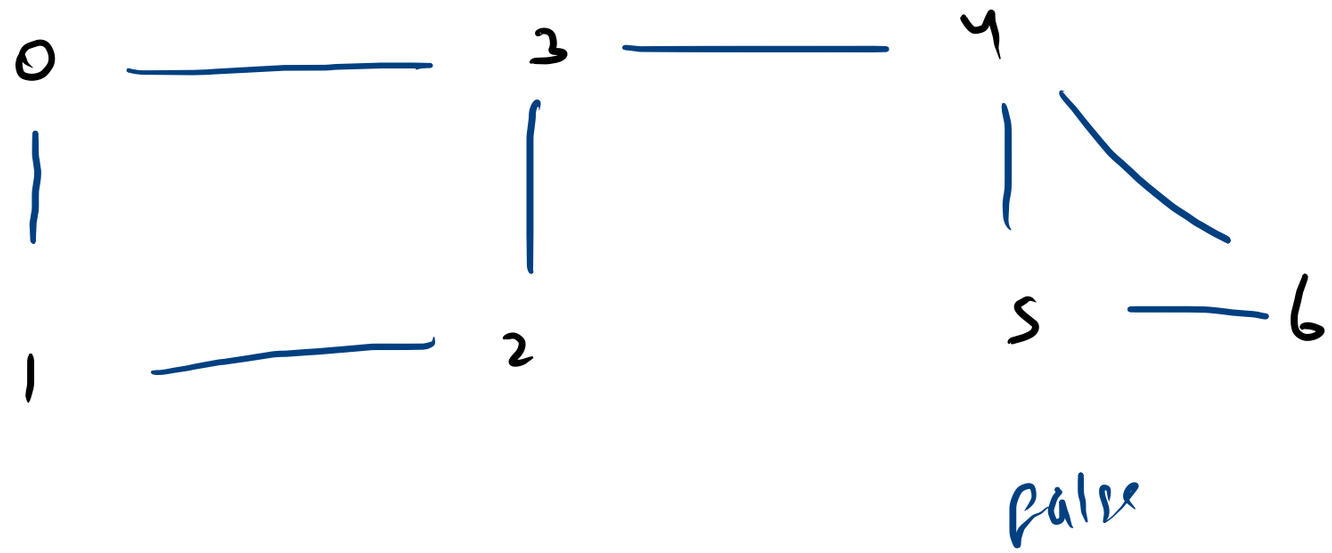




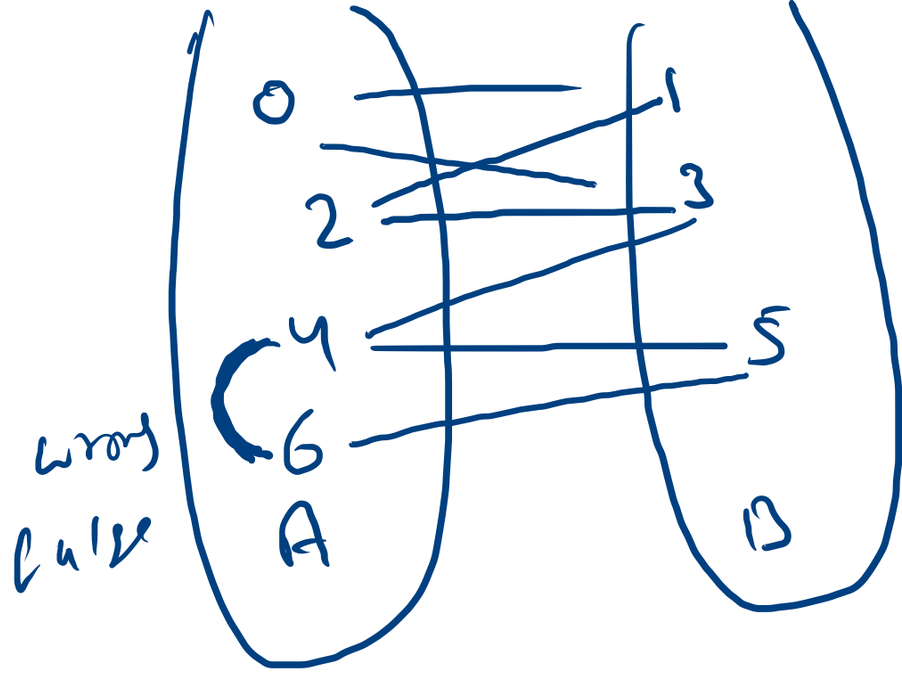
bfs child have same parent



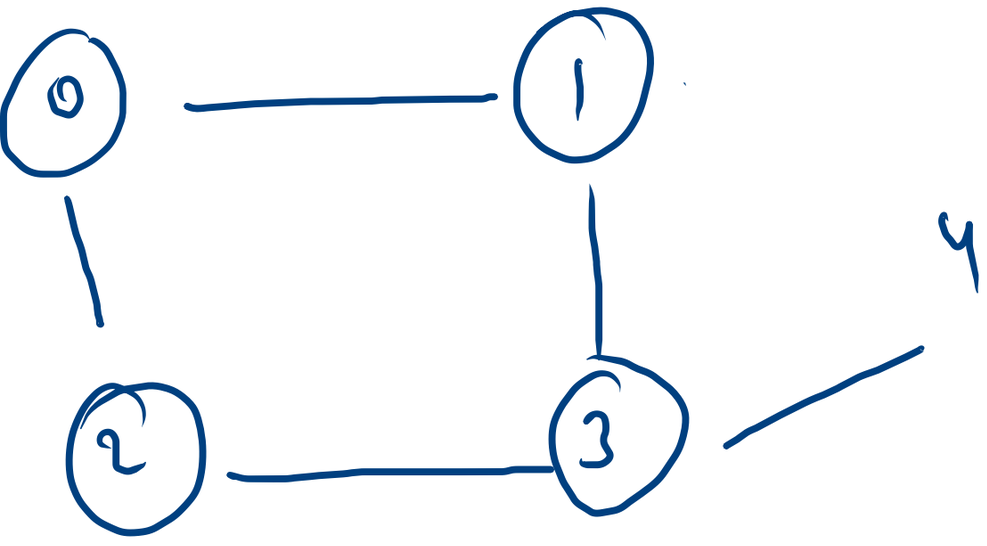


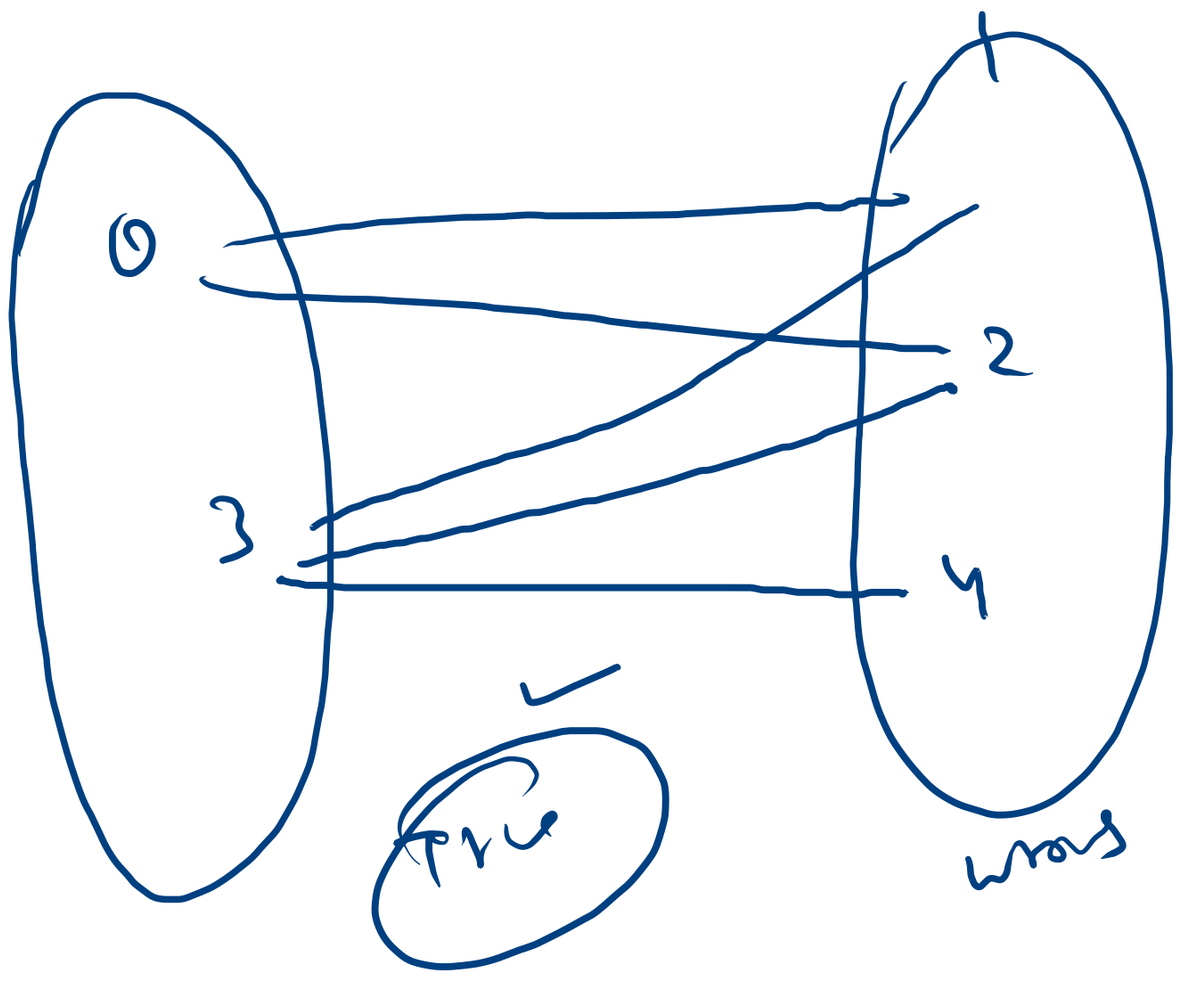


False



wrong
False





non cycle

true

1 — 2 — 3 — 4

4 — 3

1 — 2

A B

cycle

true

even

1 — 2

3 — 4

checking cycle

4 — 3

1 — 2

A B

false odd

1 — 2

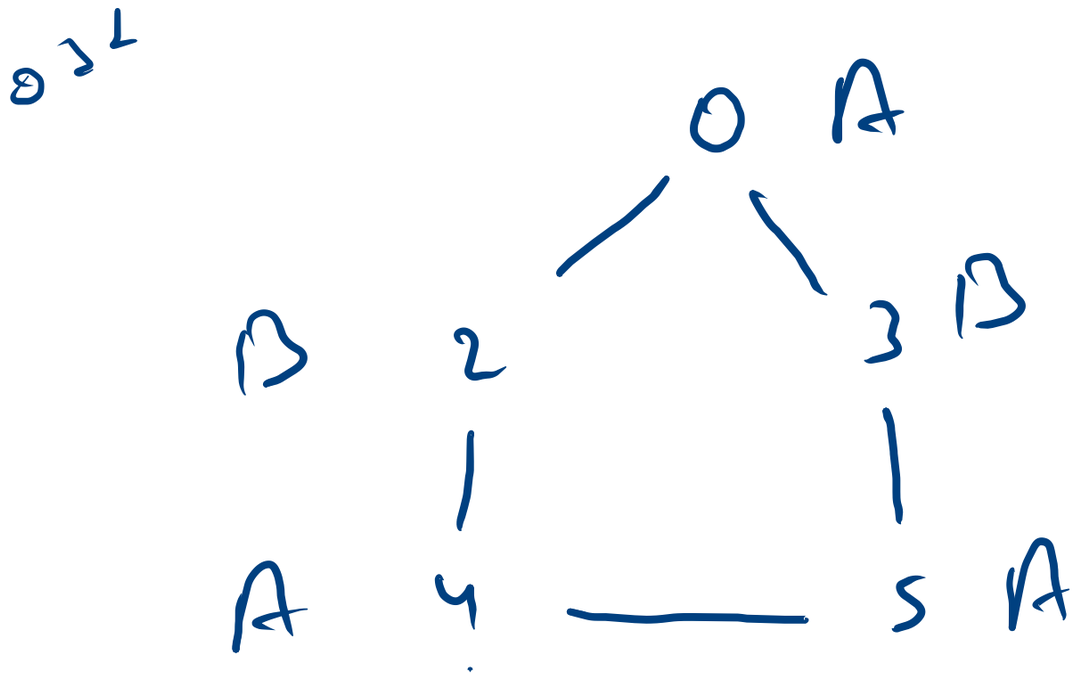
3

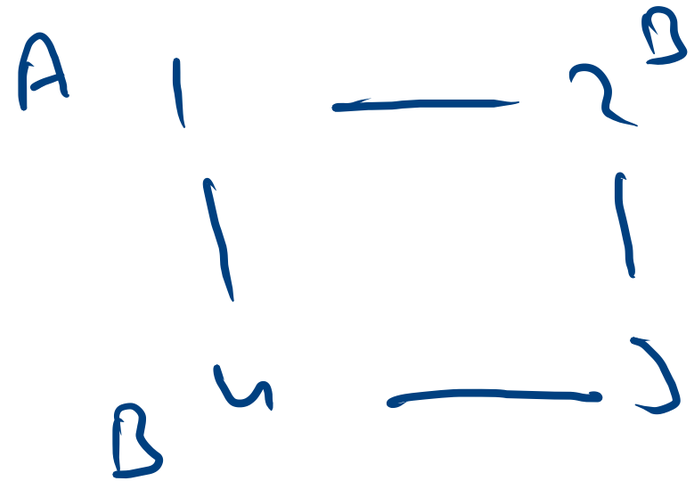
3 — 2

1

A B

Rule


O A
B 2 3 B
A 4 5 A


A 1 2 B
B 4 3


2
1 3
A B


3 2
1 4

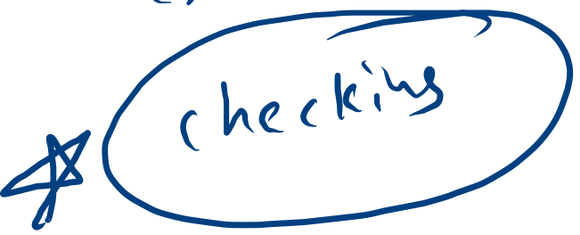
even

3 4

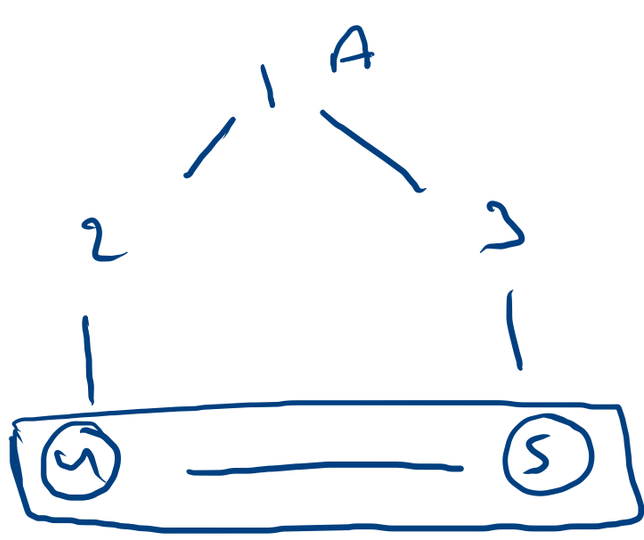
1 2

A B

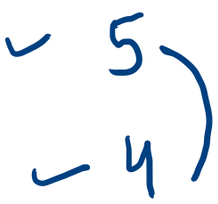
cycle



odd



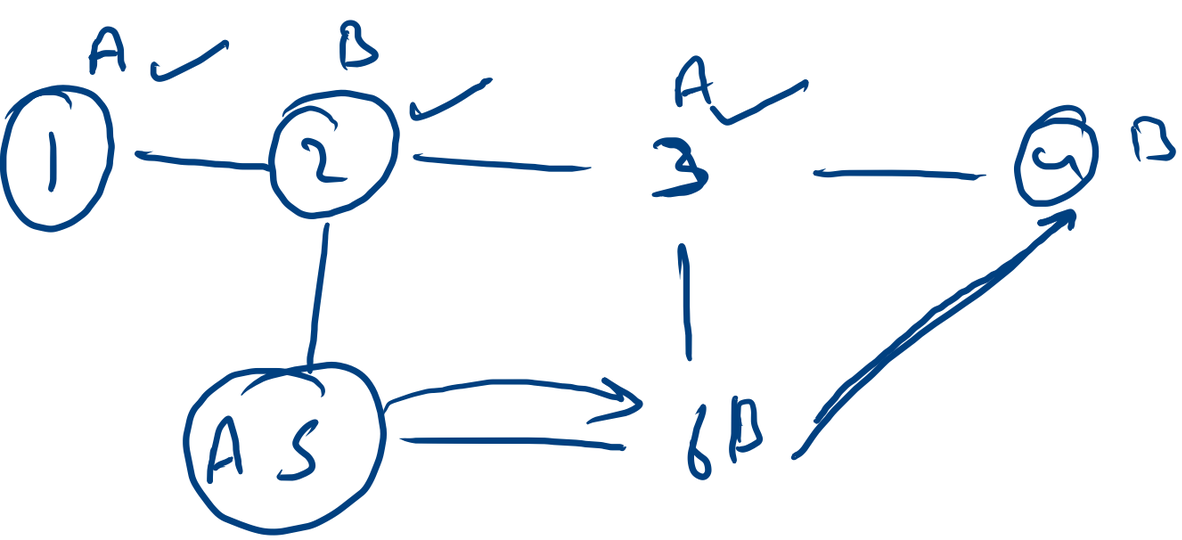
false



3

1 2

A B



checks → same → calc

nbr

not in group
assign opposit group

is in group

check same → false

3 4

5 6

1 2

A B