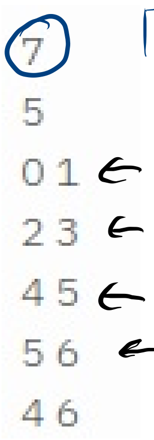
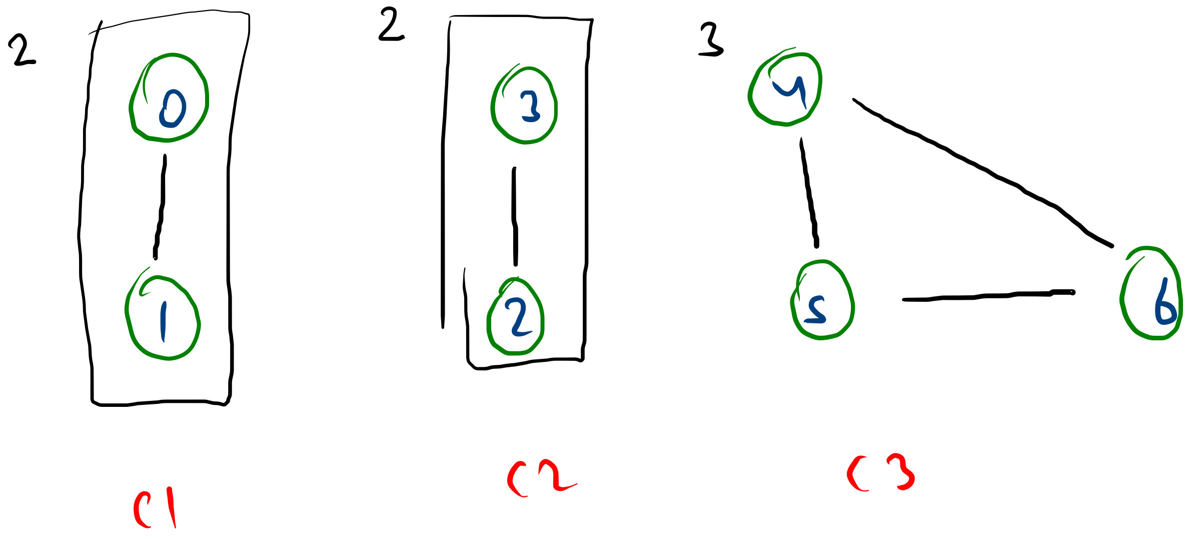
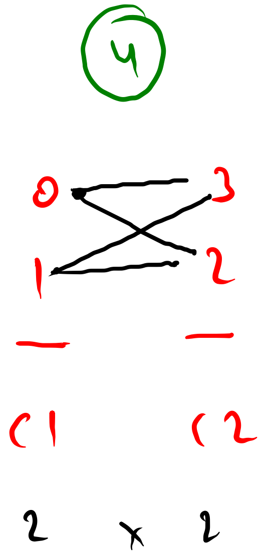
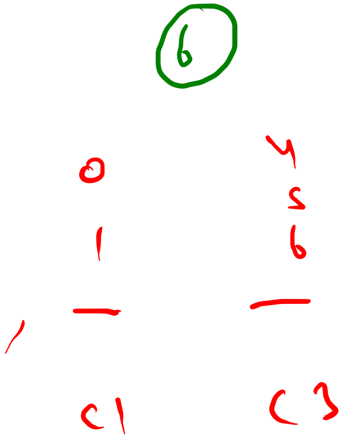


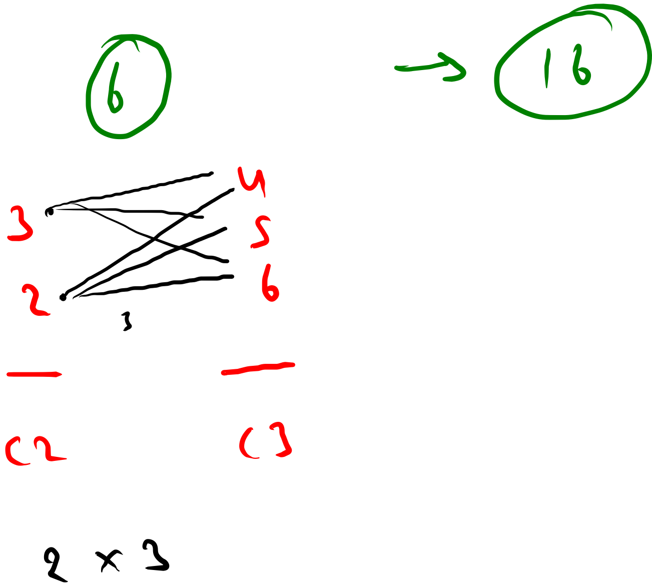
```
7
5
0 1
2 3
4 5
5 6
4 6
```

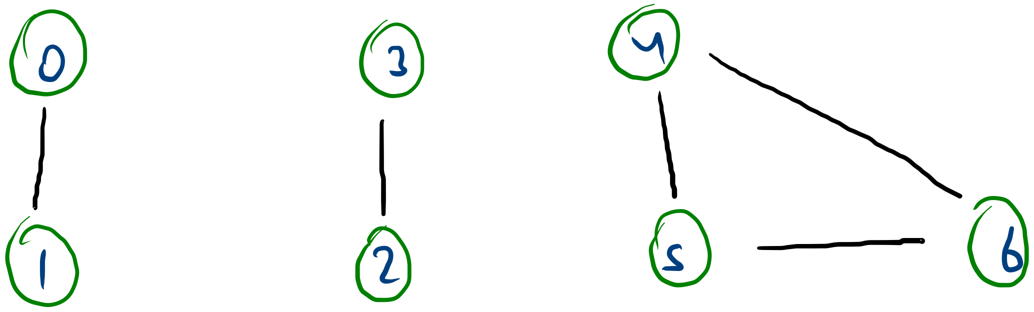
0 . . . 6

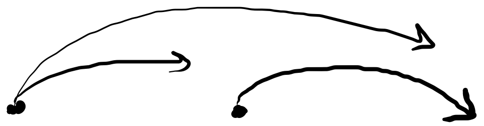










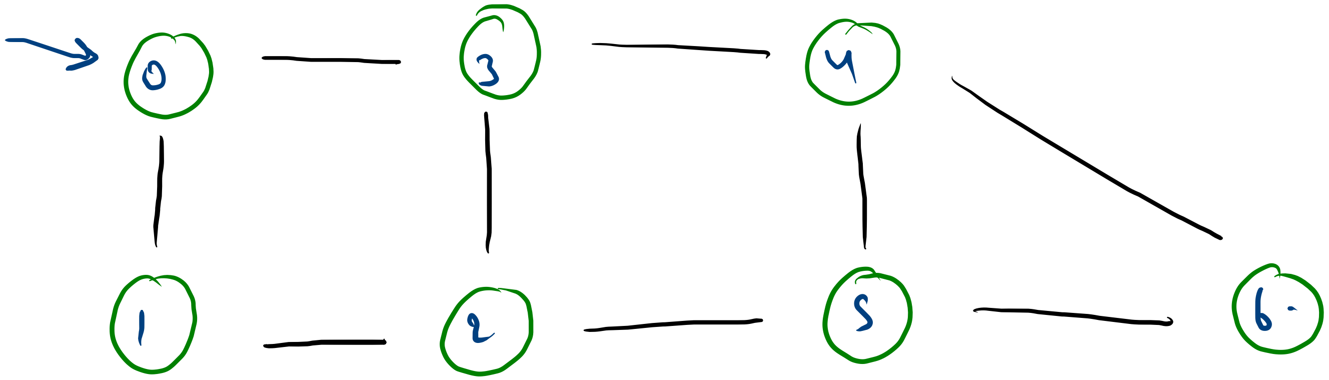


c1	c2
c1	c3
c2	c3

[2 , 2 , 3]

c1 c2 c3

c1	c2	2 × 2 = 4
c1	c3	2 × 3 = 6
c2	c3	2 × 3 = 6
		16



visited True

[call neighbour

visited False

src = 0

ans

0 1 2 3 4 5 6 • ← path no direct edge

0 1 2 3 4 6 5 •

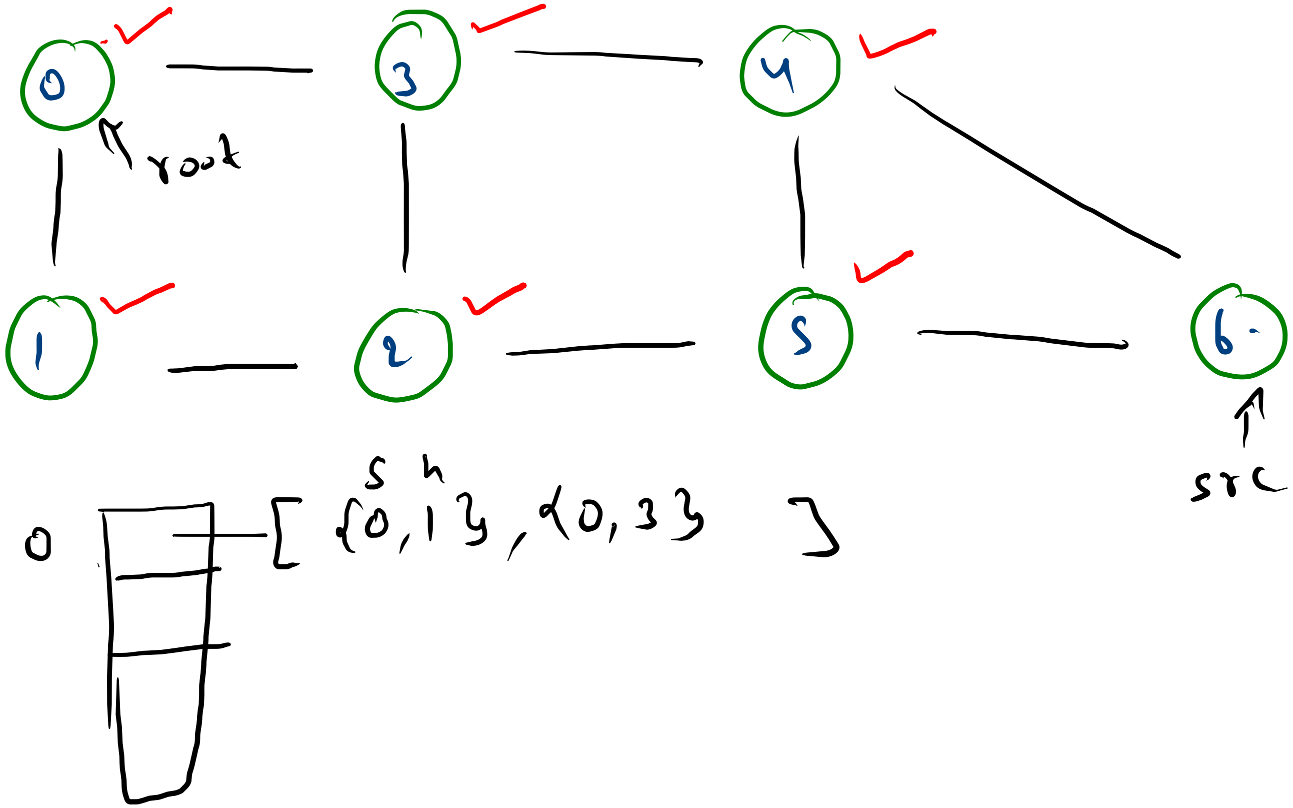
0 3 4 6 5 2 1 ★ ← direct edge cycle

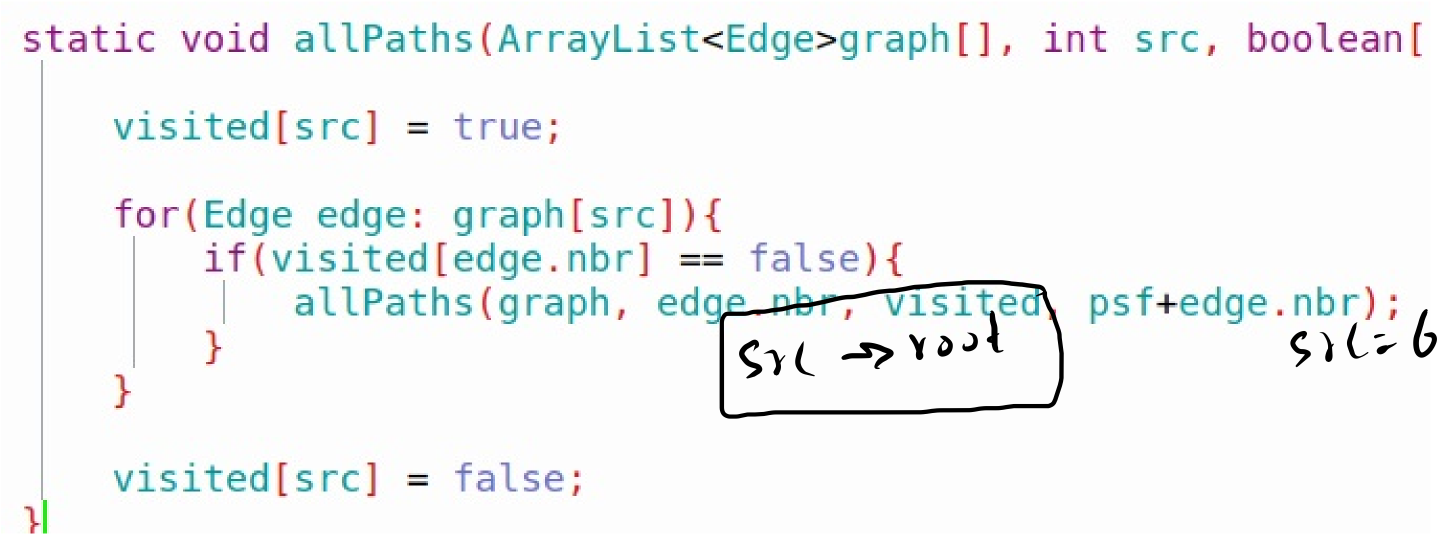
0 1 2 5 6 4 3 ★

src

all path

dest

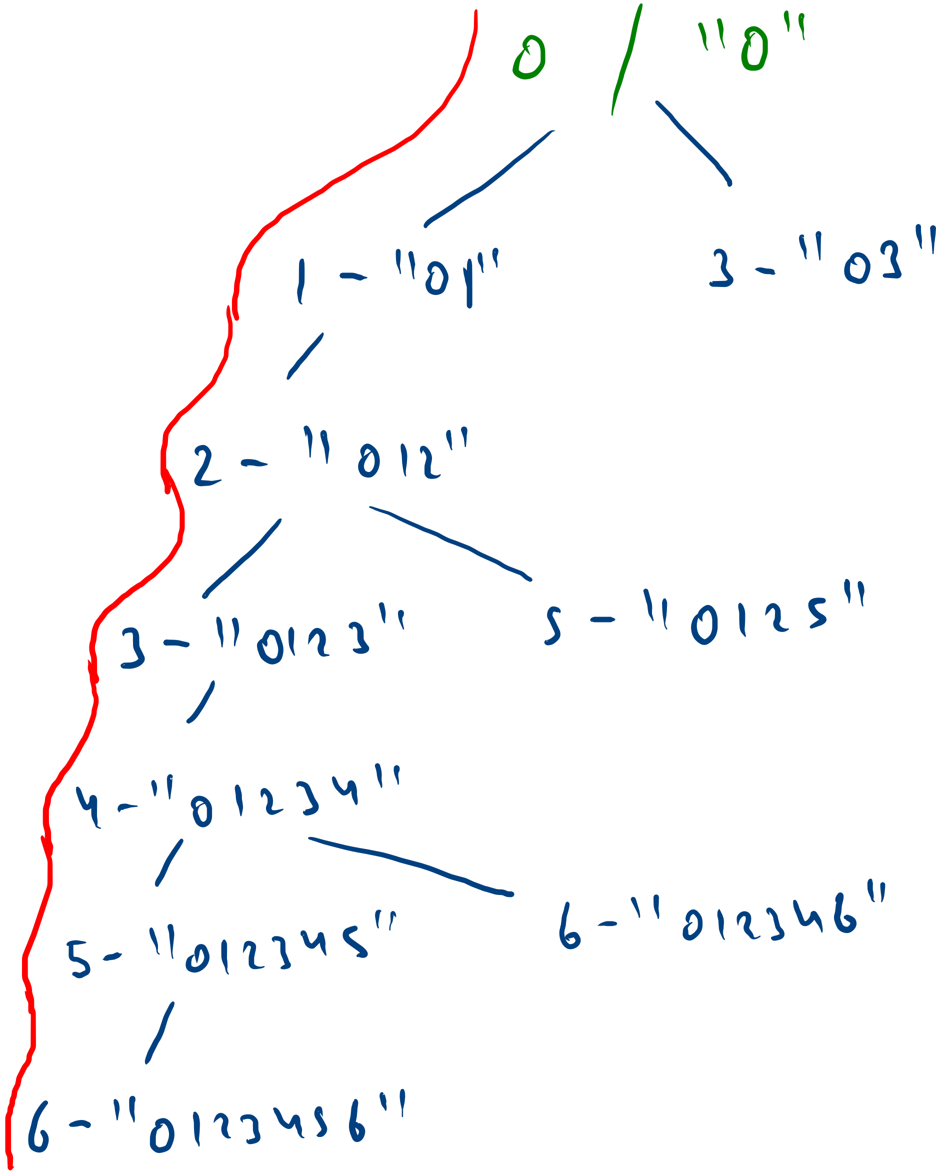


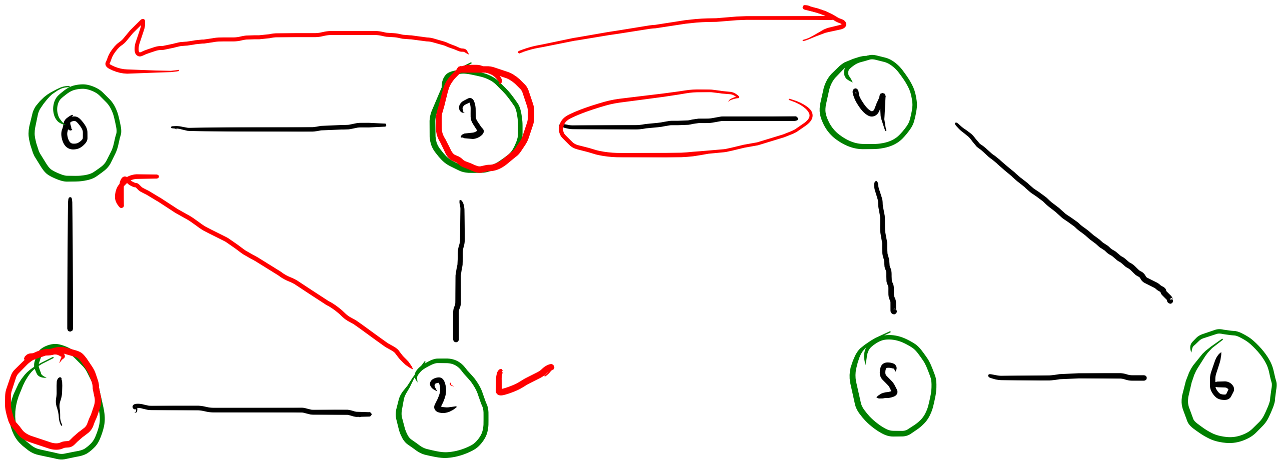


```
static void allPaths(ArrayList<Edge>graph[], int src, boolean[

    visited[src] = true;

    for(Edge edge: graph[src]){
        if(visited[edge.nbr] == false){
            allPaths(graph, edge.nbr, visited, psf+edge.nbr);
        }
    }

    visited[src] = false;
}
```

src → root

src=6





0 edge distanc

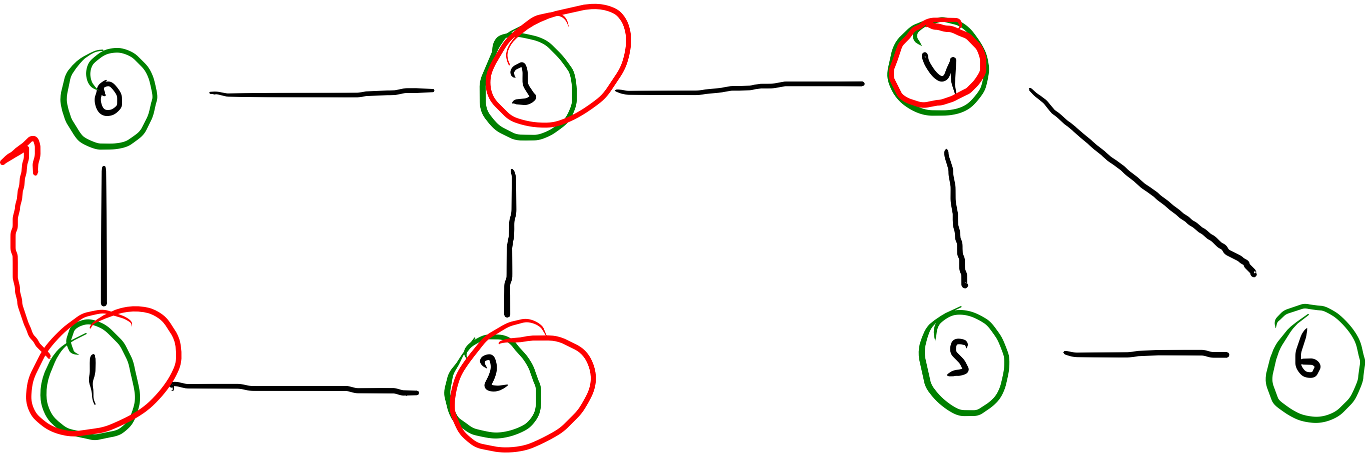
src→ 2

0 [2 @ 2

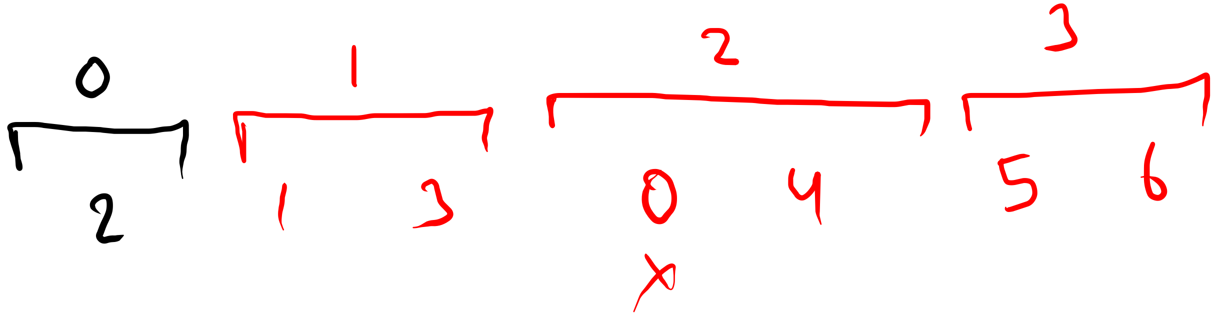
1 [3 @ 23
1 @ 21

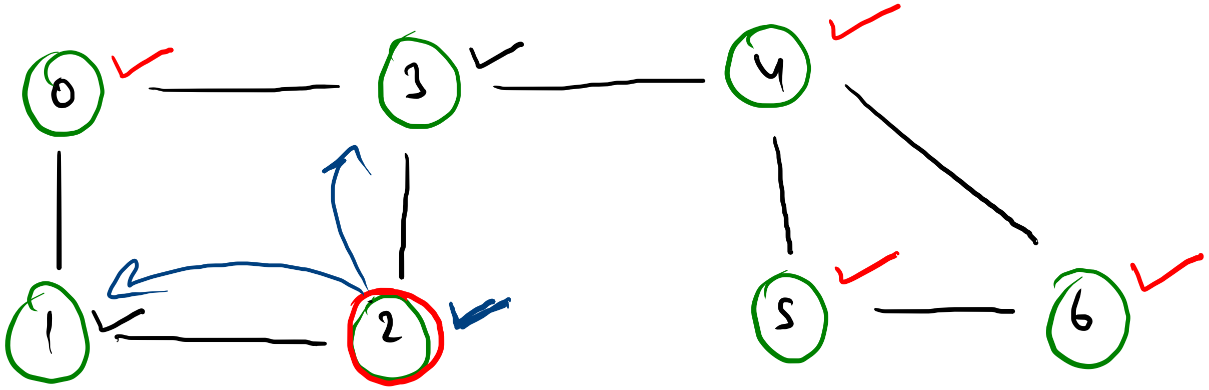
2 [0 @ 210 , 0 @ 230
4 @ 234

3 [5 @ 2345
6 @ 2346



edge



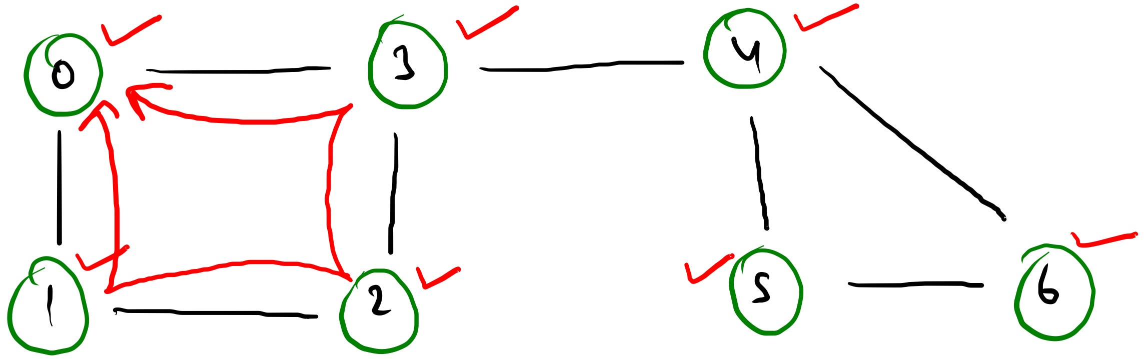


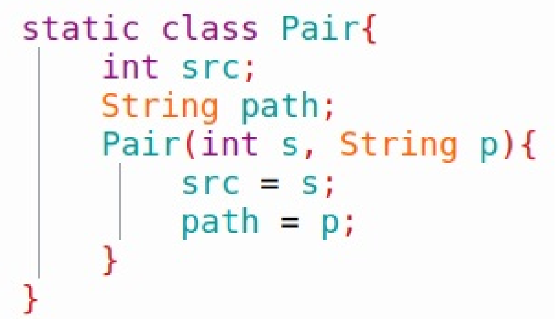
γ m p addchild

0	1		2			3		4
~~2~~	~~1~~	~~3~~	~~0~~	~~0~~	~~4~~	~~5~~	~~6~~	~~6~~
~~"2"~~	~~"21"~~	~~"23"~~	~~"210"~~	~~"230"~~	~~"234"~~	~~"2345"~~	~~"2346"~~	~~"23456"~~

P

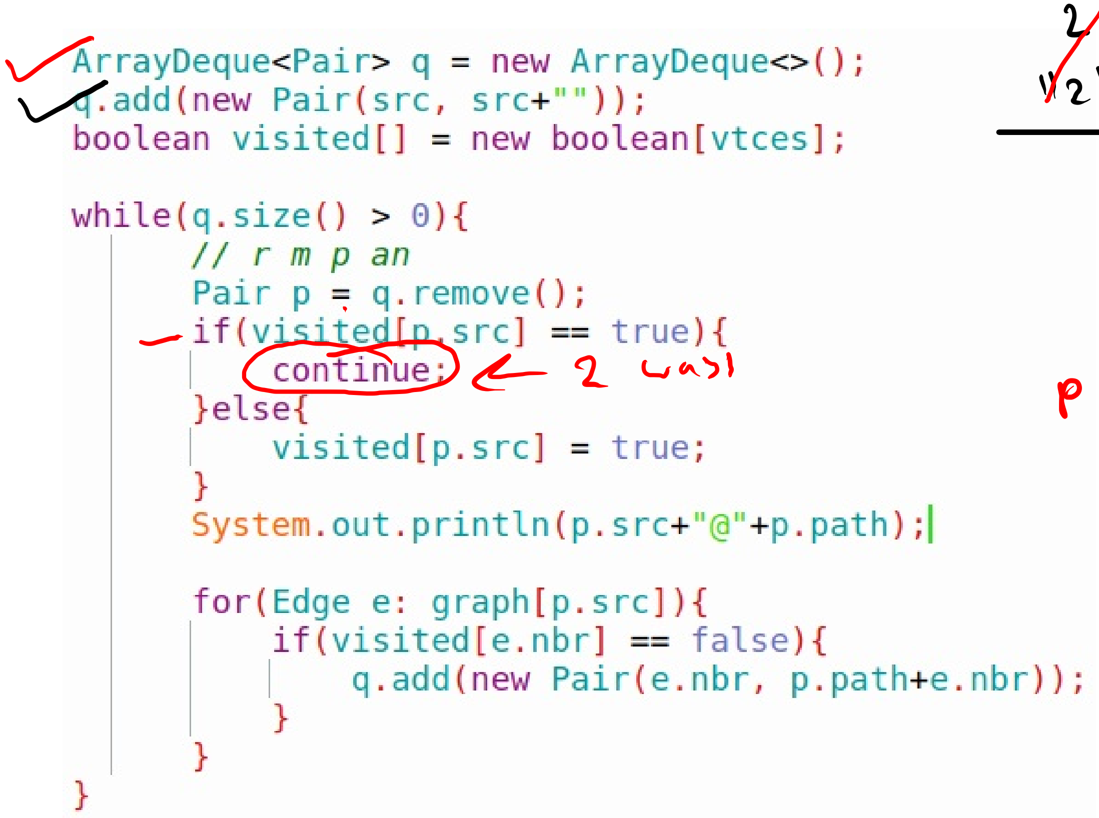
src -> 2





```
static class Pair{
    int src;
    String path;
    Pair(int s, String p){
        src = s;
        path = p;
    }
}
```

~~2~~	~~1~~	~~3~~	~~0~~	~~0~~	~~4~~	~~5~~	~~6~~	~~6~~
~~"2"~~	~~"21"~~	~~"23"~~	~~"210"~~	~~"230"~~	~~"234"~~	~~"2345"~~	~~"2346"~~	~~"23456"~~

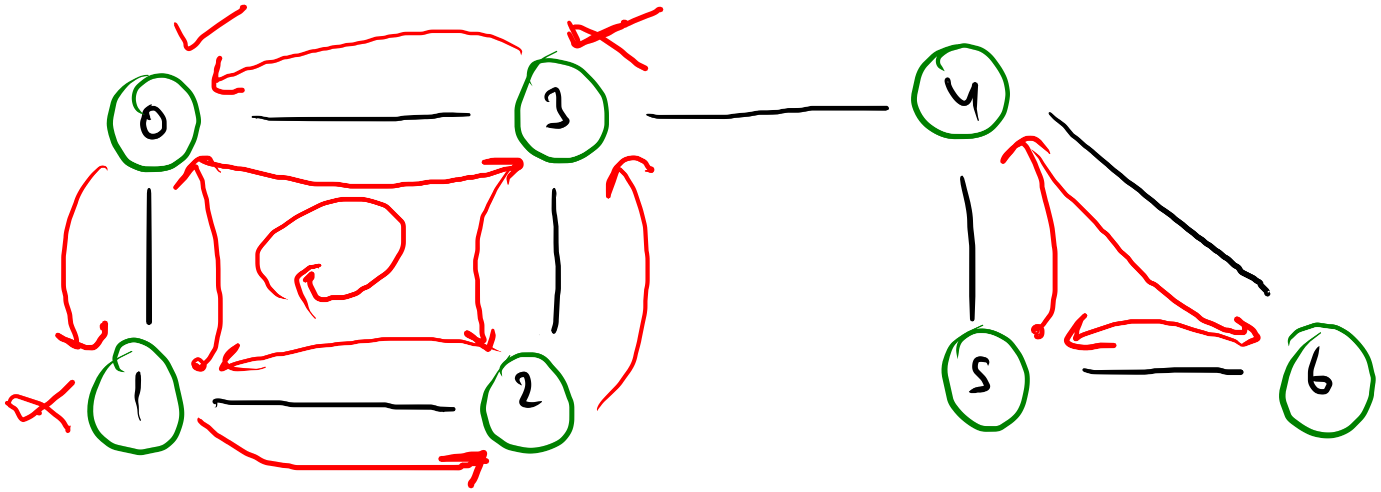


```
ArrayDeque<Pair> q = new ArrayDeque<>();
q.add(new Pair(src, src+""));
boolean visited[] = new boolean[vtces];

while(q.size() > 0){
    // r m p an
    Pair p = q.remove();
    if(visited[p.src] == true){
        continue;
    }else{
        visited[p.src] = true;
    }
    System.out.println(p.src+"@"+p.path);

    for(Edge e: graph[p.src]){
        if(visited[e.nbr] == false){
            q.add(new Pair(e.nbr, p.path+e.nbr));
        }
    }
}
```

← 2 wasl

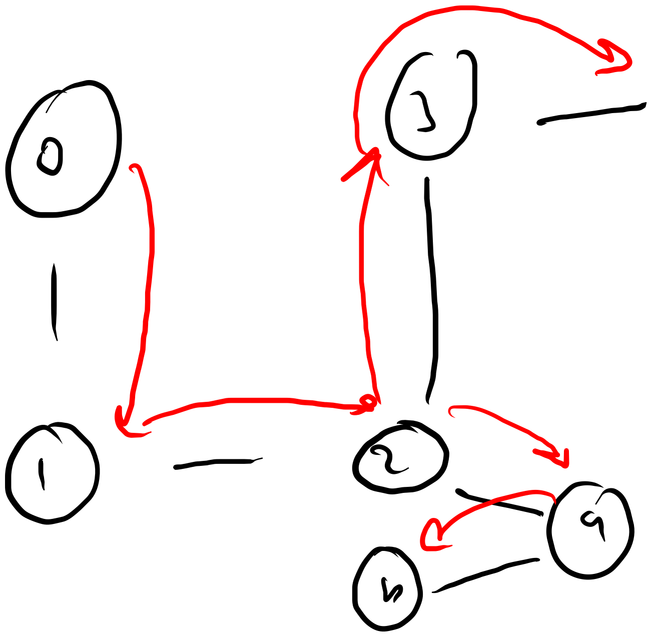
p [0 "210"]

2 @ 2
1 @ 21
3 @ 23
0 @ 210
4 @ 234
5 @ 2345
6 @ 2346

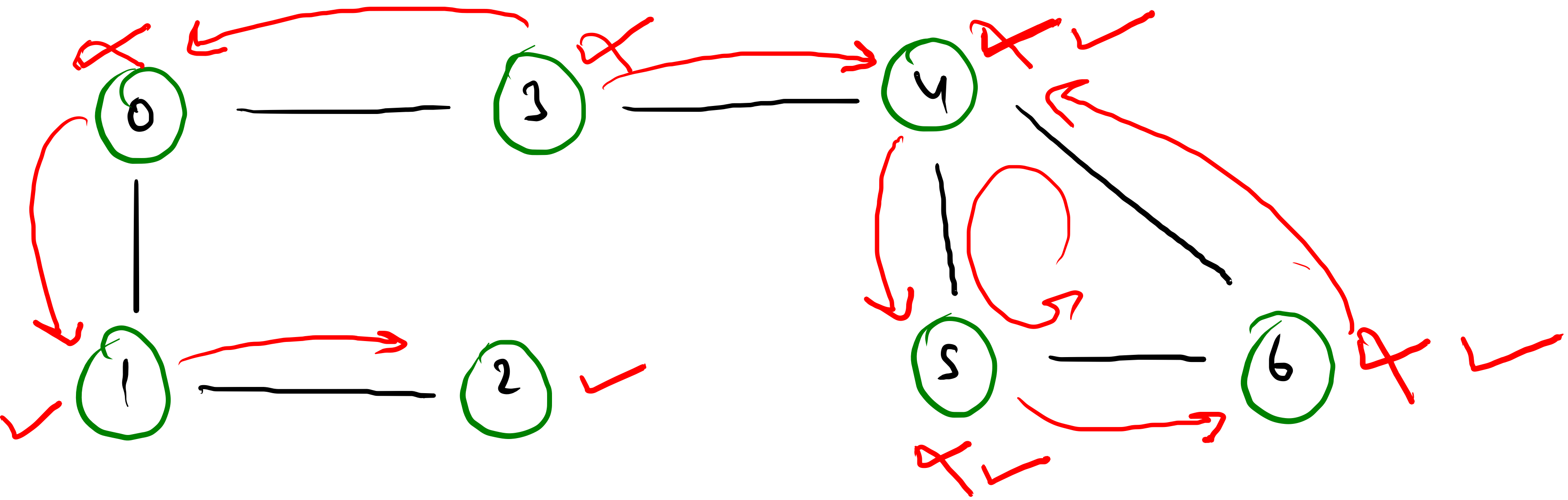


cycle

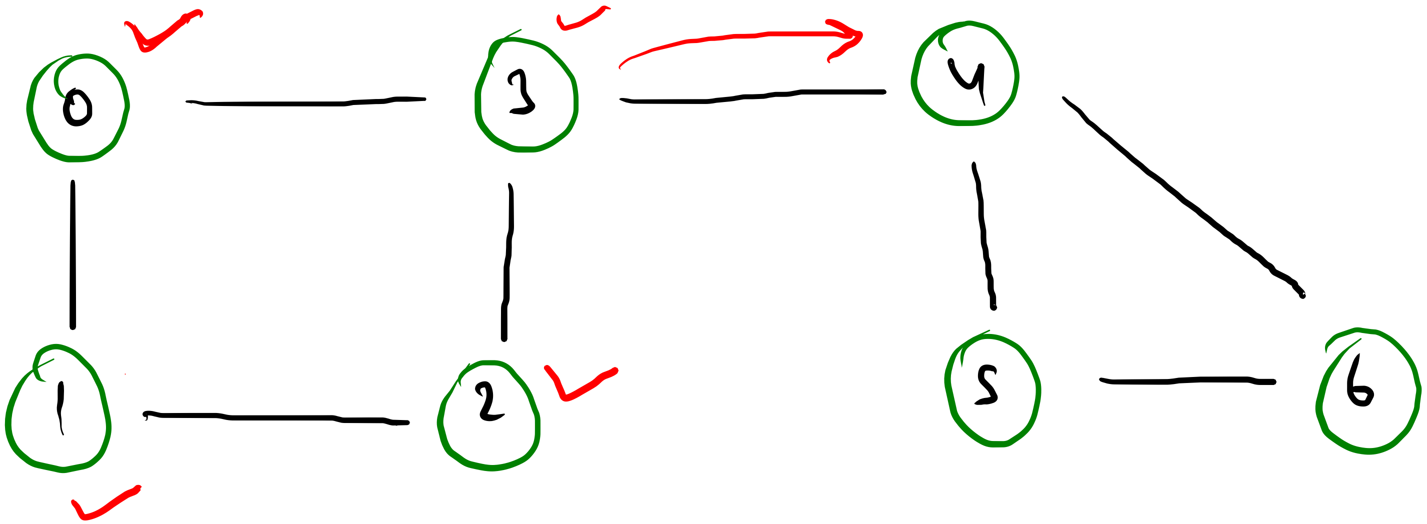
cycle

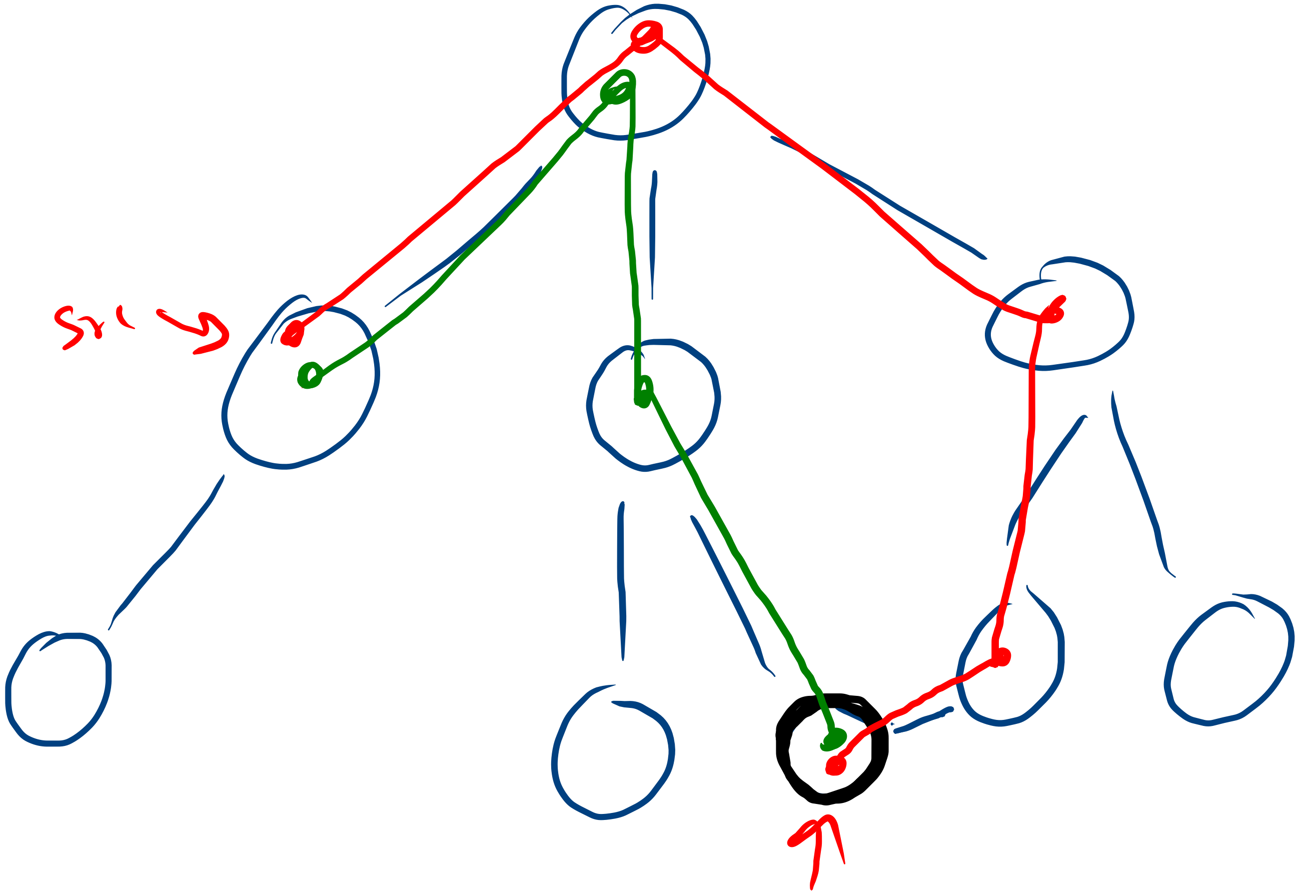


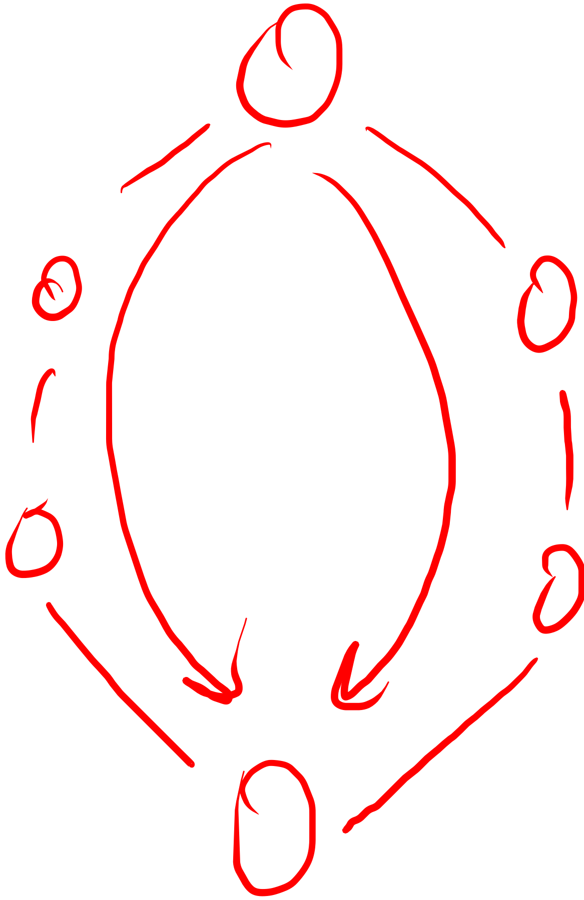
no problem

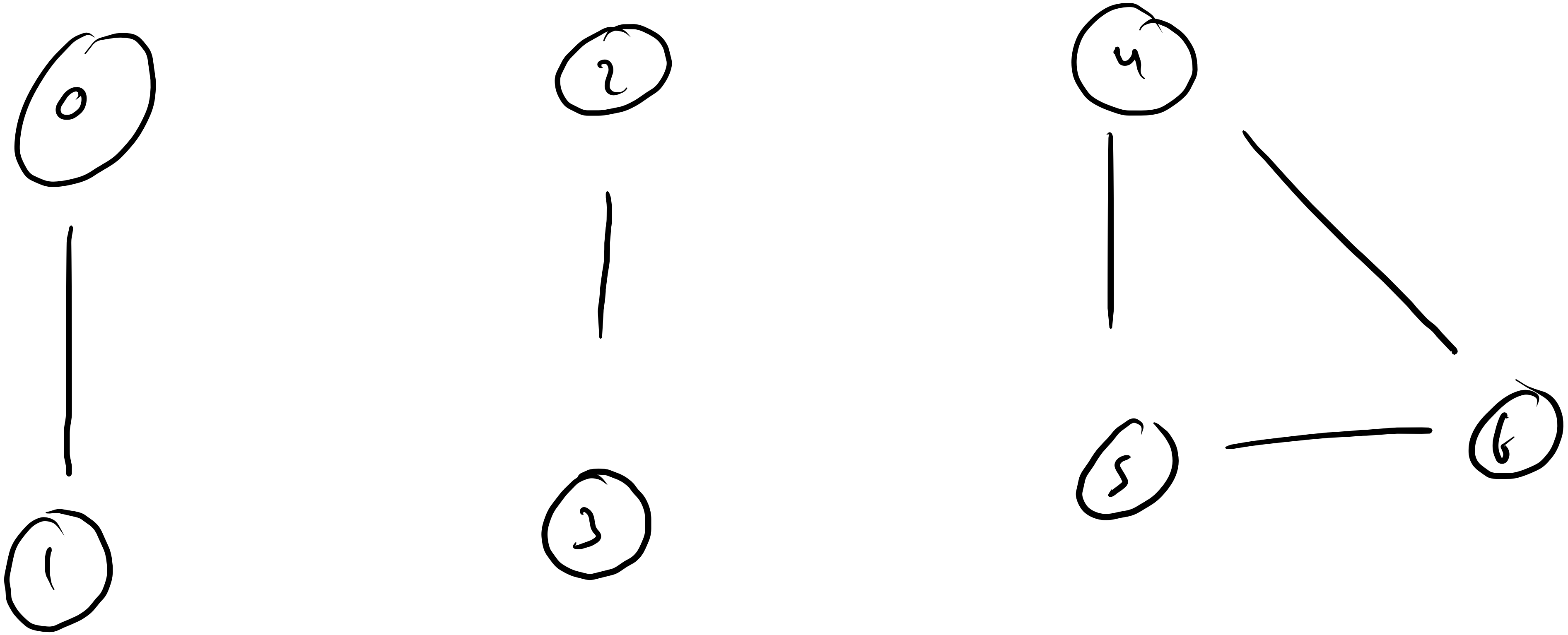


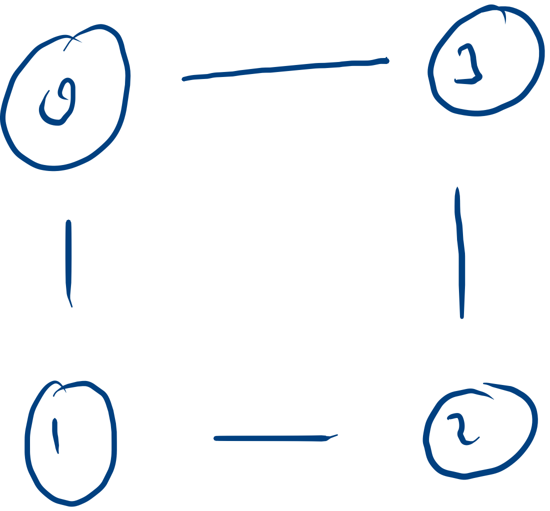
odd length cycle

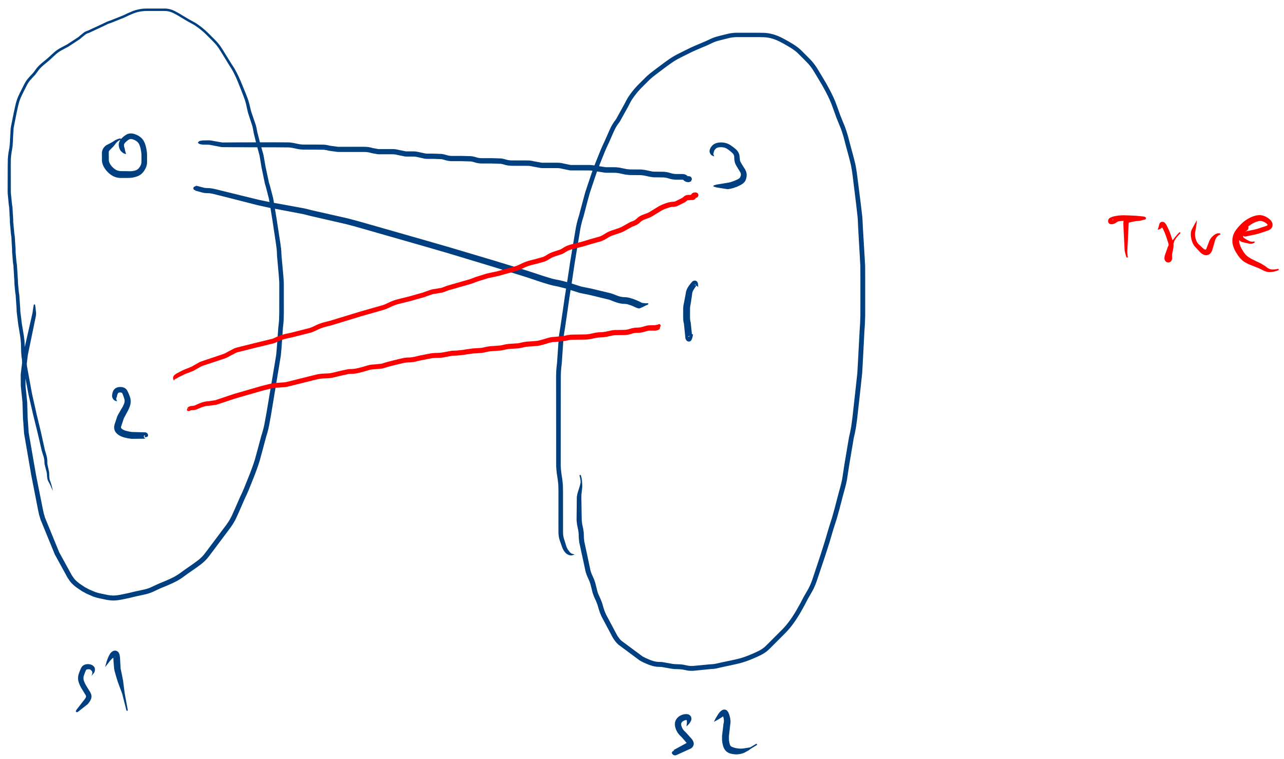


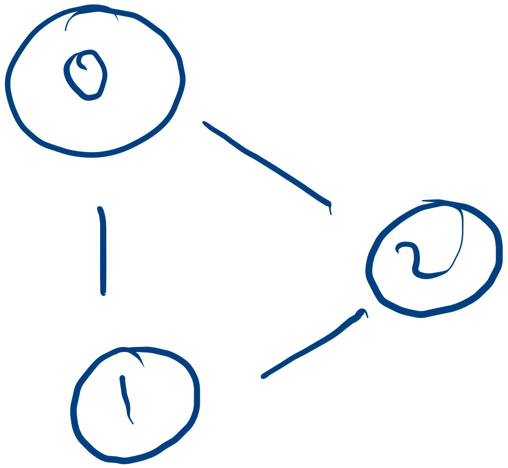

src

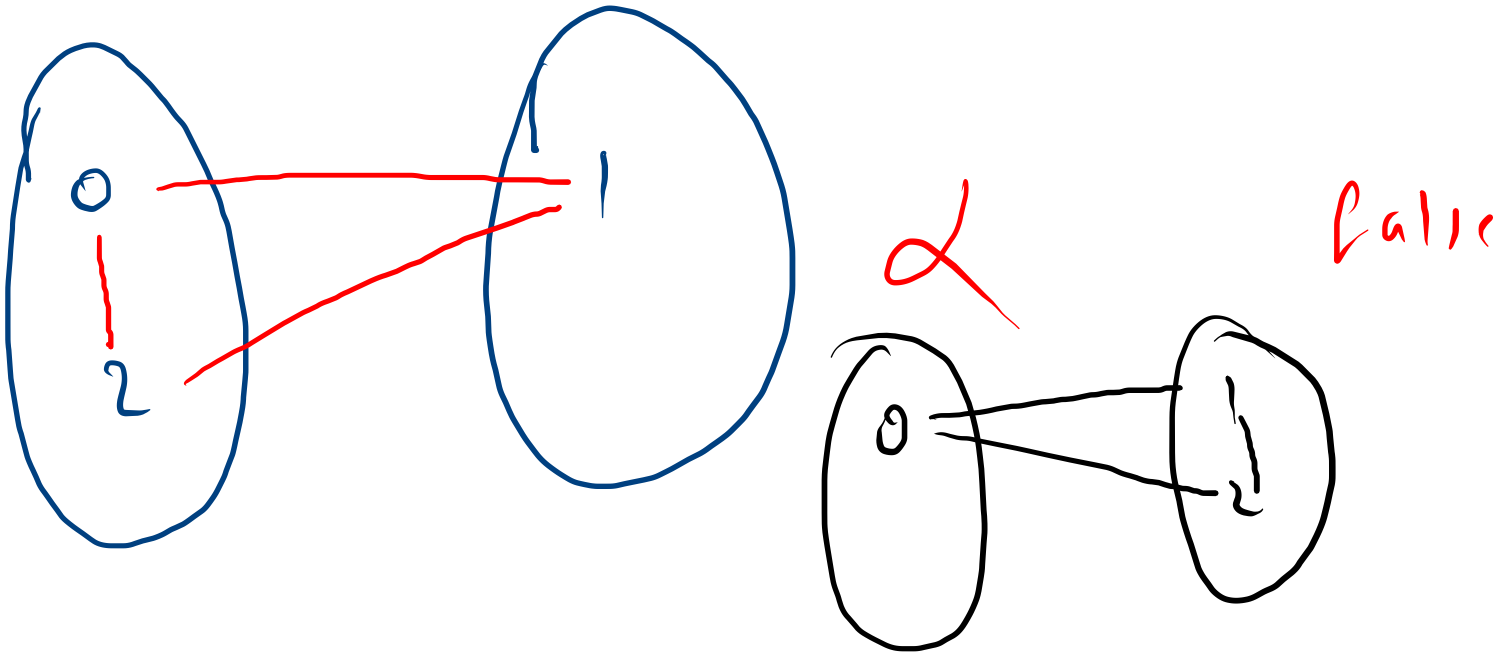



0
1
2
3
4
5
6


0
3
1
2


0
2
3
1
s1
s2
True

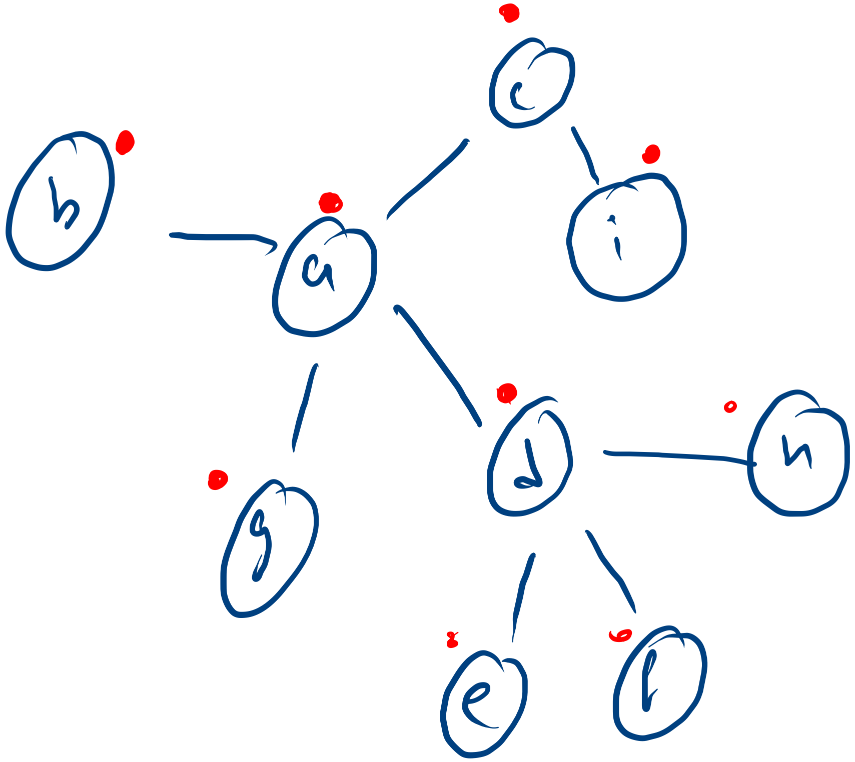

0
2
1

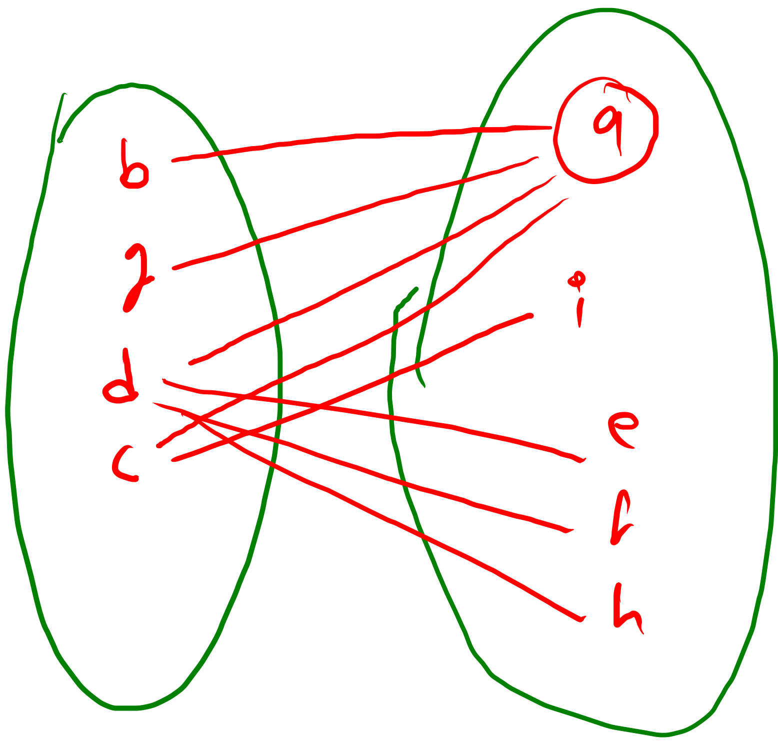

0
2
1
α
0
1
2
Fals

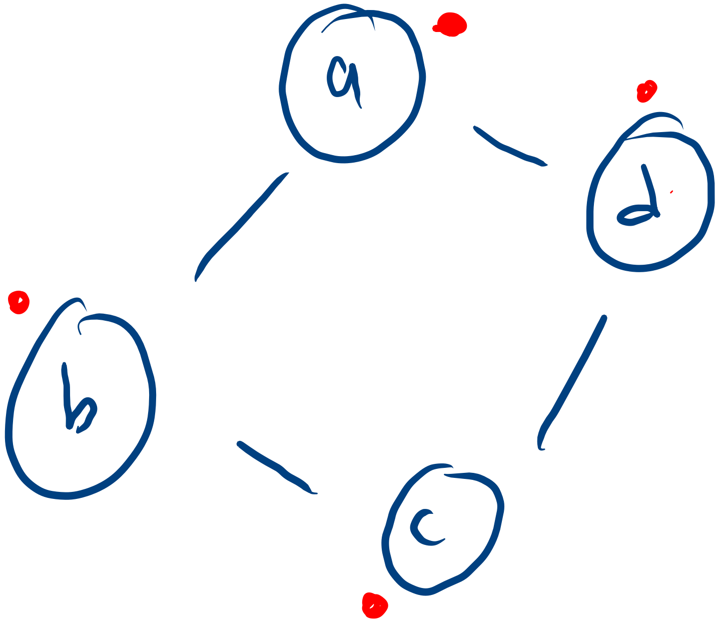
cyle → odd ✗

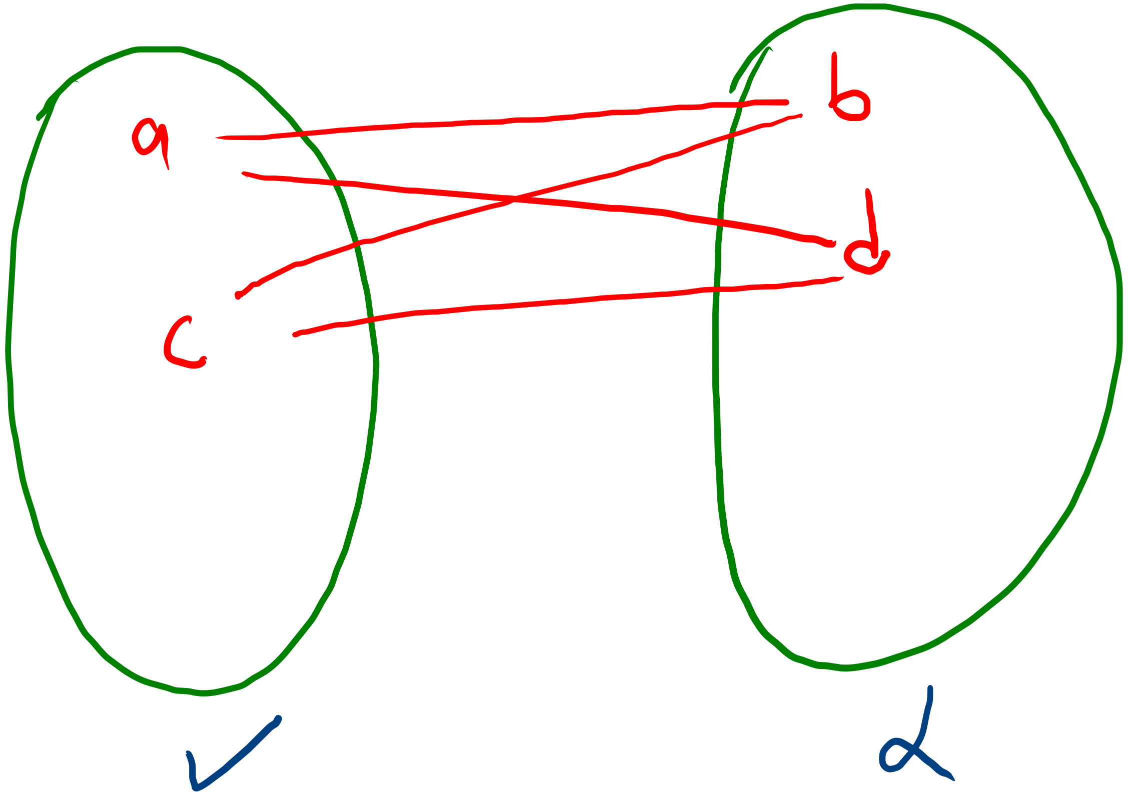
→ even ✓

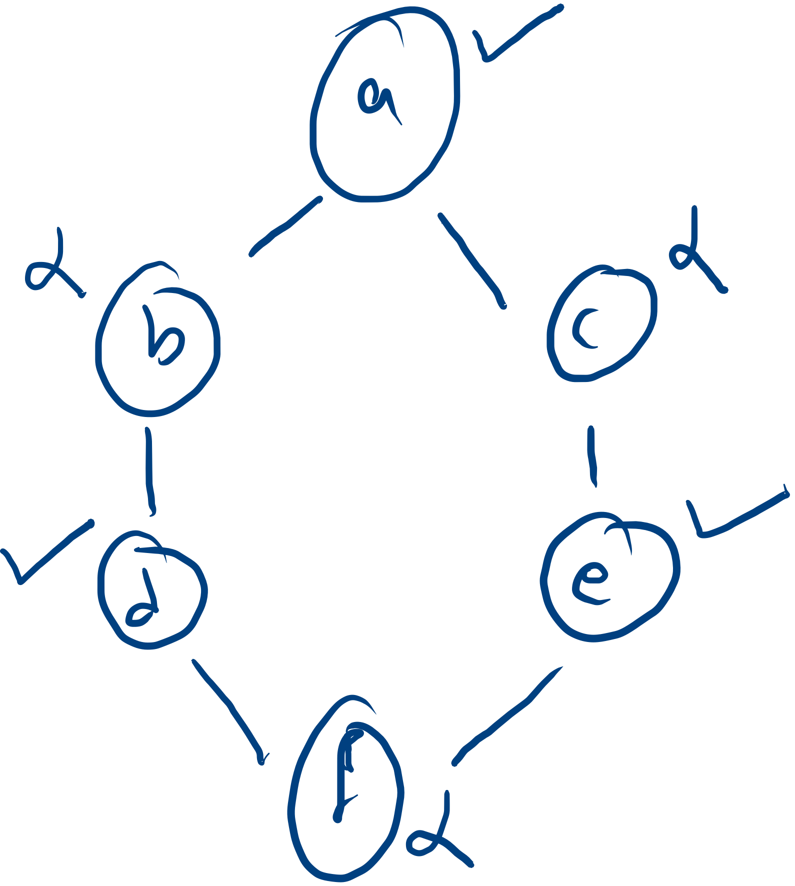
nocycle → ✓

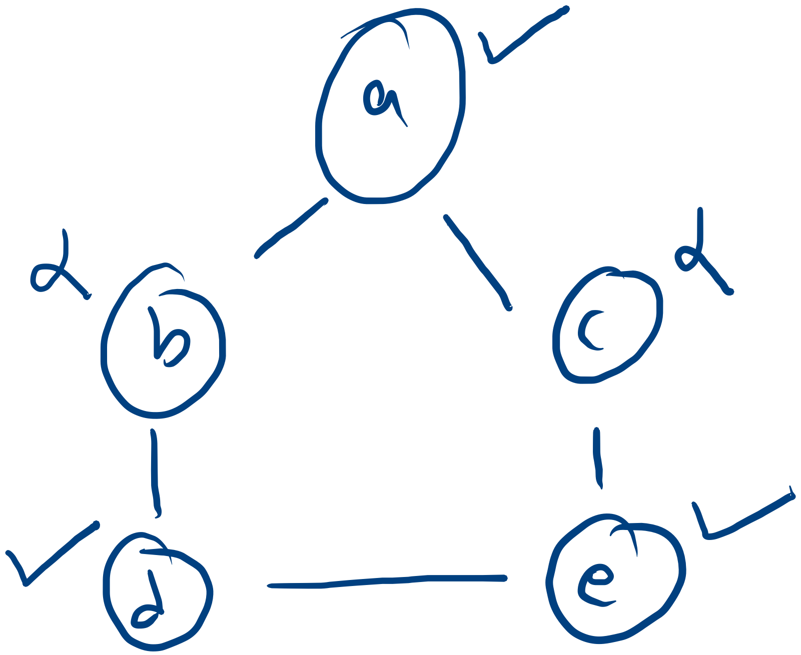


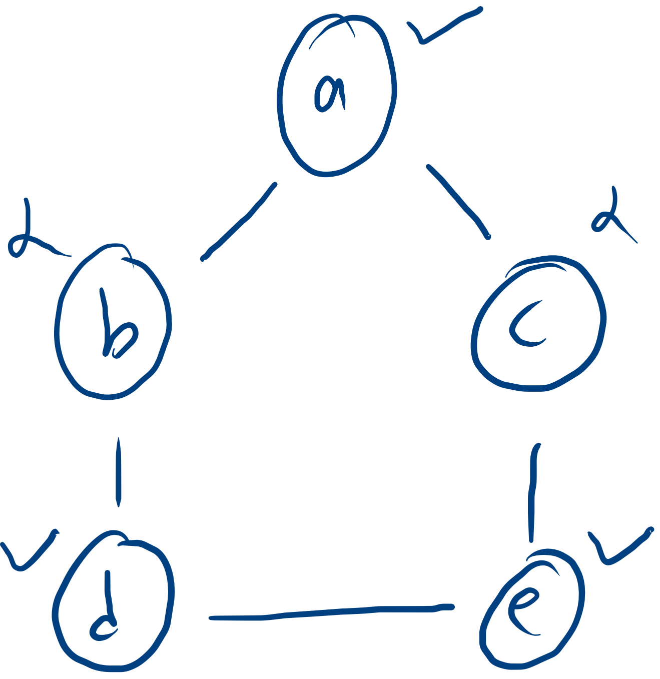


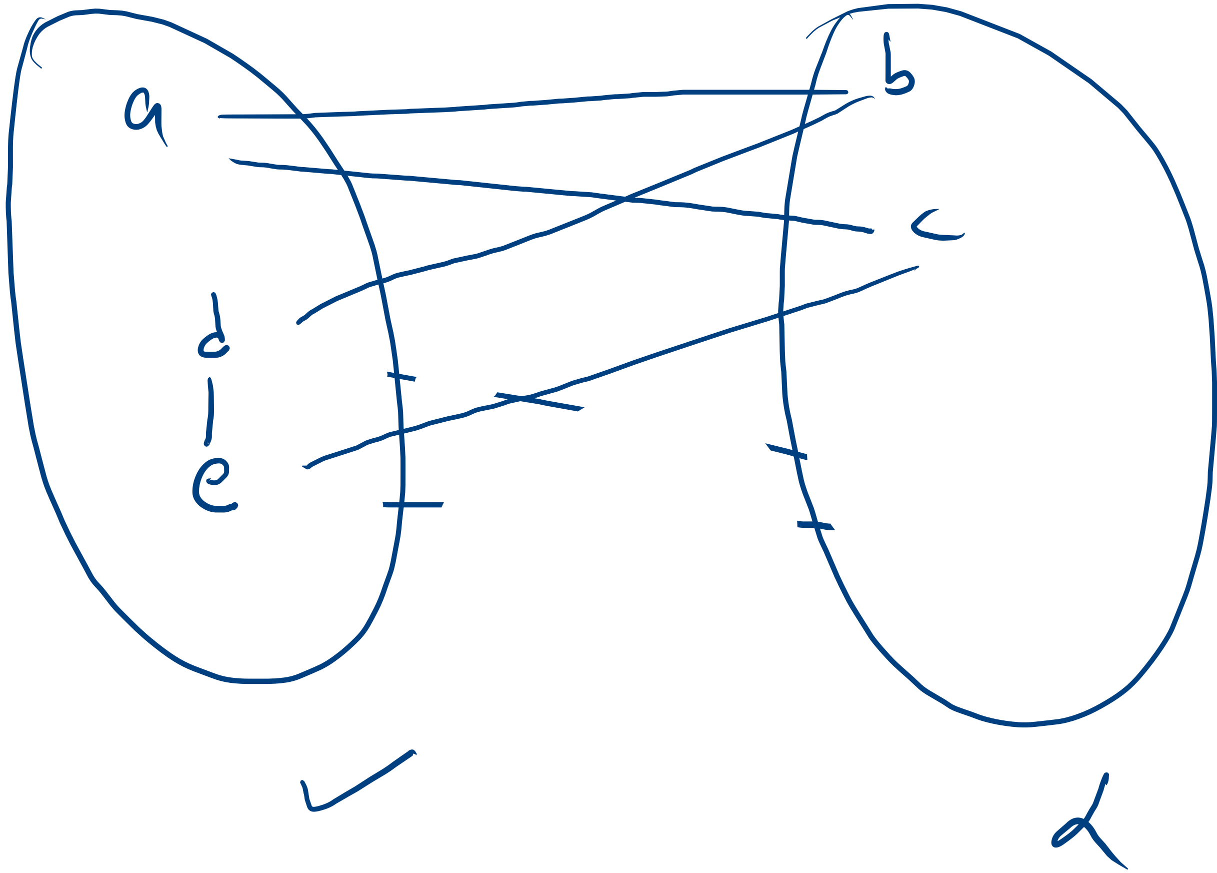

a
d
b
c

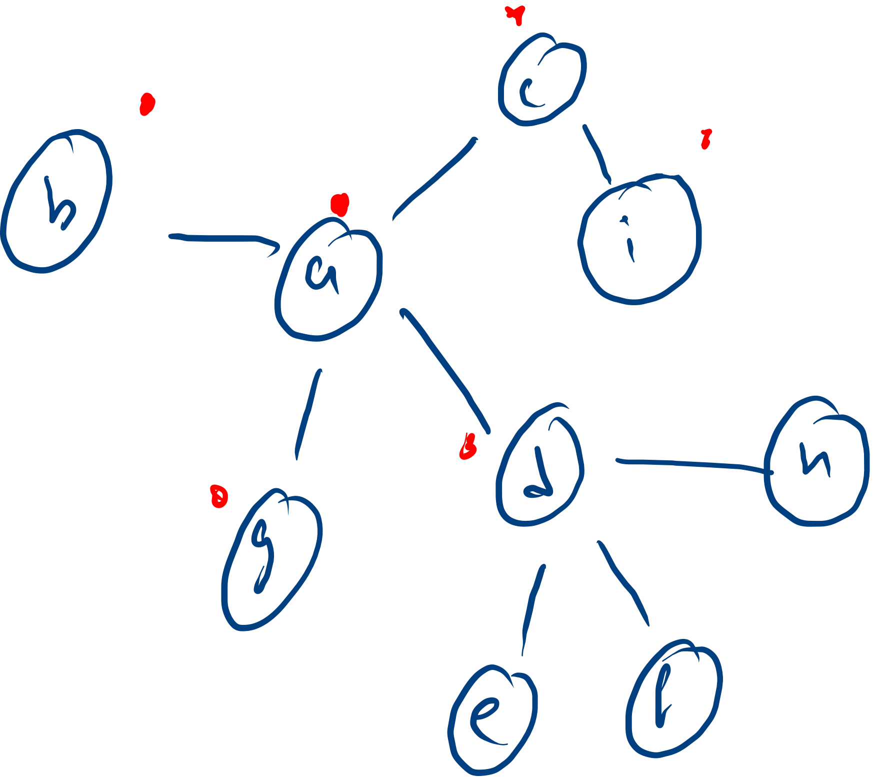

a
c
b
d
✓
α

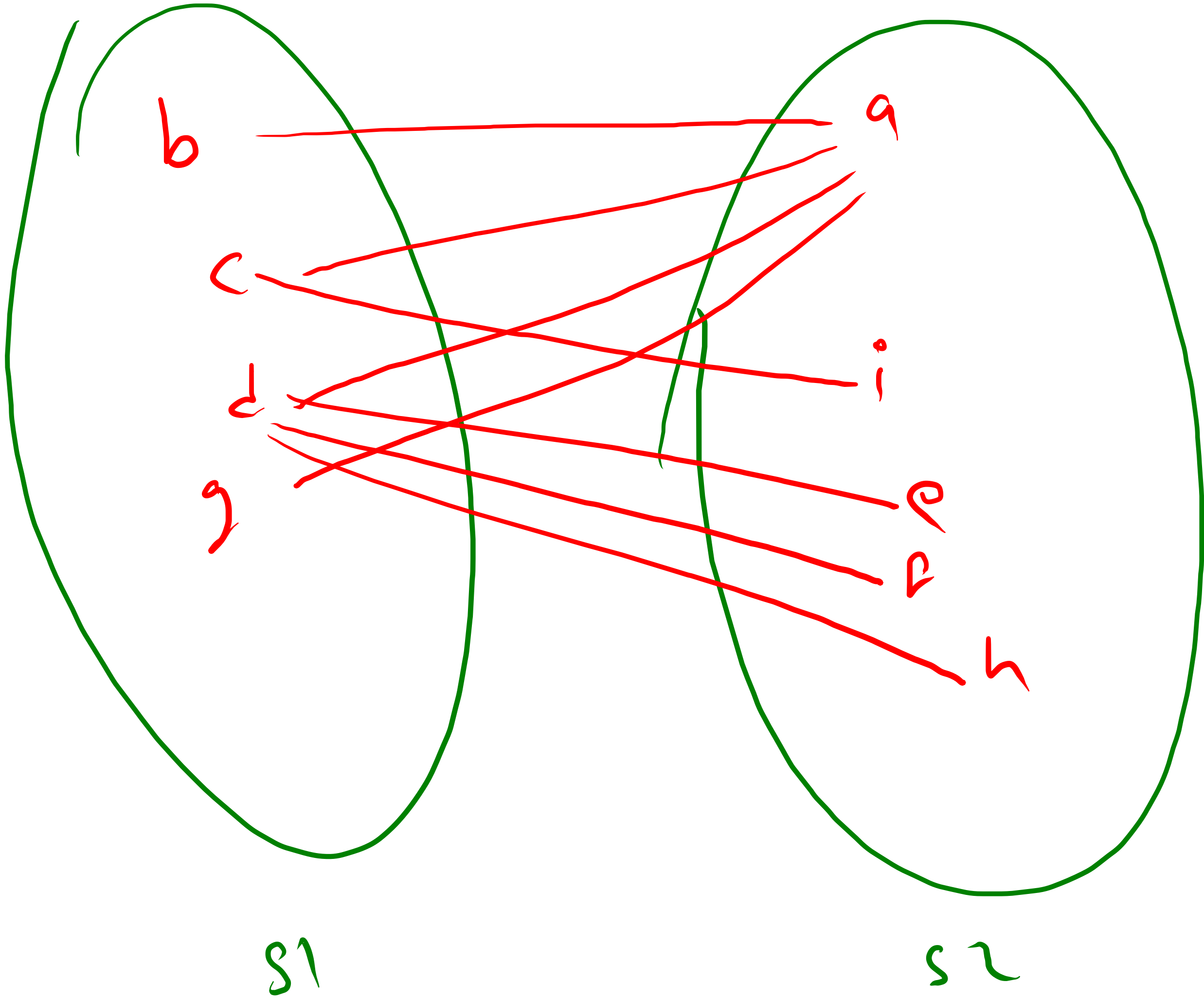

a ✓
α b
c α
✓ d
e ✓
f α

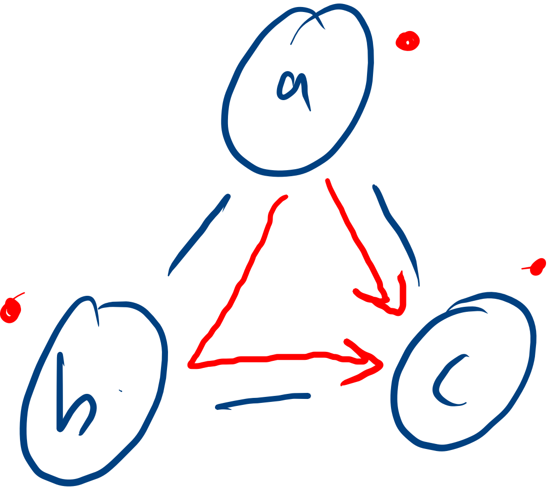

a ✓
α b
c α
✓ d
e ✓

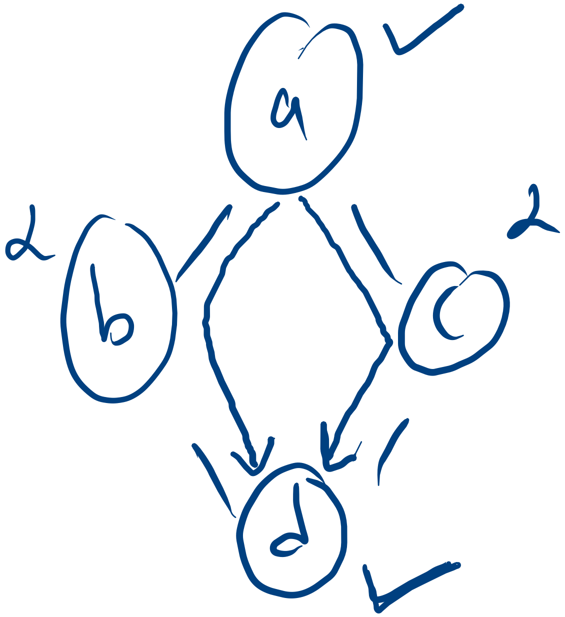

a
b
c
d
e

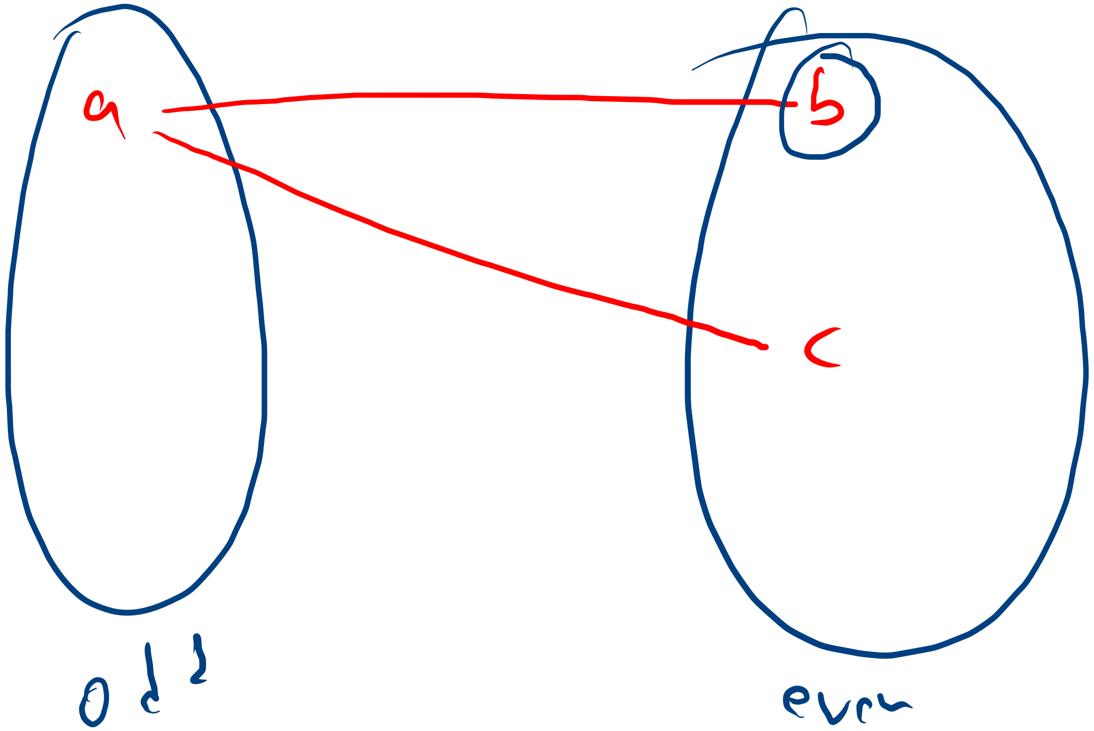

a
d
e
b
c


b
a
c
i
g
d
h
e
f

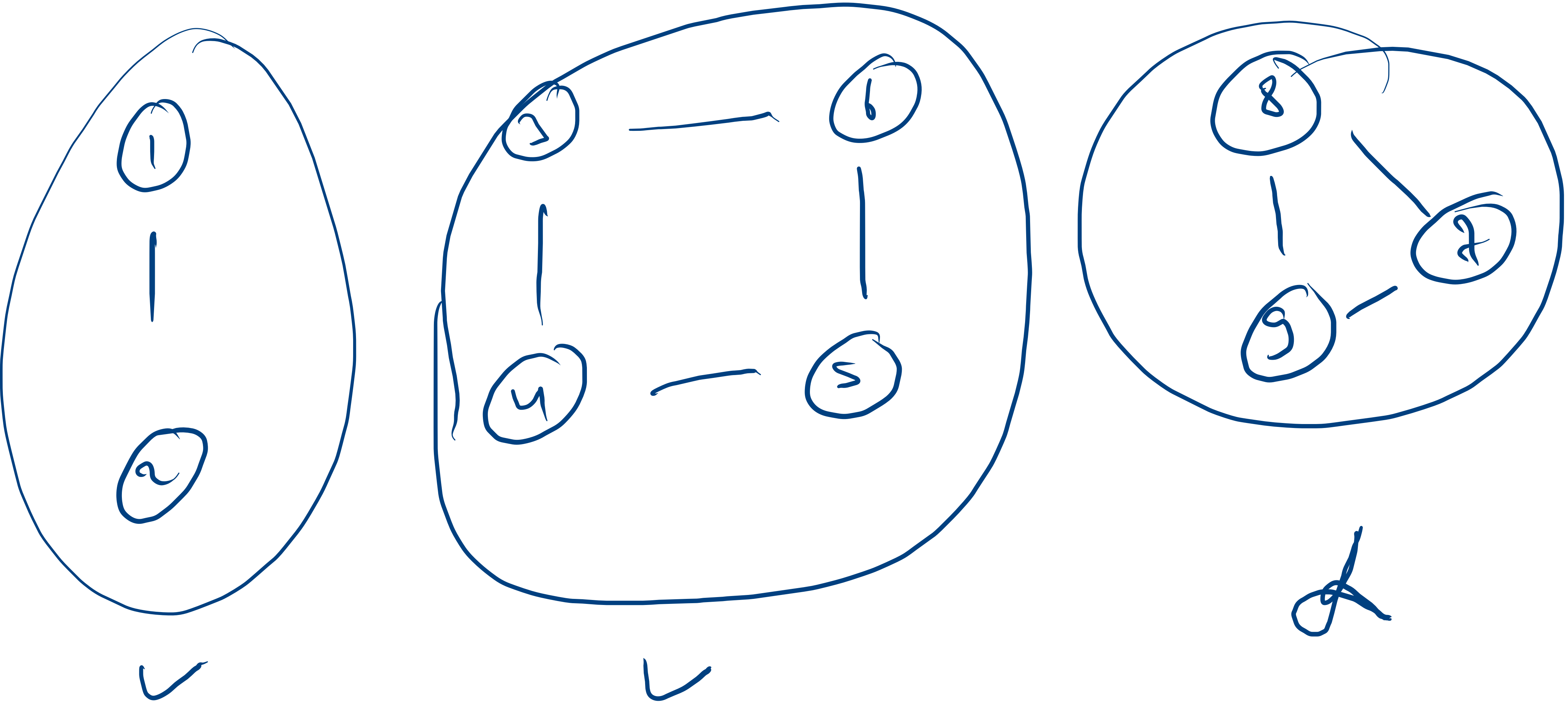

b
c
d
g
a
i
e
f
h
S1
S2

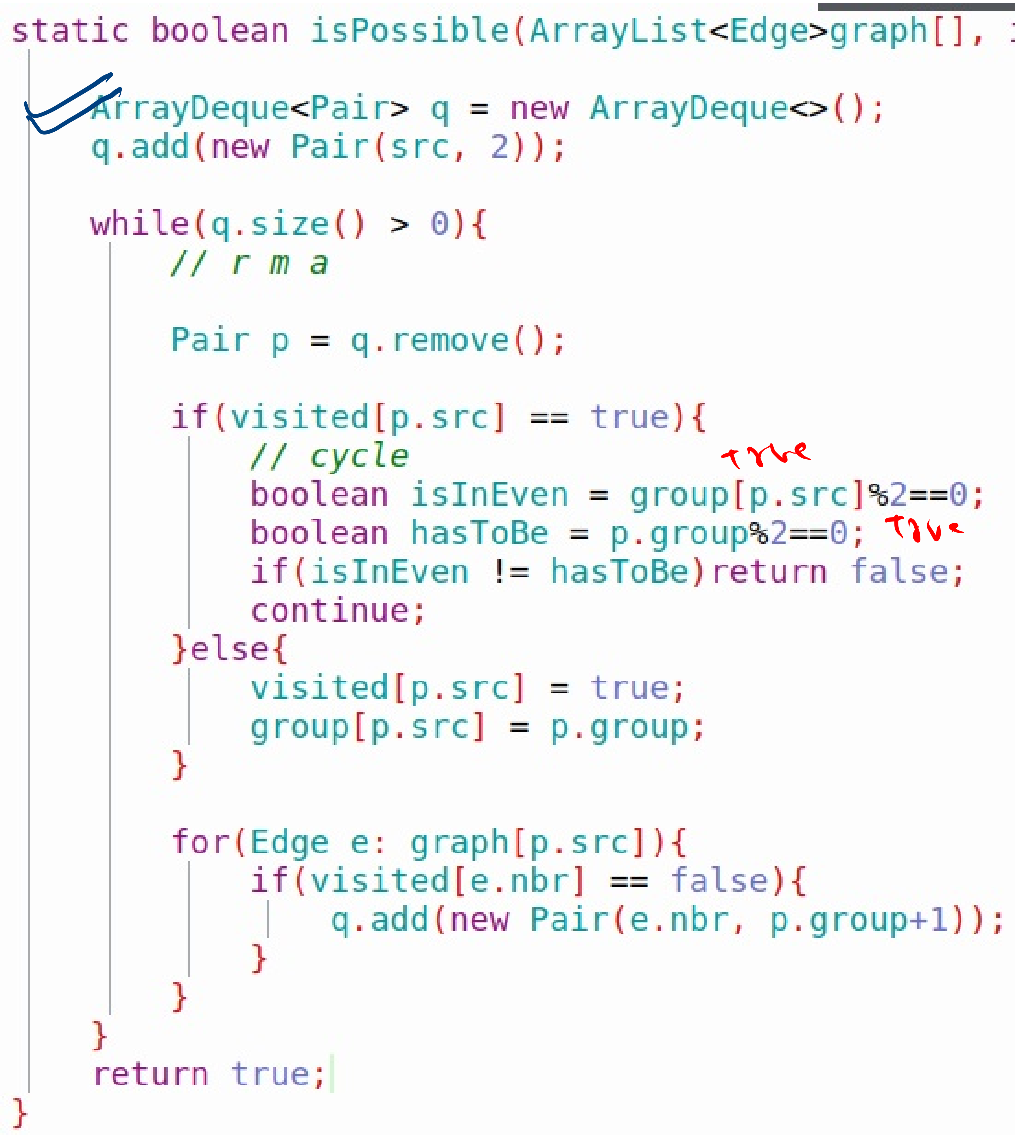






~~a~~	~~b~~	~~c~~	c	
~~1~~	~~2~~	~~2~~	1	

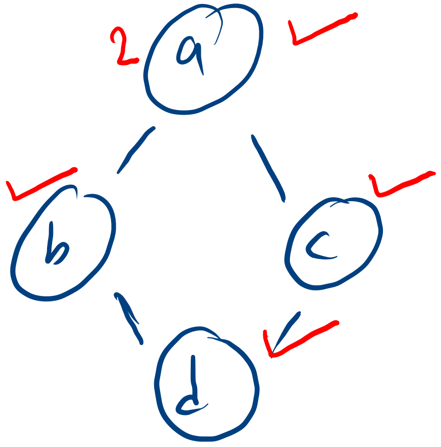

1
2
3
6
4
5
8
7
9
✓
✓

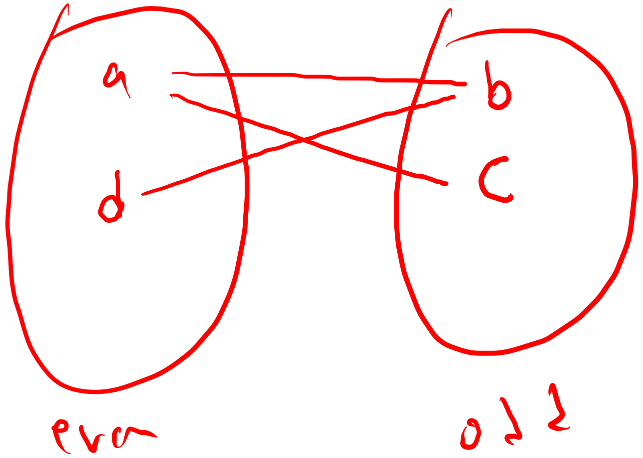


```
static boolean isPossible(ArrayList<Edge>graph[], 

    ArrayDeque<Pair> q = new ArrayDeque<>();
    q.add(new Pair(src, 2));

    while(q.size() > 0){
        // r m a

        Pair p = q.remove();

        if(visited[p.src] == true){
            // cycle
            boolean isInEven = group[p.src]%2==0;
            boolean hasToBe = p.group%2==0;
            if(isInEven != hasToBe)return false;
            continue;
        }else{
            visited[p.src] = true;
            group[p.src] = p.group;
        }

        for(Edge e: graph[p.src]){
            if(visited[e.nbr] == false){
                q.add(new Pair(e.nbr, p.group+1));
            }
        }
    }
    return true;
}
```

true

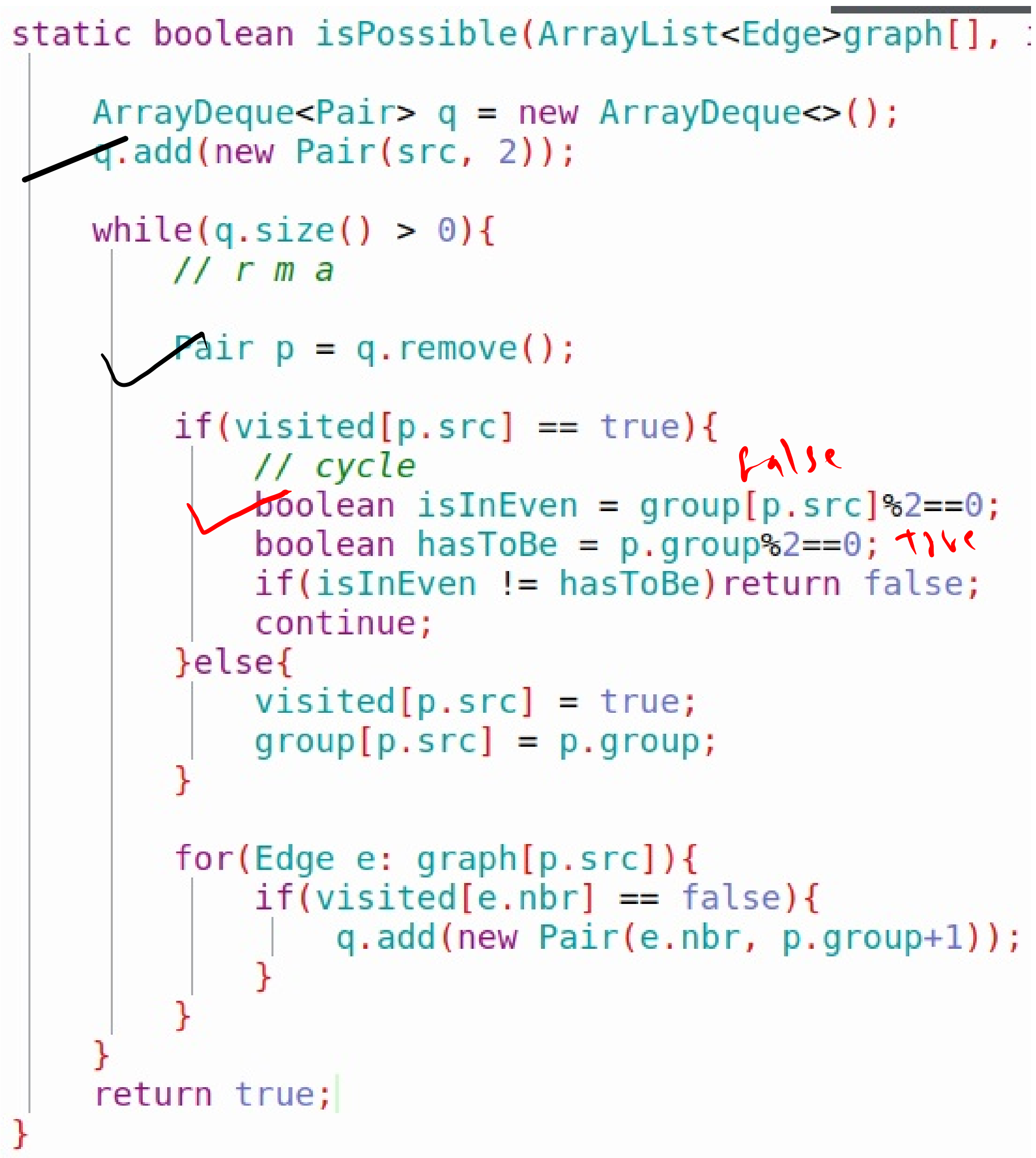
true





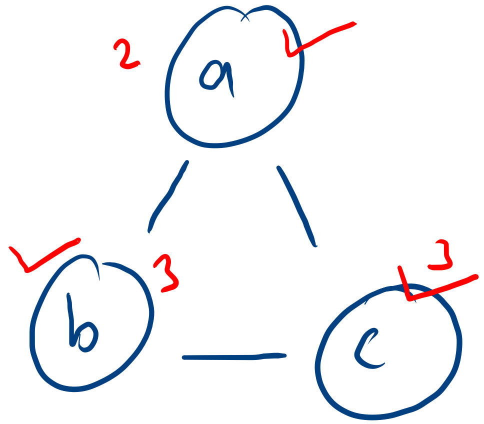
vertex	~~a~~	~~b~~	~~c~~	~~d~~	~~d~~
group	~~2~~	~~3~~	~~3~~	~~4~~	~~4~~

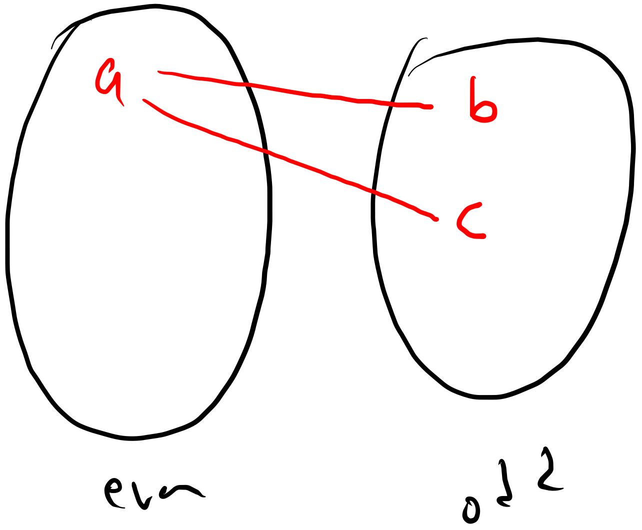
p



```
static boolean isPossible(ArrayList<Edge>graph[], 

    ArrayDeque<Pair> q = new ArrayDeque<>();
    q.add(new Pair(src, 2));

    while(q.size() > 0){
        // r m a

        Pair p = q.remove();

        if(visited[p.src] == true){
            // cycle
            boolean isInEven = group[p.src]%2==0;
            boolean hasToBe = p.group%2==0;
            if(isInEven != hasToBe)return false;
            continue;
        }else{
            visited[p.src] = true;
            group[p.src] = p.group;
        }

        for(Edge e: graph[p.src]){
            if(visited[e.nbr] == false){
                q.add(new Pair(e.nbr, p.group+1));
            }
        }
    }
    return true;
}
```

false
true


a
2
b
3
c
3


a
b
c
even
odd