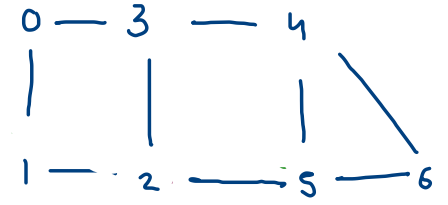
Hamiltonian Path And Cycle

Src = 2



Note -> A hamiltonian path is such which visits all vertices without visiting any twice. A hamiltonian path becomes a cycle if there is an edge between first and last vertex.

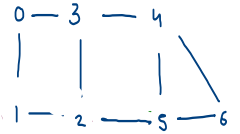
2 1 0 3 4 6 5 *

2 1 3 4 5 6 .

2 5 6 4 3 0 1 *

```
if(psf.length() ==  graph.length) {
    //psf -> hamiltonian path or hamiltonian cycle
    return;
}

vis[src] = true;

for(Edge edge : graph[src]) {
    int nbr = edge.nbr;

    if(vis[nbr] == false) {
        hamiltonian(graph,nbr,vis,psf + nbr);
    }
}
```

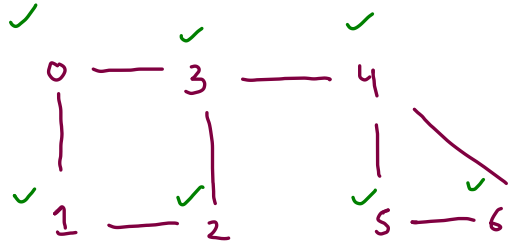
```java
if(psf.length() ==  graph.length) {
    //psf -> hamiltonian path or hamiltonian cycle
    boolean isHC = false; //is hamiltonian cycle

    for(Edge edge : graph[osrc]) {
        int nbr = edge.nbr;

        if(nbr == src) {
            isHC = true;
            break;
        }
    }

    System.out.print(psf);

    if(isHC == true) {
        System.out.println("*");
    }
    else {
        System.out.println(".");
    }
    return;
}

vis[src] = true;

for(Edge edge : graph[src]) {
    int nbr = edge.nbr;

    if(vis[nbr] == false) {
        hamiltonian(graph,nbr,osrc,vis,psf + nbr);
    }
}

vis[src] = false;
```



Breadth First Traversal

remove
mark
work
add nbr (add unvisited nbr)

BFS

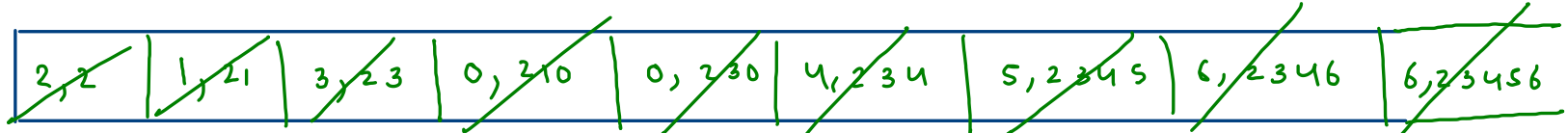


2@2
1@21
3@23
0@210
4@234
5@2345
6@2346

2@2
1@21
3@23
0@210

4@234
5@2345
6@2346

Src = 2



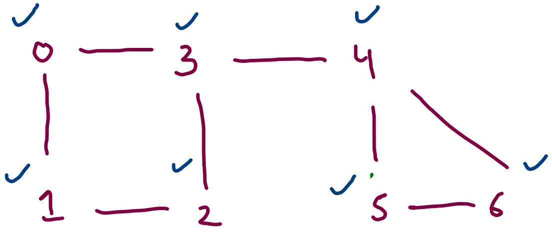
shortest path
↓
in terms of edges

0
1
continue
2
3
continue
4

min moves are ensured by BFS

src = 2

BFS linewise



lev →

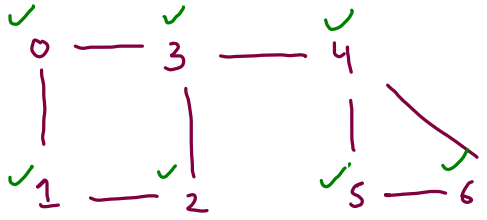
count times
- remove
- mark*
- work
- add nbr*

syso line, lev++;

c = 1

l = ~~0~~ ~~1~~ ~~2~~ ~~3~~ 4

0 → 2
1 → 1, 3
2 → 0, 4
3 → 5, 6
4 → ×

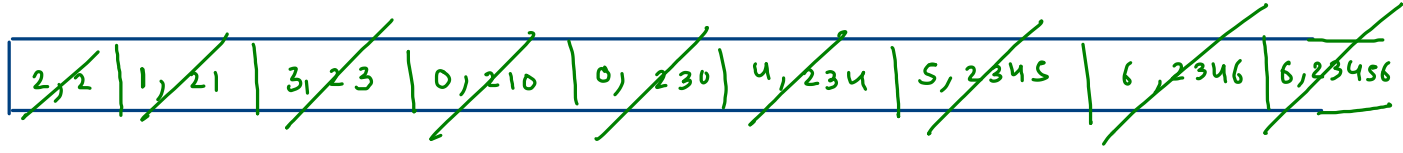


task : single src all dest shortest path (in terms of edges)

```
boolean[]vis = new boolean[graph.length];
ArrayDeque<Pair>q = new ArrayDeque<>();

q.add(new Pair(src,"" + src));

while(q.size() > 0) {
    //remove
    Pair rem = q.remove();

    //mark*
    if(vis[rem.v] == true) {
        continue;
    }
    vis[rem.v] = true;

    //work
    System.out.println(rem.v + "@" + rem.psf);

    //add nbr*
    for(Edge edge : graph[rem.v]) {
        int nbr = edge.nbr;

        if(vis[nbr] == false) {
            q.add(new Pair(nbr,rem.psf + nbr));
        }
    }
}
```

src = 2

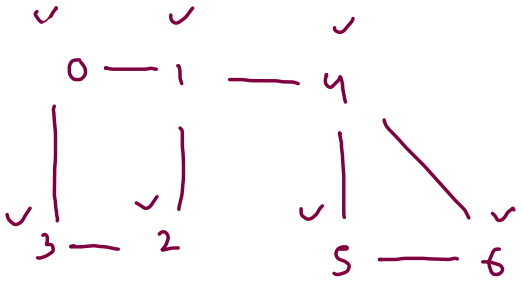
2 @ 2
1 @ 21
3 @ 23
0 @ 210
4 @ 234
5 @ 2345
6 @ 2346

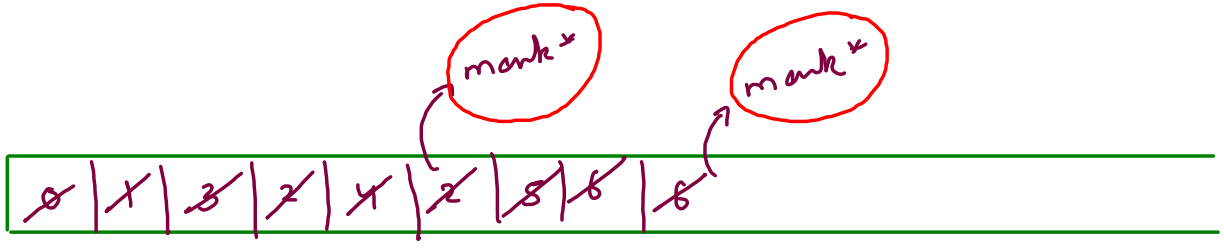


Is Graph Cyclic

if any of the comp is cyclic -> graph cyclic

for a graph to be acyclic, every comp is acyclic.



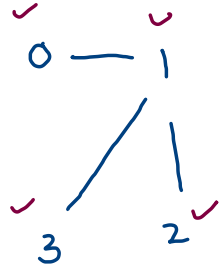


remove

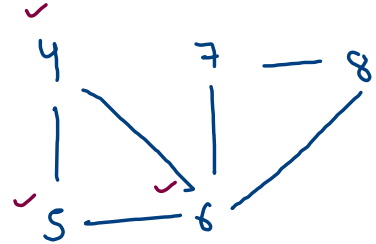
mark*

add nbr*

```java
public static boolean isGraphCyclic(ArrayList<Edge>[]graph) {
    boolean[]vis = new boolean[graph.length];

    for(int i=0; i < graph.length;i++) {
        if(vis[i] == false) {
            boolean sca = isCompCyclic(graph,i,vis);

            if(sca == true) {
                return true;
            }
        }
    }

    return false;
}
```

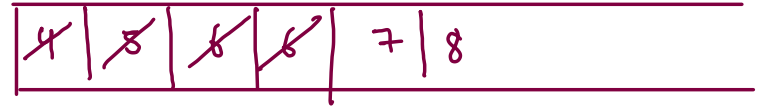


false



true

```java
public static boolean isCompCyclic(ArrayList<Edge>[]graph,int src,boolean[]vis) {
    ArrayDeque<Integer>q = new ArrayDeque<>();
    q.add(src);

    while(q.size() > 0) {
        //remove
        int rem = q.remove();

        //mark*
        if(vis[rem] == true) {
            return true;
        }
        vis[rem] = true;

        //add unvisited nbr
        for(Edge edge : graph[rem]) {
            int nbr = edge.nbr;
            if(vis[nbr] == false) {
                q.add(nbr);
            }
        }
    }

    return false;
}
```

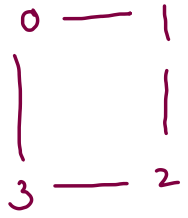


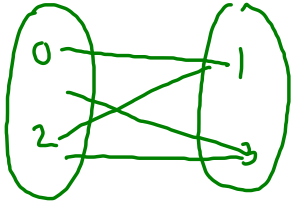
Is Graph Bipartite

mutually exclusive: $S1 \cap S2 = \phi$

exhaustive: $S1 \cup S2$ = all vertices

Note -> A graph is called bipartite if it is possible to split it's vertices in two sets of mutually exclusive and exhaustive vertices such that all edges are across sets.



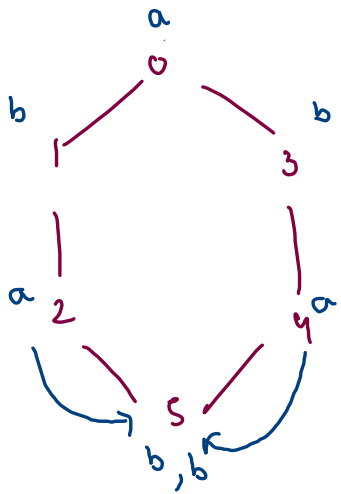


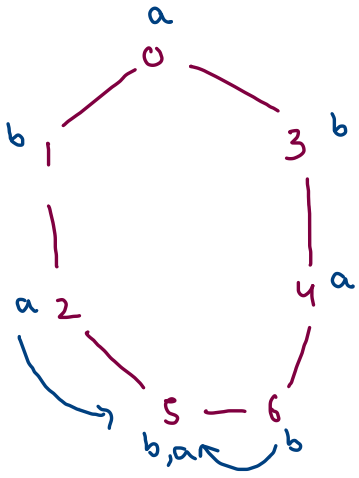
for a graph to be bipartite, every comp should be bipartite

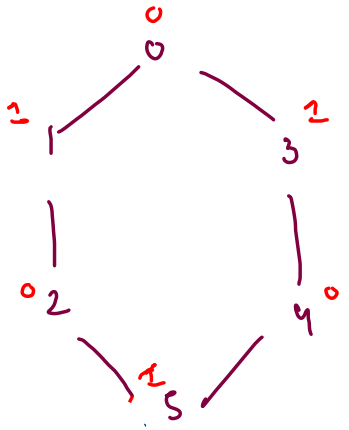
(i) if a comp is acyclic -> comp is bipartite

(ii) if a comp cyclic : a) even no. of vertices in cycle -> bipartite

b) odd no. of vertices in cycle -> non-bipartite.

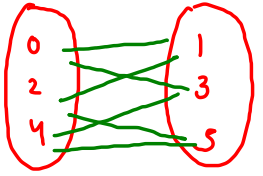


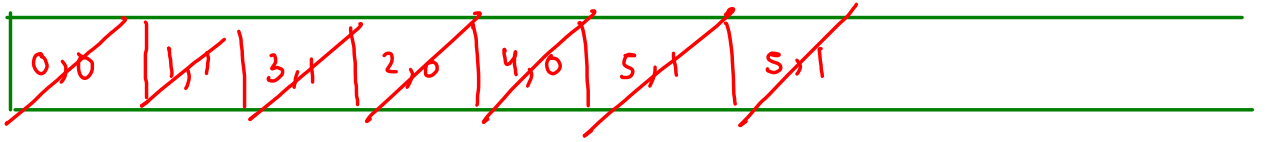


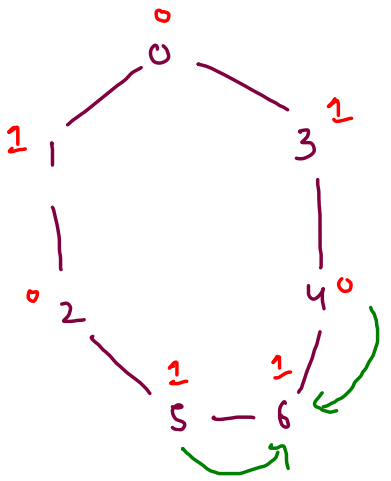


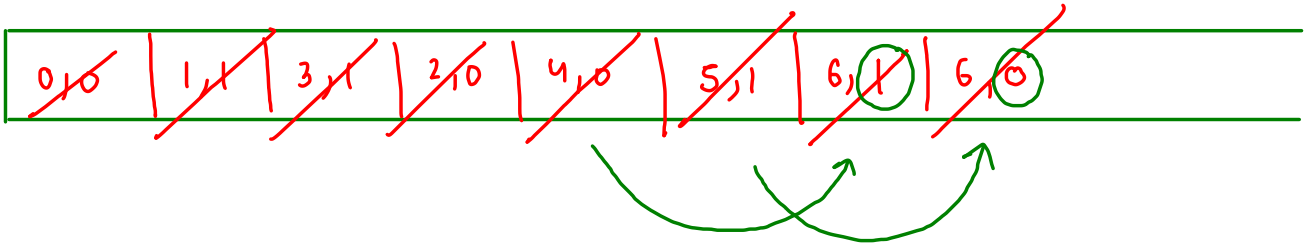
vis: last time set no.

Set 0, 1



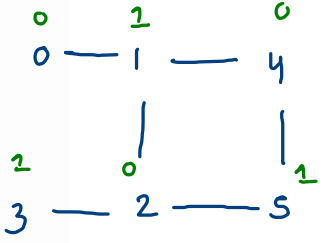



0
1
2
3
4
5
6

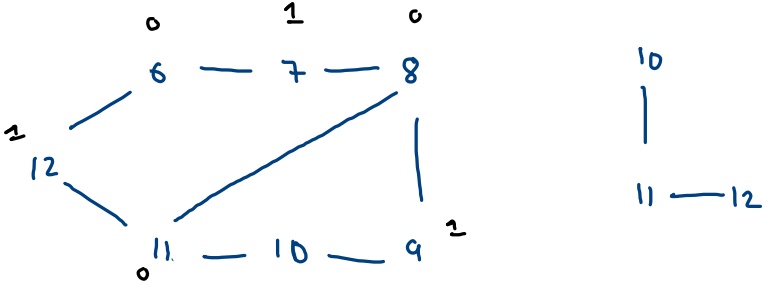

0,0
1,1
3,1
2,0
4,0
5,1
6,1
6,0

```java
public static boolean isCompBipartite(int src,ArrayList<Edge>[]graph,int[]vis) {
    ArrayDeque<Pair>q = new ArrayDeque<>();
    q.add(new Pair(src,0));

    while(q.size() > 0) {
        Pair rem = q.remove();

        if(vis[rem.v] != -1) {
            int osn = vis[rem.v]; //old set number
            int nsn = rem.sn; //new set number

            if(osn != nsn) {
                return false;
            }
            continue;
        }
        vis[rem.v] = rem.sn;

        for(Edge edge : graph[rem.v]) {
            int nbr = edge.nbr;

            if(vis[nbr] == -1) {
                int sn = (rem.sn == 0) ? 1 : 0;
                q.add(new Pair(nbr,sn));
            }
        }
    }

    return true;
}
```

```java
public static boolean isGraphBipartite(ArrayList<Edge>[]graph) {
    int[]vis = new int[graph.length];
    Arrays.fill(vis,-1);

    for(int i=0; i < graph.length;i++) {
        if(vis[i] == -1) {
            boolean sca = isCompBipartite(i,graph,vis);

            if(sca == false) {
                return false;
            }
        }
    }

    return true;
}
```



true



false