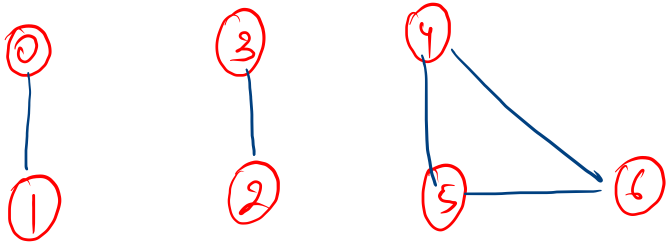
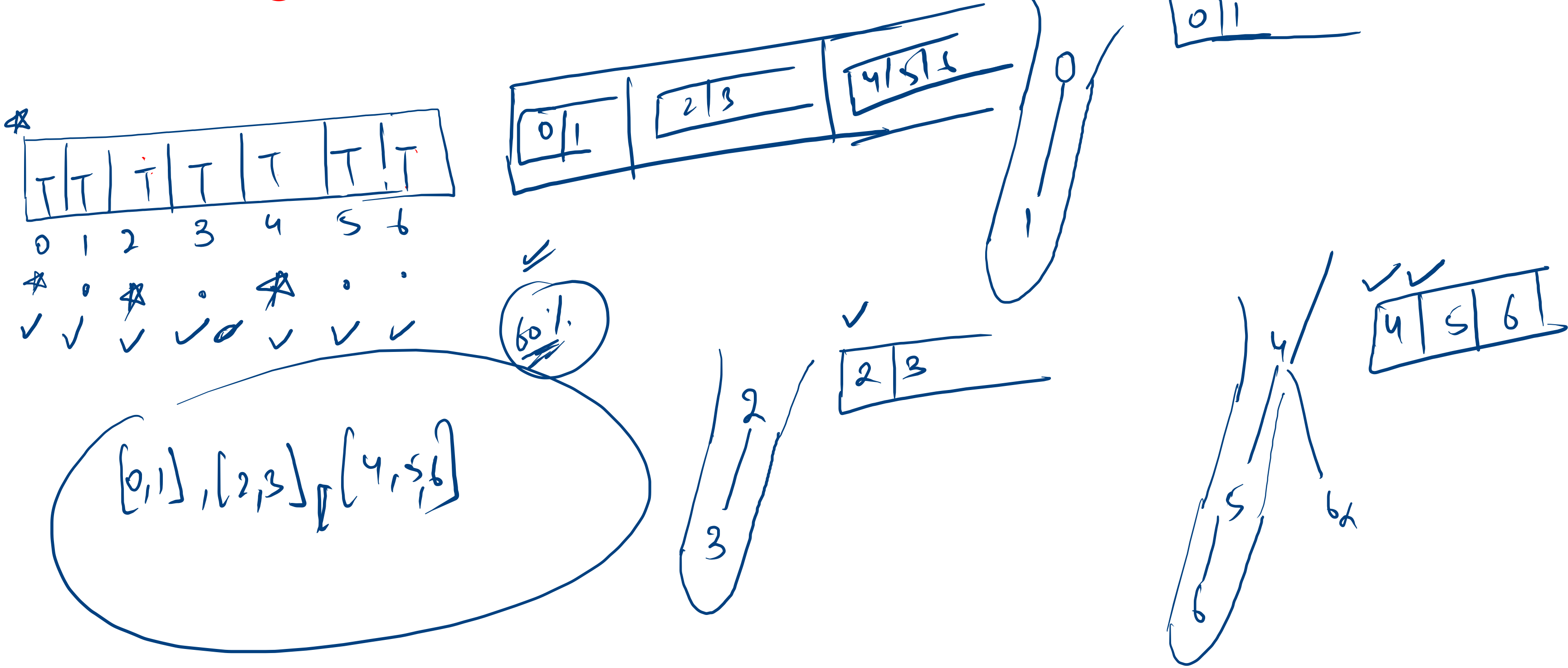
[[0,1], [2,3], [4,5,6]]

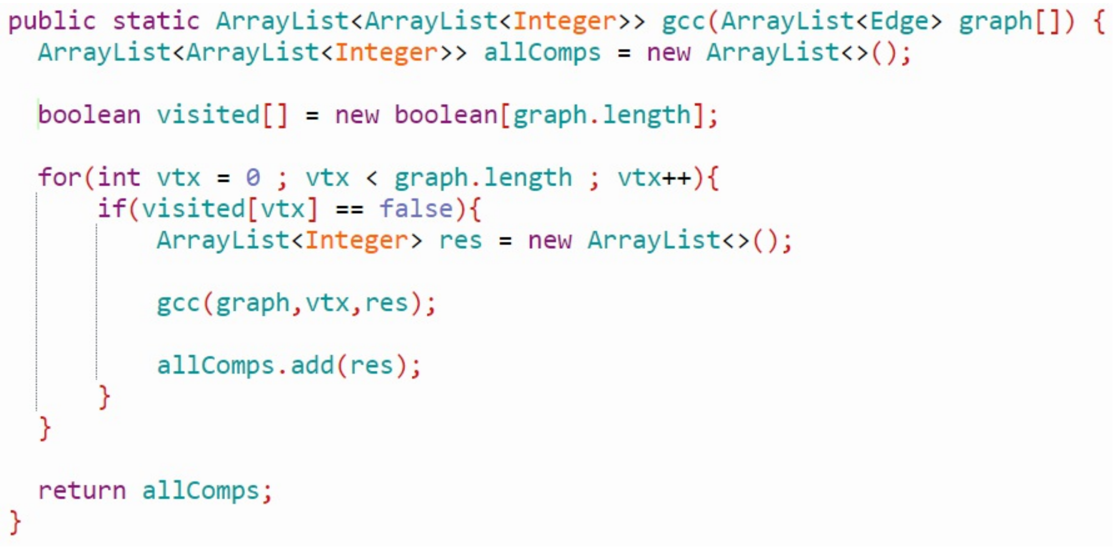




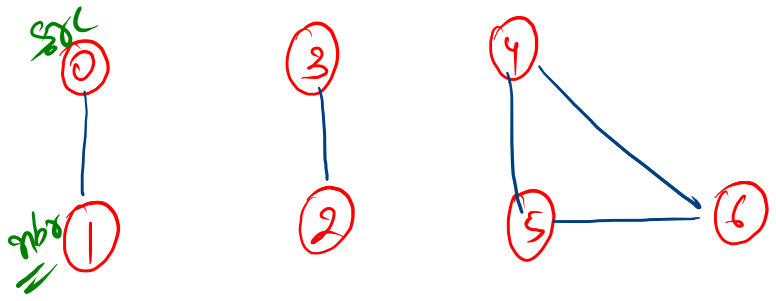
7 → vtx

5 → Edge

$$\begin{bmatrix} 0 & 1 & 10 \\ 2 & 3 & 10 \\ 4 & 5 & 10 \\ 5 & 6 & 10 \\ 4 & 6 & 10 \end{bmatrix}$$



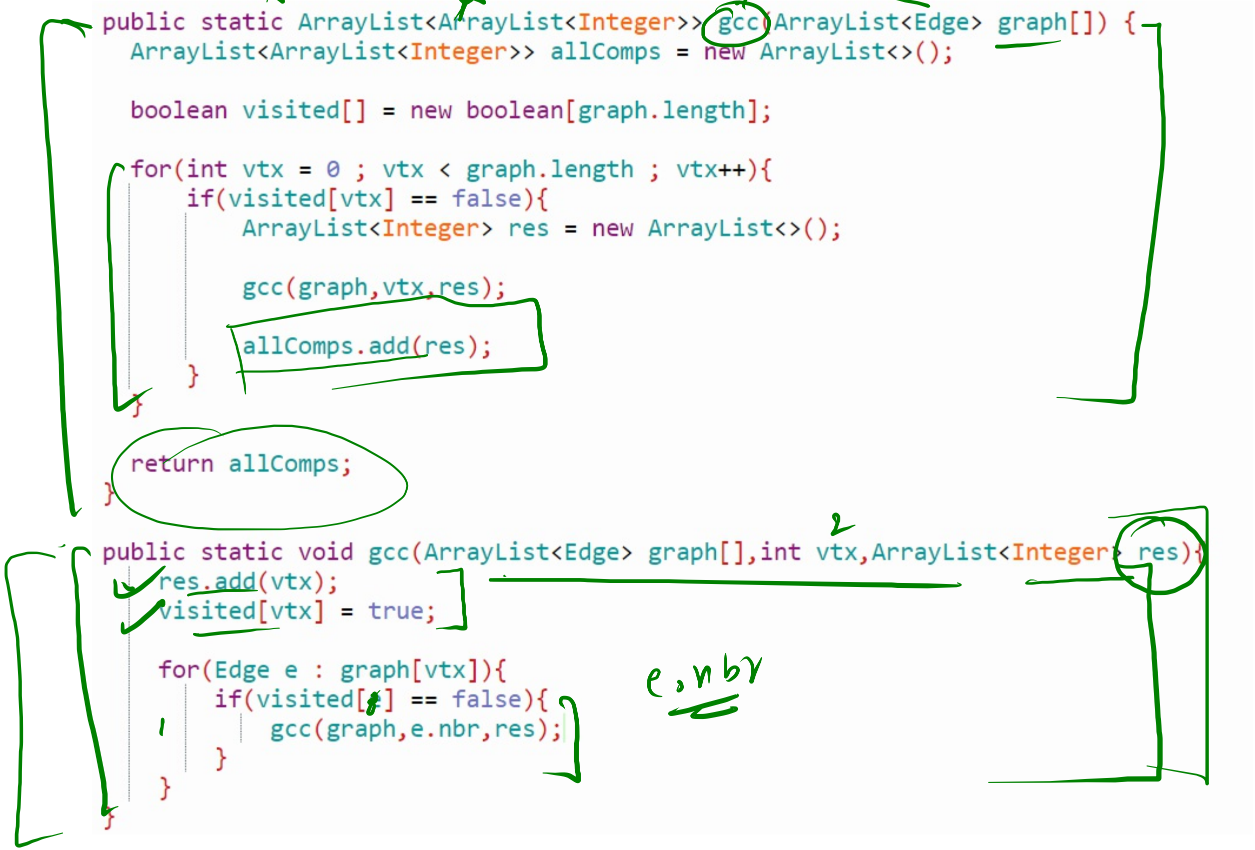
```java
public static ArrayList<ArrayList<Integer>> gcc(ArrayList<Edge> graph[]) {
  ArrayList<ArrayList<Integer>> allComps = new ArrayList<>();

  boolean visited[] = new boolean[graph.length];

  for(int vtx = 0 ; vtx < graph.length ; vtx++){
      if(visited[vtx] == false){
          ArrayList<Integer> res = new ArrayList<>();

          gcc(graph,vtx,res);

          allComps.add(res);
      }
  }

  return allComps;
}
```



allComps = [0 | 1] [2 | 3] [4 | 5 | 6]

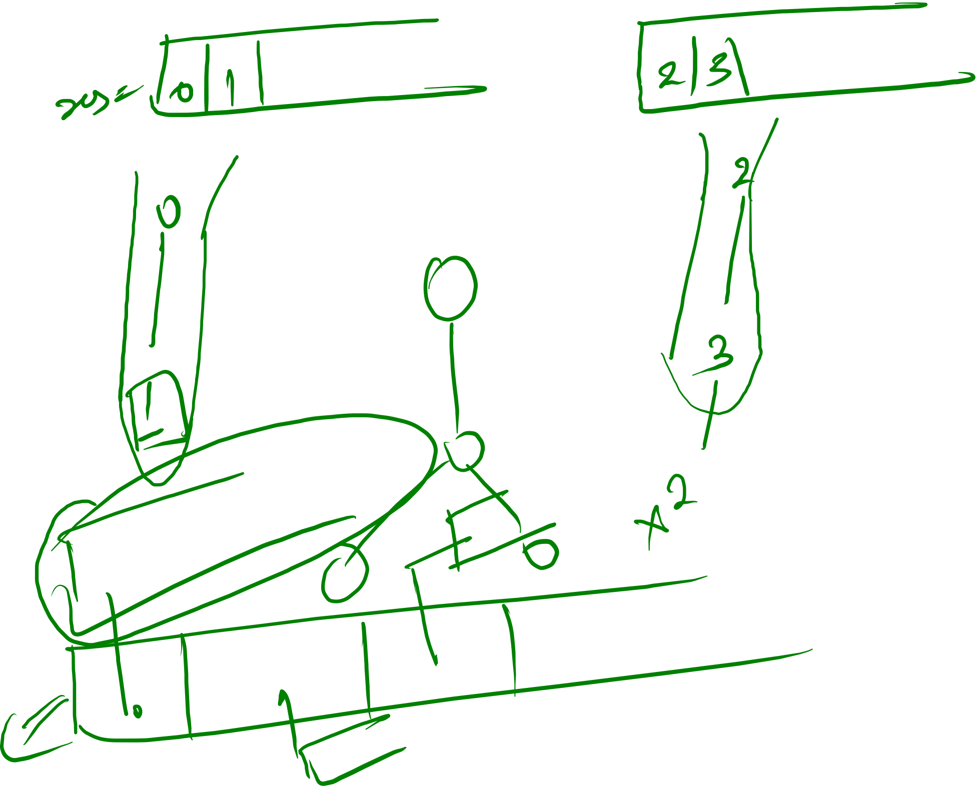
visited = T | T | T | T | ~~F~~ T | ~~F~~ T | ~~F~~ T

0 1 2 3 4 5 6



```
public static ArrayList<ArrayList<Integer>> gcc(ArrayList<Edge> graph[]) {
  ArrayList<ArrayList<Integer>> allComps = new ArrayList<>();

  boolean visited[] = new boolean[graph.length];

  for(int vtx = 0 ; vtx < graph.length ; vtx++){
      if(visited[vtx] == false){
          ArrayList<Integer> res = new ArrayList<>();

          gcc(graph,vtx,res);

          allComps.add(res);
      }
  }

  return allComps;
}

public static void gcc(ArrayList<Edge> graph[],int vtx,ArrayList<Integer> res){
    res.add(vtx);
    visited[vtx] = true;

    for(Edge e : graph[vtx]){
        if(visited[e] == false){
            gcc(graph,e.nbr,res);
        }
    }
}
```

e.nbr



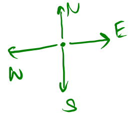
0 ⇒ land

1 ⇒ water/ocean

visit?

No. of islands?

3



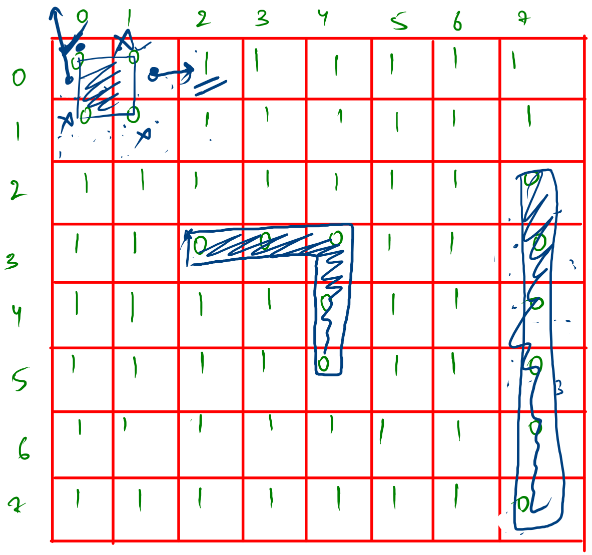
```
8
8
0 0 1 1 1 1 1 1
0 0 1 1 1 1 1 1
1 1 1 1 1 1 1 0
1 1 0 0 0 1 1 0
1 1 1 1 0 1 1 0
1 1 1 1 0 1 1 0
1 1 1 1 1 1 1 0
1 1 1 1 1 1 1 0
```

NEWS

```
8
8
0 0 1 1 1 1 1 1
0 0 1 1 1 1 1 1
1 1 1 1 1 1 1 0
1 1 0 0 0 1 1 0
1 1 1 1 0 1 1 0
1 1 1 1 0 1 1 0
1 1 1 1 1 1 1 0
1 1 1 1 1 1 1 0
```

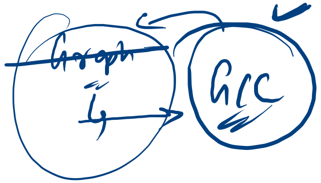
land + unvisited

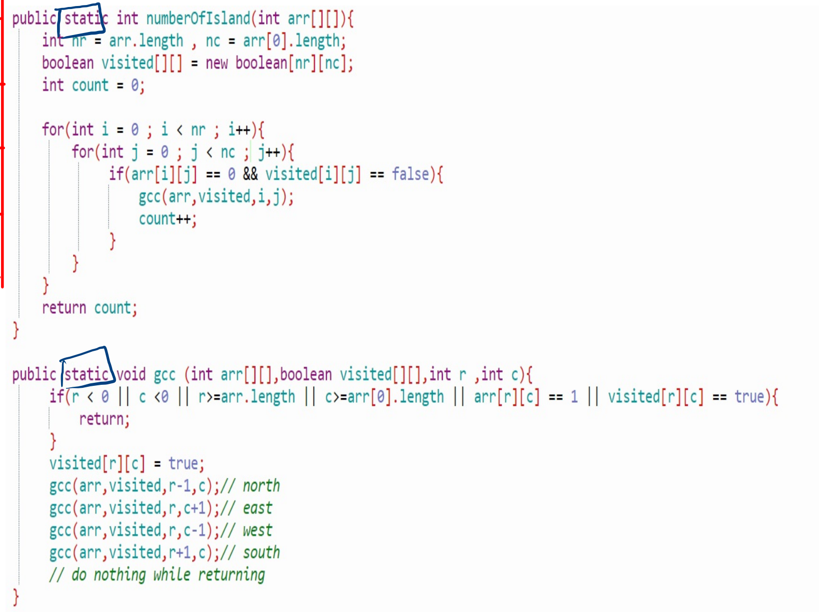
GCC



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

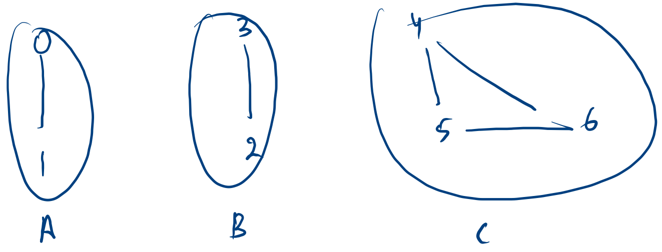
	0	1	2	3	4	5	6	7
0	T	T						
1	T	T						
2								T
3			T	T	T			T
4					T			T
5					T			T
6								T
7								T



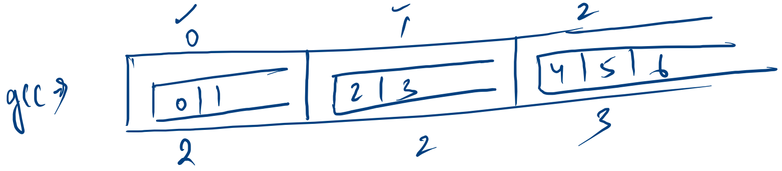


```
public static int numberOfIsland(int arr[][]){
    int nr = arr.length , nc = arr[0].length;
    boolean visited[][] = new boolean[nr][nc];
    int count = 0;

    for(int i = 0 ; i < nr ; i++){
        for(int j = 0 ; j < nc ; j++){
            if(arr[i][j] == 0 && visited[i][j] == false){
                gcc(arr,visited,i,j);
                count++;
            }
        }
    }
    return count;
}

public static void gcc (int arr[][],boolean visited[][],int r ,int c){
     if(r < 0 || c <0 || r>=arr.length || c>=arr[0].length || arr[r][c] == 1 || visited[r][c] == true){
         return;
     }
     visited[r][c] = true;
     gcc(arr,visited,r-1,c);// north
     gcc(arr,visited,r,c+1);// east
     gcc(arr,visited,r,c-1);// west
     gcc(arr,visited,r+1,c);// south
     // do nothing while returning
}
```

7 => n
5 =>
0 1
2 3
4 5
5 6
4 6



no. of ways to select 2 people from groups such that both person shouldn't belong to same club



✓ ✓
A B = 0-2, 0-3, 1-2, 1-3 => 4

✓ ✓
B C = 2-4, 2-5, 2-6 => 6
3-4, 3-5, 3-6

✓ ✓
A C = 0-4, 0-5, 0-6 => 6
1-4, 1-5, 1-6

=> 16 ✓

Two =>

A.B + B.C + ~~A.C~~

=> (2.2) + (2.3) + (2.3)

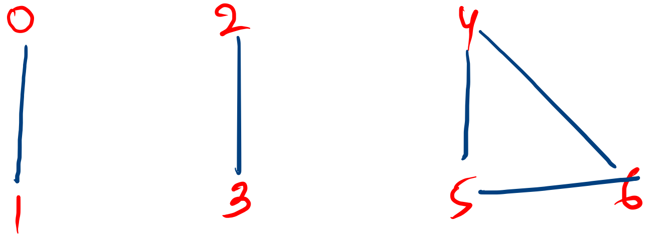
=> 4 + 6 + 6 => 16

2.3

$2^3 \Rightarrow 8$

2.3.6

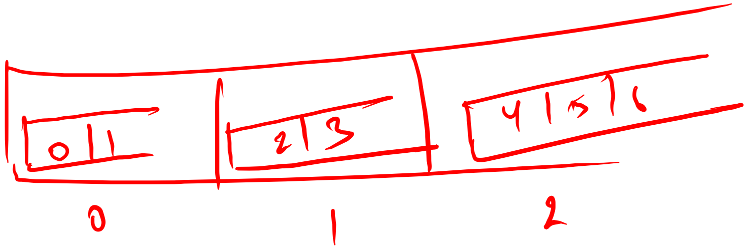
$^3C_1$ = 3 ways



$^{7}C_{2} = \frac{7!}{5!\,2!} \Rightarrow \frac{7.6}{2}$

$$\boxed{^{n}C_{2} \Rightarrow \frac{n.(n-1)}{2}}$$

i



j

$\Rightarrow {}^{7}C_{2} - {}^{2}C_{2} - {}^{2}C_{2} - {}^{3}C_{2}$

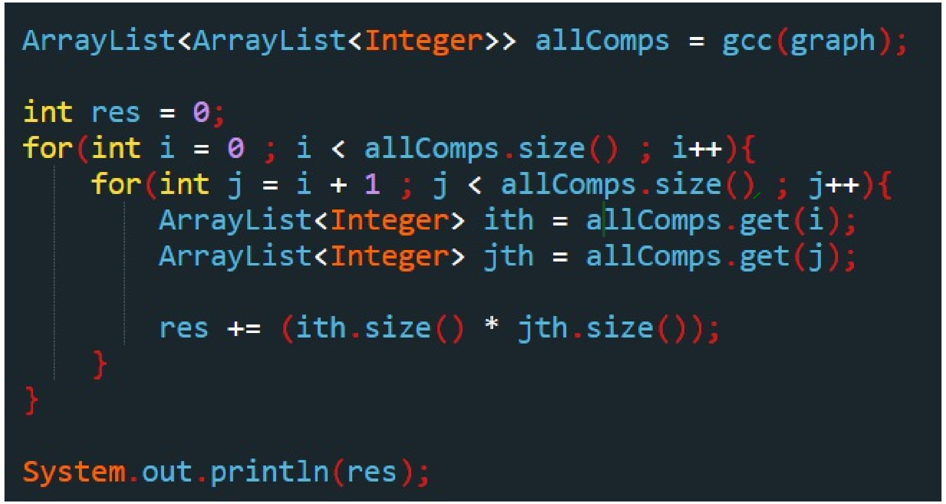
$\Rightarrow 21 - 1 - 1 - 3$

$\Rightarrow 16$

0 1 0 2 1 2

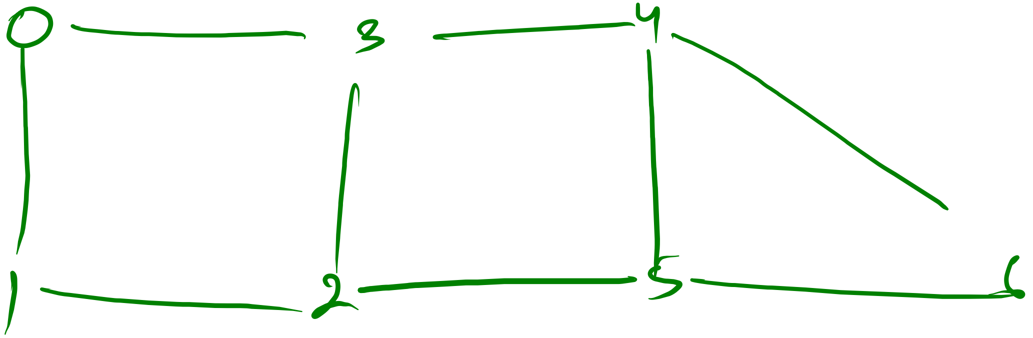
$res = 0 + (2.2) + (2.3) + (2.3)$

$\Rightarrow 16$



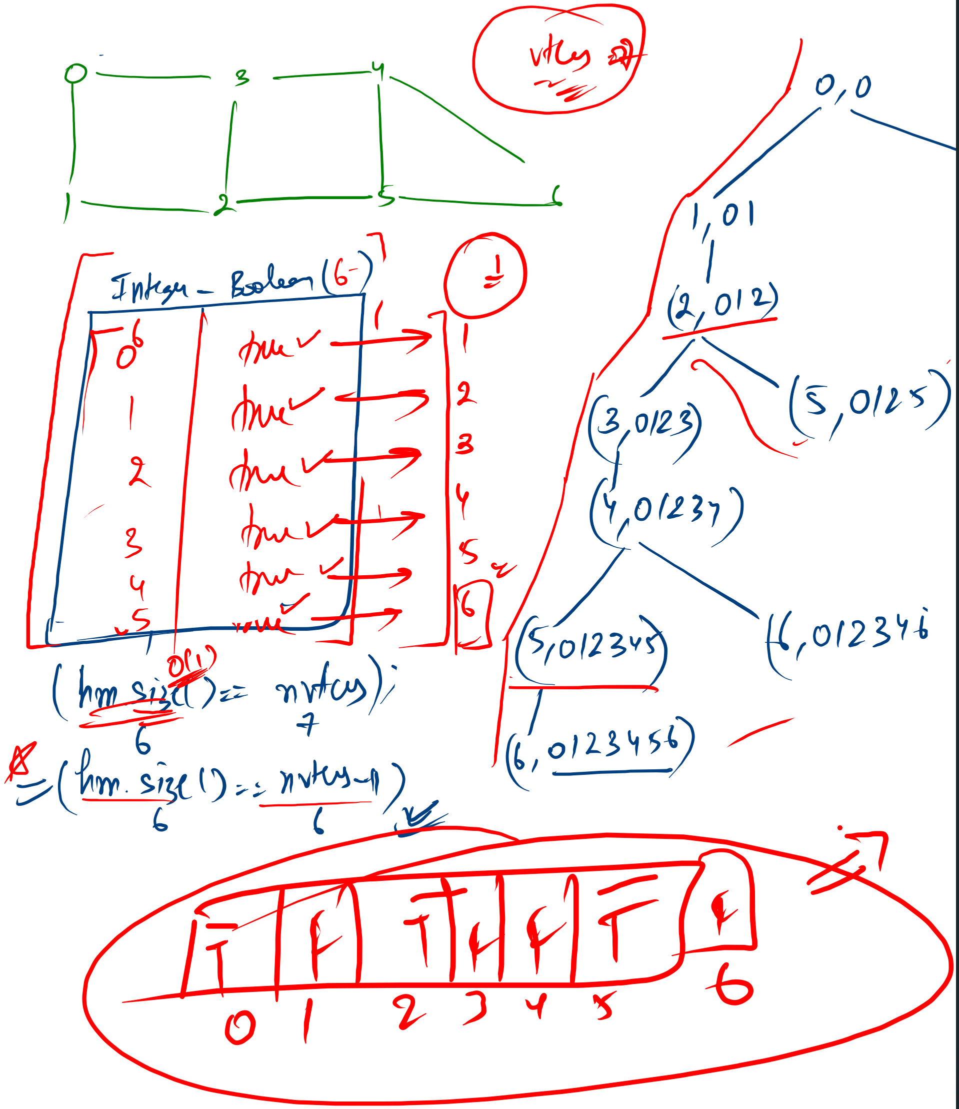
```java
ArrayList<ArrayList<Integer>> allComps = gcc(graph);

int res = 0;
for(int i = 0 ; i < allComps.size() ; i++){
    for(int j = i + 1 ; j < allComps.size() ; j++){
        ArrayList<Integer> ith = allComps.get(i);
        ArrayList<Integer> jth = allComps.get(j);

        res += (ith.size() * jth.size());
    }
}

System.out.println(res);
```

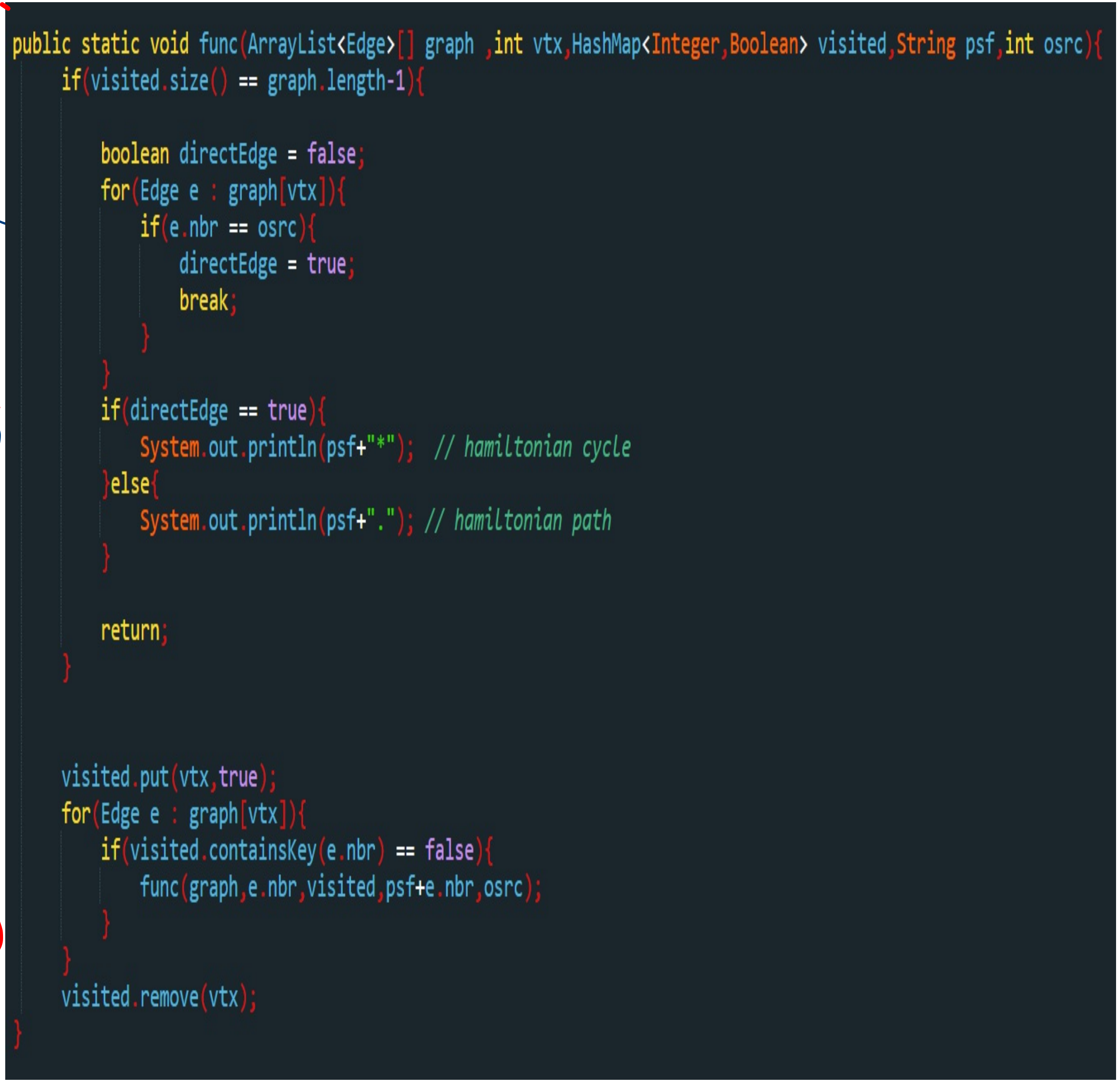
Hamiltonian path → a path which visits each & every vertex, exactly once



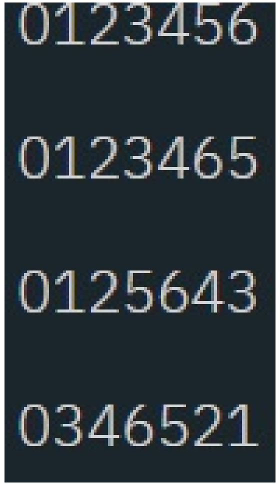
Hamiltonian Cycle

	Hp	Hc
0 1 2 3 4 5 6. →	✓	✗
0 1 2 3 4 6 5. →	✓	✗
0 1 2 5 6 4 3 0 →	✓	✓
0 3 4 6 5 2 1 0 →	✓	✓





```
public static void func(ArrayList<Edge>[] graph ,int vtx,HashMap<Integer,Boolean> visited,String psf,int osrc){
    if(visited.size() == graph.length-1){

        boolean directEdge = false;
        for(Edge e : graph[vtx]){
            if(e.nbr == osrc){
                directEdge = true;
                break;
            }
        }
        if(directEdge == true){
            System.out.println(psf+"*");  // hamiltonian cycle
        }else{
            System.out.println(psf+"."); // hamiltonian path
        }

        return;
    }


    visited.put(vtx,true);
    for(Edge e : graph[vtx]){
        if(visited.containsKey(e.nbr) == false){
            func(graph,e.nbr,visited,psf+e.nbr,osrc);
        }
    }
    visited.remove(vtx);
}
```



```
0123456

0123465

0125643

0346521
```