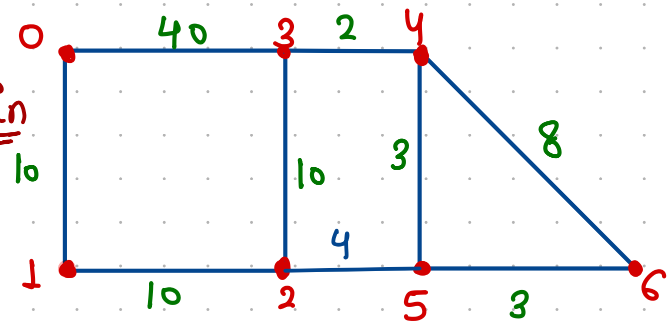
Multisolver:

1. min path ✓
2. max path ✓
3. ceil path
4. Floor path
5. k^{th} largest path

Ceil path
→ qualified min



Factor < [elements]
↓
Min → ceil of factor 'k'

Floor path
↳ qualified max

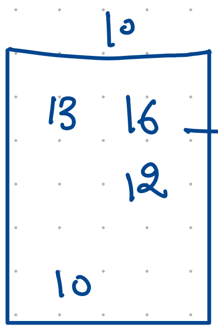
```
if ( wsf < k) {
   // floor → max.

} else if (wsf > k) {
   // ceil → min

}
```

{elements} < factor
↳ max val → floor of factor k

k = 4

10 7 3 13 || 12 4 9 16

Priority Queue of type min



→ k largest Element] we can hold it on priority Queue

k^{th} Largest

array → ~~10~~ ~~7~~ ~~9~~ ~~4~~ 13 16 ~~8~~ ~~3~~ 12 17

$K=4$

k^{th} Largest Element

priority Queue
↓
top/peek
→ min
→ max

Min priority Queue

17
13, 16
12

pq → min

$\boxed{12}$ ↑ top

Smallest among 4 Largest Element } 4^{th} Largest element

Heap

Major code of k^{th} largest

```
if(pq.size() < k) {
    pq.add(val);
} else {
    pq.add(val);
    pq.remove() ⟶ remove peek element.
}
```

pq → min
max

Method 2. (without priority queue). ⟶ using floor path concept

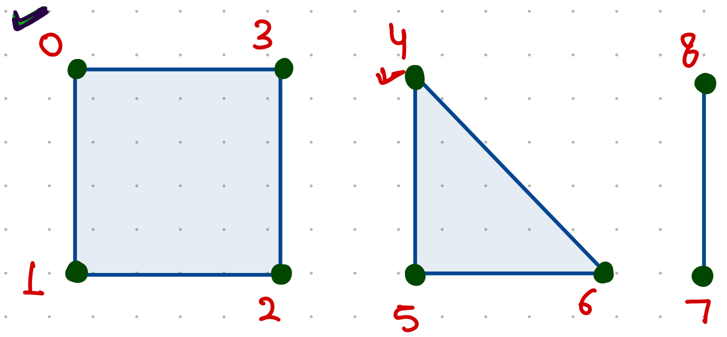
K-times call → floor path.

Factor = ∞ max1 max2 max3

Floor path = max1 max2 max3 max4. ⟶ Result

Get connected component :

connected → From a vertex, if we can visit all vertices then graph is connected



connected components :

components → [[0, 1, 2, 3], [4, 5, 6], [7, 8]]

ArrayList <ArrayList <Integer>> components.

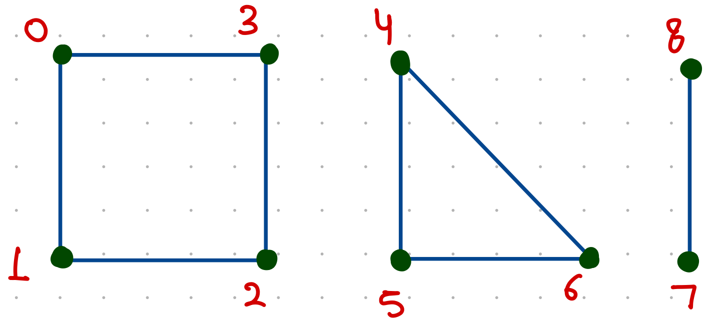
calling function, ⟶ Traversal function.

NOTE:

① There are two function Required. First is calling function which helps to call disconnected vertex

② Another one is helpful to get connected comp.

③ Do not unmark in connect comp function, otherwise we will reiterate again & again on same vertex.



vis →

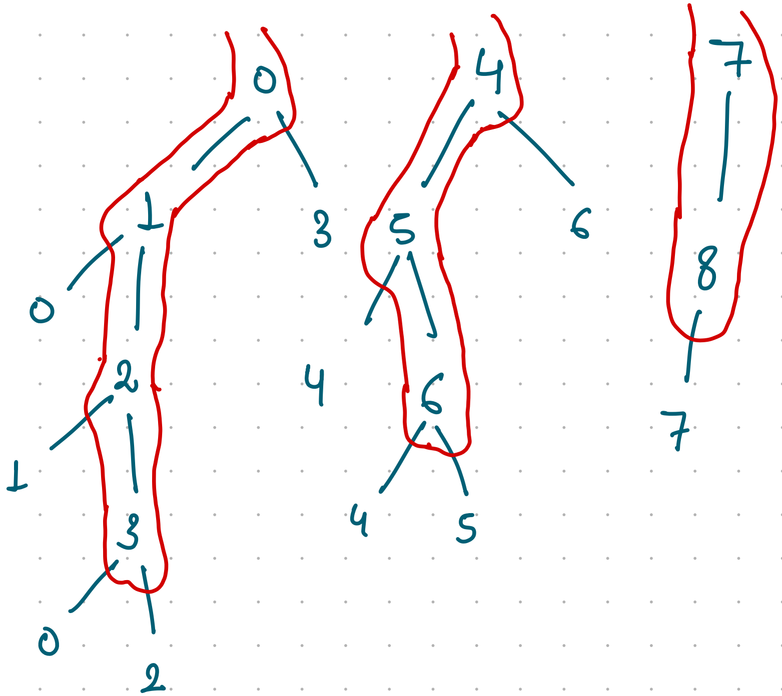
0	1	2	3	4	5	6	7	8
T	T	T	T	T	T	T	T	T

V = 0 1 2 3 4 5 6 7 8

```java
public static void gcc(ArrayList<Edge>[] graph, int src, boolean[] vis, ArrayList<Integer> comp) {
    vis[src] = true;
    comp.add(src);
    for(Edge e : graph[src]) {
        if(vis[e.nbr] == false) {
            gcc(graph, e.nbr, vis, comp);
        }
    }
}

public static ArrayList<ArrayList<Integer>> getConnectedComponents(ArrayList<Edge>[] graph) {
    ArrayList<ArrayList<Integer>> comps = new ArrayList<>(); // components
    boolean[] vis = new boolean[graph.length];

    for(int v = 0; v < graph.length; v++) {
        ArrayList<Integer> comp = new ArrayList<>(); // component
        gcc(graph, v, vis, comp);
        comps.add(comp);
    }
    return comps;
}
```

if (vis[v] == false){

}



comp → [7, 8]

comps → [0, 1, 2, 3], [4, 5, 6], [7, 8]

Final Result [[0, 1, 2, 3], [4, 5, 6], [7, 8]] Ans

Is graph connected:

✓ Connected: From any single vertex, if we can visit all vertex then graph is connected.

gcc ✓ comps.size() > 1 → $\underbrace{\text{Graph is not connected}}$

gcc ✓ If you are calling more than one time to gcc function then graph is not connected

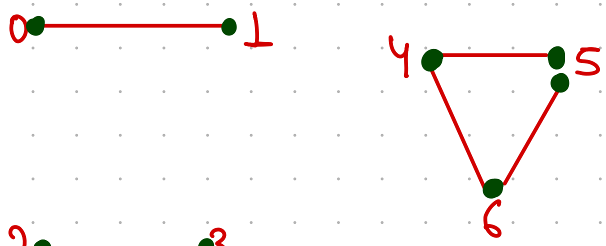
otherwise → graph is connected.

```java
public static void gcc(ArrayList<Edge>[] graph, int src, boolean[] vis) {
    vis[src] = true;
    for(Edge e : graph[src]) {
        if(vis[e.nbr] == false) {
            gcc(graph, e.nbr, vis);
        }
    }
}

public static boolean isGraphConnected(ArrayList<Edge>[] graph) {
    boolean[] vis = new boolean[graph.length];
    int count = 0;
    for(int v = 0; v < graph.length; v++) {
        if(vis[v] == false) {
            if(count == 1) return  false;
            count++;
            gcc(graph, v, vis);
        }
    }
    return true;
}
```

Perfect friend:

n - pair   In every pair, students are belongs to same pair

0 - 1 ✓
2 - 3 ✓
4 - 5 ✓
5 - 6 ✓
4 - 6 ✓



Find no of ways to select '2' students, such that both are belongs to different club

component → [0, 1], [2, 3] [4, 5, 6] → $O(n^2)$

2 2 3

S1 S2 S3

S1 X S2 → 0-2, 0-3, 1-2, 1-3] → 4

S1 x S3 → 0-4, 0-5, 0-6, 1-4, 1-5, 1-6 →] 6

S2 x S3 → 2-4, 2-5, 2-6, 3-4, 3-5, 3-6 →] 6

total no. of ways = 16

components size → S1 S2 S3 S4 S5

How to find no. of pairs.

comp size →

S1	S2	S3	S4	S5	S6
S1xS2	S2xS3	S3xS4	S4xS5	S5xS6	0
S1xS3	S2xS4	S3xS5	S4xS6	S5(S6)	0
S1xS4	S2xS5	S3xS6	S4(S5+S6)		
S1xS5	S2xS6	S3(S4+S5+S6)			
S1xS6	S2(S3+S4+S5+S6)				
S1(S2+S3+S4+S5+S6)					

res = 0 ← sum=0

sum = S2+S3+S4+S5+S6	sum = S3+S4+S5+S6	sum = S4+S5+S6	sum = S5+S6	Sum = 6	sum=0
res += S1 x sum	res += S2 x sum	res += S3 x sum	res += S4 x sum	res += S5 x sum	res += S6 x sum
sum += S1	sum = S2	sum += S3	sum += S4	sum += S5	sum += S6

travel from back to front →