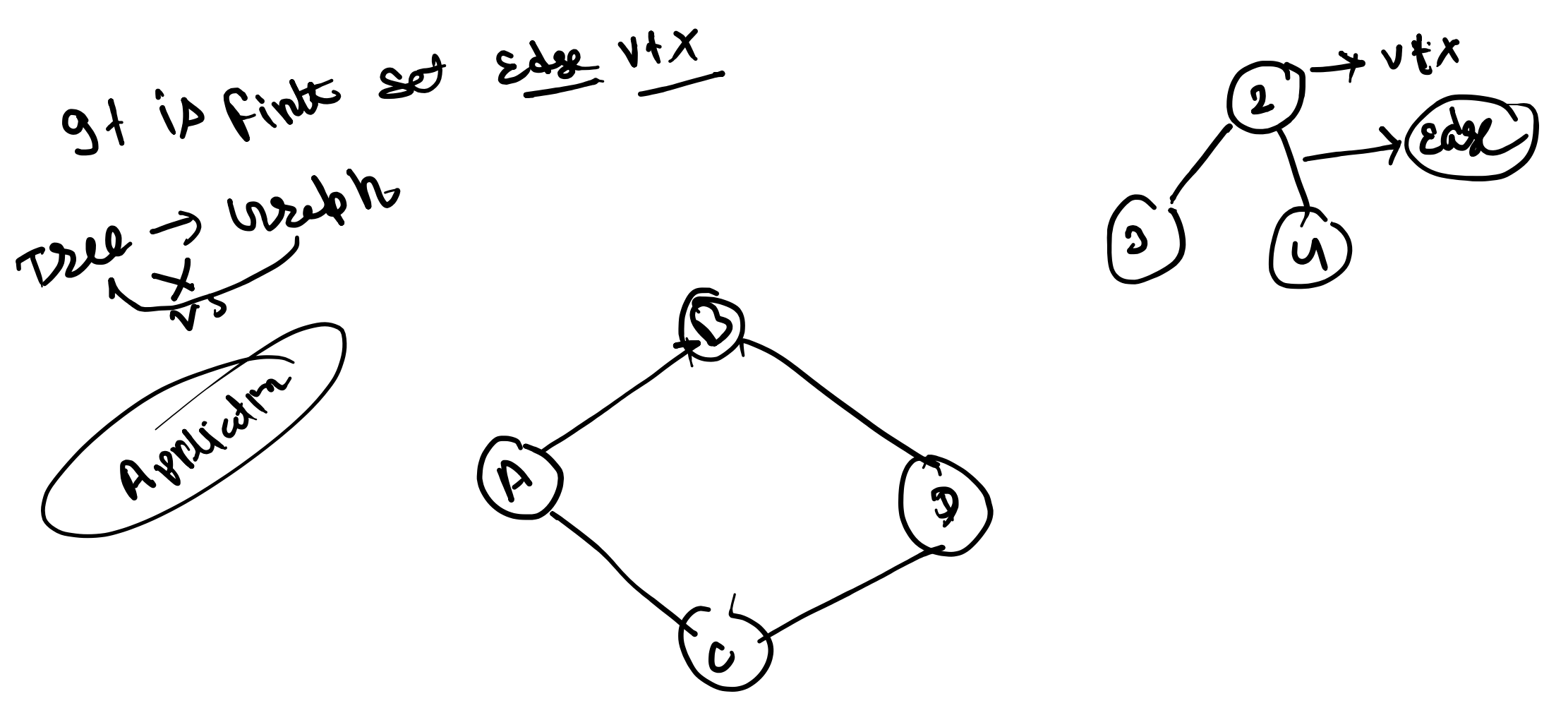
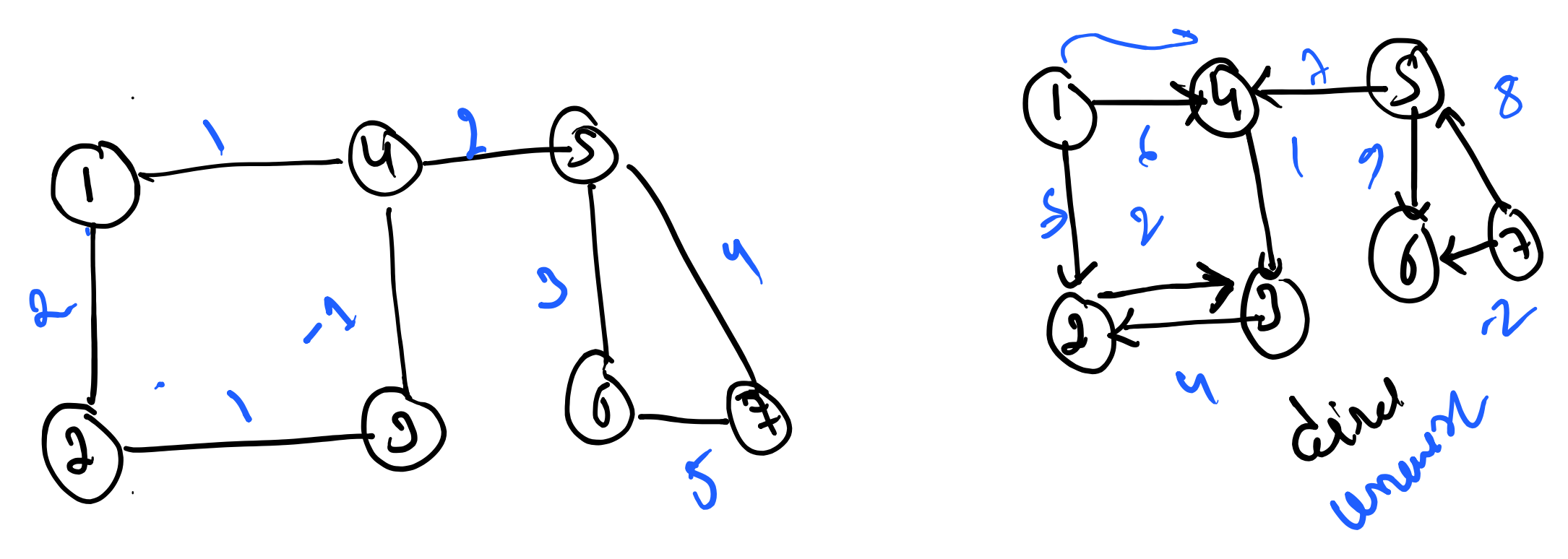
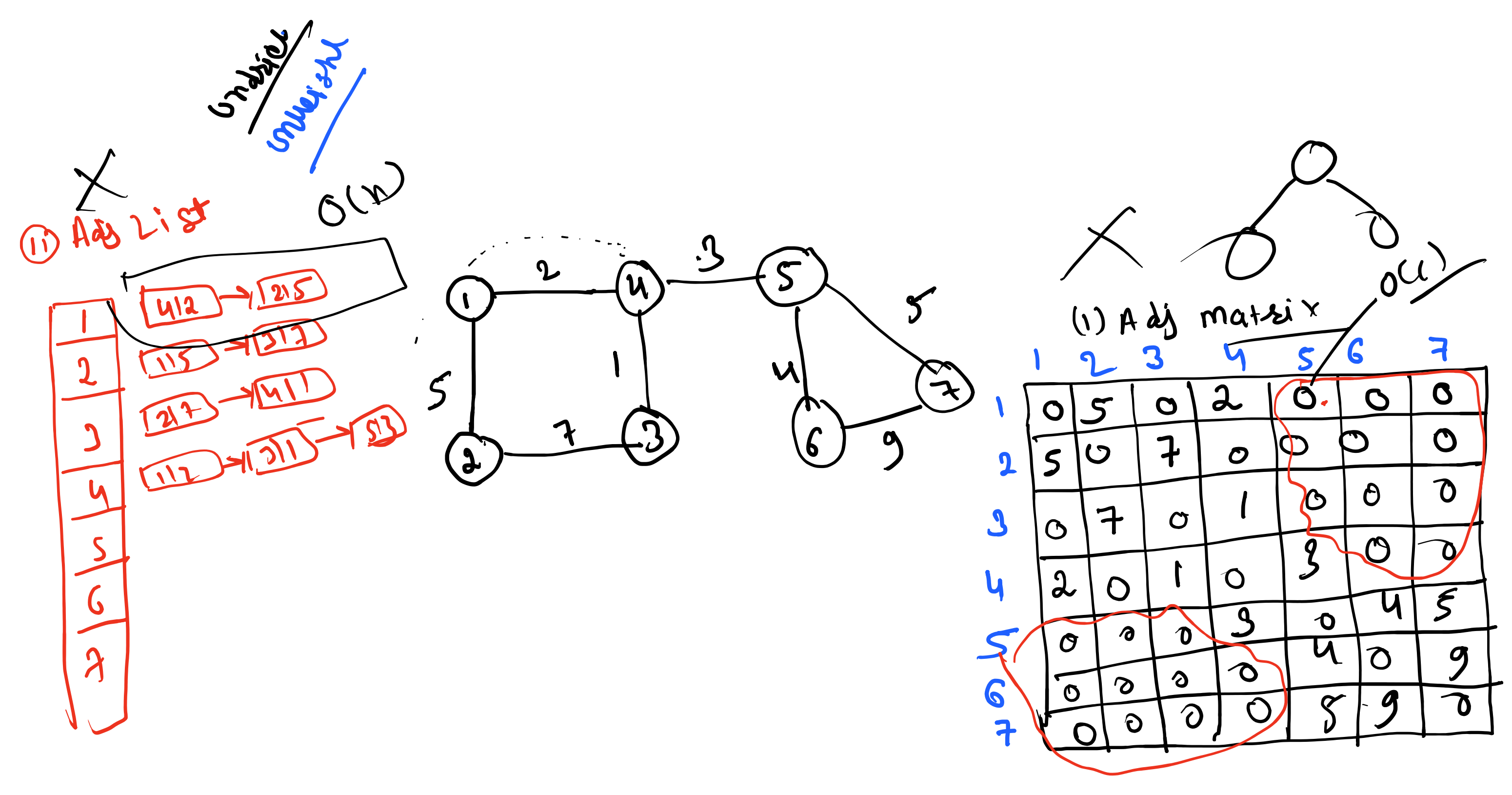
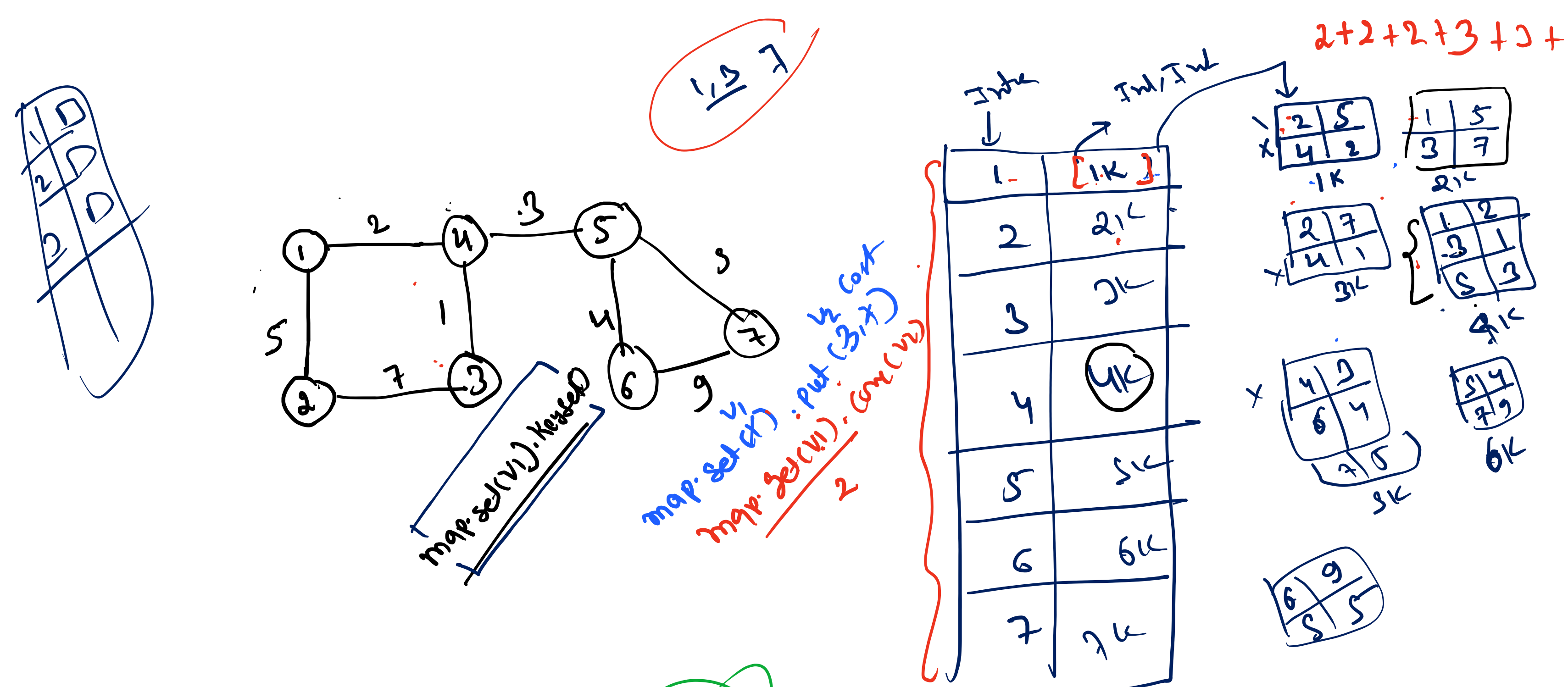
Lec-41

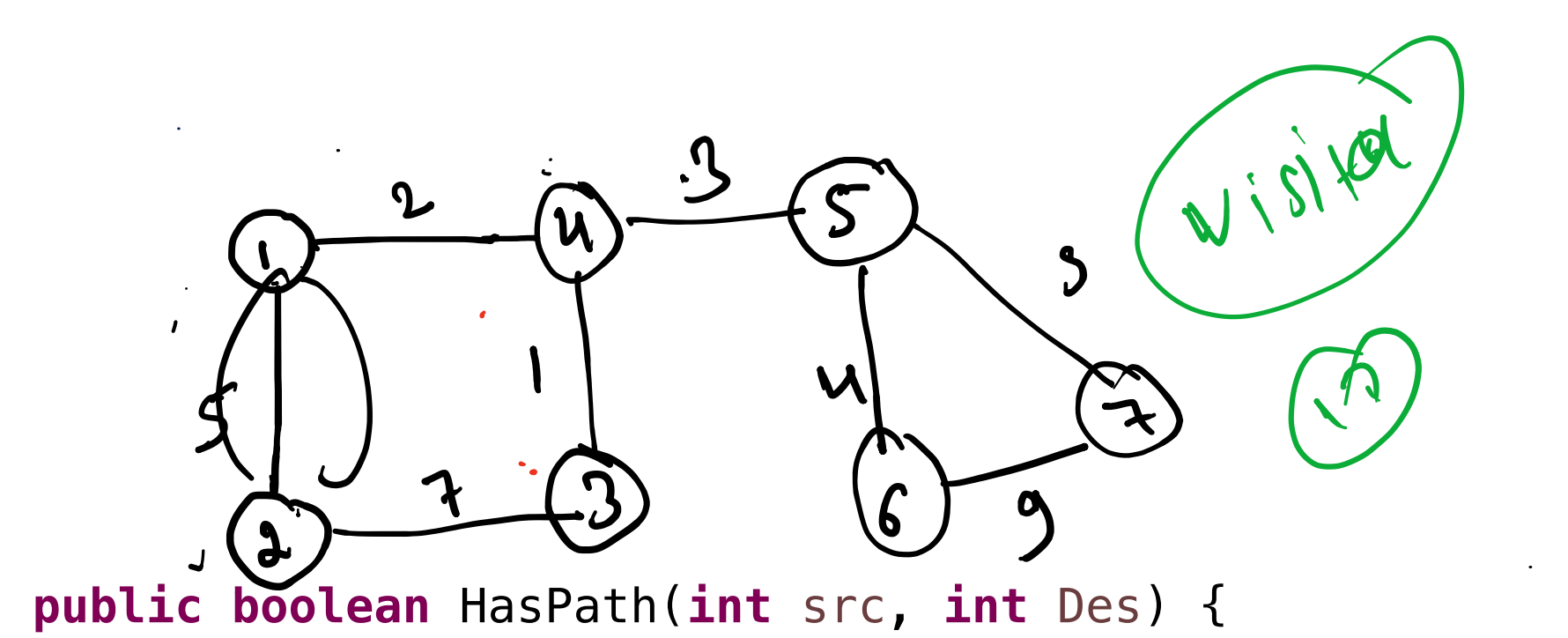
01/03/25 12:56 PM


Gt is finite set Edge vtx
Tree → Graph
Application
vtx
edge
A
B
C
D

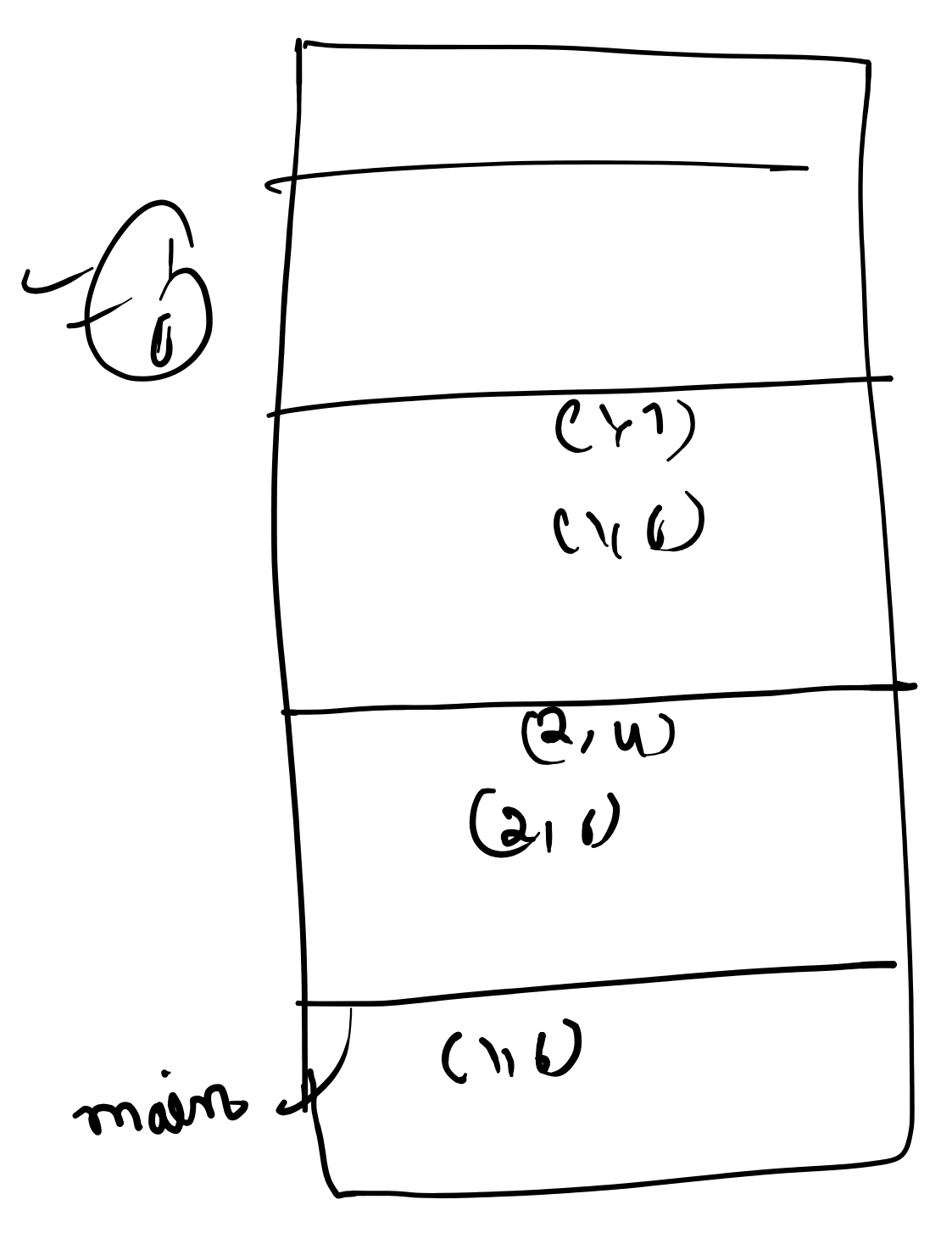

1
2
3
4
5
6
7
dird
unweighted


Undirected
weighted
O(n)
(1) Adj List
(1) Adj matrix
O(1)

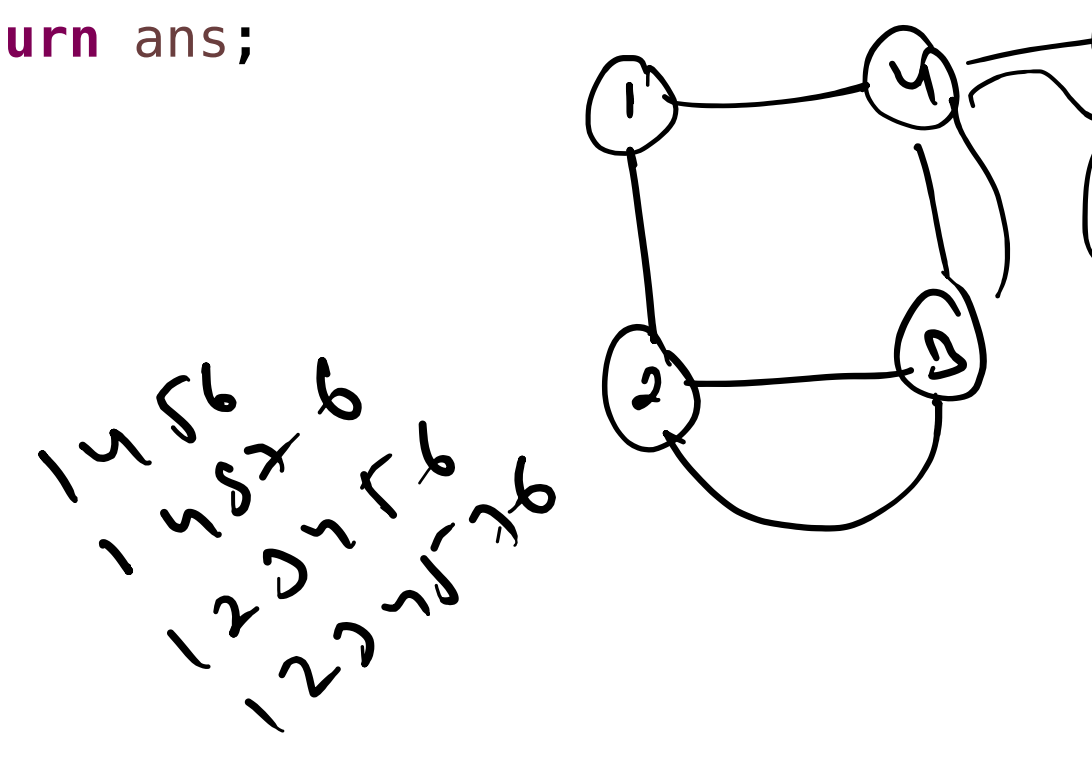

1,3 7
map.get(v1).keySet()
map.get(v1).put(3,7)
map.get(v1).get(v2)
2+2+2+3+3+2+2=16/2
1K
2K
3K
4K
5K
6K
7K

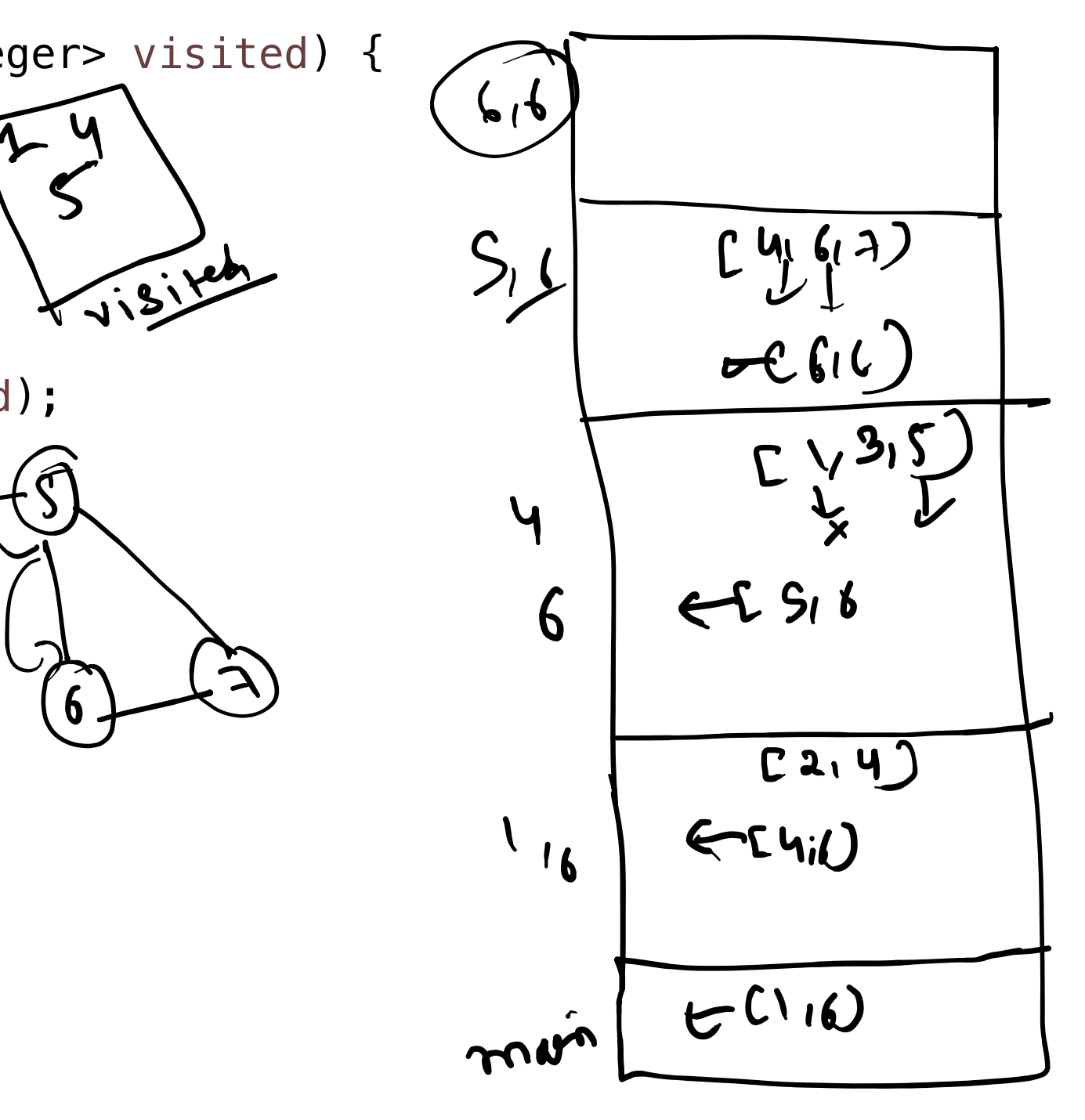


```
public boolean HasPath(int src, int Des) {
    if (src == Des) {
        return true;
    }
    for (int nbrs : map.get(src).keySet()) {
        boolean ans = HasPath(nbrs, Des);
        if (ans == true) {
            return ans;
        }
    }
    return false;

}
```

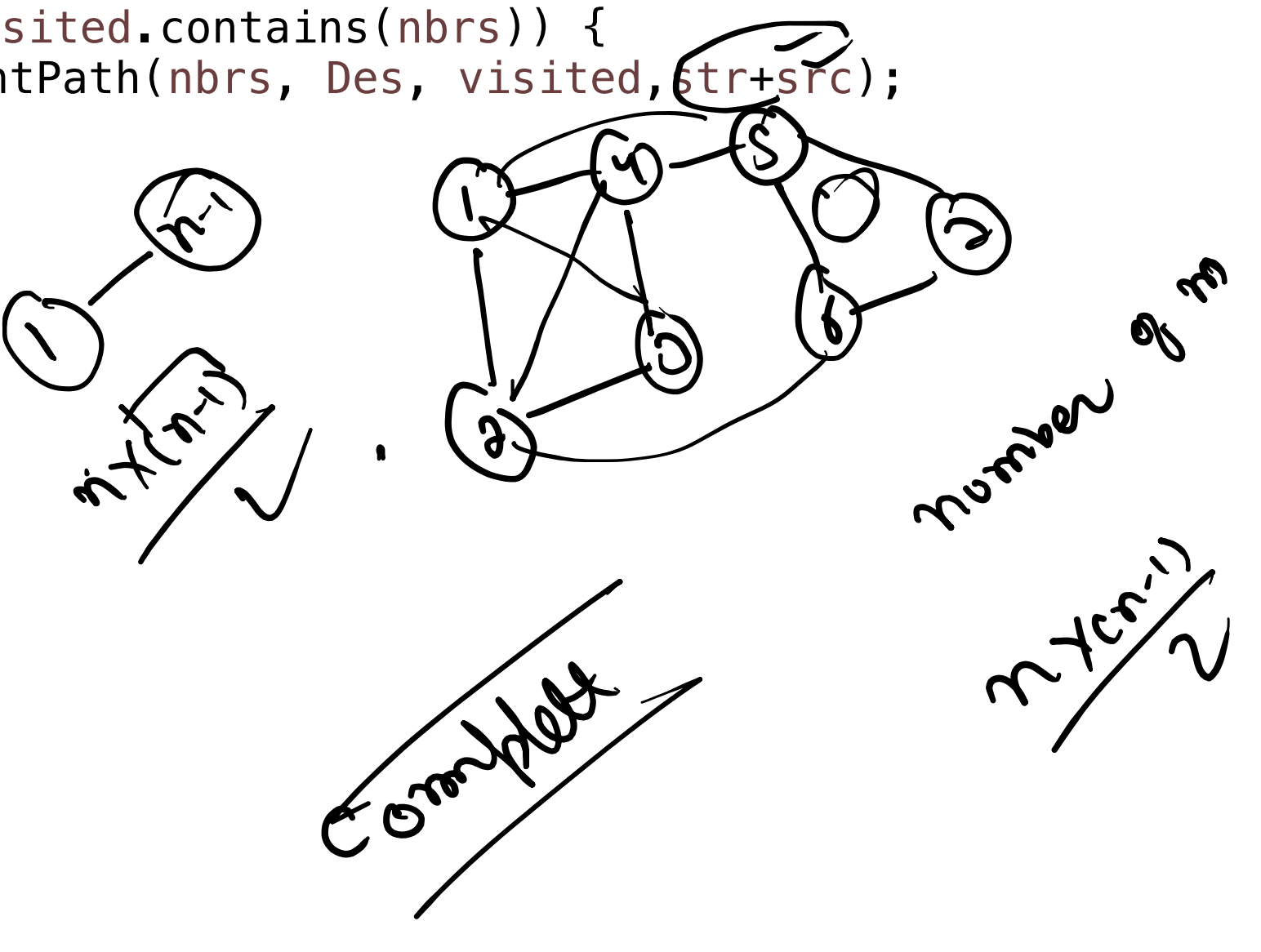

(4,7)
(1,6)
(2,4)
(2,1)
main
(1,6)

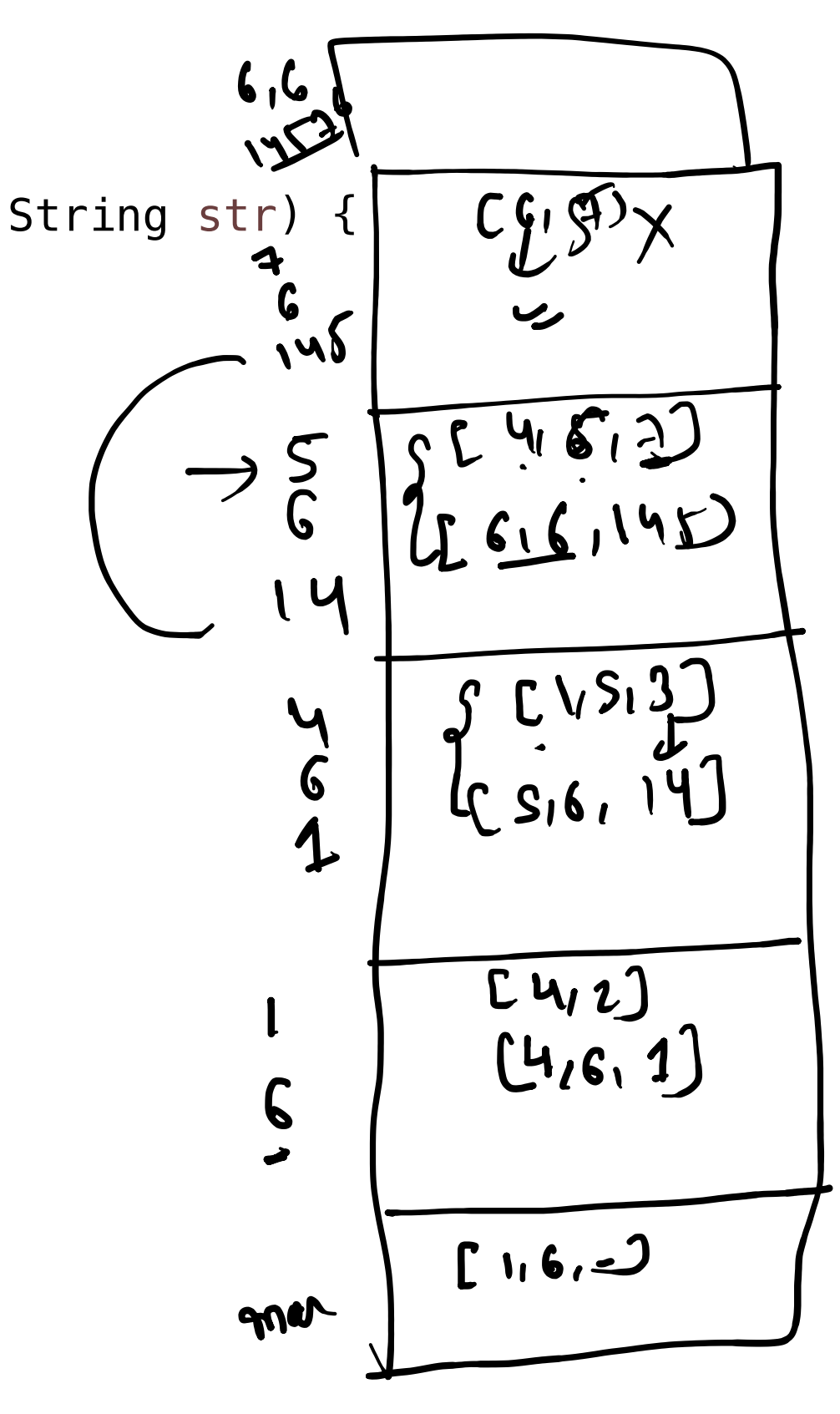
```
public boolean HasPath(int src, int Des, HashSet<Integer> visited) {
    if (src == Des) {
        return true;
    }
    visited.add(src);
    for (int nbrs : map.get(src).keySet()) {
        if (!visited.contains(nbrs)) {
            boolean ans = HasPath(nbrs, Des, visited);
            if (ans == true) {
                return ans;
            }
        }
    }
    return false;
}
```


1456 6
14576
123456
123476

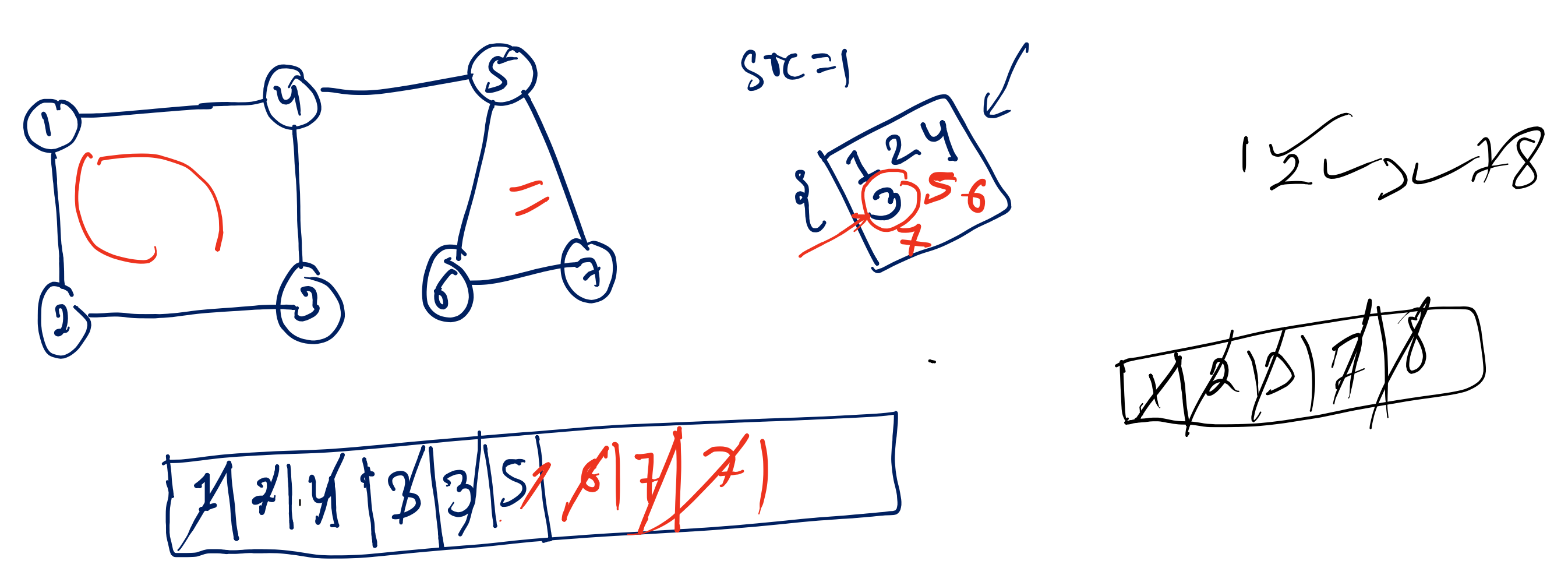

1 2 4 5
visited
6,6
5,6
[4,6,7]
[6,6]
[1,3,5]
[5,6]
[2,4]
[4,1]
main
[1,6]

```
public void PrintPath(int src, int Des, HashSet<Integer> visited,String str) {
    if (src == Des) {
        System.out.println(str);
        return;
    }
    visited.add(src);
    for (int nbrs : map.get(src).keySet()) {
        if (!visited.contains(nbrs)) {
            PrintPath(nbrs, Des, visited, str+src);

        }
    }
}
```

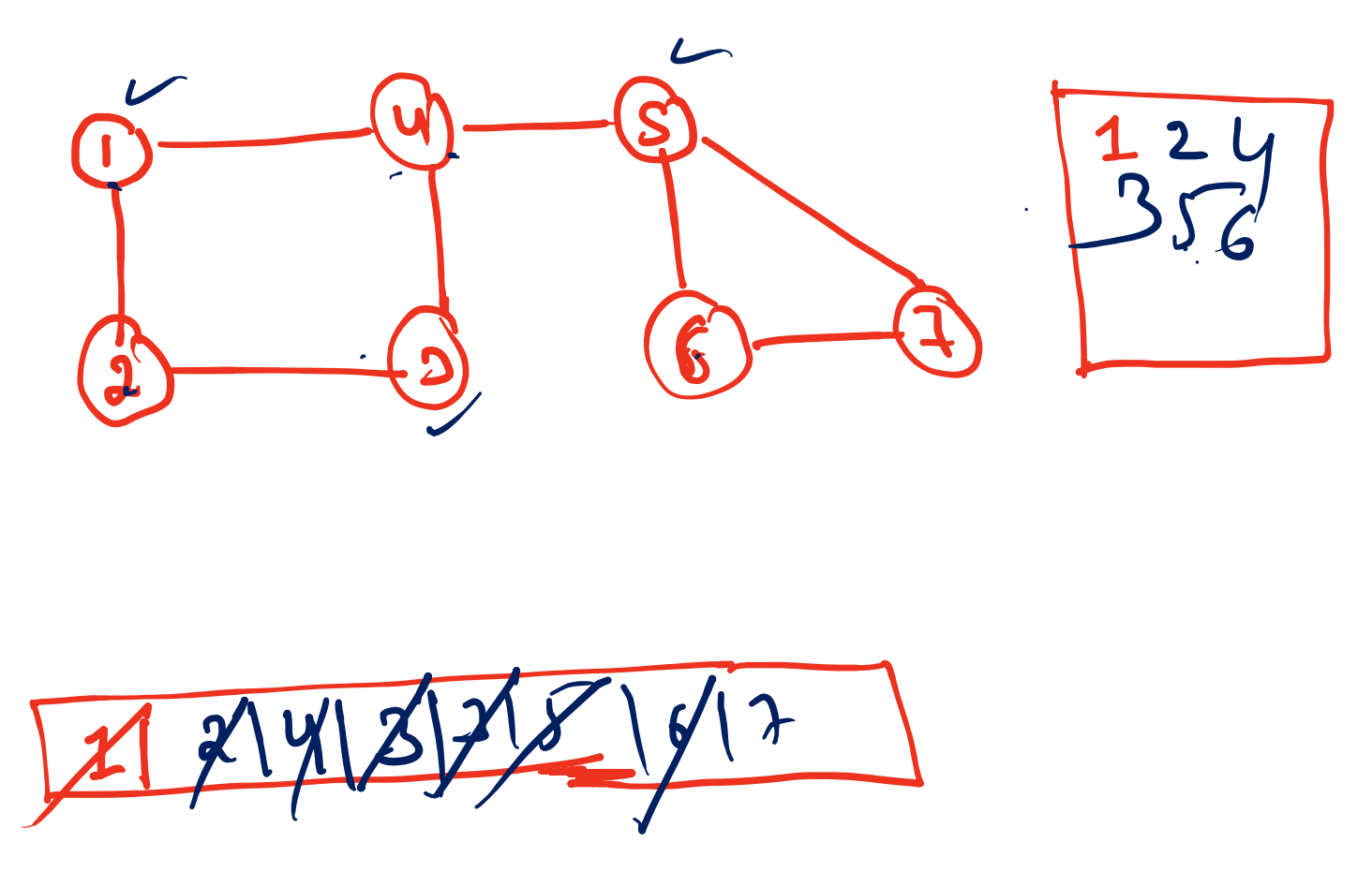

n-1
n×(n-1)
Complete
number of m
n×(n-1)/2


6,6
[6,6]
[4,6,7]
[6,6,145]
[1,5,3]
[5,6,14]
[4,2]
[4,6,1]
main
[1,6,-]

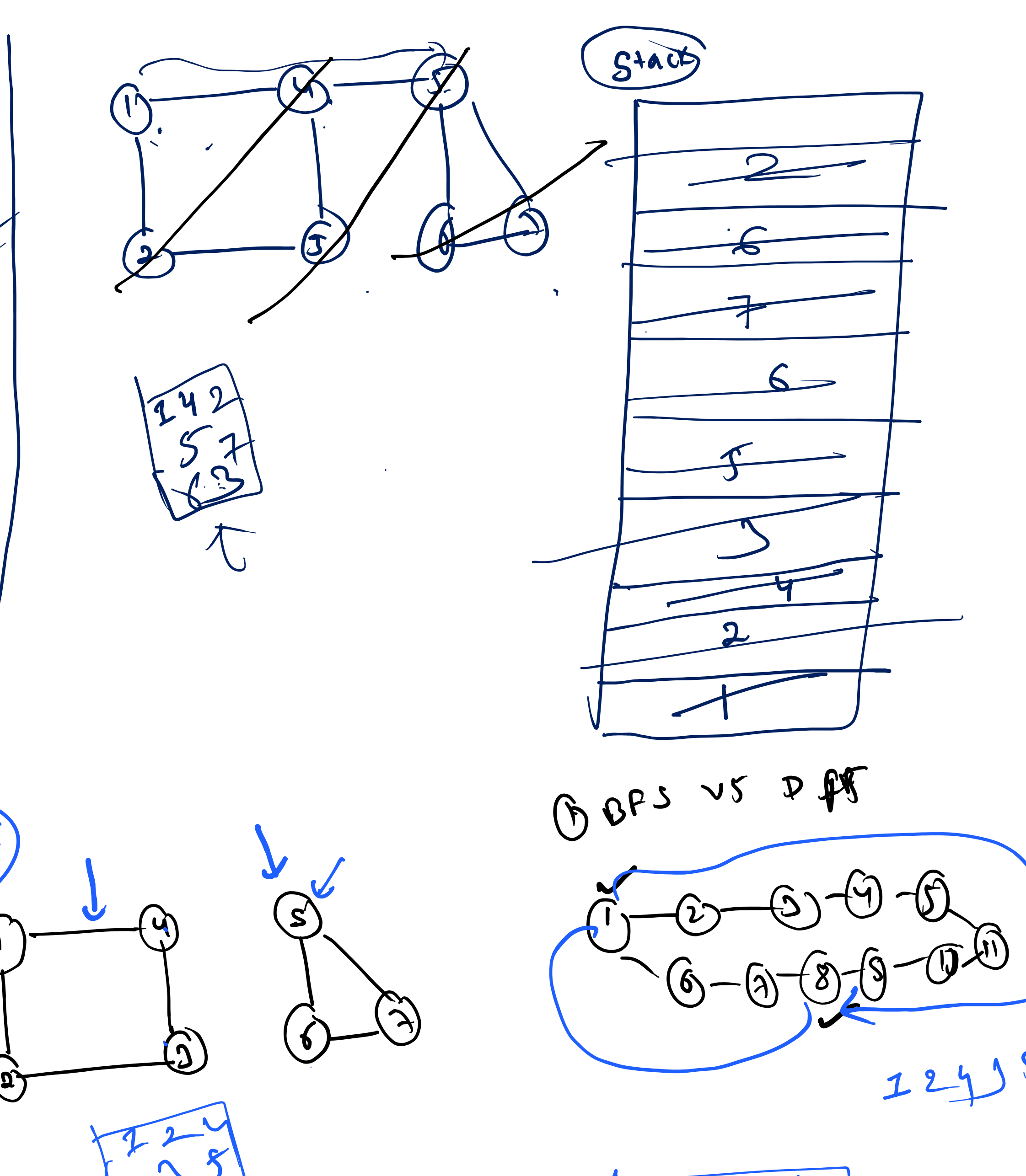
1. remove
2. Ignore
3. visited
4. self work
5. Add nbrs unvisited

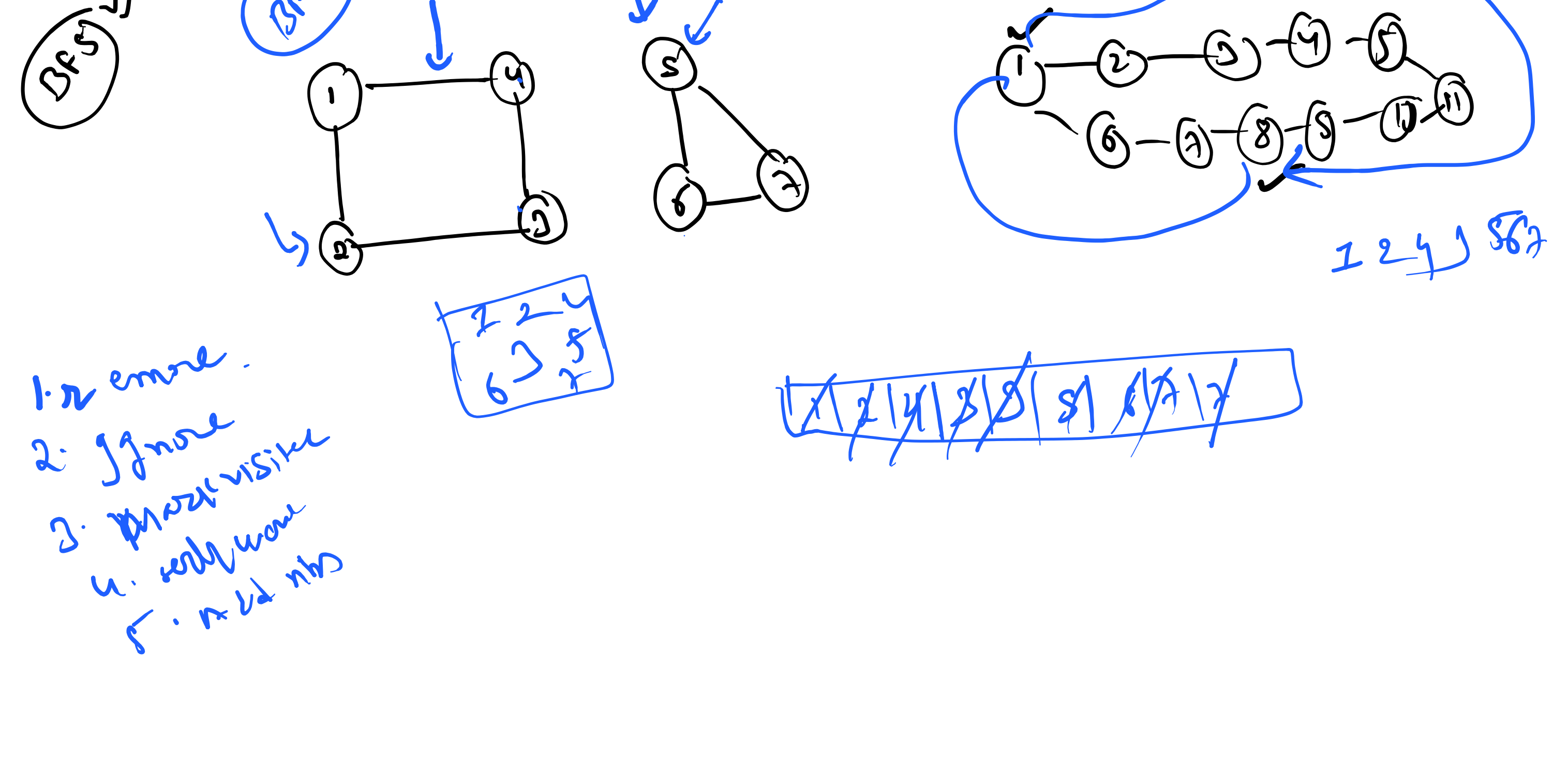

src=1
1 2 4
3 5 6
7

```
public boolean BFS(int src, int des) {
    Queue<Integer> q = new LinkedList<>();
    HashSet<Integer> visited = new HashSet<>();
    q.add(src);
    while (!q.isEmpty()) {
        // 1. remove
        int rv = q.poll();
        // 2. Ignore if Already Visited
        if (visited.contains(rv)) {
            continue;
        }
        // 3. Marked Visited
        visited.add(rv);
        // 4. self work
        if (rv == des) {
            return true;
        }
        // 5. add unvisited nbrs
        for (int nbrs : map.get(rv).keySet()) {
            if (!visited.contains(nbrs)) {
                q.add(nbrs);
            }
        }

    }
    return false;
}
```



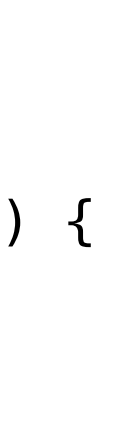
1. remove
2. Ignore
3. marked visited
4. self work
5. Add nbrs


Stack
1 4 2
5 7
6 3


DFS
BFT
1 2 4
6 3 5
7
BFS vs DFS
1 2 4 3 5 6 7

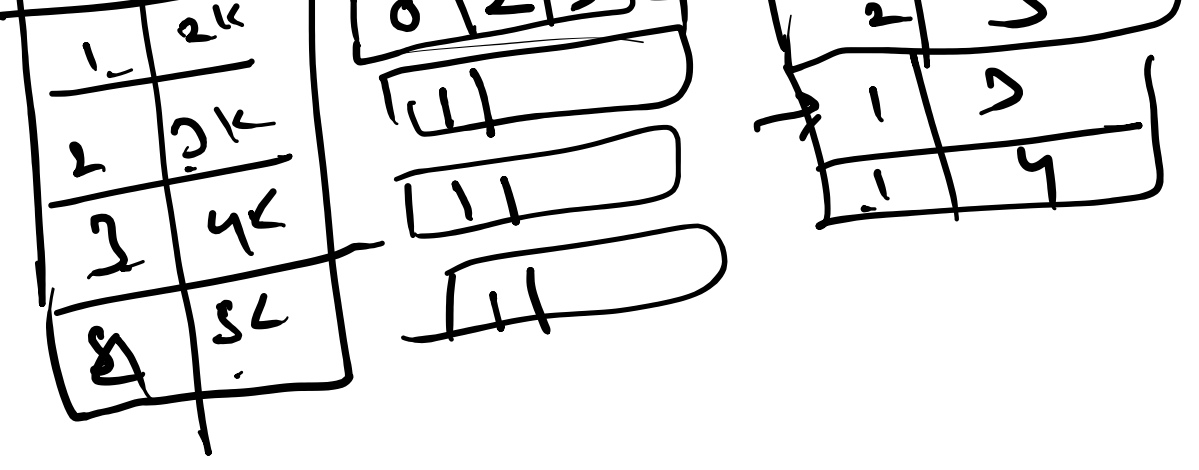
1. remove
2. Ignore
3. Mark visited
4. self work
5. add nbrs

```
public void BFT() {
    Queue<Integer> q = new LinkedList<>();
    HashSet<Integer> visited = new HashSet<>();
    for (int src : map.keySet()) {

        if(visited.contains(src)) {
            continue;
        }
        q.add(src);
        while (!q.isEmpty()) {
            // 1. remove
            int rv = q.poll();
            // 2. Ignore if Already Visited
            if (visited.contains(rv)) {
                continue;
            }
            // 3. Marked Visited
            visited.add(rv);
            // 4. self work
            System.out.print(rv+" ");
            // 5. add unvisited nbrs
            for (int nbrs : map.get(rv).keySet()) {
                if (!visited.contains(nbrs)) {
                    q.add(nbrs);
                }
            }

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


1243567
1 2 4
3 5 6

n = 5, edges = [[0,1],[1,2],[2,3],[1,3],[1,4]]

```
public boolean validTree(int n, int[][] edges) {
    HashMap<Integer, List<Integer>> map = new HashMap<>();
    for (int i = 0; i < n; i++) {
        map.put(i, new ArrayList<>());
    }
    for (int i = 0; i < edges.length; i++) {
        int v1=edges[i][0];
        int v2=edges[i][1];
        map.get(v1).add(v2);
        map.get(v2).add(v1);
    }
}
```


0
1K
2K
3K
4K
5K