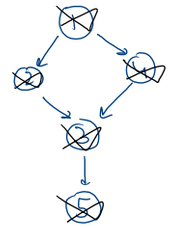
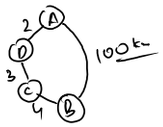
```
public boolean dfs(ArrayList<Integer>[] graph,boolea
    //Mark
    path[curr] = true;
    vis[curr] = true;

    //Work - nothing
    //Try to visit any unvisited nbrs
    for(int nbr:graph[curr]){
        if(path[nbr]) return true;
        else if(!vis[nbr]){
            boolean isCyclic = dfs(graph,vis,path,nbr
            if(isCyclic) return true;
        }
    }
    //Unmark the path array
    path[curr] = false; //the snake is backtracking
    return false;
}
```

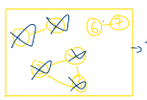
Vis: T T T T T
1 2 3 4 5

Path: 1 2 3 4 5



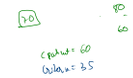


Components of A Graph



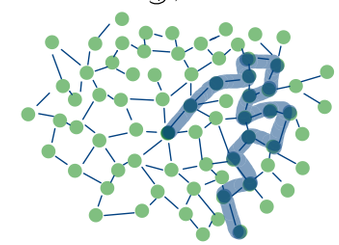
→ 3 components

3 Just apply BFS on every unvisited node.

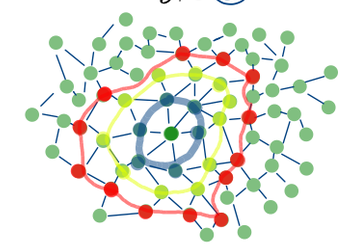


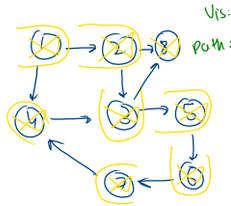
Cpatnt = 60
Criterion = 35

DFS



BFS





Vis: T T T T T T T T
1 2 3 4 5 6 7 8

Path: T T T T T T T T
1 2 3 4 5 6 7 8

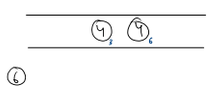
4
7
6
5
3
2
1

* Mark
~~* Work~~
* Visit unvisited
* unmark path

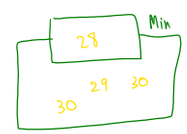
If any of my nbrs is true in the path array, this means snake is cutting itself

while(q.size()>0) {
//Remove
//Mark
if already marked return True
Mark it
//work - - -
// Add unvisited NBRs.

Detect Cycle undirected

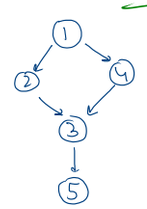


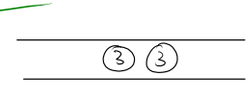
K = 4



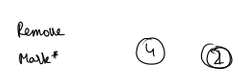
pq.size() <= 4

DFS





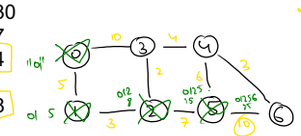
Remove
Mark*
work
Add unvisited



0123456 @ 30
012346 @ 17
012546 @ 24
01256 @ 25
032546 @ 28
03256 @ 29
03456 @ 30
0346 @ 17

src = 0
dst = 6

Print all Paths from src → Destination



Ar< Ar< Pair >>

0: {1,5, 3,10}
1: {0,5}
2: {}
3: {0,10}
4: {}
5: {}
6: {}

DFS + Path
BFS

0 3 4 6 ⇒ 17
0 1 2 5 6 ⇒ 25
0 1 2 3 4 5 6 ⇒ 30
0 1 2 3 4 6 @ ⇒ 17