BFS+Union-find+follow up

##Union-find

1.
2.
3.

##(前四个都是BFS+Union-find)

200. Number of Islands
201. Number of Islands II
202. Graph Valid Tree
203. Connected Component in Undirected Graph

> 还有个 130. Surrounded Regions，先拉黑

####200. Number of Islands

1. 选择方法BFS: 每次我遇到1，都bfs遍历所有他相邻的1，得到整个岛，把整个岛归零，岛的个数+1。实现：
    1. func1: go over the island; 遇到1就bfs所有相邻1，算作一个island
    2. func2: bfs each 1 in grid(use queue, coordinate arrys); 不需要size，但需要坐标函数，每个1找相邻上下左右四个方向的1，找到了把1换成0，
    3. func3: if this block is in the grid or out of bound
2. Union-find
    1. initialize father[i] = i;
    2. Union, do not forget count--

```
if (root_a != root_b) {
    father[root_a] = root_b;
    count --;
}
```

3. set count/query count, get total by going over the matrix, count all point==1

```
public int query() {
    return count;
}

public void set_count(int total) {
    count = total;
}
```

4. four directions

```
if (grid[i][j]=='1') {
    if (i > 0 && grid[i - 1][j]=='1') {
        union_find.connect(i * m + j, (i - 1) * m +
j);
    }
    if (i <  n - 1 && grid[i + 1][j]=='1') {
        union_find.connect(i * m + j, (i + 1) * m +
j);
    }
    if (j > 0 && grid[i][j - 1]=='1') {
        union_find.connect(i * m + j, i * m + j - 1);
    }
    if (j < m - 1 && grid[i][j + 1]=='1') {
        union_find.connect(i * m + j, i * m + j + 1);
    }
}
```

####0. Number of Islands II

1. union_find的步骤和代码基本跟上面一样，可以考虑当做模板，上面题的count是额外加的，所以这题 union find这个类里不需要count，但我们也

需要计数

2. 本题计数的逻辑是：每次有operator，count++，当判断当前的1和之前的1可以union成一个island，count--

```
/**
 * Definition for a point.
 * class Point {
 *     int x;
 *     int y;
 *     Point() { x = 0; y = 0; }
 *     Point(int a, int b) { x = a; y = b; }
 * }
 */
public class Solution {
    /**
     * @param n an integer
     * @param m an integer
     * @param operators an array of point
     * @return an integer array
     */
    int converttoId(int x, int y, int m){
        return x*m + y;
    }
    public List<Integer> numIslands2(int n, int m, Point[]
operators) {
        List<Integer> re = new ArrayList<Integer>();
        if(operators == null) {
            return re;
        }

        int[] dx = {0,-1, 0, 1};
        int[] dy = {1, 0, -1, 0};
        int[][] isLand = new int[n][m];
```

```
        UnionFind uf = new UnionFind(n,m);
        int count = 0;
        for(int i = 0; i<operators.length; i++ ){
            count++;
            int curx = operators[i].x;
            int cury = operators[i].y;
            if(isLand[curx][cury]!=1){
                isLand[curx][cury]=1;
                int id = converttoId(curx,cury,m);
                for(int dir = 0; dir<4; dir++){
                    int nextx = curx+dx[dir];
                    int nexty = cury+dy[dir];
                    if(0 <= nextx && nextx < n && 0 <=
nexty && nexty < m && isLand[nextx][nexty] == 1) {
                        int nextID =
converttoId(nextx,nexty,m);

                        int curfa =
uf.compressed_find(id);
                        int nextfa =
uf.compressed_find(nextID);
                        if(curfa!=nextfa){
                            count--;
                            uf.union(id,nextID);
                        }
                    }

                }

            }
            //add count to re when operated
            re.add(count);
        }
        return re;
```

```
    }

    class UnionFind{
        HashMap<Integer, Integer> father = new
HashMap<Integer, Integer>();
        //initialization
        public UnionFind(int n, int m){
            for(int i = 0 ; i < n; i++) {
                for(int j = 0 ; j < m; j++) {
                    int id = converttoId(i,j,m);
                    father.put(id, id);
                }
            }
        }
        //find parent
        int compressed_find(int x){
            int parent =  father.get(x);
            while(parent!=father.get(parent)) {
                parent = father.get(parent);
            }
            int temp = -1;
            int fa = x;
            //update parent for all element in this set
            while(fa!=father.get(fa)) {
                temp = father.get(fa);
                father.put(fa, parent) ;
                fa = temp;
            }
            return parent;

        }
        //union
        void union(int x, int y){
            int fa_x = compressed_find(x);
```

```
            int fa_y = compressed_find(y);
            if(fa_x != fa_y)
                father.put(fa_x, fa_y);
        }
    }
}
```

####1. Graph Valid Tree

1. 题目分析，什么时候graph valid tree
    1. edges = nodes-1
    2. n-1 edges connect the whole tree
2. BFS
    1. initialize graph :Map<Integer, Set<Integer>> graph = initializeGraph(n, edges); 把给定的node和neighbors信息存入这样一个map钟
    2. BFS并同时计数，通过最后counter==n(number of nodes)来判断是否valid
3. Union-find
    1. if (uf.compressed_find(edges[i][0]) == uf.compressed_find(edges[i][1])) return false; //有环，返回false，退出
    2. compressed_find(int x) 这个方法，当x的parent不是x时，找x的parent，是第一个while； 找到后把之前所有跟x在同一set的parent都改成x现在的parent

####2. Connected Component in Undirected Graph

1. leetcode版本：求个数，number of Connected Component。其中n：n points = n islands = n trees = n roots.

```
public int countComponents(int n, int[][] edges) {
    int[] roots = new int[n];
    for(int i = 0; i < n; i++) roots[i] = i;
```

```
    for(int[] e : edges) {
        int root1 = find(roots, e[0]);
        int root2 = find(roots, e[1]);
        if(root1 != root2) {
            roots[root1] = root2;  // union
            n--;
        }
    }
    return n;
}

public int find(int[] roots, int id) {
    while(roots[id] != id) {
        roots[id] = roots[roots[id]];  // optional: path
compression
        id = roots[id];
    }
    return id;
}
```

2. lintcode版本，要输出所有结果的，用bfs，（用union-find的话输出不太方便）每个节点bfs可以找到他所在的component。 很正常的bfs，这里用一个visited map来记录那些节点被访问过了。但答案里有一点很奇怪，排序了

##单纯的BFS

####127. Word Ladder

1. 题目解释和input：Given two words (beginWord and endWord), and a dictionary's word list, find the length of shortest transformation sequence from beginWord to endWord。Notice：
    1. Only one letter can be changed at a time.
    2. Each transformed word must exist in the word list.
    3. input wordList is List instead of set
2. 代码注意：

1. convert list to set to keep non-duplicates
2. bfs: while(){len++;...}; two for-loops: size, validNexts; another hashSet to record all words have visited
3. validNext, replace all characters in a word one by one by replace the char from 'a' to 'z' for(char c = 'a'; c<='z'; c++){... a special for loop,
4. replace a letter

####133. Clone Graph

1. get all nodes (by BFS)
2. mapping old nodes to new
3. set all neighbors, all nodes have been newed when constructed mapping, so all newNodes, newNighbors do not need to be redeclare again, just mapping.get(), we get the new node
4. return mapping.get(node); 这个return看起来很正常，但是刚看到题的时候有点懵逼，不知道该返回什么

####490. the maze 给一个maze，从起点走到终点，注意一点是选定一个方向之后撞墙才停， 返回boolean判断能否到达destination。可以用bfs和dfs，这里选择bfs。因为是付费题，所以直接从discuss区拿了答案来。

(topological sort) ####207. Course Schedule

1. initialization

```
 //idx of edges: node, element(arraylist) of edges:
node.neighbors
int[] indegree = new int[numCourses];
List[] edges = new List[numCourses];
// initialization: each node has a list to store its
neighbors
for (int i = 0;i < numCourses; i++){
    edges[i] = new ArrayList<Integer>();
}
```

2. compute indegree

```
//eg:pair[0,1] means to take course 0 you have to first
take course 1,0-indegree=1
for(int i = 0; i<prerequisites.length; i++){
    indegree[prerequisites[i][0]]++;
    edges[prerequisites[i][1]].add(prerequisites[i][0]);
}
```

3. deal with nodes, whoes indegree==0

```
//add all nodes with indegree==0, no prerequisites needed
Queue<Integer> queue = new LinkedList();
for(int i = 0; i < indegree.length; i++){
    if (indegree[i] == 0) {
        queue.add(i);
    }
}
```

4. bfs

```
//bfs from nodes above, update indegree once visited
int count = 0;
while(!queue.isEmpty()){
    int course = queue.poll();
    count ++;
    int n = edges[course].size();
    for(int i = 0; i < n; i++){
        int pointer = (int)edges[course].get(i);
        indegree[pointer]--;
        if (indegree[pointer] == 0) {
            queue.add(pointer);
        }
    }
```

```
}
```

5. return

```
//if it has cycle, there will be some node with indegree>0
left, return false, and count<numCourse
return count == numCourses;
```

####210. Course Schedule II 跟上体基本一样，两点区别

1. 需要一个array来存储每次count计数时对应的course
2. return count == numCourses ? order: new int[]{};

####310. Minimum Height Trees 是个不会的题，找高度最小的树，返回这些树的root，一开始想的是从leaf开始bfs。后来看代码确实bfs思想，算了一遍可以理解代码，但不太理解思路

```
public List<Integer> findMinHeightTrees(int n, int[][]
edges) {
    // if (n == 1) return Collections.singletonList(0);

    List<Integer> leaves = new ArrayList<>();
    if (n==1) {
                leaves.add(0);
                return leaves;
        }
        //idx: node, element-set: neighbors
    List<Set<Integer>> adj = new ArrayList<>(n);
    for (int i = 0; i < n; i++) {
        adj.add(new HashSet<>());
    }
    for (int[] edge : edges) {
        adj.get(edge[0]).add(edge[1]);
        adj.get(edge[1]).add(edge[0]);
```

```
    }
    //only one neighbors --> leaves
    for (int i = 0; i < n; i++) {
        if (adj.get(i).size() == 1){
            leaves.add(i);
        }
    }
    //核心思路在这个while里，一个node的所有edge相连node都排除之后还剩一条edge连另一个node，这个就可以当做min height tree的root了
    while (n > 2) {
        //n = nodes - leaves
        n -= leaves.size();
        List<Integer> newLeaves = new ArrayList<>();
        for (int i : leaves) {
            int j = adj.get(i).iterator().next();
            adj.get(j).remove(i);
            if (adj.get(j).size() == 1) {
                newLeaves.add(j);
            }
        }
        leaves = newLeaves;
    }
    return leaves;
}
```