BFS Practice

- ## Leetcode 542 01Matrix

Hint:

- 先将非0数转化为max，通过queue的顺序修改周边元素，若新数值小于原来数值则进行修改
- 将附近元素可能性用数组表示

```
class Solution {
    public int[][] updateMatrix(int[][] matrix) {
        Queue<int []> queue = new LinkedList<>();
        for(int row = 0; row < matrix.length; row++){
            for(int col = 0; col < matrix[0].length; col++){
                if(matrix[row][col] == 0){
                    queue.add(new int[] {row, col});
                }
                else{
                    matrix[row][col] = Integer.MAX_VALUE;
                }
            }
        }

        int dir[][] = {{-1, 0}, {1, 0}, {0 , -1}, {0, 1}};
        while(!queue.isEmpty()){
            int[] temp = queue.poll();
            for(int[] diff: dir){
                int row = temp[0] + diff[0];
                int col = temp[1] + diff[1];
                if(row < 0 || row >= matrix.length || col < 0 || col >= matrix[0].length){
                    continue;
                }
                int formervalue = matrix[row][col];
                int newvalue = matrix[temp[0]][temp[1]] + 1;
                if(formervalue <= newvalue){
                    continue;
                }
                else{
                    matrix[row][col] = newvalue;
                    queue.add(new int[]{row, col});
                }
            }
        }
        return matrix;
    }
}
```

- ## 103. Binary Tree Zigzag Level Order Traversal

Hint:

- BFS典型套路
- 设置状态判断位是从左到右还是从右到左

```
class Solution {
    public List<List<Integer>> zigzagLevelOrder(TreeNode root) {
        List<List<Integer>> ans = new ArrayList<>();
        if(root == null){
            return ans;
        }
        List<TreeNode> queue = new ArrayList<>();
        boolean startFromLeft = true;
        queue.add(root);
        while(!queue.isEmpty()){
            int size = queue.size();
            List<Integer> list = new ArrayList<>();
            for(int i = 0; i < size; i++){
                TreeNode node = queue.get(0);
                queue.remove(0);
                if(node.left != null){
                    queue.add(node.left);
                }
                if(node.right != null){
                    queue.add(node.right);
                }
                if(startFromLeft){
                    list.add(node.val);
                }
                if(!startFromLeft){
                    list.add(0, node.val);
                }
            }
            startFromLeft = !startFromLeft;
            ans.add(list);
        }
        return ans;
    }
}
```

- ## 127. Word Ladder

Hint:
1.用set记录是否路径
2.BFS经典套路，队列每一次循环该层的数量
3.用两个子函数去获取下一层的list 和 构建新string

```
class Solution {
    public int ladderLength(String beginWord, String endWord, List<String> wordList) {
        if(wordList.size() == 0){
            return 0;
        }
        if(beginWord.equals(endWord)){
            return 1;
        }

        Queue<String> queue = new LinkedList<>();
        Set<String> set = new HashSet<>();

        set.add(beginWord);
        queue.add(beginWord);
        int length = 1;
        while(!queue.isEmpty()){
            length++;
            int size = queue.size();
            for(int i = 0; i < size; i++){
                String tempWord = queue.poll();
                for(String temp: nextWordList(tempWord, wordList)){
                    if(set.contains(temp)){
                        continue;
                    }
                    if(temp.equals(endWord)){
                        return length;
                    }
                    set.add(temp);
                    queue.add(temp);
                }
            }
        }
        return 0;
    }

    private List<String> nextWordList (String word, List<String> wordList){
        List<String> ans = new ArrayList<>();
        for(int i = 0; i < word.length(); i++){
            for(char j = 'a'; j <= 'z'; j++){
                if(j == word.charAt(i)){
                    continue;
                }
                String newWord = replace(word, i, j);
                if(wordList.contains(newWord)){
                    ans.add(newWord);
                }
            }
        }
        return ans;
    }

    private String replace(String word, int pos, char letter){
        char[] ch = word.toCharArray();
        ch[pos] = letter;
        return new String(ch);
    }
}
```

- ## 133.Clone Graph

Hint:
1.用HashMap记录新旧结点
2.记录完以后分别对各个node克隆neighbors

```
public class Solution {
    public UndirectedGraphNode cloneGraph(UndirectedGraphNode node) {
        if(node == null){
            return null;
        }
        Map<UndirectedGraphNode, UndirectedGraphNode> map = new HashMap<>();
        Queue<UndirectedGraphNode> queue = new LinkedList<>();
        queue.add(node);
        while(!queue.isEmpty()){
            UndirectedGraphNode tempnode = queue.poll();
            if(!map.containsKey(tempnode)){
                UndirectedGraphNode newnode = new UndirectedGraphNode(tempnode.label);
                map.put(tempnode, newnode);
            }
            for(UndirectedGraphNode neighbor: tempnode.neighbors){
                if(!map.containsKey(neighbor)){
                    queue.add(neighbor);
                }

            }
        }

        for(UndirectedGraphNode tempnode: map.keySet()){
            UndirectedGraphNode valnode = map.get(tempnode);
            for(UndirectedGraphNode neighbor: tempnode.neighbors){
                valnode.neighbors.add(map.get(neighbor));
            }
        }
        return map.get(node);
    }
}
```

- ## 279. Perfect Squares

```
class Solution {
    public int numSquares(int n) {
        if(n <= 1){
            return n;
        }
        int ans = 0;
        Queue<Integer> queue = new LinkedList<>();
        queue.add(n);

        while(!queue.isEmpty()){
            ans++;
            int size = queue.size();
            for(int i = 0; i < size; i++){
                int node = queue.poll();
                for(int j = 1; j * j <= node; j++){
                    int res = node - j * j;
                    if(res == 0){
                        return ans;
                    }
                    queue.add(res);
                }
            }
        }
        return ans;
    }
}
```