17. Letter combination of a phone number ---- ---- ---- Medium ---- ---- ---- DFS(subset)

Given a digit string, return all possible letter combinations that the number could represent.

A mapping of digit to letters (just like on the telephone buttons) is given below.

class Solution {

String[] dicts = new String[]{"","","abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"};

public List<String> letterCombinations(String digits) {

ArrayList<String> result = new ArrayList<>();

if(digits.length() == 0){

return result;

}

String s = "";

int level = 0;

helper(digits, level, s, result);

return result;

}

private void helper(String digits, int level, String sub, ArrayList<String> result){

if(level == digits.length()){

result.add(sub);

return;

}

int index = digits.charAt(level) - '0';

for(int i = 0; i < dicts[index].length(); i++){

sub += dicts[index].charAt(i);

helper(digits,level + 1, sub, result);

sub = sub.substring(0, sub.length() - 1);

}

}

}

158. Read N Characters Given Read4 II - Call multiple times

The API: int read4(char \*buf) reads 4 characters at a time from a file.

The return value is the actual number of characters read. For example, it returns 3 if there is only 3 characters left in the file.

By using the read4 API, implement the function int read(char \*buf, int n) that reads n characters from the file.

public class Solution extends Reader4 {

private char[] buf4 = new char[4];

private int buf4Index = 4;

private int buf4Size = 4;

private boolean readNext(char[] buf, int index) {

if (buf4Index >= buf4Size) {

buf4Size = read4(buf4);

buf4Index = 0;

if (buf4Size == 0) {

return false;

}

}

buf[index] = buf4[buf4Index++];

return true;

}

public int read(char[] buf, int n) {

for (int i = 0; i < n; i++) {

if (!readNext(buf, i)) {

return i;

}

}

return n;

}

}

200. Number of Islands ---- ---- ---- BFS (connected components)

class Solution {

private class Point{

int x;

int y;

public Point(int x, int y){

this.x = x;

this.y = y;

}

}

public int numIslands(char[][] grid) {

if(grid.length == 0 || grid[0].length == 0){

return 0;

}

int result = 0;

for(int i = 0; i < grid.length; i++){

for(int j = 0; j < grid[0].length; j++){

if(grid[i][j] == '1'){

//System.out.println(i + " " + j);

findIsland(grid, i, j);

result++;

}

}

}

return result;

}

private void findIsland(char[][] grid, int x, int y){

int[] deltaX = new int[]{-1, 1, 0, 0};

int[] deltaY = new int[]{ 0, 0, -1,1};

Queue<Point> queue = new LinkedList<>();

queue.offer(new Point(x,y));

grid[x][y] = '0';

while(!queue.isEmpty()){

Point p = queue.poll();

for(int i = 0; i < 4; i++){

int newX = p.x + deltaX[i];

int newY = p.y + deltaY[i];

if(inBound(newX, newY, grid.length, grid[0].length)){

if(grid[newX][newY] == '1'){

queue.offer(new Point(newX, newY));

grid[newX][newY] = '0';

System.out.println(grid[newX][newY]);

}

}

}

}

}

private boolean inBound(int x, int y, int rows, int columns){

//System.out.println(x+" "+y);

return x >= 0 && x < rows && y >= 0 && y < columns;

}

}