64. Minimum Path Sum

题目：
1.求出从矩阵的左上到右下所需要的weight最小的值
Hint：
1.DP
2.当row或者col等于0的情况需要另外操作

```
class Solution {
    public int minPathSum(int[][] grid) {
        int dp[][] = new int [grid.length][grid[0].length];
        dp[0][0] = grid[0][0];
        for(int i = 0; i < grid.length; i++){
            for(int j = 0; j < grid[0].length; j++){
                if(i == 0 && j== 0){
                    continue;
                } else if(i == 0){
                    dp[0][j] = dp[0][j - 1] + grid[0][j];
                } else if(j == 0){
                    dp[i][0] = dp[i - 1][0] + grid[i][0];
                } else{
                    dp[i][j] = Math.min(dp[i][j - 1], dp[i - 1][j]) + grid[i][j];
                }

            }
        }
        return dp[grid.length - 1][grid[0].length - 1];
    }
}
```

139. Word Break

题目：
分割单词，检查每个分割部分是否都存在dict里
Hint：
1.Backtracking:(Time Limit Exceed)
2.DP
用array记录从0到该位数是否可split，并检查剩下的substring是否在dict中存在
3.BFS
必须用array来记录每个位置作为断点是否可行，不然会造成TLE

```
class Solution {
    public boolean wordBreak(String s, List<String> wordDict) {
        return dfs(s, 0, new HashSet<>(wordDict));

    }

    private boolean dfs(String s, int pos, Set<String> wordDict ){
        if(pos == s.length()){
            return true;
        }
        for(int i = pos + 1; i <= s.length(); i++){
            if(wordDict.contains(s.substring(pos, i)) && dfs(s, i, wordDict)){
                return true;
            }
        }
        return false;
    }
}
```

```
public boolean wordBreak(String s, List<String> wordDict) {
        boolean[] canSplit = new boolean [s.length() + 1];
        canSplit[0] = true;
        for(int i = 1; i <= s.length(); i++){
            for(int j = 0; j < i; j++){
                if(canSplit[j] && wordDict.contains(s.substring(j, i))){
                    canSplit[i] = true;
                    break;
                }
            }
        }
        return canSplit[s.length()];
    }
```

```
class Solution {
    public boolean wordBreak(String s, List<String> wordDict) {
        Set<String> dict = new HashSet<>(wordDict);
        Queue<Integer> queue = new LinkedList<>();
        queue.add(0);
        boolean[] visited = new boolean[s.length()];
        while(!queue.isEmpty()){
            int start = queue.poll();
            if(!visited[start]){
                for(int i = start + 1; i <= s.length(); i++){
                    if(dict.contains(s.substring(start, i))){
                        queue.add(i);
                        if(i == s.length()){
                            return true;
                        }
                    }

                }
                visited[start] = true;
            }
        }
        return false;
    }
}
```