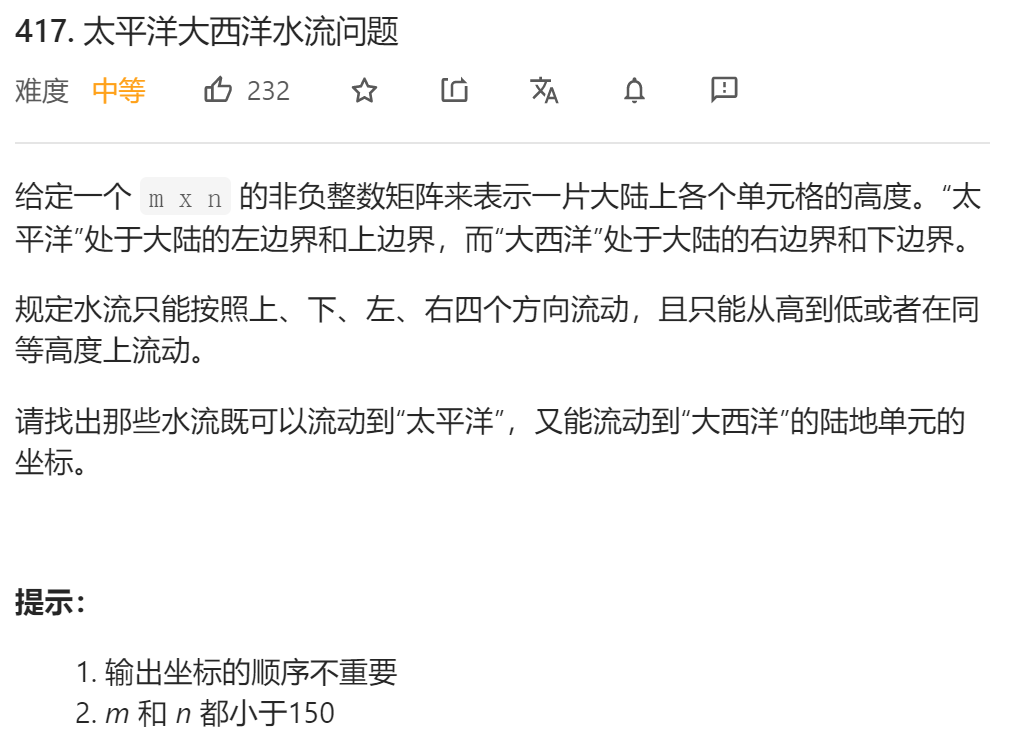
# 深度优先算法【可能出现无线递归】

## 题目描述：



## 可能出现无限递归的代码

class Solution {  
 private:  
 //0表示未搜索，1表示可以仅可以流入太平洋  
 //2表示仅可以流入大西洋，3表示两个洋都可以流入  
 vector<vector<int>>visited;  
 vector<vector<int>>res;  
 int m,n;  
  
 vector<bool>dfs(vector<vector<int>>& heights,int x,int y){  
 vector<bool>ret(2,false);  
 //终止条件  
 if(x<0||y<0){  
 ret[0]=true;  
 return ret;  
 }else if(x>=m||y>=n){  
 ret[1]=true;  
 return ret;  
 }  
 if(visited[x][y]==1){  
 ret[0]=true;  
 return ret;  
 }else if(visited[x][y]==2){  
 ret[1]=true;  
 return ret;  
 }else if(visited[x][y]==3){  
 ret[0]=ret[1]=true;  
 return ret;  
 }  
  
 //if（visited[x][y]==0)  
 vector<bool>up(2,false);  
 vector<bool>down(2,false);  
 vector<bool>left(2,false);  
 vector<bool>right(2,false);  
 if(x+1>=m||heights[x][y]>=heights[x+1][y])  
 up=dfs(heights,x+1,y);  
 if(x-1<0||heights[x][y]>=heights[x-1][y])  
 down=dfs(heights,x-1,y);  
 if(y-1<0||heights[x][y]>=heights[x][y-1])  
 left=dfs(heights,x,y-1);  
 if(y+1>=n||heights[x][y]>=heights[x][y+1])  
 right=dfs(heights,x,y+1);  
  
 ret[0]=up[0]||down[0]||left[0]||right[0];  
 ret[1]=up[1]||down[1]||left[1]||right[1];  
 if(ret[0]&&ret[1]){  
 visited[x][y]=3;  
 }else if(ret[1]==true){  
 visited[x][y]=2;  
 }else if(ret[0]==true){  
 visited[x][y]=1;  
 }  
 if(ret[0]&&ret[1]){  
 vector<int>ans(2,0);  
 ans[0]=x;  
 ans[1]=y;  
 res.push\_back(ans);  
 }  
 return ret;  
 }  
public:  
 vector<vector<int>> pacificAtlantic(vector<vector<int>>& heights) {  
 //if(heights.size()==0||heights[0].size()==0)  
 //return res;  
 m=heights.size();  
 n=heights[0].size();  
 visited.resize(m,vector<int>(n,0));  
 //改一下遍历循序，从外而内  
 //防止出现无限递归  
 for(int i=0;i<m;i++){  
 for(int j=0;j<n;j++){  
 if(!visited[i][j])  
 dfs(heights,i,j);  
 }  
 }  
   
 return res;  
 }  
};

## 未完待续

class Solution {  
public:  
const int dir[5]={-1,0,1,0,-1};  
 vector<vector<int>> pacificAtlantic(vector<vector<int>>& heights) {  
 vector<vector<int>> res;  
 if(heights.empty() || heights[0].empty()) return res;  
 vector<vector<int>> visited(heights.size(),vector<int>(heights[0].size(),0));  
 const int a=1,p=2,m=heights.size(),n=heights[0].size();  
 for(int i=0;i<n;++i)  
 {  
 visited[0][i]|=a;  
 visited[m-1][i]|=p;  
 dfs(heights,visited,0,i);  
 dfs(heights,visited,m-1,i);  
 }  
 //cout<<"?";  
 for(int i=0;i<m;++i)  
 {  
 visited[i][0]|=a;  
 visited[i][n-1]|=p;  
 dfs(heights,visited,i,0);  
 dfs(heights,visited,i,n-1);  
 }  
 for(int row = 0; row<visited.size(); ++row)  
 {  
 for(int col = 0;col<visited[0].size(); ++col)  
 {  
 //cout<<visited[row][col]<<" ";  
 if(visited[row][col]==3) res.push\_back({row,col});  
 }  
 // cout<<endl;  
 }  
 return res;  
 }  
 void dfs(const vector<vector<int>> &heights,vector<vector<int>> & visited,int row,int col)  
 {  
 for(int i = 0; i<4; ++i)  
 {  
 int r = row + dir[i];  
 int c = col + dir[i+1];  
 if(r>=0 && r<heights.size() && c>=0 && c<heights[0].size())  
 {  
 if(visited[r][c]!=visited[row][col] && heights[r][c]>=heights[row][col])  
 {  
 visited[r][c]|=visited[row][col];  
 dfs(heights,visited,r,c);  
 }  
 }  
 }  
 }  
};