在二维地图上， `0`代表海洋， `1`代表陆地，我们最多只能将一格 `0` 海洋变成 `1`变成陆地。

进行填海之后，地图上最大的岛屿面积是多少？（上、下、左、右四个方向相连的 `1` 可形成岛屿）

示例 1:

输入： `[[1, 0], [0, 1]]`
输出： `3`
解释： 将一格`0`变成`1`，最终连通两个小岛得到面积为 `3` 的岛屿。

示例 2:

输入： `[[1, 1], [1, 0]]`
输出： `4`
解释： 将一格`0`变成`1`，岛屿的面积扩大为 `4`。

示例 3:

输入： `[[1, 1], [1, 1]]`
输出： `4`
解释： 没有`0`可以让我们变成`1`，面积依然为 `4`。

说明:

- `1 <= grid.length = grid[0].length <= 50`
- `0 <= grid[i][j] <= 1`

思路，个人的解决想法，先遍历一遍，把所有岛都统一标记出来，并将标记和岛屿面积做好映射，然后进行第二次遍历，每个为海的点都能进行一次转换陆地的尝试，并将转换陆地后的岛屿面积统计出来，和最大的岛屿面积进行比较。需要注意的是，在统计周围岛屿面积时，不要把相同的岛屿重复统计，我这里是用map处理，将已连接的岛屿用map标记出来，因为map在循环中，所以寿命为当前循环，不需要管别的。

```
class Solution {
public:
    int dfs(vector<vector<int>>& grid,int i,int j,int wz){
        int num=0;
        grid[i][j]=wz;
        if(i>0&&grid[i-1][j]==1){
            num+=dfs(grid,i-1,j,wz);
        }
        if(i<grid.size()-1&&grid[i+1][j]==1){
            num+=dfs(grid,i+1,j,wz);
```

```
    }
    if(j>0&&grid[i][j-1]==1){
        num+=dfs(grid,i,j-1,wz);
    }
    if(j<grid[0].size()-1&&grid[i][j+1]==1){
        num+=dfs(grid,i,j+1,wz);
    }
    return 1+num;
}
int largestIsland(vector<vector<int>>& grid) {
    map<int,int> area;
    int m=grid.size();
    int n=grid[0].size();
    int max=0;
    for(int i=0,wz=2;i<m;i++){
        for(int j=0;j<n;j++){
            if(grid[i][j]==1){
                area[wz]=dfs(grid,i,j,wz);
                if(area[wz]>max){
                    max=area[wz];
                }
                wz++;
            }
        }
    }
    for(int i=0;i<m;i++){
        for(int j=0;j<n;j++){
            if(grid[i][j]==0){
                int num=1;
                map<int,bool> flag;
                if(i>0){
                    num+=area[grid[i-1][j]];
                    flag[grid[i-1][j]]=true;
                }
                if(i<m-1&&!flag[grid[i+1][j]]){
```

```
                    num+=area[grid[i+1][j]];
                    flag[grid[i+1][j]]=true;
                }
                if(j>0&&!flag[grid[i][j-1]]){
                    num+=area[grid[i][j-1]];
                    flag[grid[i][j-1]]=true;
                }
                if(j<n-1&&!flag[grid[i][j+1]]){
                    num+=area[grid[i][j+1]];
                    flag[grid[i][j+1]]=true;
                }
                if(num>max){
                    max=num;
                }
            }
        }
    }
    return max;
  }
};
```