贪心02

53. 最大子序和

题目

中等 30min
给你一个整数数组 `nums` ，请你找出一个具有最大和的连续子数组（子数组最少包含一个元素），返回其最大和。
子数组是数组中的一个连续部分。

示例 1：

```
输入：nums = [-2,1,-3,4,-1,2,1,-5,4]
输出：6
解释：连续子数组 [4,-1,2,1] 的和最大，为 6 。
```

示例 2：

```
输入：nums = [1]
输出：1
```

示例 3：

```
输入：nums = [5,4,-1,7,8]
输出：23
```

提示：

- `1 ⩽ nums.length ⩽ 105`
- `-104 ⩽ nums[i] ⩽ 104`

题解

很显然，两层`for`就可以做，但是这样可能会超时吧，没试过hh
这里用贪心就是维持当前区间和
全局最优：选取最大“连续和”
局部最优的情况下，并记录最大的“连续和”，可以推出全局最优。
从代码角度上来讲：遍历$nums$，从头开始用$count$ 累积，如果 E一旦加上 nums[i] 变为负数，那么就应该从 $nums[i+1]$开始从0 累积 $count$了，因为已经变为负数的 $count$，只会拖累总和。
这相当于是暴力解法中的不断调整最大子序和区间的起始位置。

```cpp
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int result = INT32_MIN;//记录最后的结果，为了避免数组里全是负数，所以设为负无穷
        int count = 0;//记录当前的区间和，因为谁也不知道后面后不会更大，所以即使现在小也不能丢弃
        int n=nums.size();
        for(int i=0;i<n;i++){
            count += nums[i];//加上当前的值
            result = max(count,result);//如果结果值更大则到当前这个区间
            if(count<0)count=0;// 如果小于0则说明这段区间没用了，无论后面是什么加上都不会更大，所以从0开始
        }
        return result;
    }
};
```

122.买卖股票的最佳时机 II

[力扣题目链接]
中等 15min

题目

给你一个整数数组 `prices` ，其中 `prices[i]` 表示某支股票第 `i` 天的价格。
在每一天，你可以决定是否购买和/或出售股票。你在任何时候 **最多** 只能持有 **一股** 股票。你也可以先购买，然后在 **同一天** 出售。
返回 *你能获得的 **最大** 利润* 。

示例 1:

```
输入：prices = [7,1,5,3,6,4]
输出：7
解释：在第 2 天（股票价格 = 1）的时候买入，在第 3 天（股票价格 = 5）的时候卖出，这笔交易所能获得利润 = 5 - 1 = 4。
随后，在第 4 天（股票价格 = 3）的时候买入，在第 5 天（股票价格 = 6）的时候卖出，这笔交易所能获得利润 = 6 - 3 = 3。
最大总利润为 4 + 3 = 7 。
```

示例 2:

```
输入：prices = [1,2,3,4,5]
输出：4
解释：在第 1 天（股票价格 = 1）的时候买入，在第 5 天 （股票价格 = 5）的时候卖出，这笔交易所能获得利润 = 5 - 1 = 4。
最大总利润为 4 。
```

示例 3:

```
输入：prices = [7,6,4,3,1]
输出：0
解释：在这种情况下，交易无法获得正利润，所以不参与交易可以获得最大利润，最大利润为 0。
```

提示:

- `1 ⩽ prices.length ⩽ 3 * 104`
- `0 ⩽ prices[i] ⩽ 104`

题解

最好的办法就是举个例子试试，试了之后发现其实我们需要收集每天的正利润就可以，**收集正利润的区间，就是股票买卖的区间，而我们只需要关注最终利润，不需要记录区间。**
局部最优：收集正利润，全局最优：求得最大利润。

```cpp
class Solution {
public:
    int maxProfit(vector<int>& prices) {
        int n=prices.size();
        int dif=0,w=0;
        for(int i=1;i<n;i++)
        {
            dif=prices[i]-prices[i-1];
            if(dif>0)w+=dif;
        }
        return w;
    }
};
```

55. 跳跃游戏

[力扣题目链接]
中等

题目

给你一个非负整数数组 `nums` ，你最初位于数组的 **第一个下标** 。数组中的每个元素代表你在该位置可以跳跃的最大长度。
判断你是否能够到达最后一个下标，如果可以，返回 `true` ；否则，返回 `false` 。

示例 1：

```
输入：nums = [2,3,1,1,4]
输出：true
解释：可以先跳 1 步，从下标 0 到达下标 1，然后再从下标 1 跳 3 步到达最后一个下标。
```

示例 2：

```
输入：nums = [3,2,1,0,4]
输出：false
解释：无论怎样，总会到达下标为 3 的位置。但该下标的最大跳跃长度是 0 ， 所以永远不可能到达最后一个下标。
```

提示：

- `1 ⩽ nums.length ⩽ 104`
- `0 ⩽ nums[i] ⩽ 105`

题解

我最初的思路是从后往前推，从最后一个区间内找到能不能跳出这个区间的，然后一直往前延申，直到到达最远处，看看能否覆盖起点
但是转念一想，不对啊，这不是彻底把起点和终点掉了个，完全没有这个必要啊，同样是找覆盖范围，从起点来就好了嘛
所以这道题可以看成跳几步无所谓，只要范围覆盖到了就成
就是说不一定非要明确一次究竟跳几步，每次取最大的跳跃步数，这个就是可以跳跃的覆盖范围。
每移动一个单位，就更新最大覆盖范围。
贪心算法局部最优解：每次取最大跳跃步数（取最大覆盖范围），整体最优解：最后得到整体最大覆盖范围，看是否能到终点。

```cpp
class Solution {
public:
    bool canJump(vector<int>& nums) {
        int n=nums.size();
        int cov=0;//用cov表示当前可以覆盖到的位置
        for(int i=0;i<=cov;i++){//只能在覆盖到的位置里面延申，所以这里是小于cov
            if(cov>=n-1)return true;//先判断，防止数组越界，n-1是因为数组下标从0开始的嘛
            cov = max(cov,i+nums[i]);//看看覆盖范围有没有变大
        }
        return false;
    }
};
```

DFS

547.省份数量

中等 20min
力扣题目链接

题目

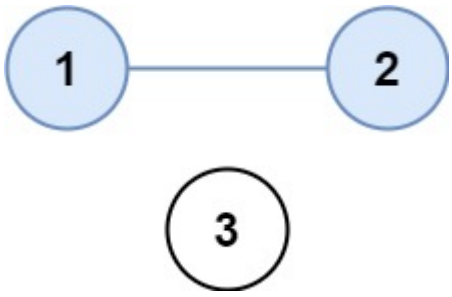
有 `n` 个城市，其中一些彼此相连，另一些没有相连。如果城市 `a` 与城市 `b` 直接相连，且城市 `b` 与城市 `c` 直接相连，那么城市 `a` 与城市 `c` 间接相连。

省份 是一组直接或间接相连的城市，组内不含其他没有相连的城市。

给你一个 `n x n` 的矩阵 `isConnected` ，其中 `isConnected[i][j] = 1` 表示第 `i` 个城市和第 `j` 个城市直接相连，而 `isConnected[i][j] = 0` 表示二者不直接相连。

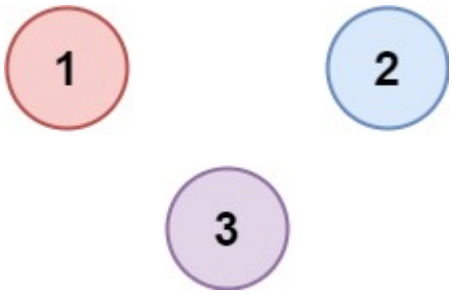
返回矩阵中 **省份** 的数量。

示例 1:




```
输入：isConnected = [[1,1,0],[1,1,0],[0,0,1]]
输出：2
```

示例 2:




```
输入：isConnected = [[1,0,0],[0,1,0],[0,0,1]]
输出：3
```

提示:

- `1 ⩽ n ⩽ 200`
- `n == isConnected.length`
- `n == isConnected[i].length`
- `isConnected[i][j]` 为 `1` 或 `0`
- `isConnected[i][i] == 1`
- `isConnected[i][j] == isConnected[j][i]`

题解

好经典的问题，搜加计数，不知道为啥主人让我做哎hh，就当复习一下dfs了，不知道为啥感觉dfs特别的美妙，你说这玩意谁发明的

```
class Solution {
public:
    void dfs(vector<vector<int>>&
isConnected,vector<int>& visit,int i,int n)
    {
        visit[i]=1;//标记该城市已经访问过
        for(int j=0;j<n;j++){//遍历邻接节点
            if(isConnected[i][j]==1&&visit[j]==0)//两
者相连且该节点没被访问过
                dfs(isConnected,visit,j,n);//搜就完了
        }
        return;
    }
    int findCircleNum(vector<vector<int>>& isConnected)
{
        int n = isConnected.size();
        vector<int> visit(n,0);
        int province=0;
        for(int i=0;i<n;i++){//遍历每个城市
            if(!visit[i]){//如果没访问过
            province++;//好耶开辟新大陆，并且搜爆
            dfs(isConnected,visit,i,n);
            }
        }
        return province;
    }

};
```