【深基16.例1】淘汰赛

题目描述

有 2^n（$n\le7$）个国家参加世界杯决赛圈且进入淘汰赛环节。已经知道各个国家的能力值，且都不相等。能力值高的国家和能力值低的国家踢比赛时高者获胜。1 号国家和 2 号国家踢一场比赛，胜者晋级。3 号国家和 4 号国家也踢一场，胜者晋级……晋级后的国家用相同的方法继续完成赛程，直到决出冠军。给出各个国家的能力值，请问亚军是哪个国家?

输入格式

第一行一个整数 n，表示一共 2^n 个国家参赛。

第二行 2^n 个整数，第 i 个整数表示编号为 i 的国家的能力值（$1\leq i \leq 2^n$）。

数据保证不存在平局。

输出格式

仅一个整数，表示亚军国家的编号。

样例 #1

样例输入 #1

```
3
4 2 3 1 10 5 9 7
```

样例输出 #1

```
1
```

解题思路一：

对于每个输入，它都是一个偶数的数组，所以它一定会存在当前下标与下标+1进行比较，每次取最大值，直到最后一定会有剩下两个人的时候，这个时候我们退出比较，选择其中的较小值，并且获得其中的对于国家的编号就可以了

```
#include <iostream>
#include <vector>
#include <cmath>
using namespace std;

class Solution
{
public:
    // 获取亚军的编号
    int GetSecond(vector<int>& nums)
    {
        int n = nums.size();
```

```
        if (n < 2) return -1; // 至少需要两个国家才能有亚军
        vector<pair<int, int>> array; // pair: {能力值, 编号}
        for (int i = 0; i < n; ++i)
        {
            array.push_back({ nums[i], i + 1 });
        }
        while (array.size() > 2)
        {
            array = cmptwo(array);
        }
        // 最后剩下两支队伍，取能力值较低的作为亚军
        int second = (array[0].first > array[1].first) ? array[1].second :
array[0].second;
        return second;
    }

private:
    vector<pair<int, int>> cmptwo(vector<pair<int, int>>& array)
    {
        vector<pair<int, int>> ans;
        for (size_t i = 0; i < array.size(); i += 2)
        {
            if (array[i].first > array[i + 1].first)
            {
                ans.push_back(array[i]);
            }
            else
            {
                ans.push_back(array[i + 1]);
            }
        }
        return ans;
    }
};

int main()
{
    int n;
    cin >> n;
    int num_teams = pow(2, n);
    vector<int> test(num_teams);
    for (int i = 0; i < num_teams; ++i)
    {
        cin >> test[i];
    }
    Solution solution;
    int ans = solution.GetSecond(test);
    cout << ans << endl;
    return 0;
}
```

解题思路二：

二分。由于输入的数据一定是偶数，那么它一定可以均匀地分成两部分，即左半部分和右半部分，最后我们只要比较两个部分的最大值，选取最大值较小的那个作为季军输出即可

```cpp
#include <bits/stdc++.h>

using namespace std;

//用结构体来存储国家信息
struct gj {
    int hm;//号码
    int nl=0;//能力值
};

int main() {
    int n;
    gj max_l, max_r;//左半边最强，右半边最强
    gj a;//读入时的临时变量a

    cin >> n;

    //找左半边最强
    for (int i = 0; i < 1<<(n-1); i++) {
        cin >> a.nl;
        if (a.nl > max_l.nl) {
            max_l.nl = a.nl;
            max_l.hm = i + 1;
        }
    }

    //找右半边最强
    for (int i = 1<<(n-1); i < 1<<n; i++) {
        cin >> a.nl;
        if (a.nl > max_r.nl) {
            max_r.nl = a.nl;
            max_r.hm = i + 1;
        }
    }

    //输出较弱的号码，即亚军。
    if (max_l.nl > max_r.nl)cout << max_r.hm;
    else cout << max_l.hm;

    return 0;
}
```

【深基16.例3】二叉树深度

题目描述

有一个 $n(n \le 10^6)$ 个结点的二叉树。给出每个结点的两个子结点编号（均不超过 n），建立一棵二叉树（根节点的编号为 1），如果是叶子结点，则输入 `0 0`。

建好这棵二叉树之后，请求出它的深度。二叉树的**深度**是指从根节点到叶子结点时，最多经过了几层。

输入格式

第一行一个整数 n，表示结点数。

之后 n 行，第 i 行两个整数 l、r，分别表示结点 i 的左右子结点编号。若 $l=0$ 则表示无左子结点，$r=0$ 同理。

输出格式

一个整数，表示最大结点深度。

样例 #1

样例输入 #1

```
7
2 7
3 6
4 5
0 0
0 0
0 0
0 0
```

样例输出 #1

```
4
```

解题思路一：

虽然题目中的结构是二叉树，但事实上我们并不需要构建一颗二叉树，定义一个tree类，存储左右子树的值，使用一个hashmap存储对应的下标和树即可，即tree[i]=Tree(left,right)。(tree是一个<int,Tree*>的哈希表);那么从根节点开始计算深度实际上就是一个简单的递归问题，自底向上就可以得到我们需要的结果的值

```
#include <iostream>
#include <vector>
#include <unordered_map>
using namespace std;

class TreeNode {
public:
    int left;
    int right;
    TreeNode(int l = 0, int r = 0) : left(l), right(r) {}
};

unordered_map<int, TreeNode> tree;

// 递归计算二叉树深度
int getDepth(int node) {
    if (node == 0) return 0; // 叶子节点
    return 1 + max(getDepth(tree[node].left), getDepth(tree[node].right));
}

int main() {
    int n;
    cin >> n;
```

```
    // 构建树
    for (int i = 1; i <= n; ++i) {
        int l, r;
        cin >> l >> r;
        tree[i] = TreeNode(l, r);
    }

    // 从根节点开始计算深度
    int depth = getDepth(1);
    cout << depth << endl;

    return 0;
}
```

解题思路二：

动态规划+DFS：事实上，在这份代码在中，我们依然不构建真正意义上的二叉树，而是使用深度优先的搜索方式。对于vectore我们有，使用e[i].push_back(x|y)来存簇某个节点的子节点，当我们需要使用时，我们用e[x][i]表示访问第x个节点的第i个子节点，len是每个节点的长度。那么对于每个节点的深度有，字节点的深度一定是当前节点的深度+1，递归遍历字节点，我们得到节点的深度一定是当前节点与子节点深度的最大值即dp[x]=max(dp[y],y是x的子树)

```
#include<bits/stdc++.h>
using namespace std;
vector<int> e[100005];
int dep[100005],f[100005];
//深度优先
void dfs(int x){
    int len=e[x].size();
    //初始化
    f[x]=dep[x];
    for(int i=0;i<len;++i){
        //子节点的深度等于当前节点的深度+1
        dep[e[x][i]]=dep[x]+1;
        //递归遍历子节点
        dfs(e[x][i]);
        //状态转移方程
        f[x]=max(f[x],f[e[x][i]]);
    }
}
int main(){
    int n,x,y;
    cin>>n;
    for(int i=1;i<=n;++i){
        cin>>x>>y;
        if(x!=0) e[i].push_back(x);
        if(y!=0) e[i].push_back(y);
    }
    dfs(1);
    cout<<f[1]+1;
    return 0;
}
```