这个题很有说法，可以帮你好好理解函数参数前的 & 到底什么意思，也能让你对 dfs 的回溯真正明白。来看你写的这几版代码。首先 pseduo-palin 就是出现奇数次的元素最多一个，偶数次随便。

思路

> 周赛的时候我试了好几遍一直有2个超时，刚刚试了一下把for(auto t: s)里加了&再提交就过了，百度搜了一下
> “值传递时需要将整个变量的值复制一份，而引用传递只需要传递内存地址,避免了复制的开销”，所以看来&速度更快一点。。。

Ver1.

```cpp
void count_1_or_2(TreeNode* root,unordered_map<int,int>place,int&num)
{
    if(root==NULL)return;
    place[root->val]++;
    if(root->left==NULL&&root->right==NULL)
    {
        int ji=0;
        for(auto&m:place)
        {
            if(m.second%2==1)
            {
                ji++;
                if(ji>1)return;
            }
        }
        num++;
    }
    count_1_or_2(root->left,place,num);
    count_1_or_2(root->right,place,num);
}
    int pseudoPalindromicPaths (TreeNode* root) {
        int num=0;
        unordered_map<int,int>place;
        count_1_or_2(root,place,num);
        return num;
    }
```

→到一个地方记录出现次数（`place[root->val]++;`）

如果是叶（`if(root->left==NULL&&root->right==NULL)`）

奇出现2次及以上不行（`if(ji>1)return;`）

行，计数+1（`num++;`）

为什么他 num 加了 & place 不加 & ?

num 加上 &，一方面是函数传来还能把他拉来，他经过一个函数是会变的

比如 在这里 place = {[1,3] [2,4]}, num=3

→经过这个函数 place = {[1,3] [2,4] [3,5]} num=5

→那到这里 place 仍为 {[1,3] [2,4]}

因为回来原本函数，place 无&是复制变量值，回来就是原来的 num 在地址上变，值真变3. to num=5。

这样可以做，但不加&会很久，超时。

Ver2.

与Ver1 唯一区别

```cpp
void count_1_or_2(TreeNode* root,unordered_map<int,int>&place,int&num)
{
    if(root==NULL)return;
    place[root->val]++;
    if(root->left==NULL&&root->right==NULL)
    {
        int ji=0;
        for(auto&m:place)
        {
            if(m.second%2==1)
            {
                ji++;
                if(ji>1)return;
            }
        }
        num++;
    }
    count_1_or_2(root->left,place,num);
    count_1_or_2(root->right,place,num);
}
    int pseudoPalindromicPaths (TreeNode* root) {
        int num=0;
        unordered_map<int,int>place;
        count_1_or_2(root,place,num);
        return num;
    }
```

加了 &，代码错了，为什么？现在到这里 place 也是 了。但我们 place 从这里回来是把整个左支处理完回来了，我需要他进行时是现在的状态而非处理完左。而 num 是贯穿全局最后还要拉出去的 return 的，目的不同

Ver3.

记录目前哪些数出次奇次。现在当前值出现次数+1，他原来奇现就偶，原来偶现就奇。

```
void count_1_or_2(TreeNode* root,vector<int>who_is_ji,int&num)
{
    if(root==NULL)return;
    if(find(who_is_ji.begin(),who_is_ji.end(),root->val)==who_is_ji.end())
    {
        who_is_ji.push_back(root->val);
    }
    else
    {
        who_is_ji.erase(find(who_is_ji.begin(),who_is_ji.end(),root->val));
    }
    if(root->left==NULL&&root->right==NULL)
    {
        if(who_is_ji.size()<=1)num++;
    }
    count_1_or_2(root->left,who_is_ji,num);
    count_1_or_2(root->right,who_is_ji,num);
}
    int pseudoPalindromicPaths (TreeNode* root) {
        int num=0;
        vector<int>who_is_ji;
        count_1_or_2(root,who_is_ji,num);
        return num;
```

放，没找到，加进去

find返回iter，正好是erase要的。

找到了，踢掉。

这个思路比上面好些，vector比map好搞，但是&也慢，但勉强能过。

Ver4

与Ver3区别

```
void count_1_or_2(TreeNode* root,vector<int>&who_is_ji,int&num)
{
    if(root==NULL)return;
    if(find(who_is_ji.begin(),who_is_ji.end(),root->val)==who_is_ji.end())
    {
        who_is_ji.push_back(root->val);
    }
    else
    {
        who_is_ji.erase(find(who_is_ji.begin(),who_is_ji.end(),root->val));
    }
    if(root->left==NULL&&root->right==NULL)
    {
        if(who_is_ji.size()<=1)num++;
    }
    vector<int>bitch;bitch.assign(who_is_ji.begin(),who_is_ji.end());
    count_1_or_2(root->left,who_is_ji,num);
    who_is_ji.assign(bitch.begin(),bitch.end());
    count_1_or_2(root->right,who_is_ji,num);
}
    int pseudoPalindromicPaths (TreeNode* root) {
        int num=0;
        vector<int>who_is_ji;
        count_1_or_2(root,who_is_ji,num);
        return num;
    }
```

就是为了解决 的问题。加上&是为了提速，故出来得让who_is_ji保持不变，找用bitch记住他的值，再变回去，再进去。

就是一种 回溯

需要他回来变为原来样子再给他去做下一步

比Ver3快了很多，但仍很慢，因为他们那帮人是用位运算的XOR完成。有强踢掉 没了补上的操作的。有兴趣看看。