题目描述

输入两个整数序列，第一个序列表示栈的压入顺序，请判断第二个序列是否为该栈的弹出顺序。假设压入栈的所有数字均不相等。例如序列1,2,3,4,5是某栈的压入顺序，序列4，5,3,2,1是该压栈序列对应的一个弹出序列，但4,3,5,1,2就不可能是该压栈序列的弹出序列。

代码(我的):

class Solution {

public:

bool IsPopOrder(vector<int> pushV, vector<int> popV) {

bool mark = true;

if (pushV.size() == 0||popV.size()==0||pushV.size()!=popV.size()) mark = false;

else if (!equal(pushV,popV))mark = false; //元素个数相等,但内容不相等(边界条件)

else check(pushV, popV, &mark);

return mark;

}

bool equal(vector<int> push, vector<int> pop){

bool flag=true;

sort(push.begin(),push.end());

sort(pop.begin(), pop.end());

for (int i = 0; i<push.size(); i++)

{

if (push[i]!=pop[i]){

flag = false;

break;

}

}

return flag;

}

int check(vector<int> push, vector<int> pop, bool \*mark){

int i;

if (pop.size()>2){ //递归出口

if (pop[0] == push[0]){

push.erase(push.begin());//删除首元素

pop.erase(pop.begin());

check(push, pop, mark);

}

else if (pop[0] == push[push.size() - 1]){//第一个出栈和入栈顺序的最后一个元素相等

for (int i = 0; i < push.size(); i++)

{

if (push[i] != pop[push.size() - i - 1])\*mark = false;

}

if (\*mark == false)return 0;

else{

pop.erase(pop.begin());

push.pop\_back();//最后一个元素出栈

check(push, pop, mark);

}

}

else{

vector<int>::iterator iter,begin;//如果是入栈队列中间的某一个，则依次对比匹配

iter = find(push.begin(), push.end(), pop[0]);

for (begin = push.begin(); begin < iter-1; begin++){

if (pop[1] == \*begin){

\*mark = false;

break;

}

}

if (\*mark == false)return 0;

else{

push.erase(iter);

pop.erase(pop.begin());

check(push, pop, mark);

}

}

}

return 0;

// else return 0;

}

};

主要思想:A.出栈元素如果是入栈的最后一个元素，则其他出栈元素与入栈序列刚好向反

B.如果出栈元素是入栈的第一个元素，则立即递归判定

C.如果既不是第一个元素也不是最后一个元素，则当近邻当前出栈元素的下一个出栈元素一定不是入栈队列中该元素之前之前for (begin = push.begin(); begin < iter-1; begin++)的元素，但可以是近邻的前一个元素。所以逐个进行匹配

特殊说明：**如果用else return 0的话函数没有返回值，系统报错**

代码2：

class Solution {

public:

    bool IsPopOrder(vector<int> pushV,vector<int> popV) {

        if(pushV.size()==NULL)

            return false;

        vector<int> stack;

1.popV[j]表示要验证的出栈序列，每次都和入栈元素对比，所有元素依次入栈。

2.如果不等则可能还有元素没进栈，如果全部进栈，而后又不相等则while不执行。

3. j为正确出栈的元素个数(j也可以作为(判断标准)

        for(int i=0,j=0;i<pushV.size();){

            stack.push\_back(pushV[i++]);

            while(j<popV.size()&&stack.back()==popV[j]){

                 stack.pop\_back();

                 j++;

             }

        }

        return stack.empty();

    }

};

算法思想:

1.建立一个临时栈

2.popV中读一个数据，如果等于临时栈的top()，临时栈pop(),同时读popV下一个值

3.如果临时栈为空或者top()值不等于popV读得值，则读pushV中的值，如果相等，同时指向下一个值，不相等则将pushV的值入临时栈，然后读下一个值

4.读到的popV的值不等于临时栈的top()，同时pushV也空了，这时就返回false，否则返回true