

04-E 混洗

#数据结构邓神

什么是栈混洗

栈混洗

❖ 考查栈 $A = \langle a_1, a_2, \dots, a_n \rangle$ 、 $B = S = \emptyset$ //左端为栈顶

❖ 只允许 将A的顶元素弹出并压入S，或 //S.push(A.pop())
将S的顶元素弹出并压入B //B.push(S.pop())

❖ 若经过一系列以上操作后，A中元素全部转入B中

$B = [a_{k1}, \dots, a_{kn}]$ //右端为栈顶

则称之为A的一个栈混洗 (stack permutation)

< 表示栈顶部

] 表示栈底部

S 为中转栈

只允许两种操作

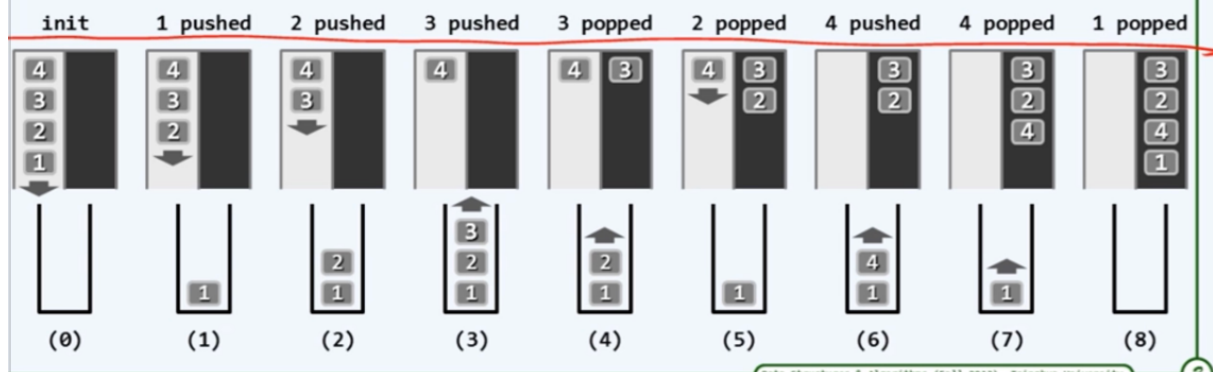
1. 弹出A 轧入S

2. 弹出 S 轧入B

计数

❖ 同一输入序列，可有多种栈混洗

[1, 2, 3, 4 >, [4, 3, 2, 1 >, [3, 2, 4, 1 >,



对于长度为N的序列，可能的栈混洗总数为多少呢？

$SP(N) < N!$ 一定小于全排列

栈混洗

❖ 考查栈 $A = \langle a_1, a_2, \dots, a_n \rangle$ 、 $B = S = \emptyset$

//左端为栈顶

❖ 只允许 将A的顶元素弹出并压入S，或
将S的顶元素弹出并压入B

//S.push(A.pop())

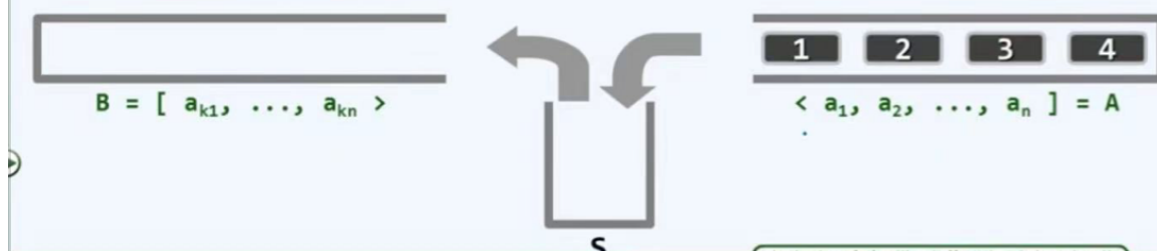
//B.push(S.pop())

❖ 若经过一系列以上操作后，A中元素全部转入B中

$B = \langle a_{k1}, \dots, a_{kn} \rangle$

//右端为栈顶

则称之为A的一个栈混洗 (stack permutation)



$= \text{catalan}(n) = (2n)! / ((n+1)!(n!))$

甄别

$\langle 1, 2, 3, \dots, n \rangle$

任一排列 $\langle p_1, p_2, p_3, \dots, p_n \rangle$

是否为栈混洗

简单情况 $\langle 1, 2, 3 \rangle$

栈混洗 5 种

全排列 6 种

缺少的是 [3,1,2]

为什么是他呢?

禁形

❖ 观察: 任意三个元素能否按某相对次序出现于混洗中, 与其它元素无关 // 故可推而广之...

❖ 对于任何 $1 \leq i < j < k \leq n$, $[\dots, \boxed{k}, \dots, \boxed{i}, \dots, \boxed{j}, \dots]$ 必非栈混洗

充要性

A permutation is a stack permutation iff (if and only if 充分必要条件)

it does NOT involve the permutation

$i < j < k$

也就是说不含 $k i j$ 这种排列

算法

我们可以直接导出一个 $O(n^3)$ 的算法

枚举所有可能

禁形 简化条件

❖ $[p_1, p_2, p_3, \dots, p_n]$ 是 $[1, 2, 3, \dots, n]$ 的栈混洗, 当且仅当
对于任意 $i < j$ 不含模式 $[\dots, \boxed{j+1}, \dots, \boxed{i}, \dots, \boxed{j}, \dots]$

如此可以获得一个 $O(n^2)$ 的算法

借助栈结构简化为 $O(n)$

❖ $O(n)$ 算法: 直接借助栈 A、B 和 S, 模拟混洗过程

// 为何可行

每次 S.pop() 之前, 检测 s 是否已空; 或需弹出的元素在 s 中, 却非顶元素

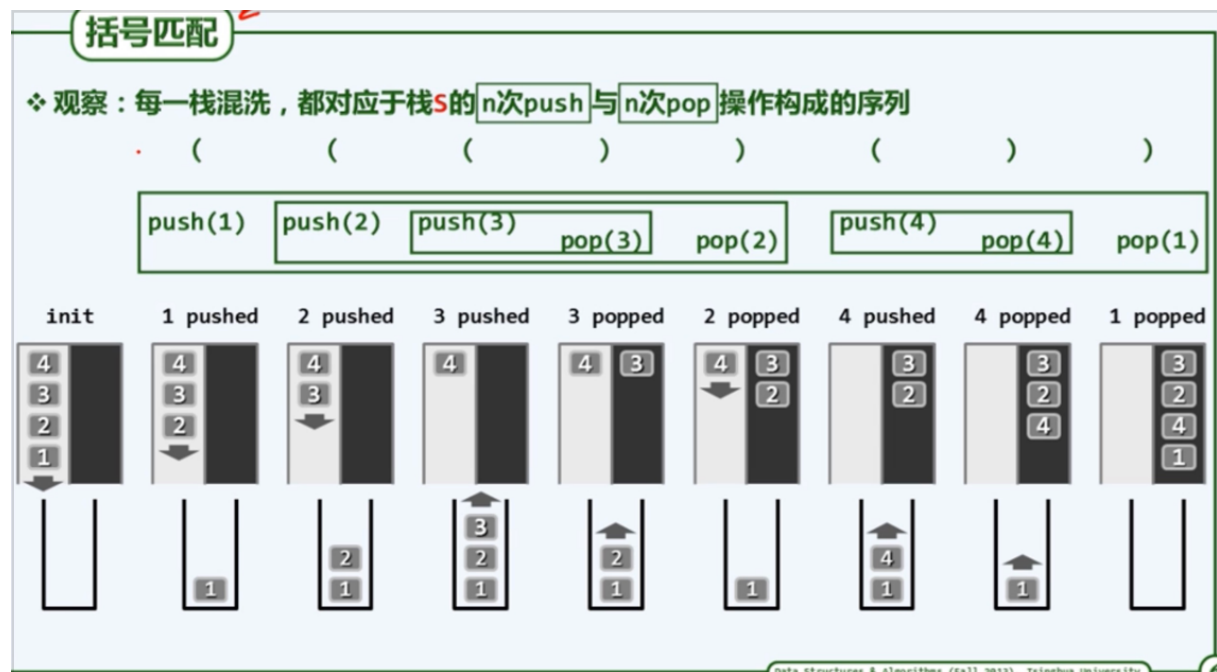
```
bool stackPermutation(stack<int> &A, stack<int> &B) { // 栈混洗 O(n) 模拟算法
    stack<int> S, temp;
    while (!B.empty()) {
        temp.push(B.top());
```

```

        B.pop();
    }
    while (!A.empty()) {
        S.push(A.top());
        A.pop();
        if (temp.top() == S.top()) {
            temp.pop();
            S.pop();
            while (!S.empty()) {
                if (temp.top() == S.top()) {
                    temp.pop();
                    S.pop();
                }
            }
            else return false;
        }
    }
    return S.empty();
}

```

括号匹配与 栈混洗



合法的栈混洗系列和合法的括号匹配表达式右一一对应的关系