严锦 1700011049

3.1

```
template<class T>
class myStack {
private:
    queue<T> A, B;
    int size;
public:
    T top() {
        T tmp;
        if (!A.empty()) {
            for (int i = 0; i < size; ++i) {
                tmp = A.front();
                A.pop();
                B.push(tmp);
            }
            return tmp;
        } else if (!B.empty()) {
            for (int i = 0; i < size; ++i) {
                tmp = B.front();
                B.pop();
                A.push(tmp);
            }
            return tmp;
        }
    }
    void pop(){
        T tmp;
        if (!A.empty()) {
            for (int i = 0; i < size - 1; ++i) {
                tmp = A.front();
                A.pop();
                B.push(tmp);
            }
            A.pop();
        } else if (!B.empty()) {
            for (int i = 0; i < size - 1; ++i) {
                tmp = B.front();
                B.pop();
                A.push(tmp);
```

```
            }
            B.pop();
        }
        size--;
    }
    void push(const T& elem) {
        if (!A.empty())
            A.push(elem);
        else if (!B.empty())
            B.push(elem);
        else {
            A.push(elem);
        }
        size++;
    }
    bool empty(){
        return (A.empty() && B.empty());
    }
};

```

时间效率

- top()

  需要将A或B中的元素全部转移到另一个队列中，时间复杂度为$\mathcal{O}(n)$

- pop()

  需要将A或B中的元素全部转移到另一个队列中，并删除队尾元素，时间复杂度为$\mathcal{O}(n)$

- push()

  直接在队尾添加元素，时间复杂度为$\mathcal{O}(1)$

- empty()

  直接调用队列的empty()接口，时间复杂度为$\mathcal{O}(1)$

3.2

证明

- 充分性

```
bool checkPermutation(vector<int> seq){
    stack<int> stk;
    stk.push(0);
    int length = (int) seq.size();
```

```
    int cnt = 1;
    for (int i = 0; i < length; ++i) {
        if (seq[i] > stk.top()) {
            //如果第i个序列元素大于栈顶元素，就循环入栈，直到栈顶元素与之相等后，再返回栈顶元素，就可以将该元素出栈
            while (stk.top() < seq[i]) {
                stk.push(cnt++);
            }
            stk.pop();
        } else if (seq[i] == stk.top()) {
            //如果第i个序列元素等于栈顶元素，就直接返回栈顶元素，此即该元素的出栈结果
            stk.pop();
        } else if (seq[i] < stk.top()) {
            //如果第i个序列元素小于栈顶元素，则该元素一定存在于栈的内部，不可能优先于栈顶元素出栈，因此该出栈序列不合法
            return false;
        }
    }
    return true;
}
```

以上是用于测试出栈序列合法性的算法。显然，当不存在下标$i,j,k \ \rightarrow \ i<j<k \ \wedge P_j<P_k<P_i$时，上述算法返回真。

- 必要性

  设存在下标$i,j,k \ \rightarrow \ i<j<k \ \wedge P_j<P_k<P_i$，则$P_i>P_j$，而$P_i$却比$P_j$先出栈，违反了栈先入先出的规则，因此假设不成立。

出栈序列数

在入栈出栈操作中，push和pop操作彼此对应。因此对于含有2n个元素的出栈序列，必定含有n对彼此对应的push和pop操作。将这对操作从操作序列中删除后，剩余n-1对彼此对应的push和pop操作，问题规模缩小。即由n对push和pop操作组成的一个入栈序列，可以分解为$S_n = (S_k)S_{n-k-1}$

由于$k \in [0,n)$，所以有:

$$T(0)=T(1)=1$$
$$T(n)=\sum_{k=0}^{n-1}T(k)\cdot T(n-k-1)$$

这是一个Catalan数递推式，解得:

$$T(n)=\frac{1}{n+1}\binom{2n}{n}$$