表、栈和队列 | Lists, Stacks and Queues

抽象数据结构 | Abstract Data Type, ADT

抽象数据结构就像面向对象里的接口（抽象类）。抽象数据结构首先是一种数据结构，因此它定义了对象的格式和允许的操作及其效果。但其 **抽象** 在于它操作的实现是没有在 ADT 中给出的。

表 ADT | The List ADT

Objects: $A_1, A_2, \ldots, A_n$

Operations: Print, MakeEmpty, Insert, Delete, Find, FindKth

实现

a. 数组实现。插入和删除代价较高；表的大小需要事先已知，因此一般不用。

b. 链表实现。链接

应用

- 多项式 ADT | Polynomial ADT

```
typedef struct Node *PtrToNode;
struct Node  {
    int Coefficient;
    int Exponent;
    PtrToNode Next;
};
typedef PtrToNode Polynomial;
```

```
8
9  Polynomial Read(){
10     /* Input should be sorted in decreasing order of exponents.
    */
11     typedef PtrToNode ptr;
12     ptr head = (ptr)malloc(sizeof(Node)), tail = NULL;
13     head->Next = NULL, tail = head;
14
15     int length; scanf("%d", &length);
16     while(length--){
17         tail->Next = (PtrToNode)malloc(sizeof(Node));
18         tail = tail->Next, tail->Next = NULL;
19         scanf("%d%d", &tail->Coefficient, &tail->Exponent);
20     }
21
22     return head;
23 }
24
25 void Print(Polynomial L){
26     if(L == NULL){
27         printf("\n");
28         return;
29     }
30
31     Polynomial temp = L->Next;
32     while(temp != NULL){
33         printf("%d %d ", temp->Coefficient, temp->Exponent);
34         temp = temp->Next;
35     }
36     printf("\n");
37     return;
38 }
```

- 多重表 | Multilists
- 基数排序 | Radix Sort

栈 ADT | The Stack ADT

栈是一个后进先出（LIFO）表，限制了插入和删除只能在表的末端（成为栈顶，top）进行。典型的操作是 Push, Pop 和 Top（读取栈顶元素的值）。

实现

a. 链表实现。与链表实现线性表一样，栈通过一个头结点指示。当栈为空时，该节点指向 NULL。Push, Pop 和 Top 通过对链表前段进行插入、删除或读取实现。
b. 数组实现。数组实现是显然的。

> 需要提示的是：在之前，我们用数组实现栈时通常习惯把栈顶指针指向栈顶元素的上面一个元素（即下次 push 的位置）；但是课程及课本的实现方式是确切指到栈顶元素（即刚刚 push 进去的东西）。因此在某些实现思路上存在偏差。这一偏差在作业 **6-2 Two Stacks In One Array** 中体现了出来。此记录。

应用

- 后缀表达式 | Postfix Expression

一道有趣的作业 ： 7-1 Pop Sequence

Given a stack which can keep *M* numbers at most. Push *N* numbers in the order of 1, 2, 3, ..., *N* and pop randomly. You are supposed to tell if a given sequence of numbers is a possible pop sequence of the stack. For example, if *M* is 5 and *N* is 7, we can obtain 1, 2, 3, 4, 5, 6, 7 from the stack, but not 3, 2, 1, 7, 5, 6, 4.

Input Specification:

Each input file contains one test case. For each case, the first line contains 3 numbers (all no more than 1000): *M* (the maximum capacity of the stack), *N* (the length of push sequence), and *K* (the number of pop sequences to be checked). Then *K* lines follow, each contains a pop sequence of *N* numbers. All the numbers in a line are separated by a space.

Output Specification:

For each pop sequence, print in one line "YES" if it is indeed a possible pop sequence of the stack, or "NO" if not.

Sample Input:

```
5 7 5
1 2 3 4 5 6 7
3 2 1 7 5 6 4
7 6 5 4 3 2 1
5 6 4 3 7 2 1
1 7 6 5 4 3 2
```

Sample Output:

```
YES
```

```
NO
NO
YES
NO
```

大模拟可以简单地用 $O(n)$ 完成，优化的意义并不很大。但这里是否有更优的做法?

队列 ADT | The Queue ADT

队列是一个先进先出（FIFO）表。入队（Enqueue）在队尾（rear）插入一个元素，出队（Dequeue）则删除队头（front）的一个元素。

课本中只介绍了队列的数组实现，这种实现是显然的。同时，为了防止“假溢出”，课本介绍了循环队列的写法（并提出“它可能会使运行时间加倍”）。

EOF