Version 2.00

编译日期: **2019-09-18**

任何建议及错误信息请发送至邮箱
1049188593@qq.com

目　录

1

线性表队列栈的计算

- 知识点：讲解相关知识点。
- 题型：直接上真题。

1.1 知识点和方法论

1.1.1 知识点

- 线性表中的位子序列从1开始，计算机中的数组从0开始。
- 卡特兰数：n 个不同元素入栈，出栈序列的个数为 $\frac{1}{n+1}C_{2n}^{n} = \frac{1}{n+1}\frac{(2n)!}{n!\times n!}$
- 后缀表达式也成为，逆波兰式。
- 在考试中简单的栈实现和队列实现
  - 申明一个栈并初始化 int stack[maxSize]; int top=-1;
  - 元素入栈 stack[++top] = x;
  - 元素出栈 x= stack[top–];

1.1.2 方法论

- 如何想不出来只遍历一遍的方法，使用普通方法也可得到大部分的分数。
- 中缀表达式转为后缀表达式：

  1. 栈中只存放符号
  2. 处理括号

  i. 若为’(’ 存入栈中。

ii. 若为')' 将'(' 之前的符号依次弹出
3. 运用四则运算法则，比如
i. 栈顶'+' 小于扫描到的'/' ，'/' 入栈
ii. 栈顶'/' 大于扫描到的'+' ，'/' 出栈，'+' 入栈
4. 检验从后往前依次读取计算，看是否和预料到的相同。

1.2 真题实战

1.2.1 王道 2019 年线性表第 10 题

若长度为 n 的非空线性表采用顺序存储结构，在表的第 i 个位置插入一个数据元素，i 的合法值应该是 ( ) 解:
线性表中的位子序列从1开始，计算机中的数组从0开始
隐含条件是原先第 i 个位子如果有元素，其和其后的元素后移。易知是 $1 \leq i \leq n+1$

1.2.2 2010 年 408

设将 n(n>1) 个整数存放到一维数组 R 中。试设计一个在时间和空间两方面都尽可能高效的算法。将 R 中保存的序列循环左移 p(0<p<n) 个位置，即将 R 中的数据由 (X0,X1,X2...Xn-1) 变换为 (Xp,Xp+1...Xn-1,X0,X1,...,Xp-1). 要求:
1. 给出算法的基本设计思想.
2. 根据设计思想，采用 C 或 C++ 语言描述算法，关键之处给出注释。
3. 说明你所设计算法的时间复杂度和空间复杂度.
解:(可能一开始想不出特别好的方法或者当你速度特别慢的时候，记得先把一种最容易的方法写上去，如果是正确的但是时间和空间复杂度不是最优秀的也可以得到 10/15 分的成绩)
1) 算法设计思想:
1. 先把数组 0 - (p-1) 个反转
2. 再把数组 p - 最后反转
3. 最后吧整个数组反转
2)

```
#include <iostream>
```

```
#include <algorithm>
using namespace std;
// reverse(first, last)
// (1) first表示要排序数组的起始地址;
// (2) last表示数组结束地址的下一位;
// A 数组   p 左移个数   len 数组长度
void shiftLeft(int A[], int p, int len) { // 写出这个函数带上
       头文件 algorithm 就可以算对了
    reverse(A, A + p);
    reverse(A + p, A + len);
    reverse(A, A + len);
}

int main() {
    int a[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
    shiftLeft(a, 4, 9);
    for (int i = 0; i < 9; i++) {
        cout << a[i] << ” ”;
    }
    cout << endl;
    system(”pause”);
}
```

3)

时间复杂度

整个序列反转一次，相当于对一半的元素执行了交换，(temp = a; a = b; b =temp;) 执行次数为 n / 2 * 3. 总共相当于两次执行整个序列的反转。执行次数是 n * 3, 时间复杂度易知是 O(n) , 空间复杂度, 因为没有额外增加其他的空间, 算法原地工作, 所以是 O(1).

———

另解:

1) 算法设计思想:

1. 先把数组 0 - (p-1) 个存放在一个额外的数组中，再原数组向左平移 p 个单位，最后吧额外数组中的元素复制到后面

2)

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
// reverse(first, last)
//（1）first表示要排序数组的起始地址;
//（2）last表示数组结束地址的下一位;
// A 数组  p 左移个数  len 数组长度
void shiftLeft1(int A[], int p, int len) {
    reverse(A, A + p);
    reverse(A + p, A + len);
    reverse(A, A + len);
}

void shiftLeft2(int A[], int p, int len) {
    vector<int> v;
    for (int i = 0; i < p; i++) {
        int tmp = A[i];
        v.push_back(tmp);
    }
    for (int i = p,j = 0; i < len; i++, j++) {
        A[j] = A[i];
    }
    for (int i = len - p,j =0; i < len; i++,j++) {
        A[i] = v[j];
    }
}


int main() {
    int a[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
    shiftLeft2(a, 4, 9);
    for (int i = 0; i < 9; i++) {
        cout << a[i] << ” ”;
    }
    cout << endl;
```

```
    system(”pause”);
}
```

3)
时间复杂度
前 p 个元素复制了两次 2*p, 后 n - p 个元素转移了一次，总共 n + p 次，时间复杂度是 O(n)
空间复杂度，增加了 p 个空间，所以是 O(p) .

1.2.3 2009 年线性表 408

已知一个带有表头节点的单链表，节点结构为 data, link (杭电考试一般会给出结构体)
假设该链表只给出了头指针 list。在不改变链表的前提下，请设计一个尽可能高效的算法，查找链表中倒数第 K 个位置上的节点 (K 为正整数)。若查找成功，算法出书该节点的 data 域的值，并返回 1；否则只返回 0.
1. 给出算法的基本设计思想.
2. 描述算法的纤细时间步骤.
3. 根据设计思想，采用 C 或 C++ 语言描述算法，关键之处给出注释。

解:
1)
算法的基本设计思想是:(王道上面写的很详细，照搬了)
关键一遍遍历，设定两个指针，一个指针只管往下遍历，另一个指针，在第一个指针，开始遍历次数统计为 k 的时候开始遍历。第二个指针如果运动了的话，那么就证明倒数第 k 个节点存在。
2)
1. 初始化: count = 0, p 和 q 指针指向第一个节点。
2. 开始遍历: p 只要没有遇到 NULL 节点就一直往下遍历，同时 count 开始计数，当 count >= k 时 q 才开始遍历。3. 判断：如果 count < k 时，说明没有倒数第 k 个节点，返回 0 结束。否则输出第 k 个节点的值，然后返回 1. 3)

```
#include <iostream>
```

```
using namespace std;

typedef struct LNode {// 关键处的注释 略
    int data;
    struct LNode *link;
}LNode;

int search_k(LNode* list, int k) {
    LNode *p = list->link, *q = list->link;
    int count = 0;
    while (p != NULL) {
        if (count < k) count++;
        else q = q->link;
        p = p->link;
    }
    if (count < k)
    return 0;
    else {
        printf(”%d”, q->data);
        return 1;
    }
}

int main() {
    //创建节点

    //创建头结点
    LNode *list = (LNode *)malloc(sizeof(LNode));
    list->data = -1;
    list->link = NULL;
    LNode *q = list;
    for (int i = 0; i < 10; i++) {
        LNode *p = (LNode *)malloc(sizeof(LNode));
        p->data = i;
        p->link = NULL;
        q->link = p;
        q = p;
    }
```

```
    q = list;
    while (q) {
        cout << q->data << endl;
        q = q->link;
    }
    system(”pause”);
}
```

———-

其他想法:

把遍历的结果存放入一个额外数组中，然后，直接判断 len - k 是否小于 0，但是结果只能得到 10 分。

1.2.4 2012 年线性表 408

假定采用带头结点的单链表保存单词，当两个单词有相同的后缀是，则可共享相同的后缀存储空间，例如，"loading" 和"being" 的存储映像如下所示。

设 str1 和 str2 分别指向两个单词所在单链表的头结点，链表的节点结构为 [data, next], 请设计一个时间上尽可能高效的算法，找出由 str1 和 str2 所指向两个链表共同后缀的起始位置。

1. 给出算法的基本设计思想.
2. 根据设计思想，采用 C 或 C++ 语言描述算法，关键之处给出注释。
3. 说明你所设计算法的时间复杂度.

解:

1)

算法的基本设计思想是:(王道上面写的很详细，照搬了)

1. 计算两个单词的长度。
2. 将 p 和 q 分别指向两个单词, 同时将长的单词的指针，和短的指针长度对齐，然后开始遍历直到他们找到共同的节点。

2)

```
typedef struct LNode {
    char data;
```

```
    struct LNode *next;
}LNode;
/*求链表长度*/
int listlen(LNode* head) {
    int len = 0;
    while (head->next != NULL) {
        len++;
        head = head->next;
    }
    return len;
}
/* 找出统统后缀的起始地址 */
LNode * find_addr(LNode *str1, LNode *str2) {
    int m, n;
    LNode *q, *p;
    m = listlen(str1); // p 对应 m
    n = listlen(str2); // q 对应 n
    for (p = str1; m > n; m--)
        p = p->next;
    for (q = str2; m < n; n--)
        q = q->next;
    //现在p和q在倒数相同的位置上，确定他们的共同位置
    while (p->next != NULL && p->next != q->next) {
        p = p->next;
        q = q->next;
    }
    return q->next;
}
```

3)

时间复杂度：

一般指最坏时间复杂度 由此得知，计算两个长度为 m+n, 对齐假设啥都不干，只有最后一个字符是他们共同的节点，那么就可以得出右遍历了一次，最大时间复杂度是 $2\times(m+n)$, 所以时间复杂度是 O(m + n). 空间复杂度是 O(1), 原地操作

1.2.5 王道有意义的题

3 个不同元素一次进栈，能得到 ( ) 不同的出栈序列。

解:

使用卡特兰数公式：$\frac{1}{3+1}\frac{(6)!}{3!\times 3!}$ = 5 种不同的出栈序列。

1.2.6 2014 年计算机 408

假设栈初始为空，将中缀表达式 a/b+(c*d - e*f)/g 转换为等价的后缀表达式的过程中，当扫描到 f 时，栈中的元素依次是 ( ).

解：

栈：push(/),'/' > '+' pop('/') push('+'),push('(', push'*','*' > '-' push '-' pop '*','-' < '*' push '*', == +,'(',-,'*'

打印序列: a,b,'/',c,d,'*',e,f

继续

栈：pop'*' pop'-' del'(', '+' < '/' push '/',pop '/', pop '+'

打印序列: '*"-',g,'/"+'