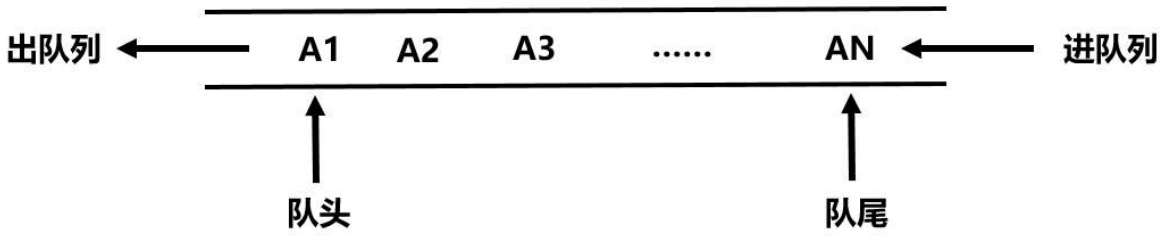
大家好，很高兴我们可以继续交流学习算法相关的面试题。在上一小节中，我们主要对排序与查找算法，常见链表以及二叉树的面试题目进行了分析与交流。在本小节中，我们主要对**队列，堆栈，字符串与数组**等知识点进行交流。针对各个知识点最高频的面试题目来进行解析，希望遇到经典的基础算法题目时，大家可以把握机会，**“手撕”成功**。

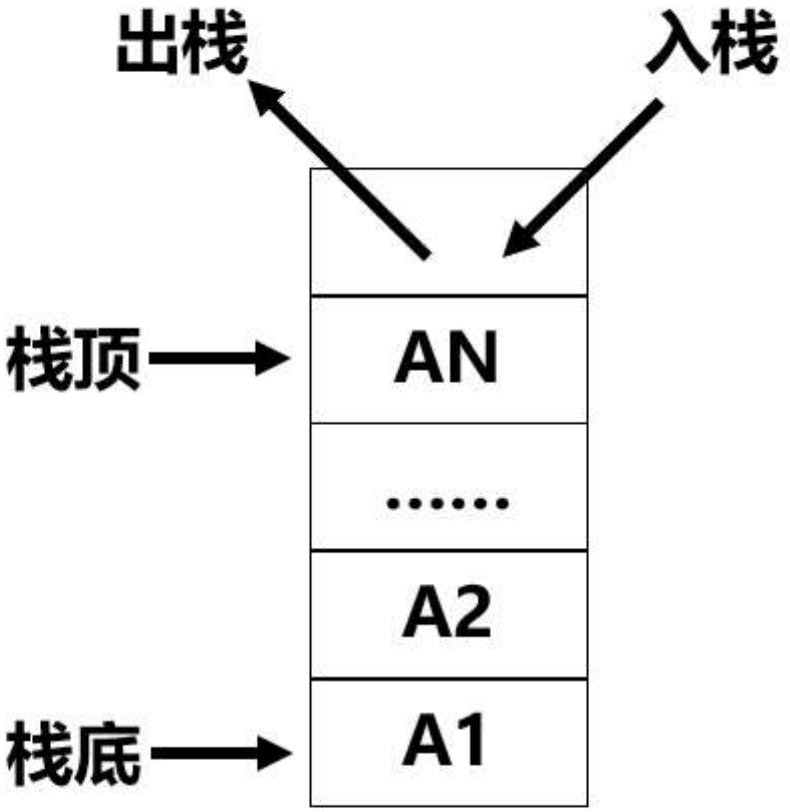
（1）队列与堆栈：

队列是一种特殊的线性表，特殊之处在于它只允许在表的前端（front）进行删除操作，而在表的后端（rear）进行插入操作。进行插入操作的端称为**队尾，进行删除操作的端称为队头。队列具有先进先出**的特性，示意图如下所示：



队列示意图

堆栈是一种具有**后进先出特性**的数据结构，因为其只可以在**栈顶**进行数据的入栈和出栈操作。堆栈的数据结构示意图如下所示：



堆栈示意图

在上一小节介绍的二叉树的深度优先与宽度优先的遍历中，我们使用到了队列和堆栈来做为其辅助的数据结构，分别利用了其**先进先出以及后进后出**的特性。在算法题目的考察中，不光将队列和堆栈做为辅助数据结构考察，还会直接对其相关特性进行考察，典型的算法题目如下：

- **包含min函数的栈**
- **用两个栈实现队列**
- **用两个队列实现栈**
- **自定义实现堆栈**

限于文章篇幅，我们这里来介绍**包含min函数的栈**题目。

题目描述：

定义栈的数据结构，请在该类型中实现一个能够得到栈最小元素的min函数。在该栈中，调用min、push和pop方法，且各个函数的时间复杂度均为O(1)。

算法思路：

题目要求我们的各个方法均为O(1)复杂度，我们考虑增加辅助空间来实现，即增加一个专门用来存储min值的辅助栈。

实现步骤：

- 比如，data依次入栈：5, 4, 3, 8, 10, 11, 12, 1。则辅助栈依次入栈：5, 4, 3，no，no, no, no, 1。其中，no代表此次不入栈。也就是说每次入栈的时候，如果入栈的元素比min中的栈顶元素小或等于则入栈，否则不入栈。
- 当出栈的时候，我们比较辅助栈与当前出栈的值是否相等。如果相等，则辅助栈栈顶元素也需要出栈。
- 当需要获取栈中最小元素的时候，我们直接获取到辅助栈的栈顶元素即可。

3

算法实现如下：

```
import java.util.Stack;
public class Main {

    Stack<Integer> stack = new Stack<>();
    Stack<Integer> minStack = new Stack<>();

/**
 * 首先需要对stack执行入栈操作，
 * 判断minStack中是否需要入栈操作
 */
    public void push(int node) {
        stack.push(node);
        if(minStack.isEmpty()||minStack.peek()>=node)
            minStack.push(node);
    }

/**
 * 判断minStack中是否需要出栈操作
 */
    public void pop() {
        if(stack.peek()==minStack.peek()){
            minStack.pop();
        }
        stack.pop();
    }

    public int top() {
        return stack.peek();
    }

/**
 * 直接peek minStack
 * @return
 */
    public int min() {
        return minStack.peek();
    }
}
```

这里需要注意的是，在获取栈中的min值时，我们应该使用minStack.peek方法而不是minStack.pop方法。peek方法仅仅是获取数值，但是pop方法则会执行出栈操作。

（2）字符串相关算法：

字符串相关的算法题目绝对是面试考察的重点内容。字符串在我们的日常开发中无处不在，在面试中涉及到的相关题目一般均可以通过递归或者动态规划来解决。高频的算法题目包括：

- **最长不含重复元素的子串**
- **最长公共子序列**
- **最长公共子串**
- **最长递增子序列**
- **最长公共前缀**
- **自定义函数实现字符串转整数的功能**

在开始交流字符串面试题目之前，我们先来简单介绍下**子串和子序列的区别**吧。

- **子串**：字符串中任意个**连续的字符**组成的子序列。
- **子序列**：字符串中**按照前后顺序**取出的任意个字符组成，**不要求连续**。

算法题目：

请从字符串中找出一个最长的不包含重复字符的子字符串，计算该最长子字符串的长度。假设该字符串中只包含‘a’-'z'的字符。

例如，给出字符串abcabcbb，那么符合要求的字串为abc，其长度为3。

算法思路：

可以使用**HashMap和双指针**来实现该算法。HashMap中不断更新维护每个字符出现的位置。使用count变量进行计数，当统计的最长子字符串被重复字符破坏时，比较count和max的大小，判断是否需要更新max变量。不管是哪种情况，每次都需要更新map中的信息，并且需要更新count变量。

算法实现：

```
package pak3;

import java.util.HashMap;
import java.util.Map;

/**
 *  求最长不含重复字符的子字符串
 * @author ywq
 */
public class Main {
    public static void main(String[] args) {
        System.out.println(lengthOfLongestSubstring("arbacacfr"));
        System.out.println(lengthOfLongestSubstring("hkcpmprxxxqw"));
```

```
        System.out.println(lengthOfLongestSubstring("dvdf"));
        System.out.println(lengthOfLongestSubstring("tmmzuxt"));
        System.out.println(lengthOfLongestSubstring("jbpnbwwd"));

    }

    public static int lengthOfLongestSubstring(String s) {
        if(s==null||s.length()==0)
            return 0;
        // 建立一个HashMap用来存放字符和位置信息
        Map<Character,Integer> map = new HashMap<Character, Integer>();
        int max = 0; // 用来记录最大值
        int count = 0; // 用来统计长度
        char[] c = s.toCharArray();
        for(int i =0;i<c.length;i++){
            if(!map.containsKey(c[i])){
                map.put(c[i], i);
                count++;
            }else {
                // 若map中已经包含该字符，分为两种情况讨论
                Integer index = map.get(c[i]);
                // 情况1：上次出现的该字符并不在当前所统计的最长字符串中，只需要
                if(i-index>count){
                    count++;
                    map.put(c[i], i);
                    continue;
                }
                // 情况2：上次出现的该字符影响了当前最长不重复的子字符串
                // 则更新位置信息、max变量和count计数
                map.put(c[i], i);
                if(count>max){
                    max = count;
                }
                count = i - index;
            }
        }
        // 防止出现没有重复字符的情况，此时max = 0
        return max>count?max:count;
    }
}
```

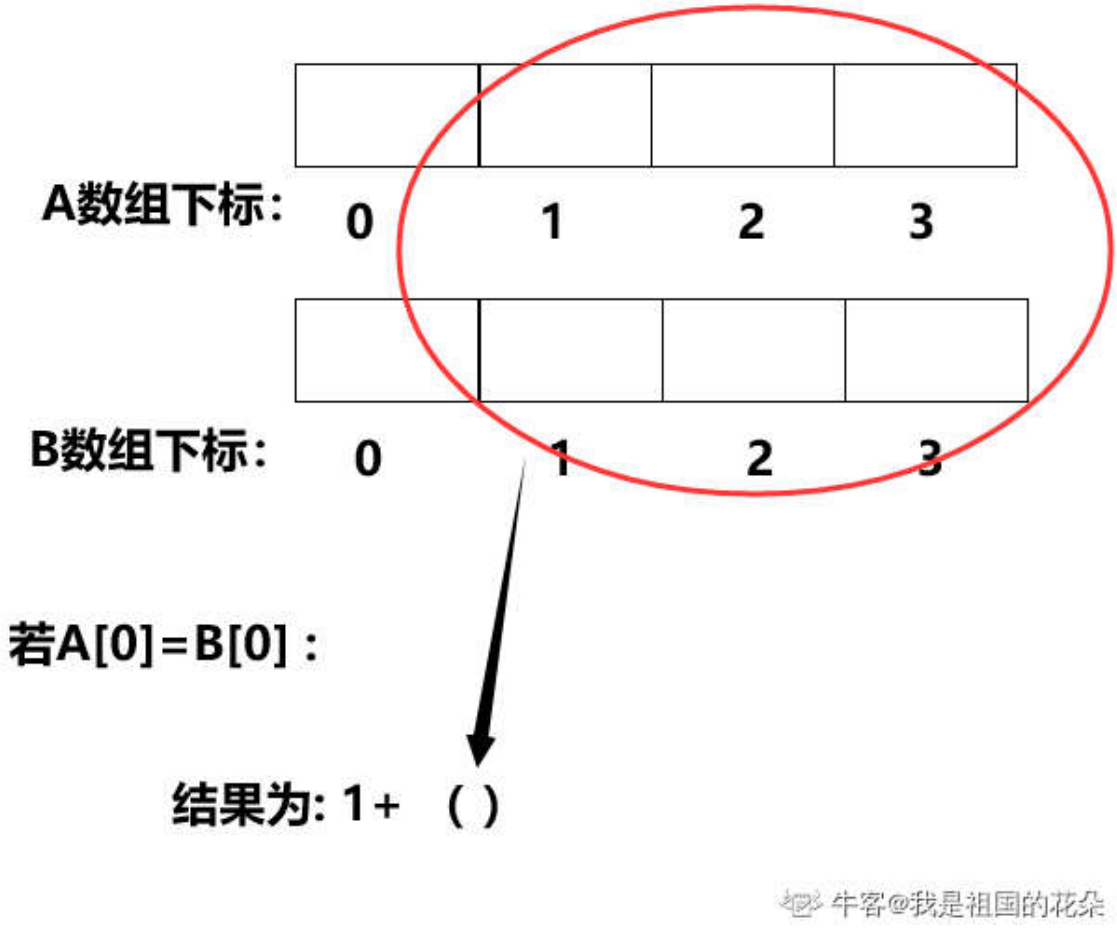
接下来，我们再来看一个**最长公共子序列**的问题吧。

题目描述：给定两个不字符串，求出最长公共子序列。

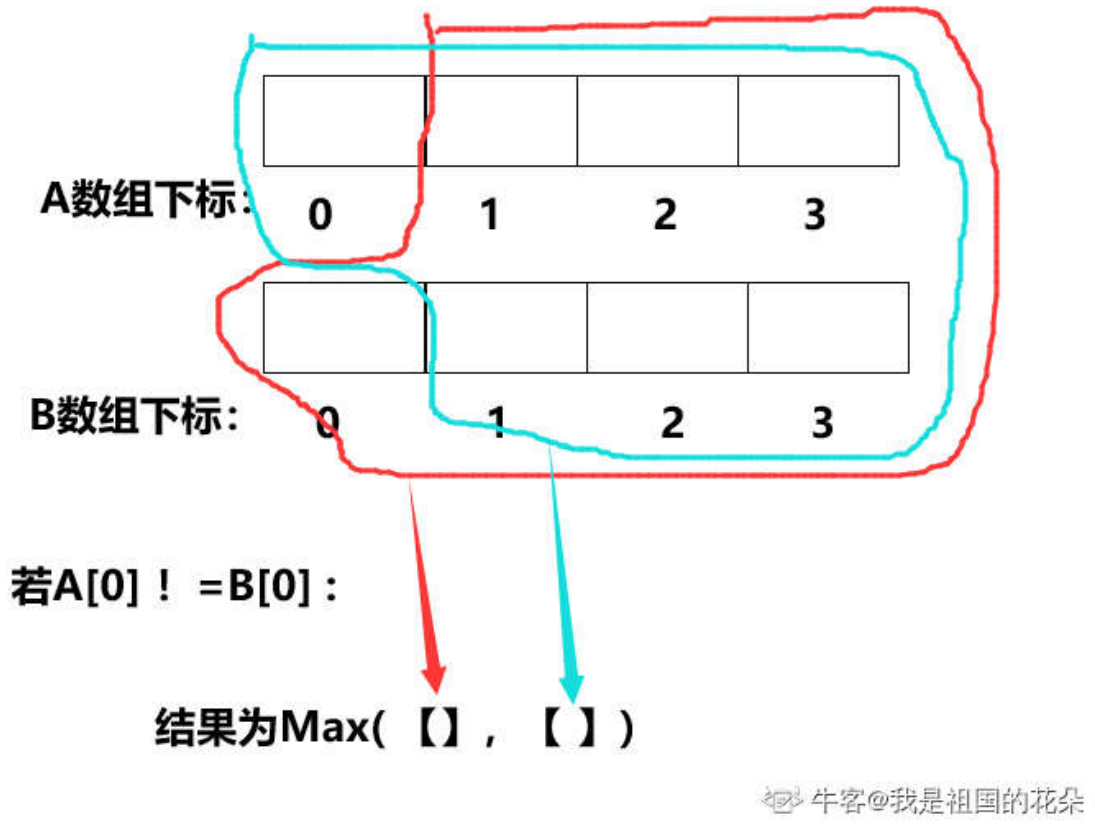
思路：

可以使用递归的方式来实现该算法。我们先比较字符小标为0的位置是否相等，根据两种情况分别进行后续的递归。画图分析如下：

情况一：



情况二:



算法实现:

```
1  /**
2       * 最长公共子序列，返回值为长度
3       * @param x
```

```
     * @param y
     * @return
     */
    int longestPublicSubSequence(String x, String y){
        if(x.length() == 0 || y.length() == 0){
            return 0;
        }
        if(x.charAt(0) == y.charAt(0)){
            return 1 + longestPublicSubSequence(x.substring(1), y.substring
        }else{
            return Math.max(longestPublicSubSequence(x.substring(1), y.subs
                    longestPublicSubSequence(x.substring(0), y.substring(1)
        }
        }
```

关于递归实现，我们需要确定递归结束的条件，以及递归公式即可实现算法。

（3）数组相关的算法：

数组是一种极其常见的数据结构，在面试中和数组相关的算法题出现频率相当高。对数组相关算法题目的考察，一般对时间复杂度和空间复杂度有所要求。所以，最常用的方法就是设置两个指针，分别指向不同的位置，不断调整指针指向来实现O（N）时间复杂度内实现算法。常见的面试题目：

- **调整数组顺序使奇数位于偶数前面**
- **拼接一个最大/小的数字**
- **合并两个有序数组**
- **查找多数元素**
- **数组中的重复元素**

接下来，我们进行典型题目的分析。

题目描述：

调整数组顺序使奇数位于偶数前面。输入一个整数数组，实现一个函数来调整该数组中数字的顺序，使得所有奇数位于数组的前半部分，所有偶数位于数组的后半部分。

思路：

关于数组的操作，我们首先考虑使用双指针。使用left和right指针，left指针从前往后移动，直到遇到偶数。right指针向前移动，直到遇到一个奇数。交换两个指针所指向的元素。通过多次交换来实现顺序的调整。

算法实现：

```java
/**
     * 可以满足奇数位于偶数前面的算法，但是奇数和奇数、偶数和偶数的相对位置不能保证
     * 时间复杂度O(N)，空间O(1)
     * @param arr
     */
    private static void reOrderArray(int[] arr){
        if(arr==null||arr.length<2)
            return ;
        int left = 0;
        int right = arr.length-1;
        while(left<right){
            while((arr[left]&1)==1){
                left++;
            }
            while((arr[right]&1)==0){
                right--;
            }
            // 如果不加此处的if判断语句，会导致right已经在left前面了，但是依然进
            // 即将已经在前面的奇数和后面的偶数进行了置换！！！
            if(left<right){
                int temp = arr[left];
                arr[left] = arr[right];
                arr[right] = temp;
            }
        }
    }
```

3

限于篇幅，其余数组相关面试题目的解析，这里就不加以阐述了。数组是面试中极其重要的考察点，我们不光要完成算法，还需要尽可能的考虑算法时间和空间复杂度，所以使用多个指针操作一般情况下会是较好的解决办法。

接下来，我们再来看一个**青蛙跳台阶**的问题吧，这是一个常见的对斐波那契数列的考察。

题目一描述：青蛙跳台阶问题。

一只青蛙一次可以跳上1级台阶，也可以跳上2级台阶。求该青蛙跳上一个n级台阶总共有多少种跳法?

分析：

当n = 1， 只有1中跳法；当n = 2时，有2种跳法；当n = 3 时，有3种跳法；当n = 4时，有5种跳法；当n = 5时，有8种跳法；.....规律类似于Fibonacci数列：

$$Fib(n)=\begin{cases}1 & n=1\\ 2 & n=2\\ Fib(n-1)+Fib(n-2) & n>2\end{cases}$$

所以，我们马上可以写出如下的递归实现代码：

```
public int Fibonacci(int n){
    if(n<=2)
        return n;
    return Fibonacci(n-1)+Fibonacci(n-2);
}
```

当我们写出递归代码时，面试官应该会建议我们对递归代码进行优化，因为递归代码中有太多的重复运算。所以，我们考虑使用使用变量保存住中间结果。 代码实现如下：

```
public int jumpFloor(int number) {
    if(number<=2)
        return number;
    int jumpone=2; // 离所求的number的距离为1步的情况，有多少种跳法
    int jumptwo=1; // 离所求的number的距离为2步的情况，有多少种跳法
    int sum=0;
    for(int i=3;i<=number;i++){
        sum=jumptwo+jumpone;
        jumptwo=jumpone;
        jumpone=sum;
    }
    return sum;
}
```

接下来，我们继续看青蛙跳台阶的变态版。

题目二：青蛙变态跳台阶问题

一只青蛙一次可以跳上1级台阶，也可以跳上2级……它也可以跳上n级。求该青蛙跳上一个n级的台阶总共有多少种跳法。

思路：

- 先跳到n-1级，再一步跳到n级，有f(n-1)种；
- 先跳到n-2级，再一步跳到n级，有f(n-2)种；
- 先跳到n-3级，再一步跳到n级，有f(n-3)种；
- ……
- 先跳到第1级，再一步跳到n级，有f(1)种；

所以，可以推出如下的公式：

- f(n)=f(n-1)+f(n-2)+f(n-3)+•••+f(1)
- f(n-1)=f(n-2)+f(n-3)+•••+f(1)
- 推出f(n)=2*f(n-1)

3

算法实现如下：

```
public int jumpFloor2(int num) {
    if(num<=2)
        return num;
    int jumpone=2; // 前面一级台阶的总跳法数
    int sum=0;
    for(int i=3;i<=num;i++){
        sum = 2*jumpone;
        jumpone = sum;
    }
    return sum;
}
```

总结：

正如我们在上一小节开头所说的，算法题目是需要大家在牛客网以及LeetCode等平台，通过大量的练习才可以比较熟练的掌握。在本专刊的两个算法小节中，我们仅仅是给出了面试中最为经典的算法题目的分析与交流。也就是说，当你遇到这些经典算法的时候，一定要把握住机会，“手撕”成功，那么你会离Offer越来越近哦。在下一小节中，我们将交流学习常见的设计模式。设计模式的熟练使用，可以帮助我们写出更加通用与优雅的代码。

限于作者水平，文章中难免会有不妥之处。大家在学习过程中遇到我没有表达清楚或者表述有误的地方，欢迎随时在文章下边指出，我会及时关注，随时改正。另外，大家有任何话题都可以在下边留言，我们一起交流探讨。

讨论 评论

柳杰201905011049420 1#

调整数组顺序那个怎么感觉和快排差不多

发表于 2020-01-15 11:35:03 赞(0) 回复(0)

牧水s N 2#

打卡

发表于 2020-02-16 22:05:06 赞(0) 回复(0)

牧水s

3#

最长公共子序列（LCS）

定义：在序列X和序列Y中同时出现的元素，按照脚标从小到大排列的这样的序列。

String x = "ABCBDABGGGTT";

String y = "BDCABATGGGTT";

x,y的最长公共子序列为BCBAGGGTT；

arbacacfr

hkcpmprxxxqw这两个是2cr

发表于 2020-02-16 22:11:25

赞(0) 回复(0)

3