计算机算法设计与分析之
递归与分治策略

易凯

2017 年 4 月 1 日

班　　级 软件 53 班
学　　号 2151601053
邮　　箱 williamyi96@gmail.com
联系电话 13772103675
个人网站 https://williamyi96.github.io
williamyi.tech
实验日期 2017 年 4 月 1 日
提交日期 2017 年 6 月 6 日

目录

1 重点掌握内容

1.1 递归概念及其使用

递归思想的核心是**分而治之**。其中，递归就是直接或者间接地调用自身的算法。其优势是递归算法结构清晰，可读性强，且容易用数学归纳法证明算法的正确性。而其运行效率相对较低，无论是耗费的计算时间还是占用的存储空间都比非递归算法要多。因此，提升此类问题运行效率的关键在于消除递归调用，将其转变为非递归算法。

在一个算法调用另一个算法之前，系统需要完成三件事： 1. 将所有实参指针、返回地址等信息传递给被调用算法；

2. 为被调用算法的局部变量分配存储区；

3. 将控制转移到被调用算法的入口。

在从被调用算法返回调用算法时，需要完成三件事情： 1. 保存被调用算法的计算结果；

2. 释放分配给被调用算法的数据区；

3. 依照被调用算法保存的返回地址将控制转移到调用算法

1.2 分治法基本思想

```
1  divide-and-conquer(P) {
2      if (P <= n0) adhoc(P);
3      divide P into smaller subinstances P1, P2, P3, P4,..., Pk
4      for (i=1; i<=k; i++)
5          yi = divide-and-conquer(Pi);
6      return merge(y1,y2,...,yk);
7  }
```

2 典型题详解

2.1 二分搜索正确性判断

题目描述 下面的 7 个算法与本章中的二分搜索算法 BinarySearch 略有不同。试判断这 7 个算法的正确性，如果不正确，请说明产生错误的原因。

BinarySearch

原始二分查找算法

```
int BinarySearch(int arr[], const int x, int n) {
    int left = 0, right = n -1;
    while(left <= right) {
        int middle = (left + right) / 2;
        if(x == arr[middle]) return middle;
        if(x > arr[middle]) left = middle + 1;
        else right = middle - 1;
    }
    return -1;
}
```

改进二分查找算法

```
package binarysearch;

public class BinarySearch {
        public int rank(int key, int[] a) {
        int lo = 0;
        int hi = a.length - 1;
        //lo represent low, hi represent high
        while (lo <= hi) {
            int mid = lo + (hi - lo) / 2;
            //use this form instead of (hi + lo)/2 because it can prevent overflow
            if(key < a[mid]) hi = mid - 1;
            else if(key > a[mid]) lo = mid + 1;
            else return mid;
        }
        return -1;
```

```
    }
}
```

测试代码

```
package binarysearch;

import java.util.Arrays;
import java.util.Scanner;

public class BinarySearchTest {
        public static void main(String[] args) {
                Scanner sc = new Scanner(System.in);
                BinarySearch bs = new BinarySearch();

                int a[] = {11,23,54,68,15,18,19,56,35};
                Arrays.sort(a);

                System.out.println("The sorted array is: ");
                for(int i = 0; i < a.length; i++) {
                        System.out.print(a[i] + "  ");
                }
                System.out.println();

                System.out.print("Please input the searching number: ");
                int key = sc.nextInt();
                System.out.print("The index of the it is: " + bs.rank(key, a));
        }
}
```

BinarySearch1

代码呈现

```
int BinarySearch(int arr[], const int x, int n) {
    int left = 0, right = n -1;
    while(left <= right) {
        int middle = (left + right) / 2;
        if(x == arr[middle]) return middle;
```

```
6           if(x > arr[middle]) left = middle;
7           else right = middle;
8       }
9       return -1;
10  }
```

正确性分析

由于在查找的过程中没 left 和 right 的移动，因此当程序查找失败时会进入死循环。

BinarySearch2

原因与上述问题大致相同，仍然为游标设置不正确，在程序运行时会产生最大位返回错误。

BinarySearch3

原因与上述问题大致相同，仍然为游标设置不正确，在程序运行时会产生最大位返回错误。

BinarySearch4

游标设置不当，如果查找失败程序会陷入死循环。

BinarySearch5

该程序正确。

BinarySearch6

游标设置不当，在程序运行时会产生最大位返回错误。

BinarySearch7

游标设置不当，如果查找失败则程序会陷入死循环。

2.2 二分搜索改写

题目描述

设 a[0:n-1] 是已经排好序的数组。请改写二分搜索算法，使得当搜索元素 x 不在数组中时，返回小于 x 的最大元素位置 i 和大于 x 的最小元素位置 j。当搜索元素在数组中时，i 与 j 相同，均为 x 在数组中是位置。

题目分析

此题的实现方式是在二分搜索的基础之上，将 index 的左右两个值 (如果查找失败) 或者是 index 对应值返回并输出。

算法描述与实现

```
1  package modefiedbinarysearch;
2
3  import java.util.Arrays;
4  import java.util.Scanner;
5
6  public class ModefiedBinarySearch {
7      public static int[] ExBinarySearch(int[] a, int index, int left, int right) {
8          int b[] = {0,0};
9          int middle;
10         while(left <= right) {
11             middle = (left + right) / 2;
12             if(index == a[middle]) {b[0] = middle; b[1] = middle; return b;}
13             if(index > a[middle]) left = middle + 1;
14             else right = middle - 1;
15         }
16         b[0] = left - 1; b[1] = right + 1;
17         return b;
18     }
19
20     public static void main(String[] args) {
21         Scanner in = new Scanner(System.in);
22         int a[] = {0,1,2,3,7,8,9};
23         System.out.println("Please input the number that you want to search");
24         int index = in.nextInt();
25         int left = a[0], right = a[a.length-1];
26         //System.out.println(index);
27         System.out.println(Arrays.toString(ExBinarySearch(a, index, left, right)));
28     }
29 }
```

反思拓展

此题为何在输出的数字在最大值和最小值范围之内，程序可以正常运行，而当输入的 index 的值大于最大值时，会报错呢?? 留待修补。

2.3 大数乘法优化

题目描述

给定两个大整数 u 和 v，他们分别有 m 和 n 位数字，且 m<n。用通常的乘法求 uv 的值需要 O(mn) 的时间，可以将 u 和 v 均看作是有 n 位数字的大整数，用本章中介绍的分治法，在 $O(n^{log3})$ 时间内计算 uv 的值。当 m 比 n 小很多时，用这种方法显然效率不高。试设计一个算法，在上述情况下用 $O(nm^{log(1.5)})$ 时间求出 uv 的值。

算法描述与实现

书本方法的时间复杂度实现

```
1  package bigintmultiply;
2
3  import java.util.Scanner;
4  import java.util.regex.Matcher;
5  import java.util.regex.Pattern;
6
7  /**
8   * Big Int Multiply
9   * @author William Yi
10  * Created on 3/18, 2017
11  */
12
13 public class BigIntMultiply {
14     //如果两个数的规模是这个级别的，那么我们直接进行相乘
15     private final static int SIZE = 4;
16
17     // 此方法要保证入参len为X、Y的长度最大值
18     private static String bigIntMultiply(String X, String Y, int len) {
19         // 最终返回结果
20         String str = "";
21         // 补齐X、Y，使之长度相同
```

```
22          X = formatNumber(X, len);
23          Y = formatNumber(Y, len);
24
25          // 少于4位数，可直接计算
26          if (len <= SIZE)    return "" + (Integer.parseInt(X) * Integer.parseInt(Y));
27
28          // 将X、Y分别对半分成两部分
29          int len1 = len / 2;
30          int len2 = len - len1;
31          String A = X.substring(0, len1);
32          String B = X.substring(len1);
33          String C = Y.substring(0, len1);
34          String D = Y.substring(len1);
35
36          // 乘法法则，分块处理
37          int lenM = Math.max(len1, len2);
38          String AC = bigIntMultiply(A, C, len1);
39          String AD = bigIntMultiply(A, D, lenM);
40          String BC = bigIntMultiply(B, C, lenM);
41          String BD = bigIntMultiply(B, D, len2);
42
43          // 处理BD，得到原位及进位
44          String[] sBD = dealString(BD, len2);
45          // 处理AD+BC的和
46          String ADBC = addition(AD, BC);
47          // 加上BD的进位
48          if (!"0".equals(sBD[1]))  ADBC = addition(ADBC, sBD[1]);
49          // 得到ADBC的进位
50          String[] sADBC = dealString(ADBC, lenM);
51          // AC加上ADBC的进位
52          AC = addition(AC, sADBC[1]);
53          // 最终结果
54          str = AC + sADBC[0] + sBD[0];
55          return str;
56      }
57      // 两个数字串按位加
58      private static String addition(String ad, String bc) {
59          // 返回的结果
60          String str = "";
```

```
61          // 两字符串长度要相同
62          int lenM = Math.max(ad.length(), bc.length());
63          ad = formatNumber(ad, lenM);
64          bc = formatNumber(bc, lenM);
65          // 按位加，进位存储在temp中
66          int flag = 0;
67          // 从后往前按位求和
68          for (int i = lenM - 1; i >= 0; i--) {
69              int t =  flag + Integer.parseInt(ad.substring(i, i + 1))
70                      + Integer.parseInt(bc.substring(i, i + 1));
71
72              // 如果结果超过9，则进位当前位只保留个位数
73              if (t > 9) { flag = 1; t = t - 10; }
74              else { flag = 0; }
75
76              // 拼接结果字符串
77              str = "" + t + str;
78          }
79          if (flag != 0) str = "" + flag + str;
80          return str;
81      }
82      // 处理数字串，分离出进位；
83      // String数组第一个为原位数字，第二个为进位
84      private static String[] dealString(String ac, int len1) {
85          String[] str = {ac, "0"};
86          if (len1 < ac.length()) {
87              int t = ac.length() - len1;
88              str[0] = ac.substring(t);
89              str[1] = ac.substring(0, t);
90          }
91          else  {
92              // 要保证结果的length与入参的len一致，少于则高位补0
93              String result = str[0];
94              for (int i = result.length(); i < len1; i++) result = "0" + result;
95              str[0] = result;
96          }
97          return str;
98      }
99      // 乘数、被乘数位数对齐
```

```
100     private static String formatNumber(String x, int len) {
101         while (len > x.length())  x = "0" + x;
102         return x;
103     }
104 
105     public static void main(String[] args)  {
106         // 正则表达式：不以0开头的数字串
107         String pat = "^[1-9]\\d*$";
108         Pattern p = Pattern.compile(pat);
109         // 获得乘数A
110         System.out.println("Please input A(not start with 0): ");
111         Scanner sc = new Scanner(System.in);
112         String A = sc.nextLine();
113         Matcher m = p.matcher(A);
114         if (!m.matches()) {
115             System.out.println("Illegal Number!");
116             return;
117         }
118         // 获得乘数B
119         System.out.println("Please input B(not start with 0)");
120         sc = new Scanner(System.in);
121         String B = sc.nextLine();
122         m = p.matcher(B);
123         if (!m.matches()) {
124             System.out.println("Illegal Number!");
125             return;
126         }
127         System.out.println(A + " * " + B + " = "
128             + bigIntMultiply(A, B, Math.max(A.length(), B.length())));
129     }
130 }
```

测试结果

```
1 Please input A(not start with 0):
2 456798615674964
3 Please input B(not start with 0)
4 4561418154748
5 456798615674964 * 4561418154748 = 2083649498603535117421329072
```

拓展深化的时间复杂度实现

在 m 比 n 小很多时，将 v 分为 n/m 段，其中每段的长度为 m 位。计算 uv 需要 n/m 次 m 位乘法运算。每次 m 位乘法可以用分治法进行实现，耗时为 $O(m^{log3})$。因此，算法所需的计算时间为 $O(\frac{n}{m}m^{log3} = O(nm^{log1.5})$

反思拓展

在实现该大整数乘法的时候，使用到的是正则式的匹配，因此对于 java 的理解可以说又上了一个台阶，虽然这部分只需要考察基本思想，但是对于编程能力的提升以及思维的拓展不失为一种较好的方式。

2.4 矩阵相乘算法设计

题目描述

对任何非零偶数 n，总可以找到奇数 m 和正整数 k，使得 $n = m2^k$。为了求出两个 n 阶矩阵的乘积，可以把一个 n 阶矩阵分成 m*m 个子矩阵，每个矩阵都有 $2^k \times 2^k$ 个元素。当需要求 $2^k \times 2^k$ 个子矩阵的积时，使用 Strassen 算法。设计一个传统方法和 Strassen 算法相结合的矩阵相乘算法，对任何偶数 n，都可以求出两个 n 阶矩阵的乘积。并分析算法的时间复杂性。

算法分析与设计

用传统方法计算两个 m 阶矩阵的乘积需要计算 $O(n^3)$ 次两个 $2^k \times 2^k$ 矩阵的乘积。用 Strassen 矩阵乘法计算两个 $2^k \times 2^k$ 矩阵的乘积需要的时间为 $O(7^k)$。因此计算的整体时间为 $O(7^k m^3)$。

2.5 多项式分治

题目描述

设 $P(x) = a_0 + a_1 x + ... + a_d x^d$ 是一个 d 次多项式。假设已有一个算法能在 $O(i)$ 时间内计算一个 i 次多项式与一个一次多项式的乘积，以及一个算法能在 O(ilogi) 时间内计算两个 i 次多项式的乘积。对于任意给定的 d 个整数 $n_1, n_2, ...n_d$。用分治法设计一个有效的算法，计算出满足 $P(n_1) = P(n_2) = .. = P(n_d) = 0$ 且最高次项系数为 1 的 d 次多项式 P(x), 并分析算法的效率。

算法分析与设计

此题的基本思路是已知线性方程组的解的情况下，反求其相关参数的情况。因此需要至少 $O(n^2)$ 的时间复杂度。但是该过程如何通过递归进行实现了，此处有待商榷。

2.6 线性空间子数组换位

题目描述

设 a[0:n-1] 是有 n 个元素的数组，k(0<=k<+n-1) 是一个非负整数。试设计一个算法将子数组 a[0:k-1] 与 a[k:n-1] 换位。要求算法在最坏情况下耗时 O(n), 且只用到 O(1) 的辅助空间。

算法分析与设计

```
public class ReversePartitialArray {
    void reversePartitialArray(int arr[], int k) {
        int n = arr.length();
        int len = max(k, n-k);
        for(int i = 0; i < len; i++) {
            int tmp = arr[i];
            swap(a[i],a[i+k]);
        }
    }
}
```

此处如何设计出一个较好的方法在线性时间之内完成相关的移位过程？

2.7 Gray 码构造

题目描述

Gray 码是一个长度为 2^n 的序列，序列中无相同元素，每个元素都是长度为 n 位的 (0,1) 串，相邻元素恰好只有一位不同，用分治策略设计一个算法对任意的 n 构造相应的 gray 码。

算法分析与设计

将 Gray 码的序列存储在数组之中，然后通过相关操作进行遍历。

```
package gray;
```

```
2
3  import java.util.Scanner;
4
5  public class Gray {
6          static int[] arr = new int[100];
7
8          static void gray(int n) {
9                  if(n == 1) {arr[1] = 0; arr[2] = 1; return;}
10                 gray(n-1);
11                 for(int k = 2^(n-1),i = k; i > 0; i--) a[2*k-i+1] = a[i] + k;
12                 System.out.println();
13         }
14
15         static void out(int n) {
16                 char[] str = new char[32];
17                 int m = 2^n;
18                 for(int i = 1; i <= m; i++) {
19                         //将arr[i]转换为2进制的字符串传给str
20                         itoa(arr[i],str,2);
21                         int s = strlen(str);
22                         for(int j = 0; i < n-s; j++)
23                                 System.out.print(0);
24                         System.out.println();
25
26                 }
27         }
28
29         public static void main(String[] args) {
30                 int n = 0;
31                 Scanner sc = new Scanner(System.in);
32
33                 while(sc.nextInt()!= 0) {
34                         int b = 2^(n-1);
35                         gray(n);
36                         out(n);
37                 }
38         }
39 }
```

2.8 有重复元素的排列问题

题目描述

设 $R = r_1, r_2, ..., r_n$ 是要进行排列的 n 个元素。其中元素 $r_1, r_2, ..., r_n$ 可能相同。试设计一个算法，列出 R 的所有不同排列。

从算法设计的角度分析即是：给定 n 以及待排列的 n 个元素，计算出这 n 个元素的所有不同排列。

数据输入与结果输出

数据输入

在指定文件中，文件的第一行是元素个数 n，接下来的一行是待排列的 n 个元素。

结果输出

将计算出的 n 个元素的所有不同排列输出到文件中，文件的最后一行中的数是排列的总数。

2.9 试题分析

此题在之前全排列代码的基础之上增加了 HasSame 进行输出字符串之中是否有重复元素的比较

2.10 代码实现

```
1  package robustperm;
2
3  import java.util.Scanner;
4
5  public class RobustPerm {
6          void robustPerm(char arr[], int start, int end) {
7                  if(start==end) {
8                          for(int i = 0; i <= start; i++) System.out.println(arr[i]);
9                  } else{
10                         for(int i = start; i <= end; i++) {
```

```
11                                  if(HasSame(arr, start, end)) {
12                                          Swap(arr[start], arr[i]);
13                                          robustPerm(arr, start + 1, end);
14                                          Swap(arr[start], arr[i]);
15                                  }
16                          }
17                  }
18          }
19
20          void Swap(char a, char b) {
21                  char tmp;
22                  tmp = a;
23                  a = b;
24                  b = tmp;
25          }
26
27          boolean HasSame(char arr[], int start, int end) {
28                  if(start <= end)
29                          for(int k = start; k < end; k++)
30                                  if(arr[k] == arr[end]) return false;
31                  return true;
32          }
33
34          public static void main(String[] args) {
35                  Scanner sc = new Scanner(System.in);
36                  System.out.println("Please input the string you want to permutaion:");
37                  final char[] arr = new char[100];
38                  for(int i = 0; i < 100; i++) {
39                          arr[i] = '0';
40                  }
41                  String str = sc.next();
42                  for(int i = 0; i < str.length(); i++) {
43                          arr[i] = str.charAt(i);
44                  }
45                  int start = 0, end = str.length();
46                  robustPerm(arr[], start, end);
47          }
48  }
```

反思总结

在最后一个 arr[] 的调用时出现的一些小问题如何进行修正可以达到较好的效果？

3 参考资料

1. What's the simplest way to print a Java array?. http://stackoverflow.com/questions/409784/whats-the-simplest-way-to-print-a-java-array

2. Large Int Multiply and java implementations. http://blog.csdn.net/oh_maxy/article/details/10903929

3. Thomas H.Cormen, Charles E.Leiserson, Ronald L.Rivest, Clifford Stein. Introduction to Algorithms.