算法设计与分析作业

习题 1 设 $P(x) = a_0 + a_1x + \cdots + a_dx^d$ 是一个 d 次多项式. 假设已有一算法能在 $O(i)$ 时间内计算一个 i 次多项式与一个 1 次多项式的乘积, 以及一个算法能在 $O(i\log i)$ 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 $n_1, n_2, \cdots, n_d$, 用分治法设计一个有效算法, 计算出 $P(n_1) = P(n_2) = \cdots = P(n_d) = 0$ 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.

解. 将 $P(x)$ 写为

$$P(x) = \prod_{i=1}^{d}(x - n_i) = \prod_{i=1}^{\lfloor d/2\rfloor}(x - n_i) \prod_{i=\lfloor d/2\rfloor+1}^{d}(x - n_i) = P_1(x)P_2(x)$$

用分治法将 d 次多项式转化为两个 $d/2$ 次多项式的乘积.

设这个算法计算 d 次多项式所需时间为 $T(d)$, 那么 $T(d)$ 满足

$$T(d) = \begin{cases} O(1) & d = 1 \\ 2T(d/2) + O(d\log d) & d > 1 \end{cases}$$

那么 $T(d) = O(d\log^2 d)$.

习题 2 设 $T[0 : n-1]$ 是 n 个元素的数组. 对任一元素 x, 设 $S(x) = \{i \mid T[i] = x\}$. 当 $|S(x)| > n/2$ 时, 称 x 为 T 的主元素. 设计一个线性时间算法, 确定 $T[0 : n-1]$ 是否有一个主元素.

解. 如果 T 有主元素 x, 那么 x 是 T 的中位数, 即若 T 的中位数不是主元素, 那么 T 无主元素.

那么用线性时间寻找中位数算法可以在线性时间内判定 T 是否有主元素.

习题 3 若在习题 2-9 中, 数组 T 中的元素不存在序关系, 只能测试任意两个元素是否相等, 试设计一个有效算法确定 T 是否有一个主元素. 算法的计算复杂性应为 $O(n\log n)$. 更进一步, 能找到一个线性时间算法吗?

解. 用分治法寻找 $T[1 : n]$ 的主元素, 设 x 是主元素, 那么 $S_x = \{i \mid T[i] = x\}$, 则 $|S_x| > n/2$. 将 $T[1 : n]$ 分为数组 $T[1 : n/2]$ 和 $T[(n/2 + 1 : n]$, 那么 x 是 $T[1 : n/2]$ 的主

元素或者 $T[n/2+1:n]$ 的主元素, 此时只需对两个数组线性扫描, 那么

$$T(n)=\begin{cases}O(1) & n\leqslant 4\\ 2T(n/2)+O(n) & n>4\end{cases}$$

则 $T(n)=O(n\log n)$.

用下面的算法可以在 $O(n)$ 时间内找出 $T[1:n]$ 的主元素.

对 $1\leqslant i\leqslant n/2$, 当 $T[2i-1]=T[2i]$, 将 $T[2i]$ 村存入数组 A, 否则不进行任何操作.

那么 A 中元素个数 $|A|\leqslant n/2$, 若 x 是 $T[1:n]$ 的主元素, 那么 x 是 A 的主元素或者 $T[n]$ 是 $T[1:n]$ 的主元素.

程序:

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

using namespace std;
#define N 10

int Partition(int* T, int l, int r)
{
        if (l == r)
        {
                return T[l];
        }

        int m = (l + r) / 2;
        int n1 = Partition(T, l, m);
        int n2 = Partition(T, m + 1, r);

        if (n1 == n2)
                return n1;
```

```
22          else
23          {
24                  int cnt1 = 0, cnt2 = 0;
25                  for (int i = l; i <= r; i++)
26                  {
27                          if (T[i] == n1)
28                                  cnt1++;
29                          if (T[i] == n2)
30                                  cnt2++;
31                  }
32                  int half = (l + r) / 2;
33                  if (cnt1 > half)
34                          return n1;
35                  else if (cnt2 > half)
36                          return n2;
37                  else
38                          return INT_MAX;
39          }
40  }
41
42  int main()
43  {
44          int T[N] = { 5,5,5,5,5,5,1,3,4,6 };
45          cout << "数组T的元素如下: " << endl
46          for (int i = 0; i < N; i++)
47          {
48                  cout << T[i] << " ";
49          }
50          cout << endl;
51          if (Partition(T, 0, N - 1) != INT_MAX)
```

```
52                 cout << "该数组的主元素为：" ;
53         << Partition(T, 0, N - 1) << endl;
54         else
55                 cout << "该数组没有主元素" << endl;
56         return 0;
57 }
```

结果:

```
1 数组T的元素如下：
2 5 5 5 5 5 5 1 3 4 6
3 该数组的主元素为：5
```

习题 4 设 $X[0:n-1]$ 和 $Y[0:n-1]$ 为两个数组, 每个数组中含有 n 个已排好序的数. 试设计 $O(\log n)$ 时间的算法, 找出 X 和 Y 的 $2n$ 个数的中位数.

解. 设 $X[i_1:j_1]$ 和 $Y[i_2:j_2]$ 是 X 和 Y 的排好序的子数组, 且 $j_1-i_1=j_2-i_2$.

若 $X[i_1] \leqslant Y[j_2]$, 则中位数 m 满足 $X[i_1] \leqslant m \leqslant Y[j_2]$, 若 $X[i_1] \geqslant Y[j_2]$, 则 $Y[j_2] \leqslant m \leqslant X[i_1]$.

设 $m_1=(i_1+j_2)/2$,$m_2=(i_2+j_2)/2$. 那么 $m_1+m_2=i_1+j_2$.

当 $X[m_1]=Y[m_2]$ 时,$m=X[m_1]=Y[m_2]$.

当 $X[m_1]<Y[m_2]$ 时, 设 m_1 是 $X[m_1:j_1]$ 和 $Y[j_2:m_2]$ 的中位数, 则 $m=m_1$.

当 $X[m_1]>Y[m_2]$ 时, 设 m_2 是 $Xi_1:m_1$ 和 $Y[m_2:j_2]$ 的中位数, 类似地有 $m=m_2$.

可得时间复杂度是 $O(\log n)$.

程序:

```
1 #include <iostream>
2 using namespace std;
3 void find(int a[], int l1, int r1, int b[], int l2, int r2)
4 {
5         int m1, m2;
6         if (r1 - l1 == 1) {
7                 m1 = a[l1] < b[l2] ? b[l2] : a[l1];
```

```
                m2 = a[r1] < b[r2] ? a[r1] : b[r2];
                if (m1 < m2)cout << m1 << endl;
                else cout << m2 << endl;
                return;
        }
        if (r1 == l1)
        {
                int k = a[r1] < b[r2] ? a[r1] : b[r2];
                cout << k << endl;
                return;
        }
        int mid1 = (l1 + r1) / 2, mid2 = (l2 + r2 + 1) / 2;
        if (a[mid1] == b[mid2])
        {
                cout << a[mid1] << endl;
                return;
        }
        if (a[mid1] < b[mid2])find(a, mid1, r1, b, l2, mid2);
        if (a[mid1] > b[mid2])find(a, l1, mid1, b, mid2, r2);
}
int main()
{
        int a[100001], b[100001];
        int n;
        cout << "输入n" << endl;
        cin >> n;
        cout << "输入a" << endl;
        for (int i = 0; i < n; i++)
        {
                cin >> a[i];
```

```
        }
        cout << "输入b" << endl;
        for (int i = 0; i < n; i++)
        {
                cin >> b[i];
        }
        cout << "结果" << endl;
        find(a, 0, n - 1, b, 0, n - 1);
}
```

结果:

```
输入n
5
输入a
1 6 8 3 4
输入b
9 4 6 8 5
结果
6
```