日期	科目	班级	姓名	学号
2022 年 10 月 4 日	算法设计与分析	强基数学 002	吴天阳	2204210460

第二次作业

(2-7) 解答. 设 $P_d(x)=(x-n_1)(x-n_2)\cdots(x-n_d)$，其满足 $P(n_1)=P(n_2)=\cdots=P(n_d)=0$ 且最高次项为 1 的 d 次的多项式，考虑使用分治法对其进行计算，不妨令 $d=2^k$，利用递归式 $P_d(x)=P_{d/2}(x)\cdot Q_{d/2}(x)$，其中 $P_{d/2}(x)=(x-n_1)(x-n_2)\cdots(x-n_{2^{k-1}})$，$Q_{d/2}(x)=(x-n_{2^{k-1}+1})(x-n_{2^{k-1}+2})+\cdots(x-n_{2^k})$. 则时间复杂度为

$$
\begin{aligned}
T(d)&=\frac{d}{2}\times 1+\frac{d}{2^2}\times 2\log 2+\cdots+\frac{d}{2^k}\times 2^{k-1}\log 2^{k-1}\\
&=\frac{d}{2}(1+1+2+3+\cdots+k-1)=\mathcal{O}\left(\frac{d}{2}\left(\frac{k(k-1)}{2}+1\right)\right)=\mathcal{O}(d\log^2 d).
\end{aligned}
$$

(2-9), (2-10) 解答. 若数组中存在主元 a，令 $k=\lceil\log x\rceil$，当 n 为奇数时，至少必有以下两种情况之一（n 为偶数时，只会出现情况一）：

1. 主元在 $T[0\sim n-2]$ 中，则必有至少三个相同项相邻且为主元，于是可通过以下递归算法查找主元. 存在 $i_0\in[1,\lfloor n/2\rfloor]$，使得 $T[2i_0-1]=T[2i_0]$，所以只需枚举 $n/2$ 项，建立新的数组 Q，当 $T[2i_0-1]=T[2i_0]$ 时，将下标 $2i_0$ 加入数组 Q 中，再递归地在 Q 中找主元. 时间复杂度为 $\mathcal{O}\left(n\left(\frac{1}{2}+\frac{1}{2^2}+\cdots+\frac{1}{2^k}\right)\right)=\mathcal{O}(n-1)$.

2. 主元为 $T[n-1]$，可以直接通过遍历整个数组判断该元素是否是主元. 结合第一种递归，则时间复杂度为 $\mathcal{O}\left(n\left(1+\frac{1}{2}+\cdots+\frac{1}{2^{k-1}}\right)\right))=\mathcal{O}(2n-1)$

综上，该算法的总时间复杂度为 $\mathcal{O}(n)$.

以下为算法部分，`check(x,y)` 用于交互查询 `a[x]` 是否与 `a[y]` 相等，其他函数只能通过调用该函数进行查询.

```cpp
#include <cstdio>
#include <vector>
using namespace std;
int a[] = {1, 1, 5, 5, 1, 5, 1};  // 有主元
// int a[] = {1, 1, 5, 5, 1, 5};  // 没有主元
// int a[] = {1, 1, 5, 5, 1, 1};  // 有主元
// int a[] = {1, 3, 2, 5, 1, 5, 1};  // 没有主元
bool check(int x, int y) {
    // 这个就是交互用的，专门用于返回 a[x],a[y] 是否相等，
    // 其他函数只能通过该函数访问数组
    return a[x] == a[y];
}
```

```cpp
// v 存储当前可能是主元的下标
bool find(vector<int> v) {  // 只能通过下标数组判断
    int n = v.size();
    if (n == 0) {
        return false;
    }
    vector<int> Q;
    for (int i = 1; i < n; i += 2) {
        if (check(v[i-1], v[i])) {
            Q.push_back(v[i]);
        }
    }
    if (n % 2 == 1) {
        int cnt = 1;
        for (int i = 0; i < n-1; i++) {
            if (check(v[i], v[n-1])) {
                cnt++;
            }
        }
        if (cnt > n/2) {
            return true;
        }
    }
    return find(Q);
}
int main() {
    int n = sizeof(a) / sizeof(int);
    vector<int> v;  // 需要判断的下标数组
    for (int i = 0; i < n; i++) {
        v.push_back(i);
    }
    if (find(v)) {
        printf(" 有主元\n");
    } else {
        printf(" 没有主元\n");
    }
}
```

(2-28) 解答. 通过由于两个数组都是有序数组，可通过二分法查找中位数，假设当前第一个数组的查找范围为 $[l_1, r_1)$ 中位数为 mid_1，第二个枚举范围为 $[l_2, r_2)$ 中位数为 mid_2，分三种情况:

1. 若 $mid_1 < mid_2$，则进一步递归查找，第一个数组查找范围为 $\left[\frac{l_1+r_2}{2}, r_1\right)$ 第二个数组查找范围为 $\left[l_2, \frac{l_2+r_2}{2}\right)$.

2. 若 $mid_1 > mid_2$，则进一步递归查找，第一个数组查找范围为 $\left[l_1, \frac{l_1+r_2}{2}\right)$ 第二个数组查找范围为 $\left[\frac{l_2+r_2}{2}, r_2\right)$.

3. 若 $mid_1 = mid_2$，则找到公共中位数返回 mid_1.

边界条件：若没有第三种返回条件，则最终会遍历到数组的边界，直接返回两个数组端点值的均值. 故时间复杂度为 $\mathcal{O}(\log n)$.

注：上述除法均为向下取整.

```cpp
#include <bits/stdc++.h>
using namespace std;
int a[] = {1,2,3,4,5};  // 中位数为 3.5
int b[] = {3,4,5,6,7};
// int a[] = {1,2,3,4,5};  // 中位数为 5.5
// int b[] = {6,7,8,9,10};
// int a[] = {3,4,5,6,7};  // 中位数为 3.5
// int b[] = {1,2,3,4,5};

double calc_mid(int *a, int l, int r) { // 计算数组 a[l~r] 的中位数
    int k = (l + r) / 2;
    if ((r - l + 1) % 2 == 0) {
        return (a[k] + a[k+1]) / 2.0;
    } else {
        return a[k];
    }
}
double get_mid(int l1, int r1, int l2, int r2) {
    if (l1 == r1-1) {  // 枚举到端点，中位数在两个数组之间
        return (a[l1] + b[l2]) / 2.0;
    }
    double mid1 = calc_mid(a, l1, r1);
    double mid2 = calc_mid(b, l2, r2);
    if (abs(mid1 - mid2) < 1e-6) {
```

```c
        return mid1;
    }
    if (mid1 < mid2) {
        return get_mid((l1+r1)/2, r1, l2, (l2+r2)/2);
    } else return get_mid(l1, (l1+r1)/2, (l2+r2)/2, r2);
}

int main() {
    int n = sizeof(a) / sizeof(int);
    printf(" 中位数为%.2f\n", get_mid(0, n, 0, n));
    return 0;
}
```