数据结构与算法分治算法

陈宇琪

2020 年 5 月 5 日

摘要

主要内容：分治的基本例题。

提交要求：除了 EOJ 上的题目在 EOJ 上提交之外，其余 10 道题目到超星上提交。

注意：填空题第 4 题和编程题第 3 题有一定难度，**请至少完成其中一题**。

分数分配：3+3+3+10+10+10+5+10+10+10，其中两个较难的题取较高得分记入总分。

折合分数：$sc^{'} = (92 - (100 - sc))/92 * 100$，其中 sc 为超星上批改分数，$92 = \lceil 64/0.7 \rceil$。

请大家尽快用自已的语言回答问题，有一些瑕疵没有问题的！

作业 DDL：2020-04-19

目录

1　知识点补充

1.1　快排核心思想

- Divide: Partition the array into two subarrays around a pivot x such that elements in lower subarray $\leq x \leq$ elements in upper subarray
- Conquer: Recursively sort the two subarrays.
- Combine: Trivial (because in place).

图 1: Quick Sort

从图中可以看出快排每次分治都选择数组中的一个数作为 pivot，将 $\leq$pivot 的数放在 pivot 左边，将 $\geq$pivot 的数放在 pivot 后边，当然对于等于 pivot 的数放在哪里都不影响结果，之后递归求解图中蓝色和黄色部分的数组即可。**也可以理解为每次将 pivot 放到它该在的位置上。**

1.2　快排的关键

快排的关键是寻找 $O(N)$ 的 Partition 算法。

Listing 1: **quick sort.cpp**

```
Quicksort(A,p,r)
{
        if (p<r)
        {
                q = Partition(A,p,r);
                Quicksort(A,p,q-1);
                Quicksort(a,q+1,r);
        }
}
```

代码中的 q 就是 pivot 所在的位置。注意代码中假设当前待排序区间为 $[p,r]$。

1.3　一个简单的 Partition 实现

在一个数组中找到 $\leq$pivot 的数和 $>$pivot 的数可以使用辅助 vector 数组通过一个遍历求得。

Listing 2: **naive partition.cpp**

```
int partition(vector <int> &a,int p,int r)
{
    int k=a[r];
    vector <int> lhs;
    vector <int> rhs;
    for (int i=p;i<=r;i++)
    {
```

```
        if (i==k) continue;
        if (a[i]<=a[k])
            lhs.push_back(a[i]);
        else rhs.push_back(a[i]);
    }
    int i=p;
    for (int j=0;j<lhs.size();j++)
        a[i++]=lhs[j];
    int q=i;
    a[i++]=a[k];
    for (int j=0;j<rhs.size();j++)
        a[i++]=rhs[j];
    return q;
}
```

1.4 一些常见的 Partition 实现

上面的 Partition 实现利用了辅助数组，浪费了一定的空间，但是快排可以实现**就地排序**。根据 PPT 上的图示，我们可以写出如下的代码：

Listing 3: **simple partition.cpp**

```
int partition(vector <int> &v,int p,int r)
{
        int x=v[r];
        int i=p-1;
        for (int j=p;j<=r-1;j++)
        {
                if (v[j]<x)
                {
                        i++;
                        swap(v[i],v[j]);
                }
        }
        swap(v[i+1],v[r]);
        return i+1;
}
```

当然还有另一种常见的写法是使用双指针的形式：
（过程描述参考附录）

Listing 4: **simple partition.cpp**

```
int partition(vector <int> &v,int p,int r)
{
        int temp=v[p];
        int i=p;
        int j=r;
        while (i<j)
        {
                while (i<j && v[j]>temp)
                        j--;
```

```
            v[i]=v[j];
            while (i<j && v[i]<=temp)
                i++;
            v[j]=v[i];
        }
        v[i]=temp;
        return i;
}
```

1.5 快排优化

快速排序算法对于 99.9999% 的数据都可以快速的进行排序，而且排序时就地排序的，不浪费额外的内存，但是对于精心构造的数据往往表现不好。

解决快排的问题的方法有很多，最简单的方法是将输入数据进行随机打乱。

Listing 5: **random shuffle.cpp**

```
int main()
{
        int n,x;
        cin>>n;
        for (int i=0;i<n;i++)
        {
            cin>>x;
            v.push_back(x);
        }
        random_shuffle(v.begin(),v.end());
        quicksort(v,0,n-1);
        for (int i=0;i<n;i++)
            cout<<v[i]<<' ';
        cout<<endl;
}
```

或者在选择 pivot 时候随机选择数组中的一个数。

Listing 6: **random select.cpp**

```
int partition(vector <int> &v,int p,int r)
{
    int k=rand()%(r-p+1)+p;
    swap(v[k],v[r]);
        int x=v[r];
        int i=p-1;
        for (int j=p;j<=r-1;j++)
        {
            if (v[j]<x)
            {
                i++;
                swap(v[i],v[j]);
            }
        }
        swap(v[i+1],v[r]);
```

```
        return i+1;
}
```

随机化选择可以破坏数据中的特征，使得快速排序的速度更加稳定。

当然也有一些基于统计量的优化（选择一些分位数作为 pivot）。

2 选择题

1、快速排序在下列（）情况下最易发挥其长处。

A．被排序的数据中含有多个相同排序码 B．被排序的数据已基本有序

C．被排序的数据完全无序 D．被排序的数据中的最大值和最小值相差悬殊

2、下述几种排序方法中，要求内存最大的是（）。

A．希尔排序 B．快速排序 C．归并排序 D．堆排序

3、下述几种排序方法中，（）是稳定的排序方法。

A．希尔排序 B．快速排序 C．归并排序 D．堆排序

3 简述题

1、根据 PPT 的描述写出对于待排序的数组 [45,23,12,67,31,39,41] 使用**快速排序**的递归归求解过程。（画出每一次根据哪一个元素进行求解，每一次**递归求解的子数组的元素情况**，请画一棵详细的**递归调用树**来记录这些信息）

2、分析快速排序的复杂度：指出最好复杂度和最坏复杂度。结合 PPT 说明在什么情况下取到最坏情况（可以用数据说明）。

3、假设现在对一个 int 数组排序，且假设数组中数的绝对值不超过 V，修改快速排序算法使得算法在最坏情况下的复杂度为 $O(n \times \log(V))$。

提示：如果一个子数组中所有数均相同，则可以提前结束。假设一个子数组中的元素取值在 $[a,b]$ 之间，是否可以在分治的时候，使得两个子数组的取值分别在 $[a,\frac{a+b}{2}]$ 和 $[\frac{a+b}{2},b]$ 之间。

4、请**结合生活场景**，再举一个分治算法在现实生活中的简单例子（不允许举 PPT 上有过的例子），请先描述一下背景，再具体解释其中蕴含的分治思想。

4 基础编程题

1、（完整代码）对于一个给定数组 a，使用分治算法实现二分查找，具体而言要求在 $O(\log(N))$ 的复杂度内判断数组 a 中是否存在 x。

2、（完整代码）对于一个给定数组 a 和一个数 s，判断 a 中是否存在两个不同的数 p,q，使得 $p+q=s$，你可以认为数组中没有重复的数，复杂度要求 $O(N\log(N))$。

提示：使用第一题的二分查找，为了防止重复扣分，你可以使用 STL 的 lower_bound 来完成这道题目。

3、（递归函数）对于给定数组，使用分治的思想计算 $\max_{1\le l\le r\le N}[\min\{a_l,...,a_r\}\times(r-l+1)]$，复杂度要求 $O(N\times\log(N))$。

提示：假设提供一个函数可以在 $O(1)$ 查询 $[L,R]$ 区间内的最小值以及最小值所在的位置。

提交函数申明：int minmax(const vector <int> &v);

提供函数申明：pair<int,int> query(int l,int r);

说明：pair<int,int> 为区间 $[l,r]$ 中最小值和最小值所在位置的元组，请确保调用时 $l\le r$。

备注 1：使用 RMQ 对数组 v 进行预处理就可以在 $O(1)$ 时间内查询区间最小值，有兴趣的同学可以提交完整代码。

备注 2：EOJ 上有对应的题目。

4、完成 EOJ 上相关习题，**请至少完成其中 2 题**，最后一题为加分题。

图 2: EOJ 相关习题

5 附录：RMQ 算法

5.1 预处理

假设二维数组 $dp[i][j]$ 表示从第 i 位开始连续 2^j 个数中的最小值，则 $dp[i][j] = min(dp[i][j-1], dp[i+2^{j-1}][j-1])$，这里使用了倍增的思想。

5.2 查询

对于区间查询 $[l, r]$，假设 $k = \lfloor \log_2(r-l+1) \rfloor$，则 $RMQ[l, r] = min(dp[l][k], dp[r-2^k+1][k])$。

5.3 算法复杂度

RMQ 的时间复杂度和空间复杂度都是 $O(N \times \log(N))$。

5.4 RMQ 推广

假设定义一个运算符 $\cdot$ 满足 $a \cdot a = a$，且满足交换律和结合律，则可以使用 RMQ 求 $a_l \cdot ... \cdot a_r$。

例如可以用 RMQ 求解区间最大，区间最小，区间或，区间与。但是 RMQ 不能求解区间异或，所以需要线段树完成。

6 参考答案

6.1 选择题

CCC

6.2 简答题

4、根据原始 PPT 上的做法，答案应该是这样的。

5、快速排序的最好复杂度 $O(n \log n)$。对于最坏复杂度：对于选择待排序队列中第一个或者最后一个作为 pivot 的情况，当数据已经正序排好序或逆序排好序 $O(n^2)$

6、其实题目已经把算法说的很明白了，这个做法的本质不是想要使得每次划分尽可能均分，而是尽可能缩小数的范围，所以提前结束是十分重要的。

7、比如整理资料，有时资料需要按照时间顺序排序，但是由于平时不整理导致资料很乱。这个时候可以先按照月份划分成 12 份，但根据每个月内的时间先后进一步排序。假设每个月的工作量都差不多的情况下，第一次的划分可以认为是平均划分的。

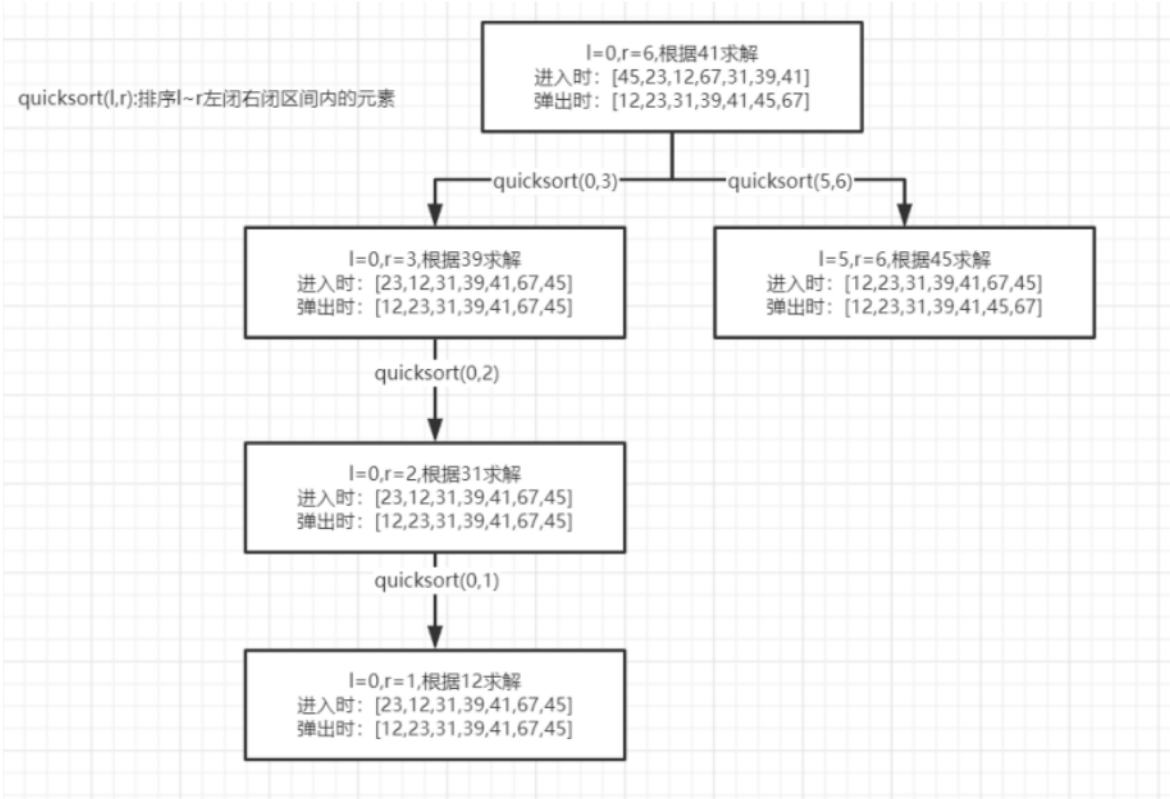


图 3: 第 4 题答案

6.3 编程题

8、二分查找可以用递归写，也可以不用递归。

Listing 7: **ans8.cpp**

```
vector<int> arr;

int binsearch(int lo, int hi, int x)
{
        int mid = (lo + hi) / 2;
        if (arr[mid] == x)
        {
                return 1;
        }
        else if (lo < hi)
        {
                if (x < arr[mid]) return binsearch(lo, mid, x);
                else return binsearch(mid + 1, hi, x);
        }
        return 0;
}
```

典型错误的答案：

Listing 8: **wrong answer for ques8.cpp**

```
bool binary_search(int* bg, int* ed, int val)
```

```
{
        auto mid = bg + (ed - bg) / 2;
        if (*mid == val) {
                return true;
        }
        return binary_search(bg, mid, val) | binary_search(mid + 1, ed, val);
}
```

9、题目中强调了 $p \neq q$。

Listing 9: **ans9.cpp**

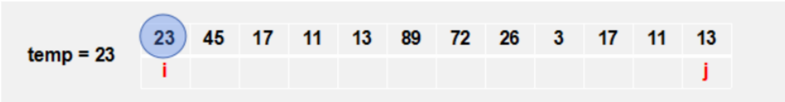
```
bool two_sum(int a[], size_t size, int sum)
{
        sort(a, a + size);       // O(nlogn)
        for (size_t i = 0; i < size; ++i) {
                if (binary_search(a + i + 1, a + size, sum - a[i])) {
                        return true;
                }       // O(n) * O(logn) = O(nlogn)
        }       // O(nlogn) + O(nlogn) = O(nlogn)
        return false;
}
```

10、最小的数使得跨区间的最优值可以很简单的计算出来。

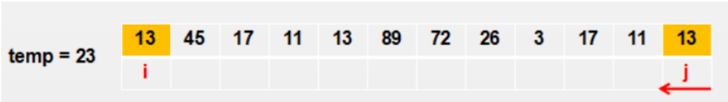
Listing 10: **ans10.cpp**

```
long long int min_max(vector<int> const &arr, int l, int r) {
    if (l <= r) {
        if (r == l) return arr[l - 1];
        pair<int, int> mid = query(l, r);
        long long int re = max(min_max(arr, l, mid.second - 1), min_max(arr, mid.second + 1, r));
        return max(re, ((long long int)mid.first) * (r - l + 1));
    }
    return 0;
}
```

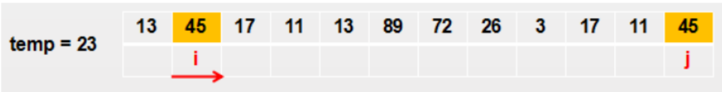
将数组第一个数23赋给temp变量，指针 i 指向数组第一个元素，指针 j 指向数组最后一个元素



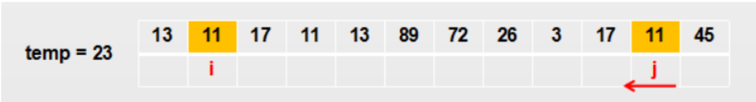
从 j 开始遍历（从右往左），遇到13时，因为13<=temp，因此将arr[j]填入arr[i]中，即此时指针 i 指向的数为13；



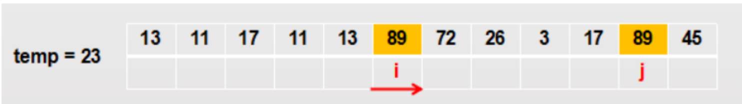
再从 i 遍历（从左往右），遇到45时，因为45>temp，因此将arr[i]填入arr[j]中，此时指针 j 指向的数为45；



继续从 j 遍历，遇到11时，因为11<=temp，因此将arr[j]填入arr[i]中，即此时指针 i 指向的数为11；



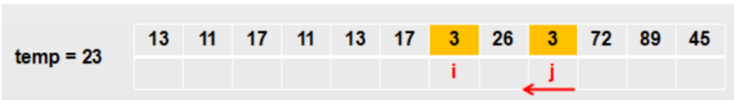
从 i 遍历，遇到89时，因为89>temp，因此将arr[i]填入arr[j]中，此时指针 j 指向的数为89；



从 j 遍历，遇到17时，因为17<=temp，因此将arr[j]填入arr[i]中，即此时指针 i 指向的数为17；

从 i 遍历，遇到72时，因为72>temp，因此将arr[i]填入arr[j]中，此时指针 j 指向的数为72；

从 j 遍历，遇到3时，因为3<=temp，因此将arr[j]填入arr[i]中，即此时指针 i 指向的数为3；



从 i 遍历，遇到26时，因为26>temp，因此将arr[i]填入arr[j]中，此时指针 j 指向的数为26；

从 j 遍历，和 i 重合；

将 temp（基准数23）填入arr[i]中。