算法设计与分析 实验1

计科210X 甘晴void 202108010XXX

目录

```
算法设计与分析<br/><br>实验1
```

```
(1) 分治法查找最大最小值
```

问题描述

想法

代码

评测

算法分析

(2) 分治法实现合并排序

问题描述

想法

代码

评测

算法分析

(3) 实现题1-3 最多约数问题

引言

题目描述

做法1(直接遍历求解+比较)

代码

评测

做法2(使用约数定理计算)

代码

评测

做法3(约数定理+DFS)

代码

评测

实验感悟

(1) 分治法查找最大最小值

问题描述

使用分治法查找给定数组a中的最大值与最小值

想法

可以看作二分查找的变形,区别在于不是有序的(我们是找值,也不需要有序),而且不是查找某个特定值(要找最小值和最大值)。

自上而下:将当前区域划分为前后两个区域,调用递归函数分别对这两个子区域进行划分, 直到划分至不再可分为止(递归边界,只剩1个数),其最大值与最小值都是其本身。

自下而上:对于当前区域,将左子区域与右子区域分别比较最大值与最小值,获得当前区域的最小值与最大值,返回至父层。

代码

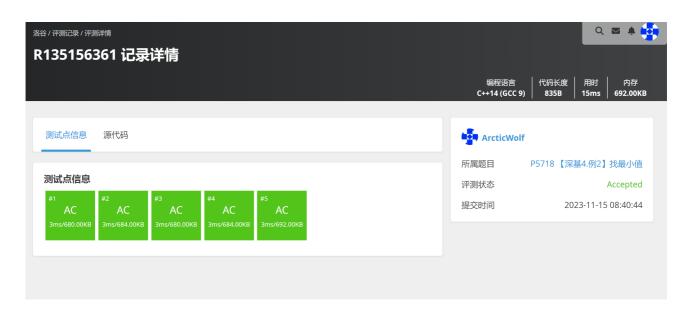
```
#include <iostream>
#include <math.h>
using namespace std;
void BinarySearch(int x[], int left, int right, int &maxi, int
&mini)
{
    if (left == right)
    {
        maxi = x[left];
        mini = x[left];
        return;
    }
    else
    {
        int max1, min1, max2, min2;
        int mid = (left + right) / 2;
        // cout << left << " " << mid << " " << right << endl;
        BinarySearch(x, left, mid, max1, min1);
        BinarySearch(x, mid + 1, right, max2, min2);
        maxi = max(max1, max2);
```

```
mini = min(min1, min2);
         return;
    }
}
int main()
{
    int n; // 数据量
    cin >> n;
    int x[n];
    for (int i = 0; i < n; i++)
        cin >> x[i];
    int maxi = x[0];
    int mini = x[0];
    BinarySearch(x, 0, n - 1, maxi, mini);
    cout << "max= " << maxi << endl;</pre>
    cout << "min= " << mini << endl;</pre>
}
```

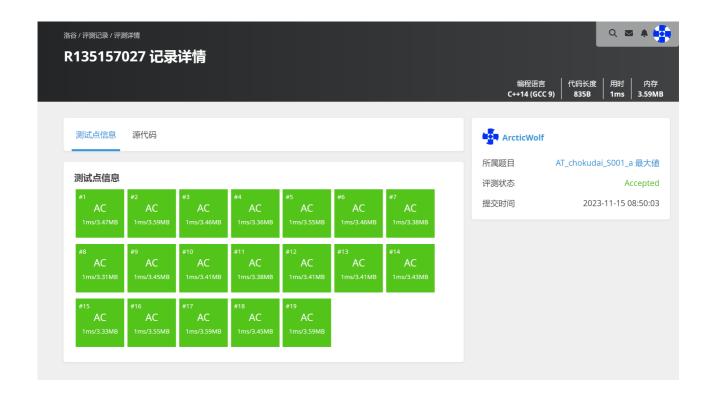
评测

洛谷没有直接查找最大/最小值的原题,但是有类似的题目,可以测试一下。

找最小值(https://www.luogu.com.cn/problem/P5718)



找最大值(https://www.luogu.com.cn/problem/AT chokudai S001 a)



算法分析

近似于二分查找,时间复杂度O(n)。

空间复杂度O(n)。

(2) 分治法实现合并排序

问题描述

使用分治法实现一个乱序数组a的排序

想法

二分排序属于是排序中的基础算法了。其主要思想在于使用分治算法。

自上而下:将当前区域划分为前后两个区域,调用递归函数分别对这两个子区域进行划分, 直到划分至不再可分为止(递归边界,只剩1个数),其自然有序。

自下而上:对于当前区域,将左子区域与右子区域进行合并,此时左子区域与右子区域都是有序的,进行O(n)的合并后本层区域有序,返回父层。

```
#include <iostream>
#include <math.h>
using namespace std;
void copy(int x[],int y[],int left,int right)
{
    for (int i=left; i<=right; i++) x[i]=y[i];</pre>
}
void Merge(int x[],int y[],int left,int mid,int right)
{
    // 合并x[left:mid] x[mid+1,right] 到 y[left:right]
    //cout<< "merge:"<<left<<" "<<mid<<" "<<right<<endl;
    int i=left,j=mid+1,k=left;
    while ((i<=mid) && (j<=right))</pre>
    {
        if (x[i] \le x[j]) y[k++] = x[i++];
        else y[k++]=x[j++];
        if (i>mid) //左边已经合并完了,剩下右边直接加入
            for (int t=j; t<=right; t++) y[k++]=x[t];</pre>
        else if (j>right) //右边已经合并完了,剩下左边直接加入
            for (int t=i; t<=mid; t++) y[k++]=x[t];
    }
    //cout<<"merge_result"<<endl;</pre>
    //for (int i=left;i<=right;i++)cout<<y[i]<<" ";</pre>
    //cout<<endl;</pre>
}
void MergeSort(int x[], int left, int right)
{
    if (left<right)</pre>
    {
        int mid=(left+right)/2;
        int y[right+1];
        //cout<<"left mid right "<<left<<" "<<mid<<" "<<
right<<endl;
        //cout<<"mergesort"<<left<<" "<<mid<<endl;</pre>
        MergeSort(x,left,mid);
        //cout<<"mergesort"<<mid+1<<" "<<right<<endl;</pre>
        MergeSort(x,mid+1,right);
```

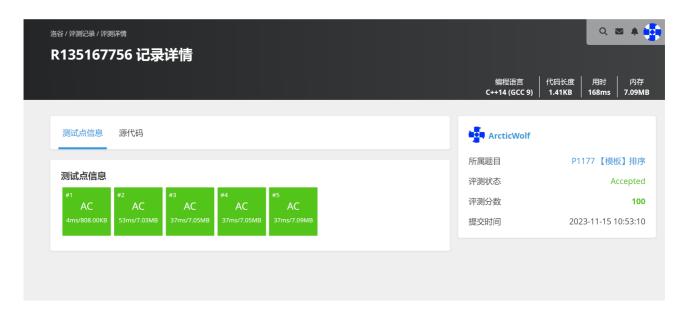
```
Merge(x,y,left,mid,right);
    copy(x,y,left,right);
}

int main()
{
    int n; // 数据量
    cin >> n;
    int x[n];
    for (int i = 0; i < n; i++)
        cin >> x[i];
    MergeSort(x,0,n-1);
    for (int i=0; i<n;i++) cout<<x[i]<<" ";
}</pre>
```

评测

数据规模达到10^9,该算法还是可以满足的。

https://www.luogu.com.cn/problem/P1177



算法分析

时间复杂度预期O(nlogn),最坏情况O(n^2)。

空间复杂度O(n)。

(3) 实现题1-3 最多约数问题

引言

书上这道题与洛谷上的这道题,不同基本一致点在于洛谷的这道题限制了范围,而书上的题目并未设置范围。可以理解为洛谷的题目是书上题目的强化版,故我直接使用洛谷题目来作为实验内容。

题目描述

数学家们喜欢各种类型的有奇怪特性的数。例如,他们认为 945 是一个有趣的数,因为它是第一个所有约数之和大于本身的奇数。

为了帮助他们寻找有趣的数,你将写一个程序扫描一定范围内的数,并确定在此范围内约数 个数最多的那个数。不幸的是,这个数和给定的范围的都比较大,用简单的方法寻找可能需 要较多的运行时间。所以请确定你的算法能在几秒内完成最大范围内的扫描。

输入格式

只有一行,给出扫描的范围,由下界L和上界U确定。满足2<L<U<10^9。

输出格式

对于给定的范围,输出该范围内约数个数 D 最多的数 P。若有多个,则输出最小的那个。 请输出 Between L and U, P has a maximum of D divisors., 其中 L,U,P,D 的含义同前面所述。

样例 #1

样例输入#1

1000 2000

样例输出#1

Between 1000 and 2000, 1680 has a maximum of 40 divisors.

做法1(直接遍历求解+比较)

可以直接采用最朴素的想法,遍历这(b-a+1)个数并分别计算它们的约数数量,再储存最多的。

在处理约数数量的时候,采取很朴素的计算方法:从1到sqrt(n)看是否为n的因子。这个算法的时间复杂度应该是nsqrt(n)。

代码

```
#include <stdio.h>
using namespace std;
int a,b;
              // 符合要求的数的最大约数
int ans = 0;
int ans_num = 0;  // 符合要求的数
void func(){//暴力出奇迹
    for(int i=a;i<=b;i++){</pre>
        int ret=0;
        for(int j=1;j*j<=i;j++){
            if(i\%j==0) ret+=2;
            if(j*j==i) ret--;
        }
        if(ret>ans){
            ans_num=i;
            ans=ret;
        }
    }
}
int main()
{
    scanf("%d %d",&a, &b);
    func();
    printf("Between %d and %d, %lld has a maximum of %lld
divisors.\n",a,b,ans_num,ans);
    return 0;
}
```

使用洛谷评测结果如下:

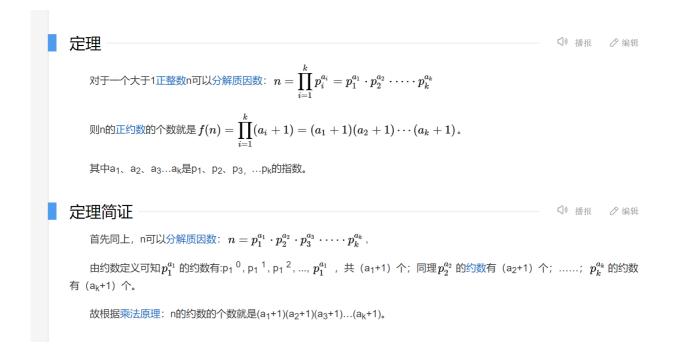


很明显看到4个超时,说明对于较大的数据,这个算法有时间复杂度上的缺陷。这个应该是 O(n^2)的。

做法2(使用约数定理计算)

可以采用朴素的想法,遍历这(b-a+1)个数并分别计算它们的约数数量,再储存最多的。

但是在处理的时候使用到约数定理。(百度可知)



这要求我们先维护一个素数表,保存一定范围内的素数,然后再把原数对素数分别进行反复整除,由此得到原数的质因数分解。一旦分解出质因数,其约数数量也就可以通过公式得到。

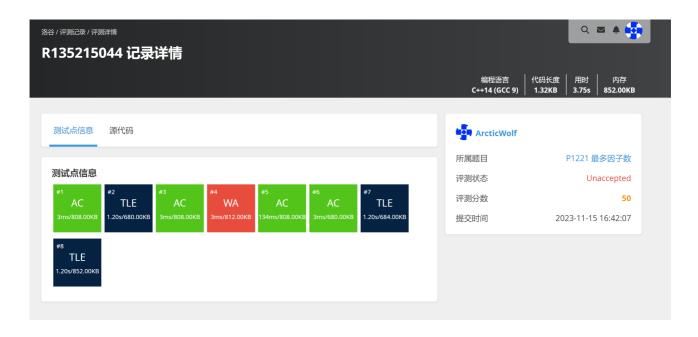
代码

```
#include <stdio.h>
using namespace std;
const int max_num = 1000;
int prime[max_num]; // 维护一张质数表
int prime_total = 0; // 质数表中质数的个数
int a, b;
int ans = 0; // 符合要求的数的最大约数
int ans_num = 0; // 符合要求的数
// 使用筛法筛出一定范围内的质数
void get_primes(int max_prime)
{
    bool flag[max_prime + 1];
    for (int i = 2; i \leftarrow \max_{prime}; i++)
        flag[i] = true;
    for (int i = 2; i \leftarrow \max_{prime}; i++)
        if (flag[i])
        {
            for (int j = i + i; j \leftarrow max\_prime; j \leftarrow i)
                flag[j] = false;
        }
    for (int i = 2; i \leftarrow \max_{prime}; i++)
        if (flag[i])
            prime[prime_total++] = i;
    prime_total--;
}
void search()
{
    for (int i = a; i <= b; i++)
    {
        long long now = i; // 当前待处理数
        long long total = 1; // 当前数的公约数数量
        for (int j = 0; j <= prime_total; j++)</pre>
```

```
{
            int num = 1;
            while (now % prime[j] == 0)
            {
                now /= prime[j];
                num++;
            }
            total *= num;
        }
        if (total > ans)
        {
            ans = total;
            ans_num = i;
        }
    }
}
int main()
{
    scanf("%d %d", &a, &b);
    get_primes(100);
    search();
    printf("Between %d and %d, %lld has a maximum of %lld
divisors.\n", a, b, ans_num, ans);
    return 0;
}
```

评测

评测结果如下:



中间的错误点应该是涉及到较大的质数了(这个问题在后面一个做法中提出,一旦使用约数定理来计算这个,难免会遇到这个问题),可以看到总体上会比之前要好一些。

做法3(约数定理+DFS)

【更大数据量,限制时间】

将寻找在范围内约数的过程看作一个深度优先搜索,某数的不同质因数可以看作不同层,每一层是一个质因数,对于每一层而言,可以选取0个,1个,.....直到快要超出右边界为止。

举例来说,对于一个确定的数140

- 第一层是质数2,取了2个,现在是1*4=4;
- 第二层是质数3,取了0个;
- 第三层是质数5,取了1个,现在是4*5=20;
- 第四层是质数7,取了1个,现在是20*7=140;
- 总结来说140=2^2+3^0+5^1+7^1

对于一个范围[3,5]

- $3=2^0+3^1+5^0$;
- 4=2^2+3^0+5^0;
- $5=2^0+3^0+5^1$;

使用这样的方法搜索所有的可能结果,直到所有的结果都会被计算完,然后就可以比较出最大值。由于这种方法试图使用自下而上做出结果,会比之前的方法快。(略去了计算每一个数的质因数的时间)。

```
#include <stdio.h>
using namespace std;
const int max_num = 1000;
int prime[max_num]; // 维护一张质数表
int prime_total = 0;// 质数表中质数的个数
int a,b;
int ans = 0; // 符合要求的数的最大约数
int ans_num = 0;
                 // 符合要求的数
// 使用筛法筛出一定范围内的质数
void get_primes(int max_prime){
    bool flag[max_prime+1];
   for (int i=2;i<=max_prime;i++) flag[i] = true;</pre>
    for (int i=2;i<=max_prime;i++)</pre>
       if(flag[i]){
           for (int j=i+i; j<=max_prime; j+=i)</pre>
               flag[i]=false;
    }
    for (int i=2;i<=max_prime;i++)</pre>
       if (flag[i]) prime[prime_total++]=i;
   prime_total--;
}
//对每一个状态进行列举,并比较记录各个状态中最小的部分
void search(int depth,long long num,long long sum)
{
   // 对于合法枚举结果的比较处理 (到达终点)
    if (a \le num & num \le b)
       if (sum>ans) {ans=sum; ans_num=num;}
       else if (sum==ans && num<ans_num) {ans_num=num;}</pre>
    }
   if (depth>prime_total) return; // 搜索超出规定层数,直接返回
   if (num*prime[depth]>b) return; // 如果本层的质数乘上去都超范围了,那
么之后的质数乘上去必然超范围,故直接返回,且本层没有结果
    long long new_num = num;
   // 以当前num为基础,分别乘上i个prime[depth],进行深搜
    search(depth+1, num, sum);
    int i=0;
    while(new_num*prime[depth]<=b){</pre>
```

```
new_num*=prime[depth];
    i++;
    search(depth+1,new_num,sum*(i+1));
}

int main()
{
    scanf("%d %d",&a, &b);
    get_primes(b);
    search(0,1,1);
    printf("Between %d and %d, %lld has a maximum of %lld divisors.\n",a,b,ans_num,ans);
    return 0;
}
```

★说明:这段代码能够处理所有情况,但是它的代价就是时间会超很多,主要在于素数表中存在的素数量过多了,要遍历很多素数。如果将get_primes()的输入参数调整为100,能解决大多数问题。这是因为即使对于2*3*5*7*11*13*17*19*23*29的结果也大于10^9。

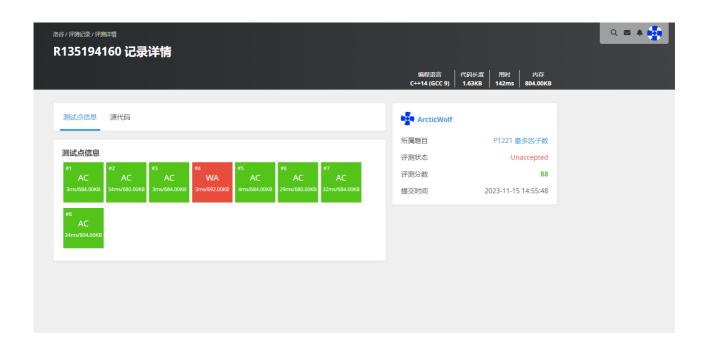
无法解决的是如果限定a==b(即a=b相差很小),并且其中有一个因数是很大的质数(比如下面评测中遇到的例子)。

这是一个需要权衡的问题,暂时不能找到一种很好的算法把这两个问题同时解决。

评测

最多因子数(https://www.luogu.com.cn/problem/P1221)

使用洛谷评测,会发现有一个点有问题。

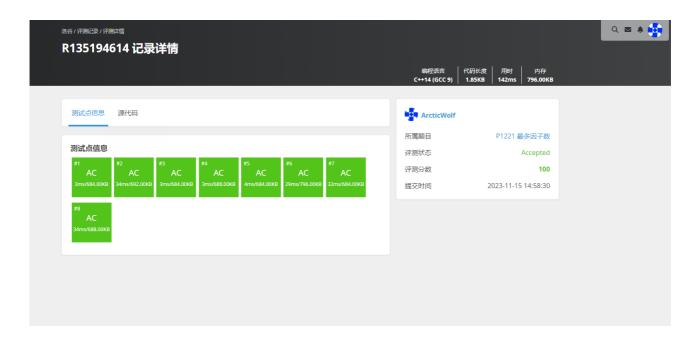


该点是a==b==131074,它的因子是1,2,65537,131074,包含一个很大的质数65537,而我们对质数的考虑最多到100(100往上可能造成TLE)

故对特例进行特判(不得已而为之,照理不能这样做的)

```
if(a == b && b == 131074){//小小的特判,因为这个数好像撞到了100以上的质数,如果想过的话可以求出更大的质数 ans_num = 131074,ans = 4; printf("Between %lld and %lld, %lld has a maximum of %lld divisors.",a,b,ans_num,ans); return 0; }
```

这样的结果可以达到AC



还有一种方法是在a和b很接近的时候进行暴力求解(参考前面一种方法),根据a和b的接近程度来决定使用哪种方法,这是一种比较好的求解方法了。

实验感悟

只是课上听讲跟实际上手操作还是有差距的,我已经在很多课程上有这个感悟了,算法是一个计科学生不得不掌握的底层技能,还是要多花时间的。