分治与二分

化繁为简 分而治之

中国传媒大学 PRIMO PAN （潘东逸杰）FROM CUC-ANIMA (DIGITAL MEDIA ART)

About Me

Primo Pan 潘东逸杰

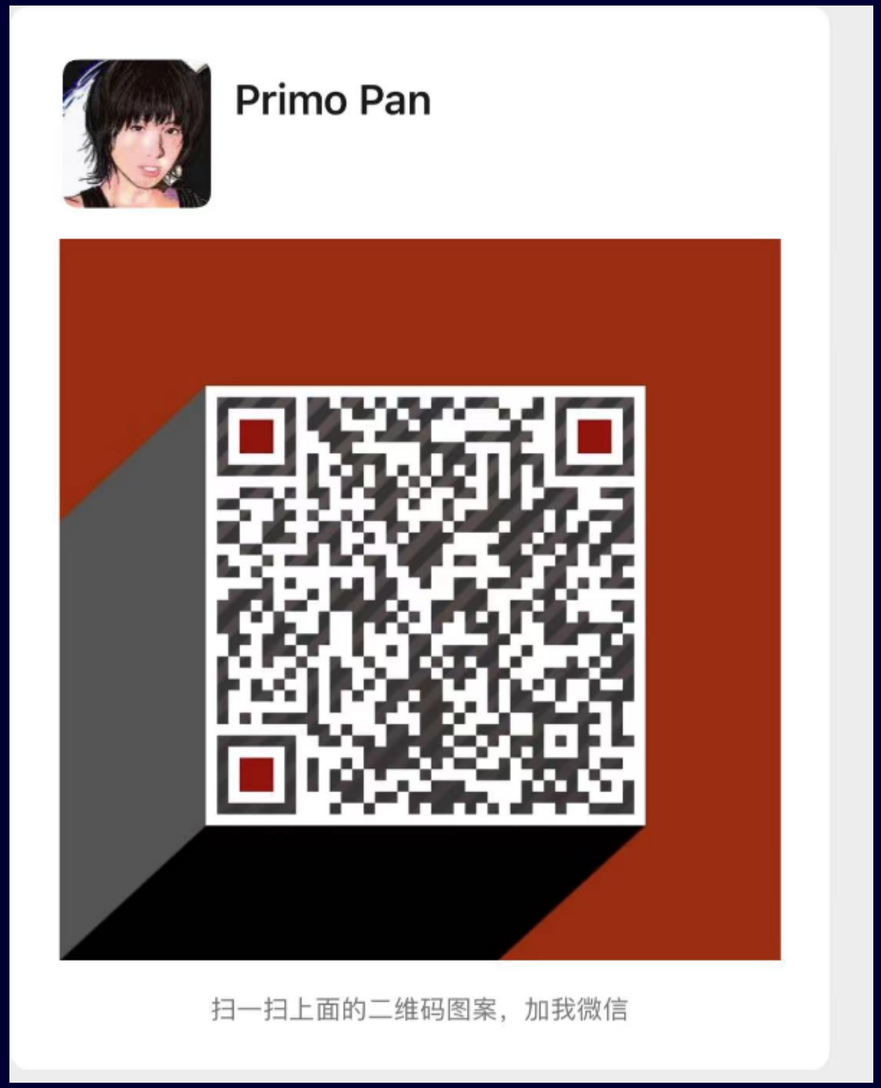
动画与数字艺术学院　20数媒网络

Codeforces ID: primojaypan

QQ：898021802

WeChat: primojaypan

About Me



从高中课本讲起————二分查找

二分查找（Binary Search）

给一个长度为16的整形数组，已知数组内元素单调递增，现给定一个数m，查询该数组是否含有m这个元素，如果有，输出m元素的下标，如果没有，输出ORZ.

二分查找（Binary Search）

给一个长度为16的整形数组，已知数组内元素单调递增，现给定一个数m，查询该数组是否含有m这个元素，如果有，输出m元素的下标，如果没有，输出ORZ.

Index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Array	1	2	5	9	11	12	17	19	24	25	29	30	33	39	40	114514

m=1,m=114514,m=19?

二分查找（Binary Search）

```cpp
#include <iostream>
#include <cstdio>
#include <cstring>

using namespace std;

int A[17]={0,1,2,5,9,11,12,17,19,24,25,29,30,33,39,40,114514};
int m;
int main()
{
    cin>>m;
    //linear search
    for (int i=1;i<=16;i++)
    {
        if (A[i]==m)
        {
            printf("%d\n",i);
            return 0;
        }
    }
    printf("NULL\n");
}

```

二分查找（Binary Search）

Index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Array	1	2	5	9	11	12	17	19	24	25	29	30	33	39	40	114514

一个一个找？看上去没啥问题，循环一下即可，大家都会的吧～
但如果数组范围更大呢，不是16，而是1E6,1E8，我们还要做1E8次循环吗？

注意观察，我们要查找的这个数组，具有什么性质？
有114514？很臭？
（BUSHI
它具有单调性，我们需要把单调性这个性质用起来！

二分查找（Binary Search）

Index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Array	1	2	5	9	11	12	17	19	24	25	29	30	33	39	40	114514

一个一个找？看上去没啥问题，循环一下即可，大家都会的吧～
但如果数组范围更大呢，不是16，而是1E6,1E8，我们还要做1E8次循环吗？

我们现在来玩一个游戏，我在心里想一个1～1000的整数，你来猜。
我每次会告诉你，你猜的这个数，和我想的这个数，是大了还是小了。
如果你猜到了，游戏结束。
作为猜数者，你会怎么做？

二分查找（Binary Search）

Index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Array	1	2	5	9	11	12	17	19	24	25	29	30	33	39	40	114514

一个一个找？看上去没啥问题，循环一下即可，大家都会的吧～
但如果数组范围更大呢，不是16，而是1E6,1E8，我们还要做1E8次循环吗？

显然，每次取中间那个数，然后根据我给你的信息，缩小你猜数的区间；
如果你猜了500，我告诉你大了，你就会把区间缩小到1～499
然后再猜中位数，依此类推，很快就能猜到答案～～

二分查找（Binary Search）

```cpp
#include <iostream>
#include <cstdio>
#include <cstring>

using namespace std;

int A[17]={0,1,2,5,9,11,12,17,19,24,25,29,30,33,39,40,114514};
int m;
int main()
{
    cin>>m;
    int L=1,R=16;

    while (L<=R)
    {
        int mid=(L+R)/2;
        if (A[mid]==m)
        {
            printf("%d\n",mid);
            return 0;
        }else if (A[mid]>m) R=mid-1;
        else L=mid+1;

    }
    puts("NULL");
    return 0;
}
```

猜数字游戏?

你在心里想一个不超过1000的正整数，我可以保证在10次以内找到它——只要你每次告诉我我猜的数比你想的大一点，小一点或者正好猜中。

首先我猜500，除了运气特别好正好猜中外，不管你说“太大”还是“太小”，我都可以把可行范围缩小一半，如果“太大”，那么答案就在1～499之间，如果太小，答案就在501～1000之间，只要每次选择中点去猜，范围就缩小了一半！

猜数字游戏?

为啥能确保我们在10次以内能够猜到呢?

$log_2(1000) < 10$

如果n是10000，100000，10000000?

不难得出，最坏情况的查找次数即为$log_2(n)$

所以二分过程的时间复杂度为$O(logn)$

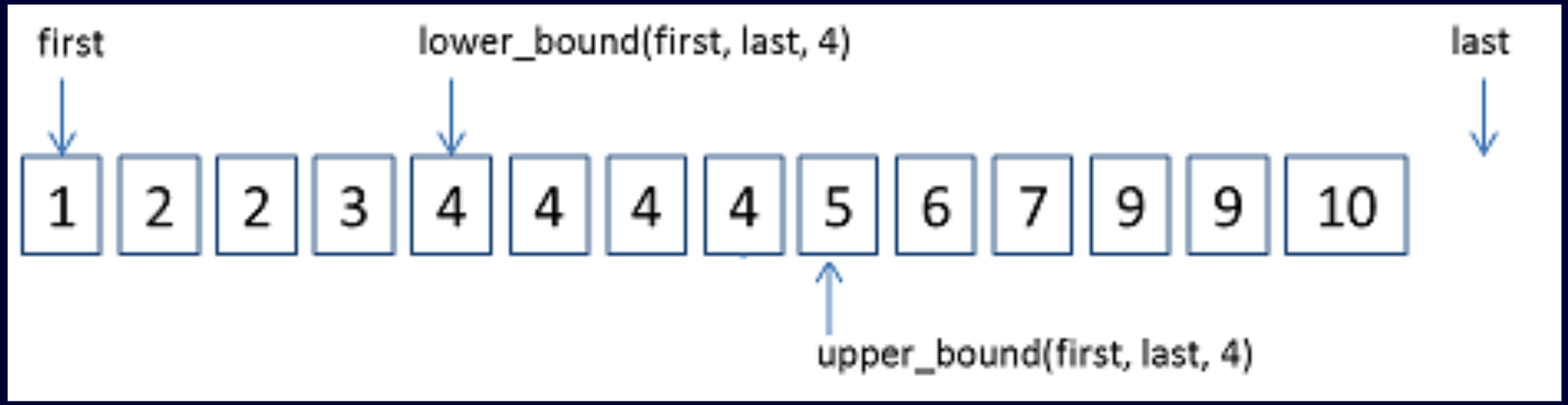
STL库函数 （#include <algorithm>）

C++的库函数中为我们准备了二分查找的库函数.

lower_bound() 和 upper_bound()

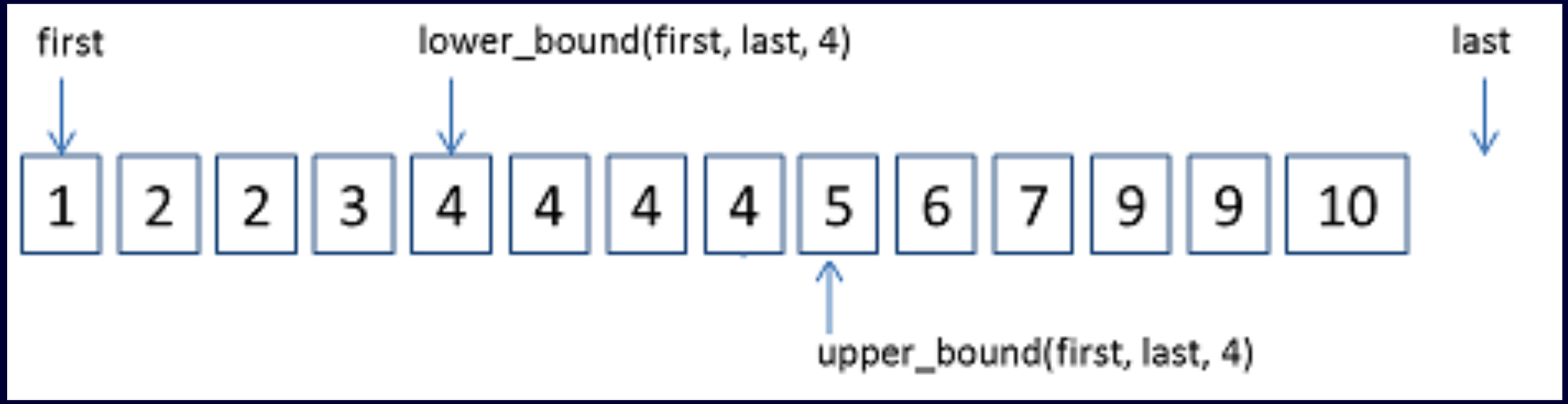
lower_bound(a, a+n, x)：返回数组a[0]~a[n-1]中，【大于等于】x的数中，最小的数的指针

upper_bound(a, a+n, x)：返回数组a[0]~a[n-1]中，【大于】x的数中，最小的数的指针



STL库函数

初始下标为1

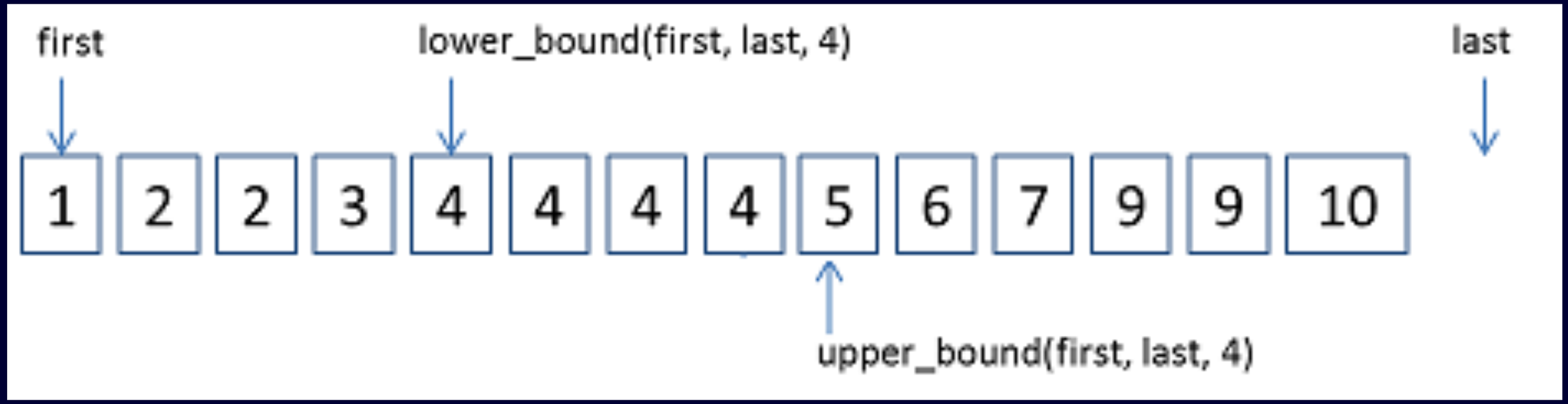


index_1=lower_bound(a+1,a+1+14,4)-a;

index_2=upper_bound(a+1,a+1+14,4)-a;

STL库函数

初始下标为0



index_1=lower_bound(a,a+14,4)-a;

index_2=upper_bound(a,a+14,4)-a;

STL库函数

x=lower_bound(begin, end, val)-数组名;

y=upper_bound(begin, end, val)-数组名;

Practice

洛谷 / 题目列表 / 题目详情

P2249 【深基13.例1】查找

提交答案 | 加入题单 | 复制题目

题目描述

展开

输入 $n(n \leq 10^6)$ 个不超过 10^9 的单调不减的（就是后面的数字不小于前面的数字）非负整数 $a_1, a_2, \ldots, a_n$，然后进行 $m(m \leq 10^5)$ 次询问。对于每次询问，给出一个整数 $q(q \leq 10^9)$，要求输出这个数字在序列中第一次出现的编号，如果没有找到的话输出 -1 。

输入格式

第一行 2 个整数 n 和 m，表示数字个数和询问次数。

第二行 n 个整数，表示这些待查询的数字。

第三行 m 个整数，表示询问这些数字的编号，从 1 开始编号。

输出格式

m 个整数表示答案。

输入输出样例

输入 #1	复制	输出 #1	复制
11 3 1 3 3 3 5 7 9 11 13 15 15 1 3 6		1 2 -1	

离散化（Discrete）

输入n个数($n <= 10000$)，保证每个数x满足（$1e9 <= x <= 1e9 + 1000$），要求你给出这个范围内，每一个出现过的数字的出现次数，从小到大输出。

离散化（Discrete）

如果每个数的范围在0到1000以内，我们完全可以以它们为下标，cnt[x]就表示x的出现次数，但现在x太大，我们无法开一个长度为1e9的数组，怎么办呢?

离散化（Discrete）

这个时候，我们就需要用到离散化。

通俗地讲就是当有些数据因为本身很大或者类型不支持，自身无法作为数组的下标来方便地处理，而影响最终结果的只有元素之间的相对大小关系时，我们可以将原来的数据按照从大到小编号来处理问题，即离散化。

离散化（Discrete）

离散化数组

将一个数组离散化，并进行查询是比较常用的应用场景：

```
// a[i] 为初始数组,下标范围为 [1, n]
// len 为离散化后数组的有效长度
std::sort(a + 1, a + 1 + n);

len = std::unique(a + 1, a + n + 1) - a - 1;
// 离散化整个数组的同时求出离散化后本质不同数的个数。
```

离散化（Discrete）

在完成上述离散化之后可以使用 `std::lower_bound` 函数查找离散化之后的排名（即新编号）：

```
1  std::lower_bound(a + 1, a + len + 1, x) - a;  // 查询 x 离散化后对应的编号
```

离散化（Discrete）

```
41  int A[1005];
42  int B[10005];
43  int C[10005];
44  int n;
45  long long N;
46  int main()
47  {
48      cin>>n;
49      for (int i=1;i<=n;i++)
50      {
51          cin>>B[i];
52          C[i]=B[i];//将B中数据迁移至C中
53      }
54      sort(C+1,C+1+n);//排序
55      N=unique(C+1,C+n+1)-C-1;//将C中所有存在的元素按顺序放到C[1]到C[N]中
```

离散化（Discrete）

```
54      sort(C+1,C+1+n);//排序
55      N=unique(C+1,C+n+1)-C-1;//将C中所有存在的元素按顺序放到C[1]到C[N]中
56      for (int i=1;i<=n;i++)
57      {
58          int x=lower_bound(C+1,C+1+N,B[i])-C;//寻找B[i]在C[1]到C[N]中的位置
59          A[x]++;//A是计数数组
60      }
61
62      for (int i=1;i<=N;i++)
63      {
64          cout<<C[i]<<" "<<A[i]<<endl;
65      }
66      return 0;
67  }
68
```

二分答案

很多时候，题目会让你去求解一个ans，这个ans有以下一些特征。比如求一个满足条件A的最大值/最小值；或者求一个**最大的最小值/最小的最大值**，并且我们**直接求解答案A非常困难**，并且ans满足**单调性**，那么我们不妨直接二分查找最后的答案，测试它是不是满足条件。这种直接二分最后答案的算法就叫**二分答案**。

二分答案

Luogu P1873 砍树

伐木工人米尔科需要砍倒 M 米长的木材。这是一个对米尔科来说很容易的工作，因为他有一个漂亮的新伐木机，可以像野火一样砍倒森林。不过，米尔科只被允许砍倒单行树木。

米尔科的伐木机工作过程如下：米尔科设置一个高度参数 H（米），伐木机升起一个巨大的锯片到高度 H，并锯掉所有的树比 H 高的部分（当然，树木不高于 H 米的部分保持不变）。米尔科就行到树木被锯下的部分。

例如，如果一行树的高度分别为 $20, 15, 10, 17$，米尔科把锯片升到 15 米的高度，切割后树木剩下的高度将是 $15, 15, 10, 15$，而米尔科将从第 1 棵树得到 5 米木材，从第 4 棵树得到 2 米木材，共 7 米木材。

米尔科非常关注生态保护，所以他不会砍掉过多的木材。这正是他尽可能高地设定伐木机锯片的原因。你的任务是帮助米尔科找到伐木机锯片的最大的整数高度 H，使得他能得到木材至少为 M 米。即，如果再升高 1 米锯片，则他将得不到 M 米木材。

二分答案

解题思路

我们可以在 1 到 10^9 中枚举答案，但是这种朴素写法肯定拿不到满分，因为从 1 枚举到 10^9 太耗时间。我们可以在 $[1,\ 10^9]$ 的区间上进行二分作为答案，然后检查各个答案的可行性（一般使用贪心法）。**这就是二分答案。**

二分答案

```
10
11  int find() {
12    int l = 1, r = 1e9 + 1;    // 因为是左闭右开的，所以 10^9 要加 1
13    while (l + 1 < r) {        // 如果两点不相邻
14      int mid = (l + r) / 2;   // 取中间值
15      if (check(mid))          // 如果可行
16        l = mid;               // 升高锯片高度
17      else
18        r = mid;  // 否则降低锯片高度
19    }
20    return l;  // 返回左边值
21  }
```

二分答案

想一想，为啥区间是左闭右开？为啥返回L(左端值）？

二分答案

合法			不合法			
最小值	L	MID	R	……	……	最大值
		或者				

合法			不合法			
最小值	……	……	L	MID	R	最大值

然后会

合法			不合法			
最小值	……	L,MID	R	……	……	最大值
		或者				

合法			不合法			
最小值	……	……	L	MID,R	……	最大值

二分答案

返回L?返回R?区间开闭？二分答案具体怎么写？稍不留神就会死循环或者WRONG ANSWER，好难啊！

具体问题，具体分析！多去动手试一试，积累经验！写多了就不觉得难了！

二分答案

题目描述 （LUOGU P1281）

现在要把 M 本有顺序的书分给 K 个人复制（抄写），每一个人的抄写速度都一样，一本书不允许给两个（或以上）的人抄写，分给每一个人的书，必须是连续的，比如不能把第一、第三、第四本书给同一个人抄写。

现在请你设计一种方案，使得复制时间最短。复制时间为抄写页数最多的人用去的时间。

二分答案

输入格式

第一行两个整数 m, k。

第二行 m 个整数，第 i 个整数表示第 i 本书的页数。

输出格式

共 k 行，每行两个整数，第 i 行表示第 i 个人抄写的书的起始编号和终止编号。 k 行的起始编号应该从小到大排列，如果有多解，则尽可能让前面的人少抄写。

输入输出样例

输入 #1 复制

```
9 3
1 2 3 4 5 6 7 8 9
```

输出 #1 复制

```
1 5
6 7
8 9
```

二分答案

题目描述 （LUOGU P1281）

现在要把 M 本有顺序的书分给 K 个人复制（抄写），每一个人的抄写速度都一样，一本书不允许给两个（或以上）的人抄写，分给每一个人的书，必须是连续的，比如不能把第一、第三、第四本书给同一个人抄写。

现在请你设计一种方案，使得复制时间**最短**。复制时间为抄写页数**最多**的人用去的时间。

简而言之，就是让抄书时间**最多**的人的用时**最短**?

这是一个**最小的最大值**问题！满足二分答案的性质！

怎么做呢?

二分答案

简而言之，就是让抄书时间最多的人的用时最短？
这是一个最小的最大值问题！满足二分答案的性质！
怎么做呢？

先确定二分答案的上下界吧，下界是啥？
就是抄书时间最多的人，至少要用多长时间？
显然，是耗时最长的一本书的时间！

那最大值呢？
假设我们就只有一个倒霉鬼要把所有书都抄完；
那么最大值就是所有书的耗时之和！

二分答案

先确定二分答案的上下界吧，下界是啥?
就是抄书时间最多的人，至少要用多长时间?
显然，是耗时最长的一本书的时间!

那最大值呢?
假设我们就只有一个倒霉鬼要把所有书都抄完;
那么最大值就是所有书的耗时之和!

好的，现在我们有了L和R，每次就二分出了一个MID值
接着，我们去检验这个MID值

二分答案

好的，现在我们有了L和R，每次就二分出了一个MID值
接着，我们去检验这个MID值

进入到CHECK时间！

想一想我们二分出的MID值是个啥
是所有人里面，抄书时间最长的人不能超过的这个值。
那么我们每次扫一遍数组，记录当前这个人抄书的时间
一旦他/她/它所用的时间超过了MID，我们就得换人，把人数减去1

如果到最后，我们的K个人还够用，那么这个MID就是可行的
RETURN TRUE
如果K个人不够用了，那么在循环中,K一定已经用完了
一旦K用完，立刻RETURN FALSE!

二分答案

```
    for (int i = 1; i <= m; i++){
        scanf("%d", &a[i]);
        r+=a[i];
        l=max(l,a[i]);
    }
    while (l <= r)
    {
        mid = (l + r) / 2;
        if (check(mid))
            r = mid - 1, ans = mid;
        else
            l = mid + 1;
    }
```

二分答案

```
bool check(int p)
{
    int now = k;//还剩多少人能抄书(包括正在抄书的人)
    int cnt = 0;//这个人要抄书的页数
    for (int i = m; i > 0; i--)
    {
        cnt += a[i];
        if (cnt > p)
        {
            cnt = a[i];
            now--;
        }

        if (now <= 0)return false;//书还没抄完，但已经没有人可以去抄书了
    }
    if(cnt > p)return false;
    return true;
}
```

Deactivate breakpoints

二分答案

那么，聪明的，你告诉我
这样做的时间复杂度怎么算捏?

刚才提过二分的复杂度是O(logN)
线形扫一遍的复杂度是O(N)

学霸题，算复杂度。
头顶标注法，从上往下数
结果乘起来
答案等于O(N logN)

二分答案

HAVE A TRY

LUOGU P1182

20MIN

二分答案

总结一下二分答案的模板吧

```
while(l<r){
    int mid=(l+r)>>1;
    if(check(mid))r=mid;
    else l=mid+1;
}
cout<<l<<endl;
```

分治（Divide-and-Conquer）

分治

分治分治，分而治之。
如果一个问题，我们把它缩小一定的范围，这个问题的性质却并没有变化。
我们就可以把它分解成若干的子问题。
子问题又分成子问题。
子子孙孙，（无穷尽也）到边界停止，然后再合并，MERGE!
分而治之，逐个击破，就是分治算法的精髓。

分治

分治分治，分而治之。
如果一个问题，我们把它缩小一定的范围，这个问题的性质却并没有变化。
我们就可以把它分解成若干的子问题。
子问题又分成子问题。
子子孙孙，（无穷尽也）到边界停止，然后再合并，MERGE!
分而治之，逐个击破，就是分治算法的精髓。
上节课杨队讲过的快速排序、归并排序，都利用了分治思想，不再赘述

各位思考一下，我们刚刚讲的二分，是否具有分治的味道?
分治多基于递归，所以各位一定要熟练掌握递归的写法!

分治

HANOI TOWER

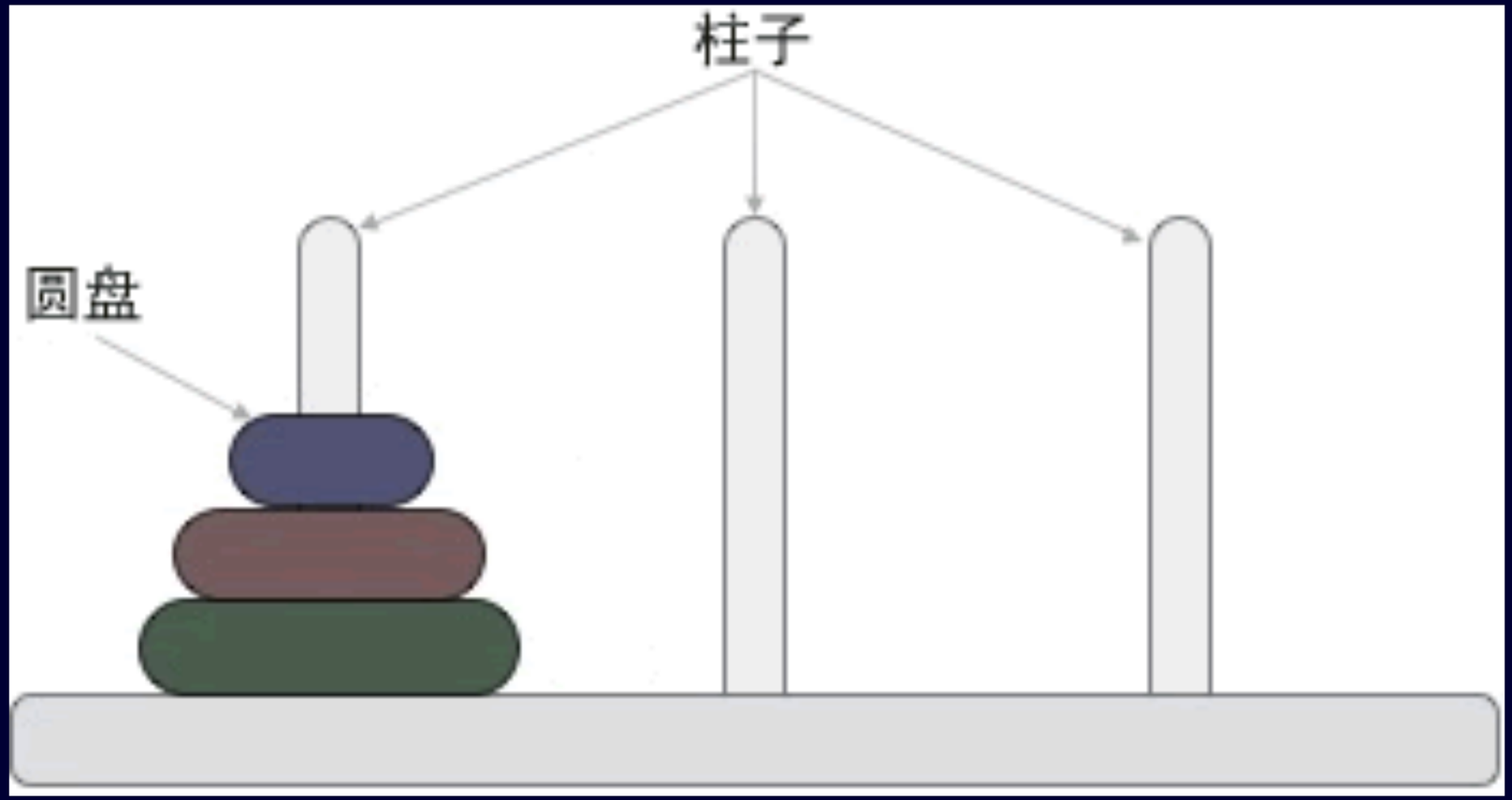
汉诺塔问题源自印度一个古老的传说，印度教的“创造之神”梵天创造世界时做了 3 根金刚石柱，其中的一根柱子上按照从小到大的顺序摞着 64 个黄金圆盘。梵天命令一个叫婆罗门的门徒将所有的圆盘移动到另一个柱子上，移动过程中必须遵守以下规则：

每次只能移动柱子最顶端的一个圆盘；

每个柱子上，小圆盘永远要位于大圆盘之上；

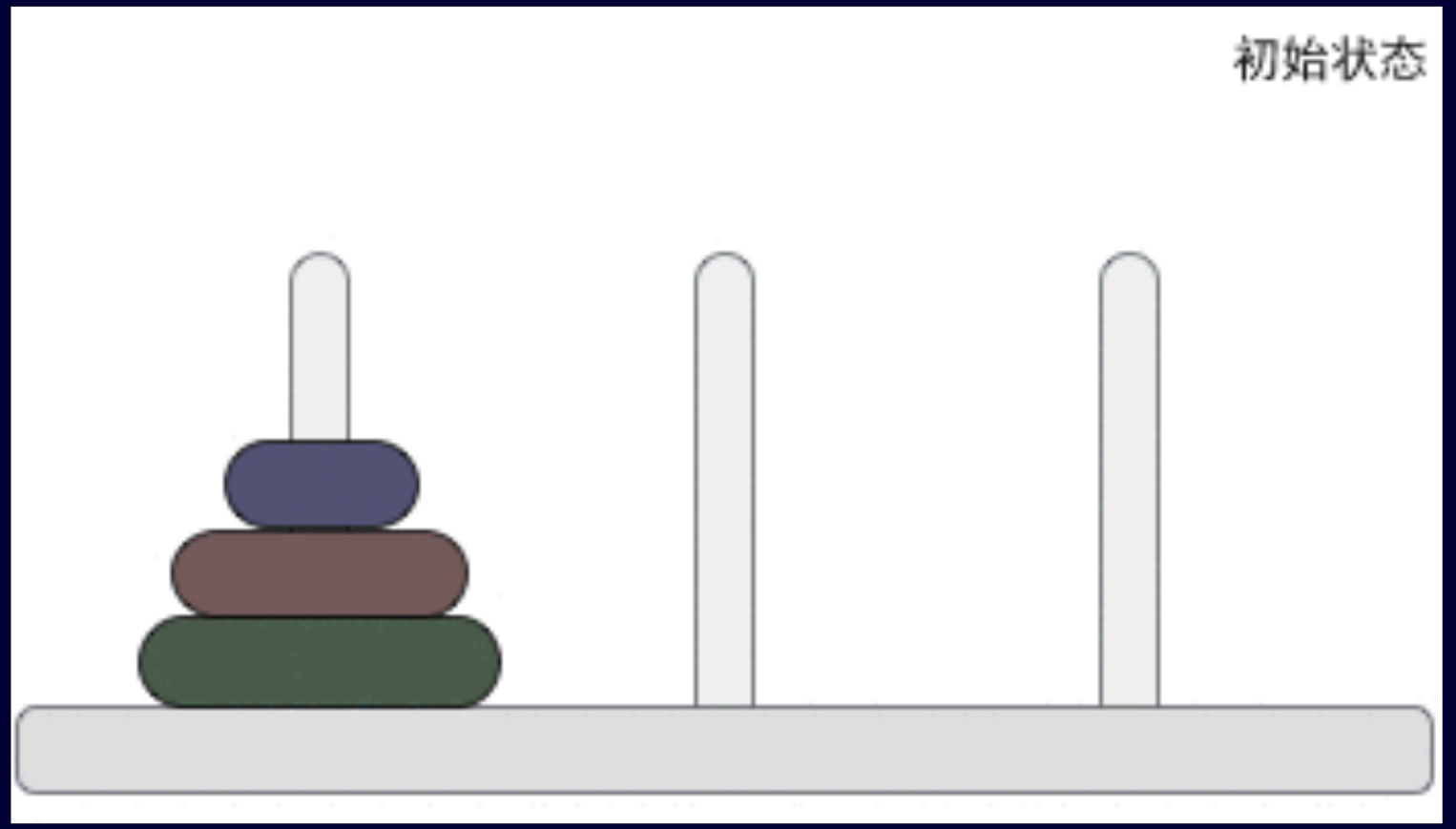
分治

HANOI TOWER



分治

HANOI TOWER



分治

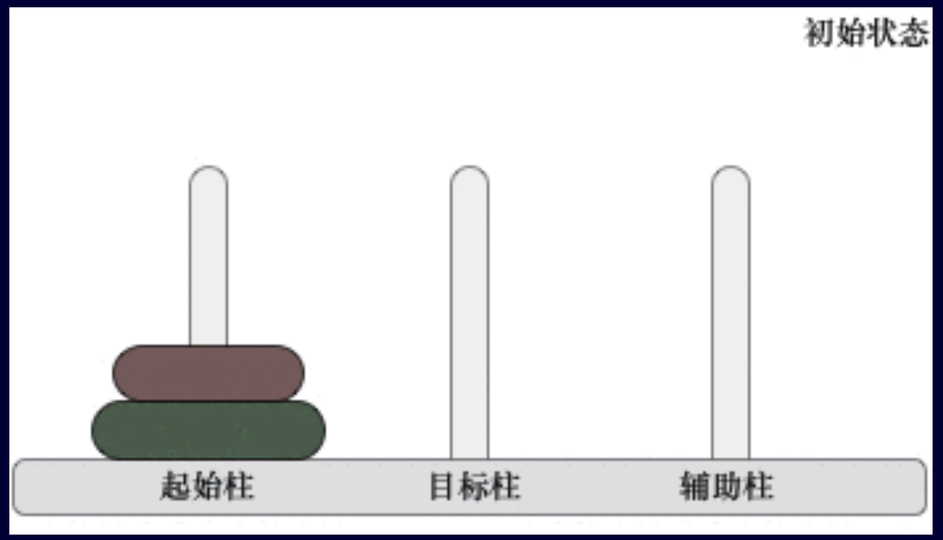
HANOI TOWER

为了方便讲解，我们将 3 个柱子分别命名为起始柱、目标柱和辅助柱。

实际上，解决汉诺塔问题是有规律可循的：

1) 当起始柱上只有 1 个圆盘时，我们可以很轻易地将它移动到目标柱上；

2) 当起始柱上有 2 个圆盘时，移动过程如下图所示：



分治

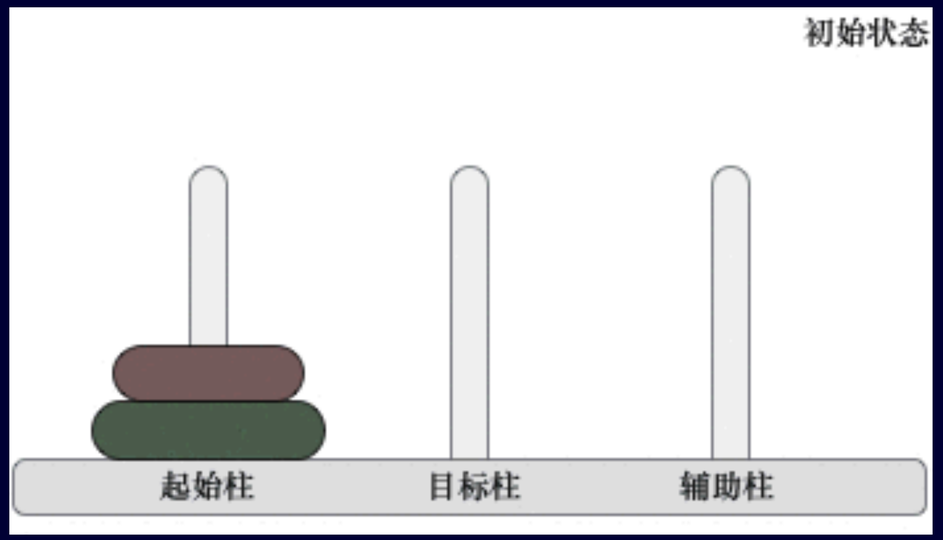
HANOI TOWER

为了方便讲解，我们将 3 个柱子分别命名为起始柱、目标柱和辅助柱。

实际上，解决汉诺塔问题是有规律可循的：

1) 当起始柱上只有 1 个圆盘时，我们可以很轻易地将它移动到目标柱上；

2) 当起始柱上有 2 个圆盘时，移动过程如下图所示：



分治

HANOI TOWER

移动过程是：先将起始柱上的 1 个圆盘移动到辅助柱上，然后将起始柱上遗留的圆盘移动到目标柱上，最后将辅助柱上的圆盘移动到目标柱上。

3) 当起始柱上有 3 个圆盘时，移动过程如图 2 所示，仔细观察会发现，移动过程和 2 个圆盘的情况类似：先将起始柱上的 2 个圆盘移动到辅助柱上，然后将起始柱上遗留的圆盘移动到目标柱上，最后将辅助柱上的圆盘移动到目标柱上。

分治

HANOI TOWER

通过分析以上 3 种情况的移动思路，可以总结出一个规律：对于 N 个圆盘的汉诺塔问题，移动圆盘的过程是：
将起始柱上的 N-1 个圆盘移动到辅助柱上；
将起始柱上遗留的 1 个圆盘移动到目标柱上；
将辅助柱上的所有圆盘移动到目标柱上。

由此，N 个圆盘的汉诺塔问题就简化成了 N-1 个圆盘的汉诺塔问题。按照同样的思路，N-1 个圆盘的汉诺塔问题还可以继续简化，直至简化为移动 3 个甚至更少圆盘的汉诺塔问题。

分治

HANOI TOWER

```
// NUM 表示移动圆盘的数量，SOURCE、TARGET、AUXILIARY 分别表示起始柱、目标柱和辅助柱
HANOI(NUM , SOURCE , TARGET , AUXILIARY):
  IF NUM == 1:    // 如果圆盘数量仅有 1 个，则直接从起始柱移动到目标柱
    PRINT(从 SOURCE 移动到 TARGET)
  ELSE:
    // 递归调用 HANOI 函数，将 NUM-1 个圆盘从起始柱移动到辅助柱上，整个过程的实现可以借助目标柱
    HANOI(NUM-1 , SOURCE , AUXILIARY , TARGET)
    // 将起始柱上剩余的最后一个大圆盘移动到目标柱上
    PRINT(从 SOURCE 移动到 TARGET)
    // 递归调用 HANOI 函数，将辅助柱上的 NUM-1 圆盘移动到目标柱上，整个过程的实现可以借助起始柱
    HANOI(N-1 , AUXILIARY , TARGET , SOURCE)
```

THANKS~~