二分与分治

陈业元

June, 2019

- 求a^n?

- 求a^n?
- $n \leq 10^6$?

- 求a^n?
- $n \leq 10^6$?
- 循环即可

- 求a^n?
- $n \leq 10^6$?
- 循环即可
- $n \leq 10^{1000000}$?

- 求a^n?
- $n \leq 10^6$?
- 循环即可
- $n \leq 10^{1000000}$?
- 快速幂!

快速幂

快速幂解决的问题是：给定a和n，求a^n的值。算法的时间复杂度为$O(\log n)$。

快速幂的思想是基于分治的，它的正确性依据只是幂数的运算法则，非常简单。

如果n是偶数，可以把a^n转化为$(a^2)^{n/2}$。

如果n是奇数，可以把a^n转化为$a * a^{n-1}$。

显然，这可以将运算次数降低到$O(\log n)$的级别。

至于代码实现，我们初始化答案为$res = 1$，它的意义是令$res = a^0$。当n是偶数时，我们令$a = a^2$，$n = n/2$；当n是奇数时，我们令$res = res * a$，$n = n - 1$。不断重复直到$n = 0$为止。由于当n为奇数时，$n - 1$必为偶数，因此我们可以将奇数和偶数规约成一种情况：每次如果n为奇数那么将$res = res * a$,然后$a = a^2, n = \lfloor \frac{n}{2} \rfloor$

至于代码实现，我们初始化答案为$res = 1$，它的意义是令$res = a^0$。
当n是偶数时，我们令$a = a^2$，$n = n/2$；当n是奇数时，我们令$res = res * a$，$n = n - 1$。不断重复直到$n = 0$为止。由于当n为奇数时，$n - 1$必为偶数，因此我们可以将奇数和偶数规约成一种情况：每次如果n为奇数那么将$res = res * a$,然后$a = a^2, n = \lfloor \frac{n}{2} \rfloor$。
快速幂的原理也可以从二进制的角度去分析，但这样就比较复杂了。
首先我们要知道，任意一个正整数x都可以表示为$x = \sum_{i=0}^{\log n} c_i * 2^i$的形式，其中$c_i = 0, 1$。例如13的二进制表示为1101，
即$13 = 2^3 + 2^2 + 0 + 2^0$。
根据上面的原理，我们可以对a^n的幂次n进行分解，即：
$a^n = a^{\sum_{i=0}^{\log n} c_i * 2^i} = \prod_{i=0}^{\log n} a^{c_i * 2^i}$。
这证明了运算次数可以降低到$O(\log n)$的级别。

```
int pow(int a,int b)
{
        int ans=1;
        while(b)
        {
                if(b&1)ans=1LL*ans*a%p;
                a=1LL*a*a%p;
                b>>=1;
        }
        return ans;
}
```

求$(a+b\sqrt{c})^n$，所有数字值域10^9

求$(a+b\sqrt{c})^n$，所有数字值域10^9

- $(a_1+b_1\sqrt{c})(a_2+b_2\sqrt{c})=a_1a_2+b_1b_2c+(a_1b_2+a_2b_1)\sqrt{c}$

求$(a+b\sqrt{c})^n$，所有数字值域10^9

- $(a_1+b_1\sqrt{c})(a_2+b_2\sqrt{c})=a_1a_2+b_1b_2c+(a_1b_2+a_2b_1)\sqrt{c}$
- 永远是$x+y\sqrt{c}$的形式，我们开一个专门的结构体存储这种数字即可，然后进行普通快速幂

矩阵快速幂

求n阶矩阵$A = a_{ij}$的p次方，矩阵乘法一次$O(n^3)$

矩阵快速幂

求n阶矩阵$A = a_{ij}$的p次方，矩阵乘法一次$O(n^3)$

时间复杂度$O(n^3 logp)$

求斐波那契数列的第n项，$n \leq 10^{10000}$

求斐波那契数列的第n项，$n \leq 10^{10000}$

- $\begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_{n-1} \\ F_{n-2} \end{pmatrix}$

求斐波那契数列的第n项，$n \leq 10^{10000}$

- $\begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_{n-1} \\ F_{n-2} \end{pmatrix}$
- $\begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n-1} \begin{pmatrix} 1 \\ 0 \end{pmatrix}$

求斐波那契数列的第n项，$n \leq 10^{10000}$

- $\begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_{n-1} \\ F_{n-2} \end{pmatrix}$
- $\begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n-1} \begin{pmatrix} 1 \\ 0 \end{pmatrix}$
- 矩阵快速幂即可，复杂度$O(logn)$

请求出下列数列的第$n \leq 10^{10000}$项

- $a_n = 3a_{n-1} + 4a_{n-2} + 8a_{n-3}$
- $a_n = 3a_{n-1} + 4a_{n-2} + 8n^2 + 6n + 9$
- $a_n = 3S_{n-1} + 5a_{n-1} + 6, S_n = \sum_{i=1}^{n} a_i$
- $a_n = a_{n-1}a_{n-2}$

请求出下列数列的第$n \leq 10^{10000}$项

- $a_n = 3a_{n-1} + 4a_{n-2} + 8a_{n-3}$
- $a_n = 3a_{n-1} + 4a_{n-2} + 8n^2 + 6n + 9$
- $a_n = 3S_{n-1} + 5a_{n-1} + 6, S_n = \sum_{i=1}^{n} a_i$
- $a_n = a_{n-1}a_{n-2}$
- bonus:最后一个题目中,$a_n = a_1^{F(n-1)} a_2^{F(n-2)}$

请求出下列数列的第$n \leq 10^{10000}$项

- $a_n = 3a_{n-1} + 4a_{n-2} + 8a_{n-3}$
- $a_n = 3a_{n-1} + 4a_{n-2} + 8n^2 + 6n + 9$
- $a_n = 3S_{n-1} + 5a_{n-1} + 6, S_n = \sum_{i=1}^{n} a_i$
- $a_n = a_{n-1}a_{n-2}$
- bonus:最后一个题目中,$a_n = a_1^{F(n-1)} a_2^{F(n-2)}$
- 哪些递推数列不能用这类方法呢？

- 有$n \leq 10^{18}$个砖块你可以给每个砖块染红/黄/蓝/绿四种颜色之一，求有多少种染法使得最终红色砖块数量和绿色砖块数量都是偶数

- 有$n \leq 10^{18}$个砖块你可以给每个砖块染红/黄/蓝/绿四种颜色之一，求有多少种染法使得最终红色砖块数量和绿色砖块数量都是偶数
- 从左往右染色，设a_i, b_i, c_i分别表示当染了i个方块后，红绿皆为偶，红绿皆为奇，红绿其中一个为奇的方案数，则:

$$\begin{cases} a_n = 2a_{n-1} + c_{n-1} \\ b_n = 2b_{n-1} + c_{n-1} \\ c_n = 2a_{n-1} + 2b_{n-1} + 2c_{n-1} \end{cases}$$

用矩阵转移即可

- 有一个5阶矩阵A,求$S_n = \sum_{i=1}^{n} A^i$

- 有一个5阶矩阵A,求$S_n = \sum_{i=1}^{n} A^i$
- 矩阵是可以分块的，对于这个，我们可以有：

$$\begin{pmatrix} S_n \\ A^{n+1} \end{pmatrix} = \begin{pmatrix} I & I \\ 0 & A \end{pmatrix} \begin{pmatrix} S_{n-1} \\ A^n \end{pmatrix}$$

这样就又可以愉快地矩阵快速幂了

二分查找

概念

二分查找可以在$O(\log n)$的时间内在一个有序连续数组内查找某个元素，比朴素线性查找要快得多，是一种极为常用的基础算法。

二分查找

概念

二分查找可以在$O(\log n)$的时间内在一个有序连续数组内查找某个元素，比朴素线性查找要快得多，是一种极为常用的基础算法。

原理

最基本的二分查找解决的问题是：考虑一个长为n的有序数组a，给定一个数x，要求查找是否存在一个元素a_i，满足$a_i = x$，如果存在则给出其下标i。算法的时间复杂度为$O(\log n)$。

二分查找的思想是基于分治的，但是因为它极为常用、变化多端、却又很简单，所以它通常被从分治中剥离开来。它的正确性要求保证数组a是有序的。

为便于描述，设数组下标为$1..n$，按升序排列。

算法维护一个窗口$[left, right]$，初始时窗口为$[1, n]$。

令$mid = (left + right)/2$，容易发现，因为数组是有序的，

当$a_{mid} < x$时，显然我们所求的下标满足$i \in [mid + 1, right]$。正确性很显然，我们假设$i \in [left, mid - 1]$，那么数组就不是有序的了，显然矛盾。同理，当$a_{mid} > x$时，满足$i \in [left, mid - 1]$。

注意到每次窗口大小都会减半，因此最多$O(\log n)$次就可以将窗口大小缩减到1，此时必定能$O(1)$地判断出该元素是否存在。

二分查找非常简单，并且极为常用，在有序条件下我们经常可以用二分来代替枚举，降低时间复杂度。

```
int binary_search(int left, int right, int val)
{
        while (left <= right)
        {
                int mid = (left + right) / 2;
                if (a[mid] == val)
                        return mid;
                if (a[mid] < val)
                        left = mid + 1;
                else
                        right = mid - 1;
        }
}
```

二分查找的扩展非常多，比如在一个升序数组中，求出比x大的最小的元素，或是求出比x小的最大的元素，等等。

事实上，这些扩展都是大同小异的，只要你真正地理解了二分查找的原理，这些变体你可以轻松地自己实现。如果你还不能轻松做到，说明你对二分查找的领悟还不够到位。

下面举几个常见的例子。为便于描述，设数组下标为$1..n$，按升序排序

1. *lower_bound*：求出第一个满足$a_i \geq x$的元素的下标i，若不存在则返回$n+1$。

2. *upper_bound*：求出第一个满足$a_i > x$的元素的下标i，若不存在则返回$n+1$。

```
int lower_bound(int left, int right, int val)
{
        int res = left;
        while (left <= right)
        {
                int mid = (left + right) / 2;
                if (a[mid] < val){//upper_bound <=
                        left = mid + 1;
                        res = left;
                }
                else  right = mid - 1;
        }
        return res;
}
```

有些题目并不是显式地存在一个可排序的数组，但是它的答案具有单调性，此时我们仍然可以运用二分策略。

最简单的例子，比如我们有一个比较复杂的单调函数$f(x) = t$，给定t的值，让你求它的解x。

设答案的取值范围为$x \in [left, right]$，那么每次对于当前值mid，计算出$f(mid)$的值，和t进行比较，据此修改窗口。其余的部分就是上述的二分查找了，如出一辙。

这个策略俗称为二分答案+check。只要问题的解具有单调性，我们就可以二分答案，然后我们通常会写一个check函数（因为这个过程比较复杂，直接整个塞进二分的代码里会非常恶心），据此进行二分。

- 1.初始上限下限需要足够

- 1.初始上限下限需要足够
- 二分求n的平方根，$(n < m),[0, \lceil\sqrt{m}\rceil]$

- 1.初始上限下限需要足够
- 二分求n的平方根，$(n < m)$,$[0, \lceil\sqrt{m}\rceil]$
- 2.$l = mid + 1, r = mid$。防止死循环,永远当作在找后缀区间的首位

- 1.初始上限下限需要足够
- 二分求n的平方根，$(n < m)$,$[0, \lceil\sqrt{m}\rceil]$
- 2.$l = mid + 1, r = mid$。防止死循环,永远当作在找后缀区间的首位
- 3.小数二分时，谨慎设置eps,必要时可用循环替代

- 1.初始上限下限需要足够
- 二分求n的平方根，$(n < m)$,$[0, \lceil\sqrt{m}\rceil]$
- 2.$l = mid + 1, r = mid$。防止死循环,永远当作在找后缀区间的首位
- 3.小数二分时，谨慎设置eps,必要时可用循环替代
- 4.擅用STL中的各种$lower_bound$和$upper_bound$

```
//normal binary search
while(l<r)
{
mid=(l+r)>>1;
if(check(mid))l=mid+1;
else  r=mid;
}
//decimal binary search
while(r-l>eps)
//binary search using 'for'
for(i=0;i<30;i++)
```

```
//binary search in STL
pos=lower_bound(a,a+n)-a;
pos=lower_bound(vec.begin(),vec.end())-vec.begin();
iter=mp.lower_bound(x);
iter=set.lower_bound(x);
```

- 数轴上有$n \leq 10^5$个点，点的坐标x_i的绝对值小于10^9,你有$k \leq 10^5$块等长的木板，你想要将这些点全部覆盖，那么这些木板最短需要多长呢？

- 数轴上有$n \leq 10^5$个点，点的坐标x_i的绝对值小于10^9,你有$k \leq 10^5$块等长的木板，你想要将这些点全部覆盖，那么这些木板最短需要多长呢？
- solution:显然，长度为x的木板若能盖住，则长度大于x的木板也能盖住，所以可以对长度x二分，每次看当前长度是否可行，可行性检查是$O(n)$的贪心问题，复杂度$O(nlogn)$

经典应用1：最大最小值，最小最大值

经典应用1：最大最小值，最小最大值

- 农民约翰有用$C \leq 10^5$只牛，然后他有$n \leq 10^5$个隔间，每个隔间都有自己的坐标位置，如何安排把牛安排进隔间才能使牛两两之间距离的最小值最大，求最大的最小值

经典应用1：最大最小值，最小最大值

- 农民约翰有用$C \leq 10^5$只牛，然后他有$n \leq 10^5$个隔间，每个隔间都有自己的坐标位置，如何安排把牛安排进隔间才能使牛两两之间距离的最小值最大，求最大的最小值
- 二分最小值，等价于两两之间的距离都大于这个最小值，贪心从前往后放置牛即可。一个小case，当$C > n$时答案为0

- 有$n \leq 10^5$个人从左到右站成一排，每个人都属于一个组织，一共只有$k \leq 10^5$个组织，用数字$x \in [1, k]$表示，第i个人属于组织s_i,一开始每个人手里都有一颗宝石，现在依次发出了$q \leq 10^5$条指令，每条指令形如$a_i \in [1, k], b_i \in [0, 1]$,表示每个$a_i$组织的人都会把他手中的所有宝石交给他左边/右边的人，如果是第一个人往左传或者第n个人往右传，那么很遗憾的是他的所有宝石就丢掉了,当所有指令结束后，问还有多少宝石没有丢掉

- 有$n \leq 10^5$个人从左到右站成一排，每个人都属于一个组织，一共只有$k \leq 10^5$个组织，用数字$x \in [1, k]$表示，第i个人属于组织s_i,一开始每个人手里都有一颗宝石，现在依次发出了$q \leq 10^5$条指令，每条指令形如$a_i \in [1, k], b_i \in [0, 1]$,表示每个$a_i$组织的人都会把他手中的所有宝石交给他左边/右边的人，如果是第一个人往左传或者第n个人往右传，那么很遗憾的是他的所有宝石就丢掉了,当所有指令结束后，问还有多少宝石没有丢掉
- 核心观察：如果把所有宝石编号，那么它们从左到右的相对顺序不会变化！因此我们只需二分从左往右第一个没有丢掉的宝石和从右往左第一个没有丢掉的宝石即可，时间复杂度$O(qlogn)$

- 给定一个第一象限的$n \leq 10^5$个点的凸多边形，第i个点为$A_i = (x_i, y_i)$。共$q \leq 10^5$次询问，每次给定第二象限的一个点P，求这个点与这个凸多边形的两条切线

- 给定一个第一象限的$n \leq 10^5$个点的凸多边形，第i个点为$A_i = (x_i, y_i)$。共$q \leq 10^5$次询问，每次给定第二象限的一个点P，求这个点与这个凸多边形的两条切线
- 将这个凸多边形的x坐标最小/最大的点拿出来，将多边形分割成上凸壳和下凸壳，可知两条切线对应的切点一定分别在上凸壳和下凸壳上。

- 给定一个第一象限的$n \leq 10^5$个点的凸多边形，第i个点为$A_i = (x_i, y_i)$。共$q \leq 10^5$次询问，每次给定第二象限的一个点P，求这个点与这个凸多边形的两条切线
- 将这个凸多边形的x坐标最小/最大的点拿出来，将多边形分割成上凸壳和下凸壳，可知两条切线对应的切点一定分别在上凸壳和下凸壳上。
- 设切点为Ai,$(A_i - A_{i-1}) \times (A_i - P)$与$(A_{i+1} - A_i) \times (A_i - P)$符号相反，这个点可以二分得到。时间复杂度$O(qlogn)$

分治法

- 求一个数列中的最大子段和？

分治法

- 求一个数列中的最大子段和？
- 暴力枚举$O(n^2)$个区间，时间复杂度$O(n^2)$

分治法

- 求一个数列中的最大子段和？
- 暴力枚举$O(n^2)$个区间，时间复杂度$O(n^2)$
- 分治:$solve(l, r)$

  在$[l, r]$中，设$mid = (l + r)/2$.有三类子段：

分治法

- 求一个数列中的最大子段和？
- 暴力枚举$O(n^2)$个区间，时间复杂度$O(n^2)$
- 分治:$solve(l, r)$

  在$[l, r]$中，设$mid = (l + r)/2$.有三类子段：
- 1.在$[l, mid]$中的子段，这一部分可以在$solve(l, mid)$中分治处理

分治法

- 求一个数列中的最大子段和？
- 暴力枚举$O(n^2)$个区间，时间复杂度$O(n^2)$
- 分治:$solve(l, r)$

  在$[l, r]$中，设$mid = (l + r)/2$.有三类子段：
- 1.在$[l, mid]$中的子段，这一部分可以在$solve(l, mid)$中分治处理
- 2.在$[mid + 1, r]$中的子段，这一部分可以在$solve(mid + 1, r)$中分治处理

分治法

- 求一个数列中的最大子段和？
- 暴力枚举$O(n^2)$个区间，时间复杂度$O(n^2)$
- 分治:$solve(l, r)$

  在$[l, r]$中，设$mid = (l + r)/2$.有三类子段：
- 1.在$[l, mid]$中的子段，这一部分可以在$solve(l, mid)$中分治处理
- 2.在$[mid + 1, r]$中的子段，这一部分可以在$solve(mid + 1, r)$中分治处理
- 3.包含mid的子段，我们只需要从mid开始，往左和往右分别求出最大的前缀/后缀，加起来即可。

分治法

- 求一个数列中的最大子段和？
- 暴力枚举$O(n^2)$个区间，时间复杂度$O(n^2)$
- 分治:$solve(l, r)$

  在$[l, r]$中，设$mid = (l + r)/2$.有三类子段：
- 1.在$[l, mid]$中的子段，这一部分可以在$solve(l, mid)$中分治处理
- 2.在$[mid + 1, r]$中的子段，这一部分可以在$solve(mid + 1, r)$中分治处理
- 3.包含mid的子段，我们只需要从mid开始，往左和往右分别求出最大的前缀/后缀，加起来即可。
- 基本流程：$solve(l, r)$
- $solve(l, mid)$
- $solve(mid + 1, r)$

分治法

- 求一个数列中的最大子段和？
- 暴力枚举$O(n^2)$个区间，时间复杂度$O(n^2)$
- 分治:$solve(l, r)$

  在$[l, r]$中，设$mid = (l + r)/2$.有三类子段：
- 1.在$[l, mid]$中的子段，这一部分可以在$solve(l, mid)$中分治处理
- 2.在$[mid + 1, r]$中的子段，这一部分可以在$solve(mid + 1, r)$中分治处理
- 3.包含mid的子段，我们只需要从mid开始，往左和往右分别求出最大的前缀/后缀，加起来即可。
- 基本流程：$solve(l, r)$
- $solve(l, mid)$
- $solve(mid + 1, r)$

分治法

- 求一个数列中的最大子段和？
- 暴力枚举$O(n^2)$个区间，时间复杂度$O(n^2)$
- 分治:$solve(l, r)$

  在$[l, r]$中，设$mid = (l + r)/2$.有三类子段：
- 1.在$[l, mid]$中的子段，这一部分可以在$solve(l, mid)$中分治处理
- 2.在$[mid + 1, r]$中的子段，这一部分可以在$solve(mid + 1, r)$中分治处理
- 3.包含mid的子段，我们只需要从mid开始，往左和往右分别求出最大的前缀/后缀，加起来即可。
- 基本流程：$solve(l, r)$
- $solve(l, mid)$
- $solve(mid + 1, r)$

- 初始有一张$2^n \times 2^n$的棋盘，$n \leq 10$,一开始这个棋盘上有一个位置(x, y)被占据了，其他地方都是空的，请设计一种方案，用L型多米诺骨牌铺满所有空着的地方

- 初始有一张$2^n \times 2^n$的棋盘，$n \leq 10$,一开始这个棋盘上有一个位置(x, y)被占据了，其他地方都是空的，请设计一种方案，用L型多米诺骨牌铺满所有空着的地方
- $solve(n, P)$

  将棋盘分成四块A, B, C, D，把含有P的那块称为A

  放置一块骨牌使得B, C, D都被占据了一个格子

  $solve(n-1, P1)$

  $solve(n-1, P2)$

  $solve(n-1, P3)$

  $solve(n-1, P4)$

- 初始有一张$2^n \times 2^n$的棋盘，$n \leq 10$,一开始这个棋盘上有一个位置(x, y)被占据了，其他地方都是空的，请设计一种方案，用L型多米诺骨牌铺满所有空着的地方
- $solve(n, P)$

  将棋盘分成四块A, B, C, D，把含有P的那块称为A

  放置一块骨牌使得B, C, D都被占据了一个格子

  $solve(n-1, P1)$

  $solve(n-1, P2)$

  $solve(n-1, P3)$

  $solve(n-1, P4)$
- 注意边界条件，当$n = 0$时直接返回

归并排序与逆序对

归并排序：利用分治法排序

归并排序与逆序对

归并排序：利用分治法排序

算法流程：调用$solve(l, r)$

- $A = solve(l, mid)$,将数组前一半排序

归并排序与逆序对

归并排序：利用分治法排序

算法流程：调用$solve(l, r)$

- $A = solve(l, mid)$,将数组前一半排序
- $B = solve(mid + 1, r)$,将数组后一半排序

归并排序与逆序对

归并排序：利用分治法排序

算法流程：调用$solve(l, r)$

- $A = solve(l, mid)$,将数组前一半排序
- $B = solve(mid + 1, r)$,将数组后一半排序
- $return \quad merge(A, B)$,将A, B两个有序数组按照顺序归并成新的有序数组，这一步是$O(r - l)$的

归并排序与逆序对

归并排序：利用分治法排序

算法流程：调用$solve(l, r)$

- $A = solve(l, mid)$,将数组前一半排序
- $B = solve(mid + 1, r)$,将数组后一半排序
- $return \quad merge(A, B)$,将A, B两个有序数组按照顺序归并成新的有序数组，这一步是$O(r - l)$的
- $T(n) = 2T(\frac{n}{2}) + O(n) = O(nlogn)$

```
int n,i,a[105],tmp[105];
void solve(int l,int r)
{
        int hd1,hd2,hd,i;
        if(l>=r)return;
        int mid;
        mid=(l+r)>>1;
        solve(l,mid);
        solve(mid+1,r);
        hd1=l;hd2=mid+1;hd=l;
        while(hd1<=mid||hd2<=r)
        {
                if(hd1>mid)tmp[hd++]=a[hd2++];
```

```
                else
                {
                        if(hd2>r)tmp[hd++]=a[hd1++];
                        else
                        {
                        if(a[hd1]<a[hd2])
                                tmp[hd++]=a[hd1++];
                        else
                                tmp[hd++]=a[hd2++];
                        }
                }
        }
        for(i=l;i<=r;i++)a[i]=tmp[i];
}
```

逆序对，数组中的(i,j)对数满足$i<j$且$a_i>a_j$

逆序对，数组中的(i,j)对数满足$i<j$且$a_i>a_j$

在归并排序的*merge*步骤，每加入一个A中元素，都将比这个元素小的B中元素的数量加入答案。(为什么呢？)

以下是进阶内容

- 给定一个长度为n的数组，判断这个数组是否满足：对于任意一个子段，这个子段中至少有一个只出现了一次的元素。

- 给定一个长度为n的数组，判断这个数组是否满足：对于任意一个子段，这个子段中至少有一个只出现了一次的元素。
- solution：首先有一种思路是：找到一个在数组中只出现了一次的元素，设为k，然后判断$[l,k]$和$[k+1,r]$即可。但如果只是朴素的从头到尾找这个k的话，最坏时间复杂度是$O(n^2)$

- 给定一个长度为n的数组，判断这个数组是否满足：对于任意一个子段，这个子段中至少有一个只出现了一次的元素。
- solution：首先有一种思路是：找到一个在数组中只出现了一次的元素，设为k，然后判断$[l,k]$和$[k+1,r]$即可。但如果只是朴素的从头到尾找这个k的话，最坏时间复杂度是$O(n^2)$
- 进行如下改进，从两头向中间同时找这个k，时间复杂度的递推式变为$T(n)=maxT(k)+T(n-k)+minO(k),O(n-k)$,则$T(n)=O(nlogn)$这个时间复杂度可以用决策树证明

经典应用2：0/1分数规划

- 有$n \leq 10^5$块金属，第i块金属有体积和质量分别是v_i, m_i,求一个金属块的子集,要求子集中至少有$k \leq 10^5$块金属，使得它们的质量和除以体积和最大，只需要输出最大的质量和与体积和的比即可

经典应用2：0/1分数规划

- 有$n \leq 10^5$块金属，第i块金属有体积和质量分别是v_i, m_i,求一个金属块的子集,要求子集中至少有$k \leq 10^5$块金属，使得它们的质量和除以体积和最大，只需要输出最大的质量和与体积和的比即可
- 分数规划：$\lambda = \frac{\sum m}{\sum v}$,即$\sum (m - \lambda v) = 0$

经典应用2：0/1分数规划

- 有$n \leq 10^5$块金属，第i块金属有体积和质量分别是v_i, m_i,求一个金属块的子集,要求子集中至少有$k \leq 10^5$块金属，使得它们的质量和除以体积和最大，只需要输出最大的质量和与体积和的比即可
- 分数规划：$\lambda = \frac{\sum m}{\sum v}$,即$\sum (m - \lambda v) = 0$
- 也就是说，对于一个λ,如果存在一个子集使得$\sum (m - \lambda v) > 0$则最终答案一定大于λ，否则最终答案小于等于λ,因此我们二分λ即可，每次将一个物品的价值设置为$m - \lambda v$，然后取前k大价值的物品加起来，看是否大于零即可。

- 有$n \leq 10^5$个电池，初始电量为$a_i \leq 10^9$，每分钟消耗$b_i \leq 10^9$，要求在接下来的$m \leq 10^5$分钟内每个电池都必须保证电量非负。你可以买一个充电宝，花x元钱可以买到一个每分钟充x电量的充电宝，每分钟你都可以用这个充电宝给某个电池充电(必须充电完整的一分钟)。每个电池容量无上限，问最少花多少钱完成任务？若无论如何都无法保持住n个电池电量，输出-1

- 有$n \leq 10^5$个电池，初始电量为$a_i \leq 10^9$，每分钟消耗$b_i \leq 10^9$，要求在接下来的$m \leq 10^5$分钟内每个电池都必须保证电量非负。你可以买一个充电宝，花x元钱可以买到一个每分钟充x电量的充电宝，每分钟你都可以用这个充电宝给某个电池充电(必须充电完整的一分钟)。每个电池容量无上限，问最少花多少钱完成任务？若无论如何都无法保持住n个电池电量，输出-1
- 二分x是显然的，然后我们需要验证每分钟充x电量的充电宝是否能满足需求，贪心即可，在每一分钟，我们用充电宝优先给**最快即将没电**的电池续命，这个可以用数据结构(*map*)维护最快即将没电的电池，时间复杂度$O(mlognlogA)$

- 有$n \leq 10^5$个电池，初始电量为$a_i \leq 10^9$，每分钟消耗$b_i \leq 10^9$，要求在接下来的$m \leq 10^5$分钟内每个电池都必须保证电量非负。你可以买一个充电宝，花x元钱可以买到一个每分钟充x电量的充电宝，每分钟你都可以用这个充电宝给某个电池充电(必须充电完整的一分钟)。每个电池容量无上限，问最少花多少钱完成任务？若无论如何都无法保持住n个电池电量，输出-1
- 二分x是显然的，然后我们需要验证每分钟充x电量的充电宝是否能满足需求，贪心即可，在每一分钟，我们用充电宝优先给**最快即将没电**的电池续命，这个可以用数据结构(*map*)维护最快即将没电的电池，时间复杂度$O(mlognlogA)$
- bonus:请试着提出一个时间复杂度$O(mlogA)$的算法