二分答案

主 讲：马 骋　王 静

时 间：2023年7月5日

第一部分　二分查找

引例1：猜数字

任意写一个 1 到1000000000之间的整数，请你猜出这个数字。

每猜一次，都会反馈给你答案是偏大、偏小还是刚好。

请问有什么高效的方案?

最多猜多少次可以准确地给出答案?

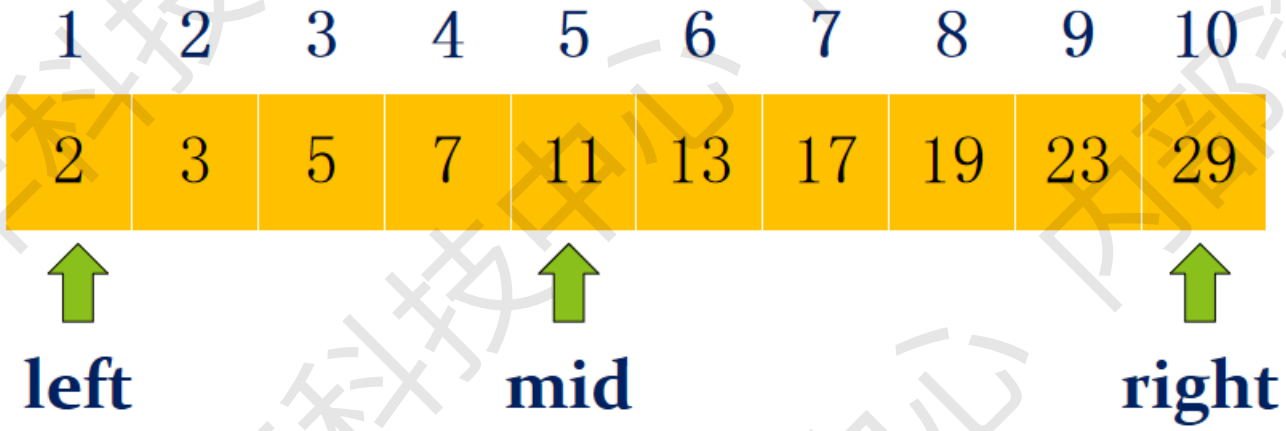
从查找说起

已知某含有 n 个元素的有序序列，如何判断某个元素 x 是否在此序列中?

- 顺序查找　　数据可以无序，时间复杂度 O(N)
- 哈希表　　空间要求高，时间复杂度 O(1)
- 二分查找　　数据要求有序，时间复杂度 O(logN)

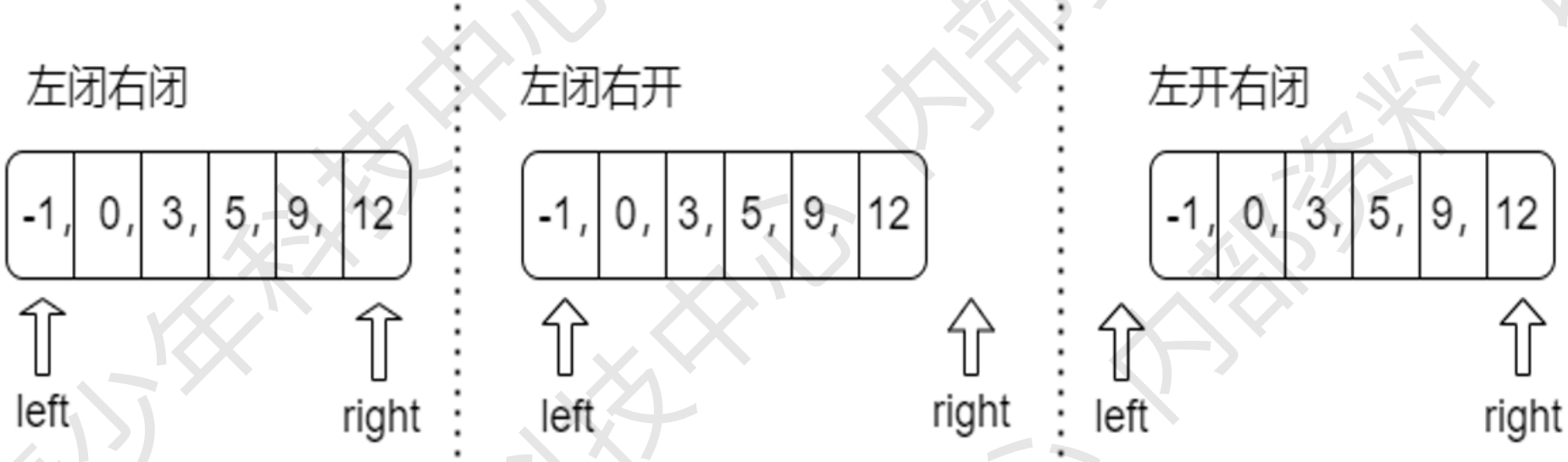
二分查找

n=10， x=17　　mid=(left+right)/2



左闭右闭条件下二分查找参考程序

```
int search(int nums[], int size, int target) {
  int left = 0;
  int right = size - 1;
  while (left <= right) { // 当 left>right 时没有找到目标值
    int middle = left + ((right - left) / 2);
    if (nums[middle] > target)
      right = middle-1; //target 在左区间，在[left, middle-1]中
    else if (nums[middle] < target)
      left = middle + 1;
    else
      return middle;
  }
  return -1;
}
```



左闭右开条件下二分查找参考程序

```
int search(int nums[], int size, int target) {
  int left = 0;
  int right = size;
  while (left < right) { // 当 left=right 时，没有找到目标值
    int middle = left + ((right - left) / 2);
    if (nums[middle] > target)
      right = middle; //target 在左区间，在[left, middle)中
    else if (nums[middle] < target)
      left = middle + 1;
    else
      return middle;
  }
  return -1;
}
```

STL中的二分查找

使用STL二分查找时，需要保证数据有序（默认从小到大）

- binary_search(begin, end, num)：[begin,end)区间二分查找等于 num 的数，找到返回 true，否则返回 false。
- lower_bound(begin, end, num)：[begin,end)区间二分查找第一个大于或等于 num 的数字，找到返回该数字的地址，不存在则返回 end。
- upper_bound(begin, end, num)：[begin,end)区间二分查找第一个大于 num 的数字，找到返回该数字的地址，不存在则返回 end。

STL中的二分查找

例如：对于数组 a[8] = {1,2,3,5,5,5,8,9}，其中 n=8 表示数组中元素的个数。

数组中是否存在数字“9”：	binary_search(a, a+n, 9)	true
找到第一个“≥5”的元素位置：	lower_bound(a, a+n, 5)-a	3
找到第一个“＞5”的元素位置：	upper_bound(a, a+n, 5)-a	6
找到最后一个“＜5”的元素位置：	lower_bound(a, a+n, 5)-a-1	2
找到最后一个“≤5”的元素位置：	upper_bound(a, a+n, 5)-a-1	5

二分查找的场景

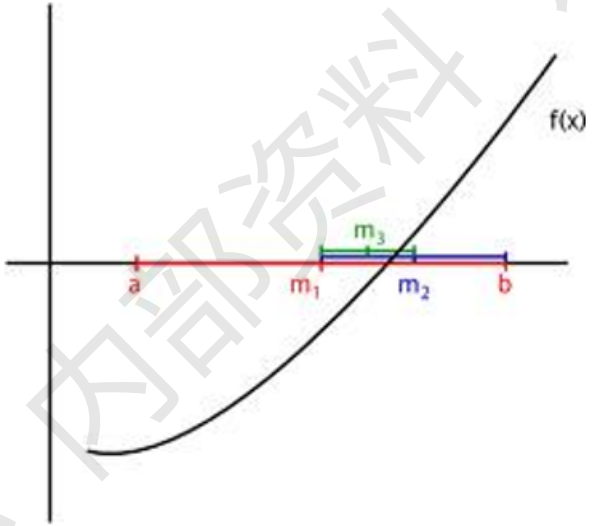
- 连续整数区间：主要是正整数
- 连续实数区间：有精度要求
- 递增整数（实数）序列

【例1】求方程的根

求方程的根：$f(x) = ax^3 + bx^2 + cx + d = 0$。

给出该方程中各项的系数（a,b,c,d均为实数）并约定该方程存在三个不同实根（根的范围在 −100 至 100 之间），且根与根之差的绝对值≥1。

要求由小到大依次在同一行输出这三个实根（根与根之间留有空格），并精确到小数点后 2 位。



【算法分析】

枚举根的值域中的每一个整数x(-100≤x≤100)。由于根与根之差的绝对值≥1，因此设定搜索区间[x1，x2)，其中 x1=x，x2=x+1。若

(1) f(x1)=0，则确定 x1 为 f(x) 的根；

(2) f(x1)*f(x2)>0，则确定根x不在区间 [x1，x2) 内，设定 [x2，x2+1) 为下一个搜索区间；

(3) f(x1)*f(x2)<0，则确定根x在区间 [x1，x2) 内。

如果确定根x在区间[x1，x2)内，可以采用二分法求得在该区间找到根的确切位置。

第二部分　二分答案

二分答案

二分答案的题目一般满足以下特征：

1. 答案的范围已知且具有单调性，“**最小值最大化**”或“**最大值最小化**”。
2. 答案可以通过二分查找枚举（连续的整数、递增的整数等）。
3. 通过检验答案，可以方便地判断出答案是过大还是过小（二分答案的关键）

通过二分答案，可以将最优性问题转化为可行性问题。

二分答案的基本框架

1. 判断答案的范围

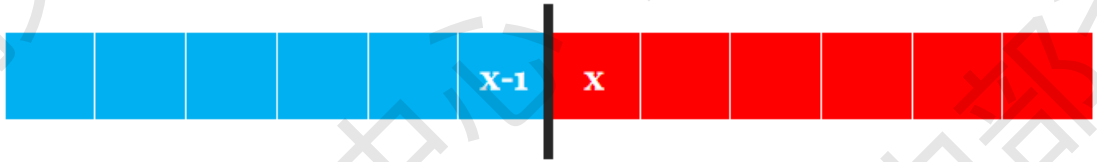
2. 二分查找答案区间

3. 验证答案（冲突判断）

二分答案的基本框架

以最小值最大化为例，如下图所示：

x 是不符合要求的最小值，x-1 是符合要求的最大值，中间是分界线。



初始的时候，left表示**符合要求的最小值**，right表示**不符合要求的最大值**，中间是灰色区域。

二分答案的基本框架


```
left = minBlue, right = maxRed;
while (left + 1 < right)
{
  mid = (left + right) / 2;
  if check(mid)      // 符合条件，在蓝色区域
    left = mid;
  else               // 不符合条件，在红色区域
    right = mid;
}
return left;
```

【例2】洗衣服

洗完衣服后，你就要弄干衣服。衣服在自然条件下用 1 个单位的时间可以晒干 A 点湿度。现在买了 1 台烘衣机，使用烘衣机可以让你用 1 个单位的时间使 1件衣服除开自然晒干的 A 点湿度外，还可烘干 B 点湿度，但在 1 个单位的时间内只能对1件衣服使用。

N 件衣服因为种种原因而不一样湿，现在告诉你每件衣服的湿度，要你求出弄干所有衣服的最少时间（湿度为0为干）。

【例2】洗衣服

输入格式：共 N+1 行

第一行 N，A，B；

接下来N行，每行一个数，表示衣服的湿度；

1 <= 湿度 A、B <= 500000,

1 <= N <= 500000。

输出格式：一行一个整数，**表示最少时间**。

输入样例：

3 2 1

1

2

3

输出样例：

1

样例分析：

1个单位时间内，用机器处理第3件衣服，其他两件衣服自然晒干。

【例2】洗衣服

衣服在自然条件下单位时间可以晒干 A 点湿度，使用烘衣机可以叠加烘干 B 点湿度。

仅拥有 1 台烘衣机，单位时间内只能 1 件衣服使用。

N件衣服湿度不同（输入），求出弄干所有衣服的最少时间（湿度为 0 为干）。

【算法分析】

方法1：使用贪心

显然，按照贪心思想，烘衣机应该处理湿度最大的衣服。每个单位时间，找出湿度最大的衣服使用烘干机处理，这样可以确保答案最优。

贪心算法思路

设单位时间内自然晒干的湿度是 A ，烘衣机烘干的湿度是 B，当前时间是 k 。

1. 找出当前最大的湿度值 MaxB，可以用排序，可以用哈希，也可以用优先队列。
2. 将 MaxB 减去 B 后放回到队伍中，找出烘干机处理后的最大湿度值 MaxA。
3. 如果 MaxA >= A*k , 则 k 即为最优解，否则 k+1 进入下一轮循环。

【核心代码】

```
int c[maxn];
......

for (int i = 1; i <= n; i ++)
{
  cin >> c[i];
}

while(1)
{
  ans++;
  sort(c+1, c+1+n);
  c[n] = c[n] - b;
  if (max(c[n], c[n-1]) <= a*ans)
  {
    cout << ans;
    return 0;
  }
}
```

使用数组排序处理数据
时间复杂度 $O(n^2 \log n)$

```
priority_queue<int> q;
......

for (int i = 1; i <= n; i ++)
{
  cin >> x;
  q.push(x);
}

while(1)
{
  ans ++;
  q.push(q.top() - b);
  q.pop();

  if (q.top() <= a * ans)
  {
    cout << ans;
    return 0;
  }
}
```

使用优先队列处理数据
时间复杂度 $O(n*\log n)$

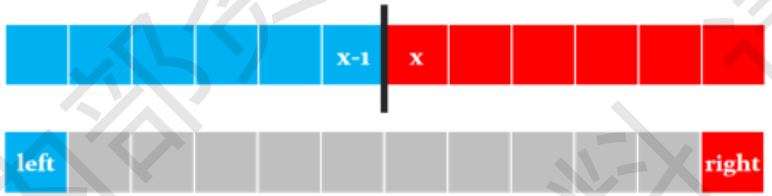
【算法分析】

方法2：二分答案

二分答案可行性分析

- 是不是最优化问题：可行的最小时间（最大的值最小问题）
- 数据是否具有单调性：烘衣机使用时间越长，衣服干得越快
- 如何验证答案可行性：烘衣机使用时间→衣服是否能全部晾干

【算法分析】



二分答案程序基本框架

1. 设答案范围 (L, R] 之间。

其中，L不可取，R可取。初始范围是 (0, 500000]。

2. 对答案做二分查找。

mid = (L+R)/2 。如果 mid 可行，则答案范围修改为(L, mid]，如果 mid 不可行，则答案范围修改为 (mid，R]，直至 L+1 == R为止，输出 ans = R 。

3. 验证 mid 是否可行。可以使用 check 函数验证。

【算法分析】

使用 check 函数验证答案

设第 i 件衣服的湿度是 h[i]，需要使用烘干机时间为 cnt

则 cnt*b+mid*a=h[i]，其中 a 是自然晒干的湿度，b是烘衣机烘干的湿度

cnt = (h[i]-mid*a)/b 向上取整

∑cnt>mid　　mid时间内无法干　　mid不可行　　check 为 false

∑cnt<=mid　　mid时间内可以干　　mid可行　　check 为 true

【核心代码】

```
int left=0, right=500000, mid;
while (left+1<right)
{
  mid = (left+right)/2;
  if (check(mid))
    right = mid;
  else
    left = mid;
}
cout << right;
```

```
bool check(int ans)
{
  int cnt=0;
  for (int i=1; i<=n; i++)
  {
    if (c[i] > a*ans)
      cnt += (c[i]-a*ans+b-1)/b;
  }
  if (cnt <= ans)
    return true;
  else
    return false;
}
```

【例3】网线主管

仙境的居民们决定举办一场程序设计区域赛。他们决定将选手的电脑用星形拓扑结构连接在一起，即将它们全部连到一个单一的中心服务器。为了组织这个完全公正的比赛，裁判委员会主席提出要将所有选手的电脑等距离地围绕在服务器周围放置。

为购买网线，裁判委员会联系了当地的一个网络解决方案提供商，要求能够提供一定数量的等长网线。裁判委员会希望**网线越长越好**，这样选手们之间的距离可以尽可能远一些。

【例3】网线主管

该公司的网线主管承接了这个任务。他知道库存中每条网线的长度（精确到厘米），并且只要告诉他所需的网线长度（精确到厘米），他都能够完成对网线的切割工作。但是，这次所需的网线长度并不知道，这让网线主管不知所措。

你需要编写一个程序，帮助网线主管确定一个最长的网线长度，并且按此长度对库存中的网线进行切割，能够得到指定数量的网线。

【例3】网线主管

输入格式：共 N+1 行。

第一行包含两个整数N和K，以单个空格隔开。N（1<=N<=10000）是库存中的网线数，K（1<=K<=10000）是需要的网线数量。接下来N行，每行一个数，为库存中每条网线的长度（单位：米）。所有网线的长度至少1m，至多100km。输入中的所有长度都精确到厘米，即保留到小数点后两位。

输出格式：共一行。

网线主管能够从库存的网线中切出指定数量的网线的最长长度（单位：米）。必须精确到厘米，即保留到小数点后两位。若无法得到长度至少为1cm的指定数量的网线，则必须输出“0.00”。

【例3】网线主管

输入样例：
4 11
8.02
7.43
4.57
5.39

已知库存N根网线，每条网线的长度w_i（单位：米），需要的**等长**网线的数量k。

确定最长的网线长度（精确到厘米）。

为方便操作，转换单位，把所有的米转换成厘米。

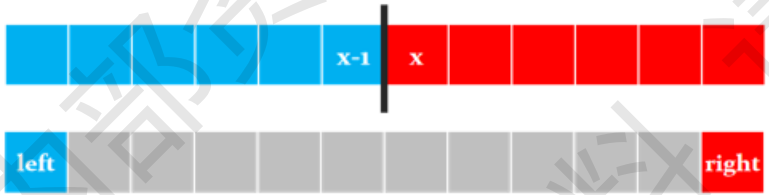
输出样例：
2.00

【算法分析】

二分答案可行性分析

- 是不是最优化问题：等长网线的最长长度（最小的值最大问题）
- 数据是否具有单调性：切割长度越长，切割的成品数量越少
- 如何验证答案可行性：切割长度→成品的数量是否能满足要求

【算法分析】



二分答案程序基本框架

1. 设答案范围 [L, R) 之间。

其中，L可取，R不可取。初始范围是 [0, 10000000)。

2. 对答案做二分查找。

mid = (L+R)/2 。如果 mid 可行，则答案范围修改为[mid, R)，如果 mid 不可行，则答案范围修改为 [L, mid)，直至 L+1 == R为止，输出 ans = L 。

3. 验证 mid 是否可行。可以使用 check 函数验证。

【算法分析】

使用 check 函数验证答案

设切割长度为mid（**单位：厘米**）

若第i个库存，长度w[i]，可以切割出成品 cnt

则 cnt = w[i]/mid　　向下取整

∑cnt >=K　　可以切割出k份　　mid可行　　check为true

∑cnt< K　　不能切割出k份　　mid不可行　　check为false

【核心代码】

```
int left = 0, right = 10000000, mid;
while (left+1<right)
{
  mid = (left+right)/2;
  if (check(mid))
    left = mid;
  else
    right = mid;
}
cout << left*0.01 << endl;
```

```
bool check(int ans)
{
  int cnt=0;
  for (int i=1; i<=n; i++)
  {
    cnt += w[i]/ans;
  }
  if (cnt >= k)
    return true;
  else
    return false;
}
```

【例3】河中跳房子

每年奶牛们都要举办各种特殊版本的跳房子比赛，包括在河里从一个岩石跳到另一个岩石。这项激动人心的活动在一条长长的笔直河道中进行，在起点和离起点L远(1≤L≤1,000,000,000)的终点处均有一个岩石。在起点和终点之间，有N(0≤N≤50,000)个岩石，每个岩石与起点的距离分别为Di(0<Di<L)。

在比赛过程中，奶牛轮流从起点出发，尝试到达终点，每一步只能从一个岩石跳到另一个岩石。当然，实力不济的奶牛是没有办法完成目标的。

【例3】河中跳房子

农夫约翰为他的奶牛们感到自豪并且年年都观看了这项比赛。但随着时间的推移，看着其他农夫的胆小奶牛们在相距很近的岩石之间缓慢前行，他感到非常厌烦。他计划移走一些岩石，使得从起点到终点的过程中，最短的跳跃距离最长。他可以移走除起点和终点外的至多 M (0≤M≤N)个岩石。

请帮助约翰确定移走这些岩石后，最长可能的最短跳跃距离是多少?

【例3】河中跳房子

输入数据：共 N+1 行

第一行包含三个整数L, N, M，相邻两个整数之间用单个空格隔开。

接下来N行，每行一个整数，表示每个岩石与起点的距离。岩石按与起点距离从近到远给出，且不会有两个岩石出现在同一个位置。

输出数据：共一行

一个整数，最长可能的最短跳跃距离。

【例3】河中跳房子

在起点和终点之间，有 N 块岩石（不含起点 0 和终点 L 的岩石，距起点 Di）。比赛过程中，选手们从起点出发，每一步跳向相邻的岩石，直至到达终点。

至多移走 M 块中间的岩石，使得选手们在比赛过程中的最短跳跃距离尽可能长。

求最短跳跃距离的最大值。

【例3】河中跳房子

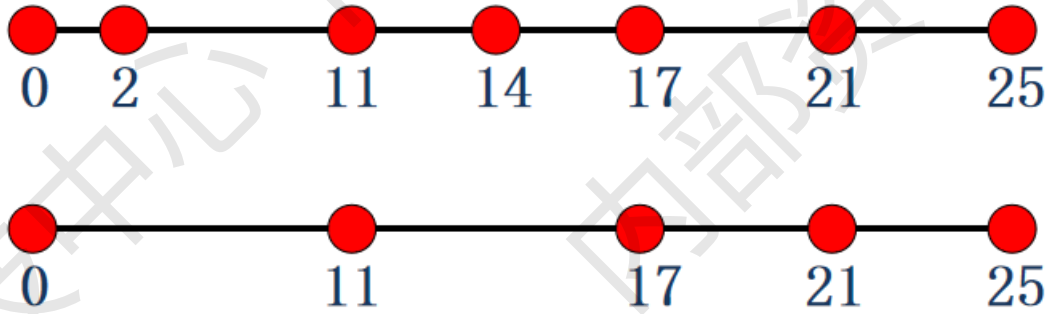
样例输入

25 5 2
2
11
14
17
21

样例输出

4

样例解析

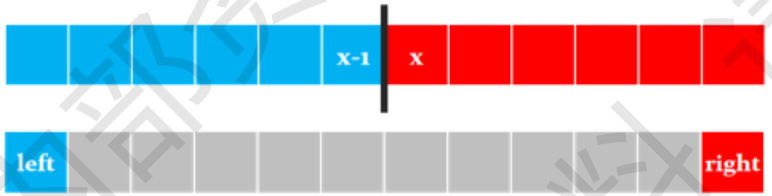


【算法分析】

二分答案可行性分析

- 是不是最优化问题：求最短跳跃距离的最大值（最小的值最大问题）
- 数据是否具有单调性：最短距离越长，移走石子越多
- 如何验证答案可行性：最短距离→移走的石子数量是否可行

【算法分析】



二分答案程序基本框架

1. 设答案范围 [L, R) 之间。

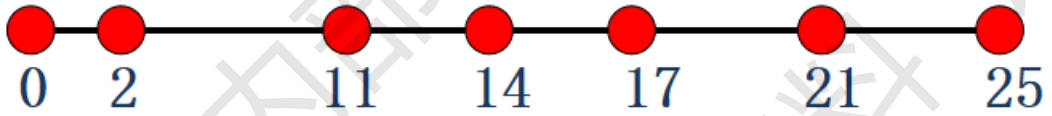
其中，L可取，R不可取。初始范围是 [1, L+1)。

2. 对答案做二分查找，得到 mid，根据check的值，修改 L 和 R 的值。

3. 验证 mid 是否可行。使用 check 函数验证。

【算法分析】

使用 check 函数验证答案



设最短跳越距离为mid

当前石头是Di，则需要移走的石头个数：

Di+1-Di<mid　移走石头i+1

Di+2-Di<mid　移走石头i+2

Di+3-Di<mid　移走石头i+2

.....

Di+x-Di>=mid：跳跃，当前石头变成Di+x

移走的石头总数>=K　　mid可行

移走的石头总数<K　　mid不可行

【核心代码】

```
int left = 1, right = l+1, mid;
while (left+1<right)
{
  mid = (left+right)/2;
  if (check(mid))
    left = mid;
  else
    right = mid;
}
cout << left << endl;
```

```
bool check(int ans)
{
  int pre=0, cnt=0;
  for (int i=1; i<=n; i++)
  {
    if (a[i]-pre<ans)
      cnt++;
    else
      pre = a[i];
  }
  if (cnt <= m)
    return true;
  else
    return false;
}
```

二分答案小结

- 在解决二分答案的题目时，通常需要分析出答案的**单调性**和**有界性**，并且需要设计出正确高效的**验证算法**。
- 由于题目的千变万化，贪心、动态规划、模拟、图论等都有可能成为验证算法。
- 二分答案的题目涉猎很广，变化丰富，因此在近年各类比赛中备受出题者的青睐。
- 在解决题目的过程中，我们要对单调性、“**最大值最小**”，“**最小值最大**”等字眼保持敏感。

感

谢

观

看