二分答案

河南省实验中学信息技术组

2026 年 01 月 24 日

二分

- 二分是一种非常精妙的算法，经常能为我们提供解决的问题的突破口，而且能节约算法时间。
- 二分的实现方法多种多样，但是写出正确的二分算法是一件不容易的事[1]。
- 对于整数域上的二分，需要注意终止边界、左右区间取舍时的开闭情况，避免漏掉答案或造成死循环；对于实数域上的二分，需要注意精度问题。

[1] 据说，只有 10% 的程序员能写对二分。

整数域上的二分

- 在单调递增序列 a 中查找 $\geq x$ 的数中最小的一个数的下标：

```
int l = 1, r = n;
while(l < r)
{
    int m = (l + r) / 2;
    if(a[m] >= x) r = m;
    else l = m + 1;
}
int ans = l;
```

- 在单调递增序列 a 中查找 $\leq x$ 的数中最大的一个数的下标：

```
int l = 1, r = n;
while(l < r)
{
    int m = (l + r + 1) / 2;
    if(a[m] <= x) l = m;
    else r = m - 1;
}
int ans = l;
```

- 上述两种二分写法保证最终答案处于闭区间 $[l, r]$ 以内，循环以 $l == r$ 结束，每次二分的中间值 m 会归属于左半段与右半段二者之一。

整数域上的二分

- m 取值和 $[l,r]$ 的范围的缩小要匹配，否则很容易造成漏解、死循环等问题。
- 为了正确书写，可以固定二分流程为：
  - 分析具体问题，确定左右半段哪一段是可行区间，以及 m 归属哪一半段。
  - 如果答案在右半段，那么令 $m=(l+r)/2$(尽可能往左半段落)，根据情况令 $r=m$ 或 $l=m+1$；如果答案在左半段，那么令 $m=(l+r+1)/2$(尽可能往右半段落)，根据情况令 $l=m$ 或 $r=m-1$。
  - 二分终止条件是 $l==r$，该值就是答案所在位置。
- C++ STL 中`lower_bound`函数实现了在递增序列中查找$\geq x$的最小的值所在的位置，`upper_bound`函数实现了递增序列中查找$>x$的最小的值所在的位置。

实数域上的二分

- 实数域上的二分需要确定精度 eps，以 $l + eps < r$ 为条件，每次根据在 m 位置上的值的大小决定如何缩小区间。

```
while(l + eps < r)
{
    double m = (l + r) / 2;
    if(cal(m) <= x) l = m;
    else r = m;
}
```

- 一般需要保留 k 位小数时，取 $eps = 10^{-(k+2)}$。
- 如果精度不容易确定或表示时，会采用循环固定次数的二分方法。

```
for(int i = 1; i <= 100; ++i)
{
    double m = (l + r) / 2;
    if(cal(m) <= x) l = m;
    else r = m;
}
```

【例】设计书架

【题目描述】

有 n 本书，第 i 本书的厚度为 a_i。现在将它们按照顺序摆放在一个 m 行的书架上，现在请你设计一款书架使得书能被摆放在书架里且书架宽度最小。

【输入格式】

第一行两个整数 $n, m(1 \leq m \leq n \leq 2 \times 10^5)$，分别表示书本的数目和书架的行数。

第二行 n 个整数，表示这 n 本书的厚度，输入保证 $1 \leq a_i \leq 10^5$。

【输出格式】

一行一个整数表示最小的书架宽度。

【输入样例 1】

```
3 2
2 1 3
```

【输出样例 1】

```
3
```

【输入样例 2】

```
10 4
4 7 6 1 4 1 1 8 2 10
```

【输出样例 2】

```
12
```

【例】设计书架

- 该问题通过推导、找规律等方法很难求出书架的最小宽度。
- 但是如果问这 n 本书能否摆放在一个宽度为 k 的书架上？这显然是比较简单的。
- 那么可以从小到大枚举书架的宽度，直到书能被正好放在书架上时，书架的宽度就是答案。

【例】设计书架

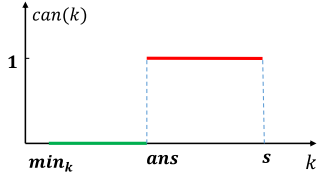
```
bool can(long long k)
{
    int p = 1; // 当前正在使用第 p 层书架
    long long r = k;   // 当前层的书架剩余宽度
    for(int i = 1; i <= n; ++i)
    {
        if(a[i] <= r) r -= a[i];
        else ++p, r = k - a[i];
    }
    return p <= m; // 使用的书架层数不能多于 m
}

long long k = max(1ll, s / m);  // 枚举的书架宽度
for(int i = 1; i <= n; ++i) k = max(k, a[i]);
while(!can(k)) ++k;  // 只要书架宽度不足就增加
cout << k;
```

- 每次尝试都需要将所有书依次放在书架上，尝试的次数如果过多，那么我们将会花费大量时间。有没有方法能减少尝试的次数呢？

【例】设计书架

- 显然，在从小到大枚举书架宽度的过程中，开始时书无法全部放入书架，但当书架的宽度增加到一定程度，书就可以全部放入书架，而且后续枚举更宽的书架时显然也是可以放入全部书籍的。
- 将求解书架宽度的问题转化为给定一个书架宽度 k，判断这 n 本书能否放入一个宽度为 k 的书架上的问题，求满足条件的最小的 k。由此得出的评价函数如下：

$$can(k) = \begin{cases} 0, & k < ans \\ 1, & k \geq ans \end{cases}$$



- 评价函数在 $[min_k, s]$ 中单调递增，于是问题变为：查找使得 $can(k) = 1$ 的最小的 k。因此可以二分枚举书架的宽度 k，判断这 n 本书能否放入一个宽度为 k 的的书架上，根据 $can(k)$ 的值，缩小答案所在的区间。

【例】设计书架

```
bool can(long long k)
{
    int p = 1; // 当前正在使用第 p 层书架
    long long r = k;   // 当前层的书架剩余宽度
    for(int i = 1; i <= n; ++i)
    {
        if(a[i] <= r) r -= a[i];
        else ++p, r = k - a[i];
    }
    return p <= m; // 使用的书架层数不能多于 m
}

long long l = max(1ll, s / m), r = s;
for(int i = 1; i <= n; ++i) l = max(l, a[i]);
while(l < r)
{
    int k = (l + r) / 2;
    if(can(k)) r = k;
    else l = k + 1;
}
cout << l;
```

【例】愤怒的牛

【题目描述】

农夫约翰建造了一座有 n 间牛舍的小屋，牛舍排在一条直线上，第 i 间牛舍在 x_i 的位置，但是约翰的 m 头牛对小屋很不满意，因此经常互相攻击。为了防止牛之间互相伤害，约翰决定自己分配牛舍使任意两头牛之间的最小距离尽可能的大。那么，这个最大的最小距离是多少呢?

【输入格式】

第 1 行有两个整数 n 和 $m(2 \le m \le n \le 10^5)$;

接下来 n 行，第 i 行一个整数 $x_i(0 \le x_i \le 10^9)$，表示第 i 间牛舍的位置。

【输出格式】

一个整数，表示最大的最小距离。

【输入样例】

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

【输出样例】

```
3
```

【样例解释】

共有 5 间牛舍，它们的位置为 $1, 2, 4, 8, 9$，需要安排 3 头牛，为了让奶牛们尽可能离的远，显然分配到 $1, 4, 9$ 最合适。

【例】愤怒的牛

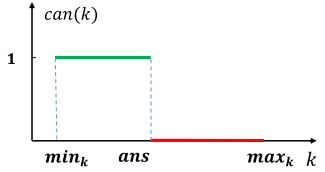
- 如果要求奶牛之间的间距过大，牛舍可能无法安排所有奶牛，而随着要求的间距减少，牛舍就可以安排所有奶牛，要求解的就是这个最大满足条件的间距。
- 那么可以从大到小枚举要求的间距，判断牛舍能否安排所有奶牛，直到能安排所有奶牛。

```
1  bool can(int k) // 在要求最小间距为 k 的情况下能否安排所有奶牛
2  {
3      int p = 1; // 当前可以安排 p 头牛
4      int last = 1;  // 上一头牛所在的牛舍
5      for(int i = 2; i <= n; ++i)
6          if(a[i] - a[last] >= k) ++p, last = i;
7      return p >= m;
8  }
9
10 sort(a + 1, a + n + 1);
11 int k = a[n] - a[1];   // 牛之间的最小间隔
12 while(!can(k)) --k;
13 cout << k;
```

【例】愤怒的牛

- 上述算法将求解间距的问题转化为给定一个间距 k，判断奶牛能否全部安排的问题，求最大的满足条件的 k。评价函数如下：

$$can(k) = \begin{cases} 1, & k \le ans \\ 0, & k > ans \end{cases}$$



- 评价函数在 $[min_k, max_k]$ 中单调递减，于是问题变为：查找使得 $can(k) = 1$ 的最大的 k。因此可以二分枚举间距 k，判断奶牛能否全部安排，根据 $can(k)$ 的值，缩小答案所在的区间。

```
sort(a + 1, a + n + 1);
int l = a[n] - a[1], r = a[n] - a[1];
for(int i = 1; i <= n - 1; ++i) l = min(l, a[i + 1] - a[i]);
while(l < r)
{
    int k = (l + r + 1) / 2;
    if(can(k)) l = k;
    else r = k - 1;
}
cout << l;
```

【例】最长子段

【题目描述】
给定 n 个整数 a_i，求一个尽可能长的非空连续子段使得子段和 $\leq S$。
【输入格式】
第一行两个整数 $n, S(1 \leq n \leq 2 \times 10^5, 1 \leq S \leq 10^9)$。
第二行 n 个整数 $a_i(-10^9 \leq a_i \leq 10^9)$。
【输出格式】
一个整数，最长非空连续子段的长度。
【输入样例】

```
6 10
-1 2 7 -4 9 12
```

【输出样例】

```
4
```

【样例解释】
最长的满足条件的子段是 $-1, 2, 7, -4$。

【例】最长子段

- 枚举 1：枚举所有子段，判断子段和是否满足 $\leq S$，求最长的子段。

```
int ans = 0;
for(int r = 1; r <= n; ++r)
    for(int l = 1; l <= r; ++l)
        if(s[r] - s[l - 1] <= S) ans = max(ans, r - l + 1);
cout << ans;
```

- 枚举 2：从大到小枚举子段长度，判断对应长度的子段是否满足 $\leq S$，求最长的子段。

```
int ans = 0;
for(int len = n; len >= 1; --len)
{
    for(int r = len; r <= n; ++r)
    {
        int l = r - len + 1;
        if(s[r] - s[l - 1] <= S) ans = max(ans, len);
    }
}
cout << ans;
```

- 时间复杂度：$O(N^2)$。

【例】最长子段

- 显然可以从大到小枚举子段长度 k，判断是否存在长度为 k 的子段满足子段和 $\leq S$。

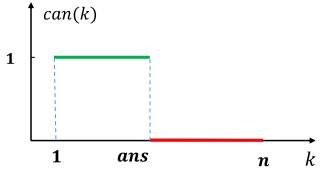
```
1  // s[] 是前缀和数组
2  bool can(int k)    // 是否存在长度为 k 的子段满足条件
3  {
4      for(int r = k; r <= n; ++r)
5          if(s[r] - s[r - k] <= S) return true;
6      return false;
7  }
8
9  int k = n;
10 while(!can(k)) --k;
11 cout << k;
```

- 上述判定能否作为二分答案的评价函数？为什么？

【例】最长子段

- 上述评价函数不具备单调性，不能作为二分答案的评价函数。
- 修改评价函数为：对于枚举的长度 k，判断是否存在长度大于等于 k 的子段满足字段和 $\leq S$。评价函数如下：

$$can(k) = \begin{cases} 1, & k \leq ans \\ 0, & k > ans \end{cases}$$



- 评价函数在 $[1, n]$ 中单调递减，于是问题变为：查找使得 $can(k) = 1$ 的最大的 k。因此可以二分 k 的大小，判断是否存在长度大于等于 k 的子段满足字段和 $\leq S$，根据得出的 $can(k)$ 的值，缩小答案所在区间。

【例】最长子段

```
// s[] 是前缀和数组
bool can(int k) // 是否存在长度大于等于 k 的子段满足条件
{
    for(int r = k; r <= n; ++r)
        for(int l = 1; l <= r - k + 1; ++l)
            if(s[r] - s[l - 1] <= S) return true;
    return false;
}

int l = 1, r = n;
while(l < r)
{
    int k = (l + r + 1) / 2;
    if(can(k)) l = k;
    else r = k - 1;
}
cout << l;
```

- 时间复杂度：$\mathrm{O}(N^2 \log N)$。

【例】最长子段

- 上述算法的时间复杂度主要受限与评价函数的时间复杂度，如何降低?
- 对于枚举的区间右端点 r，如果想让长度 $\geq k$ 的子段和尽可能小，那么只需要让区间左端点的前缀和最大即可，也即 $\max\limits_{0\leq l\leq r-k} s[l]$。
- 通过观察可以发现，随着右端点 r 的枚举，每一次只有一个新的区间左端点进入范围，那么只需要一个变量维护前缀和最小值即可。

```
1  // s[] 是前缀和数组
2  bool can(int k) // 是否存在长度大于等于 k 的子段满足条件
3  {
4      long long x = s[0]; // 前缀和最小值
5      for(int r = k; r <= n; ++r)
6      {
7          x = max(x, s[r - k]); // 对于当前右端点有一个新的区间左端点
8          if(s[r] - x <= S) return true;
9      }
10     return false;
11 }
```

- 时间复杂度：评价函数的时间复杂度为 $\mathrm{O}(N)$，故而整体复杂度为 $\mathrm{O}(N\log N)$。

【例】Best Cow Fences

【题目描述】

农夫约翰的农场由 n 个牧场构成，每一个牧场内有一定数量的奶牛 a_i。约翰想要在这些牧场中用栅栏划出一组相邻的牧场来构成一个特殊的牧区，这个牧区要至少包含 F 个牧场，牧区内牧场的平均奶牛数量为该划分方案的平均值。求所有有划分方案的平均值中的最大值。

【输入格式】

第一行：两个空格隔开的整数 $n(1 \leq n \leq 10^5)$ 和 $F(1 \leq F \leq n)$;

接下来一行 n 个整数，表示牧场内牛的数量 $(0 \leq a_i \leq 2000)$。

【输出格式】

输出只有一行，一个整数表示最大平均值的 1000 倍 (下取整)。

【输入样例】

```
10 6
6 4 2 10 3 8 5 9 4 1
```

【样例解释】

最长的满足条件的子段是 $10, 3, 8, 5, 9, 4$，平均值为 6.5，故而答案为 6500。

【输出样例】

```
6500
```

【例】Best Cow Fences

- 问题实际上要求的是所有长度大于 F 的子段的平均值的最大值。

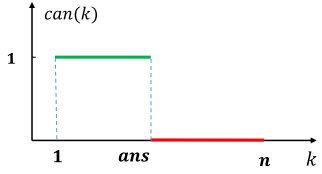
```
double k = 0;
for(int r = F; r <= n; ++r)
    for(int l = 1; l <= r - F + 1; ++l)
        k = max(k, (s[r] - s[l - 1]) / (r - l + 1));
cout << (int)(k * 1000);
```

- 时间复杂度：$O(N^2)$。
- 如何用二分答案优化？二分什么？评价函数是什么？

【例】Best Cow Fences

- 二分平均值 k，判断是否存在一个长度 $\geq F$ 的子段的平均数大于等于 k，求满足条件的最大的平均值。评价函数 $can(k$ 如下：

$$can(k) = \begin{cases} 1, & k \leq ans \\ 0, & k > ans \end{cases}$$



- 评价函数在 $[1, 2000]$ 中单调递减，于是问题变为：查找使得 $can(k) = 1$ 的最大的 k。二分平均值 k，判断是否存在一个长度 $\geq F$ 的子段平均数 $\geq k$，根据 $can(k)$ 的值，缩小答案所在的区间。

【例】Best Cow Fences

```
1  bool can(double k)  // 是否存在平均值大于等于 k 的区间
2  {
3      for(int r = F; r <= n; ++r)
4          for(int l = 1; l <= r - F + 1; ++l)
5              if((s[r] - s[l - 1]) / (r - l + 1) >= k) return true;
6      return false;
7  }
8
9  double l = 1, r = 2000;
10 while(l + 1e-6 < r)
11 {
12     double k = (l + r) / 2;
13     if(can(k)) l = k;
14     else r = k;
15 }
16 cout << (int)(r * 1000);
```

- 时间复杂度：$O(N^2 \log N)$，比直接枚举的时间复杂度还要差。

【例】Best Cow Fences

- 优化：每一次二分平均值 k 时，将每个牧场的奶牛数量都减去 k，那么问题就变为牧区内牧场奶牛数总和非负。那么问题变为：是否存在一个长度 $\geq F$ 的子段，子段和非负 (≥ 0)，这与最长子段问题类似。

```
1  bool can(double k)   // 是否存在平均值大于等于 k 的区间
2  {
3      // 每一次重新求减去 k 后的前缀和
4      for(int i = 1; i <= n; ++i) s[i] = s[i - 1] + a[i] - k;
5      double p = s[0];   // 最小前缀和
6      for(int r = F; r <= n; ++r)
7      {
8          p = min(p, s[r - F]);
9          if(s[r] - p >= 0) return true;
10     }
11     return false;
12 }
```

- 时间复杂度：$\mathrm{O}(N \log N)$。

【例】子段和问题

【题目描述】

给定一个序列 a 中包含 n 个正整数，现在给你一个正整数 S。请你编写一个程序求一个长度最小的连续子段，使得子段的和大于等于 S。

【输入格式】

第一行两个整数 $n, S(10 \leq n \leq 10^5, 1 \leq S \leq 10^8)$。

第二行包含 n 个正整数 $a_i(1 \leq a_i \leq 10^4)$。

【输出格式】

输出一个整数，表示满足条件的最小长度，如果不存在，输出 0。

【输入样例】

```
10 15
5 1 3 5 10 7 4 9 2 8
```

【样例解释】

子段 $10, 7$ 之和 $17 \geq 15$，而且是最短的。

【输出样例】

```
2
```

【例】子段和问题

- 枚举所有子段，找到长度最小的连续子段，且字段和大于 $\geq S$。
- 因为要求子段和 (区间和)，所以先求出序列 a_i 的前缀和 s_i。

```
1  int ans = N;
2  // 枚举算法 1
3  for(int r = 1; r <= n; ++r)  // 枚举子段右端点
4      for(int l = 1; l <= r; ++l) // 枚举子段左端点
5          if(s[r] - s[l - 1] >= S) ans = min(ans, r - l + 1);
6
7  // 枚举算法 2
8  for(int len = 1; len <= n; ++len) // 枚举子段长度
9      for(int l = 1; l <= n - len  + 1; ++l)
10     {
11         int r = l + len -1;
12         if(s[r] - s[l - 1] >= S) ans = min(ans, r - l + 1);
13     }
14 if(ans == N) cout << 0;
15 else cout << ans;
```

- 时间复杂度：O(N^2)，如何优化?

【例】子段和问题

- $a_i \geq 0$，所以子段长度越大，子段和更容易 $\geq S$。
- 考虑枚举算法 2，评价函数为：对于枚举的子段长度 k，判断是否存在长度为 k 的子段使得字段和 $\geq S$。显然，这个评价函数在 $[1, n]$ 上是单调递增的。
- 问题转化为：查找使得 $can(k) = 1$ 的最小的 k，可以使用二分答案。

```
bool can(int k)    // 是否存在长度为 k 的子段子段和>=S
{
    for(int r = k; r <= n; ++r) if(s[r] - s[r - k] >= S) return true;
    return false;
}
int l = 1, r = n;
while(l < r)
{
    int k = (l + r) / 2;
    if(can(k)) r = k;
    else l = k + 1;
}
if(l == n && s[n] < S) cout << 0;
else cout << ans;
```

- 时间复杂度：$\mathrm{O}(N \log N)$。

【例】子段和问题

- $a_i \geq 0$，所以子段长度越大，子段和更容易 $\geq S$。
- 考虑枚举算法 2，对于枚举的子段右端点 j，左端点越小，子段和更容易 $\geq S$，那么需要在所有 $s[j]-s[k-1] \geq S$ 的 k 中找到最大的一个即可，可以二分枚举 k。

```
int ans = N;
for(int j = 1; j <= n; ++j)
{
    int l = 1, r = j;  // 二分找到 s[j] - s[k - 1] >= S 的最大的 k
    while(l < r)
    {
        int k = (l + r + 1) / 2;
        if(s[j] - s[k - 1] >= S) l = k;
        else r = k - 1;
    }
    if(s[j] - s[l - 1] >= S) ans = min(ans, j - l + 1);
}
if(l == n && s[n] < S) cout << 0;
else cout << ans;
```

- 时间复杂度：$\mathrm{O}(N \log N)$。

【例】子段和问题

- 对于枚举的右端点 r，设其满足要求的最大的左端点为 l，那么当 $r \to r+1$ 时，l 可能不动或者向右移动，绝对不会向左移动。
- 故而，对于枚举的右端点 r，不需要每次二分查找左端点 l，而是只需要维护 l，在 $s[l+1,r] \geq S$ 的情况下尽可能向右移动，那么停止的位置就是对应枚举的右端点 r 的最大的左端点。

```
int ans = N;
int l = 1;
for(int r = 1; r <= n; ++r)
{
    while(l < r && s[r] - s[l] >= S) ++l;  // s[l+1,r] < S 结束
    if(s[r] - s[l - 1] >= S) ans = min(ans, r - l + 1);
}
if(ans == N) cout << 0;
else cout << ans;
```

- 时间复杂度：O(N)。

【例】化装晚会

【题目描述】
农夫约翰想让奶牛们参加化装晚会，但是不幸的是他只有一件服装。服装只好适合两头体重和小于等于 S 的情况，约翰有 n 头牛要来参见这场舞会，这些牛编号从 1 到 n，其中第 i 头牛的体重为 $a[i]$，约翰想知道有多少对牛能穿这件服装。

【输入格式】
第一行两个整数 $n, S(2 \leq n \leq 2 \times 10^4, 1 \leq S \leq 10^6)$。
第二行包含 n 个正整数 $a_i(1 \leq a_i \leq 10^6)$。

【输出格式】
输出一个整数，表示选择的所有方案数。注意奶牛顺序不同的两种方案是被视为相同的。

【输入样例】

```
4 6
3 5 2 1
```

【输出样例】

```
4
```

【样例解释】
4 种选择分别为：奶牛 1 和奶牛 3；奶牛 1 和奶牛 4；奶牛 2 和奶牛 4；奶牛 3 和奶牛 4。

【例】化装晚会

- 枚举所有的奶牛对，判断是否符合体重之和 $\leq S$ 即可。
- 注意避免重复枚举。

```
int cnt = 0;
for(int i = 1; i <= n; ++i)
    for(int j = i + 1; j <= n; ++j)
        if(a[i] + a[j] <= S) ++cnt;
cout << cnt;
```

- 时间复杂度：$\mathrm{O}(N^2)$。

【例】化装晚会

- 将奶牛按照体重从小到大排序。
- 枚举左侧的奶牛 i，在奶牛 i 的右侧寻找能与之配对的奶牛 j，那 $a[i]+a[j]$ 是单调递增的，因此可以二分找到最远的 $a[i]+a[j]\le S$ 的奶牛位置 j，则符合条件的奶牛对数增加 $j-i$。

```
sort(a + 1, a + n + 1);
int cnt = 0;
for(int i = 1; i <= n - 1; ++i)
{
    int l = i + 1, r = n;   // 二分查找 a[i]+a[j]<=S 的最大的 k
    while(l < r)
    {
        int k = (l + r + 1) / 2;
        if(a[i] + a[k] <= S) l = k;
        else r = k - 1;
    }
    if(a[i] + a[l] <= S) cnt += (l - i);
}
cout << cnt;
```

- 时间复杂度：$\mathrm{O}(N\log N)$。

【例】化装晚会

- 因为奶牛已经按照体重从小到大排序，所以在从左向右枚举左侧的奶牛 i 时，$a[i]$ 在递增，所以与之匹配的最远的奶牛 j 的 $a[j]$ 应该是递减的，那么在整个枚举的过程中 j 只会向左侧移动。
- 所以只要维护一个 j，在 i 右移时，对 j 进行左移即可。

```
sort(a + 1, a + n + 1);
int cnt = 0;
int j = n;
for(int i = 1; i <= n - 1; ++i)
{
    while(i < j && a[i] + a[j] > S) --j;
    if(i < j && a[i] + a[j] <= S) cnt += (j - i);
}
cout << cnt;
```

- 排序时间复杂度 $\mathrm{O}(N \log N)$，枚举奶牛时间复杂度 $\mathrm{O}(N)$，所以算法的整体时间复杂度为 $\mathrm{O}(N \log N)$。

总结

- 二分答案就是将求解问题转化为一个评价问题，而且评价函数有单调性，可以用二分的方法来解决。
- 二分答案的关键在于确定评价函数和二分的方式。
- 常见问题：最大值最小或最小值最大。

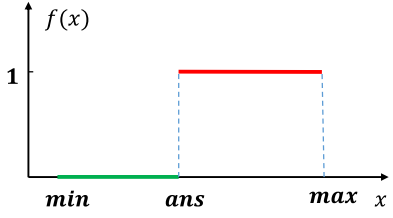


图: 最大值最小化

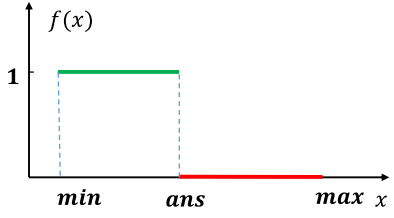


图: 最小值最大化

- 在某些情况下，利用双指针扫描的方法可以加快算法的速度。

练习

- 查找 $\geq x$ 的最小数 (COGS 2882)
- 查找 $\leq x$ 的最小数 (COGS 2887)
- 设计书架 (COGS 3427)
- 愤怒的牛 (COGS 3195)
- 最长子段 (COGS 3428)
- Best Cow Fences(COGS 3188)
- 山头狙击战 (COGS 1092)
- 数列分段 2(COGS 3196)
- 子段和问题 (COGS 3980)
- 化装晚会 (COGS 140)
- 零落尘 (COGS 3978)
- Innovative Business(COGS 2851)

练习

- 防线 (COGS 1022)
- Corral the Cows(COGS 1991)
- 聪明的质检员 [NOIP 2011](COGS 631)
- 借教室 [NOIP 2012](COGS 1266)
- Brownie Slicing G(COGS 532)