算法入门

湖南师大附中 许力

二分思想

二分查找

- 二分思想其实是分治思想的进一步发展或者应用，可以归入分治一类，也可以单列

- 二分查找是我们最早接触到的二分思想：
- 通过每次把待查找区间缩减一半的策略，我们能实现在O(logn)的复杂度内查找O(n)的**单调**区间

二分答案

- 但其实在计数类问题中，因为我们要得到的答案的所有可能的解也会构成一个单调区间

- 因此在这一类问题中，二分也是一种重要的思考方向：二分答案

二分答案

- 二分答案大致的思路：我们会先预设一个答案，然后检验这个答案是否符合题意。

- 只有两种可能：

1. 符合：那我们有可能猜小了，也许更大一点儿也能行
2. 不符合：那我们肯定猜大了，需要减小一点儿

- 然后根据检验的结果再修正、再检验
- 直到大一点小一点都不行，那就正好是要找的答案

二分答案

- 因此这一类问题，本质上也是“猜数游戏”

- 只有两点区别：

1. 猜数游戏可以直接回答猜大猜小，而问题复杂以后，需要我们自己写个检验函数：假定答案就是它，带入看是否合法（符合题意）
2. 猜数游戏有明确的解空间：比如100以内的正整数，而这里没有，需要我们来限定解空间的上下限

陶陶摘苹果II

- 果园里有n棵果树排成一行，其中第i棵果树上结了a_i个苹果。现在按照果树编号依次摘取苹果。如果果篮的剩余容量大于等于当前果树所结的果子，那么就可以将该树上的苹果全收下来，否则就要拿一个新的篮子（也就是一棵树上的所有苹果必须完整地放入一个果篮）
- 现在最多能用k个果篮，要按照以上方法收完所有苹果，陶陶需要最小什么容量规格的果篮？

n,k≤100,000

Sample input	Sample output
9 3 //n、k 1 2 3 4 5 6 7 8 9 //a_i	17

分析

- 如果果篮容量足够大，就可以一个篮子装下所有树上的苹果，不管有没有k个篮子的数量限制（因为k是上限）
- 如果果篮容量足够小，小到只需要装下苹果最多的那棵果树上的苹果就可以，但这样果篮数量很可能会突破k的上限

- 因此，这个问题实际上是问：果篮容量至少要大到多少，才可以在k个篮子的数量限制下完成任务
- 这是最小值最大化的问题

分析

- 明确了要二分答案之后，我们首先要确定解空间的范围
- 因为篮子至少要能完整装下苹果最多的那棵树上的苹果（否则那棵树的苹果将没办法被摘取）
- 所以解空间的范围即是：$[\max(a_i), \sum(a_i)]$

```
for (int i = 1; i <= n; i ++)
{
    scanf("%d", &a[i]);
    l = max(l, a[i]);    // l是解空间的左端点
    r += a[i];     // r是解空间的右端点
}
```

分析

- 然后我们就开始用二分法“猜数”
- 当然往中间猜：

```
int mid = (l + r) / 2;
```

- 如果mid是合法的，那么我们可能浪费了一点篮子的容量，缩小一点；反之，必须增加一点篮子的容量

```
while (l < r)
{
    int mid = (l + r) / 2;      // 二分答案 mid
    if (mid是合法的) r = mid; // mid是合法的则减小容量
    else l = mid + 1;  // mid不合法则增加容量
}
```

分析

- 那怎么知道mid是不是合法呢？
- 我们需要为此特地写一个检验函数：我们假定正确答案就是mid

```
int ans = 0, v = 0;  //ans是篮子个数，v是篮子已装容量
for (int i = 1; i <= n; i ++)
{
    if (a[i] <= mid - v) v += a[i]; //能放下
        else {ans ++; v = a[i];}  //放不下，拿一个新篮子
}
```

- 于是我们可以统计出当篮子容量为mid的时候，需要的篮子数量ans

分析

- 然后判断这个ans有没有突破k的限制

```
return ans <= k;
```

- 一个细节，考虑到二分是区间收缩，需要左右端点，因此循环范围上做一点调整，否则无法处理最后一棵果树

```
int ans = 0, v = 0;  //ans是篮子个数，v是篮子已装容量
for (int i = 1; i <= n + 1; i ++)
{
    if (i <= n && a[i] <= mid - v) v += a[i]; //能放下
        else {ans ++; v = a[i];}  //放不下，拿一个新篮子
}
```

参考代码

```cpp
#include<bits/stdc++.h>
using namespace std;
int a[100010], n, k, l, r;
bool check(int x)
{
    int ans = 0, v = 0;
    for (int i = 1; i <= n + 1; i ++)
    {
        if (i <= n && a[i] <= x - v) v += a[i];
            else {ans ++; v = a[i];}
    }
    return ans <= k;
}
```

```cpp
int main()
{
    scanf("%d%d", &n, &k);
    for (int i = 1; i <= n; i ++)
    {
        scanf("%d", &a[i]);
        l = max(l, a[i]);
        r += a[i];
    }
    while (l < r)
    {
        int mid = (l + r) / 2;
        if (check(mid)) r = mid;
        else l = mid + 1;
    }
    printf("%d", l);
    return 0;
}
```

二分答案

- 能够二分答案的题，通常带有这样的特征

1. 要求最值最优化：

- 即最大值最小化，或者最小值最大化

2. 解空间具有单调性：

- 如果x满足条件，那么所有比x大的都满足条件
- 如果x不满条件，那么所有比x小的也都不满足条件

二分答案

- 二分答案的题通常还带有一个隐蔽的特征：
- 因为二分的复杂度是O(logn)的，所以相比O(n)的贪心、$O(n^2)$/O(nlogn)的动态规划，能支持更大的数据范围。
- 所以一旦数据范围巨大无比，就需要考虑是不是有O(n)/O(logn)，甚至O(1)的做法

跳石头

- 长为L的河道中有n块石头供落脚，要求至多移走m块石头（以上均不含起点和终点的石头）使得最短跳跃距离尽可能长。
- 现在给出每块石头与起点的距离，问跳跃距离为多少

n,m≤50,000，L≤1,000,000,000

Sample input	Sample output
25 5 2 //L、n、m 2 11 14 17 21 //d_i	4

分析

- 标准的二分答案题

- 如果不限制移走石子数量，最短距离可以是两个石子间的最小距离
- 反之，最短距离也可以是一次跳过所有石子（不考虑实际情况）

- 因此，这道题实际也是问最小值的最大化

分析

- 于是我们二分这个跳跃距离：

```
int l = 1, r = L;
while (l < r)
{
    int mid = (l + r) / 2;
    if (mid是合法的) ans = mid, l = mid + 1;
        else r = mid;
}
```

分析

- 检验答案是否合法

```
int cnt = 0, last = 0;
// cnt为移走的石子数量，last为上一个石子编号
for (int i = 1; i <= n; i ++)
    if (a[last] + x > a[i]) cnt ++;  //说明可以跳到 i，把 i移走换下一个 i
        else last = i;  //无法移走，下一步就从 i起跳

    return cnt <= m;
 }
```

- 注意这里因为起点和终点都不在讨论之列，所以无需像上题那样处理边界

参考代码

```
#include<bits/stdc++.h>
using namespace std;
int a[5000010], L, n, m, ans;
bool check(int x)  //检验跳跃距离 x是否合法
{
    int cnt = 0, last = 0;
    // cnt为移走的石子数量，last为上一个石子编号
    for (int i = 1; i <= n; i ++)
        if (a[last] + x > a[i]) cnt ++;
    //说明可以跳到 i，把 i移走换下一个 i
            else last = i;
    //无法移走，下一步就从 i起跳
    return cnt <= m;
 }
```

```
int main()
{
    scanf("%d%d%d", &L, &n, &m);
    for (int i = 1; i <= n; i ++)
        scanf("%d", &a[i]);
    a[n+1] = L;
    int l = 1, r = L;
    while (l < r)
    {
        int mid = (l + r) / 2;
        if (check(mid)) ans = mid, l = mid + 1;
            else r = mid;
    }
    printf("%d", ans);
    return 0;
}
```

课外加练

- luogu uva714　　抄书
- luogu 2678　　跳石头
- luogu 1577　　切绳子
- luogu 1824　　进击的奶牛
- luogu 2440　　木材加工
- luogu 1873　　砍树
- luogu 2005　　A/B problem

最小子序列长度

- 给定长度为n的数列，以及整数m。求和不小于m的连续子序列长度的最小值

n≤1,000,000

Sample input	Sample output
10 20 //n、m 5 1 3 5 10 7 4 9 2 8 //a_i	3

分析

- 暴力做法？枚举每个左端点，然后区间向右扫，直到和超过m就记录下长度并break掉（显然继续扫是没有意义的）
- 然后讨论下一个左端点，维护所有左端点得到的长度的最小值

- 复杂度$O(n^2)$

暴力大法

```cpp
#include<bits/stdc++.h>
using namespace std;
int a[1000010], n, m, ans = 0x3f3f3f3f;
int main()
{
    scanf("%d%d", &n, &m);
    for (int i = 0; i < n; i ++) scanf("%d", &a[i]);
    for (int i = 0; i < n; i ++)
    {
        int sum = 0;
        for (int j = i; j < n; j ++)
            if (sum >= m) {ans = min(ans, j - i); break;}
                else sum += a[j];
    }
    printf("%d", ans);
    return 0;
}
```

参考代码

- 那么可以怎么优化呢？

- 类似区间问题，双端点扫描一直是一把利器，就看怎么用得灵活
- 我们首先让r向右扫动，并把扫过的数加入和

```
r ++; sum += a[r];
```

5	1	3	5	10	7	4	9	2	8

l r

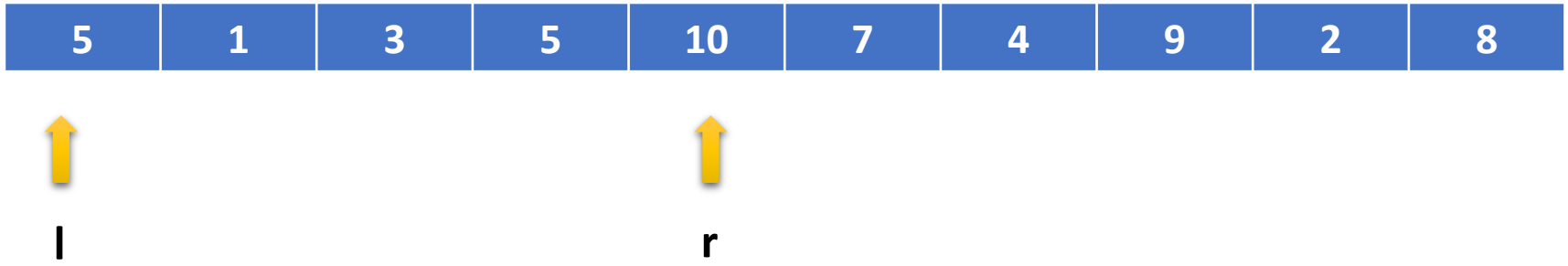
参考代码

- 直到sum>=m时停下来

- 这时候r不再急于向右扫动，我们让l动一步，
- 记得要从sum里面减掉a[l]

```
l ++; sum -= a[l];
```

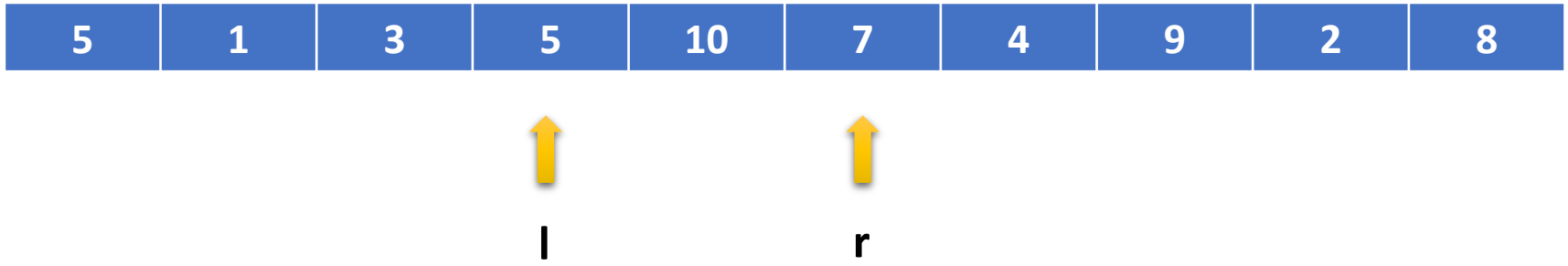
- 这时sum值必然是减小的



参考代码

- 我们的原则就是：

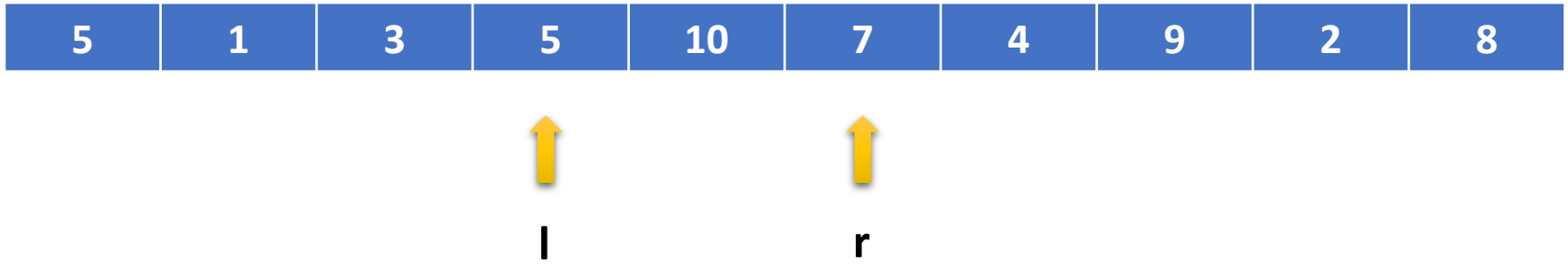
1. sum<m，说明l、r之间的间距太近，和不够，r扫动，并不断加上r扫过的值
2. sum>=m，说明l、r之间离得太远了，和太多很浪费，这样是不可能取到最优解的，于是l扫动，并不断减去l扫过的值



参考代码

- 在这个过程中维护区间长度的最小值（此时区间长度大部分时间都是符合题意，且比较优的）

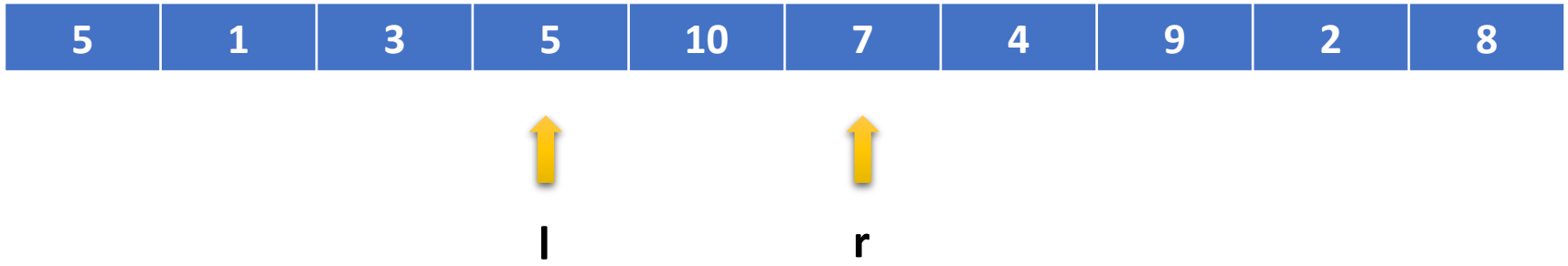
```
ans = min(ans, r - l + 1);
```



参考代码

- 我们既可以把l、r看成你追我赶的过程，也可以看作是有一把弹性长度的尺在数轴下方平移

- 这个算法叫“尺取法”，因为l、r合起来才扫过数列的长度×2（实际往往不到），因此复杂度O(n)
- 我们之前在“珠心算测验”一题中曾用过这个方法



参考代码

注意sum的起始值
另外我的代码没有考虑无解的情况，请大家自己补充上

```
#include<bits/stdc++.h>
using namespace std;
int a[1000010], n, m, l, r, ans = 0x3f3f3f3f;
int main()
{
    scanf("%d%d", &n, &m);
    for (int i = 0 ; i < n; i ++) scanf("%d", &a[i]);
    int sum = a[0];
    while (l <= r && r < n)
    {
        if (sum >= m)
        {
            ans = min(ans, r - l + 1);
            l ++; sum -= a[l];  //和过大，l向右扫
        }
        else {r ++; sum += a[r];} //和过小，r向右扫
    }
    printf("%d", ans);
    return 0;
}
```

效率对比

- 测试数据规模n=1,000,000， m=999,999,999

试题 选手

名称	排名	ruler	总分	总用时(s)	测试时间
尺取法	1	100	100	0.953	2019/1/17 17:07:54
暴力	1	100	100	50.281	2019/1/17 17:08:46

拓展一下

- 尺取法有什么限制？
- 比如：5、1、3、5、-10、7、4、9、2、8

- 如果有负数，还适用尺取法吗？
- 如果不能，还有什么办法？

课外加练

- luogu 1147　　连续自然数和
- luogu 1102　　A-B数对
- luogu 1638　　逛画展