算法入门

目录

贪心思想

贪心

- 贪心思想是指在对问题求解时，不从整体上加以考虑，而是做出在当前看来最好的选择

- 比如：有n件物品，每个物品都有其价值，现在要从中选出m件物品，使得其价值总和最大，怎么做？
- 按价值sort，然后取前m项
- 在这里，就是一种典型的贪心思想，只取当前能取到的最优值，必然能导致最终的最优值
- 反之，任何一次取的不是当前最优，都无法保证最终是最优的

贪心

- 所以：什么情况适用贪心？局部最优能导致全局最优

- 难道还有局部最优不能导致全局最优的？

背包

- 有n件物品，每件物品有一个体积。现在有一个容积为m的箱子，问最多能放多少体积的物品进去
- 样例：4件物品，体积分别为2、3、4、5，箱子容积为10

- 按照贪心思想，选取物品为5+4，但正确答案为5+3+2
- 这种情况，局部最优就无法导致全局最优

找零

- 有n种硬币，每种不限量，用于给顾客找零。现在要求找给顾客m元，要求使用硬币最少的方案
- 样例：硬币面值为1、4、5，要找给顾客8元

- 按照贪心思想，选取方案为5+1+1+1，正确答案4+4
- 类似的情况，因为限制条件增加，使得局部最优无法产生全局最优

漂流

- 同学们利用暑假去漂流。租用的独木舟是一样的规格，最多乘2人，而且承重有限制。现在给出每个人的体重，问最少需要租用几条独木舟？
- 样例：独木舟承重100，体重为45、60、65、50、75、60、70、55、50

1. 按体重sort，从大的开始讨论，如果之前的独木舟无法承载，就再租一条
2. 如果前面有多条独木舟可以载他，就选承载空间最小的一条

删数问题

- 给定一个大整数n，去掉其中任意m个数字后成为一个新的整数。现在要问新整数最小是多少？

$n \leqslant 10^{250}$

Sample input	Sample output
178543 4 //n，m	13

分析

- 首先可以明确，每次删除最大的数字，是错误的贪心策略
- 比如36204，要删掉两个数字，如果按照该贪心策略，得320
- 而正确答案是204

分析

- 那么我们讨论一下：36204

1. 因为删掉一个数，相当于前导的数被降权一位

- 依据这条规则，从最左开始扫一遍，发现3<6，先删6，得3204，比删3得6204更优

分析

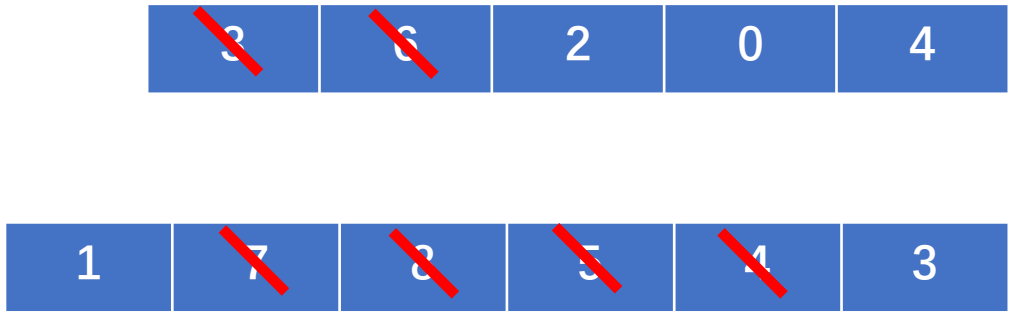
- 那么我们讨论一下：36204

2. 高位数变小，比低位数变小更优

- 依据这条规则，3204中，因为3>2，数字开始变小，这种情况下，不管2以及2后面是多大的数，都不如删3更优
- 比如3209，删9得320，删3得209，后者更优

分析

- 因此我们总结一下就是:
- 如果数字一直在上升（不下降），那么删掉这个上升序列中最后那一个，因为它最大
- 如果数字下降了，那么删掉当前这个数



参考代码


```cpp
#include<bits/stdc++.h>
using namespace std;
string n;
int main()
{
    int m; cin >> n >> m;
    int len = n.size();
    while (m --)
    {
        int k = len - 1;
        for (int i = 0; i < len - 1; i ++)
            if (n[i] > n[i+1]) {k = i; break;}
        n.erase(k, 1);
        len --;
    }
    while (n[0] == '0') n.erase(0, 1);  //清理高位零
    if (n != "") cout << n;  //有可能最后 n被删空
        else cout << "0";
    return 0;
}
```

最大乘积

- 一个正整数一般可以分为几个互不相同的自然数的和，如：3=1+2，4=1+3，5 = 1+4=2+3，6=1+5 = 2+4，…
- 将指定的正整数n分解成若干个互不相同的自然数的和，且使这些自然数的乘积最大

3≤n≤10000

Sample input	Sample output
10	2 3 5 30 //乘积

分析

- 如果要暴力做效率很低，因为不仅拆出来的数要逐一穷举，连拆出来的数的个数也要穷举

- 我们假设a=n-1，b=n+1，则$a \times b=(n-1) \times (n+1)=n^2-1$
- 而若和不变，但a=n-2，b=n+2，则$a \times b=(n-2) \times (n+2)=n^2-4$

- 所以一个数拆分出来的自然数，大小越接近，乘积越大

分析

- 同时，因子数越多，也能使得乘积更大
- 比如4×5<2×3×4，6×7<3×4×6

- 因此我们的贪心策略是：
- 将n分解成2×3×4×……的自然数序列
- 但是很可能无法分尽：比如15，只能分成2×3×4×5×1，最后这个1是浪费掉的，又或者11，分成2×3×4×2，最后这个2是重复的

分析

- 所以我们的贪心策略还需要进一步完善：
- 分到最后一个数a_i，如果$a_i<a_{i-1}$，说明最后这个a_i是不合要求的（和前面的重复）或者不优的（出现1）

- 但我们又需要尽可能维持自然数序列（因为这样最优）
- 那我们就使得sum=2+3+4+……，一旦发现sum>=n就停下来

分析

- 停下来之后有三种情况：

1. **sum=n**，此时的自然数序列即为答案
2. **sum-n=k**，此时k必然和之前的数重复（为什么？）我们从之前的数列中去掉k，剩下的序列即为答案

- 比如2+3+4+5=14>10，且14-10=4，去掉4后得2+3+5即为答案
- 又如2+3+4+5=14>11，且14-11=3，去掉3后得2+4+5即为答案

3. **sum-n=1**，此时序列中是没有1的，意味着因子数只能减少1个，去掉2，并且把最后一个数+1

- 比如2+3+4=9>8，且9-8=1，去掉2后，4+1，得3+5即为答案
- 又如2+3+4+5=14>13，且14-13=1，去掉2后，5+1，得3+4+6即为答案

参考代码

- 略

最大子段和

- 给定长度为n的序列，选出连续的一段非空子序列，使得这段子序列的和最大

n≤200,000

Sample input	Sample output
7 //n 2 -4 3 -1 2 -4 3	4

分析

- 显然不能单纯以负数的出现作为评判标准
- 比如样例数据：2、-4、3、-1、2、-4、3，和最大的子序列就包括了3、-1、2

- 我们考虑这样一个贪心策略：如果一个新数加入到现有的区间和中，使得和为正，那它一定是当前考查目标区间的一部分；反之，则要把该数连同之前的和都舍弃

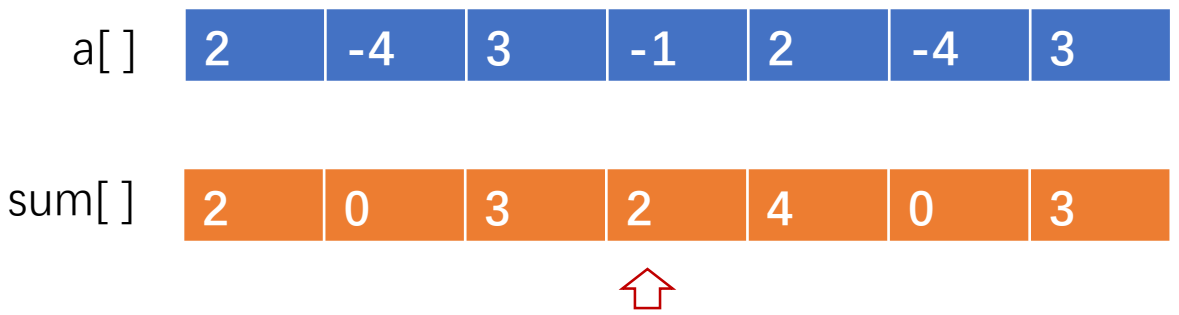
分析

- 为什么这样贪心是对的?
- 因为与其保留一个负数，不如从零开始更优
- 怎么理解？就是区间和无论是增长还是减少，但只要还为正，就都是对最终答案有贡献的，应该保留
- 所以有负数并不是问题，只要不是负得太多

分析

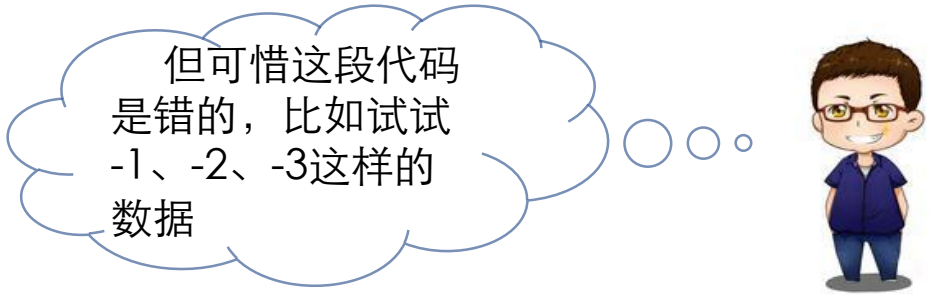
- 我们来看样例数据

```
sum[i] = max(0, sum[i-1] + a[i]);
```



这里虽然和减少，但依然有对最终的答案有贡献

参考代码




```cpp
#include<bits/stdc++.h>
using namespace std;
int a[200010], sum[200010], n, ans;
int main()
{
    scanf("%d", &n);
    for(int i = 1; i <= n; i ++)
    {
        scanf("%d", &a[i]);
        sum[i] = max(0, sum[i-1] + a[i]);
        ans = max(sum[i], ans);
    }
    printf("%d", ans);
    return 0;
}
```

修正一下

- 我们不应该武断的以“0”为默认最小值
- 就用数组首项即可，或者按题意用-10000

参考代码

```cpp
#include<bits/stdc++.h>
using namespace std;
int a[200010], sum[200010], n, ans = -10000;
int main()
{
    scanf("%d", &n);
    for(int i = 1; i <= n; i ++)
    {
        scanf("%d", &a[i]);
        sum[i] = max(a[i], sum[i-1] + a[i]);
        ans = max(sum[i], ans);
    }
    printf("%d", ans);
    return 0;
}
```

铺设道路

春春是一名道路工程师，负责铺设一条长度为 n 的道路。

铺设道路的主要工作是填平下陷的地表。整段道路可以看作是 n 块首尾相连的区域，一开始，第 i 块区域下陷的深度为 d_i 。

春春每天可以选择一段连续区间 $[L,R]$ ，填充这段区间中的每块区域，让其下陷深度减少 1。在选择区间时，需要保证，区间内的每块区域在填充前下陷深度均不为 0 。

春春希望你能帮他设计一种方案，可以在最短的时间内将整段道路的下陷深度都变为 0 。

Sample input	Sample output
6 //n 4 3 2 5 3 5 // d_i	9

分析

- 暴力怎么做?

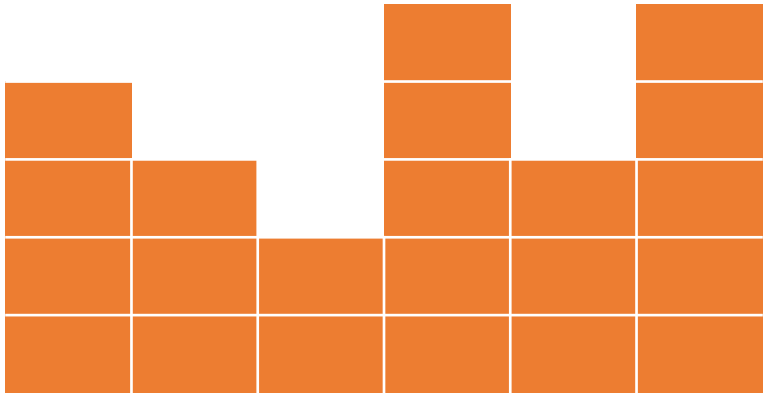
1. 从头到尾扫一遍，每次所有元素减1
2. 一旦发现了0，区间就被打断
3. 被打断后的区间重新做同样的操作
4. 直到全部是0为止

分析

- 不那么暴力的暴力怎么做?

1. 既然要持续减1，那不如每次减掉区间内的最小值，一次减干净
2. 每减1次，必然会出现0
3. 新产生的区间做同样的操作
4. 这样复杂度从$O(n^2)$降到O(nlogn)

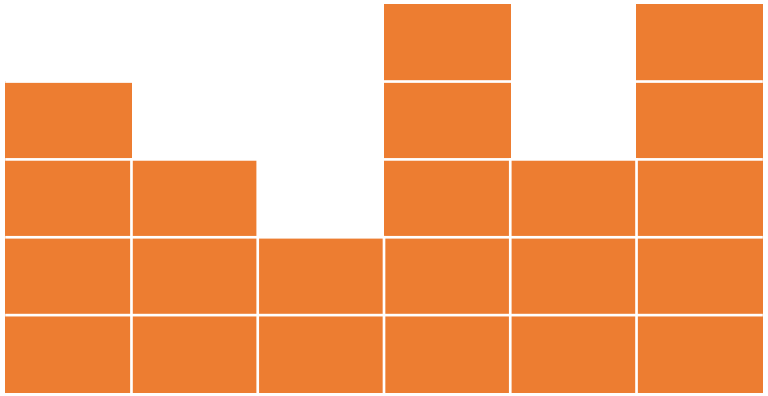
分析



- 我们从左边起点开始扫一遍

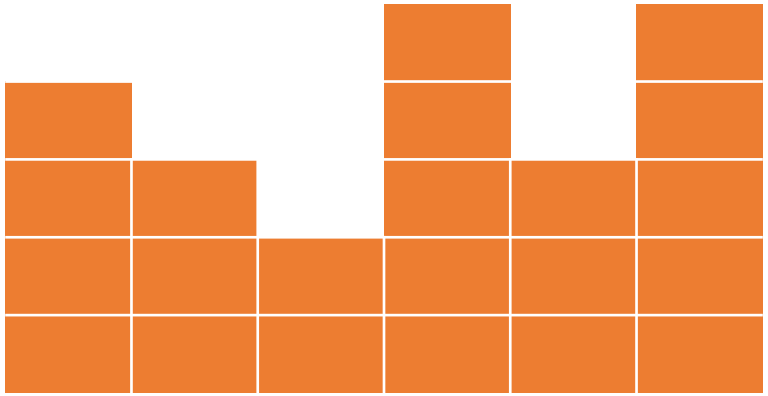
- 如果L>R（右边的坑比较浅），说明右边的R不必单独处理，在处理之前的L时会附带处理

分析



- 我们从左边起点开始扫一遍
- 如果L<R（右边的坑比较深），那么它们之间的差值R-L，是无论如何也无法附带处理的，这就必须单独消耗R-L的天数
- 因此答案就是每一对R-L（R>L时）的和

分析



- 为什么这样的答案是最优的?

- 我们要天数尽量少，就需要每天处理的区间尽量长一点
- 而持续下降的区间可以毫无顾忌的延续下去（L>R）
- 反之，则要考虑某些比较浅的坑深度为0（被填平了）

参考代码

```
6
4 3 2 5 3 5
9
--------------------------------
Process exited after 3.945 seconds wi
th return value 0
请按任意键继续. . .
```

- 复杂度O(n)

```cpp
#include<bits/stdc++.h>
using namespace std;
int a[100010], n, ans;
int main()
{
    scanf("%d", &n);
    for (int i = 1; i <= n; i ++)
    {
        scanf("%d", &a[i]);
        if (a[i] > a[i-1]) ans += a[i] - a[i-1];
    }
    printf("%d", ans);
    return 0;
}
```

比赛日程

- 怎样提高OI水平？我认为：参加的比赛越多，水平越高
- 现在给出n场比赛的开始和结束时间a_i、b_i，问最多能参加多少场？

n⩽1,000,000

Sample input	Sample output
3 //n 0 2 // a_i、b_i，下同 2 4 1 3	2

分析

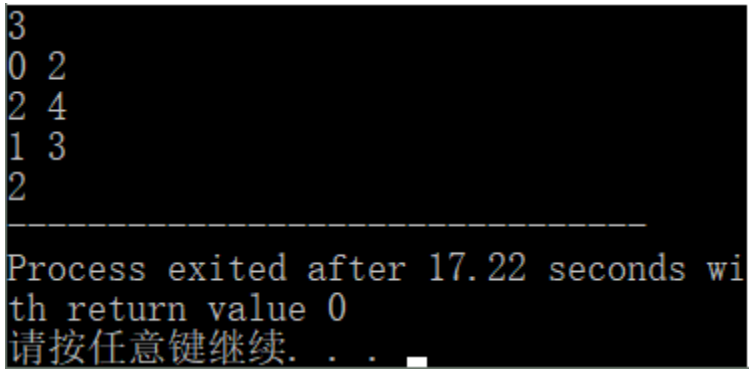
- 要想参加比赛尽量多，结束越早的比赛就越应该参加
- 因为只有这样，剩余的可支配时间才会更多，才可能容纳接下来更多的比赛

- 所以我们需要按结束时间排序，尽量选结束时间早的

分析

- 所以我们需要按结束时间排序，尽量选结束时间早的
- 但同时还需要避免冲突，因此也需要记录上场比赛的结束时间，与现在比赛的开始时间做比较
- 那我们开一个结构体数组存储每场比赛的开始和结束时间

参考代码



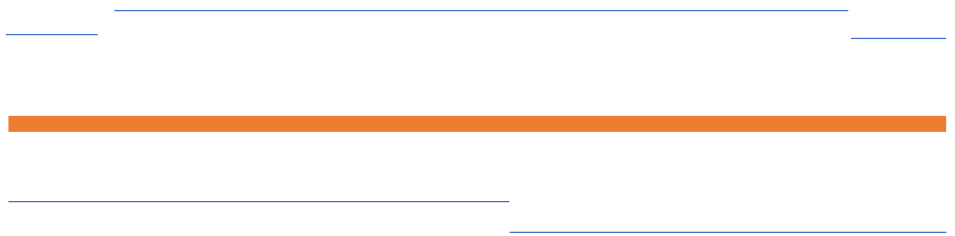

```
#include<bits/stdc++.h>
using namespace std;
int n, pos, ans; // pos记录上场比赛的结束时间
struct match
{
    int a, b;    //开始和结束时间
} m[1000010];

bool cmp(match x, match y)
{
    return x.b < y.b;  //结束时间早的靠前
}
```

```
int main()
{
    scanf("%d", &n);
    for (int i = 0; i < n; i ++)
        scanf("%d%d", &m[i].a, &m[i].b);
    sort(m, m + n, cmp);
    for (int i = 0; i < n; i ++)
        if (pos <= m[i].a)
            {pos = m[i].b; ans ++;} //更新 pos
    printf("%d", ans);
    return 0;
}
```

区间覆盖

- 在数轴上有n个区间，现在问你至少要选出多少个区间并给这些区间染色，才能使所有区间都被染上颜色

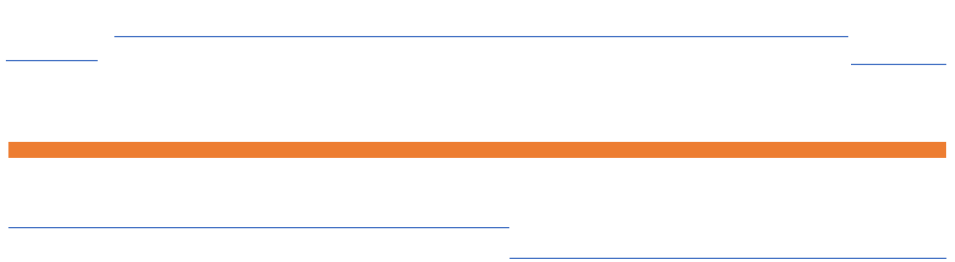
Sample input	Sample output
5 //n 1 2 // a_i、b_i，下同 2 8 8 9 1 5 5 9	2



分析

- 要想给尽量少的区间染色，就使得所有区间被染色
- 那应该优先给那些长度最大、覆盖面最广的区间染色？

- 但从样例看这个贪心是错的，原因和之前“背包”、“找零”两个问题一样，尽量长的区间有可能需要更多的零碎区间来填补空缺，从而无法达到全局最优

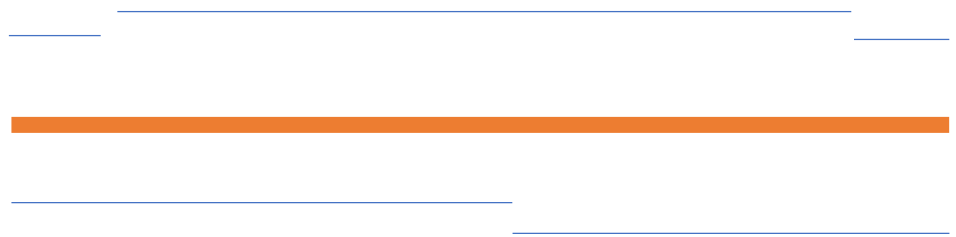


- 那么是否这道题无法应用贪心？

分析

- 我们可以借鉴刚才“比赛日程”的思路

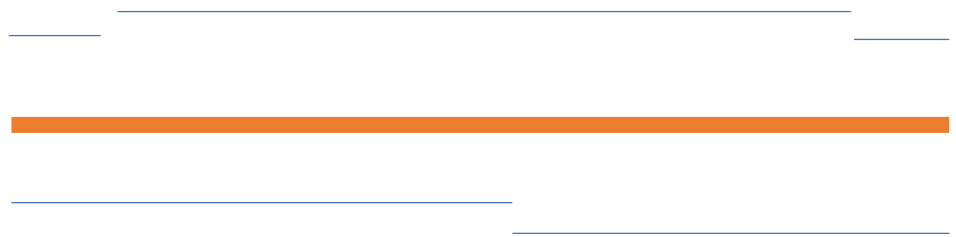
- 首先，最左端点的区间[1,b_1]是必选的，因为如果这样的区间不存在或者不被选，则整个问题无解
- 如果最左端点的区间有多个，势必选择右端点更靠右的



分析

- 我们可以借鉴刚才“比赛日程”的思路

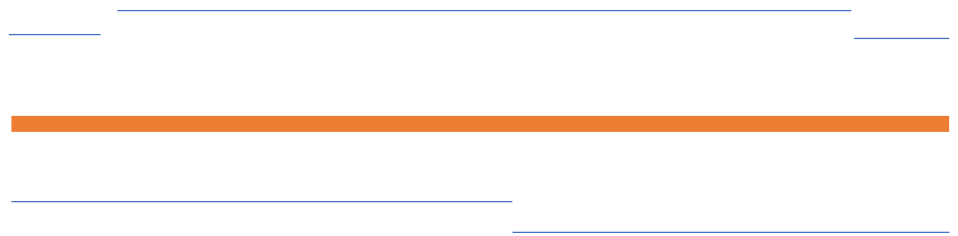
- 接下来我们需要选择左端点在区间[1,b_1]内的，否则会留出空隙
- 如果左端点在[1,b_1]的区间有多个，同样选择右端点更靠右的



分析

- 我们可以借鉴刚才“比赛日程”的思路

- 假设第二个区间是$[a_2,b_2]$，后续依此类推
- 直到右端点是最右端点为止



参考代码

- 略

一类相遇问题

- 和数学中类似，相遇问题也是计算机科学中常遇到的
- 并且它们的求解具有一些共性

蚂蚁相遇

- n只蚂蚁以1的速度在长为L的竹竿上爬行。当蚂蚁爬行到竹竿的尽头，就会掉落。而且竹竿太细，蚂蚁相遇时无法交错，只能各自回头爬行
- 现在已知每只蚂蚁距离竹竿左端点的距离，问所有蚂蚁都掉落所需的最短时间和最长时间
- n，L≤1,000,000

Sample input	Sample output
2 //数据组数 10 3 //L，n 2 6 7 214 7 //L，n 11 12 7 13 176 23 191	4 8 //min，max，下同 38 207

分析

- 完全不知道蚂蚁的朝向，而且蚂蚁之间也会因碰面而互相改变朝向
- 朝向一旦改变，就会发生来回折返
- 因此要暴力模拟所有的蚂蚁太困难

分析

- 第一问，最短时间
- 直觉上，要想时间最短，需要所有的蚂蚁都朝离自己最近的端点爬行，即[0,L/2]区间的蚂蚁向左端点爬行，[L/2,L]区间的蚂蚁向右端点爬行
- 这种情况下，蚂蚁不会相遇（为什么？）
- 蚂蚁没有相遇就没有折返，那么每只蚂蚁的爬行时间对它自己来说就是最短的
- 这样总和显然也是最短的

分析

- 第一问，最短时间
- 我们设pos为当前蚂蚁的位置，那么这只蚂蚁掉落的最短时间为：

```
time = min(pos, L - pos);
```

- 总的最短时间呢？
- 注意总的最短时间既不是最早掉落的蚂蚁决定，更不是所有蚂蚁耗时的和，而是由最后掉落的那只蚂蚁决定的
- 那显然还要维护一个最大值

```
time = min(pos, L - pos);
mint = max(mint, time);
```

分析

- 第二问，最长时间
- 直觉上，要想时间最长，需要所有的蚂蚁都尽可能地碰头，来回折返
- 这种情况下，还存在确定的最长时间吗？

- 我们现在开启上帝视角：你能区别出每只蚂蚁吗？显然不能，也不需要
- 因为蚂蚁的速度完全一样，所以我们可以这样考虑“相遇折返”这件事：

分析

- 第二问，最长时间
- 我们现在开启上帝视角：你能区别出每只蚂蚁吗？显然不能，也不需要
- 因为蚂蚁的速度完全一样，所以我们可以这样考虑“相遇折返”这件事：
- 本来应该是这样：
- 但我现在告诉你：我画这个效果的时候，蚂蚁一点都没改动就直接用了

分析

- 第二问，最长时间
- 我们现在开启上帝视角：你能区别出每只蚂蚁吗？显然不能，也不需要
- 因为蚂蚁的速度完全一样，所以我们可以这样考虑“相遇折返”这件事：
- 但我现在告诉你：我画这个效果的时候，蚂蚁一点都没改动就直接用了
- 那说明我们完全可以看成这样：

分析

- 第二问，最长时间
- 那说明我们完全可以看成这样：
- 这说明什么？说明我们完全可以不理会相遇折返这件事，它们即便真的相遇，也等效于互相穿过。就当每只蚂蚁前面没有任何阻碍

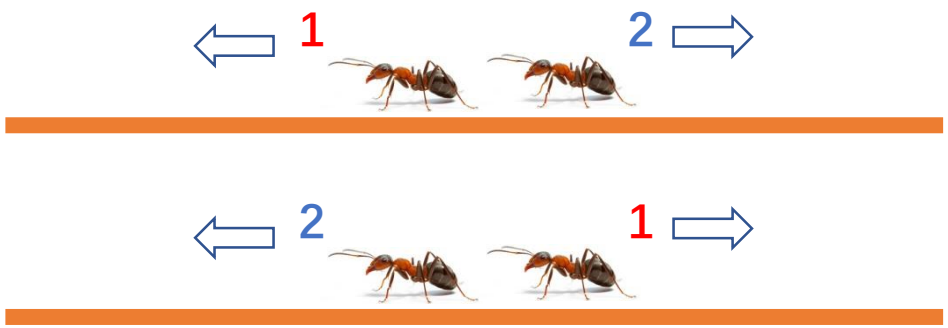
分析

- 第二问，最长时间
- 那么最长时间怎么求？
- 只要求每只蚂蚁掉落的最长时间（就是往离它最远的端点爬行）
- 总的最长时间？
- 还是由最后掉落的那只蚂蚁决定

分析

- 第二问，最长时间

```
time = max(pos, L - pos);
maxt = max(maxt, time);
```



参考代码

```
2
10 3
2 6 7
4 8
214 7
11 12 7 13 176 23 191
38 207

--------------------------------
```

```cpp
#include<bits/stdc++.h>
using namespace std;
int t, L, n, pos;
int main()
{
    scanf("%d", &t);
    while (t --)
    {
        scanf("%d%d", &L, &n);
        int mint = 0, maxt = 0;  //注意 mint的初值也为 0
        for(int i = 0; i < n; i ++)
        {
            scanf("%d", &pos);
            int time = min(pos, L - pos);
            mint = max(mint, time);  //最短时间由那只最后掉落的蚂蚁决定
            time = max(pos, L - pos);
            maxt = max(maxt, time);  //同上
        }
        printf("%d %d\n", mint, maxt);
    }
    return 0;
}
```

课外加练

- luogu 1106　　删数问题
- luogu 1249　　最大乘积（高精度）
- luogu 1115　　最大子段和
- luogu 5019　　铺设道路
- luogu 1969　　积木大赛
- luogu 1803　　线段覆盖
- luogu 1223　　排队接水
- luogu 1007　　独木桥