洛谷P1007

独木桥

题目背景

战争已经进入到紧要时间。你是运输小队长，正在率领运输部队向前线运送物资。运输任务像做题一样的无聊。你希望找些刺激，于是命令你的士兵们到前方的一座独木桥上欣赏风景，而你留在桥下欣赏士兵们。士兵们十分愤怒，因为这座独木桥十分狭窄，只能容纳 1 个人通过。假如有 2 个人相向而行在桥上相遇，那么他们 2 个人将无法绕过对方，只能有 1 个人回头下桥，让另一个人先通过。但是，可以有多个人同时呆在同一个位置。

题目描述

突然，你收到从指挥部发来的信息，敌军的轰炸机正朝着你所在的独木桥飞来！为了安全，你的部队必须撤下独木桥。独木桥的长度为 L，士兵们只能呆在坐标为整数的地方。所有士兵的速度都为 1，但一个士兵某一时刻来到了坐标为 0 或 $L+1$ 的位置，他就离开了独木桥。

每个士兵都有一个初始面对的方向，他们会以匀速朝着这个方向行走，中途不会自己改变方向。但是，如果两个士兵面对面相遇，他们无法彼此通过对方，于是就分别转身，继续行走。转身不需要任何的时间。

由于先前的愤怒，你已不能控制你的士兵。甚至，你连每个士兵初始面对的方向都不知道。因此，你想要知道你的部队最少需要多少时间就可能全部撤离独木桥。另外，总部也在安排阻拦敌人的进攻，因此你还需要知道你的部队最多需要多少时间才能全部撤离独木桥。

输入格式

第一行共一个整数 L，表示独木桥的长度。桥上的坐标为 $1,2,\cdots,L$。

第二行共一个整数 N，表示初始时留在桥上的士兵数目。

第三行共有 N 个整数，分别表示每个士兵的初始坐标。

输出格式

共一行，输出 2 个整数，分别表示部队撤离独木桥的最小时间和最大时间。2 个整数由一个空格符分开。

样例

样例输入

```
4
2
1 3
```

样例输出

```
2 4
```

提示

对于 100% 的数据，满足初始时，没有两个士兵同在一个坐标，$1 \leq L \leq 5 \times 10^3$，$0 \leq N \leq 5 \times 10^3$，且数据保证 $N \leq L$。

题解

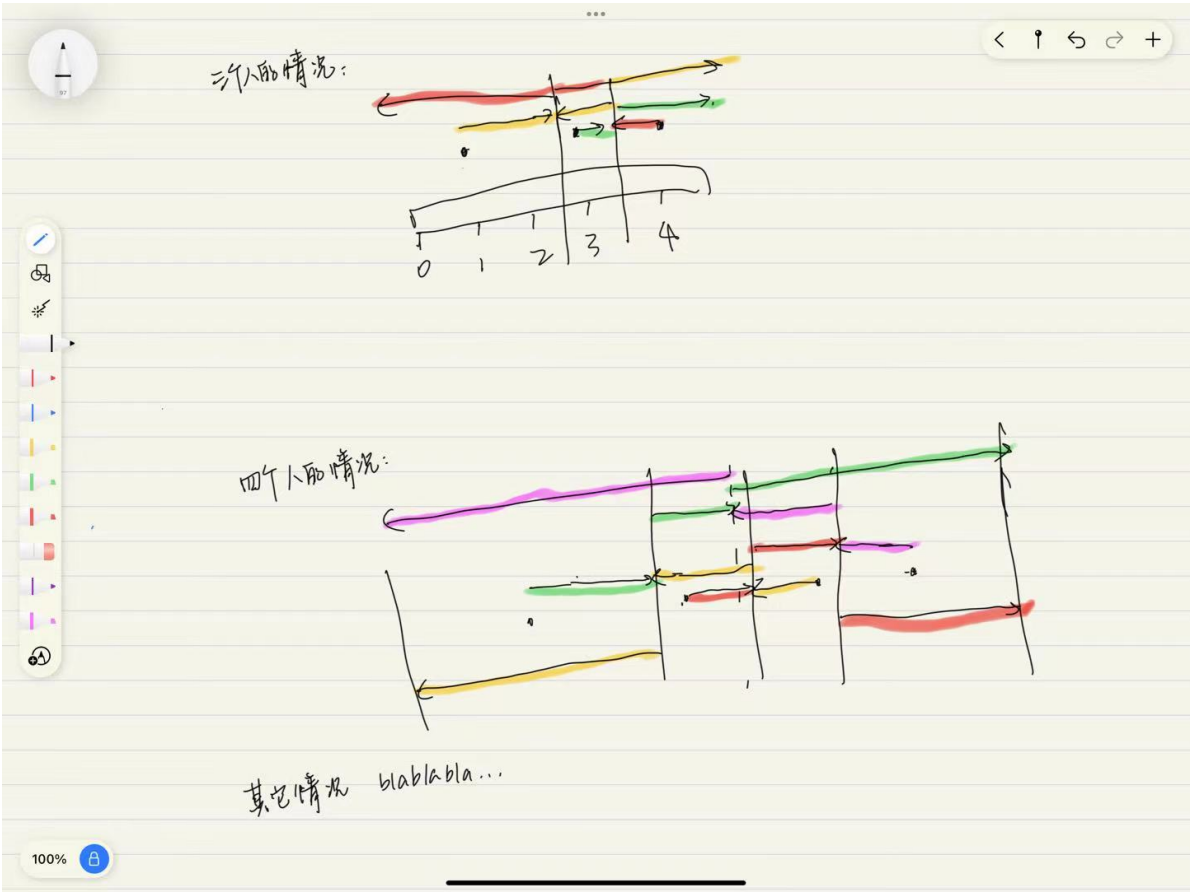
emmm这个题主要是想明白就好了，关键点在于当两个人相遇时，下一秒各自回头并且向前走，但当你仔细画一画图操作一下会发现，可以看成两个士兵互相穿过去的情况。

那么最短的时间就是离中点最近的人向外走

最长时间就是离中点最远的人向中点走，并越过中点走到出口。

其实最多和最少的分割点就是L+1的中点，在中点左边的士兵只需都往左走即可达到最短时间时间为士兵的坐标a，往右走就是所需最大时间L+1-a. 而在中点右边的情况与左边处理方法相反，原理是一样的。

接着如果总最小时间应该是左边和右边士兵中时间最小中最大的（离中点最近），最大时间是左边和右边士兵中最大的（离中点最远）。




```
#include <stdio.h>
int a[10000];

int main()
{
    int L,N;
```

```
    int i;
    int len1,len2;
    int ans1 = 0;
    int ans2 = 0;
    scanf("%d%d",&L,&N);
    for(i=0;i<N;i++){
        scanf("%d",&a[i]);
        len1 = a[i] < L+1 - a[i] ? a[i] : L+1 - a[i];
        if(ans1 < len1)
            ans1 = len1;
        len2 = a[i] > L+1 - a[i] ? a[i] : L+1 - a[i];
        if(ans2 < len2)
            ans2 = len2;
    }
    printf("%d %d", ans1,ans2);
    return 0;

}
```

好吧本来看到这个题有那个“贪心算法”的标签的，结果好像也没有用。

洛谷P1031

均分纸牌

题目描述

有N堆纸牌，编号分别为 $1,2,\ldots,N$。每堆上有若干张，但纸牌总数必为 N 的倍数。可以在任一堆上取若干张纸牌，然后移动。

移牌规则为：在编号为 1 堆上取的纸牌，只能移到编号为 2 的堆上；在编号为 N 的堆上取的纸牌，只能移到编号为 $N-1$ 的堆上；其他堆上取的纸牌，可以移到相邻左边或右边的堆上。

现在要求找出一种移动方法，用最少的移动次数使每堆上纸牌数都一样多。

例如 $N=4$ 时，4 堆纸牌数分别为 $9,8,17,6$。

移动 3 次可达到目的：

- 从第三堆取 4 张牌放到第四堆，此时每堆纸牌数分别为 $9,8,13,10$。
- 从第三堆取 3 张牌放到第二堆，此时每堆纸牌数分别为 $9,11,10,10$。
- 从第二堆取 1 张牌放到第一堆，此时每堆纸牌数分别为 $10,10,10,10$。

输入格式

第一行共一个整数 N，表示纸牌堆数。
第二行共 N 个整数 $A_1,A_2,\cdots,A_N$，表示每堆纸牌初始时的纸牌数。

输出格式

共一行，即所有堆均达到相等时的最少移动次数。

样例

样例输入

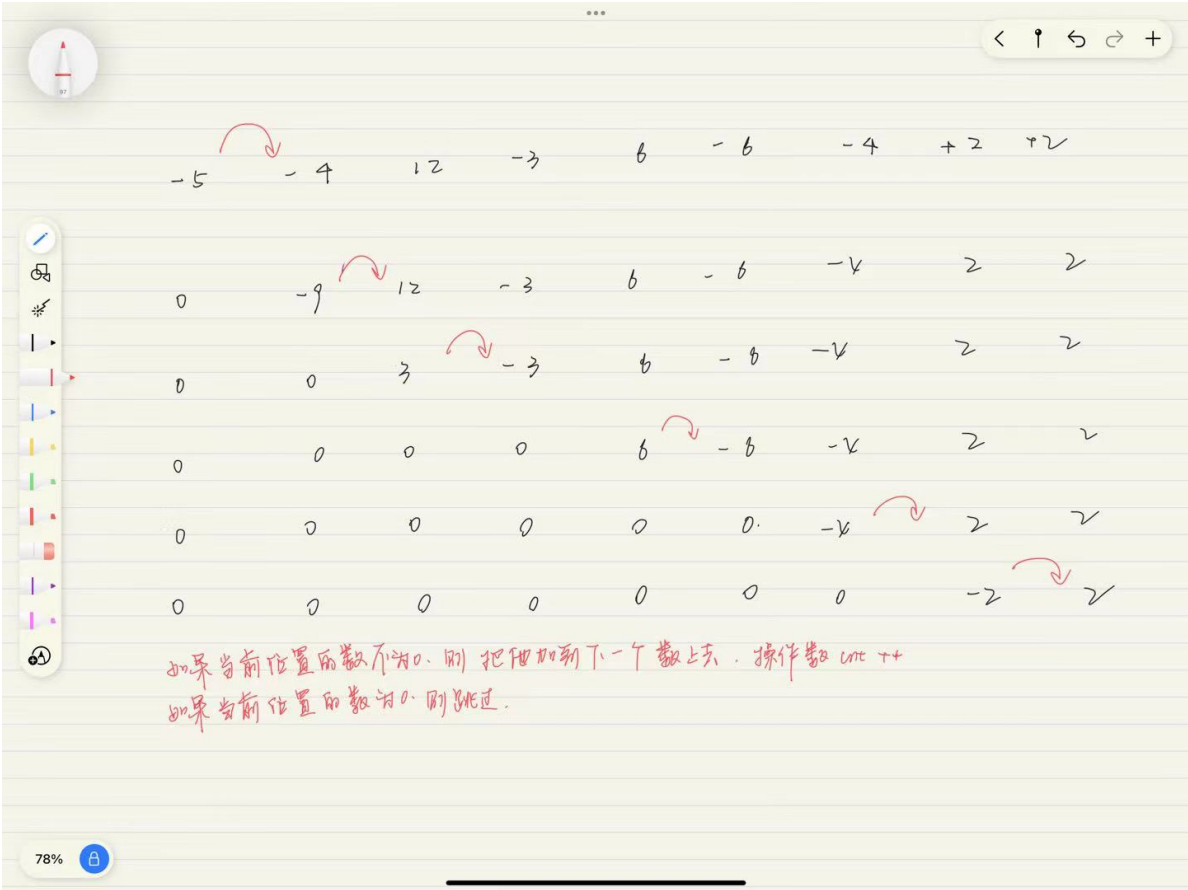
```
4
9 8 17 6
```

样例输出

```
3
```

提示

对于 100% 的数据，$1 \leq N \leq 100$，$1 \leq A_i \leq 10000$。




```c
#include <stdio.h>
int array[105];

int main()
{
    int N;
    int i;
    int sum=0;
    int cnt=0;//移动的次数
    scanf("%d", &N);
    for(i=0;i<N;i++){
        scanf("%d", &array[i]);
        sum+=array[i];//求和
    }
```

```
15
16      sum = sum / N;//求平均值
17      for(i=0;i<N;i++){
18          array[i]-=sum;
19      }                   //标准化一下，多于平均数的算正，少的算负，因为好算
20
21      for(i=0;i<N-1;i++){//具体解释看图
22          if(array[i]!=0){
23          array[i+1]+=array[i];
24          cnt++;
25          }
26      }
27
28      printf("%d",cnt);
29      return 0;
30  }
```

扩展：贪心算法

贪心算法的定义：
贪心算法是指在对问题求解时，总是做出在当前看来是最好的选择。也就是说，不从整体最优上加以考虑，只做出在某种意义上的局部最优解。贪心算法不是对所有问题都能得到整体最优解，关键是贪心策略的选择，选择的贪心策略必须具备无后效性，即某个状态以前的过程不会影响以后的状态，只与当前状态有关。
解题的一般步骤是：
1.建立数学模型来描述问题；
2.把求解的问题分成若干个子问题；
3.对每一子问题求解，得到子问题的局部最优解；
4.把子问题的局部最优解合成原来问题的一个解。
如果大家比较了解动态规划，就会发现它们之间的相似之处。最优解问题大部分都可以拆分成一个个的子问题，把解空间的遍历视作对子问题树的遍历，则以某种形式对树整个的遍历一遍就可以求出最优解，大部分情况下这是不可行的。贪心算法和动态规划本质上是对子问题树的一种修剪，两种算法要求问题都具有的一个性质就是子问题最优性(组成最优解的每一个子问题的解，对于这个子问题本身肯定也是最优的)。动态规划方法代表了这一类问题的一般解法，我们自底向上构造子问题的解，对每一个子树的根，求出下面每一个叶子的值，并且以其中的最优值作为自身的值，其它的值舍弃。而贪心算法是动态规划方法的一个特例，可以证明每一个子树的根的值不取决于下面叶子的值，而只取决于当前问题的状况。换句话说，不需要知道一个节点所有子树的情况，就可以求出这个节点的值。由于贪心算法的这个特性，它对解空间树的遍历不需要自底向上，而只需要自根开始，选择最优的路，一直走到底就可以了。

1.活动选择问题
这是《算法导论》上的例子，也是一个非常经典的问题。有n个需要在同一天使用同一个教室的活动a1,a2,…,an，教室同一时刻只能由一个活动使用。每个活动ai都有一个开始时间si和结束时间fi 。一旦被选择后，活动ai就占据半开时间区间[si,fi)。如果[si,fi]和[sj,fj]互不重叠，ai和aj两个活动就可以被安排在这一天。该问题就是要安排这些活动使得尽量多的活动能不冲突的举行。例如下图所示的活动集合S，其中各项活动按照结束时间单调递增排序。

i	1	2	3	4	5	6	7	8	9	10	11
S[i]	1	3	0	5	3	5	6	8	8	2	12
f[i]	4	5	6	7	8	9	10	11	12	13	14

考虑使用 贪心算法 的解法。为了方便，我们用不同颜色的线条代表每个活动，线条的长度就是活动所占据的时间段，蓝色的线条表示我们已经选择的活动；红色的线条表示我们没有选择的活动。
如果我们每次都选择开始时间最早的活动，不能得到最优解：

如果我们每次都选择持续时间最短的活动，不能得到最优解：

可以用数学归纳法证明，我们的贪心策略应该是每次选取结束时间最早的活动。直观上也很好理解，按这种方法选择相容活动为未安排活动留下尽可能多的时间。这也是把各项活动按照结束时间单调递增排序的原因。

2.钱币找零问题

这个问题在我们的日常生活中就更加普遍了。假设1元、2元、5元、10元、20元、50元、100元的纸币分别有c0, c1, c2, c3, c4, c5, c6张。现在要用这些钱来支付K元，至少要用多少张纸币？用贪心算法的思想，很显然，每一步尽可能用面值大的纸币即可。在日常生活中我们自然而然也是这么做的。在程序中已经事先将Value按照从小到大的顺序排好。

3.再论背包问题

在 从零开始学动态规划中我们已经谈过三种最基本的背包问题：零一背包，部分背包，完全背包。很容易证明，背包问题不能使用贪心算法。然而我们考虑这样一种背包问题：在选择物品i装入背包时，可以选择物品的一部分，而不一定要全部装入背包。这时便可以使用贪心算法求解了。计算每种物品的单位重量价值作为贪心选择的依据指标，选择单位重量价值最高的物品，将尽可能多的该物品装入背包，依此策略一直地进行下去，直到背包装满为止。在零一背包问题中贪心选择之所以不能得到最优解原因是贪心选择无法保证最终能将背包装满，部分闲置的背包空间使每公斤背包空间的价值降低了。在程序中已经事先将单位重量价值按照从大到小的顺序排好。

4.多机调度问题

n个作业组成的作业集，可由m台相同机器加工处理。要求给出一种作业调度方案，使所给的n个作业在尽可能短的时间内由m台机器加工处理完成。作业不能拆分成更小的子作业；每个作业均可在任何一台机器上加工处理。这个问题是NP完全问题，还没有有效的解法(求最优解)，但是可以用贪心选择策略设计出较好的近似算法(求次优解)。当n<=m时，只要将作业时间区间分配给作业即可；当n>m时，首先将n个作业从大到小排序，然后依此顺序将作业分配给空闲的处理机。也就是说从剩下的作业中，选择需要处理时间最长的，然后依次选择处理时间次长的，直到所有的作业全部处理完毕，或者机器不能再处理其他作业为止。如果我们每次是将需要处理时间最短的作业分配给空闲的机器，那么可能就会出现其它所有作业都处理完了只剩所需时间最长的作业在处理的情况，这样势必效率较低。在下面的代码中没有讨论n和m的大小关系，把这两种情况合二为一了。

5.小船过河问题

POJ1700是一道经典的贪心算法例题。题目大意是只有一艘船，能乘2人，船的运行速度为2人中较慢一人的速度，过去后还需一个人把船划回来，问把n个人运到对岸，最少需要多久。先将所有人过河所需的时间按照升序排序，我们考虑把单独过河所需要时间最多的两个旅行者送到对岸去，有两种方式：

1.最快的和次快的过河，然后最快的将船划回来；次慢的和最慢的过河，然后次快的将船划回来，所需时间为：t[0]+2*t[1]+t[n-1];

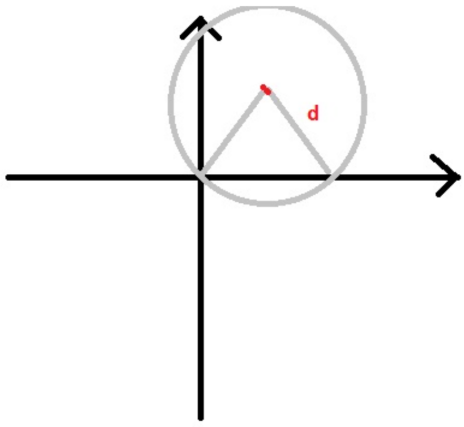
2.最快的和最慢的过河，然后最快的将船划回来，最快的和次慢的过河，然后最快的将船划回来，所需时间为：2*t[0]+t[n-2]+t[n-1]。

算一下就知道，除此之外的其它情况用的时间一定更多。每次都运送耗时最长的两人而不影响其它人，问题具有贪心子结构的性质。

AC代码：

6.区间覆盖问题

POJ1328是一道经典的贪心算法例题。题目大意是假设海岸线是一条无限延伸的直线。陆地在海岸线的一侧，而海洋在另一侧。每一个小的岛屿是海洋上的一个点。雷达坐落于海岸线上，只能覆盖d距离，所以如果小岛能够被覆盖到的话，它们之间的距离最多为d。题目要求计算出能够覆盖给出的所有岛屿的最少雷达数目。对于每个小岛，我们可以计算出一个雷达所在位置的区间。



问题转化为如何用尽可能少的点覆盖这些区间。先将所有区间按照左端点大小排序，初始时需要一个点。如果两个区间相交而不重合，我们什么都不需要做；如果一个区间完全包含于另外一个区间，我们需要更新区间的右端点；如果两个区间不相交，我们需要增加点并更新右端点。

7.销售比赛

在学校OJ上做的一道比较好的题，这里码一下。假设有偶数天，要求每天必须买一件物品或者卖一件物品，只能选择一种操作并且不能不选，开始手上没有这种物品。现在给你每天的物品价格表，要求计算最大收益。首先要明白，第一天必须买，最后一天必须卖，并且最后手上没有物品。那么除了第一天和最后一天之外我们每次取两天，小的买大的卖，并且把卖的价格放进一个最小堆。如果买的价格比堆顶还大，就交换。这样我们保证了卖的价格总是大于买的价格，一定能取得最大收益。

参考网址：(231条消息) 从零开始学贪心算法houjingyi233的博客-CSDN博客贪心算法