提高组高分试题 第三组题解

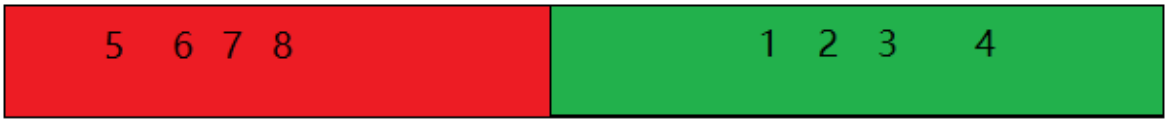
新婚快乐

题解

本文将提供一个常数略大的O(nlogn)的算法，有兴趣的同学可以尝试离线算法。

首先我们先考虑整个问题，对于每一辆车在整个行驶的过程中都可能会碰到红灯，在碰到红灯的等待期间会可能会和其他车辆回归同一起跑线。我们不妨先将问题简化，假设车在第i个红绿灯路口出发到终点的时间是多少。对于简化后的问题可以应用动态规划的思路，设dp[i]为车从第i个红绿灯路口刚好绿灯开始出发到终点的时间，设x为车启动后遇到的第二个红灯(x>i)，则dp[i] = dp[x] + dis(i,x) + wait(x)。dis 表示两个红绿灯之间的距离，wait表示在红灯处的等待时间。现在我们遇到的问题主要是如何确定当前点出发后遇到的下一个红灯。

为解决这个问题，我们将所有点到起点的距离模上(re+ge) 一个红绿灯周期 (以下皆用 mod代替) ，是其中的点落在[0,mod-1]这一区间。



由于数据范围的原因，整个过程需要离散化 其中距离取模排序后的第 1 2 3 4 在绿灯区 5 6 7 8在红灯区。也就意味着如果从头出发，车辆可以顺利通过第 1 2 3 4 个红绿灯。在红灯区我们们会被编号最小的红绿灯挡住，接下来的任务就是找到红灯区编号最小的红绿灯。

根据dp的顺序，我们可以将整个序列倒序枚举并用一个线段树维护红灯区编号最小值。

但是，我们又面临了一个新的问题，我们在线段树里存的是每个点到起点的距离，而不是两个点的相对距离。假设对于点i，此时线段树中存的是所有x > i 的点距离起点的值而不是和i的相对距离。为了解决这一问题，我们可以避免区间加减这一问题，设线段树内当前值为Ax 相对距离为 Ax-dis(i,0)。设红灯区的左右端点分别为L，R。

若 L < Ax-dis(i,0) < R 及点x在红灯区，我们可以采取一个小变形，及将访问区间改为 L+ dis(i,0) 和R+ dis(i,0) 即可。

由于模运算的性质，红绿灯区可能会变成两种情况。

1.红灯区在区间中间 2.绿灯区在区间中间

可以分类讨论。

接下来我们考虑查询，对于延误的时间t，我们可以采用上文的更换区间的方式解决，之后我们利用线段树查询出我们最先被卡住的红灯x，答案即为 dis(0,x)+wait(x)+dp[x], wait(x)为等待时间。

综合考虑我们只使用了一个线段树维护信息，综合上离散化总复杂度 O(nlogn)。

本人坚信本题存在时间复杂度更优或实现更为简单的数论算法，但受限于作者的水平只能利用数据结构来解决。本文中的红灯区绿灯区等概念可利用模运算推导，在此略去。

标准代码

c++ 11

```cpp
#include <cstdio>
#include <iostream>
#include <algorithm>
#include <cstring>
#include <cmath>
#define LL long long
#define ls rt<<1
#define rs rt<<1|1
using namespace std;

const int maxn = 2e5+100;

LL a[maxn],ge,re,t,mod,mp[maxn],mp1[maxn],h[maxn],lst,bs[maxn],ans;
int n,q,ps[maxn];
struct node
{
    LL val;
    int rk,id;
};
node b[maxn];

bool cmp(node a,node b)
{
    return a.val < b.val;
}
bool cmp1(node a,node b)
{
    return a.id < b.id;
}
int T[maxn*4];
void update(int rt,int l,int r,int x,int val)
{
    if(l==r)
    {
        T[rt] = val;
        return ;
    }
    int mid = (l+r)>>1;
    if(x<=mid) update(ls,l,mid,x,val);
    else update(rs,mid+1,r,x,val);
    T[rt] = min(T[ls],T[rs]);
}

int query(int rt,int l,int r,int L,int R)
{
    if(L<=l&&r<=R)
    {
        return T[rt];
    }
    int mid =(l+r)>>1,res=1e9+100;
    if(L<=mid) res = min(res,query(ls,l,mid,L,R));
    if(R>mid) res = min(res,query(rs,mid+1,r,L,R));
    return res;
}
int get_pos(LL L,LL R)
{
    LL posl = lower_bound(h+1,h+1+n,L) - h;
    LL posr = lower_bound(h+1,h+1+n,R) - h;
```

```
    LL posr1 = upper_bound(h+1,h+1+n,R) - h;
    if(posr==posr1) posr--;
    else posr = posr1-1;
    if(posr>n) posr = n;
    if(posl<1) posl = 1;
    //cout<<posl<<" PPP "<<posr<<endl;
    if(posl>posr) return 1e9+100;
    else return query(1,1,n,posl,posr);
}
int main()
{
    memset(T,0x3f,sizeof(T));
    scanf("%d%lld%lld",&n,&ge,&re);
    mod =ge+re;
    for(int i=1;i<=n;i++)
    {
        scanf("%lld",&a[i]);
        mp[i] = a[i];
        mp[i] += mp[i-1];
        b[i].val=mp[i]%mod;
        //out<<b[i].val<<" ";
        b[i].id = i;
        h[i]=b[i].val;
    }
    //cout<<endl;
    scanf("%lld",&lst);
    sort(h+1,h+1+n);
    sort(b+1,b+1+n,cmp);
    for(int i=1;i<=n;i++) b[i].rk=i;
    sort(b+1,b+1+n,cmp1);
    for(int i=n;i>=1;i--)
    {
        LL xi = mp[i];
        xi%=mod;
        int res=1e9+100;
        LL L = (ge+xi)%mod,R=(ge+re+xi-1)%mod;
        if(R<L)
        {
            int res1 = get_pos(L,mod);
            int res2 = get_pos(0,R);
            res = min(res1,res2);
        }
        else res = get_pos(L,R);
        //cout<<"pos:: "<<res<<endl;
        if(res<1e9)
        {
            LL cur = mp[res]-mp[i];
            bs[i] = bs[res] + cur +(mod-cur%mod);
        }
        else bs[i] = mp[n]-mp[i];
        //cout<<b[i].rk<<endl;
        update(1,1,n,b[i].rk,i);
    }
    //cout<<mp[n]<<endl;
    scanf("%d",&q);
    while(q--)
    {
        scanf("%lld",&t);
```

```
        LL pt = t%mod;
        LL L = (ge-pt+mod)%mod,R=(ge+re-pt-1)%mod;
        int res = 1e9+100;
        if(R<L)
        {
            int res1 = get_pos(L,mod);
            int res2 = get_pos(0,R);
            res = min(res1,res2);
        }
        else res = get_pos(L,R);
        if(res>1e9)
        {
            ans = t+lst+mp[n];
        }
        else
        {
            LL cur = t + mp[res];
            ans = cur + lst + bs[res] + (mod-cur%mod)%mod;
        }
        printf("%lld\n",ans);
    }
    return 0;
}
```

能量传输

题解

这道题是一个利用中位数性质的题，不过我们可以在讨论正解前先考虑一下其他算法。

不难发现，本题可以建立一个非常简单的费用流模型。我们不妨先建立一个超级源S，超级汇T，设每个人能量为$val[i]$。S和每个点连一条流量为$val[i]$,费用为0的边，每个点和T连一条流量为ave(平均值)，费用为0的边。在能交换的两个点之间连一条流量为(inf)费用为1的边。之后跑一边最小费用最大流即可。

期望得分 25

我们不妨再考虑一下这个模型，我们设vi为之前一点向当前点的流量，$v_i > 0$时为$i-1$给i的流量，$v_i < 0$时为i给$i-1$的流量。我们发现只要确定了第一个点的流量，其他点的流量均可用v_1表示。

$ave = val[i] + v_i - v_{i+1}$即 $v_2 = val[1] + v_1 - ave$ 。通过递推我们可得 $v_i = \sum_{i=0}^{i-1} val[i] + v_1 - (i-1) * ave$ 我们把v_1提出来，答案即为$\sum_{i=0}^{n} |v_i|$，我们可以去找去掉v1后的中位数，令v_1 加其等于0。时间复杂度$O(n)$ 或 $O(nlogn)$

1、中位数的性质

给定一个数列，中位数有这样的性质 ：所有数与中位数的绝对差之和最小

2、中位数性质的简单证明

首先，给定一个从小到大的数列，$x_1, x_2, \ldots\ldots, x_n$,设$x$是从$x_1$到$x_n$与其绝对差之和最小的数，则显然x位于$x_1$与$x_n$之间。那么，由于，$x_1$，$x_n$与它们之间的任意一点的距离之和都相等，且都等于$x_n - x_1$，因此接下来可以不考虑$x_1$与$x_n$，而考虑剩下的从$x_2$到$x_{n-1}$的数，同样显然有$x$必然位于$x_2$和$x_{n-1}$之间，依次类推，最后得出的结论是x就是该数列中间的那个数，或者是中间的那两个数之一，而

这个数就是中位数。

结论：数列的中位数就是该数列各个数与其绝对差之和最小的数。

PS:寻找中位数有$O(n)$的算法，本题并没有强制要求。有兴趣的同学可以考虑或学习一种以快速排序为基础的寻找序列第k大算法。

期望得分 100

标准代码

c++ 11

```
#include<cstdio>
#include<cstdlib>
#include<cstring>
#include<iostream>
#include<algorithm>
using namespace std;

int n,k;
long long ans=0,mea;
int a[1001010];
bool tf[1000010];
int nowa[1000010];
long long temp[1000010];

long long get_ans(int x){
    int t=0;
    while(!tf[x]){
        tf[x]=true;
        nowa[++t]=a[x];
        x+=k;if(x>n) x-=n;
    }
    temp[1]=0;
    for(int i=2;i<=t;i++) temp[i]=temp[i-1]+nowa[i-1]-mea;
    sort(temp+1,temp+1+t);
    long long s1=-temp[(t+1)/2],nowans=0;
    for(int i=1;i<=t;i++) nowans+=abs(temp[i]+s1);
    return nowans;
}

int main(){
    scanf("%d %d",&n,&k);k++;
    for(int i=1;i<=n;i++) scanf("%d",&a[i]),mea+=a[i];mea/=n;
    for(int i=1;i<=n;i++){
        if(tf[i]) continue;
        ans+=get_ans(i);
    }
    printf("%lld\n",ans);
    return 0;
}
```

矿物运输

题解

我们不妨先将问题简化，先考虑只有15%的链状数据。不难发现在链状数据情况下，整个问题就是一个裸的阶梯NIM游戏。

什么是阶梯NIM游戏？

有n个位置1...n，每个位置上有ai个石子。有两个人轮流操作。操作步骤是：挑选1...n中任一一个存在石子的位置ii，将至少1个石子移动至i-1位置（也就是最后所有石子都堆在在0这个位置）。谁不能操作谁输。求先手必胜还是必败。

问题等价于，求位置为奇数的ai的异或和，若异或和等于0，则先手必败，否则先手必胜。其实这个游戏恰好等价于：将每个奇数位置的数x看成一堆有x个石子的石子堆，然后玩NIM游戏。

证明略，可自行从网上获取。

接下来我们看一下这道题，从题面可以看出来这是多个有向图游戏的和，而多个有向图(G)游戏的和SG函数值等于它包含的各个子游戏SG函数值的异或和，即SG(G) = SG(G1) xor SG(G2) xor ...SG(Gm)。

所以我们不妨将深度为奇数的所有点矿石数求异或，结果大于零（根据NIM游戏规则）则输出win（先手必胜），否则输出lose。

总时复杂度，单次dfs O(n)。

标准代码

c++ 11

```
#include <cstdio>
#include <iostream>
#include <algorithm>
#include <cstring>
#include <vector>
#include <cmath>
using namespace std;

const int maxn = 3e5+100;

int T,n,fa;
int val[maxn];
int ans = 0;
vector<int> f[maxn];

void dfs(int u,int fa,int dep)
{
    if(dep&1) ans^=val[u];
    for(int i=0;i<f[u].size();i++)
    {
        int v = f[u][i];
        if(v==fa) continue;
        dfs(v,u,dep+1);
    }
}

int main()
{
    scanf("%d",&T);
    while(T--)
    {
```

```
        ans = 0;
        scanf("%d",&n);
        for(int i=0;i<=n;i++) f[i].clear();
        for(int i=1;i<n;i++)
        {
            scanf("%d",&fa);
            f[i].push_back(fa);
            f[fa].push_back(i);
        }
        for(int i=0;i<n;i++) scanf("%d",val+i);
        dfs(0,-1,0);
        if(ans) puts("win");
        else puts("lose");
    }
    return 0;
}
```