2050 Programming Competition Solution

开场白

判断一个数字 n 是否是若干个 2050 拼起来的

将 n 看成一个字符串，如果字符串长度不是 4 的倍数则显然不是

否则，依次判断每 4 位是否是 2050 即可

大多数 WA 的同学都是被诸如 205020 这种数据坑了，或者把题面看成了判断 n 是否是 2050 的倍数

时间复杂度：$O(\log n)$

Code:

```
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <ctime>
#include <cmath>
#include <iostream>
#include <algorithm>
#include <assert.h>

using namespace std;

char str[1000011];
int test, n;

int main() {
  scanf("%d", &test);
  assert(1<=test&&test<=10);
  for (; test--; ) {
    scanf("%s", str);
    n = strlen(str);
    assert(1<=n&&n<=100000);
    if (n % 4)  // 不是 4 的倍数则无解
      printf("No\n");
    else {
      bool ok = true;
      //依次判断每 4 位是否是 2050
      for (int i = 0; i < n && ok; i += 4)
        if (str[i] != '2' || str[i + 1] != '0' || str[i + 2] != '5' ||
str[i + 3] != '0')
          ok = false;
```

```
        if (ok)
          printf("Yes\n");
        else
          printf("No\n");
      }
    }
  }
```

时间间隔

一个比较直接的方法是：我们计算出这个时刻离 2050-01-01 00:00:00 一共 x 天，h 小时，m 分钟，s 秒，这个可以通过枚举年月，然后计算每个月有几天来计算

那么答案就是 $(60 \times 60 \times 24 \times x + 60 \times 60 \times h + 60 \times m + s) \ mod \ 100$

可以发现，因为 $60 \times 60 \times 24 \ mod \ 100 = 0$，所以 x 的值实际上是没用的

所以实际上，我们在哪一天是不影响我们的答案的

如果我们的输入是 YYYY-MM-DD HH:MM:SS，因为日期不影响答案，所以我们可以假设我们在 2049 年 12 月 31 日的 HH:MM:SS。

这样我们的答案就是 $(-(HH \times 60 \times 60 + MM \times 60 + SS)) \ mod \ 100$

注：因为 $60 \times 60 \ mod \ 100 = 0$，所以实际上小时对答案也没影响，那么也可以假设成 2049 年 12 月 31 日的 23:MM:SS

时间复杂度：$O(1)$

大多数错误提交的原因是直接输出了 $(MM \times 60 + SS) \ mod \ 100$，没有取负号

Code:

```
#include<stdio.h>
#include<cstring>
#include<algorithm>
#include<vector>
#include<map>
#include<assert.h>
#include<set>
#include<cmath>
#include<queue>
#include<cstdlib>
#include<iostream>
#include<bitset>
#define pii pair<int,int>
#define fi first
#define se second
#define pb push_back
#define rep(i,j,k) for(int i=(int)(j);i<=(int)(k);i++)
#define per(i,j,k) for(int i=(int)(j);i>=(int)(k);i--)
```

```
using namespace std;
typedef long long LL;
typedef double db;
void Main(){
    string t;cin>>t;
    int h,m,s;
    scanf("%d:%d:%d",&h,&m,&s);
    int ans=(s+m*60)%100;
    printf("%d\n",(100-ans)%100);
}
int main(){
    int t;scanf("%d",&t);
    while(t--)Main();
    return 0;
}
```

分宿舍

如果没有情侣房的话，实际上可以对男女分别考虑（双人间和三人间只能住同性），我们考虑计算一个数组：f_i 表示有 i 个同性要住到双人间或三人间内，最少需要多少钱

我们如何计算 f 呢，考虑一个简单的方法，我们可以枚举买了几个双人间和几个三人间，也就是：

$f_i = Min_{2x+3y\geq i}2a+3b$

于是我们得到了一个 $O(n^3)$ 计算 f 的方法

进一步地，我们令：

$g_i = Min_{2x+3y=i}2a+3b$

那么 $f_i = Min_{j\geq i}g_j$

首先计算 g 只要 $O(n^2)$，因为确定了 i 和 x 后，有 $y=(i-2x)/3$

之后再 $O(n)$ 或者 $O(n^2)$ 地通过 g 计算 f 即可

现在我们得到了在 $O(n^2)$ 时间内计算 f 的方法，当 $k=0$ 时，答案就是 f_n+f_m

那么在 $k>0$ 时，我们枚举有 i 对情侣住情侣房，那么相当于剩下了 $n+k-i$ 个男生，$m+k-i$ 个女生，所以这个方案的答案是 $ic+f_{n+k-i}+f_{m+k-i}$

所以答案就是 $Min_{i=0...k}ic+f_{n+k-i}+f_{m+k-i}$

时间复杂度：$O(n^2)$

当然，细心的同学也可以发现：g 的计算过程其实可以用一个简单的背包在 $O(n)$ 内计算出来，所以本题也可以在 $O(n)$ 的时间内解决

Code($O(n)$):

```
#include<stdio.h>
#include<cstring>
```

```cpp
#include<algorithm>
#include<vector>
#include<map>
#include<assert.h>
#include<set>
#include<cmath>
#include<queue>
#include<cstdlib>
#include<iostream>
#include<bitset>
#define pii pair<int,int>
#define fi first
#define se second
#define pb push_back
#define rep(i,j,k) for(int i=(int)(j);i<=(int)(k);i++)
#define per(i,j,k) for(int i=(int)(j);i>=(int)(k);i--)
using namespace std;
typedef long long LL;
typedef double db;
const int N=3225;
int n,m,k;LL a,b,c;
LL f[N];
void Main(){
    scanf("%d%d%d%lld%lld%lld",&n,&m,&k,&a,&b,&c);
    assert(0<=n&&n<=1000);
    assert(0<=m&&m<=1000);
    assert(0<=k&&k<=1000);
    assert(0<=a&&a<=1000000000);
    assert(0<=b&&b<=1000000000);
    assert(0<=c&&c<=1000000000);
    rep(i,0,n+k+m+20)f[i]=(1ll<<60);
    f[0]=0;
    rep(i,2,n+k+m+20)f[i]=min(f[i],f[i-2]+a);//背包，物品体积为2，费用为a
    rep(i,3,n+k+m+20)f[i]=min(f[i],f[i-3]+b);//背包，物品体积为3，费用为b
    per(i,n+k+m+19,0)f[i]=min(f[i],f[i+1]);  //考虑不住满的情况
    LL ans=1ll<<60;
    rep(i,0,k)ans=min(ans,i*1ll*c+f[n+k-i]+f[m+k-i]);   //枚举情侣房个数，计算
答案
    printf("%lld\n",ans);
}
int main(){
    int t;scanf("%d",&t);
    assert(1<=t&&t<=50);
    while(t--)Main();
    return 0;
}
```

Code ($O(n^2)$):

```
#include<stdio.h>
#include<cstring>
#include<algorithm>
#include<vector>
#include<map>
#include<assert.h>
#include<set>
#include<cmath>
#include<queue>
#include<cstdlib>
#include<iostream>
#include<bitset>
#define pii pair<int,int>
#define fi first
#define se second
#define pb push_back
#define rep(i,j,k) for(int i=(int)(j);i<=(int)(k);i++)
#define per(i,j,k) for(int i=(int)(j);i>=(int)(k);i--)
using namespace std;
typedef long long LL;
typedef double db;
const int N=3225;
int n,m,k;LL a,b,c;
LL f[N];
void Main(){
    scanf("%d%d%d%lld%lld%lld",&n,&m,&k,&a,&b,&c);
    assert(0<=n&&n<=1000);
    assert(0<=m&&m<=1000);
    assert(0<=k&&k<=1000);
    assert(0<=a&&a<=1000000000);
    assert(0<=b&&b<=1000000000);
    assert(0<=c&&c<=1000000000);
    rep(i,0,n+k+m+20)f[i]=(1ll<<60);
    rep(i,0,n+k+m+20)rep(x,0,i/2)if((i-2*x)%3==0){
        int y=(i-2*x)/3;
        //枚举 i,x, 计算 y
        f[i]=min(f[i],a*1ll*x+b*1ll*y);
    }
    //计算 f
    per(i,n+k+m+19,0)f[i]=min(f[i],f[i+1]);
    LL ans=1ll<<60;
    rep(i,0,k)ans=min(ans,i*1ll*c+f[n+k-i]+f[m+k-i]);
    printf("%lld\n",ans);
}
int main(){
    int t;scanf("%d",&t);
    assert(1<=t&&t<=50);
    while(t--)Main();
    return 0;
```

```
}
```

PASS

对于一个选手而言，要获得 PASS 必须要同时满足以下两个条件：

- 在自己的学校里是前 $\lfloor\frac{x}{k}\rfloor$ 优秀的
- 总排名在前 50% 内

所以对于一个学校，假设它的人数是 x，其中进入前 50% 的学生有 y 个，它获得的 PASS 个数就是 $min(\lfloor\frac{x}{k}\rfloor,y)$

对于每个学校，我们统计相应的 y 即可

时间复杂度：$O(n)$

```
#include<stdio.h>
#include<cstring>
#include<algorithm>
#include<vector>
#include<map>
#include<assert.h>
#include<set>
#include<cmath>
#include<queue>
#include<cstdlib>
#include<iostream>
#include<bitset>
#define pii pair<int,int>
#define fi first
#define se second
#define pb push_back
#define rep(i,j,k) for(int i=(int)(j);i<=(int)(k);i++)
#define per(i,j,k) for(int i=(int)(j);i>=(int)(k);i--)
using namespace std;
typedef long long LL;
typedef double db;
const int N=10005;
int n,m,k;
int a[N];
int cnt[N];
int d[N];
void Main(){
    scanf("%d%d%d",&n,&m,&k);
    assert(1<=n&&n<=10000);
    assert(1<=m&&m<=1000);
    assert(1<=k&&k<=20);
    rep(i,1,m)cnt[i]=d[i]=0;
```

```
    rep(i,1,n){
        scanf("%d",&a[i]);
        assert(1<=a[i]&&a[i]<=m);
        cnt[a[i]]++;            //  cnt[x]表示第 x 个学校的人数
        if(i*2<=n)d[a[i]]++;    //  d[x] 表示第 x 个学校进入前 50% 的学生的人数
    }
    int ans=0;
    rep(i,1,m){
        ans+=min(cnt[i]/k,d[i]);
    }
    printf("%d\n",ans);
}
int main(){
    int t;scanf("%d",&t);
    assert(1<=t&&t<=10);
    while(t--)Main();
    return 0;
}
```

球赛

因为乒乓球比赛的规则，获胜一方必须领先至少 2 分才行，所以比分在极端情况下是没有上限的

但是我们观察到，当 $x \geq 10$ 时，$x : x$ 这种比分实际上和 $9 : 9$ 相同

同样的，当 $x \geq 10$ 时，$x + 1 : x$ 这种比分实际上和 $10 : 9$ 相同，都是领先方下一次赢就结束，没赢就开始僵局

所以我们通过上述的转化，可以将比分的值限制在 $0...10$ 内

接下来考虑动态规划：

令 $f_{i,x,y}$ 表示，假设我们填完前 i 局中的所有问号，且当前的比分是 $x : y$ 的话，我们最多完成了几局了

考虑转移，实际上只要考虑第 $i + 1$ 局是谁赢，然后通过乒乓球的规则和上述规则得到新的比分 $x' : y'$，然后转移到 $f_{i+1,x',y'}$ 即可，转移时还要判断一局是否结束，这个等价于判断 $x' = y' = 0$

时间复杂度：$O(11^2 n)$

Code:

```
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <ctime>
#include <cmath>
#include <iostream>
#include <algorithm>
#include <assert.h>
```

```
using namespace std;

int test, n, f[10011][11][11];
char str[100011];

struct node {
  int x, y, z;
};

node calc(int i, int j) {
  // 计算 i:j 这个比分会转化成什么样，结果是 node{x,y,z}，表示比分变成 x:y，z 表示有
没有新完成一局
  node tmp;
  if (i == 11 || j == 11) {
    tmp.z = 1;
    tmp.x = tmp.y = 0;
  } else {
    tmp.z = 0;
    if (i == 10 && j == 10)
      tmp.x = tmp.y = 9;
    else
      tmp.x = i, tmp.y = j;
  }
  return tmp;
}

int main() {
  scanf("%d", &test);
  assert(1<=test&&test<=51);
  for (; test--; ) {
    scanf("%s", str);
    n = strlen(str);
    assert(1<=n&&n<=10000);
    memset(f, 128, sizeof(f));
    for(int i=0;i<n;i++){
      assert((str[i]=='A')||(str[i]=='B')||(str[i]=='?'));
    }
    f[0][0][0] = 0;
    for (int i = 1; i <= n; i++)
      for (int j = 0; j <= 10; j++)
        for (int k = 0; k <= 10; k++)
          if (f[i - 1][j][k] < (1 << 30)) {
            if (str[i - 1] == 'A' || str[i - 1] == '?') {
              node tmp = calc(j + 1, k);
              f[i][tmp.x][tmp.y] = max(f[i][tmp.x][tmp.y], f[i - 1][j][k] +
tmp.z);
            }
            if (str[i - 1] == 'B' || str[i - 1] == '?') {
```

```
                node tmp = calc(j, k + 1);
                f[i][tmp.x][tmp.y] = max(f[i][tmp.x][tmp.y], f[i - 1][j][k] +
tmp.z);
              }
            }
    int ans = 0;
    for (int j = 0; j <= 10; j++)
      for (int k = 0; k <= 10; k++)
        ans = max(ans, f[n][j][k]);
    printf("%d\n", ans);
  }
}
```

冰水挑战

因为我们是从 $i = 1...n$ 依次去选择挑不挑战的，所以一个简单的想法是：令 $f_{i,j,c}$ 表示是否有一种可能，使得我在考虑完前 i 个挑战后，我接受了 j 个挑战，最后体力剩下 c

转移只需要考虑第 $i + 1$ 个挑战是否接受，然后根据题目规则去计算出新的体力

但是由于 c 的范围非常大，所以这个动态规划的时间复杂度是我们无法接受的

但我们可以发现，在 i, j 确定的情况下，对于我们来说 c 肯定是越大越好

所以我们考虑一个新的动态规划，令 $f_{i,j}$ 表示在考虑完前 i 个挑战后，如果我们接受了 j 个的话，最后剩余体力的最大值

考虑转移：

第一种转移是不接受第 $i + 1$ 个挑战，也就是令 $f_{i+1,j} = max(f_{i+1,j}, f_{i,j} + c_{i+1})$

第二种转移时接受挑战，首先这要求 $min(f_{i,j}, b_{i+1}) > a_{i+1}$，转移是令 $f_{i+1,j} = max(f_{i+1,j}, min(f_{i,j}, b_{i+1}) - a_{i+1} + c_{i+1})$

时间复杂度：$O(n^2)$

Code:

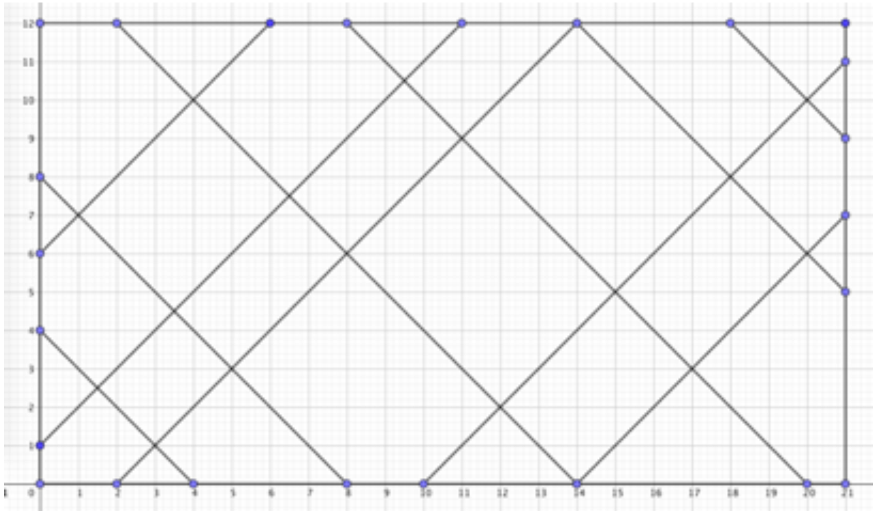
```
#include<stdio.h>
#include<cstring>
#include<algorithm>
#include<vector>
#include<map>
#include<assert.h>
#include<set>
#include<cmath>
#include<queue>
#include<cstdlib>
#include<iostream>
#include<bitset>
```

```
#define pii pair<int,int>
#define fi first
#define se second
#define pb push_back
#define rep(i,j,k) for(int i=(int)(j);i<=(int)(k);i++)
#define per(i,j,k) for(int i=(int)(j);i>=(int)(k);i--)
using namespace std;
typedef long long LL;
typedef double db;
const int N=1005;
LL f[N][N];
int n,a[N],b[N],c[N],m;
void Main(){
    scanf("%d%d",&n,&m);
    assert(1<=n&&n<=1000);
    assert(1<=m&&m<=1000000000);


    rep(i,1,n)scanf("%d%d%d",&a[i],&b[i],&c[i]);
    rep(i,1,n){
        assert(0<=a[i]&&a[i]<=1000000000);
        assert(0<=b[i]&&b[i]<=1000000000);
        assert(0<=c[i]&&c[i]<=1000000000);
    }
    rep(i,0,n)rep(j,0,n)f[i][j]=0;
    f[0][0]=m;
    rep(i,0,n-1)rep(j,0,i)if(f[i][j]){
        f[i+1][j]=max(f[i+1][j],f[i][j]+c[i+1]);
        if(min(f[i][j],b[i+1]*1ll)>a[i+1])
            f[i+1][j+1]=max(f[i+1][j+1],min(f[i][j],b[i+1]*1ll)-
a[i+1]+c[i+1]);
    }
    per(i,n,0)if(f[n][i]){
        printf("%d\n",i);
        return;
    }
}
int main(){
    int t;scanf("%d",&t);
    assert(1<=t&&t<=50);
    while(t--)Main();
    return 0;
}
```

大厦

样例：



实际上我们可以发现：两条正向 LED 灯 x, y 和两条反向 LED 灯 a, b 要构成一个合法矩形的充要条件是：

- x 和 a 的交点在矩形内
- x 和 b 的交点在矩形内
- y 和 a 的交点在矩形内
- y 和 b 的交点在矩形内

因为这四个交点实际上就是矩形的 4 个顶点

考虑计算两条线的交点：$x+y=c$ 和 $x-y=d$ 的交点：

解方程可得：$x=(c+d)/2$，$y=(c-d)/2$

那么一个简单的想法就是：枚举这四条直线是哪 4 条，然后判断四个交点是否在矩形内部

时间复杂度：$O(n^2m^2)$

优化1：

令 $S(x)$ 表示 $\{y|x\text{和}y\text{的交点在矩形内}\}$

那么我们枚举正向 LED 灯 x 和 y，则 a 和 b 只能在 $S(x)\cap S(y)$ 里选

假设 $w=|S(x)\cap S(y)|$，则合法的 (a,b) 的选择方案是 $w(w-1)/2$

我们 $O(nm)$ 预处理出所有 $S(x)$，然后枚举 x 和 y，用 $bitset$ 算出 $|S(x)\cap S(y)|$ 即可

时间复杂度：$O(n^2m/32)$

验题人写了一下这个算法，似乎是可以过的

优化2:

考虑直线 $x+y=c$，如果 $x-y=d$ 要与他有交，则要满足 $(c+d)/2\in[0,w]$，$(c-d)/2\in[0,h]$

通过这两个限制，我们可以解出 d 的范围，相当于必须有 $d\in[L(c),R(c)]$ 内

其中 $L(c)=max(-c,c-2h)$，$R(c)=min(2w-c,c)$，这个可以推推不等式得到

于是如果枚举了两条正向直线 c,d，那么 $S(x)\cap S(y)$ 的方程值都在 $[max(L(c),L(d)),min(R(c),R(d))]$ 内

所以就变成了一个区间数点问题，通过恰当的预处理和离散化可以在 $O(n^2)$ 内计算

优化3：

其实这题可以用线段树+扫描线做到 $O(n\log n)$，这不是重点所以就不细讲了，大家想了解的话可以看下面的代码或者去问杜瑜皓

Code(O(n^3/w))

```
#include<stdio.h>
#include<cstring>
#include<algorithm>
#include<vector>
#include<map>
#include<assert.h>
#include<set>
#include<cmath>
#include<queue>
#include<cstdlib>
#include<iostream>
#include<bitset>
#define pii pair<int,int>
#define fi first
#define se second
#define pb push_back
#define rep(i,j,k) for(int i=(int)(j);i<=(int)(k);i++)
#define per(i,j,k) for(int i=(int)(j);i>=(int)(k);i--)
using namespace std;
typedef long long LL;
typedef double db;
const int N=1005;
const int P=1000000007;
int w,h,n,m;
int a[N],b[N];
bitset<N> f[N];
bool ins(LL c,LL d){
    LL x=(c+d);
    LL y=(c-d);
    return 0<=x&&x<=2ll*w&&0<=y&&y<=2ll*h;
}
void Main(){
    scanf("%d%d%d%d",&w,&h,&n,&m);
    rep(i,1,n)scanf("%d",&a[i]);
    rep(i,1,m)scanf("%d",&b[i]);
    rep(i,1,n){
        f[i].reset();
```

```
        rep(j,1,m)if(ins(a[i],b[j]))f[i][j]=1;
    }
    int ans=0;
    rep(i,1,n)rep(j,i+1,n){
        int w=(f[i]&f[j]).count();
        ans=(ans+w*1ll*(w-1)/2)%P;
    }
    printf("%d\n",ans);
}
int main(){
    int t;scanf("%d",&t);
    while(t--)Main();
    return 0;
}
```

Code(O(nlogn))

```
#include<stdio.h>
#include<cstring>
#include<algorithm>
#include<vector>
#include<map>
#include<assert.h>
#include<set>
#include<cmath>
#include<queue>
#include<cstdlib>
#include<iostream>
#include<bitset>
using namespace std;
#define rep(i,a,n) for (int i=a;i<n;i++)
#define per(i,a,n) for (int i=n-1;i>=a;i--)
#define pb push_back
#define mp make_pair
#define all(x) (x).begin(),(x).end()
#define fi first
#define se second
#define SZ(x) ((int)(x).size())
typedef vector<int> VI;
typedef long long ll;
typedef pair<int,int> PII;
const ll mod=1000000007;
ll powmod(ll a,ll b) {ll res=1;a%=mod; assert(b>=0); for(;b;b>>=1)
{if(b&1)res=res*a%mod;a=a*a%mod;}return res;}
ll gcd(ll a,ll b) { return b?gcd(b,a%b):a;}
// head
```

```
const int N=101000;
const ll inf=1ll<<60;
ll w,h,c[N],d;
int n,m,_;

struct node {
    ll fg,s1,s2;
    int s0;
}nd[4*N];
void upd(int p) {
    nd[p].s1=nd[p+p].s1+nd[p+p+1].s1;
    nd[p].s2=nd[p+p].s2+nd[p+p+1].s2;
}
void setf(int p,ll v) {
    nd[p].fg+=v;
    nd[p].s2+=v*(v-1)/2*nd[p].s0+v*nd[p].s1;
    nd[p].s1+=v*nd[p].s0;
}
void build(int p,int l,int r) {
    nd[p].fg=0;
    nd[p].s0=r-l+1;
    nd[p].s1=0;
    nd[p].s2=0;
    if (l==r) {
    } else {
        int md=(l+r)>>1;
        build(p+p,l,md);
        build(p+p+1,md+1,r);
        upd(p);
    }
}
void push(int p) {
    if (nd[p].fg) {
        setf(p+p,nd[p].fg);
        setf(p+p+1,nd[p].fg);
        nd[p].fg=0;
    }
}
ll query(int p,int l,int r,int tl,int tr) {
    if (tl==l&&tr==r) return nd[p].s2;
    else {
        push(p);
        int md=(l+r)>>1;
        if (tr<=md) return query(p+p,l,md,tl,tr);
        else if (tl>md) return query(p+p+1,md+1,r,tl,tr);
        else return query(p+p,l,md,tl,md)+query(p+p+1,md+1,r,md+1,tr);
    }
}
void modify(int p,int l,int r,int tl,int tr,int v) {
```

```
    if (tl>tr) return;
    if (tl==l&&tr==r) return setf(p,v);
    else {
        push(p);
        int md=(l+r)>>1;
        if (tr<=md) modify(p+p,l,md,tl,tr,v);
        else if (tl>md) modify(p+p+1,md+1,r,tl,tr,v);
        else modify(p+p,l,md,tl,md,v),modify(p+p+1,md+1,r,md+1,tr,v);
        upd(p);
    }
}

void solve() {
    scanf("%lld%lld%d%d",&w,&h,&n,&m);
    assert(1<=w&&w<=1000000000);
    assert(1<=h&&h<=1000000000);
    assert(0<=n&&n<=1000);
    assert(0<=m&&m<=1000);
    vector<ll> v;
    vector<pair<ll,ll>> evt;
    rep(i,0,n) {
        scanf("%lld",c+i);
        assert(1<=c[i]&&c[i]<=w+h-1);
        v.pb(c[i]);
        evt.pb(mp(c[i],inf));
    }
    sort(all(v));
    for(int i=0;i+1<v.size();i++)assert(v[i]!=v[i+1]);
    vector<ll> g;
    rep(i,0,m) {
        scanf("%lld",&d);
        assert(1-h<=d&&d<=w-1);
        g.pb(d);
        ll p1=min(w+h,min(2*w-d,2*h+d));
        ll p2=max(0ll,max(-d,d));
        evt.pb(mp(p2,p1));
    }
    sort(all(g));
    for(int i=0;i+1<g.size();i++)assert(g[i]!=g[i+1]);

    sort(all(evt));
    build(1,0,n-1);
    ll ans=0;
    for (auto pr:evt) {
        ll x=pr.fi,y=pr.se;
        if (y==inf) {
            int id=lower_bound(all(v),x)-v.begin();
            if (id<n-1) ans=(ans+query(1,0,n-1,id+1,n-1))%mod;
        } else {
```

```
            int id=upper_bound(all(v),y)-v.begin()-1;
            if (id>=0) modify(1,0,n-1,0,id,1);
        }
    }
    printf("%lld\n",ans);
}

int main() {
    int _;
    scanf("%d",&_);
    assert(1<=_&&_<=10);
    for (;_;_--) {
        solve();
    }
}
```

骑行

计算出新的限速

实际上我们不能完全依靠读入的限速，因为我们的加速度也有限制，如果我们在某一段里疯狂飙车，可能会来不及减速，导致在下一段里超速

令 s'_i 为新的限速，表示在第 i 段的开头时，只要我的速度不超过 s'_i ，后面就一定能通过减速来满足后面段的限速

接下来考虑如何计算 s'_i

首先必然有 $s'_n = s_n$

对于 s'_i，假设 $s'_{i+1...n}$ 已经计算好了

那么其实我们只要保证我们能通过减速，在第 $i+1$ 段开头时速度能控制到 s'_{i+1} 就行了

那么考虑最极限的情况，从第 i 段开头时我们就一直以最大加速度去减速，则

$\int_0^{(s'_i-s'_{i+1})/a}(s'_i-ax)dx=w_i$

其实计算这个东西并不需要微积分这么高深的东西，我们可以用初中物理教的面积法：以 x 轴为时间，y 轴为速度，那么速度曲线下面的面积就是路程（接下来我们会大量用到该方法，如果不熟悉请复习一下初中物理）

所以通过计算梯形面积，有 $2w_i=(s'_i+s'_{i+1})(s'_i-s'_{i+1})/a_i$

解得 $s'_i=\sqrt{(s'_{i+1})^2+2w_ia_i}$

当然，解出来的 s'_i 需要对 s_i 取个 min

贪心地去跑

接下来我们考虑最快的骑法是什么样的，首先加速度肯定每时每刻要么是 0，要么是 a_i，要么是 $-a_i$，0 是因为已经达到了限速上限，其他情况下如果绝对值不为 0 的话显然可以进行调整使得绝对值变成 a_i

例如：如果 $a_i = 2$，最优解中有一段我的加速度是 1，那么我可以令加速度为 2，让他在一半的时间内达成一样的效果，跑的也更快了

考虑我们现在要跑第 i 段路，为了方便我们令 $s_i = s'_i$，假设我们第 i 段路开头时的速度是 $speed$，显然有 $speed \le s_i$

接下来要进行一些分类讨论

第一种情况，可以一直疯狂加速

如果 $s_i \le s_{i+1}$，我们判断一下一直加速的话会不会超过限速

假设我们一直加速到 s_i，可以通过面积法计算梯形面积来计算我们跑了多少：

$\frac{1}{2}(speed + s_i)(s_i - speed)/a_i$

假设这个值是 L

如果 $L \le w_i$，说明我可以疯狂加速到 s_i

之后因为限速问题，我们要保持 s_i 的速度跑完剩下的路程，且因为 $s_i \le s_{i+1}$，到第 $i+1$ 段时我们并没有超速

这样所用的时间是 $(s_i - speed)/a_i + (w_i - L)/s_i$

如果 $L > w_i$，说明我们还没加速到 s_i 就已经跑完了第 i 段

假设跑完时速度是 p，那么根据面积法可以列出方程：

$2w_i = (p + speed)(p - speed)/a_i$

解得 $p = \sqrt{2w_i a_i + speed^2}$

这样所用的时间是 $(p - speed)/a_i$

第二种情况，先跑再减速

这种情况是由于 $s_i > s_{i+1}$

首先，就算 $s_i > s_{i+1}$，我们可以计算出上一个情况中的 p，如果 $p \le s_{i+1}$，那么我们的策略仍然是全程加速度拉满，飙车

否则我们一定是先一直加速，然后保持速度跑一段距离，然后再减速

(1) 需要保持速度跑一段距离

思考下什么情况下需要保持速度跑一段距离，那肯定是因为加速到了 s_i 了

那么通过面积法，我们假设保持速度跑了 t 的时间，那么

$w_i = ts_i + (speed + s_i)((s_i - speed)/a_i)/2 + (s_i + s_{i+1})(s_i - s_{i+1})/(2a_i)$

（这条曲线下的图形并不是一个梯形，但是稍微拆一下也算得出来

可以通过方程把 t 直接解出来

如果 $t<0$，则说明并不属于这个情况，需要转情况 (2)，否则用时就是：$t+(s_i-speed)/a_i+(s_i-s_{i+1})/a_i$

(2)中间不保持速度跑

那么相当于我是加速到了 p，然后直接开始减速到 s_{i+1}

我们继续列方程

$w_i=(speed+p)(p-speed)/(2a_i)+(p+s_{i+1})(p-s_{i+1})/(2_a i)$

可以直接解出 p

然后用时就是 $(p-speed)/a_i+(p-s_{i+1})/a_i$

于是我们讨论完所有情况了！

时间复杂度：$O(n)$

Code:

```
#include<stdio.h>
#include<cstring>
#include<algorithm>
#include<vector>
#include<map>
#include<assert.h>
#include<set>
#include<cmath>
#include<queue>
#include<cstdlib>
#include<iostream>
#include<bitset>
#define pii pair<int,int>
#define fi first
#define se second
#define pb push_back
#define rep(i,j,k) for(int i=(int)(j);i<=(int)(k);i++)
#define per(i,j,k) for(int i=(int)(j);i>=(int)(k);i--)
using namespace std;
typedef long long LL;
typedef double db;
const int N=1100;
db w[N],s[N],a[N];
const db eps=1e-7;
int n;
void Main(){
    scanf("%d",&n);
    assert(1<=n&&n<=1000);
    rep(i,1,n){
```

```
        scanf("%lf%lf%lf",&w[i],&s[i],&a[i]);
        assert(1<=w[i]&&w[i]<=1000);
        assert(1<=s[i]&&s[i]<=1000);
        assert(1<=a[i]&&a[i]<=1000);
    }
    per(i,n-1,1){
        s[i]=min(s[i],sqrt(2*a[i]*w[i]+s[i+1]*s[i+1]));
    }
    db spd=0;
    db nt=0;
    s[n+1]=1e9;
    rep(i,1,n){
        if(!(spd<=s[i])){
            printf("%.5lf %.5lf\n",spd,s[i]);
            assert(0);
        }
        //spd<=s[i]<=s[i+1]
        if(s[i]<=s[i+1]){
            //情况1, ww 就是题解中的 L
            db ww=(s[i]*s[i]-spd*spd)/(2*a[i]);
            if(ww>=w[i]){
                db ns=sqrt(2*a[i]*w[i]+spd*spd);
                nt+=(ns-spd)/a[i];
                spd=ns;
            }
            else{
                nt+=(s[i]-spd)/a[i]+((w[i]-ww)/s[i]);
                spd=s[i];
            }
        }
        else{
            //s[i]>s[i+1]
            db ns=sqrt(2*a[i]*w[i]+spd*spd);
            //算出 p, 看看全力加速后会不会超过s[i+1]
            if(ns<=s[i+1]){
                nt+=(ns-spd)/a[i];
                spd=ns;
            }
            else{
                //情况2
                db ww=(spd+s[i])*((s[i]-spd)/a[i])/2;
                ww+=(s[i]+s[i+1])*((s[i]-s[i+1])/a[i])/2;
                if(ww<=w[i]){
                    nt+=(s[i]-spd)/a[i]+(s[i]-s[i+1])/a[i]+(w[i]-ww)/s[i];
                    spd=s[i+1];
                }
                else{
                    db hs=sqrt((2*a[i]*w[i]+spd*spd+s[i+1]*s[i+1])/2);
                    assert(hs+eps>=spd);
```

```
                        assert(hs<=s[i]+eps);
                        assert(hs+eps>=s[i+1]);
                        nt+=(hs-spd)/a[i]+(hs-s[i+1])/a[i];
                        spd=s[i+1];
                    }
                }
            }
        }
    printf("%.5lf\n",nt);
}
int main(){
    int t;scanf("%d",&t);
    assert(1<=t&&t<=100);
    while(t--)Main();
    return 0;
}
```

跨洋飞行

首先我们令 low_x 表示离第 x 个机场最近的机场到它的距离

那么相当于如果我们要在机场 x 降落的话，我们至少要有 low_x 的油

首先我们证明一个结论

结论：

```
存在一种最优方案，使得对于每个经过的机场 x，我们在机场 x 降落时，都恰好剩下 low(x) 的油
```

这个结论的思想很简单：就是在有需要的时候才加恰当数量的油，进行极限飞行

证明：

考虑归纳法，对于起点，我们的结论显然正确（没在起点降落）

假设我们在 x 降落时剩下的油有 $low_x + p$

那么假设前一个降落的机场是 y，离开 y 时的油量就是 $low_x + p + dist(x, y)$

我们可以发现，令离开 y 时的油量为 $low_x + dist(x, y)$，然后在 x 在加 p 的油，不会影响结果，且满足了我们的结论

做法:

那么现在我们知道，如果我们降落到了 x,那我们这时的油量是 low_x（为了方便，特殊地令 $low_1 = 0$）

令 f_i 表示我们降落到第 i 个机场最少需要多少时间

如果接下来要前往 j 的话，那油就要加到 $low_j + dist(x, j)$，也就是我们需要加 $low_j + dist(x, j) - low_x$ 的油

计算这需要多少时间，假设需要 w，那么连一条边 $< i, j, w >$

之后求出 1 到 n 的最短路即可，注意要考虑降落和起飞的时间

时间复杂度：$O(n^2)$

Code:

```
#include<stdio.h>
#include<cstring>
#include<algorithm>
#include<vector>
#include<map>
#include<assert.h>
#include<set>
#include<cmath>
#include<queue>
#include<cstdlib>
#include<iostream>
#include<bitset>
#define pii pair<int,int>
#define fi first
#define se second
#define pb push_back
#define rep(i,j,k) for(int i=(int)(j);i<=(int)(k);i++)
#define per(i,j,k) for(int i=(int)(j);i>=(int)(k);i--)
using namespace std;
typedef long long LL;
typedef double db;
const int N=1005;
const db eps=1e-7;
int n,a,b,c,d,u;
db dis[N][N];
int x[N],y[N];
db f[N];
bool vis[N];
db low[N];
void Main(){
    scanf("%d%d%d%d%d%d",&n,&a,&b,&c,&d,&u);
    assert(2<=n&&n<=1000);
    assert(0<=a&&a<=100000);
    assert(0<=b&&b<=100000);
    assert(0<=c&&c<=100000);
    assert(0<=d&&d<=100000);
    assert(0<=u&&u<=100000);
    rep(i,1,n){
        scanf("%d%d",&x[i],&y[i]);
```

```
        assert(0<=x[i]&&x[i]<=100000);
        assert(0<=y[i]&&y[i]<=100000);
    }
    rep(i,1,n)rep(j,1,n)dis[i][j]=sqrt((x[i]-x[j])*1.*(x[i]-x[j])+(y[i]-y[j])*1.*(y[i]-y[j]));
    rep(i,1,n){
        low[i]=1e20;
        rep(j,1,n)if(i^j)low[i]=min(low[i],dis[i][j]);
    }
    low[1]=0;
    rep(i,1,n)f[i]=1e20;
    f[1]=low[1]*c;
    if(low[1]>u+eps)assert(0);
    memset(vis,0,sizeof vis);
    int num=0;
    rep(rd,1,n){
        int x=0;
        rep(i,1,n)if(!vis[i])if((x==0)||(f[i]<f[x]))x=i;

        rep(j,1,n)if(dis[x][j]+low[j]<=u){
            if(j==n&&f[j]>f[x]+a+b+dis[x][j]*d+(dis[x][j]-low[x]+low[j])*c)num=rd;
            f[j]=min(f[j],f[x]+a+b+dis[x][j]*d+(dis[x][j]-low[x]+low[j])*c);
        }
        vis[x]=1;
    }
    assert(f[n]<(1e19));
    printf("%.7lf\n",f[n]);
}
int main(){
    int t;scanf("%d",&t);
    assert(1<=t&&t<=50);
    while(t--)Main();
    return 0;
}
```

探索行星

这是一道比较复杂度的题

没有误差的情况

我们考虑没有误差的情况，也就是说 z 全部为 0，我们的所有指令都是准确的

假设我们的坐标是 (x,y,z)，那么我们所站立的地点的法向量也是 $f=(x,y,z)$，这个法向量可以认为是我们头顶所指的方向

然后我们面朝的方向是 f 和球的切平面上的一个向量，我们设他是 $f+w$

然后我们左手所指向的方向也是 f 和球的切平面上的一个向量，我们设他是 $f+l$

实际上，f,w,l 构成了一个坐标系

当我们逆时针旋转 x 度时，f 是不变的，而 w,l 都以 f 为轴旋转了 x 度

三维旋转可以表示成矩阵乘法的形式，我们假设我们的状态是 $T=[f_x,f_y,f_z,l_x,l_y,l_z,w_x,w_y,w_z,1]$

则令 w,l 都以 f 为轴旋转 x 度，状态 T 会变成 $TRot_f(x)$

其中 $Rot_f(x)$ 是一个矩阵，这个可以根据一些数学知识去构造

当我们前进 x 时，首先 l 不变，也就是我们左手所指的方向是不变的（这个大家可以脑补下），而 w,f 相当于绕着 l 旋转了 x 度，这个变换也可以写成一个矩阵 $Rot_l(x)$，所以状态会变成 $TRot_l(x)$

所以我们可以把操作写成 $(type_i,x_i)$，则最后的状态是 $T\prod_{i=1}^{n}Rot_{type_i}(x_i)$

于是我们通过一顿矩阵乘法，计算出了我们应该到达的坐标 (s_x,s_y,s_z)

距离的计算

考虑如何计算 (x,y,z) 和 (s_x,s_y,s_z) 的距离的平方

$$(x-s_x)^2+(y-s_y)^2+(z-s_z)^2=x^2+y^2+z^2+s_x^2+s_y^2+s_z^2-2(xs_x+ys_y+zs_z)$$

注意我们是在一个半径为 1 的球上，所以 $x^2+y^2+z^2=s_x^2+s_y^2+s_z^2=1$

所以距离就是 2 减去两倍的点积，当我们知道 s_x,s_y,s_z 时，这是一个关于 x,y,z 的线性函数

期望的计算

实际上我们可以发现，我们上面所有的过程都是线性的

所以对于每一步，如果旋转角度在 $[y-z,y+z]$ 内随机选的话，我们可以计算出矩阵的期望：

$$E_{x\in[y-z,y+z]}[Rot_{type_i}(x)]=\frac{1}{2z}\int_{y-z}^{y+z}Rot_{type_i}(x)dx$$

对矩阵的积分实际上就是对矩阵里每个元素积分，显然本题中的矩阵里的元素都是简单的三角函数，利用高数的知识积分一下就行了

所以期望的坐标就是 $T\prod_{i=1}^{n}E[Rot_{type_i}(x_i)]$

时间复杂度：$O(n)$

Code(由于代码是另一个出题人的，所以可能与题解不符):

```
#include <bits/stdc++.h>
using namespace std;
#define rep(i,a,n) for (int i=a;i<n;i++)
#define per(i,a,n) for (int i=n-1;i>=a;i--)
```

```
#define pb push_back
#define mp make_pair
#define all(x) (x).begin(),(x).end()
#define fi first
#define se second
#define SZ(x) ((int)(x).size())
typedef vector<int> VI;
typedef long long ll;
typedef pair<int,int> PII;
const ll mod=1000000007;
ll powmod(ll a,ll b) {ll res=1;a%=mod; assert(b>=0); for(;b;b>>=1)
{if(b&1)res=res*a%mod;a=a*a%mod;}return res;}
ll gcd(ll a,ll b) { return b?gcd(b,a%b):a;}
// head

typedef double db;

struct matrix {
    db a[3][3];
};

matrix unit() {
    matrix c;
    rep(i,0,3) rep(j,0,3) c.a[i][j]=(i==j)?1:0;
    return c;
}
matrix operator * (const matrix &a,const matrix &b) {
    matrix c;
    rep(i,0,3) rep(j,0,3) {
        c.a[i][j]=0;
        rep(k,0,3) c.a[i][j]+=a.a[i][k]*b.a[k][j];
    }
    return c;
}

int n,p1[110],_;
db p2[110],p3[110];

void solve() {
    scanf("%d",&n);
    assert(1<=n&&n<=100);
    matrix pos1=unit(),pos2=unit();
    rep(i,0,n) {
        scanf("%d%lf",p1+i,p2+i);
        assert(p1[i]==1||p1[i]==2);
        assert(0<=p2[i]&&p2[i]<=1000);
        if (p1[i]==1) {
            db th=p2[i];
            matrix trans=unit();
```

```
            trans.a[0][0]=cosl(th);
            trans.a[0][2]=-sinl(th);
            trans.a[2][0]=sinl(th);
            trans.a[2][2]=cosl(th);
            pos1=trans*pos1;
            pos2=trans*pos2;
        } else {
            scanf("%lf",p3+i);
            assert(0<=p3[i]&&p3[i]<=1000);
            db th=p2[i];
            matrix trans=unit();
            trans.a[0][0]=cosl(th);
            trans.a[0][1]=sinl(th);
            trans.a[1][0]=-sinl(th);
            trans.a[1][1]=cosl(th);
            pos1=trans*pos1;
            if (fabs(p3[i])<=1e-5) {
                pos2=trans*pos2;
            } else {
                db e=p3[i];
                trans.a[0][0]=(sinl(th+e)-sinl(th-e))/2/e;
                trans.a[0][1]=(cosl(th-e)-cosl(th+e))/2/e;
                trans.a[1][0]=(cosl(th+e)-cosl(th-e))/2/e;
                trans.a[1][1]=(sinl(th+e)-sinl(th-e))/2/e;
                pos2=trans*pos2;
            }
        }
        db ans=2;
        rep(i,0,3) ans-=2*pos1.a[2][i]*pos2.a[2][i];
        printf("%.8lf\n",max(ans,(db)0.0));
    }
}
int main() {
    scanf("%d",&_);
    assert(1<=_&&_<=100);
    for (;_;_--) {
        solve();
    }
}
```