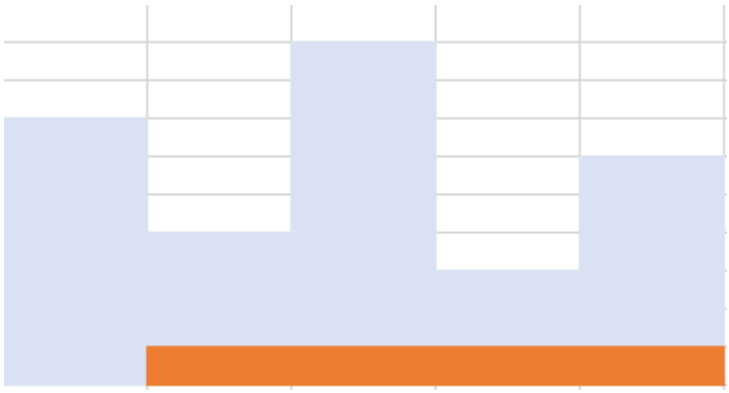
第八届ACM-ICPC暨CCCC-GPLT校内选拔赛决赛解题报告

Problem A: Arrangement For Snacks-crazyX

题目可以变换为一下模型：

- n 个数排成一排
- 有一种操作，选择长度为 k 的一段，这一段的数减 1, 令其为操作 i，使得 $i, i+1, \cdots, i+k-1$ 位置上的数减 1
- 这 n 个数不能变为负数

做法就是，从左到右，依次选 k 个数，求出这 k 个数的最小值，然后用这 k 个数减去最小值，表示操作了 k 次。继续往选后面的数，进行相同的操作。



为了方便大家理解，继续转换模型，上述表示$a = [7, 4, 10, 3, 6]$ 这样一个数列，橘黄色的块表示进行的一次操作。这样就转换成最多能够放多少个这样的块，且块不会超过淡蓝色的范围。对于一个能够放下的块，如果它能够左移就左移，从而可以空出空间来，给其他的块。

根据以上的分析，我们可以得到 $O(n * n)$ 的复杂度，即从左往右，考虑第i 个位置最多的操作，对位置i 操作完后再考虑 $i + 1$的操作数。

```cpp
/*
* Filename:    std.cpp
* Created:     Wednesday, December 06, 2017 07:03:10 PM
* Author:      crazyX
* More:
*
*/
#include <bits/stdc++.h>

#define mp make_pair
#define pb push_back
#define fi first
#define se second
#define SZ(x) ((int) (x).size())
#define all(x) (x).begin(), (x).end()
#define sqr(x) ((x) * (x))
#define clr(a,b) (memset(a,b,sizeof(a)))
#define y0 y3487465
#define y1 y8687969
#define fastio std::ios::sync_with_stdio(false)

using namespace std;

#if __cplusplus <= 199711L
    #warning The program needs at least a C++11 compliant compiler
#else
    template<typename T, typename... Args>
    T min(T value, Args... args) { return min(value, min(args...)); }

    template<typename T, typename... Args>
    T max(T value, Args... args) { return max(value, min(args...)); }
#endif

typedef long long ll;

const int INF = 1e9 + 7;
const int maxn = 1e3 + 7;

int n, k, T, a[maxn];

int getRegMin(int l, int r) {
    int res = INF;
    for (int i = l; i <= r; i += 1)
        res = min(res, a[i]);
    return res;
}

void addReg(int l, int r, int v) {
    for (int i = l; i <= r; i += 1)
        a[i] += v;
}

int main()
{
#ifdef AC
    freopen("4.in", "r", stdin);
    freopen("4.out", "w", stdout);
#endif
    scanf("%d", &T);
    while (T--) {
        scanf("%d%d", &n, &k);
        for (int i = 1; i <= n; i += 1) scanf("%d", a + i);
        int ans = 0;
        for (int i = 1; i + k - 1 <= n; i += 1) {
```

```
            int v = getRegMin(i, i + k - 1);
            ans += v;
            addReg(i, i + k - 1, -v);
        }
        printf("%d\n", ans);
    }
    return 0;
}
```

上述这个过程无非是两种操作，一种是长度为 k 的区间查询最小值，一种是长度为 k 的区间减的修改。可以通过用一种叫**线段树**的数据结构来优化。线段树可以用 $O(logn)$ 的复杂度来进行区间修改和区间查询。如果这个题 n 比较大，可以用此方法做这个题。大家有兴趣可以查一查。

Problem B: CrazyX and RSA-marryjianjian

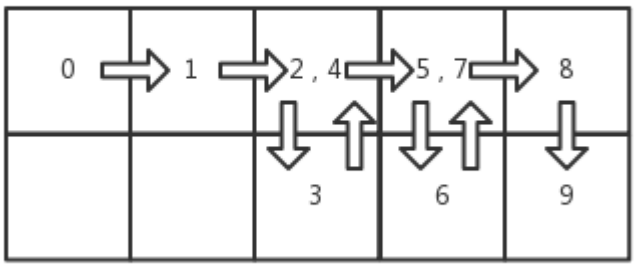
RSA为背景, 但其实RSA的数字都不会这么小的, 这里为了方便计算底数和幂指数都是小于等于100的, 所以直接暴力的计算就可以了, 一边乘一边模...至于快速幂, 这里用不用速度上其实是没有什么差别的, 因为幂指数很小, 但是模数范围是 $1e5$, 所以有可能在快速幂计算的时候会爆 int , 其实本来没想着在这里坑一下的, 一不小心...

```
#include <cstdio>
#include <iostream>
using namespace std;

int po(int c, int d, int n){
    int ans = 1;
    for(int i=0;i<d;i++){
        ans = ans * c % n;
    }
    return ans;
}

int main(){
    int c, d, n;
    while(scanf("%d%d%d", &c, &d, &n)==3){
        printf("%d\n", po(c, d, n));
    }
    return 0;
}
```

Problem C: Kindergarten-yuanzhaolin

在一个$n*m$的地图中，一个人想从左上角位置$(1,1)$处，以最短的时间移动到右下角(n,m)处，每秒可以走上、下、左、右四个方向。
每行第一个格子的左边有一个摄像头，每个摄像头都有一个工作周期t_i，工作过程是：工作$1s$后，再停止工作$(t_i-1)\ s$，给出每个摄像头在这个人开始移动后的第一次工作时刻$s_i(0\le s_i<ti)$
问：在保证不被任何一个摄像头拍到的情况下(在$T=0$时刻，允许被第一行摄像头拍到)，移动到(n,m)的最短时间是多少?

解法 首先，该题目要求到(n,m)的最短时间，很容易想到用宽度优先搜索的方式来求解，不过传统bfs算法解决迷宫问题是基于一个理论——“同一个格子没有必要经过两次。但是在本题中，很明显可以发现，最优解中可能是存在需要一个格子经过两次的情况的。如样例中：



图中$(1,3)$和$(1,4)$分别经过两次。所以用传统的bfs方法直接求解是错误的。 但是结合对每个摄像头的工作周期进行分析，我们可以发现： 假设$lcm(t_1,t_2,\dots,t_n)=L$ 对于两个时刻$x_1,x_2(x_1<x_2)\ x_1\equiv x_2\ \ mod\ L\Longrightarrow(\forall\ 1\le i\le n\quad x_1\equiv x_2\ \ mod\ t_i)$ $\therefore$ 我们可以认为$mod\ L$取值相同的时刻，各个摄像头在此刻及未来对于这些时间状态的影响是完全等价的。 因此我们可以尝试将当前已经走过的时间对L取模作为另外一个维度的状态参量，与当前搜索的横纵坐标一起组成状态空间进行搜索。 本题因为t_i值较小，$max[lcm(t_1,t_2,\dots,t_n)]=max(2,3,4,5,6)=60$ 所以总空间复杂度为$\mathcal{O}(60nm)\approx24000000$ 总时间复杂度同为$\mathcal{O}(60nm)$ 足以在题目要求时限内通过。

```
#include <bits/stdc++.h>
const int maxn=3e2;
using namespace std;
int vis[70][maxn][maxn];
const int dx[5]={0,0,1,-1,0};
const int INF=1e9+10;
const int dy[5]={1,-1,0,0,0};
int n,m;
int t[maxn],s[maxn];
int B;
int gcd(int a,int b)
{
    if(b==0) return a;
    return gcd(b,a%b);
}
int lcm(int a,int b)
{
    return a*b/gcd(a,b);
}
struct ST
{
    int b,x,y,t;
    ST()
    {
    }
    ST(int b,int x,int y,int t):b(b),x(x),y(y),t(t)
    {
    }
};
bool legal(int x,int y,int b)
{
    if(x<=0||x>n||y<=0||y>m) return false;
    if(b%t[x]==s[x]) return false;
    return true;
}
void solve()
{
    memset(vis,-1,sizeof(vis));
    B=1;
    for(int i=1;i<=n;i++) B=lcm(B,t[i]);
    queue<ST> Q;
    Q.push(ST(0,1,1,0));
    vis[0][1][1]=0;
    int res=INF;
    while(!Q.empty())
    {
        ST p=Q.front();
        Q.pop();
        for(int i=0;i<5;i++)
        {
            ST q=ST((p.b+1)%B,p.x+dx[i],p.y+dy[i],p.t+1);
            if(legal(q.x,q.y,q.b)&&vis[q.b][q.x][q.y]==-1)
            {
                vis[q.b][q.x][q.y]=q.t;
                Q.push(q);
            }
        }
    }
    for(int i=0;i<B;i++)
    {
        if(vis[i][n][m]!=-1)
            res=min(res,vis[i][n][m]);
    }
    if(res==INF)
```

```
        res=-1;
    printf("%d\n",res);
}
int main()
{
    /*
    freopen("../data/3.in","r",stdin);
    freopen("../data/3.out","w",stdout);
    */
    int T;
    scanf("%d",&T);
    for(int ca=1;ca<=T;ca++)
    {
        scanf("%d%d",&n,&m);
        for(int i=1;i<=n;i++)
            scanf("%d",&t[i]);
        for(int i=1;i<=n;i++)
            scanf("%d",&s[i]);
        solve();
    }
    return 0;
}
```

Problem D: Laptops-Eopxt

题目大意 给出两台电脑的价格和质量，问是否存在一台比另一台便宜但是质量更好。

做法 简单的语法题，注意不要只判断(a1 > a2 && b1 < b2)而忘了判断(a1 < a2 && b1 > b2)。

```
#include <bits/stdc++.h>
using namespace std;

int main() {
#ifdef Eopxt
    freopen("in.txt", "r", stdin);
    freopen("out.txt", "w", stdout);
#endif // Eopxt
    int cas;
    scanf("%d", &cas);
    while (cas--) {
        int a1, b1, a2, b2;
        scanf("%d%d%d%d", &a1, &b1, &a2, &b2);
        if ((a1 > a2 && b1 < b2) || (a1 < a2 && b1 > b2)) printf("Happy Smile\n");
        else printf("Poor Smile\n");
    }
    return 0;
}
```

Problem E: Oscar Buy an Apartment-marryjianjian

不难想到, $n == k$ 的时候和 $k == 0$, 满足条件的最小和最大都是 0.
其他情况下, 当所有的已经被买了的房子都挤在一边的时候, 可以得到最小可能的值都是 1. 而一个已经买了的房子最多可能在和两个位置相邻, 所以在房子足够多的情况下, $2 * k$就是最大值的答案, 当房子不够的时候, 剩下的位置一定存在都可以满足条件的情况, 这时候最大值的答案就是 $n - k$.
而对于 $2 * k$ 和 $n - k$ 的分界情况的判断就是 $3 * k$ 是否大于 n (相等时, $2 * k$ 和 $n - k$ 也相等). 不过这里需要看一下 k 的范围也是 $1 \leq k \leq 1e9$, 而 $3 * 1e9$ 有可能爆 int, 事实是的确存在这样的数据, 当然如果你采用除法去判断就不会有这种情况了.

```
#include <cstdio>
#include <iostream>
#include <algorithm>
using namespace std;
typedef long long ll;

int main(){
    ll n, k, a, b;
    while(scanf("%lld%lld", &n, &k) == 2){
        if(n == k || k == 0){
            printf("0 0\n");
            continue;
        }
        a = 1l;
        if(3 * k > n){
            b = n - k;
        }
        else{
            b = 2 * k;
        }
        printf("%lld %lld\n", a, b);
    }
    return 0;
}
```

Problem F: Random Statistic-yuanzhaolin

给出集合A,B,C的计算公式如下： $A=\sum_{i=1}^{|S|}x_i$ $B=(\sum_{i=1}^{|S|}x_i)^2$ $C=\max_{i=1}^{|S|}x_i$ 从一个大小为n的集合中随意选出一些数构成新的集合S，问集合S的A,B,C三个值的期望。

解法

定义

为原集合中每个元素定义一个0-1分布。
$X_i\sim(a_i,1/2)$ a_i为原集合中第i个元素的值：

$E(A)$的计算

$E(A)=E(X_1+X_2+\ldots+X_n)$
由于X_i之间相互独立，可以将上式拆解为：
$E(A)=E(X_1)+E(X_2)+\ldots+E(X_n)$
$=\frac{1}{2}(a_1+a_2+\ldots+a_n)$

$E(B)$的计算

$E(B)=E[(X_1+X_2+\ldots+X_n)^2]$
$=E[X_1^2+X_2^2+\ldots+X_n^2+\sum_{i,j}\sum_{i<j}2X_iX_j]$
$=E[X_1^2]+E[X_2^2]+\ldots+E[X_n^2]+\sum_{i,j}\sum_{i<j}E[2X_iX_j]]$
$\because$第i个数出现在选中的集合中的概率为$\frac{1}{2}$
$\therefore E(X_i^2)=\frac{a_i^2}{2}$
$\because$第i个数和第j个数同时出现在选中的集合中的概率为$\frac{1}{4}$
$\therefore E(X_iX_j)=\frac{a_ia_j}{4}$
$\therefore E(B)=\sum_{i=1}^{n}\frac{a_i^2}{2}+\sum_{i,j}\sum_{i<j}\frac{a_ia_j}{2}$

$E(C)$的计算

对于$E(C)$的求解我们可以先将原数组按照从大到小的顺序排序。 满足:
$a_1\geq a_2\geq\ldots\geq a_n$
当a_1被选中时,即有$\frac{1}{2}$的概率C的值必然为a_1，当a_1没被选中的情况下，则有$\frac{1}{2}$的概率C的值为a_2，以此类推。另外，题目说明了当集合为空集时C的值定义为0，因此。可以得到$E(C)$的计算公式如下：
$E(C)=\frac{1}{2}a_1+\frac{1}{4}a_2+\ldots+\frac{1}{2^i}a_i+\ldots+\frac{1}{2^n}a_n+\frac{1}{2^n}\times 0$
代码实现过程中，直接用一个double类型的变量来存储系数，没计算一项之后乘以0.5即可。

```
#include <bits/stdc++.h>
using namespace std;
const int maxn=1100;
int n;
double x[maxn];
int main()
{
    int T;
    scanf("%d",&T);
    for(int ca=1;ca<=T;ca++)
    {
        cin>>n;
        double A=0,B=0,C=0;
        for(int i=1;i<=n;i++)
        {
            scanf("%lf",&x[i]);
            A+=x[i];
            B+=x[i]*x[i]/2;
        }
        A/=2;
        for(int i=1;i<=n-1;i++)
            for(int j=i+1;j<=n;j++)
                B+=x[i]*x[j]/2;
        sort(x+1,x+n+1);
        double t=1;
        for(int i=n;i>=1;i--)
        {
            t/=2;
            C+=t*x[i];
        }
        printf("%.4lf %.4lf %.4lf\n",A,B,C);
    }
    return 0;
}
```

Problem G: Geometry-crazyX

N 维找 $N+1$ 个点，我们考虑从$N+1$ 维找$N+1$ 个点

设$N+1$ 个点坐标为$A_i(1<=i<=n+1)$

可以构造出这样一组坐标($N+1$维)：

$A_1(n+1,0,0,\ldots,0)$

$A_2(0,n+1,0,\ldots,0)$

$\ldots$

$A_{n+1}(0,0,0,\ldots,n+1)$

注意到这$n+1$个点满足坐标方程$x_1+x_2+\ldots+x_{n+1}=n+1$, 这个是$n+1$维空间下的一个超平面， 即这些点都在一个$N$维空间内

同时它们的均值点为$O(1,1,1,\ldots,1)$也满足上述方程

显然，$|OA_i|$全部相等并且构成的所有夹角都相等，即满足题意。

对于求夹角，我们可以直接在$N+1$维坐标下进行计算。

任选两个向量$\overrightarrow{OA_1}=(n,-1,-1,\ldots,-1),\overrightarrow{OA_2}=(-1,n,-1,\ldots,-1)$

利用余弦定理公式： $cos\theta=\frac{\vec{v}_1\vec{v}_2}{|\vec{v}_1||\vec{v}_2|}$

$$cosx=\frac{-n-n+1*(n-1)}{\sqrt{n^2+n}\sqrt{n^2+n}}=-\frac{1}{n}$$

题目为了降低难度，给大家能猜结论的空间，直接要求输出$cosx$的倒数并且提醒了答案必为整数，输出$-n$即可。

PS:

超平面：n维欧氏空间中余维度等于1的线性子空间

n维空间F^n中的超平面由方程：$a_1x_1+\ldots+a_nx_n=b$定义，超平面可以看做是二维平面内的直线，三维空间内的面在高维度的推广。

```cpp
# include <bits/stdc++.h>

using namespace std;

int main(void) {
# ifdef owly
    freopen("1.in", "r", stdin);
    freopen("1.out", "w", stdout);
# endif

    int T;
    scanf("%d", &T);
    while (T --) {
        int n;
        scanf("%d", &n);
        printf("%d\n", -n);
    }

    return 0;
}
```

Problem H: Stones-owly

题目大意是在一个 $n \times m$ 格子里面，有一些随机分布的石头，现可以将任意的石头往右移或者往下移，问能否使得格子里的每一个单元里面都有石头。

先考虑第一行。这时，不存在上一行的石头向下移动到这一行的情况，那么就考虑向右移动使得这一行所有的空单元有石头。因此，如果一个单元有多个石头，右边缺多少个石头就往右移动多少个。对于第一行左数第一个来说，如果这个单元的石头不够它本身一个加上右边缺的个数，那么就是一定不行的，否则就是一定可以的。

之后，对于每一个格子，我们贪心的将多的石头挪下来，用于填充下一行的单元。

我们可以看出，一个石头越靠左，它能填充的范围就越大，左边的石头越多，就越可能达到题目要求。那么我们模拟的过程中，就需要尽可能的将石头留在左边，具体方法就是如果右边不缺石头，就一定不要右移。

```cpp
# include <bits/stdc++.h>
# define cmax(x,y) x=max(x,y)
# define cmin(x,y) x=min(x,y)
using namespace std;
typedef double DB;
typedef long long LL;
typedef pair<int, int> Pr;

const int N = 1000 + 100;

int a[N][N];
int n, m, k;

int b[N];

int main(void) {
# ifdef owly
    freopen("2.in", "r", stdin);
    freopen("2.out", "w", stdout);
# endif

    int T;
    scanf("%d", &T);
    while (T --) {
        scanf("%d%d%d", &n, &m, &k);
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= m; j++) a[i][j] = 0;

        int x, y;
        for (int i = 0; i < k; i++) {
            scanf("%d%d", &x, &y);
            a[x][y] ++;
        }

//        for (int i = 1; i <= n; i++)
//            for (int j = 1; j <= m; j++) printf("%d%c", a[i][j], " \n"[j==m]);

        bool ans = true;
        for (int i = 1; i <= n && ans; i++) {
            b[m + 1] = 0;
            for (int j = m; j >= 1; j--) {    // b 表示该位置加上多少个石头，可以使得右边所有的空单元有石头
                if (a[i][j]) b[j] = max(0, b[j+1] - a[i][j] + 1);  // 最多可以提供 a[i][j] - 1 个石头
                else b[j] = b[j+1] + 1;    // 如果为空，就需要多加一个石头填在 j 位置
            }

            for (int j = 1; j <= m; j++) {
                if (a[i][j] == 0) ans = false;
                if (a[i][j] - 1 < b[j+1]) ans = false;
                if (!ans) break;
                a[i][j] -= b[j+1];
                a[i][j+1] += b[j+1];
            }

            if (i == n || !ans) break;
            for (int j = 1; j <= m; j++)
                if (a[i][j] > 1) a[i+1][j] += a[i][j] - 1, a[i][j] = 1;
//            puts("");
//            for (int i = 1; i <= n; i++)
//                for (int j = 1; j <= m; j++) printf("%d%c", a[i][j], " \n"[j==m]);
        }

        puts(ans ? "YES": "NO");
```

```
    }


    return 0;
}
```

Problem I: Summer Social Practice of Electronic Department of SUOAO-Martian

题目大意是在N种元器件中选择一些元器件。每种元器件具有一定的花费、体积、价值。目的是让选择的元器件的总花费不超过额定花费C，总体积不超过额定体积B的情况下总价值最大。每一种元器件可选择的个数不限。题目保证每个元件的花费和体积都大于等于1。

这是一个二维重量的完全背包问题。 将每一种元器件依次标号为第$1,2,3...n$种。 假设$dp[n][b][c]$代表在前n种元器件里面选择总重量不超过b,总花费不超过c的选择方案中可以获取的最大价值数。$bulk[n]$表示第n个元件占有的体积，$cost[n]$代表第n个元件需要的花费。$value[n]$代表第n个元件能获得的价值。那么，在考虑第n种元器件的时候有两种选择方法，一种是选上第n种元器件x次，一种是不选第n种元器件。如果不选择第n种元器件，那么执行完这一次选取操作后的值就是考虑相同体积和重量的情况下前$n-1$种元器件最好的选取方案，也就是$dp[n-1][b][c]$；如果选择第n种元器件x次，那么此次选取之前的最大价值是在前$n-1$种元器件里选取减掉当前元器件重量的重量和减掉当前元器件体积的体积对应的最大价值，也就是 $dp[n][b-x\times bulk[n]][c-x\times cost[n]],(b\geq x\times bulk[n])(c\geq x\times cost[n])$ 。 然后再加上$x\times value[n]$ ， 就是$dp[n][b][c]$的值。我们可以不枚举x从1开始，让b和c都是从小到大扫描的话，每一次都只选取一个第n种元器件，但是也是在前n种元器件里面选择，重量和花费分别为$b-bulk[n]$和$c-cost[n]$。这样的话同样都是$dp[n][b][c]$，我们从b,c小往大扫描，每一次选择都继承上一次的$dp[**n**][b][c]$的值而不是直接从$dp[**n-1**][b][c]$里面选择。这样，dp[n][b][c]的值可以从$dp[n-1][b][c]$中和$dp[n][b-bulk[n]][c-cost[n]]+value[n]$当中取最大值。如果$b<bulk[n]$或$c<cost[n]$的话就不考虑选取第$n$种元件，也就是$x=0$。 于是，状态转移方程就是

```
if(b>=bulk[n]&&c>=cost[n])  dp[n][b][c]=max( dp[n-1][b][c] , dp[n][b-bulk[n]][c-
cost[n]]+value[n] );
else                        dp[n][b][c]=     dp[n-1][b][c];
```

这就是完全背包问题的递推思路，本题将重量从一元变量变成二元变量，但递推思路不变。具体动态规划算法的详细介绍请参考*《挑战程序设计竞赛》*第二章第三节。

本题参考代码：

```cpp
#include<bits/stdc++.h>
using namespace std;
const int maxB=30,maxC=300,maxN=100;
int dp[maxN+2][maxB+2][maxC+2];
int B,C,N;
int bulk[maxN+2],cost[maxN+2],value[maxN+2];
int main(){
    int T;
    scanf("%d",&T);
    for (int ti=1;ti<=T;++ti){
        scanf("%d%d%d",&N,&B,&C);
        for (int i=1;i<=N;++i){
            scanf("%*s%d%d%d",&bulk[i],&cost[i],&value[i]);
        }
        memset(dp,0,sizeof(dp));
        for (int i=1;i<=N;++i){
            for (int b=0;b<=B;++b){
                for (int c=0;c<=C;++c){
                    if(b>=bulk[i]&&c>=cost[i]) dp[i][b][c]=max(dp[i-1][b][c],dp[i][b-bulk[i]]
[c-cost[i]]+value[i]);
                    else dp[i][b][c]=dp[i-1][b][c];
                }
            }
        }
        printf("%d\n",dp[N][B][C]);
    }
    return 0;
}
```

Problem J: Dipper and Mabel-smile & nbyby

有n对双胞胎参加$party$，每个人除了自己的双胞胎外和其他人都不认识，然后他们互相加了好友，好友关系是双向的，除小A以外的$2n-1$个人每个人新增的好友数量构成了$0\sim 2n-2$的一个排列，问小A新增了多少个好友。

做法

我们先从简单的情况考虑起：$n=2$时，记4个人为A、B、C、D，则B、C、D这3个人的新朋友数量分别为0、1、2。首先考虑谁加了2个新好友，如果是小B，由于他和小A已经是好友了，所以他把C、D都加了好友，这样一来，C、D都至少增加了1个好友，与B、C、D三人中有1个人加了0个好友矛盾。那么不妨设C加了2个新好友，则A、B都是他的好友，并且只有他的双胞胎C可能加了0个好友，这时A、B加了1个好友，是一个符合要求的方案。

$n=3$时，记6个人为A、B、C、D、E、F，则B、C、D、E、F的新好友数量分别为0、1、2、3、4。还是考虑谁加了4个好友，如果是小B，那么C、D、E、F都是他的好友，就没人新增了0个好友了，不合题意。不妨设E加了4个好友，则A、B、C、D都是他的好友，所以加了0个好友的人是他的双胞胎F。**如果我们把这两个人去掉，那么等价于A、B、C、D这4个人构成了$n=2$时的情况。**

推广到n更大的情况，类似的，第$2n-1$个人增加了$2n-2$个好友（包括A），第$2n$个人增加了0个好友，把这两个人去掉就转换为$n-1$的情况。也就是说n增大1所产生的影响仅仅是A增加了1个好友，所以答案是$n-1$。

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
#ifdef Eopxt
    freopen("in.txt", "r", stdin);
    freopen("out.txt", "w", stdout);
#endif // Eopxt
    int cas;
    scanf("%d", &cas);
    while (cas--) {
        int n;
        scanf("%d", &n);
        printf("%d\n", n-1);
    }
    return 0;
}
```