A 中奖

签到题

思路

使用结构体来存储每个订单的 a、b、c。为了方便比较，也可以同时记录一个订单从 0 点开始，到下单时间的分钟数（60*a+b）。

接下来是选择排序，双关键字排序。枚举 i 和 j，使最后钱多的在前面。具体比较方式是，如果 i 的钱更少，或者 i 和 j 的钱相同，但 i 比 j 下单晚，则将 i 和 j 交换。

结构体可以整体赋值，交换结构体就是这个操作。

代码

```cpp
#include<iostream>
using namespace std;
struct order {
    int a, b, t, c;
} o[2010];
int main() {
    int n, m;
    cin >> n >> m;
    for (int i = 1; i <= n; i++) {
        int a, b, c;
        cin >> a >> b >> c;
        o[i].a = a;
        o[i].b = b;
        o[i].t = a * 60 + b;
        o[i].c = c;
    }

    for (int i = 1; i <= n; i++)
        for (int j = 1; j < i; j++) {
            if (o[j].c < o[i].c
            || (o[j].c == o[i].c && o[j].t > o[i].t)) {
                order p;
                p = o[i];
                o[i] = o[j];
                o[j] = p;
            }
        }
    for (int i = 1; i <= m; i++)
        cout << o[i].a << ' ' << o[i].b << ' ' << o[i].c << endl;
}
```

B 构造一个简单的数列

$a=1$ 时 $f(a,n)=a$

否则前 a 项分别为 $a,1,2,...,a-1$ 。

然后找到一个可行的 b 满足 $\gcd(a-1,b)=1$ ，再在 $a-1$ 之后填上 $b,a+1,a+2,...,b-1$ （如果 $b=a+1$ 则只填 $b,b+1,b+2...$ ），如此反复即可。

可以证明在题目条件下，若 $\gcd(a-1,b)=1$ ，则有 $\gcd(a+1,b)=1$ 。

证明如下：

```
不妨假设所有素数是一个列表，p1是2 p2是3 p3是5 以此类推
不妨假设k1k2k3...是一个数列
不妨假设x是Pk1 Pk2 Pk3...Pkn，若干素数之积
我们将x的所有素因子排序后可能会得到x=P1*P2*P3*...*Pm*P(m+2)*P(m+5)*...，此处，x最小的不存在的素因子是P(m+1)
因此y必为x+P(m+1)
此时y-(x+1)=P(m+1)-1。如果y与x+1有公因数，则该公因数一定是P(m+1)-1的因数，不妨假定其为t。
而我们又注意到t一定小于P(m+1)，换言之t一定是P1,P2,...,Pm中的一个。但容易发现P1,P2...Pm均为x的因数，故他们不可能是x+1的因
```

代码

```cpp
#include<bits/stdc++.h>
#define ll long long
using namespace std;
int T, a, n;
int f[1000000 + 10];
bitset<2000000> vis;
int main()
{
    cin >> T;
    while (T--)
    {
        vis.reset();
        int pos = 1;
        cin >> a >> n;
        vis[a]=1;
        f[1] = a;
        if (f[1] == 1)
        {
            cout << n << "\n";
            continue;
        }
        else
            f[2] = 1;
        for (int i = 3, z; i <= n; i++)
        {
            f[i] = f[i - 1] + 1;
            if (vis[f[i]])
            {
                z = f[i];
                f[i]++;
                while (__gcd(f[i], f[i - 1]) != 1)
                    f[i]++;
                vis[f[i]] = 1;
                f[i + 1] = z + 1;
                if (f[i + 1] == f[i])
                    f[i + 1]++;
                i++;
            }
        }
        cout << f[n] << "\n";
    }
    return 0;
}
```

C 同心的凸正多面球

name	l	R	V
tetrahedron	$\frac{\sqrt{2}}{4}a$	$\frac{\sqrt{6}}{4}a$	$\frac{\sqrt{2}}{12}a^3$
hexahedron	$\frac{\sqrt{2}}{2}a$	$\frac{\sqrt{3}}{2}a$	a^3
octahedron	$\frac{1}{2}a$	$\frac{\sqrt{2}}{2}a$	$\frac{\sqrt{2}}{3}a^3$
dedocahedron	$\frac{3+\sqrt{5}}{4}a$	$\frac{\sqrt{3}\left(1+\sqrt{5}\right)}{4}a$	$\frac{15+7\sqrt{5}}{4}a^3$
icosahedron	$\frac{1+\sqrt{5}}{4}a$	$\frac{\sqrt{10+2\sqrt{5}}}{4}a$	$\frac{15+5\sqrt{5}}{12}a^3$
6,6,5	$\frac{3\left(\sqrt{5}+1\right)}{4}a$	$\frac{\sqrt{58+18\sqrt{5}}}{4}a$	$\frac{125+43\sqrt{5}}{4}a^3$
10,10,3	$\frac{5+3\sqrt{5}}{4}a$	$\frac{\sqrt{74+30\sqrt{5}}}{4}a$	$\frac{515+215\sqrt{5}}{12}a^3$
3,3,3,3,4	$K_1^{'}a$	K_1a	K_2a^3
3,3,3,3,5	$K_3^{'}a$	K_3a	K_4a^3
3,5,4,5	$\frac{\sqrt{10+4\sqrt{5}}}{2}a$	$\frac{\sqrt{11+4\sqrt{5}}}{2}a$	$\frac{195+98\sqrt{5}}{12}a^3$
4,6,8	$\sqrt{\frac{6+3\sqrt{2}}{2}}a$	$\frac{\sqrt{13+6\sqrt{2}}}{2}a$	$2\left(11+7\sqrt{2}\right)a^3$
4,6,10	$\frac{\sqrt{30+12\sqrt{5}}}{2}a$	$\frac{\sqrt{31+12\sqrt{5}}}{2}a$	$2\left(19+7\sqrt{5}\right)a^3$
4,4,4,3	$\frac{\sqrt{4+2\sqrt{2}}}{2}a$	$\frac{\sqrt{5+2\sqrt{2}}}{2}a$	$\frac{10\sqrt{2}+12}{3}a^3$
8,8,3	$\frac{\sqrt{2}}{2}\left(\sqrt{2}+1\right)a$	$\frac{\sqrt{7+4\sqrt{2}}}{2}a$	$\frac{21+14\sqrt{2}}{3}$
6,6,3	$\frac{3\sqrt{2}}{4}a$	$\frac{\sqrt{22}}{4}a$	$\frac{23\sqrt{2}}{12}a^3$
6,6,4	$\frac{3}{2}a$	$\frac{\sqrt{10}}{2}a$	$8\sqrt{2}a^3$
3,4,3,4	$\frac{\sqrt{3}}{2}a$	a	$\frac{5\sqrt{2}}{3}a^3$
3,5,3,5	$\frac{\sqrt{5+2\sqrt{5}}}{2}a$	$\frac{\sqrt{5}+1}{2}a$	$\frac{45+17\sqrt{5}}{6}a^3$

正多边形只有 5 种（见上图的前 6 行）。存在一组 (l, R) 使得 $r \in [l, R]$ 时输出 YES，否则输出 NO。

代码

```cpp
#include <bits/stdc++.h>
using namespace std;
int check1(double a, double r)
{
    if (r > a * sqrt(6.0) / 4)
        return false;
    if (r < a * sqrt(2.0) / 4)
        return false;
    return 1;
}
int check2(double a, double r)
{
    if (r > a * sqrt(3.0) / 2)
        return false;
    if (r < a * sqrt(2.0) / 2)
        return false;
    return 2;
}
int check3(double a, double r)
{
    if (r > a * sqrt(2.0) / 2)
        return false;
    if (r < a / 2.0)
        return false;
    return 4;
}
int check4(double a, double r)
{
    if (r > a * sqrt(3.0) * (1 + sqrt(5.0)) / 4)
        return false;
    if (r < a * (3 + sqrt(5.0)) / 4)
        return false;
    return 8;
}
int check5(double a, double r)
{
    if (r > a * sqrt(10 + 2 * sqrt(5)) / 4.0)
        return false;
    if (r < a * (1 + sqrt(5.0)) / 4)
        return false;
    return 18;
}
int check(double a, double r)
{
    return check1(a, r) | check2(a, r) | check3(a, r) | check4(a, r) | check5(a, r);
}
int main()
{
    int a, r;
    cin >> a >> r;
```

```
    cerr << check(a, r) << endl;
    if (check(a, r))
        cout << "YES" << endl;
    else
        cout << "NO" << endl;
}
```

D 完全二叉树的可能性

如果 $n \le 20$ ，我们可以用 $dp[i][0/1][0/1]$ 表示第 i 个点为 $0/1$ 时 $0/1$ 最多的个数，此时可以用动态规划解决。可以发现，原来每层的初始状态是一样的，所以我们可以用 $dp2[k][0/1][0/1]$ 表示每一层的初始值，然后更新的 $dp[i][0/1][0/1]$ 用 map 来存。

代码

```cpp
#include <bits/stdc++.h>
using namespace std;
struct pp
{
    long long *ara;
    pp()
    {
        ara = new long long[4];
        ara[3] = ara[0] = 1;
        ara[1] = ara[2] = 0;
    };
    void update(pp a, pp b, bool flag)
    {
        if (flag)
        {
            ara[3] = max(a.ara[1] + b.ara[1], a.ara[3] + b.ara[3]) + 1;
            ara[2] = max(a.ara[0] + b.ara[0], a.ara[2] + b.ara[2]);
            ara[1] = max(a.ara[1] + b.ara[3], a.ara[1] + b.ara[3]);
            ara[0] = max(a.ara[0] + b.ara[2], a.ara[0] + b.ara[2]) + 1;
        }
        else
        {
            ara[3] = max(a.ara[1] + b.ara[3], a.ara[3] + b.ara[1]) + 1;
            ara[2] = max(a.ara[0] + b.ara[2], a.ara[2] + b.ara[0]);
            ara[1] = max(a.ara[1] + b.ara[1], a.ara[3] + b.ara[3]);
            ara[0] = max(a.ara[0] + b.ara[0], a.ara[2] + b.ara[2]) + 1;
        }
    }
    void print()
    {
        cout << max(ara[2], ara[0]) << " " << max(ara[3], ara[1]) << "\n";
    }
};
map<long long, pp> dp;
pp dp2[61];
map<long long, bool> flag;
pp getdp(long long id)
{
    if (dp.count(id))
        return dp[id];
    long long cid = 0;
    while ((1 << cid) <= id)
        cid++;
    return dp2[cid];
}

int main()
{
    long long n, q, x;
    cin >> n >> q;
```

```
    for (int i = n - 1; i >= 1; i--)
    {
        dp2[i].update(dp2[i + 1], dp2[i + 1], 0);
    }
    while (q--)
    {
        cin >> x;
        x++;
        flag[x] = !flag[x];
        while (x > 0)
        {
            dp[x].update(getdp(x << 1), getdp(x << 1 | 1), flag[x]);
            x >>= 1;
        }
        dp[1].print();
    }
    return 0;
}
```

E 排列计数

一个长度为 $2n$ 的排列有 $2n^2-n$ 个有序对。如果一个排列 $p_1,p_2,\ldots,p_n$ 有一半以上的有序对是顺序对，那么排列 $p_n,p_{n-1},\ldots,p_2,p_1$ 有一半以下的有序对是逆序对。因此本题答案为 $\frac{(2n)!}{2}$ 。

```
#include <iostream>
#include <cmath>
using namespace std;
const int mod=1e9+7;
int main()
{
	long long n;
	cin>>n;
	long long ans=1;
	for(int i=3;i<=2*n;i++) ans*=i,ans%=mod;
	cout<<ans<<endl;
	return 0;
}
```

F 金玉其外矩阵

如果 N 是 h 的倍数，并且 M 是 w 的倍数，那么此时一定可以将矩阵恰好分成若干个 $h \times w$ 的子矩阵。这些子矩阵里的数的和都是负数，而原矩阵中所有数的和此时刚好是这些子矩阵的和，故也为负数。因此此时无解。

否则，我们不失一般性地假设 N 不是 h 的倍数。可以采取如下构造方法：

对第 i 行第 j 列的数，如果 i % h != 0，那么该数为 1000 。

否则，该数为 $-1000 \times (h-1) - 1$ 。

比如，若N=4，M=3，h=3，w=2，则构造出的矩阵为

```
1000 1000 1000
1000 1000 1000
-2001 -2001 -2001
1000 1000 1000
```

容易发现，采取这种构造方法时，每一个子矩阵中都恰有 $(h-1) \times w$ 个 1000 和 w 个 $-1000 \times (h-1) - 1$ 。通过计算可知子矩阵的和一定是负数。

而每一列的和一定是正数，因为它是由 N/h 个 -1 和至少一个 1000 组成的。因此，整个矩阵的和一定是正数，故满足条件。

代码

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
        int N,M,n,m,i,j;
        scanf("%d %d %d %d",&N,&M,&n,&m);
        if((N%n==0)&&(M%m==0))
        {
                printf("N");
                return 0;
        }

        printf("Y\n");
        if(N%n!=0)
        {
                for(i=1;i<=N;i++)
                {
                        for(j=1;j<=M;j++)
                        {
                                if(i%n!=0)
                                {
                                        printf("1000");
                                }
                                else
                                {
                                        printf("%d",-(1000*(n-1)+1));
                                }
                                if(j!=M) printf(" ");
                                else printf("\n");
                        }
                }
        }
        else
        {
                for(i=1;i<=N;i++)
                {
                        for(j=1;j<=M;j++)
                        {
                                if(j%m!=0)
                                {
                                        printf("1000");
                                }
                                else
                                {
                                        printf("%d",-(1000*(m-1)+1));
                                }
                                if(j!=M) printf(" ");
                                else printf("\n");
                        }
                }
```

```
        }
        return 0;
}
```

G 登山小分队

模拟题。本题数据范围很小，$n \leq 2000$，总花费时间不会太多。因此可以模拟每一轮每一个人的状态：如果可以前进则前进，如果不能前进则留在原地。

代码

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef vector<int> vi;
typedef pair<int,int> pii;
#define sz(v) ((int)(v.size()))
#define all(v) (v).begin(),(v).end()
#define pb push_back
mt19937 mrand(random_device{}());
int rnd(int x) { return mrand() % x;}
const int N=2e3+7,inf=0x3f3f3f3f,mod=1e9+7;

vector<int>g[N];
int fa[N];

void dfs(int u,int f) {
    for(auto &it:g[u]) {
        int v=it;
        if(v==f) continue;
        fa[v]=u;
        dfs(v,u);
    }
}
bool use[N];

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n=1024;
    cin>>n;
    for(int i=0,u,v;i<n-1;++i) {
        cin>>u>>v;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    dfs(1,0);
    vector<int>curpos;
    for(int i=2;i<=n;++i) {
        if(g[i].size()==1) {
            curpos.push_back(i);
        }
    }
    int ans=0,sz=curpos.size();
    while(1) {
        bool ok=1;
        for(auto &pos:curpos) {
            if(pos!=1) {
                ok=0;
                break;
            }
```

```
        }
        if(ok) break;
        ans++;
        for(int i=1;i<=n;++i) use[i]=0;
        for(auto &pos:curpos) {
            if(pos==1) continue;
            if(!use[pos]) {
                use[pos]=1;
                pos=fa[pos];
            }
        }
    }
    cout<<ans<<'\n';
    return 0;
}
```

H 拼接的字符串

签到题。

思路

如果第一行的字符串可以由第二行字符串的一个前缀+一个后缀拼接而成，则输出 `YES` ，否则输出 `NO` 。

因此我们只需求出两个字符串最长**公共前缀(指两个字符串完全相同的前缀)**的长度和最长公共后缀的长度，然后判读一下这两个部分加起来能否覆盖整个第一行的字符串。

代码

```
#include <bits/stdc++.h>
using namespace std;
-int main()
{
    string s, s1;
    cin >> s >> s1;
    int i = 0, j = 0;
    while (i < s.size() && i < s1.size() && s[i] == s1[i])
        i++;
    while (j < s.size() && j < s1.size() && s[s.size() - 1 - j] == s1[s1.size() - 1 - j])
        j++;
    if (i + j >= s.size())
        cout << "YES" << endl;
    else
        cout << "NO" << endl;
}
```

I 没有字母的数

最直观的想法是在 $[l,r]$ 区间枚举所有的数，判断其是否包含字母。

由于有多组数据，且数据量较大，所以不能直接这样做。

但是我们可以在处理测试样例前，首先前缀和求出 $f[k]$ 表示 $[1,k]$ 区间的不包含字母的数的数量。
这一步预处理的时间复杂度是 $O(k\log_{16}k)$
对于每组测试样例，答案就很明显是 $f[r]-f[l-1]$ 。
这样做的时间复杂度是 $O(t+\text{预处理})$ ，可以通过此题。

代码

```
#include <bits/stdc++.h>
using namespace std;
bool isletter(int x){
	bool flag=1;
	while(x>1){
		int p=x%16;
		if(p>=10)flag=0;
		x/=16;
	}
	return flag;
}
int f[1000010];
int main()
{
	for(int i=1;i<=1000000;i++){
		f[i]=f[i-1];
		f[i]+=isletter(i);
	}
	int T;cin>>T;
	for(;T;T--){
		int l,r;cin>>l>>r;
		cout<<f[r]-f[l-1]<<endl;
	}
	return 0;
}
```

J 有趣的序列

首先注意到如果 $kn \le c_i < (k+1)n$ ，那么 $\lfloor \frac{c_i}{n} \rfloor + 1 = k + 1$ ，则显然 $(k+1)n \le c_{i+1} < (k+2)n$

所以有结论：这个子序列一定占连续的 x 段 $(x \le k, x \le \frac{l}{n})$

$dp[i][j]$ 表示考虑到当前是答案子序列的第 i 个数时，且当前的数在 a 序列中排名第 j 小时的方案数。

那么 $dp[i][j]$ 对答案的贡献就是 $dp[i][j] \times (\frac{l}{n} - i + 1)$

特殊情况就是 l 不一定被 n 整除，所以我们令 $r = l\%n$ ，那么在 dp 当前枚举 a 中 a_q 转移的时候，如果 $q < r$ 且 $i \le \frac{l}{n} + !!r$ 时，也可以加入答案。

dp 转移就是个前缀和转移。

注意空间问题，使用了滚动数组进行优化。

代码

```cpp
#include <bits/stdc++.h>
using namespace std;
const ll mod=1000000007;
int a[1000010], b[1000010];
int q[1000010];
int dp[2][1000010];
void add(int &v, int x) {
        v += x;
        if (v >= mod) v -= mod;
}
int main() {
        ll l;
        int n, k;
        scanf("%d%lld%d", &n, &l, &k);
        for (int i = 0; i < n; i++) {
                scanf("%d", &a[i]);
                b[i] = a[i];
        }
        sort(b, b + n);
        int m = unique(b, b + n) - b;
        for (int i = 0; i < n; i++) {
                q[i] = lower_bound(b, b + m, a[i]) - b;
        }
        long long p = l / n, r = l % n;
        long long c;
        int ans = 0;
        for (int i = 0; i < m; i++) {
                dp[1][i] = 1;
        }
        for (int i = 1; i <= k; i++) {
                c = i <= p ? p - i + 1 : 0;
                c %= mod;
                int y = i & 1, z = y ^ 1;
                for (int j = 0; j < m; j++) {
                        dp[z][j] = 0;
                }
                for (int j = 0; j < n; j++) {
                        int s = q[j];
                        add(dp[z][s], dp[y][s]);
                        add(ans, c * dp[y][s] % mod);
                        if (j < r && i <= p + !!r) {
                                add(ans, dp[y][s]);
                        }
                }
                for (int j = 1; j < m; j++) {
                        add(dp[z][j], dp[z][j-1]);
                }
        }
        printf("%d\n", ans);
```

```
        return 0;
}
```