A 玉桑的环星球

出题人:爆哥

考察点:数学推导，解方程

首先我们对于这个题，忽略掉第 i 天走 i^2 步这个条件，假定要是一天走一步的话，那么我们可以分析，对于不同的星球，每次回到了自己原来的出生点，那么必然是已经走了 $k*A_i\ (k\geq 1)$ 步，那么对于 n 个星球的话，他们要同时满足走 X 步回到各自出生点，那么就是 $k_1*A_1=k_2*A_2...=k_n*A_n=X$ 那么就发现最小的 $X=LCM(A_1,A2,...,A_n)$

现在开始用 lcm 代替 $LCM(A_1,A_2,...,A_n)$

然后继续回到问题，那么我们可以得出结论：走的步数最后一定是这些数的最小公倍数的倍数。

那么我假设最少走了 m 天之后 便可以满足他们各自重新回到了出生点，根据平方和公式得出 m 天走了 $\frac{m*(m+1)*(2m+1)}{6}$ 步，那么根据前面推导，我们可以得出 $\frac{m*(m+1)*(2m+1)}{6}=k*lcm\ (k\geq 1)$

移项之后 $m*(m+1)*(2m+1)=6*lcm*k\ (k\geq 1)$

然后这个时候我们可以得出 $m*(m+1)*(2m+1)$ 是 $6*lcm$ 的倍数这个结论

然后此时你可以发现 $m\ m+1\ 2m+1$ 这三个数俩俩互质，这一点可以通过求辗转相减求最大公约数得出。

这个时候我们假设 $6*lcm$ 一共有 q 个质数,第 i 个质数有 b_i 个 那么 $6*lcm=\prod_{i=1}^{q}p_i^{b_i}$

那么通过上面得出的互质结论，每一个 $p_i^{b_i}$ 是 $m\ m+1\ 2m+1$ 其中一个的因子

然后题目给定 $lcm<=10^{12}$ 那么 $6*lcm<10^{13}$ 所以 q 最大不超过 11

那么我们可以暴力枚举每一个 $p_i^{b_i}$ 是这三个数中的哪一个因子

假设暴力分配后的为 a_1,a_2,a_3 ,那么就有 $a_1*k_1=m,a_2*k_2=m+1,a_3*k_3=2m+1\ (k_1,k_2,k_3\geq 1)$

那么接下来只要解这三个方程即可

时间复杂度 $O(3^{11}log(\lambda))$

```
#include<bits/stdc++.h>
using namespace std;
using ll=__int128;
ll Lcm(ll c,ll x){
    if(!c) return x;
    return c/__gcd(x,c)*x;
}
ll exgcd(ll a,ll b,ll &x,ll &y){
    if(!b){
        x=1;y=0;
        return a;
    }
    ll x1,y1,d;
    d=exgcd(b,a%b,x1,y1);
    x=y1;y=x1-a/b*y1;
```

```
    return d;
}
const int N=25;
ll pr[N];
int cnt[N];
int top;
ll ans;
ll res[3];
void dfs(int id){
    if(id==top){
        ll x,y;
        exgcd(res[0],res[1],x,y);
        ll k1=-x/res[1],k2=y/res[0];
        while(x+res[1]*k1>=0) k1--;
        while(y-res[0]*k2<=0) k2--;
        k1=min(k1,k2);
        ll c=res[0]*(-x)+y*res[1];
        ll a=2ll*res[0]*res[1],b=res[2];
        ll x1,y1;
        ll d=exgcd(b,a,x1,y1);
        if(c%d) return;
        x1*=c/d;y1*=c/d;
        ll b1=b/d,a1=a/d;
        ll k3=-x1/a1;
        ll k4=(y1-k1)/b1;
        while(y1-k4*b1>k1) k4++;
        while(x1+k3*a1<=0) k3++;
        k1=y1-max(k4,k3)*b1;
        ans=min(ans,-(x+res[1]*k1)*res[0]);
        return;
    }
    for(int i=0;i<3;i++)
    {
        res[i]=res[i]*pr[id];
        dfs(id+1);
        res[i]=res[i]/pr[id];
    }
}

long long x[N];
void solve(){
    int n;
    cin>>n;
    ll lc=0;
    for(int i=1;i<=n;i++){
        cin>>x[i];
        lc=Lcm(lc,x[i]);
    }

    top=0;
    lc*=6;
    ans=lc;
    for(int i=2;1ll*i*i<=lc;i++){
        if(lc%i==0){
            pr[top]=1;
```

```
            cnt[top]=0;
            while(lc%i==0){
                lc/=i;cnt[top]++;
                pr[top]*=i;
            }
            top++;
        }
    }
    if(lc>1){
        pr[top]=lc;
        cnt[top]=1;
        top++;
    }
    res[0]=res[1]=res[2]=1;
    dfs(0);
    cout<<(long long)ans<<'\n';
}


int main(){
    ios::sync_with_stdio(false);cin.tie(0);cout.tie(0);
    int t=1;
    while(t--){
        solve();
    }
    return 0;
}
```

B 前进者

出题人:梅子酒

考察点:并查集,STL容器,线段树

题意

图 G 有 n 个点，权值为 v_i，并逐步给出权值 w_i 递增的 m 条道路。每次给出新道路后求出从任意一点出发，遍历该连通块后的最大值。每次新增道路后，第一次通过城市可以获得价值 v_i，第一次经过道路需要付出价值 w_i。

思路

考虑到题目所说的无论从任意点出发都要遍历一整个连通块，并且点权和边权都只计算一次，**所以可以知道一个连通块的价值和遍历该连通块的路径无关，只和连通块本身的点权和，边权和有关。**可以用带权并查集来维护一个连通块的点权和。逐步给出边并且边权是递增的，这恰好就是一个最小生成树的过程， 同样可以用并查集维护，将边权和视为负数和点权和一起维护。

这样我们可以用并查集近似 $O(n)$ 的时间复杂度维护任意连通块的点权和，边权和。考虑如何维护其中的最大值，有很多方法。其中标程给出的方法是线段树，维护的就是上述的点权和，边权和，只需要单点修改，最终递归回根节点取最大值后，根节点的价值就是答案。还有很多其他方法，很多STL容器都可以维护，例如`priority_queue`，`multiset` 等，可以自行了解。其中无论是线段树还是STL容器维护该数据的时间复杂度都是 $O(nlogn)$，线段树的常数更加优秀一些，总时间复杂度为 $O(n + nlogn)$。注意答案可能是负数，所以极值的设置要合适。

代码

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define ls p << 1
#define rs p << 1 | 1
const int N = 1e5 + 10;
const ll INF = 1e18; // 极大值
int n, m, p[N];
ll v[N], c[N]; // v_i 点权和，c_i 边权和
struct seg{
    int l, r;
    ll val; // 一个连通块的点权和 - 边权和
}tr[N * 4];
void pushup(int p){
    tr[p].val = max(tr[ls].val, tr[rs].val); // 将答案向上更新
}
void build(int p, int l, int r){
    tr[p] = {l, r, 0};
    if(l == r){
        tr[p].val = v[l] - c[l]; // 维护
        return ;
    }
    int mid = (l + r) >> 1;
```

```cpp
    build(ls, l, mid); build(rs, mid + 1, r);
    pushup(p);
}
void update(int p, int loc, ll k){ // 单点修改
    if(tr[p].l == tr[p].r){
        tr[p].val = k;
        return ;
    }
    int mid = tr[ls].r;
    if(loc <= mid) update(ls, loc, k);
    else update(rs, loc, k);
    pushup(p);
}

int find(int x){ // 找到祖先
    if(p[x] != x) p[x] = find(p[x]);
    return p[x];
}
void merge(int x, int y, int w){
    int fx = find(x), fy = find(y);
    if(fx != fy){ // 如果之前不连通，就需要合并
        p[fx] = fy;
        v[fy] += v[fx]; // 点权和相加
        c[fy] += c[fx] + w; // 边权和相加
        v[fx] = c[fx] = 0; // 将之前的置为 0
        update(1, fy, v[fy] - c[fy]); // 更新
        update(1, fx, -INF); // 将原来连通块置为无穷小
    }
}
int main(){
    ios::sync_with_stdio(false);
    cin.tie(0); cout.tie(0);

    cin >> n >> m;
    for(int i = 1; i <= n; i ++) cin >> v[i];

    for(int i = 1; i <= n; i ++) p[i] = i;

    build(1, 1, n);
    for(int i = 1; i <= m; i ++){
        int u, v, w;
        cin >> u >> v >> w;
        merge(u, v, w);
        cout << tr[1].val << "\n"; // 直接输出根节点的答案
    }
    return 0;
}
```

C 传递者

出题人:梅子酒

考察点:思维

题意

有 n 个人围坐，每个人可以选择左右相邻的其中一人，不能不选，也不能互相选（若 a 选了 b，b 就不能选 a）。

思路

这是一个埋了坑的诈骗题，首先一个人必须要选择左右，假定 i 选择了 $i+1$，那么由于不能互相选 $i+1$ 就只能选 $i+2$，以此类推直到 $i-1$ 选 i 形成一个环（以上编号都要对 n 取模）。那么反过来可以再形成一个圈一种新的选法。所以无论 n 有多大所有的选法最多只有 2 种，并不需要对 998244353 取模，样例和时限当然也是用来误导你的。

下面来说说坑点:

$n=1$ 时，只有一个人，只能自己选自己，与不能互相选矛盾，方案数为 0。

$n=2$ 时，只有两个人，$0,1$ 只能互相选，与不能互相选矛盾，方案数为 0。

代码

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

#define ll long long

int main(){
    ios::sync_with_stdio(false);
    cin.tie(0); cout.tie(0);

    ll n;
    cin >> n;
    if(n <= 2) cout << "0\n";
    else cout << "2\n";
    return 0;
}
```

D 求生者

出题人:梅子酒

考察点:概率DP

题意

每次给定僵尸血量 h_i，和离玉米投手的距离 x_i，玉米投手可能投出玉米粒（伤害 20）和黄油（伤害 40，定身 4 秒，可以叠加定身时间），投出的黄油的概率随僵尸的靠近提高。问僵尸靠近玉米投手之前就被击杀的概率。数据范围为：$h_i, x_i(1 \le h_i \le 4*10^3, 1 \le x_i \le 10^9)$。

思路

这是一个概率DP，我们从需要维护的状态来考虑。

1. 僵尸离玉米投手的距离，因为投出的黄油会随着这个数据变化所以一定要维护，并且这也是结束状态的一种（僵尸被击杀前就靠近玉米投手）。
2. 僵尸的血量，结束状态之一（靠近之前就被击杀）。
3. 僵尸被定住的状态，我们这里取被定住的时间来维护。

于是状态就知道了，第一维僵尸的剩余血量 h_i，第二维僵尸的位置 x_i（或者僵尸已经前进的距离），第三维被定住的时间 t_i，这样的状态意义就是 $f[i][j][k]$：当僵尸血量还剩下 i 的血量，已前进了 j 格，还剩下 k 秒定身时间的概率。但是从数据范围来看，我们的空间需要开到 $4 \times 10^3 \times 10^9 \times 300 = 1.2 \times 10^{15}$，时间复杂度也是这个量级，时空复杂度全面爆炸，我们来考虑如何优化。

我们注意到僵尸的血量只有 4×10^3，就算只投出玉米粒也只需要 $\frac{4 \times 10^3}{20} = 200$ 次就可以击杀，也就是说僵尸最多前进 200 格就一定会被击杀，所以不用考虑距离太远的僵尸的情况，第二维距离 10^9 的量级就降低到 200。
同样我们也注意到一次伤害至少也是 20 点，没必要将血量精确到个位数，我们可以将血量换算成最少需要几次玉米粒才能击杀（黄油的伤害相当于两个玉米粒），于是我们将第一维血量 10^3 的量级降低到 200。
而定身时间，最多 100 次黄油就一定能击杀任何僵尸，每次投出黄油到下一次投出黄油之间至少间隔一秒，所以定身时间最多为 $100 \times 4 - 100 = 300$。 于是时空复杂度就降低到 $200 \times 200 \times 300 = 1.2 \times 10^7$，这是我们可以接受的一个时间复杂度。

并且我们存在僵尸血量过低，或距离过远的情况，数据不一定能跑满，10 组极限数据的情况下也最多是 1.2×10^8 的量级，满足 1 秒的时间限制。具体如何转移，看代码和注释理解。

代码

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

#define ll long long

const int N = 210, mod = 998244353;

ll ksm(ll a, ll b){
```

```
    ll res = 1;
    while(b){
        if(b & 1) res = res * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return res;
}

ll f[N][N][310]; // 当僵尸血量还剩余需要 i 次攻击，已经前进 j 格，还剩下 k 秒的定身时间的概率
void add(ll& x, ll y){
    x = (x + mod + y) % mod;
}

void solve(){
    int h, x;
    cin >> h >> x;

    h = (h + 19) / 20; // 转变成至少需要的玉米粒攻击的次数

    if(h <= x){
        cout << "1\n";
        return ;
    }

    memset(f, 0, sizeof f);
    f[h][0][0] = 1;
    ll inv = ksm(100, mod - 2); // 100 的乘法逆元

    for(int i = h; i > 0; i --){
        for(int j = 0; j < x; j ++){
            for(int k = 0; k <= 300; k ++){
                if(!f[i][j][k]) continue ;
                // 概率
                ll p = min(99LL, (25LL + j / 5)) * inv % mod, q = (1 + mod - p) % mod; // 投出黄油和玉米粒的概率

                // 投出黄油的转移方程
                add(f[max(0, i - 2)][j][k + 3], f[i][j][k] * p % mod);

                // 投出玉米粒的方程转移
                if(k) add(f[i - 1][j][k - 1], f[i][j][k] * q % mod); // 减少一秒定身时间
                else add(f[i - 1][j + 1][k], f[i][j][k] * q % mod); // 没有被定身，前进一格
            }
        }
    }

    ll ans = 0;
    for(int j = 0; j <= 200; j ++){
        for(int k = 0; k <= 300; k ++){
            ans = (ans + f[0][j][k]) % mod;
        }
```

```
    }
    cout << ans << "\n";
}

int main(){
    ios::sync_with_stdio(false);
    cin.tie(0); cout.tie(0);

    int n;
    cin >> n;
    while(n --){
        solve();
    }
    return 0;
}
```

E 干饭豪的不定方程

出题人:爆哥

考察点:暴力枚举

对于 $n <= 10^9, a, t, b, k \geq 2$ 那么我们可以发现单暴力枚举 x^y 这种形式不会有很多,所以我们可以先预处理出来这些情况方案之后，在对于每个询问，枚举 b^k 来看之前多少种 $a^t = n - b^k$ 情况

```
/*
干饭豪的a^k+b^t=n
数据未更新
*/
/*
    有多少个四元组(a,t,b,k)满足下面式子
    a^t+b^k=n
    且a,b,t,k>=2
    n<=1e9
*/
#include<bits/stdc++.h>
using namespace std;

void solve(){
    int n;
    cin>>n;
    long long ans=0;
    map<long long,int>mp;
    mp.clear();

    for(int a=2;;a++){
        long long c=1ll*a*a;
        if(c>=n) break;
        for(int t=2;c<n;t++,c=c*a){
            mp[c]++;
        }
    }
    for(int a=2;;a++){
        long long c=1ll*a*a;
        if(c>=n) break;
        for(int t=2;c<n;t++,c=c*a){
            if(mp.count(n-c)){
                ans+=mp[n-c];
            }
        }
    }
    cout<<ans<<'\n';
}
int main(){
    ios::sync_with_stdio(false);cin.tie(0);cout.tie(0);
    int t=1;
    cin>>t;
    while(t--){
        solve();
```

```
    }
    return 0;
}
/*
*/
```

F 狗头豪的数列求和

出题人:爆哥

考察点:数学推导,矩阵快速幂

本题需要求 $1^k+2^k+...+n^k\ (n\le 10^{18})$

 发现 n 可能会很大不能正常的通过遍历来求和,那么我们来思考每一位

定义 $sum(x,k)$ 为前 x 项 k 次方的和

首先可以得出 $sum(x,k)=sum(x-1,k)+x^k\ (x\ge 2)$

发现 $sum(x,k)$ 是用一个递推关系的，那么思考 $(x-1)^k$ 和 x^k 会有什么关系

可以通过高中学过的二项式展开得到 $(x-1)^k=\sum_{i=0}^{k}C_k^i x^i(-1)^{k-i}$

得出 $x^k=(x-1)^k-\sum_{i=0}^{k-1}x^i(-1)^{k-i}$

那么我们发现这显然是一个线性递推关系的

$k<=10$ 那么我们可以维护一个矩阵的信息 分别处理 $1,x^1,x^2,...,x^k,sum(x,k)$ 的递推联系

便可以通过矩阵快速幂来解决这道题

时间复杂度 $O(k^3log(n))$

当然，如果你知道拉格朗日插值法，那么这个题就变的很裸，不过要注意这个题模数不一定会有逆元，就看你通过什么方式来处理了。

```
#include<bits/stdc++.h>
using namespace std;
const int N=15;
using ll=long long;
int k,mod;
long long n;
struct mat{
    int m;
    ll mp[N][N];
    mat(){
        memset(mp,0,sizeof(mp));
    }
    void re(){
        for(int i=0;i<m;i++){
            mp[i][i]=1;
        }
    }
    mat operator *(const mat &w)const{
        mat res;
        res.m=m;
        for(int k=0;k<m;k++){
            for(int i=0;i<m;i++){
                for(int j=0;j<m;j++){
                    res.mp[i][j]+=mp[i][k]*w.mp[k][j]%mod;
```

```
                        res.mp[i][j]%=mod;
                    }
                }
            }
            return res;
        }
};

mat ksm(mat a,long long b){
    mat res;
    res.m=a.m;
    res.re();
    while(b){
        if(b&1) res=res*a;
        a=a*a;
        b>>=1;
    }
    return res;
}

ll c[N][N];

void solve(){
    cin>>n>>k>>mod;
    int m=k+2;
    for(int i=0;i<=m;i++){
        for(int j=0;j<=i;j++){
            if(i==j||j==0){
                c[i][j]=1;
            }
            else{
                c[i][j]=c[i-1][j-1]+c[i-1][j];
                if(c[i][j]>=mod) c[i][j]-=mod;
            }
        }
    }
    mat a;
    a.m=m;
    for(int j=m-1;j>=1;j--){
        a.mp[j][j]=1;
        int k1=k-j+1;
        for(int i=j+1,k2=k1-1,f=1;i<m;i++,k2--,f=mod-f){
            for(int p=i;p<m;p++){
                a.mp[p][j]+=f*c[k1][k2]%mod*a.mp[p][i];
                a.mp[p][j]%=mod;
            }
        }
    }
    a.mp[0][0]=1;
    for(int i=1;i<m;i++){
        a.mp[i][0]=a.mp[i][1];
    }
    mat res;
    res.m=m;
    for(int j=0;j<m;j++){
```

```
        res.mp[0][j]=1;
    }
    res=res*ksm(a,n-1);
    cout<<res.mp[0][0]<<'\n';
}

int main(){
    ios::sync_with_stdio(false);cin.tie(0);cout.tie(0);
    int t=1;
    while(t--){
        solve();
    }
    return 0;
}
```

G 草莓豪的斐波那契

出题人:爆哥

考察点:思维

可以通过观察斐波那契序列值是指数形势增长，所以直接模拟即可

```
#include<bits/stdc++.h>
using namespace std;
void solve(){
    long long n;
    cin>>n;
    long long now=0;
    long long a=1,b=1;
    int t=0;
    while(now<n){
        now+=b;
        t++;
        a=a+b;
        swap(a,b);
    }
    cout<<t<<'\n';
}
int main(){
    ios::sync_with_stdio(false);cin.tie(0);cout.tie(0);
    int t=1;
    while(t--){
        solve();
    }
    return 0;
}
```

H [小刻觉得应该来个平衡树]

出题人:buns out

考察点:STL容器,pb_ds容器,线段树,二分

> **宣传一下我的模板库:**
>
> [Release ACM模板 · zxyDEDIRE/ACM-Code-Library (github.com)]
>
> [zxyDEDIRE/ACM-Code-Library at QAQ (github.com)]
>
> 有LaTeX源码也有PDF版本

解法非常非常多，STL的各种容器都可以，pb_ds里边的容器也都可以，或者上平衡树，或者线段树+二分。

大家的解法也都分成丰富，很值得夸奖。虽说有人搜到了pb_ds库里边的平衡树，不过记得及时掌握，pb_ds库里边也有很多有趣的小东西。也有人直接手抄了Sply平衡树，码量确实很大，sply等等平衡树并不建议现在学，STL以及pb_ds大部分容器都掌握才是最有价值的。

不过可恶的是，居然有人用vector卡过去了，没有想到vector insert这么快，没有专门造数据卡。

解法一：
直接调用pb_ds的平衡树，或者直接上平衡树(平衡树代码就不贴出来了，太长了)。

pb_ds只要会用就可以了。

```
#include <ext/pb_ds/assoc_container.hpp>
#include<bits/stdc++.h>
using namespace __gnu_cxx;
using namespace __gnu_pbds;
using namespace std;
#define endl "\n"
using pii=pair<int,int>;
const int INF=1e9+7;

typedef tree<pii, null_type, less<pii>, rb_tree_tag,
tree_order_statistics_node_update> Tree;
Tree t;


template<typename T>
int getRank(T x)
{
    return t.order_of_key(x)+1;
}

template<typename T>
T getVal(int k)
{
    auto it=t.find_by_order(k-1);
```

```
    if(it!=t.end())
        return *it;
    else
        return {INF,0};
}
int get(int x){
    auto v=getVal<pii>(x);
    return v.first;
}

void solve()
{
    int n,x;
    cin>>n;
    for(int i=1;i<=n;i++)
    {
        cin>>x;
        t.insert({x,i});
        int mid_l = ceil(1.0*i/3);
        int mid_r = ceil(2.0*i/3);
        cout<<get(mid_l)<<" "<<get(mid_r)<<endl;
    }
}
signed main(){
    ios::sync_with_stdio(false);
    cin.tie(nullptr);cout.tie(nullptr);
        solve();
    return 0;
}
```

解法二:
各种容器，我用的是multiset。

第一个容器维护中位数之前的数据，第二个容器维护中位数之后的数据。如果说第一个容器中最大的数大于 第二个容器中最小的数，那么将第一个容器中最大的数移动到第二个容器。

```
#include<bits/stdc++.h>
using namespace std;
#define endl "\n"
#define pp(x) array<int,x>
using ull=unsigned long long;
using ll=long long;
using pii=pair<int,int>;
using pdd=pair<double,double>;
const int dx[]={0,0,1,-1,1,-1,1,-1};
const int dy[]={1,-1,0,0,1,-1,-1,1};
const int mod=998244353;
const int inf=0x3f3f3f3f;
const int INF=1e9+7;
const int maxn=1e6+100;
struct node{
    multiset<int,greater<int>>sa;
    multiset<int>sb;
    int lena=0,lenb=0;
    void insert(int x){
```

```
        sa.insert(x);
        lena++;
        bal();
    }
    void bal(){
        while(sa.size()&&sb.size()&&(*sa.begin())>(*sb.begin())){
            sb.insert(*sa.begin());
            sa.erase(sa.begin());
            lena--;
            lenb++;
        }
    }
    void out(int x){
        while(lena<x){
            sa.insert(*sb.begin());
            sb.erase(sb.begin());
            lena++;
            lenb--;
        }
        while(lena>x)
        {
            sb.insert(*sa.begin());
            sa.erase(sa.begin());
            lena--;
            lenb++;
        }
        cout<<*sa.begin()<<" ";
    }
}sa,sb;
int n;
void solve()
{
    cin>>n;
    int len=0;
    while(n--)
    {
        int x;
        cin>>x;
        len++;
        int mid_1=ceil(1.0*len/3);
        int mid_2=ceil(2.0*len/3);
        sa.insert(x);
        sb.insert(x);
        sa.out(mid_1);
        sb.out(mid_2);
        cout<<endl;
    }
}
signed main(){
    ios::sync_with_stdio(false);
    cin.tie(nullptr);cout.tie(nullptr);
        solve();
    return 0;
}
/*
```

```
*/
```

解法三:
离散化+线段树+二分，也有人用这种方法过了，NICE。

这种做法具体就是离散化后，建一棵线段树，线段树维护从小到大各个值的出现次数。然后就可以二分，找到具体是哪一个值。

```
#include<bits/stdc++.h>
using namespace std;
#define endl "\n"
const int maxn=1e6+100;
struct Seg{
    int l,r,val;
    Seg operator+(const Seg&a)const{
        Seg now;
        now.l=l;now.r=a.r;
        now.val=val+a.val;
        return now;
    }
}t[maxn<<2];
int p[maxn];
int n,len;
void build(int rt,int l,int r){
    t[rt]={l,r,0};
    if(l==r)return;
    int mid=(l+r)>>1;
    build(rt<<1,l,mid);
    build(rt<<1|1,mid+1,r);
}
void add(int rt,int l){
    if(t[rt].l==t[rt].r){
        t[rt].val++;
        return ;
    }
    int mid=(t[rt].l+t[rt].r)>>1;
    if(l<=mid)add(rt<<1,l);
    else add(rt<<1|1,l);
    t[rt]=t[rt<<1]+t[rt<<1|1];
}
int query(int rt,int l,int r){
    if(l<=t[rt].l&&t[rt].r<=r)
        return t[rt].val;
    int mid=(t[rt].l+t[rt].r)>>1;
    int ans=0;
    if(l<=mid)ans+=query(rt<<1,l,r);
    if(r>mid)ans+=query(rt<<1|1,l,r);
    return ans;
}
int q(int x)
{
    int l=1,r=len,ans=r;
    auto check=[&](int mid)->bool{
        return query(1,1,mid)>=x;
    };
```

```
    while(l<r-1){
        int mid=(l+r)>>1;
        if(check(mid))ans=mid,r=mid;
        else l=mid;
    }
    while(ans>1&&check(ans-1))ans--;
    return ans;
}
void solve()
{
    cin>>n;
    vector<int>v(n);
    for(int i=0;i<n;i++)
        cin>>p[i],v[i]=p[i];
    sort(v.begin(),v.end());
    v.erase(unique(v.begin(),v.end()),v.end());
    len=v.size();
    build(1,1,len);
    for(int i=0;i<n;i++)
    {
        int x=p[i];
        x=lower_bound(v.begin(),v.end(),x)-v.begin()+1;
        add(1,x);
        int mid_1=ceil(1.0*(i+1)/3);
        int mid_2=ceil(2.0*(i+1)/3);
        int pos1=q(mid_1)-1;
        int pos2=q(mid_2)-1;
        cout<<v[pos1]<<" "<<v[pos2]<<"\n";
    }
}
signed main(){
    ios::sync_with_stdio(false);
    cin.tie(nullptr);cout.tie(nullptr);
        solve();
    return 0;
}
```

I [小刻的DP题]()

出题人:buns out

考察点:思维,状压,最短路,bfs

> **宣传一下我的模板库:**
>
> [Release ACM模板 · zxyDEDIRE/ACM-Code-Library (github.com)]()
>
> [zxyDEDIRE/ACM-Code-Library at QAQ (github.com)]()
>
> 有LaTeX源码也有PDF版本

非常经典的最短路问题，状压最短路，此题也有非常非常多的解法，时限放到了 $5s$,除了非常非常暴力的做法，一般都可以过。但是过的人有点少，是大家没有很研究图论?

基本的思想就是状压维护最短路，因为 k 只有 8 ，因此可以用二进制状态压缩维护 8 个二进制位，如果说此二进位上是 1 ，那么说明走过的路包含此位的佩洛，这个是主要思想。

那么接下来询问 q 次，可以先预处理出所有的情况，或者说每次都跑一遍 DIJ。

预处理的话 dij,bfs 都是可以的，但是每次询问每次跑一边的话，最好还是用 dij，不然会T。

```
#include<bits/stdc++.h>
using namespace std;
#define endl "\n"
using ll=long long;
const ll INF=1e18+7;
const int maxn=1e6+100;
const int N=1e5+100;
struct node{
    ll id,op,dis;
    bool operator<(const node&a)const{
        return a.dis<dis;
    }
};
vector<array<ll,3>>v[maxn];
ll dis[N][550];
bool vis[N][550];
int n,m,k,s,t,q;
void dij()
{
    for(int i=1;i<=n;i++)
        for(int j=0;j<=(1<<(k+1));j++)
            dis[i][j]=INF;
    dis[s][0]=0;
    // priority_queue<node>q;
    queue<node>q;
    q.push({s,0,0});
    while(!q.empty())
    {
        // auto [x,fl,DIS]=q.top();q.pop();
```

```
        auto [x,fl,DIS]=q.front();q.pop();
        // cout<<x<<" "<<fl<<" "<<DIS<<endl;
        if(DIS>dis[x][fl])continue;
        for(auto [y,w,_op]:v[x])
        {
            int op=(_op|fl);
            if(dis[y][op]>dis[x][fl]+w)
            {
                dis[y][op]=dis[x][fl]+w;
                q.push({y,op,dis[y][op]});
            }
        }
    }
}
void solve()
{
    cin>>n>>m>>k>>s>>t;
    for(int i=1;i<=m;i++)
    {
        int op,x,y,w;
        cin>>x>>y>>w>>op;
        op--;
        v[x].push_back({y,w,1<<op});
        v[y].push_back({x,w,1<<op});
    }
    dij();
    // for(int i=0;i<(1<<k);i++)
    //  for(int j=0;j<(1<<k);j++)
    //      if(i!=j&&(i&j)==i){
    //          for(int k=1;k<=n;k++)
    //              dis[k][j]=min(dis[k][i],dis[k][j]);
    //      }
    cin>>q;
    while(q--)
    {
        int num,fl=0,x;
        cin>>num;
        while(num--){
            cin>>x;
            x--;
            fl|=(1<<x);
        }
        ll ans=INF;
        for(int i=0;i<=(1<<k);i++)
            if((i&fl)==0)
                ans=min(ans,dis[t][i]);

        if(ans==INF)cout<<-1<<endl;
        else cout<<ans<<endl;
    }
}
signed main(){
    ios::sync_with_stdio(false);
    cin.tie(nullptr);cout.tie(nullptr);
        solve();
```

```
    return 0;
}
/*
*/
```

J [小刻的画图写话]

出题人:buns out

考察点:码力,函数

> **宣传一下我的模板库:**
>
> [Release ACM模板 · zxyDEDIRE/ACM-Code-Library (github.com)]
>
> [zxyDEDIRE/ACM-Code-Library at QAQ (github.com)]
>
> 有LaTeX源码也有PDF版本

解法一:

最方便的解法就是函数递归，翻转操作就是传入的参数异或 1，由大的图形递归到小的图形。

首先传入的四个参数就是当前正方形的坐标,左上角 $(x1, y1)$,右下角 $(x2, y2)$ ，参数 op 就是是否进行翻转操作。

```
#include<bits/stdc++.h>
using namespace std;
#define endl "\n"
const int N=3111;
char mp[N][N];
int n;
void dfs(int x1,int y1,int x2,int y2,int op)
{
    if(x1==x2&&y1==y2){
        mp[x1][y1]=('0'+op);
        return ;
    }
    int mid_x=(x1+x2)>>1;
    int mid_y=(y1+y2)>>1;
    dfs(x1,y1,mid_x,mid_y,op);
    dfs(x1,mid_y+1,mid_x,y2,op^1);
    dfs(mid_x+1,y1,x2,mid_y,op^1);
    dfs(mid_x+1,mid_y+1,x2,y2,op);
}
void solve()
{
    cin>>n;
    n=(1ll<<n);
    dfs(1,1,n,n,1);
    for(int i=1;i<=n;i++){
        for(int j=1;j<=n;j++)
            cout<<mp[i][j];
        cout<<endl;
    }
}
signed main(){
    ios::sync_with_stdio(false);
```

```
    cin.tie(nullptr);cout.tie(nullptr);
        solve();
    return 0;
}
/*

https://polygon.codeforces.com/plain-answer/answer-10.txt?
testset=tests&index=10&session=e40d2be16baa1fbcac4ac4ce94a88eb0147bd4b7&ccid=1ea
5ccabf3e3ad2d489fda4e0e37f208
https://dwz.cn/OwZR2vmA

*/
```

解法二：
先画出小图形再生成次大图形，代码较为麻烦。

```
#include<bits/stdc++.h>
using namespace std;
const int N=5222;
int mp[N][N];
int n;
int main(){
    ios::sync_with_stdio(false);
    cin.tie(0);cout.tie(0);
    cin>>n;
    mp[1][1]=1;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= (1 << (i - 1)); j++) {
            for (int k = 1; k <= (1 << (i - 1)); k++) {
                mp[j + (1 << (i - 1))][k + (1 << (i - 1))] = mp[j][k];
                mp[j][k + (1 << (i - 1))] = 1 - mp[j][k];
                mp[j + (1 << (i - 1))][k] = 1 - mp[j][k];
            }
        }
    }
    for(int i=1;i<=(1<<n);i++){
        for(int j=1;j<=(1<<n);j++)
            cout<<mp[i][j];
        cout<<endl;
    }
    return 0;
}
```