赏金猎人

GsjzTle

July 17, 2021

Contents

1 模拟

1.1 进制转换

```
int num[123];
string change(int n , int m , string s)
{
    int res = 0;
    string ans = "";
    for(auto i : s) res = res * n + num[i];
    while(res)
    {
        int x = res % m;
        if(x >= 10 && x <= 15) ans += 'A' + x - 10;
        else ans += x + '0';
        res /= m;
    }
    reverse(ans.begin() , ans.end());
    return ans;
}
signed main()
{
    for(int i = 0 ; i <= 9 ; i ++) num[i + '0'] = i;
    for(int i = 0 ; i <= 5 ; i ++) num[i + 'A'] = 10 + i;
    int n , m ;
    string s;
    cin >> n >> s >> m;
    cout << change(n , m , s) << '\n';
    return 0;
}
```

2 搜索

2.1 总结

1. 如果可以用 dfs 写记忆化搜索，复杂度比 bfs 暴搜更小。

2.2 三分

2.2.1 整数、浮点数

整数。开口向上的例子。

```
ll l = 0 , r = 1e9 ;
ll ans = 0 ;
while(l <= r)
{
    ll lmid = (2 * l + r) / 3 ;
    ll rmid = (2 * r + l + 2) / 3 ;
    if(cal(lmid) <= cal(rmid)) ans = lmid , r = rmid - 1 ;
    else ans = rmid , l = lmid + 1 ;
}
```

浮点数。开口向下的例子。

```
while(r - l > eps)
{
    double lmid = l + (r - l) / 3 ;
    double rmid = r - (r - l) / 3 ;
    if(cal(lmid) <= cal(rmid)) l = lmid ;
    else r = rmid ;
}
printf("%.5f\n" , l) ;
```

2.2.2 求极大值

```
double check(double x)
{
    double sum = 0;
    rep(i , 1 , n)
    sum = sum * x + a[i];
    return sum ;
}
while(r - l > esp)
{
    double lmid = l + (r - l) / 3;
    double rmid = r - (r - l) / 3;
    if(check(lmid) > check(rmid)) r = rmid;
    else if(check(lmid) < check(rmid)) l = lmid;
    else l = lmid , r = rmid;
}
cout << setprecision(5) << fixed << l << '\n' ;
```

2.3 折半搜索

2.3.1 总结

折半搜索是暴搜的一种优化，类似于分成两部分进行状压，然后再通过各种方法把两部分组合起来计算对答案的贡献。这其中有许多技巧。

1. 考虑清楚分割前后两部分的分界线。一般来说需要状压后两部分集合的大小接近。
2. 二分答案 + 双指针扫描两部分。

2.4 模拟退火

求费马点的例子，注意用真随机数。

```
void solve()
{
    int i ;
    point ans , next ;
    point now ;
    ans.x = (p[1].x + p[2].x + p[3].x) / 3 ;
    ans.y = (p[1].y + p[2].y + p[3].y) / 3 ;
    double t = 100 ;
    double delta = 0.99 ;
```

```
    double dE ;
    double ans1 ;
    double inf = 1000000000 ;
    while(t > 1e-8)
    {
        ans1 = inf ;
        for(i = 0 ; i < 4 ; i ++)
        {
            next.x = ans.x + row[i] * t ;
            next.y = ans.y + col[i] * t ;
            if(cal(next) < ans1)
            {
                ans1 = cal(next) ;
                now = next ;
            }
        }
        dE = cal(now) - cal(ans) ;
        if(dE < 0.0 || exp(dE / t) > random1())
        {
            ans = now ;
        }
        t *= delta ;
    }
    printf("%.12f %.12f" , ans.x , ans.y) ;
}
```

3 贪心

3.1 加工生产调度

1. $a1 < b1$ 且 $a2 < b2$ 时，按 $a1 < a2$ 排序
2. $a1 = b1$ 且 $a2 = b2$ 时，如何排序对结果没有影响
3. $a1 > b1$ 且 $a2 > b2$ 时，按 $b2 < b1$ 排序

- 开两个 $vector$，一个存 $a < b$ $(vec1)$，另一个存 $a > b$ $(vec2)$，先处理 $vec1$ 在处理 $vec2$

```
const int N = 3e5 + 10;
struct Edge{
    int nex , to;
}edge[N << 1];
int head[N] , TOT;
void add_edge(int u , int v)
{
    edge[++ TOT].nex = head[u] ;
    edge[TOT].to = v;
    head[u] = TOT;
}
struct node{
    int a , b , id;
}p[N] , q[N];
vector<node>vec1 , vec2;
```

```cpp
bool cmp1(node x , node y)
{
    return x.a < y.a;
}
bool cmp2(node x , node y)
{
    return x.b > y.b;
}
signed main()
{
    int n , m;
    cin >> n;
    rep(i , 1 , n) cin >> p[i].a;
    rep(i , 1 , n) cin >> p[i].b , p[i].id = i;
    rep(i , 1 , n)
    {
        if(p[i].a <= p[i].b) vec1.pb(p[i]);
        else vec2.pb(p[i]);
    }
    sort(all(vec1) , cmp1) , sort(all(vec2) , cmp2);
    int cnt = 0;
    for(auto j : vec1) q[++ cnt] = j;
    for(auto j : vec2) q[++ cnt] = j;
    int ta = q[1].a , tb = q[1].a + q[1].b;
    rep(i , 2 , n)
    {
        ta += q[i].a;
        tb = max(ta , tb) + q[i].b;
    }
    cout << tb << '\n';
    rep(i , 1 , n) cout << q[i].id << " ";
    cout << '\n';
    return 0;
}
```

3.2 反悔贪心

- 选择一个第 i 数之后就不能选择 $i+1, i-1$，求可以获得的最大值?
- 增加类似**退流思想（雾）**的反悔机制
- 将所有数都放入优先队列，当选择 i 之后给它加个反悔机制，即把 $a[i+1]+a[i-1]-a[i]$ 作为一个新的数插到优先队列里，其含义为选择 $i+1, i-1$ 而不选 i。

```cpp
const int N = 1e5 + 10;
int n , k , l[N] , r[N] , b[N] , vis[N];
struct node{
    int id , val;
    bool operator < (const node & b) const {
        return val > b.val;
    }
}a[N];
priority_queue<node>que;
```

```cpp
signed main()
{
    int cnt = 0;
    cin >> n >> k;
    for(int i = 1 ; i <= n ; i ++) cin >> b[i];
    for(int i = 2 ; i <= n ; i ++)
    {
        a[++ cnt].id = i - 1;
        a[cnt].val = b[i] - b[i - 1];
        l[i - 1] = i - 2;
        r[i - 1] = i;
        que.push(a[cnt]);
    }
    a[0].val = a[cnt + 1].val = 0x3f3f3f3f;
    long long res = 0;
    while(!que.empty() && k)
    {
        node u = que.top();
        que.pop();
        if(vis[u.id]) continue ;
        k -- ;
        int id = u.id , val = a[id].val , L = l[id] , R = r[id];
        res += val;
        vis[L] = vis[R] = 1;
        a[id].val = a[L].val + a[R].val - a[id].val;
        l[id] = l[L] , r[id] = r[R];
        r[l[L]] = l[r[R]] = id;
        que.push(a[id]);
    }
    cout << res << '\n';
    return 0;
}
```

4 数论

4.1 小经验

1.
   - 已知 C，求有多少对 (A,B) 满足 $A\times B=C$
   - 答案为 C 的因子个数

2.
   - 已知 C,g，求有多少对 (A,B) 满足 $\begin{cases}\gcd(A,B)=g\\A\times B=C\end{cases}$
   - 可转换式子为 $\frac{A}{g}\times\frac{B}{g}=\frac{C}{g^2}$，那么 $gcd(\frac{A}{g},\frac{B}{g})=1$
   - 所以需要将 $\frac{C}{g^2}$ 的某个质因子全部分给 $\frac{A}{g}$ 或者全部分给 $\frac{B}{g}$
   - 那么答案就是 $2^{P_{cnt}}$，其中 P_{cnt} 表示 $\frac{C}{g^2}$ 的质因子的个数

4.2 FWT

4.2.1 序列中选 1,2,...,n 个数的最大异或和分别为多少

- 给定长度为 N 的序列 ($N \leq 3 \times 10^5$, $0 \leq A_i < 2^{18}$).
- 问从中取 $1, 2, 3, ..., n$ 个数的最大异或和分别为多少

```cpp
#include<bits/stdc++.h>
#define rep(i , a , b) for(int i = a ; i <= b ; i ++)
using namespace std;
const int N = 3e5 + 10;
int n , x , a[N] , s[N] , ans[N];
void fwt(int *a)
{
    rep(i , 0 , 17)
    {
        for(int j = 0 ; j < (1 << 18) ; j ++)
        {
            if(((1 << i) & j) == 0)
            {
                a[j] = a[j] + a[j ^ (1 << i)];
                a[j ^ (1 << i)] = a[j] - 2 * a[j ^ (1 << i)];
            }
        }
    }
}
signed main()
{
    cin >> n;
    rep(i , 1 , n)
    {
        cin >> x;
        a[x] = 1;
    }
    fwt(a);
    s[0] = 1;
    rep(i , 1 , 19)
    {
        fwt(s);
        for(int j = 0 ; j < (1 << 18) ; j ++) s[j] *= a[j];
        fwt(s);
        for(int j = 0 ; j < (1 << 18) ; j ++) if(s[j])
        {
            s[j] = 1;
            ans[i] = j;
        }
    }
    rep(i , 20 , n) ans[i] = ans[i - 2];
    rep(i , 1 , n) cout << ans[i] << " \n"[i == n];
    return 0;
}
```

4.3 Min25 筛

4.3.1 快速阶层取模

- 求 $n!\bmod p$
- $1 \le n < p \le 2^{31} - 1$, $1 \le T \le 5$
- 复杂度 $\Theta(\sqrt{n}\log n)$

```cpp
#include <bits/stdc++.h>
#define int long long
#define ll long long
#define rep(i, a, b) for (int i = a; i <= b; i++)
#define per(i, b, a) for (int i = b; i >= a; i--)
#define go(u) for (int i = head[u], v = e[i].v; i; i = e[i].nx, v = e[i].v)
using namespace std;
const int N = (1 << 17) + 5;
int n , mod;
inline int add(int x, int y)
{
    return 0ll + x + y >= mod ? 0ll + x + y - mod : x + y;
}
inline int dec(int x, int y)
{
    return x - y < 0 ? x - y + mod : x - y;
}
inline int mul(int x, int y)
{
    return 1ll * x * y - 1ll * x * y / mod * mod;
}
int ksm(int x, int y)
{
    int res = 1;
    for (; y; y >>= 1, x = mul(x, x))
    (y & 1) ? res = mul(res, x) : 0;
    return res;
}
const double Pi = acos(-1.0);
struct cp
{
    double x, y;
    inline cp() {}
    inline cp(double xx, double yy) : x(xx), y(yy) {}
    inline cp operator+(const cp &b) const
    {
        return cp(x + b.x, y + b.y);
    }
    inline cp operator-(const cp &b) const
    {
        return cp(x - b.x, y - b.y);
    }
    inline cp operator*(const cp &b) const
    {
```

```
45          return cp(x * b.x - y * b.y, x * b.y + y * b.x);
46      }
47      inline cp operator*(const double &b) const
48      {
49          return cp(x * b, y * b);
50      }
51      inline cp operator~() const
52      {
53          return cp(x, -y);
54      }
55  } w[2][N];
56  int r[21][N], ifac[N], lg[N], inv[N];
57  double iv[21];
58  void Pre()
59  {
60      iv[0] = 1;
61      rep(Dd, 1, 17)
62      {
63          rep(i, 0, (1 << Dd) - 1) r[Dd][i] = (r[Dd][i >> 1] >> 1) | ((i & 1) << (Dd -
                1));
64          lg[1 << Dd] = Dd, iv[Dd] = iv[Dd - 1] * 0.5;
65      }
66      inv[0] = inv[1] = ifac[0] = ifac[1] = 1;
67      rep(i, 2, 131072) inv[i] = mul(mod - mod / i, inv[mod % i]), ifac[i] = mul(ifac[
            i - 1], inv[i]);
68      for (int i = 1, Dd = 0; i < 131072; i <<= 1, ++Dd)
69      rep(k, 0, i - 1)
70      w[1][i + k] = cp(cos(Pi * k * iv[Dd]), sin(Pi * k * iv[Dd])),
71      w[0][i + k] = cp(cos(Pi * k * iv[Dd]), -sin(Pi * k * iv[Dd]));
72  }
73  int lim, Dd;
74  void FFT(cp *A, int ty)
75  {
76      rep(i, 0, lim - 1) if (i < r[Dd][i]) swap(A[i], A[r[Dd][i]]);
77      cp t;
78      for (int mid = 1; mid < lim; mid <<= 1)
79      for (int j = 0; j < lim; j += (mid << 1))
80      rep(k, 0, mid - 1)
81      A[j + k + mid] = A[j + k] - (t = w[ty][mid + k] * A[j + k + mid]),
82      A[j + k] = A[j + k] + t;
83      if (!ty)
84      rep(i, 0, lim - 1) A[i] = A[i] * iv[Dd];
85  }
86  void MTT(int *a, int *b, int len, int *c)
87  {
88      static cp f[N], g[N], p[N], q[N];
89      lim = len, Dd = lg[lim];
90      rep(i, 0, len - 1) f[i] = cp(a[i] >> 16, a[i] & 65535), g[i] = cp(b[i] >> 16, b[
            i] & 65535);
91      rep(i, len, lim - 1) f[i] = g[i] = cp(0, 0);
92      FFT(f, 1), FFT(g, 1);
93      rep(i, 0, lim - 1)
94      {
```

```
95          cp t, f0, f1, g0, g1;
96          t = ~f[i ? lim - i : 0], f0 = (f[i] - t) * cp(0, -0.5), f1 = (f[i] + t) *
                0.5;
97          t = ~g[i ? lim - i : 0], g0 = (g[i] - t) * cp(0, -0.5), g1 = (g[i] + t) *
                0.5;
98          p[i] = f1 * g1, q[i] = f1 * g0 + f0 * g1 + f0 * g0 * cp(0, 1);
99      }
100     FFT(p, 0), FFT(q, 0);
101     rep(i, 0, lim - 1) c[i] = ((((ll)(p[i].x + 0.5) % mod << 16) % mod << 16) + ((ll
            )(q[i].x + 0.5) << 16) + ((ll)(q[i].y + 0.5))) % mod;
102 }
103 void calc(int *a, int *b, int n, int k)
104 {
105     static int f[N], g[N], h[N], sum[N], isum[N];
106     int len = 1;
107     while (len <= n + n) len <<= 1;
108     rep(i, 0, n) f[i] = mul(a[i], mul(ifac[i], ifac[n - i]));
109     for (int i = n - 1; i >= 0; i -= 2) f[i] = mod - f[i];
110     int t = dec(k, n);
111     rep(i, 0, n + n) g[i] = add(i, t);
112     sum[0] = g[0];
113     rep(i, 1, n + n) sum[i] = mul(sum[i - 1], g[i]);
114     isum[n + n] = ksm(sum[n + n], mod - 2);
115     per(i, n + n, 1) isum[i - 1] = mul(isum[i], g[i]);
116     rep(i, 1, n + n) g[i] = mul(isum[i], sum[i - 1]);
117     g[0] = isum[0];
118     rep(i, n + 1, len - 1) f[i] = 0;
119     rep(i, n + n + 1, len - 1) g[i] = 0;
120     MTT(f, g, len, h);
121     int res = 1, p1 = k - n, p2 = k;
122     rep(i, p1, p2) res = 1ll * res * i % mod;
123     res = dec(res, 0);
124     rep(i, 0, n) g[i] = (0ll + mod + p1 + i) % mod;
125     sum[0] = g[0];
126     rep(i, 1, n) sum[i] = mul(sum[i - 1], g[i]);
127     isum[n] = ksm(sum[n], mod - 2);
128     per(i, n, 1) isum[i - 1] = mul(isum[i], g[i]);
129     rep(i, 1, n) g[i] = mul(isum[i], sum[i - 1]);
130     g[0] = isum[0];
131     for (int i = 0; i <= n; p2 = add(p2, 1), ++i)
132     b[i] = mul(h[i + n], res), res = mul(res, mul(g[i], p2 + 1));
133 }
134 int solve(int bl)
135 {
136     static int a[N], b[N], c[N];
137     int s = 0;
138     for (int p = bl; p; p >>= 1)
139     ++s;
140     a[0] = 1, --s;
141     int qwq = ksm(bl, mod - 2);
142     for (int p = 0; s >= 0; --s)
143     {
144         if (p)
```

```
145         {
146             calc(a, b, p, p + 1);
147             rep(i, 0, p) a[p + i + 1] = b[i];
148             a[p << 1 | 1] = 0;
149             calc(a, b, p << 1, mul(p, qwq));
150             p <<= 1;
151             rep(i, 0, p) a[i] = mul(a[i], b[i]);
152         }
153         if (bl >> s & 1)
154         {
155             rep(i, 0, p) a[i] = mul(a[i], (1ll * bl * i + p + 1) % mod);
156             p |= 1, a[p] = 1;
157             rep(i, 1, p) a[p] = mul(a[p], (1ll * bl * p + i) % mod);
158         }
159     }
160     int res = 1;
161     rep(i, 0, bl - 1) res = mul(res, a[i]);
162     return res;
163 }
164 int GetFac(int n)
165 {
166     int s = sqrt(n), res = solve(s);
167     rep(i, s * s + 1, n) res = mul(res, i);
168     return res;
169 }
170 int Fac(int n)
171 {
172     Pre();
173     if (n > mod - 1 - n)
174     {
175         int res = ksm(GetFac(mod - 1 - n), mod - 2);
176         return (mod - 1 - n) & 1 ? res : mod - res;
177     }
178     return GetFac(n);
179 }
180 signed main()
181 {
182     ios::sync_with_stdio(false);
183     cin.tie(0), cout.tie(0);
184     int T = 1;
185     cin >> T;
186     while (T--)
187     {
188         cin >> n >> mod;
189         cout << Fac(n) << '\n';
190     }
191     return 0;
192 }
```

4.4 Pohlig-Hellman

4.4.1 总结

- 求 $a^x \equiv b(mod\ \ p)$ 的最小正整数解
- $p \leq 10^{18}$ 且 p 为质数
- $p-1$ 包含的质因子较少 and 较小时比较适用
- $pohlig-hellman$ 算法的复杂度在一般情况下比 $BSGS$ 高

4.4.2 Pohlig-Hellman

```
namespace PhoRho //Phollard-Rho
{
    ll gcd(ll a, ll b)
    {
        if (b == 0) return a;
        return gcd(b, a % b);
    }
    ll fastpow(ll x, ll p, ll mod)
    {
        ll ans = 1;
        while (p)
        {
            if (p & 1) ans = (__int128) ans * x % mod;
            x = (__int128) x * x % mod;
            p >>= 1;
        }
        return ans;
    }
    ll max_factor;
    bool MillerRabin(ll x, ll b)
    {
        ll k = x - 1;
        while (k)
        {
            ll cur = fastpow(b, k, x);
            if (cur != 1 && cur != x - 1) return false;
            if ((k & 1) == 1 || cur == x - 1) return true;
            k >>= 1;
        }
        return true;
    }
    bool prime(ll x)
    {
        if (x == 4685624825598ll || x < 2) return false;
        if (x == 2 || x == 3 || x == 7 || x == 61 || x == 24251) return true;
        return MillerRabin(x, 2) && MillerRabin(x, 3) && MillerRabin(x, 7) &&
            MillerRabin(x, 61);
    }
    ll f(ll x, ll c, ll n)
    {
        return ((__int128) x * x + c) % n;
```

```
41      }
42      ll PRho(ll x)
43      {
44          ll s = 0, t = 0, c = 1ll * rand() * (x - 1) + 1;
45          int stp = 0, goal = 1;
46          ll val = 1;
47          for (goal = 1;; goal <<= 1, s = t, val = 1)
48          {
49              for (stp = 1; stp <= goal; ++stp)
50              {
51                  t = f(t, c, x);
52                  val = (__int128) val * abs(t - s) % x;
53                  if (stp % 127 == 0)
54                  {
55                      ll d = gcd(val, x);
56                      if (d > 1) return d;
57                  }
58              }
59              ll d = gcd(val, x);
60              if (d > 1) return d;
61          }
62      }
63      void fac(ll x)
64      {
65          if (x <= max_factor || x < 2)
66          {
67              return;
68          }
69          if (prime(x))
70          {
71              max_factor = max_factor > x ? max_factor : x;
72              return;
73          }
74          ll p = x;
75          while (p >= x) p = PRho(x);
76          while ((x % p == 0)) x /= p;
77          fac(x);
78          fac(p);
79      }
80      ll divide(ll n)
81      {
82          srand((unsigned) time(0));
83          max_factor = 0;
84          fac(n);
85          return max_factor;
86      }
87  }
88  namespace DLP
89  {
90      const int N = 1111111;
91      ll fastpow(ll a, ll n) //快速幂
92      {
93          ll res = 1;
```

```
94        while (n > 0)
95        {
96            if (n & 1) res = res * a;
97            a = a * a;
98            n >>= 1;
99        }
100       return res;
101   }
102   ll fastpow(ll a, ll n, ll p) //针对特别大的数的快速幂
103   {
104       ll res = 1;
105       a %= p;
106       while (n > 0)
107       {
108           if (n & 1) res = (__int128) res * a % p;
109           a = (__int128) a * a % p;
110           n >>= 1;
111       }
112       return res;
113   }
114   int prime[N], ptot;
115   bool ispr[N];
116   struct pt
117   {
118       ll p;
119       int c;
120   };
121   void getprime() //获取10^6以内的质数
122   {
123       memset(ispr, 1, sizeof(ispr));
124       for (int i = 2; i < N; ++i)
125       {
126           if (ispr[i]) prime[++ptot] = i;
127           for (int j = 1; j <= ptot && prime[j] <= (N - 1) / i; ++j)
128           {
129               ispr[i * prime[j]] = 0;
130               if (!i % prime[j]) break;
131           }
132       }
133   }
134   bool cmp(pt x, pt y)
135   {
136       return x.p < y.p;
137   }
138   void findorg(vector<pt> &v, ll num) //num分解质因数
139   {
140       while (num >= N) //大于10^6的部分，每次用Pho-Rho算法找出最大的一个质因子，然后除掉
              即可
141       {
142           ll maxf = PhoRho::divide(num);
143           int cnt = 0;
144           while (num % maxf == 0)
145           {
```

```
146             cnt++;
147             num = num / maxf;
148         }
149         v.push_back((pt)
150         {
151             maxf, cnt
152         });
153     }
154     if (ptot == 0) getprime();
155     for (int i = 1; i <= ptot && prime[i] <= num; ++i) //剩下的就是不大于10^6的质因
        子了，直接暴力枚举
156     {
157         if (num % prime[i] == 0)
158         {
159             int cnt = 0;
160             while (num % prime[i] == 0)
161             {
162                 cnt++;
163                 num /= prime[i];
164             }
165             v.push_back((pt)
166             {
167                 prime[i], cnt
168             });
169         }
170     }
171     if (num > 1) v.push_back((pt)
172     {
173         num, 1
174     });
175     sort(v.begin(), v.end(), cmp);
176 }
177 int getorg(ll p, ll phi, vector<pt> &v) //获取ord
178 {
179     for (int k = 2;; k++)
180     {
181         int flag = 1;
182         for (int i = 0; i < (int) v.size(); ++i)
183         {
184             if (fastpow(k, phi / v[i].p, p) == 1)
185             {
186                 flag = 0;
187                 break;
188             }
189         }
190         if (flag) return k;
191     }
192 }
193 ll BSGS(ll a, ll b, ll p, ll mod) //BSGS模板
194 {
195     a %= mod , b %= mod;
196     if (b == 1) return 0;
197     if (a == 0)
```

```
198         {
199             if (b == 0) return 1;
200             else return -1;
201         }
202         ll t = 1;
203         int m = int(sqrt(1.0 * p) + 1);
204         ll base = b;
205         unordered_map<ll, ll> vis;
206         for (int i = 0; i < m; ++i)
207         {
208             vis[base] = i;
209             base = (__int128) base * a % mod;
210         }
211         base = fastpow(a, m, mod);
212         ll now = t;
213         for (int i = 1; i <= m + 1; ++i)
214         {
215             now = (__int128) now * base % mod;
216             if (vis.count(now)) return i * m - vis[now];
217         }
218         return -1;
219     }
220     ll getksi(ll g, ll h, ll p, ll c, ll n, ll mod) //得到合并后的解集，然后上BSGS
221     {
222         vector<ll> pi;
223         ll tp = 1;
224         for (int i = 0; i <= c; ++i)
225         {
226             pi.push_back(tp);
227             tp *= p;
228         }
229         ll gq = fastpow(g, pi[c - 1], mod);
230         ll inv = 0;
231         tp = 1;
232         for (int i = c - 1; i >= 0; --i)
233         {
234             ll tx = tp * BSGS(gq, fastpow((__int128) h * fastpow(g, pi[c] - inv, mod)
                    % mod, pi[i], mod), p, mod);
235             inv += tx;
236             tp *= p;
237         }
238         return inv;
239     }
240     ll exgcd(ll a, ll b, ll &x, ll &y) //exgcd模板
241     {
242         if (b == 0)
243         {
244             x = 1;
245             y = 0;
246             return a;
247         }
248         ll d = exgcd(b, a % b, y, x);
249         y -= a / b * x;
```

```
250         return d;
251     }
252     ll getinv(ll a, ll p) //扩欧求逆元
253     {
254         if (a == 0) return 1;
255         ll x, y;
256         exgcd(a, p, x, y);
257         return (x % p + p) % p;
258     }
259     ll gcd(ll x, ll y) //gcd
260     {
261         if (x % y == 0) return y;
262         return gcd(y, x % y);
263     }
264     ll ExgcdSolve(ll a, ll b, ll c, ll &x, ll &y) //求解exgcd
265     {
266         ll d;
267         if (c % (d = gcd(a, b))) return -1;
268         a /= d;
269         b /= d;
270         c /= d;
271         exgcd(a, b, x, y);
272         x = (__int128) x * c % b;
273         while (x <= 0) x += b;
274         return x;
275     }
276     ll crt(vector<ll> ksi, vector<pt> v) //crt
277     {
278         int sz = v.size();
279         ll M = 1, ans = 0;
280         vector<ll> m;
281         for (int i = 0; i < sz; ++i)
282         {
283             m.push_back(fastpow(v[i].p, v[i].c));
284             M *= m[i];
285         }
286         for (int i = 0; i < sz; ++i)
287         {
288             ll Mi = M / m[i];
289             ans = ((__int128) ans + (__int128) Mi * getinv(Mi, m[i]) * ksi[i]) % M;
290         }
291         if (ans < 0) ans += M;
292         return ans;
293     }
294     ll getx(ll h, ll g, ll N, ll mod, vector<pt> &v) //获取解集，然后用crt合并
295     {
296         vector<ll> ksi;
297         for (pt tp:v)
298         {
299             ll tg = fastpow(g, N / fastpow(tp.p, tp.c), mod);
300             ll th = fastpow(h, N / fastpow(tp.p, tp.c), mod);
301             ksi.push_back(getksi(tg, th, tp.p, tp.c, N, mod));
302         }
```

```
303         return crt(ksi, v);
304     }
305     ll solve(ll a, ll b, ll p) //求解a^x = b(mod p)的最小解
306     {
307         if (b == 1) return 0;
308         ll phiP = p - 1;
309         vector<pt> v;
310         findorg(v, phiP);
311         int rt = getorg(p, phiP, v);
312         ll x = getx(a, rt, phiP, p, v);
313         ll y = getx(b, rt, phiP, p, v);
314         ll aa = 0, bb = 0;
315         if (x == 0)
316         {
317             if (y == 0) return 1;
318             else if (y == 1) return 0;
319             else return -1;
320         }
321         else return ExgcdSolve(x, phiP, y, aa, bb);
322     }
323 };
324 signed main()
325 {
326     ll p , a , b;
327     cin >> p >> a >> b;
328     ll ans = DLP::solve(a , b , p);
329     if(~ans) cout << ans << '\n';
330     else cout << "no solution\n";
331     return 0;
332 }
```

4.5 二次剩余

4.5.1 总结

什么是二次剩余？

给出一个式子 $x^2 \equiv n$，再给出 n 和 p，如果能求得一个 x 满足该式子，即 $x^2 = n + kp, k \in z$，那么我们称 n 是模 p 的二次剩余。若不存在这样的 x，则我们称 n 是模 p 的非二次剩余。同时我们称 x 为该二次同余方程的解。

二次剩余的作用？

对于一个数 n，如果我们要求 $\sqrt{n} \bmod p$ 的值，那么我们可以看 n 是否是模 p 的二次剩余，如果是的话就会满足 $x^2 \equiv n(\mathrm{mod}p) \rightarrow x \equiv \sqrt{n}(\mathrm{mod}p)$，那么我们就可以用 x 来代替 $\sqrt{n}$，即只需要求该二次同余方程的解即可。
说白了就是如果该二次同余方程有解，那么 n 可以在模 p 的意义下开根号

二次同余方程定理

- 定理 1：对于 $x^2 \equiv n(\mathrm{mod}p)$，总共有 $\dfrac{p-1}{2}$ 个的 n 能使该方程有解（将 $n = 0$ 情况除去，由于该情况显然有 $x = 0$）

- 定理 2：$\left(\frac{a}{p}\right)=\begin{cases}1, & a \text{ 在模}p \text{ 意义下是二次剩余}\\ -1, & a \text{ 在模}p \text{ 意义下是非二次剩余}\\ 0, & a\equiv 0(\mathrm{mod}p)\end{cases}$

  $(\frac{a}{p})\equiv a^{\frac{p-1}{2}}(\mathrm{mod}p)$
- 定理 3：略
- 定理四：$(a+b)^n\equiv a^n+b^n(\mathrm{mod}n)(n\in P)$

4.5.2 2019 牛客多校 B

已知 $0<=x<=y<p, p=1e9+7$，且有

- $(x\ +\ y)=b\ \mod\ p$
- $(x\ \times\ y)=c\ \mod\ p$
- 求解任意一对 x, y，不存在输出 $-1-1$

由两式变化可得 $(y-x)^2=(b^2-4c+p)\,\%p \bmod p$，那么可以应用二次剩余定理解得 $y-x$ 的值
由题我们可以知道 $(x+y)=b$ 或者 $(x+y)=b+p$，那么直接求解即可。

```
#include<bits/stdc++.h>
#define ll long long
using namespace std;
const ll MOD = 1e9 + 7;
ll ppow(ll a, ll b, ll mod)
{
    ll ret = 1;
    a = a % mod;
    while(b)
    {
        if(b & 1) ret = ret * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return ret;
}
struct TT
{
    ll p, d;
};
ll w;
TT mul_er(TT a, TT b, ll mod)
{
    TT ans;
    ans.p = (a.p * b.p % mod + a.d * b.d % mod * w % mod) % mod;
    ans.d = (a.p * b.d % mod + a.d * b.p % mod) % mod;
    return ans;
}
TT power(TT a, ll b, ll mod)
{
    TT ret;
```

```
32      ret.p = 1, ret.d = 0;
33      while(b)
34      {
35          if(b & 1) ret = mul_er(ret, a, mod);
36          a = mul_er(a, a, mod);
37          b >>= 1;
38      }
39      return ret;
40  }
41  ll legendre(ll a, ll p)
42  {
43      return ppow(a, (p - 1) >> 1, p);
44  }
45  ll modulo(ll a, ll mod)
46  {
47      a %= mod;
48      if(a < 0) a += mod;
49      return a;
50  }
51  ll solve(ll n, ll p) //x^2 = n mod p
52  {
53      if(n == 0) return 0;
54      if(n == 1) return 1;
55      if(p == 2) return 1;
56      if(legendre(n, p) + 1 == p) return -1; //无解
57      ll a = -1, t;
58      while(true)
59      {
60          a = rand() % p;
61          t = a * a - n;
62          w = modulo(t, p);
63          if(legendre(w, p) + 1 == p) break;
64      }
65      TT temp;
66      temp.p = a;
67      temp.d = 1;
68      TT ans = power(temp, (p + 1) >> 1, p);
69      return ans.p;
70  }
71  bool getans(ll sum, ll dec, ll &x, ll &y)
72  {
73      if((sum + dec) % 2 == 0)
74      {
75          y = (sum + dec) / 2;
76          x = y - dec;
77          if(x >= 0 && x + y == sum && y < MOD) return true;
78          else return false;
79      }
80      else return false;
81  }
82  signed main()
83  {
84      int T;
```

```
85      cin >> T;
86      while(T--)
87      {
88          ll b, c;
89          cin >> b >> c;
90          ll d = solve((b * b % MOD - 4 * c % MOD + MOD) % MOD, MOD);
91          if(d == -1)
92          {
93              cout << "-1 -1\n";
94              continue;
95          }
96          ll x, y;
97          if(getans(b, d, x, y)) cout << x << " " << y << '\n';
98          else if(getans(b + MOD, d, x, y)) cout << x << " " << y << '\n';
99          else if(getans(b, MOD - d, x, y)) cout << x << " " << y << '\n';
100         else if(getans(b + MOD, MOD - d, x, y)) cout << x << " " << y << '\n';
101     }
102     return 0;
103 }
```

4.5.3 一元二次方程判断是否有解

对于一个模意义下的一元二次方程：$x^2 + bx + c = 0 (\mathrm{mod} p)$，其中 p 是质数。
每次给定一组 b, c, p $(a = 1)$，问这个方程有没有整数解，有解输出 Yes，无解输出 No

- 对于一般的二次同余方程
  $ax^2 + bx + c = 0 (\mathrm{mod} p)$
- 可以通过配方化为下式：
  $(2ax + b)^2 = b^2 - 4ac (\mathrm{mod} 4ap)$
- 设 $X = 2ax + b (\mathrm{mod} 4ap)$
- 解方程 $X^2 = b^2 - 4ac (\mathrm{mod} 4ap)$
- 可以先解出 $X^2 = b^2 - 4ac (\mathrm{mod} p)$ 的解，再去不断 $+p$ 凑 $4ap$，但经过进一步讨论，发现 $p > 2$ 时 $X^2 = b^2 - 4ac (\mathrm{mod} p)$ 有解、$X^2 = b^2 - 4ac (\mathrm{mod} 4ap)$ 一定有解（证明略）
- 当 p 为 2 时，直接特判

```
1  #include <bits/stdc++.h>
2  #define ll long long
3  using namespace std;
4  ll pow_mod(ll a, ll i, ll n)
5  {
6      if(!i) return 1 % n;
7      ll temp = pow_mod(a, i >> 1, n);
8      temp = temp * temp % n;
9      if(i & 1) temp = temp * a % n;
10     return temp;
11 }
12 ll modsqr(int a, int n)
13 {
14     ll b, k, i, x;
```

```
    if(n == 2) return a % n;
    if(pow_mod(a, (n - 1) / 2, n) == 1)
    {
        if(n % 4 == 3) x = pow_mod(a, (n + 1) / 4, n);
        else
        {
            for(b = 1; b = pow_mod(b, (n - 1) / 2, n), n == 1; b ++) ;
            i = (n - 1) / 2 , k = 0;
            do
            {
                i /= 2 , k /= 2;
                if ((pow_mod(a, i, n) * (ll)pow_mod(b, k, n) + 1) % n == 0)
                {
                    k += (n - 1) / 2;
                }
            } while (i % 2 == 0);
            x = (pow_mod(a, (i - 1) / 2, n) * (ll)pow_mod(b, k / 2, n)) % n;
        }
        if(x * 2 > n) x = n - x;
        return x;
    }
    if(a == 0) return 0;
    return -1;
}
signed main()
{
    int T = 1;
    cin >> T;
    while (T--)
    {
        ll a = 1 , b, c , p;
        cin >> b >> c >> p;
        if (p == 2)
        {
            if (c % p == 0 || (a + b + c) % p == 0) puts("Yes");
            else puts("No");
            continue;
        }
        ll t = b * b - 4 * a * c;
        t = (t % p + p) % p;
        ll s = modsqr((ll)t, p);
        bool flag = (s != -1);
        puts(flag ? "Yes" : "No");
    }
    return 0;
}
```

4.5.4 求所有解

- 求解方程 $x^2 \equiv N (\mathrm{mod}\ p)$
- 若有解，则按 mod p 后递增的顺序输出在 mod p 意义下的全部解.

- 若两解相同，只输出其中一个
- 若无解，则输出 Hola!

```
#define ll long long
ll w;
struct num
{
    ll x, y;
};
num mul(num a, num b, ll p)
{
    num ans = {0, 0};
    ans.x = ((a.x * b.x % p + a.y * b.y % p * w % p) % p + p) % p;
    ans.y = ((a.x * b.y % p + a.y * b.x % p) % p + p) % p;
    return ans;
}
ll powwR(ll a, ll b, ll p)
{
    ll ans = 1;
    while (b)
    {
        if (b & 1)
            ans = 1ll * ans % p * a % p;
        a = a % p * a % p;
        b >>= 1;
    }
    return ans % p;
}
ll powwi(num a, ll b, ll p)
{
    num ans = {1, 0};
    while (b)
    {
        if (b & 1)
            ans = mul(ans, a, p);
        a = mul(a, a, p);
        b >>= 1;
    }
    return ans.x % p;
}
ll M_sqrt(ll n, ll p)
{
    n %= p;
    if (p == 2)
        return n;
    if (powwR(n, (p - 1) / 2, p) == p - 1)
        return -1; //不存在
    ll a;
    while (1)
    {
        a = rand() % p;
        w = ((a * a % p - n) % p + p) % p;
```

```
50          if (powwR(w, (p - 1) / 2, p) == p - 1)
51              break;
52      }
53      num x = {a, 1};
54      return powwi(x, (p + 1) / 2, p);
55  }
56  signed main()
57  {
58      ios::sync_with_stdio(false);
59      int t;
60      cin >> t;
61      while(t --)
62      {
63          ll n , p;
64          cin >> n >> p ;
65          if(!n)
66          {
67              cout << 0 << '\n' ;
68              continue ;
69          }
70          ll ans1 = M_sqrt(n , p) , ans2;
71          if(ans1 == -1) cout << "Hola!\n";
72          else
73          {
74              ans2 = p - ans1;
75              if(ans1 > ans2) swap(ans1 , ans2);
76              if(ans1 == ans2) cout << ans1 << '\n';
77              else cout << ans1 << " " << ans2 << '\n' ;
78          }
79      }
80      return 0;
81  }
```

4.6 反素数

4.6.1 总结

定义

- 对于任何正整数 x，其约数的个数记作 $g(x)$，例如 $g(1)=1$，$g(6)=4$
- 如果某个正整数 x 满足：$\forall 0<i<x$，都有 $g(x)>g(i)$，则称 x 为 **反素数**，例如整数 $1,2,4,6$ 等都是反质数

性质 1

- $1\sim N$ 中的最大的反质数，就是 $1\sim N$ 中约数个数最多的数中最小的一个

证明 1

设 m 是 $1\sim N$ 中约数个数最多的数中最小的一个
根据 m 的定义，m 显然满足：
$1.\forall x<m, g(x)<g(m)$
$2.\forall x>m, g(x)\leq g(m)$
根据反素数的定义，1 说明 m 是反质数，2 说明大于 m 的数都不是反质数，证毕

性质 2

- $1 \sim N$ 中任何数的不同质因子都不会超过 10 个，且所有质因子的指数总和不超过 30 (前提是 $N <= 2 \times 10^9$)
- 证明略

性质 3

- x 为反质数的必要条件是：x 分解质因数后可写作 $2^{c1} \times 3^{c2} \times ...$, 并且 $c1 \geq c2 \geq c3 \geq ... \geq 0$
- 证明略

性质 4

- 一个反素数位全部倒过来后还是一个反素数，比如 13、31 都是反素数，17、71 都是反素数...

4.6.2 求不超过 n 的最小反素数

$1 \leq N \leq 2 \times 10^9$

```
int prime[]={2 , 3 , 5 , 7 , 11 , 13 , 17 , 19 , 23 , 29 , 31 , 37 , 41 , 43 , 47 ,
    53};
int n , ma , res;
void dfs(int pos , int cnt , int now , int up)
{
   if(now > n || pos > 15) return ;
   if(cnt > ma) ma = cnt , res = now;
   else if(cnt == ma) res = min(res , now);
   rep(i , 1 , up)
   {
      if(now * prime[pos] > n) break ;
      now = now * prime[pos];
      dfs(pos + 1 , cnt * (i + 1) , now , i);
   }
}
signed main()
{
   cin >> n;
   dfs(0 , 1 , 1 , 32);
   cout << res << '\n';
   return 0;
}
```

4.6.3 求约数个数为 n 最小反素数

```
int prime[]={2 , 3 , 5 , 7 , 11 , 13 , 17 , 19 , 23 , 29 , 31 , 37 , 41 , 43 , 47};
unsigned ll ans = INF , n;
void dfs(int pos , int val , int num , int up)
{
   if(num > n) return ;
   if(num == n && val < ans) {ans = val ; return ;}
   rep(i , 1 , up)
```

```
    {
        if(val * prime[pos] > ans) break ;
        val *= prime[pos];
        dfs(pos + 1 , val , num * (i + 1) , i);
    }
}
signed main()
{
    cin >> n;
    dfs(0 , 1 , 1 , 63);
    cout << ans << '\n';
    return 0;
}
```

4.7 各种筛

4.7.1 x 的质因子个数

```
int cnt[N];
void prime_cnt(int n)
{
    for(int i = 2 ; i <= n ; i ++)
    {
        if(cnt[i]) continue ;
        for(int j = i ; j <= n ; j += i) cnt[j] ++ ;
    }
}
```

4.7.2 矩阵的 lcm

$A_{i,j} = lcm(i,j),\ O(NM)$

```
for(int i = 1 ; i <= n ; i ++) for(int j = 1 ; j <= m ; j ++)
if(!a[i][j])
{
    for(int k = 1 ; i * k <= n && j * k <= m ; k ++)
    a[i * k][j * k] = i * k * j;
}
```

4.7.3 素数 + 欧拉函数 + 最小质因子

```
int prime[100010], minprime[100010], phi[100010];
int euler(int n)
{
    int c = 0, i, j;
    phi[1] = 1;
    for (i = 2; i <= n; i++)
    {
        if (!minprime[i])
        prime[++c] = i, minprime[i] = i, phi[i] = i - 1;
        for (j = 1; j <= c && i * prime[j] <= n; j++)
        {
```

```
            minprime[i * prime[j]] = prime[j];
            if (i % prime[j] == 0)
            {
                phi[i * prime[j]] = phi[i] * prime[j];
                break;
            }
            else
            phi[i * prime[j]] = phi[i] * (prime[j] - 1);
        }
    }
    return c;
}
```

4.8 康托展开

4.8.1 康托展开

康托展开可以求解一个排列的排名，比如：12345 排名为 1，12354 排名为 2，按字典序增加排名递增，依次类推。

拿 52413 举例子：

- 首先看第一个数 5，不管第一位是什么数，后面都有四位数，那么这四位数全排列的方式有 4! 种，而如果第一位是 1 或 2 或 3 或 4 都会比 5 开头的字典序要小，所以可以令 1，2，3，4 分别作为开头，这样的话就会有 $4 \times 4!$ 种排法要比 52413 这种排法的字典序要小。
- 那么第一个数是 1，2，3，4 时候的字典序的个数数完了是 $4 \times 4!$ 种，且这些字典序都要比 52413 的字典序要小。
- 还有其他的排列方式比 52413 的字典序要小的吗？无
- 那么就可以固定第一位 5，找下一位 2，这时 5 已经用过了，所以从剩下的 1 2 3 4 里挑选比 2 小的数，一共 1 个，后面还剩三位，也就是 3! 种排列方式，那么这时候比 52413 字典序要小的又有 $1 \times 3!$ 种，也就是当 5 在第一位，1 在第二位的时候。
- 再看第三位 4，这时 5 2 都用了，所以从剩下的 1 3 4 三个数中找比 4 小的数的个数，有两个比 4 小原理同上，所以这时候也可以有 $2 * 2!$ 种排列方式的字典序小于 52413
- 再看第四位 1，这时候会有 $0 \times 1!$ 种
- 再看第五位 3，这时候会有 $0 \times 0!$ 种

```
#include<bits/stdc++.h>
#define rep(i , a , b) for(int i = a ; i <= b ; i ++)
#define int long long
using namespace std;
const int N = 1e6 + 10 , mod = 998244353;
int tree[N << 2] , a[N] , fac[N];
int lowbit(int x)
{
    return x & (-x);
}
void update(int pos , int val)
{
    while(pos <= N - 1)
```

```
14      {
15          tree[pos] += val;
16          pos += lowbit(pos);
17      }
18  }
19  int query(int pos)
20  {
21      int res = 0;
22      while(pos)
23      {
24          res += tree[pos];
25          pos -= lowbit(pos);
26      }
27      return res;
28  }
29  void init()
30  {
31      fac[0] = 1;
32      rep(i , 1 , N - 10) fac[i] = fac[i - 1] * i % mod;
33  }
34  int calc(int n)
35  {
36      int rank = 0;
37      rep(i , 1 , n) update(i , 1);
38      rep(i , 1 , n)
39      {
40          rank += fac[n - i] * query(a[i] - 1);
41          rank %= mod;
42          update(a[i] , -1);
43      }
44      return rank + 1;
45  }
46  signed main()
47  {
48      init();
49      int n;
50      cin >> n;
51      rep(i , 1 , n) cin >> a[i];
52      cout << calc(n) << '\n';
53      return 0;
54  }
```

4.8.2 康托逆展开

康托展开是一个比较常用的哈希技巧，可以将一个排列 $a_1, a_2 \cdots a_n$ 映射到一个整数 k，这个整数 k 就是这个排列在所有排列中的名次

由于它是双射的，所以也可以从一个整数还原这个整数所对应的全排列

- 假定这个排列是由 n 个数组成的，那么有从一个整数 k 映射到第 k 小的排列的方法：
- 将 k 写成 $\sum_{i=1}^{n} rank_i(n-i)!$ 的形式，其中对于任意 $rank_i$，有 $0 \le rank_i \le i$
- 对于第 i 次操作，选择当前没有选过的第 $rank_i$ 大的数加入排列 (**权值线段树上二分**)，所得的排列即为所求

- 而根据 $rank_i \times (n-i)! <= k$ ，得 $rank_i = k \div (n-i)!$
- 有了 $rank_i$ 后，k 也要相应减去 $rank_i \times (n-i)!$ 以便求 $rank_{i-1}$
- 即 $k = k - rank_i \times (n-i)! \rightarrow k = k\%(n-i)$

4.9 扩展欧几里得

4.9.1 exgcd

求解**二元一次不定方程** $ax + by = c$ $(1 \leq a, b, c \leq 10^9)$

- 若该方程无整数解，输出 -1
- 若方程有整数解，但没有正整数解，则输出所有**整数解**中 x 的最小正整数值，y 的最小正整数值
- 若方程有整数解，且有正整数解，则输出其**正整数**解的数量，所有**正整数**解中 x 的最小值，y 的最小值，x 的最大值，y 的最大值

```
void exgcd(ll a , ll b , ll &x , ll &y)
{
   if(!b){
      x = 1 , y = 0;
      return;
   }
   exgcd(b , a % b , x , y);
   ll t = x;
   x = y , y = t - (a / b) * y;
}
signed main()
{
   int T = 1;
   cin >> T;
   while(T --)
   {
      ll a , b , c , x = 0 , y = 0;
      cin >> a >> b >> c;
      ll g = __gcd(a , b);
      if(c % g != 0) { // 无解
         cout << "-1\n";
         continue ;
      }
      exgcd(a , b , x , y);
      x *= c / g , y *= c / g;
      ll p = b / g , q = a / g , k;
      if(x < 0) k = ceil((1.0 - x) / p) , x += p * k , y -= q * k; //将 x 提高到最小
          正整数
      else k = (x - 1) / p , x -= p * k , y += q * k;    //将 x 降低到最小正整数
      if(y > 0)  //有正整数解
      {
         cout << (y - 1) / q + 1 << " ";  //将 y 减到 1 的方案数即为解的个数
         cout << x << " ";     //当前的 x 即为最小正整数 x
         cout << (y - 1) % q + 1 << " ";  //将 y 取到最小正整数
         cout << x + (y - 1) / q * p << " "; //将 x 提升到最大
```

```
35          cout << y << '\n';      //特解即为 y 最大值
36      }
37      else   //无整数解
38      {
39          cout << x << " ";         //当前的 x 即为最小的正整数 x
40          cout << y + q * (ll)ceil((1.0 - y) / q) << '\n'; //将y提高到正整数
41      }
42   }
43   return 0;
44 }
```

4.9.2 求同余方程

求关于 x 的同余方程 $ax \equiv 1 \pmod b$ 的最小正整数解，保证有解 $(2 \leq a, b \leq 2,000,000,000)$

- 式子可等价于 $ax + by = 1$，然后跑 $exgcd$ 就好了
- (因为保证有解，所以可以直接跑 $exgcd$)

```
1  signed main()
2  {
3     ll a , b , x = 0 , y = 0;
4     cin >> a >> b;
5     exgcd(a , b , x , y);
6     cout << (x % b + b) % b << '\n';
7     return 0;
8  }
```

4.10 离散对数

4.10.1 BSGS

- 求最小的 x 使得 $a^x \equiv b(mod\ p)$
- p 为质数且 $2 \leq a, b < p < 2^{31}$
- 复杂度为 $O(\sqrt{p} \times log)$ ，log 为 map 的复杂度 (卡常可改为 $unordered\_map$)

```
1  struct BSGS
2  {
3     ll power(ll a, ll b, ll c) //快速幂
4     {
5        if(b == 0) return 1 % c;
6        ll ans = 1, t = a;
7        while (b > 0)
8        {
9           if (b % 2 == 1) ans = ans * t % c;
10          b >>= 1;
11          t = t * t % c;
12       }
13       return ans;
14    }
15    ll bsgs(ll a, ll b, ll p) //bsgs
```

```
    {
        map<ll, ll> hash;
        hash.clear(); //建立一个Hash表
        b %= p;
        ll t = sqrt(p) + 1;
        for(ll i = 0; i < t; ++i) hash[(ll)b * power(a, i, p) % p] = i; //将每个j对应
            的值插入Hash表
        a = power(a, t, p);
        if(!a) return b == 0 ? 1 : -1; //特判
        for(ll i = 1; i <= t; ++i) //在Hash表中查找是否有i对应的j值
        {
            ll val = power(a, i, p);
            int j = hash.find(val) == hash.end() ? -1 : hash[val];
            if (j >= 0 && i * t - j >= 0)
            return i * t - j;
        }
        return -1; //无解返回-1
    }
} bs;
signed main()
{
    ll p , a , b;
    cin >> p >> a >> b;
    ll ans = bs.bsgs(a, b, p);
    if(~ans) cout << ans << '\n';
    else cout << "no solution\n";
    return 0;
}
```

4.10.2 EX_BSGS

- 给定 a, p, b，求满足 $a^x \equiv b \pmod{p}$ 的最小自然数 x
- $a, p, b \leq 10^9$（p 不要求为质数）
- 复杂度为 $O(\sqrt{p} + log^2 p)$，$log^2 p$ 是约分和求公约数产生的

```
#define int long long
int fpow(int a, int b, int p)
{
    int sum = 1;
    for (; b; b >>= 1, a = a * a % p)
    if (b & 1) sum = sum * a % p;
    return sum;
}
int bsgs(int a, int b, int p)
{
    map<int, int>hash;
    hash.clear();
    int m = sqrt(p) + 1;
    for (int i = 1; i <= m; i++)
    b = b * a % p, hash[b] = i;
    int tt = fpow(a, m, p);
```

```cpp
    for (int i = 1, res = 1; i <= m; i++)
    {
        res = res * tt % p;
        if (hash.find(res) != hash.end()) return (i * m - hash[res] + p) % p;
    }
    return -1;
}
void exgcd(int a, int b, int& x, int& y)
{
    if (b == 0)
    {
        x = 1, y = 0;
        return;
    }
    exgcd(b, a % b, y, x);
    y -= a / b * x;
}
int inv(int a, int b)
{
    int X0, Y0;
    exgcd(a, b, X0, Y0);
    return (X0 % b + b) % b;
}
int exbsgs(int a, int b, int p)
{
    if (b == 1 || p == 1) return 0;
    int d, cnt = 0, res = 1;
    while ((d = __gcd(a, p)) != 1)
    {
        if (b % d) return -1;
        cnt++;
        b /= d;
        p /= d;
        res = res * a / d % p;
        if (res == b) return cnt;
    }
    int ans = bsgs(a, b * inv(res, p) % p, p);
    if (ans == -1) return -1;
    return ans + cnt;
}
signed main()
{
    while(1)
    {
        int a , b , p;
        cin >> a >> p >> b;
        if(!a && !b && !p) return 0;
        int ans = exbsgs(a % p , b % p , p);
        if(~ans) cout << ans << '\n';
        else cout << "No Solution\n";
    }
    return 0;
}
```

4.11 欧拉函数

4.11.1 欧拉函数性质

1. $\sum_{i=1}^{n-1}[gcd(i,n)=1]=\varphi(n)$，定义式常用来加速运算。
2. 当 $n>2$ 时，$\varphi(n)$ 是偶数。
3. 若 a,b 互质，则 $\varphi(ab)=\varphi(a)*\varphi(b)$。
4. 若 $n\geqslant 1$，$[1,n]$ 中与 n 互质的数的和为 $max(1ll,n*\varphi(n)/2)$。
5. 若 p 是质数，则 $\varphi(p)=p-1$。
6. 若 p 是质数，且 $p\mid n$ 但 $p\nmid n$，则 $\varphi(n)=\varphi(n/p)*(p-1)$。
7. $\sum_{d|n}\varphi(d)=n$
8. $\prod_{n=1}^{\infty}(1-x^n)=\sum_{k=0}^{\infty}(-1)^k x^{\frac{k*(3k\pm 1)}{2}}$
9. 积性函数指对于所有互质的整数 a 和 b 有性质 $f(ab)=f(a)f(b)$ 的数论函数。
10. 如果 $gcd(m,n)=d$，那么 $\varphi(m\times n)=\frac{\varphi(m)\times\varphi(n)\times d}{\varphi(d)}$。

4.11.2 sqrt(N)

```
ll get_phi(ll n)
{
    ll ans = n;
    for(ll i = 2 ; i * i <= n ; i ++)
    {
        if(n % i == 0)
        {
            ans -= ans / i;
            while(n % i == 0) n /= i;
        }
    }
    if(n > 1) ans -= ans / n;
    return ans;
}
```

4.11.3 欧拉筛

- $prime[i]$ 表示第 i 个质数
- $minprime[i]$ 表示 i 的最小质因子
- $phi[i]$ 表示 i 的欧拉数（比 i 小且与 i 互质的数的个数）

```
int prime[1000010] , minprime[1000010] , phi[1000010];
int euler(int n)
{
    int c = 0 , i , j;
    phi[1] = 1;
    for(i = 2 ; i <= n ; i ++)
    {
```

```
        if(!minprime[i]) prime[++ c] = i , minprime[i] = i , phi[i] = i - 1;
        for(j = 1 ; j <= c && i * prime[j] <= n ; j ++)
        {
            minprime[i * prime[j]] = prime[j];
            if(i % prime[j] == 0)
            {
                phi[i * prime[j]] = phi[i] * prime[j];
                break;
            }
            else phi[i * prime[j]] = phi[i] * (prime[j] - 1);
        }
    }
    return c;
}
```

4.11.4 N^(0.25)+1e18

- 用到了 $Pollard_Rho + Miller_Rabin$
- 复杂度为 $O(N^{1/4})$

```
const int N = 100 , P = 5;
const int prime[P] = {2, 3, 7, 61, 24251};
ll cnt, pri[N];
ll add(ll a, ll b, ll p){
    return (a += b) >= p ? a - p : a;
}
ll mul(ll a, ll b, ll p)
{
    a %= p, b %= p;
    ll c = (long double) a * b / p;
    c = a * b - c * p;
    return (c + p) % p;
}
ll random(ll p){
    return 1ll * rand() * rand() % p;
}
ll gcd(ll a, ll b){
    return !b ? a : gcd(b, a % b);
}
ll trans(ll x, ll y, ll p){
    return add(mul(x, x, p), y, p);
}
ll fast_pow(ll a, ll b, ll p)
{
    ll res = 1;
    for (; b; b >>= 1, a = mul(a, a, p))
    if (b & 1) res = mul(res, a, p);
    return res;
}
bool Miller_Rabin_check(ll a, ll p)
{
    ll cur = p - 1, tim = 0;
```

```
33      for (; !(cur & 1); cur >>= 1, tim++);
34      ll x = fast_pow(a, cur, p);
35      for (int i = 1; i <= tim; i++)
36      {
37          ll y = mul(x, x, p);
38          if (y == 1 && (x != 1) && (x != p - 1)) return 0;
39          x = y;
40      }
41      return x == 1;
42  }
43  bool Miller_Rabin(ll x)
44  {
45      for (int i = 0; i < P; i++)
46      {
47          if (x == prime[i]) return 1;
48          if (!Miller_Rabin_check(prime[i], x)) return 0;
49      }
50      return 1;
51  }
52  void Pollard_Rho(ll x)
53  {
54      if (Miller_Rabin(x))
55      {
56          pri[++cnt] = x;
57          return;
58      }
59      ll x1 = 0, x2 = 0, c = 0, p = 1;
60      while (p == 1 || p == x)
61      {
62          x1 = trans(x1, c, x);
63          x2 = trans(trans(x2, c, x), c, x);
64          while (x1 == x2)
65          {
66              c = random(x);
67              x1 = x2 = random(x);
68              x2 = trans(x2, c, x);
69          }
70          p = gcd(abs(x1 - x2), x);
71      }
72      Pollard_Rho(p);
73      Pollard_Rho(x / p);
74  }
75  ll get_phi(ll x)
76  {
77      srand(time(0));
78      if(x == 1) return 1;
79      cnt = 0;
80      Pollard_Rho(x);
81      sort(pri + 1, pri + cnt + 1);
82      cnt = unique(pri + 1, pri + cnt + 1) - pri - 1;
83      for (int i = 1; i <= cnt; i++)
84      x = x / pri[i] * (pri[i] - 1);
85      return x;
```

```
}
signed main()
{
    ll n ;
    cin >> n;
    cout << get_phi(n) << '\n';
    return 0;
}
```

4.12 欧拉降幂

4.12.1 费马小定理

- 当 $a, p \in \mathbb{Z}$ 且 p 为质数，且 $a \not\equiv 0 \pmod{p}$ 时有：
- $a^{p-1} \equiv 1 \pmod{p}$
- 所以 $a^b \equiv a^{b \bmod (p-1)} \pmod{p}$

4.12.2 欧拉定理

- 当 $a, m \in \mathbb{Z}$，且 $\gcd(a, m) = 1$ 时有：
- $a^{\varphi(m)} \equiv 1 \pmod{m}$
- 这里 $\varphi(x)$ 是数论中的欧拉函数。
- 所以 $a^b \equiv a^{b \bmod \varphi(m)} \pmod{m}$

4.12.3 扩展欧拉定理

- 当 $a, m \in \mathbb{Z}$ 时有：
- $a^b \equiv \begin{cases} a^b & , b < \varphi(m) \\ a^{b \bmod \varphi(m) + \varphi(m)} & , b \geq \varphi(m) \end{cases} \pmod{m}$

注意要判断 b 和 $\varphi(m)$ 的关系！

4.13 排列组合

4.13.1 总结

- $exlucas$ 要求 p 不能很大
- 线性求法需要满足 $p > N$，否则会出现 $i = p$ ， $fac[i] = fac[i-1] * i \bmod p = 0$

4.13.2 排列组合公式

- $A_n^m = n(n-1)...(n-m+1) = \dfrac{n!}{(n-m)!}$
- $C_n^m = \dfrac{A_n^m}{m!} = \dfrac{n!}{m!(n-m)!} = C_n^{n-m}$
- $C_{n+1}^m = C_n^m + C_n^{m-1}$
- $C_n^0 + C_n^1 + C_n^2 + ... + C_n^n = 2^n$
- $(x+1)^n = C_n^0 + C_n^1 x + C_n^2 x^2 + ... + C_n^n x^n$

4.13.3 暴力求法

```
long long C(int a , int b)

{

    long long res = 1;

    for(int i = a , j = 1 ; j <= b ; i -- , j ++) res = res * i / j;

    return res;

}
```

4.13.4 线性 (N 要小于 p)

```
const int N = 3e5 + 10 , mod = 998244353;
int fac[N] , inv[N];
int pow_mod(int x , int n , int mod)
{
    int res = 1;
    while(n)
    {
        if(n & 1) res = res * x % mod;
        x = x * x % mod;
        n >>= 1;
    }
    return res;
}
void init()
{
    int up = N - 10;
    fac[0] = 1;
    for(int i = 1 ; i <= up ; i ++) fac[i] = fac[i - 1] * i % mod;
    inv[up] = pow_mod(fac[up] , mod - 2 , mod);
    for(int i = up - 1 ; i >= 0 ; i --) inv[i] = inv[i + 1] * (i + 1) % mod;
}
int C(int n , int m)
{
    if(n < m) return 0;
    return fac[n] * inv[m] % mod * inv[n - m] % mod;
}
signed main()
{
    init();
    int n , m ;
    cin >> n >> m ;
    cout << C(n , m) << '\n';
    return 0;
}
```

4.13.5 卢卡斯

- 调试的时候模数记得用质数!
- 复杂度为 $O(kp+logp)$，k 是 n 转化为 p 进制后的位数

```
ll n, m, p;
ll Ext_gcd(ll a, ll b, ll &x, ll &y)
{
    if (b == 0)
    {
        x = 1, y = 0;
        return a;
    }
    ll ret = Ext_gcd(b, a % b, y, x);
    y -= a / b * x;
    return ret;
}
ll Inv(ll a, int m) ///求逆元a相对于m
{
    ll d, x, y, t = (ll)m;
    d = Ext_gcd(a, t, x, y);
    if (d == 1) return (x % t + t) % t;
    return -1;
}
ll C(ll n, ll m, ll p) ///组合数学
{
    ll a = 1, b = 1;
    if (m > n) return 0;
    while (m)
    {
        a = (a * n) % p , b = (b * m) % p;
        m-- , n--;
    }
    return (ll)a * Inv(b, p) % p; /// (a/b) %p 等价于 a* (b, p) 的逆元
}
// Lucas (n, m, p) =C (n%p, m%p, p) *Lucas (n/p, m/p, p) ;  ///这里可以采用对n分段递归求
    解,
// Lucas (x, 0, p) =1;
int Lucas(ll n, ll m, ll p) ///把n分段递归求解相乘
{
    if (m == 0) return 1;
    return (ll)C(n % p, m % p, p) * (ll)Lucas(n / p, m / p, p) % p;
}
signed main()
{
    int T = 1;
    cin >> T;
    while(T --)
    {
        cin >> n >> m >> p;
        cout << Lucas(n + m , m , p) << '\n';
    }
    return 0;
```

```
}
```

4.13.6 扩展卢卡斯

```
ll pow_mod(ll x, ll n, ll mod)
{
    ll res = 1;
    while (n)
    {
        if (n & 1)
            res = res * x % mod;
        x = x * x % mod;
        n >>= 1;
    }
    return res;
}
void exgcd(ll a, ll b, ll &x, ll &y)
{
    if (!b)
    {
        x = 1, y = 0;
        return;
    }
    exgcd(b, a % b, x, y);
    ll t = x;
    x = y, y = t - (a / b) * y;
}
ll inv(ll a, ll b)
{
    ll x, y;
    return exgcd(a, b, x, y), (x % b + b) % b;
}
ll crt(ll x, ll p, ll mod)
{
    return inv(p / mod, mod) * (p / mod) * x;
}
ll fac(ll x, ll a, ll b)
{
    if (!x)
        return (1);
    ll ans = 1;
    for (ll i = 1; i <= b; i++)
        if (i % a)
            ans *= i, ans %= b;
    ans = pow_mod(ans, x / b, b);
    for (ll i = 1; i <= x % b; i++)
        if (i % a)
            ans *= i, ans %= b;
    return (ans * fac(x / a, a, b) % b);
}
ll C(ll n, ll m, ll a, ll b)
{
```

```
49      ll N = fac(n, a, b), M = fac(m, a, b), Z = fac(n - m, a, b), sum = 0, i;
50      for (i = n; i; i = i / a)
51          sum += i / a;
52      for (i = m; i; i = i / a)
53          sum -= i / a;
54      for (i = n - m; i; i = i / a)
55          sum -= i / a;
56      return (N * pow_mod(a, sum, b) % b * inv(M, b) % b * inv(Z, b) % b);
57  }
58  ll exlucas(ll n, ll m, ll p)
59  {
60      ll t = p, ans = 0, i;
61      for (i = 2; i * i <= p; i++)
62      {
63          ll k = 1;
64          while (t % i == 0)
65          {
66              k *= i, t /= i;
67          }
68          ans += crt(C(n, m, i, k), p, k), ans %= p;
69      }
70      if (t > 1)
71          ans += crt(C(n, m, t, t), p, t), ans %= p;
72      return (ans % p);
73  }
74  signed main()
75  {
76      ll n, m, p;
77      cin >> n >> m >> p;
78      cout << exlucas(n, m, p) << '\n';
79      return (0);
80  }
```

4.13.7 错排

问题：

有 n 个小球对应顺序放在 n 个箱子里，现要求重新排列，使得每个小球都不放原来的位置上，求有多少种排列方式？

错排：

1. 数学上我们用 D_n 表示有 n 个球 n 个箱子时的方案数。
2. 自己简单算一下可以得出：$D_1 = 0, D_2 = 1, D_3 = 2$
3. 来看下 $D_n(n \geq 4)$ 的情况。要推出怎么做，需要分类讨论。
4. 不妨分成 $n-1$ 种情况：n 号球放进了 1 号箱子，n 号球放进了 2 号箱子… n 号球放进了 $n-1$ 号箱子（n 号球不能放进 n 号箱子）。

注意，分类讨论时要搞清楚是否涵盖了所有的情况。我们可以把所有情况列出来：

把 n 号球放进：

- 1 号箱子

- 2 号箱子
- ……
- $(k-1)$ 号箱子
- k 号箱子
- $(k+1)$ 号箱子
- ……
- $(n-1)$ 号箱子

现在，我们只着眼于一种情况：n 号球放进了 k 号箱子。
那么此时对于 k 号小球，就有两种选择：

1. 放入 n 号箱子
2. 不放入 n 号箱子

- 当 k 号小球放入了 n 号箱子，剩下的 $n-2$ 个小球进行错排，那么答案就是 D_{n-2}
- k 号小球不放入 n 号箱子，可以理解为：
  剩余 $n-1$ 个小球，1 号小球不能放入 1 号箱子，2 号小球不能放入 2 号箱子，..... , k 号小球不能放入 n 号箱子，.....，$n-1$ 号小球不能放入 $n-1$ 号箱子。然后进行错排，那么答案就是 D_{n-1}

所以递推公式： $D_n=(n-1)\times(D_{n-1}+D_{n-2})$ **（乘上 $(n-1)$ 是因为 k 有 $n-1$ 种选择）**
通项公式： $D_n=n!\left[\frac{1}{2!}-\frac{1}{3!}+...+(-1)^n\frac{1}{n!}\right]$

4.14 裴蜀定理

4.14.1 总结

若 a,b 是整数，且 $gcd(a,b)=d$，对于任意整数 x,y，$ax+by$ 都一定是 d 的倍数
特别地, 一定存在整数 x,y，使 $ax+by=gcd(a,b)$ 成立（x,y 不一定是正整数）
重要推论： a,b **互质的充要条件是存在整数** x,y **使** $ax+by=1$

4.15 唯一分解定理 + 约数定理

4.15.1 总结

唯一分解定理

- 对一个大于 1 的整数 n，n 可以分解质因数为 $\prod_{i=1}^{k}p_i^{a_i}=p_1^{a_1}\cdot p_2^{a_2}\cdots p_k^{a_k}$

约数个数定理

- n 的正约数的个数 $d(n)=\prod_{i=1}^{k}(a_i+1)=(a_1+1)(a_2+1)\cdots(a_k+1)$

约数和定理

- $sum=(p_1^0+p_1^1+...+p_1^{a_1})(p_2^0+p_2^1+...p_2^{a_2})...(p_k^0+p_k^1+...p_k^{ak})$

4.15.2 约数之和

- 有两个自然数 A 和 B，S 是 A^B 的所有约数之和
- 求 S mod 9901

A^B 的约数之和 $= (1 + p1^1 + p1^2 + ... + p1^{B*c1}) \times (1 + p2^1 + ... + p2^{B \times c2} \times ...$

```
const int N = 3e5 + 10;
const int mod = 9901;
pair<int , int>a[N];
signed main()
{
    int A , b , cnt = 0;
    cin >> A >> b;
    if(!A) return cout << 0 << '\n' , 0;
    else if(!b) return cout << 1 << '\n' , 0;
    for(int i = 2 ; i * i <= A ; i ++)
    {
        if(A % i == 0)
        {
            int c = 0;
            while(A % i == 0) A /= i , c ++;
            a[++ cnt] = make_pair(i , c);
        }
    }
    if(A > 1) a[++ cnt] = make_pair(A , 1);
    int res = 1;
    rep(i , 1 , cnt)
    {
        int n = a[i].se * b + 1 , p = a[i].fi;
        int x = (pow_mod(p , n , mod) - 1 + mod) % mod;
        int y = (pow_mod(a[i].fi - 1 , mod - 2 , mod) + mod) % mod;
        if(!x) res *= n; // 当公差为 1 时 , sum = n * a1(不过此题不会有这种情况)
        else
        {
            res *= x , res %= mod;
            res *= y , res %= mod;
        }
    }
    cout << (res + mod) % mod << '\n';
    return 0;
}
```

4.15.3 求二元倒数方程

求方程：$\frac{1}{x} + \frac{1}{y} = \frac{1}{n!}$ 的正整数解的组数，答案对 $10^9 + 7$ 取模

$$\frac{1}{x} + \frac{1}{y} = \frac{1}{n!}$$
$$\frac{x+y}{xy} = \frac{1}{n!}$$
$$xy - n!(x + y) = 0$$
$$(n!)^2 + xy - n!(x + y) = (n!)^2$$
$$(x - n!)(y - n!) = (n!)^2$$

$a = (x - n!), b = (y - n!)$
$ab = (n!)^2$
$n! = p_1^{c_1} \times p_2^{c_2} \times ... \times p_k^{c_k}$
$(n!)^2 = p_1^{2c_1} \times p_2^{2c_2} \times ... \times p_k^{2c_k}$
因为 $n!$ 是确定的，所以确定了 a b，就能确定 x y
而且只要确定了 a，就能确定 b
a 是 $(n!)^2$ 的因数，根据约数个数定理得：
a 可选取的个数为 $(2c_1 + 1)(2c_2 + 1)...(2c_k + 1)$
即 $Ans = (2c_1 + 1)(2c_2 + 1)...(2c_k + 1)$

4.15.4 具有 N 个不同因子的最小正整数

$N <= 5e4$

```
const int N = 50050;
const int p[20] =
{
   2, 3, 5, 7, 11,
   13, 17, 19, 23, 29,
   31, 37, 41, 43, 47,
   53, 59, 61, 67, 71
};
double logp[20];
double f[505][20];
int d[505];
int cnt;
int A[100000], len;
void mul(int x)
{
   int v = 0;
   for (int i = 0; i < len; ++i)
   {
      v = (A[i] = A[i] * x + v) / 10;
      A[i] %= 10;
   }
   while (v) A[len++] = v % 10, v /= 10;
}
signed main()
{
   int n, m = 0;
   cin >> n;
   for (int i = 1; i <= n; ++i) if (!(n % i)) d[m++] = i;
   for (int i = 0; i < 20; ++i) f[0][i] = .0;
   for (int i = 0; i < 20; ++i) logp[i] = log(p[i]);
   for (int i = 1; i < m; ++i)
   {
      for (int k = 0; k < 20; ++k)
      f[i][k] = 1e9;
      for (int j = 0; j < i; ++j) if (!(d[i] % d[j]))
      {
         int t = d[i] / d[j];
         for (int k = 1; k < 20; ++k)
         f[i][k] = std::min(f[i][k], f[j][k - 1] + logp[k - 1] * (t - 1));
```

```
40          }
41      }
42      A[0] = len = 1;
43      int j = 0;
44      for (int i = 0; i < 20; ++i) if (f[m - 1][i] < f[m - 1][j]) j = i;
45      for (int i = m - 1, nxt; i; i = nxt, --j)
46      {
47          for (nxt = 0; d[i] % d[nxt] || f[i][j] < f[nxt][j - 1]
48          + logp[j - 1] * (d[i] / d[nxt] - 1) - 1e-5; ++nxt);
49          for (int k = 1; k < d[i] / d[nxt]; ++k)
50          mul(p[j - 1]);
51      }
52      while (len--) cout << A[len];
53      cout << '\n';
54      return 0;
55  }
```

4.16 线性基

4.16.1 总结

线性基是啥?

- 线性基是一个**数的集合**，并且每个序列都拥有**至少一个**线性基

线性基三大性质

1. 原序列里面的任意一个数都可以由线性基里面的一些数异或得到
2. 线性基里面的任意一些数异或起来都不能得到 0
3. 线性基里面的数的个数唯一，并且在保持性质一的前提下，数的个数是最少的

关于性质三的证明:

假如序列里面的所有元素都可以插入到线性基里面

显然如果是这种情况的话，不管是用什么顺序将序列里的数插入到线性基里，线性基中的元素一定与原序列元素数量相同。所以性质 3 成立。

假如序列里面的一些元素不能插入到线性基里面

我们设 x 不能插入到线性基里面，那么一定满足形如 $d[a] \oplus d[b] \oplus d[c] = x$ 的式子

那我们尝试将插入顺序改变，变成：$d[a]$ $d[b]$ x $d[c]$ ，那么显然，$d[c]$ 是不可能插入成功的，简单的证明：

$\because d[a] \oplus d[b] \oplus d[c] = x$

$\therefore d[a] \oplus d[b] \oplus x = d[c]$（根据上面那条并没有什么卵用的异或性质）

原来是 x 插入不进去，改变顺序后，$d[c]$ 插入不进去，也就是说，对于插入不进去的元素，改变插入顺序后，要么还是插入不进去，要么就是插入进去了

同时另一个原来插入的进去的元素插入不进去了，所以，**可以插入进去的元素数量一定是固定的不可能因为一个元素而导致多个元素不能插入，就算去掉这个元素，也只能插入那多个元素中的一个!**

如果你去掉线性基里面的任意一个数，都会使得原序列里的一些（或一个）数无法通过用线性基里的元素异或得到

所以，每一个元素都是必要的，换句话说，这里面没有多余的元素，所以，这个线性基的元素个数在保持性质 1 的前提下，一定是最少的。

4.16.2 验证存在性 + 最小值 + 最大值 + 第 K 小

给定 n 个整数（数字可能重复），求在这些数中选取任意个，使得他们的异或和最大、最小、第 K 小

- 求第 K 小之前要 *rebuild* 一下（优化线性基）
- *zero* 是用来判断 0 是否可以得到 (当某个数插入失败时 *zero* 便可以得到)
- b 为线性基, nb 用来求第 K 小异或值, tot 为 nb 元素个数 (即行数)
- 异或可得到的总数为 $sum = C_{tot}^{1} + C_{tot}^{2} + ... + C_{tot}^{tot} + zero = 2^{tot} - 1 + zero$

```
#include<bits/stdc++.h>
#define rep(i , a , b) for(int i = a ; i <= b ; i ++)
#define ll long long
using namespace std;
struct Linear_Basis
{
    ll b[63], nb[63], tot; //b为线性基 , nb用来求第 K 小异或值 , tot为 nb 元素个数
    bool zero = false; // 判断是否存在 0
    void Init() //初始化
    {
        tot = 0;
        zero = false;
        memset(b, 0, sizeof(b));
        memset(nb, 0, sizeof(nb));
    }
    void Ins(ll x) //插入
    {
        for(int i = 62; i >= 0; i--)
        {
            if(x & (1ll << i))
            {
                if(!b[i])
                {
                    b[i] = x;
                    return;
                }
                x ^= b[i];
            }
        }
        // x 插入失败
        zero = true;
        return;
    }
    bool Fin(ll x) //验证存在性
    {
        if(x == 0 && b[0]) return 1;
        for(int i = 62; i >= 1; i--)
        {
            int j = i - 1;
            if(x & (1 << j))
            {
```

```
            x ^= b[i];
            if(!x) return 1;
        }
    }
    return 0;
}
ll Max() //求最大值
{
    ll res = 0;
    for(int i = 62; i >= 0; i--)
    {
        res = max(res, res ^ b[i]);
    }
    return res;
}
ll Min() //求最小值
{
    if(zero) return 0;
    for(int i = 0; i <= 62; i++)
    {
        if(b[i]) return b[i];
    }
}
ll Query_Max(ll x) // 异或 x 可得到的最大值
{
    ll res = x;
    for(int i = 62 ; i >= 0 ; i --){
        res = max(res , res ^ b[i]);
    }
    return res;
}
ll Query_Min(ll x) // 异或 x 可得到的最小值
{
    ll res = x;
    for(int i = 0 ; i <= 62 ; i ++){
        if(b[i]) res ^= b[i];
    }
    return res;
}
void Rebuild()
{
    for(int i = 62; i >= 0; i--)
    {
        if(b[i] == 0) continue;
        for(int j = i - 1; j >= 0; j--)
        {
            if(b[j] == 0) continue;
            if(b[i] & (1ll << j)) b[i] ^= b[j];
        }
    }
    for(int i = 0; i <= 62; i++)
    {
        if(b[i]) nb[tot++] = b[i];
```

```
95          }
96      }
97      ll Kth_Max(ll k) //第K大
98      {
99          if(zero) k--;
100         ll res = 0;
101         if(k == 0) return 0;
102         if(k >= (1ll << tot)) return -1;
103         for(int i = 62; i >= 0; i--)
104         {
105             if(k & (1ll << i)) res ^= nb[i];
106         }
107         return res;
108     }
109 } LB;
110 ll a[55];
111 signed main()
112 {
113     LB.Init();
114     ll n , ma = 0;
115     cin >> n;
116     rep(i , 1 , n) cin >> a[i] , LB.Ins(a[i]);
117     cout << LB.Max() << '\n';
118     return 0;
119 }
```

4.16.3 最大异或和路径

- 求出一条从 1 号节点到 N 号节点的路径，使得路径上经过的边的权值的 XOR 和最大

大概思路：

- 首先根据异或的性质，一条边或一个环走偶数遍的异或和都是 0 ，所以对于一个环，要么不走，要么只走一遍，那么可以构想出最优路径一定是一条从 1 到 n 的 **链**上套着 **若干环**，那么我们把每个环的异或和加进线性基，用这条链的异或和去异或线性基找到最大值就可以了
- 可是这个最优解的链怎么找呢，其实只要任意找一条从 1 到 n 的链就可以了，因为若存在两条从 1 到 n 的链，则它们已经构成了一个环，无论你选择哪一条作为基链，异或上这个环都会得到另一条链的异或和，所以就不用担心最优解的正确性了。

```
1  struct Linear_Basis
2  {
3      ll b[63], nb[63], tot; //b为线性基 , nb用来求第 K 小异或值 , tot为 nb 元素个数
4      bool zero = false; // 判断是否存在 0
5      void Init() //初始化
6      {
7          tot = 0;
8          zero = false;
9          memset(b, 0, sizeof(b));
10         memset(nb, 0, sizeof(nb));
11     }
12     void Ins(ll x) //插入
```

```
13      {
14          for(int i = 62; i >= 0; i--)
15          {
16              if(x & (1ll << i))
17              {
18                  if(!b[i])
19                  {
20                      b[i] = x;
21                      return;
22                  }
23                  x ^= b[i];
24              }
25          }
26          // x 插入失败
27          zero = true;
28          return;
29      }
30      bool Fin(ll x) //验证存在性
31      {
32          if(x == 0 && b[0]) return 1;
33          for(int i = 62; i >= 1; i--)
34          {
35              int j = i - 1;
36              if(x & (1 << j))
37              {
38                  x ^= b[i];
39                  if(!x) return 1;
40              }
41          }
42          return 0;
43      }
44      ll Max() //求最大值
45      {
46          ll res = 0;
47          for(int i = 62; i >= 0; i--)
48          {
49              res = max(res, res ^ b[i]);
50          }
51          return res;
52      }
53      ll Min() //求最小值
54      {
55          if(zero) return 0;
56          for(int i = 0; i <= 62; i++)
57          {
58              if(b[i]) return b[i];
59          }
60      }
61      ll Query_Max(ll x) // 异或 x 可得到的最大值
62      {
63          ll res = x;
64          for(int i = 62 ; i >= 0 ; i --){
65              res = max(res , res ^ b[i]);
```

```
66          }
67          return res;
68      }
69      ll Query_Min(ll x) // 异或 x 可得到的最小值
70      {
71          ll res = x;
72          for(int i = 0 ; i <= 62 ; i ++){
73              if(b[i]) res ^= b[i];
74          }
75          return res;
76      }
77      void Rebuild()
78      {
79          for(int i = 62; i >= 0; i--)
80          {
81              if(b[i] == 0) continue;
82              for(int j = i - 1; j >= 0; j--)
83              {
84                  if(b[j] == 0) continue;
85                  if(b[i] & (1ll << j)) b[i] ^= b[j];
86              }
87          }
88          for(int i = 0; i <= 62; i++)
89          {
90              if(b[i]) nb[tot++] = b[i];
91          }
92      }
93      ll Kth_Max(ll k) //第K大
94      {
95          if(zero) k--;
96          ll res = 0;
97          if(k == 0) return 0;
98          if(k >= (1ll << tot)) return -1;
99          for(int i = 62; i >= 0; i--)
100         {
101             if(k & (1ll << i)) res ^= nb[i];
102         }
103         return res;
104     }
105 } LB;
106 const int N = 1e5 + 10;
107 struct Edge{
108     int nex , to;
109     ll w;
110 }edge[N << 1];
111 int head[N] , TOT;
112 void add_edge(int u , int v , ll w)
113 {
114     edge[++ TOT].nex = head[u];
115     edge[TOT].to = v;
116     edge[TOT].w = w;
117     head[u] = TOT;
118 }
```

```cpp
int n , m , vis[N];
ll dis[N];
void dfs(int u , int far)
{
    vis[u] = 1;
    for(int i = head[u] ; i ; i = edge[i].nex)
    {
        int v = edge[i].to ;
        ll w = edge[i].w;
        if(v == far) continue ;
        if(vis[v])
        {
            LB.Ins(dis[v] ^ dis[u] ^ w);
            continue ;
        }
        dis[v] = dis[u] ^ w;
        dfs(v , u);
    }
}
signed main()
{
    LB.Init();
    cin >> n >> m;
    rep(i , 1 , m)
    {
        int u , v;
        ll w;
        cin >> u >> v >> w;
        add_edge(u , v , w) , add_edge(v , u , w);
    }
    dfs(1 , 0);
    cout << LB.Query_Max(dis[n]) << '\n';
    return 0;
}
```

4.16.4 区间询问异或和或上 K 的最大值

问题:

给定一个长度为 N 的数组 A，M 次询问和一个常数 K
每次询问给出区间 $[L, R]$，要求在区间中找出若干个数
使得这些数的异或和或上 K 的值最大，即 `ans = MAX(sum | K)`

实现：线段树套线性基
思路:

区间操作 —— 线段树
异或操作 —— 线性基

首先 $ans = K|sum$，所以并不是 sum(异或和) 越大，ans 就越大

举个例子：$K = 1000$，$sum1 = 1000$，$sum2 = 0111$
那么显然 $sum1 > sum2$，$(K|sum1) < (K|sum2)$

我们发现按位或运算，只要有一个 1 就行了，两个 1 和一个 1 没有区别，所以对于 K 中二进制下为 1 的所有位，线性基得到的 sum，这一位都没必要是 1，所以我们可以把整个 A 数组的 K 是 1 的所有二进制位全部变成 0。

这样等于删掉了不需要的 1 ，因为这些 1 存在在这个线性基里时，会导致最大异或和更大，但是这个更大的数对我们最终的答案没有贡献。

所以我们可以将 k 取反，然后把所有数在加入线性基之前，全部按位异或运算一遍，这样把没用的 1 全部删掉，再加入线性基

这样求出来的 max_sum（最大异或和），每一位都能使得 K 变为更大的数了

```
const int N = 50007, mod = 1e9 + 7, INF = 2.1e9;
const int Base = 27 * 2;//线性基数组大小，开大一点，嘻嘻嘻
//s < 2 ^ {63} - 1
//说明线性基最多有62个向量
struct LinearBase
{
    int dimension = 27; // dimension 维数，就是线性基的维数 = logs, (s为元素最大值)
    //dimension一定要是logs,错一点就会WA呜呜呜
    //线性基数组
    ll a[Base + 7];//注意这里要加一点，因为下面的循环也是这个数
    LinearBase()
    {
        fill(a, a + Base + 7, 0);
    }
    LinearBase(ll *x, int n)
    {
        build(x, n);
    }
    void insert(ll t)
    {
        // 逆序枚举二进制位
        for(int i = dimension; i >= 0; -- i)
        {
            if(t == 0) return ;
            // 如果 t 的第 i 位为 0，则跳过
            if(!(t >> i & 1)) continue;
            // 如果 a[i] != 0，则用 a[i] 消去 t 的第 i 位上的 1
            if(a[i]) t ^= a[i];
            else
            {
                // 找到可以插入 a[i] 的位置
                // 用 a[0...i - 1] 消去 t 的第 [0, i) 位上的 1
                // 如果某一个 a[k] = 0 也无须担心，因为这时候第 k 位
                //不存在于线性基中，不需要保证 t 的第 k 位为 0
                for(int k = 0; k < i; ++ k)
                if(t >> k & 1) t ^= a[k];
                // 用 t 消去 a[i + 1...L] 的第 i 位上的 1
                for(int k = i + 1; k <= dimension; ++ k)
                if(a[k] >> i & 1) a[k] ^= t;
                // 插入到 a[j] 的位置上
                a[i] = t;
                break;
```

```
43          }
44          // 此时 t 的第 i 位为 0，继续寻找其最高位上的 1
45      }
46      // 如果没有插入到任何一个位置上，则表明 t 可以由 a 中若干个元素的异或和表示出，即 t
            在 span(a) 中
47  }
48  // 数组 x 表示集合 S，下标范围 [1...n]
49  void build(ll *x, int n)
50  {
51      fill(a, a + Base + 7, 0);
52      for(int i = 1; i <= n; ++ i)
53      insert(x[i]);
54  }
55  ll query_max()
56  {
57      ll res = 0;
58      for(int i = 0; i <= dimension; ++ i)
59      res ^= a[i];
60      return res;
61  }
62  void mergefrom(const LinearBase &other)
63  {
64      for(int i = 0; i <= dimension; ++ i)
65      insert(other.a[i]);
66  }
67  static LinearBase merge(const LinearBase &a, const LinearBase &b)
68  {
69      LinearBase res = a;
70      for(int i = 0; i <= 27; ++ i)
71      res.insert(b.a[i]);
72      return res;
73  }
74 };
75 int n, m, q, k;
76 ll a[N];
77 struct Tree
78 {
79  int l, r;
80  LinearBase elem;//element
81 } tr[N];
82 void build(int p, int l, int r)
83 {
84  tr[p] = {l, r};
85  if(l == r)
86  {
87      tr[p].elem.insert(a[r]);
88      return ;
89  }
90  int mid = (l + r) >> 1;
91  build(p << 1, l, mid);
92  build(p << 1 | 1, mid + 1, r);
93  tr[p].elem = tr[p].elem.merge(tr[p << 1].elem, tr[p << 1 | 1].elem);
94 }
```

```
95  LinearBase query(int p, int l, int r)
96  {
97      if(l <= tr[p].l && tr[p].r <= r) return tr[p].elem;
98      int mid = (tr[p].l + tr[p].r) >> 1;
99      LinearBase res;
100     if(l <= mid) res = res.merge(res, query(p << 1, l, r));
101     if(r > mid) res = res.merge(res, query(p << 1 | 1, l, r));
102     return res;
103 }
104 signed main()
105 {
106     ios::sync_with_stdio(false);
107     cin.tie(0) , cout.tie(0);
108     int T = 1;
109     cin >> T;
110     while(T --)
111     {
112         cin >> n >> q >> k ;
113         rep(i , 1 , n)
114         {
115             cin >> a[i];
116             /*k的存在会对求线性基最大值时的主元产生影响，所以预处理一下，a[i]只保留k为0的
                    位，这样贡献最大*/
117             for(int j = 0; j < 27; ++ j)
118             {
119                 if((k >> j & 1) && (a[i] >> j & 1))
120                 a[i] ^= (1 << j);
121             }
122         }
123         build(1, 1, n);
124         while(q -- )
125         {
126             int l, r;
127             cin >> l >> r;
128             LinearBase res = query(1, l, r);
129             cout << (res.query_max() | k) << '\n';
130         }
131     }
132     return 0;
133 }
```

4.17 整除分块

4.17.1 总结

- **模型：** $\sum_{i=1}^{n} \lfloor \frac{n}{i} \rfloor$
  假设 $n = 8$，那么可得：

i	1	2	3	4	5	6	7	8
$8/i$	8	4	2	2	1	1	1	1

- **概念：**
  表中同样的值会连续出现，而相同的值所划分的区间成为一个块。
  整除的性质使得从 1 到 n 的表可根据数值划分为不同的块，且分块数远远小于 n。
  利用这种性质，我们可以推导出每个分块具体的左右端点位置在哪，这样就可以快速求解出来了。
  复杂度 $O(\sqrt{n})$

- **推导：**
  假设我们已知某一个分块的左端点 l，要求解出该分块的右端点 r。
  设该分块的数值为 k，对于该分块中的每个数 i，有 $k=\lfloor\frac{n}{i}\rfloor=\lfloor\frac{n}{l}\rfloor$，即 $i\times k\le n$。
  那么显然满足 $i\times k\le n$ 成立的最大的 i 就是我们要的右端点 r。
  于是可得：$\begin{cases} k=\lfloor\frac{n}{l}\rfloor \\ r=\max(i), i\times k\le n \end{cases}$
  推导得：$r=\lfloor\frac{n}{k}\rfloor=\left\lfloor\frac{n}{\lfloor\frac{n}{l}\rfloor}\right\rfloor$

- **变形 0：**
  已知 n,m，求 $\sum_{i=1}^{min(n,m)}\lfloor\frac{n}{i}\rfloor\lfloor\frac{m}{i}\rfloor$
  $\begin{cases} k1=\lfloor\frac{n}{l}\rfloor, k2=\lfloor\frac{m}{l}\rfloor \\ r=\max(i), i\times k1\le n, i\times k2\le m \end{cases}$
  $r=min(\lfloor\frac{n}{k1}\rfloor,\lfloor\frac{m}{k2}\rfloor)$

- **变形 1：**
  已知 n,a,b，求 $\sum_{i=1}^{n}\lfloor\frac{n}{ai+b}\rfloor$
  令 $i'=a\times i+b$，先求出 $\lfloor\frac{n}{i'}\rfloor$ 的整除分块
  可得 $k=\lfloor\frac{n}{a\times l+b}\rfloor$，$r'=\lfloor\frac{n}{k}\rfloor=\left\lfloor\frac{n}{\lfloor\frac{n}{a\times l+b}\rfloor}\right\rfloor$
  $\begin{cases} i'=a\times i+b \\ r'=\max(i') \end{cases}\rightarrow\begin{cases} a\times i=i'-b \\ i=\dfrac{i'-b}{a} \end{cases}\rightarrow r=max(i)=max(\left\lfloor\frac{i'-b}{a}\right\rfloor)=\lfloor\frac{r'-b}{a}\rfloor$
  $$r=\left\lfloor\frac{\left\lfloor\frac{n}{\lfloor\frac{n}{a\times l+b}\rfloor}\right\rfloor-b}{a}\right\rfloor$$

- **变形 2：**
  做法同上。
  已知 n，求 $\sum_{i=1}^{n}\lfloor\frac{n}{i^2}\rfloor$
  令 $i'=i^2$，先求出 $\lfloor\frac{n}{i'}\rfloor$ 的整除分块
  可得 $k=\lfloor\frac{n}{l^2}\rfloor$，$r'=\lfloor\frac{n}{k}\rfloor=\left\lfloor\frac{n}{\lfloor\frac{n}{l^2}\rfloor}\right\rfloor$
  $\begin{cases} i=\sqrt{i'} \\ r'=\max(i') \end{cases}\rightarrow r=max(i)=max(\lfloor\sqrt{i'}\rfloor)=\lfloor r'\rfloor=\lfloor\sqrt{\left\lfloor\frac{n}{\lfloor\frac{n}{l^2}\rfloor}\right\rfloor}\rfloor$

- **变形 3：**
  已知 n，求 $\sum_{i=1}^{n}\lceil\frac{n}{i}\rceil$
  当 n 整除 i 时，$\lceil\frac{n}{i}\rceil=\lfloor\frac{n}{i}\rfloor$
  当 n 不整除 i 时，$\lceil\frac{n}{i}\rceil=\lfloor\frac{n}{i}\rfloor+1$
  于是我们可以用 $\lfloor\frac{n+i-1}{i}\rfloor$ 来代替 $\lceil\frac{n}{i}\rceil$
  原式就转换为了 $\sum_{i=1}^{n}\lfloor\frac{n+i-1}{i}\rfloor$

那么有 $\begin{cases} k = \lfloor \frac{n+l-1}{l} \rfloor \\ r = \max(i), i \times k \le n + i - 1 \end{cases}$

$\because i \times k \le n + i - 1 \Rightarrow i \times (k-1) \le n - 1$

$\therefore r = \lfloor \frac{n-1}{k-1} \rfloor = \left\lceil \frac{n-1}{\lfloor \frac{n+l-1}{l} \rfloor - 1} \right\rceil$

4.17.2 余数求和

- 已知 n, k，求解：$G(n,k) = \sum_{i=1}^{n} k \bmod i$
- $1 \le n, k \le 10^9$

$ans = n \times k - \sum_{i=1}^{n} \lfloor \frac{n}{i} \rfloor \times i$，对于每一个块都是一个公差为 $\lfloor \frac{n}{i} \rfloor$ 等差数列

```
signed main()
{
   long long n , k;
   scanf("%lld %lld" , &n , &k);
   long long ans = n * k ;
   for(long long l = 1 , r ; l <= n ; l = r + 1)
   {
      if(k / l != 0) r = min(n , k / (k / l)) ;
      else r = n;
      ans -= (k / l) * (r - l + 1) * (l + r) / 2;
   }
   printf("%lld\n" , ans);
   return 0;
}
```

4.17.3 有限小数对数

- 求出有序整数对 (x, y) 的个数，使得 $1 \le x, y \le n$ 且 $\frac{x}{y}$ 可以表示为十进制有限小数。
- $1 \le n \le 10^{12}$

对于一个有限小数，当分数为最简形式时分母只包含 2 、5 两个质因子，所以我们可以用 $\lfloor \frac{bc}{ac} \rfloor$ 来表示任何一个有限小数（其中 a 为 2,5 两个质因子构成的数，c 为不包含 2,5 两个质因子的数，b 随意）。

于是这道题就可以枚举 c，那么 ans $= \sum_{c=1}^{n} \lfloor \frac{n}{c} \rfloor \times f(\lfloor \frac{n}{c} \rfloor)$（其中 $\lfloor \frac{n}{c} \rfloor$ 为 b 的取值范围，$f(\lfloor \frac{n}{c} \rfloor)$）为 $[1, \lfloor \frac{n}{c} \rfloor]$ 内 a 的个数。

不过值得注意的是这里的 c 是不包含 2,5 两个质因子的数，所以对于每个块我们要减去包含 2,5 两个质因子的数。实现起来也很简单，根据容斥原理减去块内 2 的倍数的数，再减去块内 5 的倍数的数，然后再加上块内 10 的倍数的数即可。

至于 $f(\lfloor \frac{n}{c} \rfloor)$，我们可以先用 log 的复杂度求出所有 $1 \sim 10^{12}$ 的所有 a，并记录 a 的个数

因为随着 c 增大 $f(\lfloor \frac{n}{c} \rfloor)$ 是递减的，所以可以用单指针求出

```
const int N = 3e5 + 10;
int a[N] , m , now = 1;
int f(int x)
{
   while(a[now] > x && now ) now -- ;
   return now ;
}
signed main()
{
```

```
    int n ;
    cin >> n;
    for(int i = 1 ; i <= n ; i *= 2)
    for(int j = 1 ; j * i <= n ; j *= 5)
    a[++ m] = i * j;
    now = m;
    sort(a + 1 , a + 1 + m);
    int ans = 0;
    for(int l = 1 , r ; l <= n ; l = r + 1)
    {
        r = n / (n / l);
        int cnt1 = r - l + 1 , cnt10 = r / 10 - (l - 1) / 10;
        int cnt2 = r / 2 - (l - 1) / 2 , cnt5 = r / 5 - (l - 1) / 5;
        int sum = cnt1 - cnt2 - cnt5 + cnt10;
        ans += sum * (n / l) * f(n / l);
    }
    cout << ans << '\n';
    return 0;
}
```

4.18 中国剩余定理

4.18.1 总结

中国剩余定理是一种用于求解诸如

$$\begin{cases} x \equiv q_1 \ (mod \ p_1) \\ x \equiv q_2 \ (mod \ p_2) \\ \cdots\cdots \\ x \equiv q_k \ (mod \ p_k) \end{cases}$$

形式的同余方程组的定理

其中，$p_1, p_2, ..., p_k$ 为**两两互质**的整数，我们的目的，是找出 x 的**最小非负整数**解

> 若求的是大于某个数 (假设这个数是 d) 的最小整数解
> 则可定义 $LCM = p1 \times p2 \times ... \times pn$
> 那么当 $x <= d$ 时，让 $x+ = LCM$，$x\% = LCA$ 即可
> 不过 mod 完后 x 的值可能为 0 , 若为 0 则 LCA 为答案

相比中国剩余定理，扩展中国剩余定理可无视 $p_1, p_2, ..., p_k$ **两两互质**的条件

4.18.2 中国剩余定理

$1 \le n \le 10$，$0 \le q_i < p_i \le 1000000$，$1 \le \prod a_i \le 10^{18}$ (p 是模数 q 是余数)

```
ll n , p[20], q[20];
ll ex_gcd(ll a, ll b, ll &x, ll &y)
{
    if (!b)
    {
        x = 1, y = 0;
        return a;
    }
    ll d = ex_gcd(b, a % b, x, y);
    ll temp = x;
```

```
    x = y, y = temp - a / b * y;
    return d;
}
ll CRT(ll *p, ll *q, ll n /*方程组数*/)
{
    ll x, y, a = 0, m, LCM = 1;
    rep(i , 1 , n) LCM *= p[i];
    rep(i , 1 , n)
    {
        m = LCM / p[i];
        ex_gcd(p[i], m, x, y);
        a = (a + y * m * q[i]) % LCM;
    }
    return a > 0 ? a : a + LCM;
}
signed main()
{
    cin >> n;
    rep(i, 1, n) cin >> p[i] >> q[i];
    cout << CRT(p, q, n) << '\n';
    return 0;
}
```

4.18.3 扩展中国剩余定理

$1 \le n \le 10^5$, $1 \le a_i \le 10^{12}$, $0 \le b_i < a_i$, 保证所有 a_i 的最小公倍数不超过 10^{18}

```
const int N = 1e5 + 10;
ll n , s[N] , p[N];
ll mul(ll a,ll b,ll mod)
{
    ll res = 0;
    while(b)
    {
        if(b & 1) res = (res + a) % mod;
        a = (a + a) % mod;
        b >>= 1;
    }
    return res;
}
ll exgcd(ll a,ll b,ll &x,ll &y)
{
    if(b==0)
    {
        x = 1 , y = 0;
        return a;
    }
    ll g = exgcd(b , a % b , x , y);
    ll tp = x;
    x = y , y = tp - a / b * y;
    return g;
}
ll EX_CRT(ll *s , ll *p , ll n)
```

```
{
    ll x , y , k , LCM = p[1] , ans = s[1];
    rep(i , 2 , n)
    {
        ll a = LCM , b = p[i] , c = (s[i] - ans % b + b) % b;
        ll g = exgcd(a , b , x , y) , bg = b / g;
        if(c % g != 0) return -1;
        x = mul(x , c / g , bg);
        ans += x * LCM , LCM *= bg;
        ans = (ans % LCM + LCM) % LCM;
    }
    return ans;
}
signed main()
{
    cin >> n;
    rep(i , 1 , n) cin >> p[i] >> s[i];
    cout << EX_CRT(s , p , n) << '\n';
    return 0;
}
```

5 图论

5.1 小经验

1. - **当边权很小 (或者为大小写字母) 时，一定要考虑是否可以从边权入手**
   - 比如用边权来存图：$G[u][w].pb(v)$，w 为边权
   - 根据边权写暴力 *or* *dp*?
2. **有必要的时候可以用 *set* 建图**

5.2 K 短路

5.2.1 Astar

有向图，s 到 t 的第 k 短路。

```
int n , m , s , t , k ;
vector<pii> g1[N] , g2[N] ;
struct Dij
{
    int dis[N] ;
    priority_queue<pii , vector<pii> , greater<pii> > q ;
    void init()
    {
        memset(dis , inf , sizeof(dis)) ;
    }
    void dijkstra(int s)
    {
        dis[s] = 0 ;
        q.push(make_pair(0 , s)) ;
        while(!q.empty())
```

```
16        {
17            pii p = q.top() ;
18            q.pop() ;
19            int u = p.second ;
20            if(p.first != dis[u]) continue ; //优化，不用旧值更新。
21            for(vector<pii>::iterator x = g2[u].begin() ; x != g2[u].end() ; x ++)
22            //for(auto x : g2[u])
23            {
24                int v = (*x).fi , w = (*x).se ;
25                if(dis[v] > dis[u] + w)
26                {
27                    dis[v] = dis[u] + w ;
28                    q.push(make_pair(dis[v] , v)) ;
29                }
30            }
31        }
32    }
33 } dij ;
34 struct node
35 {
36    int u , g , h ;
37    bool operator < (const node &s) const
38    {
39        return g + h > s.g + s.h ;
40    }
41 } ;
42 priority_queue<node> q ;
43 int num[N] ;
44 int astar()
45 {
46    mem0(num) ;
47    while(!q.empty()) q.pop() ;
48    q.push(node{s , 0 , dij.dis[s]}) ;
49    while(!q.empty())
50    {
51        int u = q.top().u , g = q.top().g ;
52        q.pop() ;
53        num[u] ++ ;
54        if(num[u] == k && u == t) return g ;
55        if(num[u] > k) continue ;
56        for(vector<pii>::iterator x = g1[u].begin() ; x != g1[u].end() ; x ++)
57        //for(auto x : g1[u])
58        {
59            int v = (*x).fi , w = (*x).se ;
60            q.push(node{v , g + w , dij.dis[v]}) ;
61        }
62    }
63    return -1 ;
64 }
65 int main()
66 {
67    rr(n , m) ;
68    rep(i , 1 , m)
```

```
    {
        int u , v , w ;
        rr(u , v) , r(w) ;
        g1[u].pb({v , w}) ;
        g2[v].pb({u , w}) ;
    }
    rr(s , t) , r(k) ;
    if(s == t) k ++ ;
    dij.init() ;
    dij.dijkstra(t) ;
    we(astar()) ;
    return 0 ;
}
```

5.3 K 级祖先

5.3.1 总结

- **重剖**找 K 级祖先和**倍增**找 K 级祖先的理论复杂度都是 log
- 但**重剖**的常数要比**倍增**来的小
- 不过对于森林来说**倍增**用起来会比较方便，因为**重剖**需要额外记录每棵树的 dfn_id
- 至于**长链剖分**虽然复杂度是 $O(1)$ ，但是它得用到**倍增**，以至于内存消耗大，所以不如就写个**重剖**来的 $easy$

5.3.2 倍增法

```
int dep[N] , f[N][30];
void dfs(int u , int far)
{
    dep[u] = dep[far] + 1;
    f[u][0] = far;
    for(int i = 1 ; (1 << i) <= dep[u] ; i ++)
        f[u][i] = f[f[u][i - 1]][i - 1];
    for(int i = head[u] ; i ; i = edge[i].nex)
    {
        int v = edge[i].to;
        if(v == far) continue ;
        dfs(v , u);
    }
}
int get_far(int x , int k)
{
    if(!k) return x;
    int t = dep[x] - k;
    per(i , 20 , 0) if(dep[f[x][i]] > t) x = f[x][i];
    return f[x][0];
}
```

5.3.3 重链剖分

```
int dep[N] , sz[N] , hson[N];
int fa[N] , tp[N] , dfn[N] , dfn_id[N] , tot;
void dfs1(int u , int far)
{
    sz[u] = 1 , fa[u] = far;
    dep[u] = dep[far] + 1;
    for(int i = head[u] ; i ; i = edge[i].nex)
    {
        int v = edge[i].to;
        if(v == far) continue ;
        dfs1(v , u);
        sz[u] += sz[v];
        if(sz[v] > sz[hson[u]]) hson[u] = v;
    }
}
void dfs2(int u , int far , int top)
{
    tp[u] = top;
    dfn[u] = ++ tot;
    dfn_id[tot] = u;
    if(!hson[u]) return ;
    dfs2(hson[u] , u , top);
    for(int i = head[u] ; i ; i = edge[i].nex)
    {
        int v = edge[i].to;
        if(v == far || v == hson[u]) continue ;
        dfs2(v , u , v);
    }
}
int get_far(int x , int k)
{
    if(!k) return x;
    int t = dep[x] - k;
    if(t <= 0) return 0;
    while(dep[tp[x]] > t) x = fa[tp[x]];
    t = dep[x] - t;
    return dfn_id[dfn[x] - t];
}
```

5.3.4 长链剖分

```
const int N = 5e5 + 10;
int g[N], dep[N], f[N][21], hson[N], high[N], top[N];
vector<int> G[N], U[N], V[N];
void dfs1(int u , int far)
{
    high[u] = dep[u] = dep[far] + 1;
    for (auto v : G[u])
    {
        if(v == far) continue ;
```

```
        f[v][0] = u;
        for (int i = 0; f[v][i]; i++) f[v][i + 1] = f[f[v][i]][i];
        dfs1(v , u);
        if (high[v] > high[u]) high[u] = high[v], hson[u] = v;
    }
}
void dfs2(int u, int tp , int far)
{
    top[u] = tp;
    if (u == tp)
    {
        for (int i = 0, o = u; i <= high[u] - dep[u] ; i ++)
        U[u].push_back(o), o = f[o][0];
        for (int i = 0, o = u; i <= high[u] - dep[u] ; i ++)
        V[u].push_back(o), o = hson[o];
    }
    if (hson[u]) dfs2(hson[u], tp , u);
    for (auto v : G[u]) if (v != hson[u] && v != far) dfs2(v, v , u);
}
int get_far(int u, int k)
{
    if (!k) return u;
    u = f[u][g[k]], k -= 1 << g[k], k -= dep[u] - dep[top[u]], u = top[u];
    return k >= 0 ? U[u][k] : V[u][-k];
}
void init(int n)
{
    g[0] = -1;
    for(int i = 1 ; i <= n ; i ++) g[i] = g[i >> 1] + 1;
}
```

5.4 LCA

5.4.1 tarjan 离线

```
const int N = 5e5 + 10;
int father[N] , vis[N];
vector<int>mat[N];
vector<int>Q[N];
struct pnode{
    int x , y;
    int lca;
}query[N];
void init()
{
    rep(i , 1 ,N)
    father[i] = i , vis[i] = 0;
}
int find(int x)
{
    if(father[x] == x) return x;
    else return father[x] = find(father[x]);
```

```
}
void tarjan(int u)
{
    vis[u] = 1;
    for(auto it : Q[u])
    {
        if(query[it].x == u)
        {
            if(vis[query[it].x])
            query[it].lca = find(query[it].y);
        }
        else
        {
            if(vis[query[it].y])
            query[it].lca = find(query[it].x);
        }
    }
    for(auto v : mat[u])
    {
        if(vis[v])
        continue;
        tarjan(v);
        father[v] = u;

    }
}
signed main()
{
    init();
    int n , m , s;
    cin >> n >> m >> s;
    rep(i , 1 , n - 1)
    {
        int x , y ;
        cin >> x >> y;
        mat[x].pb(y) , mat[y].pb(x);
    }
    rep(i , 1 , m)
    {
        cin >> query[i].x >> query[i].y;
        Q[query[i].x].pb(i) , Q[query[i].y].pb(i);
    }
    tarjan(s);
    rep(i , 1 , m)
    cout << query[i].lca << '\n';
    return 0;
}
```

5.4.2 倍增法

- $f[i][k]$ 表示节点 i 的上 2^k 层的祖先是哪个点

```cpp
const int N = 5e5 + 10;
struct Edge{
    int nex , to;
}edge[N << 1];
int dep[N] , f[N][30] , head[N] , tot , ans[N] , siz[N];
void add_edge(int u , int v)
{
    edge[++ tot].nex = head[u];
    edge[tot].to = v;
    head[u] = tot;
}
void dfs(int u , int far)
{
    siz[u] = 1;
    dep[u] = dep[far] + 1;
    f[u][0] = far;
    for(int i = 1 ; (1 << i) <= dep[u] ; i ++)
        f[u][i] = f[f[u][i - 1]][i - 1];
    for(int i = head[u] ; i ; i = edge[i].nex)
    {
        int v = edge[i].to;
        if(v == far) continue ;
        dfs(v , u);
        siz[u] += siz[v];
    }
}
int lca(int x , int y)
{
    if(dep[x] < dep[y]) swap(x , y);
    per(i , 20 , 0)
    {
        if(dep[f[x][i]] >= dep[y]) x = f[x][i];
        if(x == y) return x;
    }
    per(i , 20 , 0)
        if(f[x][i] != f[y][i])
            x = f[x][i] , y = f[y][i];
    return f[x][0];
}
```

5.4.3 树剖法

```cpp
#include<bits/stdc++.h>
#define ios std::ios::sync_with_stdio(false)
#define rep(i,a,n) for (int i=a;i<=n;i++)
#define per(i,n,a) for (int i=n;i>=a;i--)
#define int long long
#define ll long long
using namespace std;
const ll INF (0x3f3f3f3f3f3f3f3fll);
const int inf (0x3f3f3f3f);
const int N = 5e5 + 10;
```

```
11 struct Edge
12 {
13     int nex , to;
14 } edge[N << 1];
15 int head[N] , TOT;
16 void add_edge(int u , int v)
17 {
18     edge[++ TOT].nex = head[u] ;
19     edge[TOT].to = v;
20     head[u] = TOT;
21 }
22 int n , m , s;
23 int dep[N] , sz[N] , hson[N] , tp[N] , fa[N] , tot;
24 void dfs1(int u , int far)
25 {
26     sz[u] = 1 , dep[u] = dep[far] + 1 , fa[u]= far;
27     for(int i = head[u] ; i ; i = edge[i].nex)
28     {
29         int v = edge[i].to;
30         if(v == far) continue ;
31         dfs1(v , u);
32         sz[u] += sz[v];
33         if(sz[v] > sz[hson[u]]) hson[u] = v;
34     }
35 }
36 void dfs2(int u , int far , int top)
37 {
38     tp[u] = top;
39     if(!hson[u]) return ;
40     dfs2(hson[u] , u , top);
41     for(int i = head[u] ; i ; i = edge[i].nex)
42     {
43         int v = edge[i].to;
44         if(v == far || v == hson[u]) continue ;
45         dfs2(v , u , v);
46     }
47 }
48 int lca(int x , int y)
49 {
50     while(tp[x] != tp[y])
51     {
52         if(dep[tp[x]] < dep[tp[y]]) swap(x , y);
53         x = fa[tp[x]];
54     }
55     if(dep[x] > dep[y]) swap(x , y);
56     return x;
57 }
58 signed main()
59 {
60     cin >> n >> m >> s;
61     rep(i , 2 , n)
62     {
63         int u , v;
```

```
64        cin >> u >> v;
65        add_edge(u , v) , add_edge(v , u);
66    }
67    dfs1(s , 0);
68    dfs2(s , 0 , s);
69    while(m --)
70    {
71        int u , v;
72        cin >> u >> v;
73        cout << lca(u , v) << '\n';
74    }
75    return 0;
76 }
```

5.5 次小生成树

5.5.1 kruskal+lca

复杂度$O(MlogM + Mlog(M))$

- sum 为最小生成树的权值和（得到 sum 之前要先 $work$！）
- 假设现在要把一条非树边 (u, v, c) 加入最小生成树，想必要去掉一条原生成树中 $u->v$ 的边，显然去掉最大边效果是最好的
- 所以在最小生成树上跑一个倍增 DP，$d[i][j]$ 表示 j 的 2^i 次祖先到 j 的路径中最大的值
- 然后再枚举每一条非树边 $vis[i] = 0$, 然后再取 min 就好

```
1  const int N = 3e5 + 10 , INF = 0x3f3f3f3f3f3f3f3fll;
2  struct node
3  {
4      int w, u, v, id, vis;
5      bool operator < (const node& A) const
6      {
7          return w < A.w;
8      }
9  } edge[N];
10 struct Edge
11 {
12     int nex, to, w;
13 } e[N << 1];
14 int head[N], TOT;
15
16 int dep[N] , fa[N][23] , f1[N][23] , f2[N][23]; // f1 维护最大值 , f2 维护次大值
17 vector<int> G[N];
18 void add_edge(int u, int v, int w)
19 {
20     e[++TOT].nex = head[u];
21     e[TOT].to = v;
22     head[u] = TOT;
23     e[TOT].w = w;
24 }
25 void dfs(int u,int far)
```

```
{
    dep[u] = dep[far] + 1;
    fa[u][0] = far;
    for (int i = 1; (1 << i) <= dep[u]; i++)
    {
        fa[u][i] = fa[fa[u][i-1]][i-1];
        f1[u][i] = max(f1[u][i - 1], f1[fa[u][i - 1]][i - 1]);
        if (f1[u][i - 1] == f1[fa[u][i - 1]][i - 1])
        f2[u][i] = max(f2[u][i - 1], f2[fa[u][i - 1]][i - 1]);
    }
    for (int i = head[u]; i ; i = e[i].nex)
    {
        int v = e[i].to, w = e[i].w;
        if (v == far) continue;
        f1[v][0] = w;
        dfs(v,u);
    }
}
int lca(int u,int v)
{
    if (dep[u] < dep[v]) swap(u,v);
    for(int i = 22; i >= 0; i--)
    {
        if (dep[fa[u][i]] >= dep[v])
        {
            u = fa[u][i];
        }
    }
    if (u == v) return u;
    for(int i = 22; i >= 0; i--)
    {
        if (fa[u][i] != fa[v][i])
        {
            u = fa[u][i];
            v = fa[v][i];
        }
    }
    return fa[u][0];
}
int get_max(int x, int y, int w)
{
    // w = -INF;  //取消注释则求非严格次小生成树
    int res = 0;
    for (int i = 22; i >= 0; i--)
    {
        if (dep[fa[x][i]] >= dep[y])
        {
            res = max(res, f1[x][i] == w ? f2[x][i]:f1[x][i]); // 防止次小生成树 = 最小
                生成树
            x = fa[x][i];
        }
    }
    return res;
```

```
}
int far[N] , mst , ans[N];
int find(int x)
{
    if (far[x] == x) return x;
    return far[x] = find(far[x]);
}
void init()
{
    for(int i = 1 ; i <= N - 10 ; i ++) far[i] = i , G[i].clear();
}
signed main()
{
    init();
    int n, m;
    cin >> n >> m;
    for (int i = 1; i <= m; i++)
    {
        int u, v, w;
        cin >> u >> v >> w;
        edge[i] = {w,u,v,i,0};
    }
    sort(edge + 1, edge + 1 + m);
    for (int i = 1; i <= n; i++) far[i] = i;
    for (int i = 1; i <= m; i++)
    {
        int x = edge[i].u, y = edge[i].v, w = edge[i].w;
        int tx = find(x), ty = find(y);
        if (tx != ty)
        {
            far[tx] = ty;
            add_edge(x, y, w);
            add_edge(y, x, w);
            mst += w;
            edge[i].vis = 1;
        }
    }
    dfs(1,0);
    int sum = INF;
    for (int i = 1; i <= m; i++)
    {
        if (edge[i].vis) continue;
        int x = edge[i].u, y = edge[i].v, w = edge[i].w;
        int z = lca(x, y);
        int d = max(get_max(x, z, w), get_max(y, z, w));
        sum = min(sum, mst - d + w);
    }
    cout << sum << '\n';
    return 0;
}
```

5.6 带花树

5.6.1 一般图最大匹配

- 给出一张 n 个点 m 条边的无向图，求该图的最大匹配
- ans 表示最大匹配数
- $link[i]$ 表示点 i 与点 $link[i]$ 匹配 ($link[i]=0$ 的点没有被匹配)
- 复杂度 $O(n(nlogn+m))$ ，不过和匈牙利一样跑不满

```
const int N = 1e3 + 10 , M = 5e4 + 10;
struct Edge{
    int nex, to;
} edge[M << 2];
int n, m, len, ans, T;
int head[M], far[N], link[N], pre[N], vis[N], ti[N];
//复杂度 O(n(nlogn+m)) ，不过和匈牙利一样跑不满。
// ans表示最大匹配边数
// link[i] 表示点 i 与点 link[i] 匹配
// link[i] = 0 的点没有被匹配
queue<int> q;
void add_edge(int x, int y)
{
    edge[++len].nex = head[x];
    edge[len].to = y;
    head[x] = len;
}
int find(int x)
{
    return far[x] == x ? x : far[x] = find(far[x]);
}
int lca(int x, int y)
{
    for(++ T ; ; swap(x, y))
    {
        if (!x) continue;
        x = find(x);
        if (ti[x] == T) return x;
        ti[x] = T;
        x = pre[link[x]];
    }
}
void flower(int x, int y, int p)
{
    while (find(x) != p)
    {
        pre[x] = y , y = link[x];
        if (vis[y] == 2) vis[y] = 1, q.push(y);
        if (find(x) == x) far[x] = p;
        if (find(y) == y) far[y] = p;
        x = pre[y];
    }
}
```

```cpp
bool bfs(int s)
{
    rep(i , 1 , n) far[i] = i;
    memset(vis, 0, sizeof(vis)) , memset(pre, 0, sizeof(pre));
    while (!q.empty()) q.pop();
    vis[s] = 1;
    q.push(s);
    while (!q.empty())
    {
        int u = q.front();
        q.pop();
        for (int i = head[u] ; i ; i = edge[i].nex)
        {
            int v = edge[i].to;
            if (find(v) == find(u) || vis[v] == 2) continue;
            if (!vis[v])
            {
                vis[v] = 2 , pre[v] = u;
                if (!link[v])
                {
                    for (int now = v, last, tmp; now; now = last)
                    {
                        last = link[tmp = pre[now]];
                        link[now] = tmp;
                        link[tmp] = now;
                    }
                    return 1;
                }
                vis[link[v]] = 1;
                q.push(link[v]);
            }
            else
            {
                int p = lca(u, v);
                flower(u, v, p);
                flower(v, u, p);
            }
        }
    }
    return 0;
}
signed main()
{
    ios::sync_with_stdio(false);
    cin >> n >> m;
    rep(i , 1 , m)
    {
        int x , y;
        cin >> x >> y;
        add_edge(x, y) , add_edge(y, x);
    }
    rep(i , 1 , n) if(!link[i]) ans += bfs(i);
    cout << ans << '\n';
```

```
    rep(i , 1 , n) cout << link[i] << " \n"[i == n];
    return 0;
}
```

5.6.2 一般图最大权匹配

- 给定一张有 n 个顶点的无向带权图，有 m 条带权边
- 求一种匹配的方案，使得最终匹配边的边权之和最大
- ma 表示最大匹配权
- $link[i]$ 表示点 i 与点 $link[i]$ 匹配 ($link[i] = 0$ 的点没有被匹配)
- 复杂度 $O(n^3)$

```
const int N = 1e3 + 10;
// N 要开 1.5 倍以上
// 答案记得开 long long
struct Edge
{
    int u, v, w;
    Edge() {}
    Edge(int a , int b , int c)
    {
        u = a , v = b , w = c;
    }
} g[N][N];
int n, m, n_x, lab[N], link[N], slack[N], st[N], pa[N], from[N][N], S[N], vis[N];
vector<int> flower[N];
deque<int> q;
int dist(Edge e)
{
    return lab[e.u] + lab[e.v] - e.w * 2;
}
void update_slack(int u, int x)
{
    if (!slack[x] || dist(g[u][x]) < dist(g[slack[x]][x]))
        slack[x] = u;
}
void set_slack(int n, int x)
{
    slack[x] = 0;
    rep(u , 1 , n) if(g[u][x].w > 0 && st[u] != x && S[st[u]] == 0) update_slack(u,
        x);
}
void q_push(int n, int x)
{
    if (x <= n) return q.push_back(x);
    for (int i = 0; i < flower[x].size(); i ++) q_push(n, flower[x][i]);
}
void set_st(int n, int x, int b)
{
    st[x] = b;
```

```
38      if(x <= n) return;
39      for (int i = 0; i < flower[x].size(); i ++) set_st(n, flower[x][i], b);
40  }
41  int get_pr(int b, int xr)
42  {
43      int pr = find(flower[b].begin(), flower[b].end(), xr) - flower[b].begin();
44      if (pr % 2 == 1) //检查它在前一层是奇点还是偶点
45      {
46          reverse(flower[b].begin() + 1, flower[b].end());
47          return (int)flower[b].size() - pr;
48      }
49      else return pr;
50  }
51  void set_match(int u, int v)
52  {
53      link[u] = g[u][v].v;
54      if (u <= n)
55          return;
56      Edge e = g[u][v];
57      int xr = from[u][e.u], pr = get_pr(u, xr);
58      for (int i = 0; i < pr; ++i)
59          set_match(flower[u][i], flower[u][i ^ 1]);
60      set_match(xr, v);
61      rotate(flower[u].begin(), flower[u].begin() + pr, flower[u].end());
62  }
63  void Aug(int n, int u, int v)
64  {
65      int xnv = st[link[u]];
66      set_match(u, v);
67      if (!xnv)
68          return;
69      set_match(xnv, st[pa[xnv]]);
70      Aug(n, st[pa[xnv]], xnv);
71  }
72  int get_lca(int u, int v)
73  {
74      static int t = 0;
75      for (++t; u || v; swap(u, v))
76      {
77          if (u == 0) continue;
78          if (vis[u] == t) return u;
79          vis[u] = t; //这种方法可以不用清空 v 阵列
80          u = st[link[u]];
81          if (u) u = st[pa[u]];
82      }
83      return 0;
84  }
85  void add_blossom(int n, int u, int LCA, int v)
86  {
87      int b = n + 1;
88      while (b <= n_x && st[b]) b++;
89      n_x = max(n_x, b);
90      lab[b] = S[b] = 0;
```

```
91      link[b] = link[LCA];
92      flower[b].clear();
93      flower[b].pb(LCA);
94      for (int x = u, y; x != LCA; x = st[pa[y]])
95      {
96          flower[b].pb(x);
97          y = st[link[x]];
98          flower[b].pb(y);
99          q_push(n, y);
100     }
101     reverse(flower[b].begin() + 1, flower[b].end());
102     for (int x = v, y; x != LCA; x = st[pa[y]])
103     {
104         flower[b].pb(x);
105         y = st[link[x]];
106         flower[b].pb(y);
107         q_push(n, y);
108     }
109     set_st(n, b, b);
110     rep(i , 1 , n_x) g[b][i].w = g[i][b].w = 0;
111     rep(i , 1 , n) from[b][i] = 0;
112     for (auto xs : flower[b])
113     {
114         rep(x , 1 , n_x) if (g[b][x].w == 0 || dist(g[xs][x]) < dist(g[b][x]))
115         {
116             g[b][x] = g[xs][x];
117             g[x][b] = g[x][xs];
118         }
119         rep(x , 1 , n)
120         if (from[xs][x])
121             from[b][x] = xs;
122     }
123     set_slack(n, b);
124 }
125 void expand_blossom(int n, int b)
126 {
127     for(auto i : flower[b]) set_st(n, i, i);
128     int xr = from[b][g[b][pa[b]].u];
129     int pr = get_pr(b, xr);
130     for (int i = 0; i < pr; i += 2)
131     {
132         int xs = flower[b][i];
133         int xns = flower[b][i + 1];
134         pa[xs] = xns;
135         S[xs] = 1;
136         S[xns] = 0;
137         slack[xs] = 0;
138         set_slack(n, xns);
139         q_push(n, xns);
140     }
141     S[xr] = 1;
142     pa[xr] = pa[b];
143     for (int i = pr + 1; i < flower[b].size() ; i ++)
```

```
144     {
145         int xs = flower[b][i];
146         S[xs] = -1;
147         set_slack(n, xs);
148     }
149     st[b] = 0;
150 }
151 bool on_found_Edge(int n, const Edge &e)
152 {
153     int u = st[e.u], v = st[e.v];
154     if (S[v] == -1)
155     {
156         pa[v] = e.u;
157         S[v] = 1;
158         int nu = st[link[v]];
159         slack[v] = slack[nu] = 0;
160         S[nu] = 0;
161         q_push(n, nu);
162     }
163     else if (S[v] == 0)
164     {
165         int LCA = get_lca(u, v);
166         if(!LCA)
167         {
168             Aug(n, u, v) , Aug(n, v, u);
169             return true;
170         }
171         else add_blossom(n, u, LCA, v);
172     }
173     return false;
174 }
175 bool matching(int n)
176 {
177     rep(i , 0 , n_x + 1) S[i] = -1 , slack[i] = 0;
178     q.clear();
179     rep(i , 1 , n_x) if(st[i] == i && !link[i])
180     {
181         S[i] = pa[i] = 0;
182         q_push(n, i);
183     }
184     if (q.empty()) return false;
185     for( ; ; )
186     {
187         while (!q.empty())
188         {
189             int u = q.front();
190             q.pop_front();
191             if (S[st[u]] == 1)
192                 continue;
193             for (int v = 1; v <= n; v++)
194                 if (g[u][v].w > 0 && st[u] != st[v])
195                 {
196                     if (dist(g[u][v]) == 0 && on_found_Edge(n, g[u][v]))
```

```
                    return true;
                else if (dist(g[u][v]) != 0)
                    update_slack(u, st[v]);
            }
        }
        int d = 0x3f3f3f3f;
        rep(b , n + 1 , n_x) if(st[b] == b && S[b] == 1) d = min(d, lab[b] / 2);
        rep(x , 1 , n_x) if(st[x] == x && slack[x])
        {
            if(S[x] == -1) d = min(d, dist(g[slack[x]][x]));
            else if(S[x] == 0) d = min(d, dist(g[slack[x]][x]) / 2);
        }
        rep(u , 1 , n)
        {
            if(S[st[u]] == 0)
            {
                if(lab[u] <= d) return false;
                lab[u] -= d;
            }
            else if(S[st[u]] == 1) lab[u] += d;
        }
        rep(b , n + 1 , n_x) if(st[b] == b)
        {
            if(S[st[b]] == 0) lab[b] += d * 2;
            else if(S[st[b]] == 1) lab[b] -= d * 2;
        }
        q.clear();
        rep(x , 1 , n_x)
        {
            if(st[x] == x && slack[x] && st[slack[x]] != x && !dist(g[slack[x]][x]))
                if(on_found_Edge(n, g[slack[x]][x]))
                    return true;
        }
        rep(b , n + 1 , n_x)
        {
            if(st[b] == b && S[b] == 1 && lab[b] == 0)
                expand_blossom(n, b);
        }
    }
    return 0;
}
pair<long long, int> solve(int n)
{
    long long sum = 0;
    int cnt = 0, ma = 0;
    rep(i , 1 , n) rep(j , 1 , n) ma = max(ma, g[i][j].w);
    rep(i , 1 , n) lab[i] = ma;
    while(matching(n)) cnt ++;
    rep(i , 1 , n)
    if (link[i] && link[i] < i)
        sum += (long long)g[i][link[i]].w;
    return make_pair(sum, cnt);
}
```

```cpp
void init(int n)
{
    n_x = n;
    rep(i, 0, n + 1)
    {
        rep(j, 0, n + 1)
        {
            g[i][j] = Edge {i, j, 0};
            from[i][j] = 0;
        }
        from[i][i] = i;
        link[i] = 0;
        st[i] = i;
        flower[i].clear();
    }
}
signed main()
{
    cin >> n >> m;
    init(n);
    rep(i, 1, m)
    {
        int u, v, w;
        cin >> u >> v >> w;
        w = max(w, g[u][v].w);
        g[u][v].w = g[v][u].w = w;
    }
    long long ma = solve(n).first;
    cout << ma << '\n';
    rep(i, 1, n) cout << link[i] << " \n"[i == n];
    return 0;
}
```

5.7 矩阵树定理

5.7.1 总结

注意：特判边界，例如大小是 0 或 1 的行列式。

1. 如果出现负数高精度，尝试打表找规律。
2. 与容斥结合。画文氏图，分析容斥系数，确定加减。
3. 给出无向图，求这个图的生成树个数。

   我们对这个图构造两个矩阵，分别是这个图的连通矩阵和度数矩阵。连通矩阵 S_1 的第 i 行第 j 列上的数字表示原无向图中编号为 i 和编号为 j 的两个点之间的边的条数。度数矩阵 S_2 只有斜对角线上有数字，即只有第 i 行第 i 列上有数字，表示编号为 i 的点的度数是多少。我们将两个矩阵相减，即 $S_2 - S_1$，我们记得到的矩阵为 T，我们将矩阵 T 去掉任意一行和一列（一般情况去掉最后一行和最后一列的写法比较多）得到 $T^{'}$，最后生成树的个数就是这个矩阵 $T^{'}$ 的行列式。高斯消元求解行列式。
4. 给出有向图和其中的一个点，求以这个点为根的生成外向树个数。外向树是入度全为 1，也就是每个节点指向儿子。

我们对这个图构造两个矩阵，分别是这个图的连通矩阵和度数矩阵。连通矩阵 S_1 的第 i 行第 j 列上的数字表示原无向图中编号为 i 和编号为 j 的两个点之间编号 i 的点指向编号为 j 的点的条数。度数矩阵 S_2 只有斜对角线上有数字，即只有第 i 行第 i 列上有数字，表示编号为 i 的点的入度是多少。我们将两个矩阵相减，即 $S_2 - S_1$，我们记得到的矩阵为 T，我们将矩阵 T 去掉根所在行和根所在列得到 $T^{'}$，最后生成树的个数就是这个矩阵 $T^{'}$ 的行列式。高斯消元求解行列式。

5. 给出有向图和其中一个点，求以这个点为根的生成内向树个数。内向树是出度全为 1，也就是每个节点指向父亲。

   我们对这个图构造两个矩阵，分别是这个图的连通矩阵和度数矩阵。连通矩阵 S_1 的第 i 行第 j 列上的数字表示原无向图中编号为 i 和编号为 j 的两个点之间编号 i 的点指向编号为 j 的点的条数。度数矩阵 S_2 只有斜对角线上有数字，即只有第 i 行第 i 列上有数字，表示编号为 i 的点的出度是多少。我们将两个矩阵相减，即 $S_2 - S_1$，我们记得到的矩阵为 T，我们将矩阵 T 去掉根所在行和根所在列得到 $T^{'}$，最后生成树的个数就是这个矩阵 $T^{'}$ 的行列式。高斯消元求解行列式。

5.7.2 无向图生成树个数

- 复杂度大概 $O(n^3)$?

```
using namespace std;
const int N = 2e2 + 10;
const ll mod = 1e9 + 7;
ll G[N][N];
int n , m;
ll pow_mod(ll x , ll n , ll mod)
{
    ll res = 1;
    while(n)
    {
        if(n & 1) res = res * x % mod;
        x = x * x % mod;
        n >>= 1;
    }
    return res;
}
void init(int n)
{
    rep(i , 0 , n) rep(j , 0 , n) G[i][j] = 0;
}
ll Gauss(int n)
{
    ll ans = 1;
    rep(i , 1 , n - 1)
    {
        rep(k , i + 1 , n - 1)
        {
            while(G[k][i])
            {
                ll d = G[i][i] / G[k][i];
                rep(j , i , n - 1) G[i][j] = (G[i][j] - d * G[k][j] % mod + mod) % mod;
                swap(G[i] , G[k]) , ans = -ans;
```

```
33
34              }
35          }
36          ans = (ans * G[i][i] % mod + mod) % mod;
37      }
38      return ans;
39  }
40  signed main()
41  {
42      ios::sync_with_stdio(false) , cin.tie(0) , cout.tie(0);
43      int T = 1;
44      cin >> T;
45      while(T --)
46      {
47          cin >> n >> m;
48          init(n);
49          rep(i , 1 , m)
50          {
51              int x , y ;
52              cin >> x >> y ;
53              G[x][x] ++ , G[y][y] ++;
54              G[x][y] -- , G[y][x] --;
55          }
56          ll res = Gauss(n);
57          cout << res << '\n';
58      }
59      return 0;
60  }
```

5.7.3 有 (无) 向图生成树的权值和

```
1   const int N = 3e2 + 10 , mod = 1e9 + 7;
2   ll a[N][N];
3   ll pow_mod(ll x , ll n , ll mod)
4   {
5       ll res = 1;
6       while(n)
7       {
8           if(n & 1) res = res * x % mod;
9           x = x * x % mod;
10          n >>= 1;
11      }
12      return res;
13  }
14  ll Gauss(int n)
15  {
16      ll ans = 1;
17      for(int i = 2 ; i <= n ; i ++)
18      {
19          for(int j = i + 1 ; j <= n ; j ++) if(!a[i][i] && a[j][i])
20          {
21              ans = -ans;
```

```
22          swap(a[i] , a[j]);
23          break;
24      }
25      ll inv = pow_mod(a[i][i] , mod - 2 , mod);
26      for(int j = i + 1 ; j <= n ; j ++)
27      {
28          ll temp = a[j][i] * inv % mod;
29          for(int k = i ; k <= n ; k ++) a[j][k] = (a[j][k] - a[i][k] * temp % mod +
                mod) % mod;
30      }
31      ans = ans * a[i][i] % mod;
32   }
33   return ans;
34 }
35 signed main()
36 {
37   int n , m , op;
38   //op = 0 ÎÞÏòÍ¼ , op = 1ÓÐÏòÍ¼
39   cin >> n >> m >> op;
40   for(int i = 1 ; i <= m ; i ++)
41   {
42      int u , v , w;
43      cin >> u >> v >> w;
44      if(!op)
45      {
46          a[u][u] = (a[u][u] + w) % mod;
47          a[v][v] = (a[v][v] + w) % mod;
48          a[u][v] = (a[u][v] - w + mod) % mod;
49          a[v][u] = (a[v][u] - w + mod) % mod;
50      }
51      else
52      {
53          a[v][v] = (a[v][v] + w) % mod;
54          a[u][v] = (a[u][v] - w + mod) % mod;
55      }
56   }
57   cout << Gauss(n) << '\n';
58   return 0;
59 }
```

5.8 判断完全图

5.8.1 判断完全图

当 N 很大需要用 *vector* 存图时

```
1 for(auto x : vec) for(auto y : vec) if(x != y && !binary_search(all(g[x]) , y)) flag
      = 0;
```

5.9 树的直径

5.9.1 两次 dfs

1. 以任意一点 P 作为根节点，通过 DFS 寻找离它最远的点 Q
2. 再次从点 Q 作为根节点，通过 DFS 寻找离它最远的 W
3. 直径即为 $dep[W]$

```
int dep[N] , pos , ma;
void dfs(int u , int far)
{
    dep[u] = dep[far] + 1;
    if(dep[u] > ma) ma = dep[u] , pos = u;
    for(int i = head[u] ; i ; i = edge[i].nex)
    {
        int v = edge[i].to;
        if(v == far) continue ;
        dfs(v , u);
    }
}
int calc()
{
    ma = 0;
    dfs(1 , 0);
    //memset(dep , 0 , sizeof(dep));
    dfs(pos , 0);
    return ma - 1;
}
```

5.10 最短路

5.10.1 Dijkstra 堆优化

复杂度 $O((N+M)logM)$

```
const int inf = 0x3f3f3f3f;
const int N = 2e5 + 10;
struct node{
    int dis , pos;
    bool operator <( const node &x )const{
        return x.dis < dis;
    }
};
struct Edge{
    int to , dis , nex;
}edge[N << 1];
int head[N], dis[N] , tot , vis[N] , n , m , s;
inline void add_edge(int u , int v , int d)
{
    edge[++ tot].dis = d;
    edge[tot].to = v;
    edge[tot].nex = head[u];
    head[u] = tot;
```

```
}
inline void dijkstra(int s)
{
    memset(dis , inf , sizeof(dis)) ;
    dis[s] = 0;
    priority_queue <node> que;
    que.push(node{0, s});
    while(!que.empty())
    {
        node tmp = que.top();
        que.pop();
        int u = tmp.pos, d = tmp.dis;
        if(vis[u]) continue;
        vis[u] = 1;
        for(int i = head[u] ; i ; i = edge[i].nex)
        {
            int v = edge[i].to;
            if(dis[v] > dis[u] + edge[i].dis)
            {
                dis[v] = dis[u] + edge[i].dis;
                if(!vis[v]) que.push(node{dis[v] , v});
            }
        }
    }
}
signed main()
{
    ios::sync_with_stdio(false);
    cin >> n >> m >> s;
    rep(i , 1 , m)
    {
        int u , v , w;
        cin >> u >> v >> w;
        add_edge(u , v , w);
    }
    dijkstra(s);
    rep(i , 1 , n) cout << dis[i] << " ";
    cout << '\n';
    return 0;
}
```

5.10.2 Dijkstra 配对堆

复杂度 $O(M + NlogN)$

```
#include <bits/stdc++.h>
#include <ext/pb_ds/priority_queue.hpp>
#define rep(i, a, n) for (int i = a; i <= n; i++)
#define per(i, n, a) for (int i = n; i >= a; i--)
#define il inline
using namespace std;
using namespace __gnu_pbds;
const int inf(0x3f3f3f3f);
```

```cpp
typedef __gnu_pbds::priority_queue<pair<int, int>, greater<pair<int, int>>,
    pairing_heap_tag> heap;
template <typename T>
void read(T &res)
{
    bool flag = false;
    char ch;
    while (!isdigit(ch = getchar()))
    (ch == '-') && (flag = true);
    for (res = ch - 48; isdigit(ch = getchar()); res = (res << 1) + (res << 3) + ch
        - 48)
    ;
    flag && (res = -res);
}
template <typename T>
void Out(T x)
{
    if (x < 0)
    putchar('-'), x = -x;
    if (x > 9)
    Out(x / 10);
    putchar(x % 10 + '0');
}
const int N = 2e5 + 10;
heap que;
heap::point_iterator id[N];
struct Edge
{
    int nex, to, w;
} edge[N << 1];
int head[N], tot;
il void add_edge(int u, int v, int w)
{
    edge[++tot].nex = head[u];
    edge[tot].to = v;
    edge[tot].w = w;
    head[u] = tot;
}
int n, m, s, dis[N];
il void dij(int st)
{
    memset(dis, inf, sizeof(dis));
    dis[st] = 0;
    id[st] = que.push(make_pair(0, st));
    while (!que.empty())
    {
        int u = que.top().second;
        que.pop();
        for (int i = head[u]; i; i = edge[i].nex)
        {
            int v = edge[i].to;
            int w = edge[i].w;
            if (w + dis[u] < dis[v])
```

```
60          {
61              dis[v] = dis[u] + edge[i].w;
62              if (id[edge[i].to] != 0)
63              que.modify(id[v], make_pair(dis[v], v));
64              else
65              id[v] = que.push(make_pair(dis[v], v));
66          }
67       }
68    }
69 }
70 signed main()
71 {
72    read(n), read(m), read(s);
73    rep(i, 1, m)
74    {
75       int u, v, w;
76       read(u), read(v), read(w);
77       add_edge(u, v, w);
78    }
79    dij(s);
80    rep(i, 1, n) Out(dis[i]), putchar(' ');
81    return 0;
82 }
```

5.11 最小 mex 生成树

5.11.1 最小 mex 生成树

- n 个点 m 条边的无向连通图，边有边权。
- 求一个的生成树，使得其边权集合的 mex 尽可能小
- $1 \le n \le 10^6, 1 \le m \le 2 \times 10^6, 0 \le w \le 10^5$（注意边权是从 0 开始，要处理一下）

可撤销并查集 + 线段树分治
若一条边边权为 w，则给 $[1, w-1]$、$[w+1, max]$ 添加上这条边。
若在叶子节点 $size[find(1)] = n$ 则表示该图连通，直接输出 **叶子节点编号** -1（边权是从 0 开始的）即可

```
1  typedef pair<int , int> pii;
2  const int N = 1e5 + 10 , M = 1e6 + 10;
3  int n , m;
4  int far[M] , size[M];
5  void init()
6  {
7     rep(i , 1 , M - 10) far[i] = i , size[i] = 1;
8  }
9  int find(int x)
10 {
11    if(x == far[x]) return x;
12    return find(far[x]);
13 }
14 void Union(int x , int y , stack<pii>&st)
15 {
```

```
    int tx = find(x) , ty = find(y);
    if(tx != ty)
    {
        if(size[tx] > size[ty]) swap(tx , ty);
        st.push(make_pair(tx , ty));
        far[tx] = ty;
        size[ty] += size[tx];
    }
}
void Redo(stack<pii>&st)
{
    while(!st.empty())
    {
        pair<int , int>u = st.top();
        st.pop();
        far[u.fi] = u.fi;
        size[u.se] -= size[u.fi];
    }
}
struct Edge{
    int u , v , w;
    bool operator < (const Edge & b) const {
        return w < b.w;
    }
}edge[M << 1];
struct Tree{
    int l , r;
    vector<Edge>vec;
}tree[N << 2];
void build(int l , int r , int rt)
{
    tree[rt].l = l , tree[rt].r = r;
    if(l == r) return ;
    int mid = l + r >> 1;
    build(l , mid , rt << 1);
    build(mid + 1 , r , rt << 1 | 1);
}
void update_range(int L , int R , int rt , Edge e)
{
    int l = tree[rt].l , r = tree[rt].r;
    if(L <= l && r <= R)
    {
        tree[rt].vec.pb(e);
        return ;
    }
    int mid = l + r >> 1;
    if(L <= mid) update_range(L , R , rt << 1 , e);
    if(R > mid) update_range(L , R , rt << 1 | 1 , e);
}
void query(int rt)
{
    int l = tree[rt].l , r = tree[rt].r;
    stack<pii>st;
```

```
    for(auto i : tree[rt].vec) Union(i.u , i.v , st);
    if(l == r)
    {
        if(size[find(1)] == n)
        {
            cout << l - 1 << '\n';
            exit(0);
        }
    }
    else query(rt << 1) , query(rt << 1 | 1);
    Redo(st);
}
signed main()
{
    init();
    cin >> n >> m;
    int up = 1e5 + 1;
    build(1 , up , 1);
    rep(i , 1 , m)
    {
        int u , v , w;
        cin >> u >> v >> w;
        w ++ ;
        edge[i] = Edge{u , v , w + 1};
        if(w != 1) update_range(1 , w - 1 , 1 , edge[i]);
        update_range(w + 1 , up , 1 , edge[i]);
    }
    query(1);
    return 0;
}
```

6 动态规划

6.1 总结

关于决策：

如：$dp[i][j]$ 表示前 i 个数使用了 j 个盒子，所能得到的最大价值

- 决策 1：
  - 第 i 个数选或不选
  - 选：$dp[i][j] = dp[k][j-1] + A[i]$（$k < i$，$A[i]$ 为所能获得的价值）
  - 不选：$dp[i][j] = dp[i-1][j]$
- 决策 2：
  - 用 $g[i][j]$ 表示前 i 个数使用了 j 个盒子，所能得到的最大价值（第 i 个数一定要选）
  - $dp[i][j]$ 表示前 i 个数使用了 j 个盒子，所能得到的最大价值（第 i 个数可选可不选）
  - $g[i][j] = Max(g[i-1][j], dp[i-1][j-1]) + A[i]$
    （i 可以和 $i-1$ 放在一个盒子里，也可以单独放在一个新盒子里）

- $dp[i][j] = Max(g[i][j], dp[i-1][j])$
  （i 选了最优解就是 $g[i][j]$ ， i 不选最优解就是 $dp[i-1][j]$）

关于定义：

1. $dp[i][j]$ 表示前 i 个数和为 j 的方案数
2. $dp[i][j]$ 表示前 i 个数和不大于（不小于）j 的方案数
3. $dp[i][j]$ 表示前 i 个数，数值小于 x 的数的个数为 j 的集合个数

6.2 01 背包

6.2.1 前 k 优解

- $dp[V][K]$ 表示前 N 个物品，背包容量为 V 的第 K 优解
- 复杂度 $O(NVK)$

```
int w[N] , v[N];

int dp[N][K] , ma1[N] , ma2[N];

signed main()

{

   rep(i , 0 , N - 10) rep(j , 0 , N - 10) dp[i][j] = -inf;

   int n , V , K;

   cin >> K >> V >> n;

   dp[0][1] = 0;

   rep(i , 1 , n) cin >> w[i] >> v[i];

   rep(i , 1 , n)

   {

      per(j , V , w[i])

      {

         rep(k , 1 , K)

         {

            ma1[k] = dp[j][k];

            ma2[k] = dp[j - w[i]][k] + v[i];

         }

```

```
37          int cnt = 0 , c1 = 1 , c2 = 1;
38
39          while(cnt < K)
40
41          {
42
43              if(ma1[c1] >= ma2[c2]) dp[j][++ cnt] = ma1[c1 ++];
44
45              else dp[j][++ cnt] = ma2[c2 ++];
46
47          }
48
49       }
50
51    }
52
53    int ans = 0;
54
55    rep(k , 1 , K) ans += dp[V][k];
56
57    if(ans < 0) ans = 0;
58
59    cout << ans << '\n';
60
61    return 0;
62
63 }
```

6.2.2 求最优解方案个数

- 本题求 01 背包的最佳方案数，那么定义两个数组：$f[N], cnt[N]$
- $f[i]$ 用来存储背包容积为 i 时的最佳方案的总价值，
- $cnt[i]$ 为背包容积为 i 时总价值为最佳的方案数
- 先初始化所有的 $cnt[i]$ 为 1，因为背包里什么也不装也是一种方案
- 外层循环 n 次，每次读入新物品的 w, v
- 求出装新物品时的总价值，与不装新物品时作对比
- 如果装新物品的方案总价值更大，那么用 $f[j-v]+wf[j-v]+w$ 来更新 $f[j]$，用 $cnt[j-v]$ 更新 $cnt[j]$
- 如果总价值相等，那么最大价值的方案数就多了 $cnt[j-v]$ 种

```
1 cin >> n >> V;
2
3 rep(i , 0 , V) cnt[i] = 1;
4
5 rep(i , 1 , n) cin >> w[i] >> v[i];
6
7 rep(i , 1 , n) per(j , V , w[i])
8
```

```
{

    if(dp[j - w[i]] + v[i] > dp[j]) dp[j] = dp[j - w[i]] + v[i] , cnt[j] = cnt[j - w
        [i]];

    else if(dp[j - w[i]] + v[i] == dp[j]) cnt[j] += cnt[j - w[i]] , cnt[j] %= MOD;

}

cout << cnt[V] << '\n';
```

6.2.3 求恰好装满背包的最优解

- 对于不需要装满的情况，初始化 dp 数组的值为 0
- 而对于需要装满的情况，只要初始化 dp 数组的值为 $-inf$ (求最大值)
- 这样就能使能够恰好装满背包的物品的价值之和为正数
- 而那些不能恰好装满背包的物品的价值之和就为 $-inf$

6.2.4 求具体方案

- 01 背包的最优解为 $dp[N][V]$
- 其含义为前 N 个物品，体积为 V 的最优解
- 当 $dp[N][V] = dp[N-1][V]$ 时，第 N 个物品未选
- 当 $dp[N][V] = dp[N-1][V-w[i]] + v[i]$ 时，第 N 个物品选了

  当要求字典序最小的方案数时
  若存在一个包含第 1 个物品的最优解
  为了确保字典序最小那么我们必然要选第一个
  那么之后问题就转化成从 $2 \sim N$ 这些物品中找到最优解
  而之前的 $dp(i,j)$ 记录的都是前 i 个物品总容量为 j 的最优解
  所以现在需要将 $dp(i,j)$ 定义为从第 i 个元素到最后一个元素总容量为 j 的最优解
  然后模拟一遍即可

```
cin >> n >> V;
rep(i , 1 , n) cin >> w[i] >> v[i];
per(i , n , 1)
{
    rep(j , 0 , V)
    {
        dp[i][j] = dp[i + 1][j];
        if(j - w[i] >= 0) dp[i][j] = max(dp[i + 1][j] , dp[i + 1][j - w[i]] + v[i]);
    }
}
int m = V;
rep(i , 1 , n)
{
    if(m - w[i] >= 0 && dp[i][m] == dp[i + 1][m - w[i]] + v[i])
    {
```

```
16          ans.push_back(i) , m -= w[i];
17      }
18  }
19  for(auto i : ans) cout << i << " ";
```

6.2.5 超大背包

- 当背包的最大容量很大的时候，常规的做法时间空间复杂度都会炸
- 但这类问题通常物品的价值或个数不会很大
- 定义 n 为物品的个数，V 为背包的容量，W 为物品体积的上限，val 为物品价值的上限
- 1. 当 n 较小 (300)、V 很大 $(1e9)$、W 很大 $(1e9)$、val 较小 (300) 的时候：
     可以定义 $f[i][j]$ 表示前 i 个物品选取的价值为 j 时的最小体积
     那么只要取最大的 j 使得 $f[n][j] <= V$ 即可
     复杂度为 $O(n^2 \times val)$
  2. 当 n 很小 (40)、V 很大 $(1e9)$、W 很大 $(1e9)$、val 很大 $(1e9)$ 的时候：
     可以考虑折半枚举 + 二分查找
     把物品分为前 $\frac{n}{2}$ 和后 $\frac{n}{2}$，然后把前 $\frac{n}{2}$ 的状态保存下来并排序，记为 vec
     再枚举后 $\frac{n}{2}$，计算当前状态的体积 v，然后再去 vec 中二分查找 $V-v$ 的最优解
     （vec 存储的是 $pair < weight, val >$，$vec[i].val = max(vec[0 \sim i].val)$）
     复杂度为 $O(2^{\frac{n}{2}} \times \log_2\left(\frac{n}{2}\right))$
  3. 当 n 较大 $(2e5)$、V 较大 $(2e5)$、W 较小 (100)、val 很大 $(1e9)$ 的时候：
     考虑分治 or 单调队列优化决策单调性
     复杂度为 $O(W \times V \times \log V)$
  4. 当 n 较大 $(2e5)$、V 较大 $(2e5)$、W 较小 (100)、val 很大 $(1e9)$ 的时候：
     考虑一个骗分做法：
     先把 n 个物品按照性价比排序，然后把性价比高的存入背包，使得背包容量 V 不断减少，直到 $O(n \times V)$ 的做法可以解决（骗分，慎用！）
- 一：

```
1   signed main()
2   {
3       int T = 1;
4       cin >> T;
5       while(T --)
6       {
7           memset(f , 0x3f , sizeof(f));
8           f[0] = 0;
9           int n , V , up = 0 , res = 0;
10          cin >> n >> V;
11          for(int i = 1 ; i <= n ; i ++) cin >> w[i] >> v[i] , up += v[i];
12          for(int i = 1 ; i <= n ; i ++)
13          {
14              for(int j = up ; j >= v[i] ; j --)
15              {
16                  f[j] = min(f[j] , f[j - v[i]] + w[i]);
17              }
18          }
```

```
        for(int i = 0 ; i <= up ; i ++) if(f[i] <= V) res = i;
        cout << res << '\n';
    }
    return 0;
}
```

- 二:

```
#define int long long
const ll INF (0x3f3f3f3f3f3f3f3fll);
const int N = 41;
int w[N] , v[N];
vector<pair<int , int>>vec;
signed main()
{
    int n , V;
    cin >> n >> V;
    rep(i , 1 , n) cin >> w[i] , v[i] = w[i];
    int m = n / 2 , res = 0;
    vec.push_back(make_pair(INF , INF));
    rep(i , 0 , (1 << m) - 1)
    {
        int weight = 0 , val = 0;
        rep(j , 0 , m - 1) if(i >> j & 1) weight += w[m - j] , val += v[m - j];
        vec.push_back(make_pair(weight , val));
    }
    sort(vec.begin() , vec.end());
    int ma = -1;
    for(int i = 0 ; i < vec.size() ; i ++)
    {
        ma = max(ma , vec[i].second);
        vec[i].second = ma;
    }
    m = n - m;
    rep(i , 0 , (1 << m) - 1)
    {
        int weight = 0 , val = 0;
        rep(j , 0 , m - 1) if(i >> j & 1) weight += w[n - j] , val += v[n - j];
        if(weight > V) continue ;
        pair<int , int>p = make_pair(V - weight , INF);
        int pos = upper_bound(vec.begin() , vec.end() , p) - vec.begin();
        res = max(res , vec[pos - 1].second + val);
    }
    cout << res << '\n';
    return 0;
}
```

- 三:

```
constexpr ll INF = 1e18;
void work(vector<ll> &a, vector<ll> &b, vector<ll> &c, int l, int r, int L, int
    R)
{
```

```
    if (l > r) return;
    int m = (l + r) / 2;
    int g = -1;
    ll val = -INF;
    for (int i = L; i <= R && i <= m; ++i)
    {
        ll v = a[i] + b[m - i];
        if (v > val)
        {
            val = v;
            g = i;
        }
    }
    c[m] = val;
    work(a, b, c, l, m - 1, L, g);
    work(a, b, c, m + 1, r, g, R);
}

void solve(int n, int m)
{
    vector<int> w(n), v(n);
    int ma = 0;
    for (int i = 0; i < n; ++i)
    {
        cin >> w[i] >> v[i];
        if (w[i] <= m) ma = max(ma, w[i]);
    }
    vector<vector<int>> vals(ma + 1);
    for (int i = 0; i < n; ++i)
    {
        if (w[i] <= m) vals[w[i]].push_back(v[i]);
    }
    vector<ll> ans(m + 1);
    for (int W = 1; W <= ma; ++W)
    {
        sort(vals[W].begin(), vals[W].end(), greater<int>());
        for (int b = 0; b < W; ++b)
        {
            int siz = (m - b) / W;
            vector<ll> a(siz + 1), c(siz + 1), d(siz + 1);
            for (int i = 0; i <= siz; ++i)
            {
                a[i] = ans[i * W + b];
            }
            for (int i = 0; i < siz; ++i)
            {
                c[i + 1] = c[i] + (i < int(vals[W].size()) ? vals[W][i] : 0);
            }
            work(a, c, d, 0, siz, 0, siz);
            for (int i = 0; i <= siz; ++i)
            {
                ans[i * W + b] = d[i];
            }
```

```
        }
    }
    cout << ans[m] << "\n";
}
signed main()
{
    int n, m;
    while (cin >> n >> m) solve(n, m);
    return 0;
}
```

- 四：

```
const int N = 2e5 + 10 , M = 1e3 + 10;
struct node{
    int w , v;
}a[N];
long long f[M];
bool cmp(node a , node b)
{
    return 1ll * a.w * b.v < 1ll * b.w * a.v;
}
signed main()
{
    int n , m ;
    while(cin >> n >> m)
    {
        for(int i = 1 ; i <= n ; i ++) cin >> a[i].w >> a[i].v;
        sort(a + 1 , a + 1 + n , cmp);
        long long ans = 0;
        int pos = 1;
        while(pos <= n && m >= 1000)
        {
            ans += a[pos].v;
            m -= a[pos].w;
            pos ++ ;
        }
        memset(f , 0 , (m + 1) << 3);
        for(int i = pos ; i <= n ; i ++)
        {
            for(int j = m ; j >= a[i].w ; j --)
                f[j] = max(f[j] , f[j - a[i].w] + a[i].v);
        }
        cout << ans + f[m] << '\n';
    }
    return 0;
}
```

6.3 完全背包

6.3.1 求最优解方案数

- em，好像只能 n^3 做？（之后要是会了再回来补吧）

- $f_{i,j}+=f_{i-1,j-k\times w_i}\quad [dp_{i,j}=dp_{i-1,j-k\times w_i}+k\times v_i]$

6.4 多重背包

6.4.1 总结

- 转换为 01 背包：（暴力拆解）
  比如物品 A 有 k 个，我们不妨将它拆解为 k 个新物品，这 k 个新物品的体积为价值都和物品 A 相同，那么这就转换为了 01 背包问题

- 转换为 01 背包：（二进制优化拆解）
  比如物品 A 有 k 个，物品 A 的体积为 w，价值为 v
  我们不妨将它拆解为：
  1 个体积为 w ，价值为 v 的物品
  1 个体积为 $2w$，价值为 $2v$ 的物品
  1 个体积为 $4w$，价值为 $4v$ 的物品
  1 个体积为 $8w$，价值为 $8v$ 的物品
  .....
  1 个体积为 2^xw，价值为 2^xw 的物品
  1 个体积为 $(k-(1+2+4+...+2^x))\times w$ ，价值为 $(k-(1+2+4+...2^x))\times v$ 的物品
  当 $2^0+2^1+...+2^{x+1}>k$ 时停止拆解，拆解的最后一个物品的系数等于 $k-2^0+...+2^x$

- 单调队列优化
  qwq 见代码

- 二、

```
int dp[M];
int n , v[M << 1] , w[M << 1];
signed main()
{
   ios::sync_with_stdio(false);
   cin.tie(0) , cout.tie(0);
   int n , V , cnt = 0;
   cin >> n >> V;
   for(int i = 1 ; i <= n ; i ++)
   {
      int ww , vv , S;
      cin >> ww >> vv >> S;
      int b = 0;
      while(S >= (1 << b))
      {
         int x = (1 << b);
         w[++ cnt] = ww * x , v[cnt] = vv * x;
         S -= (1 << b);
         b ++ ;
      }
      if(S) w[++ cnt] = S * ww , v[cnt] = S * vv;
   }
   for(int i = 1 ; i <= cnt ; i ++)
   {
      for(int j = V ; j >= w[i] ; j --) dp[j] = max(dp[j] , dp[j - w[i]] + v[i
          ]);
   }
```

```
    cout << dp[V] << '\n';
    return 0;
}
```

- 三、

```
#include <bits/stdc++.h>
using namespace std;
int n, m;
int f[20002], q[20002], g[20002];
int main()
{
   cin >> n >> m;
   for
   (int i = 1; i <= n; ++i)
   {
      int v, w, s;
      cin >> v >> w >> s;
      memcpy(g, f, sizeof(f));
      for (int j = 0; j < v; ++j)
      {
         /*
         hh为队首位置
         tt为队尾位置
         数值越大，表示位置越后面
         队首在队尾后面队列为空，即hh>tt时队列为空
         */
         int hh = 0, tt = -1;
         /*
         q[]为单调队列
         存储前个s(物品数量)中的最大值
         数组从头(hh)到尾(tt)单调递减
         */
         for (int k = j; k <= m; k += v)
         {

            // f[k] = g[k]; //这一行我没理解有什么用，去掉后也能ac？我认为前面使用过了
                memcpy,这里应该一定是相等的
            //k代表假设当前背包容量为k
            //q[hh]为队首元素（最大值的下标）
            //g[]为dp[i-1][]
            //f[]为dp[i][]
            /*
            维护一个大小为k的区间
            使最大值为前k个元素中最大
            (k - q[hh]) / v 表示拿取物品的数量（相当于最原始的多重背包dp的k）
            */
            if (hh <= tt && (k - q[hh]) / v > s)
            {
               hh ++;
            }
            /*
            若队内有值，该值就是前k个元素的最大值
```

```
47              (k - q[hh]) / v 表示拿取物品的数量（相当于最原始的多重背包dp的k）
48              q[hh]为队首元素（g[]中前k个数中最大值的下标），g[]为dp[i-1][]
49              所以 g[q[hh]]为只考虑前i-1个物品时，拿前q[hh]个物品的最大价值
50              */
51              if (hh <= tt)
52              {
53                  f[k] = max(f[k], g[q[hh]] + (k - q[hh]) / v * w);
54              }
55              /*
56              若队尾元素小于当前元素，则队尾元素出队；
57              若队内一个元素比当前元素小，则该元素一定不会被用到（单调队列思想）
58              g[q[tt]] + (k - q[tt]) / v * w
59              与
60              g[k] - (k - j) / v * w
61              分别表示队尾元素的值和当前元素的值
62              */
63              while (hh <= tt && g[q[tt]] - (q[tt] - j) / v * w <= g[k] - (k - j) / v
                    * w)
64              {
65                  tt--;
66              }
67              q[++ tt] = k;
68          }
69      }
70  }
71  cout << f[m] << '\n';
72  return 0;
73 }
```

6.5 分组背包

6.5.1 总结

定义：若干组物品，每组中的物品互相冲突（就是说每组中最多只能选一件物品），求最大价值
简单来说就是把每个节点看成一个背包啦，它的容量就是以这个节点为根的子树大小，组数就是连接的儿子个数。

6.6 树形依赖背包

6.6.1 总结

- **问题模型**：
  有 N 个产品，每个产品都有对应的体积和价值。对于其中某些产品，需要购买了它的父产品（最多只会有一个）才能购买它，问可以获得的最大价值?
- **模型分析：**
  给有关系的产品连边，最后必然会生成一棵树或者一片森林
  有个小技巧：设置一个虚点 0 成为所有树的根节点，那么森林就会化为一棵树了
  而对于树来说有个很重要的性质：
  当回溯结束后，子树已经被遍历完并处理完了。所有的操作都是从叶子节点往上爬的！所以给 dp 数组的赋初值的时候，可以考虑从叶子节点下手！

  1. 定义 $dp_{u,i,j}$ 表示以 u 为根节点，从前 i 个子节点中，选择了 j 个子节点的最大价值。

2. 像 01 背包一样压缩空间，得到：$dp_{u,j}$ 表示 u 为根节点，选择了 j 个子节点的最大价值。
3. 转移式为：$dp_{u,j} = dp_{v,k} + dp_{u,j-k} + (u \to v$ 的边权 $)$
4. 转移式中的 $k < j$（因为得保证 u 一定被选择了，不过有些题目 $k = j$ 也可，比如当 j 表示的是**叶子节点个数**的时候）
5. 若节点 u 有点权，则只需在初始化的时候令 $dp_{u,1} = val_u$ 即可
6. 若要设置虚点 0（将森林化为树），则最后答案为 $dp_{0,m+1}$（多了一个节点 0）

- **树上背包也算是分组背包：**
  分组背包：若干组物品，每组中的物品互相冲突（就是说每组中最多只能选一件物品），求最大价值简单来说就是把每个节点看成一个背包啦，它的容量就是以这个节点为根的子树大小，组数就是连接的儿子个数。

  树上背包：每组都有很多选择，选一个，两个，三个，把这些选择当做组中的元素就好了，容易看出每组中只能选一个元素，比如你选择了选一个物品，就不可能同时选择选两个物品。

```
const int N = 3e2 + 10;
struct Edge{
   int nex , to;
}edge[N << 1];
int head[N] , TOT;
void add_edge(int u , int v)
{
   edge[++ TOT].nex = head[u] ;
   edge[TOT].to = v;
   head[u] = TOT;
}
int n , m , dp[N][N] , s[N] , v[N] , sum = 0;
int dfs(int u , int far)
{
   int sum = 1;
   for(int i = head[u] ; i ; i = edge[i].nex)
   {
      int v = edge[i].to;
      if(v == far) continue ;
      sum += dfs(v , u);
      for(int j = sum ; j >= 1 ; j --)
      {
         for(int k = 0 ; k <= j - 1 ; k ++)
         dp[u][j] = max(dp[u][j] , dp[u][j - k] + dp[v][k]);
      }
   }
   return sum;
}
signed main()
{
   cin >> n >> m;
   for(int i = 1 ; i <= n ; i ++)
   {
      cin >> s[i] >> v[i];
      dp[i][1] = v[i];
```

```
36          add_edge(i , s[i]);
37          add_edge(s[i] , i);
38      }
39      dfs(0 , 0);
40      cout << dp[0][m + 1] << '\n'; // 多了一个节点 0
41      return 0;
42  }
```

6.7 状压 dp

6.7.1 总结

- 对于序列问题：
  $dp[i]$ 可以表示为：
  使用了 i 这个状态对应的数，组成了长度为 $cnt[i]$ (i 对应的数的个数) 的序列的方案数。
  那么 $dp[i]$ 就可由 $dp[i-(1<<k)]$ 转移得到（其中 (i » k & 1) = 1。
- 枚举子集：
  `for(int j = i ; j ; j = (j - 1) & i)`

6.8 区间 dp

6.9 数位 dp

6.10 经典问题

6.10.1 最长上升子序列

- 不管是求最长上升子序列还是最长不下降子序列最好都用 $upper\_bound$，用 $lower\_bound$ 求不下降子序列会出错。
- 求 lis：
  1. N^2 的 dp
  2. 贪心 + 二分
  3. 权值树状数组 or 权值线段树
- 求 lis 方案数：
  定义 len 表示最大 lis 长度，$f[i]$ 表示以 i 为结尾的 lis 长度，$cnt[i]$ 表示以 i 为结尾的 lis 个数，先预处理 $f[i]$ 和 len
  那么当 $a[i]>a[j]$ && $f[i]=f[j]+1$ 时，$cnt[i]+=cnt[j]$
  答案为 $\sum_{i=1}^{n} cnt[i][f[i]==len]$
  如果要求方案数不重复的话，只要当 $a[i]=a[j]\&\&f[i]=f[j]$ 时，令 $cnt[i]=0$ 或 $cnt[j]=0$ 即可
- 求具体方案：
  和求方案数类似，先求出 $f[i]$ 和 len
  1. 定义 $pre[i]$ 表示 i 的前驱，那么当 $f[i]=f[j]+1$ 时，$pre[i]=j$。
  2. 从 $f[now]=len$ 的位置开始，如果 $f[i]=f[now]-1\&\&a[i]<a[now]$ 则 $now=i$，不断重复此步骤直到 $f[now]=1$。
- 二维 lis：
  先对第一维升序排序，再在排好序的序列上根据第二维求 lis 即可

6.10.2 最长公共子序列

- 最长公共子序列：

$$f[i,j]=\begin{cases}\begin{cases}cc0 & i=0 \text{ or } j=0\\ f[i-1,j-1]+1 & i,j>0 \text{ and } a_i=b_j\\ \max(f[i,j-1],f[i-1,j]) & i,j>0 \text{ and } a_i\neq b_j\end{cases}\end{cases}$$

- 最长公共子串：

$$f[i,j]=\begin{cases}\begin{cases}cc0 & i=0 \text{ or } j=0\\ f[i-1,j-1]+1 & i,j>0 \text{ and } a_i=b_j\\ 0 & i,j>0 \text{ and } a_i\neq b_j\end{cases}\end{cases}$$

- LCS 转换为 LIS：
  需要满足：a,b 为排列或者 a,b 中出现的字母不重复时才可转换
  $\begin{cases}A:3\ 2\ 1\ 4\ 5\\ B:1\ 2\ 3\ 4\ 5\end{cases}\rightarrow\begin{cases}A:a\ b\ c\ d\ e\\ B:c\ b\ a\ d\ e\end{cases}$
  这样标号之后，LCS 长度显然不会改变，但是出现了一个性质：
  两个序列的子序列，一定是 A 的子序列，而 A 本身就是单调递增的，因此这个子序列是单调递增的。

- 公共子序列个数：
  非种类数
  定义 $f_{i,j}$ 表示 a 的前 i 位，b 的前 j 位的公共子序列个数
  于是 $f_{i,j}=f_{i-1,j}+f_{i,j-1}-f_{i-1,j-1}$ （想象一下二维前缀和）
  而当 $a_i=b_j$ 时：
  $\because a_i,b_j$ 可构成一个新的公共子序列，a_i,b_j 又可和 $f_{i-1,j-1}$ 对应的公共子序列构成新的公共子序列
  $\therefore f_{i,j}+=1+f_{i-1,j-1}$

$$f[i][j]=\Big\{\quad\begin{cases}\begin{cases}f[i-1][j]+f[i][j-1]+1 & a_i=b_j\\ f[i-1][j]+f[i][j-1]-f[i-1][j-1] & a_i\neq b_j\end{cases}\end{cases}$$

- 最长公共子序列个数：

  非种类数
  定义 $f_{i,j}$ 表示 a 的前 i 位，b 的前 j 位的最长公共子序列个数
  $dp_{i,j}$ 表示 a 的前 i 位，b 的前 j 位的最长公共子序列
  $dp_{i,j}$ 可由 $dp_{i-1,j-1},dp_{i-1,j},dp_{i,j-1}$ 转移得到，那么：

  1. 当 $a_i=b_j$，$dp_{i,j}$ 必然是由 $dp_{i-1,j-1}$ 转移得到，所以 $f_{i,j}+=f_{i-1,j-1}$ ，此时：
     如果 $dp_{i,j}=dp_{i-1,j}$，那么 $f_{i,j}+=f_{i-1,j}$（也说明 $dp_{i-1}j$ 不是由 $dp_{i-1,j-1}$ 转移得到）
     如果 $dp_{i,j}=dp_{i,j-1}$，那么 $f_{i,j}+=f_{i,j-1}$（也说明 $dp_{i,j-1}$ 不是由 $dp_{i-1,j-1}$ 转移得到）
  2. 当 $a_i\neq b_j$，$dp_{i,j}$ 必然是由 $dp_{i-1,j}$ 或 $dp_{i,j-1}$ 转移得到，此时：
     如果 $dp_{i,j}=dp_{i-1,j}$，那么 $f_{i,j}+=f_{i-1,j}$
     如果 $dp_{i,j}=dp_{i,j-1}$，那么 $f_{i,j}+=f_{i,j-1}$
     而如果 $dp_{i,j}=dp_{i-1,j-1}$，就说明 $dp_{i-1,j},dp_{i,j-1}$ 都是由 $dp_{i-1,j-1}$ 转移得到
     那么 $f_{i,j}$ 就会多计算一次 $f_{i-1,j-1}$ ，需要减掉这部分贡献
  3. 初始化 $f_{i,0}=1,f_{0,i}=1$：因为 $dp_{i,0},dp_{0,i}$ 都等于 0，它们的 LCS 为空集，空集只算一个

- 公共子序列个数：
  种类数
  假设求的是三个串（A、B、C）的公共子序列个数：
  定义 $f_{i,j,k}$ 表示 $a[1...i], b[1...j], c[1...k]$ 互不相同的公共子序列个数。

  - 当 $a_i = b_j = c_k = ch$ 时，会产生新的公共子序列。此时考虑将 ch 接到之前出现的所有公共子序列后，这样就产生了与之前数量相同的新公共子序列，再加上 ch 本身，故此时有 $f_{i,j,k} = 2 \times f_{i-1,j-1,k-1} + 1$。但这样是错的，因为没有考虑重复的情况。
    于是可以记录 $lasta_i, lastb_j, lastc_k$ 表示 ch 在三个字符串中上一次出现的位置，分别记为 li, lj, lk。那么在计算 $f[li][lj][lk]$ 和 $f[i][j][k]$ 时均会在 $f[li-1][lj-1][lk-1]$ 所代表的公共子串末尾接上 ch。当 li, lj, lk 均不为 0 时，这部分就是重复的，还有 ch 本身也是重复的，故要考虑去重。

    $$f_{i,j,k} = 2 \cdot f_{i-1,j-1,k-1} - f_{li-1,lj-1,lk-1}$$

  - 若 $a[i]$、$b[j]$、$c[k]$ 不相等，那么没有新的公共子串产生，此时直接将之前的值继承过来即可。但直接复制 $f[i-1][j-1][k-1]$ 显然是错的，因为未考虑在 $f[i-1][j][k]$ 等处产生的新公共子串。不过很容易可以想到容斥原理：
    $f_{i,j,k} = f_{i-1,j,k} + f_{i,j-1,k} + f_{i,j,k-1} - f_{i-1,j-1,k} - f_{i,j-1,k-1} - f_{i-1,j,k-1} + f_{i-1,j-1,k-1}$.

-

```
// 求最长公共子序列和最长公共子序列的个数
const int mod = 1e8;
const int N = 5e3 + 10;
int f[N][N] , dp[N][N];
int n , m , a[N] , b[N];
char s[N] , t[N];
signed main()
{
   cin >> s + 1 >> t + 1;
   int n = strlen(s + 1) - 1 , m = strlen(t + 1) - 1;
   for(int i = 1 ; i <= n ; i ++) a[i] = s[i] - 'A';
   for(int i = 1 ; i <= m ; i ++) b[i] = t[i] - 'A';
   int now = 0;
   for(int i = 0 ; i <= max(n , m) ; i ++) f[now][i] = 1;
   for(int i = 1 ; i <= n ; i ++)
   {
      now ^= 1;
      f[now][0] = 1;
      for(int j = 1 ; j <= m ; j ++) f[now][j] = 0 , dp[now][j] = 0;
      for(int j = 1 ; j <= m ; j ++)
      {
         if(a[i] == b[j])
         {
            dp[now][j] = dp[now ^ 1][j - 1] + 1;
            f[now][j] += f[now ^ 1][j - 1];
            if(dp[now][j] == dp[now ^ 1][j]) f[now][j] += f[now ^ 1][j];
            if(dp[now][j] == dp[now][j - 1]) f[now][j] += f[now][j - 1];
         }
         else
         {
```

```
                    dp[now][j] = max(dp[now][j - 1] , dp[now ^ 1][j]);
                    if(dp[now][j] == dp[now ^ 1][j]) f[now][j] += f[now ^ 1][j];
                    if(dp[now][j] == dp[now][j - 1]) f[now][j] += f[now][j - 1];
                    if(dp[now][j] == dp[now ^ 1][j - 1]) f[now][j] -= f[now ^ 1][j -
                        1];
                }
                f[now][j] = (f[now][j] + mod) % mod;
            }
        }
        cout << dp[now][m] << '\n';
        cout << f[now][m] << '\n';
        return 0;
    }
```

```
    //公共子序列种类数
    const int N = 1e2 + 10;
    char a[N] , b[N] , c[N];
    int lasta[30] , lastb[30] , lastc[30];
    long long f[N][N][N];
    signed main()
    {
        cin >> a + 1 >> b + 1 >> c + 1;
        int alen = strlen(a + 1) , blen = strlen(b + 1) , clen = strlen(c + 1);
        for(int i = 1 ; i <= alen ; i ++)
        {
            for(int j = 0 ; j <= 25 ; j ++) lastb[j] = 0;
            for(int j = 1 ; j <= blen ; j ++)
            {
                for(int k = 0 ; k <= 25 ; k ++) lastc[k] = 0;
                for(int k = 1 ; k <= clen ; k ++)
                {
                    if(a[i] == b[j] && b[j] == c[k])
                    {
                        f[i][j][k] = f[i - 1][j - 1][k - 1] * 2 + 1;
                        int la = lasta[a[i] - 'a'] , lb = lastb[b[j] - 'a'] , lc =
                            lastc[c[k] - 'a'];
                        if(la && lb && lc) f[i][j][k] -= f[la - 1][lb - 1][lc - 1] +
                            1;
                    }
                    else
                    {
                        f[i][j][k] = f[i - 1][j][k] + f[i][j - 1][k] + f[i][j][k - 1]
                        - f[i - 1][j - 1][k] - f[i][j - 1][k - 1] - f[i - 1][j][k -
                            1]
                        + f[i - 1][j - 1][k - 1];
                    }
                    lastc[c[k] - 'a'] = k;
                }
                lastb[b[j] - 'a'] = j;
            }
            lasta[a[i] - 'a'] = i;
        }
        cout << f[alen][blen][clen] << '\n';
```

```
37          return 0;
38      }
```

6.10.3 最短不公共子串、子序列

1. a 的一个最短的子串，它不是 b 的子串。
2. a 的一个最短的子串，它不是 b 的子序列。
3. a 的一个最短的子序列，它不是 b 的子串。
4. a 的一个最短的子序列，它不是 b 的子序列。

```
1  #include <bits/stdc++.h>
2  #define maxn 4010
3  using namespace std;
4  char s1[maxn], s2[maxn];
5  struct Automata
6  {
7      int root, tot, las;
8      int len[maxn], ch[maxn][30], fa[maxn], last[30];
9      void clear_sam()
10     {
11         tot = las = root = 1;
12         memset(fa, 0, sizeof(fa));
13         memset(ch, 0, sizeof(ch));
14         memset(len, 0, sizeof(len));
15     }
16     void clear_seq()
17     {
18         root = 2010;
19         memset(ch, 0, sizeof(ch));
20         memset(last, 0, sizeof(last));
21     }
22     void insert(int c)
23     {
24         int p = las, np = las = ++tot;
25         len[np] = len[p] + 1;
26         while (p && !ch[p][c])
27         ch[p][c] = np, p = fa[p];
28         if (!p)
29         fa[np] = root;
30         else
31         {
32             int q = ch[p][c];
33             if (len[q] == len[p] + 1)
34             fa[np] = q;
35             else
36             {
37                 int nq = ++tot;
38                 memcpy(ch[nq], ch[q], sizeof(ch[q]));
39                 len[nq] = len[p] + 1, fa[nq] = fa[q], fa[np] = fa[q] = nq;
40                 while (ch[p][c] == q)
```

```
            ch[p][c] = nq, p = fa[p];
        }
    }
}
void sam(char *s)
{
    clear_sam();
    int lenth = strlen(s + 1);
    for (int i = 1; i <= lenth; ++i)
    insert(s[i] - 'a' + 1);
}
void seq(char *s)
{
    clear_seq();
    int lenth = strlen(s + 1);
    for (int i = lenth; i; --i)
    {
        for (int j = 1; j <= 26; ++j)
        ch[i][j] = last[j];
        last[s[i] - 'a' + 1] = i;
    }
    for (int i = 1; i <= 26; ++i)
    ch[root][i] = last[i];
}
} A, B;
struct node
{
    int a, b, len;
};
bool vis[maxn][maxn];
int query()
{
    memset(vis, 0, sizeof(vis));
    queue<node> q;
    q.push((node){A.root, B.root, 0});
    vis[A.root][B.root] = true;
    while (!q.empty())
    {
        node now = q.front();
        q.pop();
        for (int i = 1; i <= 26; ++i)
        {
            int a = A.ch[now.a][i], b = B.ch[now.b][i];
            if (vis[a][b])
            continue;
            if (a && !b)
            return now.len + 1;
            vis[a][b] = true;
            q.push((node){a, b, now.len + 1});
        }
    }
    return -1;
}
```

```
int main()
{
    scanf("%s%s", s1 + 1, s2 + 1);
    A.sam(s1), B.sam(s2), printf("%d\n", query());
    A.sam(s1), B.seq(s2), printf("%d\n", query());
    A.seq(s1), B.sam(s2), printf("%d\n", query());
    A.seq(s1), B.seq(s2), printf("%d\n", query());
    return 0;
}
```

6.10.4 最长公共上升子序列

- 设 $f[i][j]$ 表示在 a 的前 i 个数中取出，且以 $b[j]$ 结尾的 $LCIS$ 长度。
  - 当 $a[i] \neq b[j]$ 时，我们不能选 $a[i]$（因为序列以 $b[j]$ 结尾，所以 $b[j]$ 必须选），答案为 $f[i-1][j]$
  - 当 $a[i] = b[j]$ 时，我们枚举 b 之前可能的结尾，检查能不能把 $a[i]$（即 $b[j]$ ）加入，答案为 $\max\limits_{a[i]>b[k]}\{f[i-1][k]+1\}$，　$0 \leq k < j$
  - 最终答案为每一个可能为 $b[j]$ 结尾的 LIS 长度的最大值，即 $\max\{f[n][j]\}, 1 \leq i \leq m$ 复杂度 $O(n^3)$
- 然后我们考虑 $f[i][j]$ 可以由哪些状态转移过来，设这些状态为 $f[i][j]$ 的状态转移集合。

  我们发现这个集合包含 $f[x][y], 0 \leq x < i, 0 \leq y \leq j$。
  考虑每次转移状态的时候这些状态的变化情况。
  我们就可以发现，已经在状态转移集合里的状态，在 i 或 j 增加时不会离开集合。
  于是开个变量 val，记录当前状态转移集合里的最大值，即 $\max\{f[i-1][k], 0 \leq k < j\}$
  在转移前面状态的同时维护后面状态的 val 即可。

```
// O(N^2)
const int N = 5e3 + 10;
int n , m , a[N] , b[N];
int f[N][N] , pre[N][N] , ma;
void show(int i , int j)
{
    if(!i) return ;
    show(i - 1 , pre[i][j]);
    if(pre[i][j] != j) cout << b[j] << " ";
    return;
}
pair<int , int> LCIS()
{
    rep(i , 1 , n)
    {
        int val = 0;
        rep(j , 1 , m)
        {
            if(a[i] == b[j])
            {
                f[i][j] = f[i - 1][val] + 1;
                pre[i][j] = val;
            }
```

```
            else
            {
                f[i][j] = f[i - 1][j];
                pre[i][j] = j;
            }
            if(b[j] < a[i] && f[i][j] > f[i][val]) val = j;
        }
    }
    int ma = 0;
    rep(j , 0 , m) if(f[n][j] > f[n][ma]) ma = j;
    return make_pair(f[n][ma] , ma);
}
signed main()
{
    cin >> n;
    rep(i , 1 , n) cin >> a[i];
    cin >> m;
    rep(i , 1 , m) cin >> b[i];
    pair<int , int> res = LCIS();
    cout << res.first << '\n';
    show(n , res.second);
    return 0 ;
}
```

```
// O(N^3)
const int N = 5e2 + 10;
int n , m , a[N] , b[N];
int f[N][N] , pre[N][N];
int ans , now;
vector<int>vec;
signed main()
{
    cin >> n;

    rep(i , 1 , n) cin >> a[i];

    cin >> m;

    rep(i , 1 , m) cin >> b[i];

    a[0] = b[0] = -1;

    rep(i , 1 , n) rep(j , 1 , m)
    {
        if(a[i] == b[j])
        {
            rep(k , 0 , j - 1)
            {
                if(a[i] > b[k] && f[i][j] < f[i - 1][k] + 1)
                {
                    f[i][j] = f[i - 1][k] + 1;
                    pre[i][j] = k;
                }
```

```
30          }
31      }
32      else
33      {
34          f[i][j] = f[i - 1][j];
35          pre[i][j] = pre[i - 1][j];
36      }
37  }
38  rep(j , 1 , m) if(ans < f[n][j])
39  {
40      ans = f[n][j];
41      now = j;
42  }
43  while(now)
44  {
45      vec.push_back(b[now]);
46      now = pre[n][now];
47  }
48  reverse(vec.begin() , vec.end());
49  cout << ans << '\n';
50  for(auto i : vec) cout << i << " ";
51  cout << '\n';
52  return 0;
53 }
```

6.10.5 整数划分

- 一个正整数 n 可以表示成若干个正整数之和，形如：$n = n_1 + n_2 + ... + n_k$，其中 $n_1 \geq n_2 \geq ... \geq n_k \geq 1$
- 我们将这样的一种表示称为正整数 n 的一种划分（不考虑 $n_1, n_2, ..., n_k$ 顺序）
- 现在给定一个正整数 n，请你求出 n 共有多少种不同的划分方法

1. 可以认为 $1 \sim n$ 每个数都有无限个，于是可以转换题意为：
   有 $1 \sim n$ 种物品，每种物品都有无限个，求背包容量为 n 时能有多少种装包方案
   那么 $f[n] += f[n-i], i \in [1, n]$
2. 定义 $f[n][m]$ 表示 n 划分成 m 个数的和的方案数
   $(n = n_1 + n_2 + ... + n_m$，其中 n_i 可以为 $0)$
   那么：
   $f[x][y] = f[x][y-1] + f[x-i][y-1] + f[x-2\times i][y-1] + ...$
   $f[x][y] = f[x][y-1] + f[x-i][y]$
   又回到了完全背包。。。
   如果题目要求划分的 m 个数不能为 0（不能为空），那么也好办：
   只要先让 m 个数都 $+1$，再把剩下的 $n-m$ 个数分给 m 个就好，$ans = f[n-m][m]$
3. 如果要求 $n_1, n_2, ..., n_k$ 互不相等
   那么不难发现先出来的数的数量一定不会超过 $O(\sqrt{n})$，即 $m = min(m, \sqrt{n})$
   复杂度为 $O(n\sqrt{n})$
4. 如果 $n_1, n_2, ..., n_k$ 可以相等，但 n 又很大，就需要用到生成函数 & 五边形定理了
   复杂度为 $O(n\sqrt{n})$

5. 如果 $n_1, n_2, ..., n_k$ 可以相等，但要求每个数出现的次数小于 k，那么还是得用上生成函数 & 五边形定理，复杂度为 $O(n\sqrt{n})$

- 一：

```
signed main()

{

    memset(f , 0 , sizeof(f));

    cin >> n;

    f[0] = 1;//一个都不选，那么就只有一种情况

    for(int i = 1 ; i <= n ; i ++)

        for(int j = i ; j <= n ; j ++)

            f[j] = (f[j] + f[j - i]) % mod;

    cout << f[n] << '\n';

    return 0;

}
```

- 二：

```
signed main()

{

    int n , k;

    cin >> n >> k;

    memset(f , 0 , sizeof(f));

    rep(i , 0 , n) f[0][i] = f[i][0] = f[i][1] = f[1][i] = 1;

    rep(i , 2 , n)

    {

        rep(j , 2 , k)

        {

            if(i >= j) f[i][j] = f[i][j - 1] + f[i - j][j];

            else f[i][j] = f[i][j - 1];

        }
```

```
26
27      }
28
29      if(n - k < 0) cout << 0 << '\n';
30
31      else cout << f[n - k][k] << '\n';
32
33      return 0;
34
35  }
```

- 三略
- 四：

```
1   const int N = 1e5 + 8;
2
3   const int mo = 1e9 + 7;
4
5   int dp[N];
6
7   signed main()
8
9   {
10
11      int n = 1e5;
12
13      dp[0] = 1;
14
15      for (int i = 1; i <= n; ++i)
16
17      {
18
19          for (int j = 1, tmp = 1; i >= (3 * j * j - j) / 2 ; ++ j , tmp *= -1)
20
21          {
22
23              int x = (3 * j * j - j) / 2;
24
25              int y = (3 * j * j + j) / 2;
26
27              dp[i] = ((dp[i] + tmp * dp[i - x]) % mo + mo) % mo;
28
29              if (i >= y) dp[i] = ((dp[i] + tmp * dp[i - y]) % mo + mo) % mo;
30
31          }
32
33      }
34
35      int T = 1; cin >> T;
36
37      while (T--)
38
39      {
```

```
40
41          int n ; cin >> n;
42
43          cout << dp[n] << '\n';
44
45      }
46
47      return 0;
48
49  }
```

- 五:

```
1   const int MOD = 1e9 + 7 ;
2
3   const int MAXN = 100001;
4
5   ll get_q(ll x)
6
7   {
8
9       ll ans = 3 * x * x - x;
10
11      return ans/2;
12
13  }
14
15  ll Q[MAXN] , P[MAXN];
16
17  void init()
18
19  {
20
21      Q[0] = 0;
22
23      for(int i = 1 ; i < MAXN ; i ++)
24
25      {
26
27          if(i & 1) Q[i] = get_q(i/2+1);
28
29          else Q[i] = get_q(i/2*(-1));
30
31      }
32
33      P[0] = P[1] = 1;
34
35      for(int i = 2 ; i < MAXN ; i ++)
36
37      {
38
39          for(int j = 1 ; ; j ++)
40
```

```
41          {
42
43              if(Q[j] > i) break;
44
45              int t = j;
46
47              if(t & 1) t = t / 2 + 1;
48
49              else t = t / 2;
50
51              if(t & 1) P[i] = P[i] + P[i - Q[j]];
52
53              else P[i] = P[i] - P[i - Q[j]];
54
55              if(P[i] > MOD) P[i] = P[i]%MOD;
56
57              if(P[i] < 0) P[i] = P[i]+MOD;
58
59          // P[i] = (P[i] % MOD + MOD) % MOD; 这样写容易超时
60
61          }
62
63      }
64
65  }
66
67  ll solve(int n , int k)
68
69  {
70
71      ll ans = 0;
72
73      for(int i = 0 ; ; i ++)
74
75      {
76
77          if(Q[i] * k > n) break;
78
79          int t = i;
80
81          if(t & 1) t = t / 2 + 1;
82
83          else t = t / 2;
84
85          if(t & 1) ans = ans - P[n - Q[i]*k];
86
87          else ans = ans + P[n - Q[i]*k];
88
89          if(ans > MOD) ans = ans % MOD;
90
91          if(ans < 0) ans = ans + MOD;
92
93      // ans = (ans % MOD + MOD) % MOD; // 这样写容易超时
```

```
94
95      }
96
97      return ans;
98
99  }
100
101 signed main()
102
103 {
104
105     init();
106
107     int T = 1;
108
109     cin >> T;
110
111     while(T --)
112
113     {
114
115         int n , k;
116
117         cin >> n >> k;
118
119         cout << solve(n , k) << '\n';
120
121     }
122
123     return 0;
124
125 }
```

7 数据结构

7.1 CDQ 分治

7.1.1 cdq

CDQ 分治是离线的动态算法，会付出 log 的代价。只考虑左区间修改对右区间查询的影响。第一维排序，第二维归并，第三维树状数组。

例题：三维偏序问题。$a_i \le a_j$, $b_i \le b_j$, $c_i \le c_j, i \ne j$ 的二元组 (i, j) 个数。

做法：$ans[i]$ 表示比 (a_i, b_i, c_i) 小的个数。每个三元组相当于是一个修改操作和一个查询操作。我们先按照 a_i 升序，然后 a_i 就变成了操作下标，然后把每个三元组拆成修改操作和查询操作，排序时如果 a_i 相同，修改操作在查询操作之前。$solve(l, r)$ 是对 b_i 降序，然后用 $[l, mid]$ 的操作向树状数组里插 c_i 的值，用 $[mid + 1, r]$ 的操作去询问。

```
1  int n , k ;
2  int ans[maxn] , num[maxn] ;
3  struct node
4  {
5      bool op ;
```

```
    int a , b , c ;
    int id , cur ;
} q[maxn << 1] ;
bool cmp1(node s , node t)
{
    if(s.a == t.a) return s.op < t.op ;
    else return s.a < t.a ;
}
bool cmp2(node s , node t)
{
    if(s.b == t.b) return s.op < t.op ;
    else return s.b < t.b ;
}
struct BIT
{
    int tree[maxm] ;
    void init()
    {
        mem0(tree) ;
    }
    int lowbit(int k)
    {
        return k & -k;
    }
    void add(int n , int x , int k)
    {
        while(x <= n)
        {
            tree[x] += k ;
            x += lowbit(x) ;
        }
    }
    int sum(int x) // sum[l , r] = sum(r) - sum(l - 1)
    {
        int ans = 0 ;
        while(x != 0)
        {
            ans += tree[x] ;
            x -= lowbit(x) ;
        }
        return ans ;
    }
    int query(int l , int r)
    {
        return sum(r) - sum(l - 1) ;
    }
} bit ;
void solve(int l , int r)
{
    if(l == r) return ;
    int mid = (l + r) / 2 ;
    solve(l , mid) ;
    solve(mid + 1 , r) ;
```

```
    sort(q + l , q + r + 1 , cmp2) ; // 如果打乱了顺序，那就只能后序遍历。
    rep(i , l , r)
    if(q[i].op == 0 && q[i].id <= mid) bit.add(k , q[i].c , 1) ;
    else if(q[i].op == 1 && q[i].id > mid) ans[q[i].cur] += bit.query(1 , q[i].c) ;
    rep(i , l , r)
    if(q[i].op == 0 && q[i].id <= mid) bit.add(k , q[i].c , -1) ;
}
int main()
{
    rr(n , k) ;
    rep(i , 1 , n)
    {
        rr(q[i].a , q[i].b) , r(q[i].c) ;
        q[i].op = 0 , q[i].cur = i ;
        q[i + n].a = q[i].a ;
        q[i + n].b = q[i].b ;
        q[i + n].c = q[i].c ;
        q[i + n].op = 1 , q[i + n].cur = i ;
    }
    sort(q + 1 , q + n * 2 + 1 , cmp1) ;
    rep(i , 1 , n * 2) q[i].id = i ;
    mem0(ans) ;
    bit.init() ;
    solve(1 , n * 2) ;
    rep(i , 1 , n) ans[i] -- ;
    return 0 ;
}
```

7.1.2 数值删除、逆序对个数（排列）

- 排列，每次从中删除一个元素，一个删除 m 次，求每次删除后的逆序对数
- 离线，时间复杂度 $O(nlog^2n)$，空间复杂度 $O(n)$

```
const int N = 1e5 + 10;
struct AC
{
    int m, v, d, id, t;
} e[N << 1];
int n, m, tot;
int pos[N], a[N], c[N];
ll ans[N];
bool cmp1(AC x, AC y)
{
    return x.d < y.d;
}
void add(int x, int k)
{
    while (x <= n)
    c[x] += k, x += (x & (-x));
}
int query(int x)
{
```

```
    int su = 0;
    while (x)
    su += c[x], x -= (x & (-x));
    return su;
}
void cdq(int l, int r)
{
    if (l == r)
    return;
    int mid = (l + r) >> 1, j = l;
    cdq(l, mid), cdq(mid + 1, r);
    sort(e + l, e + mid + 1, cmp1);
    sort(e + mid + 1, e + r + 1, cmp1);
    for (int i = mid + 1; i <= r; ++i)
    {
        while (j <= mid && e[j].d <= e[i].d)
        add(e[j].v, e[j].m), ++j;
        ans[e[i].id] += e[i].m * (query(n) - query(e[i].v));
    }
    for (int i = l; i < j; ++i)
    add(e[i].v, -e[i].m);
    j = mid;
    for (int i = r; i > mid; --i)
    {
        while (j >= l && e[j].d >= e[i].d)
        add(e[j].v, e[j].m), --j;
        ans[e[i].id] += e[i].m * query(e[i].v - 1);
    }
    for (int i = mid; i > j; --i)
    add(e[i].v, -e[i].m);
}
signed main()
{
    n = read(), m = read();
    for (int i = 1; i <= n; ++i)
    a[i] = read(), pos[a[i]] = i, e[++tot] = (AC){1, a[i], i, 0, tot};
    for (int i = 1, x; i <= m; ++i)
    x = read(), e[++tot] = (AC){
        -1, x, pos[x], i, tot};
    cdq(1, tot);
    for (int i = 1; i <= m; ++i)
    ans[i] += ans[i - 1];
    for (int i = 0; i < m; ++i)
    printf("%lld\n", ans[i]);
    return 0;
}
```

7.2 splay

7.2.1 插入 x、删除 x、查 x 的排名、查排名为 x 的数、x 的前驱、x 的后继

1. 插入 x 数 $(insert(x))$

2. 删除 x 数 (若有多个相同的数，因只删除一个) ($remove(x)$)
3. 查询 x 数的排名 (排名定义为比当前数小的数的个数 $+1$) ($x\_rank(x)$)
4. 查询排名为 x 的数 ($rank\_x(x)$)
5. 求 x 的前驱 (前驱定义为小于 x，且最大的数) ($x\_pre(x)$)
6. 求 x 的后继 (后继定义为大于 x，且最小的数) ($x\_suf(x)$)

```
const int N = 2e5 + 10;
int ch[N][2], par[N], val[N], cnt[N], size[N], ncnt, root;
bool chk(int x){
   return ch[par[x]][1] == x;
}
void pushup(int x){
   size[x] = size[ch[x][0]] + size[ch[x][1]] + cnt[x];
}
void rotate(int x)
{
   int y = par[x], z = par[y], k = chk(x), w = ch[x][k^1];
   ch[y][k] = w , par[w] = y;
   ch[z][chk(y)] = x , par[x] = z;
   ch[x][k^1] = y , par[y] = x;
   pushup(y) , pushup(x);
}
void splay(int x, int goal = 0)
{
   while (par[x] != goal)
   {
      int y = par[x], z = par[y];
      if (z != goal)
      {
         if (chk(x) == chk(y)) rotate(y);
         else rotate(x);
      }
      rotate(x);
   }
   if (!goal) root = x;
}
void insert(int x)
{
   int cur = root, p = 0;
   while (cur && val[cur] != x)
   {
      p = cur;
      cur = ch[cur][x > val[cur]];
   }
   if (cur) cnt[cur] ++;
   else
   {
      cur = ++ ncnt;
      if (p) ch[p][x > val[p]] = cur;
      ch[cur][0] = ch[cur][1] = 0;
```

```
45          par[cur] = p;
46          val[cur] = x;
47          cnt[cur] = size[cur] = 1;
48      }
49      splay(cur);
50  }
51  void find(int x)
52  {
53      int cur = root;
54      while (ch[cur][x > val[cur]] && x != val[cur])
55      {
56          cur = ch[cur][x > val[cur]];
57      }
58      splay(cur);
59  }
60  int kth(int k)
61  {
62      int cur = root;
63      while (1)
64      {
65          if (ch[cur][0] && k <= size[ch[cur][0]])
66          {
67              cur = ch[cur][0];
68          }
69          else if (k > size[ch[cur][0]] + cnt[cur])
70          {
71              k -= size[ch[cur][0]] + cnt[cur];
72              cur = ch[cur][1];
73          }
74          else return cur;
75      }
76  }
77  int pre(int x)
78  {
79      find(x);
80      if (val[root] < x) return root;
81      int cur = ch[root][0];
82      while (ch[cur][1]) cur = ch[cur][1];
83      return cur;
84  }
85  int suf(int x)
86  {
87      find(x);
88      if (val[root] > x) return root;
89      int cur = ch[root][1];
90      while (ch[cur][0]) cur = ch[cur][0];
91      return cur;
92  }
93  void remove(int x)
94  {
95      int last = pre(x), next = suf(x);
96      splay(last);
97      splay(next, last);
```

```
98      int del = ch[next][0];
99      if (cnt[del] > 1)
100     {
101         cnt[del]--;
102         splay(del);
103     }
104     else ch[next][0] = 0, pushup(next), pushup(root);
105 }
106 int x_rank(int x){
107     find(x);
108     return size[ch[root][0]];
109 }
110 int rank_x(int x){
111     return val[kth(x+1)];
112 }
113 int x_pre(int x){
114     return val[pre(x)];
115 }
116 int x_suf(int x){
117     return val[suf(x)];
118 }
119 int m, op, x;
120 signed main()
121 {
122     cin >> m;
123     insert(0x3f3f3f3f);
124     insert(0xcfcfcfcf);
125     while(m --)
126     {
127         cin >> op >> x;
128         switch (op)
129         {
130             case 1:
131             insert(x);
132             break;
133             case 2:
134             remove(x);
135             break;
136             case 3:
137             cout << x_rank(x) << '\n';
138             break;
139             case 4:
140             cout << rank_x(x) << '\n';
141             break;
142             case 5:
143             cout << x_pre(x) << '\n';
144             break;
145             case 6:
146             cout << x_suf(x) << '\n';
147             break;
148         }
149     }
150     return 0;
```

```
151 }
```

7.2.2 区间插入、区间删除、区间覆盖、区间翻转、区间求和、最大子段和

支持以下 6 种操作：

1. $INSERT(pos, tot)$，在当前数列的第 pos 个数字后插入 tot 个数字：$c_1, c_2 \cdots c_{tot}$
2. $DELETE(pos, tot)$，从当前数列的第 pos 个数字开始连续删除 tot 个数字
3. $UPDATE(pos, tot, y)$，从当前数列的第 pos 个数字开始的连续 tot 个数字统一修改为 y
4. $REVERSE(pos, tot)$，取出从当前数列的第 pos 个数字开始的 tot 个数字，翻转后放入原来的位置
5. $QUERY(pos, tot)$，计算从当前数列的第 pos 个数字开始的 tot 个数字的和并输出
6. $tree[root].mid$，求出当前数列中和最大的一段子串，并输出最大和

```
1  #include <bits/stdc++.h>
2  #define L(node) (tree[node].ch[0])   //替左儿子
3  #define R(node) (tree[node].ch[1])   //替右儿子
4  #define F(node) (tree[node].fa)    //替父亲
5  #define V(node) (tree[node].val)   //替权值
6  #define S(node) (tree[node].size)   //替子树大小
7  #define compare(node, x) (tree[node].val < x) //比较node是权值x的左儿子还是右儿子
8  using namespace std;
9  const int N = 1e6 + 10 , inf = 1e8 + 1 , TAGNONE = 1e7 + 1;
10 int root, cnt, a[N], id[N], rub[N], top, n, m;
11 struct Splay
12 {
13    int ch[2]; //左右儿子
14    int size; //子树大小
15    int fa;  //父亲
16    int tag; //赋值标记
17    int val; //权值
18    int rev; //翻转标记
19    int sum; //区间权值和
20    int l; //左区间，指区间最大前缀和
21    int r; //右区间，指区间最大后缀和
22    int mid; //中区间，指区间最大子段和
23    void clear()
24    {
25       ch[0] = ch[1] = fa = rev = 0; //清空节点信息
26       tag = TAGNONE;
27    }
28 } tree[N];
29 int rublish() //垃圾回收
30 {
31    if(top == 0) return ++ cnt;
32    int node = rub[top --];
33    return node;
34 }
35 void change_val(int node, int val) //更新点值
```

```
{
    if(!node) return;           //空节点返回
    tree[node].tag = tree[node].val = val;      //打赋值标记，更新权值
    tree[node].sum = val * tree[node].size;      //更新区间权值和
    tree[node].l = tree[node].r = max(tree[node].sum, 0); //左右区间更新
    tree[node].mid = max(tree[node].sum, val);    //最大子段和更新
}
void change_rev(int node) //更新翻转
{
    swap(tree[node].ch[0], tree[node].ch[1]); //交换左右儿子
    swap(tree[node].l, tree[node].r); //交换左右区间
    tree[node].rev ^= 1;     //打翻转标记
}
void pushup(int node) //维护信息
{
    Splay &x = tree[L(node)], &y = tree[R(node)];
    int val = tree[node].val; //实质是将左右儿子合并，x代替左儿子，y代替右儿子
    Splay &res = tree[node]; //res代替tree[node]
    res.sum = x.sum + y.sum + val;
    res.size = x.size + y.size + 1;         //权值和更新，子树大小更新
    res.mid = max(max(x.mid, y.mid), x.r + y.l + val); //最大子段和更新
    res.l = max(x.l, x.sum + y.l + val);     //区间最大前缀和更新
    res.r = max(y.r, y.sum + x.r + val);    //区间最大后缀和更新
}
void pushdown(int node) //标记下传
{
    if(tree[node].tag != TAGNONE) //判断有没有赋值标记
    {
        change_val(L(node), tree[node].tag); //更新左儿子
        change_val(R(node), tree[node].tag); //更新右儿子
        tree[node].tag = TAGNONE;   //除去标记
    }
    if(tree[node].rev) //判断有没有翻转标记
    {
        change_rev(L(node)); //更新左儿子
        change_rev(R(node)); //更新右儿子
        tree[node].rev = 0; //除去标记
    }
}
void rotate(int node) //rotate 模板
{
    int fa = F(node);
    int gfa = F(fa);
    int z = tree[fa].ch[1] == node;
    tree[gfa].ch[tree[gfa].ch[1] == fa] = node;
    tree[node].fa = gfa;
    tree[fa].ch[z] = tree[node].ch[z ^ 1];
    tree[tree[node].ch[z ^ 1]].fa = fa;
    tree[node].ch[z ^ 1] = fa;
    tree[fa].fa = node;
    pushup(fa);
    pushup(node);
}
```

```
89  void Splay(int node, int goal) //Splay 模板
90  {
91      while(tree[node].fa != goal)
92      {
93          int fa = F(node);
94          int gfa = F(fa);
95          if(gfa != goal) (compare(fa, tree[node].val)) != (compare(gfa, tree[fa].val))
                ? rotate(node) : rotate(fa);
96          rotate(node);
97      }
98      if(!goal) root = node;
99  }
100 void New(int node, int x) //新建节点
101 {
102     tree[node].mid = tree[node].sum = x; //赋值信息
103     tree[node].tag = TAGNONE;
104     tree[node].rev = 0;         //标记初始化
105     tree[node].l = tree[node].r = max(x, 0); //区间赋值
106     tree[node].size = 1;        //大小赋值
107 }
108 void build(int begin, int end, int fa) //建树
109 {
110     int mid = (begin + end) >> 1;
111     int node = id[mid], pre = id[fa];
112     if(begin == end) New(node, a[begin]); //到达底部，新建一个节点
113     if(begin < mid) build(begin, mid - 1, mid); //建左子树
114     if(mid < end) build(mid + 1, end, mid); //建右子树
115     tree[node].val = a[mid];
116     tree[node].fa = pre;
117     tree[node].tag = TAGNONE; //基本信息赋值
118     pushup(node);   //维护信息
119     tree[pre].ch[mid >= fa] = node;
120 }
121 int kth(int x) //kth模板
122 {
123     int node = root;
124     while (1)
125     {
126         pushdown(node);
127         if(tree[L(node)].size >= x) node = L(node);
128         else if(tree[L(node)].size + 1 == x) return node;
129         else x -= tree[L(node)].size + 1, node = R(node);
130     }
131 }
132 void remove(int node) //将一个子树清空
133 {
134     if (L(node)) remove(L(node)); //继续清空左子树
135     if (R(node)) remove(R(node)); //继续清空右子树
136     rub[++top] = node;
137     tree[node].clear(); //清空并仍进垃圾桶
138 }
139 int split(int k, int len) //找到那个区间的位置
140 {
```

```
141     int x = kth(k), y = kth(k + len + 1);
142     Splay(x, 0);
143     Splay(y, x);
144     return L(y);
145 }
146 int query(int x, int len) //查询区间权值和
147 {
148     int node = split(x, len);  //找到该区间
149     return tree[node].sum;    //返回答案
150 }
151 void update(int x, int len, int val) //更新区间的指
152 {
153     int node = split(x, len), y = F(node); //找到该区间
154     change_val(node, val);    //更新该区间
155     pushup(y);
156     pushup(F(y)); //维护信息
157 }
158 void rever(int x, int len) //翻转区间
159 {
160     int node = split(x, len), y = F(node); //找到该区间
161     if (tree[node].tag != TAGNONE) return; //如果已经有赋值标记就不用管了
162     change_rev(node); //翻转该区间
163     pushup(y);
164     pushup(F(y)); //维护信息
165 }
166 void eraser(int x, int len) //删除区间
167 {
168     int node = split(x, len), y = F(node); //找到该区间
169     remove(node);
170     tree[y].ch[0] = 0; //删除该区间，子树清空
171     pushup(y);
172     pushup(F(y)); //维护信息
173 }
174 void insert(int k, int len) //插入区间
175 {
176     for (int i = 1; i <= len; i++) cin >> a[i]; //输入区间
177     for (int i = 1; i <= len; i++) id[i] = rublish();
178     build(1, len, 0); //将输入的区间建成一个完全二叉树
179     int z = id[(1 + len) >> 1];
180     int x = kth(k + 1), y = kth(k + 2); //找到要插入的位置
181     Splay(x, 0) , Splay(y, x);
182     tree[z].fa = y;
183     tree[y].ch[0] = z; //将新建的子树插入树中
184     pushup(y);
185     pushup(x); //维护信息
186 }
187 signed main()
188 {
189     cin >> n >> m;
190     tree[0].mid = a[1] = a[n + 2] = -inf; //边界
191     for (int i = 1 ; i <= n ; i ++) cin >> a[i + 1];
192     for (int i = 1; i <= n + 2; i ++) id[i] = i;
193     build(1, n + 2, 0); //建成一颗Splay
```

```
194     root = (n + 3) >> 1;
195     cnt = n + 2; //指根，更新点数
196     for (int i = 1 ; i <= m ; i ++)
197     {
198         string s;
199         int pos, len, y;
200         cin >> s;
201         if(s == "MAX-SUM") cout << tree[root].mid << '\n';
202         else
203         {
204             cin >> pos >> len;
205             if(s == "INSERT") insert(pos, len);
206             if(s == "DELETE") eraser(pos, len);
207             if(s == "MAKE-SAME")
208             {
209                 cin >> y;
210                 update(pos, len, y);
211             }
212             if(s == "REVERSE") rever(pos, len);
213             if(s == "GET-SUM") cout << query(pos, len) << '\n';
214         }
215     }
216     return 0;
217 }
```

7.3 dsu on tree

7.3.1 总结

过程

- 对于当前以 $root$ 为根的树，先去处理其轻子树，处理完轻子树后将轻子树所记录的信息删除，等轻子树全部处理完后再去处理其重子树，处理完重子树后将重子树所记录的信息保留，然后回到当前以 $root$ 为根的树，开始处理以 $root$ 为根的树。
- $root$ 保留了其重子树的信息，而没有轻子树的信息，因此需要再次遍历所有轻子树，并计算 $<root$，重子树 $>$、$<root$，轻子树 $>$ 、$<$ 重子树、轻子树 $>$、$<$ 轻子树，轻子树 $>$（不同分会）之间所产生的贡献。等全部计算完后，判断 $root$ 是不是它爸爸的重儿子，如果是就保留 $root$ 及其全部子树的所有信息，若不是则删除 $root$ 及其全部子树的所有信息。

套路

- 如果计算贡献的式子中包含 $dis(u,v)$，那么就把 $dis(u,v)$ 拆解为 $dep_u + dep_v - 2dep_{LCA}$ 于是就可以以 LCA 为 $root$，u,v 为其不同分支中的点，跑 $dsu\ on\ tree$ 计算贡献

7.3.2 CF600E

题目大意

有一棵个结点的以 1 号结点为根的有根树。
每个结点都有一个颜色，颜色是以编号表示的，i 号结点的颜色编号为 c_i
如果一种颜色在以 x 为根的子树内出现次数最多，称其在以 x 为根的子树中占主导地位。

显然，同一子树中可能有多种颜色占主导地位。
你的任务是对于每一个 $i \in [1,n]$，求出以 i 为根的子树中，占主导地位的颜色的编号和。
$N <= 10^5, c_i <= n$

解题思路

dsu on tree 裸题（详见注释）

```
const int N = 3e5 + 10;
struct Edge{
    int nex , to;
}edge[N << 1];
int head[N] , TOT;
void add_edge(int u , int v) // 链式前向星建图
{
    edge[++ TOT].nex = head[u] ;
    edge[TOT].to = v;
    head[u] = TOT;
}
int sz[N]; // sz[u] 表示以 u 为根的子树大小
int hson[N]; // hson[u] 表示 u 的重儿子
int HH; // HH 表示当前根节点的重儿子
int c[N]; //表示每个节点的颜色
int cnt[N]; // cnt[i]表示当前子树中，颜色 i 出现了多少次
int ans[N]; // 每个点为根的答案
int n , res , ma;
// ma 表示出现次数最多的颜色出现的次数
// res 表示出现次数最多的颜色的颜色值总和 （两颜色出现次数相同则都要算）
void dfs(int u , int far)
{
    sz[u] = 1;
    for(int i = head[u] ; i ; i = edge[i].nex) // 链式前向星
    {
        int v = edge[i].to;
        if(v == far) continue ;
        dfs(v , u);
        sz[u] += sz[v];
        if(sz[v] > sz[hson[u]]) hson[u] = v; // 选择 u 的重儿子
    }
}
void calc(int u , int far , int val) // 统计答案
{
    if(val == 1)
    {
        cnt[c[u]] ++ ;
        if(cnt[c[u]] > ma) ma = cnt[c[u]] , res = c[u];
        else if(cnt[c[u]] == ma) res += c[u];
    }
    else cnt[c[u]] -- ;
    for(int i = head[u] ; i ; i = edge[i].nex)
    {
        int v = edge[i].to;
```

```cpp
        if(v == far || v == HH) continue ; // 如果 v 是当前根节点的重儿子，则跳过
        calc(v , u , val);
    }
}
void dsu(int u , int far , int op) // op 等于0表示不保留信息，等于1表示保留信息
{
    for(int i = head[u] ; i ; i = edge[i].nex)
    {
        int v = edge[i].to;
        if(v == far || v == hson[u]) continue ; // 如果 v 是重儿子或者父亲节点就跳过
        dsu(v , u , 0); // 先遍历轻儿子 , op = 0 :轻儿子的答案不做保留
    }
    if(hson[u]) dsu(hson[u] , u , 1) , HH = hson[u];
    // 轻儿子都遍历完了，如果存在重儿子，遍历重儿子（事实上除了叶子节点每个点都必然有重儿子）
    // op = 1 ， 保留重儿子的信息
    // 当前是以 u 为根节点的子树，所以根节点的重儿子 HH = hson[u]
    calc(u , far , 1); // 再次遍历轻儿子统计答案
    ans[u] = res; // 更新答案
    HH = 0;   // 遍历结束 , 即将返回父节点，所以取消标记 HH
    if(!op) calc(u , far , -1) , ma = 0 , res = 0; // 如果 op = -1，则 u 对于它的父亲来说是轻儿子，不需要将信息传递给它的父亲
}
signed main()
{
    cin >> n;
    for(int i = 1 ; i <= n ; i ++) cin >> c[i];
    for(int i = 2 ; i <= n ; i ++)
    {
        int u , v;
        cin >> u >> v;
        add_edge(u , v) , add_edge(v , u);
    }
    dfs(1 , 0);
    dsu(1 , 0 , 0);
    for(int i = 1 ; i <= n ; i ++) cout << ans[i] << " \n"[i == n];
    return 0;
}
```

7.3.3 CF570D

题目大意

给定一个以 1 为根的 n 个节点的树，每个点上有一个字母 $(a-z)$

每个点的深度定义为该节点到 1 号节点路径上的点数.

每次询问 a,b，查询以 a 为根的子树内深度为 b 的节点上的字母重新排列之后是否能构成回文串.

解题思路

重排之后能否构成回文串，即判断出现次数为奇数的字符个数是否小于等于 1

因为要多次询问以某个点为根深度为 b 的节点上的字母

所以可以对每个节点开个 *vector* ， 再把和它有关的询问保存在 *vector* 里

然后在 *dsu on tree* 的过程进行到该节点时遍历这个节点的 *vector* (相关询问)

再对询问做判断 → 得到答案 → 保存答案

最后将保存的答案按照读入的顺序排个序输出即可

```
const int N = 5e5 + 10;
struct Edge
{
    int nex , to;
} edge[N << 2];
int head[N] , TOT;
void add_edge(int u , int v)
{
    edge[++ TOT].nex = head[u];
    edge[TOT].to = v;
    head[u] = TOT;
}
int cnt[N][26] , hson[N] , sz[N] , HH;
int n , m , col[N] , vis[N];
vector<pair<int ,int>>Q[N] , ans;
void dfs(int u , int far)
{
    for(int i = head[u] ; i ; i = edge[i].nex)
    {
        int v = edge[i].to;
        if(v == far) continue ;
        dfs(v , u);
        sz[u] += sz[v];
        if(sz[v] > sz[hson[u]]) hson[u] = v;
    }
    sz[u] ++;
}
void calc(int u , int far , int val , int dep)
{
    cnt[dep][col[u]] += val;
    for(int i = head[u] ; i ; i = edge[i].nex)
    {
        int v = edge[i].to;
        if(v == far || v == HH) continue ;
        calc(v , u , val , dep + 1);
    }
}
void dsu(int u , int far , int op , int dep)
{
    for(int i = head[u] ; i ; i = edge[i].nex)
    {
        int v = edge[i].to;
        if(v == hson[u] || v == far) continue ;
        dsu(v , u , 0 , dep + 1);
    }
    if(hson[u]) dsu(hson[u] , u , 1 , dep + 1) , HH = hson[u];
    calc(u , far , 1 , dep);
```

```
48      for(auto i : Q[u])
49      {
50          int b = i.fi , id = i.se , res = 0;
51          rep(j , 0 , 25)
52          {
53              if(cnt[b][j] & 1) res ++ ;
54          }
55          if(res > 1) ans.pb(make_pair(id , 0));
56          else ans.pb(make_pair(id , 1));
57      }
58      HH = 0;
59      if(!op) calc(u , far , -1 , dep);
60  }
61  signed main()
62  {
63      cin >> n >> m;
64      rep(i , 2 , n)
65      {
66          int v;
67          cin >> v;
68          add_edge(i , v) , add_edge(v , i);
69      }
70      rep(i , 1 , n)
71      {
72          char ch;
73          cin >> ch;
74          col[i] = ch - 'a';
75      }
76      rep(i , 1 , m)
77      {
78          int a , b;
79          cin >> a >> b;
80          Q[a].pb(make_pair(b , i));
81      }
82      dfs(1 , 0);
83      dsu(1 , 0 , 0 , 1);
84      sort(ans.begin() , ans.end());
85      for(auto i : ans)
86      if(i.se) cout << "Yes\n" ;
87      else cout << "No\n";
88      return 0;
89  }
```

7.3.4 CF208E

题目大意

给你一片森林，每次询问一个点与多少个点拥有共同的 K 级祖先

解题思路

询问一个点有多少个共同的 K 级祖先

可以等价于问它的 K 级祖先有多少个深度为 $dep + K$ 的子节点 (dep 为 K 级祖先的深度)

那么对于每个询问要先倍增或者树链剖分求出它的 K 级祖先，再将询问保存在 K 级祖先的 $vector$ 里

然后跑一遍 dsu on tree 统计 + 保存答案，最后再将保存的答案按照读入的询问顺序排个序输出即可

```
const int N = 5e5 + 10;
struct Edge{
   int nex , to;
}edge[N << 1];
int head[N] , tot;
int a[N] , dep[N] , f[N][30] , siz[N] , hson[N] , HH;
vector<pair<int , int>>Q[N] , ans;
unordered_map<int , int>cnt;
void add_edge(int u , int v)
{
   edge[++ tot].nex = head[u];
   edge[tot].to = v;
   head[u] = tot;
}
void dfs(int u , int far)
{
   siz[u] = 1;
   dep[u] = dep[far] + 1;
   f[u][0] = far;
   for(int i = 1 ; (1 << i) <= dep[u] ; i ++)
   f[u][i] = f[f[u][i - 1]][i - 1];
   for(int i = head[u] ; i ; i = edge[i].nex)
   {
      int v = edge[i].to;
      if(v == far) continue ;
      dfs(v , u);
      siz[u] += siz[v];
      if(siz[v] > siz[hson[u]]) hson[u] = v;
   }
}
void calc(int u , int far , int val , int dep)
{
   cnt[dep] += val;
   for(int i = head[u] ; i ; i = edge[i].nex)
   {
      int v = edge[i].to;
      if(v == far || v == HH) continue ;
      calc(v , u , val , dep + 1);
   }
}
void dsu(int u , int far, int op , int dep)
{
   for(int i = head[u] ; i ; i = edge[i].nex)
   {
      int v = edge[i].to;
```

```
46        if(v == far || v == hson[u]) continue ;
47        dsu(v , u , 0 , dep + 1);
48    }
49    if(hson[u]) dsu(hson[u] , u , 1 , dep + 1) , HH = hson[u];
50    calc(u , far , 1 , dep);
51    for(auto i : Q[u])
52    {
53        int k = i.fi , id = i.se;
54        ans.pb(make_pair(id , cnt[dep + k] - 1));
55    }
56    HH = 0;
57    if(!op) calc(u , far , -1 , dep);
58 }
59 int get_far(int x , int k)
60 {
61    if(!k) return x;
62    int t = dep[x] - k;
63    per(i , 20 , 0) if(dep[f[x][i]] > t) x = f[x][i];
64    return f[x][0];
65 }
66 signed main()
67 {
68    int n , q;
69    cin >> n;
70    rep(i , 1 , n)
71    {
72        cin >> a[i];
73        if(!a[i]) continue ;
74        add_edge(i , a[i]) , add_edge(a[i] , i);
75    }
76    rep(i , 1 , n)
77    {
78        if(!a[i]) dfs(i , 0);
79    }
80    cin >> q;
81    rep(i , 1 , q)
82    {
83        int a , b;
84        cin >> a >> b;
85        int x = get_far(a , b);
86        if(!x) {ans.pb(make_pair(i , 0)) ; continue ;}
87        Q[x].pb(make_pair(b , i));
88    }
89    rep(i , 1 , n)
90    {
91        if(!a[i]) dsu(i , 0 , 0 , 1) , cnt.clear();
92    }
93    sort(ans.begin() , ans.end());
94    for(auto i : ans) cout << i.se << " ";
95    cout << '\n';
96    return 0;
97 }
```

7.3.5 CF246E

题目大意

给定一片森林，每次询问一个节点的 $K-Son$ 共有个多少不同的名字。一个节点的 $K-Son$ 即为深度是该节点深度加 K 的节点。

解题思路

比较裸的 dsu on tree

统计不同名字开个 $map<string,int>$ 即可

也可以将字符串离散成数字再操作

```
const int N = 5e5 + 10;
struct Edge{
    int nex , to;
}edge[N << 1];
int head[N] , tot;
int a[N] , dep[N] , siz[N] , hson[N] , name[N] , HH;
vector<pair<int , int>>Q[N] , ans;
unordered_map<int , int>cnt[N] , sum;
void add_edge(int u , int v)
{
    edge[++ tot].nex = head[u];
    edge[tot].to = v;
    head[u] = tot;
}
void dfs(int u , int far)
{
    siz[u] = 1;
    dep[u] = dep[far] + 1;
    for(int i = head[u] ; i ; i = edge[i].nex)
    {
        int v = edge[i].to;
        if(v == far) continue ;
        dfs(v , u);
        siz[u] += siz[v];
        if(siz[v] > siz[hson[u]]) hson[u] = v;
    }
}
void calc(int u , int far , int val , int dep)
{
    cnt[dep][name[u]] += val;
    if(cnt[dep][name[u]] == 1 && val == 1) sum[dep] ++ ;
    if(!cnt[dep][name[u]]) sum[dep] -- ;
    for(int i = head[u] ; i ; i = edge[i].nex)
    {
        int v = edge[i].to;
        if(v == far || v == HH) continue ;
        calc(v , u , val , dep + 1);
    }
}
void dsu(int u , int far, int op , int dep)
```

```
{
    for(int i = head[u] ; i ; i = edge[i].nex)
    {
        int v = edge[i].to;
        if(v == far || v == hson[u]) continue ;
        dsu(v , u , 0 , dep + 1);
    }
    if(hson[u]) dsu(hson[u] , u , 1 , dep + 1) , HH = hson[u];
    calc(u , far , 1 , dep);
    for(auto i : Q[u])
    {
        int k = i.fi , id = i.se;
        ans.pb(make_pair(id , sum[dep + k]));
    }
    HH = 0;
    if(!op) calc(u , far , -1 , dep) , sum.clear();
}
map<string , int>vis;
signed main()
{
    int n , q , tot = 0;
    cin >> n;
    rep(i , 1 , n)
    {
        string s;
        int x;
        cin >> s >> x;
        if(!vis.count(s)) vis[s] = ++ tot;
        name[i] = vis[s] , a[i] = x;
        if(!x) continue ;
        add_edge(i , x) , add_edge(x , i);
    }
    rep(i , 1 , n) if(!a[i]) dfs(i , 0);
    cin >> q;
    rep(i , 1 , q)
    {
        int a , b;
        cin >> a >> b;
        Q[a].pb(make_pair(b , i));
    }
    rep(i , 1 , n)
    {
        if(!a[i]) dsu(i , 0 , 0 , 1) ;
    }
    sort(ans.begin() , ans.end());
    for(auto i : ans) cout << i.se << '\n';
    return 0;
}
```

7.3.6 CF375D

题目大意

给定一棵 n 个节点的树，根节点为 1。每个节点上有一个颜色 c_i
m 次询问。
每次询问给出 u k：询问在以 u 为根的子树中，出现次数 $\geq k$ 的颜色有多少种。

解题思路

可以开棵权值线段树

如果当前颜色出现的次数 $cnt[i] = x$, 就把树的第 x 个位置的值 $+1$

那么对于每个询问的 k 输出树的第 k 个位置的值即可

```
const int N = 3e5 + 10;
struct Tree{
    int l , r , lazy , sum;
}tree[N << 2];
void push_up(int rt)
{
    tree[rt].sum = tree[rt << 1].sum + tree[rt << 1 | 1].sum;
}
void push_down(int rt)
{
    int x = tree[rt].lazy;
    tree[rt].lazy = 0;
    tree[rt << 1].lazy = tree[rt << 1 | 1].lazy = x;
    tree[rt << 1].sum += (tree[rt << 1].r - tree[rt << 1].l + 1) * x;
    tree[rt << 1 | 1].sum += (tree[rt << 1 | 1].r - tree[rt << 1 | 1].l + 1) * x;
}
void build(int l , int r , int rt)
{
    tree[rt].l = l , tree[rt].r = r , tree[rt].lazy = 0;
    if(l == r)
    {
        tree[rt].sum = 0;
        return ;
    }
    int mid = l + r >> 1;
    build(l , mid , rt << 1);
    build(mid + 1 , r , rt << 1 | 1);
    push_up(rt);
}
void update_range(int L , int R , int rt , int val)
{
    int l = tree[rt].l , r = tree[rt].r;
    if(L <= l && r <= R)
    {
        tree[rt].lazy += val;
        tree[rt].sum += (r - l + 1) * val;
        return ;
    }
    push_down(rt);
    int mid = l + r >> 1;
    if(L <= mid) update_range(L , R , rt << 1 , val);
    if(R > mid) update_range(L , R , rt << 1 | 1 , val);
```

```
43      push_up(rt);
44  }
45  int query_range(int L , int R , int rt)
46  {
47      int l = tree[rt].l , r = tree[rt].r;
48      if(L <= l && r <= R) return tree[rt].sum;
49      push_down(rt);
50      int mid = l + r >> 1 , ans = 0;
51      if(L <= mid) ans += query_range(L , R , rt << 1);
52      if(R > mid) ans += query_range(L , R , rt << 1 | 1);
53      return ans;
54  }
55  struct Edge{
56      int nex , to;
57  }edge[N << 1];
58  int head[N] , TOT;
59  void add_edge(int u , int v)
60  {
61      edge[++ TOT].nex = head[u] ;
62      edge[TOT].to = v;
63      head[u] = TOT;
64  }
65  int dep[N] , sz[N] , hson[N] , HH;
66  int col[N] , n , m , up;
67  int cnt[N] , sum[N];
68  vector<pair<int , int>>Q[N] , ans;
69  void dfs(int u , int far)
70  {
71      dep[u] = dep[far] + 1;
72      sz[u] = 1;
73      for(int i = head[u] ; i ; i = edge[i].nex)
74      {
75          int v = edge[i].to;
76          if(v == far) continue ;
77          dfs(v , u);
78          sz[u] += sz[v];
79          if(sz[v] > sz[hson[u]]) hson[u] = v;
80      }
81  }
82  void calc(int u , int far, int val)
83  {
84      cnt[col[u]] += val;
85      if(val == 1)
86      {
87          int k = cnt[col[u]];
88          update_range(k , k , 1 , 1);
89      }
90      if(val == -1)
91      {
92          int k = cnt[col[u]] + 1;
93          update_range(k , k , 1 , -1);
94      }
95      for(int i = head[u] ; i ; i = edge[i].nex)
```

```
96      {
97          int v = edge[i].to;
98          if(v == far || v == HH) continue ;
99          calc(v , u , val);
100     }
101 }
102 void dsu(int u , int far , int op)
103 {
104     for(int i = head[u] ; i ; i = edge[i].nex)
105     {
106         int v = edge[i].to;
107         if(v == far || v == hson[u]) continue ;
108         dsu(v , u , 0);
109     }
110     if(hson[u]) dsu(hson[u] , u , 1) , HH = hson[u];
111     calc(u , far , 1);
112     for(auto i : Q[u])
113     {
114         int id = i.fi , k = i.se;
115         int res = query_range(k , k , 1);
116         ans.pb(make_pair(id , res));
117     }
118     HH = 0;
119     if(!op) calc(u , far , -1);
120 }
121 signed main()
122 {
123     ios::sync_with_stdio(false);
124     cin.tie(0) , cout.tie(0);
125     cin >> n >> m;
126     rep(i , 1 , n) cin >> col[i];
127     rep(i , 1 , n - 1)
128     {
129         int u , v;
130         cin >> u >> v;
131         add_edge(u , v) , add_edge(v , u);
132     }
133     rep(i , 1 , m)
134     {
135         int u , k;
136         cin >> u >> k;
137         Q[u].pb(make_pair(i , k));
138     }
139     build(1 , 100000 , 1);
140     dfs(1 , 0);
141     dsu(1 , 0 , 0);
142     sort(ans.begin() , ans.end());
143     for(auto i : ans) cout << i.se << '\n';
144     return 0;
145 }
```

7.3.7 CF1009F

题目大意

给定一棵以 1 为根，n 个节点的树。设 $d(u,x)$ 为 u 子树中到 u 距离为 x 的节点数。对于每个点，求一个最小的 k，使得 $d(u,k)$ 最大。

解题思路

记录子树每个深度的节点的个数，然后取个最大节点个数的最小深度即可

```
const int N = 1e6 + 10;
struct Edge{
   int nex , to;
}edge[N << 1];
int head[N] , tot;
int a[N] , dep[N] , siz[N] , hson[N] , HH;
int ans[N] , cnt[N] , ma , res;
void add_edge(int u , int v)
{
   edge[++ tot].nex = head[u];
   edge[tot].to = v;
   head[u] = tot;
}
void dfs(int u , int far)
{
   siz[u] = 1;
   dep[u] = dep[far] + 1;
   for(int i = head[u] ; i ; i = edge[i].nex)
   {
      int v = edge[i].to;
      if(v == far) continue ;
      dfs(v , u);
      siz[u] += siz[v];
      if(siz[v] > siz[hson[u]]) hson[u] = v;
   }
}
void calc(int u , int far , int val , int dep)
{
   cnt[dep] += val;
   if(cnt[dep] > ma) ma = cnt[dep] , res = dep;
   else if(cnt[dep] == ma) res = min(res , dep);
   for(int i = head[u] ; i ; i = edge[i].nex)
   {
      int v = edge[i].to;
      if(v == far || v == HH) continue ;
      calc(v , u , val , dep + 1);
   }
}
void dsu(int u , int far, int op , int dep)
{
   for(int i = head[u] ; i ; i = edge[i].nex)
   {
      int v = edge[i].to;
```

```
44        if(v == far || v == hson[u]) continue ;
45        dsu(v , u , 0 , dep + 1);
46    }
47    if(hson[u]) dsu(hson[u] , u , 1 , dep + 1) , HH = hson[u];
48    calc(u , far , 1 , dep);
49    HH = 0;
50    ans[u] = res - dep;
51    if(!op) calc(u , far , -1 , dep) , ma = 0 , res = 0;
52 }
53 signed main()
54 {
55    int n , q , tot = 0;
56    cin >> n;
57    rep(i , 1 , n - 1)
58    {
59        int u , v;
60        cin >> u >> v;
61        add_edge(u , v) , add_edge(v , u);
62    }
63    dfs(1 , 0);
64    dsu(1 , 0 , 0 , 0);
65    rep(i , 1 , n) cout << ans[i] << '\n';
66    return 0;
67 }
```

7.3.8 wannafly Day2 E

题目大意

给你一个 n 个节点的树，求每个节点的”结实程度”

一个节点的结实程度定义为以该节点为根的子树里所有节点的编号从小到大排列后，相邻编号的平方和。

解题思路

假设一个节点的子树中所有节点编号排序后构成的序列为 $a1, a2, a3.....ak$, 那么答案为 $\sum_{i=1}^{k-1}(a_{i+1}-a_i)^2$

因为每个数只会和它的前驱后继有关，所以我们可以开个 set 来维护每次添加一个数/每次删除一个数对当前答案的影响

```
1  using namespace std;
2  const int N = 3e5 + 10 , inf = 0x3f3f3f3f;
3  struct Edge{
4     int nex , to;
5  }edge[N << 1];
6  int head[N] , TOT;
7  void add_edge(int u , int v)
8  {
9     edge[++ TOT].nex = head[u] ;
10    edge[TOT].to = v;
11    head[u] = TOT;
12 }
```

```
13 int hson[N] , dep[N] , sz[N] , HH;
14 int n , ans[N] , now;
15 set<int>se;
16 void dfs(int u , int far)
17 {
18     dep[u] = dep[far] + 1;
19     sz[u] = 1;
20     for(int i = head[u] ; i ; i = edge[i].nex)
21     {
22         int v = edge[i].to;
23         if(v == far) continue ;
24         dfs(v , u);
25         sz[u] += sz[v];
26         if(sz[v] > sz[hson[u]]) hson[u] = v;
27     }
28 }
29 void calc(int u , int far , int val)
30 {
31     if(val == 1)
32     {
33         auto it = se.upper_bound(u);
34         auto it1 = it , it2 = it;
35         it1 -- ;
36         int add = 0 , add1 = 0 , add2 = 0;
37         if(it1 != se.begin() && it2 != se.end()) add = *it2 - *it1;
38         now -= add * add;
39         if(it1 != se.begin()) add1 = u - *it1;
40         if(it2 != se.end()) add2 = *it2 - u;
41         now += add1 * add1 + add2 * add2;
42         se.insert(u);
43     }
44     else if(val == -1)
45     {
46         se.erase(u);
47         auto it = se.upper_bound(u);
48         auto it1 = it , it2 = it;
49         it1 -- ;
50         int add = 0 , add1 = 0 , add2 = 0;
51         if(it1 != se.begin()) add1 = u - *it1;
52         if(it2 != se.end()) add2 = *it2 - u;
53         now -= add1 * add1 , now -= add2 * add2;
54         if(it1 != se.begin() && it2 != se.end()) add = *it2 - *it1;
55         now += add * add;
56     }
57     for(int i = head[u] ; i ; i = edge[i].nex)
58     {
59         int v = edge[i].to;
60         if(v == far || v == HH) continue ;
61         calc(v , u , val);
62     }
63 }
64 void dsu(int u , int far , int op)
65 {
```

```
66      for(int i = head[u] ; i ; i = edge[i].nex)
67      {
68          int v = edge[i].to;
69          if(v == far || v == hson[u]) continue ;
70          dsu(v , u , 0);
71      }
72      if(hson[u]) dsu(hson[u] , u , 1) , HH = hson[u];
73      calc(u , far , 1);
74      ans[u] = now;
75      HH = 0;
76      if(!op) calc(u , far , -1) , now = 0;
77  }
78  signed main()
79  {
80      cin >> n;
81      se.insert(-inf);
82      rep(i , 2 , n)
83      {
84          int x;
85          cin >> x;
86          add_edge(i , x) , add_edge(x , i);
87      }
88      dfs(1 , 0);
89      dsu(1 , 0 , 0);
90      rep(i , 1 , n) cout << ans[i] << '\n';
91      return 0;
92  }
```

7.3.9 ccpc2020 长春站 F 题

题目大意

给定一棵包含 n 个节点的树，每个节点有个权值 a_i

求 $\sum_{i=1}^{n}\sum_{j=i+1}^{n}[a_i \oplus a_j = a_{lca(i,j)}](i \oplus j)$.

解题思路

题目保证了 $a_i \neq 0$ ，所以不存在 $a_u \oplus a_v = a_u$，即满足条件的 $a_u \oplus a_v = a_{lca(u,v)}$ 的 u, v 一定在不同分支

这点极大的简单化了本问题

于是在以 rt 为根的子树中，对于节点 u ，满足条件的点的异或值为 $a_u \oplus a_{rt}$

而 u 对答案产生的贡献只和 u 在二进制下每一位的数值有关系

于是我们可以定义 f_{ijk} 表示异或值为 i 的数, 它们在二进制下第 j 位为 k 的个数

那么对于 u , 它的贡献可以这么算

```
1  int x = a[u] ^ a[rt];
2  if(x <= 1000000) // a[i] <= 1e6
3  {
4      for(int i = 17 ; i >= 0 ; i --)
5      {
6          int k = u >> i & 1;
```

```
        ans += (1LL << i) * f[x][i][k ^ 1];
    }
}
```

到这本题就差不多结束了

别忘了一个分支内的任意节点不能相互影响，所以需要先对一个分支统计完贡献后，再添加它的信息

```
const int N = 1e5 + 10 , M = 1e6 + 10;
struct Edge{
    int nex , to;
}edge[N << 2];
int head[N] , TOT;
void add_edge(int u , int v)
{
    edge[++ TOT].nex = head[u];
    edge[TOT].to = v;
    head[u] = TOT;
}
int dep[N] , sz[N] , hson[N] , HH;
int a[N] , f[M][20][2];
ll ans;
void dfs(int u , int far)
{
    sz[u] = 1;
    dep[u] = dep[far] + 1;
    for(int i = head[u] ; i ; i = edge[i].nex)
    {
        int v = edge[i].to;
        if(v == far) continue ;
        dfs(v , u);
        sz[u] += sz[v];
        if(sz[v] > sz[hson[u]]) hson[u] = v;
    }
}
void change(int u , int far , int val)
{
    for(int i = 17 ; i >= 0 ; i --) f[a[u]][i][u >> i & 1] += val;
    for(int i = head[u] ; i ; i = edge[i].nex)
    {
        int v = edge[i].to;
        if(v == far || v == HH) continue ;
        change(v , u , val);
    }
}
void calc(int u , int far , int rt)
{
    int x = a[u] ^ a[rt];
    if(x <= 1000000)
    {
        for(int i = 17 ; i >= 0 ; i --)
        {
```

```
45          int k = u >> i & 1;
46          ans += (1LL << i) * f[x][i][k ^ 1];
47       }
48    }
49    for(int i = head[u] ; i ; i = edge[i].nex)
50    {
51       int v = edge[i].to;
52       if(v == far || v == HH) continue ;
53       calc(v , u , rt);
54    }
55 }
56 void dsu(int u , int far , int op)
57 {
58    for(int i = head[u] ; i ; i = edge[i].nex)
59    {
60       int v = edge[i].to;
61       if(v == far || v == hson[u]) continue ;
62       dsu(v , u , 0);
63    }
64    if(hson[u]) dsu(hson[u] , u , 1) , HH = hson[u];
65    for(int i = head[u] ; i ; i = edge[i].nex)
66    {
67       int v = edge[i].to ;
68       if(v == far || v == HH) continue ;
69       calc(v , u , u) , change(v , u , 1);
70    }
71    HH = 0;
72    for(int i = 17 ; i >= 0 ; i --) f[a[u]][i][(u >> i) & 1] ++ ;
73    if(!op) change(u , far , -1);
74 }
75 signed main()
76 {
77    ios::sync_with_stdio(false);
78    int n ;
79    cin >> n;
80    rep(i , 1 , n) cin >> a[i];
81    rep(i , 2 , n)
82    {
83       int u , v;
84       cin >> u >> v;
85       add_edge(u , v) , add_edge(v , u);
86    }
87    dfs(1 , 0);
88    dsu(1 , 0 , 0);
89    cout << ans << '\n';
90    return 0;
91 }
```

7.3.10 洛谷 P4149

题目大意

给出一棵树，每条边有权。求一条简单路径，使得路径和等于 k，且边的数量最小。

问最小数量是多少 (若没有满足条件的则输出 -1)

解题思路

定义 dis_u 表示节点 u 到根节点的距离，dep_u 表示节点 u 的深度

那么树上任意两点 u, v 的简单路径和等于 $dis_u + dis_v - 2 \times dis_{lca(u,v)}$

两点之间边的数量为 $dep_u + dep_v - 2 \times dep_{lca(u,v)}$

那么问题就转换成在树上找到两点 u, v 使得 $dis_u + dis_v - 2 \times dis_{lca(u,v)} = k$ 且 $dep_u + dep_v - 2 \times dep_{lca(u,v)}$ 尽可能小

假设当前根节点为 rt ，那么对于子节点 x 的来说，能与它构成满足上述式子的 y 可能来自 rt 的其它分支，也可能 $y = rt$

(相同分支内的节点答案的在根节点为 rt 的子节点的时候就已经算过了)

于是我们可以定义 mp_d 表示当前距离 1 号点距离为 d 的节点的最小深度，定义 $c = k + 2 * dis_{rt} - disx$

那么 x 提供的贡献就是 $mp_c + dep_x - 2 * dep_{rt}$

因为我们计算两点的简单路径权值和、两点简单路径的边的个数是根据它们的 lca，所以一个分支内的节点不能相互影响

所以需要先对一个分支统计完贡献后，再添加它的信息

(事实上 rt 的任意一个分支内的贡献在统计子节点为根的子树中就已经计算过了)

```
const int N = 3e5 + 10;
struct Edge{
    int nex , to , w;
}edge[N << 1];
int head[N] , TOT;
void add_edge(int u , int v , int w)
{
    edge[++ TOT].nex = head[u] ;
    edge[TOT].to = v;
    edge[TOT].w = w;
    head[u] = TOT;
}
map<int , int>mp;
int n , k , ans = 0x3f3f3f3f;
int hson[N] , HH , sz[N] , dep[N] , dis[N];
void dfs(int u , int far)
{
    sz[u] = 1;
    dep[u] = dep[far] + 1;
    for(int i = head[u] ; i ; i = edge[i].nex)
    {
        int v = edge[i].to , w = edge[i].w;
        if(v == far) continue ;
        dis[v] = w + dis[u];
        dfs(v , u);
        sz[u] += sz[v];
        if(sz[v] > sz[hson[u]]) hson[u] = v;
    }
}
```

```cpp
void change(int u , int far)
{
    if(mp.count(dis[u])) mp[dis[u]] = min(mp[dis[u]] , dep[u]);
    else mp[dis[u]] = dep[u];
    for(int i = head[u] ; i ; i = edge[i].nex)
    {
        int v = edge[i].to;
        if(v == far || v == HH) continue ;
        change(v , u);
    }
}
void calc(int u , int far , int rt)
{
    int x = k + 2 * dis[rt] - dis[u];
    if(mp.count(x)) ans = min(ans , mp[x] + dep[u] - 2 * dep[rt]);
    for(int i = head[u] ; i ; i = edge[i].nex)
    {
        int v = edge[i].to;
        if(v == far || v == HH) continue ;
        calc(v , u , rt);
    }
}
void dsu(int u , int far , int op)
{
    for(int i = head[u] ; i ; i = edge[i].nex)
    {
        int v = edge[i].to;
        if(v == far || v == hson[u]) continue ;
        dsu(v , u , 0);
    }
    if(hson[u]) dsu(hson[u] , u , 1) , HH = hson[u];
    int x = k + dis[u];
    if(mp.count(x)) ans = min(ans , mp[x] + dep[u] - 2 * dep[u]);
    mp[dis[u]] = dep[u];
    for(int i = head[u] ; i ; i = edge[i].nex)
    {
        int v = edge[i].to , w = edge[i].w;
        if(v == far || v == HH) continue ;
        calc(v , u , u) , change(v , u);
    }
    HH = 0;
    if(!op) mp.clear();
}
signed main()
{
    cin >> n >> k;
    rep(i , 1 , n - 1)
    {
        int u , v , w;
        cin >> u >> v >> w;
        u ++ , v ++ ;
        add_edge(u , v , w) , add_edge(v , u , w);
    }
```

```
    dfs(1 , 0);
    dsu(1 , 0 , 0);
    if(ans == 0x3f3f3f3f) cout << -1 << '\n';
    else cout << ans << '\n';
    return 0;
}
```

7.3.11 牛客练习赛 60E

题目大意

给你一棵以 1 为根节点，包含 n 个节点的树和一个参数 k，求每个节点的"$rating$"

$rating$ 值的计算方式是这样的，对于 u 的子树中的所有节点，如果 x,y 满足 $dis(x,y)=k$

并且 x,y 的最近公共祖先是 u 且满足 $u!=x,u!=y$，那么 u 的 $rating$ 就会增加 a_x+a_y

解题思路

因为 x,y 的最近公共祖先为 u ，所以 x,y 一定在 u 子树的不同分支

$dis(x,y)=k$ 等价于 $dep[x]+dep[y]-2\times dep[lca(x,y)]=k$

于是可以先用 cnt[dep] 记录深度为 dep 的节点出现的次数，用 sum[dep] 记录 dep 的节点的权值和

那么对于 rt 为根的子节点 u，与其相匹配的点的深度为 $d=k+2*dep[rt]-dep[u]$

它对 rt 产生的贡献就为 $cnt_d\times a_u$ ，而深度为 d 的点因为 u 的出现对 rt 的贡献都会翻倍

所以 u 节点的出现对 rt 的总贡献为 $sum[d]+a[u]*cnt[d]$

又因为与 u 节点产生贡献的节点必须和 u 不在一个分支，即一个分支内的任意节点不能相互影响

所以需要先对一个分支统计完贡献后，再添加它的信息

```
const int N = 3e5 + 10;
struct Edge{
    int nex , to;
}edge[N << 1];
int head[N] , TOT;
void add_edge(int u , int v)
{
    edge[++ TOT].nex = head[u] ;
    edge[TOT].to = v;
    head[u] = TOT;
}
int n , k , sum[N] , cnt[N] , ans[N];
int dep[N] , sz[N] , HH , hson[N] , f[N][30] , a[N];
void dfs(int u , int far)
{
    sz[u] = 1;
    dep[u] = dep[far] + 1;
    for(int i = head[u] ; i ; i = edge[i].nex)
```

```
    {
        int v = edge[i].to;
        if(v == far) continue ;
        dfs(v , u);
        sz[u] += sz[v];
        if(sz[v] > sz[hson[u]]) hson[u] = v;
    }
}
void change(int u , int far , int val)
{
    sum[dep[u]] += val * a[u];
    cnt[dep[u]] += val;
    for(int i = head[u] ; i ; i = edge[i].nex)
    {
        int v = edge[i].to;
        if(v == far || v == HH) continue ;
        change(v , u , val);
    }
}
void calc(int u , int far , int rt)
{
    int c = k + 2 * dep[rt] - dep[u];
    if(c < 0) return ;
    ans[rt] += sum[c] + a[u] * cnt[c];
    for(int i = head[u] ; i ; i = edge[i].nex)
    {
        int v = edge[i].to;
        if(v == far || v == HH) continue ;
        calc(v , u , rt);
    }
}
void dsu(int u , int far , int op)
{
    for(int i = head[u] ; i ; i = edge[i].nex)
    {
        int v = edge[i].to;
        if(v == far || v == hson[u]) continue ;
        dsu(v , u , 0);
    }
    if(hson[u]) dsu(hson[u] , u , 1) , HH = hson[u];
    for(int i = head[u] ; i ; i = edge[i].nex)
    {
        int v = edge[i].to;
        if(v == far || v == HH) continue ;
        calc(v , u , u) , change(v , u , 1);
    }
    HH = 0;
    sum[dep[u]] += a[u] , cnt[dep[u]] ++ ;
    if(!op)
    {
        change(u , far , -1);
    }
}
```

```
signed main()
{
    cin >> n >> k;
    rep(i , 1 , n) cin >> a[i];
    rep(i , 2 , n)
    {
        int u , v;
        cin >> u >> v;
        add_edge(u , v) , add_edge(v , u);
    }
    dfs(1 , 0);
    dsu(1 , 0 , 0);
    rep(i , 1 , n) cout << ans[i] << " \n"[i == n];
    return 0;
}
```

7.3.12 牛客练习赛 81D

题目大意

给定一棵包含 n 个节点的树，每个节点有个权值 a_i
求 $\sum_{u=1}^{n}\sum_{v=1}^{n}\min(a_u,a_v)dis(u,v)$

解题思路

对于节点 u

- 记权值小于 a_u 的节点有 $a_{x1},a_{x2},a_{x3},...,a_{xcnt1}$
- 记权值大于等于 a_u 的节点有 $a_{y1},a_{y2},...,a_{ycnt2}$

那么节点 u 对答案的贡献为：

1. $a_u\times(dep_u+dep_{x1}-2\times dep_{lca})+a_u\times(dep_u+dep_{x2}-2\times dep_{lca})+...$
2. $a_{y1}\times(dep_u+dep_{y1}-2\times dep_{lca})+a_{y2}\times(dep_u+dep_{y2}-2\times dep_{lca})+...$

即：

1. $a_u\times cnt1\times(dep_u-2\times dep_{lca})+a_u\times(deq_{(x1+...+xcnt1)})$
2. $a_{(y1+...+ycnt2)}\times(dep_u-2\times dep_{lca})+a_{(y1+...+ycnt2)}\times dep_{(y1+..+ycnt2)}$

定义 rt 为当前子树的根，那么 $lca=rt$

开四棵权值树状数组，分别用来维护 cnt、dep、a_i 、$a_i\times dep_i$

然后跑一遍 $dsu\ on\ tree$ 即可

```
const int N = 2e5 + 10 , mod = 998244353;
int n , ans , a[N] , dep[N] , sz[N] , HH , hson[N] , M;
struct Edge{
    int nex , to;
} edge[N << 1];
int head[N] , TOT;
void add_edge(int u , int v)
{
```

```
    edge[++ TOT].nex = head[u];
    edge[TOT].to = v;
    head[u] = TOT;
}
struct TR{
    int tr[N];
    int lowbit(int x){
        return x & (-x);
    }
    void add(int pos , int val)
    {
        while(pos <= M)
        {
            tr[pos] = (tr[pos] + val + mod) % mod;
            pos += lowbit(pos);
        }
    }
    int query(int pos)
    {
        int res = 0;
        while(pos)
        {
            res += tr[pos];
            res %= mod;
            pos -= lowbit(pos);
        }
        return res;
    }
    int get_sum(int L , int R){
        return (query(R) - query(L - 1) + mod) % mod;
    }
} tree1 , tree2 , tr1 , tr2;
vector<int>vec;
int get_id(int x){
    return lower_bound(vec.begin() , vec.end() , x) - vec.begin() + 1;
}
void dfs(int u , int far)
{
    dep[u] = dep[far] + 1 , sz[u] = 1;
    for(int i = head[u] ; i ; i = edge[i].nex)
    {
        int v = edge[i].to;
        if(v == far) continue ;
        dfs(v , u);
        sz[u] += sz[v];
        if(sz[v] > sz[hson[u]]) hson[u] = v;
    }
}
void change(int u , int far , int val)
{
    tree1.add(a[u] , dep[u] * val);
    tree2.add(a[u] , vec[a[u] - 1] * dep[u] * val);
    tr1.add(a[u] , val);
```

```
62      tr2.add(a[u] , val * vec[a[u] - 1]);
63      for(int i = head[u] ; i ; i = edge[i].nex)
64      {
65          int v = edge[i].to;
66          if(v == far || v == HH) continue ;
67          change(v , u , val);
68      }
69  }
70  void calc(int u , int far , int rt)
71  {
72      int cnt = tr1.get_sum(a[u] , M);
73      int sum = tree1.get_sum(a[u] , M);
74      int mi = vec[a[u] - 1];
75          ans += mi * dep[u] * cnt + mi * sum;
76          ans -= mi * cnt * 2 * dep[rt];
77          ans = (ans + mod) % mod;
78      sum = tree2.get_sum(1 , a[u] - 1);
79      cnt = tr2.get_sum(1 , a[u] - 1);
80          ans += sum + cnt * dep[u];
81          ans -= cnt * 2 * dep[rt];
82          ans = (ans + mod) % mod;
83      for(int i = head[u] ; i ; i = edge[i].nex)
84      {
85          int v = edge[i].to;
86          if(v == far || v == HH) continue ;
87          calc(v , u , rt);
88      }
89  }
90  void dsu(int u , int far , int op)
91  {
92      for(int i = head[u] ; i ; i = edge[i].nex)
93      {
94          int v = edge[i].to;
95          if(v == far || v == hson[u]) continue ;
96          dsu(v , u , 0);
97      }
98      if(hson[u]) dsu(hson[u] , u , 1) , HH = hson[u];
99      for(int i = head[u] ; i ; i = edge[i].nex)
100     {
101         int v = edge[i].to;
102         if(v == far || v == HH) continue;
103         calc(v , u , u) , change(v , u , 1);
104     }
105     int cnt = tr1.get_sum(a[u] , M);
106     int sum = tree1.get_sum(a[u] , M);
107     int mi = vec[a[u] - 1];
108         ans += mi * dep[u] * cnt + mi * sum;
109         ans -= mi * cnt * 2 * dep[u];
110         ans = (ans + mod) % mod;
111     sum = tree2.get_sum(1 , a[u] - 1);
112     cnt = tr2.get_sum(1 , a[u] - 1);
113         ans += sum + cnt * dep[u];
114         ans -= cnt * 2 * dep[u];
```

```
115         ans = (ans + mod) % mod;
116     tree1.add(a[u] , dep[u]);
117     tree2.add(a[u] , vec[a[u] - 1] * dep[u]);
118     tr1.add(a[u] , 1);
119     tr2.add(a[u] , vec[a[u] - 1]);
120     HH = 0;
121     if(!op) change(u , far , -1);
122 }
123 signed main()
124 {
125     read(n);
126     for(int i = 1 ; i <= n ; i ++) read(a[i]) , vec.push_back(a[i]);
127     for(int i = 1 ; i < n ; i ++)
128     {
129         int u , v;
130         read(u) , read(v);
131         add_edge(u , v) , add_edge(v , u);
132     }
133     sort(vec.begin() , vec.end());
134     vec.erase(unique(vec.begin() , vec.end()) , vec.end());
135     for(int i = 1 ; i <= n ; i ++) a[i] = get_id(a[i]);
136     M = vec.size();
137     dfs(1 , 0);
138     dsu(1 , 0 , 1);
139     Out(ans * 2 % mod) , puts("");
140     return 0;
141 }
```

7.3.13 CF741D

题目大意

一棵根为 1 的树，每条边上有一个字符（$a-v$ 共 22 种）

一条简单路径被称为 Dokhtar-kosh 当且仅当路径上的字符经过重新排序后可以变成一个回文串。

求每个子树中最长的 Dokhtar-kosh 路径的长度。

解题思路

dsu on tree + 状态压缩

1. 重排后构成回文的条件为:

. 每个字母出现的次数都为偶数 . 一个字母出现次数为奇数，其余字母出现次数为偶数

2. 字母的范围为 a ~ z ， 把其转换成二进制状态（偶数为 0，奇数为 1）

那么满足回文条件的二进制为：00...000 ， 00..001 ， 00..010 , ... , 10..000

3. 维护一个从根节点到子节点 u 的前缀异或和数组 X

那么 u 到 v 的简单路径的字母重排后的二进制形式为 X[u] ^ X[v]

4. 节点 u 的答案有三种可能:

. 它的两个不同分支的节点构成的简单路径 . 它的一个分支到它本身构成的简单路径 . 它的子节点的 ans

于是就可以定义 f_x 表示状态为 x 的节点的最大深度

那么当根节点为 rt 时, 子节点 u 和 rt 产生的贡献为 ↓

```
if((x[u] ^ x[rt]) == 0) ma = max(ma , dep[u] - dep[rt]);
rep(i , 0 , 21) if((x[u] ^ x[rt]) == (1LL << i)) ma = max(ma , dep[u] -
    dep[rt]);
```

子节点 u 和 rt 的其它分支的产生贡献为 ↓

```
if(f[x[u]]) ma = max(ma , f[x[u]] + dep[u] - 2 * dep[rt]);
rep(i , 0 , 21)
{
   int now = f[x[u] ^ (1LL << i)];
   if(now) ma = max(ma , dep[u] + now - 2 * dep[rt]);
}
```

rt 由它轻子儿子传递上来的贡献为 ↓

```
for(int i = head[rt] ; i ; i = edge[i].nex)
{
   int v = edge[i].to;
   if(v == far || v == hson[rt]) continue ;
   dsu(v , rt , 0);
   ans[rt] = max(ans[rt] , ans[v]);
}
```

rt 由它重儿子传递上来的贡献为 ↓

```
if(hson[rt]) dsu(hson[rt] , rt , 1) , ans[rt] = max(ans[rt] , ans[hson[rt
    ]]) , HH = hson[rt];
if(f[x[rt]]) ma = max(ma , f[x[rt]] - dep[rt]);
rep(i , 0 , 21) if(f[x[rt] ^ (1LL << i)]) ma = max(ma , f[x[rt] ^ (1LL <<
    i)] - dep[rt]);
```

因为要计算不同分支的两点产生的贡献，所以需要先对一个分支统计完贡献后，再添加它的信息

(事实上相同分支内的节点答案的在根节点为 rt 的子节点的时候就已经算过了)

```
const int N = 6e5 + 10;
struct Edge{
   int nex , to;
}edge[N << 1];
int head[N] , TOT;
void add_edge(int u , int v)
{
   edge[++ TOT].nex = head[u] ;
   edge[TOT].to = v;
   head[u] = TOT;
}
```

```cpp
int hson[N] , HH , sz[N] , dep[N] , x[N];
int n , ma , a[N] , ans[N] , f[N * 20];
void dfs(int u , int far , int now)
{
    sz[u] = 1;
    dep[u] = dep[far] + 1;
    x[u] = now ^ a[u];
    for(int i = head[u] ; i ; i = edge[i].nex)
    {
        int v = edge[i].to;
        if(v == far) continue ;
        dfs(v , u , x[u]);
        sz[u] += sz[v];
        if(sz[v] > sz[hson[u]]) hson[u] = v;
    }
}
void calc(int u , int far , int rt)
{
    if(f[x[u]]) ma = max(ma , f[x[u]] + dep[u] - 2 * dep[rt]);
    if((x[u] ^ x[rt]) == 0) ma = max(ma , dep[u] - dep[rt]);
    rep(i , 0 , 21)
    {
        if((x[u] ^ x[rt]) == (1LL << i)) ma = max(ma , dep[u] - dep[rt]);
        int now = f[x[u] ^ (1LL << i)];
        if(now) ma = max(ma , dep[u] + now - 2 * dep[rt]);
    }
    for(int i = head[u] ; i ; i = edge[i].nex)
    {
        int v = edge[i].to;
        if(v == far || v == HH) continue ;
        calc(v , u , rt);
    }
}
void change(int u , int far , int val)
{
    if(val == 1) f[x[u]] = max(dep[u] , f[x[u]]);
    else f[x[u]] = 0;
    for(int i = head[u] ; i ; i = edge[i].nex)
    {
        int v = edge[i].to;
        if(v == far || v == HH) continue ;
        change(v , u , val);
    }
}
void dsu(int u , int far , int op)
{
    for(int i = head[u] ; i ; i = edge[i].nex)
    {
        int v = edge[i].to;
        if(v == far || v == hson[u]) continue ;
        dsu(v , u , 0);
        ans[u] = max(ans[u] , ans[v]);
    }
```

```
65      if(hson[u]) dsu(hson[u] , u , 1) , HH = hson[u] , ans[u] = max(ans[u] , ans[hson
            [u]]);
66      if(f[x[u]]) ma = max(ma , f[x[u]] - dep[u]);
67      rep(i , 0 , 21)
68      {
69          if(f[x[u] ^ (1LL << i)]) ma = max(ma , f[x[u] ^ (1LL << i)] - dep[u]);
70      }
71      for(int i = head[u] ; i ; i = edge[i].nex)
72      {
73          int v = edge[i].to;
74          if(v == far || v == hson[u]) continue ;
75          calc(v , u , u);
76          change(v , u , 1);
77      }
78      ans[u] = max(ans[u] , ma);
79      f[x[u]] = max(f[x[u]] , dep[u]);
80      HH = 0;
81      if(!op)
82      {
83          ma = 0;
84          change(u , far , -1);
85      }
86  }
87  signed main()
88  {
89      cin >> n;
90      rep(i , 2 , n)
91      {
92          int x ; char op;
93          cin >> x >> op;
94          add_edge(i , x) , add_edge(x , i);
95          a[i] = (1LL << (int)(op - 'a'));
96      }
97      dfs(1 , 0 , 0);
98      dsu(1 , 0 , 0);
99      rep(i , 1 , n) cout << ans[i] << " \n"[i == n];
100     return 0;
101 }
```

7.4 单调栈

```
1  // 左边第一个比 ai 大的数的位置
2  for(int i = 1 ; i <= n ; i ++)
3  {
4      while(st.size() && a[st.top()] <= a[i]) st.pop();
5      if(st.size()) l[i] = st.top();
6      else l[i] = -1;
7      st.push(i);
8  }
9  while(!st.empty()) st.pop();
10 // 右边第一个比 ai 大的数的位置
11 for(int i = n ; i >= 1 ; i --)
```

```
{
    while(st.size() && a[st.top()] <= a[i]) st.pop();
    if(st.size()) r[i] = st.top();
    else r[i] = -1;
    st.push(i);
}
```

7.5 单调队列

7.5.1 理想正方形

- 有一个 $a \times b$ 的整数组成的矩阵，现请你从中找出一个 $n \times n$ 的正方形区域，使得该区域所有数中的最大值和最小值的差最小。
- 先对每行跑一遍单调队列，用 $ma[i][j]$ 记录第 i 行，$[i-n,i]$ 的最大值
  然后用 $ma[i][j]$ 对每列跑一遍单调队列即可

```
const int N = 1e3 + 10;
int A , B , n , a[N][N];
deque<int>que;
int ma[N][N] , mi[N][N];
void get_min(int mi[] , int a[] , int A)
{
    que.clear();
    for(int i = 1 ; i <= A ; i ++)
    {
        if(!que.empty() && que.front() <= i - n) que.pop_front();
        while(!que.empty() && a[que.back()] >= a[i]) que.pop_back();
        que.push_back(i);
        mi[i] = a[que.front()];
    }
}
void get_max(int ma[] , int a[] , int A)
{
    que.clear();
    for(int i = 1 ; i <= A ; i ++)
    {
        if(!que.empty() && que.front() <= i - n) que.pop_front();
        while(!que.empty() && a[que.back()] <= a[i]) que.pop_back();
        que.push_back(i);
        ma[i] = a[que.front()];
    }
}
int b[N] , c[N];
int ra[N][N] , ri[N][N];
signed main()
{
    cin >> A >> B >> n;
    for(int i = 1 ; i <= A ; i ++) for(int j = 1 ; j <= B ; j ++) cin >> a[i][j];
    for(int i = 1 ; i <= A ; i ++)
    {
        get_max(ma[i] , a[i] , B);
        get_min(mi[i] , a[i] , B);
```

```
    }
    int res = 0x3f3f3f3f;
    for(int j = n ; j <= B ; j ++)
    {
        for(int i = 1 ; i <= A ; i ++) b[i] = ma[i][j] , c[i] = mi[i][j];
        get_max(ra[j] , b , A);
        get_min(ri[j] , c , A);
        for(int i = n ; i <= A ; i ++) res = min(res , ra[j][i] - ri[j][i]);
    }
    cout << res << '\n';
    return 0;
}
```

7.6 第二分块

7.6.1 将区间内 x 改为 y、区间第 k 大

- `1 l r x y`：把区间 $[l,r]$ 中所有 x 变成 y
- `2 l r k`：查询区间 $[l,r]$ 中的 k 小值
- $1 \le n,m,a_i \le 10^5$

```
#include <bits/stdc++.h>
#define rep(i, a, b) for (int i = a; i <= b; i++)
using namespace std;
const int N = 100005, M = N - 5, S1 = 600, S2 = 280;
int sm[N / S1 + 5], tp1[N], tp2[N / S2 + 2];
int n, m, a[N], cn1[N / S1 + 5][N], cn2[N / S1 + 5][N / S2 + 5], rc[N / S1 + 5], tot
    [N / S1 + 5];
int cl[N / S1 + 5][S1 + 5], vl[N / S1 + 5][S1 + 5], wh[N / S1 + 5][N + S1], a1[S1 +
    5], a2[N], bc1, bc2;
int b1[N], b2[N], L1[N / S1 + 5], L2[N / S2 + 5], R1[N / S1 + 5], R2[N / S2 + 5], rb
    [N / S1 + 5][S1 + 5];
void init()
{
    for (int i = 1; i <= M; i++) b2[i] = (i - 1) / S2 + 1;
    for (int i = 1; i <= M + 1; i++) if (b2[i] ^ b2[i - 1])
    {
        R2[bc2] = i - 1;
        if (i ^ (M + 1)) L2[++bc2] = i;
    }
    for (int i = 1; i <= n; i++) b1[i] = (i - 1) / S1 + 1;
    for (int i = 1; i <= n + 1; i++) if (b1[i] ^ b1[i - 1])
    {
        R1[bc1] = i - 1;
        if (i ^ (M + 1)) L1[++bc1] = i;
    }
    for (int i = 1; i <= bc1; i++)
    {
        int l = R1[i] - L1[i] + 1;
        for (int j = 1; j <= M; j++) cn1[i][j] = cn1[i - 1][j];
        for (int j = 1; j <= bc2; j++) cn2[i][j] = cn2[i - 1][j];
```

```
28        for (int j = L1[i]; j <= R1[i]; j++) cn1[i][a[j]]++, cn2[i][b2[a[j]]]++;
29        for (int j = 1; j <= l; j++) cl[i][j] = a[L1[i] + j - 1], a1[j] = a2[cl[i][j
              ]] = !a2[cl[i][j]] ? ++tot[i] : a2[cl[i][j]];
30        for (int j = 1; j <= l; j++)
31           a2[cl[i][j]] = 0, vl[i][a1[j]] = cl[i][j], wh[i][cl[i][j]] = a1[j], cl[i][
                 j] = a1[j] + M, wh[i][cl[i][j]] = a1[j];
32    }
33 }
34 void modif(int id, int l, int r, int x, int y)
35 {
36    int nx, fg, rs = 0;
37    if (cn1[id][x] == cn1[id - 1][x]) return;
38    if (cn1[id][y] == cn1[id - 1][y])
39    {
40       if (rc[id]) fg = 1, nx = rb[id][rc[id]--];
41       else fg = 0, nx = ++tot[id];
42       for (int i = l; i <= r; i++) if (vl[id][wh[id][cl[id][i]]] == x) cl[id][i] =
              nx + M, rs++;
43       if (!rs)
44          return fg ? rc[id]++ : tot[id]--, void();
45       else
46          wh[id][y] = wh[id][nx + M] = nx, vl[id][nx] = y;
47    }
48    else
49       for (int i = l; i <= r; i++)
50          if (vl[id][wh[id][cl[id][i]]] == x)
51             cl[id][i] = wh[id][y] + M, rs++;
52    if (rs == cn1[id][x] - cn1[id - 1][x]) rb[id][++rc[id]] = wh[id][x], vl[id][wh[
          id][x]] = 0, wh[id][wh[id][x] + M] = 0, wh[id][x] = 0;
53    sm[id] = rs;
54 }
55 void chang(int id, int x, int y)
56 {
57    if (cn1[id][x] == cn1[id - 1][x]) return;
58    if (cn1[id][y] == cn1[id - 1][y]) return wh[id][y] = wh[id][x], vl[id][wh[id][x
          ]] = y, wh[id][x] = 0, void();
59    int l = R1[id] - L1[id] + 1;
60    for (int i = 1; i <= l; i++) if (vl[id][wh[id][cl[id][i]]] == x) cl[id][i] = wh[
          id][y] + M;
61    rb[id][++rc[id]] = wh[id][x], vl[id][wh[id][x]] = 0, wh[id][wh[id][x] + M] = 0,
          wh[id][x] = 0;
62 }
63 void updat(int l, int r, int x, int y)
64 {
65    if (x == y) return;
66    else memset(sm, 0, sizeof(sm));
67    int L = b1[l], R = b1[r], X = b2[x], Y = b2[y];
68    if (L ^ R)
69    {
70       modif(L, l - L1[L] + 1, R1[L] - L1[L] + 1, x, y), modif(R, 1, r - L1[R] + 1,
              x, y), L++, R--;
71       for (int i = L; i <= R; i++) chang(i, x, y), sm[i] = cn1[i][x] - cn1[i - 1][x
              ];
```

```
72      }
73      else modif(L, l - L1[L] + 1, r - L1[L] + 1, x, y);
74      for (int i = L - 1; i <= bc1; i++)
75          sm[i] += sm[i - 1], cn1[i][x] -= sm[i], cn1[i][y] += sm[i], cn2[i][X] -= sm[i
                ], cn2[i][Y] += sm[i];
76  }
77  int query(int l, int r, int x)
78  {
79      memset(tp2, 0, sizeof(tp2));
80      int L = b1[l], R = b1[r];
81      if (L == R)
82      {
83          for (int i = l, y; i <= r; i++) y = vl[L][wh[L][cl[L][i - L1[L] + 1]]], tp1[y
                ]++, tp2[b2[y]]++;
84          for (int i = 1; i <= bc2; x -= tp2[i], i++) if (x - tp2[i] <= 0)
85          {
86              for (int j = L2[i];; x -= tp1[j], j++) if (x - tp1[j] <= 0)
87              {
88                  for (int k = l; k <= r; k++) tp1[vl[L][wh[L][cl[L][k - L1[L] + 1]]]] =
                        0;
89                  return j;
90              }
91          }
92      }
93      for (int i = l, y; i <= R1[L]; i++) y = vl[L][wh[L][cl[L][i - L1[L] + 1]]], tp1[
            y]++, tp2[b2[y]]++;
94      for (int i = L1[R], y; i <= r; i++) y = vl[R][wh[R][cl[R][i - L1[R] + 1]]], tp1[
            y]++, tp2[b2[y]]++;
95      for (int i = 1; i <= bc2; i++) tp2[i] += cn2[R - 1][i] - cn2[L][i];
96      for (int i = 1; i <= bc2; x -= tp2[i], i++) if (x - tp2[i] <= 0)
97      {
98          for (int j = L2[i]; j <= R2[i]; j++) tp1[j] += cn1[R - 1][j] - cn1[L][j];
99          for (int j = L2[i];; x -= tp1[j], j++) if (x - tp1[j] <= 0)
100         {
101             for (int k = L2[i]; k <= R2[i]; k++) tp1[k] = 0;
102             for (int k = l; k <= R1[L]; k++) tp1[vl[L][wh[L][cl[L][k - L1[L] + 1]]]] =
                    0;
103             for (int k = L1[R]; k <= r; k++) tp1[vl[R][wh[R][cl[R][k - L1[R] + 1]]]] =
                    0;
104             return j;
105         }
106     }
107     return -1; // 出错
108 }
109 signed main()
110 {
111     cin >> n >> m;
112     rep(i, 1, n) cin >> a[i];
113     init();
114     while (m--)
115     {
116         int op, l, r, x, y;
117         cin >> op >> l >> r >> x;
```

```
118         if (op == 1)
119         {
120             cin >> y;
121             updat(l, r, x, y);
122         }
123         else
124         {
125             cout << query(l, r, x) << '\n';
126         }
127     }
128     return 0;
129 }
```

7.6.2 区间大于 x 的数减去 x、区间内 x 出现的次数

- `1 l r x`：把区间 $[l,r]$ 所有大于 x 的数减去 x
- `2 l r x`：查询区间 $[l,r]$ 内的 x 的出现次
- $1 \le l \le r \le n \leqslant 100000, 1 \le a_i, x \le 100000$（弱化版，可在线）
- $1 \le n \le 10^6$，$1 \le m \le 5 \times 10^5$（强化版，因为空间不够需要离线）

```
1  //在线
2  #include <bits/stdc++.h>
3  #define il inline
4  #define SZ 333    //块的大小
5  #define L(a) ((a)*SZ - SZ + 1) //某个块左端点
6  #define R(a) ((a)*SZ)  //某个块右端点
7  using namespace std;
8  struct node
9  {
10     int rt, nm;
11 } g[405][110005]; //每个块内值域结构体，rt是值域对应的祖先，num是这个值出现次数
12 int n, Q, bl[110005], far[110005], v[110005], a[110005], mx[110005], lz[110005];
13 //bl是分块中的某个节点属于哪个块，a是初始的数组值，v是一个并查集节点对应的权值
14 //far是并查集的祖先数组，mx是区间最大值，lz是懒标记
15 il int find(int x)
16 {
17     return x == far[x] ? x : far[x] = find(far[x]);
18 }
19 //并查集找祖先，自带路径压缩（
20 il void push(int x) //下压一个标记
21 {
22     for (int i = L(x), e = R(x); i <= e; ++i) a[i] = v[find(i)], g[x][a[i]].rt = g[x
           ][a[i]].nm = 0, a[i] -= lz[x];
23     //遍历这个块中的所有元素，把这个块的东西塞回a中
24     lz[x] = 0;
25     for (int i = L(x), e = R(x); i <= e; ++i) far[i] = 0; //把这个区间清空
26 }
27 il void init(int x) //初始化一个块
28 {
29     mx[x] = 0;
```

```
30      for (int i = L(x), e = R(x); i <= e; ++i)
31      {
32          mx[x] = max(mx[x], a[i]), ++g[x][a[i]].nm; //记录区间最大值以及某个值域位置元素
                个数
33          if (g[x][a[i]].rt) far[i] = g[x][a[i]].rt;
34          else v[i] = a[i], g[x][a[i]].rt = far[i] = i;
35          //如果这个点已经有值域节点存在了，那就直接把当前节点合并到先前节点上去
36          //否则新建一个值域节点
37      }
38  }
39  il void chng(int x, int a, int b) //把值域为a的位置和值域为b的合并
40  {
41      node &s = g[x][a], &t = g[x][b]; //找到两个节点
42      if (t.rt) far[s.rt] = t.rt;
43      else t.rt = s.rt, v[s.rt] = b;
44      t.nm += s.nm, s.nm = s.rt = 0; //直接暴力合并，并把s给删除
45      //如果b不存在就直接向a贺就好了，直接把a指向b
46  }
47  il void atag(int x, int ad) //打标记
48  {
49      if (ad <= mx[x] - lz[x] - ad)
50      {
51          for (int i = lz[x] + 1; i <= lz[x] + ad; ++i) if (g[x][i].rt) chng(x, i, i +
                ad);
52          lz[x] += ad;
53      }
54      else
55      {
56          for (int i = mx[x]; i > lz[x] + ad; i--) if (g[x][i].rt) chng(x, i, i - ad);
57          mx[x] = min(mx[x], lz[x] + ad);
58      }
59      //刚刚说的分两种情况讨论，应该还挺好懂的吧，不解释了
60  }
61  il void chang(int l, int r, int x) //区间修改操作
62  {
63      int p = bl[l], q = bl[r];
64      push(p);
65      if (p ^ q) push(q); //都先下推tag
66      if (p ^ q)
67      {
68          for (int i = l, e = R(p); i <= e; ++i) if (a[i] > x) a[i] -= x;
69          for (int i = L(q); i <= r; ++i)  if (a[i] > x) a[i] -= x; //暴力的边块
70          for (int i = p + 1; i <= q - 1; ++i) atag(i, x); //整块打标记
71          init(p), init(q); //再记录回去块的信息
72      }
73      else
74      {
75          for (int i = l; i <= r; ++i) if (a[i] > x) a[i] -= x;
76          init(p);
77      }
78      //否则直接完全的暴力!
79  }
80  il int query(int l, int r, int x) //区间查询操作
```

```
{
    //只需要对于每个块来查询值域上所对应的位置出现次数即可
    int p = bl[l], q = bl[r], res = 0; //分块基本套路，和chang差不多
    if (p ^ q)
    {
        for (int i = l, e = R(p); i <= e; ++i) if (v[find(i)] - lz[p] == x) ++res;
        for (int i = L(q); i <= r; ++i) if (v[find(i)] - lz[q] == x) ++res;
        for (int i = p + 1; i <= q - 1; i++) if (x + lz[i] <= 100000) res += g[i][x +
            lz[i]].nm; //防止越界
    }
    else for (int i = l; i <= r; ++i) if (v[find(i)] - lz[p] == x) ++res;
    return res;
}
signed main()
{
    cin >> n >> Q;
    for (int i = 1; i <= n; i++) cin >> a[i], bl[i] = (i - 1) / SZ + 1;
    for (int i = bl[1]; i <= bl[n]; i++) init(i), lz[i] = 0;
    while (Q--)
    {
        int op, l, r, x;
        cin >> op >> l >> r >> x;
        if (op == 1)
        chang(l, r, x);
        else
        cout << query(l, r, x) << '\n';
    }
    return 0;
}
```

```
// 离线
#include <bits/stdc++.h>
#define rep(i, a, b) for (int i = a; i <= b; i++)
#define il inline
using namespace std;
char buf[1 << 22], *p1 = buf, *p2 = buf, obuf[1 << 22], *O = obuf;
int read()
{
    char c = getchar();
    int z = 0, f = 1;
    while (c != '-' && (c > '9' || c < '0'))
    c = getchar();
    if (c == '-')
    f = -1, c = getchar();
    while (c >= '0' && c <= '9')
    z = (z << 1) + (z << 3) + c - '0', c = getchar();
    return z * f;
}
const int N = 1e6 + 5, M = 1205, V = 1e5 + 5;
int tag, MV, n, m, fa[N], a[N], to[V], klen, blocks, L[M], R[M], rt[V], siz[N], ans
    [500005];
il int find(int x)
{
```

```
23      return x == fa[x] ? x : fa[x] = find(fa[x]);
24  }
25  il void merge(int x, int y)
26  {
27      if (rt[y]) fa[rt[x]] = rt[y];
28      else rt[y] = rt[x], to[rt[y]] = y;
29      siz[y] += siz[x];
30      rt[x] = siz[x] = 0;
31  }
32  il void build(int o)
33  {
34      MV = tag = 0;
35      rep(i, L[o], R[o])
36      {
37          MV = max(MV , a[i]);
38          if (rt[a[i]]) fa[i] = rt[a[i]];
39          else rt[a[i]] = fa[i] = i, to[i] = a[i];
40          ++ siz[a[i]];
41      }
42  }
43  il void modify(int x)
44  {
45      if ((x << 1) <= MV - tag)
46      {
47          rep(i, tag + 1, tag + x)
48          if (rt[i]) merge(i, i + x);
49          tag += x;
50      }
51      else
52      {
53          rep(i, tag + x + 1, MV)
54          if (rt[i]) merge(i, i - x);
55          if (MV - tag > x) MV = x + tag;
56      }
57  }
58  il void restruct(int x, int l, int r, int nx)
59  {
60      rep(i, L[x], R[x])
61      a[i] = to[find(i)],
62      rt[a[i]] = siz[a[i]] = 0, a[i] -= tag;
63      rep(i, L[x], R[x])
64      to[i] = 0;
65      l = max(l, L[x]), r = min(r, R[x]);
66      rep(i, l, r)
67      if (a[i] > nx) a[i] -= nx;
68      build(x);
69  }
70  struct que
71  {
72      int op, l, r, x;
73  } q[500005];
74  il void ask(int x, int i)
75  {
```

```
76      int l = q[i].l, r = q[i].r, nx = q[i].x;
77      if (nx + tag > 5e5)
78      return;
79      if (l <= L[x] && R[x] <= r)
80      ans[i] += siz[nx + tag];
81      else
82      {
83          l = max(l , L[x]);
84          r = min(r , R[x]);
85          rep(j, l, r) if(to[find(j)] - tag == nx) ++ ans[i];
86      }
87  }
88  signed main()
89  {
90      n = read() , m = read();
91      klen = sqrt(n);
92      rep(i, 1, n) a[i] = read();
93      rep(i, 1, m)
94      {
95
96          q[i] = (que){read(), read(), read(), read()};
97      }
98      blocks = (n - 1) / klen + 1;
99      rep(i, 1, blocks) L[i] = R[i - 1] + 1,
100     R[i] = i * klen , R[blocks] = n;
101     rep(i, 1, blocks)
102     {
103         memset(rt, 0, sizeof(rt)) , memset(siz, 0, sizeof(siz));
104         build(i);
105         rep(j, 1, m)
106         {
107             if (L[i] > q[j].r || R[i] < q[j].l) continue;
108             if (q[j].op == 1)
109             {
110                 if (q[j].l <= L[i] && R[i] <= q[j].r) modify(q[j].x);
111                 else restruct(i, q[j].l, q[j].r, q[j].x);
112             }
113             else ask(i, j);
114         }
115     }
116     rep(i, 1, m) if (q[i].op == 2) cout << ans[i] << '\n';
117     return 0;
118 }
```

7.7 树状数组

7.7.1 二维树状数组

- $change(a,b,c,d,x)$，将 $(a,b),(c,d)$ 为顶点的矩形区域内的所有数字加上 x
- $query(a,b,c,d)$，将 $(a,b),(c,d)$ 为顶点的矩形区域内所有数字的和

```
1   const int N = 2e3 + 60;
```

```cpp
int tree1[N][N], tree2[N][N], tree3[N][N], tree4[N][N];
int lowbit(int x){
    return x & (-x);
}
int n , m;
void update(int x, int y, int d)
{
    for(int i = x; i <= n; i += lowbit(i)) for(int j = y; j <= m; j += lowbit(j))
    {
        tree1[i][j] += d;
        tree2[i][j] += x * d;
        tree3[i][j] += y * d;
        tree4[i][j] += x * y * d;
    }
}
int sum(int x, int y)
{
    int ans = 0;
    for(int i = x; i > 0; i -= lowbit(i)) for(int j = y; j > 0; j -= lowbit(j))
    {
        ans += (x + 1) * (y + 1) * tree1[i][j] - (y + 1) * tree2[i][j] - (x + 1) *
            tree3[i][j] + tree4[i][j];
    }
    return ans;
}
void change(int a , int b , int c , int d , int x)
{
    update(a, b, x) , update(c + 1, d + 1, x);
    update(a, d + 1,-x) , update(c + 1, b, -x);
}
int query(int a , int b , int c , int d)
{
    return sum(c, d)+sum(a-1, b-1)-sum(a-1, d)-sum(c, b-1);
}
signed main()
{
    cin >> n >> m ;
    char ch[3];
    while(cin >> ch)
    {
        int a , b , c , d , x;
        cin >> a >> b >> c >> d;
        if(ch[0] == 'L')
        {
            cin >> x;
            change(a , b , c , d , x);
        }
        else cout << query(a , b , c , d) << '\n';
    }
    return 0;
}
```

7.8 线段树

7.8.1 总结

区间离散化

常规的区间离散化即将一个大区间中的信息”折叠”，然后保留左右端点，并将左右端点的值离散化

比如：

区间 $[5,1000]$，将 $[6,999]$ 中的信息折叠，再离散化 $<5,1000>\rightarrow<1,2>$
那么最后 $[1,2]$ 即为离散化后的 $[5,1000]$

但当涉及到多个区间时这么做会出现一些问题，比如：
有两个区间分别为：$[1,2]$,$[5,6]$，离散化后它们分别为：$[1,2]$,$[3,4]$
而原本 $[1,2]$,$[5,6]$ 并不是连续区间，但离散化后 $[1,2]$,$[3,4]$ 就成了连续区间，区间之间的关系发生改变，那么离散化就是错误的，我们不能忽略两区间之间的区间

权值线段树区间离散化

再补充一下上面的问题：

- 普通权值线段树的叶子节点存储的是一个**数**的信息
- 而权值线段树区间离散化的叶子节点存储的是一个**折叠区间**的信息
- 对于一棵线段树，它的叶子节点对应的数需要连续。
  因为线段树规定两个叶子节点的父节点对应的区间是两叶子节点之间的所有数
- 所以权值线段树区间离散化的叶子节点对应的信息也必须连续
  这样才能保证父节点覆盖了子节点之间所有的信息

所以如果按照普通的离散方式建立线段树，有些存在的区间就会被忽略

于是我们可以按照左闭右开的方式离散化：
对于区间 $[1,2]$,$[5,6]$，先将其改为左闭右开的形式 $\rightarrow[1,3)$,$[5,7)$
离散化之后对应的区间就是 $[1,2)$,$[3,4)$
再还原回去 $\rightarrow[1,1]$,$[3,3]$，这样就不会造成区间的丢失

时间分治：通常搭配可撤销并查集使用（例题写在可撤销并查集那块了）

7.8.2 01 序列、区间覆盖为 0、区间覆盖为 1、区间取反、区间 1 的个数、区间最大连续 1 的个数

- `0 l r` 把 $[l,r]$ 区间内的所有数全变成 0
- `1 l r` 把 $[l,r]$ 区间内的所有数全变成 1
- `2 l r` 把 $[l,r]$ 区间内的所有数全部取反，也就是说把所有的 0 变成 1，把所有的 1 变成 0
- `3 l r` 询问 $[l,r]$ 区间内总共有多少个 1
- `4 l r` 询问 $[l,r]$ 区间内最多有多少个连续的 1

```
#include <bits/stdc++.h>
#define ll long long
using namespace std;
const int MAX_N = 2e5 + 50;

int n, m, a[MAX_N];
```

```
7
8  struct segment_tree
9  {
10     int l, r;
11     int sum, lazy, rev, pre[2], suf[2], dat[2];
12     #define ls(rt) (rt << 1)
13     #define rs(rt) (rt << 1 | 1)
14 } t[MAX_N << 2];
15 void push_up(int rt)
16 {
17     t[rt].sum = t[ls(rt)].sum + t[rs(rt)].sum;
18     for (int i = 0; i <= 1; i ++ )
19     {
20         t[rt].pre[i] = t[ls(rt)].pre[i];
21         if (i == 1 && t[ls(rt)].sum == (t[ls(rt)].r - t[ls(rt)].l + 1))
22         t[rt].pre[i] += t[rs(rt)].pre[i];
23         else if (i == 0 && t[ls(rt)].sum == 0)
24         t[rt].pre[i] += t[rs(rt)].pre[i];
25
26         t[rt].suf[i] = t[rs(rt)].suf[i];
27         if (i == 1 && t[rs(rt)].sum == (t[rs(rt)].r - t[rs(rt)].l + 1))
28         t[rt].suf[i] += t[ls(rt)].suf[i];
29         else if (i == 0 && t[rs(rt)].sum == 0)
30         t[rt].suf[i] += t[ls(rt)].suf[i];
31
32         t[rt].dat[i] = t[ls(rt)].suf[i] + t[rs(rt)].pre[i];
33         t[rt].dat[i] = max(t[rt].dat[i], t[ls(rt)].dat[i]);
34         t[rt].dat[i] = max(t[rt].dat[i], t[rs(rt)].dat[i]);
35         t[rt].dat[i] = max(t[rt].dat[i], t[rt].pre[i]);
36         t[rt].dat[i] = max(t[rt].dat[i], t[rt].suf[i]);
37     }
38 }
39 void push_down(int rt)
40 {
41     if (t[rt].lazy != -1)
42     {
43         t[rt].rev = 0;
44         int v = t[rt].lazy;
45
46         t[ls(rt)].sum = (t[ls(rt)].r - t[ls(rt)].l + 1) * v;
47         t[rs(rt)].sum = (t[rs(rt)].r - t[rs(rt)].l + 1) * v;
48
49         t[ls(rt)].lazy = t[rs(rt)].lazy = v;
50         t[ls(rt)].rev = t[rs(rt)].rev = 0;
51
52         t[ls(rt)].pre[v] = t[ls(rt)].suf[v] = t[ls(rt)].dat[v] = (t[ls(rt)].r - t[ls(
               rt)].l + 1);
53         t[ls(rt)].pre[v ^ 1] = t[ls(rt)].suf[v ^ 1] = t[ls(rt)].dat[v ^ 1] = 0;
54
55         t[rs(rt)].pre[v] = t[rs(rt)].suf[v] = t[rs(rt)].dat[v] = (t[rs(rt)].r - t[rs(
               rt)].l + 1);
56         t[rs(rt)].pre[v ^ 1] = t[rs(rt)].suf[v ^ 1] = t[rs(rt)].dat[v ^ 1] = 0;
57
```

```
58          t[rt].lazy = -1;
59      }
60      if (t[rt].rev)
61      {
62          t[ls(rt)].sum = (t[ls(rt)].r - t[ls(rt)].l + 1) - t[ls(rt)].sum;
63          t[rs(rt)].sum = (t[rs(rt)].r - t[rs(rt)].l + 1) - t[rs(rt)].sum;
64          if (t[ls(rt)].lazy != -1) t[ls(rt)].lazy ^= 1;
65          else t[ls(rt)].rev ^= 1;
66          if (t[rs(rt)].lazy != -1) t[rs(rt)].lazy ^= 1;
67          else t[rs(rt)].rev ^= 1;
68
69          swap(t[ls(rt)].dat[0], t[ls(rt)].dat[1]);
70          swap(t[ls(rt)].suf[0], t[ls(rt)].suf[1]);
71          swap(t[ls(rt)].pre[0], t[ls(rt)].pre[1]);
72
73          swap(t[rs(rt)].dat[0], t[rs(rt)].dat[1]);
74          swap(t[rs(rt)].suf[0], t[rs(rt)].suf[1]);
75          swap(t[rs(rt)].pre[0], t[rs(rt)].pre[1]);
76
77          t[rt].rev = 0;
78      }
79      return;
80  }
81  void build (int rt, int l, int r)
82  {
83      t[rt].l = l , t[rt].r = r , t[rt].lazy = -1;
84      if (l == r)
85      {
86          t[rt].sum = a[l];
87          t[rt].pre[a[l]] = t[rt].suf[a[l]] = t[rt].dat[a[l]] = 1;
88          return;
89      }
90      int mid = (l + r) >> 1;
91      build(ls(rt), l, mid);
92      build(rs(rt), mid + 1, r);
93      push_up(rt);
94  }
95  void update_range(int rt, int l, int r, int v)
96  {
97      push_down(rt);
98      if (l <= t[rt].l && t[rt].r <= r)
99      {
100         if (v == 0 || v == 1)
101         {
102             t[rt].sum = (t[rt].r - t[rt].l + 1) * v;
103             t[rt].lazy = v;
104             t[rt].pre[v] = t[rt].suf[v] = t[rt].dat[v] = (t[rt].r - t[rt].l + 1);
105             t[rt].pre[v ^ 1] = t[rt].suf[v ^ 1] = t[rt].dat[v ^ 1] = 0;
106         }
107         else if (v == 2)
108         {
109             t[rt].sum = (t[rt].r - t[rt].l + 1) - t[rt].sum;
110             t[rt].rev ^= 1;
```

```
111             swap(t[rt].pre[0], t[rt].pre[1]);
112             swap(t[rt].suf[0], t[rt].suf[1]);
113             swap(t[rt].dat[0], t[rt].dat[1]);
114         }
115         return;
116     }
117     int mid = (t[rt].l + t[rt].r) >> 1;
118     if (l <= mid) update_range(ls(rt), l, r, v);
119     if (r > mid) update_range(rs(rt), l, r, v);
120     push_up(rt);
121 }
122 ll query_sum(int rt, int l, int r)
123 {
124     if (l <= t[rt].l && t[rt].r <= r) return t[rt].sum;
125     push_down(rt);
126     int mid = (t[rt].l + t[rt].r) >> 1;
127     int ans = 0;
128     if (l <= mid) ans += query_sum(ls(rt), l, r);
129     if (r > mid) ans += query_sum(rs(rt), l, r);
130     return ans;
131 }
132 segment_tree query_len(int rt, int l, int r)
133 {
134     if (l <= t[rt].l && t[rt].r <= r) return t[rt];
135     push_down(rt);
136     int mid = (t[rt].l + t[rt].r) >> 1;
137     if (r <= mid) return query_len(ls(rt), l, r);
138     if (l > mid) return query_len(rs(rt), l, r);
139     segment_tree t1 = query_len(ls(rt), l, r), t2 = query_len(rs(rt), l, r), t3;
140     t3.sum = t1.sum + t2.sum;
141     for (int i = 0; i <= 1; i ++ )
142     {
143         t3.pre[i] = t1.pre[i];
144         if (i == 1 && t1.sum == (t1.r - t1.l + 1))
145         t3.pre[i] += t2.pre[i];
146         else if (i == 0 && t1.sum == 0)
147         t3.pre[i] += t2.pre[i];
148 
149         t3.suf[i] = t2.suf[i];
150         if (i == 1 && t2.sum == (t2.r - t2.l + 1))
151         t3.suf[i] += t1.suf[i];
152         else if (i == 0 && t2.sum == 0)
153         t3.suf[i] += t1.suf[i];
154 
155         t3.dat[i] = t1.suf[i] + t2.pre[i];
156         t3.dat[i] = max(t3.dat[i], t1.dat[i]);
157         t3.dat[i] = max(t3.dat[i], t2.dat[i]);
158         t3.dat[i] = max(t3.dat[i], t3.pre[i]);
159         t3.dat[i] = max(t3.dat[i], t3.suf[i]);
160     }
161     return t3;
162 }
163 signed main()
```

```
{
    cin >> n >> m;
    for (int i = 1; i <= n; i ++ ) cin >> a[i];
    build(1, 1, n);
    for (int i = 1; i <= m; i ++ )
    {
        int op , l , r;
        cin >> op >> l >> r;
        l ++ , r ++ ; // 本题下标是从 0 开始的
        if (op == 0) update_range(1, l, r, 0);
        if (op == 1) update_range(1, l, r, 1);
        if (op == 2) update_range(1, l, r, 2);
        if (op == 3) cout << query_sum(1, l, r) << '\n';
        if (op == 4) cout << query_len(1, l, r).dat[1] << '\n';
    }
    return 0;
}
```

7.8.3 单点修改、区间重排是否在值域上连续

- 修改 x 位置的值为 y
- 查询区间 $[l,r]$ 是否可以重排为值域上连续的一段

维护序列 $2^{a_1}, 2^{a_2}, ..., 2^{a_n}$ 的和
若 $a_x, a_{x+1}, ...a_u$ 重排后若在值域上连续，则 $\sum_{i=x}^{y} 2^i = \sum_{i=Min}^{Max} 2^i = 2^{Max+1} - 2^{Min}$

```
#define ull unsigned long long
#define ll long long
#define int long long
using namespace std;
const ll INF (0x3f3f3f3f3f3f3f3fll);
const int N = 5e5 + 10;
const int M = 5e5 + 10;
int a[M] , n , m;
ull power[N] , base = 233 , tot[N];
ull pow_mod(ll x , ll n){ull res = 1 ; while(n) {if(n & 1) res = res * x ; x = x * x
    ; n >>= 1 ;} return res;}
struct Seg_Tree{
    int l , r , mi;
    ull sum;
}tree[M << 2];
void push_up(int rt)
{
    tree[rt].mi = min(tree[rt << 1].mi , tree[rt << 1 | 1].mi);
    tree[rt].sum = tree[rt << 1].sum + tree[rt << 1 | 1].sum;
}
void build(int l , int r , int rt , int *aa)
{
    tree[rt].l = l , tree[rt].r = r;
    if(l == r)
    {
        tree[rt].mi = aa[l];
        tree[rt].sum = pow_mod(base , aa[l]);
```

```
27          return ;
28      }
29      int mid = l + r >> 1;
30      build(l , mid , rt << 1 , aa);
31      build(mid + 1 , r , rt << 1 | 1 , aa);
32      push_up(rt);
33  }
34  void update(int pos , int val , int rt)
35  {
36      int l = tree[rt].l , r = tree[rt].r;
37      if(l == r)
38      {
39          tree[rt].mi = val;
40          tree[rt].sum = pow_mod(base , val);
41          return ;
42      }
43      int mid = l + r >> 1;
44      if(pos <= mid) update(pos , val , rt << 1);
45      else if(pos > mid) update(pos , val , rt << 1 | 1);
46      push_up(rt);
47  }
48  int query_min(int L , int R , int rt)
49  {
50      int l = tree[rt].l , r = tree[rt].r;
51      if(L <= l && r <= R) return tree[rt].mi ;
52      int mid = l + r >> 1 , res = INF;
53      if(L <= mid) res = min(res , query_min(L , R , rt << 1));
54      if(R > mid) res = min(res , query_min(L , R , rt << 1 | 1));
55      return res;
56  }
57  ull query_sum(int L , int R , int rt)
58  {
59      int l = tree[rt].l , r = tree[rt].r;
60      if(L <= l && r <= R) return tree[rt].sum;
61      int mid = l + r >> 1;
62      ull res = 0;
63      if(L <= mid) res += query_sum(L , R , rt << 1);
64      if(R > mid) res += query_sum(L , R , rt << 1 | 1);
65      return res;
66  }
67  const ull mod = 19260817337;
68  void init()
69  {
70      power[0] = 1 , tot[0] = 1;
71      for(int i = 1 ; i <= N - 10 ; i ++) power[i] = power[i - 1] * base , tot[i] =
            tot[i - 1] + power[i];
72  }
73  signed main()
74  {
75      ios::sync_with_stdio(false);
76      cin.tie(0) , cout.tie(0);
77      init();
78      cin >> n >> m;
```

```
    for(int i = 1 ; i <= n ; i ++) cin >> a[i];
    build(1 , n , 1 , a);
    while(m --)
    {
        int op;
        cin >> op;
        if(op == 1)
        {
            int pos , val;
            cin >> pos >> val;
            update(pos , val , 1);
        }
        else
        {
            int l , r;
            cin >> l >> r;
            int mi = query_min(l , r , 1);
            ull sum = query_sum(l , r , 1);
            ull ad = pow_mod(base , mi - 1);
            if(sum == ad * (tot[r - l + 1] - 1)) cout << "damushen\n";
            else cout << "yuanxing\n";
        }
    }
    return 0;
}
```

7.8.4 单点修改、区间最大连续子段和

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long
const ll N = 5e5 + 10;
struct Segment_tree
{
    ll l , r;
    ll sum , lmax , rmax , ans;
} tree[N << 2];
ll a[N];
void push_up(ll rt)
{
    tree[rt].sum = tree[rt << 1].sum + tree[rt << 1 | 1].sum;
    tree[rt].lmax = max(tree[rt << 1].lmax , tree[rt << 1].sum + tree[rt << 1 | 1].
        lmax);
    tree[rt].rmax = max(tree[rt << 1 | 1].rmax , tree[rt << 1 | 1].sum + tree[rt <<
        1].rmax);
    tree[rt].ans = max(tree[rt << 1].rmax + tree[rt << 1 | 1].lmax , max(tree[rt <<
        1].ans , tree[rt << 1 | 1].ans));
}
void build(ll l , ll r , ll rt , ll *aa)
{
    tree[rt].l = l;
    tree[rt].r = r;
```

```
22      if(tree[rt].l == tree[rt].r)
23      {
24          tree[rt].sum = a[l];
25          tree[rt].lmax = a[l];
26          tree[rt].rmax = a[l];
27          tree[rt].ans = a[l];
28          return ;
29      }
30      ll mid = l + r >> 1;
31      build(l , mid , rt << 1 , aa);
32      build(mid + 1 , r , rt << 1 | 1 , aa);
33      push_up(rt);
34  }
35  void update(ll pos , ll val , ll rt)
36  {
37      if(tree[rt].l == tree[rt].r)
38      {
39          tree[rt].ans = tree[rt].sum = tree[rt].lmax = tree[rt].rmax = val;
40          return ;
41      }
42      ll mid = tree[rt].l + tree[rt].r >> 1;
43      if(pos <= mid)
44          update(pos , val , rt << 1);
45      if(pos > mid)
46          update(pos , val , rt << 1 | 1);
47      push_up(rt);
48  }
49  Segment_tree query_ans(ll l , ll r , ll rt)
50  {
51      if(l <= tree[rt].l && r >= tree[rt].r)
52          return tree[rt];
53      ll mid = tree[rt].l + tree[rt].r >> 1;
54      if(r <= mid)
55          return query_ans(l , r , rt << 1);
56      else if(l > mid)
57          return query_ans(l , r , rt << 1 | 1);
58      else
59      {
60          Segment_tree Ans , a , b;
61          a = query_ans(l , mid , rt << 1);
62          b = query_ans(mid + 1 , r , rt << 1 | 1);
63          Ans.sum = a.sum + b.sum;
64          Ans.lmax = max(a.lmax , a.sum + b.lmax);
65          Ans.rmax = max(b.rmax , a.rmax + b.sum);
66          Ans.ans = max(a.rmax + b.lmax , max(a.ans , b.ans));
67          return Ans;
68      }
69  }
70  int main()
71  {
72      ios::sync_with_stdio(false);
73      int n , m;
74      cin >> n >> m;
```

```
    for(int i = 1 ; i <= n ; i ++)
        cin >> a[i];
    build(1 , n , 1 , a);
    while(m --)
    {
        ll x;
        cin >> x;
        if(x == 1)
        {
            ll l , r;
            cin >> l >> r;
            if(l > r)
            swap(l , r);
            cout << query_ans(l , r , 1).ans << '\n';
        }
        else
        {
            ll pos , val;
            cin >> pos >> val;
            update(pos , val , 1);
        }
    }
    return 0;
}
```

7.8.5 动态区间中位数

给定一个空序列，有 N 次操作，每次向序列末尾插入 $L, L+1, L+2, ...R$
求每次操作结束后序列的中位数

区间离散化（左闭右开）+ 权值线段树（线段树空间要开八倍）

```
#include <bits/stdc++.h>
#define ll long long
using namespace std;
const int N = 4e5 + 10;
ll x[N], y[N];
struct Tree
{
    int l, r;
    ll sum, len, lazy;
} tree[N << 3];
int b[N << 1];
void pushup(int rt)
{
    tree[rt].sum = tree[rt << 1].sum + tree[rt << 1 | 1].sum;
}
void build(int l, int r, int rt)
{
    tree[rt].l = l;
    tree[rt].r = r;
    tree[rt].sum = tree[rt].lazy = 0;
    if(l == r)
    {
```

```
23          tree[rt].len = b[l + 1] - b[l];
24          return;
25      }
26      int mid = l + r >> 1;
27      build(l, mid, rt << 1);
28      build(mid + 1, r, rt << 1 | 1);
29      tree[rt].len = tree[rt << 1].len + tree[rt << 1 | 1].len;
30  }
31  void pushdown(int rt)
32  {
33      if(tree[rt].lazy != 0)
34      {
35          tree[rt << 1].lazy += tree[rt].lazy;
36          tree[rt << 1].sum += tree[rt].lazy * tree[rt << 1].len;
37          tree[rt << 1 | 1].lazy += tree[rt].lazy;
38          tree[rt << 1 | 1].sum += tree[rt].lazy * tree[rt << 1 | 1].len;
39          tree[rt].lazy = 0;
40      }
41  }
42  void update(int rt, int l, int r)
43  {
44      if(l <= tree[rt].l && tree[rt].r <= r)
45      {
46          tree[rt].sum += tree[rt].len ;
47          tree[rt].lazy ++;
48          return ;
49      }
50      pushdown(rt);
51      int mid = tree[rt].l + tree[rt].r >> 1;
52      if(l <= mid) update(rt << 1, l, r);
53      if(mid < r) update(rt << 1 | 1, l, r);
54      pushup(rt);
55  }
56  int query(int rt, ll K)
57  {
58      if(tree[rt].l == tree[rt].r)
59      {
60          ll tmp = tree[rt].sum / tree[rt].len;
61          return b[tree[rt].l] + (K - 1) / tmp;
62      }
63      pushdown(rt);
64      if(K <= tree[rt << 1].sum) return query(rt << 1, K);
65      else return query(rt << 1 | 1, K - tree[rt << 1].sum);
66  }
67  signed main()
68  {
69      ios::sync_with_stdio(false);
70      cin.tie(0) , cout.tie(0);
71      int n, a1, a2, b1, b2, c1, c2, m1, m2;
72      cin >> n;
73      cin >> x[1] >> x[2] >> a1 >> b1 >> c1 >> m1;
74      cin >> y[1] >> y[2] >> a2 >> b2 >> c2 >> m2;
75      for(int i = 3; i <= n; i++)
```

```
76      {
77          x[i] = (1ll * a1 * x[i - 1] + 1ll * b1 * x[i - 2] + c1) % m1;
78          y[i] = (1ll * a2 * y[i - 1] + 1ll * b2 * y[i - 2] + c2) % m2;
79          // 题目的奇葩求 [L,R] 方法
80      }
81      int len = 0;
82      for(int i = 1; i <= n; i++)
83      {
84          ll l, r;
85          l = min(x[i], y[i]) + 1;
86          r = max(x[i], y[i]) + 1;
87          x[i] = l;
88          y[i] = r;
89          b[++len] = l;
90          b[++len] = r + 1;
91      }
92      sort(b + 1, b + len + 1);
93      len = unique(b + 1, b + len + 1) - (b + 1);
94      for(int i = 1; i <= n; i++)
95      {
96          x[i] = lower_bound(b + 1, b + len + 1, x[i]) - b;
97          y[i] = lower_bound(b + 1, b + len + 1, y[i] + 1) - b;
98      }
99      build(1, len - 1, 1);
100     ll sum = 0;
101     for(int i = 1; i <= n; i++)
102     {
103         update(1, x[i], y[i] - 1);
104         sum += b[y[i]] - b[x[i]];
105         cout << query(1 , (sum + 1) / 2) << '\n';
106     }
107     return 0;
108 }
```

7.8.6 区间 +k、区间最大连续子段和

给定一个整数序列 a，支持两种操作：

- `1 l r x` 表示给区间 $[l,r]$ 中每个数加上 x
- `2 l r` 表示查询区间 $[l,r]$ 的最大子段和（可以为 0）

复杂度 $O((n+m)\log^3 n + q\log n)$

```
1  const int N = 400500;
2  const long long INF = 2e18;
3  int n, w[N], m;
4  long long lazy[N << 2];
5  struct LINE
6  { //一次函数
7      int k;
8      long long b;
9      LINE operator+(const LINE &tree) const
10     {
```

```
11          return (LINE){
12              k + tree.k, b + tree.b};
13      }
14  };
15  pair<LINE, long long> max(LINE tree, LINE b) //对两个一次函数取x=0时的最值，同时给出max
        的选取不发生变化的最大值C(即 x>C 时max变成另一个)
16  {
17      if (tree.k < b.k || (tree.k == b.k && tree.b < b.b))
18      swap(tree, b);
19      if (tree.b >= b.b)
20      return make_pair(tree, INF);
21      return make_pair(b, (b.b - tree.b) / (tree.k - b.k));
22  }
23  struct Node
24  {
25      LINE ls, rs, ans, sum;
26      long long x; //增量>x时子树内会有max的选取发生变化
27      Node operator+(const Node &tree) const
28      {
29          Node t;
30          t.x = min(x, tree.x);
31          pair<LINE, long long> tmp = max(ls, sum + tree.ls);
32          t.ls = tmp.first, t.x = min(t.x, tmp.second);
33          tmp = max(tree.rs, rs + tree.sum);
34          t.rs = tmp.first, t.x = min(t.x, tmp.second);
35          tmp = max(ans, tree.ans);
36          t.x = min(t.x, tmp.second);
37          tmp = max(tmp.first, rs + tree.ls);
38          t.ans = tmp.first, t.x = min(t.x, tmp.second);
39          t.sum = sum + tree.sum;
40          return t;
41      }
42  } tree[N << 2];
43  void build(int rt, int l, int r)
44  {
45      if (l == r)
46      {
47          LINE t = {1, w[l]};
48          tree[rt] = (Node){
49              t, t, t, t, INF};
50          return;
51      }
52      int mid = (l + r) >> 1;
53      build(rt << 1, l, mid), build(rt << 1 | 1, mid + 1, r);
54      tree[rt] = tree[rt << 1] + tree[rt << 1 | 1];
55  }
56  void upd(int rt, long long w)
57  {
58      lazy[rt] += w;
59      tree[rt].x -= w;
60      tree[rt].ls.b += tree[rt].ls.k * w;
61      tree[rt].rs.b += tree[rt].rs.k * w;
62      tree[rt].ans.b += tree[rt].ans.k * w;
```

```
63      tree[rt].sum.b += tree[rt].sum.k * w;
64  }
65  void upd(int rt, int l, int r, long long w)
66  {
67      if (w > tree[rt].x) //如果增量使得子树内有max发生变化则递归下去重构
68      {
69          long long t = lazy[rt] + w;
70          lazy[rt] = 0;
71          int mid = (l + r) >> 1;
72          upd(rt << 1, l, mid, t), upd(rt << 1 | 1, mid + 1, r, t);
73          tree[rt] = tree[rt << 1] + tree[rt << 1 | 1];
74      }
75      else upd(rt, w);
76  }
77  void pushdown(int rt)
78  {
79      if (lazy[rt])
80      {
81          long long t = lazy[rt];
82          lazy[rt] = 0;
83          upd(rt << 1, t), upd(rt << 1 | 1, t);
84      }
85  }
86  void update_range(int rt, int l, int r, int L, int R, int x)
87  {
88      if (L <= l && r <= R)
89      {
90          upd(rt, l, r, x);
91          return;
92      }
93      pushdown(rt);
94      int mid = (l + r) >> 1;
95      if (L <= mid) update_range(rt << 1, l, mid, L, R, x);
96      if (R > mid) update_range(rt << 1 | 1, mid + 1, r, L, R, x);
97      tree[rt] = tree[rt << 1] + tree[rt << 1 | 1];
98  }
99  Node query(int rt, int l, int r, int L, int R)
100 {
101     if (l >= L && r <= R) return tree[rt];
102     pushdown(rt);
103     int mid = (l + r) >> 1;
104     if (R <= mid) return query(rt << 1, l, mid, L, R);
105     if (L > mid) return query(rt << 1 | 1, mid + 1, r, L, R);
106     return query(rt << 1, l, mid, L, mid) + query(rt << 1 | 1, mid + 1, r, mid + 1,
            R);
107 }
108 signed main()
109 {
110     cin >> n >> m;
111     rep(i, 1, n) cin >> w[i];
112     build(1, 1, n);
113     while (m--)
114     {
```

```
115         int op, l, r, x;
116         cin >> op >> l >> r;
117         if (op == 1)
118         {
119             cin >> x;
120             update_range(1, 1, n, l, r, x);
121         }
122         else cout << max(0ll , query(1 , 1 , n , l , r).ans.b) << '\n';
123     }
124     return 0;
125 }
```

7.8.7 区间 +k、区间最大值、区间最小值、区间和

```
1  #include<bits/stdc++.h>
2  #define int long long
3  #define ll long long
4  using namespace std;
5  const int N = 1e5 + 10;
6  struct Tree
7  {
8      ll l,r,sum,lazy,maxn,minn;
9  } tree[N << 2];
10 void push_up(ll rt)
11 {
12     tree[rt].sum=tree[rt<<1].sum+tree[rt<<1|1].sum;
13     tree[rt].maxn=max(tree[rt<<1].maxn,tree[rt<<1|1].maxn);
14     tree[rt].minn=min(tree[rt<<1].minn,tree[rt<<1|1].minn);
15 }
16 void push_down(ll rt , ll len)
17 {
18     if(tree[rt].lazy)
19     {
20         tree[rt<<1].lazy+=tree[rt].lazy,tree[rt<<1|1].lazy+=tree[rt].lazy;
21         tree[rt<<1].sum+=(len-(len>>1))*tree[rt].lazy,tree[rt<<1|1].sum+=(len>>1)*
               tree[rt].lazy;
22         tree[rt<<1].minn+=tree[rt].lazy,tree[rt<<1|1].minn+=tree[rt].lazy;
23         tree[rt<<1].maxn+=tree[rt].lazy,tree[rt<<1|1].maxn+=tree[rt].lazy;
24         tree[rt].lazy=0;
25     }
26 }
27 void build(ll l , ll r , ll rt , ll *aa)
28 {
29     tree[rt].lazy=0;
30     tree[rt].l=l;
31     tree[rt].r=r;
32     if(l==r)
33     {
34         tree[rt].sum=aa[l];
35         tree[rt].minn=tree[rt].sum;
36         tree[rt].maxn=tree[rt].sum;
37         return;
```

```
38      }
39      ll mid=(l+r)>>1;
40      build(l,mid,rt<<1,aa);
41      build(mid+1,r,rt<<1|1,aa);
42      push_up(rt);
43  }
44  void update_range(ll L , ll R , ll key , ll rt)
45  {
46      if(tree[rt].r<L||tree[rt].l>R)return;
47      if(L<=tree[rt].l&&R>=tree[rt].r)
48      {
49          tree[rt].sum+=(tree[rt].r-tree[rt].l+1)*key;
50          tree[rt].minn+=key;
51          tree[rt].maxn+=key;
52          tree[rt].lazy+=key;
53          return;
54      }
55      push_down(rt,tree[rt].r-tree[rt].l+1);
56      ll mid=(tree[rt].r+tree[rt].l)>>1;
57      if(L<=mid)update_range(L,R,key,rt << 1);
58      if(R>mid)update_range(L,R,key,rt << 1 | 1);
59      push_up(rt);
60  }
61  ll query_range(ll L, ll R, ll rt)
62  {
63      if(L<=tree[rt].l&&R>=tree[rt].r)
64      {
65          return tree[rt].sum;
66      }
67      push_down(rt,tree[rt].r-tree[rt].l+1);
68      ll mid=(tree[rt].r+tree[rt].l)>>1;
69      ll ans=0;
70      if(L<=mid)ans+=query_range(L,R,rt << 1);
71      if(R>mid)ans+=query_range(L,R,rt << 1 | 1);
72      return ans;
73  }
74  ll query_min(ll L,ll R,ll rt)
75  {
76      if(L<=tree[rt].l&&R>=tree[rt].r)
77      {
78          return tree[rt].minn;
79      }
80      push_down(rt,tree[rt].r-tree[rt].l+1);
81      ll mid=(tree[rt].r+tree[rt].l)>>1;
82      ll ans=(0x3f3f3f3f3f3f3f3fll);
83      if(L<=mid)ans=min(ans,query_min(L,R,rt << 1));
84      if(R>mid)ans=min(ans,query_min(L,R,rt << 1 | 1));
85      return ans;
86  }
87  ll query_max(ll L, ll R, ll rt)
88  {
89      if(L<=tree[rt].l&&R>=tree[rt].r)
90      {
```

```
91          return tree[rt].maxn;
92      }
93      push_down(rt,tree[rt].r-tree[rt].l+1);
94      ll mid=(tree[rt].r+tree[rt].l)>>1;
95      ll ans=-(0x3f3f3f3f3f3f3f3fll);
96      if(L<=mid)ans=max(ans,query_max(L,R,rt << 1));
97      if(R>mid)ans=max(ans,query_max(L,R,rt << 1 | 1));
98      return ans;
99  }
100 int a[N];
101 signed main()
102 {
103     ios::sync_with_stdio(false);
104     int n , m;
105     cin >> n >> m;
106     for(int i = 1 ; i <= n ; i ++) cin >> a[i];
107     build(1 , n , 1 , a);
108     while(m --)
109     {
110         int op ;
111         cin >> op;
112         if(op == 1)
113         {
114             int l , r , k;
115             cin >> l >> r >> k;
116             update_range(l , r , k , 1);
117         }
118         else
119         {
120             int l , r;
121             cin >> l >> r;
122             cout << query_range(l , r , 1) << '\n';
123         }
124     }
125     return 0;
126 }
```

7.8.8 区间 gcd

- $gcd(a[1], a[2], a[3]...a[n]) = gcd(a[1], a[2] - a[1], a[3] - a[2], ..., a[n] - a[n-1])$
- $d[i] = a[i] - a[i-1]$
- $gcd(a[l, r]) = gcd(a[l], gcd(d[l+1], ..., d[r]))$
- 树状数组：左边就是前缀和：$a[i] = a[1]+(a[2]-a[1])+...+(a[i]-a[i-1]) = a[1]+d[2]+...+d[i]$
- 线段树：右边就是单点修改区间查询，对于一个差分序列修改的话，就只改 L 项和 $R+1$ 两项

```
1  #include<bits/stdc++.h>
2  #define ll long long
3  using namespace std;
4  const int N = 5e5 + 10;
```

```
ll gcd(ll a,ll b){
    return b ? gcd(b , a % b) : a;
}
struct Tree{
    int l, r;
    ll gcd;
} tree[N << 2];
ll n , m , a[N] , d[N] , t[N];
void push_up(ll rt)
{
    tree[rt].gcd = abs(gcd(tree[rt << 1].gcd, tree[rt << 1 | 1].gcd));
}
void build(ll l, ll r, ll rt, ll *a)
{
    tree[rt].l = l, tree[rt].r = r;
    if (l == r)
    {
        tree[rt].gcd = a[l];
        return;
    }
    ll mid = (l + r) >> 1;
    build(l, mid, rt << 1, a);
    build(mid + 1, r, rt << 1 | 1, a);
    push_up(rt);
}
void update(ll rt, ll pos, ll val)
{
    ll l = tree[rt].l, r = tree[rt].r;
    if (l == r)
    {
        tree[rt].gcd += val;
        return;
    }
    ll mid = (l + r) >> 1;
    if (pos <= mid) update(rt << 1, pos, val);
    else update(rt << 1 | 1, pos, val);
    push_up(rt);
}
ll query(ll L, ll R, ll rt)
{
    ll l = tree[rt].l, r = tree[rt].r;
    if (L <= l && r <= R) return tree[rt].gcd;
    ll mid = (l + r) >> 1 , g = 0;
    if (L <= mid) g = abs(gcd(g, query(L, R, rt << 1)));
    if (R > mid) g = abs(gcd(g, query(L, R, rt << 1 | 1)));
    return g;
}
ll lowbit(ll x){
    return x & -x;
}
void add(ll pos, ll x){
    while (pos <= N - 9) t[pos] += x, pos += lowbit(pos);
}
```

```cpp
ll ask(ll pos)
{
    ll res = 0;
    while (pos) res += t[pos], pos -= lowbit(pos);
    return res;
}
void update_range(ll l, ll r, ll k)
{
    update(1, l, k), update(1, r + 1, -k);
    add(l, k), add(r + 1, -k);
}
ll query_gcd(int l, int r)
{
    ll g = query(l + 1, r, 1);
    return abs(gcd(g, ask(l)));
}
signed main()
{
    ios::sync_with_stdio(false);
    cin >> n >> m;
    for (int i = 1; i <= n; i ++) cin >> a[i];
    for (int i = 1; i <= n + 1; i ++)
    {
        d[i] = a[i] - a[i - 1]; // 差分数组
        add(i, d[i]);
    }
    build(1, n + 1, 1, d); // 多开一个防止 r + 1 越界
    for (int i = 1; i <= m; i ++)
    {
        char op;
        cin >> op ;
        if (op == 'C')
        {
            int l, r;
            ll k;
            cin >> l >> r >> k;
            update_range(l, r, k);
        }
        else if (op == 'Q')
        {
            int l, r;
            cin >> l >> r;
            cout << query_gcd(l, r) << '\n';
        }
    }
    return 0;
}
```

7.8.9 区间 hash

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

```cpp
#define rep(i , a , b) for(int i = a ; i <= b ; i ++)
using namespace std;
const int N = 1e6 + 10;
const int mod = 998244353;// prime : 1610612741
struct Seg_Tree{
    int l , r , max , hash , lazy;
}tree[N << 2];
int a[N] , p[N] , q[N];
void push_up(int rt , int len1 , int len2)
{
    tree[rt].hash = tree[rt << 1].hash * p[len2] + tree[rt << 1 | 1].hash;
    tree[rt].hash %= mod;
}
void build(int l , int r , int rt , int *a)
{
    tree[rt].l = l , tree[rt].r = r , tree[rt].lazy = 0;
    if(l == r)
    {
        tree[rt].hash = a[l] % mod;
        return ;
    }
    int mid = l + r >> 1;
    build(l , mid , rt << 1 , a);
    build(mid + 1 , r , rt << 1 | 1 , a);
    push_up(rt , mid - l + 1 , r - mid);
}
void f(int rt , int l , int r , int k)
{
    tree[rt].lazy += k , tree[rt].lazy %= mod;
    tree[rt].hash += k * q[r - l];
    tree[rt].hash % mod;
}
void push_down(int rt , int l , int r)
{
    int x = tree[rt].lazy;
    int mid = l + r >> 1;
    f(rt << 1 , l , mid , x);
    f(rt << 1 | 1 , mid + 1 , r , x);
    tree[rt].lazy = 0;
}
void update_range(int L , int R , int rt , int val)
{
    int l = tree[rt].l , r = tree[rt].r;
    int mid = l + r >> 1;
    if(L <= l && r <= R)
    {
        tree[rt].lazy += val , tree[rt].lazy %= mod;
        tree[rt].hash += val * q[r - l];
        tree[rt].hash %= mod;
        return ;
    }
    push_down(rt , l , r);
    if(L <= mid) update_range(L , R , rt << 1 , val);
```

```
    if(R > mid) update_range(L , R , rt << 1 | 1 , val);
    push_up(rt , mid - l + 1 , r - mid);
}
int query_range(int rt , int L , int R)
{
    int l = tree[rt].l , r = tree[rt].r;
    int ans = 0;
    int mid = l + r >> 1;
    push_down(rt , l , r);
    if(L <= l && r <= R) return tree[rt].hash % mod;
    else if(R <= mid) return query_range(rt << 1 , L , R) % mod;
    else if(L > mid) return query_range(rt << 1 | 1 , L , R) % mod;
    else
    {
        int s = query_range(rt << 1 , L , R);
        int t = query_range(rt << 1 | 1 , L , R);
        ans = s * p[min(R , r) - mid] % mod + t;
        ans %= mod;
    }
    return ans % mod;
}
void init()
{
    q[0] = 1 , p[0] = 1;
    rep(i , 1 , N - 10) p[i] = p[i - 1] * 131 , p[i] %= mod , q[i] = (q[i - 1] + p[i
        ]) % mod;
}
signed main()
{
    ios::sync_with_stdio(false);
    cin.tie(0) , cout.tie(0);
    init();
    int n ;
    cin >> n;
    rep(i , 1 , n) cin >> a[i];
    build(1 , n , 1 , a);
// cout << query_range(1 , 1 , 5) << " " << query_range(1 , 6 , 10) << '\n';
// cout << query_range(1 , 1 , 3) << " " << query_range(1 , 4 , 6) << '\n';
    return 0;
}
```

7.8.10 区间乘法

- 将区间 $[L, R]$ 内每个数乘上 k
- 将区间 $[L, R]$ 内每个数加上 k
- 输出区间 $[L, R]$ 内每个数的和对 mod 取模所得的结果

```
const int N = 1e5 + 10;
int n, m, mod;
int a[N];
struct Segment_Tree
```

```
{
    ll sum, add, mul;
    int l, r;
} tree[N << 2];
void push_up(int rt)
{
    tree[rt].sum = (tree[rt << 1].sum + tree[rt << 1 | 1].sum) % mod;
}
void push_down(int rt)
{
    tree[rt << 1].sum = (tree[rt << 1].sum * tree[rt].mul + tree[rt].add * (tree[rt
        << 1].r - tree[rt << 1].l + 1)) % mod;
    tree[rt << 1 | 1].sum = (tree[rt << 1 | 1].sum * tree[rt].mul + tree[rt].add * (
        tree[rt << 1 | 1].r - tree[rt << 1 | 1].l + 1)) % mod;
    tree[rt << 1].mul = (tree[rt << 1].mul * tree[rt].mul) % mod;
    tree[rt << 1 | 1].mul = (tree[rt << 1 | 1].mul * tree[rt].mul) % mod;
    tree[rt << 1].add = (tree[rt << 1].add * tree[rt].mul + tree[rt].add) % mod;
    tree[rt << 1 | 1].add = (tree[rt << 1 | 1].add * tree[rt].mul + tree[rt].add) %
        mod;
    tree[rt].add = 0;
    tree[rt].mul = 1;
    return;
}
void build(int rt, int l, int r , int *a)
{
    tree[rt].l = l , tree[rt].r = r;
    tree[rt].mul = 1;
    if (l == r)
    {
        tree[rt].sum = a[l] % mod;
        return;
    }
    int mid = (l + r) >> 1;
    build(rt << 1, l, mid , a);
    build(rt << 1 | 1, mid + 1, r , a);
    push_up(rt);
    return;
}
void update_Mul(int rt, int L, int R, int k)
{
    if (L <= tree[rt].l && tree[rt].r <= R)
    {
        tree[rt].add = (tree[rt].add * k) % mod;
        tree[rt].mul = (tree[rt].mul * k) % mod;
        tree[rt].sum = (tree[rt].sum * k) % mod;
        return;
    }

    push_down(rt);
    int mid = (tree[rt].l + tree[rt].r) >> 1;
    if (L <= mid) update_Mul(rt << 1, L, R, k);
    if (R > mid) update_Mul(rt << 1 | 1, L, R, k);
    push_up(rt);
```

```
55
56      return;
57  }
58  void update_Add(int rt, int L, int R, int k)
59  {
60      if (L <= tree[rt].l && tree[rt].r <= R)
61      {
62          tree[rt].add = (tree[rt].add + k) % mod;
63          tree[rt].sum = (tree[rt].sum + k * (tree[rt].r - tree[rt].l + 1)) % mod;
64          return;
65      }
66
67      push_down(rt);
68      int mid = (tree[rt].l + tree[rt].r) >> 1;
69      if (L <= mid) update_Add(rt << 1, L, R, k);
70      if (R > mid) update_Add(rt << 1 | 1, L, R, k);
71      push_up(rt);
72
73      return;
74  }
75  ll query_range(int rt, int L, int R)
76  {
77      if (L <= tree[rt].l && tree[rt].r <= R) return tree[rt].sum;
78      push_down(rt);
79      ll val = 0;
80      int mid = (tree[rt].l + tree[rt].r) >> 1;
81      if (L <= mid) val = (val + query_range(rt << 1, L, R)) % mod;
82      if (R > mid) val = (val + query_range(rt << 1 | 1, L, R)) % mod;
83
84      return val;
85  }
86  int main()
87  {
88      cin >> n >> m >> mod;
89      for (int i = 1; i <= n; i++) cin >> a[i];
90      build(1, 1, n, a);
91      for (int i = 1; i <= m; i++)
92      {
93          int op , l , r;
94          cin >> op >> l >> r;
95          if(op == 1)
96          {
97              int k;
98              cin >> k;
99              update_Mul(1, l, r, k);
100         }
101         if(op == 2)
102         {
103             int k;
104             cin >> k;
105             update_Add(1, l, r, k);
106         }
107         if (op == 3) cout << query_range(1 , l , r) << '\n';
```

```
108     }
109     return 0;
110 }
```

7.8.11 区间覆盖 + 查询

- *pushdown* 的时候需要判断区间是否被打上过标记
- *lazy* 需要赋初值

```
1  const int N = 3e5 + 10;
2  const int INF = (0x3f3f3f3f3f3f3f3fll);
3  struct Tree{
4      int l , r , sum , lazy; // 注意要赋予初值
5  }tree[N << 2];
6  void push_up(int rt)
7  {
8      tree[rt].sum = tree[rt << 1].sum + tree[rt << 1 | 1].sum;
9  }
10 void push_down(int rt)
11 {
12     int &x = tree[rt].lazy , l = tree[rt].l , r = tree[rt].r;
13     if(x != -INF) // 需要判断是否被打上过标记
14     {
15         int mid = l + r >> 1;
16         tree[rt << 1].sum = x * (mid - l + 1) , tree[rt << 1 | 1].sum = (r - mid) * x
               ;
17         tree[rt << 1].lazy = tree[rt << 1 | 1].lazy = x;
18     }
19     x = -INF;
20 }
21 void build(int l , int r , int rt , int *a)
22 {
23     tree[rt].l = l , tree[rt].r = r , tree[rt].lazy = -INF;
24     if(l == r)
25     {
26         tree[rt].sum = a[l];
27         return ;
28     }
29     int mid = l + r >> 1;
30     build(l , mid , rt << 1 , a);
31     build(mid + 1 , r , rt << 1 | 1 , a);
32     push_up(rt);
33 }
34 void update_range(int L , int R , int rt , int val)
35 {
36     int l = tree[rt].l , r = tree[rt].r;
37     if(L <= l && r <= R)
38     {
39         tree[rt].sum = (r - l + 1) * val;
40         tree[rt].lazy = val;
41         return ;
42     }
```

```
43      push_down(rt);
44      int mid = l + r >> 1;
45      if(L <= mid) update_range(L , R , rt << 1 , val);
46      if(R > mid) update_range(L , R , rt << 1 | 1 , val);
47      push_up(rt);
48  }
49
50  int query_range(int L , int R , int rt)
51  {
52      int l = tree[rt].l , r = tree[rt].r;
53      if(L <= l && r <= R) return tree[rt].sum;
54      push_down(rt);
55      int mid = l + r >> 1 , ans = 0;
56      if(L <= mid) ans += query_range(L , R , rt << 1);
57      if(R > mid) ans += query_range(L , R , rt << 1 | 1);
58      return ans;
59  }
60
61  int n , m , a[N];
62
63  signed main()
64  {
65      ios::sync_with_stdio(false);
66
67      cin >> n ;
68
69      rep(i , 1 , n) cin >> a[i];
70
71      build(1 , n , 1 , a);
72
73      cin >> m;
74
75      while(m --)
76      {
77          int op , l , r , k;
78
79          cin >> op >> l >> r;
80
81          if(!op) cout << query_range(l , r , 1) << '\n';
82
83          else
84          {
85              cin >> k;
86
87              update_range(l , r , 1 , k);
88          }
89      }
90
91      return 0;
92  }
```

7.8.12 区间异或 x、区间求和

- 给定 n 个数的序列 a，m 次操作，操作有两种：
  1. 求 $\sum_{i=l}^{r} a_i$
  2. 把 $a_l \sim a_r$ 异或上 x
- $1 \le n \le 10^5$，$1 \le m \le 5 \times 10^4$，$0 \le a_i \le 10^6$，$1 \le x \le 10^6$

把每个数进行二进制拆分，然后按位组成 21 个新序列 ($2^{21} > 10^6$)
再对于每个序列建立一棵线段树维护（有点类似二维线段树

```
const int N = 3e5 + 10;
struct Tree{
   int lazy;
   int l , r , cnt;
}tree[N][25];
void push_up(int rt , int id)
{
   tree[rt][id].cnt = tree[rt << 1][id].cnt + tree[rt << 1 | 1][id].cnt;
}
void F(int rt , int id)
{
   int l = tree[rt][id].l , r = tree[rt][id].r;
   tree[rt][id].cnt = (r - l + 1) - tree[rt][id].cnt;
   tree[rt][id].lazy ^= 1;
}
void push_down(int rt , int id)
{
   int &x = tree[rt][id].lazy;
   if(x)
   {
      F(rt << 1 , id) , F(rt << 1 | 1 , id);
      x = 0;
   }
}
void build(int l , int r , int rt , int *a)
{
   rep(id , 0 , 22) tree[rt][id].l = l , tree[rt][id].r = r;
   if(l == r)
   {
      rep(id , 0 , 22) tree[rt][id].cnt = a[l] >> id & 1;
      return ;
   }
   int mid = l + r >> 1;
   build(l , mid , rt << 1 , a) , build(mid + 1 , r , rt << 1 | 1 , a);
   rep(id , 0 , 22) push_up(rt , id);
}
void update_range(int L , int R , int rt , int x)
{
   int l = tree[rt][0].l , r = tree[rt][0].r;
   if(L <= l && r <= R)
   {
```

```
        rep(id , 0 , 22)
        {
            int num = x >> id & 1;
            if(!num) continue ;
            tree[rt][id].cnt = (r - l + 1) - tree[rt][id].cnt;
            tree[rt][id].lazy ^= 1;
        }
        return ;
    }
    rep(id , 0 , 22) push_down(rt , id);
    int mid = l + r >> 1;
    if(L <= mid) update_range(L , R , rt << 1 , x);
    if(R > mid) update_range(L , R , rt << 1 | 1 , x);
    rep(id , 0 , 22) push_up(rt , id);
}
int query_range(int L , int R , int rt)
{
    int l = tree[rt][0].l , r = tree[rt][0].r;
    int sum = 0;
    if(L <= l && r <= R)
    {
        rep(id , 0 , 22) sum += (1ll << id) * tree[rt][id].cnt;
        return sum;
    }
    int mid = l + r >> 1;
    rep(id , 0 , 22) push_down(rt , id);
    if(L <= mid) sum += query_range(L , R , rt << 1);
    if(R > mid) sum += query_range(L , R , rt << 1 | 1);
    return sum;
}
int n , m , a[N];
signed main()
{
    cin >> n;
    rep(i , 1 , n) cin >> a[i];
    build(1 , n , 1 , a);
    cin >> m;
    while(m --)
    {
        int op , l , r , x;
        cin >> op >> l >> r;
        if(op == 1) cout << query_range(l , r , 1) << '\n';
        else
        {
            cin >> x;
            update_range(l , r , 1 , x);
        }
    }
    return 0;
}
```

7.8.13 区间与、区间或、区间最小值

- `1 l r x`：将 $A_l, ..., A_r$ 中每个元素二进制与上一个数
- `2 l r x`：将 $A_l, ..., A_r$ 中每个元素二进制或上一个数
- `3 l r`：求 $A_l, ..., A_r$ 中的最小值
- $1 \le n, m \le 5 \times 10^5$，$0 \le A_i, x_i \le 10^9$

```
#include <bits/stdc++.h>
using namespace std;
const int N = 5e5 + 10 , ALL = (1ll << 31) - 1;
int n, a[N];
struct Node
{
    int l , r;
    int mi, v0, v1;
} tree[N << 2];
inline void push_up(int rt)
{
    tree[rt].v1 = tree[rt << 1].v1 & tree[rt << 1 | 1].v1;
    tree[rt].v0 = tree[rt << 1].v0 & tree[rt << 1 | 1].v0;
    tree[rt].mi = min(tree[rt << 1].mi, tree[rt << 1 | 1].mi);
}
void build(int rt, int l, int r)
{
    tree[rt].l = l , tree[rt].r = r;
    if (l == r)
    {
        tree[rt].mi = tree[rt].v1 = a[l], tree[rt].v0 = ALL ^ a[l];
        return;
    }
    int mid = l + r >> 1;
    build(rt << 1 , l , mid);
    build(rt << 1 | 1 , mid + 1, r);
    push_up(rt);
}
void push_down(int rt)
{
    if((tree[rt].v0 | tree[rt].v1) == ALL)
    {
        tree[rt << 1].mi = tree[rt << 1 | 1].mi = tree[rt].mi;
        tree[rt << 1].v0 = tree[rt << 1 | 1].v0 = tree[rt].v0;
        tree[rt << 1].v1 = tree[rt << 1 | 1].v1 = tree[rt].v1;
    }
}
void query_Or(int rt, int L, int R, int v)
{
    v &= ALL ^ tree[rt].v1;
    if(!v) return;
    int l = tree[rt].l , r = tree[rt].r;
    if(L <= l && r <= R)
    {
```

```
45          if ((v | tree[rt].v1 | tree[rt].v0) == ALL)
46          {
47              tree[rt].v1 |= v, tree[rt].v0 = ALL ^ tree[rt].v1, tree[rt].mi = tree[rt].
                    v1;
48              return;
49          }
50          push_down(rt);
51          query_Or(rt << 1 , L , R , v);
52          query_Or(rt << 1 | 1 , L , R , v);
53          push_up(rt);
54          return;
55      }
56      push_down(rt);
57      int mid = l + r >> 1;
58      if(L <= mid) query_Or(rt << 1 , L , R , v);
59      if(R > mid) query_Or(rt << 1 | 1 , L , R , v);
60      push_up(rt);
61  }
62  void query_And(int rt, int L, int R, int v)
63  {
64      v &= ALL ^ tree[rt].v0;
65      if(!v) return;
66      int l = tree[rt].l , r = tree[rt].r;
67      if(L <= l && r <= R)
68      {
69          if ((v | tree[rt].v1 | tree[rt].v0) == ALL)
70          {
71              tree[rt].v0 |= v, tree[rt].v1 = ALL ^ tree[rt].v0, tree[rt].mi = tree[rt].
                    v1;
72              return;
73          }
74          push_down(rt);
75          query_And(rt << 1 , L , R , v);
76          query_And(rt << 1 | 1 , L , R , v);
77          push_up(rt);
78          return;
79      }
80      push_down(rt);
81      int mid = l + r >> 1;
82      if(L <= mid) query_And(rt << 1 , L , R , v);
83      if(R > mid) query_And(rt << 1 | 1 , L , R , v);
84      push_up(rt);
85  }
86  int query_min(int rt, int L, int R)
87  {
88      int l = tree[rt].l , r = tree[rt].r;
89      if(L <= l && r <= R || ALL == (tree[rt].v0 | tree[rt].v1)) return tree[rt].mi;
90      int mid = l + r >> 1 , res = ALL;
91      if(L <= mid) res = min(res , query_min(rt << 1, L , R));
92      if(R > mid) res = min(res , query_min(rt << 1 | 1, L , R));
93      return res;
94  }
95  signed main()
```

```
{
    int n , m;
    cin >> n >> m;
    for (int i = 1 ; i <= n ; i ++) cin >> a[i];
    build(1 , 1 , n);
    while(m --)
    {
        int op , L , R , x;
        cin >> op >> L >> R;
        if(op == 1)
        {
            cin >> x;
            query_And(1, L, R, (ALL ^ x));
        }
        if(op == 2)
        {
            cin >> x;
            query_Or(1, L, R, x);
        }
        if(op == 3) cout << query_min(1 , L , R) << '\n';
    }
    return 0;
}
```

7.8.14 树上区间 +k、区间 ×k、区间覆盖、区间立方和

- 将 $u->v$ 路径上的点全部赋值为 w
- 将 $u->v$ 路径上的点全部加上 w
- 将 $u->v$ 路径上的点全部乘上 w
- $u->v$ 路径上的点的立方和

```
const int N = 1e5+5;
typedef long long LL;
const int p = 1e9 + 7;
int tot, num;
int n, m, r,t,cases=0;

int w[N], a[N], sum1[N * 4], sum2[N * 4], sum3[N * 4], lazy_mul[N * 4], lazy_add[N * 4]; // w[i]=j表示时间戳为i的点的值为j, a[]输入每个节点的值, dat线段树每个点权值, lazy线段树每个点的懒标记
//int h[N], e[N * 2], ne[N * 2], idx; // 邻接表数组
int dep[N], dfn[N], wson[N], sz[N], top[N], fa[N]; // dep深度 dfn搜索序 wson重儿子 size子树大小 top链头 fa父节点
vector<int> mp[N];

// 得到sz, fa, dep, wson数组
void dfs1(int u) {
    dep[u] = dep[fa[u]]+1;
    sz[u] = 1;
    for(int i = 0; i<mp[u].size(); i++) {
```

```
17        int j=mp[u][i];
18        if(j == fa[u]) continue;
19        fa[j] = u;
20        dfs1(j);
21        sz[u] += sz[j];
22        if(sz[j] > sz[wson[u]]) wson[u] = j; // 这里要注意根节点不能设为0，否则根节点的最
              重链无法更新，始终为0
23    }
24 }
25 // 得到dfn, top数组
26 void dfs2(int u, int nowtop) {
27    dfn[u] = ++num;
28    w[num] = a[u];
29    //以搜索序重排权值
30    top[u] = nowtop;
31    if(wson[u]) dfs2(wson[u], nowtop); // 先搜索重儿子
32    for(int i = 0; i<mp[u].size(); i++) {// 然后搜索轻儿子
33        int y=mp[u][i];
34        if(y ==fa[u]||y == wson[u]) continue;
35        dfs2(y, y);
36    }
37 }
38 void solve(int rt,int len,int a,int b){ //a为add b为mul
39    // 修改乘法和加法标记
40    lazy_mul[rt] = 1ll*lazy_mul[rt] * b % p;
41    lazy_add[rt] = 1ll*lazy_add[rt] * b % p;
42    lazy_add[rt] = ((lazy_add[rt] + a) % p + p) % p;
43    // 先维护乘法标记：x^3=>(bx)^3, x^2=>(bx)^2
44    if(b!=1){ //先乘后加
45        sum1[rt] = 1ll*sum1[rt] * b % p;
46        sum2[rt] = (1ll*sum2[rt] * b % p) * b % p;
47        sum3[rt] = ((1ll*sum3[rt] * b % p) * b % p) * b % p;
48    }
49    // 再维护加法标记：(x+a)^3=x^3+3x^2*a+3xa^2+a^3, (x+a)^2=x^2+2xa+a^2
50    if(a!=0){
51        int a2 = 1ll*a * a % p, a3 = 1ll*a2 * a % p;
52        sum3[rt] = ((sum3[rt] + (LL)len * a3 % p) + p) % p;
53        sum3[rt] = ((sum3[rt] + 3ll * (LL)sum2[rt] % p * a % p) + p) % p;
54        sum3[rt] = ((sum3[rt] + 3ll * (LL)sum1[rt] % p * a2 % p) + p) % p;
55        sum2[rt] = ((sum2[rt] + 2ll * (LL)sum1[rt] % p * a % p) + p) % p;
56        sum2[rt] = ((sum2[rt] + (LL)len * a2 % p) + p) % p;
57        sum1[rt] = ((sum1[rt] + (LL)len * a % p) + p) % p;
58    }
59 }
60 void pushup(int rt) {
61    sum1[rt] = (sum1[rt << 1] + sum1[rt << 1 | 1]) % p;
62    sum2[rt] = (sum2[rt << 1] + sum2[rt << 1 | 1]) % p;
63    sum3[rt] = (sum3[rt << 1] + sum3[rt << 1 | 1]) % p;
64 }
65 // 建线段树，rt为根，l为rt点管辖的左边界， r为rt点管辖的有边界
66 void build(int rt, int l, int r) {
67    lazy_add[rt] = 0;
68    lazy_mul[rt] = 1;
```

```
69      if(l==r) {
70          int temp = w[l];
71          sum1[rt] = temp;
72          sum2[rt] = (1ll*sum1[rt] * sum1[rt]) % p;
73          sum3[rt] = (1ll*sum1[rt] * sum2[rt]) % p;
74          return ;
75      }
76      int mid=(l + r)>>1;
77      build(rt << 1, l, mid);
78      build(rt << 1 | 1, mid+1, r);
79      pushup(rt);
80  }
81
82  // 下传
83  void pushdown(int rt, int l, int r) {
84      int mid = (l + r) >> 1;
85      solve(rt << 1, mid - l + 1, lazy_add[rt], lazy_mul[rt]);
86      solve(rt << 1 | 1, r - mid, lazy_add[rt], lazy_mul[rt]);
87      lazy_add[rt] = 0;
88      lazy_mul[rt] = 1;
89  }
90  // rt为根，l为rt点管辖的左边界， r为rt点管辖的有边界， L为需要修改的左区间，R为需要修改的右
        区间
91  void modify(int rt, int l, int r, int L, int R, int a,int b) {
92      if(L <= l && r <= R) {
93          solve(rt, r - l + 1, a, b);
94          return ;
95      }
96      pushdown(rt, l, r);
97      int mid = (l + r)>>1;
98      if(L <= mid) modify(rt << 1, l, mid, L, R, a,b);
99      if(mid < R) modify(rt << 1 | 1, mid + 1, r, L, R, a,b);
100     pushup(rt);
101 }
102 // rt为根，l为rt点管辖的左边界， r为rt点管辖的有边界， L为需要查询的左区间，R为查询的右区间
103 int query(int rt, int l, int r, int L, int R) {
104     if(L <= l && r <= R) {
105         return sum3[rt];
106     }
107     pushdown(rt, l, r);
108     int mid = (l + r)>>1;
109     int ans = 0;
110     if(L <= mid) ans += query(rt << 1, l, mid, L, R), ans %= p;
111     if(mid < R) ans += query(rt << 1 | 1, mid + 1, r, L, R), ans %= p;
112     pushup(rt);
113     return ans;
114 }
115 // 求树从 x 到 y 结点最短路径上所有节点的值之和
116 int path_query_Tree(int x, int y) {
117     //两点间的修改
118     int ans = 0;
119     while(top[x] != top[y]) {// 把x点和y点整到一条重链上
120         if(dep[top[x]] < dep[top[y]]) swap(x, y); // 让x点为深的那个点
```

```
121         ans += query(1, 1, n, dfn[top[x]], dfn[x]);
122         ans %= p;
123         x = fa[top[x]]; // x每次跳一条链
124     }
125     if(dep[x] > dep[y]) swap(x, y); // 让x成为深度更浅的那个点
126     ans += query(1, 1, n, dfn[x], dfn[y]);
127     return ans % p;
128 }
129 // 树上修改 a为add ,b为mul
130 void path_modify_Tree(int x, int y, int a,int b) {
131     //树上两点距离
132     while(top[x] != top[y]) { // 把x点和y点整到一条重链上
133         if(dep[top[x]] < dep[top[y]]) swap(x, y); // 让x成为对应的头部深度更大的那个点
134         modify(1, 1, n, dfn[top[x]], dfn[x], a,b); // 累加x的所有子树和
135         x = fa[top[x]]; // x跳到原来x的头部的父节点
136     }
137     if(dep[x] > dep[y]) swap(x, y); // 让x成为深度更浅的那个点
138     modify(1, 1, n, dfn[x], dfn[y], a,b);
139 }
140 int main() {
141     scanf("%d", &t);
142     while(t--){
143         scanf("%d", &n);
144         cases++;
145         // 读入边，建树
146         for(int i=1; i<=n; ++i) mp[i].clear(),wson[i]=0;
147         //memset(h, -1, sizeof h);
148         num = 0;
149         for(int i=1, x, y; i<n; i++) {
150             scanf("%d%d", &x, &y);
151             mp[x].push_back(y);
152             mp[y].push_back(x);
153         }
154         for(int i=1; i<=n; i++) scanf("%d", &a[i]); // 读入每个点的权值
155         // 两次dfs把树按照重链剖分
156         dfs1(1); // 得到sz, fa, dep, wson数组
157         dfs2(1, 1); // 得到dfn, top数组
158         build(1, 1, n);
159         scanf("%d", &m);
160         printf("Case #%d:\n", cases);
161         // m次询问
162         for(int i=1, op, x, y, z; i<=m; i++) {
163             scanf("%d", &op);
164             if(op == 1) {
165                 scanf("%d%d%d", &x, &y, &z);
166                 path_modify_Tree(x, y, z,0); // 将树从 x到 y 结点最短路径上所有节点的值赋
                        值为z
167             }
168             else if(op == 2) {
169                 scanf("%d%d%d", &x, &y, &z);
170                 path_modify_Tree(x, y, z,1); // 将树从 x到 y 结点最短路径上所有节点的值加
                        上z
171             }
```

```
172             else if(op == 3) {
173                 scanf("%d%d%d", &x, &y, &z);
174                 path_modify_Tree(x, y, 0,z); // 将树从 x到 y 结点最短路径上所有节点的值乘
                        上z
175             }
176             else {
177                 scanf("%d%d", &x,&y);
178                 printf("%d\n", path_query_Tree(x,y));
179             }
180         }
181     }
182     return 0;
183 }
```

7.8.15 矩阵 +k、矩阵最大值

- a b k x y 将矩阵 $(x, y) \sim (x + a - 1, , y, y + b - 1)$ 的值加上 k
- 询问矩阵最大值

```
1  const int N = 2e3 + 50;
2  int n , m , Q;
3  int max(int x , int y) { return x > y ? x : y; }
4  struct segy
5  {
6      int mx[N], tag[N];
7      void change(int rt, int l, int r, int L, int R, int val)
8      {
9          mx[rt] = max(mx[rt], val);
10         if (l == L && r == R)
11         {
12             tag[rt] = max(tag[rt], val);
13             return;
14         }
15         int Mid = L + R >> 1;
16         if (r <= Mid)
17             change(rt << 1, l, r, L, Mid, val);
18         else if (l > Mid)
19             change(rt << 1 | 1, l, r, Mid + 1, R, val);
20         else
21             change(rt << 1, l, Mid, L, Mid, val), change(rt << 1 | 1, Mid + 1, r, Mid
                   + 1, R, val);
22     }
23     int query(int rt, int l, int r, int L, int R)
24     {
25         if (l == L && r == R)
26             return mx[rt];
27         int ans = tag[rt], Mid = L + R >> 1;
28         if (r <= Mid)
29             ans = max(ans, query(rt << 1, l, r, L, Mid));
30         else if (l > Mid)
31             ans = max(ans, query(rt << 1 | 1, l, r, Mid + 1, R));
32         else
```

```
33          ans = max(ans, max(query(rt << 1, l, Mid, L, Mid), query(rt << 1 | 1, Mid
                + 1, r, Mid + 1, R)));
34      return ans;
35   }
36};
37struct segx
38{
39   segy mx[N], tag[N];
40   void change(int rt, int l, int r, int L, int R, int ll, int rr, int val)
41   {
42      mx[rt].change(1, ll, rr, 1, m, val);
43      if (l == L && r == R)
44      {
45         tag[rt].change(1, ll, rr, 1, m, val);
46         return;
47      }
48      int Mid = L + R >> 1;
49      if (r <= Mid)
50         change(rt << 1, l, r, L, Mid, ll, rr, val);
51      else if (l > Mid)
52         change(rt << 1 | 1, l, r, Mid + 1, R, ll, rr, val);
53      else
54         change(rt << 1, l, Mid, L, Mid, ll, rr, val), change(rt << 1 | 1, Mid + 1,
               r, Mid + 1, R, ll, rr, val);
55   }
56   int query(int rt, int l, int r, int L, int R, int ll, int rr)
57   {
58      if (l == L && r == R)
59         return mx[rt].query(1, ll, rr, 1, m);
60      int ans = tag[rt].query(1, ll, rr, 1, m), Mid = L + R >> 1;
61      if (r <= Mid)
62         ans = max(ans, query(rt << 1, l, r, L, Mid, ll, rr));
63      else if (l > Mid)
64         ans = max(ans, query(rt << 1 | 1, l, r, Mid + 1, R, ll, rr));
65      else
66         ans = max(ans, max(query(rt << 1, l, Mid, L, Mid, ll, rr), query(rt << 1 |
               1, Mid + 1, r, Mid + 1, R, ll, rr)));
67      return ans;
68   }
69} t;
70signed main()
71{
72   ios::sync_with_stdio(false);
73   cin.tie(0) , cout.tie(0);;
74   cin >> n >> m >> Q;
75   while (Q --)
76   {
77      int a, b, k, x, y;
78      cin >> a >> b >> k >> x >> y;
79      ++x, ++y;
80      int r = t.query(1, x, x + a - 1, 1, n, y, y + b - 1) + k;
81      t.change(1, x, x + a - 1, 1, n, y, y + b - 1, r);
82   }
```

```
    cout << t.query(1, 1, n, 1, n, 1, m) << '\n';
    return 0;
}
```

7.9 吉司机线段树

7.9.1 区间 +k、区间覆盖、区间最大值、区间历史最大值

- 给出一个长度为 n 的序列，有四种操作。
- $Q\ X\ Y$：询问区间 $[x,y]$ 的最大值
- $A\ X\ Y$：询问区间 $[x,y]$ 的历史最大值
- $P\ X\ Y\ Z$：将区间 $[x,y]$ 中的每个数增加 z
- $C\ X\ Y\ Z$：将区间 $[x,y]$ 中的每个数都变成 z

```
#include<bits/stdc++.h>
#define int long long
#define rep(i , a , b) for(int i = a ; i <= b ; i ++)
using namespace std;
const int N = 1e5 + 10;
const int inf = 0x3f3f3f3f3f3f3f3fll;
int n, m , a[N];
struct Tree
{
    int l, r, ma, hma; // hma 历史最大值
    int add, hadd, se, hse; //add 加的值 , se 赋的值
    //had 和 hse分别代表 ad和 se 的历史最大值。
} tree[N << 2];
void push_up(int rt)
{
    tree[rt].ma = max(tree[rt << 1].ma, tree[rt << 1 | 1].ma);
    tree[rt].hma = max(tree[rt << 1].hma, tree[rt << 1 | 1].hma);
}
void push_down(int rt)
{
    for(int i = 0 ; i <= 1 ; i ++) // 枚举左右儿子
    {
        int now = rt << 1 | i;
        tree[now].hma = max(tree[now].hma, tree[now].ma + tree[rt].hadd);
        if(tree[now].hse != -inf) tree[now].hse = max(tree[now].hse, tree[now].se +
            tree[rt].hadd);
        else tree[now].hadd = max(tree[now].hadd, tree[rt].hadd + tree[now].add);
        tree[now].hma = max(tree[now].hma, tree[rt].hse);
        tree[now].hse = max(tree[now].hse, tree[rt].hse);
        if(tree[rt].add != 0)
        {
            tree[now].ma += tree[rt].add , tree[now].hma = max(tree[now].hma, tree[now
                ].ma);
            if (tree[now].se == -inf)
            {
                tree[now].add += tree[rt].add;
```

```
35              tree[now].hadd = max(tree[now].hadd, tree[now].add);
36          }
37          else
38          {
39              tree[now].se += tree[rt].add;
40              tree[now].hse = max(tree[now].hse, tree[now].se);
41          }
42      }
43      else if(tree[rt].se != -inf)
44      {
45          tree[now].ma = tree[rt].se;
46          tree[now].hma = max(tree[now].hma, tree[now].ma);
47          tree[now].se = tree[rt].se;
48          tree[now].hse = max(tree[rt].se, tree[now].hse);
49          tree[now].add = 0;
50      }
51  }
52  tree[rt].se = tree[rt].hse = -inf;
53  tree[rt].add = tree[rt].hadd = 0;
54  }
55  void build(int rt, int l, int r , int *a)
56  {
57      tree[rt].l = l , tree[rt].r = r , tree[rt].se = tree[rt].hse = -inf;
58      if(l == r)
59      {
60          tree[rt].ma = a[l];
61          tree[rt].hma = tree[rt].ma;
62          tree[rt].add = tree[rt].hadd = 0;
63          tree[rt].se = tree[rt].hse = -inf;
64          return;
65      }
66      int mid = (l + r) >> 1;
67      build(rt << 1, l, mid , a);
68      build(rt << 1 | 1, mid + 1, r , a);
69      push_up(rt);
70  }
71  void add(int rt, int l, int r, int x)
72  {
73      if(tree[rt].l >= l && tree[rt].r <= r)
74      {
75          tree[rt].ma += x;
76          tree[rt].hma = max(tree[rt].hma, tree[rt].ma);
77          if(tree[rt].se == -inf)
78          {
79              tree[rt].add += x;
80              tree[rt].hadd = max(tree[rt].hadd, tree[rt].add);
81          }
82          else
83          {
84              tree[rt].se += x;
85              tree[rt].hse = max(tree[rt].hse, tree[rt].se);
86          }
87          return;
```

```
88      }
89      push_down(rt);
90      int mid = (tree[rt].l + tree[rt].r) >> 1;
91      if(l <= mid) add(rt << 1, l, r, x);
92      if(r > mid) add(rt << 1 | 1, l, r, x);
93      push_up(rt);
94  }
95  void change(int rt, int l, int r, int x)
96  {
97      if(tree[rt].l >= l && tree[rt].r <= r)
98      {
99          tree[rt].ma = tree[rt].se = x;
100         tree[rt].hma = max(tree[rt].hma, tree[rt].ma);
101         tree[rt].hse = max(x, tree[rt].hse);
102         tree[rt].add = 0;
103         return;
104     }
105     push_down(rt);
106     int mid = (tree[rt].l + tree[rt].r) >> 1;
107     if(l <= mid) change(rt << 1, l, r, x);
108     if(r > mid) change(rt << 1 | 1, l, r, x);
109     push_up(rt);
110 }
111 int query_max(int rt, int l, int r)
112 {
113     if(tree[rt].l >= l && tree[rt].r <= r) return tree[rt].ma;
114     push_down(rt);
115     int mid = (tree[rt].l + tree[rt].r) >> 1;
116     if (r <= mid) return query_max(rt << 1, l, r);
117     else if (l > mid) return query_max(rt << 1 | 1, l, r);
118     else return max(query_max(rt << 1, l, mid), query_max(rt << 1 | 1, mid + 1, r));
119 }
120 int query_hmax(int rt, int l, int r)
121 {
122     if(tree[rt].l >= l && tree[rt].r <= r) return tree[rt].hma;
123     push_down(rt);
124     int mid = (tree[rt].l + tree[rt].r) >> 1;
125     if(r <= mid) return query_hmax(rt << 1, l, r);
126     else if(l > mid) return query_hmax(rt << 1 | 1, l, r);
127     else return max(query_hmax(rt << 1, l, mid), query_hmax(rt << 1 | 1, mid + 1, r)
        );
128 }
129 signed main()
130 {
131     ios::sync_with_stdio(false);
132     cin >> n;
133     rep(i , 1 , n) cin >> a[i];
134     build(1, 1, n , a);
135     cin >> m;
136     rep(i , 1 , m)
137     {
138         int x, y, z;
139         char op;
```

```
        cin >> op;
        if(op == 'A')
        {
            cin >> x >> y;
            cout << query_hmax(1 , x , y) << '\n';
        }
        else if(op == 'C')
        {
            cin >> x >> y >> z;
            change(1, x, y, z);
        }
        else if(op == 'P')
        {
            cin >> x >> y >> z;
            add(1, x, y, z);
        }
        else
        {
            cin >> x >> y;
            cout << query_max(1 , x , y) << '\n';
        }
    }
    return 0;
}
```

7.10 主席树

7.10.1 撤销 X 个操作 (可撤销先前撤销的操作)

数据结构/主席树/撤销 X 个操作 (可撤销先前撤销的操作).tex

7.10.2 区间出现次数大于某值的最小数

查询区间内出现次数大于 (r - l + 1) / k 的最小数

```
const int N = 5e5 + 10;
struct Tree{
    int l , r , sum;
}tree[N * 40];
int root[N] , Cnt;
void update(int l , int r , int pre , int &now , int pos)
{
    tree[++ Cnt] = tree[pre];
    tree[Cnt].sum ++ ;
    now = Cnt;
    if(l == r) return ;
    int mid = l + r >> 1;
    if(pos <= mid) update(l , mid , tree[pre].l , tree[now].l , pos);
    else update(mid + 1 , r , tree[pre].r , tree[now].r , pos);
}
int query(int l , int r , int L , int R , int cnt)
{
    if(l == r) return l;
    int Lsum = tree[tree[R].l].sum - tree[tree[L].l].sum;
```

```
    int Rsum = tree[tree[R].r].sum - tree[tree[L].r].sum;
    int ans = 1e18 , mid = l + r >> 1;
    if(Lsum > cnt) ans = min(ans , query(l , mid , tree[L].l , tree[R].l , cnt));
    if(Rsum > cnt) ans = min(ans , query(mid + 1 , r , tree[L].r , tree[R].r , cnt))
        ;
    return ans;
}
int a[N];
vector<int>vec;
int get_id(int x)
{
    return lower_bound(all(vec) , x) - vec.begin() + 1;
}
signed main()
{
    ios::sync_with_stdio(false);
    cin.tie(0) , cout.tie(0);
    int n , m;
    cin >> n >> m;
    rep(i , 1 , n) cin >> a[i] , vec.pb(a[i]);
    sort(all(vec));
    vec.erase(unique(all(vec)) , vec.end());
    int nn = vec.size();
    rep(i , 1 , n) a[i] = get_id(a[i]) , update(1 , nn , root[i - 1] , root[i] , a[i
        ]);
    while(m --)
    {
        int l , r , k;
        cin >> l >> r >> k;
        int res = query(1 , nn , root[l - 1] , root[r] , (r - l + 1) / k);
        if(res == 1e18) res = -1;
        else res = vec[res - 1];
        cout << res << '\n';
    }
    return 0;
}
```

7.10.3 区间第 K 大

```
const int N = 3e5 + 10;
struct Chairman_Tree
{
    int l , r , sum;
} tree[N * 40];
vector<int>vec;
int root[N] , a[N] , Ctot;
void update(int l , int r , int pre , int &now , int pos)
{
    tree[++ Ctot] = tree[pre];
    tree[Ctot].sum ++ ;
    now = Ctot;
    if(l == r) return ;
```

```
    int mid = l + r >> 1;
    if(pos <= mid) update(l , mid , tree[pre].l , tree[now].l , pos);
    else update(mid + 1 , r , tree[pre].r , tree[now].r , pos);
}
int query(int l , int r , int L , int R , int k)
{
    if(l == r) return l ;
    int mid = l + r >> 1;
    int cha = tree[tree[R].l].sum - tree[tree[L].l].sum;
    if(cha >= k) return query(l , mid , tree[L].l , tree[R].l , k);
    else return query(mid + 1 , r , tree[L].r , tree[R].r , k - cha);
}
int get_id(int x)
{
    return lower_bound(vec.begin() , vec.end() , x) - vec.begin() + 1;
}
void init()
{
    vec.clear();
    rep(i , 0 , Ctot) tree[i].sum = tree[i].l = tree[i].r = 0;
    Ctot = 0;
}
signed main()
{
    ios::sync_with_stdio(false);
    int T = 1;
    // cin >> T;
    while(T --)
    {
        init();
        int n , m ;
        cin >> n >> m;
        rep(i , 1 , n)
        {
            cin >> a[i];
            vec.push_back(a[i]);
        }
        sort(vec.begin() , vec.end());
        vec.erase(unique(vec.begin() , vec.end()) , vec.end()) ;
        rep(i , 1 , n)
        update(1 , n , root[i - 1] , root[i] , get_id(a[i]));
        while(m --)
        {
            int l , r , k;
            cin >> l >> r >> k ;
            int pos = query(1 , n , root[l - 1] , root[r] , k);
            cout << vec[pos - 1] << '\n' ;
        }
    }
    return 0;
}
```

7.10.4 区间内不同数的个数

使用主席树在线做，我们不能使用权值线段树建主席树

- 与之前主席树的权值线段树思路不同，此题的思路是建立 n 颗线段树，第 i 颗线段树存储区间 $[1,i]$ 的信息
- 其中每个节点维护 sum，表示节点对应区间中数的个数，因此每棵线段树中只保留每个数最后出现的位置
- 举个例子，序列为 $5,5,5,5,5$，则第 1 颗线段树中只有第一个位置 sum 为 1，然后第二颗线段树从第一颗线段树继承过来，由于 5 这个数字之前出现过，因此在第二颗线段树中令第一个位置的 sum 为 0，令第二个位置的 sum 为 1，来保存每个数字最后出现的位置
- 因此查询区间 $[l,r]$ 时，就在第 r 颗线段树中查询区间 $[l,n]$ 中数的个数即可

```
const int N = 1e6 + 10;
struct Chairman_Tree{
    int l , r , sum;
} tree[N * 40];
vector<int>vec;
int root[N] , a[N] , Ctot , last[N]; //last[i]表示上一次出现 i 的位置
int n , m;
void update(int l , int r , int pre , int &now , int pos , int val)
{
    tree[++ Ctot] = tree[pre];
    tree[Ctot].sum += val;
    now = Ctot;
    if(l == r) return ;
    int mid = l + r >> 1;
    if(pos <= mid) update(l , mid , tree[pre].l , tree[now].l , pos , val);
    else update(mid + 1 , r , tree[pre].r , tree[now].r , pos , val);
}
int query(int l , int r , int L , int id)
{
    if(L <= l) return tree[id].sum;
    int mid = l + r >> 1;
    if(L <= mid) return query(l , mid , L , tree[id].l) + tree[tree[id].r].sum;
    return query(mid + 1 , r , L , tree[id].r);
}
int get_id(int x){
    return lower_bound(vec.begin() , vec.end() , x) - vec.begin() + 1;
}
signed main()
{
    read(n);
    rep(i , 1 , n)
    {
        read(a[i]);
        vec.push_back(a[i]);
    }
    sort(vec.begin() , vec.end());
    vec.erase(unique(vec.begin() , vec.end()) , vec.end()) ;
    rep(i , 1 , n)
```

```
    {
        if(last[a[i]])
        {
            int help = 0;
            update(1 , n , root[i - 1] , help , last[a[i]] , -1);
            /*
            先用 help 来转移 root[i - 1] , 在把 help 树上的 last[a[i]] --
            再把 help 转移给 root[i] , 同时让 root[i] 树上的 i ++
            */
            update(1 , n , help , root[i] , i , 1);
        }
        else update(1 , n , root[i - 1] , root[i] , i , 1);
        last[a[i]] = i;
    }
    read(m);
    while(m --)
    {
        int l , r;
        read(l) , read(r);
        cout << query(1 , n , l , root[r]) << '\n';
    }
    return 0;
}
```

7.10.5 区间内只出现过一次的数

- 给定一个长度为 N 序列，M 个询问，每次询问给定一个区间 $[L, R]$
- 如果这个区间里存在只出现一次的数，输出这个数（如果有多个就输出任意一个）
- 没有就输出 0 ， $N, M <= 5 \times 10^5$

用 $pre[a[i]]$ 记录 $a[i]$ 上一次出现的位置，并把它插入到当前版本线段树 $a[i]$ 的位置
如果区间内最小的 $pre[a[i]] < L$，则 $a[i]$ 为答案
对于每个 $a[i]$ ，如果 $pre[a[i]] > 0$，则说明 $a[i]$ 先前出现过
那么就需要先把当前版本线段树 $pre[a[i]]$ 位置的值赋为 INF

```
const int N = 5e5 + 10;
struct Ctree{
    int l , r , mi = inf;
}tree[N * 40];
int root[N] , Ctot;
void push_up(int rt)
{
    tree[rt].mi = min(tree[tree[rt].l].mi , tree[tree[rt].r].mi);
}
void update(int l , int r , int pre , int &now , int pos , int val)
{
    tree[++ Ctot] = tree[pre];
    now = Ctot;
    if(l == r)
    {
        tree[now].mi = val;
        return ;
```

```
    }
    int mid = l + r >> 1;
    if(pos <= mid) update(l , mid , tree[pre].l , tree[now].l , pos , val);
    else update(mid + 1 , r , tree[pre].r , tree[now].r , pos , val);
    push_up(now);
}
int query(int l , int r , int rt , int L , int R)
{
    if(L <= l && r <= R) return tree[rt].mi;
    int mid = l + r >> 1 , mi = inf;
    if(L <= mid) mi = min(mi , query(l , mid , tree[rt].l , L , R));
    if(R > mid) mi = min(mi , query(mid + 1 , r , tree[rt].r , L , R));
    return mi;
}
int n , m , a[N] , pre[N];
vector<int>vec;
int get(int x){
    return lower_bound(all(vec) , x) - vec.begin() + 1;
}
signed main()
{
    ios::sync_with_stdio(false);
    cin.tie(0) , cout.tie(0);
    cin >> n ;
    rep(i , 1 , n) cin >> a[i] , vec.pb(a[i]);
    sort(all(vec));
    vec.erase(unique(all(vec)) , vec.end());
    rep(i , 1 , n)
    {
        a[i] = get(a[i]);
        if(!pre[a[i]]) pre[a[i]] = -i;
        if(pre[a[i]] > 0)
        {
            int t = 0;
            update(1 , n , root[i - 1] , t , pre[a[i]] , inf);
            update(1 , n , t , root[i] , i , pre[a[i]]);
        }
        else update(1 , n , root[i - 1] , root[i] , i , pre[a[i]]);
        pre[a[i]] = i;
    }
    cin >> m;
    while(m --)
    {
        int l , r;
        cin >> l >> r;
        int mi = query(1 , n , root[r] , l , r);
        if(mi >= l) cout << 0 << '\n';
        else cout << vec[a[abs(mi)] - 1] << '\n';
    }
    return 0;
}
```

7.10.6 区间询问最小不能被表示的数

- 若一个区间的集合为 $\{1,1,1,4,13\}$，则该区间最小不能被为 8
  $1=1, 2=1+1, 3=1+1+1, 4=4, 5=4+1, 6=4+1+1, 7=4+1+1+1$
- $n, m <= 100000$，$\sum a[i] <= 10^9$

```
const int N = 1e5 + 5;
const int INF = 1e9;
struct Ctree
{
    int ls, rs, v;
} t[N * 40];
int rt[N], tot = 0;
void insert(int &o, int p, int l, int r, int pos, int v)
{
    o = ++tot, t[o] = t[p], t[o].v += v;
    if (l == r) return ;
    int mid = (l + r) >> 1;
    if (pos <= mid) insert(t[o].ls, t[p].ls, l, mid, pos, v);
    else insert(t[o].rs, t[p].rs, mid + 1, r, pos, v);
}
int query(int v, int u, int l, int r, int ql, int qr)
{
    if (ql <= l && r <= qr) return t[u].v - t[v].v;
    int mid = (l + r) >> 1, res = 0;
    if (ql <= mid) res += query(t[v].ls, t[u].ls, l, mid, ql, qr);
    if (qr > mid) res += query(t[v].rs, t[u].rs, mid + 1, r, ql, qr);
    return res;
}
int n , m , a[N];
signed main ()
{
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> a[i];
    for (int i = 1; i <= n; i++) insert(rt[i], rt[i - 1], 1, INF, a[i], a[i]);
    cin >> m;
    while (m --)
    {
        int l , r , ans = 1;
        cin >> l >> r;
        for (;;)
        {
            int res = query(rt[l - 1], rt[r], 1, INF, 1, ans);
            if (res >= ans) ans = res + 1;
            else break;
        }
        cout << ans << '\n';
    }
    return 0;
}
```

7.10.7 区间众数

众数指出现次数大于区间长度一半的数

询问区间 $[L, R]$ 的众数，若不存在则输出 0

二分左右区间

- 如果左区间的数的总和大于 $len/2$ ，就往左区间找
- 否则往右区间找
- 如果都没有则直接返回 0
- 这么找下去要么找到了要么就不存在

```
const int N = 5e5 + 10;
struct Tree{
    int l , r , sum;
}tree[N * 40];
int root[N] , Cnt;
void update(int l , int r , int pre , int &now , int pos)
{
    tree[++ Cnt] = tree[pre];
    tree[Cnt].sum ++ ;
    now = Cnt;
    if(l == r) return ;
    int mid = l + r >> 1;
    if(pos <= mid) update(l , mid , tree[pre].l , tree[now].l , pos);
    else update(mid + 1 , r , tree[pre].r , tree[now].r , pos);
}
int query(int l , int r , int L , int R , int len)
{
    if(l == r) return l;
    int mid = l + r >> 1;
    int Lsum = tree[tree[R].l].sum - tree[tree[L].l].sum;
    int Rsum = tree[tree[R].r].sum - tree[tree[L].r].sum;
    int ans = 0;
    /*
    这里其实取 max 其实是可以改成 return 的
    因为如果 Lsum > len , 那么 Rsum 就必然小于 len
    */
    if(Lsum > len) ans = max(ans , query(l , mid , tree[L].l , tree[R].l , len));
    if(Rsum > len) ans = max(ans , query(mid + 1 , r , tree[L].r , tree[R].r , len))
        ;
    return ans;
}
int a[N];
vector<int>vec;
int get_id(int x){
    return lower_bound(all(vec) , x) - vec.begin() + 1;
}
signed main()
{
    ios::sync_with_stdio(false);
    cin.tie(0) , cout.tie(0);
```

```
    int n , m;
    cin >> n >> m;
    rep(i , 1 , n) cin >> a[i] , vec.pb(a[i]);
    sort(all(vec));
    vec.erase(unique(all(vec)) , vec.end());
    int nn = vec.size();
    rep(i , 1 , n) a[i] = get_id(a[i]) , update(1 , nn , root[i - 1] , root[i] , a[i
        ]);
    while(m --)
    {
        int l , r;
        cin >> l >> r;
        int res = query(1 , nn , root[l - 1] , root[r] , (r - l + 1) / 2);
        if(res) res = vec[res - 1];
        cout << res << '\n';
    }
    return 0;
}
```

7.10.8 子树中距离其不超过 K 的最小点权

m 次询问，每次询问某个点的子树中距离其不超过 k 的点的权值的最小值

强制在线

以 dfs 序为下标，深度为时间轴建一棵主席树 (主席树对应的是 N 棵线段树而不是 N 棵权值线段树)

用 bfs，按深度一层层地把点加进主席树中（插入的 pos 为其的 dfs 序，val 为点权)

这样 $root[id[dep[x]+k]]$ 对应的线段树就只包含第 1 层到第 $dep[x]+k$ 层的节点

而该棵线段树的区间 $[dfl[x], dfr[x]]$ 对应的就是 x 的子节点，在这段区间内查询最小值即可

```
struct CTree{
    int l , r , mi = inf;
}tree[N * 40];
int root[N] , Ctot;
void push_up(int rt)
{
    tree[rt].mi = min(tree[tree[rt].l].mi , tree[tree[rt].r].mi);
}
void update(int l , int r , int pre , int &now , int pos , int val)
{
    tree[++ Ctot] = tree[pre];
    now = Ctot;
    if(l == r)
    {
        tree[now].mi = val;
        return ;
    }
    int mid = l + r >> 1;
    if(pos <= mid) update(l , mid , tree[pre].l , tree[now].l , pos , val);
    else update(mid + 1 , r , tree[pre].r , tree[now].r , pos , val);
    push_up(now);
}
int query(int l , int r , int rt , int L , int R)
{
```

```
    if(L <= l && r <= R) return tree[rt].mi;
    int mid = l + r >> 1 , mi = inf;
    if(L <= mid) mi = min(mi , query(l , mid , tree[rt].l , L , R));
    if(R > mid) mi = min(mi , query(mid + 1 , r , tree[rt].r , L , R));
    return mi;
}
int n , r , m , last , x , k , ma;
int tot , a[N] , dfl[N] , dfr[N] , dep[N] , vis[N] , id[N];
void dfs(int u , int far)
{
    dep[u] = dep[far] + 1;
    ma = max(ma , dep[u]);
    dfl[u] = ++ tot;
    for(int i = head[u] ; i ; i = edge[i].nex)
    {
        int v = edge[i].to;
        if(v == far) continue ;
        dfs(v , u);
    }
    dfr[u] = tot;
}
void bfs(int st)
{
    int cnt = 0;
    queue<int>que;
    que.push(st);
    vis[st] = 1;
    update(1 , n , root[0] , root[++ cnt] , dfl[st] , a[st]);
    id[1] = cnt;
    while(!que.empty())
    {
        int u = que.front();
        que.pop();
        for(int i = head[u] ; i ; i = edge[i].nex)
        {
            int v = edge[i].to;
            if(vis[v]) continue ;
            vis[v] = 1 , que.push(v);
            update(1 , n , root[cnt] , root[cnt + 1] , dfl[v] , a[v]);
            id[dep[v]] = ++ cnt;
        }
    }
}
vector<int>vec;
int get(int x){
    return lower_bound(all(vec) , x) - vec.begin() + 1;
}
signed main()
{
    cin >> n >> r;
    rep(i , 1 , n) cin >> a[i] , vec.pb(a[i]);
    sort(all(vec));
    vec.erase(unique(all(vec)) , vec.end());
```

```
78      rep(i , 1 , n) a[i] = get(a[i]);
79      rep(i , 2 , n)
80      {
81          int u , v;
82          cin >> u >> v;
83          add_edge(u , v) , add_edge(v , u);
84      }
85      dfs(r , 0) , bfs(r);
86      cin >> m;
87      rep(i , 1 , m)
88      {
89          int p , q;
90          cin >> p >> q;
91          x = (p + last) % n + 1;
92          k = (q + last) % n;
93          int ha = query(1 , n , root[id[min(dep[x] + k , ma)]] , dfl[x] , dfr[x]) - 1;
94          if(ha < 0) while(1);
95          last = vec[ha];
96          cout << last << '\n';
97      }
98      return 0;
99  }
```

7.11 树套树

7.11.1 单点修改、区间第 k 大

如果是全局动态第 k 大，可以树状数组上二分。

注意：因为没有离散化，时间和空间的常数都很大。

时间复杂度：$O(n*logn*log(up))$，空间复杂度：$O(n*logn*log(up))$。其中 up 是权值上限。

给定 n 个整数构成的序列，将对于指定的闭区间查询其区间内的第 k 小值。支持点修改。

输入 l,r,k 表示查询 $[l,r]$ 区间内第 k 小值。

思路：第 i 棵权值线段树维护的是 $[i-lowbit(i)+1,i]$ 而不是 $[1,i]$ 的信息。修改时修改 $logn$ 棵树。询问时进行 $logn$ 次两棵线段树做差。

```
1   int n , m ;
2   int a[N] ;
3   char s[5] ;
4   int lowbit(int x) { return x & (-x) ; }
5   struct Tree
6   {
7       int up , cnt ;
8       int lson[N * 300] , rson[N * 300] , sum[N * 300] ;
9       int rt[N * 300] ;
10      vector<int> vx , vy ;
11      void init()
12      {
13          up = 1e9 , cnt = 0 ;
14          mem0(lson) , mem0(rson) , mem0(sum) , mem0(rt) ;
15      }
16      void update(int &now , int l , int r , int x , int y)
17      {
18          if(now == 0) now = ++ cnt ;
```

```
        sum[now] += y ;
        if(l == r) return ;
        int mid = (l + r) >> 1 ;
        if(x <= mid) update(lson[now] , l , mid , x , y) ;
        else update(rson[now] , mid + 1 , r , x , y) ;
    }
    void add(int i , int x , int y)
    {
        for(int j = i ; j <= n ; j += lowbit(j))
        update(rt[j] , 0 , up , x , y) ;
    }
    int query(int l , int r , int k)
    {
        if(l == r) return l ;
        int num = 0 , mid = (l + r) >> 1 ;
        int sizx = sz(vx) , sizy = sz(vy) ;
        for(auto x : vx) num -= sum[lson[x]] ;
        for(auto y : vy) num += sum[lson[y]] ;
        if(k <= num)
        {
            rep(i , 0 , sizx - 1) vx[i] = lson[vx[i]] ;
            rep(i , 0 , sizy - 1) vy[i] = lson[vy[i]] ;
            return query(l , mid , k) ;
        }
        else
        {
            rep(i , 0 , sizx - 1) vx[i] = rson[vx[i]] ;
            rep(i , 0 , sizy - 1) vy[i] = rson[vy[i]] ;
            return query(mid + 1 , r , k - num) ;
        }
    }
    void ask(int l , int r , int k)
    {
        cl(vx) , cl(vy) ;
        for(int j = l - 1 ; j >= 1 ; j -= lowbit(j)) vx.pb(rt[j]) ;
        for(int j = r ; j >= 1 ; j -= lowbit(j)) vy.pb(rt[j]) ;
        w(query(0 , up , k)) , en() ;
    }
} tree ;
signed main()
{
    rr(n , m) ;
    rep(i , 1 , n) r(a[i]) ;
    tree.init() ;
    rep(i , 1 , n) tree.add(i , a[i] , 1) ;
    s[0] = 'Q' ;
    rep(i , 1 , m)
    {
        rst(s) ;
        if(s[0] == 'Q')
        {
            int l , r , k ;
            rr(l , r) , r(k) ;
```

```
            tree.ask(l , r , k) ;
        }
        else
        {
            int x , y ;
            rr(x , y) ;
            tree.add(x , a[x] , -1) ;
            a[x] = y ;
            tree.add(x , a[x] , 1) ;
        }
    }
    return 0 ;
}
```

7.11.2 数值删除、逆序对数（排列）

- 排列，每次从中删除一个元素，一个删除 m 次，求每次删除后的逆序对数
- 时、空复杂度 $O(nlog^2n)$

```
#define rep(i , a , b) for(int i = a ; i <= b ; i ++)
#define ll long long
const int N = 1e5 + 10 , M = 3e7 + 10;
int n , m , tot;
ll ans;
int lowbit(int x){
    return x & (-x);
}
int a[N] , pos[N] , quea[N] , queb[N];
int root[N];
struct Ctree{
    int l , r , sum;
}tree[M];
void change(int &root , int l , int r , int x , int y)
{
    if(!root) root = ++ tot;
    tree[root].sum += y;
    if(l == r) return;
    int mid = (l + r) >> 1;
    if(x <= mid) change(tree[root].l , l , mid , x , y);
    else change(tree[root].r , mid + 1 , r , x , y);
}
int query(int l , int r , int x , int mode)
{
    int cnta = 0 , cntb = 0 , sum = 0;
    for(int i = l - 1 ; i ; i -= lowbit(i)) quea[++ cnta] = root[i];
    for(int i = r ; i ; i -= lowbit(i))  queb[++ cntb] = root[i];
    l = 1 , r = n;
    while(l != r)
    {
        int mid = (l + r) >> 1;
        if(x > mid)
        {
```

```
34          if(mode)
35          {
36              rep(i , 1 , cnta) sum -= tree[tree[quea[i]].l].sum;
37              rep(i , 1 , cntb) sum += tree[tree[queb[i]].l].sum;
38          }
39          rep(i , 1 , cnta) quea[i] = tree[quea[i]].r;
40          rep(i , 1 , cntb) queb[i] = tree[queb[i]].r;
41          l = mid + 1;
42      }
43      else
44      {
45          if (!mode)
46          {
47              rep(i , 1 , cnta) sum -= tree[tree[quea[i]].r].sum;
48              rep(i , 1 , cntb) sum += tree[tree[queb[i]].r].sum;
49          }
50          rep(i , 1 , cnta) quea[i] = tree[quea[i]].l;
51          rep(i , 1 , cntb) queb[i] = tree[queb[i]].l;
52          r = mid;
53      }
54  }
55  return sum;
56 }
57 signed main()
58 {
59  cin >> n >> m;
60  rep(i , 1 , n)
61  {
62      cin >> a[i];
63      pos[a[i]] = i;
64      ans += query(1 , i - 1 , a[i] , 0);
65      for(int j = i ; j <= n ; j += lowbit(j)) change(root[j] , 1 , n , a[i] , 1);
66  }
67  cout << ans << '\n';
68  rep(i , 1 , m - 1)
69  {
70      int x;
71      cin >> x;
72      ans -= query(1 , pos[x] - 1 , x , 0);
73      ans -= query(pos[x] + 1 , n , x , 1);
74      cout << ans << '\n';
75      for(int j = pos[x] ; j <= n ; j += lowbit(j)) change(root[j] , 1 , n , x ,
            -1);
76  }
77  return 0;
78 }
```

7.12 块状链表

使用：

- 头文件：`#include<ext/rope>`

- 命名空间：`using namespace __gnu_cxx;`
- 定义方法：
  - *rope*< 变量类型 > 变量名称
  - *crope* 变量名称（相当于定义成 *rope* < *char* >，即定义为 *string* 类型）

定义为 *string*：

- `insert(int pos, char *s, int n)`
  将字符串 s 的前 n 位插入 *rope* 的下标 *pos* 处，如没有参数 n 则将字符串 s 的所有位都插入 *rope* 的下标 *pos* 处（补充地址知识：如果你不想从字符串下标为 0（即第一个字符）的地址开始取 n 位，就将你想开始取的地址代入。如 $s+1$ 表示从字符串下标为 1（即第二个字符）的地址开始取 n 位。*int char* 等变量类型的数组都适用这种方法来更改数组操作的起始位置。）
- `append(char *s , int pos , int n)`
  把字符串 s 中从下标 *pos* 开始的 n 个字符连接到 *rope* 的结尾，如没有参数 n 则把字符串 s 中下标 *pos* 后的所有字符连接到 *rope* 的结尾，如没有参数 *pos* 则把整个字符串 s 连接到 *rope* 的结尾（这里已经给你起始位置参数 *pos* 了就没必要用上述的取地址方法了哈）
- *insert* 和 *append* 的区别：*insert* 能把字符串插入到 *rope* 中间，但 *append* 只能把字符串接到结尾）
- `substr(int pos , int len)`
  提取 *rope* 的从下标 *pos* 开始的 *len* 个字符
- `at(int x)` 访问 *rope* 的下标为 x 的元素
- `erase(int pos , int num)` 从 *rope* 的下标 *pos* 开始删除 *num* 个字符
- `copy(int pos , int len , string &s)` 从 *rope* 的下标 *pos* 开始的 *len* 个字符用字符串 s 代替，如果 *pos* 后的位数不够就补足
- `replace(int pos , string &x)`
  从 *rope* 的下标 *pos* 开始替换成字符串 x，x 的长度为从 *pos* 开始替换的位数，如果 *pos* 后的位数不够就补足

定义为 *int*

- `insert(int pos , int *a , int n)`
  将 *int* 数组 a 的前 n 位插入 *rope* 的下标 *pos* 处，如没有参数 n 则将数组 s 的所有位都插入 *rope* 的下标 *pos* 处
- `append(int *a , int pos , int n)`
  把数组 a 中从下标 *pos* 开始的 n 个数连接到 *rope* 的结尾，如没有参数 n 则把数组 a 中下标 *pos* 后的所有数连接到 *rope* 的结尾，如没有参数 *pos* 则把整个数组 a 连接到 *rope* 的结尾
- `substr(int pos , int len)` 提取 *rope* 的从下标 *pos* 开始的 *len* 个数
- `at(int x)` 访问 *rope* 的下标为 x 的元素
- `erase(int pos , int num)` 从 *rope* 的下标 *pos* 开始删除 *num* 个数
- `copy(int pos , int len , int *a)`
  从 *rope* 的下标 *pos* 开始的 *len* 个数用数组 a 代替，如果 *pos* 后的位数不够就补足

- `replace(int pos , int *a)`
  从 *rope* 的下标 *pos* 开始替换成数组 a，a 的长度为从 *pos* 开始替换的位数，如果 *pos* 后的位数不够就补足
- 时间复杂度：$O(n\sqrt{n}$ ，具体原理详见块状链表

可持久化：

```
rope<int>*rs[N + 10];
rs[0]=new rope<int>();
rs[0]->append(0);
rs[i] = new rope<int>(*rs[v]);
rs[i] -> replace(a , b);
rs[i] -> at(a));
```

7.13 普通并查集

7.13.1 路径压缩

```
int far[N];
int find(int x)
{
   if(x == far[x]) return x;
   return far[x] = find(far[x]);
}
void Union(int x , int y)
{
   int tx = find(x) , ty = find(y);
   if(tx != ty) far[tx] = ty;
}
void init(int n)
{
   for(int i = 1 ; i <= n ; i ++) far[i] = i;
}
```

7.13.2 按秩合并

- 按秩合并可配合路径压缩一起使用，复杂度为 $\alpha(N)$
- $\alpha(N)$ 为反阿克曼函数，它是一个比 $\log N$ 增长得还要慢的函数
- 但是路径压缩不支持撤销，而按秩合并可以

```
int far[N] , dep[N];
int find(int x)
{
   if(x == far[x]) return x;
   return find(far[x]); // 改为 far[x] = find(far[x]) 就可添上路径压缩
}
void Union(int x , int y)
{
   int tx = find(x) , ty = find(y);
   if(tx == ty) return ;
```

```
    if(dep[tx] < dep[ty]) far[tx] = ty;
    else
    {
        far[ty] = tx;
        if(dep[tx] == dep[ty]) dep[tx] ++ ;
    }
}
void init(int n)
{
    for(int i = 1 ; i <= n ; i ++) far[i] = i , dep[i] = 1;
}
```

7.13.3 启发式合并 + 路径压缩

- 启发式合并其实就是按秩合并的一种
- 不过它是按集合大小来的

```
int far[N] , size[N];
int find(int x)
{
    if(x == far[x]) return x;
    return far[x] = find(far[x]); // 改为 return find(far[x]) 则去掉了路径压缩
}
void Union(int x , int y)
{
    int tx = find(x) , ty = find(y);
    if(tx == ty) return ;
    if(size[tx] > size[ty]) swap(tx , ty);
    far[tx] = ty , size[ty] += size[tx];
}
void init(int n)
{
    for(int i = 1 ; i <= n ; i ++) far[i] = i , size[i] = 1;
}
```

7.14 种类并查集

7.15 可撤销并查集

7.15.1 可撤销并查集

- 可撤销并查集不支持路径压缩，只能按秩合并（所以要记得改 $find$ 函数）
- 实现过程就是当两个数合并的时候把它们丢入栈中，等需要撤销的时候再从栈中拿出（通常搭配线段树时间分治使用）

```
int far[N] , size[N];
int find(int x)
{
    if(x == far[x]) return x;
    return find(far[x]); // 不能用路径压缩!
}
```

```
void Union(int x , int y , stack<pii>&st)
{
    int tx = find(x) , ty = find(y);
    if(tx != ty)
    {
        if(size[tx] > size[ty]) swap(tx , ty);
        st.push(make_pair(tx , ty));
        far[tx] = ty;
        size[ty] += size[tx];
    }
}
void Redo(stack<pii>&st)
{
    while(!st.empty())
    {
        pii u = st.top();
        st.pop();
        far[u.fi] = u.fi;
        size[u.se] -= size[u.fi];
    }
}
```

7.15.2 加边、删边、判二分图

- n 个点的无向图，起初无任何边
- 若 u,v 之间无边，则为 u,v 加一条边
- 若 u,v 之间有边，则删除 u,v 之间的边
- 每次操作结束后判断该图此时是否是二分图
- $2 \leq n,q \leq 10^5$

离线。记录每条边添加的时间和删除的时间，然后套线段树时间分治

tricks

判断图是否是二分图可以用种类并查集：

给 u,v 之间加边，相当于要让 u,v 成为敌人，那么只要判断再此之前 u,v 是否是朋友即可

```
typedef pair<int , int> pii;
const int N = 2e5 + 10;
int n , m;
int far[N] , size[N];
void init()
{
    rep(i , 1 , N - 10) far[i] = i , size[i] = 1;
}
int find(int x)
{
    if(x == far[x]) return x;
    return find(far[x]);
}
void Union(int x , int y , stack<pii>&st)
{
```

```
    int tx = find(x) , ty = find(y);
    if(tx != ty)
    {
        if(size[tx] > size[ty]) swap(tx , ty);
        st.push(make_pair(tx , ty));
        far[tx] = ty;
        size[ty] += size[tx];
    }
}
void Redo(stack<pii>&st)
{
    while(!st.empty())
    {
        pair<int , int>u = st.top();
        st.pop();
        far[u.fi] = u.fi;
        size[u.se] -= size[u.fi];
    }
}
struct Tree{
    int l , r;
    vector<pii>vec;
}tree[N << 2];
void build(int l , int r , int rt)
{
    tree[rt].l = l , tree[rt].r = r;
    if(l == r) return ;
    int mid = l + r >> 1;
    build(l , mid , rt << 1);
    build(mid + 1 , r , rt << 1 | 1);
}
void update_range(int L , int R , int rt , pii p)
{
    int l = tree[rt].l , r = tree[rt].r;
    if(L <= l && r <= R)
    {
        tree[rt].vec.pb(p);
        return ;
    }
    int mid = l + r >> 1;
    if(L <= mid) update_range(L , R , rt << 1 , p);
    if(R > mid) update_range(L , R , rt << 1 | 1 , p);
}
void query(int rt , int ok)
{
    int l = tree[rt].l , r = tree[rt].r;
    stack<pii>st;
    for(auto i : tree[rt].vec)
    {
        if(find(i.fi) == find(i.se)) ok = 0;
        Union(i.fi , i.se + n , st);
        Union(i.fi + n , i.se , st);
    }
```

```
    if(l == r)
    {
        if(ok) cout << "YES\n";
        else cout << "NO\n";
    }
    else query(rt << 1 , ok) , query(rt << 1 | 1 , ok);
    Redo(st);
}
map<pair<int , int> , int>mp;
signed main()
{
    ios::sync_with_stdio(false);
    cin.tie(0) , cout.tie(0);
    init();
    cin >> n >> m;
    build(1 , m , 1);
    rep(i , 1 , m)
    {
        int u , v;
        cin >> u >> v;
        if(u > v) swap(u , v);
        auto it = mp.find(make_pair(u , v));
        if(it != mp.end())
        {
            update_range(it->se , i - 1 , 1 , make_pair(u , v));
            mp.erase(it);
        }
        else mp[make_pair(u , v)] = i;
    }
    for(auto i : mp) update_range(i.se , m , 1 , i.fi);
    query(1 , 1);
    return 0;
}
```

7.15.3 加边、删边、判联通

- n 个点的无向图，起初无任何边
- 为 u v 加一条边（保证 u,v 之间始终只有一条边）
- 删除 u,v 之间的边（保证合法）
- 判断 u,v 是否联通
- $n \le 10^5, m \le 2 \times 10^5$

离线。记录每条边出现的时间和结束的时间，然后套线段树时间分治

```
#include<bits/stdc++.h>
#define rep(i , a , b) for (int i = a ; i <= b ; i ++)
#define fi first
#define se second
#define pb push_back
using namespace std;

```

```cpp
const int N = 3e5 + 10;
typedef pair<int , int> pii;

int n , m , far[N] , size[N];
map<pair<int , int> , int>mp;
vector<pair<pii , int>>Q[N];
int find(int x)
{
    if(x == far[x]) return x;
    return find(far[x]); // 不能用路径压缩!
}
void Union(int x , int y , stack<pii>&st)
{
    int tx = find(x) , ty = find(y);
    if(tx != ty)
    {
        if(size[tx] > size[ty]) swap(tx , ty);
        st.push(make_pair(tx , ty));
        far[tx] = ty;
        size[ty] += size[tx];
    }
}
void Redo(stack<pii>&st)
{
    while(!st.empty())
    {
        pii u = st.top();
        st.pop();
        far[u.fi] = u.fi;
        size[u.se] -= size[u.fi];
    }
}
struct Tree{
    int l , r;
    vector<pair<int , int>>vec;
}tree[N << 2];
void build(int l , int r , int rt)
{
    tree[rt].l = l , tree[rt].r = r;
    tree[rt].vec.clear();
    if(l == r) return ;
    int mid = l + r >> 1;
    build(l , mid , rt << 1);
    build(mid + 1 , r , rt << 1 | 1);
}
void update_range(int L , int R , int rt , pair<int , int>p)
{
    int l = tree[rt].l , r = tree[rt].r;
    if(L <= l && r <= R)
    {
        tree[rt].vec.pb(p);
        return ;
    }
```

```
    int mid = l + r >> 1;
    if(L <= mid) update_range(L , R , rt << 1 , p);
    if(R > mid) update_range(L , R , rt << 1 | 1 , p);
}
void query(int rt)
{
    int l = tree[rt].l , r = tree[rt].r;
    stack<pii>st;
    for(auto i : tree[rt].vec) Union(i.fi , i.se , st);
    if(l == r)
    {
        if(Q[l].size())
        {
            int tx = find(Q[l][0].fi.fi) , ty = find(Q[l][0].fi.se);
            if(tx == ty) Q[l][0].se = true;
            else Q[l][0].se = false;
        }
    }
    else
    {
        int mid = l + r >> 1;
        query(rt << 1);
        query(rt << 1 | 1);
    }
    Redo(st);
}
signed main()
{
    cin >> n >> m;
    rep(i , 1 , n) far[i] = i , size[i] = 1;
    build(1 , m , 1);
    rep(i , 1 , m)
    {
        string op;
        int u , v;
        cin >> op >> u >> v;
        if(u > v) swap(u , v);
        if(op[0] == 'Q') Q[i].pb(make_pair(make_pair(u , v) , false));
        else if(op[0] == 'C')
        {
            if(mp.find(make_pair(u , v)) == mp.end()) mp[make_pair(u , v)] = i;
        }
        else
        {
            update_range(mp[make_pair(u , v)] , i , 1 , make_pair(u , v));
            mp.erase(mp.find(make_pair(u , v)));
        }
    }
    for(auto i : mp) update_range(i.se , m , 1 , i.fi);
    query(1);
    rep(i , 1 , m) if(Q[i].size()) Q[i][0].se ? cout << "Yes\n" : cout << "No\n";
    return 0;
}
```

7.16 可持久化并查集

7.16.1 总结

可持久化并查集 = 可持久化数组 + 并查集 = 主席树维护叶子结点 + 并查集

且该并查集的合并必须是**按秩合并**

解释下不能使用路径压缩的原因：

- 因为路径压缩时，再次调用 $find$ 后，是要更新一部分值的
- 在数组上搞这些操作倒是挺快，然而在主席树里，每次找一个 $farther$ 要 log 次
- 把这个节点更新又要新建 log 个节点，一共要循环找不满 log 次，理论上时间复杂度是 $O(Mlog^2N)$ 的
- 但空间也是 $O(Mlog^2N)$，所以直接 GG

7.16.2 可持久化并查集

给定 n 个集合，第 i 个集合内初始状态下只有一个数，为 i

有 m 次操作。操作分为 3 种：

- `1 a b` 合并 a,b 所在集合
- `2 k` 回到第 k 次操作（执行三种操作中的任意一种都记为一次操作）之后的状态
- `3 a b` 询问 a,b 是否属于同一集合，如果是则输出 1 ，否则输出 0

```
const int N = 3e5 + 10;
struct Tree{
    int l , r , far , dep;
}tree[N << 5];
int root[N] , Ctot;
int n , m;
void build(int &now , int l , int r)
{
    now = ++ Ctot;
    if(l == r)
    {
        tree[now].far = l , tree[now].dep = 1;
        return ;
    }
    int mid = l + r >> 1;
    build(tree[now].l , l , mid);
    build(tree[now].r , mid + 1 , r);
}
void update(int l , int r , int pre , int &now , int pos , int F)
{
    tree[++ Ctot] = tree[pre];
    now = Ctot;
    if(l == r)
    {
        tree[now].far = F;
        tree[now].dep = tree[pre].dep;
        return ;
    }
```

```
29      int mid = l + r >> 1;
30      if(pos <= mid) update(l , mid , tree[pre].l , tree[now].l , pos , F);
31      else update(mid + 1 , r , tree[pre].r , tree[now].r , pos , F);
32  }
33  int query(int l , int r , int rt , int pos)
34  {
35      if(l == r) return rt;
36      int mid = l + r >> 1;
37      if(pos <= mid) return query(l , mid , tree[rt].l , pos);
38      else return query(mid + 1 , r , tree[rt].r , pos);
39  }
40  int find(int x , int i)
41  {
42      int x_id = query(1 , n , root[i] , x);
43      int F = tree[x_id].far;
44      if(F == x) return F;
45      return find(F , i); // 无路径压缩
46  }
47  void Union(int x , int y , int i)
48  {
49      int tx = find(x , i);
50      int ty = find(y , i);
51      if(tx == ty) return ;
52      int tx_id = query(1 , n , root[i] , tx); // 找到祖先对应的编号
53      int ty_id = query(1 , n , root[i] , ty); // 此时的 root[i] 就是 root[i - 1]
54      int dep_x = tree[tx_id].dep , dep_y = tree[ty_id].dep;
55      if(dep_x > dep_y) swap(tx_id , ty_id) , swap(tx , ty);
56      update(1 , n , root[i - 1] , root[i] , tx , ty); // 为 root[i] 创建新分支
57      if(dep_x == dep_y)
58      {
59          ty_id = query(1 , n , root[i] , ty);
60          tree[ty_id].dep ++ ;
61      }
62  }
63  signed main()
64  {
65      cin >> n >> m;
66      build(root[0] , 1 , n);
67      for(int i = 1 ; i <= m ; i ++)
68      {
69          int op , a , b , k;
70          cin >> op;
71          if(op == 1)
72          {
73              cin >> a >> b;
74              root[i] = root[i - 1];
75              Union(a , b , i);
76          }
77          else if(op == 2)
78          {
79              cin >> k;
80              root[i] = root[k];
81          }
```

```
82          else if(op == 3)
83          {
84              cin >> a >> b;
85              root[i] = root[i - 1];
86              if(find(a , i) == find(b , i)) puts("1");
87              else puts("0");
88          }
89      }
90      return 0;
91  }
```

7.17 带权并查集

7.17.1 带权并查集

- 带权并查集顾名思义就是给集合加上权值
- 一般维护**点到根节点（祖先）的距离**和**树的大小（集合大小）**
- 然后通过两者计算出答案（有点和 **lca** 类似？）

例题：

约翰和贝茜在玩一个方块游戏。编号为 $1...n$（$1 \leq n \leq 30000$）个方块正放在地上，每个构成一个立方柱。

游戏开始后，约翰会给贝茜发出 P（$1 \leq P \leq 100000$）个指令。指令有两种：

1. 移动（M）：将包含 X 的立方柱移动到包含 Y 的立方柱上。
2. 询问（C）：询问含 X 的立方柱中，在 X 下方的方块数目。

解：

- $far[x]$ 表示 x 的祖先
- $dis[x]$ 表示 x 到根节点的距离
- $size[x]$ 表示 x 子树的大小，$size$ 数组只适用于祖先

在将 x 放在 y 上后，x 的祖先成了 y 集合的新祖先

所以有：

- $far[tx] = ty$
- $dis[ty] = size[tx]$
- $size[tx] += size[ty]$

而 ty 子树中的节点的 dis 此时还未更新

它们更新将和路径压缩一并进行

即调用到 $find$ 函数时：

- $f = far[y]$（f 为 y 的原祖先）
- $far[y] = find(far[y])$（将原祖先的祖先更新为新祖先）
- $dis[y] += dis[f]$（y 到新祖先的距离增加了 $dis[f]$）

那么 X 下方的方块数目 $= size[F] - dis[X] - 1$ (F 为 X 的祖先)

```
int far[N] , size[N] , dis[N];
int find(int x)
{
    if(x == far[x]) return x;
    int t = far[x]; // 原来的根祖先
    far[x] = find(far[x]); // 根祖先改变
    dis[x] += dis[t]; // 到新的根祖先的距离
    return far[x];
}
void Union(int x , int y)
{
    int tx = find(x) , ty = find(y);
    if(tx == ty) return ;
    far[ty] = tx;
    dis[ty] = size[tx];
    size[tx] += size[ty];
}
void init(int n)
{
    for(int i = 1 ; i <= n ; i ++) far[i] = i , size[i] = 1 , dis[i] = 0;
}
if(op == 'M')
{
    cin >> x >> y; // x 放在 y 上面
    Union(x , y);
}
else
{
    cin >> c;
    int f = find(c);
    cout << size[f] - dis[c] - 1 << '\n';
}
```

7.18 双指针

例子:

1. n 个数得序列 a 中挑选 2 个数使 $a[i] + a[j] \leq m$ $(i < j)$ 的方案数。

```
sort(a + 1 , a + n + 1) ;
int l = 1 , r = n ;
ll ans = 0 ; //方案数
while(l < r)
{
    if(a[l] + a[r] <= m) ans += r - l , l ++ ;
    else r -- ;
}
```

8 字符串

8.1 字典树

8.1.1 指针版（释放内存）

```
// 注意初始化 root
struct tree
{
    int ed;
    tree *nex[10];
    tree()
    {
        ed = 0;
        for(int i = 0 ; i < 10 ; i ++) nex[i] = NULL;
    }
};
tree *root;
int inser(char a[])
{
    int flag = 0 , k;
    tree *p = root;
    int len = strlen(a);
    for(int i = 0 ; i < len ; i ++)
    {
        k = a[i] - '0';
        if(p -> nex[k] == NULL)
        {
            p -> nex[k] = new tree;
            flag = 1;
        }
        p = p -> nex[k];
        if(p -> ed) return 0;
    }
    p -> ed = 1;
    if(!flag) return 0;
    return 1;
}
void del(tree *root)
{
    for(int i = 0; i < 10 ; i ++)
    {
        if(root -> nex[i] != NULL) del(root -> nex[i]);
    }
    free(root);
    return ;
}
char s[110];
int main()
{
    int T = 1;
    cin >> T;
    while(T --)
```

```
48      {
49          int n;
50          cin >> n;
51          int flag = 1;
52          root = new tree();
53          while(n --)
54          {
55              cin >> s;
56              if(flag) flag = inser(s);
57          }
58          if(!flag) printf("NO\n");
59          else printf("YES\n");
60          del(root);
61      }
62      return 0;
63  }
```

8.1.2 字典树

```
1   void insert(vector<int>vec)
2   {
3       int root = 0;
4       for(auto i : vec)
5       {
6           if(!tree[root][i]) sum1[root] ++ , tree[root][i] = ++ tot , rt[root].pb(i);
7           root = tree[root][i];
8       }
9   }
10  void query(vector<int>vec)
11  {
12      int root = 0;
13      for(auto i : vec)
14      {
15          sum2[root] = 1;
16          if(!tree[root][i]) return ;
17          root = tree[root][i];
18      }
19  }
20  void del(int root)
21  {
22      for(auto i : rt[root])
23      {
24          del(i);
25          tree[root][i] = 0;
26      }
27      rt[root].clear();
28  }
```

8.2 字符串 hash

8.2.1 总结

- 尽量要用双 *hash* 并取模
  双 *hash* 即用相同 *base* 不同 *mod* 得到两个不同的 *hash* 值
  之后的操作只有当两个 *hash* 值都满足才算 *ok*
  取模比较慢，若卡常可以考虑 *ull* 自然溢出
- 二维 *hash* 是先对每行进行 *hash*，再对每列进行 *hash*
  注意：*row* hash 的基数不能和 *col hash* 的基数相同
- 二分 + *hash* 是常用套路
  一维枚举字符串中点，二维枚举正方形中心（中心不一定是整数点，需要进行一定讨论）

8.2.2 字符串 hash

```
#define ull unsigned long long
using namespace std;
const int N = 2e5 + 10;
const int Max = 2e6 + 10;
const ull base = 131;
struct Hash{
    ull power[Max] , hash1[Max] , hash2[Max];
    ull get_hash1(int l , int r){
        return hash1[r] - hash1[l - 1] * power[r - l + 1];
    }
    ull get_hash2(int l , int r){
        return hash2[l] - hash2[r + 1] * power[r - l + 1];
    }
    void hash_init(int n)
    {
        power[1] = base;
        for(int i = 2 ; i <= n ; i ++) power[i] = power[i - 1] * base;
    }
    void hash_pre(char *s)
    {
        int len = strlen(s + 1);
        hash1[1] = s[1] - 'a' + 1 , hash1[len + 1] = '\0';
        for(int i = 2 ; i <= len ; i ++) hash1[i] = hash1[i - 1] * base + s[i] - 'a'
            + 1;
    }
    void hash_suf(char *s)
    {
        int len = strlen(s + 1);
        hash2[len] = s[len] - 'a' + 1 , hash2[len + 1] = '\0';
        for(int i = len - 1 ; i >= 1 ; i --) hash2[i] = hash2[i + 1] * base + s[i] -
            'a' + 1;
    }
}ha;
char s[N];
unordered_map<int , int>mp;
signed main()
{
```

```
    ha.hash_init(N - 10);
    int n , res = 0;
    cin >> n;
    while(n --)
    {
        cin >> s + 1;
        int len = strlen(s + 1);
        ha.hash_pre(s); // 千万别传成 s + 1
        int now = ha.get_hash1(1 , len);
        if(!mp.count(now)) res ++ , mp[now] ++ ;
    }
    cout << res << '\n';
    return 0;
```

8.2.3 二维 hash、矩阵 hash

- 求最大 GsT 正方形的边长
- GsT 正方形：旋转 180° 后和旋转前完全相同

```
#define ll long long
const int N = 3e2 + 10 , MOD = 1e9 + 9;
const int A = 13331 , B = 131;
int n , m , a[N][N];
ll p1[N] , p2[N] , h1[N][N] , h2[N][N];
ll get_hash1(int a , int b , int c , int d)
{
    int ans = (h1[c][d] - h1[a - 1][d] - h1[c][b - 1] + h1[a - 1][b - 1] + MOD) %
        MOD;
    return ans * p1[n - a] % MOD * p2[m - b] % MOD;
}
ll get_hash2(int a , int b , int c , int d)
{
    int ans = (h2[a][b] - h2[c + 1][b] - h2[a][d + 1] + h2[c + 1][d + 1] + MOD) %
        MOD;
    return ans * p1[c - 1] % MOD * p2[d - 1] % MOD;
}
void init()
{
    p1[0] = p2[0] = 1;
    for(int i = 1 ; i <= N - 10 ; i ++) p1[i] = p1[i - 1] * A % MOD , p2[i] = p2[i -
         1] * B % MOD;
}
signed main()
{
    init();
    cin >> n >> m;
    for(int i = 1 ; i <= n ; i ++) for(int j = 1 ; j <= m ; j ++)
    {
        char ch;
        cin >> ch;
        if(ch == '1') a[i][j] = 1;
    }
```

```
    for(int i = 1 ; i <= n ; i ++) for(int j = 1 ; j <= m ; j ++) h1[i][j] = p1[i] *
        p2[j] * a[i][j] % MOD;
    for(int i = 1 ; i <= n ; i ++) for(int j = 1 ; j <= m ; j ++) h2[i][j] = p1[n -
        i + 1] * p2[m - j + 1] * a[i][j] % MOD;
    for(int i = 1 ; i <= n ; i ++) for(int j = 1 ; j <= m ; j ++) h1[i][j] = h1[i -
        1][j] + h1[i][j - 1] - h1[i - 1][j - 1] + h1[i][j];
    for(int i = n ; i >= 1 ; i --) for(int j = m ; j >= 1 ; j --) h2[i][j] = h2[i +
        1][j] + h2[i][j + 1] - h2[i + 1][j + 1] + h2[i][j];
    int res = -1;
    for(int i = 1 ; i <= n ; i ++) for(int j = 1 ; j <= m ; j ++)
    {
        // (i,j) 作为正方形中点
        int l = 1 , r = min(min(i - 1 , n - i) , min(j - 1 , m - j));
        while(l <= r)
        {
            int mid = l + r >> 1;
            int x1 = i - mid , y1= j - mid , x2 = i + mid , y2 = j + mid;
            ll ha = get_hash1(x1 , y1 , x2 , y2);
            ll he = get_hash2(x1 , y1 , x2 , y2);
            if(ha == he) l = mid + 1 , res = max(res , 2 * mid + 1);
            else r = mid - 1;
        }
        // (i-0.5,j-0.5) 作为正方形的中点
        l = 1 , r = min(min(i - 1 , n - i + 1) , min(j - 1 , m - j + 1));
        while(l <= r)
        {
            int mid = l + r >> 1;
            int x1 = i - 1 - mid + 1 , y1 = j - 1 - mid + 1 , x2 = i + mid - 1 , y2 =
                j + mid - 1;
            ll ha = get_hash1(x1 , y1 , x2 , y2);
            ll he = get_hash2(x1 , y1 , x2 , y2);
            if(ha == he) l = mid + 1 , res = max(res , mid * 2);
            else r = mid - 1;
        }

    }
    cout << res << '\n';
    return 0;
}
```

9 头脑风暴

9.1 dp 合集

9.1.1 不等式约束（牛客）

- 有三个长度为 n 的数列 $\{a_1, a_2, ..., a_n\}$ $\{b_1, b_2, ..., b_n\}$ $\{c_1, c_2, ..., c_n\}$
- 需要满足约束：

  1. $a_1 \times x_1 + a_2 \times x_2 + ... + a_n \times x_n \leq P$
  2. $b_1 \times x_1 + b_2 \times x_2 + ... + b_n \times x_n \geq P$

3. $\forall i, x_i \in \{0,1\}$

- 要求最小化 $w = c_1 \times x_1 + c_2 \times x_2 + ... + c_n \times x_n$
- 其中 a_i, b_i, c_i 满足 $a_i \le b_i \le 2 \times 10^6$

注意到题目中的关键信息 $a_i \le b_i$
由此可得对于任意一种方案，$\sum_{i=1}^{n} x_i a_i \le \sum_{i=1}^{n} x_i b_i$
设 $f[i][j]$ 表示前 i 项，满足 $\sum_{i=1}^{n} x_i a_i \le j$ 且 $\sum_{i=1}^{n} x_i b_i \ge j$ 时 w 的最小值

- 对 a，$[-\infty, j - a_i] + ai <= j$
- 对 b，$[j - b_i, \infty] + b_i >= j$

所以转移的区间为 $[-\infty, j - a_i] \cup [j - b_i, \infty] = [j - b_i, j - a_i]$
所以转移方程为：$f[i][j] = \min\left(\min_{j-b_i \le k \le j+a_i}\left(f[i-1][k] + w_i\right), \min(f[i-1][j])\right)$
可以利用单调队列优化，时间复杂度为 $O(nP)$

```
const int N = 1e3 + 10 , M = 1e4 + 10;
int n , p , a[N] , b[N] , c[N] , dp[N][M];
signed main()
{
   int T = 1;
   cin >> T;
   while(T --)
   {
      cin >> n >> p;
      rep(i , 1 , n) cin >> a[i];
      rep(i , 1 , n) cin >> b[i];
      rep(i , 1 , n) cin >> c[i];
      memset(dp , 0x3f , sizeof(dp));
      dp[0][0] = 0;
      rep(i , 1 , n)
      {
         rep(j , 0 , p) dp[i][j] = dp[i - 1][j];
         deque<int>deq;
         rep(j , a[i] , p)
         {
            int x = j - a[i];
            while(deq.size() && dp[i - 1][x] <= dp[i - 1][deq.front()]) deq.
                pop_front();
            deq.push_back(x);
            while(deq.size() && deq.front() < j - b[i]) deq.pop_front();
            if(!deq.size()) continue ;
            dp[i][j] = min(dp[i - 1][j] , dp[i - 1][deq.front()] + c[i]);
         }
      }
      if(dp[n][p] > 2e9) cout << "IMPOSSIBLE!!!\n";
      else cout << dp[n][p] << '\n';
   }
   return 0;
}
```

9.1.2 三元组的约束（AT）

problem：

- 给定 M 个约束条件，问满足这 M 个约束条件的长度为 N 排列有多少个
  每个约束条件为一个三元组 (x, y, z)，要求 $a_1, a_2, .., a_x$ 小于 y 的数的个数不超过 z

solve
可以将 x 位置的约束条件存储在 $vec[x]$ 中，并定义 $dp[i][bit]$ 表示：

- 用 bit 这个状态对应的数（这些数都得用上）
- 组成长度为 i 的排列的方案数
- （这些排列需要满足 $vec[1] \sim vec[i]$ 的约束条件）

那么答案就为 $dp[n][(1 << n) - 1]$

- 而如果 bit 这个状态对应的数的个数等于 i
- 这 i 个数满足 $vec[i]$ 的约束条件

那么可得：$if(bit >> k\&1)dp[i][bit] += dp[i-1][bit-(1<<k)];$
然后考虑一波优化：
先定义 $cnt(bit)$ 表示 bit 这个状态对应的数的个数。
上面有个 dp 的转移条件是：

- bit 这个状态对应的数的个数等于 i，

也就是说只有当 $i = cnt[bit]$ 时才能进行转移（只有 $dp[cnt(bit)][bit]$ 是有效的）
那么只要枚举 bit，就可以得到对应有效的 i $(i = cnt(bit))$
于是我们可以去掉 dp 数组的一维，并去掉一层循环
即定义 $dp[bit]$ 表示：

- 用 bit 这个状态对应的数（这些数都得用上）
- 组成长度为 $cnt[bit(i)]$ 的排列的方案数
- （这些排列需要满足 $vec[1] \sim vec[cnt(bit)]$ 的约束条件）

最后答案为 $dp[(1 << n) - 1]$，转移方程类似

```
int get(int x)
{
   int res = 0;
   while(x)
   {
      if(x & 1) res ++ ;
      x >>= 1;
   }
   return res;
}
vector<pair<int , int>>vec[19];
long long dp[1 << 18];
int cnt[1 << 18];
signed main()
{
```

```
    int n , m , x , y , z;
    cin >> n >> m;
    for(int i = 1 ; i <= m ; i ++)
    {
        cin >> x >> y >> z;
        vec[x].push_back(make_pair(y , z));
    }
    for(int bit = 1 ; bit < (1 << n) ; bit ++) cnt[bit] = get(bit);
    dp[0] = 1;
    for(int bit = 0 ; bit < (1 << n) ; bit ++)
    {
        vector<int>num;
        for(int k = 0 ; k < n ; k ++)
        {
            if(bit >> k & 1) num.push_back(k + 1);
        }
        int flag = 0;
        for(auto q : vec[cnt[bit]])
        {
            int y = q.first , z = q.second , sum = 0;
            for(auto h : num)
            {
                if(h <= y) sum ++ ;
            }
            if(sum > z) { flag = 1 ; break ; }
        }
        if(flag) continue ;
        for(auto k : num) dp[bit] += dp[bit - (1 << (k - 1))];
    }
    cout << dp[(1 << n) - 1] << '\n';
    return 0;
}
```

9.2 等差子序列

9.2.1 总结

- **问题：**
  给出一个长度为 N 的正整数序列
  问是否存在一个长度不小于三的等差子序列（值域为 $1e4$）

- **思路：**
  首先，找大于 3 个的和找 3 个的没区别。
  设置两个权值 $bool$ 数组 pre, suf，假设当前时刻为 now

  若 $pre_x = 0$ 表示从 $1 \sim now$，x 这个数并未出现过，反之表示 x 这个数出现过，
  若 $suf_x = 0$ 表示从 $now \sim n$，x 这个数并未出现过，反之表示 x 这个数出现过
  那么此时以 a_{now} 为对称中心，以 n 或者 1 为边界的对称区域
  若 $pre_i = 1$ 并且 $suf_j = 1$ 并且 i、j 关于 a_{now} 对称，则说明 $2 \times a_{now} = i + j$

  ----> 问题转换成功

- **解法 1：**
  直接上暴力，复杂度 $O(n^2)$

- **解法 2：**
  用 *bitset* 优化暴力，复杂度 $O(\frac{n^2}{32})$
- **解法 3：**
  可以用权值线段树 $+hash$ 来搞（序列必须为排列，思路和前面不一样了）
  设当前枚举的中点为 x，那么只要判断 $x-len\sim x$ 和 $x\sim x+len$ 的 $hash$ 值是否相同
  1. 若相同则说明可以和 x 构成等差的项都在前面出现过了。
  2. 若不相同则说明有一组的一项在前面出现了，而另一项在前面没有出现过。因为序列是一个 $1\sim n$ 的排列，所以另一项肯定在后面出现了。
- **解法 4：**
  分块 $+NTT$ （块的大小差不多为 2000）
  还可以求出有多少对 i,j,k 使得 $a_i+a_k=2\times a_j$

```cpp
// bitset
#define rep(i , a , b) for(int i = a ; i <= b ; i ++)
using namespace std;
const int N = 2e4 + 10;
int a[N] , cnt[N];
bitset<N>pre , suf , x , y;
signed main()
{
   ios::sync_with_stdio(false);
   cin.tie(0) , cout.tie(0);
   int T = 1;
   cin >> T;
   while(T --)
   {
      memset(cnt , 0 , sizeof(cnt));
      int n , flag = 0;
      pre.reset() , suf.reset();
      cin >> n;
      rep(i , 1 , n) cin >> a[i] , suf.set(a[i]) , cnt[a[i]] ++ ;
      rep(i , 1 , n)
      {
         cnt[a[i]] -- ;
         if(!cnt[a[i]]) suf.reset(a[i]);
         int pre_add = a[i] - 1 , suf_add = 20000 - a[i];
         int add = min(pre_add , suf_add);
         x = pre >> (20001 - a[i] - add);
         y = suf >> (a[i] - add);
         if((x & y).count()) flag = 1 , i = n;
         pre.set(20000 - a[i] + 1);
         // 个人习惯，我喜欢让 1 出现在最高位，2 出现在次高位。。。
      }
      if(flag) puts("Y");
      else puts("N");
   }
   return 0;
}
```

```cpp
// 权值线段树+hash
#define ull unsigned long long
const int P = 13331 , mod = 999998639;
const int N = 2e5 + 10;
ull power[N];
int a[N];
struct Seg_ment{
    int l , r ;
    ull pre , suf;
}tree[N << 2];
void push_up(int id , int len1 , int len2)
{
    tree[id].pre = tree[id << 1].pre * power[len2] + tree[id << 1 | 1].pre;
    tree[id].pre %= mod;
    tree[id].suf = tree[id << 1 | 1].suf * power[len1] + tree[id << 1].suf;
    tree[id].suf %= mod;
}
void build(int id , int l , int r)
{
    tree[id].pre = tree[id].suf = 0;
    tree[id].l = l , tree[id].r = r;
    if(l == r) return ;
    int mid = l + r >> 1;
    build(id << 1 , l , mid);
    build(id << 1 | 1 , mid + 1 , r);
    push_up(id , mid - tree[id].l + 1 , tree[id].r - mid);
}
void update(int id , int pos , int val)
{
    if(tree[id].l == tree[id].r)
    {
        tree[id].pre = tree[id].suf = val;
        return ;
    }
    int mid = tree[id].l + tree[id].r >> 1;
    if(pos <= mid) update(id << 1 , pos , val);
    else update(id << 1 | 1 , pos , val);
    push_up(id , mid - tree[id].l + 1 , tree[id].r - mid);
}
ull query_range(int id , int l , int r , int ch)
{
    if(tree[id].l >= l && tree[id].r <= r)
    {
        if(ch == 1) return tree[id].pre ;
        return tree[id].suf;
    }
    int mid = tree[id].l + tree[id].r >> 1;
    if(r <= mid) return query_range(id << 1 , l , r , ch);
    if(l > mid) return query_range(id << 1 | 1 , l , r , ch);
    ull lson = query_range(id << 1 , l , r , ch);
    ull rson = query_range(id << 1 | 1 , l , r , ch);
    ull ans ;
```

```
    if(ch == 1) ans = lson * power[min(r , tree[id].r) - mid] + rson;
    else ans = rson * power[mid - max(tree[id].l , l) + 1] + lson;
    return ans % mod;
}
void init()
{
    power[0] = 1;
    rep(i , 1 , 100000)
    power[i] = power[i - 1] * P % mod;
}
signed main()
{
    init();
    int t;
    cin >> t;
    while(t --)
    {
        int n , flag = 0 , x;
        cin >> n;
        build(1 , 1 , n);
        rep(i , 1 , n)
        {
            cin >> x;
            update(1 , x , 1);
            if(i == 1 || i == n) continue ;
            int len = min(x , n - x + 1);
            int y = query_range(1 , x - len + 1 , x , 1);
            int z = query_range(1 , x , x + len - 1 , 2);
            if(y != z) flag = 1 ;
        }
        if(flag) cout << "Y\n";
        else cout << "N\n";
    }
    return 0;
}
```

```
//NTT+分块
#define int long long
#define M 100009
using namespace std;
int read()
{
    int f = 1, re = 0;
    char ch;
    for (ch = getchar(); !isdigit(ch) && ch != '-'; ch = getchar())
    ;
    if (ch == '-')
    {
        f = -1, ch = getchar();
    }
    for (; isdigit(ch); ch = getchar())
    re = (re << 3) + (re << 1) + ch - '0';
    return re * f;
```

```
}
const int g = 3;
const int mod = 998244353;
int rev[M];
int ksm(int a, int b)
{
    int ans = 1;
    while (b)
    {
        if (b & 1) ans = (ll)ans * a % mod;
        a = (ll)a * a % mod;
        b >>= 1;
    }
    return ans % mod;
}
void ntt(int *A, int lim, int type)
{
    for (int i = 0; i < lim; i++) if (i < rev[i]) swap(A[i], A[rev[i]]);
    for (int mid = 1; mid < lim; mid <<= 1)
    {
        int W = ksm(g, (mod - 1) / (mid << 1));
        for (int R = mid << 1, j = 0; j < lim; j += R)
        {
            int w = 1;
            for (ll k = 0; k < mid; k++, w = (ll)w * W % mod)
            {
                int x = A[j + k], y = (ll)w * A[j + k + mid] % mod;
                A[j + k] = (x + y) % mod;
                A[j + mid + k] = (x - y + mod) % mod;
            }
        }
    }
    if (type == -1)
    {
        reverse(A + 1, A + lim);
        int inv = ksm(lim, mod - 2);
        for (int i = 0; i < lim; i++) A[i] = (ll)A[i] * inv % mod;
    }
}
int a[M], b[M], c[M], n, maxn, ans, num, L[M], R[M], l[M], r[M], block;
signed main()
{
    n = read();
    block = 2000;
    for (int i = 1; i <= n; i++) a[i] = read(), maxn = max(maxn, a[i]), r[a[i]]++;
    int lim = 1, lll = 0;
    while (lim < maxn * 2) lim <<= 1, lll++;
    for (int i = 0; i < lim; i++) rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (lll -
        1));
    num = n / block;
    if (n % block) num++;
    for (int i = 1; i <= num; i++) L[i] = (i - 1) * block + 1, R[i] = i * block;
    R[num] = n;
```

```
70      for (int i = 1; i <= num; i++)
71      {
72          for (int j = L[i]; j <= R[i]; j++)   r[a[j]]--;
73          for (int j = 0; j < lim; j++)    b[j] = c[j] = 0;
74          for (int j = 0; j <= maxn; j++)   b[j] = l[j], c[j] = r[j];
75          ntt(b, lim, 1), ntt(c, lim, 1);
76          for (int j = 0; j < lim; j++)    b[j] = (ll)b[j] * c[j];
77          ntt(b, lim, -1);
78          for (int j = L[i]; j <= R[i]; j++)   ans += (ll)b[2 * a[j]];
79          for (int j = L[i]; j <= R[i]; j++)
80          {
81              for (int k = L[i]; k < j; k++) if (2 * a[j] - a[k] > 0) ans += (ll)r[2 * a
                    [j] - a[k]];
82              for (int k = j + 1; k <= R[i]; k++) if (2 * a[j] - a[k] > 0) ans += (ll)l
                    [2 * a[j] - a[k]];
83              l[a[j]]++;
84          }
85      }
86      printf("%lld\n", ans);
87      return 0;
88  }
```

9.3 回文问题

9.3.1 寻找回文路径（AT）

problem:

给定 N 个点 M 条边，每条边都对应一个小写字母

问是否存在一条从 1 到 N 的路径，使得路径上的字母构成的字符串为回文串

- 若存在则输出回文串的最短长度，若不存在则输出 -1

solve

考虑双向 $bfs+dp$ （以 dp 来思考会好理解许多）

从 $1\sim n$ 的路径构成回文相当于:

- 从 $1\sim i$ 的路径和从 $i\sim n$ 的路径构成回文

或

- 从 $1\sim i$ 的路径和从 $j\sim n$ 的路径构成回文，其中 i j 相邻

于是可以定义 $dp[i][j]$ 来判断从 1 到 i 的路径和从 n 到 j 的路径是否构成回文

- $dp[i][j]=1$ 表示构成回文串
- $dp[i][j]=0$ 表示不构成回文串

有点类似从两头一起出发往中间靠的区间 dp?

起初一头在 1 号点，另一头在 N 号点

起初 1~1 之间没有路径，N~N 之间也没有路径

所以初始化 $dp[1][n]=1$，并把 $(1,n)$ 打包存入队列进行双向 bfs

在 bfs 的过程中:

- 当 $i = j$ 或 $i\ j$ 相邻时更新答案
- 当 i 的相邻边和 j 的相邻边相同时，定义 i 的相邻节点为 ii，j 的相邻节点为 jj，那么可由 $dp[i][j] = 1$ 推出 $dp[ii][jj] = 1$，然后把 (ii, jj) 打包存入队列继续 bfs

```
const int N = 1e3 + 10;
int n , m , mp[N][N] , dp[N][N];
vector<int>G[N][30];
signed main()
{
    cin >> n >> m ;
    for(int i = 1 ; i <= m ; i ++)
    {
        int u , v;
        char ch;
        cin >> u >> v >> ch;
        G[u][ch - 'a'].push_back(v);
        G[v][ch - 'a'].push_back(u);
        mp[u][v] = mp[v][u] = 1;
    }
    int res = 1e9;
    queue<pair<pair<int , int> , int>>que;
    que.push(make_pair(make_pair(1 , n) , 0));
    dp[1][n] = 1;
    while(!que.empty())
    {
        pair<pair<int , int> , int>q = que.front();
        que.pop();
        int x = q.fi.fi , y = q.fi.se , z = q.se;
        if(z > res) break ;
        if(x == y) res = min(res , z);
        if(mp[x][y]) res = min(res , z + 1);
        for(int k = 0 ; k <= 25 ; k ++)
        {
            for(auto i : G[x][k])
            {
                for(auto j : G[y][k])
                {
                    if(dp[i][j]) continue ;
                    dp[i][j] = 1;
                    que.push(make_pair(make_pair(i , j) , z + 2));
                }
            }
        }
    }
    if(res == 1e9) res = -1;
    cout << res << '\n';
    return 0;
}
```

9.4 排列问题

9.4.1 字典序最小的（排列 & 子序列）

题意：

- 给定一个长度为 N 的序列 A 和 K
- 满足 $\forall i \in [1,N], A[i] <= K$
- 现要求选择一个长度为 K 的子序列
- 使得这个子序列是一个排列且字典序最小

思路：

我们按顺序从前往后在这 N 个数之中选 K 个数出来

一个自然的想法：

每次尽量选取字典序最小的数，让较大的数在答案序列中尽量靠后

我们可以用类似单调队列的思想去实现:

用 $ans[]$ 存储答案，当枚举到第 i 个数 $A[i]$ 的时候，

若 $ans[]$ 之中没有 $A[i]$ 时, 将其与 $ans[]$ 的最后一位 B 做比较

如果 $A[i] < b$ 且 $A[i]$ 之后还有那么删去 B，接着比较，直到在 $ans[]$ 中找到不满足条件的数为止

此时插入 $A[i]$，枚举下一个数

```
rep(i , 1 , n)
{
  if(vis[a[i]]) continue ;
  while(deq.size() && a[deq.back()] > a[i] && last[a[deq.back()]] > i) vis[a[deq.
      back()]] = 0 , deq.pop_back();
  deq.pb(i);
  vis[a[i]] = 1;
}
for(auto i : deq) cout << a[i] << " ";
```

9.5 区间多次询问问题

9.5.1 从区间中选出两个数使得异或值最大

- N 是 $5e3$ 级别，可以区间 dp：$O(N^2)$
  $f[i][j] = max(f[i][j], f[i+1][j], f[i][j-1])$
- N 是 $5e4$ 级别，考虑分块：
  记 $b_{l,r}$ 表示第 l 块到第 r 块之间的答案
  这个可以用 $trie$ $O(n\sqrt{n}\log S)$ 搞出来，S 是值域

```
#define Int int
#define MAXN 100005
#define num bel[n]

```

```cpp
template <typename T> inline void read (T &t){t = 0;char c = getchar();int f = 1;while (c < '0' || c > '9'){if (c == '-') f = -f;c = getchar();}while (c >= '0' && c <= '9'){t = (t << 3) + (t << 1) + c - '0';c = getchar();} t *= f;}
template <typename T,typename ... Args> inline void read (T &t,Args&... args){read (t);read (args...);}
template <typename T> inline void write (T x){if (x < 0){x = -x;putchar ('-');}if (x > 9) write (x / 10);putchar (x % 10 + '0');}

int n,m,cur,rt[MAXN],val[MAXN],bel[MAXN];
int ch[MAXN * 32][2],cnt[MAXN * 32],Ans[705][705];

int sta[705],fin[705];

void Insert (int a,int b,int t,int x){
   if (t < 0) return ;
   int i = (x >> t) & 1;
   ch[a][i] = ++ cur;
   ch[a][i ^ 1] = ch[b][i ^ 1];
   cnt[ch[a][i]] = cnt[ch[b][i]] + 1;
   Insert (ch[a][i],ch[b][i],t - 1,x);
}

int query (int a,int b,int t,int x){
   if (t < 0) return 0;
   int i = (x >> t) & 1;
   if (cnt[ch[a][i ^ 1]] > cnt[ch[b][i ^ 1]]) return (1 << t) + query (ch[a][i ^ 1],ch[b][i ^ 1],t - 1,x);
   else return query (ch[a][i],ch[b][i],t - 1,x);
}

signed main(){
   read (n,m);
   int siz = sqrt (n);
   memset (sta,0x7f,sizeof (sta));
   for (Int i = 1;i <= n;++ i) read (val[i]),bel[i] = (i - 1) / siz + 1;
   for (Int i = 1;i <= n;++ i) sta[bel[i]] = min (sta[bel[i]],i),fin[bel[i]] = i;
   for (Int i = 1;i <= n;++ i) rt[i] = ++ cur,Insert (rt[i],rt[i - 1],17,val[i]);
   for (Int i = 1;i <= num;++ i){
      int start = sta[i];
      for (Int j = i;j <= num;++ j){
         Ans[i][j] = Ans[i][j - 1];
         for (Int k = sta[j];k <= fin[j];++ k)
            Ans[i][j] = max (Ans[i][j],query (rt[k],rt[start - 1],17,val[k]));
      }
   }
   while (m --){
      int l,r;
      read (l,r);
      if (bel[r] - bel[l] <= 2){
         int ans = 0;
         for (Int i = l;i < r;++ i)
            ans = max (ans,query (rt[r],rt[i],17,val[i]));
         write (ans),putchar ('\n');
```

```
53        }
54        else{
55            int bl = bel[l] + 1,br = bel[r] - 1;
56            int ans = Ans[bl][br];
57            for (Int i = l;i < sta[bl];++ i) ans = max (ans,query (rt[r],rt[l - 1],17,
                  val[i]));
58            for (Int i = fin[br] + 1;i <= r;++ i) ans = max (ans,query (rt[i],rt[l -
                  1],17,val[i]));
59            write (ans),putchar ('\n');
60        }
61    }
62    return 0;
63 }
```

9.6 数论只会 gcd？

9.6.1 求给定范围内 gcd 为素数的数对有多少？

给定正整数 n，求 $1 \le x, y \le n$ 且 $\gcd(x, y)$ 为素数的数对 (x, y) 有多少对

- $\sum_{p\in \text{ prime}} \sum_{i=1}^{n} \sum_{j=1}^{n} [\gcd(i, j) = p]$
- 对 gcd 进行套路式的变形：$\sum_{p\in \text{ prime}} \sum_{i=1}^{\lfloor \frac{n}{p} \rfloor} \sum_{j=1}^{\lfloor \frac{n}{p} \rfloor} [\gcd(i, j) = 1]$
- $\sum_{p\in \text{ prime}} \left(\sum_{i=1}^{\lfloor \frac{n}{p} \rfloor} \left(2 \sum_{j=1}^{i} [\gcd(i, j) = 1] \right) - 1 \right)$，其中 -1 的原因是 $i = j = 1$ 时的答案会被重复统计
- $\sum_{p\in \text{ prime}} \left(2 \sum_{i=1}^{\lfloor \frac{n}{p} \rfloor} \varphi(i) - 1 \right)$
- $\sum_{p\in \text{ prime}} \left(2 \times sum \left[\frac{n}{p} \right] - 1 \right)$

```
1  const int N = 1e7 + 10;
2  int n , prime[N] , minprime[N] , phi[N];
3  ll sum[N];
4  int euler(int n)
5  {
6     int c=0,i,j;
7     phi[1]=1;
8     for(i=2; i<=n; i++)
9     {
10       if(!minprime[i])prime[++c]=i,minprime[i]=i,phi[i]=i-1;
11       for(j=1; j<=c&&i*prime[j]<=n; j++)
12       {
13          minprime[i*prime[j]]=prime[j];
14          if(i%prime[j]==0)
15          {
16             phi[i*prime[j]]=phi[i]*prime[j];
17             break;
18          }
19          else phi[i*prime[j]]=phi[i]*(prime[j]-1);
20       }
21    }
```

```
    for(int i = 1 ; i <= n ; i ++) sum[i] = sum[i - 1] + phi[i];
    return c;
}
signed main()
{
    ll res = 0;
    cin >> n;
    int cnt = euler(n);
    for(int i = 1 ; i <= cnt ; i ++) res += 2 * sum[n / prime[i]] - 1;
    cout << res << '\n';
    return 0;
}
```

9.6.2 求给定范围内 gcd 为素数的数对有多少？

给定 N, M，求 $1 \le x \le N$，$1 \le y \le M$ 且 $\gcd(x, y)$ 为质数的 (x, y) 有多少对。

莫比乌斯反演 + 整除分块

```
const int N = 1e7 + 10;
int vis[N] , p[N] , pn , mu[N] , sum[N];
void init(int n)
{
    mu[1] = 1;
    rep(i , 2 , n)
    {
        if(!vis[i])
        {
            p[++ pn] = i;
            mu[i] = -1;
        }
        for(int j = 1 ; j <= pn && i * p[j] <= n ; j ++)
        {
            vis[i * p[j]] = 1;
            if(i % p[j] == 0)
            {
                mu[i * p[j]] = 0;
                break;
            }
            else mu[i * p[j]] =- mu[i];
        }
    }
    rep(i , 1 ,pn) for(int j = p[i] ; j <= n ; j += p[i]) sum[j] += mu[j / p[i]];
    rep(i , 1 , n) sum[i] += sum[i - 1];
}
ll solve(int n , int m)
{
    int mi = min(n , m);
    ll ans = 0;
    for(int l = 1 , r = 0 ; l <= mi ; l = r + 1)
    {
        r = min(n / (n / l) , m / (m / l));
        ans += 1ll * (n / l) * (m / l) * (sum[r] - sum[l - 1]);
    }
```

```
    return ans;
}
signed main()
{
    init(10000000);
    int T = 1 , n , m;
    cin >> T;
    while(T --)
    {
        cin >> n >> m;
        cout << solve(n , m) << '\n';
    }
    return 0;
}
```

9.6.3 单点乘法取模 + 整体 gcd

头脑风暴/数论只会 gcd? /3. 单点乘法取模 + 整体 gcd.tex

9.6.4 给定 c,d,x 求满足 c×lcm(a,b) - d×gcd(a,b) = x 的 (a,b) 的对数

$c \times lcm(a,b) - d \times gcd(a,b) = x$
$c \times \dfrac{a \times b}{gcd(a,b)} - d \times gcd(a,b) = x$
$c \times a \times b - d \times gcd(a,b)^2 = x \times gcd(a,b)$
$c \times a \times b = x \times gcd(a,b) + d \times gcd(a,b)^2$
$c \times \dfrac{a}{gcd(a,b)} \times \dfrac{b}{gcd(a,b)} = \dfrac{x}{gcd(a,b)} + d$
因为 $c \times \dfrac{a}{gcd(a,b)} \times \dfrac{b}{gcd(a,b)}$ 为整数，d 为整数
所以 $\dfrac{x}{gcd(a,b)}$ 为整数，所以 $gcd(a,b)$ 为 x 的因子
于是 $gcd(a,b)$ 我们可以在 $\sqrt{x}$ 的复杂度内枚举
定义 $A = \dfrac{a}{gcd(a,b)}, B = \dfrac{b}{gcd(a,b)}$
那么有 $gcd(A,B) = 1$ ， $A \times B = \dfrac{\dfrac{x}{gcd(a,b)} + d}{c}$
定义 $R = \dfrac{x}{gcd(a,b)} + d$
因为 $A \times B$ 为整数，所以 R 要满足 $R \bmod c = 0$
于是问题就转换成有多少对 A, B 满足 $\begin{cases} \gcd(A,B) = 1 \\ A \times B = R \end{cases}$
对于 $A \times B = R$，只要将 R 的质因子分配给 A 或 B 即可
对于 $gcd(A,B) = 1$，只要将 R 的某个质因子全部分给 A 或者全部分给 B 即可
那么 R 的贡献就是 $2^{p_{cnt}}$ ，其中 p_{cnt} 表示 R 的质因子个数

```
const int M = 2e7 + 10;
int cnt[M];
signed main()
{
    int T , up = 2e7;
    for(int i = 2 ; i <= up ; i ++)
    {
        if(cnt[i]) continue ;
```

```
9         for(int j = i ; j <= up ; j += i) cnt[j] ++ ;
10    }
11    cin >> T;
12    while(T --)
13    {
14        int c , d , x , res = 0;
15        cin >> c >> d >> x;
16        for(int i = 1 ; i * i <= x ; i ++)
17        {
18            if(x % i == 0)
19            {
20                int R = x / i + d;
21                if(R % c == 0)
22                {
23                    R /= c;
24                    res += 1ll << cnt[R];
25                }
26                int j = x / i;
27                if(j == i) continue ;
28                R = x / j + d;
29                if(R % c == 0)
30                {
31                    R /= c;
32                    res += 1ll << cnt[R];
33                }
34            }
35        }
36        cout << res << '\n';
37    }
38    return 0;
39 }
```

9.7 以子串为单位的倍增思想

9.7.1 以子串为单位的倍增思想

题目大意

给定一个长度为 N 的序列 A 和一个常数 K

有 M 次询问

每次询问查询一个区间 $[L, R]$ 内所有数最少分成多少个连续段

使得每段的和都 $<= K$ ，若无解则输出"$Chtholly$"

解题思路

简单回忆一下倍增求 LCA 思想：

- $f[i][j]$ 表示以 i 为起点，往上跳 $i + 2^j$ 步后得到的祖先
- 因为往上跳 2^j 等价于先往上跳 2^{j-1} 步后再往上跳 2^{j-1} 步
- 所以可得：$f[i][j] = f[f[i][j-1]][j-1]$

回到这道题：

暴力的做法即遍历区间 $[l,r]$ ，贪心的让每段的长度尽可能大

考虑用倍增思想优化：

定义 $f[i][j]$ 表示：

以 i 为起点，分成 2^j 个连续段后，所能到达的最远位置的下一个位置（其中每个段的和都不超过 K）

那么不难得到：$f[i][j]=f[f[i][j-1]][j-1]$ （$f[i][0]$ 可通过二分前缀和得到

然后对于每个询问 (L,R)：

先判断区间 $[L,R]$ 是否存在 A_i 使得 $A_i>K$

这步我们维护一个权值数组的前缀和 $O1$ 判断

即当 $A_i<=K$ 时，$sum[i]=sum[i-1]$

当 $A_i>K$ 时，$sum[i]=sum[i-1]+1$

当 $sum[R]-sum[L-1]>0$ 则表示该区间存在 $A_i>K$，直接输出 $Chtholly$

若 $sum[R]-sum[L-1]=0$ 则从高位往低位枚举 j：

> 如果 $f[L][j]>R$ 则表示从 L 开始划分出 2^j 个连续段是 OK 的
> 但是 2^j 连续段可能太多了（题目要求划分的连续段个数最少
> 所以就继续往下枚举
>
> 如果 $f[L][j]<R$，则表示从 L 开始划分出 2^j 个连续段是不够的
> 那就先划分出 2^j 个连续段，然后再从 $f[L][j]$ 的位置继续划分
> 即 $ans+=1<<j$ ， $L=f[L][j]$

```
const int N = 1e6 + 10;
int f[N][22];
int n , m , k , a[N] , sum[N];
long long pre[N];
signed main()
{
    cin >> n >> m >> k;
    for(int i = 1 ; i <= n ; i ++)
    {
        cin >> a[i] , pre[i] = pre[i - 1] + a[i];
        sum[i] = sum[i - 1] + (a[i] > k);
    }
    for(int j = 0 ; j <= 21 ; j ++) f[n + 1][j] = n + 1;

    for(int j = 0 ; j <= 21 ; j ++)
    {
        for(int i = 1 ; i <= n ; i ++)
        {
            f[i][0] = upper_bound(pre + i , pre + 1 + n , k - a[i] + pre[i]) -
                pre;
            if(!j) continue ;
            f[i][j] = f[f[i][j - 1]][j - 1];
        }
    }
    while(m --)
```

```
25          {
26              int l , r , ans = 0;
27              cin >> l >> r;
28              if(sum[r] - sum[l - 1])
29              {
30                  cout << "Chtholly\n";
31                  continue ;
32              }
33              for(int j = 21 ; j >= 0 ; j --)
34              {
35                  if(f[l][j] - 1 < r)
36                  {
37                      ans += 1 << j;
38                      l = f[l][j];
39                  }
40              }
41              cout << ans + 1 << '\n';
42          }
43          return 0;
44      }
```

9.8 最大字段和问题

9.8.1 从序列中选出 K 个子串使得子串和最大

1. dp，复杂度 $O(N^2)$

   设 $f(i,j)$ 表示前 i 个选 j 段（i 不一定选）的最大价值。

   设 $g(i,j)$ 表示前 i 个选 j 段（i 一定要选）的最大价值。

   对于 g，讨论 $i-1$ 选或不选。如果 $i-1$ 选了，则 i 可以接上去，不用新增一段。

   $g(i,j)=max\{g(i-1,j),f(i-1,j-1)\}+a_i$

   对于 f，讨论 i 选或不选。

   $f(i,j)=max\{g(i,j),f(i-1,j)\}$

   这样就可以做到 $O(n^2)$ 的时间复杂度。因为第一维转移时只涉及到 i 和 $i-1$，所以可以把第一维省掉，空间复杂度 $O(n)$。

```
int n, m;
std::cin >> n >> m;
LL a[N];
for (int i = 1; i <= n; i++)
std::cin >> a[i];
LL f[N] = {0}, g[N] = {0};
for (int i = 1; i <= n; i++) {
   for (int j = m; j >= 1; j--) {
      g[j] = std::max(g[j], f[j-1]) + a[i];
      f[j] = std::max(g[j], f[j]);
   }
}
std::cout << f[m] << std::endl;
```

2. 贪心

   首先，可以发现，对于一段连续的正数或负数，要么全部选，要么全部不选。

所以，我们可以把连续的一段正数或负数缩成一个数。那么序列就变成了正负交替的。以下说明都针对缩完以后的序列。

设序列中正数的个数为 cnt，则对于 $m \geq cnt$ 的情况，最优解肯定是取所有正数。

考虑 $m = cnt - 1$ 的情况，此时我们需要从 $m = cnt$ 的情况中减少一段。

有两种方法：一种是舍弃一个正数，另一种是取一个负数，使两边的正数合并成一段。

怎么取最优？若舍弃正数 a，会损失 a 的价值。若取负数 a，会损失 $-a$ 的价值。

统一起来，就是若舍弃/取走数字 a，会损失 $|a|$ 的价值。这样，我们只要找绝对值最小的数舍弃/取走即可。

舍弃/取走一个数后，序列会变成什么样呢?

事实上，舍弃/取走一个数 a_i，相当于与两边的数合并，合并完的值是 $a_{i-1} + a_i + a_{i+1}$

例如 $1, -2, 3, -4, 5$，若舍弃 3，则序列变成 $1, -3, 5$；若取走 -4，则序列变成 $1, -2, 4$。

可以发现，若取绝对值最小的数，合并完以后的序列还是正负交替。于是我们可以用刚才的方法继续获得 $m = cnt - 2, cnt - 3, ...$ 的答案，直至 m 达到题目的要求。

取绝对值最小的数，可以用优先队列做。合并节点可以使用链表。答案为最后合并完的序列的所有正数之和。

复杂度：合并 $O(n)$ 次，每次 $O(\log n)$ ，总复杂度 $O(n \log n)$

```
struct Data {
    LL val;
    int pos, tim;

    bool operator < (const Data &t) const {
        return val > t.val;
    }
};
int pl[N], pr[N];
int tim[N] = {0};
void del(int u) {
    if (u == 0) return;
    pr[pl[u]] = pr[u];
    pl[pr[u]] = pl[u];
    tim[u] = -1;
}
int main() {
    std::ios::sync_with_stdio(false);
    int n, m;
    std::cin >> n >> m;
    static LL a[N] = {0};
    int top = 0;
    for (int i = 1; i <= n; i++) {
        LL r; std::cin >> r;
        if (r == 0) continue;
        if (top == 0) {
            if (r > 0) a[++top] = r;
            continue;
        }
        if (a[top] > 0 == r > 0) a[top] += r;
        else a[++top] = r;
    }
```

```
    if (top > 0 && a[top] < 0) top--;
    for (int i = 0; i <= top; i++) {
        pl[i] = i == 0 ? top : i - 1;
        pr[i] = i == top ? 0 : i + 1;
    }
    std::priority_queue<Data> q;
    for (int i = 1; i <= top; i++) {
        q.push({ std::abs(a[i]), i, 0 });
    }
    for (int cnt = top + 1 >> 1; cnt > m; cnt--) {
        Data d = q.top(); q.pop();
        while (tim[d.pos] != d.tim) {
            d = q.top(); q.pop();
        }
        int u = d.pos, l = pl[u], r = pr[u];
        a[u] += a[l] + a[r];
        if (l != 0 && r != 0)
        q.push({ std::abs(a[u]), u, ++tim[u] });
        else del(u);
        del(l); del(r);
    }
    LL ans = 0;
    for (int i = pr[0]; i != 0; i = pr[i])
    if (a[i] > 0) ans += a[i];
    std::cout << ans << std::endl;
    return 0;
}
```

10 杂项

10.1 对拍

10.1.1 ac.cpp

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    freopen("in","r",stdin);
    freopen("1.out","w",stdout);
    int a , b;
    cin >> a >> b;
    if(a == b) cout << "YES\n";
    else cout << "NO\n";
    return 0;
}
```

10.1.2 wa.cpp

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

```cpp
int main()
{
    freopen("in","r",stdin);
    freopen("2.out","w",stdout);
    int a , b;
    cin >> a >> b;
    if(a != b) cout << "YES\n";
    else cout << "NO\n";
    return 0;
}
```

10.1.3 data.cpp

```cpp
#include<bits/stdc++.h>
#define rep(i,a,b) for(int i=a;i<=b;i++)
#define int long long
#define debug(x) cout << #x << " : " << x << '\n';
using namespace std;
const int N = 3e5 + 10;
int random(int n)
{
    return (long long)(rand() * rand()) % n;
}
int far[N] , vis[N];
int find(int x)
{
    if(x == far[x]) return x;
    return far[x] = find(far[x]);
}
map<pair<int , int> , int>mp;
signed main()
{
    freopen("make.out","w",stdout);
    srand((unsigned int)(time(0)));

    int n = random(5) + 3 , m = random(20) + 5;
    cout << n << " " << m << '\n';

    return 0;
}
```

10.1.4 duipai.cpp

```cpp
#include<bits/stdc++.h>
using namespace std;
int main(){
    int c=0;
    do{
        if(c)printf("#%d AC\n",c);
        ++c;
        system("./make");
```

```
9         system("./1");
10        system("./2");
11    }while(!system("diff 1.out 2.out"));
12    printf("#%d WA\n",c);
13    return 0;
14 }
```

10.2 素数表

- 一千以内：
  $61, 83, 113, 151, 211, 281, 379, 509683, 911$
- 一万以下：
  $1217, 1627, 2179, 2909, 3881, 6907, 9209$
- 十万以下：
  $12281, 16381, 21841, 29123, 38833, 51787, 69061, 92083$
- 百万以下：
  $122777, 163729, 218357, 291143, 388211, 517619, 690163, 999983$
- 千万以下：$1226959, 1635947, 2181271, 2908361, 3877817, 5170427, 6893911, 9191891$
- 一亿以下：$12255871, 16341163, 21788233, 29050993, 38734667, 51646229, 68861641, 91815541$
- 十亿左右：
  $1e9+7$ 和 $1e9+9$
- 十亿以下：$122420729, 163227661, 217636919, 290182597, 386910137, 515880193, 687840301, 917120411$
- 十亿以上：
  $1222827239, 1610612741, 3221225473, 4294967291$

10.3 小 tricks

- 有 X 种连续的数，每种数有 Y 个，第一种为 L ，求第 K 小的数
  $ans = L + \frac{K-1}{Y}$
- 对于一个有限小数，当分数为最简形式时分母只包含 2 、5 两个质因子，于是我们可以用 $\lfloor \frac{bc}{ac} \rfloor$ 来表示任何一个有限小数 (其中 a 为 2,5 两个质因子构成的数, c 为不包含 2,5 两个质因子的数, b 随意)
- $\sum_{i=x}^{y} 2^i = 2^{y+1} - 2^x$
- 单调栈求以 x 为最大值的区间个数，若 x 有多个，可以考虑左闭右开
- 题目给出的数据范围可能很大，但题目存在某些限制（比如每个数都不相同），那么数据范围大了之后可能就不能满足条件，重新计算一下复杂度判断是否可以直接上暴力?

10.4 玄学优化

```
1 #pragma GCC target ("avx2,fma")
2 #pragma GCC optimize ("O3")
3 #pragma GCC optimize ("unroll-loops")
4 or
```

```
#pragma GCC optimize("Ofast,no-stack-protector,unroll-loops")
#pragma GCC target("avx")
```

10.5 头文件

```
#include <bits/stdc++.h>
#define ios std::ios::sync_with_stdio(false)
#define rep(i, a, n) for (int i = a; i <= n; i++)
#define per(i, n, a) for (int i = n; i >= a; i--)
#define repp(i, a, b) for (int i = a; i < b; i++)
#define mm(a, n) memset(a, n, sizeof(a))
#define pb push_back
#define all(x) (x).begin(), (x).end()
#define fi first
#define se second
#define il inline
#define int long long
#define ll long long
#define ull unsigned long long
#define MOD 1000000007
#define pi 3.14159265358979323
#define SZ(x) ((ll)(x).size())
#define debug(x) cout << #x << ": " << x << endl;
#define debug2(x, y) cout << #x << ": " << x << " | " << #y << ": " << y << endl;
#define debug3(x, y, z) cout << #x << ": " << x << " | " << #y << ": " << y << " | "
    << #z << ": " << z << endl;
#define debug4(a, b, c, d) cout << #a << ": " << a << " | " << #b << ": " << b << "
    | " << #c << ": " << c << " | " << #d << ": " << d << endl;
#define debug5(a, b, c, d, e) cout << #a << ": " << a << " | " << #b << ": " << b <<
    " | " << #c << ": " << c << " | " << #d << ": " << d << " | " << #e << ": " <<
    e << endl;
using namespace std;
const ll INF(0x3f3f3f3f3f3f3f3fll);
const int inf(0x3f3f3f3f);
template <typename T>
void read(T &res)
{
    bool flag = false;
    char ch;
    while (!isdigit(ch = getchar()))
    (ch == '-') && (flag = true);
    for (res = ch - 48; isdigit(ch = getchar()); res = (res << 1) + (res << 3) + ch
        - 48)
    ;
    flag && (res = -res);
}
template <typename T>
void Out(T x)
{
    if (x < 0)
    putchar('-'), x = -x;
    if (x > 9)
```

```
43      Out(x / 10);
44      putchar(x % 10 + '0');
45  }
46  ll gcd(ll a, ll b)
47  {
48      return b ? gcd(b, a % b) : a;
49  }
50  ll lcm(ll a, ll b)
51  {
52      return a * b / gcd(a, b);
53  }
54  ll pow_mod(ll x, ll n, ll mod)
55  {
56      ll res = 1;
57      while (n)
58      {
59          if (n & 1)
60          res = res * x % mod;
61          x = x * x % mod;
62          n >>= 1;
63      }
64      return res;
65  }
66  ll fact_pow(ll n, ll p)
67  {
68      ll res = 0;
69      while (n)
70      {
71          n /= p;
72          res += n;
73      }
74      return res;
75  }
76  ll mult(ll a, ll b, ll p)
77  {
78      a %= p;
79      b %= p;
80      ll r = 0, v = a;
81      while (b)
82      {
83          if (b & 1)
84          {
85              r += v;
86              if (r > p)
87              r -= p;
88          }
89          v <<= 1;
90          if (v > p)
91          v -= p;
92          b >>= 1;
93      }
94      return r;
95  }
```

```
96  ll quick_pow(ll a, ll b, ll p)
97  {
98      ll r = 1, v = a % p;
99      while (b)
100     {
101         if (b & 1)
102         r = mult(r, v, p);
103         v = mult(v, v, p);
104         b >>= 1;
105     }
106     return r;
107 }
108 bool CH(ll a, ll n, ll x, ll t)
109 {
110     ll r = quick_pow(a, x, n);
111     ll z = r;
112     for (ll i = 1; i <= t; i++)
113     {
114         r = mult(r, r, n);
115         if (r == 1 && z != 1 && z != n - 1)
116         return true;
117         z = r;
118     }
119     return r != 1;
120 }
121 bool Miller_Rabin(ll n)
122 {
123     if (n < 2)
124     return false;
125     if (n == 2)
126     return true;
127     if (!(n & 1))
128     return false;
129     ll x = n - 1, t = 0;
130     while (!(x & 1))
131     {
132         x >>= 1;
133         t++;
134     }
135     srand(time(NULL));
136     ll o = 8;
137     for (ll i = 0; i < o; i++)
138     {
139         ll a = rand() % (n - 1) + 1;
140         if (CH(a, n, x, t))
141         return false;
142     }
143     return true;
144 }
145 ll exgcd1(ll a, ll b, ll &x, ll &y)
146 {
147     if (!b)
148     {
```

```
149         x = 1, y = 0;
150         return a;
151     }
152     ll t = exgcd1(b, a % b, y, x);
153     y -= a / b * x;
154     return t;
155 }
156 ll get_inv(ll a, ll mod)
157 {
158     ll x, y;
159     ll d = exgcd1(a, mod, x, y);
160     return d == 1 ? (x % mod + mod) % mod : -1;
161 }
162 void exgcd(ll a, ll b, ll &x, ll &y)
163 {
164     if (!b)
165     {
166         x = 1, y = 0;
167         return;
168     }
169     exgcd(b, a % b, x, y);
170     ll t = x;
171     x = y, y = t - (a / b) * y;
172 }
173 ll INV(ll a, ll b)
174 {
175     ll x, y;
176     return exgcd(a, b, x, y), (x % b + b) % b;
177 }
178 ll crt(ll x, ll p, ll mod)
179 {
180     return INV(p / mod, mod) * (p / mod) * x;
181 }
182 ll FAC(ll x, ll a, ll b)
183 {
184     if (!x)
185     return 1;
186     ll ans = 1;
187     for (ll i = 1; i <= b; i++)
188     if (i % a)
189     ans *= i, ans %= b;
190     ans = pow_mod(ans, x / b, b);
191     for (ll i = 1; i <= x % b; i++)
192     if (i % a)
193     ans *= i, ans %= b;
194     return ans * FAC(x / a, a, b) % b;
195 }
196 ll C(ll n, ll m, ll a, ll b)
197 {
198     ll N = FAC(n, a, b), M = FAC(m, a, b), Z = FAC(n - m, a, b), sum = 0, i;
199     for (i = n; i; i = i / a)
200     sum += i / a;
201     for (i = m; i; i = i / a)
```

```
202     sum -= i / a;
203     for (i = n - m; i; i = i / a)
204     sum -= i / a;
205     return N * pow_mod(a, sum, b) % b * INV(M, b) % b * INV(Z, b) % b;
206 }
207 ll exlucas(ll n, ll m, ll p)
208 {
209     ll t = p, ans = 0, i;
210     for (i = 2; i * i <= p; i++)
211     {
212         ll k = 1;
213         while (t % i == 0)
214         {
215             k *= i, t /= i;
216         }
217         ans += crt(C(n, m, i, k), p, k), ans %= p;
218     }
219     if (t > 1)
220     ans += crt(C(n, m, t, t), p, t), ans %= p;
221     return ans % p;
222 }
223 int prime[100010], minprime[100010], phi[100010];
224 int euler(int n)
225 {
226     int c = 0, i, j;
227     phi[1] = 1;
228     for (i = 2; i <= n; i++)
229     {
230         if (!minprime[i])
231         prime[++c] = i, minprime[i] = i, phi[i] = i - 1;
232         for (j = 1; j <= c && i * prime[j] <= n; j++)
233         {
234             minprime[i * prime[j]] = prime[j];
235             if (i % prime[j] == 0)
236             {
237                 phi[i * prime[j]] = phi[i] * prime[j];
238                 break;
239             }
240             else
241             phi[i * prime[j]] = phi[i] * (prime[j] - 1);
242         }
243     }
244     return c;
245 }
246 struct Tree
247 {
248     ll l, r, sum, lazy, maxn, minn;
249 } tree[100000];
250 il void push_up(ll rt)
251 {
252     tree[rt].sum = tree[rt << 1].sum + tree[rt << 1 | 1].sum;
253     tree[rt].maxn = max(tree[rt << 1].maxn, tree[rt << 1 | 1].maxn);
254     tree[rt].minn = min(tree[rt << 1].minn, tree[rt << 1 | 1].minn);
```

```
}
il void push_down(ll rt, ll length)
{
    if (tree[rt].lazy)
    {
        tree[rt << 1].lazy += tree[rt].lazy;
        tree[rt << 1 | 1].lazy += tree[rt].lazy;
        tree[rt << 1].sum += (length - (length >> 1)) * tree[rt].lazy;
        tree[rt << 1 | 1].sum += (length >> 1) * tree[rt].lazy;
        tree[rt << 1].minn += tree[rt].lazy;
        tree[rt << 1 | 1].minn += tree[rt].lazy;
        tree[rt << 1].maxn += tree[rt].lazy;
        tree[rt << 1 | 1].maxn += tree[rt].lazy;
        tree[rt].lazy = 0;
    }
}
il void build(ll l, ll r, ll rt, ll *aa)
{
    tree[rt].lazy = 0;
    tree[rt].l = l;
    tree[rt].r = r;
    if (l == r)
    {
        tree[rt].sum = aa[l];
        tree[rt].minn = tree[rt].sum;
        tree[rt].maxn = tree[rt].sum;
        return;
    }
    ll mid = (l + r) >> 1;
    build(l, mid, rt << 1, aa);
    build(mid + 1, r, rt << 1 | 1, aa);
    push_up(rt);
}
il void update_range(ll L, ll R, ll key, ll rt)
{
    if (tree[rt].r < L || tree[rt].l > R)
    return;
    if (L <= tree[rt].l && R >= tree[rt].r)
    {
        tree[rt].sum += (tree[rt].r - tree[rt].l + 1) * key;
        tree[rt].minn += key;
        tree[rt].maxn += key;
        tree[rt].lazy += key;
        return;
    }
    push_down(rt, tree[rt].r - tree[rt].l + 1);
    ll mid = (tree[rt].r + tree[rt].l) >> 1;
    if (L <= mid)
    update_range(L, R, key, rt << 1);
    if (R > mid)
    update_range(L, R, key, rt << 1 | 1);
    push_up(rt);
}
```

```
il ll query_range(ll L, ll R, ll rt)
{
    if (L <= tree[rt].l && R >= tree[rt].r)
    {
        return tree[rt].sum;
    }
    push_down(rt, tree[rt].r - tree[rt].l + 1);
    ll mid = (tree[rt].r + tree[rt].l) >> 1;
    ll ans = 0;
    if (L <= mid)
    ans += query_range(L, R, rt << 1);
    if (R > mid)
    ans += query_range(L, R, rt << 1 | 1);
    return ans;
}
il ll query_min(ll L, ll R, ll rt)
{
    if (L <= tree[rt].l && R >= tree[rt].r)
    {
        return tree[rt].minn;
    }
    push_down(rt, tree[rt].r - tree[rt].l + 1);
    ll mid = (tree[rt].r + tree[rt].l) >> 1;
    ll ans = (0x3f3f3f3f3f3f3f3fll);
    if (L <= mid)
    ans = min(ans, query_min(L, R, rt << 1));
    if (R > mid)
    ans = min(ans, query_min(L, R, rt << 1 | 1));
    return ans;
}
il ll query_max(ll L, ll R, ll rt)
{
    if (L <= tree[rt].l && R >= tree[rt].r)
    {
        return tree[rt].maxn;
    }
    push_down(rt, tree[rt].r - tree[rt].l + 1);
    ll mid = (tree[rt].r + tree[rt].l) >> 1;
    ll ans = -(0x3f3f3f3f3f3f3f3fll);
    if (L <= mid)
    ans = max(ans, query_max(L, R, rt << 1));
    if (R > mid)
    ans = max(ans, query_max(L, R, rt << 1 | 1));
    return ans;
}
namespace linear_seq
{
    typedef vector<ll> VI;
    const ll N = 1e5 + 10, mod = 1e9 + 7;
    ll Rx[N], Ba[N], _c[N], _md[N];
    VI Md;
    ll PMD(ll a, ll b)
    {
```

```
361         ll Rx = 1;
362         a %= mod;
363         assert(b >= 0);
364         for (; b; b >>= 1)
365         {
366             if (b & 1)
367             Rx = Rx * a % mod;
368             a = a * a % mod;
369         }
370         return Rx;
371     }
372     void MUL(ll *a, ll *b, ll k)
373     {
374         repp(i, 0, k + k) _c[i] = 0;
375         repp(i, 0, k) if (a[i]) repp(j, 0, k) _c[i + j] = (_c[i + j] + a[i] * b[j]) %
                mod;
376         for (ll i = k + k - 1; i >= k; i--)
377         if (_c[i])
378         repp(j, 0, SZ(Md)) _c[i - k + Md[j]] = (_c[i - k + Md[j]] - _c[i] * _md[Md[j
                ]]) % mod;
379         repp(i, 0, k) a[i] = _c[i];
380     }
381     ll Solve(ll n, VI a, VI b)
382     {
383         ll ans = 0, pnt = 0, k = a.size();
384         assert(SZ(a) == SZ(b));
385         repp(i, 0, k) _md[k - 1 - i] = -a[i];
386         _md[k] = 1;
387         Md.clear();
388         repp(i, 0, k) if (_md[i] != 0) Md.push_back(i);
389         repp(i, 0, k) Rx[i] = Ba[i] = 0;
390         Rx[0] = 1;
391         while ((1ll << pnt) <= n)
392         pnt++;
393         for (ll p = pnt; p >= 0; p--)
394         {
395             MUL(Rx, Rx, k);
396             if ((n >> p) & 1)
397             {
398                 for (ll i = k - 1; i >= 0; i--)
399                 Rx[i + 1] = Rx[i];
400                 Rx[0] = 0;
401                 repp(j, 0, SZ(Md)) Rx[Md[j]] = (Rx[Md[j]] - Rx[k] * _md[Md[j]]) % mod;
402             }
403         }
404         repp(i, 0, k) ans = (ans + Rx[i] * b[i]) % mod;
405         if (ans < 0)
406         ans += mod;
407         return ans;
408     }
409     VI BM(VI s)
410     {
411         VI C(1, 1), B(1, 1);
```

```
412         ll L = 0, m = 1, b = 1;
413         repp(n, 0, SZ(s))
414         {
415             ll d = 0;
416             repp(i, 0, L + 1) d = (d + (ll)C[i] * s[n - i]) % mod;
417             if (d == 0)
418             ++m;
419             else if (2 * L <= n)
420             {
421                 VI T = C;
422                 ll c = mod - d * PMD(b, mod - 2) % mod;
423                 while (SZ(C) < SZ(B) + m)
424                 C.push_back(0);
425                 repp(i, 0, SZ(B)) C[i + m] = (C[i + m] + c * B[i]) % mod;
426                 L = n + 1 - L;
427                 B = T;
428                 b = d;
429                 m = 1;
430             }
431             else
432             {
433                 ll c = mod - d * PMD(b, mod - 2) % mod;
434                 while (SZ(C) < SZ(B) + m)
435                 C.push_back(0);
436                 repp(i, 0, SZ(B)) C[i + m] = (C[i + m] + c * B[i]) % mod;
437                 ++m;
438             }
439         }
440         return C;
441     }
442     ll Gao(VI a, ll n)
443     {
444         VI c = BM(a);
445         c.erase(c.begin());
446         n--;
447         repp(i, 0, SZ(c)) c[i] = (mod - c[i]) % mod;
448         return Solve(n, c, VI(a.begin(), a.begin() + SZ(c)));
449     }
450 };
451 const int N = 3e5 + 10;
452 struct Edge
453 {
454     int nex, to;
455 } edge[N << 1];
456 int head[N], TOT;
457 void add_edge(int u, int v)
458 {
459     edge[++TOT].nex = head[u];
460     edge[TOT].to = v;
461     head[u] = TOT;
462 }
463 signed main()
464 {
```

```
465     ios;
466
467     return 0;
468 }
```

11 赛前看一看

11.1 错误征集

1. 关于 **#define int long long**
    - 注意当题目某个类型为 **unsigned int** 时, 如果没有去掉 **#define int longl ong**，就会导致 **unsigned int** → **unsigned long long** 从而导致错误
    - **#define int long long** 后，**scanf** 和 **prinf** 需要改成 **%lld**
2. 题目中有**单个字母**读入和**数字**读入时，千万别用 $char - `0`$ 来代替数字，因为读入的数字可能**不是个位数**
3. 注意是不是只建了**单向边**
4. 题目是不是多组读入?
5. inf 是不是开得不够大或者太大了爆 $longlong$?
6. 路径压缩的 $find$ 和按秩合并的 $find$ 不同，别忘记修改!
7. 检查除 $void$ 类型的函数有没有**返回值**
8. 一个数右移过多位再 & 1 是无法得到准确值的 (int y = x » 65 & 1，此时无法确定 y 是等于 0 还是 1)
9. $reverse\ vector$ 注意判空不然会 re
10. 用位运算表示 2^n 注意加 $1LL << n$
11. $map.find$ 不会创建新元素，$map[]$ 会，注意空间
12. 涉及减法的时候取模时候要 $(+mod)\%mod$

11.2 运行结果和自己预期的不一样？

1. 可能是数组越界了?
2. 数字用字符串读入没加引号?
3. 函数没写返回值?
4. 对字符串 $auto$ 的时候忘记转换成数字了?
5. 路径压缩和按秩合并的 $find$ 函数不同，忘记改了?
6. == 写成了 = ?
7. $init$ 忘记写了?