1 约定与注意

1.1 缺省源

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

#define debug(x) cerr << #x << ": " << (x) << endl
#define rep(i,a,b) for(int i=(a);i<=(b);i++)
#define dwn(i,a,b) for(int i=(a);i>=(b);i--)
#define pb push_back
#define all(x) (x).begin(), (x).end()

#define x first
#define y second
using pii = pair<int, int>;
using ll = long long;

inline void read(int &x){
    int s=0; x=1;
    char ch=getchar();
    while(ch<'0' || ch>'9') {if(ch=='-')x=-1;ch=getchar();}
    while(ch>='0' && ch<='9') s=(s<<3)+(s<<1)+ch-'0',ch=getchar();
    x*=s;
}
```

1.2 一些可能的幼稚错误

```
const int eps=1e-6;
```

不是多测但是 cin>>cs 了。

函数返回类型为 int 但是没返回东西。

1.3 防止 uset 被卡

```
struct seed{
  static uint64_t splitmix64(uint64_t x){
    x ^= x << 13;
    x ^= x >> 7;
    x ^= x << 17;
    return x;
  }
  size_t operator () (uint64_t x) const{
    static const uint64_t FIXED_RANDOM =
chrono::steady_clock::now().time_since_epoch().count();
    return splitmix64(x + FIXED_RANDOM);
  }
};

unordered_set<int, seed> st;
```

1.4 Python 的一些操作

1.4.1 高精度开根

https://www.luogu.com.cn/problem/P2293

```
import sys

sys.set_int_max_str_digits(10005)

rt=int(input())
val=int(input())

l=0
r=1

while r**rt<=val:
  l=r; r<<=1

def fpow(x, p):
  res=1
  while p>0:
    if p&1==1:
      res=res*x
    p>>=1
    x*=x
  return res


while l<r:
  mid=l+r+1>>1
  if fpow(mid, rt)<=val: l=mid;
  else: r=mid-1;

print(l)
```

2 组合

2.1 卡特兰数

长度为 $2n$ 的合法括号序列个数。

$$C_{2n}^{n}-C_{2n}^{n-1}$$

2.2 斯特林数

- 第一类斯特林数 $s(n,k)$：长度为 n 的排列中出现 k 个环的方案数。

$$s(n,k)=s(n-1,k-1)+(n-1)s(n-1,k)$$

- 第二类斯特林数 $S(n,k)$：长度为 n 的排列中出现 k 个集合的方案数。

$$S(n,k)=S(n-1,k-1)+kS(n-1,k)$$

2.3 贝尔数

B_n 是基数为 n 的集合的划分方法的数目。

且定义 $B_0=1$。

贝尔数适合递推公式：

$$B_{n+1}=\sum_{k=0}^{n}C_n^kB_k$$

2.4 Cayley 公式

n 个点带标号无根树个数为 n^{n-2}。

n 个节点度数依次为 $D_1,D_2,...,D_n$ 的无根树数量为

$$\frac{(n-2)!}{\prod_{i=1}^{n}(D_i-1)!}$$

2.5 Burnside 引理 & Pólya 定理

2.5.1 问题引入

> 给定集合 X 和置换群 G。
>
> 代入下面的情境（意义）对下文的内容进行理解：
>
> A 表示待染色物品集合。
>
> B 表示支持染的颜色的集合。
>
> X 表示染色方案集合。
>
> G 表示支持的变换。
>
> X/G 表示本质不同的染色方案集合。
>
> X^g 表示经过一个变换 g 后保持不变的染色方案对应的集合。
>
> 比如说：给你一串共 n 个珠子，支持旋转变换 G（可以看作是置换的一种）（这意味着如果两种染色方案在旋转后一样视为本质相同），每个珠子可以被染成 m 种颜色（也就是说方案集 X 的大小为 m^n），求本质不同的染色方案数（也就是 $|X/G|$）

2.5.2 Burnside 引理：

$$|X/G|=\frac{1}{|G|}\sum_{g\in G}|X^g|$$

这意味着，对于实际的计数问题，在给出变换的种类数 $|G|$ 后，我们只需要求出染色方案在各种变换下保持不动的数量的和（即 $\sum_{g\in G}|X^g|$），那么我们就可以求出本质不同的染色方案数了。

2.5.3 Pólya 定理

考虑到 Burnside 引理需要求的 $\sum_{g\in G}|X^g|$ 在实际统计中时间复杂度较高，而很多问题都是求解 $\boldsymbol{A\to B}$ 所有可能的映射（也就是比较特殊的 X）所对应的染色方案（共 $|B|^{|A|}$ 种）中本质不同的数量。

那么，在这样的问题中，可以使用 **Pólya** 定理：

$$|X/G|=\frac{1}{|G|}\sum_{g\in G}|B|^{c(g)}$$

$c(g)$ 表示置换 g 能拆分出的循环置换数。

2.5.4 牛顿恒等式

参考：https://zh.wikipedia.org/zh-cn/%E7%89%9B%E9%A0%93%E6%81%86%E7%AD%89%E5%BC%8F

这东西是我做：https://hydro.ac/p/bzoj-P3771 的时候遇到的。

题意是给你一个序列 A，求 $k\in\{1,2,3\}$ 时，所有可能的 x（也就是满足 $C_{k(x)}>0$ 的 x）的 $\sum C_{k(x)}$ 值，其中

$$C_{k(x)}=|\left\{I\subseteq\{1,2,...,n\}\mid |I|=k,\sum_{\{i\in I\}}a_i=x\right\}|,$$

即求恰好选中 k 个不考虑顺序的物品，且价值和为 x 的方案数。

```
// 使用的模板见下面的 *FFT 相关* 部分。
void solve(){
  int n; cin>>n;
  Poly f(N), f2(N), f3(N);
  rep(i, 1, n){
    int x; read(x);
    ++f[x];
    ++f2[x*2];
    ++f3[x*3];
  }
  auto res=f; // k=1
  auto ff=f*f, fff=f*f*f;
  res+=(ff-f2)/2;
  res+=(fff-f*f2*3+f3*2)/6;

  vector<pii> ans;
  rep(i, 0, (int)res.size()-1){
    if(res[i]) ans.pb({i, round(res[i])});
  }
  for(auto &[x, y]: ans) cout<<x<<" "<<y<<"\n";
}
```

本题要求的是 e_1,e_2,e_3。

$$e_1=p_1,$$

$$e_2=\frac{p_1^2-p_2}{2}$$

$$e_3=\frac{p_1^3-3p_1p_2+2p_3}{6}$$

推广来说，有下面恒等式：$ke_k=\sum_{i=1}^{k}(-1)^{i-1}e_{k-i}p_i$。（注意这里等式中 e_k 前有系数 k）。

3 数论

3.1 线性筛

算法核心是每个数只会被最小的质因数计算一次。可以用来求 φ,μ,d 等函数。

- 莫比乌斯函数 $\mu(n)$：如果整数 n 是平方数的倍数（含有某素数的平方因子），则 $\mu(n)=0$；否则，若 n 由 k 个不同素数组成，其值为 $(-1)^k$。

- 欧拉函数 φ(n)：计算小于等于 n 且与 n 互质的正整数个数。

```
int cnt, p[N];
bool vis[N]; // vis[x] 表示 x 为合数。
int phi[N]; // 求欧拉函数。
int mu[N]; // 求莫比乌斯函数。
int d[N], num[N]; // 求约数个数，d[x] 为 x 约数个数，num[x] 为 x 的最小质因子出现次数。

void sieve(int n){
  phi[1]=1;
  mu[1]=1;
  d[1]=1;
    for(int i=2; i<=n; i++){
        if(!vis[i]){
          p[cnt++]=i;
          phi[i]=i-1;
          mu[i]=-1;
          num[i]=1;
          d[i]=2;
        }
        for(int j=0; i*p[j]<=n; j++){
          vis[i*p[j]]=true;
          if(i%p[j]==0){
            phi[i*p[j]]=phi[i]*p[j];
            mu[i*p[j]]=0;
            num[i*p[j]]=num[i]+1;
            d[i*p[j]]=d[i]/num[i*p[j]]*(num[i*p[j]]+1);
            break;
          }
          phi[i*p[j]]=phi[i]*phi[p[j]];
          mu[i*p[j]]=-mu[i];
          num[i*p[j]]=1;
          d[i*p[j]]=d[i]*2;
        }
    }
}
```

3.2 组合数及快速幂

```
#define int long long

int fpow(int x, int p){
  int res=1;
  for(; p; p>>=1, x=1LL*x*x%mod) if(p&1) res=1LL*res*x%mod;
  return res%mod;
}

int inv(int x){
  return fpow(x, mod-2);
}

int fac[N];
int invfac[N];

void getFac(int n){
  fac[0]=invfac[0]=1;
  for(int i=1; i<=n; i++) fac[i]=1LL*fac[i-1]*i%mod,
invfac[i]=1LL*invfac[i-1]*inv(i)%mod;
}

int C(int a, int b){
  if(a<0 || b<0) return 0;
  if(b>a) return 0;
  return 1LL*fac[a]*invfac[b]%mod*invfac[a-b]%mod;
}
```

3.3 线性求逆元

```
inv[i]=(p-p/i)*inv[p%i]%p;
```

3.4 Lucas

用于求解大组合数取模的问题，其中模数必须为素数。当问题规模很大，而模数是一个不大的质数的时候使用。下例为：$x,y\le 10^{18}$。p 是质数。$p\le 10^5$。

```
ll C(ll a, ll b){
    if(b>a) return 0;
    return fac[a]*inv(fac[b])%mod*inv(fac[a-b])%mod;
}

ll lucas(ll a, ll b){
    if(!b) return 1;
    return C(a%mod, b%mod)*lucas(a/mod, b/mod)%mod;
}
```

3.5 整除分块

引理：

$$\forall a,b,c\in\mathbb{Z},\left\lfloor\frac{a}{bc}\right\rfloor=\left\lfloor\frac{\left\lfloor\frac{a}{b}\right\rfloor}{c}\right\rfloor$$

结论：

对于常数 n，使

$$\left\lfloor\frac{n}{i}\right\rfloor=\left\lfloor\frac{n}{j}\right\rfloor$$

成立且满足 $i\le j\le n$ 的 j 值最大为 $\left\lfloor\frac{n}{\lfloor\frac{n}{i}\rfloor}\right\rfloor$。

例题：

https://www.luogu.com.cn/problem/P2261

```
ll n, k;

signed main(){
  cin>>n>>k;
  ll res=1LL*n*k;
  for(ll l=1, r; l<=n; l=r+1){
    if(k/l==0) break;
    r=min(n, k/(k/l));
    res-=1LL*(k/l)*(l+r)*(r-l+1)/2;
  }
  cout<<res<<endl;
  return 0;
}
```

3.5.1 向上取整的数论分块

注意到 $\lceil\frac{n}{i}\rceil\equiv\lfloor\frac{n-1}{i}\rfloor+1$，所以 n 的上取整分块和 $n-1$ 下取整分块一样。

$i=n$ 时会出现分母为 0 的错误，需要特殊处理。

3.6 min_25 筛

例题：

https://judge.yosupo.jp/problem/sum_of_multiplicative_function

Given a multiplicative function $f(x)$, satisfying $f(p^e)=be+cp$. Print $\sum_{i=1}^{N}f(i)\bmod 469762049$.

参考：https://judge.yosupo.jp/submission/274231

```
// Problem: Sum of Multiplicative Function
// Contest: Library Checker
// URL: https://judge.yosupo.jp/problem/sum_of_multiplicative_
function
// Memory Limit: 1024 MB
// Time Limit: 10000 ms
//
// Powered by CP Editor (https://cpeditor.org)

#include<bits/stdc++.h>
using i64 = long long;
using u64 = unsigned long long;

constexpr int M = 469762049;
i64 powp(i64 a, i64 n) {
    i64 ans = 1;
    while (n) {
        if (n & 1) (ans *= a) %= M;
        (a *= a) %= M;
        n >>= 1;
    }
    return ans;
}

i64 inv(i64 x) {
    return powp(x, M - 2);
}

struct Sieve {
    std::vector<int> mpf; // minimum prime factor
    std::vector<int> prime;

    Sieve(int n) : mpf(n) {
        for (int i = 2; i < n; i++) {
            if (mpf[i] == 0) {
```

```
                prime.push_back(i);
                mpf[i] = i;
            }
            for (int j = 0; j < prime.size() and i * prime[j] < n;
j++) {
                mpf[i * prime[j]] = prime[j];
                if (prime[j] == mpf[i])
                    break;
            }
        }
    }
};

struct min25 {
    i64 n, N;//原n,  根号n, 根号下质数个数
    std::vector<std::vector<i64>> sum;//M, 根号N下的质数个数
    std::vector<std::vector<i64>> g;//N << 1
    std::vector<i64> w;//N << 1

    std::vector<int> id1, id2;//N
    int tot, sq;
    int cnt;//根号下质数个数
    std::vector<int> p;//质数数组

    std::vector<std::function<i64(i64)>> f;// [editable] 求拟合用的积性
函数每一项
    std::vector<std::function<i64(i64)>> pref;// [editable] 拟合用的积
性函数每一项快速求和 [2, x]  (务必存一下那个求和公式表)
    std::vector<i64> a;// [editable] 多项式系数

    std::function<i64(i64, i64)> pk;// [editable] p^k的快速求法 (输入的
x作为p^k)
    int len;//多项式项数

    //前缀和最大范围; 求拟合用的积性函数每一项; 拟合用的积性函数每一项快速求和;
多项式系数; 快速求p^k函数
    min25(
        i64 n,
        std::vector<std::function<i64(i64)>> f,
        std::vector<std::function<i64(i64)>> pref,
        std::vector<i64> a,
        std::function<i64(i64, i64)> pk
    ) : n(n), f(f), pref(pref), a(a), pk(pk), len(a.size()) {
        N = std::max(20.0, std::sqrt(1.2 * n));
        Sieve si(N);
        p = si.prime;
        cnt = p.size();

        sum.resize(len);
        g.resize(len);
        for (int i = 0; i < len; i++) {
            sum[i].resize(cnt + 1);
            g[i].resize(N * 2 + 1);
        }
        w.resize(N * 2 + 1);
        id1.resize(N + 1);
        id2.resize(N + 1);

        for (int j = 0; j < len; j++) {
            for (int i = 1; i <= cnt; i++) {
                sum[j][i] = (sum[j][i - 1] + f[j](p[i - 1])) % M;
            }
        }
        //std::cout << 1;
    }

    int getid(i64 x) {
        if (x <= sq) return id1[x];
        return id2[n / x];
    }

    i64 func(const std::vector<i64>& x) {// 单项式组合成多项式
        i64 val = 0;
        for (int i = 0; i < len; i++) {
            (val += (M + a[i]) % M * x[i] % M + M) %= M;
        }
        return val;
    }
    i64 S(i64 x, int j) {
        if(p[j] > x) return 0;

        std::vector<i64> v1(len), v2(len);
        for (int i = 0; i < len; i++) {
            v1[i] = g[i][getid(x)];
            v2[i] = sum[i][j];
        }

        i64 sp, ans = (func(v1) - func(v2) + M) % M;
        for (int i = j + 1; i <= cnt && 1ll * p[i - 1] * p[i - 1] <=
x; i++) {
            sp = p[i - 1];
            for (int e = 1; sp <= x; sp *= p[i - 1], e++)
                (ans += pk(p[i - 1], e) * (S(x / sp, i) + (e > 1)) %
M) %= M;
        }

        return (ans + M) % M;
    }
    i64 work(i64 nn) {
        tot = 0;
        n = nn;
        sq = std::sqrt(n);
        for(i64 r, m, l = 1; l <= n; l = r + 1) {
            r = n / (n / l);
            w[tot] = m = n / l;
            for (int i = 0; i < len; i++) {
                g[i][tot] = pref[i](m);
            }
            if(m <= sq)
                id1[m] = tot;
            else
                id2[n / m] = tot;
            ++tot;
        }
        for (int i = 1; i <= cnt; i++) {
            for (int j = 0; j < tot && 1ll * p[i - 1] * p[i - 1] <=
w[j]; j++) {
                for (int k = 0; k < len; k++) {
                    g[k][j] = (g[k][j] - f[k](p[i - 1]) * (g[k]
[getid(w[j] / p[i - 1])] - sum[k][i - 1]) % M) % M;
                }
            }
        }
        return (S(n, 0) + 1) % M;
    }
};

i64 inv2, inv6;

void solve() {
    inv2 = inv(2);
    //inv6 = inv(6);
    i64 n, b, c;
    std::cin >> n >> b >> c;

    std::vector<std::function<i64(i64)>> f(2);
    auto pref = f;
    auto pk = [&](i64 p, i64 k) -> i64 {
        return (b * k % M + c * p % M) % M;
    };
    // f(p) = b + cp
    std::vector<i64> a({b, c});
    f[0] = [](i64 x) -> i64 {
        return 1;
    };
    f[1] = [](i64 x) -> i64 {
        return x % M;
    };

    pref[0] = [](i64 x) -> i64 {
        return (x - 1) % M;
    };
    pref[1] = [](i64 x) -> i64 {
        x %= M;
        return (x * (x + 1) % M * inv2 % M - 1 + M) % M;
    };

    min25 m25(n, f, pref, a, pk);
    i64 ans = m25.work(n);
    std::cout << ans << "\n";
}

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

还有一个我以前在洛谷写的另一道模板题：

https://www.luogu.com.cn/problem/P5325

定义积性函数 $f(x)$，且 $f(p^k)=p^k(p^k-1)$（p 是一个质数），求

$$\sum_{i=1}^{n} f(i)$$

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

#define rep(i,a,b) for(int i=(a);i<=(b);i++)
#define dwn(i,a,b) for(int i=(a);i>=(b);i--)
#define int long long

const int N=1e6+5, mod=1e9+7;

int n;
int lim;

bool vis[N];
int p[N], pc;

int s1[N], s2[N];

int add(int x, int y){
  return x+y>=mod? x+y-mod: x+y;
}

int sub(int x, int y){
  return x-y>=0? x-y: x-y+mod;
}

int mul(int x, int y){
  return x*y%mod;
}

int fpow(int x, int p){
  int res=1;
  for(; p; p>>=1, x=1LL*x*x%mod) if(p&1) res=1LL*res*x%mod;
  return res;
}

int f(int p, int c){
  // TODO: calc f(p^c)
  return mul(fpow(p, c), sub(fpow(p, c), 1));
}

const int inv2=fpow(2, mod-2), inv6=fpow(6, mod-2);

void init(){
  rep(i, 2, lim){
    if(!vis[i]){
      p[++pc]=i;
      // TODO: calc prefix sum of prime pos
      s1[pc]=add(s1[pc-1], i);
      s2[pc]=add(s2[pc-1], mul(i, i));
    }
    for(int j=1; p[j]*i<=lim; j++){
      vis[p[j]*i]=true;
      if(i%p[j]==0) break;
    }
  }
}

////////////////////////////////////////

int val[N], id1[N], id2[N], bc;

int SUM1(int x){
  return mul(mul(x, x+1), inv2);
}

int SUM2(int x){
  return mul(mul(x, mul(x+1, x<<1|1)), inv6);
}

int g1[N], g2[N];

void get_g(){
  for(int l=1, r; l<=n; l=r+1){
    r=min(n, n/(n/l));
    val[++bc]=n/l%mod;

    // TODO: calc g_0
    g1[bc]=sub(SUM1(val[bc]), 1);
    g2[bc]=sub(SUM2(val[bc]), 1);

    val[bc]=n/l;
    if(val[bc]<=lim) id1[val[bc]]=bc;
    else id2[n/val[bc]]=bc;
  }

  rep(j, 1, pc){
    for(int i=1; i<=bc && p[j]*p[j]<=val[i]; i++){
      int tmp=val[i]/p[j];
      int x=(tmp<=lim? id1[tmp]: id2[n/tmp]);
      // TODO: calc g
      g1[i]=sub(g1[i], mul(p[j], sub(g1[x], s1[j-1])));
      g2[i]=sub(g2[i], mul(mul(p[j], p[j]), sub(g2[x], s2[j-1])));
    }
  }
}

int S(int i, int j){
  if(p[j]>=i) return 0;
  int x=(i<=lim? id1[i]: id2[n/i]);
  // TODO: calc Prime part
  int res=sub(sub(g2[x], g1[x]), sub(s2[j], s1[j]));
  for(int k=j+1; p[k]*p[k]<=i && k<=pc; k++){
    int pe=p[k];
    for(int e=1; pe<=i; e++, pe=pe*p[k]){
      res=add(res, mul(f(p[k], e), add(S(i/pe, k), (e>1))));
    }
  }
  return res;
}

signed main(){
  cin>>n;
  lim=sqrt(n)+1;
  init();
  get_g();

  cout<<add(S(n, 0), 1);

  return 0;
}
```

下面是洛谷（P5325）采用新模板的做法，可能会慢一些：

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

#define int long long

using i64 = long long;
using u64 = unsigned long long;

constexpr int M = 1e9 + 7;

int add(int x, int y){
  return (x+y)%M;
}

int mul(int x, int y){
  return x*y%M;
}

i64 powp(i64 a, i64 n) {
    i64 ans = 1;
    while (n) {
        if (n & 1) (ans *= a) %= M;
        (a *= a) %= M;
        n >>= 1;
    }
    return ans;
}

i64 inv(i64 x) {
    return powp(x, M - 2);
}

struct Sieve {
    std::vector<int> mpf; // minimum prime factor
    std::vector<int> prime;

    Sieve(int n) : mpf(n) {
        for (int i = 2; i < n; i++) {
            if (mpf[i] == 0) {
                prime.push_back(i);
                mpf[i] = i;
            }
            for (int j = 0; j < prime.size() and i * prime[j] < n;
j++) {
                mpf[i * prime[j]] = prime[j];
                if (prime[j] == mpf[i])
                    break;
            }
        }
    }
};

struct min25 {
    i64 n, N;//原n,  根号n, 根号下质数个数
    std::vector<std::vector<i64>> sum;//M, 根号N下的质数个数
    std::vector<std::vector<i64>> g;//N << 1
    std::vector<i64> w;//N << 1

    std::vector<int> id1, id2;//N
    int tot, sq;
    int cnt;//根号下质数个数
    std::vector<int> p;//质数数组

    std::vector<std::function<i64(i64)>> f;// [editable] 求拟合用的积性
函数每一项
    std::vector<std::function<i64(i64)>> pref;// [editable] 拟合用的积
性函数每一项快速求和 [2, x]  (务必存一下那个求和公式表)
    std::vector<i64> a;// [editable] 多项式系数

    std::function<i64(i64, i64)> pk;// [editable] p^k的快速求法 (输入的
x作为p^k)
    int len;//多项式项数

    //前缀和最大范围; 求拟合用的积性函数每一项; 拟合用的积性函数每一项快速求和;
多项式系数; 快速求p^k函数
    min25(
        i64 n,
        std::vector<std::function<i64(i64)>> f,
        std::vector<std::function<i64(i64)>> pref,
        std::vector<i64> a,
        std::function<i64(i64, i64)> pk
    ) : n(n), f(f), pref(pref), a(a), pk(pk), len(a.size()) {
        N = std::max(20.0, std::sqrt(1.2 * n));
        Sieve si(N);
        p = si.prime;
        cnt = p.size();

        sum.resize(len);
        g.resize(len);
        for (int i = 0; i < len; i++) {
            sum[i].resize(cnt + 1);
            g[i].resize(N * 2 + 1);
        }
        w.resize(N * 2 + 1);
        id1.resize(N + 1);
        id2.resize(N + 1);

        for (int j = 0; j < len; j++) {
            for (int i = 1; i <= cnt; i++) {
```

```
            sum[j][i] = (sum[j][i - 1] + f[j](p[i - 1])) % M;
        }
    }
    //std::cout << 1;
}

int getid(i64 x) {
    if (x <= sq) return id1[x];
    return id2[n / x];
}

i64 func(const std::vector<i64>& x) {// 单项式组合成多项式
    i64 val = 0;
    for (int i = 0; i < len; i++) {
        (val += (M + a[i]) % M * x[i] % M + M) %= M;
    }
    return val;
}
i64 S(i64 x, int j) {
    if(p[j] > x) return 0;

    std::vector<i64> v1(len), v2(len);
    for (int i = 0; i < len; i++) {
        v1[i] = g[i][getid(x)];
        v2[i] = sum[i][j];
    }

    i64 sp, ans = (func(v1) - func(v2) + M) % M;
    for (int i = j + 1; i <= cnt && 1ll * p[i - 1] * p[i - 1] <= x; i++) {
        sp = p[i - 1];
        for (int e = 1; sp <= x; sp *= p[i - 1], e++)
            (ans += pk(p[i - 1], e) * (S(x / sp, i) + (e > 1)) % M) %= M;
    }

    return (ans + M) % M;
}
i64 work(i64 nn) {
    tot = 0;
    n = nn;
    sq = std::sqrt(n);
    for(i64 r, m, l = 1; l <= n; l = r + 1) {
        r = n / (n / l);
        w[tot] = m = n / l;
        for (int i = 0; i < len; i++) {
            g[i][tot] = pref[i](m);
        }
        if(m <= sq)
            id1[m] = tot;
        else
            id2[n / m] = tot;
        ++tot;
    }
    for (int i = 1; i <= cnt; i++) {
        for (int j = 0; j < tot && 1ll * p[i - 1] * p[i - 1] <= w[j]; j++) {
            for (int k = 0; k < len; k++) {
                g[k][j] = (g[k][j] - f[k](p[i - 1]) * (g[k][getid(w[j] / p[i - 1])] - sum[k][i - 1]) % M) % M;
            }
        }
    }
    return (S(n, 0) + 1) % M;
}
};

i64 inv2, inv6;

void solve() {
    inv2 = inv(2);
    inv6 = inv(6);
    i64 n; cin>>n;

    // f(p) = 0 + -p + p^2
    std::vector<std::function<i64(i64)>> f(3);
    auto pref = f;
    auto pk = [&](i64 p, i64 k) -> i64 {
        return mul(powp(p, k), add(powp(p, k), M-1));
    };
    std::vector<i64> a({0, M-1, 1});
    f[0] = [](i64 x) -> i64 {
        return 1;
    };
    f[1] = [](i64 x) -> i64 {
        return x%M;
    };
    f[2] = [](i64 x) -> i64 {
        return mul(x, x);
    };

    pref[0] = [](i64 x) -> i64 {
        return (x - 1) % M;
    };
    pref[1] = [](i64 x) -> i64 {
        x %= M;
        return (x * (x + 1) % M * inv2 % M - 1 + M) % M;
    };
    pref[2] = [](i64 x) -> i64 {
        x %= M;
        return (x * (x + 1) % M * add(mul(x, 2), 1) % M * inv6 % M - 1 + M) % M;
    };

    min25 m25(n, f, pref, a, pk);
    i64 ans = m25.work(n);
    std::cout << ans << "\n";
}

signed main(){
    std::cin.tie(0) -> sync_with_stdio(false);
    int t = 1;
    // std::cin >> t;
    while(t--) solve();
    return 0;
}
```

3.7 莫比乌斯反演

- 结论 1：若 $F(n)=\sum_{d|n} f(d)$，则 $f(n)=\sum_{d|n} \mu(d)F(\frac{n}{d})$
- 结论 2：若 $F(n)=\sum_{n|t} f(t)$，则 $f(n)=\sum_{n|t} \mu(\frac{t}{n})F(t)$

结论 2 用得比较多。

还有一个常用小结论（也是狄利克雷卷积的一个小特例）：

- $[x=1]\equiv\sum_{d|x}\mu(d)$

3.7.1 例题

https://www.luogu.com.cn/problem/P2522

对于给出的 n 个询问，每次求有多少个数对 (x,y)，满足 $a\le x\le b$，$c\le y\le d$，且 $\gcd(x,y)=k$。

```
const int N=50005;
int primes[N], cnt;
bool vis[N];
int mu[N];
ll sum[N];

void init(){
  for(int i=2; i<N; i++){
    if(!vis[i]) primes[cnt++]=i, mu[i]=-1;
    for(int j=0; i*primes[j]<N; j++){
      vis[i*primes[j]]=true;
      if(i%primes[j]==0) break;
      mu[i*primes[j]]=-mu[i];
    }
  }

  mu[1]=1;
  for(int i=1; i<N; i++) sum[i]=mu[i]+sum[i-1];
}

int g(int b, int l){
  return b/(b/l);
}

ll f(int a, int b, int k){
  ll res=0;
  a=a/k, b=b/k;
  int n=min(a, b);

  for(int l=1, r; l<=n; l=r+1){
    r=min(n, min(g(a, l), g(b, l)));
    res+=(sum[r]-sum[l-1])*(a/l)*(b/l);
  }
  return res;
}

int main(){
  int T; cin>>T;
  init();
  while(T--){
    int a, b, c, d, k; read(a), read(b), read(c), read(d), read(k);
    cout<<f(b, d, k)-f(a-1, d, k)-f(b, c-1, k)+f(a-1, c-1, k)<<endl;
  }
  return 0;
}
```

3.8 exgcd

用于求解 $ax+by=c$。令 $d=(a,b)$，先求解 $ax+by=d$。后将解乘上 $\frac{c}{d}$ 即可。

下面代码可以求出一组特解 x_0,y_0：

```
int exgcd(int a, int b, int &x, int &y){
    if(!b) return x=1, y=0, a;
    int d=exgcd(b, a%b, y, x);
    y-=a/b*x;
    return d;
}
```

而通解形式为

$$\begin{cases} x^*=x_0+\frac{b}{(a,b)}t \\ y^*=y_0-\frac{a}{(a,b)}t \end{cases}$$

这里提供一个 inv_gcd：

参考：https://github.com/kk2a/library/blob/main/math/inv_gcd.hpp

```
// return {gcd(a, b), x} where a * x + b * y = gcd(a, b)
std::pair<long long, long long> inv_gcd(long long a, long long b) {
    a = a % b;
    if (a == 0) return {b, 0};
    long long s = b, t = a;
    long long m0 = 0, m1 = 1;

    while (t) {
        long long u = s / t;
        std::swap(s -= t * u, t);
        std::swap(m0 -= u * m1, m1);
    }
    if (m0 < 0) m0 += b / s;
    return {s, m0};
}
```

3.9 exEuler

luogu.com.cn/problem/P5091

给你三个正整数，a,m,b，你需要求：$a^b \bmod m$。

exEuler：若 $b\ge\phi(m)$，那么 $a^b\equiv a^{b\%\phi(m)+\phi(m)}\pmod m$。

3.10 bsgs

luogu.com.cn/problem/P3846

给定一个质数 p，以及一个整数 b，一个整数 n，现在要求你计算一个最小的非负整数 l，满足 $b^l\equiv n\pmod p$。

```
#define int long long

int bsgs(int a, int p, int b){
    if(1%p==b%p) return 0;
    int k=sqrt(p)+1;
    unordered_map<int, int> hash;

    for(int i=0, j=b%p; i<k; i++){
        hash[j]=i;
        j=j*a%p;
    }
    int ak=1;
    for(int i=0; i<k; i++) ak=ak*a%p; // get a^k
    for(int i=1, j=ak; i<=k; i++){
        if(hash.count(j)) return i*k-hash[j];
        j=j*ak%p;
    }
    return -1;
}

signed main(){
    int a, p, b;
    while(cin>>p>>a>>b){
        int res=bsgs(a, p, b);
        if(res==-1) puts("no solution");
        else cout<<res<<endl;
    }
    return 0;
}
```

3.11 exbsgs

给定 a,p,b，求满足 $a^x\equiv b\pmod p$ 的最小自然数 x。

保证 $\sum\sqrt{p}\le 5\times 10^6$。

```
#define int long long

const int INF=1e8;

int exgcd(int a, int b, int &x, int &y){
    if(!b){
        x=1, y=0;
        return a;
    }
    int d=exgcd(b, a%b, y, x);
    y-=a/b*x;
    return d;
}

int bsgs(int a, int b, int p){
    if(1%p==b%p) return 0;
    int k=sqrt(p)+1;
    unordered_map<int, int> hash;

    for(int i=0, j=b%p; i<k; i++){
        hash[j]=i;
        j=j*a%p;
    }
    int ak=1;
    for(int i=0; i<k; i++) ak=ak*a%p; // get a^k
    for(int i=1, j=ak; i<=k; i++){
        if(hash.count(j)) return i*k-hash[j];
        j=j*ak%p;
    }
    return -INF;
}

int exbsgs(int a, int b, int p){
    b=(b%p+p)%p;
    if(1%p==b%p) return 0;
    int x, y;
    int d=exgcd(a, p, x, y);
    if(d>1){
        if(b%d) return -INF;
        exgcd(a/d, p/d, x, y);
        return exbsgs(a, b/d*x%(p/d), p/d)+1;
    }
    return bsgs(a, b, p);
}

signed main(){
    int a, b, p;
    while(cin>>a>>p>>b, a || b || p){
        int res=exbsgs(a, b, p);
        if(res<0) puts("No Solution");
        else cout<<res<<endl;
    }
    return 0;
}
```

3.12 exCRT

用的较多的是 exCRT，但是这里还是记录一下 CRT 的构造做法：

> 我们有 $n_1,\dots,n_k$ 两两互质的同余方程组 $x\equiv a_i\pmod{n_i}$
> - 计算所有模数积 n
> - 对于第 i 个方程：
>   1. 计算 $m_i=\frac{n}{n_i}$
>   2. 计算 m_i 在模 n_i 意义下的逆元 m_i^{-1}
>   3. 计算 $c_i=m_im_i^{-1}$（不要对 n_i 取模）
> - 方程组在模 n 意义下唯一解为 $x=\sum_{i=1}^k a_ic_i\pmod n$。

例题：

luogu.com.cn/problem/P4777

给定 n 组非负整数 a_i,b_i，方程组形式为 $x\equiv b_i\pmod{a_i}$，求解关于 x 的方程组的最小非负整数解。

```
int n, M[N], A[N]; // M 对应 a_i, A 对应 b_i

int mul(int x, int p, int mod){
    int res=0;
    for(; p; p>>=1, x=(x+x)%mod) if(p&1) res=(res+x)%mod;
    return res;
}

int exgcd(int a, int b, int &x, int &y){
    if(!b) return x=1, y=0, a;
    int d=exgcd(b, a%b, y, x);
    y-=a/b*x;
    return d;
}

int exCRT(int n, int *A, int *M){
    int x, y;
    int m=M[1], res=(A[1]%m+m)%m;
    rep(i, 2, n){
        int a=m, b=M[i], d=((A[i]-res)%b+b)%b;
        int g=exgcd(a, b, x, y);
        if(d%g) return -1;

        int P=b/g;
        x=mul(x, d/g, P);
        res+=x*m, m*=P;
        (res=res%m+m)%=m;
    }
    return res;
}

signed main(){
    cin>>n;
    rep(i, 1, n) read(M[i]), read(A[i]);
    cout<<exCRT(n, A, M)<<endl;
    return 0;
}
```

3.13 Miller-Rabin

用费马小定理和二次剩余来判断质数。

```
inline int mul(int a, int b, int mod){
    return (__int128)a*b%mod;
}

inline int fpow(int x, int p, int mod){
    int res=1;
    for(; p; p>>=1, x=mul(x, x, mod)) if(p&1) res=mul(res, x, mod);
    return res;
}

inline bool rabin(int n, int a){
    int d=n-1, ct=0;
    while(~d&1) d>>=1, ct++;
    int x=fpow(a, d, n);
    if(x=1) return true;
    rep(i,1,ct){
        if(x=n-1 || x=1) return true;
        x=mul(x, x, n);
    }
    return false;
```

```
}


inline bool isp(int n){
    if(n<2) return false;
    static const int pri[] = {2, 3, 5, 7, 11, 13, 31, 61, 24251};
    for(auto x: pri) {
        if(n=x) return true;
        if(!rabin(n, x)) return false;
    }
    return true;
}
```

3.14 Pollard-Rho 算法

luogu.com.cn/problem/P4718

用于在 $O(n^{1/4})$ 的期望时间复杂度内计算合数 n 的某个非平凡因子。

下例子对于每个数字检验是否是质数，是质数就输出 Prime；如果不是质数，输出它最大的质因子是哪个。

注意调用的时候不要去 find(1)。

调用 find 后得到的 fac[] 可能有重复，如有必要手动去重。

```
const int N=1e5+5, s=20;

mt19937_64 rd(time(NULL));
int fac[N], cnt;

inline int mul(int a, int b, int mod){
  return (__int128)a*b%mod;
}

inline int fpow(int x, int p, int mod){
  int res=1;
  for(; p; p>>=1, x=mul(x, x, mod)) if(p&1) res=mul(res, x, mod);
  return res;
}

inline bool rabin(int n, int a){
  int d=n-1, ct=0;
  while(~d&1) d>>=1, ct++;
  int x=fpow(a, d, n);
  if(x==1) return true;
  rep(i, 1, ct){
    if(x==n-1 || x==1) return true;
    x=mul(x, x, n);
  }
  return false;
}


inline bool isp(int n){
  if(n<2) return false;
  static const int pri[] = {2, 3, 5, 7, 11, 13, 31, 61, 24251};
  for(auto x: pri) {
    if(n==x) return true;
    if(!rabin(n, x)) return false;
  }
  return true;
}

inline int f(int x, int c, int mod){
  return (mul(x, x, mod)+c)%mod;
}

inline int pollard_rho(int n){
  int c=rd()%(n-1)+1; int v1=0;
  for(int s=1, t=2; ; s<<=1, t<<=1){
    int pro=1, v2=v1, cnt=0;
    rep(i,s+1,t){
      v2=f(v2, c, n), pro=mul(pro, abs(v1-v2), n), cnt++;
      if(cnt%127==0){
        cnt=0; int d=__gcd(pro, n);
        if(d>1) return d;
      }
    }
    int d=__gcd(pro, n);
    if(d>1) return d;
    v1=v2;
  }
}

inline void find(int n){
  if(isp(n)) return fac[++cnt]=n, void();
  int p;
  while((p=pollard_rho(n))==n) ;
  find(p), find(n/p);
}

signed main(){
    int T; cin>>T;
    while(T--){
        cnt=0;
        int n; read(n);
        if(isp(n)){
            puts("Prime");
            continue;
        }
        find(n);
        int res=0;
        rep(i,1,cnt) if(fac[i]!=n) res=max(res, fac[i]);
        cout<<res<<endl;
    }
    return 0;
}
```

4 线性代数

4.1 高斯消元

luogu.com.cn/problem/P3389

即求解方程 $AX=Y$ 的 X。

```
const int N=110;
const double eps=1e-8;

int n;
double b[N][N];

void guass(){
    int rk=0;
    for(int r=1, c=1; r<=n; c++, r++){
        int t=r;
        for(int i=r+1; i<=n; i++) if(fabs(b[i][c])>fabs(b[t][c]))
t=i;
        if(fabs(b[t][c])<eps) continue;
        ++rk;
        for(int i=c; i<=n+1; i++) swap(b[r][i], b[t][i]);
        for(int i=n+1; i>=c; i--) b[r][i]/=b[r][c];
        for(int i=r+1; i<=n; i++) for(int j=n+1; j>=c; j--) b[i][j]-
=b[i][c]*b[r][j];
    }
    for(int i=n; i>1; i--) for(int j=i-1; j; j--) b[j][n+1]-=b[i]
[n+1]*b[j][i], b[j][i]=0;
    if(rk!=n){
        puts("No Solution");
        return;
    }
    for(int i=1; i<=n; i++) printf("%.2lf\n", b[i][n+1]);
}

signed main(){
    cin>>n;
    rep(i,1,n) rep(j,1,n+1) cin>>b[i][j];
    guass();
    return 0;
}
```

4.2 矩阵求逆

求一个 $N\times N$ 的矩阵的逆矩阵。答案对 10^9+7 取模。

```
#define int long long

inline void read(int &x){
    int s=0; x=1;
    char ch=getchar();
    while(ch<'0' || ch>'9') {if(ch='-')x=-1;ch=getchar();}
    while(ch>='0' && ch<='9') s=(s<<3)+(s<<1)+ch-'0',ch=getchar();
    x*=s;
}

const int N=405, mod=1e9+7;

int n, a[N][N<<1];

int fpow(int x, int p){
    int res=1;
    for(; p; p>>=1, x=1LL*x*x%mod) if(p&1) res=1LL*res*x%mod;
    return res%mod;
}

int inv(int x){
    return fpow(x, mod-2);
}

void mat_inv(){
    for(int i=1; i<=n; i++){
        int r=i;
        for(int j=i+1; j<=n; j++) if(a[j][i]>a[r][i]) r=j;
        if(r!=i) swap(a[i], a[r]);
        if(!a[i][i]) return puts("No Solution"), void();

        int Inv=inv(a[i][i]);
        for(int k=1; k<=n; k++){
            if(k=i) continue;
            int p=a[k][i]*Inv%mod;
            for(int j=i; j<=(n<<1); ++j) a[k][j]=((a[k][j]-p*a[i]
[j])%mod+mod)%mod;
        }
        for(int j=1; j<=(n<<1); j++) a[i][j]=(a[i][j]*Inv%mod);
    }
    for(int i=1; i<=n; i++){
        for(int j=n+1; j<=(n<<1); j++) cout<<a[i][j]<<" ";
        cout<<endl;
    }
}

signed main(){
    cin>>n;
    rep(i, 1, n) rep(j,1,n) read(a[i][j]);
    rep(i, 1, n) a[i][i+n]=1;
    mat_inv();
    return 0;
}
```

4.3 行列式求值（取模）

luogu.com.cn/problem/P7112

计算 $|w|$。

```
const int N=610;

int n, mod;
int w[N][N];

int getDet(){
    int res=1;
    rep(i, 1, n){
        rep(j, i+1, n){
            while(w[i][i]){
                int div=w[j][i]/w[i][i];
                rep(k,i,n) w[j][k]=(w[j][k]-1LL*div*w[i]
[k]%mod+mod)%mod;
                res=-res, swap(w[i], w[j]);
            }
            res=-res, swap(w[i], w[j]);
        }
    }
    rep(i, 1, n) res=1LL*res*w[i][i]%mod;
    return (res%mod+mod)%mod;
}

signed main(){
    while(cin>>n>>mod){
        rep(i, 1, n) rep(j, 1, n) read(w[i][j]);
        cout<<getDet()<<endl;
    }
    return 0;
}
```

4.4 线性基

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。

- 传统写法：

```
const int N=55;

ll base[N];

int main(){
    int n; cin>>n;
    while(n--){
        ll k; cin>>k;
        for(int j=N-1; ~j; j--){
            if(k&(1LL<<j)){
                if(!base[j]) base[j]=k;
                k^=base[j];
            }
        }
    }

    ll ans=0;
    for(int j=N-1; ~j; j--){
        if((ans^base[j])>ans) ans=ans^base[j];
    }

    cout<<ans<<endl;

    return 0;
}
```

我之前在 Atcoder 学到的简单写法：

```
signed main(){
  int n; cin>>n;
  vector<int> w(n), d;
  rep(i,0,n-1) read(w[i]);

  for(auto i: w){
      int val=i;
      for(auto j: d) if((val^j)<val) val^=j;
      if(val) d.push_back(val);
  }

  int res=0;
  for(auto i: d) res=max(res, res^i);
  cout<<res<<endl;

  return 0;
}
```

4.5 LGV 引理

4.5.1 公式

可用来求用来处理有向无环图上不相交路径计数问题。

$\omega(p)$ 表示路径 P 上所有边权之积。

$e(u,v)$ 表示 $u\to v$ 的所有路径的 ω 值之和。

A,B 分别为起点、终点集合，两个集合大小相等。

那么答案为 $\det e(A_i,B_j)$

4.5.2 例题

www.luogu.com.cn/problem/P6657

有一个 $n\times n$ 的棋盘，左下角为 $(1,1)$，右上角为 (n,n)，若一个棋子在点 (x,y)，那么走一步只能走到 $(x+1,y)$ 或 $(x,y+1)$。

现在有 m 个棋子，第 i 个棋子一开始放在 $(a_i,1)$，最终要走到 (b_i,n)。问有多少种方案，使得每个棋子都能从起点走到终点，且对于所有棋子，走过路径上的点互不相交。输出方案数 $\bmod 998244353$ 的值。

两种方案不同当且仅当存在至少一个棋子所经过的点不同。

```
#define int long long
const int N=2e6+5, M=110, mod=998244353;

int fpow(int x, int p){
    int res=1;
    for(; p; p>>=1, x=1LL*x*x%mod) if(p&1) res=1LL*res*x%mod;
    return res%mod;
}

int inv(int x){
    return fpow(x, mod-2);
}

int fac[N];
int invfac[N];

void getFac(int n){
    fac[0]=invfac[0]=1;
    for(int i=1; i<=n; i++)
    fac[i]=1LL*fac[i-1]*i%mod, invfac[i]=1LL*invfac[i-1]*inv(i)%mod;
}

int C(int a, int b){
    if(b>a) return 0;
    return 1LL*fac[a]*invfac[b]%mod*invfac[a-b]%mod;
}

int n, m;
int a[M], b[M];

int f[M][M];

int det(){
    int res=1, sig=1;
    rep(i, 1, m){
        int t=0;
        rep(j, i, m) if(f[j][i]) t=j;
        if(!t) return 0;
        if(i!=t){
            sig*=-1;
            rep(j, i, m) swap(f[i][j], f[t][j]);
        }
        rep(j, i+1, m){
            int rate=f[j][i]*inv(f[i][i])%mod;
            rep(k, i, m) f[j][k]-=f[i][k]*rate%mod, f[j][k]=(f[j]
[k]%mod+mod)%mod;
        }
        (res*=f[i][i])%=mod;
    }
    res*=sig;
    return (res%mod+mod)%mod;
}

signed main(){
    getFac(N-1);
    int cs; cin>>cs;
    while(cs--){
        read(n), read(m);
        rep(i, 1, m) read(a[i]), read(b[i]);
        rep(i, 1, m) rep(j,1,m) f[i][j]=C(n-1+b[j]-a[i], n-1);
        cout<<det()<<endl;
    }
    return 0;
}
```

5 多项式及计数技巧

5.1 拉格朗日插值

例题：

对于 n 个点 (x_i,y_i)，如果满足 $\forall i\neq j, x_i\neq x_j$，那么经过这 n 个点可以唯一地确定一个 $n-1$ 次多项式 $y=f(x)$。

现在，给定这样 n 个点，请你确定这个 $n-1$ 次多项式，并求出 $f(k)\bmod 998244353$ 的值。

https://www.luogu.com.cn/problem/P4781

```
#define int long long
const int N=2020, mod=998244353;

int n, x[N], y[N], k;
```

```cpp
int fpow(int x, int p){
  int res=1;
  for(; p; p>>=1, x=x*x%mod) if(p&1) res=res*x%mod;
  return res;
}

int inv(int x){
  return fpow(x, mod-2);
}

int lagr(int n, int *x, int *y, int k){
  int res=0;
  rep(i,1,n){
    int nu=y[i], de=1;
    rep(j,1,n) if(j!=i) (nu*=(k-x[j])%mod+mod)%=mod, (de*=(x[i]-x[j])%mod+mod)%=mod;
    (res+=nu*inv(de)%mod)%=mod;
  }
  return res;
}

signed main(){
  cin>>n>>k;
  rep(i, 1, n) read(x[i]), read(y[i]);

  cout<<lagr(n, x, y, k)<<endl; // k 是横坐标位置

  return 0;
}
```

NTT 相关部分是同学分享的板子。

5.2 NTT 相关

```cpp
#include <bits/stdc++.h>
#define fp(i, a, b) for (int i = (a), i##_ = (b) + 1; i < i##_; ++i)
#define fd(i, a, b) for (int i = (a), i##_ = (b) - 1; i > i##_; --i)
#define MUL(a, b) ((ll)(a) * (b) % P) // 大模数时这里ll要改成__int128
#define ADD(a, b) (((a) += (b)) >= P ? (a) -= P : 0) // (a += b) %= P
#define SUB(a, b) (((a) -= (b)) < 0 ? (a) += P: 0) // ((a -= b) += P) %= P
#define int long long
using namespace std;
const int N = 3e5 + 5, P = 998244353; //模数
using ll = int64_t;
using Poly = vector<int>;
/*--------------------------------------------------------------*/
class Cipolla {
    int P, I2 {};
    using pll = pair<ll, ll>;
#define X first
#define Y second
    ll mul(ll a, ll b) const {
        return a * b % P;
    }
    pll mul(pll a, pll b) const {
        return {(a.X * b.X + I2 * a.Y % P * b.Y) % P,
                (a.X * b.Y + a.Y * b.X) % P
               };
    }
    template<class T> T POW(T a, int b, T x) {
        for (; b; b >>= 1, a = mul(a, a))
            if (b & 1) x = mul(x, a);
        return x;
    }
public:
    Cipolla(int p = 0) : P(p) {}
    pair<int, int> sqrt(int n) {
        int a = rand(), x;
        if (!(n %= P)) return {0, 0};
        if (POW(n, (P - 1) >> 1, 1ll) = P - 1) return {-1, -1};
        while (POW(I2 = ((ll) a * a - n + P) % P, (P - 1) >> 1, 1ll) = 1) a =
                rand();
        x = (int) POW(pll {a, 1}, (P + 1) >> 1, {1, 0}).X;
        if (2 * x > P) x = P - x;
        return {x, P - x};
    }
#undef X
#undef Y
};
/*--------------------------------------------------------------*/
Poly getInv(int L) {
    Poly inv(L);
    inv[1] = 1;
    fp(i, 2, L - 1)
    inv[i] = MUL((P - P / i), inv[P % i]);
    return inv;
}
ll POW(ll a, ll b = P - 2, ll x = 1) {
    for (; b; b >>= 1, a = MUL(a, a))
        if (b & 1) x = MUL(x, a);
    return x;
}
auto inv = getInv(N); // NOLINT
/*--------------------------------------------------------------*/
namespace NTT {
    const int g = 3; //原根
    Poly Omega(int L) {
        int wn = POW(g, P / L);
        Poly w(L);
        w[L >> 1] = 1;
        fp(i, L / 2 + 1, L - 1) w[i] = MUL(w[i - 1], wn);
        fd(i, L / 2 - 1, 1) w[i] = w[i << 1];
        return w;
    }
    auto W = Omega(1 << 21); // NOLINT 初始化 2的次幂大于等于NTT长度
    void DIF(int *a, int n) {
        for (int k = n >> 1; k; k >>= 1)
            for (int i = 0, y; i < n; i += k << 1)
                for (int j = 0; j < k; ++j)
                    y = a[i + j + k], a[i + j + k] = MUL(a[i + j] - y + P, W[k +
                                                                             j]),
ADD(a[i + j], y);
    }
    void DIT(int *a, int n) {
        for (int k = 1; k < n; k <<= 1)
            for (int i = 0, x, y; i < n; i += k << 1)
                for (int j = 0; j < k; ++j)
                    x = a[i + j], y = MUL(a[i + j + k], W[k + j]),
                    a[i + j + k] = x - y < 0 ? x - y + P : x - y,
ADD(a[i + j],
                        y);
        int Inv = P - (P - 1) / n;
        fp(i, 0, n - 1) a[i] = MUL(a[i], Inv);
        reverse(a + 1, a + n);
    }
}
/*--------------------------------------------------------------*/
namespace Polynomial {
    // basic operator
    int norm(int n) {
        return 1 << (32 - __builtin_clz(n - 1));
    }
    void norm(Poly &a) {
        if (!a.empty()) a.resize(norm(a.size()), 0);
        else a =
        {0};
    }
    void DFT(Poly &a) {
        NTT::DIF(a.data(), a.size());
    }
    void IDFT(Poly &a) {
        NTT::DIT(a.data(), a.size());
    }
    Poly &dot(Poly &a, Poly &b) {
        fp(i, 0, a.size() - 1) a[i] = MUL(a[i], b[i]);
        return a;
    }
    // mul / div int
    Poly &operator*=(Poly &a, int b) {
        for (auto &x : a) x = MUL(x, b);
        return
            a;
    }
    Poly operator*(Poly a, int b) {
        return a *= b;
    }
    Poly operator*(int a, Poly b) {
        return b * a;
    }
    Poly &operator/=(Poly &a, int b) {
        return a *= POW(b);
    }
    Poly operator/(Poly a, int b) {
        return a /= b;
    }
    // Poly add / sub
    Poly &operator+=(Poly &a, Poly b) {
        a.resize(max(a.size(), b.size()));
        fp(i, 0, b.size() - 1) ADD(a[i], b[i]);
        return a;
    }
    Poly operator+(Poly a, Poly b) {
        return a += b;
    }
    Poly &operator-=(Poly &a, Poly b) {
        a.resize(max(a.size(), b.size()));
        fp(i, 0, b.size() - 1) SUB(a[i], b[i]);
        return a;
    }
    Poly operator-(Poly a, Poly b) {
        return a -= b;
    }
    // Poly mul
    Poly operator*(Poly a, Poly b) {
        int n = a.size() + b.size() - 1, L = norm(n);
        if (a.size() <= 8 || b.size() <= 8) {
            Poly c(n);
            fp(i, 0, a.size() - 1) fp(j, 0, b.size() - 1)
            c[i + j] = (c[i + j] + (ll) a[i] * b[j]) % P;
            return c;
        }
        a.resize(L), b.resize(L);
        DFT(a), DFT(b), dot(a, b), IDFT(a);
        return a.resize(n), a;
    }
    // Poly inv
    Poly Inv2k(Poly a) { // |a| = 2 ^ k
        int n = a.size(), m = n >> 1;
        if (n == 1) return {POW(a[0])};
        Poly b = Inv2k(Poly(a.begin(), a.begin() + m)), c = b;
        b.resize(n), DFT(a), DFT(b), dot(a, b), IDFT(a);
        fp(i, 0, n - 1) a[i] = i < m ? 0 : P - a[i];
        DFT(a), dot(a, b), IDFT(a);
        return move(c.begin(), c.end(), a.begin()), a;
    }
    Poly Inv(Poly a) {
        int n = a.size();
        norm(a), a = Inv2k(a);
        return a.resize(n), a;
    }
    // Poly div / mod
    Poly operator/(Poly a,Poly b) {
        int k = a.size() - b.size() + 1;
        if (k < 0) return {0};
        reverse(a.begin(), a.end());
        reverse(b.begin(), b.end());
        b.resize(k), a = a * Inv(b);
        a.resize(k), reverse(a.begin(), a.end());
        return a;
    }
    pair<Poly, Poly> operator%(Poly a, const Poly& b) {
        Poly c = a / b;
        a -= b * c, a.resize(b.size() - 1);
        return {c, a};
    }
    // Poly calculus
    Poly deriv(Poly a) {
        fp(i, 1, a.size() - 1) a[i - 1] = MUL(i, a[i]);
        return a.pop_back(), a;
    }
    Poly integ(Poly a) {
        a.push_back(0);
        fd(i, a.size() - 1, 1) a[i] = MUL(inv[i], a[i - 1]);
        return a[0] = 0, a;
    }
    // Poly ln
    Poly Ln(Poly a) {
        int n = a.size();
        a = deriv(a) * Inv(a);
        return a.resize(n - 1), integ(a);
    }
    // Poly exp
    Poly Exp(Poly a) {
        int n = a.size(), k = norm(n);
        Poly b = {1}, c, d;
        a.resize(k);
        for (int L = 2; L <= k; L <<= 1) {
            d = b, b.resize(L), c = Ln(b), c.resize(L);
            fp(i, 0, L - 1) c[i] = a[i] - c[i] + (a[i] < c[i] ? P :
0);
            ADD(c[0], 1), DFT(b), DFT(c), dot(b, c), IDFT(b);
            move(d.begin(), d.end(), b.begin());
        }
        return b.resize(n), b;
    }
    // Poly sqrt
    Poly Sqrt(Poly a) {
        int n = a.size(), k = norm(n);
        a.resize(k);
        Poly b = {(new Cipolla(P))->sqrt(a[0]).first, 0}, c;
        for (int L = 2; L <= k; L <<= 1) {
            b.resize(L), c = Poly(a.begin(), a.begin() + L) *
Inv2k(b);
            fp(i, L / 2, L - 1) b[i] = MUL(c[i], (P + 1) / 2);
        }
        return b.resize(n), b;
    }
    // Poly pow
    Poly Pow(Poly &a, int b) {
        return Exp(Ln(a) * b);    // a[0] = 1
    }
    Poly Pow(Poly a, int b1, int b2) { // b1 = b % P, b2 = b %
phi(P) and b >= n iff a[0] > 0
        int n = a.size(), d = 0, k;
        while (d < n && !a[d]) ++d;
        if ((ll) d * b1 >= n) return Poly(n);
        a.erase(a.begin(), a.begin() + d);
        k = POW(a[0]), norm(a *= k);
        a = Pow(a, b1) * POW(k, P - 1 - b2);
        a.resize(n), d *= b1;
        fd(i, n - 1, 0) a[i] = i >= d ? a[i - d] : 0;
        return a;
    }
    // Get [x ^ k](f / g)
    int divAt(Poly f, Poly g, ll k) {
        int n = max(f.size(), g.size()), m = norm(n);
        for (; k; k >>= 1) {
            f.resize(m * 2, 0), DFT(f);
            g.resize(m * 2, 0), DFT(g);
            fp(i, 0, 2 * m - 1) f[i] = MUL(f[i], g[i ^ 1]);
            fp(i, 0, m - 1) g[i] = MUL(g[2 * i], g[2 * i + 1]);
            g.resize(m), IDFT(f), IDFT(g);
            for (int i = 0, j = k & 1; i < n; i++, j += 2) f[i] =
f[j];
            f.resize(n), g.resize(n);
        }
        return f[0];
    }
    // Get a[k] by a[n] = sum c[i] * a[n - i]
    int LinearRecur(Poly a, Poly c, ll k) {
        c[0] = P - 1, a = a * c, a.resize(c.size() - 1);
        return divAt(a, c, k);
    }
    //Poly cyc循环卷积
    Poly cyc(Poly a, Poly b) {
        int n = a.size();
        reverse(a.begin(), a.end());
        b.insert(b.end(), b.begin(), b.end());
        a = a * b;
        fp(i, 0, n - 1) a[i] = a[i + n - 1];
        return a.resize(n), a;
    }

    //Poly subpoly差卷积
    Poly subpoly(Poly a, Poly b) {
        int n = a.size();
        int m = b.size();
        reverse(b.begin(), b.end());
        Poly c = a * b;
        fp(i, 0, n - 1) c[i] = c[i + m - 1];
        return c.resize(n), c;
    }
}
/*--------------------------------------------------------------*/
Poly BerlekampMassey(Poly &a) {
    Poly c, las, cur;
    int k, delta, d, w;
    fp(i, 0, a.size() - 1) {
        d = P - a[i];
        fp(j, 0, c.size() - 1) d = (d + (ll) a[i - j - 1] * c[j]) %
P;
        if (!d) continue;
        if (c.empty()) {
            k = i, delta = d, c.resize(i + 1);
            continue;
        }
        cur = c, w = POW(delta, P - 2, P - d);
        if (c.size() < las.size() + i - k) c.resize(las.size() + i -
k);
        SUB(c[i - k - 1], w);
        fp(j, 0, las.size() - 1) c[i - k + j] = (c[i - k + j] + (ll)
w * las[j])
                                                % P;
        if (cur.size() <= las.size() + i - k) las = cur, k = i,
delta = d;
    }
    return c;
}
/*--------------------------------------------------------------*/
using namespace Polynomial;
```

5.3 二项式反演

g_n 表示钦定 n 个位置满足要求的方案数；f_n 表示恰好 n 个位置满足要求的方案数。

因为 f 一般难求，所以考虑利用 g 求出 f。

我们有公式：

$$g_n = \sum_{i=n}^{m} C_i^n f_i$$

$$f_n = \sum_{i=n}^{m} (-1)^{i-n} C_i^n g_i$$

使用多项式科技可以在 $O(n \log n)$ 复杂度求出 f。（上式可以化为卷积形式）

下示例代码是求字符串刚好有 k 个子段 bit 的方案数。

```cpp
using namespace Polynomial;

const int mod=P;

int n;

int fpow(int x, int p){
    if(p<0) return 0;
    int res=1;
    for(; p; p>>=1, x=1LL*x*x%mod) if(p&1) res=1LL*res*x%mod;
    return res%mod;
}

int Inv(int x){
    return fpow(x, mod-2);
}

int fac[N];
int invfac[N];

void getFac(int n){
    fac[0]=invfac[0]=1;
    for(int i=1; i<=n; i++) fac[i]=1LL*fac[i-1]*i%mod,
invfac[i]=1LL*invfac[i-1]*Inv(i)%mod;
}

int C(int a, int b){
    if(b>a) return 0;
```

```
    return 1LL*fac[a]*invfac[b]%mod*invfac[a-b]%mod;
}

Poly binoinv(Poly &G){
    Poly P, Q(n+1), R(n+1);
    int sig=1;
    for(int i=0; i<=n; i++) Q[i]=G[i]*fac[i]%mod,
R[i]=(sig*invfac[i]+mod)%mod, sig*=-1;
    P=subpoly(Q, R);
    for(int i=0; i<=n; i++) P[i]=P[i]*invfac[i]%mod;
    return P;
}

Poly calG(){
    Poly G(n+1);
    for(int k=0; k<=n; k++) G[k]=C(n-2*k, k)*fpow(26, n-3*k)%mod;
    return G;
}

signed main(){
    cin>>n;
    getFac(n);

    auto G=calG();
    auto res=binoinv(G);
    for(auto &i: res) cout<<i<<' ';
    puts("");

    return 0;
}
```

5.4 FFT 相关

```
#define fp(i, a, b) for (int i = (a), i##_ = (b) + 1; i < i##_; ++i)
#define fd(i, a, b) for (int i = (a), i##_ = (b) - 1; i > i##_; --i)
using db = double;
/*--------------------*/
struct cp {
    db x, y;
    cp(db real = 0, db imag = 0) : x(real), y(imag){};
    cp operator+(cp b) const { return {x + b.x, y + b.y}; }
    cp operator-(cp b) const { return {x - b.x, y - b.y}; }
    cp operator*(cp b) const { return {x * b.x - y * b.y, x * b.y +
y * b.x}; }
};
using vcp = vector<cp>;
using Poly = vector<int>;
namespace FFT {
    const db pi = acos(-1);
    vcp Omega(int L) {
        vcp w(L); w[1] = 1;
        for (int i = 2; i < L; i <<= 1) {
            auto w0 = w.begin() + i / 2, w1 = w.begin() + i;
            cp wn(cos(pi / i), sin(pi / i));
            for (int j = 0; j < i; j += 2)
                w1[j] = w0[j >> 1], w1[j + 1] = w1[j] * wn;
        }
        return w;
    }
    auto W = Omega(1 << 21); // 请根据具体题目中 多项式的 最大可能长度进行调
整。
    void DIF(cp *a, int n) {
        cp x, y;
        for (int k = n >> 1; k; k >>= 1)
            for (int i = 0; i < n; i += k << 1)
                for (int j = 0; j < k; ++j)
                    x = a[i + j], y = a[i + j + k],
                    a[i + j + k] = (x - y) *  W[k + j], a[i + j] = x
+ y;
    }
    void IDIT(cp *a, int n) {
        cp x, y;
        for (int k = 1; k < n; k <<= 1)
            for (int i = 0; i < n; i += k << 1)
                for (int j = 0; j < k; ++j)
                    x = a[i + j], y = a[i + j + k] * W[k + j],
                    a[i + j + k] = x - y, a[i + j] = x + y;
        const db Inv = 1. / n;
        fp(i, 0, n - 1) a[i].x *= Inv, a[i].y *= Inv;
        reverse(a + 1, a + n);
    }
}

/*--------------------*/
namespace MTT{
    Poly conv(const Poly &a, const Poly &b, const int&P) {
        int n = a.size(), m = b.size(), o = n + m - 1, l = 1 <<
(__lg(o - 1) + 1);
        vcp A(l), B(l), c0(l), c1(l);
        for (int i = 0; i < n; i++) A[i] = cp(a[i] & 0x7fff, a[i] >>
15);
        for (int i = 0; i < m; i++) B[i] = cp(b[i] & 0x7fff, b[i] >>
15);
        FFT::DIF(A.data(), l), FFT::DIF(B.data(), l);
        for (int k = 1, i = 0, j; k < l; k <<= 1)
            for (; i < k * 2; i++) {
                j = i ^ k - 1;
                c0[i] = cp(A[i].x + A[j].x, A[i].y - A[j].y) * B[i]
* 0.5;
                c1[i] = cp(A[i].y + A[j].y, -A[i].x + A[j].x) * B[i]
* 0.5;
            }
        FFT::IDIT(c0.data(), l), FFT::IDIT(c1.data(), l);
        Poly res(o);
        for (int i = 0; i < o; i++) {
            ll c00 = c0[i].x + 0.5, c01 = c0[i].y + 0.5, c10 =
c1[i].x + 0.5, c11 = c1[i].y + 0.5;
            res[i] = (c00 + ((c01 + c10) % P << 15) + (c11 % P <<
30)) % P;
        }
        return res;
    }
}
/*--------------------*/

namespace Polynomial {
    // basic operator
    void DFT(vcp &a) { FFT::DIF(a.data(), a.size()); }
    void IDFT(vcp &a) { FFT::IDIT(a.data(), a.size()); }
    int norm(int n) { return 1 << (__lg(n - 1) + 1); }

    // Poly mul
    vcp &dot(vcp &a, vcp &b) { fp(i, 0, a.size() - 1) a[i] = a[i] *
b[i]; return a; }
    Poly operator*(Poly a, Poly b) {
        int n = a.size() + b.size() - 1;
        vcp c(norm(n));
        fp(i, 0, a.size() - 1) c[i].x = a[i];
        fp(i, 0, b.size() - 1) c[i].y = b[i];
        DFT(c), dot(c, c), IDFT(c), a.resize(n);
        fp(i, 0, n - 1) a[i] = (ll)(c[i].y * .5 + .5);
        return a;
    }

    Poly &operator+=(Poly &a, Poly b) {
        a.resize(max(a.size(), b.size()));
        fp(i, 0, b.size() - 1) a[i] += b[i];
        return a;
    }

    Poly operator+(Poly a, Poly b) {
        return a += b;
    }

    Poly &operator-=(Poly &a, Poly b) {
        a.resize(max(a.size(), b.size()));
        fp(i, 0, b.size() - 1) a[i] -= b[i];
        return a;
    }

    Poly operator-(Poly a, Poly b) {
        return a -= b;
    }

    Poly &operator*=(Poly &a, int b) {
        for(auto &e: a) e *= b;
        return a;
    }
    Poly operator*(Poly a, int b) {
        return a *= b;
    }

    Poly &operator/=(Poly &a, int b) {
        for(auto &e: a) e /= b;
        return a;
    }
    Poly operator/(Poly a, int b) {
        return a /= b;
    }
}
/*--------------------*/
using namespace Polynomial;
```

5.4.1 任意模数 NTT

luogu.com.cn/problem/P4245

```
signed main(){
    int n, m, P; cin>>n>>m>>P;
    Poly A(n+1), B(m+1);
    rep(i, 0, n) read(A[i]);
    rep(i, 0, m) read(B[i]);
    A=MTT::conv(A, B, P);
    for(auto i: A) cout<<i<<' ';

    return 0;
}
```

5.4.2 多项式乘法（FFT）

luogu.com.cn/problem/P3803

```
void solve(){
  int n, m; cin>>n>>m;
    Poly A(n+1), B(m+1);
    rep(i, 0, n) read(A[i]);
    rep(i, 0, m) read(B[i]);
    A=A*B;
    for(auto i: A) cout<<i<<' ';
}
```

5.5 分治 FFT

例题：luogu.com.cn/problem/P4721

给定序列 $g_{1\dots n-1}$，求序列 $f_{0\dots n-1}$。其中 $f_i=\sum_{j=1}^{i} f_{i-j}g_j$，边界为 $f_0=1$。

答案对 998244353 取模。

```
// 注意这里接的是 *NTT 相关* 部分的代码!
using namespace Polynomial;

int n;
int f[N], g[N];

void divi(int l, int r){
  if(l>=r) return;
  int mid=l+r>>1;
  divi(l, mid);

  // init
  int len=r-l+1;
  Poly A(len), B(len);
  rep(i, l, mid) A[i-l]=f[i];
  rep(i, 1, r-l) B[i-1]=g[i];
  A=A*B;
  rep(i, mid+1, r) (f[i]+=A[i-l-1])%=P;
  divi(mid+1, r);
}

signed main(){
  cin>>n; n--;
  f[0]=1;
  rep(i, 1, n) scanf("%lld", g+i);
  divi(0, n);
  rep(i, 0, n) printf("%lld ", f[i]);
  return 0;
}
```

5.6 FWT

5.6.1 做法

https://www.luogu.com.cn/problem/P4717

给定长度为 2^n 两个序列 A, B，设

$$C_i=\sum_{j\oplus k=i} A_j\times B_k$$

分别当 $\oplus$ 是 or,and,xor 时求出 C

思路都是：将 $A, B\to fwt(A), fwt(B)$，然后通过 $fwt(C)=fwt(A)\cdot fwt(B)$，最后通过逆变换得到 C。

对于 or，$fwt(a)_i=\sum_{j|i=i} a_j$，也就是经过变换后得到的是子集和。

对于 and，$fwt(a)_i=\sum_{j|i=j} a_j$，也就是经过变换后得到的是超集和。

```
#define int long long

const int N=(1<<17)+50, mod=998244353, inv2=499122177;

int A[N], B[N], a[N], b[N];
int n;

int add(int a, int b){
    return ((a+b)%mod+mod)%mod;
}

int mul(int a, int b){
    return a*b%mod;
}

void copy(){
    memcpy(a, A, sizeof a);
    memcpy(b, B, sizeof b);
}

void OR(int *w, int inv){
    for(int o=2, k=1; o<=n; o<<=1, k<<=1)
        for(int i=0; i<n; i+=o)
            for(int j=0; j<k; j++)
                w[i+j+k]=add(w[i+j+k], mul(w[i+j], inv));
}

void AND(int *w, int inv){
    for(int o=2, k=1; o<=n; o<<=1, k<<=1)
        for(int i=0; i<n; i+=o)
            for(int j=0; j<k; j++)
                w[i+j]=add(w[i+j], mul(w[i+j+k], inv));
}

void XOR(int *w, int inv){
    for(int o=2, k=1; o<=n; o<<=1, k<<=1)
        for(int i=0; i<n; i+=o)
            for(int j=0; j<k; j++)
                w[i+j]=add(w[i+j], w[i+j+k]),
                w[i+j+k]=add(w[i+j]-w[i+j+k], -w[i+j+k]),
                w[i+j]=mul(w[i+j], inv), w[i+j+k]=mul(w[i+j+k],
inv);
}

void dot(int *a, int *b){
    for(int i=0; i<n; i++) a[i]=a[i]*b[i]%mod;
}

void out(){
    for(int i=0; i<n; i++) cout<<a[i]<<' ';
    cout<<endl;
}

signed main(){
    ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);
    cin>>n;
    n=1<<n;
    for(int i=0; i<n; i++) cin>>A[i];
    for(int i=0; i<n; i++) cin>>B[i];

    copy();
    OR(a, 1); OR(b, 1);
    dot(a, b);
    OR(a, mod-1);
    out();

    copy();
    AND(a, 1); AND(b, 1);
    dot(a, b);
    AND(a, mod-1);
    out();

    copy();
    XOR(a, 1); XOR(b, 1);
    dot(a, b);
    XOR(a, inv2);
    out();

    return 0;
}
```

> 当然求子集和超集和也可以绕开 FWT，这里顺便给出 DP 做法。
>
> 初始化就是：
>
> ```
> rep(st, 0, (1<<n)-1) f[st]=w[st];
> ```
>
> 子集和：
>
> ```
> rep(i, 0, n-1) rep(st, 0, (1<<n)-1) if(st>>i&1)
> f[st]+=f[st^(1<<i)];
> ```
>
> 超集和：
>
> ```
> rep(j, 0, n-1) dwn(st, (1<<n)-1, 0) if(st>>j&1)
> f[st^(1<<j)]+=f[st];
> ```

6 计算几何

杜老师的计算几何模板。

```
#define pb push_back
#define lop(i, a, b) for (int i = (a); i < (b); i++)
#define el '\n'

typedef long long LL;
constexpr int N = 1e5 + 10, md = 1e9 + 7;

typedef double db;
const db EPS = 1e-9;

inline int sign(db a) { return a < -EPS ? -1 : a > EPS; }

inline int cmp(db a, db b) { return sign(a - b); }

struct P {
  db x, y;
  P() {}
  P(db x, db y) : x(x), y(y) {}
  P operator+(P p) { return {x + p.x, y + p.y}; }
  P operator-(P p) { return {x - p.x, y - p.y}; }
  P operator*(db d) { return {x * d, y * d}; }
  P operator/(db d) { return {x / d, y / d}; }

  bool operator<(P p) const {
    int c = cmp(x, p.x);
    if (c)
      return c == -1;
    return cmp(y, p.y) == -1;
  }

  bool operator==(P o) const { return cmp(x, o.x) == 0 && cmp(y,
o.y) == 0; }

  db dot(P p) { return x * p.x + y * p.y; } // 点积
  db det(P p) { return x * p.y - y * p.x; } // 叉积

  db distTo(P p) { return (*this - p).abs(); } // 两点之间的距离
  db alpha() { return atan2(y, x); } // 本根文角：正半轴转多少°到该点
[-pi,pi]
  friend std::istream &operator>>(std::istream &is, P &a) {
    return is >> a.x >> a.y;
  }
  friend std::ostream &operator<<(std::ostream &os, const P &a) {
    return os << "(" << a.x << "," << a.y << ")";
  }
  db abs() { return sqrt(abs2()); }
  db abs2() { return x * x + y * y; }
```

```
  P rot90() { return P(-y, x); }     // 逆时针转90°
  P unit() { return *this / abs(); } // 方向向量
  // 极角在上半轴还是在下半轴
  int quad() const { return sign(y) == 1 || (sign(y) == 0 && sign(x)
>= 0); }
  P rot(db an) {
    return {x * cos(an) - y * sin(an), x * sin(an) + y * cos(an)};
  } // 转一个角度an
};

struct L { // ps[0] -> ps[1]
  P ps[2];
  P &operator[](int i) { return ps[i]; }
  P dir() { return ps[1] - ps[0]; }
  L(P a, P b) {
    ps[0] = a;
    ps[1] = b;
  }
  bool include(P p) { return sign((ps[1] - ps[0]).det(p - ps[0])) >
0; }
  L push() { // push eps outward
    const double eps = 1e-8;
    P delta = (ps[1] - ps[0]).rot90().unit() * eps;
    return {ps[0] + delta, ps[1] + delta};
  }
};

#define cross(p1, p2, p3)
\
  ((p2.x - p1.x) * (p3.y - p1.y) -
\
   (p3.x - p1.x) * (p2.y - p1.y)) // 三个点p1p2 X p1p3
#define crossOp(p1, p2, p3)
\
  sign(cross(p1, p2, p3)) //=0 三点共线, <0 顺时针旋转, >0 逆时针旋转

// 两条直线是否相交
bool chkLL(P p1, P p2, P q1, P q2) {
  db a1 = cross(q1, q2, p1), a2 = -cross(q1, q2, p2);
  return sign(a1 + a2) != 0;
}

// 求直线的交点
P isLL(P p1, P p2, P q1, P q2) {
  db a1 = cross(q1, q2, p1), a2 = -cross(q1, q2, p2);
  return (p1 * a2 + p2 * a1) / (a1 + a2);
}

P isLL(L l1, L l2) { return isLL(l1[0], l1[1], l2[0], l2[1]); }

// 判断 区间 [l1,r1],[l2,r2] 是否相交(l1,r1)大小关所谓
bool intersect(db l1, db r1, db l2, db r2) {
  if (l1 > r1)
    swap(l1, r1);
  if (l2 > r2)
    swap(l2, r2);
  return !(cmp(r1, l2) == -1 || cmp(r2, l1) == -1);
}

// 线段相交
bool isSS(P p1, P p2, P q1, P q2) {
  return intersect(p1.x, p2.x, q1.x, q2.x) &&
         intersect(p1.y, p2.y, q1.y, q2.y) &&
         crossOp(p1, p2, q1) * crossOp(p1, p2, q2) <= 0 &&
         crossOp(q1, q2, p1) * crossOp(q1, q2, p2) <= 0;
}

// 线段严格相交(不能交到端点且有且只能有一个交点)
bool isSS_strict(P p1, P p2, P q1, P q2) {
  return crossOp(p1, p2, q1) * crossOp(p1, p2, q2) < 0 &&
         crossOp(q1, q2, p1) * crossOp(q1, q2, p2) < 0;
}

// m 在 a和b 之间
bool isMiddle(db a, db m, db b) {
  return sign(a - m) == 0 || sign(b - m) == 0 || (a < m != b < m);
}

// 点m是不是横纵都在a和b之间
bool isMiddle(P a, P m, P b) {
  return isMiddle(a.x, m.x, b.x) && isMiddle(a.y, m.y, b.y);
}

// 点p在线段p1p2上
bool onSeg(P p1, P p2, P q) {
  return crossOp(p1, p2, q) == 0 && isMiddle(p1, q, p2);
}

// 点q严格在p1p2上(不包含端点)
bool onSeg_strict(P p1, P p2, P q) {
  return crossOp(p1, p2, q) == 0 &&
         sign((q - p1).dot(p1 - p2)) * sign((q - p2).dot(p1 - p2)) <
0;
}

// 求q到直线p1p2的投影(垂足): p1!=p2
P proj(P p1, P p2, P q) {
  P dir = p2 - p1;
  return p1 + dir * (dir.dot(q - p1) / dir.abs2());
}

// 求q以直线p1p2为轴的反射
P reflect(P p1, P p2, P q) { return proj(p1, p2, q) * 2 - q; }

// 求q到线段p1p2的最小距离
db nearest(P p1, P p2, P q) {
  if (p1 == p2)
    return p1.distTo(q);
  P h = proj(p1, p2, q);
  if (isMiddle(p1, h, p2))
    return q.distTo(h);
  return min(p1.distTo(q), p2.distTo(q));
}

// 求线段p1p2与线段q1q2的距离
db disSS(P p1, P p2, P q1, P q2) {
  if (isSS(p1, p2, q1, q2))
    return 0;
  return min(min(nearest(p1, p2, q1), nearest(p1, p2, q2)),
             min(nearest(q1, q2, p1), nearest(q1, q2, p2)));
}

// 极角排序
/*
    sort(p,p+n,[&](const P &a, const P &b) {
        int qa = a.quad(), qb = b.quad();
        if(qa != qb) return qa < qb;
        else return sign(a.det(b)) > 0;
    });
*/

db rad(P p1, P p2) { return atan2l(p1.det(p2), p1.dot(p2)); }

db incircle(P p1, P p2, P p3) {
  db A = p1.distTo(p2);
  db B = p2.distTo(p3);
  db C = p3.distTo(p1);
  return sqrtl(A * B * C / (A + B + C));
}

// 多边形
// 面积
db area(vector<P> ps) {
  db ret = 0;
  lop(i, 0, ps.size()) ret += ps[i].det(ps[(i + 1) % ps.size()]);
  return ret / 2;
}
// 点包含
int contain(vector<P> ps, P p) { // 2:inside,1:on_seg,0:outside
  int n = ps.size(), ret = 0;
  lop(i, 0, n) {
    P u = ps[i], v = ps[(i + 1) % n];
    if (onSeg(u, v, p))
      return 1;
    if (cmp(u.y, v.y) <= 0)
      swap(u, v);
    if (cmp(p.y, u.y) > 0 || cmp(p.y, v.y) <= 0)
      continue;
    ret ^= crossOp(p, u, v) > 0;
  }
  return ret * 2;
}
// 严格凸包
vector<P> convexHull(vector<P> ps) {
  int n = ps.size();
  if (n <= 1)
    return ps;
  sort(ps.begin(), ps.end());
  vector<P> qs(n * 2);
  int k = 0;
  for (int i = 0; i < n; qs[k++] = ps[i++])
    while (k > 1 && crossOp(qs[k - 2], qs[k - 1], ps[i]) <= 0)
      --k;
  for (int i = n - 2, t = k; i >= 0; qs[k++] = ps[i--])
    while (k > t && crossOp(qs[k - 2], qs[k - 1], ps[i]) <= 0)
      --k;
  qs.resize(k - 1);
  return qs;
}
// 不严格凸包
vector<P> convexHullNonStrict(vector<P> ps) {
  // caution: need to unique the Ps first
  int n = ps.size();
  if (n <= 1)
    return ps;
  sort(ps.begin(), ps.end());
  vector<P> qs(n * 2);
  int k = 0;
  for (int i = 0; i < n; qs[k++] = ps[i++])
    while (k > 1 && crossOp(qs[k - 2], qs[k - 1], ps[i]) < 0)
      --k;
  for (int i = n - 2, t = k; i >= 0; qs[k++] = ps[i--])
    while (k > t && crossOp(qs[k - 2], qs[k - 1], ps[i]) < 0)
      --k;
  qs.resize(k - 1);
  return qs;
}
// 旋转卡壳
db convexDiameter(vector<P> ps) {
  int n = ps.size();
  if (n <= 1)
    return 0;
  int is = 0, js = 0;
  lop(k, 1, n) is = ps[k] < ps[is] ? k : is, js = ps[js] < ps[k] ?
k : js;
  int i = is, j = js;
  db ret = ps[i].distTo(ps[j]);
  do {
    if ((ps[(i + 1) % n] - ps[i]).det(ps[(j + 1) % n] - ps[j]) >= 0)
      (++j) %= n;
    else
      (++i) %= n;
    ret = max(ret, ps[i].distTo(ps[j]));
  } while (i != is || j != js);
  return ret;
}
// 切多边形
vector<P> convexCut(const vector<P> &ps, P q1, P q2) {
  vector<P> qs;
  int n = ps.size();
  lop(i, 0, n) {
    P p1 = ps[i], p2 = ps[(i + 1) % n];
    int d1 = crossOp(q1, q2, p1), d2 = crossOp(q1, q2, p2);
    if (d1 >= 0)
      qs.pb(p1);
    if (d1 * d2 < 0)
      qs.pb(isLL(p1, p2, q1, q2));
  }
  return qs;
}

// min_dist

db min_dist(vector<P> &ps, int l, int r) {
  if (r - l <= 5) {
    db ret = 1e100;
    lop(i, l, r) lop(j, l, i) ret = min(ret, ps[i].distTo(ps[j]));
    return ret;
  }
  int m = (l + r) >> 1;
  db ret = min(min_dist(ps, l, m), min_dist(ps, m, r));
  vector<P> qs;
  lop(i, l, r) if (abs(ps[i].x - ps[m].x) <= ret) qs.pb(ps[i]);
  sort(qs.begin(), qs.end(), [](P a, P b) -> bool { return a.y <
b.y; });
  lop(i, 1, qs.size()) for (int j = i - 1; j >= 0 && qs[j].y >=
qs[i].y - ret;
                            --j) ret = min(ret,
qs[i].distTo(qs[j]));
  return ret;
}

// 圆和圆的关系
int type(P o1, db r1, P o2, db r2) {
  db d = o1.distTo(o2);
  if (cmp(d, r1 + r2) == 1)
    return 4;
  if (cmp(d, r1 + r2) == 0)
    return 3;
  if (cmp(d, abs(r1 - r2)) == 1)
    return 2;
  if (cmp(d, abs(r1 - r2)) == 0)
    return 1;
  return 0;
}

// 圆和直线相交
vector<P> isCL(P o, db r, P p1, P p2) {
  if (cmp(abs((o - p1).det(p2 - p1) / p1.distTo(p2)), r) > 0)
    return {};
  db x = (p1 - o).dot(p2 - p1), y = (p2 - p1).abs2(),
     d = x * x - y * ((p1 - o).abs2() - r * r);
  d = max(d, (db)0.0);
  P m = p1 - (p2 - p1) * (x / y), dr = (p2 - p1) * (sqrt(d) / y);
  return {m - dr, m + dr}; // along dir: p1->p2
}

// 圆和圆相交
vector<P> isCC(P o1, db r1, P o2,
               db r2) { // need to check whether two circles are the
same
  db d = o1.distTo(o2);
  if (cmp(d, r1 + r2) == 1)
    return {};
  if (cmp(d, abs(r1 - r2)) == -1)
    return {};
  d = min(d, r1 + r2);
  db y = (r1 * r1 + d * d - r2 * r2) / (2 * d), x = sqrt(r1 * r1 - y
* y);
  P dr = (o2 - o1).unit();
  P q1 = o1 + dr * y, q2 = dr.rot90() * x;
  return {q1 - q2, q1 + q2}; // along circle 1第一个圆逆时针方向
}

// (外公切线)extanCC,(内公切线)intanCC:-r2,(点到圆的切线)tanCP:r2=0
vector<pair<P, P>> tanCC(P o1, db r1, P o2, db r2) {
  P d = o2 - o1;
  db dr = r1 - r2, d2 = d.abs2(), h2 = d2 - dr * dr;
  if (sign(d2) == 0 || sign(h2) < 0)
    return {};
  h2 = max(0.0, h2);
  vector<pair<P, P>> ret;
  for (db sign : {-1, 1}) {
    P v = (d * dr + d.rot90() * sqrt(h2) * sign) / d2;
    ret.push_back({o1 + v * r1, o2 + v * r2});
  }
  if (sign(h2) == 0)
    ret.pop_back();
  return ret;
}

vector<P> tanCP(P o, db r, P p) {
  db x = (p - o).abs2(), d = x - r * r;
  if (sign(d) <= 0)
    return {}; // on circle => no tangent
  P q1 = o + (p - o) * (r * r / x);
  P q2 = (p - o).rot90() * (r * sqrt(d) / x);
  return {q1 - q2, q1 + q2}; // counter clock-wise
}

vector<L> extanCC(P o1, db r1, P o2, db r2) {
  vector<L> ret;
  if (cmp(r1, r2) == 0) {
    P dr = (o2 - o1).unit().rot90() * r1;
    ret.pb(L(o1 + dr, o2 + dr)), ret.pb(L(o1 - dr, o2 - dr));
  } else {
    P p = (o2 * r1 - o1 * r2) / (r1 - r2);
    vector<P> ps = tanCP(o1, r1, p), qs = tanCP(o2, r2, p);
    lop(i, 0, min(ps.size(), qs.size()))
        ret.pb(L(ps[i], qs[i])); // c1 counter-clock wise
  }
  return ret;
}

vector<L> intanCC(P o1, db r1, P o2, db r2) {
  vector<L> ret;
  P p = (o1 * r2 + o2 * r1) / (r1 + r2);
  vector<P> ps = tanCP(o1, r1, p), qs = tanCP(o2, r2, p);
  lop(i, 0, min(ps.size(), qs.size()))
      ret.pb(L(ps[i], qs[i])); // c1 counter-clock wise
  return ret;
}

db areaCT(db r, P p1, P p2) {
  vector<P> is = isCL(P(0, 0), r, p1, p2);
  if (is.empty())
    return r * r * rad(p1, p2) / 2;
  bool b1 = cmp(p1.abs2(), r * r) == 1, b2 = cmp(p2.abs2(), r * r)
== 1;
  if (b1 && b2) {
    if (sign((p1 - is[0]).dot(p2 - is[0])) <= 0 &&
        sign((p1 - is[0]).dot(p2 - is[0])) <= 0)
      return r * r * (rad(p1, is[0]) + rad(is[1], p2)) / 2 +
             is[0].det(is[1]) / 2;
    else
      return r * r * rad(p1, p2) / 2;
  }
  if (b1)
    return (r * r * rad(p1, is[0]) + is[0].det(p2)) / 2;
  if (b2)
    return (p1.det(is[1]) + r * r * rad(is[1], p2)) / 2;
  return p1.det(p2) / 2;
}
// 两条直线是否平行
bool parallel(L l0, L l1) { return sign(l0.dir().det(l1.dir())) ==
0; }
// 两条直线平行、同向
bool sameDir(L l0, L l1) {
  return parallel(l0, l1) && sign(l0.dir().dot(l1.dir())) == 1;
}

// 极角排序规则
bool cmp(P a, P b) {
  if (a.quad() != b.quad()) {
    return a.quad() < b.quad();
  } else {
    return sign(a.det(b)) > 0;
  }
}

bool operator<(L l0, L l1) {
  if (sameDir(l0, l1)) {
    return l1.include(l0[0]);
  } else {
    return cmp(l0.dir(), l1.dir());
  }
}

bool check(L u, L v, L w) { return w.include(isLL(u, v)); }

vector<P> halfPlaneIS(vector<L> &l) {
  sort(l.begin(), l.end());
  deque<L> q;
  for (int i = 0; i < (int)l.size(); ++i) {
    if (i && sameDir(l[i], l[i - 1]))
      continue;
    while (q.size() > 1 && !check(q[q.size() - 2], q[q.size() - 1],
l[i]))
```

```
    q.pop_back();
  while (q.size() > 1 && !check(q[1], q[0], l[i]))
    q.pop_front();
  q.push_back(l[i]);
 }
 while (q.size() > 2 && !check(q[q.size() - 2], q[q.size() - 1],
q[0]))
  q.pop_back();
 while (q.size() > 2 && !check(q[1], q[0], q[q.size() - 1]))
  q.pop_front();
 vector<P> ret;
 for (int i = 0; i < (int)q.size(); ++i)
  ret.push_back(isLL(q[i], q[(i + 1) % q.size()]));
 return ret;
}

// 内心
P inCenter(P A, P B, P C) {
 double a = (B - C).abs(), b = (C - A).abs(), c = (A - B).abs();
 return (A * a + B * b + C * c) / (a + b + c);
}

// 外心
P circumCenter(P a, P b, P c) {
 P bb = b - a, cc = c - a;
 double db = bb.abs2(), dc = cc.abs2(), d = 2 * bb.det(cc);
 return a - P(bb.y * dc - cc.y * db, cc.x * db - bb.x * dc) / d;
}

// 垂心
P othroCenter(P a, P b, P c) {
 P ba = b - a, ca = c - a, bc = b - c;
 double Y = ba.y * ca.y * bc.y, A = ca.x * ba.y - ba.x * ca.y,
        x0 = (Y + ca.x * ba.y * b.x - ba.x * ca.y * c.x) / A,
        y0 = -ba.x * (x0 - c.x) / ba.y + ca.y;
 return {x0, y0};
}
```

7 建图

最常用的是邻接表、链式前向星。

7.1 邻接表

直接开 vector<int> g[N] 即可，加边 (u,v) 例如 g[u].push_back(v)。一个显然的优点是代码短。

7.2 链式前向星

使用时记得初始化 h[] 为 -1，因为默认边数 tot 从 0 开始编号。它的优点包括可以简单地访问某个特定编号的边，而且在使用于网络流或者任何需要求反向边的时候可以通过将某边编号异或 1 来得到它的反向边（个人认为最方便的地方就是这个）。

```
struct Edge{
  int to, next;
}e[M];

int h[N], tot;

void add(int u, int v){
  e[tot].to=v, e[tot].next=h[u], h[u]=tot++;
}
```

8 判负环

有些方法跑得比较慢或者甚至不正确，这里推荐一个利用最短路边数的判断方法。

```
bool spfa(){
  memset(vis, false, sizeof vis);
  memset(cnt, 0, sizeof cnt);
  memset(d, 0x3f, sizeof d);
  queue<int> q;
  q.push(1);
  vis[1]=true;
  d[1]=0;

  while(q.size()){
    int u=q.front();
    q.pop();
    vis[u]=false;
    for(int i=h[u]; ~i; i=e[i].next){
      int go=e[i].to;
      if(d[go]>d[u]+e[i].w){
        d[go]=d[u]+e[i].w;
        cnt[go]=cnt[u]+1;
        if(cnt[go]>=n) return true;

        if(!vis[go]){
          vis[go]=true;
          q.push(go);
        }
      }
    }
  }

  return false;
}
```

9 差分约束

> 给出一组包含 m 个不等式，有 n 个未知数的
>
> 形如：$x_i - x_j \le y$ 的不等式组，求任意一组满足这个不等式组的解。

做法：最短路的结果对应不等式的解。可以发现求完最短路后满足 $d_v \le d_u + y$，所以图中的边 (u,v,y) 对应原不等式的 (j,i,y)。而如果题目的约束是 $x_i - x_j \ge y$ 的话，就类似地去求最长路即可。

```
bool spfa(){
  memset(d, 0x3f, sizeof d);
  memset(vis, false, sizeof vis);
  memset(cnt, 0, sizeof cnt);

  queue<int> q;
  // 建图时按照题意不同要求可以利用 0 号点来限制其他点范围。
  q.push(0);
  vis[0]=true;
  d[0]=0;

  while(q.size()){
    int u=q.front(); q.pop();
    vis[u]=false;
    for(int i=h[u]; ~i; i=e[i].next){
      int go=e[i].to;
      if(d[go]>d[u]+e[i].w){
        d[go]=d[u]+e[i].w;
        cnt[go]=cnt[u]+1;
        if(cnt[go]>=n+1) return true; // n+1 是因为 0 号点也计入考虑

        if(!vis[go]){
          vis[go]=true;
          q.push(go);
        }
      }
    }
  }
  return false;
}
```

10 Kruskal 重构树

就是利用做生成树的过程构建一棵新树（即 Kruskal 重构树）。而在做最小生成树时，原树两个点路径上边权的最大值为重构树上对应 LCA 的点权。

做法是：

```
struct Edge{
  int to, next;
}e[M];

int h[N], tot;

void add(int u, int v){
  e[tot].to=v, e[tot].next=h[u], h[u]=tot++;
}

struct Buf{
  int u, v, w;

  bool operator < (const Buf &o)const{
    return w>o.w;
  }
}E[M];

int f[N];
int find(int x){
  return x==f[x]? x: f[x]=find(f[x]);
}

int val[N];

int main(){
  cin>>n>>m;
  rep(i, 1, m){
    int u, v, w; read(u), read(v), read(w);
    E[i]={u, v, w};
  }

  sort(E+1, E+1+m);

  rep(i, 1, 2*n) f[i]=i, h[i]=-1;

  rep(i, 1, m){
    int u=E[i].u, v=E[i].v, w=E[i].w;
    int pu=find(u), pv=find(v);
    if(pu!=pv){
      ++n;
      f[pu]=f[pv]=n;
      val[n]=w;
      add(n, pu), add(pu, n);
      add(n, pv), add(pv, n);
    }
  }
  return 0;
}
```

11 欧拉路径及欧拉回路

欧拉路径（Eulerian path）是经过图中每条边恰好一次的路径，欧拉回路（Eulerian circuit）是经过图中每条边恰好一次的回路。如果一个图中存在欧拉回路，则这个图被称为欧拉图（Eulerian graph）。

性质：

对于连通图 G，以下三个性质是互相等价的：

- G 是欧拉图；
- G 中所有顶点的度数都是偶数（对于有向图，每个顶点的入度等于出度）；
- G 可被分解为若干条不共边回路的并。

例题 1：

https://www.luogu.com.cn/problem/P7771

求有向图字典序最小的欧拉路径。

```
int n, m;
vector<pii> g[N]; // pair 的 second 为边的编号
int din[N], dout[N];
int S, T;
int cur[N];
bool vis[M];

stack<int> res;

void dfs(int u){
  for(int i=cur[u]; i<g[u].size(); i=max(i+1, cur[u])){
    auto [go, id]=g[u][i];
    if(!vis[id]){
      vis[id]=true;
      cur[u]=i+1;
      dfs(go);
    }
  }
  res.push(u);
}

signed main(){
  cin>>n>>m;
  rep(i, 1, m){
    int u, v; read(u), read(v);
    dout[u]++, din[v]++;
    g[u].pb({v, i});
  }

  int chk=0;
  rep(i, 1, n){
    if(din[i]!=dout[i]){
      chk++;
      if(din[i]+1==dout[i]) S=i;
      if(dout[i]+1==din[i]) T=i;
    }
  }
  if(!(!chk || chk==2)) return puts("No"), 0;
  if(!chk) S=T=1;
  rep(i, 1, n) sort(all(g[i]));
  dfs(S);
  while(res.size()){
    cout<<res.top()<<' ';
    res.pop();
  }

  return 0;
}
```

例题 2：

https://uoj.ac/problem/117

有一天一位灵魂画师画了一张图，现在要你找出欧拉回路，即在图中找一个环使得每条边都在环上出现恰好一次。

一共两个子任务：

这张图是无向图。
这张图是有向图。

```
const int N=1e5+5, M=4e5+5;

struct Edge{
  int to, next;
}e[M];

int h[N], tot;

void add(int u, int v){
  e[tot].to=v, e[tot].next=h[u], h[u]=tot++;
}

int n, m, OP; // OP=1 undirected; OP=2 directed;
vector<int> stk;
bool vis[M>>1];
int deg[N];

void dfs(int u){
  for(int i=h[u]; ~i; i=h[u]){ // 注意这里跳的是 h[u]
    int eid=(OP==1? i>>1: i)+1; // 边读入顺序的编号, [1, m]
    if(!vis[eid]){
      vis[eid]=true;
      h[u]=e[i].next;
      dfs(e[i].to);
      if(OP==1) stk.pb((i&1? -1: 1)*eid);
      else stk.pb(eid);
    }
    else h[u]=e[i].next;
  }
}

void solve(){
  cin>>OP>>n>>m;
  rep(u, 1, n) h[u]=-1;
  rep(i, 1, m){
    int u, v; scanf("%d%d", &u, &v);
    add(u, v);
    if(OP==1) add(v, u);
    if(OP==1) ++deg[u], ++deg[v];
    else ++deg[u], --deg[v];
  }
  if(OP==1) rep(i, 1, n) if(deg[i]&1) return void(puts("NO"));
  if(OP==2) rep(i, 1, n) if(deg[i]) return void(puts("NO"));
  int cc=0;
  rep(u, 1, n) if(~h[u]) dfs(u), ++cc;
  if(cc>1) return void(puts("NO"));
  puts("YES");
  {
    reverse(all(stk));
    for(auto e: stk) cout<<e<<" ";
  }
 }

signed main(){
  solve();
  return 0;
}
```

12 树链剖分

最常用的是重链剖分。有下面的性质：

- 任意点属于且仅属于一条重链。
- 重链内的 DFS 序连续。
- 树上的每条路径都可以拆分成 $O(\log n)$ 条重链。

```
int dep[N], sz[N], son[N], fa[N]; // 维护深度、子树大小、重儿子、父亲
void dfs1(int u, int f, int d){
    dep[u]=d, sz[u]=1, fa[u]=f;
    for(int i=h[u]; ~i; i=e[i].next){
        int go=e[i].to;
        if(go==f) continue;
        dfs1(go, u, d+1);
        if(sz[go]>sz[son[u]]) son[u]=go;
        sz[u]+=sz[go];
    }
}

int top[N]; // 点所在树链的顶端
void dfs2(int u, int t){
    top[u]=t;
    if(!son[u]) return;
    dfs2(son[u], t);
    for(int i=h[u]; ~i; i=e[i].next){
        int go=e[i].to;
        if(go==fa[u] || go==son[u]) continue;
        dfs2(go, go);
    }
}
```

13 LCA

给出三种实现方法。

13.1 倍增实现

复杂度 $O(\log n)$。

```
int p[N][M];
int d[N];
// dfs 处理出深度 d 和 u 的 2^j 级祖先节点 p[u][j]
void dfs(int u, int fa){
  d[u]=d[fa]+1;
  p[u][0]=fa;
  rep(i, 1, M-1) p[u][i]=p[p[u][i-1]][i-1];
  for(int i=h[u]; ~i; i=e[i].next) if(e[i].to!=fa) dfs(e[i].to, u);
}

int lca(int x, int y){
  if(d[x]<d[y]) swap(x, y);
  dwn(i, M-1, 0) if(d[x]-(1<<i)>=d[y]) x=p[x][i];
  if(x==y) return x;
  dwn(i, M-1, 0){
    if(p[x][i]!=p[y][i]){
      x=p[x][i];
      y=p[y][i];
    }
  }
  return p[x][0];
}
```

13.2 树链剖分实现

复杂度 $O(\log n)$，常数很小。

```
int lca(int u, int v){
    while(top[u]!=top[v]){
        if(dep[top[u]]<dep[top[v]]) swap(u, v);
        u=fa[top[u]];
    }
    if(dep[u]<dep[v]) return u;
    return v;
}
```

13.3 ST 表实现

预处理 $O(n\log n)$，查询 $O(1)$。

```
int st[N][20];
int dfn[N], idx, dep[N];
int lg[N];

void dfs(int u, int fa){
  st[dfn[u]=++idx][0]=fa;
  for(int i=h[u]; ~i; i=e[i].next){
    int go=e[i].to;
    if(go==fa) continue;
    dep[go]=dep[u]+1;
    dfs(go, u);
  }
}

int cmp(int u, int v){
  return dep[u]<dep[v]? u: v;
}

void build(){
  lg[1]=0;
  rep(i, 2, n) lg[i]=lg[i>>1]+1;
  rep(j, 1, 19){
    for(int i=1; i+(1<<j)-1<=n; i++){
      st[i][j]=cmp(st[i][j-1], st[i+(1<<j-1)][j-1]);
    }
  }
}

int lca(int u, int v){
  if(u==v) return u;
  int l=min(dfn[u], dfn[v]), r=max(dfn[u], dfn[v]);
  int k=lg[r-l++];
  return cmp(st[l][k], st[r-(1<<k)+1][k]);
}
```

14 树上启发式合并

做法：

- 预处理出重儿子 son[]。
- 进行 DFS，先向非重儿子方向走，且不保留它们带来的影响。
- 再向重儿子方向走并保留其影响。
- 最后更新当前点答案即可。

例题：

> 给你一棵有根树，根位于顶点 1。
>
> 每个点都有种颜色。如果在点 u 的子树中，没有其他颜色比颜色 c 出现的次数更多，那么我们就称颜色 c 在点 u 的子树中是主要颜色。
>
> 对于每个顶点 u，找出点 u 子树上所有主要颜色的总和。

```
int sz[N], son[N];

void dfs(int u, int fa){
  sz[u]=1;
  for(int i=h[u]; ~i; i=e[i].next){
    int go=e[i].to;
    if(go==fa) continue;
    dfs(go, u);
    sz[u]+=sz[go];
    if(sz[go]>sz[son[u]]) son[u]=go;
  }
}

int res[N];
int cnt[N], ans, mx;

void upd(int u, int fa, int d){
  cnt[w[u]]+=d;
  if(d==1){
    if(cnt[w[u]]>mx) mx=cnt[w[u]], ans=w[u];
    else if(cnt[w[u]]==mx) ans+=w[u];
  }

  for(int i=h[u]; ~i; i=e[i].next){
    int go=e[i].to;
    if(go==fa) continue;
    upd(go, u, d);
  }
}

void dfs2(int u, int fa, bool del){
  for(int i=h[u]; ~i; i=e[i].next){
    int go=e[i].to;
    if(go==fa || go==son[u]) continue;
    dfs2(go, u, 1);
  }
  if(son[u]) dfs2(son[u], u, 0);

  cnt[w[u]]++;
  if(cnt[w[u]]>mx) mx=cnt[w[u]], ans=w[u];
  else if(cnt[w[u]]==mx) ans+=w[u];
  for(int i=h[u]; ~i; i=e[i].next){
    int go=e[i].to;
    if(go==fa || go==son[u]) continue;
    upd(go, u, 1);
  }
  res[u]=ans;

  if(del) upd(u, fa, -1), mx=ans=0;
}

signed main(){
  memset(h, -1, sizeof h);
  cin>>n;
  rep(i, 1, n) read(w[i]);

  rep(i, 1, n-1){
    int u, v; read(u), read(v);
    add(u, v), add(v, u);
  }

  dfs(1, -1);
  dfs2(1, -1, 0);

  rep(i, 1, n) cout<<res[i]<<' ';
  cout<<endl;

  return 0;
}
```

例题 2：

https://www.luogu.com.cn/problem/P3806（注意这道题我并没有放错地方，这里提供的是树上启发式合并的做法）

给定一棵有 n 个点的树，询问树上距离为 k 的点对是否存在。

```
const int N=10010;

int n, Q;
int qs[N], res[N];
vector<pii> g[N];
unordered_map<int, int> b;

int son[N], sz[N], d[N];
int lef[N], rig[N], ts, idfn[N];

void dfs0(int u, int fa){
  sz[u]=1;
  lef[u]=++ts;
  idfn[ts]=u;
  for(auto [go, w]: g[u]) if(go!=fa){
    d[go]=d[u]+w;
    dfs0(go, u);
    sz[u]+=sz[go];
    if(sz[go]>sz[son[u]]) son[u]=go;
  }
  rig[u]=ts;
}

void ask(int u, int v){
  rep(i, 1, Q){
    int q=qs[i];
    int x=q-d[v]+d[u]*2;
    if(b.count(x)) res[i]=true;
  }
}

void upd(int x, int typ){
  if(typ==-1){
    if(--b[x]==0) b.erase(x);
  }
  else ++b[x];
}

void dfs(int u, int fa, bool del){
  for(auto [go, w]: g[u]) if(go!=fa && go!=son[u]){
    dfs(go, u, 1);
  }
  if(son[u]) dfs(son[u], u, 0);

  for(auto [go, w]: g[u]) if(go!=fa && go!=son[u]){
    int l=lef[go], r=rig[go];
    rep(i, l, r){
      int cur=idfn[i];
      ask(u, cur);
    }
    rep(i, l, r){
      int cur=idfn[i];
      upd(d[cur], 1);
    }
  }
  ask(u, u);
  upd(d[u], 1);
  if(del){
    int l=lef[u], r=rig[u];
    rep(i, l, r){
      int cur=idfn[i];
      upd(d[cur], -1);
    }
  }
}

void solve(){
  cin>>n>>Q;
  rep(i, 1, n-1){
    int u, v, w; read(u), read(v), read(w);
    g[u].pb({v, w}), g[v].pb({u, w});
  }
  rep(i, 1, Q) read(qs[i]);
  dfs0(1, 0);
  dfs(1, 0, 0);
  rep(i, 1, Q) puts(res[i]? "AYE": "NAY");
}

signed main(){
  solve();
  return 0;
}
```

15 点分治

例题：

https://hydro.ac/p/bzoj-P3784

给你一棵树，$d(u,v)$ 表示 u 到 v 的边权和（距离）。

将所有的 $\frac{n(n-1)}{2}$ 个距离从大到小排序，让你输出前 m 个值。

```
// 大方向是枚举 lca
// 然后枚举距离的一个端点 v，另一个 u 在前面的若干子树中
// 用点分序 + 堆做这个过程

const int N=1e6+5;

int n, K;
vector<pii> g[N];

int cur_n;
int cent;
int sz[N];
bool vis[N];
int ts;
int stt, fin; // 对应 v 点前缀多个子树对应段（还包括 anc）
int d[N]; // d 是到 anc 的距离
int lef[N], rig[N];

void find_cent(int u, int fa){
  sz[u]=1;
  int ma=0;
  for(auto &[go, w]: g[u]) if(!vis[go] && go!=fa){ // 为了统一写法
    find_cent(go, u);
    sz[u]+=sz[go];
    ma=max(ma, sz[go]);
  }
  ma=max(ma, cur_n-sz[u]);
  if(ma<=cur_n/2) cent=u;
}

int cal_sz(int u, int fa){
  int sz=1;
  for(auto &[go, w]: g[u]) if(go!=fa && !vis[go]) sz+=cal_sz(go, u);
  return sz;
}

void dfs(int u, int fa, int dis){
  ++ts;
  d[ts]=dis;
  lef[ts]=stt, rig[ts]=fin;
  for(auto &[go, w]: g[u]) if(!vis[go] && go!=fa) dfs(go, u, dis+w);
}

void CD(int rt){
  vis[rt]=true;
  stt=(++ts);
  // 得到 lca 子树的序列
  for(auto &[go, w]: g[rt]) if(!vis[go]){
    fin=ts;
    dfs(go, rt, w);
  }
  for(auto &[go, w]: g[rt]) if(!vis[go]){
    cur_n=cal_sz(go, 0);
    find_cent(go, 0);
    CD(cent);
  }
}

struct SPT{
  int n;
  int f[N][20];

  int cmp(int x, int y){
    return d[x]>d[y]? x: y;
  }

  void build(int _n){
    n=_n;
    rep(i, 1, n) f[i][0]=i;
    rep(j, 1, 19){
      rep(i, 1, n-(1<<j)+1){
        f[i][j]=cmp(f[i][j-1], f[i+(1<<j-1)][j-1]);
      }
    }
  }

  int query(int l, int r){
    int k=__lg(r-l+1);
    return cmp(f[l][k], f[r-(1<<k)+1][k]);
  }
}spt;

// 映射后序列的点离 lca 的距离
int dis(int x, int y){
  return d[x]+d[y];
}

struct Info{
  int l, r, x, y;

  bool operator < (const Info &o)const{
    return dis(x, y)<dis(o.x, o.y);
  }
};

priority_queue<Info> q;

void insq(int l, int r, int y){
  if(l>r) return;
  int x=spt.query(l, r);
  q.push({l, r, x, y});
}

void solve(){
  cin>>n>>K;
  rep(i, 1, n-1){
    int u, v, w; read(u), read(v), read(w);
    g[u].pb({v, w}), g[v].pb({u, w});
  }

  {
    cur_n=n;
    find_cent(1, 0);
    CD(cent);
  }
  {
    spt.build(ts);
    rep(i, 1, ts){
      if(!lef[i]) continue;
      int l=lef[i], r=rig[i];
      // cerr<<l<<" "<<r<<"\n";
      insq(l, r, i);
    }
    // debug(ts);
  }
  {
    rep(_, 1, K){
      auto [l, r, x, y]=q.top(); q.pop();
      insq(l, x-1, y), insq(x+1, r, y);
      cout<<dis(x, y)<<"\n";
    }
  }
}

signed main(){
  solve();
  return 0;
}
```

16 虚树

个人推荐下面的做法，简洁（相较于单调栈的做法）易懂：

- 对关键点按照 DFS 序排序。
- 开一个 vector（记为 vt）维护一下虚树的点集：
  - 先将所有点丢进 vt。
  - 将 DFS 序相邻的点的 LCA 丢进 vt。
- 对 vt 按照 DFS 序排序、去重。
- 对于 vt 内相邻两点 $\mathrm{vt}_i, \mathrm{vt}_{i+1}$，对应地连接虚树树边 $(\mathrm{lca}(\mathrm{vt}_i, \mathrm{vt}_{i+1}), \mathrm{vt}_{i+1})$。

例题：

> 给你一棵树，然后进行 q 次询问。
>
> 每次询问是给出 m 个关键点，让你求至少删掉多少个点（关键点不能删除），才能让关键点间两两不可达。

做法很简单，建起虚树然后在虚树跑个 DP 就行。

```
int n, m;
int pt[N<<1];
bool key[N];
vector<int> g[N];

// 树剖求 lca 部分已略去。

void Add(int u, int v){
  g[u].pb(v), g[v].pb(u);
```

```
}

bool cmp(int a, int b){
  return dfn[a]<dfn[b];
}

// 建虚树核心
vector<int> vt;
void build_tree(){
  sort(pt+1, pt+1+m, cmp);
  vt.clear();
  rep(i, 1, m) vt.pb(pt[i]);
  rep(i, 1, m-1) vt.pb(lca(pt[i], pt[i+1]));
  sort(all(vt), cmp);
  vt.erase(unique(all(vt)), end(vt));
  for(auto u: vt) g[u].clear();
  rep(i, 0, (int)vt.size()-2) Add(lca(vt[i], vt[i+1]), vt[i+1]);
}

int res;
bool dp(int u, int fa){
  int cnt=0;
  for(auto go: g[u]) if(go!=fa){
    cnt+=dp(go, u);
  }
  if(key[u]){
    res+=cnt;
    return 1;
  }
  else{
    if(cnt<=1) return cnt;
    else{
      res++;
      return 0;
    }
  }
}

int main(){
  memset(h, -1, sizeof h);
  cin>>n;
  rep(i, 1, n-1){
    int u, v; read(u), read(v);
    add(u, v), add(v, u);
  }
  dfs1(1, -1, 1), dfs2(1, 1);

  int q; cin>>q;
  while(q--){
    read(m);
    rep(i, 1, m) read(pt[i]), key[pt[i]]=true;
    bool ed=false;
    rep(i, 1, m) if(key[fa[pt[i]]]){
      ed=true;
      puts("-1");
      rep(i, 1, m) key[pt[i]]=false;
      break;
    }
    if(ed) continue;
    build_tree();
    res=0;
    dp(vt[0], fa[vt[0]]);
    cout<<res<<endl;
    rep(i, 1, m) key[pt[i]]=false;
  }

  return 0;
}
```

17 树哈希

设计哈希函数为 $h(u)=B+f(h(\text{son}))$，然后 B 随机生成出来如果是偶数的话考虑 $+1$。

```
mt19937_64
rnd(chrono::steady_clock::now().time_since_epoch().count());
ull B=rnd();

ull hs(ull x){
  return x*x*x*19890535+19260817;
}

ull f(ull x){
  return hs(x&((1LL<<32)-1))+hs(x>>32);
}

ull h[N];

void dfs(int u, int fa){
  h[u]=B;
  for(auto go: g[u]) if(go!=fa) dfs(go, u), h[u]+=f(h[go]);
  res.insert(h[u]);
}
```

参考：https://peehs-moorhsum.blog.uoj.ac/blog/7891

模板题链接：https://uoj.ac/problem/763，可以尝试去测各种奇妙的 hash 做法。

18 图的连通性

约定这里的 `id[]` 数组代表对应点所在连通分量的编号。

18.1 强连通分量

```
int dfn[N], low[N], ts; // low[u] 为在 u 的子树中能够回溯到的最早*栈中节点*。
int stk[N], top;
bool in_stk[N];
int id[N]; // id[u] 代表对应 u 所在连通分量的编号
int scc_cnt;

void tarjan(int u){
  dfn[u]=low[u]=++ts;
  in_stk[u]=true;
  stk[++top]=u;

  for(int i=h[u]; ~i; i=e[i].next){
    int go=e[i].to;
    if(!dfn[go]){
      tarjan(go);
      low[u]=min(low[u], low[go]);
    }
    else if(in_stk[go]) low[u]=min(low[u], dfn[go]);
  }

  if(dfn[u]==low[u]){
    scc_cnt++;
    int y;
    do{
      y=stk[top--];
      in_stk[y]=false;
      id[y]=scc_cnt;
    }while(y!=u);
  }
}
```

18.2 边双连通分量

```
int dfn[N], low[N], ts;
int stk[N], top;
int id[N], bcc_cnt; // id[u] 表示 u 所在的边双连通分量编号
bool is_bridge[M]; // 是否为桥（删掉该边后图连通分量数增加了，则称这条边为桥）
void tarjan(int u, int from){ // from 记录从哪条边过来的
  dfn[u]=low[u]=++ts;
  stk[++top]=u;
  for(int i=h[u]; ~i; i=e[i].next){
    int go=e[i].to;
    if(!dfn[go]){
      tarjan(go, i);
      low[u]=min(low[u], low[go]);
      if(dfn[u]<low[go]) is_bridge[i]=is_bridge[i^1]=true;
    }
    else if(i!=(from^1)) // 非反向边
      low[u]=min(low[u], dfn[go]);
  }

  if(dfn[u]==low[u]){
    ++bcc_cnt;
    int y;
    do{
      y=stk[top--];
      id[y]=bcc_cnt;
    }while(y!=u);
  }
}
```

18.3 点双连通分量

注意，如果图中存在自环，建图的时候直接忽略（即不连自环）即可。

```
int dfn[N], low[N], ts;
int stk[N], top;
int bcc_cnt; // 点双连通分量的个数
vector<int> bcc[N]; // 点双连通分量的成员
bool cut[N]; // 是否为割点
int root;

void tarjan(int u){
    dfn[u]=low[u]=++ts;
    stk[++top]=u;
    if(u==root && h[u]==-1){
        bcc[++bcc_cnt].push_back(u);
        return;
    }

    int cnt=0;
    for(int i=h[u]; ~i; i=e[i].next){
        int go=e[i].to;
        if(!dfn[go]){
            tarjan(go);
            low[u]=min(low[u], low[go]);

            if(dfn[u]<=low[go]){
                cnt++;
                if(u!=root || cnt>1) cut[u]=true;

                ++bcc_cnt;
                int y;
                do{
                    y=stk[top--];
                    bcc[bcc_cnt].push_back(y);
                }while(y!=go);
                bcc[bcc_cnt].push_back(u);
            }
        }
        else low[u]=min(low[u], dfn[go]);
    }
}
```

调用时，直接对全局变量 root 进行枚举即可：

```
for(root=1; root<=n; root++) if(!dfn[root]) tarjan(root);
```

18.4 2-SAT

例题：

https://www.luogu.com.cn/problem/P4782

有 n 个布尔变量 $x_1 \sim x_n$，另有 m 个需要满足的条件，每个条件的形式都是「x_i 为 true / false 或 x_j 为 true / false」。比如「x_1 为真或 x_3 为假」、「x_7 为假或 x_2 为假」。

2-SAT 问题的目标是给每个变量赋值使得所有条件得到满足。

输入格式

第一行两个整数 n 和 m，意义如题面所述。

接下来 m 行每行 4 个整数 i, a, j, b，表示「x_i 为 a 或 x_j 为 b」$(a, b \in \{0, 1\})$

```
const int N=2e6+5, M=2e6+5;
int n, m;

struct node{
  int to, next;
}e[M];

int h[N], tot;

void add(int u, int v){
  e[tot].to=v, e[tot].next=h[u], h[u]=tot++;
}

int ts, dfn[N], low[N];
int stk[N], top;
int id[N], cnt;
bool ins[N];

void tarjan(int u){
  dfn[u]=low[u]=++ts;
  stk[++top]=u, ins[u]=true;
  for(int i=h[u]; ~i; i=e[i].next){
    int go=e[i].to;
    if(!dfn[go]){
      tarjan(go);
      low[u]=min(low[u], low[go]);
    }else if(ins[go]) low[u]=min(low[u], dfn[go]);
  }

  if(dfn[u]==low[u]){
    int y;
    cnt++;
    do{
      y=stk[top--], ins[y]=false, id[y]=cnt;
    }while(y!=u);
  }
}

int res[N];

int main(){
  memset(h, -1, sizeof h);
  read(n), read(m);

  while(m--){
    int i, a, j, b; read(i), read(a), read(j), read(b);
    i--, j--;
    add(2*i+!a, 2*j+b), add(2*j+!b, 2*i+a);
  }

  for(int i=0; i<2*n; i++) if(!dfn[i]) tarjan(i);

  for(int i=0; i<n; i++) if(id[2*i]==id[2*i+1]){
    puts("IMPOSSIBLE");
    return 0;
  }else res[i]= id[2*i]>id[2*i+1];

  puts("POSSIBLE");
  for(int i=0; i<n; i++) printf("%d ", res[i]);
  cout<<endl;

  return 0;
}
```

19 二分图匹配

> 最小点覆盖问题：选取点集，找一种选法使得每条边至少有一个点选取，然后最小化选取点数。对于二分图，最小点覆盖等于最大匹配。
>
> 最大独立集问题：选取点集，使得点之间没有边相连，最大化点数。这显然是最小点覆盖问题的补，故答案为点数减去最小点覆盖的答案。
>
> 二分图匹配可以使用最大流求解，复杂度 $O(\sqrt{n}m)$。

下面是直接用匈牙利算法的实现，复杂度 $O(nm)$。

```
bool vis[N];
int match[N];
bool find(int u){
    for(int i=h[u]; ~i; i=e[i].next){
        int go=e[i].to;
        if(!vis[go]){
            vis[go]=true;
            if(!match[go] || find(match[go])){
                match[go]=u;
                return true;
            }
        }
    }
    return false;
}

signed main(){
  memset(h, -1, sizeof h);
  cin>>n1>>n2>>m; // n1, n2 为左、右部点个数
  rep(i, 1, m){
    int u, v; read(u), read(v);
    add(u, v+n1);
  }

  int res=0;
  rep(i, 1, n1){
    fill(vis+1, vis+1+n1+n2, false);
    if(find(i)) res++;
  }
  cout<<res<<endl;

  return 0;
}
```

20 网络流

约定下面的 S, T 为源汇点。

20.1 最大流

Dinic 算法实现：

```
struct Edge{
  int to, c, next;
}e[M];
int h[N], tot;

void add(int u, int v, int c){
  e[tot].to=v, e[tot].c=c, e[tot].next=h[u], h[u]=tot++;
  e[tot].to=u, e[tot].c=0, e[tot].next=h[v], h[v]=tot++;
}

int n, m, S, T;

int d[N], q[N], cur[N];

bool bfs(){
  memset(d, -1, sizeof d);
  int tt=-1, hh=0;
  q[++tt]=S, d[S]=0, cur[S]=h[S];

  while(tt>=hh){
    int u=q[hh++];
    for(int i=h[u]; ~i; i=e[i].next){
      int go=e[i].to;
      if(d[go]==-1 && e[i].c){
        d[go]=d[u]+1;
        cur[go]=h[go];
        if(go==T) return true;
        q[++tt]=go;
      }
    }
  }
  return false;
}

int find(int u, int limit){
  if(u==T) return limit;
  int flow=0;
  for(int i=cur[u]; ~i && flow<limit; i=e[i].next){
    cur[u]=i;
    int go=e[i].to;
    if(d[go]==d[u]+1 && e[i].c){
      int t=find(go, min(e[i].c, limit-flow));
      if(!t) d[go]=-1;
      e[i].c-=t, e[i^1].c+=t, flow+=t;
    }
  }
  return flow;
}

int dinic(){
  int res=0, flow;
  while(bfs()) while(flow=find(S, INF)) res+=flow;
  return res;
}
```

20.2 费用流

下为最小费用最大流实现：此实现中，最坏时间复杂度为 $O(nmf)$，其中 f 为网络最大流。

```
int n, m, S, T;
struct Edge{
  int to, c, w, next;
}e[M];

int h[N], tot;

void add(int u, int v, int c, int w){
  e[tot].to=v, e[tot].c=c, e[tot].w=w, e[tot].next=h[u], h[u]=tot++;
  e[tot].to=u, e[tot].c=0, e[tot].w=-w, e[tot].next=h[v], h[v]=tot++;
}

int d[N], q[N], lim[N], pre[N];
bool vis[N];

bool spfa(){
  memset(d, 0x3f, sizeof d); memset(lim, 0, sizeof lim);
  int tt=0, hh=0;
  q[tt++]=S, d[S]=0, lim[S]=INF, vis[S]=true;

  while(tt!=hh){
    int hd=q[hh++]; if(hh==N) hh=0;
    vis[hd]=false;

    for(int i=h[hd]; ~i; i=e[i].next){
      int go=e[i].to;
      if(d[go]>d[hd]+e[i].w && e[i].c){
        d[go]=d[hd]+e[i].w;
        pre[go]=i;
        lim[go]=min(lim[hd], e[i].c);
        if(!vis[go]){
          vis[go]=true;
          q[tt++]=go; if(tt==N) tt=0;
        }
      }
    }
  }

  return lim[T]>0;
}

void EK(int &flow, int &cost){
  flow=cost=0;
  while(spfa()){
    int t=lim[T];
    flow+=t, cost+=t*d[T];
    for(int i=T; i!=S; i=e[pre[i]^1].to){
      e[pre[i]].c-=t, e[pre[i]^1].c+=t;
    }
  }
}
```

20.3 上下界网络流

20.3.1 无源汇上下界可行流

添加虚拟源汇点 S',T' 并添加附加边解决，见建图代码。

```
int n, m, S, T;
struct Edge{
    int to, c, l, next;
}e[M];
int h[N], tot;

// 约定 lc 为容量下界、uc 为容量上界
void add(int u, int v, int lc, int uc){
    e[tot].to=v, e[tot].c=uc-lc, e[tot].l=lc, e[tot].next=h[u],
h[u]=tot++;
    e[tot].to=u, e[tot].c=0, e[tot].next=h[v], h[v]=tot++;
}

int imo[N]; // 入这的容量下界和 - 出这的容量下界和

int d[N], q[N], cur[N];

bool bfs(){
    memset(d, -1, sizeof d);
    int tt=-1, hh=0;
    q[++tt]=S, d[S]=0, cur[S]=h[S];
    while(tt>=hh){
        int u=q[hh++];
        for(int i=h[u]; ~i; i=e[i].next){
            int go=e[i].to;
            if(d[go]==-1 && e[i].c){
                cur[go]=h[go];
                d[go]=d[u]+1;
                if(go==T) return true;
                q[++tt]=go;
            }
        }
    }
    return false;
}

int find(int u, int limit){
    if(u==T) return limit;
    int flow=0;
    for(int i=cur[u]; ~i && limit>flow; i=e[i].next){
        int go=e[i].to;
        cur[u]=i;
        if(e[i].c && d[go]==d[u]+1){
            int t=find(go, min(limit-flow, e[i].c));
            if(!t) d[go]=-1;
            flow+=t, e[i].c-=t, e[i^1].c+=t;
        }
    }
    return flow;
}

int dinic(){
    int res=0, flow;
    while(bfs()) while(flow=find(S, INF)) res+=flow;
    return res;
}

int main(){
    memset(h, -1, sizeof h);
    cin>>n>>m;
    S=0, T=n+1;
    for(int i=1; i<=m; i++){
        int u, v, lc, uc; cin>>u>>v>>lc>>uc;
        add(u, v, lc, uc);
        imo[u]-=lc, imo[v]+=lc;
    }

    int tot=0;
    for(int i=1; i<=n; i++)
        if(imo[i]>0) add(S, i, 0, imo[i]), tot+=imo[i];
        else if(imo[i]<0) add(i, T, 0, -imo[i]);

    if(dinic()!=tot) puts("NO");
    else{
        puts("YES");
        for(int i=0; i<2*m; i+=2) cout<<e[i].l+e[i^1].c<<endl;
    }
    return 0;
}
```

20.4 有源汇上下界可行流

原图汇点 T 向源点 S 加一条下界 0 上界 ∞ 的边即可转为无源汇上下界可行流。

20.5 有源汇上下界最大流

删去所有附加边之后的残量网络。在残量网络上再跑一次 S 到 T 的最大流，将这个结果和可行流加起来即可。

```
int cnt=0;
for(int i=0; i<=n+m+1; i++)
  if(imo[i]>0) add(S, i, imo[i]), cnt+=imo[i];
  else if(imo[i]<0) add(i, T, -imo[i]);

add(t, s, INF);

if(dinic()!=cnt) cout<<-1<<endl<<endl;
else{
  int res=e[tot-1].c;
  e[tot-1].c=0, e[tot-2].c=0; // 删最后 T 向 S 连的那条边
  S=s, T=t; // 换为原图源汇点
  res+=dinic();
  cout<<res<<endl<<endl;
}
```

21 BIT

```
struct BIT{
  int n;
  int last;
  vector<int> tr;

  BIT(int _n=0): n(_n), last(0), tr(n+1, 0) {}

  void init(int _n){
    *this=BIT(_n);
  }

  int lowbit(int x){
    return x&-x;
  }

  void add(int x, int k=1){
    last+=k;
    for(; x<=n; x+=lowbit(x)) tr[x]+=k;
  }

  int query(int x){
    if(x<=0 || x>n) return 0;
    int res=0;
    for(; x; x-=lowbit(x)) res+=tr[x];
    return res;
  }

  int kth(int k){
    int sum=0, x=0;
    for(int i=__lg(n); ~i; --i){
      x+=1<<i;
      if(x>=n || sum+tr[x]>=k) x-=1<<i;
      else sum+=tr[x];
    }
    return x+1;
  }

  int pred(int x){
    int cnt=query(x-1);
    if(!cnt) return 0;
    return kth(cnt);
  }

  int succ(int x){
    int cnt=query(x);
    if(cnt==last) return 0;
    return kth(cnt+1);
  }
}bit;
```

22 FHQ Treap

个人认为最好用的平衡树，好写、代码短而且够用。

核心操作为 merge（合并） 及 split（分裂）。

根据分裂方式可以分为以下两种。

22.1 按值分裂

```
struct Node{
    int l, r;
    int key;
    int sz, v;

    #define ls tr[u].l
    #define rs tr[u].r
}tr[N];

int root, idx;

void pushup(int u){
    tr[u].sz=tr[ls].sz+tr[rs].sz+1;
}

int add(int v){
    ++idx;
    tr[idx].key=rand(), tr[idx].sz=1, tr[idx].v=v;
    return idx;
}

int merge(int x, int y){
    if(!x || !y) return x+y;
    if(tr[x].key>tr[y].key){
        tr[x].r=merge(tr[x].r, y);
        pushup(x);
        return x;
    }
    else{
        tr[y].l=merge(x, tr[y].l);
        pushup(y);
        return y;
    }
}

void split(int u, int val, int &x, int &y){
    if(!u) return x=y=0, void();
    if(tr[u].v<=val)
        x=u, split(rs, val, rs, y);
    else
        y=u, split(ls, val, x, ls);
    pushup(u);
}

// 插入 v
void insert(int v){
    int x, y;
    split(root, v, x, y);
    root=merge(x, merge(add(v), y));
}

// 删去 v
void remove(int v){
    int x, y, z;
    split(root, v, x, z);
    split(x, v-1, x, y);
    y=merge(tr[y].l, tr[y].r);
    root=merge(merge(x, y), z);
}

// 求值的排名
int val4rk(int v){
    int x, y;
    split(root, v-1, x, y);
    int res=tr[x].sz+1;
    root=merge(x, y);
    return res;
}

// 求第 k 小的数
int rk4val(int k){
    int u=root;
    while(u){
        if(tr[ls].sz+1==k) return tr[u].v;
        else if(tr[ls].sz>=k) u=ls;
        else k-=tr[ls].sz+1, u=rs;
    }
    return -1;
}
```

```
// 求 v 的前驱
int get_prev(int v){
    int x, y;
    split(root, v-1, x, y);
    int u=x;
    while(rs) u=rs;
    int res=tr[u].v;
    root=merge(x, y);
    return res;
}

// 求 v 的后继
int get_next(int v){
    int x, y;
    split(root, v, x, y);
    int u=y;
    while(ls) u=ls;
    int res=tr[u].v;
    root=merge(x, y);
    return res;
}
```

22.2 按排名分裂

用以区间操作，例如区间翻转。

```
struct Node{
  int l, r;
  int v;
  int key, sz;
  int rev; // 翻转标记

  #define ls tr[u].l
  #define rs tr[u].r
}tr[N];

int idx, root;

int add(int v){
  ++idx;
  tr[idx].v=v, tr[idx].key=rand(), tr[idx].sz=1;
  return idx;
}

void pushup(int u){
  tr[u].sz=tr[ls].sz+tr[rs].sz+1;
}

void pushdown(int u){
  if(tr[u].rev){
    swap(ls, rs);
    tr[ls].rev^=1, tr[rs].rev^=1;
    tr[u].rev=0;
  }
}

int merge(int x, int y){
  if(!x || !y) return x+y;
  if(tr[x].key>tr[y].key){
    pushdown(x);
    tr[x].r=merge(tr[x].r, y);
    pushup(x);
    return x;
  }
  else{
    pushdown(y);
    tr[y].l=merge(x, tr[y].l);
    pushup(y);
    return y;
  }
}

void split(int u, int k, int &x, int &y){
  if(!u) return x=y=0, void();
  pushdown(u);
  if(tr[ls].sz+1<=k)
    x=u, split(rs, k-tr[ls].sz-1, rs, y);
  else
    y=u, split(ls, k, x, ls);
  pushup(u);
}

void reverse(int l, int r){
  int x, y, z;
  split(root, l-1, x, y), split(y, r-l+1, y, z);
  tr[y].rev^=1;
  root=merge(x, merge(y, z));
}

void output(int u){
  if(!u) return;
  pushdown(u); // 查询记得推标记
  output(ls); cout<<tr[u].v<<' '; output(rs);
}
```

同时提供一个稍加封装而且是动态申请节点的版本，方便操作：

```
struct Node{
    int l, r;
    int key;
    int sz, v;
```

```
    #define ls tr[u].l
    #define rs tr[u].r
}tr[N];

int root, idx;

void pushup(int u){
    tr[u].sz=tr[ls].sz+tr[rs].sz+1;
}

int add(int v){
    ++idx;
    tr[idx].key=rand(), tr[idx].sz=1, tr[idx].v=v;
    return idx;
}

int merge(int x, int y){
    if(!x || !y) return x+y;
    if(tr[x].key>tr[y].key){
        tr[x].r=merge(tr[x].r, y);
        pushup(x);
        return x;
    }
    else{
        tr[y].l=merge(x, tr[y].l);
        pushup(y);
        return y;
    }
}

void split(int u, int val, int &x, int &y){
    if(!u) return x=y=0, void();
    if(tr[u].v<=val)
        x=u, split(rs, val, rs, y);
    else
        y=u, split(ls, val, x, ls);
    pushup(u);
}

void insert(int v){
    int x, y;
    split(root, v, x, y);
    root=merge(x, merge(add(v), y));
}

void remove(int v){
    int x, y, z;
    split(root, v, x, z);
    split(x, v-1, x, y);
    y=merge(tr[y].l, tr[y].r);
    root=merge(merge(x, y), z);
}

int val4rk(int v){
    int x, y;
    split(root, v-1, x, y);
    int res=tr[x].sz+1;
    root=merge(x, y);
    return res;
}

int rk4val(int k){
    int u=root;
    while(u){
        if(tr[ls].sz+1==k) return tr[u].v;
        else if(tr[ls].sz>=k) u=ls;
        else k-=tr[ls].sz+1, u=rs;
    }
    return -1;
}

int get_prev(int v){
    int x, y;
    split(root, v-1, x, y);
    int u=x;
    while(rs) u=rs;
    int res=tr[u].v;
    root=merge(x, y);
    return res;
}

int get_next(int v){
    int x, y;
    split(root, v, x, y);
    int u=y;
    while(ls) u=ls;
    int res=tr[u].v;
    root=merge(x, y);
    return res;
}
```

23 树状数组

23.1 一维树状数组

```
int tr[N];
int lowbit(int x){
    return x&-x;
}
void add(int p){
    for(; p<N; p+=lowbit(p)) tr[p]++;
}
int query(int p){
    int res=0;
    for(; p; p-=lowbit(p)) res+=tr[p];
    return res;
}
```

23.2 二维树状数组

```
// 单点 (x, y) 修改 区间 (sx, sy, tx, ty) 查询
add(x, y, k);
int res=query(tx, ty)-query(sx-1, ty)-query(tx, sy-1)+query(sx-1,
sy-1);

// 单点 (x, y) 查询 区间 (sx, sy, tx, ty) 修改
int res=query(x, y);
add(sx, sy, k), add(sx, ty+1, -k), add(tx+1, sy, -k), add(tx+1,
ty+1, k);
```

例题：www.luogu.com.cn/problem/P4514

输入数据的第一行为 X n m，代表矩阵大小为 $n \times m$。从输入数据的第二行开始到文件尾的每一行会出现以下两种操作：

- L a b c d delta —— 代表将 $(a,b),(c,d)$ 为顶点的矩形区域内的所有数字加上 δ。
- k a b c d —— 代表求 $(a,b),(c,d)$ 为顶点的矩形区域内所有数字的和。

针对每个 k 操作，在单独的一行输出答案。

```
struct BIT{
  static int n, m;
  int tr[N][N];

  int lowbit(int x){
    return x&-x;
  }

  void add(int x, int y, int k){
    for(int i=x; i<=n; i+=lowbit(i))
      for(int j=y; j<=m; j+=lowbit(j))
        tr[i][j]+=k;
  }

  int query(int x, int y){
    int res=0;
    for(int i=x; i; i-=lowbit(i))
      for(int j=y; j; j-=lowbit(j))
        res+=tr[i][j];
    return res;
  }
};

int BIT::n=0, BIT::m=0;
BIT A, B, C, D;

void add(int x, int y, int k){
  A.add(x, y, k);
  B.add(x, y, k*x);
  C.add(x, y, k*y);
  D.add(x, y, k*x*y);
}

int query(int x, int y){
  return A.query(x, y)*(x*y+x+y+1)-B.query(x, y)*(y+1)-C.query(x,
y)*(x+1)+D.query(x, y);
}

void Add(int sx, int sy, int tx, int ty, int k){
  add(sx, sy, k), add(sx, ty+1, -k), add(tx+1, sy, -k), add(tx+1,
ty+1, k);
}

int Query(int sx, int sy, int tx, int ty){
  return query(tx, ty)-query(sx-1, ty)-query(tx, sy-1)+query(sx-1,
sy-1);
}

signed main(){
  char op; cin>>op;
  cin>>BIT::n>>BIT::m;
  while(cin>>op){
    int sx, sy, tx, ty; cin>>sx>>sy>>tx>>ty;
    if(op=='L'){
      int k; cin>>k;
      Add(sx, sy, tx, ty, k);
    }
    else cout<<Query(sx, sy, tx, ty)<<endl;
  }

  return 0;
}
```

24 线段树

24.1 势能线段树

X 说，要有数列，于是便给定了一个正整数数列。L 说，要能修改，于是便有了对一段数中每个数都开平方(下取整)的操作。k 说，要能查询，于是便有了求一段数的和的操作。

- op $= 0$ 表示给 $[l,r]$ 中的每个数开平方（下取整）。
- op $= 1$ 表示询问 $[l,r]$ 中各个数的和。

```
int n, q, w[N];

struct Node{
  int l, r;
  int mx, sum;

  #define ls(u) u<<1
  #define rs(u) u<<1|1
}tr[N<<2];

void pushup(int u){
  tr[u].mx=max(tr[ls(u)].mx, tr[rs(u)].mx);
  tr[u].sum=tr[ls(u)].sum+tr[rs(u)].sum;
}

void build(int u, int l, int r){
  tr[u]={l, r, w[l], w[r]};
  if(l==r) return;
  int mid=l+r>>1;
  build(ls(u), l, mid), build(rs(u), mid+1, r);
  pushup(u);
}

void upd(int u, int l, int r){
  if(tr[u].l==tr[u].r) return tr[u].mx=tr[u].sum=sqrt(tr[u].sum),
void();
  int mid=tr[u].l+tr[u].r>>1;
  if(l<=mid && tr[ls(u)].mx>1) upd(ls(u), l, r);
  if(mid<r && tr[rs(u)].mx>1) upd(rs(u), l, r);
  pushup(u);
}

int query(int u, int l, int r){
  if(l<=tr[u].l && tr[u].r<=r) return tr[u].sum;
  int mid=tr[u].l+tr[u].r>>1, res=0;
  if(l<=mid) res+=query(ls(u), l, r);
  if(mid<r) res+=query(rs(u), l, r);
  return res;
}

signed main(){
  read(n);
  rep(i, 1, n) read(w[i]);
  read(q);

  build(1, 1, n);

  while(q--){
    int op, l, r; read(op), read(l), read(r);
    if(l>r) swap(l, r);
    if(!op) upd(l, l, r);
    else cout<<query(1, l, r)<<endl;
  }
    return 0;
}
```

24.2 线段树分治

例题：

https://www.luogu.com.cn/problem/P5787

神犇有一个 n 个节点的图。

因为神犇是神犇，所以在 k 时间内有 m 条边会出现后消失。

神犇要求出每一时间段内这个图是否是二分图。

```
const int N=2e5+50;

int n, m, K;

struct Msg{
  int x, y, z;
}stk[N];
int top;

struct Dsu{
  int fa[N], rk[N];

  void init(){
    rep(i,1,n<<1) fa[i]=i, rk[i]=1;
  }

  int find(int x){
    return x==fa[x]? x: find(fa[x]);
  }

  bool same(int x, int y){
    return find(x)==find(y);
  }

  void merge(int x, int y){
    x=find(x), y=find(y);
    if(x==y) return;
    if(rk[x]>rk[y]) swap(x, y);
    fa[x]=y;
    stk[++top]={x, y, rk[x]==rk[y]};
    if(rk[x]==rk[y]) rk[y]++;
  }

  void resume(Msg t){
    rk[t.y]-=t.z;
    fa[t.x]=t.x;
  }
}dsu;

struct Node{
  int l, r;
  vector<pii> o;

  #define ls u<<1
  #define rs u<<1|1
}tr[N<<2];

void build(int u, int l, int r){
  tr[u]={l, r};
  if(l==r) return;
  int mid=l+r>>1;
  build(ls, l, mid), build(rs, mid+1, r);
}

void assign(int u, int l, int r, int x, int y){
  if(l<=tr[u].l && tr[u].r<=r){
    tr[u].o.pb({x, y});
    return;
  }
  int mid=tr[u].l+tr[u].r>>1;
  if(l<=mid) assign(ls, l, r, x, y);
  if(mid<r) assign(rs, l, r, x, y);
}

void divi(int u){
  bool ng=false;
  int pre=top;
  for(auto &[x, y]: tr[u].o){
    dsu.merge(x, y+n);
    dsu.merge(y, x+n);
    if(dsu.same(x, x+n) || dsu.same(y, y+n)){
      ng=true;
      break;
    }
  }
  if(ng) rep(i,tr[u].l,tr[u].r) puts("No");
  else{
    if(tr[u].l==tr[u].r) puts("Yes");
    else{
      divi(ls);
      divi(rs);
    }
  }
  while(top!=pre){
    dsu.resume(stk[top--]);
  }
}

signed main(){
  cin>>n>>m>>K;
  build(1, 1, K);
  dsu.init();

  rep(i,1,m){
    int x, y, l, r; read(x), read(y), read(l), read(r);
    if(++l>r) continue;
    assign(1, l, r, x, y);
  }
  divi(1);

  return 0;
}
```

24.3 扫描线

例题：

https://www.luogu.com.cn/problem/P1502

小卡使用了超能力（透视术）知道了墙后面每个星星的位置和亮度，但是小卡发动超能力后就很疲劳，只好拜托你告诉他最多能够有总和多亮的星星能出现在窗口上。

输入格式

本题有多组数据，第一行为 T，表示有 T 组数据。

对于每组数据：

第一行 3 个整数 n, W, H 表示有 n 颗星星，窗口宽为 W，高为 H。

接下来 n 行，每行三个整数 x_i, y_i, l_i 表示星星的坐标在 (x_i, y_i)，亮度为 l_i。

输出格式

T 个整数，表示每组数据中窗口星星亮度总和的最大值。

```
const int N=1e4+50, INF=1e10;

int n, w, h;

struct Node{
  int x, l, r, v, k;
```

```
  bool operator < (const Node &o)const{
    return x==o.x? k>o.k: x<o.x;
  }
}q[N<<1];
int tot;

#define ls tr[u].l
#define rs tr[u].r

struct Tree{
  int l, r;
  int v, add;
}tr[14*N<<2];

int idx, root;

void init(){
  tot=0;
  idx=root=0;
  memset(tr, 0, sizeof tr);
}

void pushup(int u){
  tr[u].v=max(tr[ls].v, tr[rs].v);
}

void pushdown(int u){
  if(tr[u].add){
    if(!ls) ls=++idx;
    if(!rs) rs=++idx;
    auto &L=tr[ls], &R=tr[rs];
    L.v+=tr[u].add, L.add+=tr[u].add;
    R.v+=tr[u].add, R.add+=tr[u].add;
    tr[u].add=0;
  }
}

void upd(int &u, int l, int r, int nl, int nr, int v){
  if(!u) u=++idx;
  if(nl<=l && r<=nr){
    tr[u].v+=v, tr[u].add+=v;
    return;
  }
  pushdown(u);
  int mid=l+r>>1;
  if(nl<=mid) upd(ls, l, mid, nl, nr, v);
  if(mid<nr) upd(rs, mid+1, r, nl, nr, v);
  pushup(u);
}

signed main(){
  int T; cin>>T;
  while(T--){
    init();
    read(n), read(w), read(h); w--, h--;

    rep(i,1,n){
      int x, y, v; read(x), read(y), read(v);
      q[++tot]={x, y, y+h, v, 1};
      q[++tot]={x+w, y, y+h, v, -1};
    }

    sort(q+1, q+1+tot);

    int res=0;
    rep(i,1,tot){
      upd(root, 0, INF, q[i].l, q[i].r, q[i].v*q[i].k);
      res=max(res, tr[root].v);
    }
    cout<<res<<endl;
  }
    return 0;
}
```

25 可持久化数据结构

25.1 可持久化线段树

例题：luogu.com.cn/problem/P3834
给定 n 个整数构成的序列 a，将对于指定的闭区间 $[l,r]$ 查询其区间内的第 k 小值。

```
/*
习惯约定：
u 代表结点（编号）
p 代表先前版本的位置指针
q 代表最新版本的位置指针
*/

const int N=2e5+5, M=2e5+5;
int n,m;
int a[N];
vector<int> nums; // 离散化
int root[N];

int find(int x){
  return lower_bound(nums.begin(), nums.end(), x)-nums.begin();
}

struct node{
  int l, r;
  int cnt;
}tr[4*N+17*N]; // 初始开的点数+logN * N （各版本总规模）

int idx;
// 返回建立的点的编号，两个参数分别代表左右边界。
int build(int l, int r){
  int u=++idx;
  if(l==r) return u;
  int mid=l+r>>1;
  tr[u].l=build(l, mid), tr[u].r=build(mid+1, r);
  return u;
}

// 递归地插入
int insert(int p,int l,int r,int x){
  int q=++idx;
  tr[q]=tr[p];
  if(l==r){
    tr[q].cnt++;
    return q;
  }
  int mid=l+r>>1;
  if(x<=mid) tr[q].l=insert(tr[p].l, l, mid, x); // 如果更新的位置是在左边，那么 tr[q].l 为新开点
  else tr[q].r=insert(tr[p].r, mid+1, r, x); // 否则 tr[q].r 为新开点
  tr[q].cnt=tr[tr[q].l].cnt+tr[tr[q].r].cnt; // pushup the cnt
  return q;
}

int query(int p, int q, int l, int r, int k){
  if(l==r) return r;
  int mid=l+r>>1;
  int cnt=tr[tr[q].l].cnt-tr[tr[p].l].cnt;
  if(cnt>=k) return query(tr[p].l, tr[q].l, l, mid, k);
  else return query(tr[p].r, tr[q].r, mid+1, r, k-cnt);
}

int main(){
  cin>>n>>m;

  for(int i=1; i<=n; i++)
    cin>>a[i], nums.push_back(a[i]);

  sort(nums.begin(), nums.end());
  nums.erase(unique(nums.begin(), nums.end()), nums.end());

  root[0]=build(0, nums.size()-1); // 第 0 个版本指的就是空的线段树。

  for(int i=1; i<=n; i++)
    root[i]=insert(root[i-1], 0, nums.size()-1, find(a[i]));

  while(m--){
    int l, r, k; cin>>l>>r>>k;
    cout<<nums[query(root[l-1], root[r], 0, nums.size()-1,
k)]<<endl;
  }

  return 0;
}
```

25.2 可持久化平衡树

例题：https://www.luogu.com.cn/problem/P3835
您需要写一种数据结构（可参考题目标题），来维护一个可重整数集合，其中需要提供以下操作（对于各个以往的历史版本）：

1、 插入 x
2、 删除 x（若有多个相同的数，应只删除一个，如果没有请忽略该操作）
3、 查询 x 的排名（排名定义为比当前数小的数的个数 $+1$）
4、 查询排名为 x 的数
5、 求 x 的前驱（前驱定义为小于 x，且最大的数，如不存在输出 $-2^{31}+1$）
6、 求 x 的后继（后继定义为大于 x，且最小的数，如不存在输出 $2^{31}-1$）

和原本平衡树不同的一点是，每一次的任何操作都是基于某一个历史版本，同时生成一个新的版本。（操作 **3, 4, 5, 6** 即保持原版本无变化）

每个版本的编号即为操作的序号（版本 0 即为初始状态，空树）

```
const int N=5e5+5, INF=2147483647;

struct Node{
  int l, r;
  int key;
  int sz, v;

  #define ls tr[u].l
  #define rs tr[u].r
}tr[N*60];

int root[N], idx=0;

void pushup(int u){
  tr[u].sz=tr[ls].sz+tr[rs].sz+1;
}

int add(int v){
  ++idx;
  tr[idx].key=rand(), tr[idx].sz=1, tr[idx].v=v;
  return idx;
}

int copy(int u){
  ++idx;
  tr[idx]=tr[u];
  return idx;
}

void split(int u, int val, int &x, int &y){
  if(!u) return x=y=0, void();
  if(tr[u].v<=val)
    x=copy(u), split(tr[x].r, val, tr[x].r, y), pushup(x);
  else
    y=copy(u), split(tr[y].l, val, x, tr[y].l), pushup(y);
}

int merge(int x, int y){
  if(!x || !y) return x+y;
  if(tr[x].key>tr[y].key){
    int t=copy(x);
    tr[t].r=merge(tr[t].r, y);
    pushup(t);
    return t;
  }
  else{
    int t=copy(y);
    tr[t].l=merge(x, tr[t].l);
    pushup(t);
    return t;
  }
}

void insert(int &rt, int v){
  int x, y;
  split(rt, v, x, y);
  rt=merge(merge(x, add(v)), y);
}

void remove(int &rt, int v){
  int x, y, z;
  split(rt, v, x, z);
  split(x, v-1, x, y);
  y=merge(tr[y].l, tr[y].r);
  rt=merge(merge(x, y), z);
}

int val4rk(int &rt, int v){
  int x, y;
  split(rt, v-1, x, y);
  int res=tr[x].sz+1;
  rt=merge(x, y);
  return res;
}

int rk4val(int &rt, int k){
  int u=rt;
  while(u){
    if(tr[tr[ls].sz+1==k) return tr[u].v;
    else if(tr[tr[ls].sz>=k) u=ls;
    else k-=tr[tr[ls].sz+1, u=rs;
  }
  return -1;
}

int get_prev(int &rt, int v){
  int x, y;
  split(rt, v-1, x, y);
  int u=x;
  while(rs) u=rs;
  int res=(u? tr[u].v: -INF);
  rt=merge(x, y);
  return res;
}

int get_next(int &rt, int v){
  int x, y;
  split(rt, v, x, y);
  int u=y;
  while(ls) u=ls;
  int res=(u? tr[u].v: INF);
  rt=merge(x, y);
  return res;
}

int main(){
  srand(19260817);
  ios::sync_with_stdio(false);
  int q; cin>>q;

  for(int i=1; i<=q; i++){
    int id, op, x; cin>>id>>op>>x;
    root[i]=root[id];
    if(op==1) insert(root[i], x);
    else if(op==2) remove(root[i], x);
    else if(op==3) cout<<val4rk(root[i], x)<<endl;
    else if(op==4) cout<<rk4val(root[i], x)<<endl;
    else if(op==5) cout<<get_prev(root[i], x)<<endl;
    else if(op==6) cout<<get_next(root[i], x)<<endl;
  }
  return 0;
}
```

25.3 可持久化并查集

例题： www.luogu.com.cn/problem/P3402
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。

```
struct Msg{
  int fa, rk;
};

struct Node{
  int l, r;
  Msg val;

  #define ls(u) tr[u].l
  #define rs(u) tr[u].r
}tr[N*25];

int root[N], idx;

int n, Q;

void build(int &u, int l, int r){
  u=++idx;
  if(l==r) return tr[u].val={l, 0}, void();
  int mid=l+r>>1;
  build(ls(u), l, mid), build(rs(u), mid+1, r);
}

Msg qval(int u, int l, int r, int x){
  if(l==r) return tr[u].val;
  int mid=l+r>>1;
  if(x<=mid) return qval(ls(u), l, mid, x);
  return qval(rs(u), mid+1, r, x);
}

Msg find(int rt, int x){
  auto res=qval(rt, 1, n, x);
  if(res.fa==x) return res;
  return find(rt, res.fa);
}

void updfa(int &u, int p, int l, int r, int x, int k){
  u=++idx;
  tr[u]=tr[p];
  if(l==r){
    tr[u].val.fa=k;
    return;
  }
  int mid=l+r>>1;
  if(x<=mid) updfa(ls(u), ls(p), l, mid, x, k);
  else updfa(rs(u), rs(p), mid+1, r, x, k);
}

void updrk(int u, int l, int r, int x){
  if(l==r) return tr[u].val.rk++, void();
  int mid=l+r>>1;
  if(x<=mid) updrk(ls(u), l, mid, x);
  else updrk(rs(u), mid+1, r, x);
}

int main(){
  cin>>n>>Q;
  build(root[0], 1, n);

  int t, u, v, k;
  rep(ver,1,Q){ // current version
    read(t);
    root[ver]=root[ver-1];
    if(t==1){
      read(u), read(v);
      Msg fir=find(root[ver], u), sec=find(root[ver], v);

      if(fir.fa==sec.fa) continue;
      if(fir.rk>sec.rk) swap(fir, sec);
      updfa(root[ver], root[ver-1], 1, n, fir.fa, sec.fa); // merge
      if(fir.rk==sec.rk) updrk(root[ver], 1, n, sec.fa);
    }
    else if(t==2){
      read(k);
      root[ver]=root[k];
    }
    else if(t==3){
      read(u), read(v);
      Msg fir=find(root[ver], u), sec=find(root[ver], v);
      puts(fir.fa==sec.fa? "1": "0");
    }
  }

  return 0;
}
```

25.4 可持久化 Trie

例题： https://codeforces.com/contest/2093/problem/G

一个长度为 m 的数组 b 的美观度定义为所有可能数对 $1leilejlem$ 中 b_io+b_j 的最大值，其中 $xo+y$ 表示数字 x 和 y 的[按位异或](https://en.wikipedia.org/wiki/Bitwise_operation#XOR)。我们将数组 b 的美观度记为 $f(b)$。

如果一个数组 b 满足 $f(b)\geq k$，则称该数组是美观的。

你的任务是找出数组 a 中最短美观子数组的长度。如果不存在美观的子数组，则输出 -1。

```
const int N=2e5+5, M=33;

int n, K;
int w[N];

struct LTrie{
  int root[N];
  int tr[N*M][2];
  int cnt[N*M];
  int idx;

  void clear(){
    rep(i, 0, n) root[i]=0;
    rep(i, 0, idx){
      memset(tr[i], 0, sizeof tr[i]);
      cnt[i]=0;
    }
    idx=0;
  }

  void ins(int p, int q, int k, int x){
    if(k==-1) return;
    int val=x>>k&1;
    tr[q][val^1]=tr[p][val^1];
    tr[q][val]=++idx;
    cnt[tr[q][val]]=cnt[tr[p][val]]+1;
    ins(tr[p][val], tr[q][val], k-1, x);
  }

  void insert(int p, int q, int x){
    root[q]=++idx;
    ins((p==-1? 0: root[p]), root[q], 30, x);
  }

  int qry(int p, int q, int k, int x){
    if(k==-1) return 0;
    int val=x>>k&1;
    if(cnt[tr[q][val^1]] > cnt[tr[p][val^1]]) return (1<<k)|
qry(tr[p][val^1], tr[q][val^1], k-1, x);
    else return qry(tr[p][val], tr[q][val], k-1, x);
  }

  int query(int l, int r, int x){
    return qry(root[l-1], root[r], 30, x);
  }
}T;

bool ok(int l, int r, int x){
  return T.query(l, r, x)>=K;
}

void solve(){
  cin>>n>>K;
  rep(i, 1, n) read(w[i]);
  if(!K) return void(puts("1"));
  // T.insert(-1, 0, 0);
  T.insert(-1, 1, w[1]);
  int res=n+10;
  for(int i=2, j=1; i<=n; i++){
    while(j<=i-1 && ok(j, i-1, w[i])) ++j;
    if(ok(j-1, i-1, w[i])) res=min(res, i-j+2);
    T.insert(i-1, i, w[i]);
  }
  T.clear();
  if(res>n) res=-1;
  cout<<res<<"\n";
}
```

26 ODT

例题：luogu.com.cn/problem/CF896C

请你写一种奇怪的数据结构，支持：

- $1\ l\ r\ x$：将 $[l,r]$ 区间所有数加上 x
- $2\ l\ r\ x$：将 $[l,r]$ 区间所有数改成 x
- $3\ l\ r\ x$：输出将 $[l,r]$ 区间从小到大排序后的第 x 个数是的多少(即区间第 x 小，数字大小相同算多次，保证 $1\ l=x\ l=r-l+1$)
- $4\ l\ r\ x\ y$：输出 $[l,r]$ 区间每个数字的 x 次方的和模 y 的值(即 $(\sum_{i=l}^{r}a_i^x)\bmod y$)

```
struct node{
    int l, r;
    mutable ll v;
    node(int L, int R=-1, ll V=0): l(L), r(R), v(V) {};//构造函数
    bool operator < (const node&o)const{//运算符重载
        return l<o.l;
    }
};
set<node>s;
//分裂
IT split(int p){
    IT it=s.lower_bound(node(p));
    if(it->l==p) return it; //如果p在it对应的区间里面
    it--; //否则找上一个it (p必然在相应区间上)

    //模拟分裂区间
    int L=it->l; int R=it->r; ll V=it->v;
    s.erase(it);
    s.insert(node(L, p-1, V));
    return s.insert(node(p, R, V)).first; //insert会返回一个pair类型
first是迭代器 second是bool类型
}
//推平
void assign(int l, int r, ll v=0){
    IT itr=split(r+1); IT itl=split(l);
    s.erase(itl, itr);
    s.insert(node(l, r, v));
}
//sum up
void add(int l, int r, ll v=1){
    IT itr=split(r+1); IT itl=split(l);
    for(; itl!=itr; itl++){
        itl->v+=v;
    }
}
//rank
ll rk(int l, int r, int k){
    IT itr=split(r+1); IT itl=split(l);
    //建立一个vector取出区间 first存区间值 second存区间长
    vector<pair<ll, int>>v;
    v.clear();

    for(; itl!=itr; itl++){
        v.push_back(pair<ll, int>(itl->v, itl->r-itl->l+1));
    }
    sort(v.begin(), v.end());
    vector<pair<ll, int>>::iterator vit;
    for(vit=v.begin(); vit!=v.end(); vit++){
        k-=vit->second;
        if(k<=0) return vit->first;
    }
    return -1;
}
//幂次和
ll sum_pow(int l, int r, int k, int mo){
    ll res=0;
    IT itr=split(r+1); IT itl=split(l);
    for(; itl!=itr; itl++){
        res=((itl->r-itl->l+1)*fpow(itl->v, ll(k), ll(mo))+res)%mo;
        res%=mo;
    }
    return res;
}
```

27 LCT

例题：luogu.com.cn/problem/P3690

给定 n 个点以及每个点的权值，要你处理接下来的 m 个操作。操作有四种，操作从 0 到 3 编号。点从 1 到 n 编号。

- `0 x y` 代表询问从 x 到 y 的路径上的点的权值的 $\oplus$ 和。保证 x 到 y 是联通的。
- `1 x y` 代表连接 x 到 y，若 x 到 y 已经联通则无需连接。
- `2 x y` 代表删除边 (x,y)，不保证边 (x,y) 存在。
- `3 x y` 代表将点 x 上的权值变成 y。

```
struct Node{
  int s[2], p, v;
  int sum, rev;
}tr[N];
int stk[N];

void pushrev(int x){
  swap(tr[x].s[0], tr[x].s[1]);
  tr[x].rev^=1;
}

void pushup(int x){
  tr[x].sum=tr[tr[x].s[0]].sum^tr[x].v^tr[tr[x].s[1]].sum;
}

void pushdown(int x){
  if(tr[x].rev){
    pushrev(tr[x].s[0]), pushrev(tr[x].s[1]);
    tr[x].rev=0;
  }
}

bool isroot(int x){
  return tr[tr[x].p].s[0]!=x && tr[tr[x].p].s[1]!=x;
}

void rotate(int x){
  int y=tr[x].p, z=tr[y].p;
  int k=tr[y].s[1]==x;
  if(!isroot(y)) tr[z].s[tr[z].s[1]==y]=x;
  tr[x].p=z;
  tr[tr[x].s[k^1]].p=y, tr[y].s[k]=tr[x].s[k^1];
  tr[y].p=x, tr[x].s[k^1]=y;
  pushup(y), pushup(x);
}

void splay(int x){
  int top=0, r=x;
  stk[++top]=r;
  while(!isroot(r)) stk[++top]=r=tr[r].p;
  while(top) pushdown(stk[top--]);

  while(!isroot(x)){
    int y=tr[x].p, z=tr[y].p;
    if(!isroot(y))
      if((tr[y].s[1]==x) ^ (tr[z].s[1]==y)) rotate(x);
      else rotate(y);
    rotate(x);
  }
}

void access(int x){ // 建立从原树的根到 x 的路径，同时 x 成为 (splay) 根节点。
  int z=x;
  for(int y=0; x; y=x, x=tr[x].p){
    splay(x);
    tr[x].s[1]=y, pushup(x);
  }
  splay(z);
}

void makeroot(int x){ // x 成为原树的根节点
  access(x);
  pushrev(x);
}

int findroot(int x){ // 找到 x 所在原树的根节点，再将原树根节点转到 splay 的根节点
  access(x);
  while(tr[x].s[0]) pushdown(x), x=tr[x].s[0];
  splay(x);
  return x;
}

void split(int x, int y){ // x, y 路径用 splay 维护起来，splay 根为 y
  makeroot(x);
  access(y);
}

void link(int x, int y){ // 若 x, y 不连通，连边
  makeroot(x);
  if(findroot(y)!=x) tr[x].p=y;
}

void cut(int x, int y){ // x, y 间若直接连边，删除之
  makeroot(x); // 让 y 一定是 x 的后继
  if(findroot(y)==x && tr[y].p==x && !tr[y].s[0]){
    tr[x].s[1]=tr[y].p=0;
    pushup(x);
  }
}

int main(){
  cin>>n>>m;
  for(int i=1; i<=n; i++) read(tr[i].v);

  while(m--){
    int op, x, y; read(op), read(x), read(y);
    if(op==0){
      split(x, y);
      cout<<tr[y].sum<<'\n';
    }
    else if(op==1) link(x, y);
    else if(op==2) cut(x, y);
    else if(op==3){
      splay(x);
      tr[x].v=y;
      pushup(x);
    }
  }
  return 0;
}
```

例题 2：

https://codeforces.com/contest/1681/problem/F

You are given a tree, consisting of n vertices. Each edge has an integer value written on it.

Let $f(v,u)$ be the number of values that appear exactly once on the edges of a simple path between vertices v and u.

Calculate the sum of $f(v,u)$ over all pairs of vertices v and u such that $1\leq v<u\leq n$.

主要是介绍如何利用虚边维护 LCT 的子树大小（因为 LCT 一般是维护树的路径，子树信息较难维护）。

```
const int N=5e5+5;

vector<pii> col[N];
int n;

struct Node{
  int s[2], p;
  int sz, sz_v, rev;
}tr[N];
int stk[N];

void pushrev(int x){
  swap(tr[x].s[0], tr[x].s[1]);
  tr[x].rev^=1;
}

void pushup(int x){
  tr[x].sz=tr[tr[x].s[0]].sz+1+tr[tr[x].s[1]].sz+tr[x].sz_v;
}

void pushdown(int x){
  if(tr[x].rev){
    pushrev(tr[x].s[0]), pushrev(tr[x].s[1]);
    tr[x].rev=0;
  }
}

bool isroot(int x){
  return tr[tr[x].p].s[0]!=x && tr[tr[x].p].s[1]!=x;
}

void rotate(int x){
  int y=tr[x].p, z=tr[y].p;
  int k=tr[y].s[1]==x;
  if(!isroot(y)) tr[z].s[tr[z].s[1]==y]=x;
  tr[x].p=z;
  tr[tr[x].s[k^1]].p=y, tr[y].s[k]=tr[x].s[k^1];
  tr[y].p=x, tr[x].s[k^1]=y;
  pushup(y), pushup(x);
}

void splay(int x){
  int top=0, r=x;
  stk[++top]=r=tr[r].p;
  while(!isroot(r)) stk[++top]=r=tr[r].p;
  while(top) pushdown(stk[top--]);

  while(!isroot(x)){
    int y=tr[x].p, z=tr[y].p;
    if(!isroot(y))
      if((tr[y].s[1]==x) ^ (tr[z].s[1]==y)) rotate(x);
      else rotate(y);
    rotate(x);
  }
}

void access(int x){ // 建立从原树的根到 x 的路径，同时 x 成为 (splay) 根节点。
  int z=x;
  for(int y=0; x; y=x, x=tr[x].p){
    splay(x);
    tr[x].sz_v+=tr[tr[x].s[1]].sz-tr[y].sz; // // 维护虚边的子树大小和
    tr[x].s[1]=y, pushup(x);
  }
  splay(z);
}

void makeroot(int x){ // x 成为原树的根节点
  access(x);
  pushrev(x);
}

int findroot(int x){ // 找到 x 所在原树的根节点，再将原树根节点转到 splay 的根节点
  access(x);
  while(tr[x].s[0]) pushdown(x), x=tr[x].s[0];
  splay(x);
  return x;
}

void split(int x, int y){ // x, y 路径用 splay 维护起来，splay 根为 y
  makeroot(x);
  access(y);
}

void link(int x, int y){ // 若 x, y 不连通，连边
  makeroot(x);
  makeroot(y);
  tr[x].p=y;
  tr[y].sz_v+=tr[x].sz;
}

void cut(int x, int y){ // x, y 间若直接连边，删除之
  makeroot(x); // 让 y 一定是 x 的后继
  if(findroot(y)==x && tr[y].p==x && !tr[y].s[0]){
    tr[x].s[1]=tr[y].p=0;
    pushup(x);
  }
}

int get_sz(int u){
  makeroot(u);
  return tr[u].sz;
}
```

```
ll cal(int u, int v){
  return 1LL*get_sz(u)*get_sz(v);
}

void solve(){
  cin>>n;
  rep(i, 1, n) tr[i].sz=1;
  rep(i, 1, n-1){
    int u, v, c; read(u), read(v), read(c);
    col[c].pb({u, v});
    link(u, v);
  }

  ll res=0;
  rep(c, 1, n){
    for(auto &[u, v]: col[c]) cut(u, v);
    for(auto &[u, v]: col[c]) res+=cal(u, v);
    for(auto &[u, v]: col[c]) link(u, v);
  }
  cout<<res<<endl;
}

signed main(){
  solve();
  return 0;
}
```

28 K-D Tree

例题：www.luogu.com.cn/problem/P4148

你有一个$N \times N$的棋盘，每个格子内有一个整数，初始时的时候全部为 0，现在需要维护两种操作：

- 1 x y A $1 \leq x, y \leq N$，A 是正整数。将格子 x,y 里的数字加上 A。
- 2 x1 y1 x2 y2 $1 \leq x_1 \leq x_2 \leq N$，$1 \leq y_1 \leq y_2 \leq N$。输出 x_1, y_1, x_2, y_2 这个矩形内的数字和
- 3 无 终止程序

```
struct Point{
  int x[2], w;
};

struct Node{
  int l, r;
  Point P;
  int L[2], R[2], sum, sz;

  #define ls tr[u].l
  #define rs tr[u].r
}tr[N];

int n;

int idx, root;
int buf[N], tot;
int add(){
  if(!tot) return ++idx;
  return buf[tot--];
}

void pushup(int u){
  auto &L=tr[ls], &R=tr[rs];
  tr[u].sum=tr[u].P.w+L.sum+R.sum, tr[u].sz=L.sz+R.sz+1;

  rep(i,0,1){
    tr[u].L[i]=min(tr[u].P.x[i], min(L.L[i], R.L[i]));
    tr[u].R[i]=max(tr[u].P.x[i], max(L.R[i], R.R[i]));
  }
}

const double Al=0.75;

Point pt[N];

void getSeq(int u, int cnt){
  if(ls) getSeq(ls, cnt);
  buf[++tot]=u, pt[tr[ls].sz+1+cnt]=tr[u].P;
  if(rs) getSeq(rs, cnt+tr[ls].sz+1);
}

int rebuild(int l, int r, int k){
  if(l>r) return 0;
  int mid=l+r>>1;
  int u=add();

  nth_element(pt+l, pt+mid, pt+r+1, [&](Point a, Point b){
    return a.x[k]<b.x[k];
  });
  tr[u].P=pt[mid];

  ls=rebuild(l, mid-1, k^1), rs=rebuild(mid+1, r, k^1);
  pushup(u);
  return u;
}

void maintain(int &u, int k){
  if(tr[u].sz*Al<tr[ls].sz || tr[u].sz*Al<tr[rs].sz)
    getSeq(u, 0), u=rebuild(1, tot, k);
}

void insert(int &u, Point p, int k){
  if(!u){
    u=add();
    tr[u].l=tr[u].r=0;
    tr[u].P=p, pushup(u);
    return;
  }
  if(p.x[k]<=tr[u].P.x[k]) insert(ls, p, k^1);
  else insert(rs, p, k^1);
  pushup(u);
  maintain(u, k);
}

bool In(Node t, int x1, int y1, int x2, int y2){
  return t.L[0]>=x1 && t.R[0]<=x2 && t.L[1]>=y1 && t.R[1]<=y2;
}

bool In(Point p, int x1, int y1, int x2, int y2){
  return p.x[0]>=x1 && p.x[0]<=x2 && p.x[1]>=y1 && p.x[1]<=y2;
}

bool Out(Node t, int x1, int y1, int x2, int y2){
  return t.R[0]<x1 || t.L[0]>x2 || t.R[1]<y1 || t.L[1]>y2;
}

int query(int u, int x1, int y1, int x2, int y2){
  if(In(tr[u], x1, y1, x2, y2)) return tr[u].sum;
  if(Out(tr[u], x1, y1, x2, y2)) return 0;

  int res=0;
  if(In(tr[u].P, x1, y1, x2, y2)) res+=tr[u].P.w;
  res+=query(ls, x1, y1, x2, y2)+query(rs, x1, y1, x2, y2);
  return res;
}

int main(){
  cin>>n;
  // init
  tr[0].L[0]=tr[0].L[1]=INF;
  tr[0].R[0]=tr[0].R[1]=-INF;

  int res=0, op;
  while(cin>>op, op!=3){
    if(op==1){
      int x, y, k; read(x), read(y), read(k);
      insert(root, {x^res, y^res, k^res}, 0);
    }
    else{
      int x1, y1, x2, y2; read(x1), read(y1), read(x2), read(y2);
      write(res=query(root, x1^res, y1^res, x2^res, y2^res));
      puts("");
    }
  }

  return 0;
}
```

例题 2：

https://www.luogu.com.cn/problem/P7883

求平面最近点对。

```
const int N=5e5+5, INF=0x3f3f3f3f;

struct Point{
  int x[2];
};

struct Node{
  int l, r;
  Point P;
  int L[2], R[2], sz;

  #define ls tr[u].l
  #define rs tr[u].r
}tr[N];

int n;
int idx, root;

inline void pushup(int u){
  auto &L=tr[ls], &R=tr[rs];
  tr[u].sz=L.sz+R.sz+1;

  rep(i,0,1){
    tr[u].L[i]=min(tr[u].P.x[i], min(L.L[i], R.L[i]));
    tr[u].R[i]=max(tr[u].P.x[i], max(L.R[i], R.R[i]));
  }
}

Point pt[N];

int build(int l, int r, int k){
  if(l>r) return 0;
  int mid=l+r>>1;
  int u=++idx;

  nth_element(pt+l, pt+mid, pt+r+1, [&](Point a, Point b){
    return a.x[k]<b.x[k];
  });
  tr[u].P=pt[mid];

  ls=build(l, mid-1, k^1), rs=build(mid+1, r, k^1);
  pushup(u);
  return u;
}

ll res=4e18;

inline ll Dis(Point a, Point b){
  ll dx=a.x[0]-b.x[0], dy=a.x[1]-b.x[1];
  return dx*dx+dy*dy;
}

inline ll H(Node t, Point p){
  auto sqr=[](int x)->ll{return 1LL*x*x;};

  ll x=p.x[0], y=p.x[1];
  ll res=0;
  if(x<t.L[0]) res+=sqr(x-t.L[0]);
  if(x>t.R[0]) res+=sqr(x-t.R[0]);
  if(y<t.L[1]) res+=sqr(y-t.L[1]);
  if(y>t.R[1]) res+=sqr(y-t.R[1]);
  return res;
}

void query(int u, Point p){
  if(!u) return;
  if(tr[u].P.x[0]!=p.x[0] || tr[u].P.x[1]!=p.x[1]) res=min(res,
Dis(tr[u].P, p));
  ll LV=4e18, RV=4e18;
  if(ls) LV=H(tr[ls], p);
  if(rs) RV=H(tr[rs], p);

  if(LV<RV){
    if(LV<res) query(ls, p);
    if(RV<res) query(rs, p);
  }
  else{
    if(RV<res) query(rs, p);
    if(LV<res) query(ls, p);
  }
}

int main(){
  cin>>n;
  // init
  tr[0].L[0]=tr[0].L[1]=INF;
  tr[0].R[0]=tr[0].R[1]=-INF;

  rep(i,1,n){
    int x, y; read(x), read(y);
    pt[i]={x, y};
  }
  sort(pt+1, pt+1+n, [](Point a, Point b){
    return a.x[0]==b.x[0]? a.x[1]<b.x[1]: a.x[0]<b.x[0];
  });

  root=build(1, n, 0);
  rep(i,1,n) query(root, pt[i]);
  cout<<res<<endl;

  return 0;
}
```

29 莫队

块大小一般是 `block_size = std::max<int>(1, n / std::max(1.0, std::sqrt(Q * 2.0 / 3.0)));`

29.1 普通莫队

例题：www.luogu.com.cn/problem/P1494

具体来说，小 Z 把这 N 只袜子从 1 到 N 编号，然后从编号 L 到 R 的袜子中随机选出两只来穿。尽管小 Z 并不在意两只袜子是不是完整的一双，他却很在意袜子的颜色，毕竟穿两只不同色的袜子会很尴尬。你的任务便是告诉小 Z，他有多大的概率抽到两只颜色相同的袜子。当然，小 Z 希望这个概率尽量高，所以他可能会询问多个 (L, R) 以方便自己选择。

然而数据中有 $L = R$ 的情况，请特判这种情况，输出 `0/1`。

```
const int N=50050;

int n, m, w[N];
struct Query{
  int id, l, r, k;
  bool operator < (const Query &o)const{
    return k==o.k? ((k&1)? r<o.r: r>o.r): k<o.k;
  }
}q[N];

ll res;
pair<ll, ll> ans[N];

int b[N];

void add(int val){
  res+=b[val];
  b[val]++;
}

void del(int val){
  res-=b[val]-1;
  b[val]--;
}

int main(){
  cin>>n>>m;
  rep(i,1,n) read(w[i]);

  const int len=sqrt(n+1);
  rep(i,1,m){
    int l, r; read(l), read(r);
    q[i]={i, l, r, l/len};
  }

  sort(q+1, q+1+m);

  for(int i=0, j=1, k=1; k<=m; k++){
    int l=q[k].l, r=q[k].r;
    if(l==r){
      ans[q[k].id]={0, 1};
      continue;
    }
    while(i<r) add(w[++i]);
    while(i>r) del(w[i--]);
    while(j<l) del(w[j++]);
    while(j>l) add(w[--j]);
    ll g=__gcd(res, 1LL*(r-l+1)*(r-l)/2);
    ans[q[k].id]={res/g, 1LL*(r-l+1)*(r-l)/2/g};
  }

  rep(i,1,m) cout<<ans[i].x<<'/'<<ans[i].y<<endl;

  return 0;
}
```

29.2 带修莫队

例题：www.luogu.com.cn/problem/P1903

墨墨购买了一套 N 支彩色画笔（其中有些颜色可能相同），摆成一排，你需要回答墨墨的提问。墨墨会向你发布如下指令：

1. Q L R 代表询问你从第 L 支画笔到第 R 支画笔中共有几种不同颜色的画笔。
2. R P C 把第 P 支画笔替换为颜色 C。

```
const int N=2e5+5, S=1e6+5;

struct Query{
  int id, l, r, t;
}q[N];

struct Modify{
  int p, c;
}c[N];

int cnt[S];
int w[N], ans[N];
int n, m;
int mq, mc;
int len;

int get(int x){
  return x/len;
}

bool cmp(Query &a, Query &b){
  int al=get(a.l), ar=get(a.r), bl=get(b.l), br=get(b.r);
  if(al!=bl) return al<bl;
  if(ar!=br) return ar<br;
  return a.t<b.t;
}

void add(int v, int &res){
  if(!cnt[v]) res++;
  cnt[v]++;
}

void del(int v, int &res){
  if(cnt[v]==1) res--;
  cnt[v]--;
}

int main(){
  read(n), read(m);
  for(int i=1; i<=n; i++) read(w[i]);

  for(int i=0; i<m; i++){
    char op; int l, r; cin>>op; read(l), read(r);
    if(op=='Q') mq++, q[mq]={mq, l, r, mc};
    else mc++, c[mc]={l, r};
  }

  if(mc>100) len=cbrt((double)n*mc)+1;
  else len=sqrt(n)+1;

  sort(q+1, q+1+mq, cmp);

  for(int i=0, j=1, res=0, t=0, k=1; k<=mq; k++){
    int id=q[k].id, l=q[k].l, r=q[k].r, tm=q[k].t;
    while(i<r) add(w[++i], res);
```

```
    while(i>r) del(w[i--], res);
    while(j<l) del(w[j++], res);
    while(j>l) add(w[--j], res);

    while(t<tm){
      t++;
      if(c[t].p>=j && c[t].p<=i){
        del(w[c[t].p], res);
        add(c[t].c, res);
      }
      swap(w[c[t].p], c[t].c);
    }

    while(t>tm){
      if(c[t].p>=j && c[t].p<=i){
        del(w[c[t].p], res);
        add(c[t].c, res);
      }
      swap(w[c[t].p], c[t].c);
      t--;
    }

    ans[id]=res;
  }

  for(int i=1; i<=mq; i++) printf("%d\n", ans[i]);

  return 0;
}
```

29.3 回滚莫队

例题：luogu.com.cn/problem/AT_joisc2014_c

日记中记录了连续 N 天发生的事件，大约每天发生一件。

事件有种类之分。第 i 天发生的事件的种类用一个整数 X_i 表示，X_i 越大，事件的规模就越大。

JOI 教授决定用如下的方法分析这些日记：

- 选择日记中连续的几天 $[L, R]$ 作为分析的时间段；
- 定义事件 A 的重要度 W_A 为 $A \times T_A$，其中 T_A 为该事件在区间 $[L, R]$ 中出现的次数。

现在，您需要帮助教授求出所有事件中重要度最大的事件是哪个，并输出其重要度。

```
const int N=1e5+5;

int n, Q;
int w[N];

int blen, bcnt;
int L[N], R[N];
int bel[N];

struct Query{
  int l, r, id;
}q[N];

vector<int> raw;

int find(int x){
  return lower_bound(all(raw), x)-begin(raw);
}

int cnt[N], cnt2[N];
int res[N];

signed main(){
  cin>>n>>Q;
  rep(i,1,n) read(w[i]), raw.pb(w[i]);
  sort(all(raw));
  raw.erase(unique(all(raw)), end(raw));
  rep(i,1,n) w[i]=find(w[i]);

  blen=sqrt(n+0.5);
  bcnt=n/blen;
  rep(i, 1, bcnt){
    L[i]=R[i-1]+1;
    R[i]=L[i]+blen-1;
  }
  if(bcnt*blen!=n) ++bcnt, L[bcnt]=R[bcnt-1]+1, R[bcnt]=n;
  rep(i, 1, bcnt){
    rep(j,L[i],R[i]) bel[j]=i;
  }

  rep(i, 1, Q){
    int l, r; read(l), read(r);
    q[i]={l, r, i};
  }
  sort(q+1, q+1+Q, [&](const Query &a, const Query &b){
    return bel[a.l]==bel[b.l]? a.r<b.r: bel[a.l]<bel[b.l];
  });

  for(int k=1, idx=1; k<=bcnt; k++){
    int l=R[k]+1, r=R[k];
    memset(cnt, 0, sizeof cnt);
    int cur=0;
    for(; idx<=Q && bel[q[idx].l]==k; idx++){
      int ql=q[idx].l, qr=q[idx].r, id=q[idx].id;
      if(bel[ql]==bel[qr]){
        rep(i,ql,qr) cnt2[w[i]]=0;
        rep(i,ql,qr) cnt2[w[i]]++, res[id]=max(res[id],
raw[w[i]]*cnt2[w[i]]);
        continue;
      }
      while(r<qr){
        cnt[w[++r]]++;
        cur=max(cur, raw[w[r]]*cnt[w[r]]);
      }
      int rec=cur;
      while(l>ql){
        cnt[w[--l]]++;
        cur=max(cur, raw[w[l]]*cnt[w[l]]);
      }
      res[id]=cur;
      while(l<R[k]+1) cnt[w[l++]]--;
      cur=rec;
    }
  }
  rep(i, 1, Q) cout<<res[i]<<'\n';

  return 0;
}
```

30 树套树

例题：https://www.luogu.com.cn/problem/P3380

您需要写一种数据结构（可参考题目标题），来维护一个有序数列，其中需要提供以下操作：

1. 查询 k 在区间内的排名；
2. 查询区间内排名为 k 的值；
3. 修改某一位置上的数值；
4. 查询 k 在区间内的前驱（前驱定义为严格小于 x，且最大的数，若不存在输出 **-2147483647**）；
5. 查询 k 在区间内的后继（后继定义为严格大于 x，且最小的数，若不存在输出 **2147483647**）。

对于一组元素，一个数的排名被定义为严格比它小的元素个数加一，而排名为 k 的数被定义为“将元素从小到大排序后排在第 k 位的元素值”。

上古年代写的树套树：

```
const int N=15e5+5, INF=2147483647;

int n, q, w[N];

// 伸展树 data -----------------------------------------
struct node{
  int s[2], p, v;
  int size;

  void init(int _p, int _v){
    p=_p, v=_v;
    size=1;
  }
}tr[N];
int idx;

// 开始伸展树操作 ---------------------------------------
void pushup(int u){
  tr[u].size=tr[tr[u].s[0]].size+tr[tr[u].s[1]].size+1;
}

void rotate(int x){
  int y=tr[x].p, z=tr[y].p;
  int k= tr[y].s[1]==x;
  tr[z].s[tr[z].s[1]==y]=x, tr[x].p=z;
  tr[tr[x].s[k^1]].p=y, tr[y].s[k]=tr[x].s[k^1];
  tr[y].p=x, tr[x].s[k^1]=y;
  pushup(y), pushup(x);
}

void splay(int &root, int x, int k){
  while(tr[x].p!=k){
    int y=tr[x].p, z=tr[y].p;
    if(z!=k)
      if((tr[z].s[1]==y) ^ (tr[y].s[1]==x)) rotate(x);
      else rotate(y);
    rotate(x);
  }
  if(k==0) root=x;
}

void insert(int &root, int v){
  int u=root, p=0;
  while(u) p=u, u=tr[u].s[v>tr[u].v];
  u=++idx;
  if(p) tr[p].s[v>tr[p].v]=u;
  tr[u].init(p, v);
  splay(root, u, 0);
}

int get_k(int root, int v){
  int u=root, k=0;
  while(u){
    if(v>tr[u].v) k+=tr[tr[u].s[0]].size+1, u=tr[u].s[1];
    else u=tr[u].s[0];
  }
  return k;
}

void remove(int &root, int v){
  int u=root;
  while(u && tr[u].v!=v) u=tr[u].s[v>tr[u].v];
  splay(root, u, 0);

  int pu=tr[u].s[0], su=tr[u].s[1];
  while(tr[pu].s[1]) pu=tr[pu].s[1];
  while(tr[su].s[0]) su=tr[su].s[0];
  splay(root, pu, 0), splay(root, su, pu);
  tr[su].s[0]=0;
}

void update(int &root, int p, int q){
  remove(root, p), insert(root, q);
}

int get_pre(int root, int v){
  int u=root, res=-INF;
  while(u){
    if(tr[u].v<v) res=max(res, tr[u].v), u=tr[u].s[1];
    else u=tr[u].s[0];
  }
  return res;
}

int get_suf(int root, int v){
    int u=root, res=INF;
    while(u){
      if(v<tr[u].v) res=min(res, tr[u].v), u=tr[u].s[0];
      else u=tr[u].s[1];
    }
    return res;
}

// 线段树 data -----------------------------------------
int L[N], R[N], T[N];

// 开始线段树操作 ---------------------------------------
int ls(int u){return u<<1;}
int rs(int u){return u<<1|1;}

void build(int u, int l, int r){
  // 伸展树建树
  insert(T[u], -INF), insert(T[u], INF);
  for(int i=l; i<=r; i++) insert(T[u], w[i]);

  // 线段树建树
  L[u]=l, R[u]=r;
  if(l==r) return;
  int mid=l+r>>1;
  build(ls(u), l, mid), build(rs(u), mid+1, r);
}

int query_less(int u, int l, int r, int v){
  if(l<=L[u] && R[u]<=r) return get_k(T[u], v)-1;

  int mid=L[u]+R[u]>>1, res=0;
  if(l<=mid) res+=query_less(ls(u), l, r, v);
  if(r>mid) res+=query_less(rs(u), l, r, v);
  return res;
}

void change(int u, int p, int v){
  update(T[u], w[p], v);
  if(L[u]==R[u]) return;
  int mid=L[u]+R[u]>>1;
  if(p<=mid) change(ls(u), p, v);
  else change(rs(u), p, v);
}

int query_pre(int u, int l, int r, int v){
  if(l<=L[u] && R[u]<=r) return get_pre(T[u], v);
  int mid=L[u]+R[u]>>1, res=-INF;
  if(l<=mid) res=max(res, query_pre(ls(u), l, r, v));
  if(r>mid) res=max(res, query_pre(rs(u), l, r, v));
  return res;
}

int query_suf(int u, int l, int r, int v){
  if(l<=L[u] && R[u]<=r) return get_suf(T[u], v);
  int mid=L[u]+R[u]>>1, res=INF;

  if(l<=mid) res=min(res, query_suf(ls(u), l, r, v));
  if(r>mid) res=min(res, query_suf(rs(u), l, r, v));
  return res;
}

int main(){
  read(n), read(q);
  for(int i=1; i<=n; i++) read(w[i]);

  build(1, 1, n);

  while(q--){
    int op; read(op);
    if(op==1){
      int l, r, v; read(l), read(r), read(v);
      cout<<query_less(1, l, r, v)+1<<endl;
    }else if(op==2){
      int l, r, k; read(l), read(r), read(k);
      int ok=0, ng=1e8;
      while(ok<ng){
        int mid=ok+ng+1>>1;
        if(query_less(1, l, r, mid)+1<=k) ok=mid;
        else ng=mid-1;
      }
      cout<<ok<<endl;
    }else if(op==3){
      int p, v; read(p), read(v);
      change(1, p, v);
      w[p]=v;
    }else if(op==4){
      int l, r, v; read(l), read(r), read(v);
      cout<<query_pre(1, l, r, v)<<endl;
    }else if(op==5){
      int l, r, v; read(l), read(r), read(v);
      cout<<query_suf(1, l, r, v)<<endl;
    }
  }

  return 0;
}
```

31 猫树

31.1 普通猫树（叫这个名字只是为了和下面猫树分治区分开）

https://www.luogu.com.cn/problem/SP1043

每次查询区间的最大子段和。

```
const int N=1e5+5, M=20;

int n, w[N];

struct CatTree{
  int sz;
  int f[M][N<<2], g[M][N<<2];
  int loc[N<<2];

  #define ls(u) (u<<1)
  #define rs(u) (u<<1|1)

  void get_sz(int n){
    sz=1;
    while(sz<n) sz<<=1;
  }

  void build(int u, int l, int r, int d){
    if(l==r) return void(loc[l]=u);
    int mid=l+r>>1;
    int s, sm;

    // lef
    s=sm=f[d][mid]=g[d][mid]=w[mid];
    dwn(i, mid-1, l){
      sm=max(0, sm);
      s+=w[i], sm+=w[i];
      f[d][i]=max(f[d][i+1], s);
      g[d][i]=max(g[d][i+1], sm);
    }

    // rig
    s=sm=f[d][mid+1]=g[d][mid+1]=w[mid+1];
    rep(i, mid+2, r){
      sm=max(0, sm);
      s+=w[i], sm+=w[i];
      f[d][i]=max(f[d][i-1], s);
      g[d][i]=max(g[d][i-1], sm);
    }

    build(ls(u), l, mid, d+1);
    build(rs(u), mid+1, r, d+1);
  }

  int query(int l, int r){
    if(l==r) return w[l];
    int d=__lg(loc[l])-__lg(loc[l]^loc[r]);
    return max({f[d][l]+f[d][r], g[d][l], g[d][r]});
  }
}T;

void solve(){
  cin>>n;
  rep(i, 1, n) read(w[i]);

  T.get_sz(n);
  T.build(1, 1, T.sz, 1);
  int Q; cin>>Q;
  rep(_, 1, Q){
    int l, r; read(l), read(r);
    cout<<T.query(l, r)<<"\n";
  }
}
```

```
signed main(){
  solve();
  return 0;
}
```

31.2 猫树分治

例题：

https://www.luogu.com.cn/problem/P6240

有一条小吃街，从左到右依次排列着 n 个商店，从 1 开始标号。

第 i 个商店会只出售一种小吃，热量为 h_i，美味度为 w_i。

现在有 m 个吃货要来逛街，第 i 个吃货会在 $[l_i,r_i]$ 的商店内寻找小吃，而且为了防止太胖，最多能摄入 t_i 的热量。

小吃吃多了会腻，所以同一个商店的小吃只能吃一次。

现在每个吃货想知道自己最多能得到多少美味度。

```
// 猫树分治!
// 合并背包复杂度为容量限制 M
// 那就是 build: nlogn*M
//      + query: Q*M

const int N=40040, M=202, QLIM=2e5+5;

int n, Q;
int h[N], w[N];

struct Query{
  int l, r, t;
}q[QLIM];
vector<int> QID;

int pre[N][M], suf[N][M];
int res[QLIM];

void solve(int l, int r, vector<int> &QID){
  if(l==r){
    for(auto id: QID){
      if(q[id].t>=h[l]) res[id]=w[l];
      else res[id]=0;
    }
    return;
  }
  int mid=l+r>>1;

  // DP
  // lef
  memset(suf[mid+1], 0, sizeof suf[mid+1]);
  dwn(i, mid, l){
    rep(j, 0, M-1) suf[i][j]=suf[i+1][j];
    rep(j, h[i], M-1) suf[i][j]=max(suf[i][j], suf[i+1][j-
h[i]]+w[i]);
  }
  // rig
  memset(pre[mid], 0, sizeof pre[mid]);
  rep(i, mid+1, r){
    rep(j, 0, M-1) pre[i][j]=pre[i-1][j];
    rep(j, h[i], M-1) pre[i][j]=max(pre[i][j], pre[i-1][j-
h[i]]+w[i]);
  }


  vector<int> LQ, RQ;
  for(auto id: QID){
    auto &[ql, qr, t]=q[id];
    if(qr<=mid) LQ.pb(id);
    else if(ql>mid) RQ.pb(id);
    else{
      int cur=0;
      rep(i, 0, t) cur=max(cur, suf[ql][i]+pre[qr][t-i]);
      res[id]=cur;
    }
  }
  solve(l, mid, LQ), solve(mid+1, r, RQ);
}

void solve(){
  cin>>n>>Q;
  rep(i, 1, n) read(h[i]);
  rep(i, 1, n) read(w[i]);

  rep(i, 1, Q){
    read(q[i].l), read(q[i].r), read(q[i].t);
    QID.pb(i);
  }
  solve(1, n, QID);
  rep(i, 1, Q) cout<<res[i]<<"\n";
}

signed main(){
  solve();
  return 0;
}
```

32 笛卡尔树

例题：

https://www.luogu.com.cn/problem/P5854

给定一个 $1\sim n$ 的排列 p，构建其笛卡尔树。

即构建一棵二叉树，满足：

1. 每个节点的编号满足二叉搜索树的性质。
2. 节点 i 的权值为 p_i，每个节点的权值满足小根堆的性质。

```
const int N=1e7+50;

int n, w[N];
pii tr[N];
int top, pre, stk[N];

int main(){
  cin>>n;
  rep(i,1,n) read(w[i]);
  // build
  rep(i,1,n){
    int val=w[i];
    pre=top;
    while(pre && w[stk[pre]]>val) pre--;
    if(pre<top) tr[i].x=stk[pre+1];
    if(pre) tr[stk[pre]].y=i;
    top=++pre;
    stk[top]=i;
  }

  ll fir=0, sec=0;
  rep(i,1,n) fir^=1LL*i*(tr[i].x+1), sec^=1LL*i*(tr[i].y+1);
  cout<<fir<<' '<<sec<<endl;

  return 0;
}
```

33 字符串哈希

33.1 i64 模数代替双哈希

```
#define int long long

const int mod=100000000000000021, B=10000000000777;

int mul(int a, int b){
  return (__int128)a*b%mod;
}

int add(int x, int y){
  return (x+y)%mod;
}

int get_hash(string &s){
  int res=0;
  for(auto e: s) res=add(mul(res, B), e);
  return res;
}
```

33.2 二维哈希

例题：

https://www.luogu.com.cn/problem/P10474

给定一个 M 行 N 列的 01 矩阵，以及 Q 个 A 行 B 列的 01 矩阵，你需要求出这 Q 个矩阵哪些在原矩阵中出现过。

所谓 01 矩阵，就是矩阵中所有元素不是 0 就是 1。

```
const int N=1010;

int n, m, A, B;
ull w[N][N];

struct Hash_2D{
  static const ull B1=503, B2=1013;

  ull f[N][N], g[N][N];
  ull pw1[N], pw2[N];
  set<ull> st;

  void build(ull w[N][N], int n, int m){
    pw1[0]=pw2[0]=1;
    rep(i, 1, m) pw1[i]=pw1[i-1]*B1;
    rep(i, 1, n) pw2[i]=pw2[i-1]*B2;
    rep(i, 1, n) rep(j, 1, m) f[i][j]=f[i][j-1]*B1+w[i][j];
    rep(i, 1, n) rep(j, 1, m) g[i][j]=g[i-1][j]*B2+f[i][j];
  }

  ull get(int ax, int ay, int bx, int by){
    return g[bx][by]-g[ax-1][by]*pw2[bx-ax+1]-g[bx][ay-1]*pw1[by-
ay+1]+g[ax-1][ay-1]*pw2[bx-ax+1]*pw1[by-ay+1];
  }

  void get_st(int A, int B){
    rep(ax, 1, n) rep(ay, 1, m){
      int bx=ax+A-1, by=ay+B-1;
      if(bx>n || by>m) continue;
      st.insert(get(ax, ay, bx, by));
    }
  }

  void query(int A, int B){
    rep(i, 1, A) rep(j, 1, B){
      char c; cin>>c;
      f[i][j]=f[i][j-1]*B1+ull(c);
    }
    rep(i, 1, A) rep(j, 1, B) g[i][j]=g[i-1][j]*B2+f[i][j];
    if(st.count(get(1, 1, A, B))) puts("1");
    else puts("0");
  }
}H;

void solve(){
  cin>>n>>m>>A>>B;
  rep(i, 1, n) rep(j, 1, m){
    char c; cin>>c;
    w[i][j]=int(c);
  }
  H.build(w, n, m);
  H.get_st(A, B);

  int Q; cin>>Q;
  rep(_, 1, Q) H.query(A, B);
}

signed main(){
  solve();
  return 0;
}
```

34 KMP

例题：https://www.luogu.com.cn/problem/P3375

给出两个字符串 s_1 和 s_2，若 s_1 的区间 $[l,r]$ 子串与 s_2 完全相同，则称 s_2 在 s_1 中出现了，其出现位置为 l。现在请你求出 s_2 在 s_1 中所有出现的位置。

定义一个字符串 s 的 border 为 s 的一个非 s 本身的子串 t，满足 t 既是 s 的前缀，又是 s 的后缀。对于 s_2，你还需要求出对于其每个前缀 s' 的最长 border t' 的长度。

```
string s, p;
int m, n;
int nx[N];

void solve(){
  // 输入类似于:
  // ABABABC
  // ABA

  cin>>s>>p;
  m=s.size(), n=p.size();
  s=" "+s, p=" "+p;
  rep(i, 2, n){
    int j=nx[i-1];
    while(j && p[j+1]!=p[i]) j=nx[j];
    if(p[j+1]==p[i]) ++j;
    nx[i]=j;
  }

  for(int i=1, j=0; i<=m; i++){
    while(j && p[j+1]!=s[i]) j=nx[j];
    if(p[j+1]==s[i]) ++j;
    if(j==n){
      j=nx[j];
      cout<<i-n+1<<"\n";
    }
  }
  rep(i, 1, n) cout<<nx[i]<<" ";
  puts("");
}
```

35 AC 自动机

例题： luogu.com.cn/problem/P5357

给你一个文本串 S 和 n 个模式串 $T_{1\sim n}$，请你分别求出每个模式串 T_i 在 S 中出现的次数。

```
const int N=2e5+5;

int n;

struct AC{
  int tr[N][26], idx;
  int fail[N];
  int cnt[N];
  vector<int> grp[N];
  vector<int> T[N];
  int res[N];

  void clear(int u){
    memset(tr[u], 0, sizeof tr[u]);
    fail[u]=cnt[u]=0;
  }

  void init(){
    idx=0;
    clear(0);
  }

  void insert(string &s, int index){
    int u=0;
    for(auto &c: s){
      int val=c-'a';
      int &go=tr[u][val];
      if(!go) go=++idx, clear(idx);
      u=go;
    }
    grp[u].pb(index);
  }

  void build(){
    queue<int> q;
    rep(i,0,25) if(tr[0][i]){
      q.push(tr[0][i]);
      fail[tr[0][i]]=0;
    }
    while(q.size()){
      int u=q.front(); q.pop();
      rep(i,0,25){
        int &go=tr[u][i];
        if(!go) go=tr[fail[u]][i];
        else fail[go]=tr[fail[u]][i], q.push(go);
      }
    }

    rep(i,1,idx) T[fail[i]].pb(i);
  }

  void dfs(int u){
    for(auto &go: T[u]){
      dfs(go);
      cnt[u]+=cnt[go];
    }
  }

  void query(string &s){
    int u=0;
    for(auto &c: s){
      int val=c-'a';
      u=tr[u][val];
      cnt[u]++;
    }
    dfs(0);
    rep(i,1,idx) if(grp[i].size()){
      for(auto &id: grp[i]) res[id]=cnt[i];
    }
    rep(i,1,n) cout<<res[i]<<'\n';
  }
}ac;

signed main(){
  ios::sync_with_stdio(false);
  cin>>n;
  ac.init();
  rep(i,1,n){
    string s; cin>>s;
    ac.insert(s, i);
  }
  ac.build();
  string s; cin>>s;
  ac.query(s);

  return 0;
}
```

36 后缀数组

SA 关键性质是

$$\mathrm{lcp}(\mathrm{sa}[i],\mathrm{sa}[j])=\min(\{\mathrm{height}[i+1..j]\})$$

例题：

https://uoj.ac/problem/35

读入一个长度为 n 的由小写英文字母组成的字符串，请把这个字符串的所有非空后缀按字典序从小到大排序，然后按顺序输出后缀的第一个字符在原串中的位置。位置编号为 1 到 n。

除此之外为了进一步证明你确实有给后缀排序的超能力，请另外输出 $n-1$ 个整数分别表示排序后相邻后缀的最长公共前缀的长度。

```
const int N=1e6+5;

int n, m;
int sa[N], rk[N], r2[N], cnt[N];
int height[N];
string s;

void rsort(){
  rep(i, 0, m) cnt[i]=0;
  rep(i, 1, n) cnt[rk[i]]++;
  rep(i, 1, m) cnt[i]+=cnt[i-1];
  dwn(i, n, 1) sa[cnt[rk[r2[i]]]--]=r2[i];
}

void get_sa(){
  m=256;
  rep(i, 1, n) rk[i]=s[i], r2[i]=i;
  rsort();
  for(int len=1; ; len<<=1){
    int tmp=0;
```

```
    rep(i, 1, len) r2[++tmp]=n-len+i;
    rep(i, 1, n) if(sa[i]>len) r2[++tmp]=sa[i]-len;
    rsort();
    swap(r2, rk), rk[sa[1]]=tmp=1;
    rep(i, 2, n) rk[sa[i]]=(r2[sa[i]]==r2[sa[i-1]] &&
r2[sa[i]+len]==r2[sa[i-1]+len]? tmp: ++tmp);
    m=tmp;
    if(m==n) break;
  }
}

void get_height(){
  rep(i, 1, n) rk[sa[i]]=i;
  for(int i=1, k=0; i<=n; i++){
    if(rk[i]==1) continue;
    if(k) --k;
    int j=sa[rk[i]-1];
    while(i+k<=n && j+k<=n && s[i+k]==s[j+k]) ++k;
    height[rk[i]]=k;
  }
}

void solve(){
  ios::sync_with_stdio(0);
  cin>>s;
  n=s.size();
  s=" "+s;
  get_sa();
  get_height();
  rep(i, 1, n) cout<<sa[i]<<" ";
  cout<<endl;
  rep(i, 2, n) cout<<height[i]<<" ";
  cout<<endl;
}
```

37 后缀自动机

简单来说，就是使用一个 DAG 以及一棵树维护一个字符串所有子串（压缩的）信息。

其中 DAG 的点称为状态。

endpos

个人认为 SAM 的核心在于 endpos。

子串（终点）在原串出现的下标集合称为 endpos 集合。

例如，对于原串 aabaaba，$\text{endpos}(\text{ab})=\{3,6\}$。

endpos 有如下性质：

- 对于任意子串 a,b，二者 endpos 的关系必为：
  1. $\text{endpos}(a)\subseteq\text{endpos}(b)$
  2. $\text{endpos}(a)\cap\text{endpos}(b)=\emptyset$

  之一。
- endpos 相同的两个子串属于一个等价类。（这意味着：SAM 中每个状态会对应 endpos 相同的若干个子串）
- 原串中 endpos 对应的等价类数量级为 $O(N)$。
- 对于一个状态 st 及任意的 longest(st) 的后缀 s，若有：$|\text{shortest(st)}|\le|s|\le|\text{longsest(st)}|$，则 $s\in\text{substrings(st)}$。

记 $|\text{shortest(st)}|$ 为 minlen(st)，$|\text{longest(st)}|$ 为 len(st)。

例题：luogu.com.cn/problem/P3804

给定一个只包含小写字母的字符串 S。

请你求出 S 的所有出现次数不为 1 的子串的出现次数乘上该子串长度的最大值。

```
struct Edge{
  int to, next;
}e[M];

int h[N], idx;

void add(int u, int v){
  e[idx].to=v, e[idx].next=h[u], h[u]=idx++;
}

char str[N];

struct Node{
  int len, fa; // longest length of state, link
  int ch[26];
}node[N];
int tot=1, last=1;
int f[N];

void ins(int c){
  int p=last, np=last=++tot;
  f[tot]=1;
  node[np].len=node[p].len+1;
  for(; p && !node[p].ch[c]; p=node[p].fa) node[p].ch[c]=np;
  if(!p) node[np].fa=1;
  else{
    int q=node[p].ch[c];
    if(node[q].len==node[p].len+1) node[np].fa=q;
    else{
      int nq=++tot;
      node[nq]=node[q], node[nq].len=node[p].len+1;
      node[q].fa=node[np].fa=nq;
      for(; p && node[p].ch[c]==q; p=node[p].fa) node[p].ch[c]=nq;
    }
  }
}

int res=0;

void dfs(int u){
  for(int i=h[u]; ~i; i=e[i].next){
    int go=e[i].to;
    dfs(go);
    f[u]+=f[go];
  }
  if(f[u]>1) res=max(res, f[u]*node[u].len);
}

signed main(){
  scanf("%s", str);
  for(int i=0; str[i]; i++) ins(str[i]-'a');
  memset(h, -1, sizeof h);
  for(int i=2; i<=tot; i++) add(node[i].fa, i);
  dfs(1);

  cout<<res<<endl;

  return 0;
}
```

38 PAM

例题：

给定一个字符串 s。保证每个字符为小写字母。对于 s 的每个位置，请求出以该位置结尾的回文子串个数。

这个字符串被进行了加密，除了第一个字符，其他字符都需要通过上一个位置的答案来解密。

具体地，若第 $i(i\ge1)$ 个位置的答案是 k，第 $i+1$ 个字符读入时的 ASCII 码为 c，则第 $i+1$ 个字符实际的 ASCII 码为 $(c-97+k)\bmod26+97$。所有字符在加密前后都为小写字母。

https://www.luogu.com.cn/problem/P5496

```
const int N=5e5+5, M=26;

struct PAM{
  struct Tr{
    int ch[M];
    int fail, len, cnt;
  }tr[N];

  int last, idx;
  int n, cur, s[N];

  PAM(){
    last=0; // 上一次插入的点编号
    idx=1; // 状态数
    tr[0].len=0, tr[0].fail=1;
    tr[1].len=-1, tr[1].fail=0;
  }

  int getfail(int x){
    while(s[cur-tr[x].len-1]!=s[cur]) x=tr[x].fail;
    return x;
  }

  void insert(int c){
    int p=getfail(last);
    if(!tr[p].ch[c]){
      ++idx;
      tr[idx].len=tr[p].len+2;
      tr[idx].fail=tr[getfail(tr[p].fail)].ch[c];
      tr[idx].cnt=tr[tr[idx].fail].cnt+1;
      tr[p].ch[c]=idx;
    }
    last=tr[p].ch[c];
  }

  void solve(string &str){
    n=str.size();
    int last_ans=0;
    str=" "+str;
    s[0]=M;
    for(cur=1; cur<=n; cur++){
      s[cur]=(str[cur]-'a'+last_ans)%M;
      insert(s[cur]);
      cout<<(last_ans=tr[last].cnt)<<" ";
    }
  }
}pam;

signed main(){
  string s; cin>>s;
  pam.solve(s);
  return 0;
}
```

39 DP

39.1 数位 DP

例题：

https://codeforces.com/problemset/problem/1710/C

题意

统计有多少个三元组 (a,b,c) 满足：$a\oplus b,b\oplus c,a\oplus c$ 能组成非退化三角形。其中 $a,b,c\in[0,m]$。

思路

这题形式一看就非常的数位 DP。

先考虑条件如何化为等价容易被 DP 表出的状态：

记 $a',b',c'\in[0,1]$。我们注意到 $(a'\oplus c')+(a'\oplus b')\ge(b'\oplus c')$ 是一定成立的，轮换情况同理。

所以对于非退化三角形的限制之一 $(a\oplus c)+(a\oplus b)>(b\oplus c)$，只要存在一位 (a',b',c') 满足 $(a'\oplus c')+(a'\oplus b')>(b'\oplus c')$ 即可满足该限制，其他两种限制同理。

故维护 DP 数组 $f(x,\text{fl}_a,\text{fl}_b,\text{fl}_c,\text{st}_1,\text{st}_2,\text{st}_3)$ 表示到第 x 位，fl_a 为 a 值是否贴边，st_1 表示约束 1 满足与否；其他变量同理。

做法

转移见代码。

代码

```
const int N=2e5+5, mod=998244353;

vector<int> val;
int f[N][2][2][2][2][2][2];

int add(int x, int y){
  return x+y>=mod? x+y-mod: x+y;
}

int cal(int a, int b, int c){
  return (a^c)+(a^b)>(b^c);
}

int dfs(int x, int fl_a, int fl_b, int fl_c, int st1, int st2, int
st3){
  if(x==-1) return st1 && st2 && st3;
  int &res=f[x][fl_a][fl_b][fl_c][st1][st2][st3];
  if(~res) return res;
  res=0;
  rep(a, 0, fl_a? val[x]: 1) rep(b, 0, fl_b? val[x]: 1) rep(c, 0,
fl_c? val[x]: 1){
    int nx_fl_a=(fl_a && a==val[x]);
    int nx_fl_b=(fl_b && b==val[x]);
    int nx_fl_c=(fl_c && c==val[x]);
    int nx_st1=st1|cal(a, b, c), nx_st2=st2|cal(b, a, c),
nx_st3=st3|cal(c, a, b);
    res=add(res, dfs(x-1, nx_fl_a, nx_fl_b, nx_fl_c, nx_st1, nx_st2,
nx_st3));
  }
  return res;
}

void solve(){
  string s; cin>>s;
  while(s.size()) val.pb(s.back()-'0'), s.pop_back();
  memset(f, -1, sizeof f);
  cout<<dfs(val.size()-1, 1, 1, 1, 0, 0, 0)<<"\n";
}
```

39.2 斜率优化

例题：

https://www.luogu.com.cn/problem/P2120

L 公司有 n 个工厂，由高到低分布在一座山上，工厂 1 在山顶，工厂 n 在山脚。

由于这座山处于高原内陆地区（干燥少雨），L 公司一般把产品直接堆放在露天，以节省费用。突然有一天，L 公司的总裁 L 先生接到气象部门的电话，被告知三天之后将有一场暴雨，于是 L 先生决定紧急在某些工厂建立一些仓库以免产品被淋坏。

由于地形的不同，在不同工厂建立仓库的费用可能是不同的。第 i 个工厂目前已有成品 p_i 件，在第 i 个工厂位置建立仓库的费用是 c_i。

对于没有建立仓库的工厂，其产品应被运往其他的仓库进行储藏，而由于 L 公司产品的对外销售处设置在山脚的工厂 n，故产品只能往山下运（即只能运往编号更大的工厂的仓库），当然运送产品也是需要费用的，一件产品运送一个单位距离的费用是 1。

假设建立的仓库容量都都是足够大的，可以容下所有的产品。你将得到以下数据：

- 工厂 i 距离工厂 1 的距离 x_i（其中 $x_1=0$）。
- 工厂 i 目前已有成品数量 p_i。
- 在工厂 i 建立仓库的费用 c_i。

请你帮助 L 公司寻找一个仓库建设的方案，使得总的费用（建造费用 + 运输费用）最小。

```
const int N=1e6+50;

int n;
int x[N], p[N], c[N];
int sum[N], Sum[N];
int f[N];

int q[N];
int l, r;

int X(int i){return sum[i];}
int Y(int i){return f[i]+Sum[i];}

bool del(int a, int b, int k){
  return k*(X(b)-X(a))>Y(b)-Y(a);
}

bool ins(int x, int y, int z){
  return (Y(y)-Y(x))*(X(z)-X(x))>(X(y)-X(x))*(Y(z)-Y(x));
}

signed main(){
  cin>>n;
  rep(i,1,n){
    read(x[i]), read(p[i]), read(c[i]);
    sum[i]=sum[i-1]+p[i];
    Sum[i]=Sum[i-1]+p[i]*x[i];
  }

  r=-1, l=0;
  q[++r]=0;

  rep(i,1,n){
    while(r>l && del(q[l], q[l+1], x[i])) l++;
    f[i]=f[q[l]]-x[i]*sum[q[l]]+Sum[q[l]]+x[i]*sum[i-1]-
Sum[i-1]+c[i];
    while(r>l && ins(q[r-1], q[r], i)) r--;
    q[++r]=i;
  }

  int res=f[n];
  while(!p[n] && n>1) n--, res=min(res, f[n]);
  cout<<res<<endl;

  return 0;
}
```

39.3 四边形不等式优化

若对于 $a\le b\le c\le d$ 下面

$$w(a,b)+w(b,d)\le w(a,d)+w(b,c)$$

恒成立。

则称函数 w 满足四边形不等式。

例题：

https://www.luogu.com.cn/problem/P4072

从 S 地到 T 地的路可以划分成 n 段，相邻两段路的分界点设有休息站。

Pine 计划用 m 天到达 T 地。除第 m 天外，每一天晚上 Pine 都必须在休息站过夜。所以，一段路必须在同一天中走完。

Pine 希望每一天走的路长度尽可能相近，所以他希望每一天走的路的长度的方差尽可能小。

帮助 Pine 求出最小方差是多少。

设方差是 v，可以证明，$v\times m^2$ 是一个整数。为了避免精度误差，输出结果时输出 $v\times m^2$。

```
// ans = m sum((seg[i]-avg)^2) = [sum (m seg[i]^2 - 2S seg[i])] +
S^2

const int N=3030, INF=1e18;

int n, m;
int w[N], s[N];
int S;
int f[2][N];

int sum(int l, int r){
  int seg=s[r]-s[l-1];
  return m*seg*seg-S*seg*2;
}

void divi(int *f, int *g, int l, int r, int lo, int ro){
  if(l>r || lo>ro) return;
```

```
  int mid=l+r>>1;
  pii best={INF, -1};
  rep(k, lo, min(ro, mid)) best=min(best, {f[k-1]+sum(k, mid), k});
  g[mid]=best.x;
  int idx=best.y;
  divi(f, g, l, mid-1, lo, idx);
  divi(f, g, mid+1, r, idx, ro);
}

void solve(){
  cin>>n>>m;
  rep(i, 1, n) read(w[i]), S+=w[i], s[i]=w[i]+s[i-1];
  memset(f, 0x3f, sizeof f);
  rep(i, 1, n) f[1][i]=sum(1, i);
  rep(i, 2, m) divi(f[i-1&1], f[i&1], 1, n, 1, n);
  cout<<f[m&1][n]+S*S<<"\n";
}

signed main(){
  solve();
  return 0;
}
```

40 随机化

40.1 模拟退火

想要跑满某时间可以用：

```
while ((double)clock()/CLOCKS_PER_SEC<0.80) anneal();。
```

例题：

给你 n 个二维平面上的点，让你找到一个点使得到 n 个点的距离之和最小。

```
#define x first
#define y second
typedef pair<double, double> PDD;

const int N=105;
PDD p[N];
int n;
double ans=1e9;

double rand(double l, double r){
    return (double)rand()/RAND_MAX*(r-l)+l;
}

double getd(PDD u, PDD v){
    double dx=u.x-v.x, dy=u.y-v.y;
    return sqrt(dx*dx+dy*dy);
}

double calc(PDD pt){
    double res=0;
    for(int i=1; i<=n; i++) res+=getd(pt, p[i]);
    ans=min(ans, res);
    return res;
}

void anneal(){
    PDD cur(rand(0, 1e4), rand(0, 1e4));
    for(double t=1e4; t>1e-4; t*=0.99){
        PDD np(rand(cur.x-t, cur.x+t), rand(cur.y-t, cur.y+t));
        double dt=calc(np)-calc(cur);
        if(exp(-dt/t)>rand(0, 1)) cur=np;
    }
}

int main(){
    cin>>n;
    for(int i=1; i<=n; i++){
        double x, y; cin>>x>>y;
        p[i]={x, y};
    }

    for(int i=0; i<100; i++) anneal();
    printf("%.0lf", ans);

    return 0;
}
```

例题 2：

https://www.luogu.com.cn/problem/P4044

题意简述：

一场保龄球比赛有 n 轮，每轮选手有两次投球机会击倒 10 个木瓶。每轮的得分是两次投球击倒的木瓶总数，并根据以下规则影响总分：

1. 全中（第一次投球击倒全部 10 瓶）：
   - 无需第二次投球。
   - 下一轮的得分 整体乘 **2** 计入总分。
   - 如果第 n 轮是“全中”，则增加一轮附加轮（共 $n+1$ 轮），附加轮的得分必须乘 **2**，但不会触发更多附加轮。

2. 补中（两次投球共击倒 10 瓶）：
   - 下一轮的 第一次投球得分乘 **2** 计入总分。

3. 失误（两次投球未击倒全部 10 瓶）：
   - 下一轮得分正常计入总分。

目标：重新排列所有轮次（需保持附加轮条件不变，即第 n 轮是否为“全中”与原顺序一致），使得按规则计算的总分最大化。

```
#define x first
#define y second
typedef pair<int, int> PII;

const int N=55;

int n, m;
PII p[N];

int ans;

int calc(){
  int res=0;
  for(int i=0; i<m; i++){
    res+=p[i].x+p[i].y;
    if(i==m-1) continue;

    if(p[i].x==10) res+=p[i+1].x+p[i+1].y;
    else if(p[i].x+p[i].y==10) res+=p[i+1].x;
  }
  ans=max(ans, res);
  return res;
}

void anneal(){
  bool ok=m-n; // 是否加了一场
  for(double t=1e4; t>1e-4; t*=0.99){
    int u=rand()%m, v=rand()%n;
    int cur=calc();
    swap(p[u], p[v]);

    if(p[n-1].x==10 && !ok || p[n-1].x!=10 && ok) swap(p[u],
p[v]); // 不合法情况
    else{
      int np=calc();
      int delta=np-cur;
      if(exp(delta/t)<(double)rand()/RAND_MAX) swap(p[u], p[v]);  //
跳不过去
    }
  }
}

int main(){
  cin>>n;
  for(int i=0; i<n; i++) cin>>p[i].x>>p[i].y;
  if(p[n-1].x==10) m=n+1, cin>>p[n].x>>p[n].y;
  else m=n;

  for(int i=0; i<100; i++) anneal();
  cout<<ans<<endl;

  return 0;
}
```

41 DLX

https://www.luogu.com.cn/problem/UVA1309

填 16×16 的数独。

```
const int R=16*16*16+5, C=16*16*4+5;
const int N=4*R; // 1 的个数

int n, m;
int w[R][C];

struct DLX{
    int l[N], r[N], u[N], d[N]; // 四个指针
    int s[N]; // s[j] 表示第 j 列 1 的数量
    int row[N], col[N]; // 相应标号所在行、列
    int idx; // 点的标号

    int ans[N], top;
    char res[20][20];

    void init(){ // 一开始有 m+1 个点
        memset(row, 0, sizeof row);
        memset(col, 0, sizeof col);
        memset(s, 0, sizeof s);
        top=0;
        rep(i,0,m) l[i]=i-1, r[i]=i+1, u[i]=d[i]=i;
        l[0]=m, r[m]=0;
        idx=m+1;
    }

    void add(int &hh, int &tt, int x, int y){
        row[idx]=x, col[idx]=y, s[y]++;
        u[idx]=y, d[idx]=d[y], u[d[y]]=idx, d[y]=idx;
        r[hh]=l[tt]=idx, r[idx]=tt, l[idx]=hh;
        tt=idx++;
    }

    void build(){
        rep(i,1,n){
            int hh=idx, tt=idx;
            rep(j,1,m){
                int x=w[i][j];
                if(x) add(hh, tt, i, j);
            }
        }
    }

    void remove(int p){
        r[l[p]]=r[p], l[r[p]]=l[p];
        for(int i=d[p]; i!=p; i=d[i]) for(int j=r[i]; j!=i; j=r[j])
            s[col[j]]--, u[d[j]]=u[j], d[u[j]]=d[j];
    }

    void resume(int p){
        for(int i=u[p]; i!=p; i=u[i]) for(int j=l[i]; j!=i; j=l[j])
            u[d[j]]=j, d[u[j]]=j, s[col[j]]++;
        r[l[p]]=p, l[r[p]]=p;
    }

    bool dfs(){
        if(!r[0]) return true;
        int p=r[0];
        for(int i=r[0]; i; i=r[i]) if(s[i]<s[p]) p=i; // 找到点最少的列
        remove(p);
        for(int i=d[p]; i!=p; i=d[i]){
            ans[++top]=row[i];
            for(int j=r[i]; j!=i; j=r[j]) remove(col[j]);
            if(dfs()) return true;
            for(int j=l[i]; j!=i; j=l[j]) resume(col[j]);
            top--;
        }
        resume(p);
        return false;
    }

    void solve(){
        if(dfs()){
            rep(i,1,top){
                int t=ans[i];
                int r=(t-1)/16/16+1, c=(t-1)/16%16+1, v=(t-1)%16+1;
                res[r][c]=v-1+'A';
            }

            rep(i,1,16){
              rep(j,1,16) cout<<res[i][j];
              cout<<endl;
            }
        }
        // else puts("No Solution!");
    }
}dlx;

int get(int x, int y){
    int r=(x-1)/4+1, c=(y-1)/4+1;
    return (r-1)*4+c;
}

string s[20];

int main(){
    n=16*16*16, m=4*16*16;
    bool fl=false;
    while(cin>>s[1]){
      s[1]=" "+s[1];
      rep(i,2,16) cin>>s[i], s[i]=" "+s[i];
      if(!fl) fl=true;
      else puts("");

        vector<pii> buf;

        rep(i,1,16) rep(j,1,16){
            char t=s[i][j];
            int val=(t=='-'? 0: t-'A'+1);
            rep(k,1,16){
                if(val && val!=k) continue;
                int r=(i-1)*16*16+(j-1)*16+k;
                int c1, c2, c3, c4;
                c1=(i-1)*16+j;
                c2=16*16+(i-1)*16+k;
                c3=2*16*16+(j-1)*16+k;
                c4=3*16*16+(get(i, j)-1)*16+k;

                w[r][c1]=w[r][c2]=w[r][c3]=w[r][c4]=1;
                buf.push_back({r, c1});
                buf.push_back({r, c2});
                buf.push_back({r, c3});
                buf.push_back({r, c4});
            }
        }

        dlx.init();
        dlx.build();
        dlx.solve();

        for(auto [x, y]: buf) w[x][y]=0;
    }
    return 0;
}
```

42 Slope trick

例题：

给定一个整数序列 $a_1,a_2,\ldots,a_n$，求出一个递增序列 $b_1<b_2<\cdots<b_n$，使得序列 a_i 和 b_i 的各项之差的绝对值之和 $|a_1-b_1|+|a_2-b_2|+\ldots+|a_n-b_n|$ 最小。

```
#define int ll

struct SlopeTrick{
    priority_queue<int> L;
    priority_queue<int, vector<int>, greater<int>> R;
    const int INF=1e18;
    int addL=0, addR=0, min_f=0;

    SlopeTrick(){
        L.push(-INF);
        R.push(INF);
    }

  // 加上高 x 轴的距离
    void add_const(int c){
        min_f+=c;
    }

  // 加上 max(0, x-a)
    void add_x_minus_a(int a){
        int l=L.top()+addL;
        add_const(max(l-a, 0LL));
        L.push(a-addL);
        R.push(L.top()+addL-addR);
        L.pop();
    }

  // 加上 max(0, a-x)
    void add_a_minus_x(int a){
        int r=R.top()+addR;
        add_const(max(a-r, 0LL));
        R.push(a-addR);
        L.push(R.top()+addR-addL);
        R.pop();
    }

  // 加上 |x-a|
    void add_abs(int a){
        add_x_minus_a(a);
        add_a_minus_x(a);
    }

  // 前缀 min
    void min_left(){
        while(!R.empty()) R.pop();
    }

  // 后缀 min
    void min_right(){
        while(!L.empty()) L.pop();
    }

  // f(x) -> f(x-a)
    void shift(int a){
        addL+=a;
        addR+=a;
    }
};

void solve(){
  SlopeTrick slp=SlopeTrick();
  int n; cin>>n;
  vector<int> w(n+1);
  rep(i, 1, n) read(w[i]);
  rep(i, 1, n) w[i]-=i;

  slp.add_abs(w[1]);
  vector<int> res(n+1);
  res[1]=slp.L.top();
  rep(i, 2, n){
    slp.min_left();
    slp.add_abs(w[i]);
    res[i]=slp.L.top();
  }
  cout<<slp.min_f<<endl;
  dwn(i, n-1, 1) res[i]=min(res[i], res[i+1]);
  rep(i, 1, n) cout<<res[i]+i<<" ";
}

signed main(){
  solve();
  return 0;
}
```

43 WQS 二分

例题：https://www.luogu.com.cn/problem/P1484

cyrcyr 今天在种树，他在一条直线上挖了 n 个坑。这 n 个坑都可以种树，但为了保证每一棵树都有充足的养料，cyrcyr 不会在相邻的两个坑中种树。而且由于 cyrcyr 的树种不够，他至多会种 k 棵树。假设 cyrcyr 有某种神能力，能预知自己在某个坑种树的获利会是多少（可能为负），请你帮助他计算出他的最大获利。

```
const int N=3e5+5, INF=1e15;
```

```
int n, m;
int w[N], tmp[N];
int bias;

pii Max(pii &a, pii &b){
  if(a.x!=b.x) return max(a, b);
  return pii(a.x, min(a.y, b.y));
}

bool ok(int k){
  rep(i, 1, n) tmp[i]=w[i]-k;
  int sz=n;
  vector<vector<pii>> f(sz+1, vector<pii>(2));
  f[0][0]={0, 0};
  f[0][1]={-INF, 0};
  rep(i, 1, sz){
    f[i][1]={f[i-1][0].x+tmp[i], f[i-1][0].y+1};
    f[i][0]=Max(f[i-1][0], f[i-1][1]);
  }
  pii res=Max(f[sz][0], f[sz][1]);
  bias=res.x;
  return res.y<=m;
}

void solve(){
  cin>>n>>m;
  rep(i, 1, n) read(w[i]);
  if(ok(0)){
    cout<<bias<<endl;
    return;
  }

  int l=-1e9-10, r=1e9+10;
  while(l<r){
    int mid=l+r>>1;
    if(ok(mid)) r=mid;
    else l=mid+1;
  }
  ok(l);
  int res=bias+l*m;
  cout<<res<<endl;
}

signed main(){
  solve();
  return 0;
}
```

44 分治小技巧

44.1 例题 1

https://codeforces.com/contest/1175/problem/F

题意

给你一个序列 a，求有多少个子段满足该子段是一个排列。

思路

排列有着仅出现一次的性质，其实可以哈希搞。但这里使用分治搞（专门记录一下）。

做法

一个合法方案等价于同时满足：
- $[l,r]$ 的 max 为 $r-l+1$。
- $[l,r]$ 内的 a_i 值各不相同。

因为有最值的约束，我们可以在分治的时候以任意一个 max 值所在位置作为分治的分割点 mid。

分治将当前区间分成一长一短两段：

- 枚举短的半段（设为左半段，短的半段为右半段同理），因为长度已经确定，故右半段也确定了。
- 左半必须不重复且值 $<\max$，且后继一定不在右半段对应区间里。
- 而对于右半段，也一定满足值 $<\max$，然后里面所有元素的后继位置的 min 值一定都在右半段的右侧。

那要维护的信息就是区间 max（直接维护位置，因为还要分裂区间用），后继位置的 min，前驱位置的 max。

代码

```
const int N=3e5+5, M=20;

int n, w[N];
int pre[N], suf[N];
int b[N];

void init(){
  vector<int> tmp(N, 0);
  rep(i, 1, n){
    pre[i]=tmp[w[i]];
    tmp[w[i]]=i;
  }
  fill(all(tmp), n+1);
  dwn(i, n, 1){
    suf[i]=tmp[w[i]];
    tmp[w[i]]=i;
  }
}

struct ST{
  int typ;
  int f[N][M];
  int w[N];

  int cmp(int x, int y){
    if(!typ) return w[x]<w[y]? x: y;
    return w[x]<w[y]? y: x;
  }

  void build(int n, int *_w, int _typ){
    typ=_typ;
    rep(i, 1, n) w[i]=_w[i];
    rep(j, 0, M-1){
      rep(i, 1, n-(1<<j)+1){
        if(!j) f[i][j]=i;
        else f[i][j]=cmp(f[i][j-1], f[i+(1<<j-1)][j-1]);
      }
    }
  }

  int qpos(int l, int r){
    int k=__lg(r-l+1);
    return cmp(f[l][k], f[r-(1<<k)+1][k]);
  }

  int qval(int l, int r){
    return w[qpos(l, r)];
  }
}st1, st2, st3;

int res;

void dfs(int l, int r){
  if(l>r) return;
  if(l==r){
    if(w[l]==1) ++res;
    return;
  }
  int mid=st1.qpos(l, r);
  int ma=w[mid];
  if(mid-l<r-mid){
    int tmp=n+1;
    dwn(lef, mid, l){
      int rig=lef+ma-1, val=w[lef];
      if(rig<mid || val>ma) break;
      if(++b[val]>1) break;
      if(suf[lef]<=rig){
        tmp=min(suf[lef], tmp);
        continue;
      }
      if(tmp<=rig) continue;
      if(mid<rig){
        if(st1.qval(mid+1, rig)<ma && st3.qval(mid+1, rig)>rig){
          ++res;
        }
      }
      else{
        ++res;
      }
    }
    rep(i, l, mid) b[w[i]]=0;
  }
  else{
    int tmp=0;
    rep(rig, mid, r){
      int lef=rig-ma+1, val=w[rig];
      if(lef>mid || val>ma) break;
      if(++b[val]>1) break;
      if(pre[rig]>=lef){
        tmp=max(pre[rig], tmp);
        continue;
      }
      if(tmp>=lef) continue;
      if(lef<mid){
        if(st1.qval(lef, mid-1)<ma && st2.qval(lef, mid-1)<lef){
          ++res;
        }
      }
      else{
        ++res;
      }
    }
    rep(i, mid, r) b[w[i]]=0;
  }
  dfs(l, mid-1), dfs(mid+1, r);
}

void solve(){
  cin>>n;
  rep(i, 1, n) read(w[i]);
  init();
  st1.build(n, w, 1);
  st2.build(n, pre, 1);
  st3.build(n, suf, 0);
  dfs(1, n);
  cout<<res<<"\n";
}
```

44.2 例题 2

https://codeforces.com/contest/1442/problem/D

题意

给你 n 个单调不减的序列以及最多 k 次操作。

每次操作你可以从序列开头取走一个元素并将这个元素值加入答案中。

求最大的答案。

思路

核心结论是：对于最优解选取的序列中，一定可以满足至多一个序列不选完。

> 证明：
>
> 设现在有两个没选完的序列 A, B，A 最后一个选取的元素为 a，B 最后一个选取的元素为 b。不妨设 $a>b$。
>
> 那这个选取方式一定不优于：不选 b，而是去 A 中多取一个元素。

做法

所以我们枚举这个没选满的序列 i，通过前后缀 $([1,i-1]\cup[i+1,n])$ 的背包合并结果和当前序列来更新答案。

直接合并复杂度是 $O(nk^2)$ 的，考虑使用以下分治做法：

- 记录当前层的背包状态，后把左半段物品填入背包。
- 先走右子树，走完右子树之后将背包状态重设为之前记录的背包状态，并将背包合并右半段的信息。
- 可以发现树上每一个点对应的背包都是当前区间被抠掉的形态，所以求每个叶子答案即可。

代码

```
#define int long long
const int N=3030;

int n, K;
vector<int> w[N];
int s[N];
int f[N];
int sav[15][N];
int res;

void insert(int idx){
  int sz=w[idx].size()-1;
  dwn(i, K, sz) f[i]=max(f[i], f[i-sz]+s[idx]);
}

void Save(int dep){
  rep(i, 0, K) sav[dep][i]=f[i];
}

void Load(int dep){
  rep(i, 0, K) f[i]=sav[dep][i];
}

void dfs(int l, int r, int dep){
  if(l==r){
    int sz=w[l].size()-1;
    rep(i, 0, sz) if(K-i>=0) res=max(res, w[l][i]+f[K-i]);
    return;
  }
  int mid=l+r>>1;
  Save(dep);
  rep(i, l, mid) insert(i);
  dfs(mid+1, r, dep+1);
  Load(dep);
  rep(i, mid+1, r) insert(i);
  dfs(l, mid, dep+1);
}

void solve(){
  cin>>n>>K;
  rep(i, 1, n){
    int cnt; read(cnt);
    w[i].resize(cnt+1, 0);
    rep(j, 1, cnt) read(w[i][j]), w[i][j]+=w[i][j-1];
    s[i]=w[i][cnt];
  }
  dfs(1, n, 0);
  cout<<res<<"\n";
}
```

45 一些手写库

45.1 分数类

引用：https://github.com/SSerxhs/SSerxhs-ICPC-library/blob/main/%E6%A8%A1%E6%9D%BF/%E6%9D%82%E9%A1%B9/%E5%88%86%E6%95%B0%E7%B1%BB.cpp

```
typedef long long ll;
struct frac
{
  ll x,y;
  frac(const ll &_t):x(_t),y(1) {}
  frac(const ll &_x,const ll &_y):x(_x),y(_y)
  {
    ll d=gcd(x,y);
    x/=d; y/=d;
    if (x<0||x==0&&y<0) x=-x,y=-y;
    assert(y);
  }
  frac operator+(const frac &o) const
  {
    ll d=lcm(y,o.y);
    return {d/y*x+d/o.y*o.x,d};
  }
  frac operator-(const frac &o) const
  {
    ll d=lcm(y,o.y);
    return {d/y*x-d/o.y*o.x,d};
  }
  frac operator*(const frac &o) const
  {
    return {x*o.x,y*o.y};
  }
  frac operator/(const frac &o) const
  {
    return {x*o.y,y*o.x};
  }
  frac &operator+=(const frac &o) { return *this=*this+o; }
  frac &operator-=(const frac &o) { return *this=*this-o; }
  frac &operator*=(const frac &o) { return *this=*this*o; }
  frac &operator/=(const frac &o) { return *this=*this/o; }
  bool operator<(const frac &o) const { return x*o.y<y *o.x; }
  bool operator==(const frac &o) const { return x*o.y==y*o.x; }
  bool operator>(const frac &o) const { return x*o.y>y*o.x; }
  bool operator!=(const frac &o) const { return x*o.y!=y*o.x; }
};
ostream &operator<<(ostream &cout,const frac &o)
{
  return cout<<o.x<<"/"<<o.y;
}
```

45.2 变长 bitset

引用：https://github.com/SSerxhs/SSerxhs-ICPC-library/blob/main/%E6%A8%A1%E6%9D%BF/Bitset.cpp

```
struct Bitset
{
  typedef unsigned int ui;
  typedef unsigned long long ll;
#define all(x) (x).begin(),(x).end()
  const static ll B=-1llu;
  vector<ll> a;
  int n;
  Bitset() {}
  Bitset(int _n) :n(_n),a(_n+63>>6) {}
  bool operator[](int x) const { assert(x>=0&&x<n); return
a[x>>6]>>(x&63)&1; }
  void set(int x,bool y) { assert(x>=0&&x<n);
a[x>>6]=(a[x>>6]&(B^1<<(x&63)))|((ll)y<<(x&63)); }
  void set(int x) { assert(x>=0&&x<n); a[x>>6]|=1llu<<(x&63); }
  void set() { memset(a.data(),0xff,a.size()*sizeof a[0]);
a.back()&=(1llu<<1+(n-1&63))-1; }
  void reset(int x) { a[x>>6]&=~(1llu<<(x&63)); }
  void reset() { memset(a.data(),0,a.size()*sizeof a[0]); }
  int count() const
  {
    int r=0;
    for (ll x:a) r+=__builtin_popcountll(x);
    return r;
  }
  Bitset &operator|=(const Bitset &o)
  {
    assert(n==o.n);
    for (int i=0; i<a.size(); i++) a[i]|=o.a[i];
    return *this;
  }
  Bitset operator|(Bitset o) { o|=*this; return o; }
  Bitset &operator&=(const Bitset &o)
  {
    assert(n==o.n);
    for (int i=0; i<a.size(); i++) a[i]&=o.a[i];
    return *this;
  }
  Bitset operator&(Bitset o) { o&=*this; return o; }
  Bitset &operator^=(const Bitset &o)
  {
    assert(n==o.n);
    for (int i=0; i<a.size(); i++) a[i]^=o.a[i];
    return *this;
  }
  Bitset operator^(Bitset o) { o^=*this; return o; }
  Bitset operator~() const
  {
    auto r=*this;
    for (ll &x:r.a) x=~x;
    return r;
  }
  Bitset &operator<<=(int x)
  {
    if (x>=n)
```

```
    {
      fill(all(a),0);
      return *this;
    }
    assert(x>=0);
    int y=x>>6;
    x&=63;
    for (int i=(int)a.size()-1; i>y; i--) a[i]=a[i-y]<<x|a[i-
y-1]>>64-x;
    a[y]=a[0]<<x;
    memset(a.data(),0,y*sizeof a[0]);
    // fill_n(a.begin(),y,0);
    a.back()&=(1llu<<1+(n-1&63))-1;
    return *this;
  }
  Bitset operator<<(int x)
  {
    auto r=*this;
    r<<=x;
    return r;
  }
  Bitset &operator>>=(int x)
  {
    if (x>=n)
    {
      fill(all(a),0);
      return *this;
    }
    assert(x>=0);
    int y=x>>6,R=(int)a.size()-y-1;
    x&=63;
    for (int i=0; i<R; i++) a[i]=a[i+y]>>x|a[i+y+1]<<64-x;
    a[R]=a.back()>>x;
    memset(a.data()+R+1,y*sizeof a[0]);
    // fill(R+1+all(a),0);
    return *this;
  }
  Bitset operator>>(int x)
  {
    auto r=*this;
    r>>=x;
    return r;
  }
  void range_set(int l,int r)//[l,r) to 1
  {
    if (l>>6==r>>6)
    {
      a[l>>6]|=(1llu<<r-l)-1<<(l&63);
      return;
    }
    if (l&63)
    {
      a[l>>6]|=~((1llu<<(l&63))-1);//[l&63,64)
      l=(l>>6)+1<<6;
    }
    if (r&63)
    {
      a[r>>6]|=(1llu<<(r&63))-1;
      r=(r>>6)-1<<6;
    }
    memset(a.data()+(l>>6),0xff,(r-l>>6)*sizeof a[0]);
  }
  void range_reset(int l,int r)//[l,r) to 0
  {
    if (l>>6==r>>6)
    {
      a[l>>6]&=~((1llu<<r-l)-1<<(l&63));
      return;
    }
    if (l&63)
    {
      a[l>>6]&=(1llu<<(l&63))-1;//[l&63,64)
      l=(l>>6)+1<<6;
    }
    if (r&63)
    {
      a[r>>6]&=~((1llu<<(r&63))-1);
      r=(r>>6)-1<<6;
    }
    memset(a.data()+(l>>6),0,(r-l>>6)*sizeof a[0]);
  }
  void range_set(int l,int r,bool x)//[l,r)
  {
    if (x) range_set(l,r);
    else range_reset(l,r);
  }
};
```

45.3 frac_floor and fracceil

参考：https://github.com/kk2a/library/blob/main/math/frac_floor.hpp

```
// floor(x) = ceil(x) - 1 (for all x not in Z) ...(1)
// floor(x) = -ceil(-x)   (for all x)          ...(2)

// return floor(a / b)
template <typename T, typename U> constexpr T fracfloor(T a, U b) {
    assert(b != 0);
    if (a % b == 0) return a / b;
    if (a >= 0) return a / b;
    // floor(x) = -ceil(-x)      by (2)
    //          = -floor(-x) - 1 by (1)
    return -((-a) / b) - 1;
}

// return ceil(a / b)
template <typename T, typename U> constexpr T fracceil(T a, U b) {
    assert(b != 0);
    if (a % b == 0) return a / b;
    if (a >= 0) return a / b + 1;

    // ceil(x) = -floor(-x)      by (2)
    return -((-a) / b);
}

}
```