Standard Code Library

Magic Vegtable(llsnb!)

Shanghai University

October 30, 2019

Contents

数据范围表

$unsigned\ int : 0 \sim 4294967295$

$int : -2147483648 \sim 2147483647 \quad (1 << 30) + ((1 << 30) - 1) \quad 2.1 \times 10^9$

$ull : 0 \sim 18446744073709551615 \quad 1.8 \times 10^{19}$

$ll_{max} : 9223372036854775807 \quad (1LL << 62) + ((1LL << 62) - 1) \quad 9.2 \times 10^{18}$

$0x3f3f3f3f : 1061109567 \quad 1 \times 10^9$

$0x3f3f3f3f3f3f3f3f : 4557430888798830399 \quad 4.5 \times 10^{18}$

宏定义

- debug 宏

```
#define compute
#ifdef compute
#define dbg(x...) do{cout << "\033[32;1m" << #x << "->" ; err(x);} while(0)
void err(){cout << "\033[39;0m" << endl;}
template<template<typename...> class T,typename t,typename... A>
void err(T<t> a,A... x){for (auto v:a) cout << v << ' '; err(x...);}
template<typename T,typename... A>
void err(T a,A... x){cout << a << ' '; err(x...);}
#else
#define dbg(...)
#endif
```

- 更多配色：33-黄色，34-蓝色，31-橙色

VIM 配置

- 比赛用

```
set nu
set hlsearch
set tabstop=4
syntax on
set shiftwidth=4
set cindent
set mouse=a
set cursorline
set cursorcolumn
```

- compute 用

```
set nocompatible " be iMproved, required
filetype off " required
" set the runtime path to include Vundle and initialize
set rtp+=~/.vim/bundle/Vundle.vim
call vundle#begin()
" alternatively, pass a path where Vundle should install plugins
"call vundle#begin('~/some/path/here')
" let Vundle manage Vundle, required
```

```
Plugin 'VundleVim/Vundle.vim'
Plugin 'luochen1990/rainbow'
call vundle#end() " required
filetype plugin indent on " required
" To ignore plugin indent changes, instead use:
"filetype plugin on
"
" Brief help
" :PluginList - lists configured plugins
" :PluginInstall - installs plugins; append `!` to update or just :PluginUpdate
" :PluginSearch foo - searches for foo; append `!` to refresh local cache
" :PluginClean - confirms removal of unused plugins; append `!` to auto-approve
↪ removal
"
" see :h vundle for more details or wiki for FAQ
" Put your non-Plugin stuff after this line
set nu
set tabstop=4
syntax on
set shiftwidth=4
set cin
set mouse=a
set ruler
set cursorline
set cursorcolumn
set cindent
set autoindent
let g:rainbow_active=1
" 主题 solarized
Bundle 'altercation/vim-colors-solarized'
"let g:solarized_termcolors=256
let g:solarized_termtrans=1
let g:solarized_contrast="normal"
let g:solarized_visibility="normal"
" 主题 molokai
Bundle 'tomasr/molokai'
let g:molokai_original = 1
set background=dark
set t_Co=256
"colorscheme solarized
colorscheme molokai
"colorscheme phd
"kakko comp
inoremap ( ()<Esc>i
inoremap [ []<Esc>i
inoremap { {}<Esc>i
inoremap ' ''<Esc>i
inoremap " ""<Esc>i
inoremap ) <c-r>=ClosePair(')')<CR>
inoremap } <c-r>=ClosePair('}')<CR>
inoremap ] <c-r>=ClosePair(']')<CR>
function ClosePair(char)
```

```
    if getline('.')[col('.')-1]==a:char
        return "\<Right>"
    else
        return a:char
    endif
endfunction
set completeopt=longest,menu
```

数学

欧拉筛

```
const int maxn = 1e7 + 10;
int prime[maxn] = {0}, phi[maxn] = {0}, tot;

void euler()
{
    phi[1] = 1;
    for (int i = 2; i < maxn; i++)
    {
        if (!phi[i])
        {
            prime[tot++] = i;
            phi[i] = i - 1;
        }
        for (int j = 0; j < tot && i * prime[j] < maxn; j++)
        {
            if (i % prime[j] == 0)
            {
                phi[i * prime[j]] = phi[i] * prime[j];
                break;
            }
            phi[i * prime[j]] = phi[i] * phi[prime[j]];
        }
    }
}
```

筛莫比乌斯函数

```
const int maxn = 1e7 + 10;
int prime[maxn], tot = 0, mu[maxn];
bool check[maxn];

void mobius()
{
    mu[1] = 1;
    for (int i = 2; i < maxn; i++)
    {
        if (!check[i])
        {
            prime[tot++] = i;
            mu[i] = -1;
        }
        for (int j = 0; j < tot && i * prime[j] < maxn; j++)
        {
            check[i * prime[j]] = true;
            if (i % prime[j] == 0)
            {
                mu[i * prime[j]] = 0;
                break;
            }
```

```
                mu[i * prime[j]] = -mu[i];
            }
        }
}
```

莫比乌斯反演，整除分块

$$F(n) = \Sigma_{d|n} f(d) \Rightarrow f(n) = \Sigma_{d|n} \mu(d) F(\frac{d}{n})$$

$$F(n) = \Sigma_{n|d} f(d) \Rightarrow f(n) = \Sigma_{n|d} \mu(\frac{d}{n}) F(d)$$

```
ll prime[maxn], tot = 0, mu[maxn], sum[maxn];
bool check[maxn];

void getmu()
{
    mu[1] = 1;
    for (int i = 2; i < maxn; i++)
    {
        if (!check[i])
        {
            prime[tot++] = i;
            mu[i] = -1;
        }
        for (int j = 0; j < tot && i * prime[j] < maxn; j++)
        {
            check[i * prime[j]] = true;
            if (i % prime[j] == 0)
            {
                mu[i * prime[j]] = 0;
                break;
            }
            mu[i * prime[j]] = -mu[i];
        }
    }
    for (int i = 1; i < maxn; i++) //前缀和
        sum[i] = sum[i - 1] + mu[i];
}

ll cal(int a, int b) //整除分块 i:1->a  j:1->b  gcd(i,j)=1 对数
{
    if (a > b)
        swap(a, b);
    ll l = 1, r, ans = 0;
    while (l <= a)
    {
        r = min(a / (a / l), b / (b / l));
        ans += (sum[r] - sum[l - 1]) * (a / l) * (b / l);
        l = r + 1;
    }
    return ans;
}
```

公式

一些数论公式

- 当 $x \geq \phi(p)$ 时有 $a^x \equiv a^{x \ mod \ \phi(p)+\phi(p)} \pmod p$
- $\mu^2(n) = \sum_{d^2|n} \mu(d)$
- $\sum_{d|n} \varphi(d) = n$
- $\sum_{d|n} 2^{\omega(d)} = \sigma_0(n^2)$，其中 ω 是不同素因子个数
- $\sum_{d|n} \mu^2(d) = 2^{\omega(d)}$

一些数论函数求和的例子

- $\sum_{i=1}^n i[gcd(i,n)=1] = \frac{n\varphi(n)+[n=1]}{2}$
- $\sum_{i=1}^n \sum_{j=1}^m [gcd(i,j)=x] = \sum_d \mu(d) \lfloor \frac{n}{dx} \rfloor \lfloor \frac{m}{dx} \rfloor$
- $\sum_{i=1}^n \sum_{j=1}^m gcd(i,j) = \sum_{i=1}^n \sum_{j=1}^m \sum_{d|gcd(i,j)} \varphi(d) = \sum_d \varphi(d) \lfloor \frac{n}{d} \rfloor \lfloor \frac{m}{d} \rfloor$
- $S(n) = \sum_{i=1}^n \mu(i) = 1 - \sum_{i=1}^n \sum_{d|i,d<i} \mu(d) \overset{t=\frac{i}{d}}{=} 1 - \sum_{t=2}^n S(\lfloor \frac{n}{t} \rfloor)$
  - 利用 $[n=1] = \sum_{d|n} \mu(d)$
- $S(n) = \sum_{i=1}^n \varphi(i) = \sum_{i=1}^n i - \sum_{i=1}^n \sum_{d|i,d<i} \varphi(i) \overset{t=\frac{i}{d}}{=} \frac{i(i+1)}{2} - \sum_{t=2}^n S(\frac{n}{t})$
  - 利用 $n = \sum_{d|n} \varphi(d)$
- $\sum_{i=1}^n \mu^2(i) = \sum_{i=1}^n \sum_{d^2|n} \mu(d) = \sum_{d=1}^{\lfloor \sqrt{n} \rfloor} \mu(d) \lfloor \frac{n}{d^2} \rfloor$
- $\sum_{i=1}^n \sum_{j=1}^n gcd^2(i,j) = \sum_d d^2 \sum_t \mu(t) \lfloor \frac{n}{dt} \rfloor^2$
  $\overset{x=dt}{=} \sum_x \lfloor \frac{n}{x} \rfloor^2 \sum_{d|x} d^2 \mu(\frac{x}{d})$
- $\sum_{i=1}^n \varphi(i) = \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n [i \perp j] - 1 = \frac{1}{2} \sum_{i=1}^n \mu(i) \cdot \lfloor \frac{n}{i} \rfloor^2 - 1$

斐波那契数列性质

- $F_{a+b} = F_{a-1} \cdot F_b + F_a \cdot F_{b+1}$
- $F_1 + F_3 + \dots + F_{2n-1} = F_{2n}, F_2 + F_4 + \dots + F_{2n} = F_{2n+1} - 1$
- $\sum_{i=1}^n F_i = F_{n+2} - 1$
- $\sum_{i=1}^n F_i^2 = F_n \cdot F_{n+1}$
- $F_n^2 = (-1)^{n-1} + F_{n-1} \cdot F_{n+1}$
- $gcd(F_a, F_b) = F_{gcd(a,b)}$
- 模 n 周期（皮萨诺周期）
  - $\pi(p^k) = p^{k-1}\pi(p)$
  - $\pi(nm) = lcm(\pi(n), \pi(m)), \forall n \perp m$
  - $\pi(2) = 3, \pi(5) = 20$
  - $\forall p \equiv \pm 1 \pmod{10}, \pi(p) | p-1$
  - $\forall p \equiv \pm 2 \pmod 5, \pi(p) | 2p+2$

常见生成函数

- $(1+ax)^n = \sum_{k=0}^n \binom{n}{k} a^k x^k$
- $\dfrac{1-x^{r+1}}{1-x} = \sum_{k=0}^n x^k$
- $\dfrac{1}{1-ax} = \sum_{k=0}^\infty a^k x^k$
- $\dfrac{1}{(1-x)^2} = \sum_{k=0}^\infty (k+1) x^k$

- $\dfrac{1}{(1-x)^n} = \sum_{k=0}^{\infty} \binom{n+k-1}{k} x^k$
- $e^x = \sum_{k=0}^{\infty} \dfrac{x^k}{k!}$
- $\ln(1+x) = \sum_{k=0}^{\infty} \dfrac{(-1)^{k+1}}{k} x^k$

佩尔方程

若一个丢番图方程具有以下的形式：$x^2 - ny^2 = 1$。且 n 为正整数，则称此二元二次不定方程为**佩尔方程**。

若 n 是完全平方数，则这个方程式只有平凡解 $(\pm 1, 0)$（实际上对任意的 n，$(\pm 1, 0)$ 都是解）。对于其余情况，拉格朗日证明了佩尔方程总有非平凡解。而这些解可由 $\sqrt{n}$ 的连分数求出。

$$x = [a_0; a_1, a_2, a_3] = x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{\ddots}}}}$$

设 $\frac{p_i}{q_i}$ 是 $\sqrt{n}$ 的连分数表示：$[a_0; a_1, a_2, a_3, ...]$ 的渐近分数列，由连分数理论知存在 i 使得 (p_i, q_i) 为佩尔方程的解。取其中最小的 i，将对应的 (p_i, q_i) 称为佩尔方程的基本解，或最小解，记作 (x_1, y_1)，则所有的解 (x_i, y_i) 可表示成如下形式：$x_i + y_i\sqrt{n} = (x_1 + y_1\sqrt{n})^i$。或者由以下的递回关系式得到：

$x_{i+1} = x_1 x_i + n y_1 y_i$, $y_{i+1} = x_1 y_i + y_1 x_i$。

但是：佩尔方程千万不要去推（虽然推起来很有趣，但结果不一定好看，会是两个式子）。记住佩尔方程结果的形式通常是 $a_n = k a_{n-1} - a_{n-2}$（$a_{n-2}$ 前的系数通常是 -1）。暴力 / 凑出两个基础解之后加上一个 0，容易解出 k 并验证。

Burnside & Polya

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

注：X^g 是 g 下的不动点数量，也就是说有多少种东西用 g 作用之后可以保持不变。

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

注：用 m 种颜色染色，然后对于某一种置换 g，有 $c(g)$ 个置换环，为了保证置换后颜色仍然相同，每个置换环必须染成同色。

皮克定理

$2S = 2a + b - 2$

- S 多边形面积
- a 多边形内部点数
- b 多边形边上点数

莫比乌斯反演

- $g(n) = \sum_{d|n} f(d) \Leftrightarrow f(n) = \sum_{d|n} \mu(d) g(\frac{n}{d})$
- $f(n) = \sum_{n|d} g(d) \Leftrightarrow g(n) = \sum_{n|d} \mu(\frac{d}{n}) f(d)$

低阶等幂求和

- $\sum_{i=1}^{n} i^1 = \frac{n(n+1)}{2} = \frac{1}{2}n^2 + \frac{1}{2}n$
- $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} = \frac{1}{3}n^3 + \frac{1}{2}n^2 + \frac{1}{6}n$
- $\sum_{i=1}^{n} i^3 = \left[\frac{n(n+1)}{2}\right]^2 = \frac{1}{4}n^4 + \frac{1}{2}n^3 + \frac{1}{4}n^2$
- $\sum_{i=1}^{n} i^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30} = \frac{1}{5}n^5 + \frac{1}{2}n^4 + \frac{1}{3}n^3 - \frac{1}{30}n$
- $\sum_{i=1}^{n} i^5 = \frac{n^2(n+1)^2(2n^2+2n-1)}{12} = \frac{1}{6}n^6 + \frac{1}{2}n^5 + \frac{5}{12}n^4 - \frac{1}{12}n^2$

一些组合公式

- 错排公式：$D_1 = 0, D_2 = 1, D_n = (n-1)(D_{n-1} + D_{n-2}) = n!(\frac{1}{2!} - \frac{1}{3!} + \cdots + (-1)^n \frac{1}{n!}) = \lfloor \frac{n!}{e} + 0.5 \rfloor$
- 卡塔兰数（n 对括号合法方案数，n 个结点二叉树个数，$n \times n$ 方格中对角线下方的单调路径数，凸 $n+2$ 边形的三角形划分数，n 个元素的合法出栈序列数）：$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!n!}$

连分数

- 可表示为 $\frac{p_k}{q_k} = a_1 + \frac{1}{a_2 + \frac{1}{a_3 + \frac{1}{a_4 + \ldots}}} = [a_1, a_2, a_3, a_4, \ldots]$
- 可以根据线性递得到以下式子：

$$p_k = \begin{cases} a_1 & k = 1 \\ a_1 a_2 + 1 & k = 2 \\ a_k p_{k-1} + p_{k-2} & k \geq 3 \end{cases}$$

$$q_k = \begin{cases} 1 & k = 1 \\ a_2 & k = 2 \\ a_k q_{k-1} + q_{k-2} & k \geq 3 \end{cases}$$

- 写成矩阵形式即为：

$$\begin{bmatrix} p_n & p_{n-1} \\ q_n & q_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} a_1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} a_2 & 1 \\ 1 & 0 \end{bmatrix} \ldots \begin{bmatrix} a_n & 1 \\ 1 & 0 \end{bmatrix}$$

常见狄利克雷卷积

积性函数

$$\epsilon(n) = [n = 1]$$

$$1(n) = 1$$

$$Id(n) = n$$

$$d(n) = \Sigma_{d|n} 1$$

$$\sigma(n) = \Sigma_{d|n} d$$

卷积

$$\mu * 1 = \epsilon$$

$$\phi * 1 = Id$$

$$1 * 1 = d$$

杜教筛

$O(n^{2/3})$ 求积性函数前缀和。
假设要求的积性函数为 f，前缀和为 S，能找到较为简单 (前缀和好求) 的积性函数 g，使得 $f*g=h$ 前缀和容易求出。那么有:

$$g(1)S(n)=\Sigma_{i=1}^{n}h(i)-\Sigma_{d=2}^{n}g(d)\cdot S(\lfloor n/d\rfloor)$$

其中 $g(1)$ 一般来说都是 1，h 和 g 的前缀和需要比较好算，然后就能记忆化搜索了。

```
//杜教筛欧拉函数 原理: phi * I = Id
const int maxn=1e6+10;          //通常预处理到 n^2/3 为最优
ll phi[maxn];
unordered_map<int,ll> phi2;     //hash

ll djsphi(int n)
{
    if (n < maxn)
        return phi[n];
    if (phi2[n])
        return phi2[n];
    ll ans = 1ll * n * (n + 1) / 2;
    for (int l = 2, r; l <= n; l = r + 1)
    {
        r = n / (n / l);
        ans -= djsphi(n / l) * (r - l + 1);
    }
    return phi2[n] = ans;   //记忆化
}
```

一些积性函数的表

id	$\mu(i)$	$\phi(i)$	id	$\mu(i)$	$\phi(i)$	id	$\mu(i)$	$\phi(i)$
1	1	1	16	0	8	31*	-1	30
2*	-1	1	17*	-1	16	32	0	16
3*	-1	2	18	0	6	33	1	20
4	0	2	19*	-1	18	34	1	16
5*	-1	4	20	0	8	35	1	24
6	1	2	21	1	12	36	0	12
7*	-1	6	22	1	10	37*	-1	36
8	0	4	23*	-1	22	38	1	18
9	0	6	24	0	8	39	1	24
10	1	4	25	0	20	40	0	16
11*	-1	10	26	1	12	41*	-1	40
12	0	4	27	0	18	42	-1	12
13*	-1	12	28	0	12	43*	-1	42
14	1	6	29*	-1	28	44	0	20
15	1	8	30	-1	8	45	0	24

快速幂

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

快速乘

```
ll mul(ll a, ll b, ll m)
{
    ll ret = 0;
    while (b)
    {
        if (b & 1)
        {
            ret += a;
            if (ret >= m)
                ret -= m;
        }
        a += a;
        if (a >= m)
            a -= m;
        b >>= 1;
    }
    return ret;
}
```

```
ll mul(ll a, ll b, ll m)
{
    return (a * b - ll((long double)a * b / m) * m + m) % m;
}
ll mul(ll a, ll b, ll m)
{
    ll t = a * b - ll((long double)a * b / m) * m;
    return t < 0 ? t + m : t;
}
```

扩展欧几里得算法

求解 $a \cdot x + b \cdot y = gcd(a, b)$ 的一组特解。

```
ll exgcd(ll a, ll b, ll &x, ll &y) //返回 gcd(a,b)
{
    if (b == 0)
    {
```

```
        x = 1, y = 0;
        return a;
    }
    ll d = exgcd(b, a % b, y, x);
    y -= x * (a / b);
    return d;
}
```

类欧几里得算法

$$F(a,b,c,n) = \Sigma_{i=0}^{n}\lfloor\frac{a*i+b}{c}\rfloor$$

$$G(a,b,c,n) = \Sigma_{i=0}^{n}\lfloor i*\frac{a*i+b}{c}\rfloor$$

$$H(a,b,c,n) = \Sigma_{i=0}^{n}(\lfloor\frac{a*i+b}{c}\rfloor)^2$$

通用方法：1. 当 $a \geq c$ 或者 $b \geq c$ 时，通过 $\lfloor\frac{a}{c}\rfloor = \lfloor\frac{a\%c}{c}\rfloor + \lfloor\frac{a}{c}\rfloor$ 展开化简。2. 当 $a < c$ 且 $b < c$ 时，通过枚举直线下的点，交换枚举顺序展开化简后递归求解。3. 注意边界条件：$a = 0$，$n = 0$。4. 需要自己写多项取模相加/减/乘。

```
const ll mod = 1e9 + 7, inv2 = (mod + 1) / 2, inv6 = (mod + 1) / 6;

struct node
{
    ll f, g, h;
};

node solve(ll a, ll b, ll c, ll n)
{
    node ans, tmp;
    if (a == 0)
    {
        ans.f = (n + 1) * (b / c) % mod;
        ans.g = (b / c) * n % mod * (n + 1) % mod * inv2 % mod;
        ans.h = (n + 1) * (b / c) % mod * (b / c) % mod;
        return ans;
    }
    if (a >= c || b >= c)
    {
        tmp = solve(a % c, b % c, c, n);
        ans.f = (tmp.f + (a / c) * n % mod * (n + 1) % mod * inv2 % mod + (b / c) *
         ↪ (n + 1) % mod) % mod;
        ans.g = (tmp.g + (a / c) * n % mod * (n + 1) % mod * (2 * n + 1) % mod * inv6
         ↪ % mod + (b / c) * n % mod * (n + 1) % mod * inv2 % mod) % mod;
        ans.h = ((a / c) * (a / c) % mod * n % mod * (n + 1) % mod * (2 * n + 1) %
         ↪ mod * inv6 % mod +
                 (b / c) * (b / c) % mod * (n + 1) % mod + (a / c) * (b / c) % mod *
                  ↪ n % mod * (n + 1) % mod +
                 tmp.h + 2 * (a / c) % mod * tmp.g % mod + 2 * (b / c) % mod * tmp.f
                  ↪ % mod) %
                mod;
        return ans;
```

```
    }
    ll m = (a * n + b) / c;
    tmp = solve(c, c - b - 1, a, m - 1);
    ans.f = ((n * (m % mod) % mod - tmp.f) % mod + mod) % mod;
    ans.g = ((n * (n + 1) % mod * (m % mod) % mod - tmp.f - tmp.h) % mod + mod) *
     ↪  inv2 % mod;
    ans.h = ((n * (m % mod) % mod * ((m + 1) % mod) % mod - 2 * tmp.g - 2 * tmp.f -
     ↪  ans.f) % mod + mod) % mod;
    return ans;
}
```

逆元

1. 使用费马小定理，要求模数 p 为素数。
2. 使用扩展欧几里得定理，不要求模数 p 为素数。

```
ll inv(ll a, ll p) //求 a 关于 p 的逆元
{
    ll x, y;
    ll d = exgcd(a, p, x, y);
    if (d != 1)
        return -1;
    return (x % p + p) % p;
}
```

3. 基于 $inv(a) = (p - \lfloor p/a \rfloor) * inv(p\%a)\%p$，在 $O(n)$ 时间复杂度下求出逆元表。

```
void init()
{
    inv[1] = 1;
    for (int i = 2; i < maxn; i++)
        inv[i] = (mod - mod / i) * 1LL * inv[mod % i] % mod;
}
```

组合数

$O(n)$ 时间复杂度内预处理出 n 以内的组合数。

```
void init() //inv,f,finv 都开 ll
{
    inv[1] = 1;
    for (int i = 2; i < maxn; i++)
        inv[i] = (mod - mod / i) * inv[mod % i] % mod; //inv: 逆元
    f[0] = finv[0] = 1;                                //f: 阶乘  finv: 阶乘逆元 (1/f)
    for (int i = 1; i < maxn; i++)
    {
        f[i] = f[i - 1] * i % mod;
        finv[i] = finv[i - 1] * inv[i] % mod;
    }
}

ll C(int n, int m) //C(n,m)
{
    if (m < 0 || m > n)
```

```
        return 0;
    return f[n] * finv[n - m] % mod * finv[m] % mod;
}
```

拉格朗日插值

$$L(x) = \sum_{i=0}^{n} y_i l_i(x)$$

其中

$$l_i(x) = \prod_{j=0\&j\neq i}^{n} \frac{x - x_j}{x_i - x_j}$$

1. 需要组合数中的 init。
2. arr: 插值数组 n: 项数 (从 0 开始一共 n+1 项) x: 需要求的值

```
ll lagrange(ll *arr, ll n, ll x)
{
    if (x <= n)
        return arr[x];
    ll ans = 0, sgn = n & 1 ? -1 : 1;
    for (int j = 0; j <= n; j++, sgn *= -1)
        ans = (ans + mod + sgn * f[x] % mod * finv[x - n - 1] % mod * inv[x - j] %
         ↪ mod * finv[j] % mod * finv[n - j] % mod * arr[j] % mod) % mod;
    return ans;
}
```

线性递推

BM 模板

要求: 1. 线性递推 2. 所有数都有逆元 3. k 阶线性递推需要 2k 项

```
#include<bits/stdc++.h>
using namespace std;
#define rep(i,a,n) for (int i=a;i<n;i++)
#define per(i,a,n) for (int i=n-1;i>=a;i--)
#define pb push_back
#define mp make_pair
#define all(x) (x).begin(),(x).end()
#define fi first
#define se second
#define SZ(x) ((int)(x).size())
typedef vector<int> VI;
typedef long long ll;
typedef pair<int,int> PII;
const ll mod=1000000007;        //修改成题目要求的模数
ll powmod(ll a,ll b) {ll res=1;a%=mod; assert(b>=0);
 ↪ for(;b;b>>=1){if(b&1)res=res*a%mod;a=a*a%mod;}return res;}
// head
int _,n;
namespace linear_seq {
    const int N=10010;
```

```
ll res[N],base[N],_c[N],_md[N];

vector<int> Md;
void mul(ll *a,ll *b,int k) {
    rep(i,0,k+k) _c[i]=0;
    rep(i,0,k) if (a[i]) rep(j,0,k) _c[i+j]=(_c[i+j]+a[i]*b[j])%mod;
    for (int i=k+k-1;i>=k;i--) if (_c[i])
        rep(j,0,SZ(Md)) _c[i-k+Md[j]]=(_c[i-k+Md[j]]-_c[i]*_md[Md[j]])%mod;
    rep(i,0,k) a[i]=_c[i];
}
int solve(ll n,VI a,VI b) { // a 系数 b 初值 b[n+1]=a[0]*b[n]+...
    //        printf("%d\n",SZ(b));
    ll ans=0,pnt=0;
    int k=SZ(a);
    assert(SZ(a)==SZ(b));
    rep(i,0,k) _md[k-1-i]=-a[i];_md[k]=1;
    Md.clear();
    rep(i,0,k) if (_md[i]!=0) Md.push_back(i);
    rep(i,0,k) res[i]=base[i]=0;
    res[0]=1;
    while ((1ll<<pnt)<=n) pnt++;
    for (int p=pnt;p>=0;p--) {
        mul(res,res,k);
        if ((n>>p)&1) {
            for (int i=k-1;i>=0;i--) res[i+1]=res[i];res[0]=0;
            rep(j,0,SZ(Md)) res[Md[j]]=(res[Md[j]]-res[k]*_md[Md[j]])%mod;
        }
    }
    rep(i,0,k) ans=(ans+res[i]*b[i])%mod;
    if (ans<0) ans+=mod;
    return ans;
}
VI BM(VI s) {
    VI C(1,1),B(1,1);
    int L=0,m=1,b=1;
    rep(n,0,SZ(s)) {
        ll d=0;
        rep(i,0,L+1) d=(d+(ll)C[i]*s[n-i])%mod;
        if (d==0) ++m;
        else if (2*L<=n) {
            VI T=C;
            ll c=mod-d*powmod(b,mod-2)%mod;
            while (SZ(C)<SZ(B)+m) C.pb(0);
            rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c*B[i])%mod;
            L=n+1-L; B=T; b=d; m=1;
        } else {
            ll c=mod-d*powmod(b,mod-2)%mod;
            while (SZ(C)<SZ(B)+m) C.pb(0);
            rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c*B[i])%mod;
            ++m;
        }
    }
```

```
        return C;
    }
    int gao(VI a,ll n) {//c.size()= 阶数
        VI c=BM(a);
        c.erase(c.begin());
        rep(i,0,SZ(c)) c[i]=(mod-c[i])%mod;
        return solve(n,c,VI(a.begin(),a.begin()+SZ(c)));
    }
};
int main()
{
    while (~scanf("%d",&n))
    {
        vector<int>v;
        v.push_back(1);
        v.push_back(1);
        v.push_back(2);
        v.push_back(3);
        v.push_back(5);
        v.push_back(8);
        //VI{1,1,2,3,5,8}       解出斐波那契数列
        printf("i:%d  arr:%d\n",n,linear_seq::gao(v,n-1));
    }
}
```

Fast Transfroming

FFT

- 复数类实现，n 为 2 的幂次

```
typedef double LD;
const LD PI = 3.14159265358979;
struct C
{
    LD r, i;
    C(LD r = 0, LD i = 0) : r(r), i(i) {}
    C operator+(const C &a) const
    {
        return C(r + a.r, i + a.i);
    }
    C operator-(const C &a) const
    {
        return C(r - a.r, i - a.i);
    }
    C operator*(const C &a) const
    {
        return C(r * a.r - i * a.i, r * a.i + i * a.r);
    }
};
void FFT(C x[], int n, int p)
{
    for (int i = 0, t = 0; i < n; ++i)
```

```
    {
        if (i > t)
            swap(x[i], x[t]);
        for (int j = n >> 1; (t ^= j) < j; j >>= 1)
            ;
    }
    for (int h = 2; h <= n; h <<= 1)
    {
        C wn(cos(p * 2 * PI / h), sin(p * 2 * PI / h));
        for (int i = 0; i < n; i += h)
        {
            C w(1, 0), u;
            for (int j = i, k = h >> 1; j < i + k; ++j)
            {
                u = x[j + k] * w;
                x[j + k] = x[j] - u;
                x[j] = x[j] + u;
                w = w * wn;
            }
        }
    }
    if (p == -1)
        for (int i = 0; i < n; ++i)
            x[i].r /= n;
}
void conv(C a[], C b[], int n)
{
    FFT(a, n, 1);
    FFT(b, n, 1);
    for (int i = 0; i < n; ++i)
        a[i] = a[i] * b[i];
    FFT(a, n, -1);
}
```

NTT

```
const int maxn=1e6+7;
ll wn[maxn << 2], rev[maxn << 2];
int G=3;//998244353
int NTT_init(int n_) {
    int step = 0; int n = 1;
    for ( ; n < n_; n <<= 1) ++step;
    for(int i=1;i<n;i++)
        rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (step - 1));
    int g = quick(G, (mod - 1) / n);
    wn[0] = 1;
    for (int i = 1; i <= n; ++i)
        wn[i] = wn[i - 1] * g % mod;
    return n;
}

void NTT(ll a[], int n, int f) {
```

```
    for(int i=0;i<n;i++) if (i < rev[i])
        std::swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k <<= 1) {
        for (int i = 0; i < n; i += (k << 1)) {
            int t = n / (k << 1);
            for(int j=0;j<k;j++){
                ll w = f == 1 ? wn[t * j] : wn[n - t * j];
                ll x = a[i + j];
                ll y = a[i + j + k] * w % mod;
                a[i + j] = (x + y) % mod;
                a[i + j + k] = (x - y + mod) % mod;
            }
        }
    }
    if (f == -1) {
        ll ninv = inv(n);
        for(int i=0;i<n;i++)
            a[i] = a[i] * ninv % mod;
    }
}
```

FWT

- $C_k = \sum_{i \oplus j = k} A_i B_j$

```
template<typename T>
void fwt(ll a[], int n, T f) {
    for (int d = 1; d < n; d *= 2)
        for (int i = 0, t = d * 2; i < n; i += t)
            for(int j = 0; j < d; j++)
                f(a[i + j], a[i + j + d]);
}

void AND(ll& a, ll& b) { a += b; }
void OR(ll& a, ll& b) { b += a; }
void XOR (ll& a, ll& b) {
    ll x = a, y = b;
    a = (x + y) % mod;
    b = (x - y + mod) % mod;
}
void rAND(ll& a, ll& b) { a -= b; }
void rOR(ll& a, ll& b) { b -= a; }
void rXOR(ll& a, ll& b) {
    static ll inv2 = (mod + 1) / 2;
    ll x = a, y = b;
    a = (x + y) * inv2 % mod;
    b = (x - y + mod) * inv2 % mod;
}
```

万能 FFT

```
namespace fft
{
```

```
struct num
{
    double x,y;
    num() {x=y=0;}
    num(double x,double y):x(x),y(y){}
};
inline num operator+(num a,num b) {return num(a.x+b.x,a.y+b.y);}
inline num operator-(num a,num b) {return num(a.x-b.x,a.y-b.y);}
inline num operator*(num a,num b) {return num(a.x*b.x-a.y*b.y,a.x*b.y+a.y*b.x);}
inline num conj(num a) {return num(a.x,-a.y);}

int base=1;
vector<num> roots={{0,0},{1,0}};
vector<int> rev={0,1};
const double PI=acosl(-1.0);

void ensure_base(int nbase)
{
    if(nbase<=base) return;
    rev.resize(1<<nbase);
    for(int i=0;i<(1<<nbase);i++)
        rev[i]=(rev[i>>1]>>1)+((i&1)<<(nbase-1));
    roots.resize(1<<nbase);
    while(base<nbase)
    {
        double angle=2*PI/(1<<(base+1));
        for(int i=1<<(base-1);i<(1<<base);i++)
        {
            roots[i<<1]=roots[i];
            double angle_i=angle*(2*i+1-(1<<base));
            roots[(i<<1)+1]=num(cos(angle_i),sin(angle_i));
        }
        base++;
    }
}

void fft(vector<num> &a,int n=-1)
{
    if(n==-1) n=a.size();
    assert((n&(n-1))==0);
    int zeros=__builtin_ctz(n);
    ensure_base(zeros);
    int shift=base-zeros;
    for(int i=0;i<n;i++)
        if(i<(rev[i]>>shift))
            swap(a[i],a[rev[i]>>shift]);
    for(int k=1;k<n;k<<=1)
    {
        for(int i=0;i<n;i+=2*k)
        {
            for(int j=0;j<k;j++)
            {
```

```
                    num z=a[i+j+k]*roots[j+k];
                    a[i+j+k]=a[i+j]-z;
                    a[i+j]=a[i+j]+z;
                }
            }
        }
    }

    vector<num> fa,fb;

    vector<int> multiply(vector<int> &a, vector<int> &b)
    {
        int need=a.size()+b.size()-1;
        int nbase=0;
        while((1<<nbase)<need) nbase++;
        ensure_base(nbase);
        int sz=1<<nbase;
        if(sz>(int)fa.size()) fa.resize(sz);
        for(int i=0;i<sz;i++)
        {
            int x=(i<(int)a.size()?a[i]:0);
            int y=(i<(int)b.size()?b[i]:0);
            fa[i]=num(x,y);
        }
        fft(fa,sz);
        num r(0,-0.25/sz);
        for(int i=0;i<=(sz>>1);i++)
        {
            int j=(sz-i)&(sz-1);
            num z=(fa[j]*fa[j]-conj(fa[i]*fa[i]))*r;
            if(i!=j) fa[j]=(fa[i]*fa[i]-conj(fa[j]*fa[j]))*r;
            fa[i]=z;
        }
        fft(fa,sz);
        vector<int> res(need);
        for(int i=0;i<need;i++) res[i]=fa[i].x+0.5;
        return res;
    }

    vector<int> multiply_mod(vector<int> &a,vector<int> &b,int m,int eq=0)
    {
        int need=a.size()+b.size()-1;
        int nbase=0;
        while((1<<nbase)<need) nbase++;
        ensure_base(nbase);
        int sz=1<<nbase;
        if(sz>(int)fa.size()) fa.resize(sz);
        for(int i=0;i<(int)a.size();i++)
        {
            int x=(a[i]%m+m)%m;
            fa[i]=num(x&((1<<15)-1),x>>15);
        }
```

```
        fill(fa.begin()+a.size(),fa.begin()+sz,num{0,0});
        fft(fa,sz);
        if(sz>(int)fb.size()) fb.resize(sz);
        if(eq) copy(fa.begin(),fa.begin()+sz,fb.begin());
        else
        {
            for(int i=0;i<(int)b.size();i++)
            {
                int x=(b[i]%m+m)%m;
                fb[i]=num(x&((1<<15)-1),x>>15);
            }
            fill(fb.begin()+b.size(),fb.begin()+sz,num{0,0});
            fft(fb,sz);
        }
        double ratio=0.25/sz;
        num r2(0,-1),r3(ratio,0),r4(0,-ratio),r5(0,1);
        for(int i=0;i<=(sz>>1);i++)
        {
            int j=(sz-i)&(sz-1);
            num a1=(fa[i]+conj(fa[j]));
            num a2=(fa[i]-conj(fa[j]))*r2;
            num b1=(fb[i]+conj(fb[j]))*r3;
            num b2=(fb[i]-conj(fb[j]))*r4;
            if(i!=j)
            {
                num c1=(fa[j]+conj(fa[i]));
                num c2=(fa[j]-conj(fa[i]))*r2;
                num d1=(fb[j]+conj(fb[i]))*r3;
                num d2=(fb[j]-conj(fb[i]))*r4;
                fa[i]=c1*d1+c2*d2*r5;
                fb[i]=c1*d2+c2*d1;
            }
            fa[j]=a1*b1+a2*b2*r5;
            fb[j]=a1*b2+a2*b1;
        }
        fft(fa,sz);fft(fb,sz);
        vector<int> res(need);
        for(int i=0;i<need;i++)
        {
            ll aa=fa[i].x+0.5;
            ll bb=fb[i].x+0.5;
            ll cc=fa[i].y+0.5;
            res[i]=(aa+((bb%m)<<15)+((cc%m)<<30))%m;
        }
        return res;
    }
    vector<int> square_mod(vector<int> &a,int m)
    {
        return multiply_mod(a,a,m,1);
    }
};
```

离散对数

BSGS

- ~~北上广深，拔山盖世~~应用于模数为质数

```cpp
ll BSGS(ll a, ll b, ll p) // a^x = b (mod p)
{
    a %= p;
    if (!a && !b)
        return 1;
    if (!a)
        return -1;
    static map<ll, ll> mp;
    mp.clear();
    ll m = sqrt(p + 1.5);
    ll v = 1;
    for (int i = 1; i < m + 1; ++i)
    {
        v = v * a % p;
        mp[v * b % p] = i;
    }
    ll vv = v;
    for (int i = 1; i < m + 1; ++i)
    {
        auto it = mp.find(vv);
        if (it != mp.end())
            return i * m - it->second;
        vv = vv * v % p;
    }
    return -1;
}
```

exBSGS

```cpp
ll exBSGS(ll a, ll b, ll p)  // a^x = b (mod p)
{
    a %= p; b %= p;
    if (a == 0) return b > 1 ? -1 : b == 0 && p != 1;
    LL c = 0, q = 1;
    while (1) {
        ll g = __gcd(a, p);
        if (g == 1) break;
        if (b == 1) return c;
        if (b % g) return -1;
        ++c; b /= g; p /= g; q = a / g * q % p;
    }
    static map<ll, ll> mp; mp.clear();
    ll m = sqrt(p + 1.5);
    ll v = 1;
    for(int i = 1; i< m + 1 ; i++)
    {
        v = v * a % p;
        mp[v * b % p] = i;
```

```
    }
    for(int i = 1; i< m + 1; i++)
    {
        q = q * v % p;
        auto it = mp.find(q);
        if (it != mp.end()) return i * m - it->second + c;
    }
    return -1;
}
```

二次剩余

- $x == -1$ 时无根
- 否则有两个解: x , $p - x$

```
ll a, p, w;
struct T
{
    ll x, y;
};

T mul_two(T a, T b, ll p)
{
    T ans;
    ans.x = (a.x * b.x % p + a.y * b.y % p * w % p) % p;
    ans.y = (a.x * b.y % p + a.y * b.x % p) % p;
    return ans;
}

T qpow_two(T a, ll n, ll p)
{
    T ans;
    ans.x = 1;
    ans.y = 0;
    while (n)
    {
        if (n & 1)
            ans = mul_two(ans, a, p);
        n >>= 1;
        a = mul_two(a, a, p);
    }
    return ans;
}

ll qpow(ll a, ll n, ll p)
{
    ll ans = 1;
    a %= p;
    while (n)
    {
        if (n & 1)
            ans = ans * a % p;
        n >>= 1;
```

```
        a = a * a % p;
    }
    return ans % p;
}

ll Legendre(ll a, ll p)
{
    return qpow(a, (p - 1) >> 1, p);
}

int solve(ll n, ll p)
{
    if (n == 0)
        return 0;
    if (p == 2)
        return 1;
    if (Legendre(n, p) + 1 == p)
        return -1;
    ll a, t;
    while (1)
    {
        a = rand() % p;
        t = a * a - n;
        w = (t % p + p) % p;
        if (Legendre(w, p) + 1 == p)
            break;
    }
    T tmp;
    tmp.x = a;
    tmp.y = 1;
    T ans = qpow_two(tmp, (p + 1) >> 1, p);
    return ans.x;
}
```

中国剩余定理

- 逐项合并，支持不互质，无解返回-1
- 前置 exgcd

```
ll CRT(ll *m, ll *r, ll n)
{
    if (!n)
        return 0;
    ll M = m[0], R = r[0], x, y, d;
    for (int i = 1; i < n; i++)
    {
        d = exgcd(M, m[i], x, y);
        if ((r[i] - R) % d)
            return -1;
        x = (r[i] - R) / d * x % (m[i] / d);
        R += x * M;
        M = M / d * m[i];
        R %= M;
```

```
    }
    return R >= 0 ? R : R + M;
}
```

线性基

```
template<typename T,int D>
struct Base{
    T a[D];
    int m;
    Base(){m = 0, memset(a, 0, sizeof(a));}
    void clear(){m = 0, memset(a, 0, sizeof(a));}
    bool ins(T x)
    {
        for(int i = D - 1; ~i; --i)
            if(x >> i & 1)
            {
                if(a[i]) x ^= a[i];
                else{
                    m++;
                    a[i] = x;return 1;
                }
            }
        return 0;
    }
};
//求交
template<typename T,int D>
Base<T,D> Merge(Base<T,D> A,Base<T,D> B)
{
    if(A.m==D) return B;
    if(B.m==D) return A;
    Base<T,D> All,C,D;
    All=A;
    D.ful();
    for(int i=D-1;i>=0;i--)
    {
        if(B.a[i]){
            T v=B.a[i],k=0;
            bool can=1;
            for(int j=D-1;j>=0;j--)
            {
                if(v>>j&1)
                {
                    if(All.a[j])
                    {
                        v^=All.a[j];
                        k^=D.a[j];
                    }
                    else{
                        can=0;
                        All.a[j]=v;
```

```
                        D.a[j]=k;
                        break;
                    }
                }
            }
            if(can)
            {
                T v=0;
                for(int j=D-1;j>=0;j--)
                {
                    if(k>>j&1)
                        v^=A.a[j];
                }
                C.ins(v);
            }
        }
    }
    return C;
}
```

高斯消元

```
const double eps=1e-8;
typedef vector<double> vec;
typedef vector<vec> mat;
int sz;
vec gauss_jordan(const mat& A, const vec& b)
{
    int n=A.size();
    mat B(n,vec(n+1));
    for(int i=0;i<n;i++)
        for(int j=0;j<n;j++)
            B[i][j]=A[i][j];

    for(int i=0;i<n;i++) B[i][n]=b[i];
    for(int i=0;i<n;i++)
    {
        int pivot=i;
        for(int j=i;j<n;j++)
            if(abs(B[j][i])>abs(B[pivot][i])) pivot=j;
        swap(B[i],B[pivot]);
        if(abs(B[i][i])<eps) return vec();
        for(int j=i+1;j<=n;j++) B[i][j]/=B[i][i];
        for(int j=0;j<n;j++)
        {
            if(i!=j)
            {
                for(int k=i+1;k<=n;k++)
                    B[j][k]-=B[j][i]*B[i][k];
            }
        }
    }
```

```
    vec x(n);
    for(int i=0;i<n;i++)
        x[i]=B[i][n];
    return x;
}
```

素数测试

miller_rabin

```
ll power(ll v, ll p, ll m)
{
    ll r = 1;
    while (p)
    {
        if (p & 1)
            r = r * v % m;
        v = v * v % m;
        p >>= 1;
    }

    return r;
}

bool witness(ll a, ll p)
{
    int k = 0;
    ll q = p - 1;
    while ((q & 1) == 0)
        ++k, q >>= 1;
    ll v = power(a, q, p);
    if (v == 1 || v == p - 1)
        return false; // probably prime number
    while (k-- != 0)
    {
        v = v * v % p;
        if (v == p - 1)
            return false;
    }

    return true; // composite number
}

bool miller_rabin(ll p)
{
    if (p == 1)
        return false;
    if (p == 2)
        return true;
    if (p % 2 == 0)
        return false;

    for (int i = 0; i != 50; ++i)
```

```
    {
        ll a = std::rand() % (p - 1) + 1;
        if (witness(a, p))
            return false;
    }

    return true;
}
```

Pollard Rho

```
ll pollard_rho(ll n, int a)
{
    ll x = 2, y = 2, d = 1, k = 0, i = 1;
    while(d == 1)
    {
        ++k;
        x = mul_mod(x, x, n) + a;
        d = __gcd(x >= y ? x - y : y - x, n);
        if(k == i)
        {
            y = x;
            i <<= 1;
        }
    }
    if(d == n) return pollard_rho(n, a + 1);
    return d;
}
```

博弈

- 巴什博弈：P 态为 $n \equiv 0 (mod\ m + 1)$
- 阶梯博弈：阶梯博弈等效为奇数号阶梯的 nim 博弈
- 威佐夫博弈：有两堆各若干个物品，两个人轮流从某一堆取至少一个或同时从两堆中取同样多的物品，规定每次至少取一个，多者不限，最后取光者得胜。P 态为 $(y - x) \times \frac{\sqrt{5}+1}{2} = x$
- NP 图、SG 函数找规律，博弈 dp
- 考虑模仿操作

字符串

KMP

```
char s[maxn], t[maxn];
int fail[maxn];
void getfail()
{
    memset(fail, 0, sizeof(fail));
    int len = strlen(t);
    int j = 0, k = fail[0] = -1;
    while (j < len)
    {
        while (k != -1 && t[j] != t[k])
            k = fail[k];
        fail[++j] = ++k;
    }
}

int kmp()
{
    int n = strlen(s), m = strlen(t);
    int i = 0, j = 0;
    int ret = 0;
    while (i < n)
    {
        while (j != -1 && s[i] != t[j])
            j = fail[j];
        i++, j++;
        if (j == m)
            ret++, j = fail[j];
    }
    return ret;
}
```

Manacher

```
const int maxn = 2e5;

string Mnc(string &s)
{
    string t = "$#";
    for (int i = 0; i < s.length(); ++i) //构造辅助串
    {
        t += s[i];
        t += '#';
    }

    int ml = 0, p = 0, R = 0, M = 0;
    //最大长度，最长回文中心，当前最大回文串右端，当前最长回文中心

    int len = t.length();
    vector<int> P(len, 0); //回长度数组
```

```
    for (int i = 0; i < len; ++i)
    {
        P[i] = R > i ? min(P[2 * M - i], R - i) : 1; //转移方程

        while (t[i + P[i]] == t[i - P[i]]) //长度扩张
            ++P[i];

        if (i + P[i] > R) //更新右端和中心
        {
            R = i + P[i];
            M = i;
        }
        if (ml < P[i]) //记录极大
        {
            ml = P[i];
            p = i;
        }
    }

    return s.substr((p - ml) / 2, ml - 1); //返回回文串
}
```

后缀数组

```
char s[maxn];
int sa[maxn], t[maxn], t2[maxn], c[maxn], rk[maxn], height[maxn];
//sa[],height[] 下标从 1 开始，rk[] 下标从 0 开始
void getsa(int m, int n)
{ //n 为字符串的长度，字符集的值为 0~m-1
    n++;
    int *x = t, *y = t2;
    //基数排序
    for (int i = 0; i < m; i++)
        c[i] = 0;
    for (int i = 0; i < n; i++)
        c[x[i] = s[i]]++;
    for (int i = 1; i < m; i++)
        c[i] += c[i - 1];
    for (int i = n - 1; ~i; i--)
        sa[--c[x[i]]] = i;
    for (int k = 1; k <= n; k <<= 1)
    { //直接利用 sa 数组排序第二关键字
        int p = 0;
        for (int i = n - k; i < n; i++)
            y[p++] = i;
        for (int i = 0; i < n; i++)
            if (sa[i] >= k)
                y[p++] = sa[i] - k;
        //基数排序第一关键字
        for (int i = 0; i < m; i++)
            c[i] = 0;
        for (int i = 0; i < n; i++)
```

```
            c[x[y[i]]]++;
        for (int i = 1; i < m; i++)
            c[i] += c[i - 1];
        for (int i = n - 1; ~i; i--)
            sa[--c[x[y[i]]]] = y[i];
        //根据 sa 和 y 数组计算新的 x 数组
        swap(x, y);
        p = 1;
        x[sa[0]] = 0;
        for (int i = 1; i < n; i++)
            x[sa[i]] = y[sa[i - 1]] == y[sa[i]] && y[sa[i - 1] + k] == y[sa[i] + k] ? p - 1 : p++;
        if (p >= n)
            break; //以后即使继续倍增，sa 也不会改变，推出
        m = p;              //下次基数排序的最大值
    }
    n--;
    int k = 0;
    for (int i = 0; i <= n; i++)
        rk[sa[i]] = i;
    for (int i = 0; i < n; i++)
    {
        if (k) k--;
        int j = sa[rk[i] - 1];
        while (s[i + k] == s[j + k])
            k++;
        height[rk[i]] = k;
    }
}

int dp[maxn][30];
void initrmq(int n)
{
    for (int i = 1; i <= n; i++)
        dp[i][0] = height[i];
    for (int j = 1; (1 << j) <= n; j++)
        for (int i = 1; i + (1 << j) - 1 <= n; i++)
            dp[i][j] = min(dp[i][j - 1], dp[i + (1 << (j - 1))][j - 1]);
}
int rmq(int l, int r)
{
    int k = 31 - __builtin_clz(r - l + 1);
    return min(dp[l][k], dp[r - (1 << k) + 1][k]);
}
int lcp(int a, int b)
{ // 求两个后缀的最长公共前缀
    a = rk[a], b = rk[b];
    if (a > b)
        swap(a, b);
    return rmq(a + 1, b);
}
```

后缀自动机

```cpp
char s[maxn];
int ch[maxn][26], step[maxn], pre[maxn];
int to[maxn], topo[maxn], cntr[maxn], sum[maxn];
int sz, last; // init(){sz=last=1;}
void ins(int x)
{
    int np = ++sz, p = last;
    last = np;
    step[np] = step[p] + 1;
    cntr[np] = 1;
    while (!ch[p][x] && p)
        ch[p][x] = np, p = pre[p];
    if (!p)
        pre[np] = 1;
    else
    {
        int q = ch[p][x];
        if (step[q] == step[p] + 1)
            pre[np] = q;
        else
        {
            int nq = ++sz;
            step[nq] = step[p] + 1;
            for (int i = 0; i < 26; ++i)
                ch[nq][i] = ch[q][i];
            pre[nq] = pre[q];
            pre[q] = pre[np] = nq;
            while (ch[p][x] == q && p)
                ch[p][x] = nq, p = pre[p];
        }
    }
}

void getr()
{
    for (int i = 1; i <= sz; ++i)
        ++to[step[i]]; //利用后缀自动机性质拓扑排序
    for (int i = 1; i <= sz; ++i)
        to[i] += to[i - 1];
    for (int i = 1; i <= sz; ++i)
        topo[to[step[i]]--] = i;
    for (int i = sz; i >= 1; --i)
        if (ty)
            cntr[pre[topo[i]]] += cntr[topo[i]];
        else
            cntr[i] = 1;
    cntr[1] = 0;
    for (int i = sz; i >= 1; --i)
    {
        int x = topo[i];
```

```
        sum[x] = cntr[x]; //sum: 停下或继续，你还能走出多少个子串
        for (int j = 0; j < 26; ++j)
            if (ch[x][j])
                sum[x] += sum[ch[x][j]];
    }
}
```

广义后缀自动机

```
//每个串 last 置 1
void ins(int x)
{
    x-='a';
    int p=last,np=0,nq=0,q=-1;
    if(!ch[p][x])
    {
        np = ++sz;
        step[np] = step[p] + 1;
        while (!ch[p][x] && p)
            ch[p][x] = np, p = pre[p];
    }
    if (!p)
        pre[np] = 1;
    else
    {
        q = ch[p][x];
        if (step[q] == step[p] + 1)
            pre[np] = q;
        else
        {
            int nq = ++sz;
            step[nq] = step[p] + 1;
            for (int i = 0; i < 26; ++i)
                ch[nq][i] = ch[q][i];
            pre[nq] = pre[q];
            pre[q] = pre[np] = nq;
            while (ch[p][x] == q && p)
                ch[p][x] = nq, p = pre[p];
        }
    }
    last=np ? np : nq ? nq : q;
}
```

AC 自动机

```
struct AC_auto
{
    int ch[maxn][26];
    int num[maxn], fail[maxn];
    // f 即为 fail 指针.
    int tot;
    void init()
```

```cpp
    {
        tot = 0;
        for (int i = 0; i < 26; ++i)
            ch[0][i] = 0;
    }
    void insert(char s[], int len)
    {
        int u = 0;
        for (int i = 0; i < len; i++)
        {
            if (!ch[u][s[i] - 'a'])
            {
                ch[u][s[i] - 'a'] = ++tot;
                for (int j = 0; j < 26; ++j)
                    ch[tot][j] = 0;
                num[tot] = fail[tot] = 0;
            }
            u = ch[u][s[i] - 'a'];
        }
        num[u]++;
    } //往 Trie 树里插入元素.
    void build()
    {
        queue<int> q;
        for (int i = 0; i < 26; i++)
        {
            if (ch[0][i])
                fail[ch[0][i]] = 0,
                //第一层与其他单词不可能有公共前后缀,fail 直接为根.
                    q.push(ch[0][i]);
        }
        while (!q.empty())
        {
            int u = q.front();
            q.pop();
            for (int i = 0; i < 26; i++)
                if (ch[u][i])
                {
                    fail[ch[u][i]] = ch[fail[u]][i];
                    q.push(ch[u][i]);
                }
                else
                    ch[u][i] = ch[fail[u]][i];
            //这一步直接省略了查询时的比较.
        }
    } //构建 Fail 指针.
    int query(char s[], int len)
    {
        int u = 0, ans = 0;
        for (int i = 0; i < len; i++)
        {
            u = ch[u][s[i] - 'a'];
```

```
            for (int j = u; j && num[j] != -1; j = fail[j])
                ans += num[j], num[j] = -1;
            //因为直接已经在每个单词的最后面打了标记，所以直接加上即可.
        }
        return ans;
    }
} AC;
```

最小表示法

```
int getMin(char *s)
{
    int i = 0, j = 1, l;
    int len = strlen(s);
    while (i < len && j < len)
    {
        for (l = 0; l < len; l++)
            if (s[(i + l) % len] != s[(j + l) % len])
                break;
        if (l >= len)
            break;
        if (s[(i + l) % len] > s[(j + l) % len])
        {
            if (i + l + 1 > j)
                i = i + l + 1;
            else
                i = j + 1;
        }
        else if (j + l + 1 > i)
            j = j + l + 1;
        else
            j = i + 1;
    }
    return i < j ? i : j;
}

int getMax(char *s)
{
    int len = strlen(s);
    int i = 0, j = 1, k = 0;
    while (i < len && j < len && k < len)
    {
        int t = s[(i + k) % len] - s[(j + k) % len];
        if (!t)
            k++;
        else
        {
            if (t > 0)
            {
                if (j + k + 1 > i)
                    j = j + k + 1;
                else
```

```
                    j = i + 1;
                }
                else if (i + k + 1 > j)
                    i = i + k + 1;
                else
                    i = j + 1;
                k = 0;
            }
        }
        return i < j ? i : j;
    }
```

回文自动机

```
const int maxn = 5e5 + 7;

struct PAM
{
    int next[maxn][26]; //next 指针，和字典树类似，指向的串为当前串两端加上同一个字符构成。
    int fail[maxn], cnt[maxn], num[maxn], len[maxn], s[maxn];
    int last, n, p;
    int newnode(int rt)
    {
        for (int i = 0; i < 26; i++)
            next[p][i] = 0;
        cnt[p] = 0;
        num[p] = 0;
        len[p] = rt;
        return p++;
    }

    void init()
    {
        p = last = n = 0;
        newnode(0);
        newnode(-1);
        s[n] = -1;
        fail[0] = 1;
    }

    int getFail(int x) //fail 指针的构建
    {
        while (s[n - len[x] - 1] != s[n])
            x = fail[x];
        return x;
    }

    int ins(int c) //插入字符
    {
        c -= 'a';
        s[++n] = c;
        int cur = getFail(last);
```

```
        if (!next[cur][c]) //如果不存此字符节点
        {
            int now = newnode(len[cur] + 2);          //+2: 回文所以两段同时加 1
            fail[now] = next[getFail(fail[cur])][c]; //构建此处的 fail
            next[cur][c] = now;                       //构建此处的 next
            num[now] = num[fail[now]] + 1;            //以此末尾字母结尾的回文串个数
        }
        last = next[cur][c]; //last 指针
        cnt[last]++;
        return last;
    }

    void count()
    {
        for (int i = p - 1; i >= 0; i--)
            cnt[fail[i]] += cnt[i]; //父节点累加子节点的 cnt (若 fail[v]=u, 则 u 一定是 v 的子回文串)
    }

} pam;
```

EXKMP

```
char s[maxn];
int nxt[maxn],ex[maxn]; //ex 数组即为 extend 数组
//预处理计算 nxt 数组
void GETNEXT(char *str)
{
    int i=0,j,po,len=strlen(str);
    nxt[0]=len;//初始化 nxt[0]
    while(str[i]==str[i+1]&&i+1<len)//计算 nxt[1]
        i++;
    nxt[1]=i;
    po=1;//初始化 po 的位置
    for(i=2;i<len;i++)
    {
        if(nxt[i-po]+i<nxt[po]+po)//第一种情况，可以直接得到 nxt[i] 的值
            nxt[i]=nxt[i-po];
        else//第二种情况，要继续匹配才能得到 nxt[i] 的值
        {
            j=nxt[po]+po-i;
            if(j<0)j=0;//如果 i>po+nxt[po], 则要从头开始匹配
            while(i+j<len&&str[j]==str[j+i])//计算 nxt[i]
                j++;
            nxt[i]=j;
            po=i;//更新 po 的位置
        }
    }
}

//计算 extend 数组
void EXKMP(char *s1,char *s2)
```

```
{
    int i=0,j,po,len=strlen(s1),l2=strlen(s2);
    GETNEXT(s2);//计算子串的 nxt 数组
    while(s1[i]==s2[i]&&i<l2&&i<len)//计算 ex[0]
        i++;
    ex[0]=i;
    po=0;//初始化 po 的位置
    for(i=1;i<len;i++)
    {
        if(nxt[i-po]+i<ex[po]+po)//第一种情况，直接可以得到 ex[i] 的值
            ex[i]=nxt[i-po];
        else//第二种情况，要继续匹配才能得到 ex[i] 的值
        {
            j=ex[po]+po-i;
            if(j<0)j=0;//如果 i>ex[po]+po 则要从头开始匹配
            while(i+j<len&&j<l2&&s1[j+i]==s2[j])//计算 ex[i]
                j++;
            ex[i]=j;
            po=i;//更新 po 的位置
        }
    }
}
```

数据结构

无旋 Treap

- 大根堆

维护集合

- 小于 k 分为左子树，大于等于 k 分为右子树

```
struct Treap
{
    Treap *l, *r;
    int val, prior;
    Treap(int _val) : val(_val), l(NULL), r(NULL), prior(rnd()) {}
};
typedef Treap *pt;
void split(pt o, int k, pt &l, pt &r)
{
    if (!o)
        l = r = NULL;
    else if (o->val >= k)
        split(o->l, k, l, o->l), r = o;
    else
        split(o->r, k, o->r, r), l = o;
}
void merge(pt &o, pt l, pt r)
{
    if (!l || !r)
        o = l ? l : r;
    else if (l->prior > r->prior)
        merge(l->r, l->r, r), o = l;
    else
        merge(r->l, l, r->l), o = r;
}
pt root;
```

ST 表

- 一维

```
void ST(int n) //处理出 [1,n] 的 RMQ
{
    for (int i = 1; i <= n; i++)
        dp[i][0] = arr[i];

    for (int j = 1; (1 << j) <= n; j++)
    {
        for (int i = 1; i + (1 << j) - 1 <= n; i++)
            dp[i][j] = min(dp[i][j - 1], dp[i + (1 << (j - 1))][j - 1]);
    }
}

int query(int l, int r)
```

```cpp
{
    int k = 31 - __builtin_clz(r - l + 1);
    return min(dp[l][k], dp[r - (1 << k) + 1][k]);
}
```

树状数组

- 区间最值
- $O(log^2(n))$

```cpp
int a[maxn], tree[maxn]; //a[] 存原始数据, tree[] 存树状数组
int n;
//先改 a[x], 然后 update(x)
void update(int x)
{
    int lx, i;
    while (x < n)
    {
        tree[x] = a[x];
        lx = -x & x;
        for (i = 1; i < lx; i <<= 1)
            tree[x] = max(tree[x], tree[x - i]);
        x += -x & x;
    }
}
int query(int x, int y) //[x,y] 区间最值
{
    int ret = 0;
    while (y >= x)
    {
        ret = max(a[y], ret);
        y--;
        for (; y - (-y & y) >= x; y -= -y & y)
            ret = max(tree[y], ret);
    }
    return ret;
}
```

- 区间修改、区间查询（查询前缀和的前缀和）

```cpp
int tr[maxn], trr[maxn];
void add(int x, int val)
{
    for (int i = x; i < maxn; i += i & -i)
    {
        tr[i] += val;
        trr[i] += x * val;
    }
}
void add(int l, int r, int val)
{
    add(l, val);
    add(r + 1, -val);
}
```

```
int sum(int x)
{
    int ret = 0;
    for (int i = x; i > 0; i -= i & -i)
        ret += (x + 1) * tr[i] - trr[i];
    return ret;
}
int sum(int l, int r) { return sum(r) - sum(l - 1); }
```

笛卡尔树

- $O(n)$ 建树，大根堆

```
stack<int> st;
for (int i = 0; i < n; i++)
{
    int last = -1;
    while (!st.empty() && arr[i] > arr[st.top()])
        last = st.top(), st.pop();
    if (!st.empty())
        rc[st.top()] = i, fa[i] = st.top();
    lc[i] = last;
    if (~last)
        fa[last] = i;
    st.push(i);
}
int root = -1;
for (int i = 0; i < n; i++)
    if (!~fa[i])
        root = i;
```

主席树优化建图

- 只写了点向区间连边

```
struct Node
{
    Node *l, *r;
    int id;
    Node(int _id) : id(_id), l(NULL), r(NULL) {}
};
Node *rt[maxn];
int tot; //编号
int ins[maxn];
#define Lson L, mid, o->l
#define Rson mid + 1, R, o->r
void build(int L, int R, Node *&o)
{
    o = new Node(tot++);
    if (L == R)
    {
        addedge(o->id, L);
        return;
```

```
    }
    int mid = L + R >> 1;
    build(Lson);
    build(Rson);
    addedge(o->id, o->l->id);
    addedge(o->id, o->r->id);
}
void update(int p, int l, int r, int L, int R, Node *o)
{
    if (l <= L && r >= R)
    {
        addedge(ins[p], o->id);
        return;
    }
    int mid = L + R >> 1;
    if (l <= mid)
        update(p, l, r, Lson);
    if (r > mid)
        update(p, l, r, Rson);
}
void add(int pos, int L, int R, Node *&o, Node *pre)
{
    o = new Node(tot++);
    if (L == R)
    {
        addedge(o->id, ins[pos]);
        return;
    }
    int mid = L + R >> 1;
    if (pos <= mid)
    {
        add(pos, Lson, pre->l);
        o->r = pre->r;
    }
    else
    {
        add(pos, Rson, pre->r);
        o->l = pre->l;
    }
    addedge(o->id, o->l->id);
    addedge(o->id, o->r->id);
}
```

Link-Cut Tree

- 修改点值先 makeroot

```
#define lc ch[x][0]
#define rc ch[x][1]
namespace LCT
{
int fa[maxn], ch[maxn][2], val[maxn], pre[maxn], lz[maxn];
inline bool nroot(int x)
```

```
{
    return ch[fa[x]][0] == x || ch[fa[x]][1] == x;
}
inline void pushup(int x) //维护链信息
{
    pre[x] = pre[lc] ^ pre[rc] ^ val[x];
}
inline void pushr(int x)
{
    swap(lc, rc);
    lz[x] ^= 1;
} //反转
inline void pushdown(int x)
{
    if (lz[x])
    {
        if (lc)
            pushr(lc);
        if (rc)
            pushr(rc);
        lz[x] = 0;
    }
}
void rotate(int x) //单次旋转
{
    int y = fa[x], z = fa[y], k = ch[y][1] == x, w = ch[x][!k];
    if (nroot(y))
        ch[z][ch[z][1] == y] = x;
    ch[x][!k] = y;
    ch[y][k] = w;
    if (w)
        fa[w] = y;
    fa[y] = x;
    fa[x] = z;
    pushup(y);
}
void pushall(int x) //递归下放标记
{
    if (nroot(x))
        pushall(fa[x]);
    pushdown(x);
}
void splay(int x)
{
    pushall(x);
    while (nroot(x))
    {
        int y = fa[x];
        int z = fa[y];
        if (nroot(y))
            rotate((ch[y][0] == x) ^ (ch[z][0] == y) ? x : y);
        rotate(x);
```

```
    }
    pushup(x);
}
void access(int x)
{
    for (int y = 0; x; x = fa[y = x])
    {
        splay(x);
        rc = y;
        pushup(x);
    }
}
void makeroot(int x)
{
    access(x);
    splay(x);
    pushr(x);
}
int findroot(int x)
{
    access(x);
    splay(x);
    while (lc)
        pushdown(x), x = lc;
    splay(x);
    return x;
}
void split(int x, int y)
{
    makeroot(x);
    access(y);
    splay(y);
}
void link(int x, int y)
{
    makeroot(x);
    if (findroot(y) != x)
        fa[x] = y;
}
void cut(int x, int y)
{
    makeroot(x);
    if (findroot(y) == x && fa[y] == x && !ch[y][0])
    {
        fa[y] = ch[x][1] = 0;
        pushup(x);
    }
}
}; // namespace LCT
```

Splay

```
struct Splay
{

    struct Node
    {
        int father, childs[2], key, cnt, sz;
        void init() {father = childs[0] = childs[1] = key = cnt = sz = 0;}
        void init(int fa, int lc, int rc, int k, int c, int s)
        {
            father = fa;
            childs[0] = lc;
            childs[1] = rc;
            key = k;
            cnt = c;
            sz = s;
        }
    } tre[maxn];

    int tot, root;
    void init() {tot = root = 0;}

    inline bool judge(int x) {return tre[ tre[x].father ].childs[1] == x;}

    inline void update(int x)
    {
        if(x)
        {
            tre[x].sz = tre[x].cnt;
            if(tre[x].childs[0])
                tre[x].sz += tre[ tre[x].childs[0] ].sz;
            if(tre[x].childs[1])
                tre[x].sz += tre[ tre[x].childs[1] ].sz;
        }
    }

    inline void rotate(int x)
    {
        int y = tre[x].father, z = tre[y].father, k = judge(x);
        tre[y].childs[k] = tre[x].childs[!k];
        tre[ tre[x].childs[!k] ].father = y;
        tre[x].childs[!k] = y;
        tre[y].father = x;
        tre[z].childs[ tre[z].childs[1] == y ] = x;
        tre[x].father = z;
        update(y);
    }

    void splay(int x,int goal)
    {
        for(int father; (father = tre[x].father) != goal; rotate(x) )
```

```
            if(tre[father].father != goal)
                rotate(judge(x) == judge(father) ? father : x);
        if(goal == 0)
            root = x;
    }

    void insert(int x)
    {
        if(root == 0)
        {
            tre[++tot].init(0, 0, 0, x, 1, 1);
            root = tot;
            return ;
        }
        int now = root, father = 0;
        while(1)
        {
            if(tre[now].key == x)
            {
                tre[now].cnt ++;
                update(now), update(father);
                splay(now, 0);
                break;
            }
            father = now;
            if(x > tre[now].key)
                now = tre[now].childs[1];
            else
                now = tre[now].childs[0];

            if(now == 0)
            {
                tre[++tot].init(father, 0, 0, x, 1, 1);
                if(x > tre[father].key)
                    tre[father].childs[1] = tot;
                else
                    tre[father].childs[0] = tot;
                update(father);
                splay(tot, 0);
                break;
            }
        }
    }

    int pre()
    {
        int now = tre[root].childs[0];
        while(tre[now].childs[1])
            now = tre[now].childs[1];
        return now;
    }

```

```
    int next()
    {
        int now = tre[root].childs[1];
        while(tre[now].childs[0])
            now = tre[now].childs[0];
        return now;
    }

    int rnk(int x)
    { /// 找 x 的排名
        int now = root, ans = 0;
        while(1)
        {
            if(x < tre[now].key)
                now = tre[now].childs[0];
            else
            {
                if(tre[now].childs[0])
                    ans += tre[ tre[now].childs[0] ].sz;
                if(x == tre[now].key)
                {
                    splay(now, 0);
                    return ans + 1;
                }
                ans += tre[now].cnt;
                now = tre[now].childs[1];
            }
        }
    }

    int kth(int x)
    { /// 找排名为 x 的数字
        int now = root;
        while(1)
        {
            if(tre[now].childs[0] && x <= tre[ tre[now].childs[0] ].sz )
                now = tre[now].childs[0];
            else
            {
                int lchild = tre[now].childs[0], sum = tre[now].cnt;
                if(lchild)
                    sum += tre[lchild].sz;
                if(x <= sum)
                    return tre[now].key;
                x -= sum;
                now = tre[now].childs[1];
            }
        }
    }

    void del(int x)
    {
```

```
        find(x);
        if(tre[root].cnt > 1)
        {
            tre[root].cnt --;
            update(root);
            return ;
        }
        if(!tre[root].childs[0] && !tre[root].childs[1])
        {
            tre[root].init();
            root = 0;
            return ;
        }
        if(!tre[root].childs[0])
        {
            int old_root = root;
            root = tre[root].childs[1];
            tre[root].father = 0;
            tre[old_root].init();
            return ;
        }
        if(!tre[root].childs[1])
        {
            int old_root = root;
            root = tre[root].childs[0];
            tre[root].father = 0;
            tre[old_root].init();
            return ;
        }
        int pre_node = pre(), old_root = root;
        splay(pre_node, 0);
        tre[root].childs[1] = tre[old_root].childs[1];
        tre[ tre[old_root].childs[1] ].father = root;
        tre[old_root].init();
        update(root);
    }

    bool find(int x)
    {
        int now = root;
        while(1)
        {
            if(now == 0)
                return 0;
            if(x == tre[now].key)
            {
                splay(now, 0);
                return 1;
            }
            if(x > tre[now].key)
                now = tre[now].childs[1];
            else
```

```
                now = tre[now].childs[0];
        }
    }
} S;
```

时间分治线段树

- 内含可回滚并查集和并查集判二分图

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int,int> PII;
const int maxn=1e5+7;
struct Edge{
    int u,v;
};
vector<Edge> seg[maxn<<2];
#define lson o<<1
#define rson o<<1|1
#define Lson L,mid,lson
#define Rson mid+1,R,rson
int n,m,T;
int fa[maxn],col[maxn],sz[maxn];
void init()
{
    for(int i=0;i<=n;i++) fa[i]=i,sz[i]=1,col[i]=0;
}
int Find(int x)
{
    return fa[x]==x?x:Find(fa[x]);
}
int getval(int x)
{
    int ret=0;
    while(fa[x]!=x) ret^=col[x],x=fa[x];
    return ret;
}
void update(int l,int r,Edge e,int L=1,int R=T,int o=1)
{
    if(l<=L&&r>=R)
    {
        seg[o].push_back(e);
        return;
    }
    int mid=L+R>>1;
    if(l<=mid)
        update(l,r,e,Lson);
    if(r>mid) update(l,r,e,Rson);
}
void solve(int L=1,int R=T,int o=1,bool ok=0)
{
    int mid=L+R>>1;
```

```
    if(ok){
        if(L==R) puts(ok?"No":"Yes");
        else
        {
            solve(Lson,1);
            solve(Rson,1);
        }
    }
    else{
        vector<int> cur;
        //insert
        for(int i=0;i<seg[o].size();i++)
        {
            Edge e=seg[o][i];
            int u=e.u,v=e.v;
            int colu=getval(u),colv=getval(v);
            u=Find(u),v=Find(v);
            if(u==v&&colu==colv)
                ok=1;
            else{
                if(sz[u]>sz[v]) swap(u,v);
                sz[v]+=sz[u];
                fa[u]=v;
                col[u]=colu^colv^1;
                cur.push_back(u);
            }
        }
        if(L==R)
            puts(ok?"No":"Yes");
        else solve(Lson,ok),solve(Rson,ok);
        //deleta
        for(int i=cur.size()-1;i>=0;i--)
        {
            int u=cur[i];
            sz[fa[u]]-=sz[u],fa[u]=u,col[u]=0;
        }
    }
}
int main()
{
    scanf("%d%d%d",&n,&m,&T);
    init();
    for(int i=0,u,v,s,e;i<m;i++)
    {
        scanf("%d%d%d%d",&u,&v,&s,&e);
        s++;
        if(s<=e)
            update(s,e,Edge{u,v});
    }
    solve();
}
```

柯朵莉树

```
struct node{
    int l,r;
    mutable ll val;
    bool operator<(const node &a)const{
        return r<a.r;
    }
};
set<node> st;
void split(int p)
{
    auto it = st.lower_bound({p, p, 0});
    if(it -> l == p) return;
    int l = it -> l, r = it -> r; ll val = it -> val;
    st.erase(it);
    st.insert({l, p-1, val});
    st.insert({p, r, val});
}
void update(ll l, ll r, int v)
{
    split(l), split(r + 1);
    auto cur=st.lower_bound({l, l, 0});
    while(cur->r <= r)
    {
        auto tmp = cur;
        cur++;
        st.erase(tmp);
    }
    st.insert({l, r, v});
}
```

K-D Tree

- 最近/远点对

```
const int maxn=5e5+7;
const int inf=0x3f3f3f3f;
int cur,ans,root;

struct P
{
    int mn[2],mx[2],d[2],lch,rch;
    int& operator[](int x) {return d[x];}
    friend bool operator<(P x,P y) {return x[cur]<y[cur];}
    friend int dis(P x,P y) {return abs(x[0]-y[0])+abs(x[1]-y[1]);}
}p[maxn];

struct kdtree
{
    P t[maxn],T;
    int ans;
    void update(int k)
```

```
    {
        int l=t[k].lch,r=t[k].rch;
        for (int i=0;i<2;i++)
        {
            t[k].mn[i]=t[k].mx[i]=t[k][i];
            if (l) t[k].mn[i]=min(t[k].mn[i],t[l].mn[i]);
            if (r) t[k].mn[i]=min(t[k].mn[i],t[r].mn[i]);
            if (l) t[k].mx[i]=max(t[k].mx[i],t[l].mx[i]);
            if (r) t[k].mx[i]=max(t[k].mx[i],t[r].mx[i]);
        }
    }
    int build(int l,int r,int now)
    {
        cur=now;
        int mid=(l+r)/2;
        nth_element(p+l,p+mid,p+r+1);
        t[mid]=p[mid];
        for (int i=0;i<2;i++) t[mid].mx[i]=t[mid].mn[i]=t[mid][i];
        if (l<mid) t[mid].lch=build(l,mid-1,now^1);
        if (r>mid) t[mid].rch=build(mid+1,r,now^1);
        update(mid);
        return mid;
    }
    int getmn(P x)
    {
        int ans=0;
        for (int i=0;i<2;i++)
        {
            ans+=max(T[i]-x.mx[i],0);
            ans+=max(x.mn[i]-T[i],0);
        }
        return ans;
    }
    int getmx(P x)
    {
        int ans=0;
        for (int i=0;i<2;i++) ans+=max(abs(T[i]-x.mn[i]),abs(T[i]-x.mx[i]));
        return ans;
    }
    void querymx(int k)
    {
        ans=max(ans,dis(t[k],T));
        int l=t[k].lch,r=t[k].rch,dl=-inf,dr=-inf;
        if (l) dl=getmx(t[l]);
        if (r) dr=getmx(t[r]);
        if (dl>dr)
        {
            if (dl>ans) querymx(l);
            if (dr>ans) querymx(r);
        }
        else
        {
```

````
            if (dr>ans) querymx(r);
            if (dl>ans) querymx(l);
        }
    }
    void querymn(int k)
    {
        if (dis(t[k],T)) ans=min(ans,dis(t[k],T));
        int l=t[k].lch,r=t[k].rch,dl=inf,dr=inf;
        if (l) dl=getmn(t[l]);
        if (r) dr=getmn(t[r]);
        if (dl<dr)
        {
            if (dl<ans) querymn(l);
            if (dr<ans) querymn(r);
        }
        else
        {
            if (dr<ans) querymn(r);
            if (dl<ans) querymn(l);
        }
    }
    int query(int f,int x,int y)
    {
        T[0]=x;T[1]=y;
        if (f==0) ans=-inf,querymx(root);
        else ans=inf,querymn(root);
        return ans;
    }
}kd;
```\newpage

## 二维几何 点和向量

```c++
#include <bits/stdc++.h>
using namespace std;
#define mp make_pair
#define fi first
#define se second
#define pb push_back
typedef double db;
const db eps = 1e-6;
const db pi = acos(-1.0);
int sign(db k)
{
    if (k > eps)
        return 1;
    else if (k < -eps)
        return -1;
    return 0;
}
int cmp(db k1, db k2) { return sign(k1 - k2); }
````

```cpp
int inmid(db k1, db k2, db k3) { return sign(k1 - k3) * sign(k2 - k3) <= 0; } // k3 在 [k1,k2] 内
struct point
{
    db x, y;
    point operator+(const point &k1) const { return (point){k1.x + x, k1.y + y}; }
    point operator-(const point &k1) const { return (point){x - k1.x, y - k1.y}; }
    point operator*(db k1) const { return (point){x * k1, y * k1}; }
    point operator/(db k1) const { return (point){x / k1, y / k1}; }
    int operator==(const point &k1) const { return cmp(x, k1.x) == 0 && cmp(y, k1.y) == 0; }
    // 逆时针旋转
    point turn(db k1) { return (point){x * cos(k1) - y * sin(k1), x * sin(k1) + y * cos(k1)}; }
    point turn90() { return (point){-y, x}; }
    bool operator<(const point k1) const    //x 为第一关键词 y 为第二关键词
    {
        int a = cmp(x, k1.x);
        if (a == -1)
            return 1;
        else if (a == 1)
            return 0;
        else
            return cmp(y, k1.y) == -1;
    }
    db abs() { return sqrt(x * x + y * y); }
    db abs2() { return x * x + y * y; }
    db dis(point k1) { return ((*this) - k1).abs(); }
    point unit()
    {
        db w = abs();
        return (point){x / w, y / w};
    }
    void scan()
    {
        double k1, k2;
        scanf("%lf%lf", &k1, &k2);
        x = k1;
        y = k2;
    }
    void print() { printf("%.11lf %.11lf\n", x, y); }
    db getw() { return atan2(y, x); }
    point getdel()
    {
        if (sign(x) == -1 || (sign(x) == 0 && sign(y) == -1))
            return (*this) * (-1);
        else
            return (*this);
    }
    int getP() const { return sign(y) == 1 || (sign(y) == 0 && sign(x) == -1); }
};
```

```
int inmid(point k1, point k2, point k3) { return inmid(k1.x, k2.x, k3.x) && inmid(k1.y, k2.y, k3.y); }
db cross(point k1, point k2) { return k1.x * k2.y - k1.y * k2.x; }
db dot(point k1, point k2) { return k1.x * k2.x + k1.y * k2.y; }
db rad(point k1, point k2) { return atan2(cross(k1, k2), dot(k1, k2)); }
// -pi -> pi
int compareangle(point k1, point k2)
{
    return k1.getP() < k2.getP() || (k1.getP() == k2.getP() && sign(cross(k1, k2)) > 0);
}
point proj(point k1, point k2, point q)
{ // q 到直线 k1,k2 的投影
    point k = k2 - k1;
    return k1 + k * (dot(q - k1, k) / k.abs2());
}
point reflect(point k1, point k2, point q) { return proj(k1, k2, q) * 2 - q; }  q 对于直线 k1,k2 的对称点
int clockwise(point k1, point k2, point k3)
{ // k1 k2 k3 逆时针 1 顺时针 -1 否则 0
    return sign(cross(k2 - k1, k3 - k1));
}
int checkLL(point k1, point k2, point k3, point k4)
{ // 求直线 (L) 线段 (S)k1,k2 和 k3,k4 的交点
    return cmp(cross(k3 - k1, k4 - k1), cross(k3 - k2, k4 - k2)) != 0;
}
point getLL(point k1, point k2, point k3, point k4)
{
    db w1 = cross(k1 - k3, k4 - k3), w2 = cross(k4 - k3, k2 - k3);
    return (k1 * w2 + k2 * w1) / (w1 + w2);
}
int 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;
}
int checkSS(point k1, point k2, point k3, point k4)
{ // 非规范相交 <= 0 ; 规范相交 < 0
    return intersect(k1.x, k2.x, k3.x, k4.x) && intersect(k1.y, k2.y, k3.y, k4.y) &&
           sign(cross(k3 - k1, k4 - k1)) * sign(cross(k3 - k2, k4 - k2)) <= 0 &&
           sign(cross(k1 - k3, k2 - k3)) * sign(cross(k1 - k4, k2 - k4)) <= 0;
}
db disSP(point k1, point k2, point q)
{ // 点 ( q ) 到线段 ( k1 , k2 ) 距离
    point k3 = proj(k1, k2, q);
    if (inmid(k1, k2, k3))
        return q.dis(k3);
    else
        return min(q.dis(k1), q.dis(k2));
```

```
}
db disSS(point k1, point k2, point k3, point k4)
{
    if (checkSS(k1, k2, k3, k4))
        return 0;
    else
        return min(min(disSP(k1, k2, k3), disSP(k1, k2, k4)), min(disSP(k3, k4, k1), disSP(k3, k4, k2)));
}
int onS(point k1, point k2, point q) //点 q 在点 k1,k2 之间
{
    return inmid(k1, k2, q) && sign(cross(k1 - q, k2 - k1)) == 0;
}
```

多边形

```
db area(vector<point> A)
{ // 多边形用 vector<point> 表示 , 逆时针
    db ans = 0;
    for (int i = 0; i < A.size(); i++)
        ans += cross(A[i], A[(i + 1) % A.size()]);
    return ans / 2;
}
int checkconvex(vector<point> A)
{ // 逆时针
    int n = A.size();
    A.push_back(A[0]);
    A.push_back(A[1]);
    for (int i = 0; i < n; i++)
        if (sign(cross(A[i + 1] - A[i], A[i + 2] - A[i])) == -1)
            return 0;
    return 1;
}
int contain(vector<point> A, point q)
{ // 2 内部 1 边界 0 外部
    int pd = 0;
    A.push_back(A[0]);
    for (int i = 1; i < A.size(); i++)
    {
        point u = A[i - 1], v = A[i];
        if (onS(u, v, q))
            return 1;
        if (cmp(u.y, v.y) > 0)
            swap(u, v);
        if (cmp(u.y, q.y) >= 0 || cmp(v.y, q.y) < 0)
            continue;
        if (sign(cross(u - v, q - v)) < 0)
            pd ^= 1;
    }
    return pd << 1;
}
vector<point> ConvexHull(vector<point> A, int flag = 1) //凸包
```

```
{                                                          // flag=0 不严格 flag=1 严格
    int n = A.size();
    vector<point> ans(n * 2);
    sort(A.begin(), A.end());
    int now = -1;
    for (int i = 0; i < A.size(); i++)
    {
        while (now > 0 && sign(cross(ans[now] - ans[now - 1], A[i] - ans[now - 1])) <
         ↪  flag)
            now--;
        ans[++now] = A[i];
    }
    int pre = now;
    for (int i = n - 2; i >= 0; i--)
    {
        while (now > pre && sign(cross(ans[now] - ans[now - 1], A[i] - ans[now - 1]))
         ↪  < flag)
            now--;
        ans[++now] = A[i];
    }
    ans.resize(now);
    return ans;
}
db convexDiameter(vector<point> A)
{ // 凸包直径
    int now = 0, n = A.size();
    db ans = 0;
    for (int i = 0; i < A.size(); i++)
    {
        now = max(now, i);
        while (1)
        {
            db k1 = A[i].dis(A[now % n]), k2 = A[i].dis(A[(now + 1) % n]);
            ans = max(ans, max(k1, k2));
            if (k2 > k1)
                now++;
            else
                break;
        }
    }
    return ans;
}
```

圆和线段

```
struct circle
{
    point o;
    db r;
    void scan()
    {
        o.scan();
```

```
        scanf("%lf", &r);
    }
    int inside(point k) { return cmp(r, o.dis(k)); }
};
struct line
{
    // p[0]->p[1]
    point p[2];
    line(point k1, point k2)
    {
        p[0] = k1;
        p[1] = k2;
    }
    point &operator[](int k) { return p[k]; }
    int include(point k) { return sign(cross(p[1] - p[0], k - p[0])) > 0; }
    point dir() { return p[1] - p[0]; }
    line push()
    { // 向外 ( 左手边 ) 平移 eps
        const db eps = 1e-6;
        point delta = (p[1] - p[0]).turn90().unit() * eps;
        return {p[0] - delta, p[1] - delta};
    }
};
point getLL(line k1, line k2) { return getLL(k1[0], k1[1], k2[0], k2[1]); }
int parallel(line k1, line k2) { return sign(cross(k1.dir(), k2.dir())) == 0; }
int sameDir(line k1, line k2) { return parallel(k1, k2) && sign(dot(k1.dir(),
 ↪  k2.dir())) == 1; }
int operator<(line k1, line k2)
{
    if (sameDir(k1, k2))
        return k2.include(k1[0]);
    return compareangle(k1.dir(), k2.dir());
}
int checkpos(line k1, line k2, line k3) { return k3.include(getLL(k1, k2)); }

int checkposCC(circle k1, circle k2)
{ // 返回两个圆的公切线数量
    if (cmp(k1.r, k2.r) == -1)
        swap(k1, k2);
    db dis = k1.o.dis(k2.o);
    int w1 = cmp(dis, k1.r + k2.r), w2 = cmp(dis, k1.r - k2.r);
    if (w1 > 0)
        return 4;
    else if (w1 == 0)
        return 3;
    else if (w2 > 0)
        return 2;
    else if (w2 == 0)
        return 1;
    else
        return 0;
}
```

```cpp
vector<point> getCL(circle k1, point k2, point k3)
{ // 沿着 k2->k3 方向给出 , 相切给出两个
    point k = proj(k2, k3, k1.o);
    db d = k1.r * k1.r - (k - k1.o).abs2();
    if (sign(d) == -1)
        return {};
    point del = (k3 - k2).unit() * sqrt(max((db)0.0, d));
    return {k - del, k + del};
}
vector<point> getCC(circle k1, circle k2)
{ // 沿圆 k1 逆时针给出 , 相切给出两个
    int pd = checkposCC(k1, k2);
    if (pd == 0 || pd == 4)
        return {};
    db a = (k2.o - k1.o).abs2(), cosA = (k1.r * k1.r + a - k2.r * k2.r) / (2 * k1.r * sqrt(max(a, (db)0.0)));
    db b = k1.r * cosA, c = sqrt(max((db)0.0, k1.r * k1.r - b * b));
    point k = (k2.o - k1.o).unit(), m = k1.o + k * b, del = k.turn90() * c;
    return {m - del, m + del};
}
vector<point> TangentCP(circle k1, point k2)
{ // 沿圆 k1 逆时针给出
    db a = (k2 - k1.o).abs(), b = k1.r * k1.r / a, c = sqrt(max((db)0.0, k1.r * k1.r - b * b));
    point k = (k2 - k1.o).unit(), m = k1.o + k * b, del = k.turn90() * c;
    return {m - del, m + del};
}
vector<line> TangentoutCC(circle k1, circle k2)
{
    int pd = checkposCC(k1, k2);
    if (pd == 0)
        return {};
    if (pd == 1)
    {
        point k = getCC(k1, k2)[0];
        return {(line){k, k}};
    }
    if (cmp(k1.r, k2.r) == 0)
    {
        point del = (k2.o - k1.o).unit().turn90().getdel();
        return {(line){k1.o - del * k1.r, k2.o - del * k2.r}, (line){k1.o + del * k1.r, k2.o + del * k2.r}};
    }
    else
    {
        point p = (k2.o * k1.r - k1.o * k2.r) / (k1.r - k2.r);
        vector<point> A = TangentCP(k1, p), B = TangentCP(k2, p);
        vector<line> ans;
        for (int i = 0; i < A.size(); i++)
            ans.push_back((line){A[i], B[i]});
        return ans;
    }
```

```
}
vector<line> TangentinCC(circle k1, circle k2)
{
    int pd = checkposCC(k1, k2);
    if (pd <= 2)
        return {};
    if (pd == 3)
    {
        point k = getCC(k1, k2)[0];
        return {(line){k, k}};
    }
    point p = (k2.o * k1.r + k1.o * k2.r) / (k1.r + k2.r);
    vector<point> A = TangentCP(k1, p), B = TangentCP(k2, p);
    vector<line> ans;
    for (int i = 0; i < A.size(); i++)
        ans.push_back((line){A[i], B[i]});
    return ans;
}
vector<line> TangentCC(circle k1, circle k2)
{
    int flag = 0;
    if (k1.r < k2.r)
        swap(k1, k2), flag = 1;
    vector<line> A = TangentoutCC(k1, k2), B = TangentinCC(k1, k2);
    for (line k : B)
        A.push_back(k);
    if (flag)
        for (line &k : A)
            swap(k[0], k[1]);
    return A;
}
db getarea(circle k1, point k2, point k3)
{
    // 圆 k1 与三角形 k2 k3 k1.o 的有向面积交
    point k = k1.o;
    k1.o = k1.o - k;
    k2 = k2 - k;
    k3 = k3 - k;
    int pd1 = k1.inside(k2), pd2 = k1.inside(k3);
    vector<point> A = getCL(k1, k2, k3);
    if (pd1 >= 0)
    {
        if (pd2 >= 0)
            return cross(k2, k3) / 2;
        return k1.r * k1.r * rad(A[1], k3) / 2 + cross(k2, A[1]) / 2;
    }
    else if (pd2 >= 0)
    {
        return k1.r * k1.r * rad(k2, A[0]) / 2 + cross(A[0], k3) / 2;
    }
    else
    {
```

```
        int pd = cmp(k1.r, disSP(k2, k3, k1.o));
        if (pd <= 0)
            return k1.r * k1.r * rad(k2, k3) / 2;
        return cross(A[0], A[1]) / 2 + k1.r * k1.r * (rad(k2, A[0]) + rad(A[1], k3)) / 2;
    }
}
circle getcircle(point k1, point k2, point k3)
{
    // 三点求圆
    db a1 = k2.x - k1.x, b1 = k2.y - k1.y, c1 = (a1 * a1 + b1 * b1) / 2;
    db a2 = k3.x - k1.x, b2 = k3.y - k1.y, c2 = (a2 * a2 + b2 * b2) / 2;
    db d = a1 * b2 - a2 * b1;
    point o = (point){k1.x + (c1 * b2 - c2 * b1) / d, k1.y + (a1 * c2 - a2 * c1) / d};
    return (circle){o, k1.dis(o)};
}
circle getScircle(vector<point> A)
{
    //随机增量法 最小圆覆盖
    random_shuffle(A.begin(), A.end());
    circle ans = (circle){A[0], 0};
    for (int i = 1; i < A.size(); i++)
        if (ans.inside(A[i]) == -1)
        {
            ans = (circle){A[i], 0};
            for (int j = 0; j < i; j++)
                if (ans.inside(A[j]) == -1)
                {
                    ans.o = (A[i] + A[j]) / 2;
                    ans.r = ans.o.dis(A[i]);
                    for (int k = 0; k < j; k++)
                        if (ans.inside(A[k]) == -1)
                            ans = getcircle(A[i], A[j], A[k]);
                }
        }
    return ans;
}
```

其他

```
vector<line> getHL(vector<line> &L)
{ // 求半平面交 , 半平面是逆时针方向 , 输出按照逆时针
    sort(L.begin(), L.end());
    deque<line> q;
    for (int i = 0; i < (int)L.size(); i++)
    {
        if (i && sameDir(L[i], L[i - 1]))
            continue;
        while (q.size() > 1 && !checkpos(q[q.size() - 2], q[q.size() - 1], L[i]))
            q.pop_back();
        while (q.size() > 1 && !checkpos(q[1], q[0], L[i]))
```

```
            q.pop_front();
        q.push_back(L[i]);
    }
    while (q.size() > 2 && !checkpos(q[q.size() - 2], q[q.size() - 1], q[0]))
        q.pop_back();
    while (q.size() > 2 && !checkpos(q[1], q[0], q[q.size() - 1]))
        q.pop_front();
    vector<line> ans;
    for (int i = 0; i < q.size(); i++)
        ans.push_back(q[i]);
    return ans;
}
db closepoint(vector<point> &A, int l, int r)
{ // 最近点对 , 先要按照 x 坐标排序
    if (r - l <= 5)
    {
        db ans = 1e20;
        for (int i = l; i <= r; i++)
            for (int j = i + 1; j <= r; j++)
                ans = min(ans, A[i].dis(A[j]));
        return ans;
    }
    int mid = ((l + r)) >> 1;
    db ans = min(closepoint(A, l, mid), closepoint(A, mid + 1, r));
    vector<point> B;
    for (int i = l; i <= r; i++)
        if (abs(A[i].x - A[mid].x) <= ans)
            B.push_back(A[i]);
    sort(B.begin(), B.end(), [](point k1, point k2) { return k1.y < k2.y; });
    for (int i = 0; i < B.size(); i++)
        for (int j = i + 1; j < B.size() && B[j].y - B[i].y < ans; j++)
            ans = min(ans, B[i].dis(B[j]));
    return ans;
}
vector<point> convexcut(vector<point> A, point k1, point k2)
{
    // 保留 k1,k2,p 逆时针的所有点
    int n = A.size();
    A.push_back(A[0]);
    vector<point> ans;
    for (int i = 0; i < n; i++)
    {
        int w1 = clockwise(k1, k2, A[i]), w2 = clockwise(k1, k2, A[i + 1]);
        if (w1 >= 0)
            ans.push_back(A[i]);
        if (w1 * w2 < 0)
            ans.push_back(getLL(k1, k2, A[i], A[i + 1]));
    }
    return ans;
}
int checkPoS(vector<point> A, point k1, point k2)
{
```

```
    // 多边形 A 和直线 ( 线段 )k1->k2 严格相交 , 注释部分为线段
    struct ins
    {
        point m, u, v;
        int operator<(const ins &k) const { return m < k.m; }
    };
    vector<ins> B;
    //if (contain(A,k1)==2||contain(A,k2)==2) return 1;
    vector<point> poly = A;
    A.push_back(A[0]);
    for (int i = 1; i < A.size(); i++)
        if (checkLL(A[i - 1], A[i], k1, k2))
        {
            point m = getLL(A[i - 1], A[i], k1, k2);
            if (inmid(A[i - 1], A[i], m) /*&&inmid(k1,k2,m)*/)
                B.push_back((ins){m, A[i - 1], A[i]});
        }
    if (B.size() == 0)
        return 0;
    sort(B.begin(), B.end());
    int now = 1;
    while (now < B.size() && B[now].m == B[0].m)
        now++;
    if (now == B.size())
        return 0;
    int flag = contain(poly, (B[0].m + B[now].m) / 2);
    if (flag == 2)
        return 1;
    point d = B[now].m - B[0].m;
    for (int i = now; i < B.size(); i++)
    {
        if (!(B[i].m == B[i - 1].m) && flag == 2)
            return 1;
        int tag = sign(cross(B[i].v - B[i].u, B[i].m + d - B[i].u));
        if (B[i].m == B[i].u || B[i].m == B[i].v)
            flag += tag;
        else
            flag += tag * 2;
    }
    //return 0;
    return flag == 2;
}
```

图论

树链剖分

- 维护点权，边权要下放

```
int fa[maxn], dep[maxn], maxson[maxn], son[maxn]; //dfs 数组
int top[maxn], dfn[maxn], tot;                    //link 数组
int dfs(int u)
{
    int ret = 0;
    for (int i = head[u]; i != -1; i = edge[i].nxt)
    {
        int v = edge[i].to;
        if (v == fa[u])
            continue;
        fa[v] = u;
        dep[v] = dep[u] + 1;
        int sz = dfs(v);
        ret += sz;
        if (sz > maxson[u])
        {
            maxson[u] = sz;
            son[u] = v;
        }
    }
    return ret + 1;
}
void link(int u, int t)
{
    dfn[u] = ++tot;
    top[u] = t;
    if (son[u])
        link(son[u], t);
    for (int i = head[u]; i != -1; i = edge[i].nxt)
    {
        int v = edge[i].to;
        if (v == fa[u] || v == son[u])
            continue;
        link(v, v);
    }
}
void hld()
{
    dfs(1);
    link(1, 1);
}
```

边双联通分量

- 将无向图缩成一棵树
- 有重边需要做一些改动
- 稍微改一下就是强联通缩点了

```cpp
int dfn[maxn], low[maxn], bel[maxn];
int n, m;
int ti, scc; //时间戳与联通分量计数
stack<int> st;
void dfs(int u, int fa)
{
    dfn[u] = low[u] = ++ti;
    st.push(u);
    for (int i = head[u]; i != -1; i = edge[i].nxt)
    {
        int v = edge[i].to;
        if (v == fa)
            continue;
        if (!dfn[v])
        {
            dfs(v, u);
            low[u] = min(low[u], low[v]);
        }
        else if (!bel[v])
            low[u] = min(low[u], dfn[v]);
    }
    if (dfn[u] == low[u])
    {
        scc++;
        while (1)
        {
            int t = st.top();
            st.pop();
            bel[t] = scc;
            if (u == t)
                break;
        }
    }
}
void DCC()
{
    for (int i = 1; i <= n; i++)
        if (!dfn[i])
            dfs(i, -1);
    for (int i = 0; i < cur; i++) //遍历所有边建图
    {
        int u = edge[i].from, v = edge[i].to;
        if (bel[u] != bel[v])
            addedge2(bel[u], bel[v]);
    }
}
```

虚树

- 注意每次清空虚树图
- 根节点必定是关键点

```cpp
const int maxn = 5e5 + 7;
```

```cpp
int ti; //时间戳
int ts[maxn];
int depth[maxn];
int far[maxn][22];
int dfn[maxn];
void dfs(int u, int fa = -1)
{
    dfn[u] = ++ti; //dfs 序
    ts[ti] = u;    //括号序列时间戳映射
    for (int i = head[u]; i != -1; i = edge[i].nxt)
    {
        int v = edge[i].to;
        if (v == fa)
            continue;
        depth[v] = depth[u] + 1;
        far[v][0] = u; //可能会用到的倍增数组
        dfs(v, u);
        ts[++ti] = u;
    }
}
int ST[maxn][22]; //LCA 转 RMQ 用 ST 表求
bool cmp(int x, int y) { return depth[x] < depth[y]; }
void RMQ()
{
    for (int i = 1; i <= ti; i++)
        ST[i][0] = ts[i];
    for (int j = 1; (1 << j) <= ti; j++)
    {
        for (int i = 1; i + (1 << (j - 1)) - 1 <= ti; i++)
        {
            if (cmp(ST[i][j - 1], ST[i + (1 << (j - 1))][j - 1]))
                ST[i][j] = ST[i][j - 1];
            else
                ST[i][j] = ST[i + (1 << (j - 1))][j - 1];
        }
    }
}
int LCA(int u, int v)
{
    int l = dfn[u], r = dfn[v];
    if (l > r)
        swap(l, r);
    int k = 31 - __builtin_clz(r - l + 1);
    if (cmp(ST[l][k], ST[r - (1 << k) + 1][k]))
        return ST[l][k];
    return ST[r - (1 << k) + 1][k];
}
void Virtual(vector<int> &all) //关键点
{
    int st[maxn]; //栈模拟访问
    int top = 0;
    for (auto &u : all)
```

```
    {
        if (top == 0)
            st[++top] = u;
        else
        {
            int lca = LCA(st[top], u);
            while (top > 1 && dfn[st[top - 1]] >= dfn[lca])
            { //栈中至少有两个元素，则开始向上连边
                access(st[top],st[top-1]));
                top--;
            }
            if (lca != st[top]) //最后将 lca 也放进去
            {
                access(st[top], lca);
                st[top] = lca;
            }
            st[++top] = u;
        }
    }
    while (top > 1) //所有元素出栈
    {
        access(st[top], st[top - 1]);
        --top;
    }
}
```

网络流

DINIC

```
const int maxn = 1e3 + 7;
const int INF = 0x3f3f3f3f;
struct Edge
{
    int from, to, cap, flow;
};
struct Dinic
{
    int n, m, s, t;
    vector<Edge> edges;
    vector<int> G[maxn];
    bool vis[maxn];
    int d[maxn];
    int cur[maxn];

    void AddEdge(int from, int to, int cap, int c = 0)
    {
        edges.push_back(Edge{from, to, cap, 0});
        edges.push_back(Edge{to, from, c, 0});
        m = edges.size();
        G[from].push_back(m - 2);
        G[to].push_back(m - 1);
    }
```

```
    bool BFS()
    {
        memset(vis, 0, sizeof(vis));
        queue<int> q;
        q.push(s);
        d[s] = 0;
        vis[s] = 1;
        while (!q.empty())
        {
            int u = q.front();
            q.pop();
            for (int i = 0; i < G[u].size(); i++)
            {
                Edge &e = edges[G[u][i]];
                if (!vis[e.to] && e.cap > e.flow)
                {
                    vis[e.to] = 1;
                    d[e.to] = d[u] + 1;
                    q.push(e.to);
                }
            }
        }
        return vis[t];
    }
    int DFS(int u, int dist)
    {
        if (u == t || dist == 0)
            return dist;
        int flow = 0, f;
        for (int &i = cur[u]; i < G[u].size(); i++)
        {
            Edge &e = edges[G[u][i]];
            if (d[u] + 1 == d[e.to] && (f = DFS(e.to, min(dist, e.cap - e.flow))) >
             ↪  0)
            {
                e.flow += f;
                edges[G[u][i] ^ 1].flow -= f;
                flow += f;
                dist -= f;
                if (!dist)
                    break;
            }
        }
        return flow;
    }
    int Maxflow(int s, int t)
    {
        this->s = s;
        this->t = t;
        int flow = 0;
        while (BFS())
```

```
        {
            memset(cur, 0, sizeof(cur));
            flow += DFS(s, INF);
        }
        return flow;
    }
};
```

费用流

```
const int maxn = 1e3 + 7;
const int INF = 0x3f3f3f3f;
struct Edge
{
    int from, to, cap, flow, cost;
};
struct MCMF
{
    int n, m, s, t;
    vector<Edge> edges;
    vector<int> G[maxn];
    int inq[maxn];
    int d[maxn]; //最短路数组
    int p[maxn]; //记录路径
    int a[maxn]; //记录流量
    void init(int n)
    {
        this->n = n;
        for (int i = 0; i < n; i++)
            G[i].clear();
        edges.clear();
    }
    void addedge(int from, int to, int cap, int cost)
    {
        edges.push_back(Edge{from, to, cap, 0, cost});
        edges.push_back(Edge{to, from, 0, 0, -cost});
        m = edges.size();
        G[from].push_back(m - 2);
        G[to].push_back(m - 1);
    }
    bool spfa(int s, int t, int &flow, int &cost)
    {
        for (int i = 0; i < n; i++)
            d[i] = INF;
        memset(inq, 0, sizeof(inq));
        d[s] = 0;
        inq[s] = 1;
        p[s] = 0;
        a[s] = INF;
        queue<int> q;
        q.push(s);
        while (!q.empty())
```

```
        {
            int u = q.front();
            q.pop();
            inq[u] = 0;
            for (int i = 0; i < G[u].size(); i++)
            {
                Edge &e = edges[G[u][i]];
                if (e.cap > e.flow && d[e.to] > d[u] + e.cost)
                {
                    d[e.to] = d[u] + e.cost;             //松弛
                    p[e.to] = G[u][i];                   //记录上一个点
                    a[e.to] = min(a[u], e.cap - e.flow); //流量控制
                    if (!inq[e.to])
                    {
                        q.push(e.to);
                        inq[e.to] = 1;
                    }
                }
            }
        }
        if (d[t] == INF)
            return false; //不存在最短路
        flow += a[t];
        cost += d[t] * a[t];
        int u = t;
        while (u != s)
        {
            edges[p[u]].flow += a[t];
            edges[p[u] ^ 1].flow -= a[t];
            u = edges[p[u]].from;
        }
        return true;
    }
    int Mincost(int s, int t)
    {
        int flow = 0, cost = 0;
        while (spfa(s, t, flow, cost))
            ;
        return cost;
    }
};
```

最小树形图

```
struct Edge//边的权和顶点
{
    int u, v;
    ll w;
}edge[maxn * maxn];
int pre[maxn], id[maxn], vis[maxn], pos;
ll in[maxn];//存最小入边权,pre[v] 为该边的起点

```

```
ll Directed_MST(int root, int V, int E)
{
    ll ret = 0;//存最小树形图总权值
    while(true)
    {
        int i;
        //1. 找每个节点的最小入边
        for( i = 0; i < V; i++)
            in[i] = INF;//初始化为无穷大
        for( i = 0; i < E; i++)//遍历每条边
        {
            int u = edge[i].u;
            int v = edge[i].v;
            if(edge[i].w < in[v] && u != v)//说明顶点 v 有条权值较小的入边  记录之
            {
                pre[v] = u;//节点 u 指向 v
                in[v] = edge[i].w;//最小入边
                if(u == root)//这个点就是实际的起点
                    pos = i;
            }
        }
        for( i = 0; i < V; i++)//判断是否存在最小树形图
        {
            if(i == root)
                continue;
            if(in[i] == INF)
                return -1;//除了根以外有点没有入边，则根无法到达它  说明它是独立的点 一定不
                  ↪ 能构成树形图
        }
        //2. 找环
        int cnt = 0;//记录环数
        memset(id, -1, sizeof(id));
        memset(vis, -1, sizeof(vis));
        in[root] = 0;
        for( i = 0; i < V; i++) //标记每个环
        {
            ret += in[i];//记录权值
            int v = i;
            while(vis[v] != i && id[v] == -1 && v != root)
            {
                vis[v] = i;
                v = pre[v];
            }
            if(v != root && id[v] == -1)
            {
                for(int u = pre[v]; u != v; u = pre[u])
                    id[u] = cnt;//标记节点 u 为第几个环
                id[v] = cnt++;
            }
        }
        if(cnt == 0)
            break; //无环   则 break
```

```
        for( i = 0; i < V; i++)
            if(id[i] == -1)
                id[i] = cnt++;
            //3. 建立新图   缩点，重新标记
            for( i = 0; i < E; i++)
            {
                int u = edge[i].u;
                int v = edge[i].v;
                edge[i].u = id[u];
                edge[i].v = id[v];
                if(id[u] != id[v])
                    edge[i].w -= in[v];
            }
            V = cnt;
            root = id[root];
    }
    return ret;
}
```

树的最大匹配

```
int f[N],g[N];
void dfs(int cur,int fa)
{
    f[cur]=g[cur]=0;
    for(int i=head[cur];~i;i=e[i].next)
    {
        if(e[i].v==fa)
            continue;
        int v=e[i].v;
        dfs(v,cur);
        g[cur]+=max(f[v],g[v]);
    }
    for(int i=head[cur];~i;i=e[i].next)
    {
        if(e[i].v==fa)
            continue;
        int v=e[i].v;
        f[cur]=max(f[cur],g[cur]-max(f[v],g[v])+g[v]+1);
    }
}

int maxmatch()
{
    dfs(1,-1);
    return max(f[1],g[1]);
}
```

支配树

```
vector<int> G[N], rG[N];
vector<int> dt[N];
```

```

namespace tl
{
    int fa[N], idx[N], clk, ridx[N];
    int c[N], best[N], semi[N], idom[N];

    void init(int n)
    {
        clk = 0;
        for(int i=0;i<=n;++i)
        {
            idom[i]=0;
            c[i]=-1;
            idx[i]=0;
            dt[i].clear();
            semi[i] = best[i] = i;
        }
    }
    void dfs(int u)
    {
        idx[u] = ++clk; ridx[clk] = u;
        for (int& v: G[u])
            if (!idx[v])
            {
                fa[v] = u;
                dfs(v);
            }
    }
    int fix(int x)
    {
        if (c[x] == -1)
            return x;
        int &f = c[x], rt = fix(f);
        if (idx[semi[best[x]]] > idx[semi[best[f]]])
            best[x] = best[f];
        return f = rt;
    }
    void go(int rt)
    {
        dfs(rt);
        for(int i=clk;i>=2;--i)
        {
            int x = ridx[i], mn = clk + 1;
            for (int& u: rG[x])
            {
                if (!idx[u])
                    continue; // 可能不能到达所有点
                fix(u);
                mn = min(mn, idx[semi[best[u]]]);
            }
            c[x] = fa[x];
            dt[semi[x] = ridx[mn]].push_back(x);
```

```
            x = ridx[i - 1];
            for (int& u: dt[x])
            {
                fix(u);
                if (semi[best[u]] != x)
                    idom[u] = best[u];
                else
                    idom[u] = x;
            }
            dt[x].clear();
        }
        for(int i=2;i<=clk;++i)
        {
            int u = ridx[i];
            if (idom[u] != semi[u])
                idom[u] = idom[idom[u]];
            dt[idom[u]].push_back(u);
        }
    }
}
```

斯坦纳树

```
const int maxn=35;
struct Edge{
    int to,w;
};
vector<Edge> G[maxn];
int n,m;
PII tar[4];
int st[maxn];
int dp[maxn][1<<8];
int dis[1<<8];
bool inq[maxn];
queue<int> q;
int cur=0;
bool check(int S)
{
    bool ok=1;
    for(int i=0;i<4;i++)
    {
        if(S&st[tar[i].first])
            ok&=((S&st[tar[i].second])!=0);
    }
    return ok;
}
void spfa(int S)
{
    while(!q.empty())
    {
        int u=q.front();
        q.pop();
```

```
        inq[u]=0;
        for(auto e:G[u])
        {
            int v=e.to;
            if(dp[v][S|st[v]]>dp[u][S]+e.w)
            {
                dp[v][S|st[v]]=dp[u][S]+e.w;
                if(inq[v]||(st[v]|S)!=S) continue;
                inq[v]=1;
                q.push(v);
            }
        }
    }
}
void solve()
{
    memset(dp,0x3f,sizeof(dp));//求解斯坦纳树
    for(int i=0;i<n;i++)
    {
        if(st[i])
            dp[i][st[i]]=0;
    }
    for(int S=0;S<(1<<cur);S++)
    {
        memset(inq,0,sizeof(inq));
        for(int i=0;i<n;i++)
        {
            if(st[i]&&(st[i]&S)==0) continue;
            for(int T=(S-1)&S;T;T=(T-1)&S)
                dp[i][S]=min(dp[i][S],dp[i][T|st[i]]+dp[i][(S-T)|st[i]]);
            if(dp[i][S]!=INF)
            {
                inq[i]=1;
                q.push(i);
            }
        }
        spfa(S);
    }
    memset(dis,0x3f,sizeof(dis));//计算答案
    for(int S=0;S<(1<<cur);S++)
    {
        for(int i=0;i<n;i++)
            dis[S]=min(dis[S],dp[i][S]);
    }
    for(int S=0;S<(1<<cur);S++)
    {
        if(!check(S)) continue;//满足要求的状态
        for(int T=(S-1)&S;T;T=(T-1)&S)
        {
            if(!check(T)||!check(S-T)) continue;
            dis[S]=min(dis[S],dis[T]+dis[S-T]);
        }
```

```
    }
    cout<<dis[(1<<cur)-1]<<endl;
}
```

2-SAT

```
const int maxn=8e3+7; //HDU 826ms
struct Twosat   //O(n^2) 得到字典序最小的方案
{
    //编号从 0~2n-1 顺序每两个为一对相斥的
    int n;
    vector<int> g[maxn*2];
    bool mark[maxn*2];
    int s[maxn*2],c;//一个栈
    bool dfs(int x)
    {
        if(mark[x^1])
            return false;
        if(mark[x])
            return true;
        mark[x]=true;
        s[c++]=x;
        for(int to:g[x])
            if(!dfs(to))
                return false;
        return true;
    }
    void init(int n)
    {
        this->n=n;
        for(int i=0;i<n*2;i++)
        {
            g[i].clear();
            mark[i]=0;
        }
    }
    void add(int x,int y)
    {//这个函数随题意变化
        g[x].push_back(y^1);//选了 x 就必须选 y^1
        g[y].push_back(x^1);
    }
    bool solve()
    {
        for(int i=0;i<n*2;i+=2)
            if(!mark[i]&&!mark[i+1])
            {
                c=0;
                if(!dfs(i))
                {
                    while(c>0)
                        mark[s[--c]]=false;
                    if(!dfs(i+1))
```

```
                    return false;
                }
            }
        return true;
    }
}sat;
```

网络流基本建图技巧

- 最大流 = 最小割
- 拆点技巧
- 最小点覆盖 = 最大匹配
- 最大独立集 = 点数-最小覆盖集
- 最小路径覆盖最小路径覆盖 = 原图节点数-最大匹配拆点分出入度变二分图。一开始把每个点都视为一条路径，每次匹配相当于合并两条路径，匹配几次路径数就少了多少。
- 上下界网络流先让每条边有下界那么多的流量，然后再去增大某些弧的流量使流量平衡。具体操作即为平衡与补流。对于入流大于出流的点，由 S 向其建边补流，对于出流大于入流的点，向 T 连边平衡。具体操作时用一个数组统计入流与出流差即可。对于有源汇的情况，加一条由 t 到 s，容量为 INF 的边，改造成循环流即可。
- 最小流首先按照有源汇可行流的方法建模, 但是不要建立 <t,s> 这条弧。然后在这个图上, 跑从附加源 ss 到附加汇 tt 的最大流。这时候再添加弧 <t,s> 再跑从 ss 到 tt 的最大流, 就是原图的最小流。
- 区间 k 覆盖最小费用最大流, [i,j] 区间覆盖 `add_edge(i,j,1,-w)` 点之间连 `add_edge(i,i+1,k,0)`
- 优化建图目前已知的有二进制优化和线段树优化。
- 最大权闭合子图建图：设置超级源汇 S 和所有的正权的点连接权值为点权的边，所有点权为负的点和 T 连接权值为点权绝对值的边。然后如果选择了某个 v 点才可以选 u 点，那么把 u 向 v 连接一条权值为 INF 的边。最大点权和 = 正点权和-最小割

一些公式

- Carley format : $(n+1)^{n-1}, T_{n,k} = kn^{n-k-1}$

杂项

简单随机数

```
inline int rnd()
{
    static int seed=2333;
    return seed=(int)seed*482711LL%2147483647;
}
```

输入输出挂

- 只支持整数
- 支持负数
- 不带空格

```
template <typename T>
bool scan_d(T &num)
{
        char in;bool IsN=false;
        in=getchar();
        if(in==EOF) return false;
        while(in!='-'&&(in<'0'||in>'9')) in=getchar();
        if(in=='-'){ IsN=true;num=0;}
        else num=in-'0';
        while(in=getchar(),in>='0'&&in<='9'){
                num*=10,num+=in-'0';
        }
        if(IsN) num=-num;
        return true;
}
template <typename T>
void o(T p) {
    static int stk[70], tp;
    if (p == 0) { putchar('0'); return; }
    if (p < 0) { p = -p; putchar('-'); }
    while (p) stk[++tp] = p % 10, p /= 10;
    while (tp) putchar(stk[tp--] + '0');
}
```

HDU 专用

```
#define reads(n) FastIO::read(n)
namespace FastIO {
    const int SIZE = 1 << 16;
    char buf[SIZE], obuf[SIZE], str[60];
    int bi = SIZE, bn = SIZE, opt;
    int read(char *s) {
        while (bn) {
            for (; bi < bn && buf[bi] <= ' '; bi++);
            if (bi < bn) break;
            bn = fread(buf, 1, SIZE, stdin);
            bi = 0;
```

```
        }
        int sn = 0;
        while (bn) {
            for (; bi < bn && buf[bi] > ' '; bi++) s[sn++] = buf[bi];
            if (bi < bn) break;
            bn = fread(buf, 1, SIZE, stdin);
            bi = 0;
        }
        s[sn] = 0;
        return sn;
    }
    template<typename T>
    bool read(T& x) {
        int n = read(str), bf;
        if (!n) return 0;
        int i = 0; if (str[i] == '-') bf = -1, i++; else bf = 1;
        for (x = 0; i < n; i++) x = x * 10 + str[i] - '0';
        if (bf < 0) x = -x;
        return 1;
    }
};
```

日期

```
// Routines for performing computations on dates.  In these routines,
// months are exprsesed as integers from 1 to 12, days are expressed
// as integers from 1 to 31, and years are expressed as 4-digit
// integers.

string dayOfWeek[] = {"Mo", "Tu", "We", "Th", "Fr", "Sa", "Su"};

// converts Gregorian date to integer (Julian day number)

int DateToInt (int m, int d, int y){
  return
    1461 * (y + 4800 + (m - 14) / 12) / 4 +
    367 * (m - 2 - (m - 14) / 12 * 12) / 12 -
    3 * ((y + 4900 + (m - 14) / 12) / 100) / 4 +
    d - 32075;
}

// converts integer (Julian day number) to Gregorian date: month/day/year

void IntToDate (int jd, int &m, int &d, int &y){
  int x, n, i, j;

  x = jd + 68569;
  n = 4 * x / 146097;
  x -= (146097 * n + 3) / 4;
  i = (4000 * (x + 1)) / 1461001;
  x -= 1461 * i / 4 - 31;
  j = 80 * x / 2447;
```

```
  d = x - 2447 * j / 80;
  x = j / 11;
  m = j + 2 - 12 * x;
  y = 100 * (n - 49) + i + x;
}

// converts integer (Julian day number) to day of week

string IntToDay (int jd){
  return dayOfWeek[jd % 7];
}
```

子集枚举

```
for (int T = S; T; T = (T - 1) & S)
```

- 枚举大小为 k 的子集

```
template<typename T>
void subset(int k, int n, T&& f) {
    int t = (1 << k) - 1;
    while (t < 1 << n)
    {
        f(t);
        int x = t & -t, y = t + x;
        t = ((t & ~y) / x >> 1) | y;
    }
}
```

大数

```
const int DLEN=4,MAXSIZE=100,MAXN=9999;
class BigNum
{
private:
    int a[MAXSIZE];    //可以控制大数的位数
    int len;       //大数长度
public:
    BigNum(){ len = 1;memset(a,0,sizeof(a)); }   //构造函数
    BigNum(const int);       //将一个 int 类型的变量转化为大数
    BigNum(const char*);     //将一个字符串类型的变量转化为大数
    BigNum(const BigNum &);  //拷贝构造函数
    BigNum &operator=(const BigNum &);   //重载赋值运算符，大数之间进行赋值运算

    friend istream& operator>>(istream&,  BigNum&);   //重载输入运算符
    friend ostream& operator<<(ostream&,  BigNum&);   //重载输出运算符

    BigNum operator+(const BigNum &) const;   //重载加法运算符，两个大数之间的相加运算
    BigNum operator-(const BigNum &) const;   //重载减法运算符，两个大数之间的相减运算
    BigNum operator*(const BigNum &) const;   //重载乘法运算符，两个大数之间的相乘运算
    BigNum operator/(const int   &) const;    //重载除法运算符，大数对一个整数进行相除运算

    BigNum operator^(const int  &) const;    //大数的 n 次方运算
```

```
    int    operator%(const int  &) const;    //大数对一个 int 类型的变量进行取模运算
    bool   operator>(const BigNum & T)const;   //大数和另一个大数的大小比较
    bool   operator>(const int & t)const;      //大数和一个 int 类型的变量的大小比较

    void print();       //输出大数
};
BigNum::BigNum(const int b)     //将一个 int 类型的变量转化为大数
{
    int c,d = b;
    len = 0;
    memset(a,0,sizeof(a));
    while(d > MAXN)
    {
        c = d - (d / (MAXN + 1)) * (MAXN + 1);
        d = d / (MAXN + 1);
        a[len++] = c;
    }
    a[len++] = d;
}
BigNum::BigNum(const char*s)     //将一个字符串类型的变量转化为大数
{
    int t,k,index,l,i;
    memset(a,0,sizeof(a));
    l=strlen(s);
    len=l/DLEN;
    if(l%DLEN)
        len++;
    index=0;
    for(i=l-1;i>=0;i-=DLEN)
    {
        t=0;
        k=i-DLEN+1;
        if(k<0)
            k=0;
        for(int j=k;j<=i;j++)
            t=t*10+s[j]-'0';
        a[index++]=t;
    }
}
BigNum::BigNum(const BigNum & T) : len(T.len)  //拷贝构造函数
{
    int i;
    memset(a,0,sizeof(a));
    for(i = 0 ; i < len ; i++)
        a[i] = T.a[i];
}
BigNum & BigNum::operator=(const BigNum & n)   //重载赋值运算符，大数之间进行赋值运算
{
    int i;
    len = n.len;
    memset(a,0,sizeof(a));
    for(i = 0 ; i < len ; i++)
```

```
        a[i] = n.a[i];
    return *this;
}
istream& operator>>(istream & in,  BigNum & b)   //重载输入运算符
{
    char ch[MAXSIZE*4];
    int i = -1;
    in>>ch;
    int l=strlen(ch);
    int count=0,sum=0;
    for(i=l-1;i>=0;)
    {
        sum = 0;
        int t=1;
        for(int j=0;j<4&&i>=0;j++,i--,t*=10)
        {
            sum+=(ch[i]-'0')*t;
        }
        b.a[count]=sum;
        count++;
    }
    b.len =count++;
    return in;

}
ostream& operator<<(ostream& out,  BigNum& b)   //重载输出运算符
{
    int i;
    cout << b.a[b.len - 1];
    for(i = b.len - 2 ; i >= 0 ; i--)
    {
        cout.width(DLEN);
        cout.fill('0');
        cout << b.a[i];
    }
    return out;
}

BigNum BigNum::operator+(const BigNum & T) const   //两个大数之间的相加运算
{
    BigNum t(*this);
    int i,big;      //位数
    big = T.len > len ? T.len : len;
    for(i = 0 ; i < big ; i++)
    {
        t.a[i] +=T.a[i];
        if(t.a[i] > MAXN)
        {
            t.a[i + 1]++;
            t.a[i] -=MAXN+1;
        }
    }
```

```
    if(t.a[big] != 0)
        t.len = big + 1;
    else
        t.len = big;
    return t;
}
BigNum BigNum::operator-(const BigNum & T) const   //两个大数之间的相减运算
{
    int i,j,big;
    bool flag;
    BigNum t1,t2;
    if(*this>T)
    {
        t1=*this;
        t2=T;
        flag=0;
    }
    else
    {
        t1=T;
        t2=*this;
        flag=1;
    }
    big=t1.len;
    for(i = 0 ; i < big ; i++)
    {
        if(t1.a[i] < t2.a[i])
        {
            j = i + 1;
            while(t1.a[j] == 0)
                j++;
            t1.a[j--]--;
            while(j > i)
                t1.a[j--] += MAXN;
            t1.a[i] += MAXN + 1 - t2.a[i];
        }
        else
            t1.a[i] -= t2.a[i];
    }
    t1.len = big;
    while(t1.a[len - 1] == 0 && t1.len > 1)
    {
        t1.len--;
        big--;
    }
    if(flag)
        t1.a[big-1]=0-t1.a[big-1];
    return t1;
}

BigNum BigNum::operator*(const BigNum & T) const   //两个大数之间的相乘运算
{
```

```
    BigNum ret;
    int i,j,up;
    int temp,temp1;
    for(i = 0 ; i < len ; i++)
    {
        up = 0;
        for(j = 0 ; j < T.len ; j++)
        {
            temp = a[i] * T.a[j] + ret.a[i + j] + up;
            if(temp > MAXN)
            {
                temp1 = temp - temp / (MAXN + 1) * (MAXN + 1);
                up = temp / (MAXN + 1);
                ret.a[i + j] = temp1;
            }
            else
            {
                up = 0;
                ret.a[i + j] = temp;
            }
        }
        if(up != 0)
            ret.a[i + j] = up;
    }
    ret.len = i + j;
    while(ret.a[ret.len - 1] == 0 && ret.len > 1)
        ret.len--;
    return ret;
}
BigNum BigNum::operator/(const int & b) const   //大数对一个整数进行相除运算
{
    BigNum ret;
    int i,down = 0;
    for(i = len - 1 ; i >= 0 ; i--)
    {
        ret.a[i] = (a[i] + down * (MAXN + 1)) / b;
        down = a[i] + down * (MAXN + 1) - ret.a[i] * b;
    }
    ret.len = len;
    while(ret.a[ret.len - 1] == 0 && ret.len > 1)
        ret.len--;
    return ret;
}
int BigNum::operator %(const int & b) const    //大数对一个 int 类型的变量进行取模运算
{
    int i,d=0;
    for (i = len-1; i>=0; i--)
    {
        d = ((d * (MAXN+1))% b + a[i])% b;
    }
    return d;
}
```

```
BigNum BigNum::operator^(const int & n) const    //大数的 n 次方运算
{
    BigNum t,ret(1);
    int i;
    if(n<0)
        exit(-1);
    if(n==0)
        return 1;
    if(n==1)
        return *this;
    int m=n;
    while(m>1)
    {
        t=*this;
        for( i=1;i<<1<=m;i<<=1)
        {
            t=t*t;
        }
        m-=i;
        ret=ret*t;
        if(m==1)
            ret=ret*(*this);
    }
    return ret;
}
bool BigNum::operator>(const BigNum & T) const   //大数和另一个大数的大小比较
{
    int ln;
    if(len > T.len)
        return true;
    else if(len == T.len)
    {
        ln = len - 1;
        while(a[ln] == T.a[ln] && ln >= 0)
            ln--;
        if(ln >= 0 && a[ln] > T.a[ln])
            return true;
        else
            return false;
    }
    else
        return false;
}
bool BigNum::operator >(const int & t) const    //大数和一个 int 类型的变量的大小比较
{
    BigNum b(t);
    return *this>b;
}

void BigNum::print()    //输出大数
{
    int i;
```

```
    //cout << a[len - 1];
    printf("%d",a[len-1]);
    for(i = len - 2 ; i >= 0 ; i--)
    {
        /*cout.width(DLEN);
        cout.fill('0');
        cout << a[i];*/
        printf("%04d",a[i]);
    }
    //cout << endl;
    printf("\n");
}
```