ACM/ICPC Template Manaual

CSL

October 8, 2017

Contents

0 Include

```cpp
#include <bits/stdc++.h>
using namespace std;
#define clr(a, x) memset(a, x, sizeof(a))
#define mp(x, y) make_pair(x, y)
#define pb(x) push_back(x)
#define X first
#define Y second
#define fastin \
   ios_base::sync_with_stdio(0); \
   cin.tie(0);
typedef long long ll;
typedef long double ld;
typedef pair<int, int> PII;
typedef vector<int> VI;
const int INF = 0x3f3f3f3f;
const int mod = 1e9 + 7;
const double eps = 1e-6;
```

vim 配置

```vim
syntax on
set cindent
set nu
set tabstop = 4
set shiftwidth = 4
set background = dark
map<C-A> ggVG"+y
map<F5>: call Run()<CR>
func !Run()
   exec "w"
   exec "!g++ -Wall % -o %<"
   exec "!./%<"
endfunc
```

1 Math

1.1 Prime

1.1.1 Eratosthenes Sieve

$O(n \log\log n)$ 筛出 maxn 内所有素数
$notprime[i] = 0/1$ 0 为素数 1 为非素数

```
const int maxn = "Edit";
bool notprime[maxn] = {1, 1}; // 0 && 1 为非素数
void GetPrime()
{
   for (int i = 2; i < maxn; i++)
      if (!notprime[i] && i <= maxn / i) // 筛到√n为止
         for (int j = i * i; j < maxn; j += i)
            notprime[j] = 1;
}
```

1.1.2 Eular Sieve

$O(n)$ 得到欧拉函数 $phi[]$、素数表 $prime[]$、素数个数 tot
传入的 n 为函数定义域上界

```
const int maxn = "Edit";
bool vis[maxn];
int tot, phi[maxn], prime[maxn];
void CalPhi(int n)
{
   clr(vis, 0);
   phi[1] = 1;
   tot = 0;
   for (int i = 2; i < n; i++)
   {
      if (!vis[i])
         prime[tot++] = i, phi[i] = i - 1;
      for (int j = 0; j < tot; j++)
      {
         if (i * prime[j] > n) break;
         vis[i * prime[j]] = 1;
         if (i % prime[j] == 0)
         {
            phi[i * prime[j]] = phi[i] * prime[j];
            break;
         }
         else
            phi[i * prime[j]] = phi[i] * (prime[j] - 1);
      }
   }
}
```

1.1.3 Prime Factorization

函数返回素因数个数
数组以 $fact[i][0]^{fact[i][1]}$ 的形式保存第 i 个素因数

```
ll fact[100][2];
int getFactors(ll x)
{
   int cnt = 0;
   for (int i = 0; prime[i] <= x / prime[i]; i++)
   {
      fact[cnt][1] = 0;
      if (x % prime[i] == 0)
      {
         fact[cnt][0] = prime[i];
         while (x % prime[i] == 0)
            fact[cnt][1]++, x /= prime[i];
         cnt++;
      }
   }
   if (x != 1)
      fact[cnt][0] = x, fact[cnt++][1] = 1;
   return cnt;
}
```

1.1.4 Miller Rabin

$O(s\log n)$ 内判定 2^{63} 内的数是不是素数，s 为测定次数

```
bool Miller_Rabin(ll n, int s)
{
   if (n == 2) return 1;
   if (n < 2 || !(n & 1)) return 0;
   int t = 0;
   ll x, y, u = n - 1;
   while ((u & 1) == 0) t++, u >>= 1;
   for (int i = 0; i < s; i++)
   {
      ll a = rand() % (n - 1) + 1;
      ll x = Pow(a, u, n);
      for (int j = 0; j < t; j++)
      {
         ll y = Mul(x, x, n);
         if (y == 1 && x != 1 && x != n - 1) return 0;
         x = y;
      }
      if (x != 1) return 0;
   }
   return 1;
}
```

1.1.5 Segment Sieve

对区间 $[a,b)$ 内的整数执行筛法。
函数返回区间内素数个数
is_prime[i-a]=true 表示 i 是素数
$a < b \leq 10^{12}, b-a \leq 10^6$

```
int segment_sieve(ll a, ll b)
{
   int tot = 0;
   for (ll i = 0; i * i < b; ++i)
      is_prime_small[i] = true;
   for (ll i = 0; i < b - a; ++i)
      is_prime[i] = true;
   for (ll i = 2; i * i < b; ++i)
      if (is_prime_small[i])
      {
         for (ll j = 2 * i; j * j < b; j += i)
            is_prime_small[j] = false;
         for (ll j = max(2LL, (a + i - 1) / i) * i; j < b; j += i)
            is_prime[j - a] = false;
      }
   for (ll i = 0; i < b - a; ++i)
      if (is_prime[i]) prime[tot++] = i + a;
   return tot;
}
```

1.2 Eular-phi

1.2.1 Eular

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

1.2.2 Sieve

```
const int N = "Edit";
int phi[N] = {0, 1};
void CalEuler()
{
   for (int i = 2; i < N; i++)
      if (!phi[i])
         for (int j = i; j < N; j += i)
         {
            if (!phi[j]) phi[j] = j;
            phi[j] = phi[j] / i * (i - 1);
         }
}
```

1.3 Exgcd-Inv

1.3.1 Extended Euclidean

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

1.3.2 ax+by=c

```
// 引用返回通解: X = x + k * dx, Y = y - k * dy
// 引用返回的x是最小非负整数解，方程无解函数返回0
#define Mod(a,b) (((a)%(b)+(b))%(b))
bool solve(ll a, ll b, ll c, ll &x, ll &y, ll &dx, ll &dy)
{
    if (a == 0 && b == 0) return 0;
    ll x0, y0;
    ll d = exgcd(a, b, x0, y0);
    if (c % d != 0) return 0;
    dx = b / d;
    dy = a / d;
    x = Mod(x0 * c / d, dx);
    y = (c - a * x) / b;
//  y = Mod(y0 * c / d, dy); x = (c - b * y) / a;
    return 1;
}
```

1.3.3 Multiplicative Inverse Modulo

利用 exgcd 求 a 在模 m 下的逆元，需要保证 $gcd(a,m) == 1$.

```
ll inv(ll a, ll m)
{
    ll x, y;
    ll d = exgcd(a, m, x, y);
    return d == 1 ? (x + m) % m : -1;
}
```

$a < m$ 且 m 为素数时，有以下两种求法

```
ll inv(ll a, ll m) { return a == 1 ? 1 : inv(m % a, m) * (m - m / a) % m; }
ll inv(ll a, ll m) { return Pow(a, m - 2, m); }
```

1.4 Modulo-Linear-Equation

1.4.1 Chinese Remainder Theory

```
// X = r[i] (mod m[i]); 要求m[i]两两互质
// 引用返回通解X = re + k * mo;
void crt(ll r[], ll m[], ll n, ll &re, ll &mo)
{
    mo = 1, re = 0;
    for (int i = 0; i < n; i++) mo *= m[i];
```

```
    for (int i = 0; i < n; i++)
    {
        ll x, y, tm = mo / m[i];
        ll d = exgcd(tm, m[i], x, y);
        re = (re + tm * x * r[i]) % mo;
    }
    re = (re + mo) % mo;
}
```

1.4.2 ExCRT

```
// X = r[i] (mod m[i]); m[i]可以不两两互质
// 引用返回通解X = re + k * mo; 函数返回是否有解
bool excrt(ll r[], ll m[], ll n, ll &re, ll &mo)
{
    ll x, y;
    mo = m[0], re = r[0];
    for (int i = 1; i < n; i++)
    {
        ll d = exgcd(mo, m[i], x, y);
        if ((r[i] - re) % d != 0) return 0;
        x = (r[i] - re) / d * x % (m[i] / d);
        re += x * mo;
        mo = mo / d * m[i];
        re %= mo;
    }
    re = (re + mo) % mo;
    return 1;
}
```

1.5 Combinatorics

1.5.1 Combination

$0 \le m \le n \le 1000$

```
const int maxn = 1010;
ll C[maxn][maxn];
void CalComb()
{
    C[0][0] = 1;
    for (int i = 1; i < maxn; i++)
    {
        C[i][0] = 1;
        for (int j = 1; j <= i; j++)
            C[i][j] = (C[i - 1][j - 1] + C[i - 1][j]) % mod;
    }
}
```

$0 \le m \le n \le 10^5$, 模 p 为素数

```
const int maxn = 100010;
ll f[maxn];
void CalFact()
{
    f[0] = 1;
    for (int i = 1; i < maxn; i++)
```

```
        f[i] = (f[i - 1] * i) % mod;
}
ll C(int n, int m) { return f[n] * inv(f[m] * f[n - m] % mod, mod) % mod; }
```

1.5.2 Lucas

$1 \leq n, m \leq 1000000000, 1 < p < 100000$, p 是素数

```
const int maxp = 100010;
ll f[maxp];
void CalFact(ll p)
{
   f[0] = 1;
   for (int i = 1; i <= p; i++)
      f[i] = (f[i - 1] * i) % p;
}
ll Lucas(ll n, ll m, ll p)
{
   ll ret = 1;
   while (n && m)
   {
      ll a = n % p, b = m % p;
      if (a < b) return 0;
      ret = (ret * f[a] * Pow(f[b] * f[a - b] % p, p - 2, p)) % p;
      n /= p, m /= p;
   }
   return ret;
}
```

1.5.3 Big-combination

$0 \leq n \leq 10^9, 0 \leq m \leq 10^4, 1 \leq k \leq 10^9 + 7$

```
vector<int> v;
int dp[110];
ll Cal(int l, int r, int k, int dis)
{
   ll res = 1;
   for (int i = l; i <= r; i++)
   {
      int t = i;
      for (int j = 0; j < v.size(); j++)
      {
         int y = v[j];
         while (t % y == 0)
            dp[j] += dis, t /= y;
      }
      res = res * (ll)t % k;
   }
   return res;
}
ll Comb(int n, int m, int k)
{
   clr(dp, 0);
   v.clear();
   int tmp = k;
```

```cpp
    for (int i = 2; i * i <= tmp; i++)
    {
        if (tmp % i == 0)
        {
            int num = 0;
            while (tmp % i == 0)
                tmp /= i, num++;
            v.pb(i);
        }
    }
    if (tmp != 1) v.pb(tmp);
    ll ans = Cal(n - m + 1, n, k, 1);
    for (int j = 0; j < v.size(); j++)
        ans = ans * Pow(v[j], dp[j], k) % k;
    ans = ans * inv(Cal(2, m, k, -1), k) % k;
    return ans;
}
```

1.5.4 Polya

推论：一共 n 个置换，第 i 个置换的循环节个数为 $gcd(i,n)$

$N*N$ 的正方形格子，$c^{n^2}+2c^{\frac{n^2+3}{4}}+c^{\frac{n^2+1}{2}}+2c^{n\frac{n+1}{2}}+2c^{\frac{n(n+1)}{2}}$

正六面体，$\frac{m^8+17m^4+6m^2}{24}$ 正四面体，$\frac{m^4+11m^2}{12}$

```cpp
// 长度为n的项链串用c种颜色染
ll solve(int c, int n)
{
    if (n == 0) return 0;
    ll ans = 0;
    for (int i = 1; i <= n; i++)
        ans += Pow(c, __gcd(i, n));
    if (n & 1)
        ans += n * Pow(c, n + 1 >> 1);
    else
        ans += n / 2 * (1 + c) * Pow(c, n >> 1);
    return ans / n / 2;
}
```

1.6 FastMul-FastPow

```cpp
ll Mul(ll a, ll b, ll mod)
{
    ll t = 0;
    for (; b; b >>= 1, a = (a << 1) % mod)
        if (b & 1) t = (t + a) % mod;
    return t;
}
ll Pow(ll a, ll n, ll mod)
{
    ll t = 1;
    for (; n; n >>= 1, a = (a * a % mod))
        if (n & 1) t = (t * a % mod);
    return t;
}
```

1.7 Mobius-Inversion

1.7.1 Mobius

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

```
ll ans;
const int maxn = "Edit";
int n, x, prime[maxn], tot, mu[maxn];
bool check[maxn];
void calmu()
{
   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; j++)
      {
         if (i * prime[j] >= maxn) break;
         check[i * prime[j]] = true;
         if (i % prime[j] == 0)
         {
            mu[i * prime[j]] = 0;
            break;
         }
         else
            mu[i * prime[j]] = -mu[i];
      }
   }
}
```

1.7.2 Number of coprime

```
// 有n个数（n<=100000），问这n个数中互质的数的对数
clr(b, 0);
_max = 0;
ans = 0;
for (int i = 0; i < n; i++)
{
   scanf("%d", &x);
   if (x > _max) _max = x;
   b[x]++;
}
int cnt;
for (int i = 1; i <= _max; i++)
{
   cnt = 0;
   for (ll j = i; j <= _max; j += i)
      cnt += b[j];
   ans += 1LL * mu[i] * cnt * cnt;
}
printf("%lld\n", (ans - b[1]) / 2);
```

1.7.3 VisibleTrees

```
// gcd(x,y)==1的对数 x<=n, y<=m
int main()
{
   calmu();
   int n, m;
   scanf("%d %d", &n, &m);
   if (n < m) swap(n, m);
   ll ans = 0;
   for (int i = 1; i <= m; ++i)
   {
      ans += (ll)mu[i] * (n / i) * (m / i);
   }
   printf("%lld\n", ans);
   return 0;
}
```

1.8 Others

1.8.1 Josephus

```
int num, m, r = 0;
for (int k = 1; k <= num; ++k) r = (r + m) % k;
cout << r + 1 << endl;
```

1.8.2 Digit

```
// n^n最左边一位数
int leftmost(int n)
{
   double m = n * log10((double)n);
   double g = m - (long long)m;
   g = pow(10.0, g);
   return (int)g;
}

// n!位数
int count(ll n)
{
   if (n == 1) return 1;
   return (int)ceil(0.5 * log10(2 * M_PI * n) + n * log10(n) - n * log10(M_E));
}
```

1.8.3 FFT

```
const double PI = acos(-1.0);
//复数结构体
struct Complex
{
   double x, y; //实部和虚部 x+yi
   Complex(double _x = 0.0, double _y = 0.0) { x = _x, y = _y; }
   Complex operator-(const Complex& b) const { return Complex(x - b.x, y - b.y); }
   Complex operator+(const Complex& b) const { return Complex(x + b.x, y + b.y); }
```

```
    Complex operator*(const Complex& b) const { return Complex(x * b.x - y * b.y, x * b.
        y + y * b.x); }
};
/*
* 进行FFT和IFFT前的反转变换。
* 位置i和 (i二进制反转后位置) 互换
* len必须取2的幂
*/
void change(Complex y[], int len)
{
    for (int i = 1, j = len / 2; i < len - 1; i++)
    {
        if (i < j) swap(y[i], y[j]);
        //交换互为小标反转的元素，i<j保证交换一次
        //i做正常的+1，j左反转类型的+1,始终保持i和j是反转的
        int k = len / 2;
        while (j >= k) j -= k, k /= 2;
        if (j < k) j += k;
    }
}
/*
* 做FFT
* len必须为2^k形式，
* on==1时是DFT，on==-1时是IDFT
*/
void fft(Complex y[], int len, int on)
{
    change(y, len);
    for (int h = 2; h <= len; h <<= 1)
    {
        Complex wn(cos(-on * 2 * PI / h), sin(-on * 2 * PI / h));
        for (int j = 0; j < len; j += h)
        {
            Complex w(1, 0);
            for (int k = j; k < j + h / 2; k++)
            {
                Complex u = y[k];
                Complex t = w * y[k + h / 2];
                y[k] = u + t, y[k + h / 2] = u - t;
                w = w * wn;
            }
        }
    }
    if (on == -1)
        for (int i = 0; i < len; i++) y[i].x /= len;
}
```

1.9 Formula

1. 约数定理：若 $n = \prod_{i=1}^{k} p_i^{a_i}$，则
   (a) 约数个数 $f(n) = \prod_{i=1}^{k}(a_i + 1)$
   (b) 约数和 $g(n) = \prod_{i=1}^{k}(\sum_{j=0}^{a_i} p_i^j)$
2. 小于 n 且互素的数之和为 $n\varphi(n)/2$
3. 若 $gcd(n,i) = 1$，则 $gcd(n, n-i) = 1 (1 \le i \le n)$

4. 错排公式：$D(n) = (n-1)(D(n-2) + D(n-1)) = \sum_{i=2}^{n} \frac{(-1)^k n!}{k!} = [\frac{n!}{e} + 0.5]$
5. 威尔逊定理：$p\ is\ prime\ \Rightarrow (p-1)! \equiv -1(mod\ p)$
6. 欧拉定理：$gcd(a,n) = 1 \Rightarrow a^{\varphi(n)} \equiv 1(mod\ n)$
7. 欧拉定理推广：$gcd(n,p) = 1 \Rightarrow a^n \equiv a^{n\%\varphi(p)}(mod\ p)$
8. 素数定理：对于不大于 n 的素数个数 $\pi(n)$，$\lim\limits_{n\to\infty} \pi(n) = \frac{n}{\ln n}$
9. 位数公式：正整数 x 的位数 $N = log10(n) + 1$
10. 斯特灵公式 $n! \approx \sqrt{2\pi n}(\frac{n}{e})^n$
11. 设 $a > 1, m, n > 0$, 则 $gcd(a^m - 1, a^n - 1) = a^{gcd(m,n)} - 1$
12. 设 $a > b, gcd(a,b) = 1$, 则 $gcd(a^m - b^m, a^n - b^n) = a^{gcd(m,n)} - b^{gcd(m,n)}$

$$G = gcd(C_n^1, C_n^2, ..., C_n^{n-1}) = \begin{cases} n, & n \text{ is prime} \\ 1, & n \text{ has multy prime factors} \\ p, & n \text{ has single prime factor } p \end{cases}$$

$gcd(Fib(m), Fib(n)) = Fib(gcd(m,n))$

13. 若 $gcd(m,n) = 1$, 则:
    (a) 最大不能组合的数为 $m * n - m - n$
    (b) 不能组合数个数 $N = \frac{(m-1)(n-1)}{2}$
14. $(n+1)lcm(C_n^0, C_n^1, ..., C_n^{n-1}, C_n^n) = lcm(1, 2, ..., n+1)$
15. 若 p 为素数，则 $(x + y + ... + w)^p \equiv x^p + y^p + ... + w^p(mod\ p)$
16. 卡特兰数：1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012
    $h(0) = h(1) = 1, h(n) = \frac{(4n-2)h(n-1)}{n+1} = \frac{C_{2n}^n}{n+1} = C_{2n}^n - C_{2n}^{n-1}$

$$a_{n+m} = \sum_{i=0}^{m-1} b_i a_{n+i} \Rightarrow \begin{pmatrix} a_{n+m} \\ a_{n+m-1} \\ \vdots \\ a_{n+1} \end{pmatrix} = \begin{pmatrix} b_{m-1} & \cdots & b_1 & b_0 \\ 1 & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & 0 \end{pmatrix} \begin{pmatrix} a_{n+m-1} \\ a_{n+m-2} \\ \vdots \\ a_n \end{pmatrix}$$

$$a_{n+m} = \sum_{i=0}^{m-1} b_i a_{n+i} + c \Rightarrow \begin{pmatrix} a_{n+m} \\ a_{n+m-1} \\ \vdots \\ a_{n+1} \\ 1 \end{pmatrix} = \begin{pmatrix} b_{m-1} & \cdots & b_1 & b_0 & c \\ 1 & \cdots & 0 & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & 1 & 0 & 0 \\ 0 & \cdots & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a_{n+m-1} \\ a_{n+m-2} \\ \vdots \\ a_n \\ 1 \end{pmatrix}$$

2 String-Processing

2.1 KMP

```
// 返回y中x的个数
int ne[N];
void initkmp(char x[], int m)
{
   int i, j;
   j = ne[0] = -1;
   i = 0;
   while (i < m)
   {
      while (j != -1 && x[i] != x[j])
         j = ne[j];
      ne[++i] = ++j;
   }
}
int kmp(char x[], int m, char y[], int n)
{
   int i, j, ans;
   i = j = ans = 0;
   initkmp(x, m);
   while (i < n)
   {
      while (j != -1 && y[i] != x[j]) j = ne[j];
      i++;
      j++;
      if (j >= m)
      {
         ans++;
         j = ne[j];
      }
   }
   return ans;
}
```

2.2 ExtendKMP

```
//next[i]:x[i...m-1]与x[0...m-1]的最长公共前缀
//extend[i]:y[i...n-1]与x[0...m-1]的最长公共前缀
void pre_EKMP(char x[], int m)
{
   next[0] = m;
   int j = 0;
   while (j + 1 < m && x[j] == x[j + 1])j++;
   next[1] = j;
   int k = 1;
   for (int i = 2; i < m; i++)
   {
      int p = next[k] + k - 1;
      int L = next[i - k];
      if (i + L < p + 1)next[i] = L;
      else
      {
         j = max(0, p - i + 1);
         while (i + j < m && x[i + j] == x[j])j++;
```

```
            next[i] = j;
            k = i;
        }
    }
}
void EKMP(char x[], int m, char y[], int n)
{
    pre_EKMP(x, m, next);
    int j = 0;
    while (j < n && j < m && x[j] == y[j])j++;
    extend[0] = j;
    int k = 0;
    for (int i = 1; i < n; i++)
    {
        int p = extend[k] + k - 1;
        int L = next[i - k];
        if (i + L < p + 1)extend[i] = L;
        else
        {
            j = max(0, p - i + 1);
            while (i + j < n && j < m && y[i + j] == x[j])j++;
            extend[i] = j;
            k = i;
        }
    }
}
```

2.3 Manacher

```
// O(n)求解最长回文子串
const int N = "Edit";
char s[N], str[N << 1];
int p[N << 1];
void Manacher(char s[], int &n)
{
    str[0] = '$';
    str[1] = '#';
    for (int i = 0; i < n; i++)
    {
        str[(i << 1) + 2] = s[i];
        str[(i << 1) + 3] = '#';
    }
    n = 2 * n + 2;
    str[n] = 0;
    int mx = 0, id;
    for (int i = 1; i < n; i++)
    {
        p[i] = mx > i ? min(p[2 * id - i], mx - i) : 1;
        for (; str[i - p[i]] == str[i + p[i]]; p[i]++);
        if (p[i] + i > mx)
        {
            mx = p[i] + i;
            id = i;
        }
    }
}
int solve(char s[])
{
```

```
    int n = strlen(s);
    Manacher(s, n);
    int res = 0;
    for (int i = 0; i < n; i++)
        res = max(res, p[i]);
    return res - 1;
}
```

2.4 Aho-Corasick Automaton

```
//不要忘记build!
const int maxn = "Edit";
struct Trie
{
    int ch[maxn][26], f[maxn], val[maxn];
    int sz, rt;
    int newnode() { clr(ch[sz], -1), val[sz] = 0; return sz++; }
    void init() { sz = 0, rt = newnode(); }
    inline int idx(char c) { return c - 'A'; };
    void insert(const char* s)
    {
        int u = 0, n = strlen(s);
        for (int i = 0; i < n; i++)
        {
            int c = idx(s[i]);
            if (ch[u][c] == -1) ch[u][c] = newnode();
            u = ch[u][c];
        }
        val[u]++;
    }
    void build()
    {
        queue<int> q;
        f[rt] = rt;
        for (int c = 0; c < 26; c++)
        {
            if (~ch[rt][c])
                f[ch[rt][c]] = rt, q.push(ch[rt][c]);
            else
                ch[rt][c] = rt;
        }
        while (!q.empty())
        {
            int u = q.front();
            q.pop();
            // val[u] |= val[f[u]];
            for (int c = 0; c < 26; c++)
            {
                if (~ch[u][c])
                    f[ch[u][c]] = ch[f[u]][c], q.push(ch[u][c]);
                else
                    ch[u][c] = ch[f[u]][c];
            }
        }
    }
    //返回主串中有多少模式串
    int query(const char* s)
    {
```

```
        int u = rt, n = strlen(s);
        int res = 0;
        for (int i = 0; i < n; i++)
        {
            int c = idx(s[i]);
            u = ch[u][c];
            int tmp = u;
            while (tmp != rt)
            {
                res += val[tmp];
                val[tmp] = 0;
                tmp = f[tmp];
            }
        }
        return res;
    }
};
```

2.5 Suffix Array

```
//倍增算法构造后缀数组,复杂度O(nlogn)
const int maxn = "Edit";
char s[maxn];
int sa[maxn], t[maxn], t2[maxn], c[maxn], rnk[maxn], height[maxn];
//n为字符串的长度,字符集的值为0~m-1
void build_sa(int m, int n)
{
    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 = 0; 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++) rnk[sa[i]] = i;
    for (int i = 0; i < n; i++)
    {
        if (k) k--;
        int j = sa[rnk[i] - 1];
        while (s[i + k] == s[j + k]) k++;
        height[rnk[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 = 0;
    while ((1 << (k + 1)) <= r - l + 1) k++;
    return min(dp[l][k], dp[r - (1 << k) + 1][k]);
}
// 求两个后缀的最长公共前缀
int lcp(int a, int b)
{
    a = rnk[a], b = rnk[b];
    if (a > b) swap(a, b);
    a++;
    return rmq(a, b);
}
```

3 Data-Structure

3.1 Binary-Indexed-Tree

$O(\log n)$ 查询和修改数组的前缀和

```
// 注意下标应从1开始 n是全局变量
const int maxn = "Edit";
int bit[N], n;
int sum(int x)
{
   int s = 0;
   for (int i = x; i; i -= i & -i)
      s += bit[i];
   return s;
}
void add(int x, int v)
{
   for (int i = x; i <= n; i += i & -i)
      bit[i] += v;
}
```

3.2 Segment-Tree

```
#define lson rt << 1 // 左儿子
#define rson rt << 1 | 1 // 右儿子
#define Lson l, m, lson // 左子树
#define Rson m + 1, r, rson // 右子树
void PushUp(int rt); // 用lson和rson更新rt
void PushDown(int rt[, int m]); // rt的标记下移，m为区间长度（若与标记有关）
void build(int l, int r, int rt); // 以rt为根节点，对区间[l, r]建立线段树
void update([...,] int l, int r, int rt) // rt[l, r]内寻找目标并更新
int query(int L, int R, int l, int r, int rt) // rt-[l, r]内查询[L, R]
```

3.2.1 Single-point Update

```
const int maxn = "Edit";
int sum[maxn << 2];
void PushUp(int rt) { sum[rt] = sum[lson] + sum[rson]; }
void build(int l, int r, int rt)
{
   if (l == r)
   {
      scanf("%d", &sum[rt]);
      return;
   } // 建立的时候直接输入叶节点
   int m = (l + r) >> 1;
   build(Lson);
   build(Rson);
   PushUp(rt);
}
void update(int p, int add, int l, int r, int rt)
{
   if (l == r)
   {
```

```
        sum[rt] += add;
        return;
    }
    int m = (l + r) >> 1;
    if (p <= m) update(p, add, Lson);
    else update(p, add, Rson);
    PushUp(rt);
}
int query(int L, int R, int l, int r, int rt)
{
    if (L <= l && r <= R) return sum[rt];
    int m = (l + r) >> 1, s = 0;
    if (L <= m) s += query(L, R, Lson);
    if (m < R) s += query(L, R, Rson);
    return s;
}
```

3.2.2 Interval Update

```
// seg[rt]用于存放懒惰标记，注意PushDown时标记的传递
const int maxn = "Edit";
int seg[maxn << 2], sum[maxn << 2];

void PushUp(int rt) { sum[rt] = sum[lson] + sum[rson]; }
void PushDown(int rt, int m)
{
    if (seg[rt] == 0) return;
    seg[lson] += seg[rt];
    seg[rson] += seg[rt];
    sum[lson] += seg[rt] * (m - (m >> 1));
    sum[rson] += seg[rt] * (m >> 1);
    seg[rt] = 0;
}
void build(int l, int r, int rt)
{
    seg[rt] = 0;
    if (l == r)
    {
        scanf("%lld", &sum[rt]);
        return;
    }
    int m = (l + r) >> 1;
    build(Lson);
    build(Rson);
    PushUp(rt);
}
void update(int L, int R, int add, int l, int r, int rt)
{
    if (L <= l && r <= R)
    {
        seg[rt] += add;
        sum[rt] += add * (r - l + 1);
        return;
    }
    PushDown(rt, r - l + 1);
    int m = (l + r) >> 1;
    if (L <= m) update(L, R, add, Lson);
    if (m < R) update(L, R, add, Rson);
```

```
    PushUp(rt);
}
int query(int L, int R, int l, int r, int rt)
{
    if (L <= l && r <= R) return sum[rt];
    PushDown(rt, r - l + 1);
    int m = (l + r) >> 1, ret = 0;
    if (L <= m) ret += query(L, R, Lson);
    if (m < R) ret += query(L, R, Rson);
    return ret;
}
```

3.3 Partition-Tree

```
#define Lson l, m, dep + 1
#define Rson m + 1, r, dep + 1

int tree[20][maxn]; //表示每层每个位置的值
int sorted[maxn]; //已经排序好的数
int toleft[20][maxn]; //toleft[p][i]表示第i层从1到i有数分入左边
void build(int l, int r, int dep)
{
    if (l == r) return;
    int m = (l + r) >> 1, same = m - l + 1; //表示等于中间值而且被分入左边的个数
    for (int i = l; i <= r; i++)
        if (tree[dep][i] < sorted[m])
            same--;
    int lpos = l;
    int rpos = m + 1;
    for (int i = l; i <= r; i++)
    {
        if (tree[dep][i] < sorted[m])
            tree[dep + 1][lpos++] = tree[dep][i];
        else if (tree[dep][i] == sorted[m] && same > 0)
        {
            tree[dep + 1][lpos++] = tree[dep][i];
            same--;
        }
        else
            tree[dep + 1][rpos++] = tree[dep][i];
        toleft[dep][i] = toleft[dep][l - 1] + lpos - l;
    }
    build(Lson);
    build(Rson);
}
//查询区间第k小的数
int query(int L, int R, int k, int l, int r, int dep)
{
    if (L == R) return tree[dep][L];
    int m = (l + r) >> 1;
    int cnt = toleft[dep][R] - toleft[dep][L - 1];
    if (cnt >= k)
    {
        int newl = l + toleft[dep][L - 1] - toleft[dep][l - 1];
        int newr = newl + cnt - 1;
        return query(newl, newr, k, Lson);
    }
    else
```

```
    {
        int newr = R + toleft[dep][r] - toleft[dep][R];
        int newl = newr - (R - L - cnt);
        return query(newl, newr, k - cnt, Rson);
    }
}
```

3.4 Functional-Segment-Tree

```
// 静态查询区间第k小的值
const int maxn = "Edit";
int a[maxn], rt[maxn];
int cnt;
int lson[maxn << 5], rson[maxn << 5], sum[maxn << 5];
#define Lson l, m, lson[x], lson[y]
#define Rson m + 1, r, rson[x], rson[y]

void update(int p, int l, int r, int& x, int y)
{
    lson[++cnt] = lson[y], rson[cnt] = rson[y], sum[cnt] = sum[y] + 1, x = cnt;
    if (l == r) return;
    int m = (l + r) >> 1;
    if (p <= m) update(p, Lson);
    else update(p, Rson);
}
int query(int l, int r, int x, int y, int k)
{
    if (l == r) return l;
    int m = (l + r) >> 1;
    int s = sum[lson[y]] - sum[lson[x]];
    if (s >= k) return query(Lson, k);
    else return query(Rson, k - s);
}
```

3.5 RMQ

```
const int maxn = "Edit";
int mmax[maxn][30], mmin[maxn][30];
int a[maxn], n, k;
void init()
{
    for (int i = 1; i <= n; i++) mmax[i][0] = mmin[i][0] = a[i];
    for (int j = 1; (1 << j) <= n; j++)
        for (int i = 1; i + (1 << j) - 1 <= n; i++)
        {
            mmax[i][j] = max(mmax[i][j - 1], mmax[i + (1 << (j - 1))][j - 1]);
            mmin[i][j] = min(mmin[i][j - 1], mmin[i + (1 << (j - 1))][j - 1]);
        }
}
// op=0/1 返回[l,r]最大/小值
int rmq(int l, int r, int op)
{
    int k = 0;
    while ((1 << (k + 1)) <= r - l + 1) k++;
    if (op == 0)
        return max(mmax[l][k], mmax[r - (1 << k) + 1][k]);
```

```
    return min(mmin[l][k], mmin[r - (1 << k) + 1][k]);
}

// 二维RMQ
void init()
{
    for (int i = 0; (1 << i) <= n; i++)
        for (int j = 0; (1 << j) <= m; j++)
        {
            if (i == 0 && j == 0) continue;
            for (int row = 1; row + (1 << i) - 1 <= n; row++)
                for (int col = 1; col + (1 << j) - 1 <= m; col++)
                    //当x或y等于0的时候，就相当于一维的RMQ了
                    if (i == 0)
                        dp[row][col][i][j] = max(dp[row][col][i][j - 1],
                                                 dp[row][col + (1 << (j - 1))][i][j - 1]);
                    else if (j == 0)
                        dp[row][col][i][j] = max(dp[row][col][i - 1][j],
                                                 dp[row + (1 << (i - 1))][col][i - 1][j]);
                    else
                        dp[row][col][i][j] = max(dp[row][col][i][j - 1],
                                                 dp[row][col + (1 << (j - 1))][i][j - 1]);
        }
}
int rmq(int x1, int y1, int x2, int y2)
{
    int kx = 0, ky = 0;
    while (x1 + (1 << (1 + kx)) - 1 <= x2) kx++;
    while (y1 + (1 << (1 + ky)) - 1 <= y2) ky++;
    int m1 = dp[x1][y1][kx][ky];
    int m2 = dp[x2 - (1 << kx) + 1][y1][kx][ky];
    int m3 = dp[x1][y2 - (1 << ky) + 1][kx][ky];
    int m4 = dp[x2 - (1 << kx) + 1][y2 - (1 << ky) + 1][kx][ky];
    return max(max(m1, m2), max(m3, m4));
}
```

4 Graph-Theory

4.1 Union-find-Set

```
const int maxn = "Edit";
int n, fa[maxn], ra[maxn];
void init()
{
   for (int i = 0; i <= n; i++) fa[i] = i, ra[i] = 0;
}
int find(int x)
{
   return fa[x] != x ? fa[x] = find(fa[x]) : x;
}
void unite(int x, int y)
{
   x = find(x), y = find(y);
   if (x == y) return;
   if (ra[x] < ra[y])
      fa[x] = y;
   else
   {
      fa[y] = x;
      if (ra[x] == ra[y]) ra[x]++;
   }
}
bool same(int x, int y) { return find(x) == find(y); }
```

4.2 Minimal-Spanning-Tree

4.2.1 Kruskal

```
vector<pair<int, PII> > G;
void add_edge(int u, int v, int d) { G.pb(mp(d, mp(u, v))); }
int Kruskal(int n)
{
   init(n);
   sort(G.begin(), G.end());
   int m = G.size();
   int num = 0, ret = 0;
   for (int i = 0; i < m; i++)
   {
      pair<int, PII> p = G[i];
      int x = p.Y.X;
      int y = p.Y.Y;
      int d = p.X;
      if (!same(x, y))
      {
         unite(x, y);
         num++;
         ret += d;
      }
      if (num == n - 1) break;
   }
   return ret;
}
```

4.2.2 Prim

```
// 耗费矩阵cost[][],标号从0开始,0~n-1
// 返回最小生成树的权值,返回-1表示原图不连通
const int maxn = "Edit";
bool vis[maxn];
int lowc[maxn];
int Prim(int cost[][maxn], int n)
{
   int ans = 0;
   clr(vis, 0);
   vis[0] = 1;
   for (int i = 1; i < n; i++)
      lowc[i] = cost[0][i];
   for (int i = 1; i < n; i++)
   {
      int minc = INF;
      int p = -1;
      for (int j = 0; j < n; j++)
         if (!vis[j] && minc > lowc[j])
         {
            minc = lowc[j];
            p = j;
         }
      if (minc == INF) return -1;
      vis[p] = 1;
      ans += minc;
      for (int j = 0; j < n; j++)
         if (!vis[j] && lowc[j] > cost[p][j])
            lowc[j] = cost[p][j];
   }
   return ans;
}
```

4.3 Shortest-Path

4.3.1 Dijkstra

```
// pair<权值, 点>
// 记得初始化
const int maxn = "Edit";
typedef pair<int, int> PII;
typedef vector<PII> VII;
VII G[maxn];
int vis[maxn], dis[maxn];
void init(int n)
{
   for (int i = 0; i < n; i++)
      G[i].clear();
}
void add_edge(int u, int v, int w)
{
   G[u].pb(mp(w, v));
}
void Dijkstra(int s, int n)
{
   clr(vis, 0);
   clr(dis, 0x3f);
```

```
    dis[s] = 0;
    priority_queue<PII, VII, greater<PII> > q;
    q.push(mp(dis[s], s));
    while (!q.empty())
    {
        PII t = q.top();
        int x = t.Y;
        q.pop();
        if (vis[x]) continue;
        vis[x] = 1;
        for (int i = 0; i < G[x].size(); i++)
        {
            int y = G[x][i].Y;
            int w = G[x][i].X;
            if (!vis[y] && dis[y] > dis[x] + w)
            {
                dis[y] = dis[x] + w;
                q.push(mp(dis[y], y));
            }
        }
    }
}
```

4.3.2 SPFA

```
// G[u] = mp(v, w)
// SPFA()返回0表示存在负环
const int maxn = "Edit";
vector<PII> G[maxn];
bool vis[maxn];
int dis[maxn];
int inqueue[maxn];
void init(int n)
{
    for (int i = 0; i < n; i++) G[i].clear();
}
void add_edge(int u, int v, int w) { G[u].pb(mp(v, w)); }
bool SPFA(int s, int n)
{
    clr(vis, 0);
    clr(dis, 0x3f);
    clr(inqueue, 0);
    dis[s] = 0;
    queue<int> q; // 待优化的节点入队
    q.push(s);
    while (!q.empty())
    {
        int x = q.front();
        q.pop();
        vis[x] = false;
        for (int i = 0; i < G[x].size(); i++)
        {
            int y = G[x][i].X, w = G[x][i].Y;
            if (dis[y] > dis[x] + w)
            {
                dis[y] = dis[x] + w;
                if (!vis[y])
                {
```

```
                q.push(y);
                vis[y] = true;
                if (++inqueue[y] >= n) return 0;
            }
        }
    }
}
return 1;
}
```

4.3.3 Floyd

$O(n^3)$ 求出任意两点间最短路

```
const int maxn = "Edit";
int G[maxn][maxn];
void init(int n)
{
   clr(G, 0x3f);
   for (int i = 0; i < n; i++) G[i][i] = 0;
}
void add_edge(int u, int v, int w) { G[u][v] = min(G[u][v], w); }
void Floyd(int n)
{
   for (int k = 0; k < n; k++)
      for (int i = 0; i < n; i++)
         for (int j = 0; j < n; j++)
            G[i][j] = min(G[i][j], G[i][k] + G[k][j]);
}
```

4.4 Topo-Sort

4.4.1 Matrix

```
// 存图前记得初始化
// Ans存放拓排结果，G为邻接矩阵，deg为入度信息
// 排序成功返回1，存在环返回0
const int maxn = "Edit";
int Ans[maxn]; // 存放拓扑排序结果
int G[maxn][maxn]; // 存放图信息
int deg[maxn]; // 存放点入度信息
void init() { clr(G, 0), clr(deg, 0), clr(Ans, 0); }
void add_edge(int u, int v)
{
   if (G[u][v]) return;
   G[u][v] = 1, deg[v]++;
}
bool Toposort(int n)
{
   int tot = 0;
   queue<int> q;
   for (int i = 0; i < n; ++i)
      if (deg[i] == 0) q.push(i);
   while (!q.empty())
   {
      int v = q.front();
```

```
        q.pop();
        Ans[tot++] = v;
        for (int i = 0; i < n; ++i)
            if (G[v][i] == 1)
                if (--deg[i] == 0) q.push(i);
    }
    if (tot < n) return false;
    return true;
}
```

4.4.2 List

```
// 存图前记得初始化
// Ans排序结果，G邻接表，deg入度，map用于判断重边
// 排序成功返回1，存在环返回0
const int maxn = "Edit";
typedef pair<int, int> PII;
int Ans[maxn];
vector<int> G[maxn];
int deg[maxn];
map<PII, bool> S;
void init(int n)
{
    S.clear();
    for (int i = 0; i < n; i++) G[i].clear();
    clr(deg, 0), clr(Ans, 0);
}
void add_edge(int u, int v)
{
    if (S[mp(u, v)]) return;
    G[u].pb(v);
    S[mp(u, v)] = 1;
    deg[v]++;
}
bool Toposort(int n)
{
    int tot = 0;
    queue<int> q;
    for (int i = 0; i < n; ++i)
        if (deg[i] == 0) q.push(i);
    while (!q.empty())
    {
        int v = q.front();
        que.pop();
        Ans[tot++] = v;
        for (int i = 0; i < G[v].size(); ++i)
        {
            int t = G[v][i];
            if (--deg[t] == 0) q.push(t);
        }
    }
    if (tot < n) return false;
    return true;
}
```

4.5 LCA

4.5.1 Tarjan

```
//Tarjan离线算法求LCA
const int maxn = "Edit";
int par[maxn]; //并查集
int ans[maxn]; //存储答案
vector<int> G[maxn]; //邻接表
vector<int> query[maxn], num[maxn]; //存储查询信息
bool vis[maxn]; //是否被遍历
inline void init(int n)
{
   for (int i = 1; i <= n; i++)
   {
      G[i].clear();
      query[i].clear();
      num[i].clear();
      par[i] = i;
      vis[i] = 0;
   }
}
inline void add_edge(int u, int v) { G[u].pb(v); }
inline void add_query(int id, int u, int v)
{
   query[u].pb(v), query[v].pb(u);
   num[u].pb(id), num[v].pb(id);
}
void tarjan(int u)
{
   vis[u] = 1;
   for (int i = 0; i < G[u].size(); i++)
   {
      int v = G[u][i];
      if (vis[v]) continue;
      tarjan(v);
      unite(u, v);
   }
   for (int i = 0; i < query[u].size(); i++)
   {
      int v = query[u][i];
      if (!vis[v]) continue;
      ans[num[u][i]] = find(v);
   }
}
```

4.6 Biconnected-Component

```
//割顶的bccno无意义
const int maxn = "Edit";
int pre[maxn], iscut[maxn], bccno[maxn], dfs_clock, bcc_cnt;
vector<int> G[maxn], bcc[maxn];
stack<PII> s;
void init(int n)
{
   for (int i = 0; i < n; i++) G[i].clear();
}
inline void add_edge(int u, int v) { G[u].pb(v), G[v].pb(u); }
```

```
int dfs(int u, int fa)
{
   int lowu = pre[u] = ++dfs_clock;
   int child = 0;
   for (int i = 0; i < G[u].size(); i++)
   {
      int v = G[u][i];
      PII e = mp(u, v);
      if (!pre[v])
      {
         //没有访问过v
         s.push(e);
         child++;
         int lowv = dfs(v, u);
         lowu = min(lowu, lowv); //用后代的low函数更新自己
         if (lowv >= pre[u])
         {
            iscut[u] = true;
            bcc_cnt++;
            bcc[bcc_cnt].clear(); //注意! bcc从1开始编号
            for (;;)
            {
               PII x = s.top();
               s.pop();
               if (bccno[x.X] != bcc_cnt)
                  bcc[bcc_cnt].pb(x.X), bcc[x.X] = bcc_cnt;
               if (bccno[x.Y] != bcc_cnt)
                  bcc[bcc_cnt].pb(x.Y), bcc[x.Y] = bcc_cnt;
               if (x.X == u && x.Y == v) break;
            }
         }
      }
      else if (pre[v] < pre[u] && v != fa)
      {
         s.push(e);
         lowu = min(lowu, pre[v]); //用反向边更新自己
      }
   }
   if (fa < 0 && child == 1) iscut[u] = 0;
   return lowu;
}
void find_bcc(int n)
{
   //调用结束后s保证为空，所以不用清空
   clr(pre, 0), clr(iscut, 0), clr(bccno, 0);
   dfs_clock = bcc_cnt = 0;
   for (int i = 0; i < n; i++)
      if (!pre[i]) dfs(i, -1);
}
```

4.7 Strongly-Connected-Component

```
const int maxn = "Edit";
vector<int> G[maxn];
int pre[maxn], lowlink[maxn], sccno[maxn], dfs_clock, scc_cnt;
stack<int> S;
inline void add_edge(int u, int v) { G[u].pb(v); }
void dfs(int u)
```

```
{
   pre[u] = lowlink[u] = ++dfs_clock;
   S.push(u);
   for (int i = 0; i < G[u].size(); i++)
   {
      int v = G[u][i];
      if (!pre[v])
      {
         dfs(v);
         lowlink[u] = min(lowlink[u], lowlink[v]);
      }
      else if (!sccno[v])
         lowlink[u] = min(lowlink[u], pre[v]);
   }
   if (lowlink[u] == pre[u])
   {
      scc_cnt++;
      for (;;)
      {
         int x = S.top();
         S.pop();
         sccno[x] = scc_cnt;
         if (x == u) break;
      }
   }
}
void find_scc(int n)
{
   dfs_clock = 0, scc_cnt = 0;
   clr(sccno, 0), clr(pre, 0);
   for (int i = 0; i < n; i++)
      if (!pre[i]) dfs(i);
}
```

4.8 Bipartite-Graph-Matching

1) 一个二分图中的最大匹配数等于这个图中的最小点覆盖数

König 定理是一个二分图中很重要的定理, 它的意思是, 一个二分图中的最大匹配数等于这个图中的最小点覆盖数。如果你还不知道什么是最小点覆盖, 我也在这里说一下: 假如选了一个点就相当于覆盖了以它为端点的所有边, 你需要选择最少的点来覆盖所有的边。

2) 最小路径覆盖 =|G|-最大匹配数

在一个 N*N 的有向图中, 路径覆盖就是在图中找一些路经, 使之覆盖了图中的所有顶点, 且任何一个顶点有且只有一条路径与之关联;

(如果把这些路径中的每条路径从它的起始点走到它的终点, 那么恰好可以经过图中的每个顶点一次且仅一次); 如果不考虑图中存在回路, 那么每每条路径就是一个弱连通子集.

由上面可以得出:

1. 一个单独的顶点是一条路径;

2. 如果存在一路径 $p_1, p_2, ...p_k$, 其中 p_1 为起点,p_k 为终点, 那么在覆盖图中, 顶点 $p_1, p2, ...p_k$ 不再与其它的顶点之间存在有向边.

最小路径覆盖就是找出最小的路径条数, 使之成为 G 的一个路径覆盖.

路径覆盖与二分图匹配的关系: 最小路径覆盖 =|G|-最大匹配数;

3) 二分图最大独立集 = 顶点数-二分图最大匹配

独立集: 图中任意两个顶点都不相连的顶点集合。

4.8.1 Hungry(Matrix)

```
/*
二分图匹配(匈牙利算法的DFS实现)(邻接矩阵形式)
初始化:g[][]两边顶点的划分情况
建立g[i][j]表示i->j的有向边就可以了,是左边向右边的匹配
g没有边相连则初始化为0
uN是匹配左边的顶点数,vN是匹配右边的顶点数
调用:res=hungary();输出最大匹配数
优点:适用于稠密图,DFS找增广路,实现简洁易于理解
时间复杂度:O(VE)
顶点编号从0开始的
*/
const int maxn = "Edit";
int uN, vN; //u,v的数目,使用前面必须赋值
int g[maxn][maxn]; //邻接矩阵
int linker[maxn];
bool used[maxn];
bool dfs(int u)
{
   for (int v = 0; v < vN; v++)
      if (g[u][v] && !used[v])
      {
         used[v] = true;
         if (linker[v] == -1 || dfs(linker[v]))
         {
            linker[v] = u;
            return true;
         }
      }
   return false;
}
int hungary()
{
   int res = 0;
   clr(linker, -1);
   for (int u = 0; u < uN; u++)
   {
      clr(used, 0);
      if (dfs(u)) res++;
   }
   return res;
}
```

4.8.2 Hungry(List)

```
/*
匈牙利算法邻接表形式
使用前用init()进行初始化
加边使用函数addedge(u,v)
*/
const int maxn = "Edit";
int n;
vector<int> G[maxn];
int linker[maxn];
bool used[maxn];
inline void init(int n)
{
```

```
    for (int i = 0; i < n; i++) G[i].clear();
}
inline void addedge(int u, int v) { G[u].pb(v); }
bool dfs(int u)
{
    for (int i = 0; i < G[u].size(); i++)
    {
        int v = G[u][i];
        if (!used[v])
        {
            used[v] = true;
            if (linker[v] == -1 || dfs(linker[v]))
            {
                linker[v] = u;
                return true;
            }
        }
    }
    return false;
}
int hungary()
{
    int ans = 0;
    clr(linker, -1);
    for (int u = 0; u < n; v++)
    {
        clr(vis, 0);
        if (dfs(u)) ans++;
    }
    return ans;
}
```

4.8.3 Hopcroft-Carp

```
/*
二分图匹配(Hopcroft-Carp算法)
复杂度O(sqrt(n)*E)
邻接表存图,vector实现
vector先初始化,然后加边
uN 为左端的顶点数,使用前赋值(点编号0开始)
*/
const int maxn = "Edit";
vector<int> G[maxn];
int uN;
int Mx[maxn], My[maxn];
int dx[maxn], dy[maxn];
int dis;
bool used[maxn];
inline void init(int n)
{
    for (int i = 0; i < n; i++) G[i].clear();
}
inline void addedge(int u, int v) { G[u].pb(v); }
bool SearchP()
{
    queue<int> Q;
    dis = INF;
    clr(dx, -1);
```

```
    clr(dy, -1);
    for (int i = 0; i < uN; i++)
        if (Mx[i] == -1)
        {
            Q.push(i);
            dx[i] = 0;
        }
    while (!Q.empty())
    {
        int u = Q.front();
        Q.pop();
        if (dx[u] > dis) break;
        int sz = G[u].size();
        for (int i = 0; i < sz; i++)
        {
            int v = G[u][i];
            if (dy[v] == -1)
            {
                dy[v] = dx[u] + 1;
                if (My[v] == -1)
                    dis = dy[v];
                else
                {
                    dx[My[v]] = dy[v] + 1;
                    Q.push(My[v]);
                }
            }
        }
    }
    return dis != INF;
}
bool DFS(int u)
{
    int sz = G[u].size();
    for (int i = 0; i < sz; i++)
    {
        int v = G[u][i];
        if (!used[v] && dy[v] == dx[u] + 1)
        {
            used[v] = true;
            if (My[v] != -1 && dy[v] == dis) continue;
            if (My[v] == -1 || DFS(My[v]))
            {
                My[v] = u, Mx[u] = v;
                return true;
            }
        }
    }
    return false;
}
int MaxMatch()
{
    int res = 0;
    clr(Mx, -1), clr(My, -1);
    while (SearchP())
    {
        clr(used, false);
        for (int i = 0; i < uN; i++)
            if (Mx[i] == -1 && DFS(i)) res++;
```

```
    }
    return res;
}
```

4.9 2-SAT

```
struct TwoSAT
{
    int n;
    vector<int> G[maxn << 1];
    bool mark[maxn << 1];
    int S[maxn << 1], c;
    void init(int n)
    {
        this->n = n;
        for (int i = 0; i < (n << 1); i++) G[i].clear();
        clr(mark, 0);
    }
    bool dfs(int x)
    {
        if (mark[x ^ 1]) return false;
        if (mark[x]) return true;
        mark[x] = true;
        S[c++] = x;
        for (int i = 0; i < G[x].size(); i++)
            if (!dfs(G[x][i])) return false;
        return true;
    }
    //x = xval or y = yval
    void add_clause(int x, int xval, int y, int yval)
    {
        x = (x << 1) + xval;
        y = (y << 1) + yval;
        G[x ^ 1].pb(y);
        G[y ^ 1].pb(x);
    }
    bool solve()
    {
        for (int i = 0; i < (n << 1); 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;
    }
};
```

4.10 Network-Flow

4.10.1 EdmondKarp

```
const int maxn = "Edit";
struct Edge
{
   int from, to, cap, flow;
   Edge(int u, int v, int c, int f) : from(u), to(v), cap(c), flow(f) {}
};
struct EdmonsKarp //时间复杂度O(v*E*E)
{
   int n, m;
   vector<Edge> edges; //边数的两倍
   vector<int> G[maxn]; //邻接表，G[i][j]表示节点i的第j条边在e数组中的序号
   int a[maxn]; //起点到i的可改进量
   int p[maxn]; //最短路树上p的入弧编号
   void init(int n)
   {
      for (int i = 0; i < n; i++) G[i].clear();
      edges.clear();
   }
   void AddEdge(int from, int to, int cap)
   {
      edges.pb(Edge(from, to, cap, 0));
      edges.pb(Edge(to, from, 0, 0)); //反向弧
      m = edges.size();
      G[from].pb(m - 2);
      G[to].pb(m - 1);
   }
   int Maxflow(int s, int t)
   {
      int flow = 0;
      for (;;)
      {
         clr(a, 0);
         queue<int> q;
         q.push(s);
         a[s] = INF;
         while (!q.empty())
         {
            int x = q.front();
            q.pop();
            for (int i = 0; i < G[x].size(); i++)
            {
               Edge& e = edges[G[x][i]];
               if (!a[e.to] && e.cap > e.flow)
               {
                  p[e.to] = G[x][i];
                  a[e.to] = min(a[x], e.cap - e.flow);
                  q.push(e.to);
               }
            }
            if (a[t]) break;
         }
         if (!a[t]) break;
         for (int u = t; u != s; u = edges[p[u]].from)
         {
            edges[p[u]].flow += a[t];
            edges[p[u] ^ 1].flow -= a[t];
         }
         flow += a[t];
      }
```

```
        return flow;
    }
};
```

4.10.2 Dinic

```
const int maxn = "Edit";
struct Edge
{
    int from, to, cap, flow;
    Edge(int u, int v, int c, int f) : from(u), to(v), cap(c), flow(f) {}
};
struct Dinic
{
    int n, m, s, t; //结点数，边数（包括反向弧），源点编号和汇点编号
    vector<Edge> edges; //边表。edge[e]和edge[e^1]互为反向弧
    vector<int> G[maxn]; //邻接表，G[i][j]表示节点i的第j条边在e数组中的序号
    bool vis[maxn]; //BFS使用
    int d[maxn]; //从起点到i的距离
    int cur[maxn]; //当前弧下标
    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)
    {
        edges.pb(Edge(from, to, cap, 0));
        edges.pb(Edge(to, from, 0, 0));
        m = edges.size();
        G[from].pb(m - 2);
        G[to].pb(m - 1);
    }
    bool BFS()
    {
        clr(vis, 0);
        clr(d, 0);
        queue<int> q;
        q.push(s);
        d[s] = 0;
        vis[s] = 1;
        while (!q.empty())
        {
            int x = q.front();
            q.pop();
            for (int i = 0; i < G[x].size(); i++)
            {
                Edge& e = edges[G[x][i]];
                if (!vis[e.to] && e.cap > e.flow)
                {
                    vis[e.to] = 1;
                    d[e.to] = d[x] + 1;
                    q.push(e.to);
                }
            }
        }
        return vis[t];
```

```
    }
    int DFS(int x, int a)
    {
        if (x == t || a == 0) return a;
        int flow = 0, f;
        for (int& i = cur[x]; i < G[x].size(); i++)
        {
            //从上次考虑的弧
            Edge& e = edges[G[x][i]];
            if (d[x] + 1 == d[e.to] && (f = DFS(e.to, min(a, e.cap - e.flow))) > 0)
            {
                e.flow += f;
                edges[G[x][i] ^ 1].flow -= f;
                flow += f;
                a -= f;
                if (a == 0) break;
            }
        }
        return flow;
    }
    int Maxflow(int s, int t)
    {
        this->s = s;
        this->t = t;
        int flow = 0;
        while (BFS())
        {
            clr(cur, 0);
            flow += DFS(s, INF);
        }
        return flow;
    }
};
```

4.10.3 ISAP

```
const int maxn = "Edit";
struct Edge
{
    int from, to, cap, flow;
    Edge(int u, int v, int c, int f) : from(u), to(v), cap(c), flow(f) {}
};
struct ISAP
{
    int n, m, s, t; //结点数，边数（包括反向弧），源点编号和汇点编号
    vector<Edge> edges; //边表。edges[e]和edges[e^1]互为反向弧
    vector<int> G[maxn]; //邻接表，G[i][j]表示结点i的第j条边在e数组中的序号
    bool vis[maxn]; //BFS使用
    int d[maxn]; //起点到i的距离
    int cur[maxn]; //当前弧下标
    int p[maxn]; //可增广路上的一条弧
    int num[maxn]; //距离标号计数
    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)
{
   edges.pb(Edge(from, to, cap, 0));
   edges.pb(Edge(to, from, 0, 0));
   int m = edges.size();
   G[from].pb(m - 2);
   G[to].pb(m - 1);
}
int Augumemt()
{
   int x = t, a = INF;
   while (x != s)
   {
      Edge& e = edges[p[x]];
      a = min(a, e.cap - e.flow);
      x = edges[p[x]].from;
   }
   x = t;
   while (x != s)
   {
      edges[p[x]].flow += a;
      edges[p[x] ^ 1].flow -= a;
      x = edges[p[x]].from;
   }
   return a;
}
void BFS()
{
   clr(vis, 0);
   clr(d, 0);
   queue<int> q;
   q.push(t);
   d[t] = 0;
   vis[t] = 1;
   while (!q.empty())
   {
      int x = q.front();
      q.pop();
      int len = G[x].size();
      for (int i = 0; i < len; i++)
      {
         Edge& e = edges[G[x][i]];
         if (!vis[e.from] && e.cap > e.flow)
         {
            vis[e.from] = 1;
            d[e.from] = d[x] + 1;
            q.push(e.from);
         }
      }
   }
}
int Maxflow(int s, int t)
{
   this->s = s;
   this->t = t;
   int flow = 0;
   BFS();
   clr(num, 0);
   for (int i = 0; i < n; i++) num[d[i]]++;
```

```
        int x = s;
        clr(cur, 0);
        while (d[s] < n)
        {
            if (x == t)
            {
                flow += Augumemt();
                x = s;
            }
            int ok = 0;
            for (int i = cur[x]; i < G[x].size(); i++)
            {
                Edge& e = edges[G[x][i]];
                if (e.cap > e.flow && d[x] == d[e.to] + 1)
                {
                    ok = 1;
                    p[e.to] = G[x][i];
                    cur[x] = i;
                    x = e.to;
                    break;
                }
            }
            if (!ok) //Retreat
            {
                int m = n - 1;
                for (int i = 0; i < G[x].size(); i++)
                {
                    Edge& e = edges[G[x][i]];
                    if (e.cap > e.flow)
                        m = min(m, d[e.to]);
                }
                if (--num[d[x]] == 0) break; //gap优化
                num[d[x] = m + 1]++;
                cur[x] = 0;
                if (x != s) x = edges[p[x]].from;
            }
        }
        return flow;
    }
};
```

4.10.4 MinCost MaxFlow

```
const int maxn = "Edit";
struct Edge
{
    int from, to, cap, flow, cost;
    Edge(int u, int v, int c, int f, int w) : from(u), to(v), cap(c), flow(f), cost(w)
        {}
};
struct MCMF
{
    int n, m;
    vector<Edge> edges;
    vector<int> G[maxn];
    int inq[maxn]; //是否在队列中
    int d[maxn]; //bellmanford
    int p[maxn]; //上一条弧
```

```
int a[maxn]; //可改进量
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.pb(Edge(from, to, cap, 0, cost));
   edges.pb(Edge(to, from, 0, 0, -cost));
   m = edges.size();
   G[from].pb(m - 2);
   G[to].pb(m - 1);
}
bool BellmanFord(int s, int t, int& flow, ll& cost)
{
   for (int i = 0; i < n; i++) d[i] = INF;
   clr(inq, 0);
   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;
            p[e.to] = G[u][i];
            a[e.to] = min(a[u], e.cap - e.flow);
            if (!inq[e.to])
            {
               q.push(e.to);
               inq[e.to] = 1;
            }
         }
      }
   }
   if (d[t] == INF) return false; // 当没有可增广的路时退出
   flow += a[t];
   cost += (ll)d[t] * (ll)a[t];
   for (int u = t; u != s; u = edges[p[u]].from)
   {
      edges[p[u]].flow += a[t];
      edges[p[u] ^ 1].flow -= a[t];
   }
   return true;
}
int MincostMaxflow(int s, int t, ll& cost)
{
   int flow = 0;
```

```
    cost = 0;
    while (BellmanFord(s, t, flow, cost));
    return flow;
  }
};
```

5 Computational-Geometry

5.1 Basic-Function

```
#define zero(x) ((fabs(x) < eps ? 1 : 0))
#define sgn(x) (fabs(x) < eps ? 0 : ((x) < 0 ? -1 : 1))

struct point
{
   double x, y;
   point(double a = 0, double b = 0) { x = a, y = b; }
   point operator-(const point& b) const { return point(x - b.x, y - b.y); }
   point operator+(const point& b) const { return point(x + b.x, y + b.y); }
   // 两点是否重合
   bool operator==(point& b) { return zero(x - b.x) && zero(y - b.y); }
   // 点积(以原点为基准)
   double operator*(const point& b) const { return x * b.x + y * b.y; }
   // 叉积(以原点为基准)
   double operator^(const point& b) const { return x * b.y - y * b.x; }
   // 绕P点逆时针旋转a弧度后的点
   point rotate(point b, double a)
   {
      double dx, dy;
      (*this - b).split(dx, dy);
      double tx = dx * cos(a) - dy * sin(a);
      double ty = dx * sin(a) + dy * cos(a);
      return point(tx, ty) + b;
   }
   // 点坐标分别赋值到a和b
   void split(double& a, double& b) { a = x, b = y; }
};
struct line
{
   point s, e;
   line() {}
   line(point ss, point ee) { s = ss, e = ee; }
};
```

5.2 Position

5.2.1 Point-Point

```
double dist(point a, point b) { return sqrt((a - b) * (a - b)); }
```

5.2.2 Line-Line

```
// <0, *> 表示重合; <1, *> 表示平行; <2, P> 表示交点是P;
pair<int, point> spoint(line l1, line l2)
{
   point res = l1.s;
   if (sgn((l1.s - l1.e) ^ (l2.s - l2.e)) == 0)
      return mp(sgn((l1.s - l2.e) ^ (l2.s - l2.e)) != 0, res);
   double t = ((l1.s - l2.s) ^ (l2.s - l2.e)) / ((l1.s - l1.e) ^ (l2.s - l2.e));
   res.x += (l1.e.x - l1.s.x) * t;
   res.y += (l1.e.y - l1.s.y) * t;
   return mp(2, res);
```

```
}
```

5.2.3 Segment-Segment

```
bool segxseg(line l1, line l2)
{
   return
      max(l1.s.x, l1.e.x) >= min(l2.s.x, l2.e.x) &&
      max(l2.s.x, l2.e.x) >= min(l1.s.x, l1.e.x) &&
      max(l1.s.y, l1.e.y) >= min(l2.s.y, l2.e.y) &&
      max(l2.s.y, l2.e.y) >= min(l1.s.y, l1.e.y) &&
      sgn((l2.s - l1.e) ^ (l1.s - l1.e)) * sgn((l2.e-l1.e) ^ (l1.s - l1.e)) <= 0 &&
      sgn((l1.s - l2.e) ^ (l2.s - l2.e)) * sgn((l1.e-l2.e) ^ (l2.s - l2.e)) <= 0;
}
```

5.2.4 Line-Segment

```
//l1是直线,l2是线段
bool segxline(line l1, line l2)
{
   return sgn((l2.s - l1.e) ^ (l1.s - l1.e)) * sgn((l2.e - l1.e) ^ (l1.s - l1.e)) <= 0;
}
```

5.2.5 Point-Line

```
point pointtoline(point P, line L)
{
   point res;
   double t = ((P - L.s) * (L.e - L.s)) / ((L.e - L.s) * (L.e - L.s));
   res.x = L.s.x + (L.e.x - L.s.x) * t;
   res.y = L.s.y + (L.e.y - L.s.y) * t;
   return dist(P, res);
}
```

5.2.6 Point-Segment

```
point pointtosegment(point p, line l)
{
   point res;
   double t = ((p - l.s) * (l.e - l.s)) / ((l.e - l.s) * (l.e - l.s));
   if (t >= 0 && t <= 1)
   {
      res.x = l.s.x + (l.e.x - l.s.x) * t;
      res.y = l.s.y + (l.e.y - l.s.y) * t;
   }
   else
      res = dist(p, l.s) < dist(p, l.e) ? l.s : l.e;
   return res;
}
```

5.2.7 Point on Segment

```
bool PointOnSeg(point p, line l)
{
   return
      sgn((l.s - p) ^ (l.e-p)) == 0 &&
      sgn((p.x - l.s.x) * (p.x - l.e.x)) <= 0 &&
      sgn((p.y - l.s.y) * (p.y - l.e.y)) <= 0;
}
```

5.3 Polygon

5.3.1 Area

```
double area(point p[], int n)
{
   double res = 0;
   for (int i = 0; i < n; i++)
      res += (p[i] ^ p[(i + 1) % n]) / 2;
   return fabs(res);
}
```

5.3.2 Point in Convex

```
// 点形成一个凸包, 而且按逆时针排序(如果是顺时针把里面的<0改为>0)
// 点的编号 : [0,n)
// -1 : 点在凸多边形外
// 0 : 点在凸多边形边界上
// 1 : 点在凸多边形内
int PointInConvex(point a, point p[], int n)
{
   for (int i = 0; i < n; i++)
      if (sgn((p[i] - a) ^ (p[(i + 1) % n] - a)) < 0)
         return -1;
      else if (PointOnSeg(a, line(p[i], p[(i + 1) % n])))
         return 0;
   return 1;
}
```

5.3.3 Point in Polygon

```
// 射线法,poly[]的顶点数要大于等于3,点的编号0~n-1
// -1 : 点在凸多边形外
// 0 : 点在凸多边形边界上
// 1 : 点在凸多边形内
int PointInPoly(point p, point poly[], int n)
{
   int cnt;
   line ray, side;
   cnt = 0;
   ray.s = p;
   ray.e.y = p.y;
   ray.e.x = -100000000000.0; // -INF,注意取值防止越界
   for (int i = 0; i < n; i++)
   {
```

```
      side.s = poly[i];
      side.e = poly[(i + 1) % n];
      if (PointOnSeg(p, side)) return 0;
      //如果平行轴则不考虑
      if (sgn(side.s.y - side.e.y) == 0)
         continue;
      if (PointOnSeg(sid e.s, r ay))
         cnt += (sgn(side.s.y - side.e.y) > 0);
      else if (PointOnSeg(side.e, ray))
         cnt += (sgn(side.e.y - side.s.y) > 0);
      else if (segxseg(ray, side))
         cnt++;
   }
   return cnt % 2 == 1 ? 1 : -1;
}
```

5.3.4 Judge Convex

```
//点可以是顺时针给出也可以是逆时针给出
//点的编号1~n-1
bool isconvex(point poly[], int n)
{
   bool s[3];
   clr(s, 0);
   for (int i = 0; i < n; i++)
   {
      s[sgn((poly[(i + 1) % n] - poly[i]) ^ (poly[(i + 2) % n] - poly[i])) + 1] = 1;
      if (s[0] && s[2]) return 0;
   }
   return 1;
}
```

5.4 Integer-Points

5.4.1 On Segment

```
int OnSegment(line l) { return __gcd(fabs(l.s.x - l.e.x), fabs(l.s.y - l.e.y)) + 1; }
```

5.4.2 On Polygon Edge

```
int OnEdge(point p[], int n)
{
   int i, ret = 0;
   for (i = 0; i < n; i++)
      ret += __gcd(fabs(p[i].x - p[(i + 1) % n].x), fabs(p[i].y - p[(i + 1) % n].y));
   return ret;
}
```

5.4.3 Inside Polygon

```
int InSide(point p[], int n)
{
   int i, area = 0;
   for (i = 0; i < n; i++)
```

```
      area += p[(i + 1) % n].y * (p[i].x - p[(i + 2) % n].x);
   return (fabs(area) - OnEdge(n, p)) / 2 + 1;
}
```

5.5 Circle

5.5.1 Circumcenter

```
point waixin(point a, point b, point c)
{
   double a1 = b.x - a.x, b1 = b.y - a.y, c1 = (a1 * a1 + b1 * b1) / 2;
   double a2 = c.x - a.x, b2 = c.y - a.y, c2 = (a2 * a2 + b2 * b2) / 2;
   double d = a1 * b2 - a2 * b1;
   return point(a.x + (c1 * b2 - c2 * b1) / d, a.y + (a1 * c2 - a2 * c1) / d);
}
```

6 Dynamic-Programming

6.1 Subsequence

6.1.1 Max Sum

```
// 传入序列a和长度n，返回最大子序列和
int MaxSeqSum(int a[], int n)
{
   int rt = 0, cur = 0;
   for (int i = 0; i < n; i++)
   {
      cur += a[i];
      rt = max(cur, rt);
      cur = max(0, cur);
   }
   return rt;
}
```

6.1.2 Longest Increase

```
// 序列下标从1开始，LIS()返回长度，序列存在lis[]中
#define N 100100
int n, len, a[N], b[N], f[N];
int Find(int p, int l, int r)
{
   int mid;
   while (l <= r)
   {
      mid = (l + r) >> 1;
      if (a[p] > b[mid]) l = mid + 1;
      else r = mid - 1;
   }
   return f[p] = l;
}
int LIS(int lis[])
{
   int len = 1;
   f[1] = 1;
   b[1] = a[1];
   for (int i = 2; i <= n; i++)
   {
      if (a[i] > b[len]) b[++len] = a[i], f[i] = len;
      else b[Find(i, 1, len)] = a[i];
   }
   for (int i = n, t = len; i >= 1 && t >= 1; i--)
      if (f[i] == t)
         lis[--t] = a[i];
   return len;
}
```

6.1.3 Longest Common Increase

```
// 序列下标从1开始
int LCIS(int a[], int b[], int n, int m)
{
```

```
    clr(dp, 0);
    for (int i = 1; i <= n; i++)
    {
        int ma = 0;
        for (int j = 1; j <= m; j++)
        {
            dp[i][j] = dp[i - 1][j];
            if (a[i] > b[j]) ma = max(ma, dp[i - 1][j]);
            if (a[i] == b[j]) dp[i][j] = ma + 1;
        }
    }
    return *max_element(dp[n] + 1, dp[n] + 1 + m);
}
```

7 Others

7.1 Matrix

7.1.1 Matrix FastPow

```
typedef vector<ll> vec;
typedef vector<vec> mat;
mat mul(mat& A, mat& B)
{
   mat C(A.size(), vec(B[0].size()));
   for (int i = 0; i < A.size(); i++)
      for (int k = 0; k < B.size(); k++)
         if (A[i][k]) // 对稀疏矩阵的优化
            for (int j = 0; j < B[0].size(); j++)
               C[i][j] = (C[i][j] + A[i][k] * B[k][j]) % mod;
   return C;
}
mat Pow(mat A, ll n)
{
   mat B(A.size(), vec(A.size()));
   for (int i = 0; i < A.size(); i++) B[i][i] = 1;
   for (; n; n >>= 1, A = mul(A, A))
      if (n & 1) B = mul(B, A);
   return B;
}
```

7.1.2 Gauss Elimination

```
void gauss()
{
   int now = 1, to;
   double t;
   for (int i = 1; i <= n; i++)
   {
      /*for (to = now; !a[to][i] && to <= n; to++);
      //做除法时减小误差, 可不写
      if (to != now)
         for (int j = 1; j <= n + 1; j++)
            swap(a[to][j], a[now][j]);*/
      t = a[now][i];
      for (int j = 1; j <= n + 1; j++) a[now][j] /= t;
      for (int j = 1; j <= n; j++)
         if (j != now)
         {
            t = a[j][i];
            for (int k = 1; k <= n + 1; k++) a[j][k] -= t * a[now][k];
         }
      now++;
   }
}
```

求线性基

```
for (int i = 1; i <= m; i++)
   for (int j = 63; ~j; j--)
      if ((a[i] >> j) & 1)
         if (!ins[j])
```

```
        {
            ins[j] = a[i];
            break;
        }
        else
            a[i] ^= ins[j];
```

7.2 BigNum

7.2.1 High-precision

```
// 加法 乘法 小于号 输出
struct bint
{
    int l;
    short int w[100];
    bint(int x = 0)
    {
        l = x == 0;
        clr(w, 0);
        while (x != 0)
            w[l++] = x % 10, x /= 10;
    }
    bool operator<(const bint& x) const
    {
        if (l != x.l) return l < x.l;
        int i = l - 1;
        while (i >= 0 && w[i] == x.w[i]) i--;
        return (i >= 0 && w[i] < x.w[i]);
    }
    bint operator+(const bint& x) const
    {
        bint ans;
        ans.l = l > x.l ? l : x.l;
        for (int i = 0; i < ans.l; i++)
        {
            ans.w[i] += w[i] + x.w[i];
            ans.w[i + 1] += ans.w[i] / 10;
            ans.w[i] = ans.w[i] % 10;
        }
        if (ans.w[ans.l] != 0) ans.l++;
        return ans;
    }
    bint operator*(const bint& x) const
    {
        bint res;
        int up, tmp;
        for (int i = 0; i < l; i++)
        {
            up = 0;
            for (int j = 0; j < x.l; j++)
            {
                tmp = w[i] * x.w[j] + res.w[i + j] + up;
                res.w[i + j] = tmp % 10;
                up = tmp / 10;
            }
            if (up != 0) res.w[i + x.l] = up;
        }
```

```
        res.l = l + x.l;
        while (res.w[res.l - 1] == 0 && res.l > 1) res.l--;
        return res;
    }
    void print()
    {
        for (int i = l - 1; i >= 0; i--)
            printf("%d", w[i]);
        printf("\n");
    }
};
```

7.2.2 Complete High-precision

```
#define N 10000
class bint
{
private:
    int a[N]; // 用 N 控制最大位数
    int len; // 数字长度
public:
    // 构造函数
    bint() { len = 1, clr(a, 0); }
    // int -> bint
    bint(int n)
    {
        len = 0;
        clr(a, 0);
        int d = n;
        while (n)
            d = n / 10 * 10, a[len++] = n - d, n = d / 10;
    }
    // char[] -> int
    bint(const char s[])
    {
        clr(a, 0);
        len = 0;
        int l = strlen(s);
        for (int i = l - 1; ~i; i--) a[len++] = s[i];
    }
    // 拷贝构造函数
    bint(const bint& b)
    {
        clr(a, 0);
        len = b.len;
        for (int i = 0; i < len; i++) a[i] = b.a[i];
    }
    // 重载运算符 bint = bint
    bint& operator=(const bint& n)
    {
        len = n.len;
        for (int i = 0; i < len; i++) a[i] = n.a[i];
        return *this;
    }
    // 重载运算符 bint + bint
    bint operator+(const bint& b) const
    {
        bint t(*this);
```

```
    int res = b.len > len ? b.len : len;
    for (int i = 0; i < res; i++)
    {
        t.a[i] += b.a[i];
        if (t.a[i] >= 10) t.a[i + 1]++, t.a[i] -= 10;
    }
    t.len = res + a[res] == 0;
    return t;
}
// 重载运算符 bint - bint
bint operator-(const bint& b) const
{
    bool f = *this > b;
    bint t1 = f ? *this : b;
    bint t2 = f ? b : *this;
    int res = t1.len, j;
    for (int i = 0; i < res; 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--] += 9;
            t1.a[i] += 10 - t1.a[i];
        }
        else
            t1.a[i] -= t2.a[i];
    t1.len = res;
    while (t1.a[len - 1] == 0 && t1.len > 1) t1.len--, res--;
    if (f) t1.a[res - 1] = 0 - t1.a[res - 1];
    return t1;
}
// 重载运算符 bint * bint
bint operator*(const bint& b) const
{
    bint t;
    int i, j, up, tmp, tmp1;
    for (i = 0; i < len; i++)
    {
        up = 0;
        for (j = 0; j < b.len; j++)
        {
            tmp = a[i] * b.a[j] + t.a[i + j] + up;
            if (tmp > 9)
                tmp1 = tmp - tmp / 10 * 10, up = tmp / 10, t.a[i + j] = tmp1;
            else
                up = 0, t.a[i + j] = tmp;
        }
        if (up) t.a[i + j] = up;
    }
    t.len = i + j;
    while (t.a[t.len - 1] == 0 && t.len > 1) t.len--;
    return t;
}
// 重载运算符 bint / int
bint operator/(const int& b) const
{
    bint t;
    int down = 0;
```

```
        for (int i = len - 1; ~i; i--)
            t.a[i] = (a[i] + down * 10) / b, down = a[i] + down * 10 - t.a[i] * b;
        t.len = len;
        while (t.a[t.len - 1] == 0 && t.len > 1) t.len--;
        return t;
    }
    // 重载运算符 bint ^ n (n次方快速幂, 需保证n非负)
    bint operator^(const int n) const
    {
        bint t(*this), rt(1);
        if (n == 0) return 1;
        if (n == 1) return *this;
        int m = n;
        for (; m; m >>= 1, t = t * t)
            if (m & 1) rt = rt * t;
    }
    return rt;
    // 重载运算符 bint > bint 比较大小
    bool operator>(const bint& b) const
    {
        int p;
        if (len > b.len) return 1;
        if (len == b.len)
        {
            p = len - 1;
            while (a[p] == b.a[p] && p >= 0) p--;
            return p >= 0 && a[p] > b.a[p];
        }
        return 0;
    }
    // 重载运算符 bint > int 比较大小
    bool operator>(const int& n) const { return *this > bint(n); }
    // 输出
    void out()
    {
        printf("%d", a[len - 1]);
        for (int i = len - 2; ~i; i--) printf("%d", a[i]);
        puts("");
    }
};
```

7.3 Mo

莫队算法, 可以解决一类静态, 离线区间查询问题。分成 $\sqrt{x}$ 块, 分块排序。

```
struct query { int L, R, id; };
void solve(query node[], int m)
{
    tmp = 0;
    clr(num, 0);
    clr(ans, 0);
    sort(node, node + m, [](query a, query b) { return a.l / unit < b.l / unit || a.l /
        unit == b.l / unit && a.r < b.r; });
    int L = 1, R = 0;
    for (int i = 0; i < m; i++)
    {
        while (node[i].L < L) add(a[--L]);
```

```
        while (node[i].L > L) del(a[L++]);
        while (node[i].R < R) del(a[R--]);
        while (node[i].R > R) add(a[++R]);
        ans[node[i].id] = tmp;
    }
}
```

7.4 Fast-Scanner

```
// 适用于正负整数
template <class T>
inline bool scan_d(T &ret)
{
    char c;
    int sgn;
    if (c = getchar(), c == EOF) return 0; //EOF
    while (c != '-' && (c < '0' || c > '9')) c = getchar();
    sgn = (c == '-') ? -1 : 1;
    ret = (c == '-') ? 0 : (c - '0');
    while (c = getchar(), c >= '0' && c <= '9') ret = ret * 10 + (c - '0');
    ret *= sgn;
    return 1;
}
inline void out(int x)
{
    if (x > 9) out(x / 10);
    putchar(x % 10 + '0');
}
```