ACM/ICPC Template Manaual

CSL

October 9, 2017

Contents

0 Include

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

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

```cpp
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

```cpp
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

```
1   ll fact[100][2];
2   int getFactors(ll x)
3   {
4       int cnt = 0;
5       for (int i = 0; prime[i] <= x / prime[i]; i++)
6       {
7           fact[cnt][1] = 0;
8           if (x % prime[i] == 0)
9           {
10              fact[cnt][0] = prime[i];
11              while (x % prime[i] == 0) fact[cnt][1]++, x /= prime[i];
12              cnt++;
13          }
14      }
15      if (x != 1) fact[cnt][0] = x, fact[cnt++][1] = 1;
16      return cnt;
17  }
```

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

```
1   bool Miller_Rabin(ll n, int s)
2   {
3       if (n == 2) return 1;
4       if (n < 2 || !(n & 1)) return 0;
5       int t = 0;
6       ll  x, y, u = n - 1;
7       while ((u & 1) == 0) t++, u >>= 1;
8       for (int i = 0; i < s; i++)
9       {
10          ll a = rand() % (n - 1) + 1;
11          ll x = Pow(a, u, n);
12          for (int j = 0; j < t; j++)
13          {
14              ll y = Mul(x, x, n);
15              if (y == 1 && x != 1 && x != n - 1) return 0;
16              x = y;
17          }
18          if (x != 1) return 0;
19      }
20      return 1;
21  }
```

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
$a < b \leq 10^{12}, b - a \leq 10^6$

```
1   const int maxn = "Edit";
2   bool is_prime_small[maxn], is_prime[maxn];
3   int prime[maxn];
4   int segment_sieve(ll a, ll b)
5   {
6       int tot = 0;
```

```
7        for (ll i = 0; i * i < b; ++i)
8            is_prime_small[i] = true;
9        for (ll i = 0; i < b - a; ++i)
10           is_prime[i] = true;
11       for (ll i = 2; i * i < b; ++i)
12           if (is_prime_small[i])
13           {
14               for (ll j = 2 * i; j * j < b; j += i)
15                   is_prime_small[j] = false;
16               for (ll j = max(2LL, (a + i - 1) / i) * i; j < b; j += i)
17                   is_prime[j - a] = false;
18           }
19       for (ll i = 0; i < b - a; ++i)
20           if (is_prime[i]) prime[tot++] = i + a;
21       return tot;
22   }
```

1.2 Eular phi

1.2.1 Eular

```
1    ll Euler(ll n)
2    {
3        ll rt = n;
4        for (int i = 2; i * i <= n; i++)
5            if (n % i == 0)
6            {
7                rt -= rt / i;
8                while (n % i == 0) n /= i;
9            }
10       if (n > 1) rt -= rt / n;
11       return rt;
12   }
```

1.2.2 Sieve

```
1    const int N = "Edit";
2    int phi[N] = {0, 1};
3    void CalEuler()
4    {
5        for (int i = 2; i < N; i++)
6            if (!phi[i])
7                for (int j = i; j < N; j += i)
8                {
9                    if (!phi[j]) phi[j] = j;
10                   phi[j] = phi[j] / i * (i - 1);
11               }
12   }
```

1.3 Basic Number Theory

1.3.1 Extended Euclidean

```
1    ll exgcd(ll a, ll b, ll &x, ll &y)
2    {
3        ll d = a;
4        if (b) d = exgcd(b, a % b, y, x), y -= x * (a / b);
```

4

```
5        else x = 1, y = 0;
6        return d;
7    }
```

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

```
1    #define Mod(a, b) (((a) % (b) + (b)) % (b))
2    bool solve(ll a, ll b, ll c, ll& x, ll& y, ll& dx, ll& dy)
3    {
4        if (a == 0 && b == 0) return 0;
5        ll x0, y0;
6        ll d = exgcd(a, b, x0, y0);
7        if (c % d != 0) return 0;
8        dx = b / d, dy = a / d;
9        x = Mod(x0 * c / d, dx);
10       y = (c - a * x) / b;
11       //  y = Mod(y0 * c / d, dy); x = (c - b * y) / a;
12       return 1;
13   }
```

1.3.3   Multiplicative Inverse Modulo

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

```
1    ll inv(ll a, ll m)
2    {
3        ll x, y;
4        ll d = exgcd(a, m, x, y);
5        return d == 1 ? (x + m) % m : -1;
6    }
```

$a < m$ 且 m 为素数时，有以下两种求法

```
1    ll inv(ll a, ll m) { return a == 1 ? 1 : inv(m % a, m) * (m - m / a) % m; }
2    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 $X = re + k * mo$

```
1    void crt(ll r[], ll m[], ll n, ll &re, ll &mo)
2    {
3        mo = 1, re = 0;
4        for (int i = 0; i < n; i++) mo *= m[i];
5        for (int i = 0; i < n; i++)
6        {
7            ll x, y,  tm = mo / m[i];
8            ll d = exgcd(tm, m[i], x, y);
9            re = (re + tm * x * r[i]) % mo;
10       }
11       re = (re + mo) % mo;
12   }
```

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

```
1  bool excrt(ll r[], ll m[], ll n, ll &re, ll &mo)
2  {
3      ll x, y;
4      mo = m[0], re = r[0];
5      for (int i = 1; i < n; i++)
6      {
7          ll d = exgcd(mo, m[i],  x, y);
8          if ((r[i] - re) % d != 0) return 0;
9          x = (r[i] - re) / d * x % (m[i] / d);
10         re += x * mo;
11         mo = mo / d * m[i];
12         re %= mo;
13     }
14     re = (re + mo) % mo;
15     return 1;
16 }
```

1.5 Combinatorics

1.5.1 Combination

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

```
1  const int maxn = 1010;
2  ll C[maxn][maxn];
3  void CalComb()
4  {
5      C[0][0] = 1;
6      for (int i = 1; i < maxn; i++)
7      {
8          C[i][0] = 1;
9          for (int j = 1; j <= i; j++) C[i][j] = (C[i - 1][j - 1] + C[i - 1][j]) % mod;
10     }
11 }
```

$0 \le m \le n \le 10^5$, 模 p 为素数

```
1  const int maxn = 100010;
2  ll f[maxn];
3  void CalFact()
4  {
5      f[0] = 1;
6      for (int i = 1; i < maxn; i++) f[i] = (f[i - 1] * i) % mod;
7  }
8  ll C(int n, int m) { return f[n] * inv(f[m] * f[n - m] % mod, mod) % mod; }
```

1.5.2 Lucas

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

```
1  const int maxp = 100010;
2  ll f[maxp];
3  void CalFact(ll p)
```

```
 4  {
 5      f[0] = 1;
 6      for (int i = 1; i <= p; i++) f[i] = (f[i - 1] * i) % p;
 7  }
 8  ll Lucas(ll n, ll m, ll p)
 9  {
10      ll ret = 1;
11      while (n && m)
12      {
13          ll a = n % p, b = m % p;
14          if (a < b) return 0;
15          ret = (ret * f[a] * Pow(f[b] * f[a - b] % p, p - 2, p)) % p;
16          n /= p, m /= p;
17      }
18      return ret;
19  }
```

1.5.3 Big Combination

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

```
 1  vector<int> v;
 2  int dp[110];
 3  ll Cal(int l, int r, int k, int dis)
 4  {
 5      ll res = 1;
 6      for (int i = l; i <= r; i++)
 7      {
 8          int t = i;
 9          for (int j = 0; j < v.size(); j++)
10          {
11              int y = v[j];
12              while (t % y == 0) dp[j] += dis, t /= y;
13          }
14          res = res * (ll)t % k;
15      }
16      return res;
17  }
18  ll Comb(int n, int m, int k)
19  {
20      clr(dp, 0);
21      v.clear();
22      int tmp = k;
23      for (int i = 2; i * i <= tmp; i++)
24          if (tmp % i == 0)
25          {
26              int num = 0;
27              while (tmp % i == 0) tmp /= i, num++;
28              v.pb(i);
29          }
30      if (tmp != 1) v.pb(tmp);
31      ll ans = Cal(n - m + 1, n, k, 1);
32      for (int j = 0; j < v.size(); j++) ans = ans * Pow(v[j], dp[j], k) % k;
33      ans = ans * inv(Cal(2, m, k, -1), k) % k;
34      return ans;
35  }
```

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，$\frac{m^4+11m^2}{12}$

```
1  // 长度为n的项链串用c种颜色染
2  ll solve(int c, int n)
3  {
4      if (n == 0) return 0;
5      ll ans = 0;
6      for (int i = 1; i <= n; i++) ans += Pow(c, __gcd(i, n));
7      if (n & 1) ans += n * Pow(c, n + 1 >> 1);
8      else ans += n / 2 * (1 + c) * Pow(c, n >> 1);
9      return ans / n / 2;
10 }
```

1.6 Fast Power

```
1  ll Mul(ll a, ll b, ll mod)
2  {
3      ll t = 0;
4      for (; b; b >>= 1, a = (a << 1) % mod)
5          if (b & 1) t = (t + a) % mod;
6      return t;
7  }
8  ll Pow(ll a, ll n, ll mod)
9  {
10     ll t = 1;
11     for (; n; n >>= 1, a = (a * a % mod))
12         if (n & 1) t = (t * a % mod);
13     return t;
14 }
```

1.7 Mobius Inversion

1.7.1 Mobius

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

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

```
1  ll ans;
2  const int maxn = "Edit";
3  int n, x, prime[maxn], tot, mu[maxn];
4  bool check[maxn];
5  void calmu()
6  {
7      mu[1] = 1;
8      for (int i = 2; i < maxn; i++)
9      {
10         if (!check[i]) prime[tot++] = i, mu[i] = -1;
11         for (int j = 0; j < tot; j++)
12         {
13             if (i * prime[j] >= maxn) break;
14             check[i * prime[j]] = true;
15             if (i % prime[j] == 0)
```

```
16                {
17                    mu[i * prime[j]] = 0;
18                    break;
19                }
20                else mu[i * prime[j]] = -mu[i];
21            }
22        }
23  }
```

1.7.2 Number of Coprime-pair

有 n 个数 $(n \leq 100000)$，问这 n 个数中互质的数的对数

```
1   ll solve()
2   {
3       int b[100005];
4       ll _max, ans = 0;
5       clr(b, 0);
6       for (int i = 0; i < n; i++)
7       {
8           scanf("%d", &x);
9           if (x > _max) _max = x;
10          b[x]++;
11      }
12      for (int i = 1; i <= _max; i++)
13      {
14          int cnt = 0;
15          for (ll j = i; j <= _max; j += i) cnt += b[j];
16          ans += 1LL * mu[i] * cnt * cnt;
17      }
18      return ans - b[1]) / 2;
19  }
```

1.7.3 VisibleTrees

$gcd(x, y) = 1$ 的对数, $x \leq n, y \leq m$

```
1   ll solve(int n, int m)
2   {
3       if (n < m) swap(n, m);
4       ll ans = 0;
5       for (int i = 1; i <= m; ++i) ans += (ll)mu[i] * (n / i) * (m / i);
6       return ans;
7   }
```

1.8 Fast Transformation

1.8.1 FFT

```
1   const double PI = acos(-1.0);
2   //复数结构体
3   struct Complex
4   {
5       double x, y; //实部和虚部 x+yi
6       Complex(double _x = 0.0, double _y = 0.0) { x = _x, y = _y; }
7       Complex operator-(const Complex& b) const { return Complex(x - b.x, y - b.y); }
8       Complex operator+(const Complex& b) const { return Complex(x + b.x, y + b.y); }
```

```
 9        Complex operator*(const Complex& b) const { return Complex(x * b.x - y * b.y, x * b
          .y + y * b.x); }
10    };
11    /*
12    * 进行FFT和IFFT前的反转变换。
13    * 位置i和  (i二进制反转后位置) 互换
14    * len必须取2的幂
15    */
16    void change(Complex y[], int len)
17    {
18        for (int i = 1, j = len / 2; i < len - 1; i++)
19        {
20            if (i < j) swap(y[i], y[j]);
21            //交换互为小标反转的元素，i<j保证交换一次
22            //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
23            int k = len / 2;
24            while (j >= k) j -= k, k /= 2;
25            if (j < k) j += k;
26        }
27    }
28    /*
29    * 做FFT
30    * len必须为2^k形式,
31    * on==1时是DFT, on==-1时是IDFT
32    */
33    void fft(Complex y[], int len, int on)
34    {
35        change(y, len);
36        for (int h = 2; h <= len; h <<= 1)
37        {
38            Complex wn(cos(-on * 2 * PI / h), sin(-on * 2 * PI / h));
39            for (int j = 0; j < len; j += h)
40            {
41                Complex w(1, 0);
42                for (int k = j; k < j + h / 2; k++)
43                {
44                    Complex u = y[k];
45                    Complex t = w * y[k + h / 2];
46                    y[k] = u + t, y[k + h / 2] = u - t;
47                    w = w * wn;
48                }
49            }
50        }
51        if (on == -1)
52            for (int i = 0; i < len; i++) y[i].x /= len;
53    }
```

1.9 Others

1.9.1 Digit

n^n 最左边一位数

```
1    int leftmost(int n)
2    {
3        double m = n * log10((double)n);
4        double g = m - (ll)m;
5        return (int)pow(10.0, g);
6    }
```

$n!$ 位数

```
1  int count(ll n)
2  {
3      if (n == 1) return 1;
4      return (int)ceil(0.5 * log10(2 * M_PI * n) + n * log10(n) - n * log10(M_E));
5  }
```

1.9.2 Josephus

n 个人围成一圈，从第一个开始报数，第 m 个将被杀掉

```
1  int josephus(int n, int m)
2  {
3      int r = 0;
4      for (int k = 1; k <= n; ++k) r = (r + m) % k;
5      return r + 1;
6  }
```

1.10 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 $gcd(n, n-i) = 1 (1 \leq i \leq n)$

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 $\pi(n)$，$\lim_{n \to \infty} \pi(n) = \frac{n}{\ln n}$

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
    $h(0) = h(1) = 1, h(n) = \frac{(4n-2)h(n-1)}{n+1} = \frac{C_{2n}^n}{n+1} = C_{2n}^n - C_{2n}^{n-1}$

17. 线性递推:

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

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

2 String Processing

2.1 KMP

```
1  // 返回y中x的个数
2  const int N = "Edit";
3  int next[N];
4  void initkmp(char x[], int m)
5  {
6      int i = 0, j = next[0] = -1;
7      while (i < m)
8      {
9          while (j != -1 && x[i] != x[j]) j = next[j];
10         next[++i] = ++j;
11     }
12 }
13 int kmp(char x[], int m, char y[], int n)
14 {
15     int i, j, ans;
16     i = j = ans = 0;
17     initkmp(x, m);
18     while (i < n)
19     {
20         while (j != -1 && y[i] != x[j]) j = next[j];
21         i++, j++;
22         if (j >= m) ans++, j = next[j];
23     }
24     return ans;
25 }
```

2.2 ExtendKMP

```
1  //next[i]:x[i...m-1]与x[0...m-1]的最长公共前缀
2  //extend[i]:y[i...n-1]与x[0...m-1]的最长公共前缀
3  const int N = "Edit";
4  int next[N], extend[N];
5  void pre_ekmp(char x[], int m)
6  {
7      next[0] = m;
8      int j = 0;
9      while (j + 1 < m && x[j] == x[j + 1]) j++;
10     next[1] = j;
11     int k = 1;
12     for (int i = 2; i < m; i++)
13     {
14         int p = next[k] + k - 1;
15         int L = next[i - k];
16         if (i + L < p + 1)
17             next[i] = L;
18         else
19         {
20             j = max(0, p - i + 1);
21             while (i + j < m && x[i + j] == x[j]) j++;
22             next[i] = j;
23             k = i;
24         }
25     }
26 }
```

```
27  void ekmp(char x[], int m, char y[], int n)
28  {
29      pre_ekmp(x, m, next);
30      int j = 0;
31      while (j < n && j < m && x[j] == y[j]) j++;
32      extend[0] = j;
33      int k = 0;
34      for (int i = 1; i < n; i++)
35      {
36          int p = extend[k] + k - 1;
37          int L = next[i - k];
38          if (i + L < p + 1)
39              extend[i] = L;
40          else
41          {
42              j = max(0, p - i + 1);
43              while (i + j < n && j < m && y[i + j] == x[j]) j++;
44              extend[i] = j, k = i;
45          }
46      }
47  }
```

2.3 Manacher

O(n) 求解最长回文子串

```
1   const int N = "Edit";
2   char s[N], str[N << 1];
3   int p[N << 1];
4   void Manacher(char s[], int& n)
5   {
6       str[0] = '$', str[1] = '#';
7       for (int i = 0; i < n; i++) str[(i << 1) + 2] = s[i], str[(i << 1) + 3] = '#';
8       n = 2 * n + 2;
9       str[n] = 0;
10      int mx = 0, id;
11      for (int i = 1; i < n; i++)
12      {
13          p[i] = mx > i ? min(p[2 * id - i], mx - i) : 1;
14          while (str[i - p[i]] == str[i + p[i]]) p[i]++;
15          if (p[i] + i > mx) mx = p[i] + i, id = i;
16      }
17  }
18  int solve(char s[])
19  {
20      int n = strlen(s);
21      Manacher(s, n);
22      return *max_elememt(p, p + n) - 1;
23  }
```

2.4 Aho-Corasick Automaton

```
1   const int maxn = "Edit";
2   struct Trie
3   {
4       int ch[maxn][26], f[maxn], val[maxn];
5       int sz, rt;
```

```
6        int newnode() { clr(ch[sz], -1), val[sz] = 0; return sz++; }
7        void init() { sz = 0, rt = newnode(); }
8        inline int idx(char c) { return c - 'A'; };
9        void insert(const char* s)
10       {
11           int u = 0, n = strlen(s);
12           for (int i = 0; i < n; i++)
13           {
14               int c = idx(s[i]);
15               if (ch[u][c] == -1) ch[u][c] = newnode();
16               u = ch[u][c];
17           }
18           val[u]++;
19       }
20       void build()
21       {
22           queue<int> q;
23           f[rt] = rt;
24           for (int c = 0; c < 26; c++)
25           {
26               if (~ch[rt][c])
27                   f[ch[rt][c]] = rt, q.push(ch[rt][c]);
28               else
29                   ch[rt][c] = rt;
30           }
31           while (!q.empty())
32           {
33               int u = q.front();
34               q.pop();
35               // val[u] |= val[f[u]];
36               for (int c = 0; c < 26; c++)
37               {
38                   if (~ch[u][c])
39                       f[ch[u][c]] = ch[f[u]][c], q.push(ch[u][c]);
40                   else
41                       ch[u][c] = ch[f[u]][c];
42               }
43           }
44       }
45       //返回主串中有多少模式串
46       int query(const char* s)
47       {
48           int u = rt, n = strlen(s);
49           int res = 0;
50           for (int i = 0; i < n; i++)
51           {
52               int c = idx(s[i]);
53               u = ch[u][c];
54               int tmp = u;
55               while (tmp != rt)
56               {
57                   res += val[tmp];
58                   val[tmp] = 0;
59                   tmp = f[tmp];
60               }
61           }
62           return res;
63       }
64   };
```

2.5 Suffix Array

```
1   //倍增算法构造后缀数组,复杂度O(nlogn)
2   const int maxn = "Edit";
3   char s[maxn];
4   int sa[maxn], t[maxn], t2[maxn], c[maxn], rank[maxn], height[maxn];
5   //n为字符串的长度,字符集的值为0~m-1
6   void build_sa(int m, int n)
7   {
8       n++;
9       int *x = t, *y = t2;
10      //基数排序
11      for (int i = 0; i < m; i++) c[i] = 0;
12      for (int i = 0; i < n; i++) c[x[i] = s[i]]++;
13      for (int i = 1; i < m; i++) c[i] += c[i - 1];
14      for (int i = n - 1; ~i; i--) sa[--c[x[i]]] = i;
15      for (int k = 1; k <= n; k <<= 1)
16      {
17          //直接利用sa数组排序第二关键字
18          int p = 0;
19          for (int i = n - k; i < n; i++) y[p++] = i;
20          for (int i = 0; i < n; i++)
21              if (sa[i] >= k) y[p++] = sa[i] - k;
22          //基数排序第一关键字
23          for (int i = 0; i < m; i++) c[i] = 0;
24          for (int i = 0; i < n; i++) c[x[y[i]]]++;
25          for (int i = 0; i < m; i++) c[i] += c[i - 1];
26          for (int i = n - 1; ~i; i--) sa[--c[x[y[i]]]] = y[i];
27          //根据sa和y数组计算新的x数组
28          swap(x, y);
29          p = 1;
30          x[sa[0]] = 0;
31          for (int i = 1; i < n; i++)
32              x[sa[i]] = y[sa[i - 1]] == y[sa[i]] && y[sa[i - 1] + k] == y[sa[i] + k] ? p
     - 1 : p++;
33          if (p >= n) break; //以后即使继续倍增，sa也不会改变，退出
34          m = p;              //下次基数排序的最大值
35      }
36      n--;
37      int k = 0;
38      for (int i = 0; i <= n; i++) rank[sa[i]] = i;
39      for (int i = 0; i < n; i++)
40      {
41          if (k) k--;
42          int j = sa[rank[i] - 1];
43          while (s[i + k] == s[j + k]) k++;
44          height[rank[i]] = k;
45      }
46  }
47
48  int dp[maxn][30];
49  void initrmq(int n)
50  {
51      for (int i = 1; i <= n; i++)
52          dp[i][0] = height[i];
53      for (int j = 1; (1 << j) <= n; j++)
54          for (int i = 1; i + (1 << j) - 1 <= n; i++)
55              dp[i][j] = min(dp[i][j - 1], dp[i + (1 << (j - 1))][j - 1]);
56  }
```

```
57  int rmq(int l, int r)
58  {
59      int k = 0;
60      while ((1 << (k + 1)) <= r - l + 1) k++;
61      return min(dp[l][k], dp[r - (1 << k) + 1][k]);
62  }
63  // 求两个后缀的最长公共前缀
64  int lcp(int a, int b)
65  {
66      a = rank[a], b = rank[b];
67      if (a > b) swap(a, b);
68      return rmq(a + 1, b);
69  }
```

3 Data Structure

3.1 Binary Indexed Tree

$O(\log n)$ 查询和修改数组的前缀和

```
1  // 注意下标应从1开始 n是全局变量
2  const int maxn = "Edit";
3  int bit[N], n;
4  int sum(int x)
5  {
6      int s = 0;
7      for (int i = x; i; i -= i & -i)
8          s += bit[i];
9      return s;
10 }
11 void add(int x, int v)
12 {
13     for (int i = x; i <= n; i += i & -i)
14         bit[i] += v;
15 }
```

3.2 Segment Tree

```
1  #define lson rt << 1          // 左儿子
2  #define rson rt << 1 | 1      // 右儿子
3  #define Lson l, m, lson       // 左子树
4  #define Rson m + 1, r, rson   // 右子树
5  void PushUp(int rt);          // 用lson和rson更新rt
6  void PushDown(int rt[, int m]);              // rt的标记下移，m为区间长度（若与标记有关）
7  void build(int l, int r, int rt);            // 以rt为根节点，对区间[l, r]建立线段树
8  void update([...,] int l, int r, int rt)     // rt[l, r]内寻找目标并更新
9  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

```
1  const int maxn = "Edit";
2  int sum[maxn << 2];
3  void PushUp(int rt) { sum[rt] = sum[lson] + sum[rson]; }
4  void build(int l, int r, int rt)
5  {
6      if (l == r)
7      {
8          scanf("%d", &sum[rt]);
9          return;
10     } // 建立的时候直接输入叶节点
11     int m = (l + r) >> 1;
12     build(Lson);
13     build(Rson);
14     PushUp(rt);
15 }
16 void update(int p, int add, int l, int r, int rt)
17 {
18     if (l == r)
19     {
20         sum[rt] += add;
21         return;
```

```
22        }
23        int m = (l + r) >> 1;
24        if (p <= m) update(p, add, Lson);
25        else update(p, add, Rson);
26        PushUp(rt);
27  }
28  int query(int L, int R, int l, int r, int rt)
29  {
30        if (L <= l && r <= R) return sum[rt];
31        int m = (l + r) >> 1, s = 0;
32        if (L <= m) s += query(L, R, Lson);
33        if (m < R) s += query(L, R, Rson);
34        return s;
35  }
```

3.2.2 Interval Update

```
1   // seg[rt]用于存放懒惰标记，注意PushDown时标记的传递
2   const int maxn = "Edit";
3   int seg[maxn << 2], sum[maxn << 2];
4
5   void PushUp(int rt) { sum[rt] = sum[lson] + sum[rson]; }
6   void PushDown(int rt, int m)
7   {
8         if (seg[rt] == 0) return;
9         seg[lson] += seg[rt];
10        seg[rson] += seg[rt];
11        sum[lson] += seg[rt] * (m - (m >> 1));
12        sum[rson] += seg[rt] * (m >> 1);
13        seg[rt] = 0;
14  }
15  void build(int l, int r, int rt)
16  {
17        seg[rt] = 0;
18        if (l == r)
19        {
20              scanf("%lld", &sum[rt]);
21              return;
22        }
23        int m = (l + r) >> 1;
24        build(Lson);
25        build(Rson);
26        PushUp(rt);
27  }
28  void update(int L, int R, int add, int l, int r, int rt)
29  {
30        if (L <= l && r <= R)
31        {
32              seg[rt] += add;
33              sum[rt] += add * (r - l + 1);
34              return;
35        }
36        PushDown(rt, r - l + 1);
37        int m = (l + r) >> 1;
38        if (L <= m) update(L, R, add, Lson);
39        if (m < R) update(L, R, add, Rson);
40        PushUp(rt);
41  }
```

```
42  int query(int L, int R, int l, int r, int rt)
43  {
44      if (L <= l && r <= R) return sum[rt];
45      PushDown(rt, r - l + 1);
46      int m = (l + r) >> 1, ret = 0;
47      if (L <= m) ret += query(L, R, Lson);
48      if (m < R) ret += query(L, R, Rson);
49      return ret;
50  }
```

3.3 Partition Tree

```
1   #define Lson l, m, dep + 1
2   #define Rson m + 1, r, dep + 1
3
4   int tree[20][maxn];    //表示每层每个位置的值
5   int sorted[maxn];      //已经排序好的数
6   int toleft[20][maxn]; //toleft[p][i]表示第i层从1到i有数分入左边
7   void build(int l, int r, int dep)
8   {
9       if (l == r) return;
10      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
11      for (int i = l; i <= r; i++)
12          if (tree[dep][i] < sorted[m])
13              same--;
14      int lpos = l;
15      int rpos = m + 1;
16      for (int i = l; i <= r; i++)
17      {
18          if (tree[dep][i] < sorted[m])
19              tree[dep + 1][lpos++] = tree[dep][i];
20          else if (tree[dep][i] == sorted[m] && same > 0)
21          {
22              tree[dep + 1][lpos++] = tree[dep][i];
23              same--;
24          }
25          else
26              tree[dep + 1][rpos++] = tree[dep][i];
27          toleft[dep][i] = toleft[dep][l - 1] + lpos - l;
28      }
29      build(Lson);
30      build(Rson);
31  }
32  //查询区间第k小的数
33  int query(int L, int R, int k, int l, int r, int dep)
34  {
35      if (L == R) return tree[dep][L];
36      int m = (l + r) >> 1;
37      int cnt = toleft[dep][R] - toleft[dep][L - 1];
38      if (cnt >= k)
39      {
40          int newl = l + toleft[dep][L - 1] - toleft[dep][l - 1];
41          int newr = newl + cnt - 1;
42          return query(newl, newr, k, Lson);
43      }
44      else
45      {
46          int newr = R + toleft[dep][r] - toleft[dep][R];
```

```
47            int newl = newr - (R - L - cnt);
48            return query(newl, newr, k - cnt, Rson);
49        }
50 }
```

3.4 Functional Segment Tree

```
1  // 静态查询区间第k小的值
2  const int maxn = "Edit";
3  int a[maxn], rt[maxn];
4  int cnt;
5  int lson[maxn << 5], rson[maxn << 5], sum[maxn << 5];
6  #define Lson l, m, lson[x], lson[y]
7  #define Rson m + 1, r, rson[x], rson[y]
8
9  void update(int p, int l, int r, int& x, int y)
10 {
11     lson[++cnt] = lson[y], rson[cnt] = rson[y], sum[cnt] = sum[y] + 1, x = cnt;
12     if (l == r) return;
13     int m = (l + r) >> 1;
14     if (p <= m) update(p, Lson);
15     else update(p, Rson);
16 }
17 int query(int l, int r, int x, int y, int k)
18 {
19     if (l == r) return l;
20     int m = (l + r) >> 1;
21     int s = sum[lson[y]] - sum[lson[x]];
22     if (s >= k) return query(Lson, k);
23     else return query(Rson, k - s);
24 }
```

3.5 RMQ

```
1  const int maxn = "Edit";
2  int mmax[maxn][30], mmin[maxn][30];
3  int a[maxn], n, k;
4  void init()
5  {
6      for (int i = 1; i <= n; i++) mmax[i][0] = mmin[i][0] = a[i];
7      for (int j = 1; (1 << j) <= n; j++)
8          for (int i = 1; i + (1 << j) - 1 <= n; i++)
9          {
10             mmax[i][j] = max(mmax[i][j - 1], mmax[i + (1 << (j - 1))][j - 1]);
11             mmin[i][j] = min(mmin[i][j - 1], mmin[i + (1 << (j - 1))][j - 1]);
12         }
13 }
14 // op=0/1 返回[l,r]最大/小值
15 int rmq(int l, int r, int op)
16 {
17     int k = 0;
18     while ((1 << (k + 1)) <= r - l + 1) k++;
19     if (op == 0)
20         return max(mmax[l][k], mmax[r - (1 << k) + 1][k]);
21     return min(mmin[l][k], mmin[r - (1 << k) + 1][k]);
22 }
23
```

```
24  // 二维RMQ
25  void init()
26  {
27      for (int i = 0; (1 << i) <= n; i++)
28          for (int j = 0; (1 << j) <= m; j++)
29          {
30              if (i == 0 && j == 0) continue;
31              for (int row = 1; row + (1 << i) - 1 <= n; row++)
32                  for (int col = 1; col + (1 << j) - 1 <= m; col++)
33                      //当x或y等于0的时候，就相当于一维的RMQ了
34                      if (i == 0)
35                          dp[row][col][i][j] = max(dp[row][col][i][j - 1],
36                                                   dp[row][col + (1 << (j - 1))][i][j - 1]);
37                      else if (j == 0)
38                          dp[row][col][i][j] = max(dp[row][col][i - 1][j],
39                                                   dp[row + (1 << (i - 1))][col][i - 1][j]);
40                      else
41                          dp[row][col][i][j] = max(dp[row][col][i][j - 1],
42                                                   dp[row][col + (1 << (j - 1))][i][j - 1]);
43          }
44  }
45  int rmq(int x1, int y1, int x2, int y2)
46  {
47      int kx = 0, ky = 0;
48      while (x1 + (1 << (1 + kx)) - 1 <= x2) kx++;
49      while (y1 + (1 << (1 + ky)) - 1 <= y2) ky++;
50      int m1 = dp[x1][y1][kx][ky];
51      int m2 = dp[x2 - (1 << kx) + 1][y1][kx][ky];
52      int m3 = dp[x1][y2 - (1 << ky) + 1][kx][ky];
53      int m4 = dp[x2 - (1 << kx) + 1][y2 - (1 << ky) + 1][kx][ky];
54      return max(max(m1, m2), max(m3, m4));
55  }
```

3.6 Treap

```
1   struct Node
2   {
3       Node* ch[2]; // 左右子树
4       int r;       // 随机优先级
5       int v;       // 值
6       int s;       // 结点总数
7       Node(int v = 0) : v(v)
8       {
9           ch[0] = ch[1] = NULL;
10          r = rand();
11          s = 1;
12      }
13      int cmp(int x) const
14      {
15          if (x == v) return -1;
16          return x < v ? 0 : 1;
17      }
18      void maintain()
19      {
20          s = 1;
21          if (ch[0] != NULL) s += ch[0]->s;
22          if (ch[1] != NULL) s += ch[1]->s;
23      }
```

```
24  };
25  void rotate(Node*& o, int d)
26  {
27      Node* k = o->ch[d ^ 1];
28      o->ch[d ^ 1] = k->ch[d];
29      k->ch[d] = o;
30      o->maintain();
31      k->maintain();
32      o = k;
33  }
34  void insert(Node*& o, int x)
35  {
36      if (o == NULL) o = new Node(x);
37      else
38      {
39          int d = (x < o->v ? 0 : 1); // 不要用cmp函数，因为可能会有相同结点
40          insert(o->ch[d], x);
41          if (o->ch[d]->r > o->r) rotate(o, d ^ 1);
42      }
43      o->maintain();
44  }
45  Node* find(Node* o, int x)
46  {
47      if (o == NULL) return NULL;
48      if (x == o->v) return o;
49      return x < o->v ? find(o->ch[0], x) : find(o->ch[1], x);
50  }
51  // 要确保结点存在
52  void remove(Node*& o, int x)
53  {
54      int d = o->cmp(x);
55      int ret = 0;
56      if (d == -1)
57      {
58          Node* u = o;
59          if (o->ch[0] != NULL && o->ch[1] != NULL)
60          {
61              int d2 = (o->ch[0]->r > o->ch[1]->r ? 1 : 0);
62              rotate(o, d2);
63              remove(o->ch[d2], x);
64          }
65          else
66          {
67              if (o->ch[0] == NULL) o = o->ch[1];
68              else o = o->ch[0];
69              delete u;
70          }
71      }
72      else remove(o->ch[d], x);
73      if (o != NULL) o->maintain();
74  }
75  int kth(Node* o, int k)
76  {
77      if (o == NULL || k <= 0 || k > o->s) return 0;
78      int s = (o->ch[0] == NULL ? 0 : o->ch[0]->s);
79      if (k == s + 1) return o->v;
80      else if (k <= s) return kth(o->ch[0], k);
81      else return kth(o->ch[1], k - s - 1);
82  }
```

```
83   // 在以o为根的子树中，值比x小的结点总数加1
84   int rank(Node* o, int x)
85   {
86       if (o == NULL) return 1;
87       if (x <= o->v) return rank(o->ch[0], x);
88       return rank(o->ch[1], x) + (o->ch[0] == NULL ? 0 : o->ch[0]->s) + 1;
89   }
```

3.7 Slpay

```
1    struct Node
2    {
3        Node* ch[2];
4        int s;
5        int flip;
6        int v;
7        int cmp(int k) const
8        {
9            int d = k - ch[0]->s;
10           if (d == 1) return -1;
11           return d <= 0 ? 0 : 1;
12       }
13       void maintain() { s = ch[0]->s + ch[1]->s + 1; }
14       void pushdown()
15       {
16           if (flip)
17           {
18               flip = 0;
19               swap(ch[0], ch[1]);
20               ch[0]->flip = !ch[0]->flip;
21               ch[1]->flip = !ch[1]->flip;
22           }
23       }
24   };
25   Node* null = new Node();
26   void rotate(Node*& o, int d)
27   {
28       Node* k = o->ch[d ^ 1];
29       o->ch[d ^ 1] = k->ch[d];
30       k->ch[d] = o;
31       o->maintain();
32       k->maintain();
33       o = k;
34   }
35   void splay(Node*& o, int k)
36   {
37       o->pushdown();
38       int d = o->cmp(k);
39       if (d == 1) k -= o->ch[0]->s + 1;
40       if (d != -1)
41       {
42           Node* p = o->ch[d];
43           p->pushdown();
44           int d2 = p->cmp(k);
45           int k2 = (d2 == 0 ? k : k - p->ch[0]->s - 1);
46           if (d2 != -1)
47           {
48               splay(p->ch[d2], k2);
```

```
49                if (d == d2) rotate(o, d ^ 1);
50                else rotate(o->ch[d], d);
51            }
52            rotate(o, d ^ 1);
53        }
54    }
55    // 合并left和right。假定left的所有元素比right小。注意right可以是null，但left不可以
56    Node* merge(Node* left, Node* right)
57    {
58        splay(left, left->s);
59        left->ch[1] = right;
60        left->maintain();
61        return left;
62    }
63    // 把o的前k小结点放在left里，其他的放在right里。1<=k<=o->s。当k=o->s时，right=null
64    void split(Node* o, int k, Node*& left, Node*& right)
65    {
66        splay(o, k);
67        left = o;
68        right = o->ch[1];
69        o->ch[1] = null;
70        left->maintain();
71    }
72    const int maxn = "Edit";
73    struct SplaySequence
74    {
75        int n;
76        Node seq[maxn];
77        Node* root;
78        Node* build(int sz)
79        {
80            if (!sz) return null;
81            Node* L = build(sz / 2);
82            Node* o = &seq[++n];
83            o->v = n; // 节点编号
84            o->ch[0] = L;
85            o->ch[1] = build(sz - sz / 2 - 1);
86            o->flip = o->s = 0;
87            o->maintain();
88            return o;
89        }
90        void init(int sz)
91        {
92            n = 0;
93            null->s = 0;
94            root = build(sz);
95        }
96    };
```

4 Graph Theory

4.1 Union-Find Set

```cpp
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

```cpp
typedef pair<int, PII> Edge;
vector<Edge> 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++)
    {
        Edge p = G[i];
        int x = p.Y.X, 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

```
1   // 耗费矩阵cost[][],标号从0开始,0~n-1
2   // 返回最小生成树的权值,返回-1表示原图不连通
3   const int maxn = "Edit";
4   bool vis[maxn];
5   int lowc[maxn];
6   int Prim(int cost[][maxn], int n)
7   {
8       int ans = 0;
9       clr(vis, 0);
10      vis[0] = 1;
11      for (int i = 1; i < n; i++)
12          lowc[i] = cost[0][i];
13      for (int i = 1; i < n; i++)
14      {
15          int minc = INF;
16          int p = -1;
17          for (int j = 0; j < n; j++)
18              if (!vis[j] && minc > lowc[j])
19              {
20                  minc = lowc[j];
21                  p = j;
22              }
23          if (minc == INF) return -1;
24          vis[p] = 1;
25          ans += minc;
26          for (int j = 0; j < n; j++)
27              if (!vis[j] && lowc[j] > cost[p][j])
28                  lowc[j] = cost[p][j];
29      }
30      return ans;
31  }
```

4.3 Shortest Path

4.3.1 Dijkstra

```
1   // pair<权值, 点>
2   // 记得初始化
3   const int maxn = "Edit";
4   typedef pair<int, int> PII;
5   typedef vector<PII> VII;
6   VII G[maxn];
7   int vis[maxn], dis[maxn];
8   void init(int n)
9   {
10      for (int i = 0; i < n; i++) G[i].clear();
11  }
12  void add_edge(int u, int v, int w) { G[u].pb(mp(w, v)); }
13  void Dijkstra(int s, int n)
14  {
15      clr(vis, 0), clr(dis, 0x3f);
16      dis[s] = 0;
17      priority_queue<PII, VII, greater<PII> > q;
18      q.push(mp(dis[s], s));
19      while (!q.empty())
20      {
21          PII t = q.top();
```

```
22          int x = t.Y;
23          q.pop();
24          if (vis[x]) continue;
25          vis[x] = 1;
26          for (int i = 0; i < G[x].size(); i++)
27          {
28              int y = G[x][i].Y, w = G[x][i].X;
29              if (!vis[y] && dis[y] > dis[x] + w)
30              {
31                  dis[y] = dis[x] + w;
32                  q.push(mp(dis[y], y));
33              }
34          }
35      }
36  }
```

4.3.2 SPFA

```
1   // G[u] = mp(v, w)
2   // SPFA()返回0表示存在负环
3   const int maxn = "Edit";
4   vector<PII> G[maxn];
5   bool vis[maxn];
6   int dis[maxn];
7   int inqueue[maxn];
8   void init(int n)
9   {
10      for (int i = 0; i < n; i++) G[i].clear();
11  }
12  void add_edge(int u, int v, int w) { G[u].pb(mp(v, w)); }
13  bool SPFA(int s, int n)
14  {
15      clr(vis, 0), clr(dis, 0x3f), clr(inqueue, 0);
16      dis[s] = 0;
17      queue<int> q; // 待优化的节点入队
18      q.push(s);
19      while (!q.empty())
20      {
21          int x = q.front();
22          q.pop();
23          vis[x] = false;
24          for (int i = 0; i < G[x].size(); i++)
25          {
26              int y = G[x][i].X, w = G[x][i].Y;
27              if (dis[y] > dis[x] + w)
28              {
29                  dis[y] = dis[x] + w;
30                  if (!vis[y])
31                  {
32                      q.push(y);
33                      vis[y] = true;
34                      if (++inqueue[y] >= n) return 0;
35                  }
36              }
37          }
38      }
39      return 1;
40  }
```

4.3.3 Floyd

$O(n^3)$ 求出任意两点间最短路

```
1   // 领接矩阵存图需注意判断重边
2   const int maxn = "Edit";
3   int G[maxn][maxn];
4   void init(int n)
5   {
6       clr(G, 0x3f);
7       for (int i = 0; i < n; i++) G[i][i] = 0;
8   }
9   void add_edge(int u, int v, int w) { G[u][v] = min(G[u][v], w); }
10  void Floyd(int n)
11  {
12      for (int k = 0; k < n; k++)
13          for (int i = 0; i < n; i++)
14              for (int j = 0; j < n; j++)
15                  G[i][j] = min(G[i][j], G[i][k] + G[k][j]);
16  }
```

4.4 Topo Sort

4.4.1 Matrix

```
1   // 存图前记得初始化
2   // Ans存放拓排结果, G为邻接矩阵, deg为入度信息
3   // 排序成功返回1, 存在环返回0
4   const int maxn = "Edit";
5   int Ans[maxn];      // 存放拓扑排序结果
6   int G[maxn][maxn]; // 存放图信息
7   int deg[maxn];      // 存放点入度信息
8   void init() { clr(G, 0), clr(deg, 0), clr(Ans, 0); }
9   void add_edge(int u, int v)
10  {
11      if (G[u][v]) return;
12      G[u][v] = 1, deg[v]++;
13  }
14  bool Toposort(int n)
15  {
16      int tot = 0;
17      queue<int> q;
18      for (int i = 0; i < n; ++i)
19          if (deg[i] == 0) q.push(i);
20      while (!q.empty())
21      {
22          int v = q.front();
23          q.pop();
24          Ans[tot++] = v;
25          for (int i = 0; i < n; ++i)
26              if (G[v][i] == 1)
27                  if (--deg[i] == 0) q.push(i);
28      }
29      if (tot < n) return false;
30      return true;
31  }
```

4.4.2 List

```
1  // 存图前记得初始化
2  // Ans排序结果，G邻接表，deg入度，map用于判断重边
3  // 排序成功返回1，存在环返回0
4  const int maxn = "Edit";
5  int Ans[maxn];
6  vector<int> G[maxn];
7  int deg[maxn];
8  map<PII, bool> S;
9  void init(int n)
10 {
11     S.clear();
12     for (int i = 0; i < n; i++) G[i].clear();
13     clr(deg, 0), clr(Ans, 0);
14 }
15 void add_edge(int u, int v)
16 {
17     if (S[mp(u, v)]) return;
18     G[u].pb(v);
19     S[mp(u, v)] = 1;
20     deg[v]++;
21 }
22 bool Toposort(int n)
23 {
24     int tot = 0;
25     queue<int> q;
26     for (int i = 0; i < n; ++i)
27         if (deg[i] == 0) q.push(i);
28     while (!q.empty())
29     {
30         int v = q.front();
31         que.pop();
32         Ans[tot++] = v;
33         for (int i = 0; i < G[v].size(); ++i)
34         {
35             int t = G[v][i];
36             if (--deg[t] == 0) q.push(t);
37         }
38     }
39     if (tot < n) return false;
40     return true;
41 }
```

4.5 LCA

4.5.1 Tarjan

```
1  // Tarjan离线算法
2  // 时间复杂度O(n+q)
3  const int maxn = "Edit";
4  int par[maxn];                      //并查集
5  int ans[maxn];                      //存储答案
6  vector<int> G[maxn];                //邻接表
7  vector<int> query[maxn], num[maxn]; //存储查询信息
8  bool vis[maxn];                     //是否被遍历
9  inline void init(int n)
10 {
11     for (int i = 1; i <= n; i++)
```

```
12      {
13          G[i].clear();
14          query[i].clear();
15          num[i].clear();
16          par[i] = i;
17          vis[i] = 0;
18      }
19  }
20  inline void add_edge(int u, int v) { G[u].pb(v); }
21  inline void add_query(int id, int u, int v)
22  {
23      query[u].pb(v), query[v].pb(u);
24      num[u].pb(id), num[v].pb(id);
25  }
26  void tarjan(int u)
27  {
28      vis[u] = 1;
29      for (int i = 0; i < G[u].size(); i++)
30      {
31          int v = G[u][i];
32          if (vis[v]) continue;
33          tarjan(v);
34          unite(u, v);
35      }
36      for (int i = 0; i < query[u].size(); i++)
37      {
38          int v = query[u][i];
39          if (!vis[v]) continue;
40          ans[num[u][i]] = find(v);
41      }
42  }
```

4.5.2 DFS+ST

```
 1  // DFS+ST在线算法
 2  // 时间复杂度O(nlogn+q)
 3  const int maxn = "Edit";
 4  vector<int> G[maxn];
 5  int dfs_clock;
 6  int pos[maxn], f[maxn << 1], dep[maxn << 1], dp[maxn << 1][30];
 7  inline void init(int n)
 8  {
 9      for (int i = 0; i < n; i++) G[i].clear();
10      dfs_clock = 0;
11  }
12  inline void add_edge(int u, int v) { G[u].pb(v); }
13  void dfs(int u, int pre, int depth)
14  {
15      f[++dfs_clock] = u;       //记录遍历顺序
16      pos[u] = dfs_clock;       //记录某个节点在f中第一次出现的位置
17      dep[dfs_clock] = depth; //记录路径
18      for (int i = 0; i < G[i].size(); i++)
19      {
20          int v = G[u][i];
21          if (v == pre) continue;
22          dfs(v, u, depth + 1);
23          f[++dfs_clock] = u;
24          dep[dfs_clock] = depth;
```

```
25         }
26    }
27    void initrmq(int n) // n = dfs_clock
28    {
29        for (int i = 1; i <= n; i++) dp[i][0] = i;
30        for (int j = 1; (1 << j) <= n; j++)
31            for (int i = 0; i + (1 << j) - 1 <= tot; i++)
32            {
33                if (dep[dp[i][j - 1]] < dep[dp[i + (1 << (j - 1))][j - 1]])
34                    dp[i][j] = dp[i][j - 1];
35                else
36                    dp[i][j] = dp[i + (1 << (j - 1))][j - 1];
37            }
38    }
39    int rmq(int l, int r)
40    {
41        l = pos[l], r = pos[r];
42        if (l > r) swap(l, r);
43        int k = 0;
44        while ((1 << (k + 1)) <= r - l + 1) k++;
45        return (dep[l][k] < dep[r - (1 << k) + 1][k]) ? dp[l][k] : dp[r - (1 << k) + 1][k];
46    }
```

4.6   Biconnected Component

```
1    //割顶的bccno无意义
2    const int maxn = "Edit";
3    int pre[maxn], iscut[maxn], bccno[maxn], dfs_clock, bcc_cnt;
4    vector<int> G[maxn], bcc[maxn];
5    stack<PII> s;
6    void init(int n)
7    {
8        for (int i = 0; i < n; i++) G[i].clear();
9    }
10   inline void add_edge(int u, int v) { G[u].pb(v), G[v].pb(u); }
11   int dfs(int u, int fa)
12   {
13       int lowu = pre[u] = ++dfs_clock;
14       int child = 0;
15       for (int i = 0; i < G[u].size(); i++)
16       {
17           int v = G[u][i];
18           PII e = mp(u, v);
19           if (!pre[v])
20           {
21               //没有访问过v
22               s.push(e);
23               child++;
24               int lowv = dfs(v, u);
25               lowu = min(lowu, lowv); //用后代的low函数更新自己
26               if (lowv >= pre[u])
27               {
28                   iscut[u] = true;
29                   bcc_cnt++;
30                   bcc[bcc_cnt].clear(); //注意! bcc从1开始编号
31                   for (;;)
32                   {
33                       PII x = s.top();
```

```
34                    s.pop();
35                    if (bccno[x.X] != bcc_cnt)
36                        bcc[bcc_cnt].pb(x.X), bccno[x.X] = bcc_cnt;
37                    if (bccno[x.Y] != bcc_cnt)
38                        bcc[bcc_cnt].pb(x.Y), bccno[x.Y] = bcc_cnt;
39                    if (x.X == u && x.Y == v) break;
40                }
41            }
42        }
43        else if (pre[v] < pre[u] && v != fa)
44        {
45            s.push(e);
46            lowu = min(lowu, pre[v]); //用反向边更新自己
47        }
48    }
49    if (fa < 0 && child == 1) iscut[u] = 0;
50    return lowu;
51 }
52 void find_bcc(int n)
53 {
54     //调用结束后S保证为空，所以不用清空
55     clr(pre, 0), clr(iscut, 0), clr(bccno, 0);
56     dfs_clock = bcc_cnt = 0;
57     for (int i = 0; i < n; i++)
58         if (!pre[i]) dfs(i, -1);
59 }
```

4.7 Strongly Connected Component

```
1  const int maxn = "Edit";
2  vector<int> G[maxn];
3  int pre[maxn], lowlink[maxn], sccno[maxn], dfs_clock, scc_cnt;
4  stack<int> S;
5  inline void add_edge(int u, int v) { G[u].pb(v); }
6  void dfs(int u)
7  {
8      pre[u] = lowlink[u] = ++dfs_clock;
9      S.push(u);
10     for (int i = 0; i < G[u].size(); i++)
11     {
12         int v = G[u][i];
13         if (!pre[v])
14         {
15             dfs(v);
16             lowlink[u] = min(lowlink[u], lowlink[v]);
17         }
18         else if (!sccno[v])
19             lowlink[u] = min(lowlink[u], pre[v]);
20     }
21     if (lowlink[u] == pre[u])
22     {
23         scc_cnt++;
24         for (;;)
25         {
26             int x = S.top();
27             S.pop();
28             sccno[x] = scc_cnt;
29             if (x == u) break;
```

```
30              }
31          }
32 }
33 void find_scc(int n)
34 {
35      dfs_clock = 0, scc_cnt = 0;
36      clr(sccno, 0), clr(pre, 0);
37      for (int i = 0; i < n; i++)
38          if (!pre[i]) dfs(i);
39 }
```

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

2. 最小路径覆盖 $=|G|$-最大匹配数

   在一个 $N \times N$ 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:

   (a) 一个单独的顶点是一条路径;

   (b) 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)

```
1 /*
2 二分图匹配(匈牙利算法的DFS实现)(邻接矩阵形式)
3 初始化:g[][]两边顶点的划分情况
4 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
5 g没有边相连则初始化为0
6 uN是匹配左边的顶点数,vN是匹配右边的顶点数
7 调用:res=hungary();输出最大匹配数
8 优点:适用于稠密图,DFS找增广路,实现简洁易于理解
9 时间复杂度:O(VE)
10 顶点编号从0开始的
11 */
12 const int maxn = "Edit";
13 int uN, vN;          //u,v的数目,使用前面必须赋值
14 int g[maxn][maxn]; //邻接矩阵
15 int linker[maxn];
16 bool used[maxn];
17 bool dfs(int u)
18 {
19      for (int v = 0; v < vN; v++)
20          if (g[u][v] && !used[v])
21          {
22              used[v] = true;
23              if (linker[v] == -1 || dfs(linker[v]))
24              {
25                  linker[v] = u;
```

```
26                    return true;
27                }
28            }
29        return false;
30  }
31  int hungary()
32  {
33      int res = 0;
34      clr(linker, -1);
35      for (int u = 0; u < uN; u++)
36      {
37          clr(used, 0);
38          if (dfs(u)) res++;
39      }
40      return res;
41  }
```

4.8.2   Hungry(List)

```
1   /*
2   匈牙利算法邻接表形式
3   使用前用init()进行初始化
4   加边使用函数addedge(u,v)
5   */
6   const int maxn = "Edit";
7   int n;
8   vector<int> G[maxn];
9   int linker[maxn];
10  bool used[maxn];
11  inline void init(int n)
12  {
13      for (int i = 0; i < n; i++) G[i].clear();
14  }
15  inline void addedge(int u, int v) { G[u].pb(v); }
16  bool dfs(int u)
17  {
18      for (int i = 0; i < G[u].size(); i++)
19      {
20          int v = G[u][i];
21          if (!used[v])
22          {
23              used[v] = true;
24              if (linker[v] == -1 || dfs(linker[v]))
25              {
26                  linker[v] = u;
27                  return true;
28              }
29          }
30      }
31      return false;
32  }
33  int hungary()
34  {
35      int ans = 0;
36      clr(linker, -1);
37      for (int u = 0; u < n; v++)
38      {
39          clr(vis, 0);
```

```
40            if (dfs(u)) ans++;
41        }
42        return ans;
43    }
```

4.8.3 Hopcroft-Carp

```
1    /*
2    二分图匹配(Hopcroft-Carp算法)
3    复杂度O(sqrt(n)*E)
4    邻接表存图,vector实现
5    vector先初始化,然后加边
6    uN 为左端的顶点数,使用前赋值(点编号0开始)
7    */
8    const int maxn = "Edit";
9    vector<int> G[maxn];
10   int uN;
11   int Mx[maxn], My[maxn];
12   int dx[maxn], dy[maxn];
13   int dis;
14   bool used[maxn];
15   inline void init(int n)
16   {
17       for (int i = 0; i < n; i++) G[i].clear();
18   }
19   inline void addedge(int u, int v) { G[u].pb(v); }
20   bool SearchP()
21   {
22       queue<int> Q;
23       dis = INF;
24       clr(dx, -1);
25       clr(dy, -1);
26       for (int i = 0; i < uN; i++)
27           if (Mx[i] == -1)
28           {
29               Q.push(i);
30               dx[i] = 0;
31           }
32       while (!Q.empty())
33       {
34           int u = Q.front();
35           Q.pop();
36           if (dx[u] > dis) break;
37           int sz = G[u].size();
38           for (int i = 0; i < sz; i++)
39           {
40               int v = G[u][i];
41               if (dy[v] == -1)
42               {
43                   dy[v] = dx[u] + 1;
44                   if (My[v] == -1)
45                       dis = dy[v];
46                   else
47                   {
48                       dx[My[v]] = dy[v] + 1;
49                       Q.push(My[v]);
50                   }
51               }
```

```
52              }
53          }
54          return dis != INF;
55      }
56      bool DFS(int u)
57      {
58          int sz = G[u].size();
59          for (int i = 0; i < sz; i++)
60          {
61              int v = G[u][i];
62              if (!used[v] && dy[v] == dx[u] + 1)
63              {
64                  used[v] = true;
65                  if (My[v] != -1 && dy[v] == dis) continue;
66                  if (My[v] == -1 || DFS(My[v]))
67                  {
68                      My[v] = u, Mx[u] = v;
69                      return true;
70                  }
71              }
72          }
73          return false;
74      }
75      int MaxMatch()
76      {
77          int res = 0;
78          clr(Mx, -1), clr(My, -1);
79          while (SearchP())
80          {
81              clr(used, false);
82              for (int i = 0; i < uN; i++)
83                  if (Mx[i] == -1 && DFS(i)) res++;
84          }
85          return res;
86      }
```

4.8.4 Hungry(Multiple)

```
1   const int maxn = "Edit";
2   const int maxm = "Edit";
3   int uN, vN;           //u,v的数目,使用前面必须赋值
4   int g[maxn][maxm]; //邻接矩阵
5   int linker[maxm][maxn];
6   bool used[maxm];
7   int num[maxm]; //右边最大的匹配数
8   bool dfs(int u)
9   {
10      for (int v = 0; v < vN; v++)
11          if (g[u][v] && !used[v])
12          {
13              used[v] = true;
14              if (linker[v][0] < num[v])
15              {
16                  linker[v][++linker[v][0]] = u;
17                  return true;
18              }
19              for (int i = 1; i <= num[0]; i++)
20                  if (dfs(linker[v][i]))
```

```
21                        {
22                            linker[v][i] = u;
23                            return true;
24                        }
25                }
26        return false;
27    }
28    int hungary()
29    {
30        int res = 0;
31        for (int i = 0; i < vN; i++) linker[i][0] = 0;
32        for (int u = 0; u < uN; u++)
33        {
34            clr(used, 0);
35            if (dfs(u)) res++;
36        }
37        return res;
38    }
```

4.8.5   Kuhn-Munkres

```
1    const int maxn = "Edit";
2    int nx, ny;                          //两边的点数
3    int g[maxn][maxn];                   //二分图描述
4    int linker[maxn], lx[maxn], ly[maxn]; //y中各点匹配状态,x,y中的点标号
5    int slack[N];
6    bool visx[N], visy[N];
7    bool dfs(int x)
8    {
9        visx[x] = true;
10       for (int y = 0; y < ny; y++)
11       {
12           if (visy[y]) continue;
13           int tmp = lx[x] + ly[y] - g[x][y];
14           if (tmp == 0)
15           {
16               visy[y] = true;
17               if (linker[y] == -1 || dfs(linker[y]))
18               {
19                   linker[y] = x;
20                   return true;
21               }
22           }
23           else if (slack[y] > tmp)
24               slack[y] = tmp;
25       }
26       return false;
27    }
28    int KM()
29    {
30       clr(linker, -1), clr(ly, 0);
31       for (int i = 0; i < nx; i++)
32       {
33           lx[i] = -INF;
34           for (int j = 0; j < ny; j++)
35               if (g[i][j] > lx[i]) lx[i] = g[i][j];
36       }
37       for (int x = 0; x < nx; x++)
```

```
38      {
39          clr(slack, 0x3f);
40          for (;;)
41          {
42              clr(visx, 0), clr(visy, 0);
43              if (dfs(x)) break;
44              int d = INF;
45              for (int i = 0; i < ny; i++)
46                  if (!visy[i] && d > slack[i]) d = slack[i];
47              for (int i = 0; i < nx; i++)
48                  if (visx[i]) lx[i] -= d;
49              for (int i = 0; i < ny; i++)
50                  if (visy[i])
51                      ly[i] += d;
52                  else
53                      slack[i] -= d;
54          }
55      }
56      int res = 0;
57      for (int i = 0; i < ny; i++)
58          if (~linker[i]) res += g[linker[i]][i];
59      return res;
60  }
```

4.9   Network Flow

```
1  struct Edge
2  {
3      int from, to, cap, flow;
4      Edge(int u, int v, int c, int f) : from(u), to(v), cap(c), flow(f) {}
5  };
```

4.9.1   EdmondKarp

```
1  const int maxn = "Edit";
2  struct EdmonsKarp //时间复杂度O(v*E*E)
3  {
4      int n, m;
5      vector<Edge> edges;    //边数的两倍
6      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
7      int a[maxn];             //起点到i的可改进量
8      int p[maxn];             //最短路树上p的入弧编号
9      void init(int n)
10     {
11         for (int i = 0; i < n; i++) G[i].clear();
12         edges.clear();
13     }
14     void AddEdge(int from, int to, int cap)
15     {
16         edges.pb(Edge(from, to, cap, 0));
17         edges.pb(Edge(to, from, 0, 0)); //反向弧
18         m = edges.size();
19         G[from].pb(m - 2);
20         G[to].pb(m - 1);
21     }
22     int Maxflow(int s, int t)
23     {
```

```
24          int flow = 0;
25          for (;;)
26          {
27              clr(a, 0);
28              queue<int> q;
29              q.push(s);
30              a[s] = INF;
31              while (!q.empty())
32              {
33                  int x = q.front();
34                  q.pop();
35                  for (int i = 0; i < G[x].size(); i++)
36                  {
37                      Edge& e = edges[G[x][i]];
38                      if (!a[e.to] && e.cap > e.flow)
39                      {
40                          p[e.to] = G[x][i];
41                          a[e.to] = min(a[x], e.cap - e.flow);
42                          q.push(e.to);
43                      }
44                  }
45                  if (a[t]) break;
46              }
47              if (!a[t]) break;
48              for (int u = t; u != s; u = edges[p[u]].from)
49              {
50                  edges[p[u]].flow += a[t];
51                  edges[p[u] ^ 1].flow -= a[t];
52              }
53              flow += a[t];
54          }
55          return flow;
56      }
57  };
```

4.9.2 Dinic

```
1  const int maxn = "Edit";
2  struct Dinic
3  {
4      int n, m, s, t;        //结点数，边数（包括反向弧），源点编号和汇点编号
5      vector<Edge> edges;    //边表。edge[e]和edge[e^1]互为反向弧
6      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
7      bool vis[maxn];        //BFS使用
8      int d[maxn];           //从起点到i的距离
9      int cur[maxn];         //当前弧下标
10     void init(int n)
11     {
12         this->n = n;
13         for (int i = 0; i < n; i++) G[i].clear();
14         edges.clear();
15     }
16     void AddEdge(int from, int to, int cap)
17     {
18         edges.pb(Edge(from, to, cap, 0));
19         edges.pb(Edge(to, from, 0, 0));
20         m = edges.size();
21         G[from].pb(m - 2);
```

```
22          G[to].pb(m - 1);
23      }
24      bool BFS()
25      {
26          clr(vis, 0);
27          clr(d, 0);
28          queue<int> q;
29          q.push(s);
30          d[s] = 0;
31          vis[s] = 1;
32          while (!q.empty())
33          {
34              int x = q.front();
35              q.pop();
36              for (int i = 0; i < G[x].size(); i++)
37              {
38                  Edge& e = edges[G[x][i]];
39                  if (!vis[e.to] && e.cap > e.flow)
40                  {
41                      vis[e.to] = 1;
42                      d[e.to] = d[x] + 1;
43                      q.push(e.to);
44                  }
45              }
46          }
47          return vis[t];
48      }
49      int DFS(int x, int a)
50      {
51          if (x == t || a == 0) return a;
52          int flow = 0, f;
53          for (int& i = cur[x]; i < G[x].size(); i++)
54          {
55              //从上次考虑的弧
56              Edge& e = edges[G[x][i]];
57              if (d[x] + 1 == d[e.to] && (f = DFS(e.to, min(a, e.cap - e.flow))) > 0)
58              {
59                  e.flow += f;
60                  edges[G[x][i] ^ 1].flow -= f;
61                  flow += f;
62                  a -= f;
63                  if (a == 0) break;
64              }
65          }
66          return flow;
67      }
68      int Maxflow(int s, int t)
69      {
70          this->s = s;
71          this->t = t;
72          int flow = 0;
73          while (BFS())
74          {
75              clr(cur, 0);
76              flow += DFS(s, INF);
77          }
78          return flow;
79      }
80  };
```

4.9.3 ISAP

```
const int maxn = "Edit";
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++)
```

```
58                    {
59                        Edge& e = edges[G[x][i]];
60                        if (!vis[e.from] && e.cap > e.flow)
61                        {
62                            vis[e.from] = 1;
63                            d[e.from] = d[x] + 1;
64                            q.push(e.from);
65                        }
66                    }
67                }
68            }
69        int Maxflow(int s, int t)
70        {
71            this->s = s;
72            this->t = t;
73            int flow = 0;
74            BFS();
75            clr(num, 0);
76            for (int i = 0; i < n; i++)
77                if (d[i] < INF) num[d[i]]++;
78            int x = s;
79            clr(cur, 0);
80            while (d[s] < n)
81            {
82                if (x == t)
83                {
84                    flow += Augumemt();
85                    x = s;
86                }
87                int ok = 0;
88                for (int i = cur[x]; i < G[x].size(); i++)
89                {
90                    Edge& e = edges[G[x][i]];
91                    if (e.cap > e.flow && d[x] == d[e.to] + 1)
92                    {
93                        ok = 1;
94                        p[e.to] = G[x][i];
95                        cur[x] = i;
96                        x = e.to;
97                        break;
98                    }
99                }
100               if (!ok) //Retreat
101               {
102                   int m = n - 1;
103                   for (int i = 0; i < G[x].size(); i++)
104                   {
105                       Edge& e = edges[G[x][i]];
106                       if (e.cap > e.flow) m = min(m, d[e.to]);
107                   }
108                   if (--num[d[x]] == 0) break; //gap优化
109                   num[d[x] = m + 1]++;
110                   cur[x] = 0;
111                   if (x != s) x = edges[p[x]].from;
112               }
113           }
114           return flow;
115       }
116   };
```

4.9.4 MinCost MaxFlow

```cpp
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;
```

```
57                         }
58                     }
59                 }
60             }
61             if (d[t] == INF) return false; // 当没有可增广的路时退出
62             flow += a[t];
63             cost += (ll)d[t] * (ll)a[t];
64             for (int u = t; u != s; u = edges[p[u]].from)
65             {
66                 edges[p[u]].flow += a[t];
67                 edges[p[u] ^ 1].flow -= a[t];
68             }
69             return true;
70         }
71         int MincostMaxflow(int s, int t, ll& cost)
72         {
73             int flow = 0;
74             cost = 0;
75             while (BellmanFord(s, t, flow, cost));
76             return flow;
77         }
78 };
```

4.10   2-SAT

```
1  struct TwoSAT
2  {
3      int n;
4      vector<int> G[maxn << 1];
5      bool mark[maxn << 1];
6      int S[maxn << 1], c;
7      void init(int n)
8      {
9          this->n = n;
10         for (int i = 0; i < (n << 1); i++) G[i].clear();
11         clr(mark, 0);
12     }
13     bool dfs(int x)
14     {
15         if (mark[x ^ 1]) return false;
16         if (mark[x]) return true;
17         mark[x] = true;
18         S[c++] = x;
19         for (int i = 0; i < G[x].size(); i++)
20             if (!dfs(G[x][i])) return false;
21         return true;
22     }
23     //x = xval or y = yval
24     void add_clause(int x, int xval, int y, int yval)
25     {
26         x = (x << 1) + xval;
27         y = (y << 1) + yval;
28         G[x ^ 1].pb(y);
29         G[y ^ 1].pb(x);
30     }
31     bool solve()
32     {
33         for (int i = 0; i < (n << 1); i += 2)
```

```
34              if (!mark[i] && !mark[i + 1])
35              {
36                  c = 0;
37                  if (!dfs(i))
38                  {
39                      while (c > 0) mark[S[--c]] = false;
40                      if (!dfs(i + 1)) return false;
41                  }
42              }
43          return true;
44      }
45  };
```

4.11 Eular Path

- 基本概念:
    - 欧拉图: 能够没有重复地一次遍历所有边的图。(必须是连通图)
    - 欧拉路: 上述遍历的路径就是欧拉路。
    - 欧拉回路: 若欧拉路是闭合的（一个圈，从起点开始遍历最终又回到起点），则为欧拉回路。
- 无向图 G 有欧拉路径的充要条件
    - G 是连通图
    - G 中奇顶点（连接边的数量为奇数）的数量等于 0 或 2.
- 无向图 G 有欧拉回路的充要条件
    - G 是连通图
    - G 中每个顶点都是偶顶点
- 有向图 G 有欧拉路径的充要条件
    - G 是连通图
    - u 的出度比入度大 1，v 的出度比入度小 1，其他所有点出度和入度相同。(u 为起点，v 为终点)
- 有向图 G 有欧拉回路的充要条件
    - G 是连通图
    - G 中每个顶点的出度等于入度

4.11.1 Fleury

```
1  // 若有两个点的度数是奇数，则此时这两个点只能作为欧拉路径的起点和终点。
2  const int maxn = "Edit";
3  int G[maxn][maxn];
4  int deg[maxn][maxn];
5  vector<int> Ans;
6  inline void init() { clr(G, 0), clr(deg, 0); }
7  inline void AddEdge(int u, int v) { deg[u]++, deg[v]++, G[u][v]++, G[v][u]++; }
8  void Fleury(int s)
9  {
10     for (int i = 0; i < n; i++)
11         if (G[s][i])
12         {
13             G[s][i]--, G[i][s]--;
14             Fleury(i);
15         }
16     Ans.pb(s);
17 }
```

5 Computational Geometry

5.1 Basic Function

```
1  #define zero(x) ((fabs(x) < eps ? 1 : 0))
2  #define sgn(x) (fabs(x) < eps ? 0 : ((x) < 0 ? -1 : 1))
3
4  struct point
5  {
6      double x, y;
7      point(double a = 0, double b = 0) { x = a, y = b; }
8      point operator-(const point& b) const { return point(x - b.x, y - b.y); }
9      point operator+(const point& b) const { return point(x + b.x, y + b.y); }
10     // 两点是否重合
11     bool operator==(point& b) { return zero(x - b.x) && zero(y - b.y); }
12     // 点积(以原点为基准)
13     double operator*(const point& b) const { return x * b.x + y * b.y; }
14     // 叉积(以原点为基准)
15     double operator^(const point& b) const { return x * b.y - y * b.x; }
16     // 绕P点逆时针旋转a弧度后的点
17     point rotate(point b, double a)
18     {
19         double dx, dy;
20         (*this - b).split(dx, dy);
21         double tx = dx * cos(a) - dy * sin(a);
22         double ty = dx * sin(a) + dy * cos(a);
23         return point(tx, ty) + b;
24     }
25     // 点坐标分别赋值到a和b
26     void split(double& a, double& b) { a = x, b = y; }
27  };
28  struct line
29  {
30      point s, e;
31      line() {}
32      line(point ss, point ee) { s = ss, e = ee; }
33  };
```

5.2 Position

5.2.1 Point-Point

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

5.2.2 Line-Line

```
1  // <0, *> 表示重合; <1, *> 表示平行; <2, P> 表示交点是P;
2  pair<int, point> spoint(line l1, line l2)
3  {
4      point res = l1.s;
5      if (sgn((l1.s - l1.e) ^ (l2.s - l2.e)) == 0)
6          return mp(sgn((l1.s - l2.e) ^ (l2.s - l2.e)) != 0, res);
7      double t = ((l1.s - l2.s) ^ (l2.s - l2.e)) / ((l1.s - l1.e) ^ (l2.s - l2.e));
8      res.x += (l1.e.x - l1.s.x) * t;
9      res.y += (l1.e.y - l1.s.y) * t;
10     return mp(2, res);
11 }
```

5.2.3 Segment-Segment

```
1  bool segxseg(line l1, line l2)
2  {
3      return
4          max(l1.s.x, l1.e.x) >= min(l2.s.x, l2.e.x) &&
5          max(l2.s.x, l2.e.x) >= min(l1.s.x, l1.e.x) &&
6          max(l1.s.y, l1.e.y) >= min(l2.s.y, l2.e.y) &&
7          max(l2.s.y, l2.e.y) >= min(l1.s.y, l1.e.y) &&
8          sgn((l2.s - l1.e) ^ (l1.s - l1.e)) * sgn((l2.e-l1.e) ^ (l1.s - l1.e)) <= 0 &&
9          sgn((l1.s - l2.e) ^ (l2.s - l2.e)) * sgn((l1.e-l2.e) ^ (l2.s - l2.e)) <= 0;
10 }
```

5.2.4 Line-Segment

```
1  //l1是直线,l2是线段
2  bool segxline(line l1, line l2)
3  {
4      return sgn((l2.s - l1.e) ^ (l1.s - l1.e)) * sgn((l2.e - l1.e) ^ (l1.s - l1.e)) <=
         0;
5  }
```

5.2.5 Point-Line

```
1  point pointtoline(point P, line L)
2  {
3      point res;
4      double t = ((P - L.s) * (L.e - L.s)) / ((L.e - L.s) * (L.e - L.s));
5      res.x = L.s.x + (L.e.x - L.s.x) * t, res.y = L.s.y + (L.e.y - L.s.y) * t;
6      return dist(P, res);
7  }
```

5.2.6 Point-Segment

```
1  point pointtosegment(point p, line l)
2  {
3      point res;
4      double t = ((p - l.s) * (l.e - l.s)) / ((l.e - l.s) * (l.e - l.s));
5      if (t >= 0 && t <= 1)
6          res.x = l.s.x + (l.e.x - l.s.x) * t, res.y = l.s.y + (l.e.y - l.s.y) * t;
7      else
8          res = dist(p, l.s) < dist(p, l.e) ? l.s : l.e;
9      return res;
10 }
```

5.2.7 Point on Segment

```
1  bool PointOnSeg(point p, line l)
2  {
3      return
4          sgn((l.s - p) ^ (l.e-p)) == 0 &&
5          sgn((p.x - l.s.x) * (p.x - l.e.x)) <= 0 &&
6          sgn((p.y - l.s.y) * (p.y - l.e.y)) <= 0;
7  }
```

5.3 Polygon

5.3.1 Area

```
1  double area(point p[], int n)
2  {
3      double res = 0;
4      for (int i = 0; i < n; i++) res += (p[i] ^ p[(i + 1) % n]) / 2;
5      return fabs(res);
6  }
```

5.3.2 Point in Convex

```
1  // 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)
2  // 点的编号 : [0,n)
3  // -1 : 点在凸多边形外
4  // 0  : 点在凸多边形边界上
5  // 1  : 点在凸多边形内
6  int PointInConvex(point a, point p[], int n)
7  {
8      for (int i = 0; i < n; i++)
9          if (sgn((p[i] - a) ^ (p[(i + 1) % n] - a)) < 0)
10             return -1;
11         else if (PointOnSeg(a, line(p[i], p[(i + 1) % n])))
12             return 0;
13     return 1;
14 }
```

5.3.3 Point in Polygon

```
1  // 射线法,poly[]的顶点数要大于等于3,点的编号0~n-1
2  // -1 : 点在凸多边形外
3  // 0  : 点在凸多边形边界上
4  // 1  : 点在凸多边形内
5  int PointInPoly(point p, point poly[], int n)
6  {
7      int cnt;
8      line ray, side;
9      cnt = 0;
10     ray.s = p;
11     ray.e.y = p.y;
12     ray.e.x = -100000000000.0; // -INF,注意取值防止越界
13     for (int i = 0; i < n; i++)
14     {
15         side.s = poly[i], side.e = poly[(i + 1) % n];
16         if (PointOnSeg(p, side)) return 0;
17         //如果平行轴则不考虑
18         if (sgn(side.s.y - side.e.y) == 0)
19             continue;
20         if (PointOnSeg(sid e.s, r ay))
21             cnt += (sgn(side.s.y - side.e.y) > 0);
22         else if (PointOnSeg(side.e, ray))
23             cnt += (sgn(side.e.y - side.s.y) > 0);
24         else if (segxseg(ray, side))
25             cnt++;
26     }
27     return cnt % 2 == 1 ? 1 : -1;
28 }
```

5.3.4 Judge Convex

```
1  //点可以是顺时针给出也可以是逆时针给出
2  //点的编号1~n-1
3  bool isconvex(point poly[], int n)
4  {
5      bool s[3];
6      clr(s, 0);
7      for (int i = 0; i < n; i++)
8      {
9          s[sgn((poly[(i + 1) % n] - poly[i]) ^ (poly[(i + 2) % n] - poly[i])) + 1] = 1;
10         if (s[0] && s[2]) return 0;
11     }
12     return 1;
13 }
```

5.4 Integer Points

5.4.1 On Segment

```
1  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

```
1  int OnEdge(point p[], int n)
2  {
3      int i, ret = 0;
4      for (i = 0; i < n; i++)
5          ret += __gcd(fabs(p[i].x - p[(i + 1) % n].x), fabs(p[i].y - p[(i + 1) % n].y));
6      return ret;
7  }
```

5.4.3 Inside Polygon

```
1  int InSide(point p[], int n)
2  {
3      int i, area = 0;
4      for (i = 0; i < n; i++)
5          area += p[(i + 1) % n].y * (p[i].x - p[(i + 2) % n].x);
6      return (fabs(area) - OnEdge(n, p)) / 2 + 1;
7  }
```

5.5 Circle

5.5.1 Circumcenter

```
1  point waixin(point a, point b, point c)
2  {
3      double a1 = b.x - a.x, b1 = b.y - a.y, c1 = (a1 * a1 + b1 * b1) / 2;
4      double a2 = c.x - a.x, b2 = c.y - a.y, c2 = (a2 * a2 + b2 * b2) / 2;
5      double d = a1 * b2 - a2 * b1;
6      return point(a.x + (c1 * b2 - c2 * b1) / d, a.y + (a1 * c2 - a2 * c1) / d);
7  }
```

6 Dynamic Programming

6.1 Subsequence

6.1.1 Max Sum

```
1  // 传入序列a和长度n, 返回最大子序列和
2  int MaxSeqSum(int a[], int n)
3  {
4      int rt = 0, cur = 0;
5      for (int i = 0; i < n; i++)
6          cur += a[i], rt = max(cur, rt), cur = max(0, cur);
7      return rt;
8  }
```

6.1.2 Longest Increase

```
1   // 序列下标从1开始, LIS()返回长度, 序列存在lis[]中
2   const int N = "Edit";
3   int len, a[N], b[N], f[N];
4   int Find(int p, int l, int r)
5   {
6       while (l <= r)
7       {
8           int mid = (l + r) >> 1;
9           if (a[p] > b[mid])
10              l = mid + 1;
11          else
12              r = mid - 1;
13      }
14      return f[p] = l;
15  }
16  int LIS(int lis[], int n)
17  {
18      int len = 1;
19      f[1] = 1, b[1] = a[1];
20      for (int i = 2; i <= n; i++)
21      {
22          if (a[i] > b[len])
23              b[++len] = a[i], f[i] = len;
24          else
25              b[Find(i, 1, len)] = a[i];
26      }
27      for (int i = n, t = len; i >= 1 && t >= 1; i--)
28          if (f[i] == t) lis[--t] = a[i];
29      return len;
30  }
31
32  // 简单写法(下标从0开始,只返回长度)
33  int dp[N];
34  int LIS(int a[], int n)
35  {
36      clr(dp, 0x3f);
37      for (int i = 0; i < n; i++) *lower_bound(dp, dp + n, a[i]) = a[i];
38      return lower_bound(dp, dp + n, INF) - dp;
39  }
```

6.1.3 Longest Common Increase

```
1   // 序列下标从1开始
2   int LCIS(int a[], int b[], int n, int m)
3   {
4       clr(dp, 0);
5       for (int i = 1; i <= n; i++)
6       {
7           int ma = 0;
8           for (int j = 1; j <= m; j++)
9           {
10              dp[i][j] = dp[i - 1][j];
11              if (a[i] > b[j]) ma = max(ma, dp[i - 1][j]);
12              if (a[i] == b[j]) dp[i][j] = ma + 1;
13          }
14      }
15      return *max_element(dp[n] + 1, dp[n] + 1 + m);
16  }
```

6.2 Digit Statistics

```
1   int a[20];
2   ll dp[20][state];
3   ll dfs(int pos, /*state变量*/, bool lead /*前导零*/, bool limit /*数位上界变量*/)
4   {
5       //递归边界，既然是按位枚举，最低位是0，那么pos==-1说明这个数枚举完了
6       if (pos == -1) return 1;
7       /*这里一般返回1，表示枚举的这个数是合法的，那么这里就需要在枚举时必须每一位都要满足题目条件，
8       也就是说当前枚举到pos位，一定要保证前面已经枚举的数位是合法的。*/
9       if (!limit && !lead && dp[pos][state] != -1) return dp[pos][state];
10      /*常规写法都是在没有限制的条件记忆化，这里与下面记录状态是对应*/
11      int up = limit ? a[pos] : 9; //根据limit判断枚举的上界up
12      ll ans = 0;
13      for (int i = 0; i <= up; i++) //枚举，然后把不同情况的个数加到ans就可以了
14      {
15          if () ...
16          else if () ...
17          ans += dfs(pos - 1, /*状态转移*/, lead && i == 0, limit && i == a[pos])
18          //最后两个变量传参都是这样写的
19          /*当前数位枚举的数是i，然后根据题目的约束条件分类讨论
20          去计算不同情况下的个数，还有要根据state变量来保证i的合法性*/
21      }
22      //计算完，记录状态
23      if (!limit && !lead) dp[pos][state] = ans;
24      /*这里对应上面的记忆化，在一定条件下时记录，保证一致性，
25      当然如果约束条件不需要考虑lead，这里就是lead就完全不用考虑了*/
26      return ans;
27  }
28  ll solve(ll x)
29  {
30      int pos = 0;
31      while (x) //把数位都分解出来
32          a[pos++] = x % 10, x /= 10;
33      return dfs(pos - 1 /*从最高位开始枚举*/, /*一系列状态 */, true, true);
34      //刚开始最高位都是有限制并且有前导零的，显然比最高位还要高的一位视为0
35  }
```

7 Others

7.1 Matrix

7.1.1 Matrix FastPow

```
 1  typedef vector<ll> vec;
 2  typedef vector<vec> mat;
 3  mat mul(mat& A, mat& B)
 4  {
 5      mat C(A.size(), vec(B[0].size()));
 6      for (int i = 0; i < A.size(); i++)
 7          for (int k = 0; k < B.size(); k++)
 8              if (A[i][k]) // 对稀疏矩阵的优化
 9                  for (int j = 0; j < B[0].size(); j++)
10                      C[i][j] = (C[i][j] + A[i][k] * B[k][j]) % mod;
11      return C;
12  }
13  mat Pow(mat A, ll n)
14  {
15      mat B(A.size(), vec(A.size()));
16      for (int i = 0; i < A.size(); i++) B[i][i] = 1;
17      for (; n; n >>= 1, A = mul(A, A))
18          if (n & 1) B = mul(B, A);
19      return B;
20  }
```

7.1.2 Gauss Elimination

```
 1  void gauss()
 2  {
 3      int now = 1, to;
 4      double t;
 5      for (int i = 1; i <= n; i++, now++)
 6      {
 7          /*for (to = now; !a[to][i] && to <= n; to++);
 8          //做除法时减小误差，可不写
 9          if (to != now)
10              for (int j = 1; j <= n + 1; j++)
11                  swap(a[to][j], a[now][j]);*/
12          t = a[now][i];
13          for (int j = 1; j <= n + 1; j++) a[now][j] /= t;
14          for (int j = 1; j <= n; j++)
15              if (j != now)
16              {
17                  t = a[j][i];
18                  for (int k = 1; k <= n + 1; k++) a[j][k] -= t * a[now][k];
19              }
20      }
21  }
```

7.2 Tricks

7.2.1 Stack-Overflow

```
 1  // 解决爆栈问题
 2  #pragma comment(linker, "/STACK:1024000000,1024000000")
```

7.2.2 Fast-Scanner

```
1   // 适用于正负整数
2   template <class T>
3   inline bool scan_d(T &ret)
4   {
5       char c;
6       int sgn;
7       if (c = getchar(), c == EOF) return 0; //EOF
8       while (c != '-' && (c < '0' || c > '9')) c = getchar();
9       sgn = (c == '-') ? -1 : 1;
10      ret = (c == '-') ? 0 : (c - '0');
11      while (c = getchar(), c >= '0' && c <= '9') ret = ret * 10 + (c - '0');
12      ret *= sgn;
13      return 1;
14  }
15  inline void out(int x)
16  {
17      if (x > 9) out(x / 10);
18      putchar(x % 10 + '0');
19  }
```

7.2.3 Strok-Sscanf

```
1   // 空格作为分隔输入,读取一行的整数
2   gets(buf);
3   int v;
4   char *p = strtok(buf, " ");
5   while (p)
6   {
7       sscanf(p, "%d", &v);
8       p = strtok(NULL," ");
9   }
```

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。

```
1   struct query { int L, R, id; };
2   void solve(query node[], int m)
3   {
4       tmp = 0;
5       clr(num, 0);
6       clr(ans, 0);
7       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; });
8       int L = 1, R = 0;
9       for (int i = 0; i < m; i++)
10      {
11          while (node[i].L < L) add(a[--L]);
12          while (node[i].L > L) del(a[L++]);
13          while (node[i].R < R) del(a[R--]);
14          while (node[i].R > R) add(a[++R]);
15          ans[node[i].id] = tmp;
16      }
17  }
```

7.4 BigNum

7.4.1 High-precision

```
1  // 加法 乘法 小于号 输出
2  struct bint
3  {
4      int l;
5      short int w[100];
6      bint(int x = 0)
7      {
8          l = x == 0, clr(w, 0);
9          while (x) w[l++] = x % 10, x /= 10;
10     }
11     bool operator<(const bint& x) const
12     {
13         if (l != x.l) return l < x.l;
14         int i = l - 1;
15         while (i >= 0 && w[i] == x.w[i]) i--;
16         return (i >= 0 && w[i] < x.w[i]);
17     }
18     bint operator+(const bint& x) const
19     {
20         bint ans;
21         ans.l = l > x.l ? l : x.l;
22         for (int i = 0; i < ans.l; i++)
23         {
24             ans.w[i] += w[i] + x.w[i];
25             ans.w[i + 1] += ans.w[i] / 10;
26             ans.w[i] = ans.w[i] % 10;
27         }
28         if (ans.w[ans.l] != 0) ans.l++;
29         return ans;
30     }
31     bint operator*(const bint& x) const
32     {
33         bint res;
34         int up, tmp;
35         for (int i = 0; i < l; i++)
36         {
37             up = 0;
38             for (int j = 0; j < x.l; j++)
39             {
40                 tmp = w[i] * x.w[j] + res.w[i + j] + up;
41                 res.w[i + j] = tmp % 10;
42                 up = tmp / 10;
43             }
44             if (up != 0) res.w[i + x.l] = up;
45         }
46         res.l = l + x.l;
47         while (res.w[res.l - 1] == 0 && res.l > 1) res.l--;
48         return res;
49     }
50     void print()
51     {
52         for (int i = l - 1; ~i; i--) printf("%d", w[i]);
53         puts("");
54     }
55 };
```

7.4.2 Complete High-precision

```
1   #define N 10000
2   class bint
3   {
4   private:
5       int a[N]; // 用 N 控制最大位数
6       int len;  // 数字长度
7   public:
8       // 构造函数
9       bint() { len = 1, clr(a, 0); }
10      // int -> bint
11      bint(int n)
12      {
13          len = 0;
14          clr(a, 0);
15          int d = n;
16          while (n)
17              d = n / 10 * 10, a[len++] = n - d, n = d / 10;
18      }
19      // char[] -> int
20      bint(const char s[])
21      {
22          clr(a, 0);
23          len = 0;
24          int l = strlen(s);
25          for (int i = l - 1; ~i; i--) a[len++] = s[i];
26      }
27      // 拷贝构造函数
28      bint(const bint& b)
29      {
30          clr(a, 0);
31          len = b.len;
32          for (int i = 0; i < len; i++) a[i] = b.a[i];
33      }
34      // 重载运算符 bint = bint
35      bint& operator=(const bint& n)
36      {
37          len = n.len;
38          for (int i = 0; i < len; i++) a[i] = n.a[i];
39          return *this;
40      }
41      // 重载运算符 bint + bint
42      bint operator+(const bint& b) const
43      {
44          bint t(*this);
45          int res = b.len > len ? b.len : len;
46          for (int i = 0; i < res; i++)
47          {
48              t.a[i] += b.a[i];
49              if (t.a[i] >= 10) t.a[i + 1]++, t.a[i] -= 10;
50          }
51          t.len = res + a[res] == 0;
52          return t;
53      }
54      // 重载运算符 bint - bint
55      bint operator-(const bint& b) const
56      {
57          bool f = *this > b;
```

```
58          bint t1 = f ? *this : b;
59          bint t2 = f ? b : *this;
60          int res = t1.len, j;
61          for (int i = 0; i < res; i++)
62              if (t1.a[i] < t2.a[i])
63              {
64                  j = i + 1;
65                  while (t1.a[j] == 0) j++;
66                  t1.a[j--]--;
67                  while (j > i) t1.a[j--] += 9;
68                  t1.a[i] += 10 - t1.a[i];
69              }
70              else
71                  t1.a[i] -= t2.a[i];
72          t1.len = res;
73          while (t1.a[len - 1] == 0 && t1.len > 1) t1.len--, res--;
74          if (f) t1.a[res - 1] = 0 - t1.a[res - 1];
75          return t1;
76      }
77      // 重载运算符 bint * bint
78      bint operator*(const bint& b) const
79      {
80          bint t;
81          int i, j, up, tmp, tmp1;
82          for (i = 0; i < len; i++)
83          {
84              up = 0;
85              for (j = 0; j < b.len; j++)
86              {
87                  tmp = a[i] * b.a[j] + t.a[i + j] + up;
88                  if (tmp > 9)
89                      tmp1 = tmp - tmp / 10 * 10, up = tmp / 10, t.a[i + j] = tmp1;
90                  else
91                      up = 0, t.a[i + j] = tmp;
92              }
93              if (up) t.a[i + j] = up;
94          }
95          t.len = i + j;
96          while (t.a[t.len - 1] == 0 && t.len > 1) t.len--;
97          return t;
98      }
99      // 重载运算符 bint / int
100     bint operator/(const int& b) const
101     {
102         bint t;
103         int down = 0;
104         for (int i = len - 1; ~i; i--)
105             t.a[i] = (a[i] + down * 10) / b, down = a[i] + down * 10 - t.a[i] * b;
106         t.len = len;
107         while (t.a[t.len - 1] == 0 && t.len > 1) t.len--;
108         return t;
109     }
110     // 重载运算符 bint ^ n (n次方快速幂，需保证n非负)
111     bint operator^(const int n) const
112     {
113         bint t(*this), rt(1);
114         if (n == 0) return 1;
115         if (n == 1) return *this;
116         int m = n;
```

```
117            for (; m; m >>= 1, t = t * t)
118                if (m & 1) rt = rt * t;
119        }
120        return rt;
121        // 重载运算符 bint > bint 比较大小
122        bool operator>(const bint& b) const
123        {
124            int p;
125            if (len > b.len) return 1;
126            if (len == b.len)
127            {
128                p = len - 1;
129                while (a[p] == b.a[p] && p >= 0) p--;
130                return p >= 0 && a[p] > b.a[p];
131            }
132            return 0;
133        }
134        // 重载运算符 bint > int 比较大小
135        bool operator>(const int& n) const { return *this > bint(n); }
136        // 输出
137        void out()
138        {
139            for (int i = len - 1; ~i; i--) printf("%d", a[i]);
140            puts("");
141        }
142    };
```

7.5 VIM

```
1   syntax on
2   set cindent
3   set nu
4   set tabstop = 4
5   set shiftwidth = 4
6   set background = dark
7   map<C-A> ggVG"+y
8   map<F5>: call Run()<CR>
9   func !Run()
10      exec "w"
11      exec "!g++ -Wall % -o %<"
12      exec "!./%<"
13  endfunc
```