ACM/ICPC Template Manaual

Shanghai University

X.D. Network Ingenuity

September 25, 2018

Contents

0   Header

```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  #define fastin                          \
5      ios_base::sync_with_stdio(0); \
6      cin.tie(0);
7  typedef long long ll;
8  typedef long double ld;
9  typedef pair<int, int> PII;
10 typedef vector<int> VI;
11 const int INF = 0x3f3f3f3f;
12 const int mod = 1e9 + 7;
13 const double eps = 1e-6;
14
15 #ifndef ONLINE_JUDGE
16 #define dbg(args...)                                 \
17     do                                               \
18     {                                                \
19         cout << "\033[32;1m" << #args << " -> "; \
20         err(args);                                   \
21     } while (0)
22 #else
23 #define dbg(...)
24 #endif
25 void err()
26 {
27     cout << "\033[39;0m" << endl;
28 }
29 template <template <typename...> class T, typename t, typename... Args>
30 void err(T<t> a, Args... args)
31 {
32     for (auto x : a) cout << x << ' ';
33     err(args...);
34 }
35 template <typename T, typename... Args>
36 void err(T a, Args... args)
37 {
38     cout << a << ' ';
39     err(args...);
40 }
41
42 int main()
43 {
44 #ifndef ONLINE_JUDGE
45     freopen("test.in", "r", stdin);
46     freopen("test.out", "w", stdout);
47 #endif
48
49     return 0;
50 }
```

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

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

$d(n)$ 函数

```cpp
const int maxn = "Edit";
int prime[maxn], tot;
int d[maxn], e[maxn]; //d正除数个数, e最小质因子个数
bool check[maxn];
void CalD()
{
    d[1] = 1;
```

```
8        for (int i = 2; i < maxn; i++)
9        {
10           if (!check[i])
11           {
12               prime[tot++] = i;
13               e[i] = 1, d[i] = 2;
14           }
15           for (int j = 0; j < tot; j++)
16           {
17               if (i * prime[j] >= maxn) break;
18               check[i * prime[j]] = true;
19               if (i % prime[j] == 0)
20               {
21                   e[i * prime[j]] = e[i] + 1;
22                   d[i * prime[j]] = d[i] / e[i] * (e[i] + 1);
23                   break;
24               }
25               else
26               {
27                   e[i * prime[j]] = 1;
28                   d[i * prime[j]] = 2 * d[i];
29               }
30           }
31       }
32   }
```

$\sigma\lambda(n)$ 函数, $\lambda = 1$

```
1    const int maxn = "Edit";
2    int prime[maxn], tot;
3    int sig[maxn], e[maxn]; //sig正除数, e不含能整除i的最小质因子的正除数和
4    bool check[maxn];
5    void CalSig()
6    {
7        sig[1] = 1;
8        for (int i = 2; i < maxn; i++)
9        {
10           if (!check[i])
11           {
12               prime[tot++] = i;
13               e[i] = 1, sig[i] = i + 1;
14           }
15           for (int j = 0; j < tot; j++)
16           {
17               if (i * prime[j] >= maxn) break;
18               check[i * prime[j]] = true;
19               if (i % prime[j] == 0)
20               {
21                   sig[i * prime[j]] = sig[i] * prime[j] + e[i];
22                   e[i * prime[j]] = e[i];
23                   break;
24               }
25               else
26               {
27                   sig[i * prime[j]] = sig[i] * (prime[j] + 1);
28                   e[i * prime[j]] = sig[i];
29               }
30           }
31       }
32   }
```

1.1.3 Prime Factorization

```
1  vector<pair<ll, int>> getFactors(ll x)
2  {
3      vector<pair<ll, int>> fact;
4      for (int i = 0; prime[i] <= x / prime[i]; i++)
5      {
6          if (x % prime[i] == 0)
7          {
8              fact.emplace_back(prime[i], 0);
9              while (x % prime[i] == 0) fact.back().second++, x /= prime[i];
10         }
11     }
12     if (x != 1) fact.emplace_back(x, 1);
13     return fact;
14 }
```

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
$1 < a < b \le 10^{12}, b - a \le 10^6$

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

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

1.2 Euler phi

1.2.1 Euler

```
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);
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  }
```

$a < p$ 且 p 为素数时，有以下两种求法
费马小定理

```
1  ll inv(ll a, ll p) { return Pow(a, p - 2, p); }
```

贾志鹏线性筛

```
1  for (int i = 2; i < n; i++) inv[i] = inv[p % i] * (p - p / i) % p;
```

1.3.4 Discrete Logarithm

求解 $a^x \equiv b \pmod{p}$, p 可以不是质数

```
1  ll exbsgs(ll a, ll b, ll p)
2  {
3      if (b == 1LL) return 0;
4      ll t, d = 1, k = 0;
5      while ((t = gcd(a, p)) != 1)
6      {
7          if (b % t) return -1;
8          ++k, b /= t, p /= t, d = d * (a / t) % p;
9          if (b == d) return k;
10     }
11     map<ll, ll> dic;
12     ll m = ceil(sqrt(p));
13     ll a_m = Pow(a, m, p), mul = b;
14     for (ll j = 1; j <= m; ++j) mul = mul * a % p, dic[mul] = j;
15     for (ll i = 1; i <= m; ++i)
16     {
```

```
17        d = d * a_m % p;
18        if (dic[d]) return i * m - dic[d] + k;
19      }
20      return -1;
21  }
```

1.4 Modulo Linear Equation

1.4.1 Chinese Remainder Theory

$X \equiv r_i \pmod{m_i}$; 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

$X \equiv r_i \pmod{m_i}$; 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  ll inv[maxn]; // 阶乘的逆元
 4  void CalFact()
 5  {
 6      f[0] = 1;
 7      for (int i = 1; i < maxn; i++) f[i] = (f[i - 1] * i) % p;
 8      inv[maxn - 1] = Pow(f[maxn - 1], p - 2, p);
 9      for (int i = maxn - 2; ~i; i--) inv[i] = inv[i + 1] * (i + 1) % p;
10  }
11  ll C(int n, int m) { return f[n] * inv[m] % p * inv[n - m] % p; }
```

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[maxn];
 3  ll inv[maxn]; // 阶乘的逆元
 4  void CalFact()
 5  {
 6      f[0] = 1;
 7      for (int i = 1; i < maxn; i++) f[i] = (f[i - 1] * i) % p;
 8      inv[maxn - 1] = Pow(f[maxn - 1], p - 2, p);
 9      for (int i = maxn - 2; ~i; i--) inv[i] = inv[i + 1] * (i + 1) % p;
10  }
11  ll Lucas(ll n, ll m, ll p)
12  {
13      ll ret = 1;
14      while (n && m)
15      {
16          ll a = n % p, b = m % p;
17          if (a < b) return 0;
18          ret = ret * f[a] % p * inv[b] % p * inv[a - b] % p;
19          n /= p, m /= p;
20      }
21      return ret;
22  }
```

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       memset(dp, 0, sizeof(dp));
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.push_back(i);
29           }
30       if (tmp != 1) v.push_back(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，$\frac{m^4+11m^2}{12}$

长度为 n 的项链串用 c 种颜色染 $\sum_{d|n} \frac{\varphi(n/d)c^d}{n}$

```
1    ll solve(int c, int n)
2    {
3        if (n == 0) return 0;
4        ll ans = 0;
5        for (int i = 1; i <= n; i++) ans += Pow(c, __gcd(i, n));
6        if (n & 1) ans += n * Pow(c, n + 1 >> 1);
7        else ans += n / 2 * (1 + c) * Pow(c, n >> 1);
8        return ans / n / 2;
9    }
```

每种颜色至少涂多少个，求方案数

```
1    ll polya(int a)//a为循环节长度
2    {
3        ll dp[65][65] = {0}; //前者为颜色，后者为未填充格子个数
4        int tot = 60 / a, limit = 0;
5        dp[0][tot] = 1;
6        for (int i = 1; i <= n; i++)
```

```
7          {
8              int tmp = (c[i] + a - 1) / a;
9              int up2 = tot - limit;
10             int up1 = up2 - tmp;              //最多空tot-(limit + tmp)
11             for (int j = 0; j <= up1; j++) //最少空0个, 即填满
12             {
13                 for (int k = tmp; j + k <= up2; k++) //至少选tmp个, 最多选tot - limit -j
14                     (dp[i][j] += dp[i - 1][j + k] * C[j + k][k]) %= p;
15             }
16             limit += tmp;
17         }
18         return dp[n][0];
19 }
```

每种颜色要有多少个, 求恰好满足的方案数

```
1 bool check(int b) //a[i]是每种颜色有多少个, b是循环节长度
2 {
3     for (int i = 0; i < n; i++)
4         if (a[i] % b) return false;
5     return true;
6 }
7 ll solve(int tot, int b) //tot是总数, b是循环节长度
8 {
9     if (!check(b)) return 0;
10    ll res = 1, cnt = tot / b; //cnt循环节个数
11    for (int i = 0; i < 6; i++)
12    {
13        res *= C[cnt][a[i] / b];
14        cnt -= a[i] / b;
15    }
16    return res;
17 }
```

1.6 Fast Power

```
1 inline ll Mul(ll a, ll b, ll m)
2 {
3     if (m <= 1000000000)
4         return a * b % m;
5     else if (m <= 1000000000000ll)
6         return (((a * (b >> 20) % m) << 20) + (a * (b & ((1 << 20) - 1)))) % m;
7     else
8     {
9         ll d = (ll)floor(a * (long double)b / m + 0.5);
10        ll ret = (a * b - d * m) % m;
11        if (ret < 0) ret += m;
12        return ret;
13    }
14 }
15 ll Pow(ll a, ll n, ll m)
16 {
17     ll t = 1;
18     for (; n; n >>= 1, a = (a * a % m))
19         if (n & 1) t = (t * a % m);
20     return t;
21 }
```

1.7 Mobius Inversion

1.7.1 Mobius

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

```
1  const int maxn = "Edit";
2  int prime[maxn], tot, mu[maxn];
3  bool check[maxn];
4  void CalMu()
5  {
6      mu[1] = 1;
7      for (int i = 2; i < maxn; i++)
8      {
9          if (!check[i]) prime[tot++] = i, mu[i] = -1;
10         for (int j = 0; j < tot; j++)
11         {
12             if (i * prime[j] >= maxn) break;
13             check[i * prime[j]] = true;
14             if (i % prime[j] == 0)
15             {
16                 mu[i * prime[j]] = 0;
17                 break;
18             }
19             else
20                 mu[i * prime[j]] = -mu[i];
21         }
22     }
23 }
```

1.7.2 Examples

有 n 个数 $(n \le 100000, 1 \le a_i \le 10^6)$, 问这 n 个数中互质的数的对数

```
1  const int maxn = "Edit";
2  int b[maxn];
3  ll solve(int n)
4  {
5      ll ans = 0;
6      for (int i = 0, x; i < n; i++) scanf("%d", &x), b[x]++;
7      for (int i = 1; i < maxn; i++)
8      {
9          int cnt = 0;
10         for (int j = i; j < maxn; j += i) cnt += b[j];
11         ans += 1LL * mu[i] * cnt * cnt;
12     }
13     return (ans - b[1]) / 2;
14 }
```

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

```
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 <= n; i++) ans += (ll)mu[i] * (n / i) * (m / i);
6      /*
7      数论分块写法(sum为莫比乌斯函数的前缀和)
```

```
 8      for (int i = 1; i <= n; i = pos + 1)
 9      {
10          pos = min(n / (n / i), m / (m / i));
11          ans += 1LL * (sum[pos] - sum[i - 1]) * (n / i) * (m / i);
12      }
13      */
14      return ans;
15  }
```

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  void change(Complex y[], int len)
12  {
13      for (int i = 1, j = len / 2; i < len - 1; i++)
14      {
15          if (i < j) swap(y[i], y[j]);
16          int k = len / 2;
17          while (j >= k) j -= k, k /= 2;
18          if (j < k) j += k;
19      }
20  }
21  /*
22  * len必须为2^k形式,
23  * on==1时是DFT, on==-1时是IDFT
24  */
25  void fft(Complex y[], int len, int on)
26  {
27      change(y, len);
28      for (int h = 2; h <= len; h <<= 1)
29      {
30          Complex wn(cos(-on * 2 * PI / h), sin(-on * 2 * PI / h));
31          for (int j = 0; j < len; j += h)
32          {
33              Complex w(1, 0);
34              for (int k = j; k < j + h / 2; k++)
35              {
36                  Complex u = y[k];
37                  Complex t = w * y[k + h / 2];
38                  y[k] = u + t, y[k + h / 2] = u - t;
39                  w = w * wn;
40              }
41          }
42      }
43      if (on == -1)
44          for (int i = 0; i < len; i++) y[i].x /= len;
45  }
```

1.8.2　NTT

模数 P 为费马素数，G 为 P 的原根。$G^{\frac{P-1}{n}}$ 具有和 $w_n = e^{\frac{2i\pi}{n}}$ 相似的性质。具体的 P 和 G 可参考 1.11

```cpp
const int mod = 119 << 23 | 1;
const int G = 3;
int wn[20];
void getwn()
{ //  千万不要忘记
    for (int i = 0; i < 20; i++) wn[i] = Pow(G, (mod - 1) / (1 << i), mod);
}
void change(int y[], int len)
{
    for (int i = 1, j = len / 2; i < len - 1; i++)
    {
        if (i < j) swap(y[i], y[j]);
        int k = len / 2;
        while (j >= k) j -= k, k /= 2;
        if (j < k) j += k;
    }
}
void ntt(int y[], int len, int on)
{
    change(y, len);
    for (int h = 2, id = 1; h <= len; h <<= 1, id++)
    {
        for (int j = 0; j < len; j += h)
        {
            int w = 1;
            for (int k = j; k < j + h / 2; k++)
            {
                int u = y[k] % mod;
                int t = 1LL * w * (y[k + h / 2] % mod) % mod;
                y[k] = (u + t) % mod, y[k + h / 2] = ((u - t) % mod + mod) % mod;
                w = 1LL * w * wn[id] % mod;
            }
        }
    }
    if (on == -1)
    {
        //  原本的除法要用逆元
        int inv = Pow(len, mod - 2, mod);
        for (int i = 1; i < len / 2; i++) swap(y[i], y[len - i]);
        for (int i = 0; i < len; i++) y[i] = 1LL * y[i] * inv % mod;
    }
}
```

1.8.3　FWT

```cpp
void fwt(int f[], int m)
{
    int n = __builtin_ctz(m);
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < m; ++j)
            if (j & (1 << i))
            {
                int l = f[j ^ (1 << i)], r = f[j];
                f[j ^ (1 << i)] = l + r, f[j] = l - r;
```

```
10              // or: f[j] += f[j ^ (1 << i)];
11              // and: f[j ^ (1 << i)] += f[j];
12          }
13 }
14 void ifwt(int f[], int m)
15 {
16     int n = __builtin_ctz(m);
17     for (int i = 0; i < n; ++i)
18         for (int j = 0; j < m; ++j)
19             if (j & (1 << i))
20             {
21                 int l = f[j ^ (1 << i)], r = f[j];
22                 f[j ^ (1 << i)] = (l + r) / 2, f[j] = (l - r) / 2;
23                 // 如果有取模需要使用逆元
24                 // or: f[j] -= f[j ^ (1 << i)];
25                 // and: f[j ^ (1 << i)] -= f[j];
26             }
27 }
```

1.9 Numerical Integration

1.9.1 Adaptive Simpson's Rule

$\int_a^b f(x)dx \approx \frac{b-a}{6}[f(a) + 4f(\frac{a+b}{2}) + f(b)]$
$|S(a,c) + S(c,b) - S(a,b)|/15 < \epsilon$

```
1 double F(double x) {}
2 double simpson(double a, double b)
3 { // 三点Simpson法
4     double c = a + (b - a) / 2;
5     return (F(a) + 4 * F(c) + F(b)) * (b - a) / 6;
6 }
7 double asr(double a, double b, double eps, double A)
8 { //自适应Simpson公式 (递归过程) 。已知整个区间[a,b]上的三点Simpson值A
9     double c = a + (b - a) / 2;
10    double L = simpson(a, c), R = simpson(c, b);
11    if (fabs(L + R - A) <= 15 * eps) return L + R + (L + R - A) / 15.0;
12    return asr(a, c, eps / 2, L) + asr(c, b, eps / 2, R);
13 }
14 double asr(double a, double b, double eps) { return asr(a, b, eps, simpson(a, b)); }
```

1.9.2 Berlekamp-Massey

```
1 const int maxn = 1 << 14;
2 ll res[maxn], base[maxn], _c[maxn], _md[maxn];
3 vector<int> Md;
4 void mul(ll* a, ll* b, int k)
5 {
6     for (int i = 0; i < k + k; i++) _c[i] = 0;
7     for (int i = 0; i < k; i++)
8         if (a[i])
9             for (int j = 0; j < k; j++) _c[i + j] = (_c[i + j] + a[i] * b[j]) % mod;
10    for (int i = k + k - 1; i >= k; i--)
11        if (_c[i])
12            for (int j = 0; j < Md.size(); j++) _c[i - k + Md[j]] = (_c[i - k + Md[j]]
    - _c[i] * _md[Md[j]]) % mod;
13    for (int i = 0; i < k; i++) a[i] = _c[i];
14 }
```

```
15  int solve(ll n, VI a, VI b)
16  {
17      ll ans = 0, pnt = 0;
18      int k = a.size();
19      assert(a.size() == b.size());
20      for (int i = 0; i < k; i++) _md[k - 1 - i] = -a[i];
21      _md[k] = 1;
22      Md.clear();
23      for (int i = 0; i < k; i++)
24          if (_md[i] != 0) Md.push_back(i);
25      for (int i = 0; i < k; i++) res[i] = base[i] = 0;
26      res[0] = 1;
27      while ((1LL << pnt) <= n) pnt++;
28      for (int p = pnt; p >= 0; p--)
29      {
30          mul(res, res, k);
31          if ((n >> p) & 1)
32          {
33              for (int i = k - 1; i >= 0; i--) res[i + 1] = res[i];
34              res[0] = 0;
35              for (int j = 0; j < Md.size(); j++) res[Md[j]] = (res[Md[j]] - res[k] * _md
    [Md[j]]) % mod;
36          }
37      }
38      for (int i = 0; i < k; i++) ans = (ans + res[i] * b[i]) % mod;
39      if (ans < 0) ans += mod;
40      return ans;
41  }
42  VI BM(VI s)
43  {
44      VI C(1, 1), B(1, 1);
45      int L = 0, m = 1, b = 1;
46      for (int n = 0; n < s.size(); n++)
47      {
48          ll d = 0;
49          for (int i = 0; i <= L; i++) d = (d + (ll)C[i] * s[n - i]) % mod;
50          if (d == 0)
51              ++m;
52          else if (2 * L <= n)
53          {
54              VI T = C;
55              ll c = mod - d * Pow(b, mod - 2) % mod;
56              while (C.size() < B.size() + m) C.push_back(0);
57              for (int i = 0; i < B.size(); i++) C[i + m] = (C[i + m] + c * B[i]) % mod;
58              L = n + 1 - L, B = T, b = d, m = 1;
59          }
60          else
61          {
62              ll c = mod - d * Pow(b, mod - 2) % mod;
63              while (C.size() < B.size() + m) C.push_back(0);
64              for (int i = 0; i < B.size(); i++) C[i + m] = (C[i + m] + c * B[i]) % mod;
65              ++m;
66          }
67      }
68      return C;
69  }
70  int gao(VI a, ll n)
71  {
72      VI c = BM(a);
```

```
73      c.erase(c.begin());
74      for (int i = 0; i < c.size(); i++) c[i] = (mod - c[i]) % mod;
75      return solve(n, c, VI(a.begin(), a.begin() + c.size()));
76  }
```

1.9.3   Simplex

输入矩阵 a 描述线性规划的标准形式。

a 为 $m+1$ 行 $n+1$ 列，其中行 $0 \sim m-1$ 为不等式，行 m 为目标函数（最大化）。

列 $0 \sim n-1$ 为变量 $0 \sim n-1$ 的系数，列 n 为常数项。

约束为 $a_{i,0}x_0 + a_{i,1}x_1 + \cdots \le a_{i,n}$，目标为 $\max(a_{m,0}x_0 + a_{m,1}x_1 + \cdots + a_{m,n-1}x_{n-1} - a_{m,n})$

注意：变量均有非负约束 $x[i] \ge 0$

```
 1  const int maxm = 500; // 约束数目上限
 2  const int maxn = 500; // 变量数目上限
 3  const double INF = 1e100;
 4  const double eps = 1e-10;
 5  struct Simplex
 6  {
 7      int n;                  // 变量个数
 8      int m;                  // 约束个数
 9      double a[maxm][maxn]; // 输入矩阵
10      int B[maxm], N[maxn]; // 算法辅助变量
11      void pivot(int r, int c)
12      {
13          swap(N[c], B[r]);
14          a[r][c] = 1 / a[r][c];
15          for (int j = 0; j <= n; j++)
16              if (j != c) a[r][j] *= a[r][c];
17          for (int i = 0; i <= m; i++)
18              if (i != r)
19              {
20                  for (int j = 0; j <= n; j++)
21                      if (j != c) a[i][j] -= a[i][c] * a[r][j];
22                  a[i][c] = -a[i][c] * a[r][c];
23              }
24      }
25      bool feasible()
26      {
27          for (;;)
28          {
29              int r, c;
30              double p = INF;
31              for (int i = 0; i < m; i++)
32                  if (a[i][n] < p) p = a[r = i][n];
33              if (p > -eps) return true;
34              p = 0;
35              for (int i = 0; i < n; i++)
36                  if (a[r][i] < p) p = a[r][c = i];
37              if (p > -eps) return false;
38              p = a[r][n] / a[r][c];
39              for (int i = r + 1; i < m; i++)
40                  if (a[i][c] > eps)
41                  {
```

```
42                      double v = a[i][n] / a[i][c];
43                      if (v < p) r = i, p = v;
44                  }
45              pivot(r, c);
46          }
47      }
48      // 解有界返回1, 无解返回0, 无界返回-1。b[i]为x[i]的值, ret为目标函数的值
49      int simplex(int n, int m, double x[maxn], double& ret)
50      {
51          this->n = n, this->m = m;
52          for (int i = 0; i < n; i++) N[i] = i;
53          for (int i = 0; i < m; i++) B[i] = n + i;
54          if (!feasible()) return 0;
55          for (;;)
56          {
57              int r, c;
58              double p = 0;
59              for (int i = 0; i < n; i++)
60                  if (a[m][i] > p) p = a[m][c = i];
61              if (p < eps)
62              {
63                  for (int i = 0; i < n; i++)
64                      if (N[i] < n) x[N[i]] = 0;
65                  for (int i = 0; i < m; i++)
66                      if (B[i] < n) x[B[i]] = a[i][n];
67                  ret = -a[m][n];
68                  return 1;
69              }
70              p = INF;
71              for (int i = 0; i < m; i++)
72                  if (a[i][c] > eps)
73                  {
74                      double v = a[i][n] / a[i][c];
75                      if (v < p) r = i, p = v;
76                  }
77              if (p == INF) return -1;
78              pivot(r, c);
79          }
80      }
81 };
```

1.10 Others

约瑟夫问题
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 }
```

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    }
```

1.11 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：$p\ is\ prime \Rightarrow (p - 1)! \equiv -1 \pmod{p}$

6. 欧拉定理：$\gcd(a, n) = 1 \Rightarrow a^{\varphi(n)} \equiv 1 \pmod{n}$

7. 欧拉定理推广：$\gcd(n, p) = 1 \Rightarrow a^n \equiv a^{n\%\varphi(p)} \pmod{p}$

8. 模的幂公式：$a^n \pmod{m} = \begin{cases} a^n \mod m & n < \varphi(m) \\ a^{n\%\varphi(m)+\varphi(m)} \mod m & n \ge \varphi(m) \end{cases}$

9. 素数定理：对于不大于 n 的素数个数 $\pi(n)$，$\lim_{n \to \infty} \pi(n) = \frac{n}{\ln n}$

10. 位数公式：正整数 x 的位数 $N = \log_{10}(n) + 1$

11. 斯特灵公式 $n! \approx \sqrt{2\pi n}(\frac{n}{e})^n$

12. 设 $a > 1, m, n > 0$，则 $\gcd(a^m - 1, a^n - 1) = a^{\gcd(m,n)} - 1$

13. 设 $a > b, \gcd(a, b) = 1$，则 $\gcd(a^m - b^m, a^n - b^n) = a^{\gcd(m,n)} - b^{\gcd(m,n)}$

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

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

14. 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 $N = \frac{(m-1)(n-1)}{2}$

15. $(n + 1)lcm(C_n^0, C_n^1, ..., C_n^{n-1}, C_n^n) = lcm(1, 2, ..., n + 1)$

16. 若 p 为素数，则 $(x + y + ... + w)^p \equiv x^p + y^p + ... + w^p \pmod{p}$

17. 卡特兰数：1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012
    $h(0) = h(1) = 1, h(n) = \frac{(4n-2)h(n-1)}{n+1} = \frac{C_{2n}^n}{n+1} = C_{2n}^n - C_{2n}^{n-1}$

18. 伯努利数：$B_n = -\frac{1}{n+1}\sum_{i=0}^{n-1} C_{n+1}^i B_i$

$$\sum_{i=1}^{n} i^k = \frac{1}{k+1}\sum_{i=1}^{k+1} C_{k+1}^i B_{k+1-i}(n+1)^i$$

19. 二项式反演：

$$f_n = \sum_{i=0}^{n}(-1)^i \binom{n}{i} g_i \Leftrightarrow g_n = \sum_{i=0}^{n}(-1)^i \binom{n}{i} f_i$$

$$f_n = \sum_{i=0}^{n} \binom{n}{i} g_i \Leftrightarrow g_n = \sum_{i=0}^{n}(-1)^{n-i} \binom{n}{i} f_i$$

20. FFT 常用素数

$r\,2^k+1$	r	k	g
3	1	1	2
5	1	2	2
17	1	4	3
97	3	5	5
193	3	6	5
257	1	8	3
7681	15	9	17
12289	3	12	11
40961	5	13	3
65537	1	16	3
786433	3	18	10
5767169	11	19	3
7340033	7	20	3
23068673	11	21	3
104857601	25	22	3
167772161	5	25	3
469762049	7	26	3
998244353	119	23	3
1004535809	479	21	3
2013265921	15	27	31
2281701377	17	27	3
3221225473	3	30	5
75161927681	35	31	3
77309411329	9	33	7
206158430209	3	36	22
2061584302081	15	37	7
2748779069441	5	39	3
6597069766657	3	41	5
39582418599937	9	42	5
79164837199873	9	43	5
263882790666241	15	44	7
1231453023109121	35	45	3
1337006139375617	19	46	3
3799912185593857	27	47	5
4222124650659841	15	48	19
7881299347898369	7	50	6
31525197391593473	7	52	3
180143985094819841	5	55	6
1945555039024054273	27	56	5
4179340454199820289	29	57	3

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() { memset(ch[sz], -1, sizeof(ch[sz])), 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;
12         for (int i = 0; s[i]; 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;
49         int res = 0;
50         for (int i = 0; s[i]; 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   struct Suffix_Array
4   {
5       char s[maxn];
6       int sa[maxn], t[maxn], t2[maxn], c[maxn], rank[maxn], height[maxn];
7       void build_sa(int m, int n)
8       { //n为字符串的长度,字符集的值为0~m-1
9           n++;
10          int *x = t, *y = t2;
11          //基数排序
12          for (int i = 0; i < m; i++) c[i] = 0;
13          for (int i = 0; i < n; i++) c[x[i] = s[i]]++;
14          for (int i = 1; i < m; i++) c[i] += c[i - 1];
15          for (int i = n - 1; ~i; i--) sa[--c[x[i]]] = i;
16          for (int k = 1; k <= n; k <<= 1)
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，推出
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 = 31 - __builtin_clz(r - l + 1);
60          return min(dp[l][k], dp[r - (1 << k) + 1][k]);
61      }
62      int lcp(int a, int b)
63      { // 求两个后缀的最长公共前缀
64          a = rank[a], b = rank[b];
65          if (a > b) swap(a, b);
66          return rmq(a + 1, b);
67      }
68  };
```

2.6 Suffix Automation

```
1   const int maxn = "Edit";
2   struct SAM
3   {
4       int len[maxn << 1], link[maxn << 1], ch[maxn << 1][26];
5       int num[maxn << 1]; //每个结点所代表的字符串的出现次数
6       int sz, rt, last;
7       int newnode(int x = 0)
8       {
9           len[sz] = x;
10          link[sz] = -1;
11          memset(ch[sz], -1, sizeof(ch[sz]));
12          return sz++;
13      }
14      void init() { sz = last = 0, rt = newnode(); }
15      void reset() { last = 0; }
16      void extend(int c)
17      {
18          int np = newnode(len[last] + 1);
19          int p;
20          for (p = last; ~p && ch[p][c] == -1; p = link[p]) ch[p][c] = np;
21          if (p == -1)
22              link[np] = rt;
23          else
24          {
25              int q = ch[p][c];
26              if (len[p] + 1 == len[q])
27                  link[np] = q;
28              else
29              {
30                  int nq = newnode(len[p] + 1);
31                  memcpy(ch[nq], ch[q], sizeof(ch[q]));
32                  link[nq] = link[q], link[q] = link[np] = nq;
33                  for (; ~p && ch[p][c] == q; p = link[p]) ch[p][c] = nq;
34              }
35          }
36          last = np;
37      }
38      int topcnt[maxn], topsam[maxn << 1];
39      void build(const char* s)
40      { // 加入串后拓扑排序
41          memset(topcnt, 0, sizeof(topcnt));
42          for (int i = 0; i < sz; i++) topcnt[len[i]]++;
43          for (int i = 0; i < maxn - 1; i++) topcnt[i + 1] += topcnt[i];
```

```
44          for (int i = 0; i < sz; i++) topsam[--topcnt[len[i]]] = i;
45          int u = rt;
46          for (int i = 0; s[i]; i++) num[u = ch[u][s[i] - 'a']] = 1;
47          for (int i = sz - 1; ~i; i--)
48          {
49              int u = topsam[i];
50              if (~link[u]) num[link[u]] += num[u];
51          }
52      }
53  };
```

2.7 Palindromic Tree

```
 1  const int maxn = "Edit";
 2  struct Palindromic_Tree
 3  {
 4      int ch[maxn][26], f[maxn], len[maxn], s[maxn];
 5      int cnt[maxn]; // 结点表示的本质不同的回文串的个数(调用count()后)
 6      int num[maxn]; // 结点表示的最长回文串的最右端点为回文串结尾的回文串个数
 7      int last, sz, n;
 8      int newnode(int x)
 9      {
10          memset(ch[sz], 0, sizeof(ch[sz]));
11          cnt[sz] = num[sz] = 0, len[sz] = x;
12          return sz++;
13      }
14      void init()
15      {
16          sz = 0;
17          newnode(0), newnode(-1);
18          last = n = 0, s[0] = -1, f[0] = 1;
19      }
20      int get_fail(int u)
21      {
22          while (s[n - len[u] - 1] != s[n]) u = f[u];
23          return u;
24      }
25      void add(int c)
26      { // c-='a'
27          s[++n] = c;
28          int u = get_fail(last);
29          if (!ch[u][c])
30          {
31              int np = newnode(len[u] + 2);
32              f[np] = ch[get_fail(f[u])][c];
33              num[np] = num[f[np]] + 1;
34              ch[u][c] = np;
35          }
36          last = ch[u][c];
37          cnt[last]++;
38      }
39      void count()
40      {
41          for (int i = sz - 1; ~i; i--) cnt[f[i]] += cnt[i];
42      }
43  };
```

2.8   Hash

```
1  typedef unsigned long long ull;
2  const ull Seed_Pool[] = {146527, 19260817};
3  const ull Mod_Pool[] = {1000000009, 998244353};
4  struct Hash
5  {
6      ull SEED, MOD;
7      vector<ull> p, h;
8      Hash() {}
9      Hash(const string& s, const int& seed_index, const int& mod_index)
10     {
11         SEED = Seed_Pool[seed_index];
12         MOD = Mod_Pool[mod_index];
13         int n = s.length();
14         p.resize(n + 1), h.resize(n + 1);
15         p[0] = 1;
16         for (int i = 1; i <= n; i++) p[i] = p[i - 1] * SEED % MOD;
17         for (int i = 1; i <= n; i++) h[i] = (h[i - 1] * SEED % MOD + s[i - 1]) % MOD;
18     }
19     ull get(int l, int r) { return (h[r] - h[l] * p[r - l] % MOD + MOD) % MOD; }
20     ull substr(int l, int m) { return get(l, l + m); }
21 };
```

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
2  template <class T>
3  struct BIT
4  {
5      vector<T> bit;
6      int n;
7      void init(int n)
8      {
9          this->n = n;
10         bit.assign(n + 1, 0);
11     }
12     void update(int x, T v)
13     {
14         for (; x <= n; x + = x & -x) bit[x] += v
15     }
16     void query(int x)
17     {
18         T ret = 0;
19         for (; x; x -= x & -x) ret += bit[x];
20         return ret;
21     }
22     // 做权值树状数组时求第k小
23     int kth(int k)
24     {
25         int ret = 0, cnt = 0;
26         for (int i = 20; ~i; i--)
27         {
28             ret ^= (1 << i);
29             if (ret > n || cnt + bit[ret] >= k)
30                 ret ^= (1 << i);
31             else
32                 cnt += bit[ret];
33         }
34         return ret + 1;
35     }
36 };
```

3.2 Segment Tree

线段树必须要能够裸写，此处仅留矩形面积周长系列备忘。

3.2.1 Area Combination

```
1  // 矩形面积并
2  map<double, int> Hash;
3  map<int, double> rHash;
4  struct line
5  {
6      double l, r, h;
7      int val;
8      line(double l = 0, double r = 0, double h = 0, int val = 0) : l(l), r(r), h(h), val
       (val) {}
```

```
 9      bool operator<(const line& A) const { return h < A.h; }
10   };
11   struct Node
12   {
13       int cover;
14       double len;
15   };
16   const int maxn = 1000;
17   Node seg[maxn << 2];
18   void build(int rt, int l, int r)
19   {
20       seg[rt].cover = seg[rt].len = 0;
21       if (l == r) return;
22       int mid = l + r >> 1;
23       build(lson, l, mid);
24       build(rson, mid + 1, r);
25   }
26   void pushup(int rt, int l, int r)
27   {
28       if (seg[rt].cover > 0)
29           seg[rt].len = rHash[r + 1] - rHash[l]; // [l,r)
30       else if (l == r)
31           seg[rt].len = 0;
32       else
33           seg[rt].len = seg[lson].len + seg[rson].len;
34   }
35   void update(int rt, int l, int r, int L, int R, int val)
36   {
37       if (L <= l && R >= r)
38       {
39           seg[rt].cover += val;
40           pushup(rt, l, r);
41           return;
42       }
43       int mid = l + r >> 1;
44       if (mid >= L) update(lson, l, mid, L, R, val);
45       if (mid + 1 <= R) update(rson, mid + 1, r, L, R, val);
46       pushup(rt, l, r);
47   }
48   int main()
49   {
50       int n, kase = 0;
51       while (~scanf("%d", &n))
52       {
53           if (!n) break;
54           double x1, x2, y1, y2;
55           vector<line> a;
56           set<double> xval;
57           for (int i = 0; i < n; i++)
58           {
59               scanf("%lf%lf%lf%lf", &x1, &y1, &x2, &y2);
60               a.emplace_back(x1, x2, y1, 1);
61               a.emplace_back(x1, x2, y2, -1);
62               xval.insert(x1);
63               xval.insert(x2);
64           }
65           // 离散化
66           Hash.clear(), rHash.clear();
67           int cnt = 0;
```

```
68          for (auto& v : xval)
69          {
70              Hash[v] = ++cnt;
71              rHash[cnt] = v;
72          }
73          sort(a.begin(), a.end());
74          build(1, 1, cnt);
75          double ans = 0;
76          for (int i = 0; i < a.size() - 1; i++)
77          {
78              update(1, 1, cnt, Hash[a[i].l], Hash[a[i].r] - 1,
79                      a[i].val); //[l,r)
80              ans += (a[i + 1].h - a[i].h) * seg[1].len;
81          }
82          printf("Test case #%d\n", ++kase);
83          printf("Total explored area: %.2lf\n\n", ans);
84      }
85  }
```

3.2.2 Area Intersection

```
1  // 矩形面积交
2  map<double, int> Hash;
3  map<int, double> rHash;
4  struct Lines
5  {
6      double l, r, h;
7      int val;
8      bool operator<(const Lines& A) const { return h < A.h; }
9  };
10 struct Node
11 {
12     int cnt;      //  覆盖次数
13     double len1; // 覆盖次数大于0的长度
14     double len2; // 覆盖次数大于1的长度
15 };
16 Node seg[maxn << 2];
17 void build(int rt, int l, int r)
18 {
19     seg[rt].cnt = seg[rt].len1 = seg[rt].len2 = 0;
20     if (l == r) return;
21     int mid = l + r >> 1;
22     build(lson, l, mid);
23     build(rson, mid + 1, r);
24 }
25 inline void pushup(int rt, int l, int r)
26 {
27     if (seg[rt].cnt > 1)
28         seg[rt].len1 = seg[rt].len2 = rHash[r + 1] - rHash[l];
29     else if (seg[rt].cnt == 1)
30     {
31         seg[rt].len1 = rHash[r + 1] - rHash[l];
32         if (l == r)
33             seg[rt].len2 = 0;
34         else
35             seg[rt].len2 = seg[lson].len1 + seg[rson].len1;
36     }
37     else
```

```
38          {
39              if (l == r)
40                  seg[rt].len1 = seg[rt].len2 = 0;
41              else
42              {
43                  seg[rt].len1 = seg[lson].len1 + seg[rson].len1;
44                  seg[rt].len2 = seg[lson].len2 + seg[rson].len2;
45              }
46          }
47      }
48      void update(int rt, int l, int r, int L, int R, int val)
49      {
50          if (L <= l && R >= r)
51          {
52              seg[rt].cnt += val;
53              pushup(rt, l, r);
54              return;
55          }
56          int mid = l + r >> 1;
57          if (L <= mid) update(lson, l, mid, L, R, val);
58          if (R >= mid + 1) update(rson, mid + 1, r, L, R, val);
59          pushup(rt, l, r);
60      }
61      int main()
62      {
63          int T;
64          scanf("%d", &T);
65          while (T--)
66          {
67              int n;
68              scanf("%d", &n);
69              double x1, x2, y1, y2;
70              vector<Lines> line;
71              set<double> X;
72              for (int i = 1; i <= n; i++)
73              {
74                  scanf("%lf%lf%lf%lf", &x1, &y1, &x2, &y2);
75                  line.push_back({x1, x2, y1, 1});
76                  line.push_back({x1, x2, y2, -1});
77                  X.insert(x1);
78                  X.insert(x2);
79              }
80              sort(line.begin(), line.end());
81              int cnt = 0;
82              Hash.clear();
83              rHash.clear();
84              for (auto& v : X) Hash[v] = ++cnt, rHash[cnt] = v;
85              build(1, 1, cnt);
86              double area = 0;
87              for (int i = 0; i < line.size() - 1; i++)
88              {
89                  update(1, 1, cnt, Hash[line[i].l], Hash[line[i].r] - 1, line[i].val);
90                  area += seg[1].len2 * (line[i + 1].h - line[i].h);
91              }
92              printf("%.2lf\n", area);
93          }
94      }
```

3.2.3　Perimeter Combination

```
1   // 矩形周长并
2   int n, m[2];
3   int sum[maxn << 2], cnt[maxn << 2], all[2][maxn];
4   struct Seg
5   {
6       int l, r, h, d;
7       Seg() {}
8       Seg(int l, int r, int h, int d) : l(l), r(r), h(h), d(d) {}
9       bool operator<(const Seg& rhs) const { return h < rhs.h; }
10  } a[2][maxn];
11  #define lson l, m, rt << 1
12  #define rson m + 1, r, rt << 1 | 1
13  void pushup(int p, int l, int r, int rt)
14  {
15      if (cnt[rt])
16          sum[rt] = all[p][r + 1] - all[p][l];
17      else if (l == r)
18          sum[rt] = 0;
19      else
20          sum[rt] = sum[rt << 1] + sum[rt << 1 | 1];
21  }
22  void update(int p, int L, int R, int v, int l, int r, int rt)
23  {
24      if (L <= l && r <= R)
25      {
26          cnt[rt] += v;
27          pushup(p, l, r, rt);
28          return;
29      }
30      int m = l + r >> 1;
31      if (L <= m) update(p, L, R, v, lson);
32      if (R > m) update(p, L, R, v, rson);
33      pushup(p, l, r, rt);
34  }
35  int main()
36  {
37      while (scanf("%d", &n) == 1)
38      {
39          for (int i = 1; i <= n; ++i)
40          {
41              int x1, y1, x2, y2;
42              scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
43              all[0][i] = x1, all[0][i + n] = x2;
44              all[1][i] = y1, all[1][i + n] = y2;
45              a[0][i] = Seg(x1, x2, y1, 1);
46              a[0][i + n] = Seg(x1, x2, y2, -1);
47              a[1][i] = Seg(y1, y2, x1, 1);
48              a[1][i + n] = Seg(y1, y2, x2, -1);
49          }
50          n <<= 1;
51          sort(all[0] + 1, all[0] + 1 + n);
52          m[0] = unique(all[0] + 1, all[0] + 1 + n) - all[0] - 1;
53          sort(all[1] + 1, all[1] + 1 + n);
54          m[1] = unique(all[1] + 1, all[1] + 1 + n) - all[1] - 1;
55          sort(a[0] + 1, a[0] + 1 + n);
56          sort(a[1] + 1, a[1] + 1 + n);
57          int ans = 0;
```

```
58        for (int i = 0; i < 2; ++i)
59        {
60            int t = 0, last = 0;
61            memset(cnt, 0, sizeof cnt);
62            memset(sum, 0, sizeof sum);
63            for (int j = 1; j <= n; ++j)
64            {
65                int l = lower_bound(all[i] + 1, all[i] + 1 + m[i], a[i][j].l) - all[i];
66                int r = lower_bound(all[i] + 1, all[i] + 1 + m[i], a[i][j].r) - all[i];
67                if (l < r) update(i, l, r - 1, a[i][j].d, 1, m[i], 1);
68                t += abs(sum[1] - last);
69                last = sum[1];
70            }
71            ans += t;
72        }
73        printf("%d\n", ans);
74    }
75    return 0;
76 }
```

3.3 Splay Tree

```
1  #define key_value ch[ch[root][1]][0]
2  const int maxn = "Edit";
3  struct Splay
4  {
5      int a[maxn];
6      int sz[maxn], ch[maxn][2], fa[maxn];
7      int key[maxn], rev[maxn];
8      int root, tot;
9      int stk[maxn], top;
10     void init(int n)
11     {
12         tot = 0, top = 0;
13         root = newnode(0, -1);
14         ch[root][1] = newnode(root, -1);
15         for (int i = 0; i < n; i++) a[i] = i + 1;
16         key_value = build(0, n - 1, ch[root][1]);
17         pushup(ch[root][1]);
18         pushup(root);
19     }
20     int newnode(int p = 0, int k = 0)
21     {
22         int x = top ? stk[top--] : ++tot;
23         fa[x] = p;
24         sz[x] = 1;
25         ch[x][0] = ch[x][1] = 0;
26         key[x] = k;
27         rev[x] = 0;
28         return x;
29     }
30     void pushdown(int x)
31     {
32         if (rev[x])
33         {
34             swap(ch[x][0], ch[x][1]);
35             if (ch[x][0]) rev[ch[x][0]] ^= 1;
36             if (ch[x][1]) rev[ch[x][1]] ^= 1;
```

```
37              rev[x] = 0;
38          }
39      }
40      void pushup(int x) { sz[x] = sz[ch[x][0]] + sz[ch[x][1]] + 1; }
41      void rotate(int x, int d)
42      {
43          int y = fa[x];
44          pushdown(y), pushdown(x);
45          ch[y][d ^ 1] = ch[x][d];
46          fa[ch[x][d]] = y;
47          if (fa[y]) ch[fa[y]][ch[fa[y]][1] == y] = x;
48          fa[x] = fa[y];
49          ch[x][d] = y;
50          fa[y] = x;
51          pushup(y);
52      }
53      void splay(int x, int goal = 0)
54      {
55          pushdown(x);
56          while (fa[x] != goal)
57          {
58              if (fa[fa[x]] == goal)
59                  rotate(x, ch[fa[x]][0] == x);
60              else
61              {
62                  int y = fa[x];
63                  int d = ch[fa[y]][0] == y;
64                  ch[y][d] == x ? rotate(x, d ^ 1) : rotate(y, d);
65                  rotate(x, d);
66              }
67          }
68          pushup(x);
69          if (goal == 0) root = x;
70      }
71      int kth(int r, int k)
72      {
73          pushdown(r);
74          int t = sz[ch[r][0]] + 1;
75          if (t == k) return r;
76          return t > k ? kth(ch[r][0], k) : kth(ch[r][1], k - t);
77      }
78      int build(int l, int r, int p)
79      {
80          if (l > r) return 0;
81          int mid = l + r >> 1;
82          int x = newnode(p, a[mid]);
83          ch[x][0] = build(l, mid - 1, x);
84          ch[x][1] = build(mid + 1, r, x);
85          pushup(x);
86          return x;
87      }
88      void select(int l, int r)
89      {
90          splay(kth(root, l), 0);
91          splay(kth(ch[root][1], r - l + 2), root);
92      }
93      // 各种操作
94  };
```

3.4 Functional Segment Tree

静态查询区间第 k 小的值
必要时进行离散化

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

3.5 Sparse Table

```
1   const int maxn = "Edit";
2   int dp[maxn][20];
3   int a[maxn];
4   void init(int n)
5   {
6       for (int i = 1; i <= n; i++) dp[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               dp[i][j] = max(dp[i][j - 1], dp[i + (1 << (j - 1))][j - 1]);
10  }
11  // 返回[l,r]最大值
12  int rmq(int l, int r, int op)
13  {
14      int k = 31 - __builtin_clz(r - l + 1);
15      return max(dp[l][k], dp[r - (1 << k) + 1][k]);
16  }
```

二维 RMQ

```
1   void init(int n, int m)
2   {
3       for (int i = 0; (1 << i) <= n; i++)
4           for (int j = 0; (1 << j) <= m; j++)
5           {
6               if (i == 0 && j == 0) continue;
7               for (int row = 1; row + (1 << i) - 1 <= n; row++)
8                   for (int col = 1; col + (1 << j) - 1 <= m; col++)
9                       if (i)
10                          dp[row][col][i][j] = max(dp[row][col][i - 1][j],
```

```
11                                              dp[row + (1 << (i - 1))][col][i - 1][j]);
12                      else
13                          dp[row][col][i][j] = max(dp[row][col][i][j - 1],
14                                              dp[row][col + (1 << (j - 1))][i][j - 1]);
15              }
16  }
17  int rmq(int x1, int y1, int x2, int y2)
18  {
19      int kx = 31 - __builtin_clz(x2 - x1 + 1);
20      int ky = 31 - __builtin_clz(y2 - y1 + 1);
21      int m1 = dp[x1][y1][kx][ky];
22      int m2 = dp[x2 - (1 << kx) + 1][y1][kx][ky];
23      int m3 = dp[x1][y2 - (1 << ky) + 1][kx][ky];
24      int m4 = dp[x2 - (1 << kx) + 1][y2 - (1 << ky) + 1][kx][ky];
25      return max({m1, m2, m3, m4});
26  }
```

3.6 Heavy-Light Decomposition

```
1   const int maxn = "Edit";
2   struct HLD
3   {
4       int n, dfs_clock;
5       int sz[maxn], top[maxn], son[maxn], dep[maxn], fa[maxn], id[maxn];
6       vector<int> G[maxn];
7       // vector<pair<PII, int>> edges; 维护边权时，将其下放为儿子结点的点权
8       void init(int n)
9       {
10          this->n = n, memset(son, -1, sizeof(son)), dfs_clock = 0;
11          for (int i = 0; i <= n; i++) G[i].clear();
12      }
13      void add_edge(int u, int v) { G[u].push_back(v), G[v].push_back(u); }
14      void dfs(int u, int p, int d)
15      {
16          dep[u] = d, fa[u] = p, sz[u] = 1;
17          for (auto& v : G[u])
18          {
19              if (v == p) continue;
20              dfs(v, u, d + 1);
21              sz[u] += sz[v];
22              if (son[u] == -1 || sz[v] > sz[son[u]]) son[u] = v;
23          }
24      }
25      void link(int u, int t)
26      {
27          top[u] = t, id[u] = ++dfs_clock;
28          if (son[u] == -1) return;
29          link(son[u], t);
30          for (auto& v : G[u])
31              if (v != son[u] && v != fa[u]) link(v, v);
32      }
33      int query_path(int u, int v)
34      { // 数据结构相关操作，一般使用线段树或者树状数组
35          int ret = 0;
36          while (top[u] != top[v])
37          {
38              if (dep[top[u]] < dep[top[v]]) swap(u, v);
39              ret += query(id[top[u]], id[u]);
```

```
40              u = fa[top[u]];
41          }
42          if (dep[u] > dep[v]) swap(u, v);
43          ret += query(id[u], id[v]);
44          /* 边权
45          if (u == v) return ret;
46          if (dep[u] > dep[v]) swap(u, v);
47          ret += query(id[son[u]], id[v]);
48          */
49          return ret;
50      }
51  };
```

3.7   Link-Cut Tree

动态维护一个森林

```
1   const int maxn = "Edit";
2   struct LCT
3   {
4       int val[maxn], sum[maxn]; // 基于点权
5       int rev[maxn], ch[maxn][2], fa[maxn];
6       int stk[maxn];
7       inline void init(int n)
8       { // 初始化点权
9           for (int i = 1; i <= n; i++) scanf("%d", val + i);
10          for (int i = 1; i <= n; i++)
11              fa[i] = ch[i][0] = ch[i][1] = rev[i] = 0;
12      }
13      inline bool isroot(int x) { return ch[fa[x]][0] != x && ch[fa[x]][1] != x; }
14      inline bool get(int x) { return ch[fa[x]][1] == x; }
15      void pushdown(int x)
16      {
17          if (!rev[x]) return;
18          swap(ch[x][0], ch[x][1]);
19          if (ch[x][0]) rev[ch[x][0]] ^= 1;
20          if (ch[x][1]) rev[ch[x][1]] ^= 1;
21          rev[x] ^= 1;
22      }
23      void pushup(int x) { sum[x] = val[x] + sum[ch[x][0]] + sum[ch[x][1]]; }
24      void rotate(int x)
25      {
26          int y = fa[x], z = fa[fa[x]], d = get(x);
27          if (!isroot(y)) ch[z][get(y)] = x;
28          fa[x] = z;
29          ch[y][d] = ch[x][d ^ 1], fa[ch[y][d]] = y;
30          ch[x][d ^ 1] = y, fa[y] = x;
31          pushup(y), pushup(x);
32      }
33      void splay(int x)
34      {
35          int top = 0;
36          stk[++top] = x;
37          for (int i = x; !isroot(i); i = fa[i]) stk[++top] = fa[i];
38          for (int i = top; i; i--) pushdown(stk[i]);
39          for (int f; !isroot(x); rotate(x))
40              if (!isroot(f = fa[x])) rotate(get(x) == get(f) ? f : x);
41      }
```

```
42    void access(int x)
43    {
44        for (int y = 0; x; y = x, x = fa[x]) splay(x), ch[x][1] = y, pushup(x);
45    }
46    int find(int x)
47    {
48        access(x), splay(x);
49        while (ch[x][0]) x = ch[x][0];
50        return x;
51    }
52    void makeroot(int x) { access(x), splay(x), rev[x] ^= 1; }
53    void link(int x, int y) { makeroot(x), fa[x] = y, splay(x); }
54    void cut(int x, int y) { makeroot(x), access(y), splay(y), fa[x] = ch[y][0] = 0; }
55    void update(int x, int v) { val[x] = v, access(x), splay(x); }
56    int query(int x, int y)
57    {
58        makeroot(y), access(x), splay(x);
59        return sum[x];
60    }
61  };
```

3.8 Virtual Tree

```
1   const int maxn = "Edit";
2   vector<int> vtree[maxn];
3   void build(vector<int>& vec)
4   {
5       sort(vec.begin(), vec.end(), [&](int x, int y) { return dfn[x] < dfn[y]; });
6       static int s[maxn];
7       int top = 0;
8       s[top] = 0;
9       vtree[0].clear();
10      for (auto& u : vec)
11      {
12          int vlca = lca(u, s[top]);
13          vtree[u].clear();
14          if (vlca == s[top])
15              s[++top] = u;
16          else
17          {
18              while (top && dep[s[top - 1]] >= dep[vlca])
19              {
20                  vtree[s[top - 1]].push_back(s[top]);
21                  top--;
22              }
23              if (s[top] != vlca)
24              {
25                  vtree[vlca].clear();
26                  vtree[vlca].push_back(s[top--]);
27                  s[++top] = vlca;
28              }
29              s[++top] = u;
30          }
31      }
32      for (int i = 0; i < top; ++i) vtree[s[i]].push_back(s[i + 1]);
33  }
```

3.9 Cartesian Tree

```
1  const int maxn = "Edit";
2  int lson[maxn], rson[maxn], fa[maxn];
3  void build(int n)
4  {
5      stack<int> s;
6      for (int i = 0; i < n; i++)
7      {
8          int last = -1;
9          while (!s.empty() && a[i] > a[s.top()]) last = s.top(), s.pop();
10         if (!s.empty()) rson[s.top()] = i, fa[i] = s.top();
11         lson[i] = last;
12         if (~last) fa[last] = i;
13         s.push(i);
14     }
15 }
```

4 Graph Theory

4.1 Shortest Path

```
1  struct Edge
2  {
3      int from, to, dist;
4      Edge() {}
5      Edge(int u, int v, int d) : from(u), to(v), dist(d) {}
6  };
```

4.1.1 Dijkstra

```
1  struct HeapNode
2  {
3      int d, u;
4      bool operator<(const HeapNode& rhs) const
5      {
6          return d > rhs.d;
7      }
8  };
9  const int maxn = "Edit";
10 struct Dijkstra
11 {
12     int n, m;                // 点数和边数
13     vector<Edge> edges;      // 边列表
14     vector<int> G[maxn];     // 每个节点出发的边编号 (从0开始编号)
15     bool done[maxn];         // 是否已永久标号
16     int d[maxn];             // s到各点的距离
17     int p[maxn];             // 最短路中的一条边
18     void init(int n)
19     {
20         this->n = n;
21         for (int i = 0; i < n; i++) G[i].clear(); // 清空邻接表
22         edges.clear();                            // 清空边表
23     }
24     void AddEdge(int from, int to, int dist)
25     { // 如果是无向图, 每条无向边需调用两次AddEdge
26         edges.emplace_back(from, to, dist);
27         m = edges.size();
28         G[from].push_back(m - 1);
29     }
30     void dijkstra(int s)
31     {
32         priority_queue<HeapNode> q;
33         for (int i = 0; i < n; i++) d[i] = INF;
34         d[s] = 0;
35         memset(done, 0, sizeof(done));
36         q.push({0, s});
37         while (!q.empty())
38         {
39             HeapNode x = q.top();
40             q.pop();
41             int u = x.u;
42             if (done[u]) continue;
43             done[u] = true;
44             for (auto& id : G[u])
45             {
```

```
46                    Edge& e = edges[id];
47                    if (d[e.to] > d[u] + e.dist)
48                    {
49                        d[e.to] = d[u] + e.dist;
50                        p[e.to] = id;
51                        q.push({d[e.to], e.to});
52                    }
53                }
54            }
55        }
56  };
```

4.1.2 Bellman-Ford

```
1   const int maxn = "Edit";
2   struct BellmanFord
3   {
4       int n, m;
5       vector<Edge> edges;
6       vector<int> G[maxn];
7       bool inq[maxn]; // 是否在队列中
8       int d[maxn];     // s到各个点的距离
9       int p[maxn];     // 最短路中的上一条弧
10      int cnt[maxn];  // 进队次数
11      void init(int n)
12      {
13          this->n = n;
14          for (int i = 0; i < n; i++) G[i].clear();
15          edges.clear();
16      }
17      void AddEdge(int from, int to, int dist)
18      {
19          edges.emplace_back(from, to, dist);
20          m = edges.size();
21          G[from].push_back(m - 1);
22      }
23      bool bellmanford(int s)
24      {
25          queue<int> q;
26          memset(inq, 0, sizeof(inq));
27          memset(cnt, 0, sizeof(cnt));
28          for (int i = 0; i < n; i++) d[i] = INF;
29          d[s] = 0;
30          inq[s] = true;
31          q.push(s);
32          while (!q.empty())
33          {
34              int u = q.front();
35              q.pop();
36              inq[u] = false;
37              for (auto& id : G[u])
38              {
39                  Edge& e = edges[id];
40                  if (d[u] < INF && d[e.to] > d[u] + e.dist)
41                  {
42                      d[e.to] = d[u] + e.dist;
43                      p[e.to] = id;
44                      if (!inq[e.to])
```

```
45                      {
46                          q.push(e.to);
47                          inq[e.to] = true;
48                          if (++cnt[e.to] > n) return false;
49                      }
50                  }
51              }
52          }
53          return true;
54      }
55  };
```

4.2   Minimal Spanning Tree

4.2.1   Zhu Liu

```
1   const int maxn = "Edit";
2   // 固定根的最小树型图，邻接矩阵写法
3   struct MDST
4   {
5       int n;
6       int w[maxn][maxn]; // 边权
7       int vis[maxn];      // 访问标记，仅用来判断无解
8       int ans;            // 计算答案
9       int removed[maxn]; // 每个点是否被删除
10      int cid[maxn];      // 所在圈编号
11      int pre[maxn];      // 最小入边的起点
12      int iw[maxn];       // 最小入边的权值
13      int max_cid;        // 最大圈编号
14      void init(int n)
15      {
16          this->n = n;
17          for (int i = 0; i < n; i++)
18              for (int j = 0; j < n; j++) w[i][j] = INF;
19      }
20      void AddEdge(int u, int v, int cost)
21      {
22          w[u][v] = min(w[u][v], cost); // 重边取权最小的
23      }
24      // 从s出发能到达多少个结点
25      int dfs(int s)
26      {
27          vis[s] = 1;
28          int ans = 1;
29          for (int i = 0; i < n; i++)
30              if (!vis[i] && w[s][i] < INF) ans += dfs(i);
31          return ans;
32      }
33      // 从u出发沿着pre指针找圈
34      bool cycle(int u)
35      {
36          max_cid++;
37          int v = u;
38          while (cid[v] != max_cid)
39          {
40              cid[v] = max_cid;
41              v = pre[v];
42          }
43          return v == u;
```

```
44          }
45          // 计算u的最小入弧，入弧起点不得在圈c中
46          void update(int u)
47          {
48              iw[u] = INF;
49              for (int i = 0; i < n; i++)
50                  if (!removed[i] && w[i][u] < iw[u])
51                  {
52                      iw[u] = w[i][u];
53                      pre[u] = i;
54                  }
55          }
56          // 根结点为s，如果失败则返回false
57          bool solve(int s)
58          {
59              memset(vis, 0, sizeof(vis));
60              if (dfs(s) != n) return false;
61              memset(removed, 0, sizeof(removed));
62              memset(cid, 0, sizeof(cid));
63              for (int u = 0; u < n; u++) update(u);
64              pre[s] = s;
65              iw[s] = 0; // 根结点特殊处理
66              ans = max_cid = 0;
67              for (;;)
68              {
69                  bool have_cycle = false;
70                  for (int u = 0; u < n; u++)
71                      if (u != s && !removed[u] && cycle(u))
72                      {
73                          have_cycle = true;
74                          // 以下代码缩圈，圈上除了u之外的结点均删除
75                          int v = u;
76                          do
77                          {
78                              if (v != u) removed[v] = 1;
79                              ans += iw[v];
80                              // 对于圈外点i，把边i->v改成i->u（并调整权值）；v->i改为u->i
81                              // 注意圈上可能还有一个v'使得i->v'或者v'->i存在，
82                              // 因此只保留权值最小的i->u和u->i
83                              for (int i = 0; i < n; i++)
84                                  if (cid[i] != cid[u] && !removed[i])
85                                  {
86                                      if (w[i][v] < INF)
87                                          w[i][u] = min(w[i][u], w[i][v] - iw[v]);
88                                      w[u][i] = min(w[u][i], w[v][i]);
89                                      if (pre[i] == v) pre[i] = u;
90                                  }
91                              v = pre[v];
92                          } while (v != u);
93                          update(u);
94                          break;
95                      }
96                  if (!have_cycle) break;
97              }
98              for (int i = 0; i < n; i++)
99                  if (!removed[i]) ans += iw[i];
100             return true;
101         }
102     };
```

4.3 LCA

4.3.1 DFS+ST

DFS+ST 在线算法
时间复杂度 $O(nlogn + q)$

```
1  const int maxn = "Edit";
2  vector<int> G[maxn], sp;
3  int dep[maxn], dfn[maxn];
4  PII dp[21][maxn << 1];
5  void init(int n)
6  {
7      for (int i = 0; i < n; i++) G[i].clear();
8      sp.clear();
9  }
10 void dfs(int u, int fa)
11 {
12     dep[u] = dep[fa] + 1;
13     dfn[u] = sp.size();
14     sp.push_back(u);
15     for (auto& v : G[u])
16     {
17         if (v == fa) continue;
18         dfs(v, u);
19         sp.push_back(u);
20     }
21 }
22 void initrmq()
23 {
24     int n = sp.size();
25     for (int i = 0; i < n; i++) dp[0][i] = {dfn[sp[i]], sp[i]};
26     for (int i = 1; (1 << i) <= n; i++)
27         for (int j = 0; j + (1 << i) - 1 < n; j++)
28             dp[i][j] = min(dp[i - 1][j], dp[i - 1][j + (1 << (i - 1))]);
29 }
30 int lca(int u, int v)
31 {
32     int l = dfn[u], r = dfn[v];
33     if (l > r) swap(l, r);
34     int k = 31 - __builtin_clz(r - l + 1);
35     return min(dp[k][l], dp[k][r - (1 << k) + 1]).second;
36 }
```

4.3.2 Tarjan

Tarjan 离线算法
时间复杂度 $O(n + q)$

```
1  const int maxn = "Edit";
2  int par[maxn];           //并查集
3  int ans[maxn];           //存储答案
4  vector<int> G[maxn];     //邻接表
5  vector<PII> query[maxn]; //存储查询信息
6  bool vis[maxn];          //是否被遍历
7  inline void init(int n)
8  {
9      for (int i = 1; i <= n; i++)
10     {
```

```
11          G[i].clear(), query[i].clear();
12          par[i] = i, vis[i] = 0;
13      }
14  }
15  inline void add_edge(int u, int v) { G[u].push_back(v); }
16  inline void add_query(int id, int u, int v)
17  {
18      query[u].emplace_back(v, id);
19      query[v].emplace_back(u, id);
20  }
21  void tarjan(int u)
22  {
23      vis[u] = 1;
24      for (auto& v : G[u])
25      {
26          if (vis[v]) continue;
27          tarjan(v);
28          unite(u, v);
29      }
30      for (auto& q : query[u])
31      {
32          int &v = q.X, &id = q.Y;
33          if (!vis[v]) continue;
34          ans[id] = find(v);
35      }
36  }
```

4.4 Depth-First Traversal

4.4.1 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].push_back(v), G[v].push_back(u); }
11 int dfs(int u, int fa)
12 {
13     int lowu = pre[u] = ++dfs_clock;
14     int child = 0;
15     for (auto& v : G[u])
16     {
17         PII e = {u, v};
18         if (!pre[v])
19         {
20             //没有访问过v
21             s.push(e);
22             child++;
23             int lowv = dfs(v, u);
24             lowu = min(lowu, lowv); //用后代的low函数更新自己
25             if (lowv >= pre[u])
26             {
27                 iscut[u] = true;
28                 bcc_cnt++;
```

```
29                    bcc[bcc_cnt].clear(); //注意! bcc从1开始编号
30                    for (;;)
31                    {
32                        PII x = s.top();
33                        s.pop();
34                        if (bccno[x.first] != bcc_cnt)
35                            bcc[bcc_cnt].push_back(x.first), bcc[x.first] = bcc_cnt;
36                        if (bccno[x.second] != bcc_cnt)
37                            bcc[bcc_cnt].push_back(x.second), bcc[x.second] = bcc_cnt;
38                        if (x.first == u && x.second == v) break;
39                    }
40                }
41            }
42            else if (pre[v] < pre[u] && v != fa)
43            {
44                s.push(e);
45                lowu = min(lowu, pre[v]); //用反向边更新自己
46            }
47        }
48        if (fa < 0 && child == 1) iscut[u] = 0;
49        return lowu;
50 }
51 void find_bcc(int n)
52 {
53     //调用结束后s保证为空，所以不用清空
54     memset(pre, 0, sizeof(pre));
55     memset(iscut, 0, sizeof(iscut));
56     memset(bccno, 0, sizeof(bccno));
57     dfs_clock = bcc_cnt = 0;
58     for (int i = 0; i < n; i++)
59         if (!pre[i]) dfs(i, -1);
60 }
```

4.4.2 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 init(int n)
 6 {
 7     for (int i = 0; i < n; i++) G[i].clear();
 8 }
 9 inline void add_edge(int u, int v) { G[u].push_back(v); }
10 void dfs(int u)
11 {
12     pre[u] = lowlink[u] = ++dfs_clock;
13     S.push(u);
14     for (auto& v : G[u])
15     {
16         if (!pre[v])
17         {
18             dfs(v);
19             lowlink[u] = min(lowlink[u], lowlink[v]);
20         }
21         else if (!sccno[v])
22             lowlink[u] = min(lowlink[u], pre[v]);
23     }
```

```
24      if (lowlink[u] == pre[u])
25      {
26          scc_cnt++;
27          for (;;)
28          {
29              int x = S.top();
30              S.pop();
31              sccno[x] = scc_cnt;
32              if (x == u) break;
33          }
34      }
35  }
36  void find_scc(int n)
37  {
38      dfs_clock = 0, scc_cnt = 0;
39      memset(sccno, 0, sizeof(sccno)), memset(pre, 0, sizeof(pre));
40      for (int i = 0; i < n; i++)
41          if (!pre[i]) dfs(i);
42  }
```

4.4.3 2-SAT

```
1   const int maxn = "Edit";
2   struct TwoSAT
3   {
4       int n;
5       vector<int> G[maxn << 1];
6       bool mark[maxn << 1];
7       int S[maxn << 1], c;
8       void init(int n)
9       {
10          this->n = n;
11          for (int i = 0; i < (n << 1); i++) G[i].clear();
12          memset(mark, 0, sizeof(mark));
13      }
14      bool dfs(int x)
15      {
16          if (mark[x ^ 1]) return false;
17          if (mark[x]) return true;
18          mark[x] = true;
19          S[c++] = x;
20          for (auto& y : G[x])
21              if (!dfs(y)) return false;
22          return true;
23      }
24      //x = xval or y = yval
25      void add_clause(int x, int xval, int y, int yval)
26      {
27          x = (x << 1) + xval;
28          y = (y << 1) + yval;
29          G[x ^ 1].push_back(y);
30          G[y ^ 1].push_back(x);
31      }
32      bool solve()
33      {
34          for (int i = 0; i < (n << 1); i += 2)
35              if (!mark[i] && !mark[i + 1])
36              {
```

```
37                c = 0;
38                if (!dfs(i))
39                {
40                    while (c > 0) mark[S[--c]] = false;
41                    if (!dfs(i + 1)) return false;
42                }
43            }
44        return true;
45    }
46 };
```

4.5 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

4.5.1 Fleury

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。

```
1  const int maxn = "Edit";
2  int G[maxn][maxn];
3  int deg[maxn][maxn];
4  vector<int> ans;
5  inline void init() { memset(G, 0, sizeof(G)), memset(deg, 0, sizeof(deg)); }
6  inline void AddEdge(int u, int v) { deg[u]++, deg[v]++, G[u][v]++, G[v][u]++; }
7  void Fleury(int s)
8  {
9      for (int i = 0; i < n; i++)
10         if (G[s][i])
11         {
12             G[s][i]--, G[i][s]--;
13             Fleury(i);
14         }
15     ans.push_back(s);
16 }
```

4.6 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。

4.6.1 Hungry(Matrix)

时间复杂度:$O(VE)$.
顶点编号从 0 开始

```
const int maxn = "Edit";
int uN, vN;          //uN是匹配左边的顶点数,vN是匹配右边的顶点数
int g[maxn][maxn]; //邻接矩阵g[i][j]表示i->j的有向边就可以了,是左边向右边的匹配
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;
    memset(linker, -1, sizeof(linker));
    for (int u = 0; u < uN; u++)
    {
        memset(used, 0, sizeof(used));
        if (dfs(u)) res++;
    }
    return res;
}
```

4.6.2 Hungry(List)

使用前用 init() 进行初始化
加边使用函数 addedge(u,v)

```cpp
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].push_back(v); }
bool dfs(int u)
{
    for (auto& v : G[u])
    {
        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;
    memset(linker, -1, sizeof(linker));
    for (int u = 0; u < n; u++)
    {
        memset(used, 0, sizeof(used));
        if (dfs(u)) ans++;
    }
    return ans;
}
```

4.6.3 Hopcroft-Carp

复杂度 $O(\sqrt{n} * E)$
uN 为左端的顶点数, 使用前赋值 (点编号 0 开始)

```cpp
const int maxn = "Edit";
vector<int> G[maxn];
int uN, dis;
int Mx[maxn], My[maxn];
int dx[maxn], dy[maxn];
bool used[maxn];
inline void init(int n)
{
    for (int i = 0; i < n; i++) G[i].clear();
}
```

```cpp
11  inline void addedge(int u, int v) { G[u].push_back(v); }
12  bool bfs()
13  {
14      queue<int> q;
15      dis = INF;
16      memset(dx, -1, sizeof(dx)), memset(dy, -1, sizeof(dy));
17      for (int i = 0; i < uN; i++)
18          if (Mx[i] == -1) q.push(i), dx[i] = 0;
19      while (!q.empty())
20      {
21          int u = q.front();
22          q.pop();
23          if (dx[u] > dis) break;
24          for (auto& v : G[u])
25          {
26              if (dy[v] == -1)
27              {
28                  dy[v] = dx[u] + 1;
29                  if (My[v] == -1)
30                      dis = dy[v];
31                  else
32                  {
33                      dx[My[v]] = dy[v] + 1;
34                      q.push(My[v]);
35                  }
36              }
37          }
38      }
39      return dis != INF;
40  }
41  bool dfs(int u)
42  {
43      for (auto& v : G[u])
44      {
45          if (!used[v] && dy[v] == dx[u] + 1)
46          {
47              used[v] = true;
48              if (My[v] != -1 && dy[v] == dis) continue;
49              if (My[v] == -1 || dfs(My[v]))
50              {
51                  My[v] = u, Mx[u] = v;
52                  return true;
53              }
54          }
55      }
56      return false;
57  }
58  int MaxMatch()
59  {
60      int res = 0;
61      memset(Mx, -1, sizeof(Mx)), memset(My, -1, sizeof(My));
62      while (bfs())
63      {
64          memset(used, false, sizeof(used));
65          for (int i = 0; i < uN; i++)
66              if (Mx[i] == -1 && dfs(i)) res++;
67      }
68      return res;
69  }
```

4.6.4 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         memset(used, 0, sizeof(used));
35         if (dfs(u)) res++;
36     }
37     return res;
38 }
```

4.6.5 Kuhn-Munkres

```
1  const int maxn = "Edit";
2  int n;
3  int cost[maxn][maxn];
4  int lx[maxn], ly[maxn], match[maxn], slack[maxn];
5  int prev[maxn];
6  bool vy[maxn];
7
8  void augment(int root)
9  {
10     fill(vy + 1, vy + n + 1, false);
11     fill(slack + 1, slack + n + 1, INF);
12     int py;
13     match[py = 0] = root;
14     do
15     {
```

```
16          vy[py] = true;
17          int x = match[py], yy;
18          int delta = INF;
19          for (int y = 1; y <= n; y++)
20          {
21              if (!vy[y])
22              {
23                  if (lx[x] + ly[y] - cost[x][y] < slack[y])
24                      slack[y] = lx[x] + ly[y] - cost[x][y], prev[y] = py;
25                  if (slack[y] < delta) delta = slack[y], yy = y;
26              }
27          }
28          for (int y = 0; y <= n; y++)
29          {
30              if (vy[y])
31                  lx[match[y]] -= delta, ly[y] += delta;
32              else
33                  slack[y] -= delta;
34          }
35          py = yy;
36      } while (match[py] != -1);
37      do
38      {
39          int pre = prev[py];
40          match[py] = match[pre], py = pre;
41      } while (py);
42  }
43  int KM()
44  {
45      for (int i = 1; i <= n; i++)
46      {
47          lx[i] = ly[i] = 0;
48          match[i] = -1;
49          for (int j = 1; j <= n; j++) lx[i] = max(lx[i], cost[i][j]);
50      }
51      int answer = 0;
52      for (int root = 1; root <= n; root++) augment(root);
53      for (int i = 1; i <= n; i++) answer += lx[i], answer += ly[i];
54      return answer;
55  }
```

4.7 Network Flow

```
1  struct Edge
2  {
3      int from, to, cap, flow;
4      Edge(int u, int v, int c, int f)
5          : from(u), to(v), cap(c), flow(f) {}
6  };
```

费用流

```
1  struct Edge
2  {
3      int from, to, cap, flow, cost;
4      Edge(int u, int v, int c, int f, int w)
5          : from(u), to(v), cap(c), flow(f), cost(w) {}
6  };
```

建模技巧

二分图带权最大独立集。给出一个二分图，每个结点上有一个正权值。要求选出一些点，使得这些点之间没有边相连，且权值和最大。

解：在二分图的基础上添加源点 S 和汇点 T，然后从 S 向所有 X 集合中的点连一条边，所有 Y 集合中的点向 T 连一条边，容量均为该点的权值。X 结点与 Y 结点之间的边的容量均为无穷大。这样，对于图中的任意一个割，将割中的边对应的结点删掉就是一个符合要求的解，权和为所有权减去割的容量。因此，只需要求出最小割，就能求出最大权和。

公平分配问题。把 m 个任务分配给 n 个处理器。其中每个任务有两个候选处理器，可以任选一个分配。要求所有处理器中，任务数最多的那个处理器所分配的任务数尽量少。不同任务的候选处理器集 $\{p_1, p_2\}$ 保证不同。

解：本题有一个比较明显的二分图模型，即 X 结点是任务，Y 结点是处理器。二分答案 x，然后构图，首先从源点 S 出发向所有的任务结点引一条边，容量等于 1，然后从每个任务结点出发引两条边，分别到达它所能分配到的两个处理器结点，容量为 1，最后从每个处理器结点出发引一条边到汇点 T，容量为 x，表示选择该处理器的任务不能超过 x。这样网络中的每个单位流量都是从 S 流到一个任务结点，再到处理器结点，最后到汇点 T。只有当网络中的总流量等于 m 时才意味着所有任务都选择了一个处理器。这样，我们通过 $O(\log m)$ 次最大流便算出了答案。

区间 k 覆盖问题。数轴上有一些带权值的左闭右开区间。选出权和尽量大的一些区间，使得任意一个数最多被 k 个区间覆盖。

解：本题可以用最小费用流解决，构图方法是把每个数作为一个结点，然后对于权值为 w 的区间 $[u, v)$ 加边 $u{\to}v$，容量为 1，费用为 $-w$。再对所有相邻的点加边 $i{\to}i+1$，容量为 k，费用为 0。最后，求最左点到最右点的最小费用最大流即可，其中每个流量对应一组互不相交的区间。如果数值范围太大，可以先进行离散化。

最大闭合子图。给定带权图 G（权值可正可负），求一个权和最大的点集，使得起点在该点集中的任意弧，终点也在该点集中。

解：新增附加源 s 和附加汇 t，从 s 向所有正权点引一条边，容量为权值；从所有负权点向汇点引一条边，容量为权值的相反数。求出最小割以后，$S - \{s\}$ 就是最大闭合子图。

最大密度子图。给出一个无向图，找一个点集，使得这些点之间的边数除以点数的值（称为子图的密度）最大。

解：如果两个端点都选了，就必然要选边，这就是一种推导。如果把每个点和每条边都看成新图中的结点，可以把问题转化为最大闭合子图。

4.7.1 EdmondKarp

```cpp
const int maxn = "Edit";
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.emplace_back(from, to, cap, 0);
        edges.emplace_back(to, from, 0, 0); //反向弧
        m = edges.size();
        G[from].push_back(m - 2);
        G[to].push_back(m - 1);
    }
```

```
22      int Maxflow(int s, int t)
23      {
24          int flow = 0;
25          for (;;)
26          {
27              memset(a, 0, sizeof(a));
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  };
```

4.7.2 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.emplace_back(from, to, cap, 0);
19          edges.emplace_back(to, from, 0, 0);
```

```
20          m = edges.size();
21          G[from].push_back(m - 2);
22          G[to].push_back(m - 1);
23      }
24      bool BFS()
25      {
26          memset(vis, 0, sizeof(vis));
27          memset(d, 0, sizeof(d));
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          { //从上次考虑的弧
55              Edge& e = edges[G[x][i]];
56              if (d[x] + 1 == d[e.to] && (f = DFS(e.to, min(a, e.cap - e.flow))) > 0)
57              {
58                  e.flow += f;
59                  edges[G[x][i] ^ 1].flow -= f;
60                  flow += f;
61                  a -= f;
62                  if (a == 0) break;
63              }
64          }
65          return flow;
66      }
67      int Maxflow(int s, int t)
68      {
69          this->s = s, this->t = t;
70          int flow = 0;
71          while (BFS())
72          {
73              memset(cur, 0, sizeof(cur));
74              flow += DFS(s, INF);
75          }
76          return flow;
77      }
78  };
```

4.7.3 ISAP

```
1   const int maxn = "Edit";
2   struct ISAP
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。edges[e]和edges[e^1]互为反向弧
6       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
7       bool vis[maxn];          //BFS使用
8       int d[maxn];             //起点到i的距离
9       int cur[maxn];           //当前弧下标
10      int p[maxn];             //可增广路上的一条弧
11      int num[maxn];           //距离标号计数
12      void init(int n)
13      {
14          this->n = n;
15          for (int i = 0; i < n; i++) G[i].clear();
16          edges.clear();
17      }
18      void AddEdge(int from, int to, int cap)
19      {
20          edges.emplace_back(from, to, cap, 0);
21          edges.emplace_back(to, from, 0, 0);
22          int m = edges.size();
23          G[from].push_back(m - 2);
24          G[to].push_back(m - 1);
25      }
26      int Augumemt()
27      {
28          int x = t, a = INF;
29          while (x != s)
30          {
31              Edge& e = edges[p[x]];
32              a = min(a, e.cap - e.flow);
33              x = edges[p[x]].from;
34          }
35          x = t;
36          while (x != s)
37          {
38              edges[p[x]].flow += a;
39              edges[p[x] ^ 1].flow -= a;
40              x = edges[p[x]].from;
41          }
42          return a;
43      }
44      void BFS()
45      {
46          memset(vis, 0, sizeof(vis));
47          memset(d, 0, sizeof(d));
48          queue<int> q;
49          q.push(t);
50          d[t] = 0;
51          vis[t] = 1;
52          while (!q.empty())
53          {
54              int x = q.front();
55              q.pop();
56              int len = G[x].size();
57              for (int i = 0; i < len; i++)
```

```
58                  {
59                      Edge& e = edges[G[x][i] ^ 1];
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          memset(num, 0, sizeof(num));
76          for (int i = 0; i < n; i++)
77              if (d[i] < INF) num[d[i]]++;
78          int x = s;
79          memset(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 };
```

4.7.4 MinCost MaxFlow

```
1   const int maxn = "Edit";
2   struct MCMF
3   {
4       int n, m;
5       vector<Edge> edges;
6       vector<int> G[maxn];
7       int inq[maxn]; //是否在队列中
8       int d[maxn];   //bellmanford
9       int p[maxn];   //上一条弧
10      int a[maxn];   //可改进量
11      void init(int n)
12      {
13          this->n = n;
14          for (int i = 0; i < n; i++) G[i].clear();
15          edges.clear();
16      }
17      void AddEdge(int from, int to, int cap, int cost)
18      {
19          edges.emplace_back(from, to, cap, 0, cost);
20          edges.emplace_back(to, from, 0, 0, -cost);
21          m = edges.size();
22          G[from].push_back(m - 2);
23          G[to].push_back(m - 1);
24      }
25      bool BellmanFord(int s, int t, int& flow, ll& cost)
26      {
27          for (int i = 0; i < n; i++) d[i] = INF;
28          memset(inq, 0, sizeof(inq));
29          d[s] = 0;
30          inq[s] = 1;
31          p[s] = 0;
32          a[s] = INF;
33          queue<int> q;
34          q.push(s);
35          while (!q.empty())
36          {
37              int u = q.front();
38              q.pop();
39              inq[u] = 0;
40              for (int i = 0; i < G[u].size(); i++)
41              {
42                  Edge& e = edges[G[u][i]];
43                  if (e.cap > e.flow && d[e.to] > d[u] + e.cost)
44                  {
45                      d[e.to] = d[u] + e.cost;
46                      p[e.to] = G[u][i];
47                      a[e.to] = min(a[u], e.cap - e.flow);
48                      if (!inq[e.to])
49                      {
50                          q.push(e.to);
51                          inq[e.to] = 1;
52                      }
53                  }
54              }
55          }
56          if (d[t] == INF) return false; // 当没有可增广的路时退出
57          flow += a[t];
```

```
58              cost += (ll)d[t] * (ll)a[t];
59              for (int u = t; u != s; u = edges[p[u]].from)
60              {
61                  edges[p[u]].flow += a[t];
62                  edges[p[u] ^ 1].flow -= a[t];
63              }
64              return true;
65          }
66          int MincostMaxflow(int s, int t, ll& cost)
67          {
68              int flow = 0;
69              cost = 0;
70              while (BellmanFord(s, t, flow, cost));
71              return flow;
72          }
73      };
```

4.7.5   Upper-Lower Bound

上下界网络流建图方法

记号说明

- $f(u,v)$ 表示 $u \to v$ 的实际流量
- $b(u,v)$ 表示 $u \to v$ 的流量下界
- $c(u,v)$ 表示 $u \to v$ 的流量上界

无源汇可行流

建图

- 新建附加源点 S 和 T
- 原图中的边 $u \to v$, 限制为 $[b,c]$, 建边 $u \to v$, 容量为 $c-b$
- 记 $d(i) = \sum b(u,i) - \sum b(i,v)$
- 若 $d(i) > 0$, 建边 $S \to i$, 流量为 $d(i)$
- 若 $d(i) < 0$, 建边 $i \to T$, 流量为 $-d(i)$

求解

- 跑 $S \to T$ 的最大流, 如果满流, 则原图存在可行流。
- 此时, 原图中每一条边的流量为新图中对应边的流量加上这条边的下界。

有源汇可行流

建图

- 在原图中建边 $t \to s$, 流量限制为 $[0, +\infty)$, 这样就改造成了无源汇的网络流图。
- 之后就可以像求解无源汇可行流一样建图了。

求解　同无源汇可行流

有源汇最大流

建图　同有源汇可行流

求解

- 先跑一遍 $S \to T$ 的最大流，求出可行流
- 记此时 $\sum f(s, i) = sum_1$
- 将 $t \to s$ 这条边拆掉，在新图上跑 $s \to t$ 的最大流
- 记此时 $\sum f(s, i) = sum_2$
- 最终答案即为 $sum_1 + sum_2$

有源汇最小流

建图　同无源汇可行流

求解

- 求 $S \to T$ 最大流
- 建边 $t \to s$，容量为 $+\infty$
- 再跑一遍 $S \to T$ 的最大流，答案即为 $f(t, s)$

有源汇的最大流和最小流也可以通过二分答案求得，
即二分 $t \to s$ 的下界（最大流）和上界（最小流）复杂度多了个 $O(\log n)$ 这里不再赘述。

蓝书上的做法

- 先用无源汇可行流建图的方法求出可行流，然后用传统 $s - t$ 增广路算法即可得到最大流。把 t 看成源点，s 看成汇点后求出的 $t - s$ 最大流就是最小流。
- 注意：原先每条弧 $u \to v$ 的反向弧容量为 0，而在有容量下界的情形中，反向弧的容量应该等于流量下界。

有源汇费用流

建图

- 新建附加源点 S 和 T
- 原图中的边 $u \to v$，限制为 $[b, c]$，费用为 $cost$，建边 $u \to v$，容量为 $c - b$，费用为 $cost$
- 记 $d(i) = \sum b(u, i) - \sum b(i, v)$
- 若 $d(i) > 0$，建边 $S \to i$，流量为 $d(i)$，**费用为** 0
- 若 $d(i) < 0$，建边 $i \to T$，流量为 $-d(i)$，**费用为** 0
- 建边 $t \to s$，流量为 $+\infty$，费用为 0。

求解

- 跑 $S \to T$ 的最小费用最大流
- 答案为求出的费用加上原图中边的下界乘以边的费用

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 {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 {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  double 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  double 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 dist(p, 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      memset(s, 0, sizeof(s));
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(p, n)) / 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  }
```

5.6 RuJia Liu's

5.6.1 Point

```cpp
1  struct Point
2  {
3      double x, y;
4      Point(double x = 0, double y = 0) : x(x), y(y) {}
5  };
6
7  typedef Point Vector;
8
9  //向量+向量=向量, 点+向量=点
10 Vector operator+(Vector A, Vector B) { return Vector(A.x + B.x, A.y + B.y); }
11 //点-点=向量
12 Vector operator-(Point A, Point B) { return Vector(A.x - B.x, A.y - B.y); }
13 //向量*数=向量
14 Vector operator*(Vector A, double p) { return Vector(A.x * p, A.y * p); }
15 //向量/数=向量
16 Vector operator/(Vector A, double p) { return Vector(A.x / p, A.y / p); }
17
18 bool operator<(const Point& a, const Point& b)
19 {
20     return a.x < b.x || (a.x == b.x && a.y < b.y);
21 }
22
23 const double eps = 1e-10;
24 double dcmp(double x)
25 {
26     if (fabs(x) < eps)
27         return 0;
28     else
29         return x < 0 ? -1 : 1;
30 }
31
32 bool operator==(const Point& a, const Point& b)
33 {
34     return dcmp(a.x - b.x) == 0 && dcmp(a.y - b.y) == 0;
35 }
36
37 /*
38  * 基本运算:
39  * 点积
40  * 叉积
41  * 向量旋转
42  */
43 double Dot(Vector A, Vector B) { return A.x * B.x + A.y * B.y; }
44 double Length(Vector A) { return sqrt(Dot(A, A)); }
45 double Angle(Vector A, Vector B) { return acos(Dot(A, B) / Length(A) / Length(B)); }
46
47 double Cross(Vector A, Vector B) { return A.x * B.y - A.y * B.x; }
48 double Area2(Point A, Point B, Point C) { return Cross(B - A, C - A); }
49
50 //rad是弧度
51 Vector Rotate(Vector A, double rad)
52 {
53     return Vector(A.x * cos(rad) - A.y * sin(rad),
54                   A.x * sin(rad) + A.y * cos(rad));
55 }
```

```cpp
56
57   //调用前请确保A不是零向量
58   Vector Normal(Vector A)
59   {
60       double L = Length(A);
61       return Vector(-A.y / L, A.x / L);
62   }
63
64   /*
65    *  点和直线:
66    *  两直线交点
67    *  点到直线的距离
68    *  点到线段的距离
69    *  点在直线上的投影
70    *  线段相交判定
71    *  点在线段上判定
72    */
73
74   //调用前保证两条直线P+tv和Q+tw有唯一交点。当且仅当Cross(v, w)非0
75   Point GetLineIntersection(Point P, Vector v, Point Q, Vector w)
76   {
77       Vector u = P - Q;
78       double t = Cross(w, u) / Cross(v, w);
79       return P + v * t;
80   }
81
82   double DistanceToLine(Point P, Point A, Point B)
83   {
84       Vector v1 = B - A, v2 = P - A;
85       return fabs(Cross(v1, v2)) / Length(v1); //如果不取绝对值, 得到的是有向距离
86   }
87
88   double DistanceToSegment(Point P, Point A, Point B)
89   {
90       if (A == B) return Length(P - A);
91       Vector v1 = B - A, v2 = P - A, v3 = P - B;
92       if (dcmp(Dot(v1, v2)) < 0) return Length(v2);
93       if (dcmp(Dot(v1, v3)) > 0) return Length(v3);
94       return fabs(Cross(v1, v2)) / Length(v1);
95   }
96
97   Point GetLineProjection(Point P, Point A, Point B)
98   {
99       Vector v = B - A;
100      return A + v * (Dot(v, P - A) / Dot(v, v));
101  }
102
103  bool SegmentProperIntersection(Point a1, Point a2, Point b1, Point b2)
104  {
105      double c1 = Cross(a2 - a1, b1 - a1), c2 = Cross(a2 - a1, b2 - b1),
106             c3 = Cross(b2 - b1, a1 - b1), c4 = Cross(b2 - b1, a2 - b1);
107      return dcmp(c1) * dcmp(c2) < 0 && dcmp(c3) * dcmp(c4) < 0;
108  }
109
110  bool OnSegment(Point p, Point a1, Point a2)
111  {
112      return dcmp(Cross(a1 - p, a2 - p)) == 0 && dcmp(Dot(a1 - p, a2 - p)) < 0;
113  }
```

5.6.2 Circle

```
1   struct Line
2   {
3       Point p;      //直线上任意一点
4       Vector v;     //方向向量。它的左边就是对应的半平面
5       double ang;   //极角。即从x正半轴旋转到向量v所需要的角（弧度）
6       Line() {}
7       Line(Point p, Vector v) : p(p), v(v) { ang = atan2(v.y, v.x); }
8       bool operator<(const Line& L) const //  排序用的比较运算符
9       {
10          return ang < L.ang;
11      }
12      Point point(double t) { return p + v * t; }
13  };
14
15  struct Circle
16  {
17      Point c;
18      double r;
19      Circle(Point c, double r) : c(c), r(r) {}
20      Point point(double a) { return c.x + cos(a) * r, c.y + sin(a) * r; }
21  };
22
23  int getLineCircleIntersection(Line L, Circle C, double& t1, double& t2, vector<Point>&
        sol)
24  {
25      double a = L.v.x, b = L.p.x - C.c.x, c = L.v.y, d = L.p.y - C.c.y;
26      double e = a * a + c * c, f = 2 * (a * b + c * d), g = b * b + d * d - C.r * C.r;
27      double delta = f * f - 4 * e * g; //判别式
28      if (dcmp(delta) < 0) return 0;      //相离
29      if (dcmp(delta) == 0)               //相切
30      {
31          t1 = t2 = -f / (2 * e);
32          sol.push_back(L.point(t1));
33          return 1;
34      }
35      //相交
36      t1 = (-f - sqrt(delta)) / (2 * e);
37      t2 = (-f + sqrt(delta)) / (2 * e);
38      sol.push_back(t1);
39      sol.push_back(t2);
40      return 2;
41  }
42
43  double angle(Vector v) { return atan2(v.y, v.x); }
44
45  int getCircleCircleIntersection(Circle C1, Circle C2, vector<Point>& sol)
46  {
47      double d = Length(C1.c - C2.c);
48      if (dcmp(d) == 0)
49      {
50          if (dcmp(C1.r - C2.r) == 0) return -1; //两圆重合
51          return 0;
52      }
53      if (dcmp(C1.r + C2.r - d) < 0) return 0;        //内含
54      if (dcmp(fabs(C1.r - C2.r) - d) > 0) return 0; //外离
55
56      double a = angle(C2.c - C1.c); //向量C1C2的极角
```

```
57         double da = acos((C1.r * C1.r + d * d - C2.r * C2.r) / (2 * C1.r * d));
58         //C1C2到C1P1的角
59         Point p1 = C1.point(a - da), p2 = C1.point(a + da);
60
61         sol.push_back(p1);
62         if (p1 == p2) return 1;
63         sol.push_back(p2);
64         return 2;
65     }
66
67     //过点p到圆C的切线, v[i]是第i条切线的向量, 返回切线条数
68     int getTangents(Point p, Circle C, Vector* v)
69     {
70         Vector u = C.c - p;
71         double dist = Length(u);
72         if (dist < C.r)
73             return 0;
74         else if (dcmp(dist - C.r) == 0)
75         { //p在圆上, 只有一条切线
76             v[0] = Rotate(u, M_PI / 2);
77             return 1;
78         }
79         else
80         {
81             double ang = asin(C.r / dist);
82             v[0] = Rotate(u, -ang);
83             v[1] = Rotate(u, +ang);
84             return 2;
85         }
86     }
87
88     //两圆的公切线
89     //返回切线的条数。-1表示无穷条切线。
90     //a[i]和b[i]分别是第i条切线在圆A和圆B上的切点
91     int getTangents(Circle A, Circle B, Point* a, Point* b)
92     {
93         int cnt = 0;
94         if (A.r < B.r)
95         {
96             swap(A, B);
97             swap(a, b);
98         }
99         int d2 = (A.c.x - B.c.x) * (A.c.x - B.c.x) + (A.c.y - B.c.y) * (A.c.y - B.c.y);
100        int rdiff = A.r - B.r;
101        int rsum = A.r + B.r;
102        if (d2 < rdiff * rdiff) return 0; //内含
103        double base = atan2(B.c.y - A.c.y, B.c.x - A.c.x);
104        if (d2 == 0 && A.r == B.r) return -1; //无限多条切线
105        if (d2 == rdiff * rdiff)
106        { //内切, 一条切线
107            a[cnt] = A.point(base);
108            b[cnt] = B.point(base);
109            cnt++;
110            return 1;
111        }
112        //有外共切线
113        double ang = acos(A.r - B.r) / sqrt(d2);
114        a[cnt] = A.point(base + ang);
115        b[cnt] = B.point(base + ang);
```

```
116        cnt++;
117        a[cnt] = A.point(base + ang);
118        b[cnt] = B.point(base - ang);
119        cnt++;
120        if (d2 == rsum * rsum)
121        {
122            a[cnt] = A.point(base);
123            b[cnt] = B.point(M_PI + base);
124            cnt++;
125        }
126        else if (d2 > rsum * rsum)
127        {
128            double ang = acos((A.r + B.r) / sqrt(d2));
129            a[cnt] = A.point(base + ang);
130            b[cnt] = B.point(M_PI + base + ang);
131            cnt++;
132            a[cnt] = A.point(base - ang);
133            b[cnt] = B.point(M_PI + base - ang);
134            cnt++;
135        }
136        return cnt;
137    }
138
139    //三角形外接圆（三点保证不共线）
140    Circle CircumscribedCircle(Point p1, Point p2, Point p3)
141    {
142        double Bx = p2.x - p1.x, By = p2.y - p1.y;
143        double Cx = p3.x - p1.x, Cy = p3.y - p1.y;
144        double D = 2 * (Bx * Cy - By * Cx);
145        double cx = (Cy * (Bx * Bx + By * By) - By * (Cx * Cx + Cy * Cy)) / D + p1.x;
146        double cy = (Bx * (Cx * Cx + Cy * Cy) - Cx * (Bx * Bx + By * By)) / D + p1.y;
147        Point p = Point(cx, cy);
148        return Circle(p, Length(p1 - p));
149    }
150
151    //三角形内切圆
152    Circle InscribedCircle(Point p1, Point p2, Point p3)
153    {
154        double a = Length(p2 - p3);
155        double b = Length(p3 - p1);
156        double c = Length(p1 - p2);
157        Point p = (p1 * a + p2 * b + p3 * c) / (a + b + c);
158        return Circle(p, DistanceToLine(p, p1, p2));
159    }
```

5.6.3 Polygon

```
1    typedef vector<Point> Polygon;
2    //多边形的有向面积
3    double PolygonArea(Polygon po)
4    {
5        int n = po.size();
6        double area = 0.0;
7        for (int i = 1; i < n - 1; i++)
8            area += Cross(po[i] - po[0], po[i + 1] - po[0]);
9        return area / 2;
10   }
11
```

```
12   //点在多边形内判定
13   int isPointInPolygon(Point p, Polygon poly)
14   {
15       int wn = 0; //绕数
16       int n = poly.size();
17       for (int i = 0; i < n; i++)
18       {
19           if (OnSegment(p, poly[i], poly[(i + 1) % n])) return -1; //边界上
20           int k = dcmp(Cross(poly[(i + 1) % n] - poly[i], p - poly[i]));
21           int d1 = dcmp(poly[i].y - p.y);
22           int d2 = dcmp(poly[(i + 1) % n].y - p.y);
23           if (k > 0 && d1 <= 0 && d2 > 0) wn++;
24           if (k < 0 && d2 <= 0 && d1 > 0) wn--;
25       }
26       if (wn != 0) return 1; //内部
27       return 0;              //外部
28   }
29
30   //凸包(Andrew算法)
31   //如果不希望在凸包的边上有输入点，把两个 <= 改成 <
32   //如果不介意点集被修改，可以改成传递引用
33   Polygon ConvexHull(vector<Point> p)
34   {
35       sort(p.begin(), p.end());
36       p.erase(unique(p.begin(), p.end()), p.end());
37       int n = p.size(), m = 0;
38       Polygon res(n + 1);
39       for (int i = 0; i < n; i++)
40       {
41           while (m > 1 && Cross(res[m - 1] - res[m - 2], p[i] - res[m - 2]) <= 0) m--;
42           res[m++] = p[i];
43       }
44       int k = m;
45       for (int i = n - 2; i >= 0; i--)
46       {
47           while (m > k && Cross(res[m - 1] - res[m - 2], p[i] - res[m - 2]) <= 0) m--;
48           res[m++] = p[i];
49       }
50       m -= n > 1;
51       res.resize(m);
52       return res;
53   }
54
55   //半平面交
56   vector<Point> HalfplaneIntersection(vector<Line>& L)
57   {
58       int n = L.size();
59       sort(L.begin(), L.end()); // 按极角排序
60
61       int first, last;    // 双端队列的第一个元素和最后一个元素的下标
62       vector<Point> p(n); // p[i]为q[i]和q[i+1]的交点
63       vector<Line> q(n);  // 双端队列
64       vector<Point> ans;  // 结果
65
66       q[first = last = 0] = L[0]; // 双端队列初始化为只有一个半平面L[0]
67       for (int i = 1; i < n; i++)
68       {
69           while (first < last && !OnLeft(L[i], p[last - 1])) last--;
70           while (first < last && !OnLeft(L[i], p[first])) first++;
```

```
71          q[++last] = L[i];
72          if (fabs(Cross(q[last].v, q[last - 1].v)) < eps)
73          { // 两向量平行且同向，取内侧的一个
74              last--;
75              if (OnLeft(q[last], L[i].p)) q[last] = L[i];
76          }
77          if (first < last) p[last - 1] = GetLineIntersection(q[last - 1], q[last]);
78      }
79      while (first < last && !OnLeft(q[first], p[last - 1])) last--; // 删除无用平面
80      if (last - first <= 1) return vector<Point>();                  // 空集
81      p[last] = GetLineIntersection(q[last], q[first]);               // 计算首尾两个半平面的
    交点
82
83      return vector<Point>(q.begin() + first, q.begin() + last + 1);
84  }
```

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     memset(dp, 0x3f, sizeof(dp));
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

```cpp
// 序列下标从1开始
int LCIS(int a[], int b[], int n, int m)
{
    memset(dp, 0, sizeof(dp));
    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);
}
```

6.2 Digit Statistics

```cpp
int a[20];
ll dp[20][state];
ll dfs(int pos, /*state变量*/, bool lead /*前导零*/, bool limit /*数位上界变量*/)
{
    //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
    if (pos == -1) return 1;
    /*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，
    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。*/
    if (!limit && !lead && dp[pos][state] != -1) return dp[pos][state];
    /*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*/
    int up = limit ? a[pos] : 9; //根据limit判断枚举的上界up
    ll ans = 0;
    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
    {
        if () ...
        else if () ...
        ans += dfs(pos - 1, /*状态转移*/, lead && i == 0, limit && i == a[pos])
        //最后两个变量传参都是这样写的
        /*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
        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*/
    }
    //计算完，记录状态
    if (!limit && !lead) dp[pos][state] = ans;
    /*这里对应上面的记忆化，在一定条件下时记录，保证一致性，
    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*/
    return ans;
}
ll solve(ll x)
{
    int pos = 0;
    do //把数位都分解出来
        a[pos++] = x % 10;
    while (x /= 10);
    return dfs(pos - 1 /*从最高位开始枚举*/, /*一系列状态 */, true, true);
    //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
}
```

6.3 Slope Optimization

问题 设 $f(i) = \min(y[k] - s[i] \times x[k]), k \in [1, i-1]$, 现在要求出所有 $f(i), i \in [1, n]$

考虑两个决策 j 和 k, 如果 j 比 k 优, 则

$$y[j] - s[i] \times x[j] < y[k] - s[i] \times x[k]$$

化简得:

$$\frac{y_j - y_k}{x_j - x_k} < s_i$$

不等式左边是个斜率, 我们把它设为 slope(j, k)

我们可以维护一个单调递增的队列, 为什么呢?

因为如果 slope$(q[i-1], q[i]) >$ slope$(q[i], q[i+1])$, 那么当前者成立时, 后者必定成立。即 $q[i]$ 决策优于 $q[i-1]$ 决策时, $q[i+1]$ 必然优于 $q[i]$, 因此 $q[i]$ 就没有存在的必要了。所以我们要维护递增的队列。

那么每次的决策点 i, 都要满足

$$\begin{cases} \text{slope}(q[i-1], q[i]) < s[i] \\ \text{slope}(q[i], q[i+1]) \geq s[i] \end{cases}$$

一般情况去二分这个 i 即可。

如果 $s[i]$ 是单调不降的, 那么对于决策 j 和 $k(j < k)$ 来说, 如果决策 k 优于决策 j, 那么对于 $i \in [k+1, n]$, 都存在决策 k 优于决策 j, 因此决策 j 就可以舍弃了。这样的话我们可以用单调队列进行优化, 可以少个 log。

单调队列滑动窗口最大值

```cpp
// k为滑动窗口的大小
deque<int> q;
for (int i = 0, j = 0; i + k <= d; i++)
{
    while (j < i + k)
    {
        while (!q.empty() && a[q.back()] < a[j]) q.pop_back();
        q.push_back(j++);
    }
    while (q.front() < i) q.pop_front();
    // a[q.front()]为当前滑动窗口的最大值
}
```

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  fgets(buf, BUFSIZE, stdin);
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 Algorithm

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      memset(ans, 0, sizeof(ans));
5      sort(node, node + m, [](query a, query b) {
6          return a.l / unit < b.l / unit
7                  || a.l / unit == b.l / unit && a.r < b.r;
8      });
9      int L = 1, R = 0;
10     for (int i = 0; i < m; i++)
11     {
12         while (node[i].L < L) add(a[--L]);
13         while (node[i].L > L) del(a[L++]);
14         while (node[i].R < R) del(a[R--]);
15         while (node[i].R > R) add(a[++R]);
16         ans[node[i].id] = tmp;
17     }
18  }
```

7.4 BigNum

7.4.1 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, memset(w, 0);
        while (x) 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; i--) printf("%d", w[i]);
        puts("");
    }
};
```

7.4.2 Complete High-precision

```
1  import java.math.BigInteger;
```

7.5 Misc

7.5.1 Standard Template Library

```
1  template <class InputIterator, class OutputIterator>
2    OutputIterator copy (InputIterator first, InputIterator last, OutputIterator result);
3
4  template <class InputIterator1, class InputIterator2,
5           class OutputIterator, class Compare>
6    OutputIterator merge (InputIterator1 first1, InputIterator1 last1,
7                          InputIterator2 first2, InputIterator2 last2,
8                          OutputIterator result, Compare comp);
9
10 template <class InputIterator, class Function>
11    Function for_each (InputIterator first, InputIterator last, Function fn);
12
13 template <class InputIterator, class OutputIterator, class UnaryOperation>
14   OutputIterator transform (InputIterator first1, InputIterator last1,
15                             OutputIterator result, UnaryOperation op);
16
17 template< class ForwardIterator, class T >
18 void iota( ForwardIterator first, ForwardIterator last, T value );
```

7.5.2 Policy-Based Data Structures

红黑树

声明/头文件
```
1  #include <ext/pb_ds/tree_policy.hpp>
2  #include <ext/pb_ds/assoc_container.hpp>
3  using namespace __gnu_pbds;
4  typedef tree<pt, null_type, less<pt>, rb_tree_tag, tree_order_statistics_node_update>
       rbtree;
```

使用方法
```
1  pt                                      // 关键字类型
2  null_type                               // 无映射(低版本g++为null_mapped_type)
3  less<int>                               // 从小到大排序
4  rb_tree_tag                             // 红黑树 (splay_tree_tag)
5  tree_order_statistics_node_update       // 结点更新
6  T.insert(val);                          // 插入
7  T.erase(iterator);                      // 删除
8  T.order_of_key();                       // 查找有多少数比它小
9  T.find_by_order(k);                     // 有k个数比它小的数是多少
10 a.join(b);                              // b并入a 前提是两棵树的key的取值范围不相交
11 a.split(v, b);                          // key小于等于v的元素属于a, 其余的属于b
12 T.lower_bound(x);                       // >=x的min的迭代器
13 T.upper_bound((x);                      // >x的min的迭代器
```

7.5.3 Subset Enumeration

枚举真子集

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

枚举大小为 k 的子集

```
1  void subset(int k, int n)
2  {
3      int t = (1 << k) - 1;
4      while (t < (1 << n))
5      {
6          // do something
7          int x = t & -t, y = t + x;
8          t = ((t & ~y) / x >> 1) | y;
9      }
10  }
```

7.5.4 Date Magic

```
1  string dayOfWeek[] = {"Mo", "Tu", "We", "Th", "Fr", "Sa", "Su"};
2
3  // converts Gregorian date to integer (Julian day number)
4  int DateToInt(int m, int d, int y)
5  {
6      return 1461 * (y + 4800 + (m - 14) / 12) / 4
7              + 367 * (m - 2 - (m - 14) / 12 * 12) / 12
8              - 3 * ((y + 4900 + (m - 14) / 12) / 100) / 4
9              + d - 32075;
10  }
11
12  // converts integer (Julian day number) to Gregorian date: month/day/year
13  void IntToDate(int jd, int& m, int& d, int& y)
14  {
15      int x, n, i, j;
16      x = jd + 68569;
17      n = 4 * x / 146097;
18      x -= (146097 * n + 3) / 4;
19      i = (4000 * (x + 1)) / 1461001;
20      x -= 1461 * i / 4 - 31;
21      j = 80 * x / 2447;
22      d = x - 2447 * j / 80;
23      x = j / 11;
24      m = j + 2 - 12 * x;
25      y = 100 * (n - 49) + i + x;
26  }
27
28  // converts integer (Julian day number) to day of week
29  string IntToDay(int jd) { return dayOfWeek[jd % 7]; }
```

7.6 Configuration

7.6.1 VSCode

launch.json

```
1  {
2      "version": "0.2.0",
3      "configurations": [
```

```
 4          {
 5              "name": "(gdb) Launch",
 6              "type": "cppdbg",
 7              "request": "launch",
 8              "program": "${workspaceRoot}/a.out",
 9              "args": [],
10              "stopAtEntry": false,
11              "cwd": "${fileDirname}",
12              "environment": [],
13              "externalConsole": true,
14              "MIMode": "gdb",
15              "setupCommands": [
16                  {
17                      "description": "Enable pretty-printing for gdb",
18                      "text": "-enable-pretty-printing",
19                      "ignoreFailures": true
20                  }
21              ],
22              "preLaunchTask": "build"
23          }
24      ]
25  }
```

task.json

```
 1  {
 2      // See https://go.microsoft.com/fwlink/?LinkId=733558
 3      // for the documentation about the tasks.json format
 4      "version": "2.0.0",
 5      "tasks": [
 6          {
 7              "label": "build",
 8              "type": "shell",
 9              "command": "g++",
10              "args": [
11                  "-g",
12                  "-std=c++17",
13                  "${file}"
14              ],
15              "group": {
16                  "kind": "build",
17                  "isDefault": true
18              },
19              "problemMatcher": {
20                  "owner": "cpp",
21                  "fileLocation": "absolute",
22                  "pattern": {
23                      "regexp": "^(.*):(\\d+):(\\d+):\\s+(warning|error):\\s+(.*)$",
24                      "file": 1,
25                      "line": 2,
26                      "column": 3,
27                      "severity": 4,
28                      "message": 5
29                  }
30              }
31          }
32      ]
33  }
```

7.6.2 Vim

```
1  syntax on
2  set cindent
3  set nu
4  set tabstop=4
5  set shiftwidth=4
6  set background=dark
7  set mouse=a
8
9  map<C-A> ggvG"+y
10 map<F5> :call Run()<CR>
11
12 func! Run()
13     exec "w"
14     exec "!g++ -std=c++11 -O2 % -o %<"
15     exec "!time ./%<"
16 endfunc
17
18 autocmd BufNewFile *.cpp 0r ~/include.cpp
19 autocmd BufNewFile *.cpp normal G
20
21 inoremap ( ()<Esc>i
22 inoremap [ []<Esc>i
23 inoremap { {<CR>}<Esc>O
24 inoremap ' ''<Esc>i
25 inoremap " ""<Esc>i
26
27 inoremap ) <c-r>=ClosePair(')')<CR>
28 inoremap ] <c-r>=ClosePair(']')<CR>
29
30 func ClosePair(char)
31     if getline('.')[col('.')-1]==a:char
32         return "\<Right>"
33     else
34         return a:char
35     endif
36 endfunc
```