

Algorithm Library

stlvdv

2021 年 10 月 21 日

目录

1	多项式	1
1.1	FFT - tourist	1
1.2	形式幂级数	6
2	数论	6
2.1	简单的防爆模板	6
2.2	筛法	7
2.2.1	线性素数筛	7
2.2.2	线性欧拉函数筛	7
2.2.3	线性约数个数函数筛	7
2.2.4	线性素因子个数函数筛	8
2.2.5	线性约数和函数筛	9
2.2.6	线性莫比乌斯函数筛	9
2.3	扩展欧几里得	10
2.3.1	线性同余方程最小非负整数解	10
2.4	欧拉定理	11
2.5	欧拉函数	11
2.6	中国剩余定理	11
2.6.1	CRT	11
2.6.2	EXCRT	11
2.7	BSGS	12
2.8	迪利克雷卷积	12
2.9	杜教筛	12
3	线性代数	12
3.1	高斯-约旦消元法	12
3.2	高斯消元法-bitset	13
3.3	线性基	14
3.4	矩阵树定理	17
3.5	LGV 引理	18
4	组合数学	18
4.1	组合数预处理	18
4.2	卢卡斯定理	19
4.3	小球盒子模型	19
4.4	斯特林数	19
4.4.1	第一类斯特林数	19
4.4.2	第二类斯特林数	20
5	博弈论	20

6	图论	20
6.1	并查集	20
6.2	最小树形图	21
6.3	最近公共祖先	23
6.4	强连通分量	24
6.5	最大流	26
6.6	最小费用最大流	30
6.7	全局最小割	36
6.8	二分图最大权匹配	36
6.9	一般图最大匹配	37
6.10	2-sat	37
6.11	最大团	37
7	数据结构	39
7.1	树状数组	39
8	字符串	39
8.1	KMP	39
8.2	Z-Function	39
8.3	Manacher	39
8.4	Trie	40
8.5	01-Trie	41
9	计算几何	42
10	杂项	42
10.1	蔡勒公式	42

1 多项式

1.1 FFT - tourist

```

1  /* copy from tourist */
2  namespace FFT {
3      typedef double dbl;
4
5      struct num {
6          dbl x, y;
7          num() { x = y = 0; }
8          num(dbl x, dbl y) : x(x), y(y) { }
9      };
10
11     inline num operator+(num a, num b) { return num(a.x + b.x, a.y + b.y); }
12     inline num operator-(num a, num b) { return num(a.x - b.x, a.y - b.y); }
13     inline num operator*(num a, num b) { return num(a.x * b.x - a.y * b.y, a
        .x * b.y + a.y * b.x); }
14     inline num conj(num a) { return num(a.x, -a.y); }
15
16     int base = 1;
17     vector<num> roots = { {0, 0}, {1, 0} };
18     vector<int> rev = { 0, 1 };
19
20     const dbl PI = acos(-1.0);
21
22     void ensure_base(int nbase) {
23         if (nbase <= base) {
24             return;
25         }
26         rev.resize(1 << nbase);
27         for (int i = 0; i < (1 << nbase); i++) {
28             rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (nbase - 1));
29         }
30         roots.resize(1 << nbase);
31         while (base < nbase) {
32             dbl angle = 2 * PI / (1 << (base + 1));
33             for (int i = 1 << (base - 1); i < (1 << base); i++) {
34                 roots[i << 1] = roots[i];
35                 dbl angle_i = angle * (2 * i + 1 - (1 << base));
36                 roots[(i << 1) + 1] = num(cos(angle_i), sin(angle_i));
37             }
38             base++;
39         }

```

```

40     }
41
42     void fft(vector<num>& a, int n = -1) {
43         if (n == -1) {
44             n = a.size();
45         }
46         assert((n & (n - 1)) == 0);
47         int zeros = __builtin_ctz(n);
48         ensure_base(zeros);
49         int shift = base - zeros;
50         for (int i = 0; i < n; i++) {
51             if (i < (rev[i] >> shift)) {
52                 swap(a[i], a[rev[i] >> shift]);
53             }
54         }
55         for (int k = 1; k < n; k <= 1) {
56             for (int i = 0; i < n; i += 2 * k) {
57                 for (int j = 0; j < k; j++) {
58                     num z = a[i + j + k] * roots[j + k];
59                     a[i + j + k] = a[i + j] - z;
60                     a[i + j] = a[i + j] + z;
61                 }
62             }
63         }
64     }
65
66     vector<num> fa, fb;
67
68     vector<long long> multiply(vector<int>& a, vector<int>& b) {
69         int need = a.size() + b.size() - 1;
70         int nbase = 1;
71         while ((1 << nbase) < need) nbase++;
72         ensure_base(nbase);
73         int sz = 1 << nbase;
74         if (sz > (int)fa.size()) {
75             fa.resize(sz);
76         }
77         for (int i = 0; i < sz; i++) {
78             int x = (i < (int)a.size() ? a[i] : 0);
79             int y = (i < (int)b.size() ? b[i] : 0);
80             fa[i] = num(x, y);
81         }
82         fft(fa, sz);

```

```

83     num r(0, -0.25 / (sz >> 1));
84     for (int i = 0; i <= (sz >> 1); i++) {
85         int j = (sz - i) & (sz - 1);
86         num z = (fa[j] * fa[j] - conj(fa[i] * fa[i])) * r;
87         if (i != j) {
88             fa[j] = (fa[i] * fa[i] - conj(fa[j] * fa[j])) * r;
89         }
90         fa[i] = z;
91     }
92     for (int i = 0; i < (sz >> 1); i++) {
93         num A0 = (fa[i] + fa[i + (sz >> 1)]) * num(0.5, 0);
94         num A1 = (fa[i] - fa[i + (sz >> 1)]) * num(0.5, 0) * roots[(sz
95             >> 1) + i];
96         fa[i] = A0 + A1 * num(0, 1);
97     }
98     fft(fa, sz >> 1);
99     vector<long long> res(need);
100    for (int i = 0; i < need; i++) {
101        if (i % 2 == 0) {
102            res[i] = fa[i >> 1].x + 0.5;
103        } else {
104            res[i] = fa[i >> 1].y + 0.5;
105        }
106    }
107    return res;
108 }
109
110 vector<long long> square(const vector<int>& a) {
111     int need = a.size() + a.size() - 1;
112     int nbase = 1;
113     while ((1 << nbase) < need) nbase++;
114     ensure_base(nbase);
115     int sz = 1 << nbase;
116     if ((sz >> 1) > (int)a.size()) {
117         fa.resize(sz >> 1);
118     }
119     for (int i = 0; i < (sz >> 1); i++) {
120         int x = (2 * i < (int)a.size() ? a[2 * i] : 0);
121         int y = (2 * i + 1 < (int)a.size() ? a[2 * i + 1] : 0);
122         fa[i] = num(x, y);
123     }
124     fft(fa, sz >> 1);
125     num r(1.0 / (sz >> 1), 0.0);

```

```

125     for (int i = 0; i <= (sz >> 2); i++) {
126         int j = ((sz >> 1) - i) & ((sz >> 1) - 1);
127         num fe = (fa[i] + conj(fa[j])) * num(0.5, 0);
128         num fo = (fa[i] - conj(fa[j])) * num(0, -0.5);
129         num aux = fe * fe + fo * fo * roots[(sz >> 1) + i] * roots[(sz
            >> 1) + i];
130         num tmp = fe * fo;
131         fa[i] = r * (conj(aux) + num(0, 2) * conj(tmp));
132         fa[j] = r * (aux + num(0, 2) * tmp);
133     }
134     fft(fa, sz >> 1);
135     vector<long long> res(need);
136     for (int i = 0; i < need; i++) {
137         if (i % 2 == 0) {
138             res[i] = fa[i >> 1].x + 0.5;
139         } else {
140             res[i] = fa[i >> 1].y + 0.5;
141         }
142     }
143     return res;
144 }
145
146 vector<int> multiply_mod(vector<int>& a, vector<int>& b, int m, int eq =
    0) {
147     int need = a.size() + b.size() - 1;
148     int nbase = 0;
149     while ((1 << nbase) < need) nbase++;
150     ensure_base(nbase);
151     int sz = 1 << nbase;
152     if (sz > (int)fa.size()) {
153         fa.resize(sz);
154     }
155     for (int i = 0; i < (int)a.size(); i++) {
156         int x = (a[i] % m + m) % m;
157         fa[i] = num(x & ((1 << 15) - 1), x >> 15);
158     }
159     fill(fa.begin() + a.size(), fa.begin() + sz, num{ 0, 0 });
160     fft(fa, sz);
161     if (sz > (int)fb.size()) {
162         fb.resize(sz);
163     }
164     if (eq) {
165         copy(fa.begin(), fa.begin() + sz, fb.begin());

```

```

166     } else {
167         for (int i = 0; i < (int)b.size(); i++) {
168             int x = (b[i] % m + m) % m;
169             fb[i] = num(x & ((1 << 15) - 1), x >> 15);
170         }
171         fill(fb.begin() + b.size(), fb.begin() + sz, num{ 0, 0 });
172         fft(fb, sz);
173     }
174     dbl ratio = 0.25 / sz;
175     num r2(0, -1);
176     num r3(ratio, 0);
177     num r4(0, -ratio);
178     num r5(0, 1);
179     for (int i = 0; i <= (sz >> 1); i++) {
180         int j = (sz - i) & (sz - 1);
181         num a1 = (fa[i] + conj(fa[j]));
182         num a2 = (fa[i] - conj(fa[j])) * r2;
183         num b1 = (fb[i] + conj(fb[j])) * r3;
184         num b2 = (fb[i] - conj(fb[j])) * r4;
185         if (i != j) {
186             num c1 = (fa[j] + conj(fa[i]));
187             num c2 = (fa[j] - conj(fa[i])) * r2;
188             num d1 = (fb[j] + conj(fb[i])) * r3;
189             num d2 = (fb[j] - conj(fb[i])) * r4;
190             fa[i] = c1 * d1 + c2 * d2 * r5;
191             fb[i] = c1 * d2 + c2 * d1;
192         }
193         fa[j] = a1 * b1 + a2 * b2 * r5;
194         fb[j] = a1 * b2 + a2 * b1;
195     }
196     fft(fa, sz);
197     fft(fb, sz);
198     vector<int> res(need);
199     for (int i = 0; i < need; i++) {
200         long long aa = fa[i].x + 0.5;
201         long long bb = fb[i].x + 0.5;
202         long long cc = fa[i].y + 0.5;
203         res[i] = (aa + ((bb % m) << 15) + ((cc % m) << 30)) % m;
204     }
205     return res;
206 }
207
208 vector<int> square_mod(vector<int>& a, int m) {

```



```

209     return multiply_mod(a, a, m, 1);
210 }
211 };

```

1.2 形式幂级数

2 数论

2.1 简单的防爆模板

```

1 namespace SimpleMod {
2     constexpr int MOD = (int)1e9 + 7;
3     inline int norm(long long a) { return (a % MOD + MOD) % MOD; }
4     inline int add(int a, int b) { return a + b >= MOD ? a + b - MOD : a + b
5         ; }
6     inline int sub(int a, int b) { return a - b < 0 ? a - b + MOD : a - b; }
7     inline int mul(int a, int b) { return (int)((long long)a * b % MOD); }
8     inline int powmod(int a, long long b) {
9         int res = 1;
10        while (b > 0) {
11            if (b & 1) res = mul(res, a);
12            a = mul(a, a);
13            b >>= 1;
14        }
15        return res;
16    }
17    inline int inv(int a) {
18        a %= MOD;
19        if (a < 0) a += MOD;
20        int b = MOD, u = 0, v = 1;
21        while (a) {
22            int t = b / a;
23            b -= t * a; swap(a, b);
24            u -= t * v; swap(u, v);
25        }
26        assert(b == 1);
27        if (u < 0) u += MOD;
28        return u;
29    }
30 }

```

2.2 筛法

2.2.1 线性素数筛

```

1  vector<bool> isPrime; // true 表示非素数  false 表示是素数
2  vector<int> prime; // 保存素数
3  int sieve(int n) {
4      isPrime.resize(n + 1, false);
5      isPrime[0] = isPrime[1] = true;
6      for (int i = 2; i <= n; i++) {
7          if (!isPrime[i]) prime.emplace_back(i);
8          for (int j = 0; j < (int)prime.size() && prime[j] * i <= n; j++) {
9              isPrime[prime[j] * i] = true;
10             if (!(i % prime[j])) break;
11         }
12     }
13     return (int)prime.size();
14 }

```

2.2.2 线性欧拉函数筛

```

1  bool is_prime[SIZE];
2  int prime[SIZE], phi[SIZE]; // phi[i] 表示 i 的欧拉函数值
3  int Phi(int n) { // 线性筛素数的同时线性求欧拉函数
4      phi[1] = 1; is_prime[1] = true;
5      int p = 0;
6      for (int i = 2; i <= n; i++) {
7          if (!is_prime[i]) prime[p++] = i, phi[i] = i - 1;
8          for (int j = 0; j < p && prime[j] * i <= n; j++) {
9              is_prime[prime[j] * i] = true;
10             if (!(i % prime[j])) {
11                 phi[i * prime[j]] = phi[i] * prime[j];
12                 break;
13             }
14             phi[i * prime[j]] = phi[i] * (prime[j] - 1);
15         }
16     }
17     return p;
18 }

```

2.2.3 线性约数个数函数筛

```

1  bool is_prime[SIZE];

```

```

2  int prime[SIZE], d[SIZE], num[SIZE]; // d[i] 表示 i 的因子数  num[i] 表示 i
    的最小质因子出现次数
3  int getFactors(int n) { // 线性筛因子数
4      d[1] = 1; is_prime[1] = true;
5      int p = 0;
6      for (int i = 2; i <= n; i++) {
7          if (!is_prime[i]) prime[p++] = i, d[i] = 2, num[i] = 1;
8          for (int j = 0; j < p && prime[j] * i <= n; j++) {
9              is_prime[prime[j] * i] = true;
10             if (!(i % prime[j])) {
11                 num[i * prime[j]] = num[i] + 1;
12                 d[i * prime[j]] = d[i] / num[i * prime[j]] * (num[i * prime[
                    j]] + 1);
13                 break;
14             }
15             num[i * prime[j]] = 1;
16             d[i * prime[j]] = d[i] + d[i];
17         }
18     }
19     return p;
20 }

```

2.2.4 线性素因子个数函数筛

```

1  bool is_prime[SIZE];
2  int prime[SIZE], num[SIZE]; // num[i] 表示 i 的质因子数
3  int getPrimeFactors(int n) { // 线性筛质因子数
4      is_prime[1] = true;
5      int p = 0;
6      for (int i = 2; i <= n; i++) {
7          if (!is_prime[i]) prime[p++] = i, num[i] = 1;
8          for (int j = 0; j < p && prime[j] * i <= n; j++) {
9              is_prime[prime[j] * i] = true;
10             if (!(i % prime[j])) {
11                 num[i * prime[j]] = num[i];
12                 break;
13             }
14             num[i * prime[j]] = num[i] + 1;
15         }
16     }
17     return p;
18 }

```

2.2.5 线性约数和函数筛

```

1  bool is_prime[SIZE];
2  int prime[SIZE], f[SIZE], g[SIZE]; // f[i] 表示 i 的约数和
3  int getSigma(int n) {
4      g[1] = f[1] = 1; is_prime[1] = true;
5      int p = 0;
6      for (int i = 2; i <= n; i++) {
7          if (!is_prime[i]) prime[p++] = i, f[i] = g[i] = i + 1;
8          for (int j = 0; j < p && prime[j] * i <= n; j++) {
9              is_prime[prime[j] * i] = true;
10             if (!(i % prime[j])) {
11                 g[i * prime[j]] = g[i] * prime[j] + 1;
12                 f[i * prime[j]] = f[i] / g[i] * g[i * prime[j]];
13                 break;
14             }
15             f[i * prime[j]] = f[i] * f[prime[j]];
16             g[i * prime[j]] = 1 + prime[j];
17         }
18     }
19     return p;
20 }

```

2.2.6 线性莫比乌斯函数筛

```

1  bool is_prime[SIZE];
2  int prime[SIZE], mu[SIZE]; // mu[i] 表示 i 的莫比乌斯函数值
3  int getMu(int n) { // 线性筛莫比乌斯函数
4      mu[1] = 1; is_prime[1] = true;
5      int p = 0;
6      for (int i = 2; i <= n; i++) {
7          if (!is_prime[i]) prime[p++] = i, mu[i] = -1;
8          for (int j = 0; j < p && prime[j] * i <= n; j++) {
9              is_prime[prime[j] * i] = true;
10             if (!(i % prime[j])) {
11                 mu[i * prime[j]] = 0;
12                 break;
13             }
14             mu[i * prime[j]] = -mu[i];
15         }
16     }
17     return p;
18 }

```

2.3 扩展欧几里得

2.3.1 线性同余方程最小非负整数解

exgcd 求 $ax + by = c$ 的最小非负整数解详解:

1. 求出 a, b 的最大公约数 $g = \gcd(a, b)$ ，根据裴蜀定理检查是否满足 $c \% g = 0$ ，不满足则无解；
2. 调整系数 a, b, c 为 $a' = \frac{a}{g}, b' = \frac{b}{g}, c' = \frac{c}{g}$ ，这是因为 $ax + by = c$ 和 $a'x + b'y = c'$ 是完全等价的；
3. 实际上 exgcd 求解的方程是 $a'x + b'y = 1$ ，求解前需要注意让系数 $a', b' \geq 0$ （举个例子，如果系数 b' 原本 < 0 ，我们可以翻转 b' 的符号然后令解 (x, y) 为 $(x, -y)$ ，但是求解的时候要把 y 翻回来)；
4. 我们通过 exgcd 求出一组解 (x_0, y_0) ，这组解满足 $a'x_0 + b'y_0 = 1$ ，为了使解合法我们需要令 $x_0 = c'x_0, y_0 = c'y_0$ ，于是有 $a'(c'x_0) + b'(c'y_0) = c'$ ；
5. 考虑到 $a'x_0 + b'y_0 = 1$ 等价于同余方程 $a'x_0 \equiv 1 \pmod{b'}$ ，因此为了求出最小非负整数解，我们最后还需要对 b' 取模；
6. 最后注意特判 $c' = 0$ 的情况，如果要求解 y 且系数 b 发生了翻转，将其翻转回来。

```

1 long long exgcd(long long a, long long b, long long& x, long long& y) {
2     if (!b) {
3         x = 1, y = 0;
4         return a;
5     }
6     long long g = exgcd(b, a % b, y, x);
7     y -= (a / b) * x;
8     return g;
9 }
10
11 ll x, y; // 最小非负整数解
12 bool solve(ll a, ll b, ll c) { // ax+by=c
13     ll g = gcd(a, b);
14     if (c % g) return false;
15     a /= g, b /= g, c /= g;
16     bool flag = false;
17     if (b < 0) b = -b, flag = true;
18     exgcd(a, b, x, y);
19     x = (x * c % b + b) % b;
20     if (flag) b = -b;
21     y = (c - a * x) / b;
22     if (!c) x = y = 0; // ax+by=0
23     return true;
24 }
```

2.4 欧拉定理

$$a^b \equiv \begin{cases} a^{b \bmod \varphi(p)}, & \gcd(a, p) = 1 \\ a^b, & \gcd(a, p) \neq 1, b < \varphi(p) \\ a^{b \bmod \varphi(p) + \varphi(p)}, & \gcd(a, p) \neq 1, b \geq \varphi(p) \end{cases} \pmod{p}$$

2.5 欧拉函数

2.6 中国剩余定理

2.6.1 CRT

```

1 // 求解形如 x = ci (mod mi) 的线性方程组 (mi, mj)必须两两互质
2 long long CRT(vector<long long>& c, vector<long long>& m) {
3     long long M = m[0], ans = 0;
4     for (int i = 1; i < (int)m.size(); ++i) M *= m[i];
5     for (int i = 0; i < (int)m.size(); ++i) { // Mi * ti * ci
6         long long mi = M / m[i];
7         long long ti = inv(mi, m[i]); // mi 模 m[i] 的逆元
8         ans = (ans + mi * ti % M * c[i] % M) % M;
9     }
10    ans = (ans + M) % M; // 返回模 M 意义下的唯一解
11    return ans;
12 }
```

2.6.2 EXCRT

```

1 long long exgcd(long long a, long long b, long long& x, long long& y) {
2     if (!b) {
3         x = 1, y = 0;
4         return a;
5     }
6     long long g = exgcd(b, a % b, y, x);
7     y -= (a / b) * x;
8     return g;
9 }
10
11 long long mulmod(long long x, long long y, const long long z) { // x * y % z
12     // 防爆
13     return (x * y - (long long)(((long double)x * y + 0.5) / (long double)z)
14         * z + z) % z;
15 }
16
17 // 求解形如 x = ci (mod mi) 的线性方程组
```

```

16 long long EXCRT(vector<long long>& c, vector<long long>& m) {
17     long long M = m[0], ans = c[0];
18     for (int i = 1; i < (int)m.size(); ++i) { // M * x - mi * y = ci - C
19         long long x, y, C = ((c[i] - ans) % m[i] + m[i]) % m[i]; // ci - C
20         long long G = exgcd(M, m[i], x, y);
21         if (C % G) return -1; // 无解
22         long long P = m[i] / G;
23         x = mulmod(C / G, x, P); // 防爆求最小正整数解 x
24         ans += x * M;
25         M *= P; // LCM(M, mi)
26         ans = (ans % M + M) % M;
27     }
28     return ans;
29 }

```

2.7 BSGS

2.8 迪利克雷卷积

$$g(1)S(n) = \sum_{i=1}^n (f * g)(i) - \sum_{i=2}^n g(i)S(\lfloor \frac{n}{i} \rfloor)$$

2.9 杜教筛

$$(f * g)(n) = \sum_{d|n} f(d)g(\frac{n}{d}) = \sum_{xy=n} f(x)g(y)$$

3 线性代数

3.1 高斯-约旦消元法

```

1  /*
2   * 高斯-约旦消元法
3   * 可以修改为解异或方程组 修改策略为
4   * a+b -> a^b
5   * a-b -> a^b
6   * a*b -> a&b
7   * a/b -> a*(b==1)
8   * */
9  constexpr double eps = 1e-7;
10 double a[SIZE][SIZE], ans[SIZE];
11 void gauss(int n) {
12     vector<bool> vis(n, false);
13     for (int i = 0; i < n; i++) {

```

```

14     for (int j = 0; j < n; j++) {
15         if (vis[j]) continue;
16         if (fabs(a[j][i]) > eps) {
17             vis[i] = true;
18             for (int k = 0; k <= n; k++) swap(a[i][k], a[j][k]);
19             break;
20         }
21     }
22     if (fabs(a[i][i]) < eps) continue;
23     for (int j = 0; j <= n; j++) {
24         if (i != j && fabs(a[j][i]) > eps) {
25             double res = a[j][i] / a[i][i];
26             for (int k = 0; k <= n; k++) a[j][k] -= a[i][k] * res;
27         }
28     }
29 }
30 }
31
32 int check(int n) { // 解系检测
33     int status = 1;
34     for (int i = n - 1; i >= 0; i--) {
35         if (fabs(a[i][i]) < eps && fabs(a[i][n]) > eps) return -1; // 无解
36         if (fabs(a[i][i]) < eps && fabs(a[i][n]) < eps) status = 0; // 无穷
           解
37         ans[i] = a[i][n] / a[i][i];
38         if (fabs(ans[i]) < eps) ans[i] = 0;
39     }
40     return status; // 唯一解或无穷解
41 }

```

3.2 高斯消元法-bitset

```

1  constexpr int SIZE = 1001;
2  bitset<SIZE> a[SIZE];
3  int ans[SIZE];
4  void gauss(int n) { // bitset版高斯消元 用于求解异或线性方程组
5      bitset<SIZE> vis;
6      for (int i = 0; i < n; i++) {
7          for (int j = 0; j < n; j++) {
8              if (vis[j]) continue;
9              if (a[j][i]) {
10                 vis.set(i);
11                 swap(a[i], a[j]);

```



```

12         break;
13     }
14 }
15 if (!a[i][i]) continue;
16 for (int j = 0; j <= n; j++) {
17     if (i != j && (a[j][i] & a[i][i])) {
18         a[j] ^= a[i];
19     }
20 }
21 }
22 }

```

3.3 线性基

```

1 struct linearBasis {
2     /* 线性基性质:
3      * 1. 若a[i]!=0 (即主元i存在)
4      *    则线性基中只有a[i]的第i位是1 (只存在一个主元)
5      *    且此时a[i]的最高位就是第i位
6      * 2. 将数组a插入线性基 假设有|B|个元素成功插入
7      *    则数组a中每个不同的子集异或和都出现  $2^{(n-|B|)}$  次
8      * */
9     static const int MAXL = 60;
10    long long a[MAXL + 1];
11    int id[MAXL + 1];
12    int zero;
13    /* 0的标志位 =1则表示0可以被线性基表示出来
14     * 求第k大元素时 需要注意题意中线性基为空时是否可以表示0
15     * 默认不可以表示 必要时进行修改即可
16     * */
17    linearBasis() {
18        zero = 0;
19        fill(a, a + MAXL + 1, 0);
20    }
21    long long& operator[] (int k) { return a[k]; }
22    bool insert(long long x) {
23        for (int j = MAXL; ~j; j--) {
24            if (!(x & (1LL << j))) { // 如果 x 的第 j 位为 0, 则跳过
25                continue;
26            }
27            if (a[j]) { // 如果 a[j] != 0, 则用 a[j] 消去 x 的第 j 位上的 1
28                x ^= a[j];
29            } else { // 找到插入位置

```

```

30         for (int k = 0; k < j; k++) {
31             if (x & (1LL << k)) { // 如果x存在某个低位线性基的主元k
                                     则消去
32                 x ^= a[k];
33             }
34         }
35         for (int k = j + 1; k <= MAXL; k++) {
36             if (a[k] & (1LL << j)) { // 如果某个高位线性基存在主元j
                                     则消去
37                 a[k] ^= x;
38             }
39         }
40         a[j] = x;
41         return true;
42     }
43 }
44 zero = 1;
45 return false;
46 }
47 long long query_max() { // 最大值
48     long long res = 0;
49     for (int i = MAXL; ~i; i--) {
50         res ^= a[i];
51     }
52     return res;
53 }
54 long long query_max(long long x) { // 线性基异或x的最大值
55     for (int i = MAXL; ~i; i--) {
56         if ((x ^ a[i]) > x) {
57             x ^= a[i];
58         }
59     }
60     return x;
61 }
62 long long query_min() { // 最小值
63     for (int i = 0; i < MAXL; i++) {
64         if (a[i]) {
65             return a[i];
66         }
67     }
68     return -1; // 线性基为空
69 }
70 long long query_min(long long x) { // 线性基异或x的最小值

```

```

71     for (int i = MAXL; ~i; i--) {
72         if ((x ^ a[i]) < x) {
73             x ^= a[i];
74         }
75     }
76     return x;
77 }
78 int count(long long x) { // 元素 x 能否被线性基内元素表示
79     int res = 0;
80     vector<long long> b(MAXL + 1);
81     for (int i = 0; i <= MAXL; i++) {
82         b[i] = a[i];
83     }
84     res = this->insert(x);
85     for (int i = 0; i <= MAXL; i++) {
86         a[i] = b[i];
87     }
88     return !res; // 成功插入则无法表示 失败则可以表示
89 }
90 int size() { // 线性基有效元素数量
91     int res = 0;
92     for (int i = 0; i <= MAXL; i++) {
93         if (a[i]) {
94             res++;
95         }
96     }
97     return res;
98 }
99 long long kth_element(long long k) { // 第k大元素
100     vector<long long> b;
101     for (int i = 0; i <= MAXL; i++) {
102         if (a[i]) {
103             b.push_back(a[i]);
104         }
105     }
106     if (zero) {
107         if (--k == 0) {
108             return 0;
109         }
110     }
111     if (k >= (1LL << this->size())) { // k超过了线性基可以表示的最大数量
112         return -1;
113     }

```

```

114     long long res = 0;
115     for (int i = 0; i <= MAXL; i++) {
116         if (k & (1LL << i)) {
117             res ^= b[i];
118         }
119     }
120     return res;
121 }
122 long long rank(long long x) { // 元素x在线性基内的排名 (默认不考虑0)
123     vector<long long> b;
124     for (int i = 0; i <= MAXL; i++) {
125         if (a[i]) {
126             b.push_back(1LL << i);
127         }
128     }
129     long long res = 0;
130     for (int i = 0; i < (int)b.size(); i++) {
131         if (x & b[i]) {
132             res |= (1LL << i);
133         }
134     }
135     return res;
136 }
137 void clear() {
138     zero = 0;
139     fill(a, a + MAXL + 1, 0);
140 }
141 };

```

3.4 矩阵树定理

```

1  /*
2  * 矩阵树定理
3  * 有向图: 若 u->v 有一条权值为 w 的边 基尔霍夫矩阵 a[v][v] += w, a[v][u] -=
      w
4  * 生成树数量为除去 根所在行和列 后的n-1阶行列式的值
5  * 无向图: 若 u->v 有一条权值为 w 的边 基尔霍夫矩阵 a[v][v] += w, a[v][u] -=
      w, a[u][u] += w, a[u][v] -= w
6  * 生成树数量为除去 任意一行和列 后的n-1阶行列式的值
7  * 无权图则边权默认为1
8  * */
9  typedef long long ll;
10 typedef unsigned long long u64;

```

```

11 int a[SIZE][SIZE];
12 int gauss(int a[][SIZE], int n) { // 任意模数求行列式  $O(n^2(n + \log(\text{mod})))$ 
13     int ans = 1;
14     for (int i = 1; i <= n; i++) {
15         int* x = 0, * y = 0;
16         for (int j = i; j <= n; j++) {
17             if (a[j][i] && (x == NULL || a[j][i] < x[i])) {
18                 x = a[j];
19             }
20         }
21         if (x == 0) {
22             return 0;
23         }
24         for (int j = i; j <= n; j++) {
25             if (a[j] != x && a[j][i]) {
26                 y = a[j];
27                 for (;;) {
28                     int v = md - y[i] / x[i], k = i;
29                     for (; k + 3 <= n; k += 4) {
30                         y[k + 0] = (y[k + 0] + u64(x[k + 0]) * v) % md;
31                         y[k + 1] = (y[k + 1] + u64(x[k + 1]) * v) % md;
32                         y[k + 2] = (y[k + 2] + u64(x[k + 2]) * v) % md;
33                         y[k + 3] = (y[k + 3] + u64(x[k + 3]) * v) % md;
34                     }
35                     for (; k <= n; ++k) {
36                         y[k] = (y[k] + u64(x[k]) * v) % md;
37                     }
38                     if (!y[i]) break;
39                     swap(x, y);
40                 }
41             }
42         }
43         if (x != a[i]) {
44             for (int k = i; k <= n; k++) {
45                 swap(x[k], a[i][k]);
46             }
47             ans = md - ans;
48         }
49         ans = 1LL * ans * a[i][i] % md;
50     }
51     return ans;
52 }

```

3.5 LGV 引理

4 组合数学

4.1 组合数预处理

```

1 namespace BinomialCoefficient {
2     vector<int> fac, ifac, iv;
3     // 组合数预处理 option=1则还会预处理线性逆元
4     void prepareFactorials(int maximum = 1000000, int option = 0) {
5         fac.assign(maximum + 1, 0);
6         ifac.assign(maximum + 1, 0);
7         fac[0] = ifac[0] = 1;
8         if (option) { // O(3n)
9             iv.assign(maximum + 1, 1);
10            for (int p = 2; p * p <= MOD; p++)
11                assert(MOD % p != 0);
12            for (int i = 2; i <= maximum; i++)
13                iv[i] = mul(iv[MOD % i], (MOD - MOD / i));
14            for (int i = 1; i <= maximum; i++) {
15                fac[i] = mul(i, fac[i - 1]);
16                ifac[i] = mul(iv[i], ifac[i - 1]);
17            }
18        } else { // O(2n + log(MOD))
19            for (int i = 1; i <= maximum; i++)
20                fac[i] = mul(fac[i - 1], i);
21            ifac[maximum] = inv(fac[maximum]);
22            for (int i = maximum; i; i--)
23                ifac[i - 1] = mul(ifac[i], i);
24        }
25    }
26    inline int binom(int n, int m) {
27        if (n < m || n < 0 || m < 0) return 0;
28        return mul(fac[n], mul(ifac[m], ifac[n - m]));
29    }
30 }

```

4.2 卢卡斯定理

4.3 小球盒子模型

4.4 斯特林数

4.4.1 第一类斯特林数

第一类斯特林数 $\begin{bmatrix} n \\ k \end{bmatrix}$ 表示将 n 个不同元素划分入 k 个非空圆排列的方案数。

$$\begin{bmatrix} n \\ k \end{bmatrix} = \begin{bmatrix} n-1 \\ k-1 \end{bmatrix} + (n-1) \begin{bmatrix} n-1 \\ k \end{bmatrix}$$

边界是 $\begin{bmatrix} 0 \\ 0 \end{bmatrix} = 1$ 。

第一类斯特林数三角形，从 $s(1, 1)$ 开始：

1									
1	1								
2	3	1							
6	11	6	1						
24	50	35	10	1					
120	274	225	85	15	1				
720	1764	1624	735	175	21	1			
5040	13068	13132	6769	1960	322	28	1		
40320	109584	118124	67284	22449	4536	546	36	1	
362880	1026576	1172700	723680	269325	63273	9450	870	45	1

4.4.2 第二类斯特林数

第二类斯特林数 $\begin{Bmatrix} n \\ k \end{Bmatrix}$ 表示将 n 个不同元素划分为 k 个非空子集的方案数。

$$\begin{Bmatrix} n \\ k \end{Bmatrix} = \begin{Bmatrix} n-1 \\ k-1 \end{Bmatrix} + k \begin{Bmatrix} n-1 \\ k \end{Bmatrix}$$

边界是 $\begin{Bmatrix} 0 \\ 0 \end{Bmatrix} = 1$ 。

基于容斥原理的递推方法：

$$\begin{Bmatrix} n \\ k \end{Bmatrix} = \frac{1}{k!} \sum_{i=0}^k (-1)^i \binom{k}{i} (k-i)^n$$

第二类斯特林数三角形，从 $S(1, 1)$ 开始：

```

1
1 1
1 3 1
1 7 6 1
1 15 25 10 1
1 31 90 65 15 1
1 63 301 350 140 21 1
1 127 966 1701 1050 266 28 1
1 255 3025 7770 6951 2646 462 36 1
1 511 9330 34105 42525 22827 5880 750 45 1

```

5 博弈论

6 图论

6.1 并查集

```

1 struct dsu {
2 private:
3     // number of nodes
4     int n;
5     // root node: -1 * component size
6     // otherwise: parent
7     std::vector<int> pa;
8 public:
9     dsu(int n_ = 0) : n(n_), pa(n_, -1) {}
10    // find node x's parent
11    int find(int x) {
12        return pa[x] < 0 ? x : pa[x] = find(pa[x]);
13    }
14    // merge node x and node y
15    // if x and y had already in the same component, return false, otherwise
16    // return true
17    // Implement (union by size) + (path compression)
18    bool unite(int x, int y) {
19        int xr = find(x), yr = find(y);
20        if (xr != yr) {
21            if (-pa[xr] < -pa[yr]) std::swap(xr, yr);
22            pa[xr] += pa[yr];
23            pa[yr] = xr; // y -> x
24            return true;
25        }
26        return false;

```



```

26     }
27     // size of the connected component that contains the vertex x
28     int size(int x) {
29         return -pa[find(x)];
30     }
31 };

```

6.2 最小树形图

```

1  namespace ZL {
2      // a 尽量开大，之后的边都塞在这个里面
3      const int N = 100010, M = 100010, inf = 1e9;
4      struct edge {
5          int u, v, w, use, id;
6          edge(int u_ = 0, int v_ = 0, int w_ = 0, int use_ = 0, int id_ = 0)
7              : u(u_), v(v_), w(w_), use(use_), id(id_) {}
8      } b[M], a[2000100];
9      int n, m, ans, pre[N], id[N], vis[N], root, In[N], h[N], len, way[M];
10     // 从root 出发能到达每一个点的最小树形图
11     // 先调用init 然后把边add 进去，需要方案就getway, way[i] 为1 表示使用
12     // 最小值保存在ans
13     void init(int _n, int _root) { // 点数 根节点
14         n = _n; m = 0; b[0].w = inf; root = _root;
15     }
16     void add(int u, int v, int w) {
17         m++;
18         b[m] = edge(u, v, w, 0, m);
19         a[m] = b[m];
20     }
21     int work() {
22         len = m;
23         for (;;) {
24             for (int i = 1; i <= n; i++) { pre[i] = 0; In[i] = inf; id[i] = 0; vis[i] = 0; h[i] = 0; }
25             for (int i = 1; i <= m; i++) {
26                 if (b[i].u != b[i].v && b[i].w < In[b[i].v]) {
27                     pre[b[i].v] = b[i].u; In[b[i].v] = b[i].w; h[b[i].v] = b[i].id;
28                 }
29             }
30             for (int i = 1; i <= n; i++) if (pre[i] == 0 && i != root)
31                 return 0;
32             int cnt = 0; In[root] = 0;

```

```

32     for (int i = 1; i <= n; i++) {
33         if (i != root) a[h[i]].use++; int now = i; ans += In[i];
34         while (vis[now] == 0 && now != root) { vis[now] = i; now =
           pre[now]; }
35         if (now != root && vis[now] == i) {
36             cnt++; int kk = now;
37             while (1) {
38                 id[now] = cnt; now = pre[now];
39                 if (now == kk) break;
40             }
41         }
42     }
43     if (cnt == 0) return 1;
44     for (int i = 1; i <= n; i++) if (id[i] == 0) id[i] = ++cnt;
45     // 缩环，每一条接入的边都会茶包原来接入的那条边，所以要调整边权
46     // 新加的边是u，茶包的边是v
47     for (int i = 1; i <= m; i++) {
48         int k1 = In[b[i].v], k2 = b[i].v;
49         b[i].u = id[b[i].u];
50         b[i].v = id[b[i].v];
51         if (b[i].u != b[i].v) {
52             b[i].w -= k1; a[++len].u = b[i].id; a[len].v = h[k2]; b[
               i].id = len;
53         }
54     }
55     n = cnt; root = id[root];
56 }
57 return 1;
58 }
59 void getway() {
60     for (int i = 1; i <= m; i++) way[i] = 0;
61     for (int i = len; i > m; i--) { a[a[i].u].use += a[i].use; a[a[i].v
           ].use -= a[i].use; }
62     for (int i = 1; i <= m; i++) way[i] = a[i].use;
63 }
64 }

```

6.3 最近公共祖先

```

1  constexpr int SIZE = 200010;
2  constexpr int DEPTH = 21; // 最大深度 2^DEPTH - 1
3  int pa[SIZE][DEPTH], dep[SIZE];
4  vector<int> g[SIZE]; // 邻接表

```

```

5 void dfs(int rt, int fin) { //预处理深度和祖先
6     pa[rt][0] = fin;
7     dep[rt] = dep[pa[rt][0]] + 1; //深度
8     for (int i = 1; i < DEPTH; i++) { // rt 的  $2^i$  祖先等价于 rt 的  $2^{(i-1)}$ 
          祖先的  $2^{(i-1)}$  祖先
9         pa[rt][i] = pa[pa[rt][i-1]][i-1];
10    }
11    int sz = g[rt].size();
12    for (int i = 0; i < sz; ++i) {
13        if (g[rt][i] == fin) continue;
14        dfs(g[rt][i], rt);
15    }
16 }
17
18 int LCA(int x, int y) {
19     if (dep[x] > dep[y]) swap(x, y);
20     int dif = dep[y] - dep[x];
21     for (int j = 0; dif; ++j, dif >>= 1) {
22         if (dif & 1) {
23             y = pa[y][j]; //先跳到同一高度
24         }
25     }
26     if (y == x) return x;
27     for (int j = DEPTH - 1; j >= 0 && y != x; j--) { //从底往上跳
28         if (pa[x][j] != pa[y][j]) { //如果当前祖先不相等 我们就需要更新
29             x = pa[x][j];
30             y = pa[y][j];
31         }
32     }
33     return pa[x][0];
34 }

```

6.4 强连通分量

```

1 namespace SCC {
2     // Compressed Sparse Row
3     template <class E> struct csr {
4         std::vector<int> start;
5         std::vector<E> elist;
6         explicit csr(int n, const std::vector<std::pair<int, E>>& edges)
7             : start(n + 1), elist(edges.size()) {
8             for (auto e : edges) {
9                 start[e.first + 1]++;

```

```

10         }
11         for (int i = 1; i <= n; i++) {
12             start[i] += start[i - 1];
13         }
14         auto counter = start;
15         for (auto e : edges) {
16             elist[counter[e.first]++] = e.second;
17         }
18     }
19 };
20
21 struct scc_graph {
22 public:
23     explicit scc_graph(int n) : _n(n) {}
24
25     int num_vertices() { return _n; }
26
27     void add_edge(int from, int to) { edges.push_back({ from, {to} }); }
28
29     // @return pair of (# of scc, scc id)
30     std::pair<int, std::vector<int>> scc_ids() {
31         auto g = csr<edge>(_n, edges);
32         int now_ord = 0, group_num = 0;
33         std::vector<int> visited, low(_n), ord(_n, -1), ids(_n);
34         visited.reserve(_n);
35         auto dfs = [&](auto self, int v) -> void {
36             low[v] = ord[v] = now_ord++;
37             visited.push_back(v);
38             for (int i = g.start[v]; i < g.start[v + 1]; i++) {
39                 auto to = g.elist[i].to;
40                 if (ord[to] == -1) {
41                     self(self, to);
42                     low[v] = std::min(low[v], low[to]);
43                 } else {
44                     low[v] = std::min(low[v], ord[to]);
45                 }
46             }
47             if (low[v] == ord[v]) {
48                 while (true) {
49                     int u = visited.back();
50                     visited.pop_back();
51                     ord[u] = _n;
52                     ids[u] = group_num;

```

```

53         if (u == v) break;
54     }
55     group_num++;
56 }
57 };
58 for (int i = 0; i < _n; i++) {
59     if (ord[i] == -1) dfs(dfs, i);
60 }
61 for (auto& x : ids) {
62     x = group_num - 1 - x;
63 }
64 return { group_num, ids };
65 }
66
67 // O(N + M)
68 // It returns the list of the SCC in topological order.
69 std::vector<std::vector<int>>> scc() {
70     auto ids = scc_ids();
71     int group_num = ids.first;
72     std::vector<int> counts(group_num);
73     for (auto x : ids.second) counts[x]++;
74     std::vector<std::vector<int>>> groups(ids.first);
75     for (int i = 0; i < group_num; i++) {
76         groups[i].reserve(counts[i]);
77     }
78     for (int i = 0; i < _n; i++) {
79         groups[ids.second[i]].push_back(i);
80     }
81     return groups;
82 }
83
84 private:
85     int _n;
86     struct edge {
87         int to;
88     };
89     std::vector<std::pair<int, edge>>> edges;
90 };
91 }

```

6.5 最大流

```

1 template <class T> struct simple_queue {

```

```

2      std::vector<T> payload;
3      int pos = 0;
4      void reserve(int n) { payload.reserve(n); }
5      int size() const { return int(payload.size()) - pos; }
6      bool empty() const { return pos == int(payload.size()); }
7      void push(const T& t) { payload.push_back(t); }
8      T& front() { return payload[pos]; }
9      void clear() {
10         payload.clear();
11         pos = 0;
12     }
13     void pop() { pos++; }
14 };
15
16 template <class Cap> struct mf_graph {
17 public:
18     mf_graph() : _n(0) {}
19     mf_graph(int n) : _n(n), g(n) {}
20
21     // returns an integer k such that this is the k-th edge that is added.
22     int add_edge(int from, int to, Cap cap) {
23         assert(0 <= from && from < _n);
24         assert(0 <= to && to < _n);
25         assert(0 <= cap);
26         int m = int(pos.size());
27         pos.push_back({ from, int(g[from].size()) });
28         int from_id = int(g[from].size());
29         int to_id = int(g[to].size());
30         if (from == to) to_id++;
31         g[from].push_back(_edge{ to, to_id, cap });
32         g[to].push_back(_edge{ from, from_id, 0 });
33         return m;
34     }
35
36     struct edge {
37         int from, to;
38         Cap cap, flow;
39     };
40
41     edge get_edge(int i) {
42         int m = int(pos.size());
43         assert(0 <= i && i < m);
44         auto _e = g[pos[i].first][pos[i].second];

```

```

45     auto _re = g[_e.to][_e.rev];
46     return edge{ pos[i].first, _e.to, _e.cap + _re.cap, _re.cap };
47 }
48 std::vector<edge> edges() {
49     int m = int(pos.size());
50     std::vector<edge> result;
51     for (int i = 0; i < m; i++) {
52         result.push_back(get_edge(i));
53     }
54     return result;
55 }
56 void change_edge(int i, Cap new_cap, Cap new_flow) {
57     int m = int(pos.size());
58     assert(0 <= i && i < m);
59     assert(0 <= new_flow && new_flow <= new_cap);
60     auto& _e = g[pos[i].first][pos[i].second];
61     auto& _re = g[_e.to][_e.rev];
62     _e.cap = new_cap - new_flow;
63     _re.cap = new_flow;
64 }
65
66 // max flow from s to t
67 // O(M*N^2) general
68 // O(min(M*N^2/3, M^3/2)) if capacities of edges are 1
69 Cap flow(int s, int t) {
70     return flow(s, t, std::numeric_limits<Cap>::max());
71 }
72 Cap flow(int s, int t, Cap flow_limit) {
73     assert(0 <= s && s < _n);
74     assert(0 <= t && t < _n);
75     assert(s != t);
76
77     std::vector<int> level(_n), iter(_n);
78     simple_queue<int> que;
79
80     auto bfs = [&]() {
81         std::fill(level.begin(), level.end(), -1);
82         level[s] = 0;
83         que.clear();
84         que.push(s);
85         while (!que.empty()) {
86             int v = que.front();
87             que.pop();

```

```

88         for (auto e : g[v]) {
89             if (e.cap == 0 || level[e.to] >= 0) continue;
90             level[e.to] = level[v] + 1;
91             if (e.to == t) return;
92             que.push(e.to);
93         }
94     }
95 };
96 auto dfs = [&](auto self, int v, Cap up) {
97     if (v == s) return up;
98     Cap res = 0;
99     int level_v = level[v];
100    for (int& i = iter[v]; i < int(g[v].size()); i++) {
101        _edge& e = g[v][i];
102        if (level_v <= level[e.to] || g[e.to][e.rev].cap == 0)
103            continue;
104        Cap d =
105            self(self, e.to, std::min(up - res, g[e.to][e.rev].cap));
106        if (d <= 0) continue;
107        g[v][i].cap += d;
108        g[e.to][e.rev].cap -= d;
109        res += d;
110        if (res == up) break;
111    }
112    return res;
113 };
114
115 Cap flow = 0;
116 while (flow < flow_limit) {
117     bfs();
118     if (level[t] == -1) break;
119     std::fill(iter.begin(), iter.end(), 0);
120     while (flow < flow_limit) {
121         Cap f = dfs(dfs, t, flow_limit - flow);
122         if (!f) break;
123         flow += f;
124     }
125 }
126 return flow;
127 }
128
129 std::vector<bool> min_cut(int s) {

```



```

129     std::vector<bool> visited(_n);
130     simple_queue<int> que;
131     que.push(s);
132     while (!que.empty()) {
133         int p = que.front();
134         que.pop();
135         visited[p] = true;
136         for (auto e : g[p]) {
137             if (e.cap && !visited[e.to]) {
138                 visited[e.to] = true;
139                 que.push(e.to);
140             }
141         }
142     }
143     return visited;
144 }
145
146 private:
147     int _n;
148     struct _edge {
149         int to, rev;
150         Cap cap;
151     };
152     std::vector<std::pair<int, int>> pos;
153     std::vector<std::vector<_edge>> g;
154 };

```

6.6 最小费用最大流

```

1 namespace MCMF {
2     template <class T> struct simple_queue {
3         std::vector<T> payload;
4         int pos = 0;
5         void reserve(int n) { payload.reserve(n); }
6         int size() const { return int(payload.size()) - pos; }
7         bool empty() const { return pos == int(payload.size()); }
8         void push(const T& t) { payload.push_back(t); }
9         T& front() { return payload[pos]; }
10        void clear() {
11            payload.clear();
12            pos = 0;
13        }
14        void pop() { pos++; }

```

```

15     };
16
17     template <class E> struct csr {
18         std::vector<int> start;
19         std::vector<E> elist;
20         explicit csr(int n, const std::vector<std::pair<int, E>>& edges)
21             : start(n + 1), elist(edges.size()) {
22             for (auto e : edges) {
23                 start[e.first + 1]++;
24             }
25             for (int i = 1; i <= n; i++) {
26                 start[i] += start[i - 1];
27             }
28             auto counter = start;
29             for (auto e : edges) {
30                 elist[counter[e.first]++] = e.second;
31             }
32         }
33     };
34
35     template <class Cap, class Cost> struct mcf_graph {
36     public:
37         mcf_graph() {}
38         explicit mcf_graph(int n) : _n(n) {}
39
40         int add_edge(int from, int to, Cap cap, Cost cost) {
41             assert(0 <= from && from < _n);
42             assert(0 <= to && to < _n);
43             assert(0 <= cap);
44             assert(0 <= cost);
45             int m = int(_edges.size());
46             _edges.push_back({ from, to, cap, 0, cost });
47             return m;
48         }
49
50         struct edge {
51             int from, to;
52             Cap cap, flow;
53             Cost cost;
54         };
55
56         edge get_edge(int i) {
57             int m = int(_edges.size());

```

```

58         assert(0 <= i && i < m);
59         return _edges[i];
60     }
61     std::vector<edge> edges() { return _edges; }
62
63     std::pair<Cap, Cost> flow(int s, int t) {
64         return flow(s, t, std::numeric_limits<Cap>::max());
65     }
66     std::pair<Cap, Cost> flow(int s, int t, Cap flow_limit) {
67         return slope(s, t, flow_limit).back();
68     }
69     std::vector<std::pair<Cap, Cost>> slope(int s, int t) {
70         return slope(s, t, std::numeric_limits<Cap>::max());
71     }
72     std::vector<std::pair<Cap, Cost>> slope(int s, int t, Cap flow_limit
73 ) {
74         assert(0 <= s && s < _n);
75         assert(0 <= t && t < _n);
76         assert(s != t);
77
78         int m = int(_edges.size());
79         std::vector<int> edge_idx(m);
80
81         auto g = [&]() {
82             std::vector<int> degree(_n), redge_idx(m);
83             std::vector<std::pair<int, _edge>> elist;
84             elist.reserve(2 * m);
85             for (int i = 0; i < m; i++) {
86                 auto e = _edges[i];
87                 edge_idx[i] = degree[e.from]++;
88                 redge_idx[i] = degree[e.to]++;
89                 elist.push_back({ e.from, {e.to, -1, e.cap - e.flow, e.
90                     cost} });
91                 elist.push_back({ e.to, {e.from, -1, e.flow, -e.cost} })
92                     ;
93             }
94             auto _g = csr<_edge>(_n, elist);
95             for (int i = 0; i < m; i++) {
96                 auto e = _edges[i];
97                 edge_idx[i] += _g.start[e.from];
98                 redge_idx[i] += _g.start[e.to];
99                 _g.elist[edge_idx[i]].rev = redge_idx[i];
100                 _g.elist[redge_idx[i]].rev = edge_idx[i];

```

```

98         }
99         return _g;
100     }();
101
102     auto result = slope(g, s, t, flow_limit);
103
104     for (int i = 0; i < m; i++) {
105         auto e = g.elist[edge_idx[i]];
106         _edges[i].flow = _edges[i].cap - e.cap;
107     }
108
109     return result;
110 }
111
112 private:
113     int _n;
114     std::vector<edge> _edges;
115
116     // inside edge
117     struct _edge {
118         int to, rev;
119         Cap cap;
120         Cost cost;
121     };
122
123     std::vector<std::pair<Cap, Cost>> slope(csr<_edge>& g,
124         int s,
125         int t,
126         Cap flow_limit) {
127         // variants (C = maxcost):
128         //  $-(n-1)C \leq \text{dual}[s] \leq \text{dual}[i] \leq \text{dual}[t] = 0$ 
129         // reduced cost (= e.cost + dual[e.from] - dual[e.to])  $\geq 0$  for
130             all edge
131
132         // dual_dist[i] = (dual[i], dist[i])
133         std::vector<std::pair<Cost, Cost>> dual_dist(_n);
134         std::vector<int> prev_e(_n);
135         std::vector<bool> vis(_n);
136         struct Q {
137             Cost key;
138             int to;
139             bool operator<(Q r) const { return key > r.key; }
140         };

```

```

140     std::vector<int> que_min;
141     std::vector<Q> que;
142     auto dual_ref = [&]() {
143         for (int i = 0; i < _n; i++) {
144             dual_dist[i].second = std::numeric_limits<Cost>::max();
145         }
146         std::fill(vis.begin(), vis.end(), false);
147         que_min.clear();
148         que.clear();
149
150         // que[0..heap_r) was heapified
151         size_t heap_r = 0;
152
153         dual_dist[s].second = 0;
154         que_min.push_back(s);
155         while (!que_min.empty() || !que.empty()) {
156             int v;
157             if (!que_min.empty()) {
158                 v = que_min.back();
159                 que_min.pop_back();
160             } else {
161                 while (heap_r < que.size()) {
162                     heap_r++;
163                     std::push_heap(que.begin(), que.begin() + heap_r);
164                 }
165                 v = que.front().to;
166                 std::pop_heap(que.begin(), que.end());
167                 que.pop_back();
168                 heap_r--;
169             }
170             if (vis[v]) continue;
171             vis[v] = true;
172             if (v == t) break;
173             // dist[v] = shortest(s, v) + dual[s] - dual[v]
174             // dist[v] >= 0 (all reduced cost are positive)
175             // dist[v] <= (n-1)C
176             Cost dual_v = dual_dist[v].first, dist_v = dual_dist[v].
                second;
177             for (int i = g.start[v]; i < g.start[v + 1]; i++) {
178                 auto e = g.elist[i];
179                 if (!e.cap) continue;
180                 // |-dual[e.to] + dual[v]| <= (n-1)C

```

```

181         // cost <= C - -(n-1)C + 0 = nC
182         Cost cost = e.cost - dual_dist[e.to].first + dual_v;
183         if (dual_dist[e.to].second - dist_v > cost) {
184             Cost dist_to = dist_v + cost;
185             dual_dist[e.to].second = dist_to;
186             prev_e[e.to] = e.rev;
187             if (dist_to == dist_v) {
188                 que_min.push_back(e.to);
189             } else {
190                 que.push_back(Q{ dist_to, e.to });
191             }
192         }
193     }
194 }
195 if (!vis[t]) {
196     return false;
197 }
198
199 for (int v = 0; v < _n; v++) {
200     if (!vis[v]) continue;
201     // dual[v] = dual[v] - dist[t] + dist[v]
202     //          = dual[v] - (shortest(s, t) + dual[s] - dual[
203         t]) +
204     //          (shortest(s, v) + dual[s] - dual[v]) = -
205         shortest(s,
206         //          t) + dual[t] + shortest(s, v) = shortest(s, v
207         ) -
208     //          shortest(s, t) >= 0 - (n-1)C
209     dual_dist[v].first -= dual_dist[t].second - dual_dist[v
210         ].second;
211 }
212 return true;
213 };
214 Cap flow = 0;
215 Cost cost = 0, prev_cost_per_flow = -1;
216 std::vector<std::pair<Cap, Cost>> result = { {Cap(0), Cost(0)}
217     };
218 while (flow < flow_limit) {
219     if (!dual_ref()) break;
220     Cap c = flow_limit - flow;
221     for (int v = t; v != s; v = g.elist[prev_e[v]].to) {
222         c = std::min(c, g.elist[g.elist[prev_e[v]].rev].cap);
223     }

```

```

219         for (int v = t; v != s; v = g.elist[prev_e[v]].to) {
220             auto& e = g.elist[prev_e[v]];
221             e.cap += c;
222             g.elist[e.rev].cap -= c;
223         }
224         Cost d = -dual_dist[s].first;
225         flow += c;
226         cost += c * d;
227         if (prev_cost_per_flow == d) {
228             result.pop_back();
229         }
230         result.push_back({ flow, cost });
231         prev_cost_per_flow = d;
232     }
233     return result;
234 }
235 };
236 }

```

6.7 全局最小割

6.8 二分图最大权匹配

```

1 namespace KM {
2     typedef long long ll;
3     const int maxn = 510;
4     const int inf = 1e9;
5     int vx[maxn], vy[maxn], lx[maxn], ly[maxn], slack[maxn];
6     int w[maxn][maxn]; // 以上为权值类型
7     int pre[maxn], left[maxn], right[maxn], NL, NR, N;
8     void match(int& u) {
9         for (; u; std::swap(u, right[pre[u]]))
10             left[u] = pre[u];
11     }
12     void bfs(int u) {
13         static int q[maxn], front, rear;
14         front = 0; vx[q[rear = 1] = u] = true;
15         while (true) {
16             while (front < rear) {
17                 int u = q[++front];
18                 for (int v = 1; v <= N; ++v) {
19                     int tmp;
20                     if (vy[v] || (tmp = lx[u] + ly[v] - w[u][v]) > slack[v])
21                         continue;

```

```

22         pre[v] = u;
23         if (!tmp) {
24             if (!left[v]) return match(v);
25             vy[v] = vx[q[++rear] = left[v]] = true;
26         } else slack[v] = tmp;
27     }
28 }
29 int a = inf;
30 for (int i = 1; i <= N; ++i)
31     if (!vy[i] && a > slack[i]) a = slack[u = i];
32 for (int i = 1; i <= N; ++i) {
33     if (vx[i]) lx[i] -= a;
34     if (vy[i]) ly[i] += a;
35     else slack[i] -= a;
36 }
37 if (!left[u]) return match(u);
38 vy[u] = vx[q[++rear] = left[u]] = true;
39
40 }
41
42 }
43 void exec() {
44     for (int i = 1; i <= N; ++i) {
45         for (int j = 1; j <= N; ++j) {
46             slack[j] = inf;
47             vx[j] = vy[j] = false;
48         }
49         bfs(i);
50     }
51 }
52 ll work(int nl, int nr) { // NL, NR 为左右点数, 返回最大权匹配的权值和
53     NL = nl; NR = nr;
54     N = std::max(NL, NR);
55     for (int u = 1; u <= N; ++u)
56         for (int v = 1; v <= N; ++v)
57             lx[u] = std::max(lx[u], w[u][v]);
58     exec();
59     ll ans = 0;
60     for (int i = 1; i <= N; ++i)
61         ans += lx[i] + ly[i];
62     return ans;
63 }
64 void output() { // 输出左边点与右边哪个点匹配, 没有匹配输出0

```



```

65     for (int i = 1; i <= NL; ++i)
66         printf("%d ", (w[i][right[i]] ? right[i] : 0));
67     printf("\n");
68 }
69 }

```

6.9 一般图最大匹配

6.10 2-sat

6.11 最大团

```

1  /*
2  * 最大团 Bron-Kerbosch algorithm
3  * 最劣复杂度  $O(3^{(n/3)})$ 
4  * 采用位运算形式实现
5  * */
6  namespace Max_clique {
7  #define ll long long
8  #define TWOL(x) (1ll << (x))
9      const int N = 60;
10     int n, m;          // 点数 边数
11     int r = 0;          // 最大团大小
12     ll G[N];           // 以二进制形式存图
13     ll clique = 0;      // 最大团 以二进制形式存储
14     void BronK(int S, ll P, ll X, ll R) { // 调用时参数这样设置: 0, TWOL(n)
15         -1, 0, 0
16         if (P == 0 && X == 0) {
17             if (r < S) {
18                 r = S;
19                 clique = R;
20             }
21         }
22         if (P == 0) return;
23         int u = __builtin_ctzll(P | X);
24         ll c = P & ~G[u];
25         while (c) {
26             int v = __builtin_ctzll(c);
27             ll pv = TWOL(v);
28             BronK(S + 1, P & G[v], X & G[v], R | pv);
29             P ^= pv; X |= pv; c ^= pv;
30         }
31     }
32     void init() {

```

```

32     cin >> n >> m;
33     for (int i = 0; i < m; i++) {
34         int u, v;
35         cin >> u >> v;
36         --u, --v;
37         G[u] |= TWOL(v);
38         G[v] |= TWOL(u);
39     }
40     BronK(0, TWOL(n)-1, 0, 0);
41     cout << r << ' ' << clique << '\n';
42 }
43 }

```

7 数据结构

7.1 树状数组

8 字符串

8.1 KMP

```

1 namespace KMP {
2     vector<int> getPrefixTable(string s) { // 求前缀表
3         int n = s.length();
4         vector<int> nxt(n, 0);
5         for (int i = 1; i < n; i++) {
6             int j = nxt[i - 1];
7             while (j > 0 && s[i] != s[j]) {
8                 j = nxt[j - 1];
9             }
10            if (s[i] == s[j]) j++;
11            nxt[i] = j;
12        }
13        return nxt;
14    }
15
16    vector<int> kmp(string s, string t) { // 返回所有匹配位置的集合
17        int n = s.length(), m = t.length();
18        vector<int> res;
19        vector<int> nxt = getPrefixTable(t);
20        for (int i = 0, j = 0; i < n; i++) {
21            while (j > 0 && j < m && s[i] != t[j]) {
22                j = nxt[j - 1];

```

```

23     }
24     if (s[i] == t[j]) j++;
25     if (j == m) {
26         res.push_back(i + 1 - m);
27         j = nxt[m - 1];
28     }
29 }
30 return res;
31 }
32 }

```

8.2 Z-Function

8.3 Manacher

```

1 namespace Manacher {
2     static constexpr int SIZE = 1e5 + 5; // 预设为原串长度
3     int len = 1; // manacher 预处理后字符串的长度
4     char stk[SIZE << 1]; // manacher 预处理字符串 需要2倍空间+1
5     void init(string s) { // 初始化stk
6         stk[0] = '*'; len = 1;
7         for (int i = 0; i < s.length(); ++i) {
8             stk[len++] = s[i];
9             stk[len++] = '*';
10        }
11    }
12    int manacher() { // 返回最长回文子串长度
13        vector<int> rad(len << 1); // 存储每个点作为对称中心可拓展的最大半径
14        int md = 0; // 最远回文串对称中心下标
15        for (int i = 1; i < len; ++i) {
16            int& r = rad[i] = 0;
17            if (i <= md + rad[md]) {
18                r = min(rad[2 * md - i], md + rad[md] - i);
19            }
20            while (i - r - 1 >= 0 && i + r + 1 < len &&
21                stk[i - r - 1] == stk[i + r + 1]) ++r;
22            if (i + r >= md + rad[md]) md = i;
23        }
24        int res = 0;
25        for (int i = 0; i < len; ++i) {
26            if (rad[i] > res) {
27                res = rad[i];
28            }
29        }

```

```

30     return res;
31 }
32 }

```

8.4 Trie

```

1 struct trie {
2     int cnt;
3     vector<vector<int>> nxt;
4     vector<bool> vis;
5     /* 初始化的时候size需要设置为字符串总长之和 26是字符集大小 */
6     trie(int size_ = 0) : cnt(0), vis(size_, false), nxt(size_, vector<int>
7         >(26, 0)) {}
8     void insert(string s) { // 插入字符串
9         int p = 0;
10        for (int i = 0; i < (int)s.length(); i++) {
11            int c = s[i] - 'a';
12            if (!nxt[p][c]) nxt[p][c] = ++cnt;
13            p = nxt[p][c];
14        }
15        vis[p] = true;
16    }
17    bool find(string s) { // 查找字符串
18        int p = 0;
19        for (int i = 0; i < (int)s.length(); i++) {
20            int c = s[i] - 'a';
21            if (!nxt[p][c]) return false;
22            p = nxt[p][c];
23        }
24        return vis[p];
25    }
};

```

8.5 01-Trie

```

1 template<typename T> struct xorTrie {
2     int HIGHBIT, cnt;
3     vector<vector<int>> nxt;
4     vector<bool> vis;
5     xorTrie(int n_ = 0, int highbit_ = 30) : HIGHBIT(highbit_), cnt(0) {
6         int size_ = upperBoundEstimate(n_);
7         nxt.resize(size_, vector<int>(2, 0));
8         vis.resize(size_, false);

```

```

9      }
10     int upperBoundEstimate(int n) { // 求内存上界
11         int hbit = log2(n);
12         return n * (HIGHBIT - hbit + 1) + (1 << (hbit + 1)) - 1;
13     }
14     void insert(T x) { // 插入
15         int p = 0;
16         for (int i = HIGHBIT; ~i; i--) {
17             int s = ((x >> i) & 1);
18             if (!nxt[p][s]) nxt[p][s] = ++cnt;
19             p = nxt[p][s];
20         }
21         vis[p] = true;
22     }
23     bool find(T x) { // 查询
24         int p = 0;
25         for (int i = HIGHBIT; ~i; i--) {
26             int s = ((x >> i) & 1);
27             if (!nxt[p][s]) return false;
28             p = nxt[p][s];
29         }
30         return vis[p];
31     }
32 };

```

9 计算几何

10 杂项

10.1 蔡勒公式