

图论

- 1 构造圆方树
- 2 最小树形图：朴素算法
- 3 blossom algorithm
- 4 euler_tour

计算几何

- 5 最小圆覆盖

数论

- 6 线性筛 & 杜教筛
- 7 类 Euclid 算法

网络流

- 8 dinic
- 9 费用流

字符串

- 10 后缀排序：DC3
- 11 AC 自动机
- 12 后缀排序：倍增算法
- 13 后缀排序：SA-IS
- 14 pam

数据结构

- 15 ntt
- 16 fft
- 17 lct
- 18 左偏树

最优化

- 19 单纯型

ADDITIONAL DOCUMENTS

1. 构造圆方树

G 用于存图，T 是构造的圆方树。只有一个点的点双没有添加方点。

```
1 static vector<int> G[NMAX + 10], T[NMAX + 10];
2 void bcc(int u, int f = 0) {
3     static stack<Pair> stk;
4     static bool marked[NMAX + 10];
5     static int in[NMAX + 10], low[NMAX + 10], cur;
6     in[u] = low[u] = ++cur;
7     for (int v : G[u]) {
8         if (v == f) f = 0; // 应对重边
9         else if (in[v]) low[u] = min(low[u], in[v]);
10        else {
11            stk.push(Pair(u, v)); // stk 内存储 DFS 树上的边
12            bcc(v, u);
13            low[u] = min(low[u], low[v]);
14            if (low[v] > in[u]) { // 割边 u - v
15                T[u].push_back(v);
16                T[v].push_back(u);
17                stk.pop();
18            } else if (low[v] >= in[u]) { // 可能有点双了
19                cnt++;
20                int linked = 0, p = n + cnt; // linked 点数, p 圆方树上的新方点
21                auto add = [p, &linked](int x) {
22                    if (!marked[x]) {
23                        marked[x] = true;
24                        T[p].push_back(x);
25                        T[x].push_back(p);
26                        linked++;
27                    }
28                };
29                while (!stk.empty()) {
```

```

29         Pair x = stk.top();
30         stk.pop();
31         add(x.u);
32         add(x.v);
33         if (x.u == u && x.v == v) break;
34     }
35     for (int v : T[p]) marked[v] = false;
36     if (linked == 0) cnt--; // 假点双
37 }

```

2. 最小树形图：朴素算法

给定一张 n 个点 m 条边的带权有向图，求以 r 为根的最小树形图上的边权总和，如果不存在输出 -1 。时间复杂度为 $O(nm)$ 。调用 `mdst(r)` 获得答案，调用前需清空 `id` 数组。如要求不定根的最小树形图，可以额外添加一个节点，向原图中的每个点连接一条边权为 ∞ 的边。

```

1 static int n, m, G[NMAX + 10], nxt[MMAX + 10];
2 static struct Edge { int u, v, w; } E[MMAX + 10], *in[NMAX + 10];
3 static int id[NMAX + 10], mark[NMAX + 10];
4 int find(int x) { return id[x] ? id[x] = find(id[x]) : x; }
5 int dfs(int x) {
6     mark[x] = 1; int ret = 1;
7     for (int i = G[x]; i; i = nxt[i])
8         if (!mark[E[i].v]) ret += dfs(E[i].v);
9     return ret;
10 }
11 inline int detect(int x) {
12     mark[x] = x;
13     for (int y = in[x]->u; in[y]; y = in[y]->u)
14         if (mark[y]) return mark[y] == x ? y : 0;
15     else mark[y] = x;
16     return 0;
17 }
18 int mdst(int r) {
19     if (dfs(r) < n) return -1;
20     int ret = 0;
21     while (true) {
22         memset(in, 0, sizeof(in));
23         memset(mark, 0, sizeof(mark));
24         for (auto *e = E + 1; e <= E + m; e++)
25             if (e->u != e->v && e->v != r && (!in[e->v] || e->w < in[e->v]->w))
26                 in[e->v] = e;
27         int p = 0, t = 0;
28         for (int x = 1; x <= n; x++, t |= p) if (!mark[x] && in[x]) {
29             if (!(p = detect(x))) continue;
30             ret += in[p]->w;
31             for (int x = in[p]->u; x != p; x = in[x]->u)
32                 id[find(x)] = p, ret += in[x]->w;
33             for (auto *e = E + 1; e <= E + m; e++) {
34                 int u = find(e->u), v = find(e->v);
35                 if (u != p && v == p) e->w -= in[e->v]->w;
36                 e->u = u; e->v = v;
37             }
38             if (!t) break;
39         }
40         for (int x = 1; x <= n; x++) if (in[x]) ret += in[x]->w;
41         return ret;
42 }

```

3. blossom algorithm

```

1 #include <stdio.h>
2 #include <algorithm>
3 #include <queue>
4 using namespace std;
5 const int maxn = 510;
6 struct node {
7     int v;
8     node *next;
9 } pool[maxn*maxn*2], *g[maxn];
10 int top, n, m, match[maxn];
11 int kind[maxn], pre[maxn], vis[maxn], c[maxn];
12 queue < int > q;
13 int f[maxn], ans;
14 void add ( int u , int v ) {node *tmp = &pool[++top]; tmp->v = v; tmp->next = g[u]; g[u] = tmp;}
15 int find ( int x ) {int i, t; for ( i = x ; c[i] > 0 ; i = c[i] ) ; while ( c[x] > 0 ) {t = c[x]; c[x] = i; x = t;} return i;}
16 void getpath ( int x , int tar , int root ) {
17     int t;
18     while ( x != root ) {t = match[x]; match[tar] = x; match[x] = tar; tar = t; x = pre[t];}
19     match[tar] = x; match[x] = tar;
20 }
21 int lca ( int u , int v , int root ) {
22     int i; for ( i = 1 ; i <= n ; i++ ) f[i] = 0;
23     while ( find ( u ) != root ) {u = find ( u ); f[u] = 1; if ( !match[u] ) break; u = pre[match[u]];}
24     f[root] = 1;

```

```

25     while ( find ( v ) != root ) {v = find ( v );if ( f[v] == 1 ) return v;if ( !match[v] ) break;v
= pre[match[v]];}
26     return root;
27 }
28 void blossom ( int x , int y , int l ) {
29     while ( find ( x ) != l ) {pre[x] = y;y = match[x];if ( kind[match[x]] == 2 ) {kind[match[x]] =
1;q.push ( match[x] );}if ( find ( x ) == x ) c[find(x)] = 1;if ( find ( match[x] ) == match[x] ) c[
find(match[x])] = 1;x = pre[y];}
30 }
31 void bfs ( int x ) {
32     int k , i , z;
33     for ( i = 1 ; i <= n ; i++ ) {
34         kind[i] = pre[i] = vis[i] = 0;c[i] = -1;
35     }
36     while ( q.size () ) q.pop ();q.push ( x );kind[x] = 1; vis[x] = 1;
37     while ( q.size () ) {
38         k = q.front (); q.pop ();
39         for ( node *j = g[k] ; j ; j = j -> next ) {
40             if ( !vis[j->v] ) {
41                 if ( !match[j->v] ) {
42                     getpath ( k , j -> v , x );
43                     return ;
44                 }
45                 else {
46                     kind[j->v] = 2;
47                     kind[match[j->v]] = 1;
48                     pre[j->v] = k;
49                     vis[j->v] = 1; vis[match[j->v]] = 1;
50                     q.push ( match[j->v] );
51                 }
52             }
53             else {
54                 if ( find ( k ) == find ( j -> v ) ) continue;
55                 if ( kind[find(j->v)] == 1 ) {
56                     z = lca ( k , j -> v , x );
57                     blossom ( k , j -> v , z );
58                     blossom ( j -> v , k , z );
59                 }
60             }
61         }
62     }
63 }
64 void work () {
65     int i , u , v;
66     scanf ( "%d%d" , &n , &m );
67     for ( i = 1 ; i <= m ; i++ ) {
68         scanf ( "%d%d" , &u , &v );
69         add ( u , v ); add ( v , u );
70     }
71     for ( i = 1 ; i <= n ; i++ ) {
72         if ( !match[i] ) bfs ( i );
73     }
74     for ( i = 1 ; i <= n ; i++ ) if ( match[i] ) ans++;
75     printf ( "%d\n" , ans / 2 );
76     for ( i = 1 ; i <= n ; i++ ) printf ( "%d%c" , match[i] , i==n?'\\n':' ' );
77 }
78 int main () {
79     work ();
80     return 0;
81 }

```

4. euler_tour

```

1 stack < int > s;
2 void dfs ( int i ) {
3     for ( node *j = g[i] ; j ; j = j -> next ) if ( !j -> taboo ) {
4         s.push ( j -> f );
5         j -> taboo = 1;
6         dfs ( j -> v );
7         ans[++index] = s.top ();
8         s.pop ();
9     }
10 }

```

5. 最小圆覆盖

```

1
2 #include <stdio.h>
3 #include <algorithm>
4 #include <math.h>
5
6 using namespace std;
7
8 const int maxn = 120000;
9 struct point {
10     double x , y;

```

```

11 } a[maxn] , c , tmp1 , tmp2;
12 int n;
13 double r;
14 double tmp;
15 double dis ( point x1 , point x2 ) {return sqrt ( (x1.x-x2.x)*(x1.x-x2.x) + (x1.y-x2.y)*(x1.y-x2.y)
);};
16 double det ( point x1 , point x2 , point x3 ) {return (x2.x-x1.x) * (x3.y-x1.y) - (x3.x-x1.x) * (x2.
y-x1.y);};
17 double abs ( double x ) {if ( x < 0 ) return -x;return x;}
18 point getcen ( point x1 , point x2 , point x3 ) {
19     double A , B , C , D , E , F;point ret;
20     if ( x1.x == x2.x ) A = 0.0, B = 1.0, C = (x1.y+x2.y)/2.0;
21     else {
22         A = 1.0/((x1.y-x2.y) / (x1.x-x2.x));B = 1.0;
23         C = -(x1.y+x2.y)/2.0 - A * (x1.x+x2.x)/2.0;
24     }
25     if ( x1.x == x3.x ) D = 0.0, E = 1.0, F = (x1.y+x3.y)/2.0;
26     else {
27         D = 1.0/((x1.y-x3.y) / (x1.x-x3.x));E = 1.0;
28         F = -(x1.y+x3.y)/2.0 - D * (x1.x+x3.x)/2.0;
29     }
30     ret.x = (B * F - C * E) / (A * E - B * D);
31     ret.y = (A * F - C * D) / (B * D - A * E);
32     return ret;
33 }
34 void work () {
35     int i , j , k;
36     srand(67890);
37     scanf ( "%d" , &n );
38     for ( i = 1 ; i <= n ; i++ ) scanf ( "%lf%lf" , &a[i].x , &a[i].y );
39     random_shuffle ( a + 1 , a + 1 + n );
40     if ( n == 2 ) {
41         printf ( "%.3lf\n" , dis ( a[1] , a[2] ) / 2.0 );
42         return ;
43     }
44     c.x = a[1].x;c.y = a[1].y;r = 0.0;
45     for ( i = 2 ; i <= n ; i++ ) {
46         if ( dis ( c , a[i] ) - r > 1e-9 ) {
47             c.x = a[i].x;c.y = a[i].y;r = 0.0;
48             for ( j = 1 ; j < i ; j++ ) {
49                 if ( dis ( c , a[j] ) - r > 1e-9 ) {
50                     c.x = (a[i].x + a[j].x) / 2.0;
51                     c.y = (a[i].y + a[j].y) / 2.0;
52                     r = dis ( a[i] , a[j] ) / 2.0;
53                     tmp = r; tmp1 = c;
54                     for ( k = 1 ; k <= j - 1 ; k++ ) {
55                         if ( dis ( tmp1 , a[k] ) - tmp > 1e-9 ) {
56                             if ( abs(det ( a[i] , a[j] , a[k] )) < 1e-9 ) continue;
57                             tmp2 = getcen ( a[i] , a[j] , a[k] );
58                             tmp = dis ( tmp2 , a[i] );
59                             tmp1 = tmp2;
60                         }
61                     }
62                     c = tmp1; r = tmp;
63                 }
64             }
65         }
66     }
67     printf ( "%.3lf\n" , r );
68 }
69 int main () {
70     work ();
71     return 0;
72 }

```

6. 线性筛 & 杜教筛

计算积性函数 $f(n)$ 的前缀和 $F(n) = \sum_{k=1}^n f(k)$: 先选定辅助函数 $g(n)$ 进行 Dirichlet 卷积, 得到递推公式:

$$F(n) = \frac{1}{g(1)} \left(\sum_{k=1}^n (f \times g)(k) - \sum_{k=2}^n g(k) F\left(\left\lfloor \frac{n}{k} \right\rfloor\right) \right)$$

对于 Euler 函数 $\varphi(n)$, 选定 $g(n) = 1$, 得:

$$\Phi(n) = \frac{n(n+1)}{2} - \sum_{k=2}^n \Phi\left(\left\lfloor \frac{n}{k} \right\rfloor\right)$$

对于 Mobius 函数 $\mu(n)$, 选定 $g(n) = 1$, 得:

$$M(n) = 1 - \sum_{k=2}^n M\left(\left\lfloor \frac{n}{k} \right\rfloor\right)$$

如果没有预处理, 时间复杂度为 $\Theta(n^{3/4})$, 空间复杂度为 $\Theta(\sqrt{n})$ 。如果预处理前 $\Theta(n^{2/3})$ 项前缀和, 则时空复杂度均变为 $\Theta(n^{2/3})$ 。下面的代码以 Euler 函数为例, 能够在 1s 内计算 10^{10} 内的数据。可以多次调用。

```

1 #define S 17000000 // for F(10^10)
2 static int pc, pr[S + 10];

```

```

3 static i64 phi[S + 10];
4 static unordered_map<i64, i64> dat;
5 inline void sub(i64 &a, i64 b) { a -= b; if (a < 0) a += MOD; }
6 inline i64 c2(i64 n) { n %= MOD; return n * (n + 1) % MOD * INV2 % MOD; }
7 i64 F(i64 n) { // 杜教筛
8     if (n <= S) return phi[n];
9     if (dat.count(n)) return dat[n];
10    i64 &r = dat[n] = c2(n);
11    for (i64 i = 2, l; i <= n; i = l + 1) {
12        i64 p = n / i;
13        l = n / p;
14        sub(r, (l - i + 1) * F(p) % MOD); // (l - i + 1) % MOD?
15    }
16    return r;
17 }
18 phi[1] = 1; // 线性筛
19 for (int i = 2; i <= S; i++) {
20     if (!phi[i]) {
21         pr[pc++] = i;
22         phi[i] = i - 1;
23     }
24     for (int j = 0; pr[j] * i <= S; j++) {
25         int p = pr[j];
26         if (i % p) phi[i * p] = phi[i] * (p - 1);
27         else {
28             phi[i * p] = phi[i] * p;
29             break;
30         }
31     }
32 }
33 for (int i = 2; i <= S; i++) add(phi[i], phi[i - 1]);

```

7. 类 Euclid 算法

类 Euclid 算法在模意义下计算：

$$\sum_{k=0}^n k^p \left\lfloor \frac{ak+b}{c} \right\rfloor^q$$

其中所有参数非负，在计算过程中始终保证 $K = p + q$ 不增， $a, c \geq 1$ 且 $b \geq 0$ 。需要 Bernoulli 数 ($B_1 = +1/2$) 来计算自然数幂前缀和 $S_p(x) = \sum_{k=1}^x k^p = \sum_{k=1}^{p+1} a_k^{(p)} x^k$ ，其中 $a_k^{(p)} = \frac{1}{p+1} \binom{p+1}{k} B_{p+1-k}$ 。代码中 has 为访问标记数组，每次使用前需清空，val 为记忆化使用的数组，qpow 是快速幂，S 是自然数幂前缀和，A 记录了 $a_k^{(p)}$ ，C 是组合数。时空复杂度为 $O(K^3 \log \max\{a, c\})$ 。

算法主要分为三个情况，其中 $a \geq c$ 和 $b \geq c$ 的情况比较简单。当 $a, b < c$ 时，用 $j = \lfloor (ak+b)/c \rfloor$ 进行代换，注意最终要转化为 $\lfloor (c(j-1) + c - b - 1)/a \rfloor < k \leq \lfloor (cj + c - b - 1)/a \rfloor$ ，再进行一次分部求和即可。注意处理 $k \leq n$ 这个条件。

```

1 i64 F(i64 n, i64 a, i64 b, i64 c, int p, int q, int d = 0) {
2     if (n < 0) return 0;
3     if (has[d][p][q]) return val[d][p][q];
4     has[d][p][q] = true;
5     i64 &ret = val[d][p][q] = 0; // 后面的 d 均加 1
6     if (!q) ret = S(n, p) + (!p); // 注意 p = 0 的边界情况
7     else if (!a) ret = qpow(b / c, q) * (S(n, p) + (!p)) % MOD;
8     else if (a >= c) {
9         i64 m = a / c, r = a % c, mp = 1;
10        for (int j = 0; j <= q; j++, mp = mp * m % MOD)
11            add(ret, C[q][j] * mp % MOD * F(n, r, b, c, p + j, q - j, d) % MOD);
12    } else if (b >= c) {
13        i64 m = b / c, r = b % c, mp = 1;
14        for (int j = 0; j <= q; j++, mp = mp * m % MOD)
15            add(ret, C[q][j] * mp % MOD * F(n, a, r, c, p, q - j, d) % MOD);
16    } else {
17        i64 m = (a * n + b) / c;
18        for (int k = 0; k < q; k++) {
19            i64 s = 0;
20            for (int i = 1; i <= p + 1; i++)
21                add(s, A[p][i] * F(m - 1, c, c - b - 1, a, k, i, d) % MOD);
22            add(ret, C[q][k] * s % MOD);
23        }
24        ret = (qpow(m, q) * S(n, p) - ret) % MOD;
25    } return ret;
26 }

```

8. dinic

```

1 void add ( int u , int v , int f ) {
2     node *tmp1 = &pool[++top] , *tmp2 = &pool[++top];
3     tmp1 -> v = v; tmp1 -> f = f; tmp1 -> next = g[u]; g[u] = tmp1; tmp1 -> rev = tmp2;
4     tmp2 -> v = u; tmp2 -> f = 0; tmp2 -> next = g[v]; g[v] = tmp2; tmp2 -> rev = tmp1;
5 }
6 bool makelevel () {
7     int i , k;
8     queue < int > q;
9     for ( i = 1 ; i <= 1 + n + n + 1 ; i++ ) level[i] = -1;
10    level[1] = 1; q.push ( 1 );
11    while ( q.size () != 0 ) {
12        k = q.front (); q.pop ();
13        for ( node *j = g[k] ; j ; j = j -> next )

```

```

14         if ( j -> f && level[j->v] == -1 ) {
15             level[j->v] = level[k] + 1;
16             q.push ( j -> v );
17             if ( j -> v == 1 + n + n + 1 ) return true;
18         }
19     }
20     return false;
21 }
22 int find ( int k , int key ) {
23     if ( k == 1 + n + n + 1 ) return key;
24     int i , s = 0;
25     for ( node *j = g[k] ; j ; j = j -> next )
26         if ( j -> f && level[j->v] == level[k] + 1 && s < key ) {
27             i = find ( j -> v , min ( key - s , j -> f ) );
28             j -> f -= i;
29             j -> rev -> f += i;
30             s += i;
31         }
32     if ( s == 0 ) level[k] = -1;
33     return s;
34 }
35 void dinic () {
36     int ans = 0;
37     while ( makelevel () == true ) ans += find ( 1 , 99999 );
38     //printf ( "%d\n" , ans );
39     if ( ans == sum ) printf ( "^_\n" );
40     else printf ( "T_T\n" );
41 }

```

9. 费用流

```

1 void add ( int u , int v , int f , int c ) {
2     node *tmp1 = &pool[++top] , *tmp2 = &pool[++top];
3     tmp1 -> v = v; tmp1 -> f = f; tmp1 -> c = c; tmp1 -> next = g[u]; g[u] = tmp1; tmp1 -> rev = tmp
4     2; tmp2 -> v = u; tmp2 -> f = 0; tmp2 -> c = -c; tmp2 -> next = g[v]; g[v] = tmp2; tmp2 -> rev = tm
5     p1;
6 }
7 bool spfa () {
8     int i , k;
9     queue < int > q;
10    for ( i = 1 ; i <= 1 + n*m*3 + 1 ; i++ ) dis[i] = 9999999 , f[i] = 0;
11    dis[1] = 0; f[1] = 1; q.push ( 1 );
12    while ( q.size () != 0 ) {
13        k = q.front (); q.pop (); f[k] = 0;
14        for ( node *j = g[k] ; j ; j = j -> next )
15            if ( j -> f && dis[j->v] > dis[k] + j -> c ) {
16                dis[j->v] = dis[k] + j -> c;
17                from[j->v] = j;
18                if ( f[j->v] == 0 ) q.push ( j -> v );
19                f[j->v] = 1;
20            }
21    }
22    if ( dis[1+n*m*3+1] != 9999999 ) return true;
23    return false;
24 }
25 int find () {
26     int i , f = 999999 , s = 0;
27     for ( i = 1+n*m*3+1 ; i != 1 ; i = from[i] -> rev -> v ) f = min ( f , from[i] -> f );
28     flow += f;
29     for ( i = 1+n*m*3+1 ; i != 1 ; i = from[i] -> rev -> v ) from[i] -> f -= f , from[i] -> rev -> f
30     += f;
31     return f * dis[1+n*m*3+1];
32 }
33 void dinic () {
34     int ans = 0;
35     while ( spfa () == true ) ans += find ();
36     //printf ( "%d\n" , flow );
37     if ( flow == sum && sum == sum1 ) printf ( "%d\n" , ans );
38     else printf ( "-1\n" );
39 }

```

10. 后缀排序：DC3

DC3 后缀排序算法，时空复杂度 $\Theta(n)$ 。字符串本体 s 数组、 sa 数组和 rk 数组都要求 3 倍空间。下标从 0 开始，字符串长度为 n ，字符集 Σ 为 $[0, m]$ 。partial_sum 需要标准头文件 `numeric`。

```

1 #define CH(i, n) i < n ? s[i] : 0
2 static int ch[NMAX + 10][3], seq[NMAX + 10];
3 static int arr[NMAX + 10], tmp[NMAX + 10], cnt[NMAX + 10];
4 inline bool cmp(int i, int j) {
5     return ch[i][0] == ch[j][0] && ch[i][1] == ch[j][1] && ch[i][2] == ch[j][2];
6 }
7 inline bool sufcmp(int *s, int *rk, int n, int i, int j) {
8     if (s[i] != s[j]) return s[i] < s[j];
9     if ((i + 1) % 3 && (j + 1) % 3) return rk[i + 1] < rk[j + 1];

```

```

10     if (s[i + 1] != s[j + 1]) return s[i + 1] < s[j + 1];
11     return rk[i + 2] < rk[j + 2];
12 }
13 void radix_sort(int n, int m, int K, bool init = true) {
14     if (init) for (int i = 0; i < n; i++) arr[i] = i;
15     int *a = arr, *b = tmp;
16     for (int k = 0; k < K; k++) {
17         memset(cnt, 0, sizeof(int) * (m + 1));
18         for (int i = 0; i < n; i++) cnt[ch[a[i]][k]]++;
19         partial_sum(cnt, cnt + m + 1, cnt);
20         for (int i = n - 1; i >= 0; i--) b[--cnt[ch[a[i]][k]]] = a[i];
21         swap(a, b);
22     }
23     if (a != arr) memcpy(arr, tmp, sizeof(int) * n);
24 }
25 void suffix_sort(int *s, int n, int m, int *sa, int *rk) {
26     s[n] = 0; n++;
27     int p = 0, q = 0;
28     for (int i = 1; i < n; i += 3, p++) for (int j = 0; j < 3; j++)
29         ch[p][2 - j] = CH(i + j, n);
30     for (int i = 2; i < n; i += 3, p++) for (int j = 0; j < 3; j++)
31         ch[p][2 - j] = CH(i + j, n);
32     radix_sort(p, m, 3);
33     for (int i = 0; i < p; i++) {
34         if (!q || (q && !cmp(arr[i - 1], arr[i]))) q++;
35         s[n + arr[i]] = q;
36     }
37     if (q < p) suffix_sort(s + n, p, q, sa + n, rk + n);
38     else {
39         for (int i = 0; i < p; i++) sa[n + s[n + i] - 1] = i;
40         for (int i = 0; i < p; i++) rk[n + sa[n + i]] = i + 1;
41     }
42     m = max(m, p);
43     p = q = 0;
44     for (int i = 1; i < n; i += 3, p++) rk[i] = rk[n + p];
45     for (int i = 2; i < n; i += 3, p++) rk[i] = rk[n + p];
46     for (int i = 0; i < n; i++) if (i % 3) seq[rk[i] - 1] = i;
47     for (int i = 0; i < n; i += 3, q++) {
48         ch[i][0] = i + 1 < n ? rk[i + 1] : 0;
49         ch[i][1] = s[i];
50         arr[q] = i;
51     }
52     radix_sort(q, m, 2, false);
53     for (int i = seq[0] == n - 1, j = arr[0] == n - 1, k = 0; i < p || j < q; k++) {
54         if (i == p) sa[k] = arr[j++];
55         else if (j == q) sa[k] = seq[i++];
56         else if (sufcmp(s, rk, n, seq[i], arr[j])) sa[k] = seq[i++];
57         else sa[k] = arr[j++];
58     }
59     for (int i = 0; i < n - 1; i++) rk[sa[i]] = i + 1;
60 }

```

11. AC 自动机

时间复杂度 $O(n + m + z + n|\Sigma|)$, n 是模板串总长度, m 是目标串长度, z 是总匹配次数, Σ 是字符集。如果想移掉 $n|\Sigma|$ 这一项, 需要使用哈希表。传入的字符串下标从 0 开始。

```

1 struct Node {
2     Node() : mark(false), suf(NULL), nxt(NULL) {
3         memset(ch, 0, sizeof(ch));
4     }
5     bool mark;
6     Node *suf, *nxt, *ch[SIGMA];
7 };
8 void insert(Node *x, char *s) {
9     for (int i = 0; s[i]; i++) {
10         int c = s[i] - 'a';
11         if (!x->ch[c]) x->ch[c] = new Node;
12         x = x->ch[c];
13     }
14     x->mark = true;
15 }
16 void build_automaton(Node *r) {
17     queue<Node *> q;
18     for (int c = 0; c < SIGMA; c++) {
19         if (!r->ch[c]) continue;
20         r->ch[c]->suf = r;
21         q.push(r->ch[c]);
22     }
23     while (!q.empty()) {
24         Node *x = q.front();
25         q.pop();
26         for (int c = 0; c < SIGMA; c++) {
27             Node *v = x->ch[c]; if (!v) continue;
28             Node *y = x->suf;
29             while (y != r && !y->ch[c]) y = y->suf;

```

```

30         if (y->ch[c]) y = y->ch[c];
31         v->suf = y;
32         if (y->mark) v->nxt = y;
33         else v->nxt = y->nxt;
34         q.push(v);
35     }}}
36 void search(Node *x, char *s) {
37     for (int i = 0; s[i]; i++) {
38         int c = s[i] - 'a';
39         while (x->suf && !x->ch[c]) x = x->suf;
40         if (x->ch[c]) x = x->ch[c];
41         if (x->mark) print(i + 1, x->data);
42         for (Node *y = x->nxt; y; y = y->nxt) print(i + 1, y->data);
43     }
}

```

12. 后缀排序：倍增算法

倍增法后缀排序，时间复杂度为 $\Theta(n \log n)$ 。suffix_sort 是本体，结果输出到 sa 数组和 rk 数组（排名数组）。参数 s 是字符串，下标从 0 开始，n 是字符串长度，m 是字符集大小（一般为 255，字符集为 $\Sigma = \{0, 1, 2, \dots, m\}$ ，0 是保留的 \$ 字符）。算法运行完毕后 sa 数组里面存的是从 0 开始的下标，rk 数组里面存的是从 1 开始的排名值。

另外附带一个线性求 lcp 数组的代码。lcp 数组下标从 1 开始，实际上只有在 2 到 n 范围内的才是有效值。参数意义与 suffix_sort 相同。

```

1 static int sa[NMAX + 10], rk[NMAX + 10], lcp[NMAX + 10];
2 void suffix_sort(const char *s, int n, int m) {
3     static int x[NMAX + 10], y[NMAX + 10], cnt[NMAX + 10], i;
4     for (i = 0; i < n; i++) cnt[s[i]]++;
5     for (i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
6     for (i = 0; i < n; i++) sa[--cnt[s[i]]] = i;
7     for (i = 1, m = 1, rk[sa[0]] = 1; i < n; i++) {
8         if (s[sa[i - 1]] != s[sa[i]]) m++;
9         rk[sa[i]] = m;
10    }
11    for (int l = 1; l < n; l <= 1) {
12        memset(cnt, 0, sizeof(int) * (m + 1));
13        for (i = 0; i < n; i++) cnt[y[i] = i + l < n ? rk[i + l] : 0]++;
14        for (i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
15        for (i = n - 1; i >= 0; i--) x[--cnt[y[i]]] = i;
16        memset(cnt, 0, sizeof(int) * (m + 1));
17        for (i = 0; i < n; i++) cnt[rk[i]]++;
18        for (i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
19        for (i = n - 1; i >= 0; i--) sa[--cnt[rk[x[i]]]] = x[i];
20        for (i = 1, m = 1, x[sa[0]] = 1; i < n; i++) {
21            if (rk[sa[i - 1]] != rk[sa[i]] || y[sa[i - 1]] != y[sa[i]]) m++;
22            x[sa[i]] = m;
23        }
24        memcpy(rk, x, sizeof(int) * n);
25    }
26 void compute_lcp(const char *s, int n) {
27     int j = 0, p;
28     for (int i = 0; i < n; i++, j = max(0, j - 1)) {
29         if (rk[i] == 1) {
30             j = 0;
31             continue;
32         }
33         p = sa[rk[i] - 2];
34         while (p + j < n && i + j < n && s[p + j] == s[i + j]) j++;
35         lcp[rk[i]] = j;
36    }
}

```

13. 后缀排序：SA-IS

SA-IS 后缀数组排序。字符串存在 str 中，下标从 1 开始，长度为 n，并且 str[n + 1] 为哨兵字符，编号为 1。后缀数组放在 sa 中，下标从 1 开始。时空复杂度为 $\Theta(n)$ 。其中使用了 vector<bool> 来优化缓存命中率。

```

1 #define rep(i, l, r) for (register int i = (l); i <= (r); ++i)
2 #define rrep(i, r, l) for (register int i = (r); i >= (l); --i)
3 #define PUTS(x) sa[cur[chr[x]]++] = x
4 #define PUTL(x) sa[cur[chr[x]]++] = x
5 #define LMS(x) (!type[x - 1] && type[x])
6 #define RESET memset(sa + 1, 0, sizeof(int) * (n + 1));
7     memcpy(cur + 1, cnt + 1, sizeof(int) * m);
8 #define INDUCE rep(i, 1, m) cur[i] = cnt[i - 1] + 1;
9     rep(i, 1, n + 1) if (sa[i] > 1 && !type[sa[i] - 1]) PUTL(sa[i] - 1);
10    memcpy(cur + 1, cnt + 1, sizeof(int) * m);
11    rrep(i, n + 1, 1) if (sa[i] > 1 && type[sa[i] - 1]) PUTS(sa[i] - 1);
12 void sais(int n, int m, int *str, int *sa) {
13     static int id[NMAX + 10];
14     vector<bool> type(n + 2);
15     type[n + 1] = true;
16     rrep(i, n, 1) type[i] = str[i] == str[i + 1] ? type[i + 1] : str[i] < str[i + 1];
17     int cnt[m + 1], cur[m + 1], idx = 1, y = 0, rt, lrt, *ns = str + n + 2, *nsa = sa + n + 2;
18     memset(cnt, 0, sizeof(int) * (m + 1));
19     rep(i, 1, n + 1) cnt[chr[i]]++;
20     rep(i, 1, m) cnt[i] += cnt[i - 1];

```



```

21 RESET rep(i, 2, n + 1) if (LMS(i)) PUTS(i); INDUCE
22 memset(id + 1, 0, sizeof(int) * n);
23 rep(i, 2, n + 1) if (LMS(sa[i])) {
24     register int x = sa[i];
25     for (rt = x + 1; !LMS(rt); rt++);
26     id[x] = y && rt + y == lrt + x && !memcmp(str + x, str + y, sizeof(int) * (rt - x + 1)) ? id
x : ++idx;
27     y = x, lrt = rt;
28 }
29 int len = 0, pos[(n >> 1) + 1];
30 rep(i, 1, n) if (id[i]) {
31     ns[++len] = id[i];
32     pos[len] = i;
33 }
34 ns[len + 1] = 1, pos[len + 1] = n + 1;
35 if (len == idx - 1) rep(i, 1, len + 1) nsa[ns[i]] = i;
36 else sais(len, idx, ns, nsa);
37 RESET rrep(i, len + 1, 1) PUTS(pos[nsa[i]]); INDUCE
38 }
39 static int str[NMAX * 3 + 10], sa[NMAX * 3 + 10];

```

14. pam

```

1
2 #include <stdio.h>
3 #include <algorithm>
4 #include <string.h>
5 using namespace std;
6 const int NN = 310000;
7 struct node {
8     int len, cnt, ch[30], fail;
9 } p[NN];
10 int top, n, last;
11 char z[NN];
12 long long ans;
13 void work () {
14     int i, tmp;
15     scanf ("%s", z + 1);
16     n = strlen (z + 1);
17     top = 2;
18     p[1].fail = 2; p[2].fail = 1;
19     p[1].len = 0; p[2].len = -1;
20     z[0] = '$';
21     last = 1;
22     for (i = 1; i <= n; i++) {
23         while (z[i] != z[i - p[last].len - 1]) last = p[last].fail;
24         if (!p[last].ch[z[i] - 'a' + 1]) {
25             p[last].ch[z[i] - 'a' + 1] = ++top;
26             p[top].len = p[last].len + 2;
27             tmp = p[last].fail;
28             while (z[i] != z[i - p[tmp].len - 1]) tmp = p[tmp].fail;
29             if (p[tmp].len > 1 && p[tmp].ch[z[i] - 'a' + 1]) p[tmp].fail = p[tmp].ch[z[i] - 'a' + 1];
30             else p[tmp].fail = 1;
31         }
32         last = p[last].ch[z[i] - 'a' + 1];
33         p[last].cnt++;
34     }
35     for (i = top; i >= 1; i--) p[p[i].fail].cnt += p[i].cnt;
36     for (i = 1; i <= top; i++) {
37         //printf ("%d %d\n", p[i].len, p[i].cnt);
38         ans = max (ans, (long long)p[i].len * p[i].cnt);
39     }
40     printf ("%lld\n", ans);
41 }
42 int main () {
43     work ();
44     return 0;
45 }

```

15. ntt

```

1 #include <stdio.h>
2 #include <algorithm>
3 using namespace std;
4 const long long maxn = 120000;
5 const long long mod = 998244353;
6 const long long omega = 3;
7 long long a[maxn * 4], b[maxn * 4], c[maxn * 4], d[maxn * 4];
8 long long n, m, N, in;
9 long long pow (long long f, long long x) {long long s = 1; while (x) {if (x % 2) s = (s * f) % m; f = (f * f) % mod; x >>= 1;} return s;}
10 long long inv (long long x) {return pow (x, mod - 2);}
11 long long rev (long long x) {long long i, y; i = 1; y = 0; while (i < N) {y = y * 2 + (x % 2); i <<= 1; x >>= 1;} return y;}
12 void br (long long *x) {long long i; for (i = 0; i < N; i++) d[rev(i)] = x[i]; for (i = 0; i < N; i++) x[i] = d[i];}

```

```

13 void FFT ( Long Long *x , Long Long f ) {
14     Long Long i , j , s , k;
15     Long Long w , wm , u , t;
16     br ( x );
17     for ( s = 2 ; s <= N ; s *= 2 ) {
18         k = s / 2;
19         wm = pow ( omega , (mod-1) / s );
20         if ( f == -1 ) wm = inv ( wm );
21         for ( i = 0 ; i < N ; i += s ) {
22             w = 1;
23             for ( j = 1 ; j <= k ; j++ ) {
24                 u = x[i+j-1]; t = (x[i+j-1+k]*w) % mod;
25                 x[i+j-1] = (u + t) % mod;
26                 x[i+j-1+k] = (u - t + mod) % mod;
27                 w = (w*wm) % mod;
28             }
29         }
30     }
31     if ( f == -1 ) for ( i = 0 ; i < N ; i++ ) x[i] = (x[i] * in) % mod;
32 }
33 void work () {
34     Long Long i;
35     scanf ( "%lld%lld" , &n , &m );
36     N = 1;
37     while ( N < n + m + 2 ) N = N * 2;
38     for ( i = 0 ; i <= n ; i++ ) scanf ( "%lld" , &a[i] );
39     for ( i = 0 ; i <= m ; i++ ) scanf ( "%lld" , &b[i] );
40     in = inv ( N );
41     FFT ( a , 1 ); FFT ( b , 1 );
42     for ( i = 0 ; i < N ; i++ ) c[i] = (a[i]*b[i]) % mod;
43     FFT ( c , -1 );
44     for ( i = 0 ; i <= n + m ; i++ ) printf ( "%lld%c" , c[i] , i==n+m?'\\n':' ' );
45 }
46 int main () {
47     work ();
48     return 0;
49 }

```

16. fft

```

1 #include <stdio.h>
2 #include <algorithm>
3 #include <math.h>
4 using namespace std;
5 const int maxn = 120000;
6 const double pi = acos(-1);
7 struct complex {
8     double r , i;
9 } a[maxn*4] , b[maxn*4] , c[maxn*4] , d[maxn*4];
10 complex operator + ( complex x1 , complex x2 ) {complex y;y.r = x1.r + x2.r;y.i = x1.i + x2.i;return y;}
11 complex operator - ( complex x1 , complex x2 ) {complex y;y.r = x1.r - x2.r;y.i = x1.i - x2.i;return y;}
12 complex operator * ( complex x1 , complex x2 ) {complex y;y.r = x1.r * x2.r - x1.i * x2.i;y.i = x1.r * x2.i + x1.i * x2.r;return y;}
13 int n , m , N;
14 int rev ( int x ) {int i , y;i = 1; y = 0;while ( i < N ) {y = y * 2 + (x%2);x >>= 1; i <<= 1;}return y;}
15 void br ( complex *x ) {int i;for ( i = 0 ; i < N ; i++ ) d[rev(i)] = x[i];for ( i = 0 ; i < N ; i++ ) x[i] = d[i];}
16 void FFT ( complex *x , int f ) {
17     int i , j , s , k;
18     complex w , wm , u , t;
19     br ( x );
20     for ( s = 2 ; s <= N ; s *= 2 ) {
21         k = s / 2;
22         wm.r = cos(2*pi/s); wm.i = sin(2*pi/s) * f;
23         for ( i = 0 ; i < N ; i += s ) {
24             w.r = 1.0; w.i = 0.0;
25             for ( j = 1 ; j <= k ; j++ ) {
26                 u = x[i+j-1]; t = x[i+j-1+k] * w;
27                 x[i+j-1] = u + t;
28                 x[i+j-1+k] = u - t;
29                 w = w * wm;
30             }
31         }
32     }
33     if ( f == -1 ) for ( i = 0 ; i < N ; i++ ) x[i].r = x[i].r / N;
34 }
35 void work () {
36     int i;
37     scanf ( "%d%d" , &n , &m );
38     N = 1;
39     while ( N < n + m + 2 ) N = N * 2;
40     for ( i = 0 ; i <= n ; i++ ) scanf ( "%lf" , &a[i].r );
41     for ( i = 0 ; i <= m ; i++ ) scanf ( "%lf" , &b[i].r );

```

```

42     FFT ( a , 1 ); FFT ( b , 1 );
43     for ( i = 0 ; i < N ; i++ ) c[i] = a[i] * b[i];
44     FFT ( c , -1 );
45     for ( i = 0 ; i <= n + m ; i++ ) printf ( "%d%c" , int (c[i].r + 0.5) , i==n+m?'\\n':' ' );
46 }
47 int main () {
48     work ();
49     return 0;
50 }

```

17. lct

```

1 struct node {
2     Long Long x;
3     Long Long lm , lp , rev;
4     Long Long s , siz;
5     Long Long ch[4] , fa;
6 } p[maxn];
7 void cut ( Long Long x , Long Long kind ) {
8     p[p[x].ch[kind]].fa *= -1;
9     p[x].ch[kind] = 0;
10    update ( x );
11 }
12 void down ( Long Long x ) {
13     if ( p[x].fa > 0 ) down ( p[x].fa );
14     pushdown ( x );
15 }
16 void rotate ( Long Long x , Long Long kind ) {
17     Long Long y = p[x].fa;
18     if ( p[y].fa > 0 ) p[p[y].fa].ch[y==p[p[y].fa].ch[1]] = x;
19     p[x].fa = p[y].fa;
20     if ( p[x].ch[kind^1] ) p[p[x].ch[kind^1]].fa = y;
21     p[y].ch[kind] = p[x].ch[kind^1];
22     p[y].fa = x;
23     p[x].ch[kind^1] = y;
24     update ( y ); update ( x );
25 }
26 void splay ( Long Long x ) {
27     down ( x );
28     for ( ; p[x].fa > 0 ; rotate ( x , x==p[p[x].fa].ch[1] ) )
29         if ( p[p[x].fa].fa > 0 && (x==p[p[x].fa].ch[1]) == (p[p[x].fa].fa==p[p[p[x].fa].fa].ch[1]) )
30             rotate ( p[x].fa , x==p[p[x].fa].ch[1] );
31 }
32 void access ( Long Long x ) {
33     splay ( x );
34     cut ( x , 1 );
35     for ( ; p[x].fa != 0 ; ) {
36         splay ( -p[x].fa );
37         cut ( -p[x].fa , 1 );
38         p[-p[x].fa].ch[1] = x;
39         update ( -p[x].fa );
40         p[x].fa *= -1;
41         splay ( x );
42     }
43 }
44 void makeroot ( Long Long x ) {
45     access ( x );
46     p[x].rev ^= 1;
47     swap ( p[x].ch[0] , p[x].ch[1] );
48 }
49 void link ( Long Long x , Long Long y ) {
50     makeroot ( y );
51     p[y].fa = -x;
52 }

```

18. 左偏树

核心操作split和merge，merge时候让小的当堆顶，继续合并右子树和另外一棵树，之后维护左偏性质。

```

1 struct node {
2     int x , i , dist;
3     node *ll , *rr;
4 } pool[maxn] , *t[maxn];
5 int n , m;
6 int a[maxn];
7 int c[maxn] , f[maxn];
8 int getdist ( node *id ) {
9     if ( id == NULL ) return -1;
10    return id -> dist;
11 }
12 node *merge ( node *id1 , node *id2 ) {
13     if ( id1 == NULL ) return id2;
14     if ( id2 == NULL ) return id1;
15     if ( id1 -> x > id2 -> x ) swap ( id1 , id2 );
16     id1 -> rr = merge ( id1 -> rr , id2 );
17     if ( getdist ( id1 -> ll ) < getdist ( id1 -> rr ) ) swap ( id1 -> ll , id1 -> rr );
18     id1 -> dist = getdist ( id1 -> rr ) + 1;

```

```

19     return id1;
20 }
21 int find ( int x ) {
22     int i , t;
23     for ( i = x ; c[i] > 0 ; i = c[i] ) ;
24     while ( x != i ) {
25         t = c[x];
26         c[x] = i;
27         x = t;
28     }
29     return i;
30 }
31 void Union ( int x , int y ) {
32     t[x] = merge ( t[x] , t[y] );
33     c[x] += c[y];
34     c[y] = x;
35 }

```

19. 单纯型

```

1 #define EPS 1e-10
2 #define INF 1e100
3
4 class Simplex {
5 public:
6     void initialize() {
7         scanf("%d%d%d", &n, &m, &t);
8
9         memset(A, 0, sizeof(A));
10        for (int i = 1; i <= n; i++) {
11            idx[i] = i;
12            scanf("%Lf", A[0] + i);
13        }
14
15        for (int i = 1; i <= m; i++) {
16            idy[i] = n + i;
17            for (int j = 1; j <= n; j++) {
18                scanf("%Lf", A[i] + j);
19                A[i][j] *= -1;
20            }
21
22            scanf("%Lf", A[i]);
23        }
24    }
25
26    void solve() {
27        srand(time(0));
28
29        while (true) {
30            int x = 0, y = 0;
31            for (int i = 1; i <= m; i++) {
32                if (A[i][0] < -EPS && (!y || (rand() & 1)))
33                    y = i;
34            }
35
36            if (!y)
37                break;
38
39            for (int i = 1; i <= n; i++) {
40                if (A[y][i] > EPS && (!x || (rand() & 1)))
41                    x = i;
42            }
43
44            if (!x) {
45                puts("Infeasible");
46                return;
47            }
48
49            pivot(x, y);
50        }
51
52        while (true) {
53            double k = INF;
54            int x, y;
55            for (x = 1; x <= n; x++) {
56                if (A[0][x] > EPS)
57                    break;
58            }
59
60            if (x > n)
61                break;
62
63            for (int i = 1; i <= m; i++) {
64                double d = A[i][x] > -EPS ? INF : -A[i][0] / A[i][x];
65                if (d < k) {
66                    k = d;

```

```

67         y = i;
68     }
69 }
70
71     if (k >= INF) {
72         puts("Unbounded");
73         return;
74     }
75
76     pivot(x, y);
77 }
78
79 printf("%.10Lf\n", A[0][0]);
80
81 if (t) {
82     static double ans[NMAX + 10];
83     for (int i = 1; i <= m; i++) {
84         if (idy[i] <= n)
85             ans[idy[i]] = A[i][0];
86     }
87
88     for (int i = 1; i <= n; i++) {
89         printf("%.10Lf ", ans[i]);
90     }
91     printf("\n");
92 }
93 }
94
95 private:
96 void pivot(int x, int y) {
97     swap(idx[x], idy[y]);
98     double r = -A[y][x];
99     A[y][x] = -1;
100    for (int i = 0; i <= n; i++) {
101        A[y][i] /= r;
102    }
103
104    for (int i = 0; i <= m; i++) {
105        if (i == y)
106            continue;
107
108        r = A[i][x];
109        A[i][x] = 0;
110        for (int j = 0; j <= n; j++) {
111            A[i][j] += r * A[y][j];
112        }
113    }
114 }
115
116 int n, m, t;
117 double A[NMAX + 10][NMAX + 10];
118 int idx[NMAX + 10], idy[NMAX + 10];
119 };

```

◦ [lmj/treehash.md](#):

◦ [lmj/matrix_tree_theorem.md](#):

K =度数矩阵-邻接矩阵, K 的任意代数余子式(一般删最后一行一列, 取正号)即为生成树数量。

◦ [lmj/virtual_tree.md](#):

把需要的点按照dfs序排序, 把相邻的lca求出来, 塞进去重新排序, 之后按照顺序维护当前的链, 如果不是链就pop当前的点, 在虚树上面加边。

◦ [lmj/dominator_tree.md](#):

◦ [lmj/sam.md](#):

◦ [lmj/cdq.md](#):

◦ [lmj/tree_divide_and_conquer\(edge_and_node\).md](#):

◦ [lmj/number_theory.md](#):

反演/筛

◦ [lmj/bounded_flow.md](#):

无源汇可行流

建模方法:

首先建立一个源ss和一个汇tt, 一般称为附加源和附加汇。

对于图中的每条弧, 假设它容量上界为c, 下界b, 那么把这条边拆为三条只有上界的弧。

一条为, 容量为b;

一条为, 容量为b;

一条为, 容量为c-b。

其中前两条弧一般称为附加弧。

然后对这张图跑最大流，以ss为源，以tt为汇，如果所有的附加弧都满流，则原图有可行流。

这时，每条非附加弧的流量加上它的容量下界，就是原图中这条弧应该有的流量。

理解方法：

对于原图中的每条弧，我们把c-b

称为它的自由流量，意思就是只要它流满了下界，这些流多少都没问题。

既然如此，对于每条弧，我们强制给v提供b单位的流量，并且强制从u那里拿走b单位的流量，这一步对应着两条附加弧。

如果这一系列强制操作能完成的话，也就是有一组可行流了。

注意：这张图的最大流只是对应着原图的一组可行流，而不是原图的最大或最小流。

有源汇可行流

建模方法：

建立弧，容量下界为0，上界为 ∞ 。

然后对这个新图（实际上只是比原图多了一条边）按照无源汇可行流的方法建模，如果所有附加弧满流，则存在可行流。

求原图中每条边对应的实际流量的方法，同无源汇可行流，只是忽略掉弧

就好。

而且这时候弧的流量就是原图的总流量。

理解方法：

有源汇相比无源汇的不同就在于，源和汇是不满足流量平衡的，那么连接

之后，源和汇也满足了流量平衡，就可以直接按照无源汇的方式建模。

注意：这张图的最大流只是对应着原图的一组可行流，而不是原图的最大或最小流。

有源汇最大流

建模方法：

首先按照有源汇可行流的方法建模，如果不存在可行流，更别提什么最大流了。

如果存在可行流，那么在运行过有源汇可行流的图上（就是已经存在流量的那张图，流量不要清零），跑一遍从s到t的最大流（这里的s和t是原图的源和汇，不是附加源和附加汇），就是原图的最大流。

理解方法：

为什么要在那个已经有了流量的图上跑最大流？因为那张图保证了每条弧的容量下界，在这张图上跑最大流，实际上就是在容量下界全部满足的前提下尽量多得获得“自由流量”。

注意，在这张已经存在流量的图上，弧也是存在流量的，千万不要忽略这条弧。因为它的相反弧的流量为的流量的相反数，且的容量为0，所以这部分的流量也是会被算上的。

有源汇最小流

有源汇最小流的常见建模方法比较多，我就只说我常用的一种。

建模方法：

首先按照有源汇可行流的方法建模，但是不要建立这条弧。

然后在这个图上，跑从附加源ss到附加汇tt的最大流。

这时候再添加弧，下界为0，上界为 ∞ 。

在现在的这张图上，从ss到tt的最大流，就是原图的最小流。

理解方法：

我们前面提到过，有源汇可行流的流量只是对应一组可行流，并不是最大或者最小流。

并且在跑完有源汇可行流之后，弧的流量就是原图的流量。

从这个角度入手，我们想让弧的流量尽量小，就要尽量多的消耗掉那些“本来不需要经过”的流量。

于是我们在添加之前，跑一遍从ss到tt的最大流，就能尽量多的消耗那些流量啦QwQ。

<https://www.cnblogs.com/mlystdcall/p/6734852.html>

◦ 1mj/Mo's_algorithm.md:

带修莫队：把时间当成一维，排序时左右端点的块和时间一起排序，模拟时间。

树上莫队：按照欧拉序，如果询问x,y,若lca(x,y)=x, 则查询st[x]到st[y], 否则ed[x],st[y], 再加上lca, 出现两次的点不算。

◦ 1mj/idea.md:

启发式合并

离线

hash

数据结构上跑图论算法