

图论

1. 边双联通 tarjan 2. 点双联通 tarjan
3. 有向图强联通 tarjan 4. 倍增lca 5. 仙人掌 DP
6. 构造圆方树 7. 最小树形图: 朴素算法
8. blossom algorithm 9. euler_tour
10. block_forest_data_structure

计算几何

11. 最小圆覆盖

数论

12. 线性筛 & 杜教筛 13. 类 Euclid 算法
14. 幂级数前缀和 15. continued_fraction
16. min_25_sieve 17. Pohlig_Hellman

网络流

18. dinic 19. 费用流

字符串

20. manacher 21. SA_sll 22. KMP 23. PAM_sll
24. 回文自动机 25. 后缀排序: DC3 26. AC 自动机
27. 后缀树 28. 后缀排序: 倍增算法
29. 后缀排序: SA-IS 30. pam

数据结构

31. 权值splay 32. 序列splay 33. ntt 34. fft 35. lct
36. 左偏树

杂项

37. 三分_上凸函数 38. 线性空间求交 39. 单纯型

其它文档

1. 边双联通 tarjan

```

1 const int N = 5010; // 3352只用1010即可
2 struct node{
3     int v,w,id;
4     node(int v = 0,int w = 0,int id = 0):v(v),w(w),id(id){};
5 };
6 vector<node>G[N];
7 int pre[N];
8 int low[N];
9 int dfs_num;int ans ;int n,m;
10 void init(){
11     mem(pre,0); mem(low,0);
12     for(int i=0;i<=n;i++) G[i].clear();
13     dfs_num = 0;ans = INF;
14 }
15 int dfs(int u,int fa){
16     low[u] = pre[u] = ++dfs_num;
17     for(int i=0;i<G[u].size();i++){
18         int v = G[u][i].v;
19         int id = G[u][i].id;
20         if(id == fa) continue;
21         if(!pre[v]){
22             dfs(v,id);//注意这里 第二个参数是 id
23             low[u] = min(low[u],low[v]);//用后代的low更新
24             当前的
25         }
26         else
27             low[u] = min(low[u],pre[v]);//利用后代v的反向
28             边更新low
29 }
30 int main(){
31     int t;
32     while(scanf("%d%d",&n,&m)!=EOF&& (n || m)){
33         int a,b,c;

```

```

32     init();
33     for(int i=1;i<=m;i++){
34         scanf("%d%d",&a,&b);
35         G[a].push_back(node(b,0,i));
36         G[b].push_back(node(a,0,i));
37     }
38     for(int i=1;i<=n;i++){
39         if(!pre[i])
40             dfs(i,0);
41         //cout<<i<<endl;
42     }
43     int degree[N];mem(degree,0);
44     for(int i=1;i<=n;i++){
45         for(int j=0;j<G[i].size();j++){
46             int v = G[i][j].v;
47             if(low[i] != low[v]){
48                 degree[low[v]]++; degree[low[i]]++;
49             }
50         }
51         int l = 0;
52         for(int i=1;i<=dfs_num;i++)
53             if(degree[i] == 2)
54                 l++;
55         printf("%d\n", (l+1)/2);
56     }
57     return 0;

```

2. 点双联通 tarjan

```

1 void tarjan(int u, int fa) {
2     pre[u] = low[u] = ++dfs_clock;
3     for (int i = 0; i < (int)G[u].size(); i++) {
4         int v = G[u][i];
5         if (!pre[v]) {
6             S.push(Edge(u, v));
7             tarjan(v, u);
8             low[u] = min(pre[v], low[u]);
9             if (low[v] >= pre[u]) {
10                 bcc_cnt++;
11                 bcc[bcc_cnt].clear();
12                 for(;;) {
13                     Edge x = S.top(); S.pop();
14                     if (bccno[x.u] != bcc_cnt) {
15                         bcc[bcc_cnt].push_back(x.u);
16                         bccno[x.u] = bcc_cnt;
17                     }
18                     if (bccno[x.v] != bcc_cnt) {
19                         bcc[bcc_cnt].push_back(x.v);
20                         bccno[x.v] = bcc_cnt;
21                     }
22                 }
23                 if (x.u == u && x.v == v) break;
24             }
25             else if (pre[v] < pre[u] && v != fa) {
26                 S.push(Edge(u, v));
27                 low[u] = min(low[u], pre[v]);
28             }
29         }
30     }
31 }

```

3. 有向图强联通 tarjan

```

1 int n,m;
2 int head[N],pos;
3 struct edge{int to,next;}e[N<<1];
4 void add(int a,int b)
5 {pos++;e[pos].to=b,e[pos].next=head[a],head[a]=pos;}
6 int dfn[N],low[N],SCC;
7 bool in[N];
8 int st[N],top,T;
9 vector<int>G[N];
10 void tarjan(int u) {
11     st[++top]=u;in[u]=1;
12     dfn[u]=low[u]=++T;
13     for(int i=head[u];i;i=e[i].next) {
14         int v=e[i].to;
15         if(!dfn[v]) {
16             tarjan(v);
17             low[u]=min(low[u],low[v]);
18         }
19         else if(in[v])low[u]=min(low[u],dfn[v]);
20     }
21     if(low[u]==dfn[u]) {
22         int v;
23         ++SCC;
24         do {
25             v=st[top--];
26             in[v]=false;

```

```

27     G[SCC].push_back(v);
28     }while(v!=u);
29 }
30 int main() {
31     scanf("%d%d",&n,&m);
32     for(int i=1;i<=m;i++) {
33         int x,y;
34         scanf("%d%d",&x,&y);
35         add(x,y);
36     }
37     for(int i=1;i<=n;i++)if(!dfn[i])tarjan(i);
38 }

```

4. 倍增lca

```

1 int lca(int x,int y) {
2     if(deep[x]<deep[y])swap(x,y);
3     int t=deep[x]-deep[y];
4     for(int i=0;bin[i]<=t;i++)
5         if(t&bin[i])x=fa[x][i];
6     for(int i=16;i>=0;i--)
7         if(fa[x][i]!=fa[y][i])
8             x=fa[x][i],y=fa[y][i];
9     if(x==y)return x;
10    return fa[x][0];
11 }

```

5. 仙人掌 DP

重复使用时，只需清空 dfn、fa 和 now。每次扫出的环按一定顺序存放在 a 数组中，a[1] 是环的根。

```

1 int dfn[NMAX + 10], low[NMAX + 10], now, cnt;
2 int ed[NMAX + 10], fa[NMAX + 10], a[NMAX + 10];
3 void dfs(int x) {
4     dfn[x] = low[x] = ++now;
5     for (int v : G[x]) if (v != fa[x]) {
6         if (dfn[v]) {
7             ed[v] = x, low[x] = min(low[x], dfn[v]);
8             continue;
9         } fa[v] = x;
10        dfs(v);
11        if (low[v] > dfn[x]) ; // 割边
12        else if (low[v] == dfn[x]) {
13            a[1] = x;
14            for (cnt = 1, v = ed[x]; v != x; v = fa[v])
15                a[++cnt] = v;
16            // 环 a[1]...a[cnt]
17            else low[x] = min(low[x], low[v]);
18        }
19    }

```

6. 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                };

```

```

28         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     }

```

7. 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];
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     }

```

8. blossom algorithm

```

1 const int maxn = 510;
2 struct node {
3     int v;
4     node *next;
5 } pool[maxn*maxn*2], *g[maxn];
6 int top, n, m, match[maxn];
7 int kind[maxn], pre[maxn], vis[maxn], c[maxn];
8 queue<int> q;
9 int f[maxn], ans;
10 void add(int u, int v) { node *tmp = &pool[top++]; tmp->v = v; tmp->next = g[u]; g[u] = tmp; }
11 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;}
12 void getpath ( int x , int tar , int root ) {
13     int t;
14     while ( x != root ) {t = match[x];match[tar] =
x;match[x] = tar;tar = t;x = pre[t];}
15     match[tar] = x;match[x] = tar;
16 }
17 int lca ( int u , int v , int root ) {
18     int i;for ( i = 1 ; i <= n ; i++ ) f[i] = 0;
19     while ( find ( u ) != root ) {u = find ( u );f
[u] = 1;if ( !match[u] ) break;u = pre[match[u]]
};}
20     f[root] = 1;
21     while ( find ( v ) != root ) {v = find ( v );i
f ( f[v] == 1 ) return v;if ( !match[v] ) break;
v = pre[match[v]];}
22     return root;
23 }
24 void blossom ( int x , int y , int l ) {
25     while ( find ( x ) != l ) {pre[x] = y;y = matc
h[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];}
26 }
27 void bfs ( int x ) {
28     int k , i , z;
29     for ( i = 1 ; i <= n ; i++ ) {
30         kind[i] = pre[i] = vis[i] = 0;c[i] = -1;
31     }
32     while ( q.size () ) q.pop ();q.push ( x );kind
[x] = 1; vis[x] = 1;
33     while ( q.size () ) {
34         k = q.front (); q.pop ();
35         for ( node *j = g[k] ; j ; j = j -> next ) {
36             if ( !vis[j->v] ) {
37                 if ( !match[j->v] ) {
38                     getpath ( k , j -> v , x );
39                     return ;
40                 }
41             } else {
42                 kind[j->v] = 2;
43                 kind[match[j->v]] = 1;
44                 pre[j->v] = k;
45                 vis[j->v] = 1; vis[match[j->v]] = 1;
46                 q.push ( match[j->v] );
47             }
48         }
49         if ( find ( k ) == find ( j -> v ) ) con
tinue;
50         if ( kind[find(j->v)] == 1 ) {
51             z = lca ( k , j -> v , x );
52             blossom ( k , j -> v , z );
53             blossom ( j -> v , k , z );
54 }}}}
55 void work () {
56     int i , u , v;
57     scanf ( "%d%d" , &n , &m );
58     for ( i = 1 ; i <= m ; i++ ) {
59         scanf ( "%d%d" , &u , &v );
60         add ( u , v ); add ( v , u );
61     }
62     for ( i = 1 ; i <= n ; i++ ) {
63         if ( !match[i] ) bfs ( i );
64     }
65     for ( i = 1 ; i <= n ; i++ ) if ( match[i] ) a
ns++;
66     printf ( "%d\n" , ans / 2 );
67     for ( i = 1 ; i <= n ; i++ ) printf ( "%d%c" ,
match[i] , i==n?'\\n':' ' );
68 }

```

9. 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 }

```

10. block_forest_data_structure

又叫圆方树

这个代码用来构造仙人掌的圆方树，两个点一条边的双联通分量不会被处理为圆点 + 方点，而是两个圆点直接相连，kind = 0 为圆点。tot 是圆点 + 方点的数量。注意数组大小要开两倍来维护方点。

gt 是造好的圆方树，如果还是从 1 号点开始遍历树的话，那么方点的边表中，就是按照 dfn 顺序的那些点，也就是按照环的顺序排序的，开头是与 1 号点最近的点，可以方便地处理环。

```

1 struct node {
2     int v , u ; node *next;
3 } pooln[maxn*4] , *gn[maxn];
4 struct tree {
5     int v ; tree *next;
6 } poolt[maxn*4] , *gt[maxn*2];
7 int topt , topn;
8 int n , m , tot;
9 int kind[maxn*2] , dfn[maxn] , low[maxn] , index
;
10 stack <node*> st;
11 void add ( int u , int v ) {
12     node *tmp = &pooln[++topn];
13     tmp -> v = v ; tmp -> u = u ; tmp -> next = gn[u
]; gn[u] = tmp;
14 }
15 void addt ( int u , int v ) {
16     tree *tmp = &poolt[++topt];
17     tmp -> v = v ; tmp -> next = gt[u]; gt[u] = tmp
;
18 }
19 void tarjan ( int i , int from ) {
20     dfn[i] = low[i] = ++index;
21     for ( node *j = gn[i] ; j ; j = j -> next ) if
( j -> v != from ) {
22         if ( !dfn[j->v] || dfn[i] > dfn[j->v] ) st.p
ush(j);
23         if ( !dfn[j->v] ) {
24             tarjan ( j -> v , i );
25             low[i] = min ( low[i] , low[j->v] );
26             if ( low[j->v] >= dfn[i] ) {
27                 if ( st.top() == j ) {
28                     addt ( i , j -> v , j -> prob );
29                     addt ( j -> v , i , j -> prob );
30                     st.pop();
31                 } else {
32                     tot++;
33                     kind[tot] = 1;
34                     while ( st.top() != j ) {
35                         node *k = st.top ();
36                         st.pop();
37                         addt ( tot , k -> u , k -> prob );
38                         addt ( k -> u , tot , k -> prob );
39                     }
40                     addt ( tot , i , j -> prob );
41                     addt ( i , tot , j -> prob );
42                     st.pop();
43                 }
44                 else low[i] = min ( low[i] , dfn[j->v] );
45 }
46 void work () {
47     int i , u , v , a , b;
48     scanf ( "%d%d" , &n , &m );
49     for ( i = 1 ; i <= m ; i++ ) {
50         scanf ( "%d%d%d%d" , &u , &v , &a , &b );
51         add ( u , v );
52         add ( v , u );
53     }
54     tot = n;
55     for ( i = 1 ; i <= n ; i++ ) kind[i] = 0;
56     tarjan ( 1 , -1 );
57 }

```

11. 最小圆覆盖

```

1 const int maxn = 120000;
2 struct point {
3     double x , y;
4 } a[maxn] , c , tmp1 , tmp2;
5 int n;
6 double r;
7 double tmp;

```

```

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

```

12. 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; i <= n; i = i + 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]);

```

13. 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。

n	0	1	2	4	6	8
B_n	1	$\frac{1}{2}$	$\frac{1}{6}$	$-\frac{1}{30}$	$\frac{1}{42}$	$-\frac{1}{30}$
n	10	12	14	16	18	20
B_n	$\frac{5}{66}$	$-\frac{691}{2730}$	$\frac{7}{6}$	$-\frac{3617}{510}$	$\frac{43867}{798}$	$-\frac{174611}{330}$

```

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 % M
    OD)

```

```

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 % M
OD)
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 }

```

14. 幂级数前缀和

KMAX 表示插值多项式次数最大值, **MOD** 为模数, 要求为质数。**qpow** 是快速幂, **add** 是取模加法。**f[0]** 到 **f[K + 1]** 存放的是前缀和函数的取值, 下面的预处理是暴力快速幂求出的, 如果要线性复杂度请换成线性筛。插值方法为 Lagrange 插值法, 单次计算复杂度为 $\Theta(K)$ 。注意计算结果可能为负数。使用时可以开一个 **PowerSeries** 的数组。

```

1 static bool _initialized;
2 static int cnt;
3 static i64 _fi[KMAX + 10], _tmp[KMAX + 10];
4 struct PowerSeries {
5     static void init() {
6         _fi[0] = 1;
7         for (int i = 2; i <= KMAX + 1; i++) _fi[0] =
8         _fi[0] * i % MOD;
9         _fi[KMAX + 1] = qpow(_fi[0], MOD - 2);
10        for (int i = KMAX; i >= 0; i--) _fi[i] = _fi
11        [i + 1] * (i + 1) % MOD;
12        _initialized = true;
13    }
14    int K; i64 *f;
15    PowerSeries() : PowerSeries(cnt++) {}
16    PowerSeries(int _K) : K(_K) {
17        if (!_initialized) init();
18        f = new i64[K + 2]; f[0] = 0;
19        for (int i = 1; i <= K + 1; i++) f[i] = (f[i
20        - 1] + qpow(i, K)) % MOD;
21    }
22    ~PowerSeries() { delete[] f; }
23    i64 operator()(i64 n) const {
24        n %= MOD; _tmp[K + 2] = 1;
25        for (int i = K + 1; i >= 1; i--) _tmp[i] =
26        _tmp[i + 1] * (n - i) % MOD;
27        i64 ret = 0, pre = 1;
28        for (int i = 0, b = K & 1 ? 1 : -1; i <= K +
29        1; i++, b = -b) {
30            add(ret, b * f[i] * pre % MOD * _tmp[i + 1
31            ] % MOD * _fi[i] % MOD * _fi[K + 1 - i] % MOD);
32            pre = pre * (n - i) % MOD;
33        } return ret;
34    }
35    i64 eval(i64 n) const { return (*this)(n); }
36 };

```

15. continued fraction

连分数相关/最佳分数逼近

这个代码用来处理 $\frac{a}{b} < \frac{x}{y} < \frac{c}{d}$ ，给出一组 x, y 最小的解，注意， x 最小就对应了 y 最小，二者是等价的。（请自行保证 $\frac{a}{b} < \frac{c}{d}$ ）

结果为 num/dom，过程中，dec 保存了两个分数的连分数展开，len 是两个数组的长度。例如 [4; 1, 2] 表示的分数是 $4 + \frac{1}{1 + \frac{1}{2}}$

连分数的一些性质 $[4; 1, 4, 3] = [4; 1, 4, 2, 1] = [4; 1, 4, 3, \infty]$, 可以在最后加一个 1 上去 (只能有一个, 因为 1 不能再减 1 了), 完成了之后, 后面可以认为有无穷个 ∞ 。

求一个分数的连分数展开：把整数部分减掉，放到答案数组里，然后把剩下的真分数去倒数，重复做到 $\frac{0}{x}$ 就是结果。无理数类似，但是要想办法存数值。

代码中求的是两个公共前缀，在第一个不同处取 $\min\{a_i, b_i\} + 1$ 就是分子分母最小的解。复杂度和辗转相除类似， $O(\log n)$ 。

如果要求的是和一个分数最接近的数，即限制了分子，分母有一个界，那么同样求出这个分数的连分数表示，然后考虑每一个前缀截断一下，并且把最后一个数字 $-1, +0, +1$ 分别求一下看看哪个最接近。复杂度 $O(\log^2 n)$ ，卡时间的话可以尝试二分一下，变成 $O(\log n \log \log n)$ 。（此段的代码没实现过，不保证正确性）（理论大概是连分数展开是最优的分数逼近，所以可以这样搞（不会证，不记得对不对））

```

1 long long dec1[1200] , dec2[1200] , len1 , len2;
2 long long num , dom;
3 void getfrac ( long long *d , long long &l , long long a , long long b ) {
4     l = 1;
5     d[1] = a / b;
6     a %= b;
7     while ( a != 0 ) {
8         swap ( a , b );
9         d[++l] = a / b;
10        a %= b;
11    }
12    void work () {
13        long long i;
14        getfrac ( dec1 , len1 , a , b );
15        getfrac ( dec2 , len2 , c , d );
16        dec1[len1+1] = 2147483647777777777ll;
17        dec2[len2+1] = 2147483647777777777ll;
18        for ( i = 1 ; i <= len1 && i <= len2 ; i++ ) {
19            if ( dec1[i] != dec2[i] ) break;
20        }
21        dec1[i] = min ( dec1[i] , dec2[i] ) + 1;
22        num = dec1[i]; dom = 1;
23        for ( i-- ; i >= 1 ; i-- ) {
24            swap ( num , dom );
25            num = num + dom * dec1[i];
26        }
27        printf ( "%lld %lld\n" , num , dom );
28    }

```

16. min 25 sieve

记号同 whzzt18 年集训队论文。 $f(x)$ 表示被求和的积性函数, 并且在质数点值是一个低阶多项式。

$$h(n) = \sum_{\substack{2 \leq p \leq n \\ p \text{ prime}}} f(p)$$

$$h_{n,m} = \sum_{\substack{2 \leq x \leq n \\ x \text{ 不含} \leq m \text{ 的质因子} \\ \text{或} x \text{ 是质数}}} x^k$$

$$g_{n,m} = \sum_{\substack{2 \leq x \leq n \\ x \text{ 不含} \leq m \text{ 的质因子} \\ \text{或} x \text{ 是质数}}} f(x)$$

注意从 2 开始。考虑线性筛的过程，每次筛掉一个最小的质数。对于 $h(n, m)$ 和 $g(n, m)$ 进行筛法时，考虑枚举 i 的最小质因子，并且合数的最小质因子不超过 \sqrt{n} 。其中 $h(n) = h(n, 0)$ ， $h(n, m)$ 是筛 $h(n)$ 的过程， $g(n, 0)$ 就是答案。从而写出递推式（假设质数点值 $f(p) = p^k$ ）

$$h(n, j) = h(n, j-1) - p_j^k \left[h\left(\left\lfloor \frac{n}{p_j} \right\rfloor, j-1\right) - h(p_{j-1}) \right]$$

其中 $p_{j-1} \leq \sqrt{n}$ 可以把 $h(p_{j-1})$ 打表, 扣掉是要把最小质因子小的去掉, 并且只有 $p_j^2 \leq n$ 时转移不为 0。从小到大按层转移。

$$g(n, i) = g(n, i + 1) +$$

$$\sum_{\substack{e \geq 1 \\ n_i^{e+1} \leq n}} \left[f(p_i^e) \left[g \left(\left\lfloor \frac{n}{p_i^e} \right\rfloor, i+1 \right) - h(p_i) \right] + f(p_i^{e+1}) \right]$$

同样的，只有 $p_i^2 \leq n$ 时存在转移，分层计算即可。初值 $h(n, 0) = \sum_{i=1}^n i^k$ 全都算上，然后把不是质数的点值筛出去， $g(n, m) = h(n)$ ，先只计算质数上的点值，然后把合数的点值逐个加入到 g 中。最后的答案是 $g(n, 0) + f(1)$ 。

```

1 typedef Long Long LL;
2 const LL NN = 420000;
3 const LL block = 100000;
4 const LL mod = 1000000007;
5 const LL inv2 = 500000004;
6 LL n, p[120000], prime[NN], tot;
7 LL value[NN], cnt, limit, pos[NN];
8 LL sumh[NN], h0[NN], h1[NN];
9 LL h[NN]; // sum of h[1..value[x]]
10 LL g[NN];
11 LL getpos ( LL x ) { return x<=limit?x:pos[n/x]; }
12 void predo () {
13     LL i, j;
14     for ( i = 2 ; i <= block ; i++ ) {
15         if ( !p[i] )
16             prime[++tot] = i;
17         for ( j = 1 ; j <= tot && i * prime[j] <= block ; j++ ) {
18             p[i*prime[j]] = 1;
19             if ( i % prime[j] == 0 ) break;
20         }
21         cnt = 0;
22         for ( i = 1 ; i * i <= n ; i++ ) value[++cnt] = i;
23         i--; limit = i;
24         for ( ; i >= 1 ; i-- ) if ( n / i != value[cnt] ) {
25             value[++cnt] = n / i;
26             pos[i] = cnt;
27         }
28         for ( i = 1 ; i <= tot ; i++ )
29             sumh[i] = (sumh[i-1] + prime[i]) % mod;
30         for ( i = 1 ; i <= cnt ; i++ ) { // cal h from 2 to i
31             h0[i] = ((value[i]-1)%mod*((value[i]+2)%mod)%mod*inv2) % mod; // modulo before multiply
32             h1[i] = (value[i] - 1) % mod;
33         }
34         for ( i = 1 ; i <= tot ; i++ ) {
35             for ( j = cnt ; prime[i] * prime[i] <= value[j] ; j-- ) {
36                 h0[j] = ( h0[j] - prime[i] * (h0[getpos(value[j])/prime[i]] - sumh[i-1]) ) % mod;
37                 if ( h0[j] < 0 ) h0[j] += mod;
38                 h1[j] = ( h1[j] - 1 * (h1[getpos(value[j])/prime[i]] - (i-1)) ) % mod;
39                 if ( h1[j] < 0 ) h1[j] += mod;
40             }
41             for ( i = 1 ; i <= cnt ; i++ ) // f(p)=p-1
42                 h[i] = ( h0[i] - h1[i] + mod ) % mod;
43         }
44         LL getf ( LL p , LL e ) { return p ^ e; }
45         void min25 () {
46             LL i, j, e, now, tmp;
47             for ( j = cnt ; j >= 1 ; j-- ) g[j] = h[j];
48             for ( i = tot ; i >= 1 ; i-- )
49                 for ( j = cnt ; prime[i] * prime[i] <= value[j] ; j-- )
50                     for ( e = 1 , now = prime[i] ; now * prime[i] <= value[j] ; e++ , now = now * prime[i] )
51                         g[j] = ( g[j] + getf(prime[i],e) * (g[getpos(value[j]/now)] - h[prime[i]] + mod) + getf(prime[i],e+1)) % mod;
52             printf ( "%lld\n" , (g[cnt] + 1) % mod );
53         }
54         void work () {
55             scanf ( "%lld" , &n );
56             predo ();
57             min25 ();
58         }

```

17. Pohlig_Hellman

用来对 smooth 的模数 p 求离散对数。如果 $p-1$ 的质因数分解中最大的素因子比较小可以使用。getlog 用来取对数，getroot 算原根，枚举时，只需要判断这个数的 $(p-1)/\text{prime factor}$

次幂是不是全部都不是 1 就可以。考虑 $n = p^e$ 阶循环群，原根 g 是生成元，现在要找 $g^x = h_0$ 。

1. 令 $x_0 = 0$
2. 计算 $r = g^{p^{e-1}}$ ，这个元素阶数为 p
3. 对 $k = 0 \dots e-1$ 计算
 1. $h_k = (g^{-x_k} h)^{p^{e-1-k}}$ ，这个元素同样是 p 阶的，在 $\langle r \rangle$ 中
 2. 用 BSGS（或者暴力）求出 $d_k \in \{0, \dots, p-1\}$ 满足 $r^{d_k} = h_k$
 3. 令 $x_{k+1} = x_k + p^k d_k$
4. 返回 x_e

即设 $x = c_0 + c_1 p + c_2 p^2 + \dots + c_{e-1} p^{e-1}$ ，每一次进行的 p^{e-1-k} 次方可以令之后的数都是 p^e 的倍数，从而都是 1，只留下 $g^{c_k p^{e-1}}$ 这一项（之前的项被 3.1. 中的逆元消去了），然后计算这一项。如果 $n = p_1^{e_1} p_2^{e_2}$ 这样，考虑对每个质因数，调用一次 $g_i = g^{n/p_i^{e_i}}$ ， $h_i = h^{n/p_i^{e_i}}$ ，得到 $x \equiv x_i \pmod{p_i^{e_i}}$ ，CRT 求解就可以了。这一步同样是把其他无关的素因子的阶通过高次幂消去。ExGCD 似乎过程是不会爆 long long 的（？）。复杂度 $O(\sum e_i (\log n + \sqrt{p_i}))$ 。

```

1 typedef Long Long LL;
2 LL pfactor[1200], totf;
3 LL gene[1200];
4 void exgcd(LL a, LL b, LL &x, LL &y) {
5     if(b==0){ x=1; y=0; return; }
6     exgcd(b, a%b, x, y);
7     LL tp=x;
8     x=y; y=tp-a/b*y;
9 }
10 LL inv ( LL a , LL mod ) {
11     LL x, y;
12     exgcd ( a , mod , x , y );
13     return (x%mod+mod)%mod;
14 }
15 LL qmul ( LL a , LL b , LL m ) {
16     a %= m; b %= m;
17     LL r = a*b, s=(Long double)(a)*b/m;
18     return ((r-m*s)%m+m)%m;
19 }
20 LL fast_mul ( LL a , LL b , LL c , LL mod ) {
21     return qmul(qmul(a,b,mod),c,mod);
22 }
23 pair<LL,LL> crt ( pair<LL,LL> a , pair<LL,LL> b ) {
24     if ( a.first == -1 ) return b;
25     a.first = fast_mul(a.first,b.second,inv(b.second,a.second),a.second*b.second) + fast_mul(b.first,a.second,inv(a.second,b.second),a.second*b.second);
26     a.second *= b.second;
27     a.first %= a.second;
28     return a;
29 }
30 LL mpow ( LL f , LL x , LL mod ) {
31     LL s = 1;
32     while ( x ) {
33         if ( x % 2 ) s = qmul(s,f,mod);
34         f = qmul(f,f,mod); x >>= 1;
35     } return s;
36 }
37 pair<LL,LL> solve ( LL g , LL h , LL mod , LL prime , LL p ) { // mod=prime^e
38     LL j, k, r = mpow ( g , mod / prime , p ) , x = 0 , hh;
39     LL ret = 0 , nowp = mod / prime , pp = 1;
40     gene[0] = 1;
41     for ( k = 1 ; k <= prime - 1 ; k++ ) {
42         gene[k] = qmul ( gene[k-1],r,p );
43     }
44     for ( k = 0 ; k <= e - 1 ; k++ ) {
45         h = qmul(h,inv(mpow(g,x,p),p),p);
46         hh = mpow ( h , nowp , p );
47         for ( j = 0 ; j <= prime - 1 ; j++ ) {
48             if ( gene[j] == hh ) break;
49         }
50         nowp = nowp / prime;

```

```

51     x = j * pp;
52     ret += x;
53     pp = pp * prime;
54 } return make_pair(ret, mod);
55 }
56 LL getlog ( LL a , LL root , LL p ) {
57     LL i , j , tp , tmp;
58     pair<LL,LL> ret , rem;
59     tp = p - 1;
60     rem.first = -1;
61     for ( i = 2 ; tp != 1 ; i++ ) {
62         if ( tp % i == 0 ) {
63             tmp = 1; j = 0;
64             while ( tp % i == 0 ) {
65                 tmp = tmp * i;
66                 j++; tp /= i;
67             }
68             ret = solve ( mpow ( root , p / tmp , p )
69 , mpow ( a , p / tmp , p ) , tmp , i , j , p );
70             rem = crt ( rem , ret );
71         }
72     } return rem.first;
73 }
74 LL getroot ( LL p ) {
75     LL i , j , tp = p - 1;
76     totf = 0;
77     for ( i = 2 ; tp != 1 ; i++ ) {
78         if ( tp % i == 0 ) {
79             pfactor[++totf] = i;
80             while ( tp % i == 0 ) tp /= i;
81         }
82     }
83     for ( i = 2 ; i < p ; i++ ) {
84         for ( j = 1 ; j <= totf ; j++ ) {
85             if ( mpow ( i , (p-1)/pfactor[j] , p ) ==
86 1 ) break;
87         }
88         if ( j == totf + 1 ) return i;
89     } return -1;
90 }
91 LL work ( LL p , LL a , LL b ) { // return x, s
uch that a^x = b (mod p)
92     LL i , j , rt , la , lb , x , y , g;
93     rt = getroot ( p );
94     la = getlog ( a , rt , p ); // rt^la = a (mod
p)
95     lb = getlog ( b , rt , p );
96     // x*la = lb (mod p-1)
97     g = __gcd ( la , p - 1 );
98     exgcd ( la , p - 1 , x , y );
99     if ( lb % g != 0 ) return -1;
100    x = (x%(p-1)+(p-1))%(p-1);
101    return qmul ( x , (lb/g) , (p - 1)/__gcd(la,p-
1) );
102 }

```

18. dinic

```

1 void add ( int u , int v , int f ) {
2     node *tmp1 = &pool[++top] , *tmp2 = &pool[++to
p];
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 tr
ue;
18            }
19    } return false;
20 }
21 int find ( int k , int key ) {
22     if ( k == 1 + n + n + 1 ) return key;
23     int i , s = 0;
24     for ( node *j = g[k] ; j ; j = j -> next )
25         if ( j -> f && level[j->v] == level[k] + 1 &

```

```

& s < key ) {
26     i = find ( j -> v , min ( key - s , j -> f
) );
27     j -> f -= i;
28     j -> rev -> f += i;
29     s += i;
30 }
31 if ( s == 0 ) level[k] = -1;
32 return s;
33 }
34 void dinic () {
35     int ans = 0;
36     while ( makelevel () == true ) ans += find ( 1
, 999999 );
37     //printf ( "%d\n" , ans );
38     if ( ans == sum ) printf ( "^_\n" );
39     else printf ( "T_T\n" );
40 }

```

19. 费用流

```

1 void add ( int u , int v , int f , int c ) {
2     node *tmp1 = &pool[++top] , *tmp2 = &pool[++to
p];
3     tmp1 -> v = v; tmp1 -> f = f; tmp1 -> c = c; t
mp1 -> next = g[u]; g[u] = tmp1; tmp1 -> rev = t
mp2;
4     tmp2 -> v = u; tmp2 -> f = 0; tmp2 -> c = -c;
tmp2 -> next = g[v]; g[v] = tmp2; tmp2 -> rev =
tmp1;
5 }
6 bool spfa () {
7     int i , k;
8     queue < int > q;
9     for ( i = 1 ; i <= 1 + n*m*3 + 1 ; i++ ) dis[i
] = 9999999 , f[i] = 0;
10    dis[1] = 0; f[1] = 1; q.push ( 1 );
11    while ( q.size () != 0 ) {
12        k = q.front (); q.pop (); f[k] = 0;
13        for ( node *j = g[k] ; j ; j = j -> next )
14            if ( j -> f && dis[j->v] > dis[k] + j -> c
) {
15                dis[j->v] = dis[k] + j -> c;
16                from[j->v] = k;
17                if ( f[j->v] == 0 ) q.push ( j -> v );
18                f[j->v] = 1;
19            }
20    }
21    if ( dis[1+n*m*3+1] != 9999999 ) return true;
22    return false;
23 }
24 int find () {
25     int i , f = 999999 , s = 0;
26     for ( i = 1+n*m*3+1 ; i != 1 ; i = from[i] ->
rev -> v ) f = min ( f , from[i] -> f );
27     flow += f;
28     for ( i = 1+n*m*3+1 ; i != 1 ; i = from[i] ->
rev -> v ) from[i] -> f -= f , from[i] -> rev ->
f += f;
29     return f * dis[1+n*m*3+1];
30 }
31 void dinic () {
32     int ans = 0;
33     while ( spfa () == true ) ans += find ();
34     //printf ( "%d\n" , flow );
35     if ( flow == sum && sum == sum1 ) printf ( "%d
\n" , ans );
36     else printf ( "-1\n" );
37 }

```

20. manacher

```

1 void manacher() {
2     //max(p[i])-1即为最大回文子串长
3     int mx=0,id=0;n=strlen(ch);
4     for(int i=n;i-->0)ch[i]=ch[i+1];
5     for(int i=1;i<=n;i++)c[i<<1]=ch[i],c[i<<1+1]=
'#';
6     m=n<<1+1;c[0]='-',c[1]='#',c[m+1]='+';
7     for(int i=1;i<=m;i++) {
8         if(mx>i)p[i]=min(p[2*i-id],mx-i);
9         while(c[p[i]+i]==c[i-p[i]])p[i]++;
10        if(i+p[i]>mx)mx=i+p[i],id=i;
11    }

```

21. SA_sl

```

1 #define N 200020
2 int wa[N],wb[N],ws[N],wv[N],sa[N],rank[N];
3 void cal_sa(int *r,int n,int m) {
4     int *x=wa,*y=wb,*t;
5     for(int i=0;i<m;i++)ws[i]=0;
6     for(int i=0;i<n;i++)ws[x[i]=r[i]]++;
7     for(int i=1;i<m;i++)ws[i]+=ws[i-1];
8     for(int i=n-1;i>=0;i--)sa[-ws[x[i]]]=i;
9     for(int j=1,p=1;p<n;j<=1,m=p) {
10         p=0;
11         for(int i=n-j;i<n;i++)y[p++]=i;
12         for(int i=0;i<n;i++)if(sa[i]>=j)y[p++]=sa[i]-j;
13         for(int i=0;i<n;i++)wv[i]=x[y[i]];
14         for(int i=0;i<m;i++)ws[i]=0;
15         for(int i=0;i<n;i++)ws[wv[i]]++;
16         for(int i=1;i<m;i++)ws[i]+=ws[i-1];
17         for(int i=n-1;i>=0;i--)sa[-ws[wv[i]]]=y[i];
18         t=x,x=y,y=t,p=1;x[sa[0]]=0;
19         for(int i=1;i<n;i++)
20             x[sa[i]]=(y[sa[i-1]]==y[sa[i]]&&y[sa[i-1]+j]==y[sa[i]+j])?p-1:p++;
21     }
22     int height[N];
23     void cal_h(int *r,int *sa,int n) {
24         int k=0;
25         for(int i=1;i<n;i++)rank[sa[i]]=i;
26         for(int i=0;i<n;i++) {
27             int j=sa[rank[i]-1];if(k>j)k=j;
28             while(r[j+k]==r[i+k])k++;
29             height[rank[i]]=k;
30         }
31         char ch[N];int r[N];
32         int main() {
33             std::cin>>ch;
34             int n=strlen(ch);
35             for(int i=0;i<n;i++)r[i]=ch[i];r[n]=0;
36             cal_sa(r,n+1,128);
37             cal_h(r,sa,n);
38             for(int i=1;i<n;i++)printf("%d ",sa[i+1]);put
39             s("");
40             for(int i=2;i<n;i++)printf("%d ",height[i]);
41         }

```

22. KMP

```

1 int p[101];
2 int main() {
3     string a,b;
4     cin>>a>>b;
5     int n=a.length(),m=b.length();
6     a=" "+a;b=" "+b;
7     int j=0;
8     for(int i=2;i<=m;i++) {
9         while(j>0&&b[j+1]!=b[i])j=p[j];
10        if(b[j+1]==b[i])j++;
11        p[i]=j;
12    }
13    j=0;
14    for(int i=1;i<=n;i++) {
15        while(j>0&&b[j+1]!=a[i])j=p[j];
16        if(b[j+1]==a[i])j++;
17        if(j==m){printf("%d",i-m+1);break;}
18    }
19    return 0;
20 }

```

23. PAM_sl

```

1 #define N 500020
2 int val[N], head[N], pos;
3 struct edge{int to,next;}e[N<<1];
4 void add(int a,int b) {pos++;e[pos].to=b,e[pos].
5 next=head[a],head[a]=pos;}
6 struct Tree {
7     char ch[N];
8     int now,cnt,odd,even;
9     int fail[N],len[N],go[N][26];
10    void init() {
11        now=cnt=0;
12        odd=++cnt,even=++cnt;
13        len[odd]=-1,len[even]=0;
14        fail[odd]=fail[even]=odd;
15        now=even;add(odd,even);
16    }
17    void insert(int pos,char c) {

```

```

17 while(ch[pos-1-len[now]]!=c)now=fail[now];
18 if(!go[now][c-'a']){
19     go[now][c-'a']=++cnt;
20     len[cnt]=len[now]+2;
21     if(now==odd)fail[cnt]=even;
22     else {
23         int t=fail[now];
24         while(ch[pos-1-len[t]]!=c)t=fail[t];
25         fail[cnt]=go[t][c-'a'];
26     }
27     add(fail[cnt],cnt);
28 }
29 now=go[now][c-'a'];
30 val[now]++;
31 }
32 void dfs(int u) {
33     for(int i=head[u];i;i=e[i].next) {
34         int v=e[i].to;
35         dfs(v);
36         val[u]+=val[v];
37     }
38     Long Long cal() {
39         Long Long ret=0;
40         for(int i=3;i<=cnt;i++)
41             ret=max(ret,1ll*len[i]*val[i]);
42         return ret;
43     }
44 }
45 int main() {
46     tree.init();
47     scanf("%s",tree.ch+1);
48     int len=strlen(tree.ch+1);
49     for(int i=1;i<=len;i++)
50         tree.insert(i,tree.ch[i]);
51     printf("%lld\n",tree.cal());
52 }

```

24. 回文自动机

```

1 int val[N];
2 int head[N],pos;
3 struct edge{int to,next;}e[N<<1];
4 void add(int a,int b)
5 {pos++;e[pos].to=b,e[pos].next=head[a],head[a]=p
6 os;}
7 struct Tree {
8     char ch[N];
9     int now,cnt,odd,even;
10    int fail[N],len[N],go[N][26];
11    void init() {
12        now=cnt=0;
13        odd=++cnt,even=++cnt;
14        len[odd]=-1,len[even]=0;
15        fail[odd]=fail[even]=odd;
16        now=even;add(odd,even);
17    }
18    void insert(int pos,char c) {
19        while(ch[pos-1-len[now]]!=c)now=fail[now];
20        if(!go[now][c-'a']) {
21            go[now][c-'a']=++cnt;
22            len[cnt]=len[now]+2;
23            if(now==odd)fail[cnt]=even;
24            else {
25                int t=fail[now];
26                while(ch[pos-1-len[t]]!=c)t=fail[t];
27                fail[cnt]=go[t][c-'a'];
28            }
29            add(fail[cnt],cnt);
30        }
31        now=go[now][c-'a'];
32        val[now]++;
33    }
34    void dfs(int u) {
35        for(int i=head[u];i;i=e[i].next) {
36            int v=e[i].to;
37            dfs(v);
38            val[u]+=val[v];
39        }
40        Long Long cal() {
41            Long Long ret=0;
42            for(int i=3;i<=cnt;i++)
43                ret=max(ret,1ll*len[i]*val[i]);
44            return ret;
45        }
46    }
47 }
48 int main() {
49     tree.init();
50     scanf("%s",tree.ch+1);
51     int len=strlen(tree.ch+1);
52     for(int i=1;i<=len;i++)
53         tree.insert(i,tree.ch[i]);
54     printf("%lld\n",tree.cal());
55 }

```


25. 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])) s
```

```
a[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 }
```

26. 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     }
37 }
38 void search(Node *x, char *s) {
39     for (int i = 0; s[i]; i++) {
40         int c = s[i] - 'a';
41         while (x->suf && !x->ch[c]) x = x->suf;
42         if (x->ch[c]) x = x->ch[c];
43         if (x->mark) print(i + 1, x->data);
44         for (Node *y = x->nxt; y; y = y->nxt) print(i + 1, y->data);
45     }
46 }
```

27. 后缀树

Ukkonen 在线添加尾部字符的后缀树构建算法。后缀树即后缀 Trie 的虚树, 树上节点数不超过两倍的字符串总长。State 是后缀树上的节点。Trans 是后缀树的边, 记录了一个区间 $[l, r]$ 表示边所对应的子串。根节点没有 fail 指针。原字符串 str 的下标从 1 开始, 字符串的最后一个字符是 EOFC, 该字符不一定要字典序最大。注意 n 比原长多 1。字符集的第一个字母为 0, 字符集 Σ 大小由 SIGMA 确定。添加字符串前需调用 `_append::reset`。时间复杂度为 $\Theta(n)$, 空间复杂度为 $\Theta(n|\Sigma|)$ 。大字符集请使用 `unordered_map`。

```
1 #define SIGMA 27
2 #define EOFC (SIGMA - 1)
3 struct State {
4     struct Trans {
5         Trans(int _l, int _r, State *_nxt) : l(_l), r(_r), nxt(_nxt) {}
6         int l, r; State *nxt;
7         int len() const { return r - l + 1; }
8     };
9     State() : fail(NULL) { memset(ch, 0, sizeof(ch)); }
10    State *fail; Trans *ch[SIGMA];
11 };
12 }
```

```

13 typedef State::Trans Trans;
14 static State *rt;
15 static char str[NMAX + 10];
16 static int n;
17 namespace _append {
18 static char dir;
19 static int len, cnt, cur;
20 static State *ap;
21 void reset() {
22     dir = -1; ap = rt;
23     len = cnt = cur = 0;
24 }
25 inline void append(char c) {
26     using namespace _append;
27     cnt++; cur++;
28     State *x, *y = NULL;
29     while (cnt) {
30         if (cnt <= len + 1) {
31             len = cnt - 1;
32             dir = len ? str[cur - len] : -1;
33         }
34         while (dir >= 0 && len >= ap->ch[dir]->len())
35         ) {
36             len -= ap->ch[dir]->len();
37             ap = ap->ch[dir]->nxt;
38             dir = len ? str[cur - len] : -1;
39         }
40         if ((dir >= 0 && str[ap->ch[dir]->l + len] =
41         = c) || (dir < 0 && ap->ch[c])) {
42             if (dir < 0) dir = c;
43             if (y) y->fail = ap;
44             len++; return;
45         }
46         if (dir < 0) {
47             ap->ch[c] = new Trans(cur, n, new State);
48             x = ap;
49         } else {
50             Trans *t = ap->ch[dir];
51             x = new State;
52             x->ch[c] = new Trans(cur, n, new State);
53             x->ch[str[t->l + len]] = new Trans(t->l +
54             len, t->r, t->nxt);
55             t->r = t->l + len - 1;
56             t->nxt = x;
57         }
58         if (y) y->fail = x;
59         if (ap != rt) ap = ap->fail;
60         y = x; cnt--;
61     }
62     inline void initialize() {
63         rt = new State;
64         _append::reset();
65         n = strlen(str) + 1;
66         for (int i = 1; i < n; i++) {
67             str[i] = 'a';
68             append(str[i]);
69         }
70         str[n] = EOF;
71         append(EOF);
72     }

```

28. 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（包括末尾添加的保留字符 \$），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，两个数组均从 0 开始索引。如果要多次使用请注意清空 cnt 数组。

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 suf_fix_sort 相同。

```

1 static int sa[NMAX + 10], rk[NMAX + 10], lcp[NMA
2 X + 10];
3 void suffix_sort(const char *s, int n, int m) {
4     static int x[NMAX + 10], y[NMAX + 10], cnt[NMA
5     X + 10], i;

```

```

4 //memset(cnt, 0, sizeof(int) * (m + 1));
5 for (i = 0; i < n; i++) cnt[s[i]]++;
6 for (i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
7 for (i = 0; i < n; i++) sa[--cnt[s[i]]] = i;
8 for (i = 1, m = 1, rk[sa[0]] = 1; i < n; i++)
9 {
10     if (s[sa[i - 1]] != s[sa[i]]) m++;
11     rk[sa[i]] = m;
12 }
13 for (int l = 1; l < n; l <= 1) {
14     memset(cnt, 0, sizeof(int) * (m + 1));
15     for (i = 0; i < n; i++) cnt[y[i]] = i + l < n
16     ? rk[i + l] : 0;
17     for (i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
18     for (i = n - 1; i >= 0; i--) x[--cnt[y[i]]]
19     = i;
20     memset(cnt, 0, sizeof(int) * (m + 1));
21     for (i = 0; i < n; i++) cnt[rk[i]]++;
22     for (i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
23     for (i = n - 1; i >= 0; i--) sa[--cnt[rk[x[i]
24     ]]] = x[i];
25     for (i = 1, m = 1, x[sa[0]] = 1; i < n; i++)
26     {
27         if (rk[sa[i - 1]] != rk[sa[i]] || y[sa[i -
28         1]] != y[sa[i]]) m++;
29         x[sa[i]] = m;
30     }
31     memcpy(rk, x, sizeof(int) * n);
32 }
33 void compute_lcp(const char *s, int n) {
34     int j = 0, p;
35     for (int i = 0; i < n; i++, j = max(0, j - 1))
36     {
37         if (rk[i] == 1) {
38             j = 0;
39             continue;
40         }
41         p = sa[rk[i] - 2];
42         while (p + j < n && i + j < n && s[p + j] ==
43         s[i + j]) j++;
44         lcp[rk[i]] = j;
45     }
46 }

```

29. 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);
2 i <= (r); ++i)
3 #define rrep(i, r, l) for (register int i = (r);
4 i >= (l); --i)
5 #define PUTS(x) sa[cur[str[x]]--] = x
6 #define PUTL(x) sa[cur[str[x]]++] = x
7 #define LMS(x) (!type[x - 1] && type[x])
8 #define RESET memset(sa + 1, 0, sizeof(int) * (n
9 + 1));
10 memcpy(cur + 1, cnt + 1, sizeof(int) * m);
11 #define INDUCE rep(i, 1, m) cur[i] = cnt[i - 1]
12 + 1;
13 rep(i, 1, n + 1) if (sa[i] > 1 && !type[sa[i]
14 - 1]) PUTL(sa[i] - 1);
15 memcpy(cur + 1, cnt + 1, sizeof(int) * m);
16 rrep(i, n + 1, 1) if (sa[i] > 1 && type[sa[i]
17 - 1]) PUTS(sa[i] - 1);
18 void sais(int n, int m, int *str, int *sa) {
19     static int id[NMAX + 10];
20     vector<bool> type(n + 2);
21     type[n + 1] = true;
22     rrep(i, n, 1) type[i] = str[i] == str[i + 1] ?
23     type[i + 1] : str[i] < str[i + 1];
24     int cnt[m + 1], cur[m + 1], idx = 1, y = 0, rt
25     , lrt, *ns = str + n + 2, *nsa = sa + n + 2;
26     memset(cnt, 0, sizeof(int) * (m + 1));
27     rep(i, 1, n + 1) cnt[str[i]]++;
28     rep(i, 1, m) cnt[i] += cnt[i - 1];
29     RESET rep(i, 2, n + 1) if (LMS(i)) PUTS(i); IN
30     DUCE
31     memset(id + 1, 0, sizeof(int) * n);
32     rep(i, 2, n + 1) if (LMS(sa[i])) {
33         register int x = sa[i];

```

```

25     for (rt = x + 1; !LMS(rt); rt++) ;
26     id[x] = y && rt + y == lrt + x && !memcmp(st
r + 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]]); I
NDUCE
38 }
39 static int str[NMAX * 3 + 10], sa[NMAX * 3 + 10]
;

```

30. pam

```

1 const int NN = 310000;
2 struct node {
3     int len, cnt, ch[30], fail;
4 } p[NN];
5 int top, n, last;
6 char z[NN];
7 long long ans;
8 void work () {
9     int i, tmp;
10    scanf ("%s", z + 1);
11    n = strlen (z + 1);
12    top = 2;
13    p[1].fail = 2; p[2].fail = 1;
14    p[1].len = 0; p[2].len = -1;
15    z[0] = '$';
16    last = 1;
17    for (i = 1; i <= n; i++) {
18        while (z[i] != z[i - p[last].len - 1]) last =
p[last].fail;
19        if (!p[last].ch[z[i] - 'a' + 1]) {
20            p[last].ch[z[i] - 'a' + 1] = ++top;
21            p[top].len = p[last].len + 2;
22            tmp = p[last].fail;
23            while (z[i] != z[i - p[tmp].len - 1]) tmp =
p[tmp].fail;
24            if (p[top].len > 1 && p[tmp].ch[z[i] - 'a' +
1]) p[top].fail = p[tmp].ch[z[i] - 'a' + 1];
25            else p[top].fail = 1;
26        }
27        last = p[last].ch[z[i] - 'a' + 1];
28        p[last].cnt++;
29    }
30    for (i = top; i >= 1; i--) p[p[i].fail].cn
t += p[i].cnt;
31    for (i = 1; i <= top; i++) {
32        //printf ("%d %d\n", p[i].len, p[i].cnt
);
33        ans = max (ans, (long long)p[i].len * p[i]
.cnt);
34    }
35    printf ("%lld\n", ans);
36 }

```

31. 权值splay

```

1 ll n, kind, rt, sz, fa[N], num[N];
2 ll tr[N][2], size[N], v[N], ans;
3 void pushup(ll k) {size[k] = size[tr[k][0]] + size[tr
[k][1]] + num[k];}
4 void rotate(ll x, ll &k) {
5     ll y = fa[x], z = fa[y], l, r;
6     l = tr[y][1] == x; r = l ^ 1;
7     if (y == k) k = x;
8     else tr[z][tr[z][1] == y] = x;
9     fa[x] = z, fa[tr[x][r]] = y, fa[y] = x;
10    tr[y][1] = tr[x][r], tr[x][r] = y;
11    pushup(y); pushup(x);
12 }
13 void splay(ll x, ll &k) {
14     while (x != k) {
15         ll y = fa[x], z = fa[y];
16         if (y != k) {
17             if (tr[y][0] == x ^ tr[z][0] == y)

```

```

18         rotate(x, k);
19         else rotate(y, k);
20     } rotate(x, k);
21 }
22 void insert(ll &k, ll x, ll last) {
23     if (!k) {k = ++sz; v[k] = x; size[k] = num[k] = 1; fa[k] = la
st; splay(k, rt); return;}
24     if (x == v[k]) num[k]++;
25     else if (x > v[k]) insert(tr[k][1], x, k);
26     else insert(tr[k][0], x, k);
27 }
28 ll t1, t2;
29 ll find(ll x, ll k) {
30     if (!k) return 0;
31     if (x == v[k]) return k;
32     else if (x > v[k]) return find(x, tr[k][1]);
33     else return find(x, tr[k][0]);
34 }
35 void ask_before(ll x, ll k) {
36     if (!k) return;
37     if (v[k] < x) {t1 = k; ask_before(x, tr[k][1]);}
38     else ask_before(x, tr[k][0]);
39 }
40 void ask_after(ll x, ll k) {
41     if (!k) return;
42     if (v[k] > x) {t2 = k; ask_after(x, tr[k][0]);}
43     // else if (v[k] == x) return;
44     else ask_after(x, tr[k][1]);
45 }
46 void del(ll x, ll k) {
47     if (num[k] > 1) {
48         num[k]--, size[k]--;
49         splay(k, rt); return;
50     }
51     t1 = t2 = -1;
52     ask_before(x, rt);
53     ask_after(x, rt);
54     if (t1 == -1 && t2 == -1) {
55         if (num[rt] == 1) rt = 0;
56         else size[rt]--, num[rt]--;
57     }
58     else if (t1 == -1) {
59         splay(t2, rt);
60         tr[rt][0] = 0;
61         pushup(rt);
62     }
63     else if (t2 == -1) {
64         splay(t1, rt);
65         tr[rt][1] = 0;
66         pushup(rt);
67     }
68     else {
69         splay(t1, rt);
70         splay(t2, tr[t1][1]);
71         tr[t2][0] = 0;
72         pushup(t2); pushup(t1);
73 }

```

32. 序列splay

```

1 int n, m, sz, rt;
2 char ch[10];
3 int tr[N][2], fa[N], v[N], sum[N];
4 int mx[N], lx[N], rx[N];
5 int st[N], size[N], top, tag[N];
6 bool rev[N];
7 void pushup(int u) {
8     size[u] = 1, sum[u] = v[u]; int l = tr[u][0], r = tr[u][1
];
9     if (l) size[u] += size[l], sum[u] += sum[l];
10    if (r) size[u] += size[r], sum[u] += sum[r];
11    mx[u] = v[u]; if (l) mx[u] = max(mx[u], mx[l]); if (r) mx
[u] = max(mx[u], mx[r]);
12    if (l && r) mx[u] = max(mx[u], rx[l] + v[u] + lx[r]);
13    else if (l) mx[u] = max(mx[u], rx[l] + v[u]);
14    else if (r) mx[u] = max(mx[u], v[u] + lx[r]);
15    lx[u] = 0; if (l) lx[u] = lx[l]; rx[u] = 0; if (r) rx[u] = rx
[r];
16    if (!l) lx[u] = max(lx[u], v[u]); if (!r) rx[u] = max(rx
[u], v[u]);
17    if (l && r) lx[u] = max(lx[u], v[u] + lx[r]); if (!r && l)
rx[u] = max(rx[u], v[u] + rx[l]);
18    if (l) lx[u] = max(lx[u], sum[l] + v[u]); if (r) rx[u] = m
ax(rx[u], sum[r] + v[u]);
19    if (l && r) lx[u] = max(lx[u], sum[l] + v[u] + lx[r]); rx[
u] = max(rx[u], sum[r] + v[u] + rx[l]);

```

```

20 }
21 void work(int k,int c) {
22     tag[k]=c,v[k]=c,sum[k]=size[k]*c;
23     mx[k]=(c>0?c*size[k]:c),lx[k]=rx[k]=(c>0?c*size[k]:0);
24 }
25 void rve(int k) {
26     rev[k]^=1;
27     swap(lx[k],rx[k]);
28     swap(tr[k][0],tr[k][1]);
29 }
30 void pushdown(int u) {
31     int l=tr[u][0],r=tr[u][1];
32     if(tag[u]!=12345) {
33         if(l)work(l,tag[u]);if(r)work(r,tag[u]);
34         tag[u]=12345;
35     }
36     if(rev[u]) {
37         if(l)rve(l);if(r)rve(r);
38         rev[u]^=1;
39     }
40 void rotate(int x,int &k) {
41     int y=fa[x],z=fa[y];
42     int l=(tr[y][1]==x),r=l^1;
43     if(y==k)k=x;
44     else tr[z][tr[z][1]==y]=x;
45     fa[x]=z,fa[y]=x,fa[tr[x][r]]=y;
46     tr[y][l]=tr[x][r],tr[x][r]=y;
47     pushup(y);pushup(x);
48 }
49 void splay(int x,int &k) {
50     while(x!=k) {
51         int y=fa[x],z=fa[y];
52         if(y!=k) {
53             if(tr[y][0]==x^tr[z][0]==y)
54                 rotate(x,k);
55             else rotate(y,k);
56         }
57         rotate(x,k);
58     }
59 int find(int k,int rk) {
60     pushdown(k);
61     int l=tr[k][0],r=tr[k][1];
62     if(size[l]>=rk)return find(l,rk);
63     else if(size[l]+1==rk)return k;
64     else return find(r,rk-size[l]-1);
65 }
66 int split(int l,int r) {
67     int x=find(rt,l),y=find(rt,r+2);
68     splay(x,rt),splay(y,tr[x][1]);
69     return tr[y][0];
70 }
71 int a[N];
72 void newnode(int k,int c)
73 {v[k]=sum[k]=c,mx[k]=c,tag[k]=12345,lx[k]=rx[k]=
(c>0?c:0),size[k]=1,rev[k]=0;}
74 int build(int l,int r) {
75     if(l>r)return 0;int mid=(l+r)>>1,now;
76     now++sz;newnode(now,a[mid-1]);
77     tr[now][0]=build(l,mid-1);if(tr[now][0])fa[tr[
now][0]]=now;
78     tr[now][1]=build(mid+1,r);if(tr[now][1])fa[tr[
now][1]]=now;
79     pushup(now);return now;
80 }
81 int Build(int l,int r) {
82     if(l>r)return 0;int mid=(l+r)>>1,now;
83     if(top)now=st[top--];else now++sz;newnode(now
,a[mid]);
84     tr[now][0]=Build(l,mid-1);if(tr[now][0])fa[tr[
now][0]]=now;
85     tr[now][1]=Build(mid+1,r);if(tr[now][1])fa[tr[
now][1]]=now;
86     pushup(now);return now;
87 }
88 void insert(int x,int tot) {
89     for(int i=0;i<tot+2;i++)a[i]=0;
90     for(int i=1;i<=tot;i++)a[i]=read();
91     int l=find(rt,x+1),r=find(rt,x+2);
92     splay(l,rt),splay(r,tr[l][1]);
93     tr[r][0]=Build(1,tot),fa[tr[r][0]]=r;
94     pushup(r),splay(r,rt);
95 }
96 void clr(int k){tag[k]=12345,tr[k][0]=tr[k][1]=f
a[k]=rev[k]=v[k]=sum[k]=mx[k]=lx[k]=rx[k]=size[k]

```

```

]=0;}
97 void rec(int k) {
98     if(!k)return;
99     rec(tr[k][0]);rec(tr[k][1]);
100     st[++top]=k,clr(k);
101 }
102 void del(int x,int tot) {
103     int l=x,r=x+tot-1,k=split(l,r);
104     int fk=fa[k];tr[fk][0]=fa[k]=0;rec(k);
105     splay(fk,rt);
106 }
107 void make_same(int x,int tot,int c)
108 {int l=x,r=x+tot-1,k=split(l,r);work(k,c);if(fa[
k])splay(fa[k],rt);}
109 void rever(int x,int tot)
110 {int l=x,r=x+tot-1,k=split(l,r);rve(k);if(fa[k])
splay(fa[k],rt);}
111 int get_sum(int x,int tot) {
112     int l=x,r=x+tot-1,k=split(l,r);
113     return sum[k];
114 }

```

33. ntt

```

1 const Long Long maxn = 120000;
2 const Long Long mod = 998244353;
3 const Long Long omega = 3;
4 Long Long a[maxn*4] , b[maxn*4] , c[maxn*4] , d[
maxn*4];
5 Long Long n , m , N , in;
6 Long Long pow ( Long Long f , Long Long x ) {Lon
g Long s = 1;while ( x ) {if ( x % 2 ) s = (s*f)
% mod;f = (f*f) % mod; x >>= 1;}return s;}
7 Long Long inv ( Long Long x ) {return pow ( x ,
mod - 2 );}
8 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;}
9 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];}
10 void FFT ( Long Long *x , Long Long f ) {
11     Long Long i , j , s , k;
12     Long Long w , wm , u , t;
13     br ( x );
14     for ( s = 2 ; s <= N ; s *= 2 ) {
15         k = s / 2;
16         wm = pow ( omega , (mod-1) / s );
17         if ( f == -1 ) wm = inv ( wm );
18         for ( i = 0 ; i < N ; i += s ) {
19             w = 1;
20             for ( j = 1 ; j <= k ; j++ ) {
21                 u = x[i+j-1]; t = (x[i+j-1+k]*w) % mod;
22                 x[i+j-1] = (u + t) % mod;
23                 x[i+j-1+k] = (u - t + mod) % mod;
24                 w = (w*wm) % mod;
25             }
26             if ( f == -1 ) for ( i = 0 ; i < N ; i++ ) x[i
] = (x[i] * in) % mod;
27         }
28     }
29 void work () {
30     Long Long i;
31     scanf ( "%lld%lld" , &n , &m );
32     N = 1;
33     while ( N < n + m + 2 ) N = N * 2;
34     for ( i = 0 ; i <= n ; i++ ) scanf ( "%lld" ,
&a[i] );
35     for ( i = 0 ; i <= m ; i++ ) scanf ( "%lld" ,
&b[i] );
36     in = inv ( N );
37     FFT ( a , 1 ); FFT ( b , 1 );
38     for ( i = 0 ; i < N ; i++ ) c[i] = (a[i]*b[i])
% mod;
39     FFT ( c , -1 );
40     for ( i = 0 ; i <= n + m ; i++ ) printf ( "%ll
d%c" , c[i] , i==n+m?'\\n':' ' );

```

34. fft

```

1 const int maxn = 120000;
2 const double pi = acos(-1);
3 struct complex {
4     double r , i;
5 } a[maxn*4] , b[maxn*4] , c[maxn*4] , d[maxn*4];
6 complex operator + ( complex x1 , complex x2 ) {

```

```

    complex y; y.r = x1.r + x2.r; y.i = x1.i + x2.i; return y; }
7 complex operator - ( complex x1 , complex x2 ) {
    complex y; y.r = x1.r - x2.r; y.i = x1.i - x2.i; return y; }
8 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; }
9 int n , m , N;
10 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; }
11 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]; }
12 void FFT ( complex *x , int f ) {
    int i , j , s , k;
    complex w , wm , u , t;
    br ( x );
    for ( s = 2 ; s <= N ; s *= 2 ) {
        k = s / 2;
        wm.r = cos(2*pi/s); wm.i = sin(2*pi/s) * f;
        for ( i = 0 ; i < N ; i += s ) {
            w.r = 1.0; w.i = 0.0;
            for ( j = 1 ; j <= k ; j++ ) {
                u = x[i+j-1]; t = x[i+j-1+k] * w;
                x[i+j-1] = u + t;
                x[i+j-1+k] = u - t;
                w = w * wm;
            }
        }
        if ( f == -1 ) for ( i = 0 ; i < N ; i++ ) x[i]
            .r = x[i].r / N;
    }
29 void work () {
    int i;
    scanf ( "%d%d" , &n , &m );
    N = 1;
    while ( N < n + m + 2 ) N = N * 2;
    for ( i = 0 ; i <= n ; i++ ) scanf ( "%lf" , &
    a[i].r );
    for ( i = 0 ; i <= m ; i++ ) scanf ( "%lf" , &
    b[i].r );
    FFT ( a , 1 ); FFT ( b , 1 );
    for ( i = 0 ; i < N ; i++ ) c[i] = a[i] * b[i];
    FFT ( c , -1 );
    for ( i = 0 ; i <= n + m ; i++ ) printf ( "%d%
    c" , int (c[i].r + 0.5) , i==n+m?'\\n':' ' );
40 }

```

35. 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[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].f
    a].ch[1] ) )
29         if ( p[p[x].fa].fa > 0 && (x==p[p[x].fa].ch[
    1]) == (p[p[x].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 void makeroot ( Long Long x ) {
44     access ( x );
45     p[x].rev ^= 1;
46     swap ( p[x].ch[0] , p[x].ch[1] );
47 }
48 void link ( Long Long x , Long Long y ) {
49     makeroot ( y );
50     p[y].fa = -x;
51 }

```

36. 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 }

```

37. 三分_上凸函数

```

1 double solve() {
2     while(1+eps<r) {
3         double mid=(l+r)/2.0;
4         double mmid=(mid+r)/2.0;
5         if(cal(mid)>cal(mmid))r=mmid;
6         else l=mid;
7     }
8     if(cal(l)<cal(r))return r;
9     else return l;
10 }

```

38. 线性空间求交

设两个线性空间 U 、 V 的基分别为 u_1, u_2, \dots, u_n 和 v_1, v_2, \dots, v_m 。考虑同时求出 $U+V$ 和 $U \cap V$ 的基：逐次将 u_i 加入。设当前扩展到 $v_1, \dots, v_m, u'_1, \dots, u'_j$ ，若 u_i 不能被它们线性表出，则令 $u'_{j+1} = u_i$ 。否则 $u_i = \sum a_j u'_j + \sum b_j v_j$ ，即 $u_i - \sum a_j u'_j = \sum b_j v_j$ ，那么等式左边可以直接加入交空间。时间复杂度 $\Theta(nm)$ 。代码是异或线性空间的求交。

```

1 #define SMAX 32
2 typedef unsigned int u32;
3 struct Basis {
4     u32 v[SMAX];
5     auto operator[](const size_t i) -> u32& {
6         return v[i];
7     };
8 auto intersect(Basis &u, Basis v) -> Basis {
9     Basis z, r;
10    for (int i = 0; i < SMAX; i++) if (u[i]) {
11        u32 x = u[i], y = u[i];
12        for (int j = 0; j < SMAX; j++) if ((x >> j)
13        & 1) {
14            if (v[j] x ^= v[j], y ^= r[j];
15            else {
16                v[j] = x, r[j] = y;
17                break;
18            }
19        }
20        if (!x) z.add(y);
21    }
22    return z;
23 }

```

```

58     if (idy[i] <= n) ans[idy[i]] = A[i][0];
59     for (int i = 1; i <= n; i++)
60         printf("%.10Lf ", ans[i]);
61     printf("\n");
62 }
63 private:
64 void pivot(int x, int y) {
65     swap(idy[x], idy[y]);
66     double r = -A[y][x];
67     A[y][x] = -1;
68     for (int i = 0; i <= n; i++) A[y][i] /= r;
69     for (int i = 0; i <= m; i++) {
70         if (i == y) continue;
71         r = A[i][x];
72         A[i][x] = 0;
73         for (int j = 0; j <= n; j++)
74             A[i][j] += r * A[y][j];
75     }
76     int n, m, t;
77     double A[NMAX + 10][NMAX + 10];
78     int idx[NMAX + 10], idy[NMAX + 10];
79 };

```

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

```

• s11/扩展网络流.md

无源汇有上下界可行流:

建图:

$M[i]$ =流入 i 点的下界流量-流出 i 点的下界流量

$S \rightarrow i, c = M[i]$ ($M[i] \geq 0$)

$i \rightarrow T, c = -M[i]$

流程:

$S \rightarrow T$ 跑最大流,当 S 连出去的边满流是存在可行流

有源汇上下界最大流:

建图:

$T \rightarrow S$,流量限制为(0,无穷大), 转化成无源汇

增设 ST 和 SD ,像无源汇那样连边

流程:

1. $ST \rightarrow SD$ 跑最大流,判断是否满流,不满流则无解

2. 去掉 ST, SD ,从 $S \rightarrow T$ 跑最大流,两遍流量和为有源汇最大流量

有源汇上下界最小流:

建图: 同最大流

流程: 1. 同最大流

1. 去掉 $ST, SD, T \rightarrow S$ 跑最大流,两次流量之差为有源汇最小流

最大权闭合子图:

问题描述: 求最大权值和的点集,使得这个点集里的任一点的后继也在该点集中

建图: 原图中的($u \rightarrow v$),建边($u \rightarrow v, \text{inf}$)

对于 $c[u] > 0$ 建边($s \rightarrow u, c[u]$)

对于 $c[u] < 0$ 建边($u \rightarrow t, -c[u]$)

流程: 建图后跑 $s \rightarrow t$ 的最小割, $\sum c[u] (c[u] > 0)$ -最小割即为答案

• xz1/manhattan.md

Manhattan 距离最小生成树: 每 45° 一个象限, 对每个点找到每个象限中离它最近的点连边, 然后做最小生成树。

优化: 只用写找直线 $y = x$ 与直线 $x = 0$ 之间的最近点的代码, 然后依次交换 x 和 y 、取反 y 、交换 x 和 y 一共做 4 次扫描线即可。

• xz1/maxdn.md

表格内的数据表示最坏情况。

$\log_{10} n$	1	2	3	4	5	6
$\omega(n)$	2	3	4	5	6	7
$d(n)$	4	12	32	64	128	240
$\log_{10} n$	7	8	9	10	11	12
$\omega(n)$	8	9	9	10	10	11
$d(n)$	448	768	1344	2304	4032	6720

$\log_{10} n$	13	14	15	16	17	18
$\omega(n)$	12	12	13	13	14	15
$d(n)$	10752	17280	26880	41472	64512	103680

◦ xzl/fwt.md

FWT 算法：分治 $A \rightarrow A_1, A_2$ ，线性变换 T ，合并时 $A = T[A_1, A_2]^T$ 。逆变换时取 T 的逆矩阵即可。

卷积类型	变换
异或卷积	$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \begin{bmatrix} 1/2 & 1/2 \\ 1/2 & -1/2 \end{bmatrix}$
或卷积	$\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}$
和卷积	$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}$

或卷积就是子集和变换。通过按子集大小分类可在 $O(n \log^2 n)$ 时间内计算子集卷积：

```
for i = 0 → n - 1: // 按大小分类
    F[c][i] = f[i]
    G[c][i] = g[i]
for i = 0 → k - 1: // 提前计算 FWT
    F[i] = fwt(F[i])
    G[i] = fwt(G[i])
for i + j = k: // 卷积
    H[k] += F[i] · G[j]
for i in xrange(k): // FWT 逆变换
    H[i] = rfwt(H[i])
for all subset S: // 得到卷积结果
    R[i] = H[popcount(S)][i]
```

◦ lmj/treeshash.md

$$\text{hash}[x] = A \cdot \prod_{v \text{ 是 } x \text{ 的儿子}} (\text{hash}[v] \oplus B) \pmod C$$

◦ 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>

- `lmj/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 $\text{lca}(x,y)=x$ ，则查询 $\text{st}[x]$ 到 $\text{st}[y]$ ，否则 $\text{ed}[x],\text{st}[y]$ ，再加上 lca ，出现两次的点不算。
- `lmj/game.md` ■
各种游戏题

- n 数码问题，考虑把 0 去掉之后的逆序对数量，如果是 $n \times n$ ， n 为偶数的话，还要加上每个数到正确的行需要的步数和。是偶数就可以恢复。
- `lmj/idea.md` ■
启发式合并
离线
hash
数据结构上跑图论算法