---

1. 边双联通 tarjan

```
const int N = 5010; // 3352只用1010即可
struct node{
  int v,w,id;
  node(int v = 0,int w = 0,int id = 0):v(v),w(w),id(id){};
};
vector<node>G[N];
int pre[N];
int low[N];
int dfs_num;int ans ;int n,m;
void init(){
  mem(pre,0); mem(low,0);
  for(int i=0;i<=n;i++) G[i].clear();
  dfs_num = 0;ans = INF;
}
int dfs(int u,int fa){
  low[u] = pre[u] = ++dfs_num;
  for(int i=0;i<G[u].size();i++){
    int v = G[u][i].v;
    int id = G[u][i].id;
    if(id == fa) continue;
    if(!pre[v]){
      dfs(v,id);//注意这里 第二个参数是 id
      low[u] = min(low[u],low[v]);//用后代的low更新当前的
    }
    else
      low[u] = min(low[u],pre[v]);//利用后代v的反向边更新low
}}
int main(){
  int t;
  while(scanf("%d%d",&n,&m)!=EOF&& (n || m)){
    int a,b,c;
    init();
    for(int i=1;i<=m;i++){
      scanf("%d%d",&a,&b);
      G[a].push_back(node(b,0,i));
      G[b].push_back(node(a,0,i));
    }
    for(int i=1;i<=n;i++){
      if(!pre[i])
        dfs(i,0);
      //cout<<i<<endl;
    }
    int degree[N];mem(degree,0);
    for(int i=1;i<=n;i++){
      for(int j=0;j<G[i].size();j++){
        int v = G[i][j].v;
        if(low[i] != low[v]){
          degree[low[v]]++; degree[low[i]]++;
    }}}
    int l = 0;
    for(int i=1;i<=dfs_num;i++)
      if(degree[i] == 2)
      l++;
    printf("%d\n",(l+1)/2);
  }
  return 0;
}
```

2. 点双联通 tarjan

```
void tarjan(int u, int fa) {
  pre[u] = low[u] = ++dfs_clock;
  for (int i = 0; i < (int)G[u].size(); i++) {
    int v = G[u][i];
    if (!pre[v]) {
      S.push(Edge(u, v));
      tarjan(v, u);
      low[u] = min(pre[v], low[u]);
      if (low[v] >= pre[u]) {
        bcc_cnt++;
        bcc[bcc_cnt].clear();
        for(;;) {
          Edge x = S.top(); S.pop();
          if (bccno[x.u] != bcc_cnt) {
            bcc[bcc_cnt].push_back(x.u);
            bccno[x.u] = bcc_cnt;
          }
          if (bccno[x.v] != bcc_cnt) {
            bcc[bcc_cnt].push_back(x.v);
            bccno[x.v] = bcc_cnt;
          }
          if (x.u == u && x.v == v) break;
    }}}
    else if (pre[v] < pre[u] && v != fa) {
      S.push(Edge(u, v));
      low[u] = min(low[u], pre[v]);
}}}
```

3. 有向图强联通 tarjan

```
int n,m;
int head[N],pos;
struct edge{int to,next;}e[N<<1];
void add(int a,int b)
{pos++;e[pos].to=b,e[pos].next=head[a],head[a]=pos;}
int dfn[N],low[N],SCC;
bool in[N];
int st[N],top,T;
vector<int>G[N];
void tarjan(int u) {
  st[++top]=u;in[u]=1;
  dfn[u]=low[u]=++T;
  for(int i=head[u];i;i=e[i].next) {
    int v=e[i].to;
    if(!dfn[v]) {
      tarjan(v);
      low[u]=min(low[u],low[v]);
    }
    else if(in[v])low[u]=min(low[u],dfn[v]);
  }
  if(low[u]==dfn[u]) {
    int v;
    ++SCC;
    do {
      v=st[top--];
      in[v]=false;
      G[SCC].push_back(v);
    }while(v!=u);
}}
int main() {
```

```cpp
  scanf("%d%d",&n,&m);
  for(int i=1;i<=m;i++) {
    int x,y;
    scanf("%d%d",&x,&y);
    add(x,y);
  }
  for(int i=1;i<=n;i++)if(!dfn[i])tarjan(i);
}
```

4. 倍增lca

```cpp
int lca(int x,int y) {
  if(deep[x]<deep[y])swap(x,y);
  int t=deep[x]-deep[y];
  for(int i=0;bin[i]<=t;i++)
    if(t&bin[i])x=fa[x][i];
  for(int i=16;i>=0;i--)
    if(fa[x][i]!=fa[y][i])
      x=fa[x][i],y=fa[y][i];
  if(x==y)return x;
  return fa[x][0];
}
```

5. 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。

```cpp
static vector<int> G[NMAX + 10], T[NMAX + 10];
void bcc(int u, int f = 0) {
  static stack<Pair> stk;
  static bool marked[NMAX + 10];
  static int in[NMAX + 10], low[NMAX + 10], cur;
  in[u] = low[u] = ++cur;
  for (int v : G[u]) {
    if (v == f) f = 0;  // 应对重边
    else if (in[v]) low[u] = min(low[u], in[v]);
    else {
      stk.push(Pair(u, v));  // stk 内存储 DFS 树上的边
      bcc(v, u);
      low[u] = min(low[u], low[v]);
      if (low[v] > in[u]) {  // 割边 u - v
        T[u].push_back(v);
        T[v].push_back(u);
        stk.pop();
      } else if (low[v] >= in[u]) {  // 可能有点双了
        cnt++;
        int linked = 0, p = n + cnt;  // linked 点数，p 圆方树上的新方点
        auto add = [p, &linked](int x) {
          if (!marked[x]) {
            marked[x] = true;
            T[p].push_back(x);
            T[x].push_back(p);
            linked++;
          }};
        while (!stk.empty()) {
          Pair x = stk.top();
          stk.pop();
          add(x.u);
          add(x.v);
          if (x.u == u && x.v == v) break;
        }
        for (int v : T[p]) marked[v] = false;
        if (linked == 0) cnt--;  // 假点双
}}}}
```

6. 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。

```cpp
int dfn[NMAX + 10], low[NMAX + 10], now, cnt;
int ed[NMAX + 10], fa[NMAX + 10], a[NMAX + 10];
void dfs(int x) {
  dfn[x] = low[x] = ++now;
  for (int v : G[x]) if (v != fa[x]) {
    if (dfn[v]) {
      ed[v] = x, low[x] = min(low[x], dfn[v]);
      continue;
    } fa[v] = x;
    dfs(v);
    if (low[v] > dfn[x]) ; // 割边
    else if (low[v] == dfn[x]) {
      a[1] = x;
      for (cnt = 1, v = ed[x]; v != x; v = fa[v])
        a[++cnt] = v;
      // 环 a[1]...a[cnt]
    } else low[x] = min(low[x], low[v]);
}}
```

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。

```cpp
static int n, m, G[NMAX + 10], nxt[MMAX + 10];
static struct Edge { int u, v, w; } E[MMAX + 10], *in[NMAX + 10];
static int id[NMAX + 10], mark[NMAX + 10];
int find(int x) { return id[x] ? id[x] = find(id[x]) : x; }
int dfs(int x) {
  mark[x] = 1; int ret = 1;
  for (int i = G[x]; i; i = nxt[i])
    if (!mark[E[i].v]) ret += dfs(E[i].v);
  return ret;
}
inline int detect(int x) {
  mark[x] = x;
  for (int y = in[x]->u; in[y]; y = in[y]->u)
    if (mark[y]) return mark[y] == x ? y : 0;
    else mark[y] = x;
  return 0;
}
int mdst(int r) {
  if (dfs(r) < n) return -1;
  int ret = 0;
  while (true) {
    memset(in, 0, sizeof(in));
    memset(mark, 0, sizeof(mark));
    for (auto *e = E + 1; e <= E + m; e++)
      if (e->u != e->v && e->v != r && (!in[e->v] || e->w < in[e->v]->w))
        in[e->v] = e;
    int p = 0, t = 0;
    for (int x = 1; x <= n; x++, t |= p) if (!mark[x] && in[x]) {
      if (!(p = detect(x))) continue;
      ret += in[p]->w;
      for (int x = in[p]->u; x != p; x = in[x]->u)
        id[find(x)] = p, ret += in[x]->w;
      for (auto *e = E + 1; e <= E + m; e++) {
        int u = find(e->u), v = find(e->v);
        if (u != p && v == p) e->w -= in[e->v]->w;
        e->u = u; e->v = v;
      }}
    if (!t) break;
  }
  for (int x = 1; x <= n; x++) if (in[x]) ret += in[x]->w;
  return ret;
}
```

8. blossom algorithm

```cpp
const int maxn = 510;
struct node {
  int v;
  node *next;
} pool[maxn*maxn*2] , *g[maxn];
int top,n , m,match[maxn];
int kind[maxn] , pre[maxn] , vis[maxn] , c[maxn];
queue < int > q;
int f[maxn],ans;
void add ( int u , int v ) {node *tmp = &pool[++top];tmp -> v = v; tmp -> next = g[u]; g[u] = tmp;}
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;}
void getpath ( int x , int tar , int root ) {
  int t;
```

```
  while ( x != root ) {t = match[x];match[tar] = x;match[x] = tar;tar = t;x = pre[t];}
  match[tar] = x;match[x] = tar;
}
int lca ( int u , int v , int root ) {
  int i;for ( i = 1 ; i <= n ; i++ ) f[i] = 0;
  while ( find ( u ) != root ) {u = find ( u );f[u] = 1;if ( !match[u] ) break;u = pre[match[u]];}
  f[root] = 1;
  while ( find ( v ) != root ) {v = find ( v );if ( f[v] == 1 ) return v;if ( !match[v] ) break;v = pre[match[v]];}
  return root;
}
void blossom ( int x , int y , int l ) {
  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)] = l;if ( find ( match[x] ) == match[x] ) c[find(match[x])] = l;x = pre[y];}
}
void bfs ( int x ) {
  int k , i , z;
  for ( i = 1 ; i <= n ; i++ ) {
    kind[i] = pre[i] = vis[i] = 0;c[i] = -1;
  }
  while ( q.size () ) q.pop ();q.push ( x );kind[x] = 1; vis[x] = 1;
  while ( q.size () ) {
    k = q.front (); q.pop ();
    for ( node *j = g[k] ; j ; j = j -> next ) {
      if ( !vis[j->v] ) {
        if ( !match[j->v] ) {
          getpath ( k , j -> v , x );
          return ;
        }
        else {
          kind[j->v] = 2;
          kind[match[j->v]] = 1;
          pre[j->v] = k;
          vis[j->v] = 1; vis[match[j->v]] = 1;
          q.push ( match[j->v] );
      }}
      else {
        if ( find ( k ) == find ( j -> v ) ) continue;
        if ( kind[find(j->v)] == 1 ) {
          z = lca ( k , j -> v , x );
          blossom ( k , j -> v , z );
          blossom ( j -> v , k , z );
}}}}}
void work () {
  int i , u , v;
  scanf ( "%d%d" , &n , &m );
  for ( i = 1 ; i <= m ; i++ ) {
    scanf ( "%d%d" , &u , &v );
    add ( u , v ); add ( v , u );
  }
  for ( i = 1 ; i <= n ; i++ ) {
    if ( !match[i] ) bfs ( i );
  }
  for ( i = 1 ; i <= n ; i++ ) if ( match[i] ) ans++;
  printf ( "%d\n" , ans / 2 );
  for ( i = 1 ; i <= n ; i++ ) printf ( "%d%c" , match[i] , i==n?'\n':' ' );
}
```

9. euler_tour

```
stack < int > s;
void dfs ( int i ) {
  for ( node *j = g[i] ; j ; j = j -> next ) if ( !j -> taboo ) {
    s.push ( j -> f );
    j -> taboo = 1;
    dfs ( j -> v );
    ans[++index] = s.top ();
    s.pop ();
  }
}
```

10. 最小圆覆盖

```
const int maxn = 120000;
struct point {
  double x , y;
} a[maxn] , c , tmp1 , tmp2;
int n;
double r;
double tmp;
double dis ( point x1 , point x2 ) {return sqrt ( (x1.x-x2.x)*(x1.x-x2.x) + (x1.y-x2.y)*(x1.y-x2.y) );}
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);}
double abs ( double x ) {if ( x < 0 ) return -x; return x;}
point getcen ( point x1 , point x2 , point x3 ) {
  double A , B , C , D , E , F;point ret;
  if ( x1.x == x2.x ) A = 0.0, B = 1.0, C = (x1.y+x2.y)/2.0;
  else {
    A = 1.0/((x1.y-x2.y) / (x1.x-x2.x));B = 1.0;
    C = -(x1.y+x2.y)/2.0 - A * (x1.x+x2.x)/2.0;
  }
  if ( x1.x == x3.x ) D = 0.0, E = 1.0, F = (x1.y+x3.y)/2.0;
  else {
    D = 1.0/((x1.y-x3.y) / (x1.x-x3.x));E = 1.0;
    F = -(x1.y+x3.y)/2.0 - D * (x1.x+x3.x)/2.0;
  }
  ret.x = (B * F - C * E) / (A * E - B * D);
  ret.y = (A * F - C * D) / (B * D - A * E);
  return ret;
}
void work () {
  int i , j , k;
  srand(67890);
  scanf ( "%d" , &n );
  for ( i = 1 ; i <= n ; i++ ) scanf ( "%lf%lf" , &a[i].x , &a[i].y );
  random_shuffle ( a + 1 , a + 1 + n );
  if ( n == 2 ) {
    printf ( "%.3lf\n" , dis ( a[1] , a[2] ) / 2.0 );
    return ;
  }
  c.x = a[1].x;c.y = a[1].y;r = 0.0;
  for ( i = 2 ; i <= n ; i++ ) {
    if ( dis ( c , a[i] ) - r > 1e-9 ) {
      c.x = a[i].x;c.y = a[i].y;r = 0.0;
      for ( j = 1 ; j < i ; j++ ) {
        if ( dis ( c , a[j] ) - r > 1e-9 ) {
          c.x = (a[i].x + a[j].x) / 2.0;
          c.y = (a[i].y + a[j].y) / 2.0;
          r = dis ( a[i] , a[j] ) / 2.0;
          tmp = r; tmp1 = c;
          for ( k = 1 ; k <= j - 1 ; k++ ) {
            if ( dis ( tmp1 , a[k] ) - tmp > 1e-9 ) {
              if ( abs(det ( a[i] , a[j] , a[k] )) < 1e-9 ) continue;
              tmp2 = getcen ( a[i] , a[j] , a[k] );
              tmp = dis ( tmp2 , a[i] );
              tmp1 = tmp2;
          }}
      c = tmp1; r = tmp;
  }}}}
  printf ( "%.3lf\n" , r );
}
```

11. 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：

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

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。

```
#define S 17000000  // for F(10^10)
static int pc, pr[S + 10];
static i64 phi[S + 10];
static unordered_map<i64, i64> dat;
inline void sub(i64 &a, i64 b) { a -= b; if (a < 0) a += MOD; }
inline i64 c2(i64 n) { n %= MOD; return n * (n + 1) % MOD * INV2 % MOD; }
i64 F(i64 n) {  // 杜教筛
  if (n <= S) return phi[n];
  if (dat.count(n)) return dat[n];
  i64 &r = dat[n] = c2(n);
  for (i64 i = 2, l; i <= n; i = l + 1) {
    i64 p = n / i;
    l = n / p;
    sub(r, (l - i + 1) * F(p) % MOD);  // (l - i + 1) % MOD?
  }
  return r;
}
phi[1] = 1;  // 线性筛
for (int i = 2; i <= S; i++) {
  if (!phi[i]) {
    pr[pc++] = i;
    phi[i] = i - 1;
  }
  for (int j = 0; pr[j] * i <= S; j++) {
    int p = pr[j];
    if (i % p) phi[i * p] = phi[i] * (p - 1);
    else {
      phi[i * p] = phi[i] * p;
      break;
}}}
for (int i = 2; i <= S; i++) add(phi[i], phi[i - 1]);
```

12. 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，$a,\ c \geqslant 1$ 且 $b \geqslant 0$。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 $\mathrm{O}(K^3 \log \max\{a,\ c\})$。

算法主要分为三个情况，其中 $a \geqslant c$ 和 $b \geqslant c$ 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 $\lfloor (c(j-1)+c-b-1)/a \rfloor < k \leqslant \lfloor (cj+c-b-1)/a \rfloor$，再进行一次分部求和即可。注意处理 $k \leqslant n$ 这个条件。

```
i64 F(i64 n, i64 a, i64 b, i64 c, int p, int q, int d = 0) {
  if (n < 0) return 0;
  if (has[d][p][q]) return val[d][p][q];
  has[d][p][q] = true;
  i64 &ret = val[d++][p][q] = 0;  // 后面的 d 均加 1
  if (!q) ret = S(n, p) + (!p);   // 注意 p = 0 的边界情况
  else if (!a) ret = qpow(b / c, q) * (S(n, p) + (!p)) % MOD;
  else if (a >= c) {
    i64 m = a / c, r = a % c, mp = 1;
    for (int j = 0; j <= q; j++, mp = mp * m % MOD)
      add(ret, C[q][j] * mp % MOD * F(n, r, b, c, p + j, q - j, d) % MOD);
  } else if (b >= c) {
    i64 m = b / c, r = b % c, mp = 1;
    for (int j = 0; j <= q; j++, mp = mp * m % MOD)
      add(ret, C[q][j] * mp % MOD * F(n, a, r, c, p, q - j, d) % MOD);
  } else {
    i64 m = (a * n + b) / c;
    for (int k = 0; k < q; k++) {
      i64 s = 0;
      for (int i = 1; i <= p + 1; i++)
        add(s, A[p][i] * F(m - 1, c, c - b - 1, a, k, i, d) % MOD);
      add(ret, C[q][k] * s % MOD);
    }
    ret = (qpow(m, q) * S(n, p) - ret) % MOD;
  } return ret;
}
```

13. 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。

```
static bool _initialized;
static int cnt;
static i64 _fi[KMAX + 10], _tmp[KMAX + 10];
struct PowerSeries {
  static void init() {
    _fi[0] = 1;
    for (int i = 2; i <= KMAX + 1; i++) _fi[0] = _fi[0] * i % MOD;
    _fi[KMAX + 1] = qpow(_fi[0], MOD - 2);
    for (int i = KMAX; i >= 0; i--) _fi[i] = _fi[i + 1] * (i + 1) % MOD;
    _initialized = true;
  }
  int K; i64 *f;
  PowerSeries() : PowerSeries(cnt++) {}
  PowerSeries(int _K) : K(_K) {
    if (!_initialized) init();
    f = new i64[K + 2]; f[0] = 0;
    for (int i = 1; i <= K + 1; i++) f[i] = (f[i - 1] + qpow(i, K)) % MOD;
  }
  ~PowerSeries() { delete[] f; }
  i64 operator()(i64 n) const {
    n %= MOD; _tmp[K + 2] = 1;
    for (int i = K + 1; i >= 1; i--) _tmp[i] = _tmp[i + 1] * (n - i) % MOD;
    i64 ret = 0, pre = 1;
    for (int i = 0, b = K & 1 ? 1 : -1; i <= K + 1; i++, b = -b) {
      add(ret, b * f[i] * pre % MOD * _tmp[i + 1] % MOD * _fi[i] % MOD * _fi[K + 1 - i] % MOD);
      pre = pre * (n - i) % MOD;
    } return ret;
  }
  i64 eval(i64 n) const { return (*this)(n); }
};
```

14. dinic

```
void add ( int u , int v , int f ) {
  node *tmp1 = &pool[++top] , *tmp2 = &pool[++top];
  tmp1 -> v = v; tmp1 -> f = f; tmp1 -> next = g[u]; g[u] = tmp1; tmp1 -> rev = tmp2;
  tmp2 -> v = u; tmp2 -> f = 0; tmp2 -> next = g[v]; g[v] = tmp2; tmp2 -> rev = tmp1;
}
bool makelevel () {
  int i , k;
  queue < int > q;
  for ( i = 1 ; i <= 1 + n + n + 1 ; i++ ) level[i] = -1;
  level[1] = 1; q.push ( 1 );
  while ( q.size () != 0 ) {
    k = q.front (); q.pop ();
    for ( node *j = g[k] ; j ; j = j -> next )
      if ( j -> f && level[j->v] == -1 ) {
        level[j->v] = level[k] + 1;
```

```
        q.push ( j -> v );
        if ( j -> v == 1 + n + n + 1 ) return true;
  }}
  return false;
}
int find ( int k , int key ) {
  if ( k == 1 + n + n + 1 ) return key;
  int i , s = 0;
  for ( node *j = g[k] ; j ; j = j -> next )
    if ( j -> f && level[j->v] == level[k] + 1 && s < key ) {
      i = find ( j -> v , min ( key - s , j -> f ) );
      j -> f -= i;
      j -> rev -> f += i;
      s += i;
    }
  if ( s == 0 ) level[k] = -1;
  return s;
}
void dinic () {
  int ans = 0;
  while ( makelevel () == true ) ans += find ( 1 , 99999 );
  //printf ( "%d\n" , ans );
  if ( ans == sum ) printf ( "^_^\n" );
  else printf ( "T_T\n" );
}
```

15. 费用流

```
void add ( int u , int v , int f , int c ) {
  node *tmp1 = &pool[++top] , *tmp2 = &pool[++top];
  tmp1 -> v = v; tmp1 -> f = f; tmp1 -> c = c; tmp1 -> next = g[u]; g[u] = tmp1; tmp1 -> rev = tmp2;
  tmp2 -> v = u; tmp2 -> f = 0; tmp2 -> c = -c; tmp2 -> next = g[v]; g[v] = tmp2; tmp2 -> rev = tmp1;
}
bool spfa () {
  int i , k;
  queue < int > q;
  for ( i = 1 ; i <= 1 + n*m*3 + 1 ; i++ ) dis[i] = 9999999, f[i] = 0;
  dis[1] = 0; f[1] = 1; q.push ( 1 );
  while ( q.size () != 0 ) {
    k = q.front (); q.pop (); f[k] = 0;
    for ( node *j = g[k] ; j ; j = j -> next )
      if ( j -> f && dis[j->v] > dis[k] + j -> c ) {
        dis[j->v] = dis[k] + j -> c;
        from[j->v] = j;
        if ( f[j->v] == 0 ) q.push ( j -> v );
        f[j->v] = 1;
      }
  }
  if ( dis[1+n*m*3+1] != 9999999 ) return true;
  return false;
}
int find () {
  int i , f = 999999 , s = 0;
  for ( i = 1+n*m*3+1 ; i != 1 ; i = from[i] -> rev -> v ) f = min ( f , from[i] -> f );
  flow += f;
  for ( i = 1+n*m*3+1 ; i != 1 ; i = from[i] -> rev -> v ) from[i] -> f -= f, from[i] -> rev -> f += f;
  return f * dis[1+n*m*3+1];
}
void dinic () {
  int ans = 0;
  while ( spfa () == true ) ans += find ();
  //printf ( "%d\n" , flow );
  if ( flow == sum && sum == sum1 ) printf ( "%d\n" , ans );
  else printf ( "-1\n" );
}
```

16. manacher

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

17. KMP

```
int p[101];
int main() {
  string a,b;
  cin>>a>>b;
  int n=a.length(),m=b.length();
  a=" "+a;b=" "+b;
  int j=0;
  for(int i=2;i<=m;i++) {
    while(j>0&&b[j+1]!=b[i])j=p[j];
    if(b[j+1]==b[i])j++;
    p[i]=j;
  }
  j=0;
  for(int i=1;i<=n;i++) {
    while(j>0&&b[j+1]!=a[i])j=p[j];
    if(b[j+1]==a[i])j++;
    if(j==m){printf("%d",i-m+1);break;}
  }
  return 0;
}
```

18. 回文自动机

```
int val[N];
int head[N],pos;
struct edge{int to,next;}e[N<<1];
void add(int a,int b)
{pos++;e[pos].to=b,e[pos].next=head[a],head[a]=pos;}
struct Tree {
  char ch[N];
  int now,cnt,odd,even;
  int fail[N],len[N],go[N][26];
  void init() {
    now=cnt=0;
    odd=++cnt,even=++cnt;
    len[odd]=-1,len[even]=0;
    fail[odd]=fail[even]=odd;
    now=even;add(odd,even);
  }
  void insert(int pos,char c) {
    while(ch[pos-1-len[now]]!=c)now=fail[now];
    if(!go[now][c-'a']) {
      go[now][c-'a']=++cnt;
      len[cnt]=len[now]+2;
      if(now==odd)fail[cnt]=even;
      else {
        int t=fail[now];
        while(ch[pos-1-len[t]]!=c)t=fail[t];
        fail[cnt]=go[t][c-'a'];
      }
      add(fail[cnt],cnt);
    }
    now=go[now][c-'a'];
    val[now]++;
  }
  void dfs(int u) {
    for(int i=head[u];i;i=e[i].next) {
      int v=e[i].to;
      dfs(v);
      val[u]+=val[v];
  }}
  long long cal() {
    long long ret=0;
    for(int i=3;i<=cnt;i++)
      ret=max(ret,1ll*len[i]*val[i]);
    return ret;
  }
}tree;
```

19. 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。

```
#define CH(i, n) i < n ? s[i] : 0
static int ch[NMAX + 10][3], seq[NMAX + 10];
static int arr[NMAX + 10], tmp[NMAX + 10], cnt[NMAX + 10];
inline bool cmp(int i, int j) {
  return ch[i][0] == ch[j][0] && ch[i][1] == ch[j][1] && ch[i][2] == ch[j][2];
}
inline bool sufcmp(int *s, int *rk, int n, int i, int j) {
  if (s[i] != s[j]) return s[i] < s[j];
  if ((i + 1) % 3 && (j + 1) % 3) return rk[i + 1] < rk[j + 1];
  if (s[i + 1] != s[j + 1]) return s[i + 1] < s[j + 1];
  return rk[i + 2] < rk[j + 2];
}
void radix_sort(int n, int m, int K, bool init = true) {
  if (init) for (int i = 0; i < n; i++) arr[i] = i;
  int *a = arr, *b = tmp;
  for (int k = 0; k < K; k++) {
    memset(cnt, 0, sizeof(int) * (m + 1));
    for (int i = 0; i < n; i++) cnt[ch[a[i]][k]]++;
    partial_sum(cnt, cnt + m + 1, cnt);
    for (int i = n - 1; i >= 0; i--) b[--cnt[ch[a[i]][k]]] = a[i];
    swap(a, b);
  }
  if (a != arr) memcpy(arr, tmp, sizeof(int) * n);
}
void suffix_sort(int *s, int n, int m, int *sa, int *rk) {
  s[n] = 0; n++;
  int p = 0, q = 0;
  for (int i = 1; i < n; i += 3, p++) for (int j = 0; j < 3; j++)
    ch[p][2 - j] = CH(i + j, n);
  for (int i = 2; i < n; i += 3, p++) for (int j = 0; j < 3; j++)
    ch[p][2 - j] = CH(i + j, n);
  radix_sort(p, m, 3);
  for (int i = 0; i < p; i++) {
    if (!q || (q && !cmp(arr[i - 1], arr[i]))) q++;
    s[n + arr[i]] = q;
  }
  if (q < p) suffix_sort(s + n, p, q, sa + n, rk + n);
  else {
    for (int i = 0; i < p; i++) sa[n + s[n + i] - 1] = i;
    for (int i = 0; i < p; i++) rk[n + sa[n + i]] = i + 1;
  }
  m = max(m, p);
  p = q = 0;
  for (int i = 1; i < n; i += 3, p++) rk[i] = rk[n + p];
  for (int i = 2; i < n; i += 3, p++) rk[i] = rk[n + p];
  for (int i = 0; i < n; i++) if (i % 3) seq[rk[i] - 1] = i;
  for (int i = 0; i < n; i += 3, q++) {
    ch[i][0] = i + 1 < n ? rk[i + 1] : 0;
    ch[i][1] = s[i];
    arr[q] = i;
  }
  radix_sort(q, m, 2, false);
  for (int i = seq[0] == n - 1, j = arr[0] == n - 1, k = 0; i < p || j < q; k++) {
    if (i == p) sa[k] = arr[j++];
    else if (j == q) sa[k] = seq[i++];
    else if (sufcmp(s, rk, n, seq[i], arr[j])) sa[k] = seq[i++];
    else sa[k] = arr[j++];
  }
  for (int i = 0; i < n - 1; i++) rk[sa[i]] = i + 1;
}
```

20. 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。

```
struct Node {
  Node() : mark(false), suf(NULL), nxt(NULL) {
    memset(ch, 0, sizeof(ch));
  }
  bool mark;
  Node *suf, *nxt, *ch[SIGMA];
};
void insert(Node *x, char *s) {
  for (int i = 0; s[i]; i++) {
    int c = s[i] - 'a';
    if (!x->ch[c]) x->ch[c] = new Node;
    x = x->ch[c];
  }
  x->mark = true;
}
void build_automaton(Node *r) {
  queue<Node *> q;
  for (int c = 0; c < SIGMA; c++) {
    if (!r->ch[c]) continue;
    r->ch[c]->suf = r;
    q.push(r->ch[c]);
  }
  while (!q.empty()) {
    Node *x = q.front();
    q.pop();
    for (int c = 0; c < SIGMA; c++) {
      Node *v = x->ch[c]; if (!v) continue;
      Node *y = x->suf;
      while (y != r && !y->ch[c]) y = y->suf;
      if (y->ch[c]) y = y->ch[c];
      v->suf = y;
      if (y->mark) v->nxt = y;
      else v->nxt = y->nxt;
      q.push(v);
}}}
void search(Node *x, char *s) {
  for (int i = 0; s[i]; i++) {
    int c = s[i] - 'a';
    while (x->suf && !x->ch[c]) x = x->suf;
    if (x->ch[c]) x = x->ch[c];
    if (x->mark) print(i + 1, x->data);
    for (Node *y = x->nxt; y; y = y->nxt) print(i + 1, y->data);
}}
```

21. 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 $\Sigma=\{0,1,2,\ldots,m\}$，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。

```
static int sa[NMAX + 10], rk[NMAX + 10], lcp[NMAX + 10];
void suffix_sort(const char *s, int n, int m) {
  static int x[NMAX + 10], y[NMAX + 10], cnt[NMAX + 10], i;
  for (i = 0; i < n; i++) cnt[s[i]]++;
  for (i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
  for (i = 0; i < n; i++) sa[--cnt[s[i]]] = i;
  for (i = 1, m = 1, rk[sa[0]] = 1; i < n; i++) {
    if (s[sa[i - 1]] != s[sa[i]]) m++;
    rk[sa[i]] = m;
  }
  for (int l = 1; l < n; l <<= 1) {
    memset(cnt, 0, sizeof(int) * (m + 1));
    for (i = 0; i < n; i++) cnt[y[i] = i + l < n ? rk[i + l] : 0]++;
    for (i = 1; i <= m; i++) cnt[i] += cnt[i - 1
```

```
];
    for (i = n - 1; i >= 0; i--) x[--cnt[y[i]]] = i;
    memset(cnt, 0, sizeof(int) * (m + 1));
    for (i = 0; i < n; i++) cnt[rk[i]]++;
    for (i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
    for (i = n - 1; i >= 0; i--) sa[--cnt[rk[x[i]]]] = x[i];
    for (i = 1, m = 1, x[sa[0]] = 1; i < n; i++) {
      if (rk[sa[i - 1]] != rk[sa[i]] || y[sa[i - 1]] != y[sa[i]]) m++;
      x[sa[i]] = m;
    }
    memcpy(rk, x, sizeof(int) * n);
}}
void compute_lcp(const char *s, int n) {
  int j = 0, p;
  for (int i = 0; i < n; i++, j = max(0, j - 1)) {
    if (rk[i] == 1) {
      j = 0;
      continue;
    }
    p = sa[rk[i] - 2];
    while (p + j < n && i + j < n && s[p + j] == s[i + j]) j++;
    lcp[rk[i]] = j;
}}
```

22. 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。

```
#define rep(i, l, r) for (register int i = (l); i <= (r); ++i)
#define rrep(i, r, l) for (register int i = (r); i >= (l); --i)
#define PUTS(x) sa[cur[str[x]]--] = x
#define PUTL(x) sa[cur[str[x]]++] = x
#define LMS(x) (!type[x - 1] && type[x])
#define RESET memset(sa + 1, 0, sizeof(int) * (n + 1));
  memcpy(cur + 1, cnt + 1, sizeof(int) * m);
#define INDUCE rep(i, 1, m) cur[i] = cnt[i - 1] + 1;
  rep(i, 1, n + 1) if (sa[i] > 1 && !type[sa[i] - 1]) PUTL(sa[i] - 1);
  memcpy(cur + 1, cnt + 1, sizeof(int) * m);
  rrep(i, n + 1, 1) if (sa[i] > 1 && type[sa[i] - 1]) PUTS(sa[i] - 1);
void sais(int n, int m, int *str, int *sa) {
  static int id[NMAX + 10];
  vector<bool> type(n + 2);
  type[n + 1] = true;
  rrep(i, n, 1) type[i] = str[i] == str[i + 1] ? type[i + 1] : str[i] < str[i + 1];
  int cnt[m + 1], cur[m + 1], idx = 1, y = 0, rt, lrt, *ns = str + n + 2, *nsa = sa + n + 2;
  memset(cnt, 0, sizeof(int) * (m + 1));
  rep(i, 1, n + 1) cnt[str[i]]++;
  rep(i, 1, m) cnt[i] += cnt[i - 1];
  RESET rep(i, 2, n + 1) if (LMS(i)) PUTS(i); INDUCE
  memset(id + 1, 0, sizeof(int) * n);
  rep(i, 2, n + 1) if (LMS(sa[i])) {
    register int x = sa[i];
    for (rt = x + 1; !LMS(rt); rt++) ;
    id[x] = y && rt + y == lrt + x && !memcmp(str + x, str + y, sizeof(int) * (rt - x + 1)) ? idx : ++idx;
    y = x, lrt = rt;
  }
  int len = 0, pos[(n >> 1) + 1];
  rep(i, 1, n) if (id[i]) {
    ns[++len] = id[i];
    pos[len] = i;
  }
  ns[len + 1] = 1, pos[len + 1] = n + 1;
  if (len == idx - 1) rep(i, 1, len + 1) nsa[ns[i]] = i;
  else sais(len, idx, ns, nsa);
  RESET rrep(i, len + 1, 1) PUTS(pos[nsa[i]]); INDUCE
}
static int str[NMAX * 3 + 10], sa[NMAX * 3 + 10];
```

23. 后缀树

```
#define SIGMA 27
#define EOFCHAR 26
static char str[NMAX + 10];
struct Node;
struct Edge {
  Edge(int l, int r, Node *nxt) : left(l), right(r), nxt(nxt) {}
  int left, right; Node *nxt;
  int length() const { return right - left + 1; }
};
struct Node {
  Node() : fail(NULL) { memset(trans, 0, sizeof(trans)); }
  Node *fail; Edge *trans[SIGMA];
};
struct BuildState {
  BuildState(Node *_r) : r(_r), ap(_r), dir(-1), len(0), remain(0), pos(0) {}
  Node *r, *ap; char dir; int len, remain, pos;
};
void append(char c, BuildState &state) {
  auto &cur = state.ap; auto &dir = state.dir;
  auto &len = state.len; auto &cnt = state.remain;
  auto &size = state.pos; cnt++; size++;
  Node *last = NULL, *x;
  while (cnt) {
    if (cnt - 1 <= len) {
      len = cnt - 1;
      dir = len ? str[size - len] : -1;
    }
    while (dir >= 0 && len >= cur->trans[dir]->length()) {
      len -= cur->trans[dir]->length();
      cur = cur->trans[dir]->nxt;
      dir = len ? str[size - len] : -1;
    }
    if ((dir >= 0 && c == str[cur->trans[dir]->left + len]) ||
      (dir < 0 && cur->trans[c])) {
      if (dir < 0) dir = c;
      if (last) last->fail = cur;
      len++;
      return;
    }
    if (dir < 0) {
      cur->trans[c] = new Edge(size, n, new Node);
      x = cur;
    } else {
      Edge *t = cur->trans[dir];
      x = new Node;
      x->trans[c] = new Edge(size, n, new Node);
      x->trans[str[t->left + len]] = new Edge(t->left + len, t->right, t->nxt);
      t->right = t->left + len - 1;
      t->nxt = x;
    }
    if (last) last->fail = x;
    last = x; cnt--;
    cur = cur->fail ? cur->fail : state.r;
}}
void initialize() {
  scanf("%d%s", &n, str + 1);
  tree = new Node;
  BuildState state(tree);
  for (int i = 1; i <= n; i++) append(str[i], state);
  append(EOFCHAR, state);
  tree->trans[EOFCHAR] = NULL;
}
```

24. pam

```
const int NN = 310000;
struct node {
```

```
  int len , cnt,ch[30] , fail;
} p[NN];
int top,n,last;
char z[NN];
long long ans;
void work () {
  int i , tmp;
  scanf ( "%s" , z + 1 );
  n = strlen ( z + 1 );
  top = 2;
  p[1].fail = 2; p[2].fail = 1;
  p[1].len = 0; p[2].len = -1;
  z[0] = '$';
  last = 1;
  for ( i = 1 ; i <= n ; i++ ) {
    while ( z[i] != z[i-p[last].len-1] ) last = p[last].fail;
    if ( !p[last].ch[z[i]-'a'+1] ) {
      p[last].ch[z[i]-'a'+1] = ++top;
      p[top].len = p[last].len + 2;
      tmp = p[last].fail;
      while ( z[i] != z[i-p[tmp].len-1] ) tmp = p[tmp].fail;
      if ( p[top].len > 1 && p[tmp].ch[z[i]-'a'+1] ) p[top].fail = p[tmp].ch[z[i]-'a'+1];
      else p[top].fail = 1;
    }
    last = p[last].ch[z[i]-'a'+1];
    p[last].cnt++;
  }
  for ( i = top ; i >= 1 ; i-- ) p[p[i].fail].cnt += p[i].cnt;
  for ( i = 1 ; i <= top ; i++ ) {
    //printf ( "%d %d\n" , p[i].len , p[i].cnt );
    ans = max ( ans , (long long)p[i].len * p[i].cnt );
  }
  printf ( "%lld\n" , ans );
}
```

25. 权值splay

```
ll n,kind,rt,sz,fa[N],num[N];
ll tr[N][2],size[N],v[N],ans;
void pushup(ll k){size[k]=size[tr[k][0]]+size[tr[k][1]]+num[k];}
void rotate(ll x,ll &k) {
  ll y=fa[x],z=fa[y],l,r;
  l=tr[y][1]==x;r=l^1;
  if(y==k)k=x;
  else tr[z][tr[z][1]==y]=x;
  fa[x]=z,fa[tr[x][r]]=y,fa[y]=x;
  tr[y][l]=tr[x][r],tr[x][r]=y;
  pushup(y);pushup(x);
}
void splay(ll x,ll &k) {
  while(x!=k) {
    ll y=fa[x],z=fa[y];
    if(y!=k) {
      if(tr[y][0]==x^tr[z][0]==y)
        rotate(x,k);
      else rotate(y,k);
    }rotate(x,k);
}}
void insert(ll &k,ll x,ll last) {
  if(!k){k=++sz;v[k]=x;size[k]=num[k]=1;fa[k]=last;splay(k,rt);return ;}
  if(x==v[k])num[k]++;
    else if(x>v[k])insert(tr[k][1],x,k);
  else insert(tr[k][0],x,k);
}
ll t1,t2;
ll find(ll x,ll k) {
  if(!k)return 0;
  if(x==v[k])return k;
  else if(x>v[k])return find(x,tr[k][1]);
  else return find(x,tr[k][0]);
}
void ask_before(ll x,ll k) {
  if(!k)return ;
  if(v[k]<x){t1=k;ask_before(x,tr[k][1]);}
  else ask_before(x,tr[k][0]);
}
void ask_after(ll x,ll k) {
  if(!k)return ;
  if(v[k]>x){t2=k;ask_after(x,tr[k][0]);}
//  else if(v[k]==x)return ;
  else ask_after(x,tr[k][1]);
}
void del(ll x,ll k) {
  if(num[k]>1) {
    num[k]--,size[k]--;
    splay(k,rt);return;
  }
  t1=t2=-1;
  ask_before(x,rt);
  ask_after(x,rt);
  if(t1==-1&&t2==-1) {
    if(num[rt]==1)rt=0;
    else size[rt]--,num[rt]--;
  }
  else if(t1==-1) {
    splay(t2,rt);
    tr[rt][0]=0;
    pushup(rt);
  }
  else if(t2==-1) {
    splay(t1,rt);
    tr[rt][1]=0;
    pushup(rt);
  }
  else {
    splay(t1,rt);
    splay(t2,tr[t1][1]);
    tr[t2][0]=0;
    pushup(t2);pushup(t1);
}}
```

26. 序列splay

```
int n,m,sz,rt;
char ch[10];
int tr[N][2],fa[N],v[N],sum[N];
int mx[N],lx[N],rx[N];
int st[N],size[N],top,tag[N];
bool rev[N];
void pushup(int u) {
  size[u]=1,sum[u]=v[u];int l=tr[u][0],r=tr[u][1];
  if(l)size[u]+=size[l],sum[u]+=sum[l];
  if(r)size[u]+=size[r],sum[u]+=sum[r];
  mx[u]=v[u];if(l)mx[u]=max(mx[u],mx[l]);if(r)mx[u]=max(mx[u],mx[r]);
  if(l&&r)mx[u]=max(mx[u],rx[l]+v[u]+lx[r]);
  else if(l)mx[u]=max(mx[u],rx[l]+v[u]);
  else if(r)mx[u]=max(mx[u],v[u]+lx[r]);
  lx[u]=0;if(l)lx[u]=lx[l];rx[u]=0;if(r)rx[u]=rx[r];
  if(!l)lx[u]=max(lx[u],v[u]);if(!r)rx[u]=max(rx[u],v[u]);
  if(!l&&r)lx[u]=max(lx[u],v[u]+lx[r]);if(!r&&l)rx[u]=max(rx[u],v[u]+rx[l]);
  if(l)lx[u]=max(lx[u],sum[l]+v[u]);if(r)rx[u]=max(rx[u],sum[r]+v[u]);
  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]);
}
void work(int k,int c) {
  tag[k]=c,v[k]=c,sum[k]=size[k]*c;
  mx[k]=(c>0?c*size[k]:c),lx[k]=rx[k]=(c>0?c*size[k]:0);
}
void rve(int k) {
  rev[k]^=1;
  swap(lx[k],rx[k]);
  swap(tr[k][0],tr[k][1]);
}
void pushdown(int u) {
  int l=tr[u][0],r=tr[u][1];
  if(tag[u]!=12345) {
    if(l)work(l,tag[u]);if(r)work(r,tag[u]);
    tag[u]=12345;
  }
  if(rev[u]) {
    if(l)rve(l);if(r)rve(r);
    rev[u]^=1;
}}
void rotate(int x,int &k) {
  int y=fa[x],z=fa[y];
  int l=(tr[y][1]==x),r=l^1;
  if(y==k)k=x;
```

```
  else tr[z][tr[z][1]==y]=x;
  fa[x]=z,fa[y]=x,fa[tr[x][r]]=y;
  tr[y][l]=tr[x][r],tr[x][r]=y;
  pushup(y);pushup(x);
}
void splay(int x,int &k) {
  while(x!=k) {
    int y=fa[x],z=fa[y];
    if(y!=k) {
      if(tr[y][0]==x^tr[z][0]==y)
        rotate(x,k);
      else rotate(y,k);
    }
    rotate(x,k);
}}
int find(int k,int rk) {
  pushdown(k);
  int l=tr[k][0],r=tr[k][1];
  if(size[l]>=rk)return find(l,rk);
  else if(size[l]+1==rk)return k;
  else return find(r,rk-size[l]-1);
}
int split(int l,int r) {
  int x=find(rt,l),y=find(rt,r+2);
  splay(x,rt),splay(y,tr[x][1]);
  return tr[y][0];
}
int a[N];
void newnode(int k,int c)
{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;}
int build(int l,int r) {
  if(l>r)return 0;int mid=(l+r)>>1,now;
  now=++sz;newnode(now,a[mid-1]);
  tr[now][0]=build(l,mid-1);if(tr[now][0])fa[tr[now][0]]=now;
  tr[now][1]=build(mid+1,r);if(tr[now][1])fa[tr[now][1]]=now;
  pushup(now);return now;
}
int Build(int l,int r) {
  if(l>r)return 0;int mid=(l+r)>>1,now;
  if(top)now=st[top--];else now=++sz;newnode(now,a[mid]);
  tr[now][0]=Build(l,mid-1);if(tr[now][0])fa[tr[now][0]]=now;
  tr[now][1]=Build(mid+1,r);if(tr[now][1])fa[tr[now][1]]=now;
  pushup(now);return now;
}
void insert(int x,int tot) {
  for(int i=0;i<=tot+2;i++)a[i]=0;
  for(int i=1;i<=tot;i++)a[i]=read();
  int l=find(rt,x+1),r=find(rt,x+2);
  splay(l,rt),splay(r,tr[l][1]);
  tr[r][0]=Build(1,tot),fa[tr[r][0]]=r;
  pushup(r),splay(r,rt);
}
void clr(int k){tag[k]=12345,tr[k][0]=tr[k][1]=fa[k]=rev[k]=v[k]=sum[k]=mx[k]=lx[k]=rx[k]=size[k]=0;}
void rec(int k) {
  if(!k)return;
  rec(tr[k][0]);rec(tr[k][1]);
  st[++top]=k,clr(k);
}
void del(int x,int tot) {
  int l=x,r=x+tot-1,k=split(l,r);
  int fk=fa[k];tr[fk][0]=fa[k]=0;rec(k);
  splay(fk,rt);
}
void make_same(int x,int tot,int c)
{int l=x,r=x+tot-1,k=split(l,r);work(k,c);if(fa[k])splay(fa[k],rt);}
void rever(int x,int tot)
{int l=x,r=x+tot-1,k=split(l,r);rve(k);if(fa[k])splay(fa[k],rt);}
int get_sum(int x,int tot) {
  int l=x,r=x+tot-1,k=split(l,r);
  return sum[k];
}
```

27. ntt

```
const long long maxn = 120000;
const long long mod = 998244353;
const long long omega = 3;
long long a[maxn*4] , b[maxn*4] , c[maxn*4] , d[maxn*4];
long long n , m , N , in;
long long pow ( long long f , long long x ) {long long s = 1;while ( x ) {if ( x % 2 ) s = (s*f) % mod;f = (f*f) % mod; x >>= 1;}return s;}
long long inv ( long long x ) {return pow ( x , mod - 2 );}
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;}
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];}
void FFT ( long long *x , long long f ) {
  long long i , j , s , k;
  long long w , wm , u , t;
  br ( x );
  for ( s = 2 ; s <= N ; s *= 2 ) {
    k = s / 2;
    wm = pow ( omega , (mod-1) / s );
    if ( f == -1 ) wm = inv ( wm );
    for ( i = 0 ; i < N ; i += s ) {
      w = 1;
      for ( j = 1 ; j <= k ; j++ ) {
        u = x[i+j-1]; t = (x[i+j-1+k]*w) % mod;
        x[i+j-1] = (u + t) % mod;
        x[i+j-1+k] = (u - t + mod) % mod;
        w = (w*wm) % mod;
  }}}
  if ( f == -1 ) for ( i = 0 ; i < N ; i++ ) x[i] = (x[i] * in) % mod;
}
void work () {
  long long i;
  scanf ( "%lld%lld" , &n , &m );
  N = 1;
  while ( N < n + m + 2 ) N = N * 2;
  for ( i = 0 ; i <= n ; i++ ) scanf ( "%lld" , &a[i] );
  for ( i = 0 ; i <= m ; i++ ) scanf ( "%lld" , &b[i] );
  in = inv ( N );
  FFT ( a , 1 ); FFT ( b , 1 );
  for ( i = 0 ; i < N ; i++ ) c[i] = (a[i]*b[i]) % mod;
  FFT ( c , -1 );
  for ( i = 0 ; i <= n + m ; i++ ) printf ( "%lld%c" , c[i] , i==n+m?'\n':' ' );
}
```

28. fft

```
const int maxn = 120000;
const double pi = acos(-1);
struct complex {
  double r , i;
} a[maxn*4] , b[maxn*4] , c[maxn*4] , d[maxn*4];
complex operator + ( complex x1 , complex x2 ) {complex y;y.r = x1.r + x2.r;y.i = x1.i + x2.i;return y;}
complex operator - ( complex x1 , complex x2 ) {complex y;y.r = x1.r - x2.r;y.i = x1.i - x2.i;return y;}
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;}
int n , m , N;
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;}
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];}
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;
}
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':' ' );
}
```

29. lct

```
struct node {
  long long x;
  long long lm , lp , rev;
  long long s , siz;
  long long ch[4] , fa;
} p[maxn];
void cut ( long long x , long long kind ) {
  p[p[x].ch[kind]].fa *= -1;
  p[x].ch[kind] = 0;
  update ( x );
}
void down ( long long x ) {
  if ( p[x].fa > 0 ) down ( p[x].fa );
  pushdown ( x );
}
void rotate ( long long x , long long kind ) {
  long long y = p[x].fa;
  if ( p[y].fa > 0 ) p[p[y].fa].ch[y==p[p[y].fa].ch[1]] = x;
  p[x].fa = p[y].fa;
  if ( p[x].ch[kind^1] ) p[p[x].ch[kind^1]].fa = y;
  p[y].ch[kind] = p[x].ch[kind^1];
  p[y].fa = x;
  p[x].ch[kind^1] = y;
  update ( y ); update ( x );
}
void splay ( long long x ) {
  down ( x );
  for ( ; p[x].fa > 0 ; rotate ( x , x==p[p[x].fa].ch[1]) )
    if ( p[p[x].fa].fa > 0 && (x==p[p[x].fa].ch[1]) == (p[x].fa==p[p[p[x].fa].fa].ch[1]) )
      rotate ( p[x].fa , x==p[p[x].fa].ch[1] );
}
void access ( long long x ) {
  splay ( x );
  cut ( x , 1 );
  for ( ; p[x].fa != 0 ; ) {
    splay ( -p[x].fa );
    cut ( -p[x].fa , 1 );
    p[-p[x].fa].ch[1] = x;
    update ( -p[x].fa );
    p[x].fa *= -1;
    splay ( x );
}}
void makeroot ( long long x ) {
  access ( x );
  p[x].rev ^= 1;
  swap ( p[x].ch[0] , p[x].ch[1] );
}
void link ( long long x , long long y ) {
  makeroot ( y );
  p[y].fa = -x;
}
```

30. 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。

```
struct node {
  int x , i , dist;
  node *ll , *rr;
} pool[maxn] , *t[maxn];
int n , m;
int a[maxn];
int c[maxn] , f[maxn];
int getdist ( node *id ) {
  if ( id == NULL ) return -1;
  return id -> dist;
}
node *merge ( node *id1 , node *id2 ) {
  if ( id1 == NULL ) return id2;
  if ( id2 == NULL ) return id1;
  if ( id1 -> x > id2 -> x ) swap ( id1 , id2 );
  id1 -> rr = merge ( id1 -> rr , id2 );
  if ( getdist ( id1 -> ll ) < getdist ( id1 -> rr ) ) swap ( id1 -> ll , id1 -> rr );
  id1 -> dist = getdist ( id1 -> rr ) + 1;
  return id1;
}
int find ( int x ) {
  int i , t;
  for ( i = x ; c[i] > 0 ; i = c[i] ) ;
  while ( x != i ) {
    t = c[x];
    c[x] = i;
    x = t;
  }
  return i;
}
void Union ( int x , int y ) {
  t[x] = merge ( t[x] , t[y] );
  c[x] += c[y];
  c[y] = x;
}
```

31. 三分_上凸函数

```
double solve() {
  while(l+eps<r) {
    double mid=(l+r)/2.0;
    double mmid=(mid+r)/2.0;
    if(cal(mid)>cal(mmid))r=mmid;
    else l=mid;
  }
  if(cal(l)<cal(r))return r;
  else return l;
}
```

32. 线性空间求交

设两个线性空间 U、V 的基分别为 $u_1, u_2, ..., u_n$ 和 $v_1, v_2, ..., v_m$。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 $v_1, ..., v_m, u'_1, ..., u'_j$，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。

```
#define SMAX 32
typedef unsigned int u32;
struct Basis {
  u32 v[SMAX];
  auto operator[](const size_t i) -> u32& {
    return v[i];
}};
auto intersect(Basis &u, Basis v) -> Basis {
  Basis z, r;
  for (int i = 0; i < SMAX; i++) if (u[i]) {
    u32 x = u[i], y = u[i];
    for (int j = 0; j < SMAX; j++) if ((x >> j) & 1) {
      if (v[j]) x ^= v[j], y ^= r[j];
      else {
        v[j] = x, r[j] = y;
        break;
    }}
    if (!x) z.add(y);
  }
  return z;
}
```

33. 单纯型

```
#define EPS 1e-10
```

```cpp
#define INF 1e100

class Simplex {
 public:
  void initialize() {
    scanf("%d%d%d", &n, &m, &t);
    memset(A, 0, sizeof(A));
    for (int i = 1; i <= n; i++) {
      idx[i] = i;
      scanf("%Lf", A[0] + i);
    }
    for (int i = 1; i <= m; i++) {
      idy[i] = n + i;
      for (int j = 1; j <= n; j++) {
        scanf("%Lf", A[i] + j);
        A[i][j] *= -1;
      }
      scanf("%Lf", A[i]);
  }}
  void solve() {
    srand(time(0));
    while (true) {
      int x = 0, y = 0;
      for (int i = 1; i <= m; i++)
        if (A[i][0] < -EPS && (!y || (rand() & 1))) y = i;
      if (!y) break;
      for (int i = 1; i <= n; i++)
        if (A[y][i] > EPS && (!x || (rand() & 1))) x = i;
      if (!x) {
        puts("Infeasible");
        return;
      }
      pivot(x, y);
    }
    while (true) {
      double k = INF;
      int x, y;
      for (x = 1; x <= n; x++)
     if (A[0][x] > EPS) break;
      if (x > n) break;
      for (int i = 1; i <= m; i++) {
        double d = A[i][x] > -EPS ? INF : -A[i][0] / A[i][x];
        if (d < k) {
          k = d;
          y = i;
      }}
      if (k >= INF) {
        puts("Unbounded");
        return;
      }
      pivot(x, y);
    }
    printf("%.10Lf\n", A[0][0]);
    if (t) {
      static double ans[NMAX + 10];
      for (int i = 1; i <= m; i++)
        if (idy[i] <= n) ans[idy[i]] = A[i][0];
      for (int i = 1; i <= n; i++)
        printf("%.10Lf ", ans[i]);
      printf("\n");
  }}
 private:
  void pivot(int x, int y) {
    swap(idx[x], idy[y]);
    double r = -A[y][x];
    A[y][x] = -1;
    for (int i = 0; i <= n; i++) A[y][i] /= r;
    for (int i = 0; i <= m; i++) {
      if (i == y) continue;
      r = A[i][x];
      A[i][x] = 0;
      for (int j = 0; j <= n; j++)
        A[i][j] += r * A[y][j];
  }}
  int n, m, t;
  double A[NMAX + 10][NMAX + 10];
  int idx[NMAX + 10], idy[NMAX + 10];
};
```

◦ sll/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->i,c=M[i] (M[i]>=0)

i->T,c=-M[i]

流程:

S->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->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->SD跑最大流,判断是否满流,不满流则无解
2. 去掉ST,SD,从S->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->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->v),建边(u->v,inf)

对于c[u]>0 建边(s->u,c[u])

对于c[u]<0 建边(u->t,-c[u])

流程: 建图后跑s->t的最小割, $\sum c_u$-最小割即为答案

◦ xzl/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。

◦ xzl/fwt.md ■

FWT 算法：分治 $A \to A_1, A_2$，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/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

- 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。

- lmj/idea.md ■

启发式合并

离线

hash

数据结构上跑图论算法