---

1. block_forest_data_structure [lmj/block_forest_data_stru…

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。

```
struct node {
  int v , u; node *next;
} pooln[maxn*4] , *gn[maxn];
struct tree {
  int v; tree *next;
} poolt[maxn*4] , *gt[maxn*2];
int topt , topn;
int n , m , tot;
int kind[maxn*2] , dfn[maxn] , low[maxn] , index;
stack <node*> st;
void add ( int u , int v ) {
  node *tmp = &pooln[++topn];
  tmp -> v = v; tmp -> u = u; tmp -> next = gn[u]; gn[u] = tmp;
}
void addt ( int u , int v ) {
  tree *tmp = &poolt[++topt];
  tmp -> v = v; tmp -> next = gt[u]; gt[u] = tmp;
}
void tarjan ( int i , int from ) {
  dfn[i] = low[i] = ++index;
  for ( node *j = gn[i] ; j ; j = j -> next ) if ( j -> v != from ) {
    if ( !dfn[j->v] || dfn[i] > dfn[j->v] ) st.push(j);
    if ( !dfn[j->v] ) {
      tarjan ( j -> v , i );
      low[i] = min ( low[i] , low[j->v] );
      if ( low[j->v] >= dfn[i] ) {
        if ( st.top() == j ) {
          addt ( i , j -> v , j -> prob );
          addt ( j -> v , i , j -> prob );
          st.pop();
        } else {
          tot++;
          kind[tot] = 1;
          while ( st.top() != j ) {
            node *k = st.top ();
            st.pop();
            addt ( tot , k -> u , k -> prob );
            addt ( k -> u , tot , k -> prob );
          }
          addt ( tot , i , j -> prob );
          addt ( i , tot , j -> prob );
          st.pop();
    }}}
    else low[i] = min ( low[i] , dfn[j->v] );
}}
void work () {
  int i , u , v , a , b;
  scanf ( "%d%d" , &n , &m );
  for ( i = 1 ; i <= m ; i++ ) {
    scanf ( "%d%d%d%d" , &u , &v , &a , &b );
    add ( u , v );
    add ( v , u );
  }
  tot = n;
  for ( i = 1 ; i <= n ; i++ ) kind[i] = 0;
  tarjan ( 1 , -1 );
}
```

2. blossom algorithm [lmj/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':' ' );
}
```

3. euler_tour [lmj/euler_tour.cpp]

```
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 ();
  }
}
```

4. 仙人掌 DP [xzl/仙人掌 DP，图论.cpp]

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。

```
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]);
}}
```

5. 倍增lca [sll/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];
}
```

6. 有向图强联通 tarjan [sll/tarjan(SCC).cpp]

```
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() {
  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);
}
```

7. 构造圆方树 [xzl/biconnected.cpp]

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。

```
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
}}}}
```

8. 点双联通 tarjan [sll/点双连通分量.cpp]

```
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]);
}}}
```

9. 边双联通 tarjan [sll/边双连通分量.cpp]

```
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;
}
```

10. 最小树形图：朴素算法 [xzl/mdst-nm.cpp]

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。

```
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;
}
```

11. 最小树形图：Tarjan 算法 [xzl/最小树形图：Tarjan 算法...

使用可并堆优化的 Chu-Liu 算法，这里使用左偏树。in 存储原图的入边。contract 会生成一棵 contraction 树，树根为 n。Contraction 树上每个节点的所有儿子构成一个环，环上每个点的

入边存放在 ed 内。使用 expand(r, n) 从节点 r 处展开以 r 为根的最小树形图，如果返回 INF 则表示不存在树形图。contract 过程会增加节点并且改动边权，故使用 w0 保存原始边权。注意点数 n 应该开到两倍。重复使用时注意收缩完后 fa[n] 和 nxt[n] 应置 0 。contract 时间复杂度为 $\mathrm{O}(m\log n)$，expand 时间复杂度为 $\Theta(n)$ ，实测随机数据下只有边数 m 达到 5×10^5 级别时才比朴素算法快。

```
#define INF 0x3f3f3f3f
struct Edge { int u, v, w, w0; };
struct Heap {
  Heap(Edge *_e) : e(_e), rk(1), sum(0), lch(NULL), rch(NULL) {}
  Edge *e; int rk, sum;
  Heap *lch, *rch;
  void push() {
    if (lch) lch->sum += sum;
    if (rch) rch->sum += sum;
    e->w += sum; sum = 0;
}};
inline Heap *meld(Heap *x, Heap *y) {
  if (!x) return y;
  if (!y) return x;
  if (x->e->w + x->sum > y->e->w + y->sum)
    swap(x, y);
  x->push();
  x->rch = meld(x->rch, y);
  if (!x->lch || x->lch->rk < x->rch->rk)
    swap(x->lch, x->rch);
  x->rk = x->rch ? x->rch->rk + 1 : 1;
  return x;
}
inline Edge *extract(Heap *&x) {
  Edge *r = x->e;
  x->push();
  x = meld(x->lch, x->rch);
  return r;
}
static vector<Edge> in[NMAX + 10];
static int n, m, fa[2 * NMAX + 10], nxt[2 * NMAX + 10];
static Edge *ed[2 * NMAX + 10];
static Heap *Q[2 * NMAX + 10];
static UnionFind id; // id[] & id.fa
void contract() {
  static bool mark[2 * NMAX + 10];
  //memset(mark + 1, 0, 2 * n);
  //id.clear(2 * n);
  for (int i = 1; i <= n; i++) {
    queue<Heap*> q;
    for (int j = 0; j < in[i].size(); j++)
      q.push(new Heap(&in[i][j]));
    while (q.size() > 1) {
      Heap *u = q.front(); q.pop();
      Heap *v = q.front(); q.pop();
      q.push(meld(u, v));
    } Q[i] = q.front();
  } mark[1] = true;
  for (int u = 1, u0 = 1, p; Q[u]; mark[u0 = u] = true) {
    do u = id[(ed[u] = extract(Q[u]))->u];
    while (u == u0 && Q[u]);
    if (u == u0) break;
    if (!mark[u]) continue;
    for (u0 = u, n++; u != n; u = p) {
      id.fa[u] = fa[u] = n;
      if (Q[u]) Q[u]->sum -= ed[u]->w;
      Q[n] = meld(Q[n], Q[u]);
      p = id[ed[u]->u];
      nxt[p == n ? u0 : p] = u;
}}}
i64 expand(int, int);
i64 _expand(int x) {
  i64 r = 0;
  for (int u = nxt[x]; u != x; u = nxt[u])
    if (ed[u]->w0 >= INF) return INF;
    else r += expand(ed[u]->v, u) + ed[u]->w0;
  return r;
}
i64 expand(int x, int t) {
  i64 r = 0;
  for ( ; x != t; x = fa[x])
    if ((r += _expand(x)) >= INF) return INF;
  return r;
}
//contract();
//i64 ans = expand(rt, n);
```

12. 最小圆覆盖 [lmj/minimal_circle_cover.cpp]

```
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 );
}
```

13. Pohlig-Hellman 离散对数 [xzl/Pohlig-Hellman 离散对数...

Pohlig-Hellman 离散对数算法，求解同余方程 $a^x\equiv b \pmod m$ 的最小解 x 或者报告无解，要求 m 为质数。ord 用于求出 a 关于 m 的阶数。算法需要实现快速幂 qpow(a, k, m)、快速乘 qmul(a, b, m)、素数判定 isprime(n) 和使用扩展 Euclid 算法

求出的逆元 inv(x, m)。p0、k0、c0 存放的是 $m-1$ 的质因数分解，p1、k1、c1 存放的是 a 关于 m 的阶数的质因数分解。factor 是 Pollard-ρ 质因数分解算法。设阶数的质因数分解为 $p_1^{k_1}p_2^{k_2}\cdots p_n^{k_n}$，则时间复杂度为 $\mathrm{O}\left(\sum_{i=1}^n k_i(\log m+\sqrt{p_i})\right)$。

```cpp
#define KMAX 64
static i64 p0[KMAX], p1[KMAX];
static int k0[KMAX], k1[KMAX], c0, c1;
inline i64 _f(i64 x, i64 m) {
  i64 r = qmul(x, x, m) + 1;
  if (r >= m) r -= m;
  return r;
}
inline void factor(i64 n) {
  if (n == 2 || isprime(n)) p0[c0++] = n;
  else if (!(n & 1)) factor(2), factor(n >> 1);
  else {
    for (int i = 1; ; i++) {
      i64 x = i, y = _f(x, n), p = __gcd(y - x, n);
      while (p == 1) {
        x = _f(x, n);
        y = _f(_f(y, n), n);
        p = __gcd((y - x + n) % n, n) % n;
      }
      if (p != 0 && p != n) {
        factor(p), factor(n / p);
        return;
}}}}
inline void psort(i64 *p, int *k, int &c) {
  sort(p, p + c);
  int t = 0;
  for (int i = 0, j; i < c; i = j) {
    for (j = i; j < c && p[i] == p[j]; j++) ;
    p[t] = p[i];
    k[t++] = j - i;
  } c = t;
}
void ord(i64 a, i64 m, int p = 0, i64 cur = 1) {
  static i64 tmp[KMAX + 10], mi;
  static int t;
  if (p == 0) mi = LLONG_MAX;
  if (p == c0 && qpow(a, cur, m) == 1 && cur < mi) {
    mi = cur;
    memcpy(p1, tmp, sizeof(i64) * t);
    c1 = t;
  } else if (p != c0) {
    int t0 = t;
    for (int k = 0; k <= k0[p] && cur < mi; k++, cur *= p0[p]) {
      if (k) tmp[t++] = p0[p];
      ord(a, m, p + 1, cur);
    } t = t0;
}}
inline i64 log(i64 a, i64 b, i64 m, i64 p, int k) {
  typedef unordered_map<i64, i64> Map;
  static Map tb;
  i64 pw = 1, bc, bt = 1, s = 1;
  for (int i = 1; i < k; i++) pw *= p;
  i64 g = qpow(a, pw, m), ai = inv(a, m), x = 0;
  for (bc = g; s * s <= p; s++) bc = qmul(bc, g, m);
  tb.clear();
  for (i64 i = 1, t = bc; i <= s; i++, t = qmul(t, bc, m))
    tb.insert(make_pair(t, i));
  for (int i = 0; i < k; i++, pw /= p, bt *= p) {
    i64 b0 = qpow(qmul(b, qpow(ai, x, m), m), pw, m), d = -1;
    for (i64 j = 0, t = b0; j < s; j++, t = qmul(t, g, m)) {
      Map::iterator it = tb.find(t);
      if (it != tb.end()) {
        d = it->second * s - j;
        if (d >= p) d -= p;
        break;
    }}
    if (d == -1) return -1;
    x += bt * d;
  } return x;
}
inline i64 log(i64 a, i64 b, i64 m) {
  if (a == 1) return b == 1 ? 0 : -1;
  i64 m0 = 1, x = 0;
  for (int i = 0; i < c1; i++)
    for (int j = 0; j < k1[i]; j++) m0 *= p1[i];
  for (int i = 0; i < c1; i++) {
    i64 pw = p1[i];
    for (int j = 1; j < k1[i]; j++) pw *= p1[i];
    i64 mi = m0 / pw, r = log(qpow(a, mi, m), qpow(b, mi, m), m, p1[i], k1[i]);
    if (r == -1) return -1;
    x = (x + qmul(qmul(r, mi, m0), inv(mi, pw), m0)) % m0;
  } return x < 0 ? x + m0 : x;
}
//factor(m - 1);
//psort(p0, k0, c0);
//ord(a, m);
//psort(p1, k1, c1);
//i64 ans = log(a, b, m);
```

14. Pohlig_Hellman [lmj/Pohlig_Hellman.cpp]

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 $g^x=h$。

1. 令 $x_0=0$
2. 计算 $r=g^{p^{e-1}}$，这个元素阶数为 p
3. 对 $k=0...e-1$ 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 $d_k\in\{0,\ ...,\ p-1\}$ 满足 $r^{d_k}=h_k$
   3. 令 $x_{k+1}=x_k+p^kd_k$
4. 返回 x_e

即设 $x=c_0+c_1p+c_2p^2+\cdots+c_{e-1}p^{e-1}$，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 $g^{c_kp^{e-1}}$ 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 $\mathrm{O}\left(\sum e_i(\log n+\sqrt{p_i})\right)$。

```cpp
typedef long long LL;
LL pfactor[1200] , totf;
LL gene[1200];
void exgcd(LL a,LL b,LL &x,LL &y) {
  if(b==0){ x=1; y=0; return;}
  exgcd(b,a%b,x,y);
  LL tp=x;
  x=y; y=tp-a/b*y;
}
LL inv ( LL a , LL mod ) {
  LL x , y;
  exgcd ( a , mod , x , y );
  return (x%mod+mod)%mod;
}
LL qmul ( LL a , LL b , LL m ) {
  a %= m; b %= m;
  LL r = a*b,s=(long double)(a)*b/m;
  return ((r-m*s)%m+m)%m;
}
LL fast_mul ( LL a , LL b , LL c , LL mod ) {
  return qmul(qmul(a,b,mod),c,mod);
}
pair<LL,LL> crt ( pair<LL,LL> a , pair<LL,LL> b ) {
  if ( a.first == -1 ) return b;
  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);
  a.second *= b.second;
  a.first %= a.second;
```

```
  return a;
}
LL mpow ( LL f , LL x , LL mod ) {
  LL s = 1;
  while ( x ) {
    if ( x % 2 ) s = qmul(s,f,mod);
    f = qmul(f,f,mod); x >>= 1;
  } return s;
}
pair<LL,LL> solve ( LL g , LL h , LL mod , LL prime , LL e , LL p ) {//mod=prime^e
  LL j , k , r = mpow ( g , mod / prime , p ) , x = 0 , hh;
  LL ret = 0 , nowp = mod / prime , pp = 1;
  gene[0] = 1;
  for ( k = 1 ; k <= prime - 1 ; k++ ) {
    gene[k] = qmul (gene[k-1],r,p);
  }
  for ( k = 0 ; k <= e - 1 ; k++ ) {
    h = qmul(h,inv(mpow(g,x,p),p),p);
    hh = mpow ( h , nowp , p );
    for ( j = 0 ; j <= prime - 1 ; j++ ) {
      if ( gene[j] == hh ) break;
    }
    nowp = nowp / prime;
    x = j * pp;
    ret += x;
    pp = pp * prime;
  } return make_pair(ret,mod);
}
LL getlog ( LL a , LL root , LL p ) {
  LL i , j , tp , tmp;
  pair<LL,LL> ret , rem;
  tp = p - 1;
  rem.first = -1;
  for ( i = 2 ; tp != 1 ; i++ ) {
    if ( tp % i == 0 ) {
      tmp = 1; j = 0;
      while ( tp % i == 0 ) {
        tmp = tmp * i;
        j++; tp /= i;
      }
      ret = solve ( mpow ( root , p / tmp , p ) , mpow ( a , p / tmp , p ) , tmp , i , j , p );
      rem = crt ( rem , ret );
  }} return rem.first;
}
LL getroot ( LL p ) {
  LL i , j , tp = p - 1;
  totf = 0;
  for ( i = 2 ; tp != 1 ; i++ ) {
    if ( tp % i == 0 ) {
      pfactor[++totf] = i;
      while ( tp % i == 0 ) tp /= i;
  }}
  for ( i = 2 ; i < p ; i++ ) {
    for ( j = 1 ; j <= totf ; j++ ) {
      if ( mpow ( i , (p-1)/pfactor[j] , p ) == 1 ) break;
    }
    if ( j == totf + 1 ) return i;
  } return -1;
}
LL work ( LL p , LL a , LL b ) {  // return x, such that a^x = b (mod p)
  LL i , j , rt , la , lb , x , y , g;
  rt = getroot ( p );
  la = getlog ( a , rt , p );  // rt^la = a (mod p)
  lb = getlog ( b , rt , p );
  // x*la = lb (mod p-1)
  g = __gcd ( la , p - 1 );
  exgcd ( la , p - 1 , x , y );
  if ( lb % g != 0 ) return -1;
  x = (x%(p-1)+(p-1))%(p-1);
  return qmul ( x , (lb/g) , (p - 1)/__gcd(la,p-1) );
}
```

15. continued_fraction [lmj/continued_fraction.cpp]

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 $\mathrm{O}(\log^2 n)$，卡时间的话可以尝试二分一下，变成 $\mathrm{O}(\log n \log\log n)$。（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））

```
long long dec1[1200] , dec2[1200] , len1 , len2;
long long num , dom;
void getfrac ( long long *d , long long &l , long long a , long long b ) {
  l = 1;
  d[1] = a / b;
  a %= b;
  while ( a != 0 ) {
    swap ( a , b );
    d[++l] = a / b;
    a %= b;
}}
void work () {
  long long i;
  getfrac ( dec1 , len1 , a , b );
  getfrac ( dec2 , len2 , c , d );
  dec1[len1+1] = 2147483647777777777ll;
  dec2[len2+1] = 2147483647777777777ll;
  for ( i = 1 ; i <= len1 && i <= len2 ; i++ ) {
    if ( dec1[i] != dec2[i] ) break;
  }
  dec1[i] = min ( dec1[i] , dec2[i] ) + 1;
  num = dec1[i]; dom = 1;
  for ( i-- ; i >= 1 ; i-- ) {
    swap ( num , dom );
    num = num + dom * dec1[i];
  }
  printf ( "%lld %lld\n" , num , dom );
}
```

16. min_25_sieve [lmj/min_25_sieve.cpp]

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

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

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

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

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 $p_{j-1} \leqslant \sqrt{n}$ 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 $p_j^2 \leqslant n$ 时转移不为 0。从小到大按层转移。

$$g(n,\,i) = g(n,\,i+1) + \sum_{\substack{e \geqslant 1 \\ p^{e+1} \leqslant n}} \left[f(p_i^e)\left[g\left(\left\lfloor \frac{n}{p_i^e} \right\rfloor,\, i+1 \right) - h(p_i) \right] + f(p_i^{e+1}) \right]$$

同样的，只有 $p_i^2 \leqslant n$ 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 $g(n,\,0) + f(1)$。

```
typedef long long LL;
const LL NN = 420000;
const LL block = 100000;
const LL mod = 1000000007;
const LL inv2 = 500000004;
LL n, p[1200000] , prime[NN] , tot;
LL value[NN] , cnt , limit , pos[NN];
LL sumh[NN] , h0[NN] , h1[NN];
LL h[NN];  // sum of h[1..value[x]]
LL g[NN];
LL getpos ( LL x ) { return x<=limit?x:pos[n/x]; }
void predo () {
  LL i , j;
  for ( i = 2 ; i <= block ; i++ ) {
    if ( !p[i] )
      prime[++tot] = i;
    for ( j = 1 ; j <= tot && i * prime[j] <= block ; j++ ) {
      p[i*prime[j]] = 1;
      if ( i % prime[j] == 0 ) break;
  }}
  cnt = 0;
  for ( i = 1 ; i * i <= n ; i++ ) value[++cnt] = i;
  i--; limit = i;
  for ( ; i >= 1 ; i-- ) if ( n / i != value[cnt] ) {
    value[++cnt] = n / i;
    pos[i] = cnt;
  }
  for ( i = 1 ; i <= tot ; i++ )
    sumh[i] = (sumh[i-1] + prime[i]) % mod;
  for ( i = 1 ; i <= cnt ; i++ ) {//cal h from 2 to i
    h0[i] = ((value[i]-1)%mod*((value[i]+2)%mod)%mod*inv2) % mod;//modulo before multiply
    h1[i] = (value[i] - 1) % mod;
  }
  for ( i = 1 ; i <= tot ; i++ ) {
    for ( j = cnt ; prime[i] * prime[i] <= value[j] ; j-- ) {
      h0[j] = ( (h0[j] - prime[i] * (h0[getpos(value[j]/prime[i])]-sumh[i-1]) ) % mod );
      if ( h0[j] < 0 ) h0[j] += mod;
      h1[j] = ( (h1[j] - 1 * (h1[getpos(value[j]/prime[i])]-(i-1)) ) % mod );
      if ( h1[j] < 0 ) h1[j] += mod;
  }}
  for ( i = 1 ; i <= cnt ; i++ )//f(p)=p-1
    h[i] = ( h0[i] - h1[i] + mod ) % mod;
}
LL getf ( LL p , LL e ) { return p ^ e; }
void min25 () {
  LL i , j , e , now , tmp;
  for ( j = cnt ; j >= 1 ; j-- ) g[j] = h[j];
  for ( i = tot ; i >= 1 ; i-- )
    for ( j = cnt ; prime[i] * prime[i] <= value[j] ; j-- )
      for ( e = 1 , now = prime[i] ; now * prime[i] <= value[j] ; e++ , now = now * prime[i] )
        g[j] = ( g[j] + getf(prime[i],e) * (g[getpos(value[j]/now)]-h[prime[i]]+mod) + getf(prime[i],e+1) ) % mod;
  printf ( "%lld\n" , (g[cnt] + 1) % mod );
}
void work () {
  scanf ( "%lld" , &n );
  predo ();
  min25 ();
}
```

17. 幂级数前缀和 [xzl/power-series.cpp]

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); }
};
```

18. 类 Euclid 算法 [xzl/sim-euclid.cpp]

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 $O(K^3 \log \max\{a,\,c\})$。注意参数的范围防止整数溢出。**如果只是计算直线下整点数量，则主算法部分只用被注释掉的四句话。**

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。

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	$\frac{5}{66}$	$-\frac{691}{2730}$	$\frac{7}{6}$	$-\frac{3617}{510}$	$\frac{43867}{798}$	$-\frac{174611}{330}$

```
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;
    //return b / c * (n + 1) % 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);
    //return (F(n, a % c, b, c) + a / c * n % MOD * (n + 1) % MOD * INV2) % 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);
    //return (F(n, a, b % c, c) + b / c * (n + 1)) % 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 (m * n - F(m - 1, c, c - b - 1, a)) % MOD;
  } return ret;
}
```

19. 线性筛 & 杜教筛 [xzl/dyh.cpp]

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]);
```

20. dinic [lmj/dinic.cpp]

```
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" );
}
```

21. 费用流 [lmj/min_cost_max_flow.cpp]

```
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" );
}
```

22. 三分_上凸函数 [sll/三分_上凸函数.cpp] ■

```
    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;
}
```

23. 单纯型 [xzl/simplex.cpp] ■

```
#define EPS 1e-10
#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];
};
```

24. 线性空间求交 [xzl/vector-space-intersect.cpp] ■

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

25. AC 自动机 [xzl/ac-automaton.cpp] ■

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);
}}
```

26. KMP [sll/KMP.cpp]

```
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;
}
```

27. PAM_sll [sll/PAM_sll，字符串.cpp]

```
#define N 500020
int val[N], 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;
int main() {
  tree.init();
  scanf("%s",tree.ch+1);
  int len=strlen(tree.ch+1);
  for(int i=1;i<=len;i++)
    tree.insert(i,tree.ch[i]);
  tree.dfs(1);
  printf("%lld\n",tree.cal());
}
```

28. SA_sll [sll/SA_sll，字符串.cpp]

```
#define N 200020
int wa[N],wb[N],ws[N],wv[N],sa[N],rank[N];
void cal_sa(int *r,int n,int m) {
  int *x=wa,*y=wb,*t;
  for(int i=0;i<m;i++)ws[i]=0;
  for(int i=0;i<n;i++)ws[x[i]=r[i]]++;
  for(int i=1;i<m;i++)ws[i]+=ws[i-1];
  for(int i=n-1;i>=0;i--)sa[--ws[x[i]]]=i;
  for(int j=1,p=1;p<n;j<<=1,m=p) {
    p=0;
    for(int i=n-j;i<n;i++)y[p++]=i;
    for(int i=0;i<n;i++)if(sa[i]>=j)y[p++]=sa[i]-j;
    for(int i=0;i<n;i++)wv[i]=x[y[i]];
    for(int i=0;i<m;i++)ws[i]=0;
    for(int i=0;i<n;i++)ws[wv[i]]++;
    for(int i=1;i<m;i++)ws[i]+=ws[i-1];
    for(int i=n-1;i>=0;i--)sa[--ws[wv[i]]]=y[i];
    t=x,x=y,y=t,p=1;x[sa[0]]=0;
    for(int i=1;i<n;i++)
      x[sa[i]]=(y[sa[i-1]]==y[sa[i]]&&y[sa[i-1]+j]==y[sa[i]+j])?p-1:p++;
}}
int height[N];
void cal_h(int *r,int *sa,int n) {
  int k=0;
  for(int i=1;i<=n;i++)rank[sa[i]]=i;
  for(int i=0;i<n;i++) {
    int j=sa[rank[i]-1];if(k)k--;
    while(r[j+k]==r[i+k])k++;
    height[rank[i]]=k;
}}
char ch[N]; int r[N];
int main() {
  std::cin>>ch;
  int n=strlen(ch);
  for(int i=0;i<n;i++)r[i]=ch[i];r[n]=0;
  cal_sa(r,n+1,128);
  cal_h(r,sa,n);
  for(int i=1;i<=n;i++)printf("%d ",sa[i]+1);puts("");
  for(int i=2;i<=n;i++)printf("%d ",height[i]);
}
```

29. manacher [sll/manacher.cpp]

```
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;
}}
```

30. pam [lmj/pam.cpp]

```
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 );
}
```

31. 回文自动机 [sll/回文自动机.cpp]

```
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;
```

32. 后缀排序：倍增算法 [xzl/sa-nlogn.cpp]

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 $\Sigma=\{0,1,2,...,m\}$，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 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;
  //memset(cnt, 0, sizeof(int) * (m + 1));
  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;
}}
```

33. 后缀排序：DC3 [xzl/dc3.cpp]

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;
}
```

34. 后缀排序：SA-IS [xzl/sais.cpp]

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];
```

35. 后缀树 [xzl/后缀树，字符串.cpp]

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。

```
#define SIGMA 27
#define EOFC (SIGMA - 1)
struct State {
  struct Trans {
```

```
    Trans(int _l, int _r, State *_nxt)
      : l(_l), r(_r), nxt(_nxt) {}
    int l, r; State *nxt;
    int len() const { return r - l + 1; }
  };
  State() : fail(NULL) { memset(ch, 0, sizeof(ch)); }
  State *fail; Trans *ch[SIGMA];
};
typedef State::Trans Trans;
static State *rt;
static char str[NMAX + 10];
static int n;
namespace _append {
static char dir;
static int len, cnt, cur;
static State *ap;
void reset() {
  dir = -1; ap = rt;
  len = cnt = cur = 0;
}}
inline void append(char c) {
  using namespace _append;
  cnt++; cur++;
  State *x, *y = NULL;
  while (cnt) {
    if (cnt <= len + 1) {
      len = cnt - 1;
      dir = len ? str[cur - len] : -1;
    }
    while (dir >= 0 && len >= ap->ch[dir]->len()) {
      len -= ap->ch[dir]->len();
      ap = ap->ch[dir]->nxt;
      dir = len ? str[cur - len] : -1;
    }
    if ((dir >= 0 && str[ap->ch[dir]->l + len] == c) ||
      (dir < 0 && ap->ch[c])) {
      if (dir < 0) dir = c;
      if (y) y->fail = ap;
      len++; return;
    }
    if (dir < 0) {
      ap->ch[c] = new Trans(cur, n, new State);
      x = ap;
    } else {
      Trans *t = ap->ch[dir];
      x = new State;
      x->ch[c] = new Trans(cur, n, new State);
      x->ch[str[t->l + len]] = new Trans(t->l + len, t->r, t->nxt);
      t->r = t->l + len - 1;
      t->nxt = x;
    }
    if (y) y->fail = x;
    if (ap != rt) ap = ap->fail;
    y = x; cnt--;
}}
inline void initialize() {
  rt = new State;
  _append::reset();
  n = strlen(str + 1) + 1;
  for (int i = 1; i < n; i++) {
    str[i] -= 'a';
    append(str[i]);
  }
  str[n] = EOFC;
  append(EOFC);
}
```

36. fft [lmj/fft.cpp]

```
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':' ' );
}
```

37. lct [lmj/lct.cpp]

```
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;
}
```

38. ntt [lmj/ntt.cpp]

```
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':' ' );
}
```

39. 左偏树 [lmj/leftist_tree.cpp]

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;
}
```

40. 序列splay [sll/区间splay.cpp]

```
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) {
```

```cpp
    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];
}
```

41. 权值splay [sll/权值splay.cpp]

```cpp
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);
}}
```

◦ 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](c[u]>0)-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/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/spfa-opt.md ■

SPFA 优化。均为玄学，该卡掉的都可以卡掉。费用流时可以考虑一下。

- SLF: 如果入队元素 dist 小于队首元素 dist，则加入队首。使用 deque。
- SLF-swap: 如果入队后发现队尾元素 dist 小于队首元素 dist，则交换队首和队尾。避免使用双端队列。
- LLL: 入队时与队内 dist 平均值做比较来决定是进队首或者队尾。使用 deque。（效果甚微）

∘ 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 $\mathrm{O}(n\log^2 n)$ 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 ■

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

∘ 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/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