Algorithm template by cxjyxx

0. Interface

```
==========================================================
Suffix_Array

  const int SAmaxn 为字符串最大长度
  const int SAmaxm 为字符串最多字符种数

  void create(string) [s] 传入字符串s 建立s的SA
  void clear() 清空SA
==========================================================
Link_Table

  const int LTmaxn 为最大的最大点编号
  const int LTmaxm 为最多边数

  void makeline(int, int) [u, v] 建立u->v的单向边
  void makeline_double(int, int) [u, v] 建立u、v的双向边
  int be(int) [now] 返回now连出的第一条边
  int next(int) [i] 返回i这条边的下一条边
  int to(int) [i]  返回这条边指向的点
  void clear() 清空链表
==========================================================
Link_Table_V

  const int LTmaxn 为最大的最大点编号
  const int LTmaxm 为最多边数

  void makeline(int, int, int) [u, v, va] 建立u->v的权值为va单向边
  void makeline_double(int, int, int) [u, v, va] 建立u->v的权值为va双向边
  int be(int) [now] 返回now连出的第一条边
  int next(int) [i] 返回i这条边的下一条边
  int v(int) [i] 返回i这条边的权值
  int to(int) [i]  返回这条边指向的点
  void clear() 清空链表
==========================================================
Suffix_Auto_Maton

  const int SAMmaxn 为字符从最大长度*2

  void create() 初始化SAM
  void add(char, int) [c, len] 往当前SAM加入字符c len是这个字符在串中的位置 从1开始
  void clear() 清空SAM
==========================================================
KMP

  const int KMPmaxn 为字符串最大长度

  void create(string) [s] 建立s的KMP
  void clear() 清空KMP
==========================================================
AC_Auto_Maton

  const int ACAMmaxn 最多节点数
  const int ACAMmaxm 最多字符种数

  void create(char) [c] 传入最小字符c 如'a' 或 'A'
  void add(string) [s] 把该字符串s加入ACAM
```

```
  void getfail() 取得ACAM的fail值
  void clear() 清空ACAM
============================================================
Splay

  const int SPLAYmaxn 最多节点数
  const int SPLAYinf 最大权值abs

  void create() 初始化splay
  void clear(int) [u] 清空节点u
  int lower_bound(int) [k] 返回第一个key不小于k的节点编号
  void insert(int) [k] 插入key为k的节点
  int get_lower(int) [k] 返回k的前驱
  int get_upper(int) [k] 返回k的后继
  void del(int, int) [l, r] 删除key为l~r的所有节点
  int get_kth(int) [k] 返回key值第k小的节点
============================================================
Link_Cut_Tree
  //这个类内的节点编号和外部的节点边号不一样 一般传入外部编号即可 get_v函数需要传入内部节点编号
to_num函数可以把外部节点转为内部节点编号

  const int LCTmaxn 最大节点数

  void create(int, Link_Table, int[]) [n, L, a[]] 表示有n个节点
                                  连接信息存储在L中 权值信息存储在a[]中
                                  下标为1~n
  void make_root(int) [u] 把节点u转到根
  void link(int, int) [u, v] 连接两个节点u, v
  void cut(int, int) [u, v] 切断两个节点u, v间的连边
  int query_sum(int, int) [u, v] 返回u, v路径上的节点权值和
  int query_max(int, int) [u, v] 返回u, v路径上有最大权值的节点的内部编号
  int query_max(int, int) [u, v] 返回u, v路径上有最小权值的节点的内部编号
  int root(int) [u] 返回节点u所在树的根
  void modify_sum(int, int, int) [u, v, value] 把u到v路径上所有点的权值+value
  void refresh(int) [u] 刷新节点u的信息
  int get_v(int) [u] /*u为内部节点编号*/ 返回内部节点u对应权值
  int to_num(int) [u] 返回外部节点u对应内部节点编号
  void clear() 清空LCT
============================================================
Segment_Tree

  const int STmaxn 最大节点数 一般为4*n
  const int STinf 最大权值abs

  void create(int, int, int[]) [l, r, a[]] 传入ST的左右范围 和初始权值数组
  void modify_is(int, int, int) [l, r, v] 把l~r的权值都改为v
  void modify_sum(int, int, int) [l, r, v] 把l~r的权值都加v
  int query_max(int, int) [l, r] 返回l~r的最大值
  int query_min(int, int) [l, r] 返回l~r的最小值
  int query_sum(int, int) [l, r] 返回l~r的权值和
  void clear() 清空ST
============================================================
Tree_Chain_Division
  //这里的树链剖分可以采用两种create方法 一种是边权 一种是点权 后面会介绍 调用这个类的时侯 同时也要调
用 Segment_Tree、Link_Table、Link_Table_V

  const int TCDmaxn 最大节点数
```

  void create(int, Link_Table_V) [n, L] 表示这棵树有n个节点 权值为边权 保证节点1为有效节点 连边关系及权值关系为L
  void create(int, Link_Table, int[]) [n, L, v[]] 表示这棵树有n个节点 权值为点权 保证节点1为有效节点 连边关系为L 点权为v[]
  int lca(int, int) [u, v] 返回节点u, v的lca
  void modify_sum(int, int, int) [u, v, value] 把u到v路径上所有点的权值+value
  void modify_is(int, int, int) [u, v, value] 把u到v路径上所有点的权值改为value
  int query_sum(int, int) [u, v] 返回u, v路径上的节点权值和
  int query_max(int, int) [u, v] 返回u, v路径上有最大权值的节点的内部编号
  int query_max(int, int) [u, v] 返回u, v路径上有最小权值的节点的内部编号
  void clear() 清空TCD
==
KD_Tree

  KDT_Point
    xy[KDTmaxp] 点坐标
    num 点编号
    dist 内部使用 无需理解

  const int KDTmaxp 最大维度
  const int KDTmaxn 最大节点数
  const int KDTinf 最大dist值

  void create(int, KDT_Point[], int) [n, k[], p] 表示一共有n个节点要预先建树 节点信息存在k[]里 下标0~n-1 维度为p
  void insert(KDT_Point) [k] 往KD_Tree里插入节点k
  void query_max(KDT_Point, *KDT_Point[], int) [a, &ans[], k] 返回离点a的k远点 写在ans[]里
  void query_min(KDT_Point, *KDT_Point[], int) [a, &ans[], k] 返回离点a的k近点 写在ans[]里
  int dist(KDT_Point, KDT_Point) [a, b] 返回点a、b的距离
  void clear() 清空KDT
==
Network_Flow

  const int NFmaxn 最大节点数
  const int NFmaxm 最大边数
  const int NFinf 最大权值

  void create(int) [n] 一共有n个点
  void makeline(int, int, int) [u, v, va] 建立u->v权值为va的边 会建立反向边
  int query(int, int) [s, t] 以s为源点 以t为汇点 跑最大流 返回流量
  void clear() 清空NF
==
Network_Cost_Flow_Spfa
//需调用par类

  const int NCFSmaxn 最大节点数
  const int NCFSmaxm 最大边数
  const int NCFSinf_ 最大权值的两位 如(0x7f)
  const int NCFSinf 最大权值必为NCFSinf_的满状态 如（0x7f7f7f7f)

  void create() 无意义
  void makeline(int, int, int, int) [u, v, va, c] 建立u->v权值为va 费用为c的边 会建立反向边
  par query(int, int) [s, t] 以s为源点 以t为汇点 跑最小费用最大流 返回(最大流量, 最小费用)
  void clear() 清空NCFS
==
Network_Cost_Flow_Zkw

//需调用par类

  const int NCFZmaxn 最大节点数
  const int NCFZmaxm 最大边数
  const int NCFSinf_ 最大权值的两位 如(0x7f)
  const int NCFSinf 最大权值必为NCFSinf_的满状态 如 (0x7f7f7f7f)

  void create() 无意义
  void makeline(int, int, int, int) [u, v, va, c] 建立u->v权值为va 费用为c的边 会建立反向边
  par query(int, int) [s, t] 以s为源点 以t为汇点 跑最小费用最大流 返回(最大流量, 最小费用)
  void clear() 清空NCFZ
==
Network_Flow_Up_Down
//需调用Network_Flow类

  NFUDinf 最大权值

  void create(int, int, int, int, int) [n, s, t, ss, tt] n个点 源为s 汇为t 超级源为ss 超级汇为tt 只要保证ss和tt不会被用到即可
  void makeline(int, int, int, int, int) [u, v, down, up] 建立从u到v的 流量上界为up 下界为down的边 会建立反向边
  int query() 返回最大流量 如果无解 返回-1 注意 这个函数建议只调用一次 和不带上下界的不一样
  void clear() 清空NFUD
==
Network_Cost_Flow_Up_Down
//需调用Network_Cost_Flow_Zkw、par类

  NCFUDinf 最大权值

  void create(int, int, int, int) [s, t, ss, tt] 源为s 汇为t 超级源为ss 超级汇为tt 只要保证ss和tt不会被用到即可
  void makeline(int, int, int, int, int, int) [u, v, down, up, c] 建立从u到v的 流量上界为up 下界为down 费用为c的边 会建立反向边
  par query() 返回(最大流量, 最小费用) 如果无解 返回(-1, -1) 注意 这个函数建议只调用一次 和不带上下界的不一样
  void clear() 清空NCFUD
==
Mergeable_Tree
//可并堆 大根堆

  MHmaxn 最大节点数

  void create() 无意义
  int merge(int, int) [u, v] 合并以u,v为根的两个堆 返回新的根
  int make_node(int) [v] 建立一个只有一个元素 元素权值为v的堆 返回根的编号
  int pop(int) [u] 删除以u为根的堆的根 返回新根
  int top(int) [u] 返回以u为根的堆的根的权值
  void clear() 清空MT
==
Hash_Map

  HMmaxn 最多元素数

  HM_Member_V 存放需保存的值的类
  HM_Member 存放须hash的值和对应权值类

  MOD 模数
  MOD2 hash用的乘数

```
  void create() 无意义
  int hash(HM_Member&) [a] 返回a的hash值 可能需要重载
  bool check(HM_Member&) [a] 返回a是否出现在hash表中
  HM_Member_V get(HM_Member&) [a] 返回a对应的权值类 默认a出现在hash表中
  void insert(HM_Member&) [a] 把a插入hash表中 如果a已经出现在hash表中 会把原来a的权值类更新为新
的a的权值类 否则直接插入
  void clear() 清空HM
============================================================
Geometry_Base

  eps 误差保留
  pi π

  point 存放一个点
  {
       double x, y坐标

       point (double, double) 构造函数
       point +(point, point) 向量加
       point -(point, point) 向量减
       point *(point, point) 向量乘
       point /(point, point) 向量除
       double &(point, point) 叉乘
       double |(point, point) 点乘
       point +=(point) 向量加
       point -=(point) 向量减
       point *=(double) [a] x、y都乘a
       point /=(double) [a] x、y都除a
       point ==(point, point) [a, b] 返回（a、b是同一个点）
       point !=(point, point) [a, b] 返回（a、b不是同一个点）
       point <(point, point) [a, b] 如果(a.x < b.x) 或 (a.x == b.x 且 a.y < b.y) 就返回
true 否则返回false
  }

  segment 存放一个线段
  {
       point a, b 线段两个端点

       segment(point, point) 构造函数
       point ==(point, point) [a, b] 返回（a、b是同一线段）
       point !=(point, point) [a, b] 返回（a、b是不同一线段）
  }

  void create() 无意义
  double sqr(double) [a] 返回a * a
  point rotate(point, double) [a, b] 把a作为向量 顺时针旋转b(弧度制)
  segment get_vertical(segment, point) [a, b] 返回点b关于直线a的垂线
  point get_foot(segment, point) [a, b] 求出点b对于直线a的垂足
  point get_mid(segment) [a] 求线段a的中点
  double dist(point, point) [a, b] 返回a、b两点的距离
  double dist(segment, point) [a, b] 返回点b与直线a的距离(注意这里是直线)
  double dist2(segment, point) [a, b] 返回点b与线段a的距离(注意这里是线段)
  int cmp(double, double) [a, b] 返回在eps下 a、b的大小关系 (0 =) (-1 <) (1 >)
  int pos(point, point) [p, q] 把p、q作为向量 返回q关于p的位置关系 (0 平行) (-1 q在p右侧) (1
q在p左侧)
  int pos(segment, segment) [p, q] 返回两条线段作为向量的q关于p的位置关系(由线段的a点指向b点)
返回值同上
```

```
  int pos(segment, point) [p, q] 返回点q关于作为向量的线段p的位置关系 返回值同上
  bool init(segment, point) [a, b] 判断点b是否在线段a内
  int check(segment, segment) [a, b] 判断线段a、b是否相交 （0 无交点） (-1 平行相交<可能无数交
点>) (1 一个交点)
  point cross(segment, segment) [a, b] 返回a、b作为直线的交点 如果平行返回(0, 0)
  double get_k(segment) [s] 返回s的斜率 如果k垂直x轴 返回0
  double get_b(segment) [s] 返回s的截距 如果k垂直x轴 返回0
  void clear() 无意义

  typedef Geometry_Base::point gbp; 声明Geometry_Base::point类型可以用GBP 构造函数同理
  typedef Geometry_Base::segment gbs; 声明Geometry_Base::segment类型可以用GBS 构造函数同理
============================================================
Geometry_Polygon
//需调用Geometry_Base类

  GPmaxn 表示最大顶点数

  int n 记录顶点数
  int a[] 记录顶点

  void create(int, vector<gbp>) [n, a] 建立有n个顶点 分别是a的多边形
  double get_s() 返回多边形面积
  int check(gbp) [a] 返回a是否在多边形内 (-1 边界) (0 外部) (1 内部)
  double get_in_len(gbs) 返回某条线段被多边形覆盖的长度
  void clear() 清除多边形
============================================================
Geometry_Round
//需调用Geometry_Base类

  void create(gbp, double) [o, r] 建立圆心为o 半径为r的圆
  int check(gbp) [a] 返回a是否在圆内 (-1 边界) (0 外部) (1 内部)
  int check(gbs) [s] 返回直线s与圆的交点个数
  int check2(gbs) [s] 返回线段s与圆的交点个数
  vector<gbp> cross(gbs) [s] 返回直线s与圆的交点集合
============================================================
Shortest_Path
//需调用Link_Table_V类
//堆优化的dij实现

  SPmaxn 最大点数

  int query(Link_Table_V, int, int) [l, s, t] 返回在l图上 s到t的最短路 如果不联通 返回-1
============================================================
High_Num

  cut 压缩的位数
  HNmaxn 最大位数/cut

  create(int a) [a] 新建值为a的高精数
  create(string a) [a] 新建值为a的高精数 第一个字符如果为'-'则为负数 否则为正 其他字符必须为
'0'~'9'
  {
    +
    -
    *
    /
       %
       +=
```

```
      -=
      *=
      /=
      %=
      =
      ==
      !=
      <
      >
      <=
      >=
      对于两个High_Num间的 或 一个High_Num和一个int间的 以上操作符 均支持
  }
  int to_int() 返回这个高精数的int形 爆了自重
  print() 输出高精数
  clear() 清空高精数
============================================================
Discretization
//要引用functional库

  Dmaxn 最大需离散成员数

  Discretization<class T> a; 声明一个为T类型进行离散的离散化类的实例a

  int sau(T*, int, class, class) [a, n, _less = less<class>(), _equal =
equal_to<class>()] 对T类型的a数组的前n位(下标从0开始)进行排序并去重 返回去重后n的大小 注意这里是直
接在传入的数组上进行改变 后两个参数是比较函数 跟sort传入的比较函数一样 这两个函数为bool类型 都需有两
个T类型参数 第一个函数返回第一个参数是否小于第二个参数 第二个函数返回第一个参数是否等于第二个参数

  void query(T*, int, int*, class, class) [a, n, ans, _less = less<class>(), _equal =
equal_to<class>()] 对a数组的前n位进行离散化 结果存入ans数组中 后两个参数同上

  实例:
  //------------0--------------
  #include <iostream>
  #include <cstdio>
  #include <functional>
  /*
       Discretization类
  */
  Discretization<int> D;

  int n, ans[100], a[100];

  int main()
  {
       READ(a, n); //对a进行读入 这里不实现 下同
       D.query(a, n, ans); //缺省
       n = D.sau(a, n); //缺省  注意这里调用的两个函数是不相关的 只是为了演示如何使用 下同
  }
  //------------0--------------
  //------------1--------------
  #include <iostream>
  #include <cstdio>
  #include <functional>
  /*
       Discretization类
  */
```

```
  struct point
  {
       int a, b;
  };

  bool _less(point a, point b)
  {
       return a.a < b.a;
  }

  bool _same(point a, point b)
  {
       return a.b < b.b;
  }

  Discretization<point> D;
  point a[100];
  int n, ans[100];

  int main()
  {
       READ(a, n);
       D.query(a, n, ans, _less, _same);
       n = D.sau(a, n, _less, _same);
  }
  //------------1--------------
  //------------2--------------
  #include <iostream>
  #include <cstdio>
  #include <functional>
  /*
       Discretization类
  */
  Discretization<double> D;

  bool same(double a, double b)
  {
    return abs(a - b) <= 1e-8;
  }

  double a[100];
  int n, ans[100];

  int main()
  {
       READ(a, n);
       D.query(a, n, ans, less<double>(), same); //这里的less是functional自带的函数 可以直
接这样使用
       n = D.sau(a, n, less<double>(), _same); //同上
  }
  //------------2--------------
============================================================
Tarjan
//需调用Link_Table类

  Tmaxn 最大点数
  Tmaxm 最大边数 注意 如果是双向边要乘2
```

  void get_cut_node(Link_Table, int, int*) [L, n, b] 在L这张图上 共有n个点(编号为0~n-1 如果有些编号没点没关系) 在b数组中 b[i]==true 表示点i为割点 否则不是 注意b数组下标大于n - 1 的数值不确定
  void get_cut_edge(Link_Table, int, int*) [L, n, b] 在L这张图上 共有n个点(同上) 在b数组中 b[i]==true 表示L中的下标为i的边为割边(由于Link_Table中是有向边 一条割边将有两条对应边被记为割边) 注意b数组下标大于n - 1 的数值不确定
==========

1. Suffix_Array // **后缀数组**

```
//============================SA=======================

const int SAmaxn = 100005;
const int SAmaxm = 100005;
struct Suffix_Array
{
  int sa[SAmaxn], sa_t[SAmaxn], t[SAmaxn], v[SAmaxn], v_t[SAmaxn], s[SAmaxn],
rank[SAmaxn], h[SAmaxn], cnt[SAmaxm], m, n;

  bool same(int a, int b, int c) { return t[a] == t[b] && t[a + c] == t[b + c]; }

  void geth()
  {
    for (int i = 0; i < n; ++i) rank[sa[i]] = i;
    for (int i = 0, k = 0; i < n; h[rank[i++]] = k)
      if (rank[i])
        for (k = k ? k - 1 : k; s[i + k] == s[sa[rank[i] - 1] + k]; ++k);
      else k = 0;
  }

  void getsa()
  {
    for (int i = 0; i < n; ++i) ++cnt[v[i] = s[i]];
    for (int i = 1; i < m; ++i) cnt[i] += cnt[i - 1];
    for (int i = n - 1; i >= 0; --i) sa[--cnt[v[i]]] = i;
    for (int j = 1, p = 0, i; p < n; m = p, j <<= 1)
    {
      for (p = 0, i = n - j; i < n; ++i) sa_t[p++] = i;
      for (i = 0; i < n; ++i) if (sa[i] >= j) sa_t[p++] = sa[i] - j;
      for (i = 0; i < m; ++i) cnt[i] = 0;
      for (i = 0; i < n; ++i) ++cnt[v_t[i] = v[sa_t[i]]];
      for (i = 1; i < m; ++i) cnt[i] += cnt[i - 1];
      for (i = n - 1; i >= 0; --i) sa[--cnt[v_t[i]]] = sa_t[i], t[i] = v[i];
      for (i = 1, p = 1, v[sa[0]] = 0; i < n; ++i)
        v[sa[i]] = same(sa[i], sa[i - 1], j) ? p - 1 : p++;
    }
  }

  void create(int s_t[], int n_t, int m_t)
  {
    n = n_t;
    m = m_t;
    for (int i = 0; i < n; ++i) s[i] = s_t[i];
    getsa();
    geth();
  }

  void create(string s_t)
  {
    n = s_t.length();
```

```
    m = 300;
    for (int i = 0; i < n; ++i) s[i] = s_t[i];
    getsa();
    geth();
  }

  void clear()
  {
    memset(cnt, 0, sizeof(cnt));
  }
};

//============================SA=======================
```

2. Link_Table // 链表

```
//============================LinkTable=================

const int LTmaxm = 1000005;
const int LTmaxn = 1000005;

struct line
{
  int to, next;
};

struct Link_Table
{
  line li[LTmaxm];
  int be_[LTmaxn], l;

  int to(int i) { return li[i].to; }
  int next(int i) { return li[i].next; }
  int be(int i) { return be_[i]; }

  void makeline(int fr, int to)
  {
    ++l;
    li[l].next = be_[fr];
    be_[fr] = l;
    li[l].to = to;
  }

  void makeline_double(int fr, int to)
  {
    makeline(fr, to);
    makeline(to, fr);
  }

  void clear()
  {
    l = 0;
    memset(be_, 0, sizeof(be_));
  }
};

//============================LinkTable=================
```

3. Link_Table_V // **带权链表**

```
//============================LinkTable_V===============

const int LTVmaxm = 200005;
const int LTVmaxn = 200005;

struct LTVline
{
  int to, next, v;
};

struct Link_Table_V
{
  LTVline li[LTVmaxm];
  int be_[LTVmaxn], l;

  int next(int i) { return li[i].next; }
  int be(int i) { return be_[i]; }
  int to(int i) { return li[i].to; }
  int v(int i) { return li[i].v; }

  void makeline(int fr, int to, int v)
  {
    ++l;
    li[l].next = be_[fr];
    be_[fr] = l;
    li[l].to = to;
    li[l].v = v;
  }

  void makeline_double(int fr, int to, int v)
  {
    makeline(fr, to, v);
    makeline(to, fr, v);
  }

  void clear()
  {
    l = 0;
    memset(be_, 0, sizeof(be_));
  }
};

//============================LinkTable_V===============
```

4. Suffix_Auto_Maton // **后缀自动机**

```
//============================SAM=======================

const int SAMmaxn = 500005;

struct SAM_Node
{
  int ch[30], fail, l;
};

struct Suffix_Auto_Maton
{
```

```
  SAM_Node t[SAMmaxn];
  int n, last, tot;

  void add(int c, int l)
  {
    int now = ++tot, p = last;
    last = tot;
    t[now].l = l;
    for (; p && !t[p].ch[c]; p = t[p].fail) t[p].ch[c] = now;
    if (!p) t[now].fail = 1;
    else if (t[p].l + 1 == t[t[p].ch[c]].l) t[now].fail = t[p].ch[c];
    else
    {
      int r = ++tot, k = t[p].ch[c];
      t[r] = t[k];
      t[r].l = t[p].l + 1;
      t[k].fail = t[now].fail = r;
      for (; p && t[p].ch[c] == k; p = t[p].fail) t[p].ch[c] = r;
    }
  }

  void create(string s)
  {
    int n = s.length();
    tot = last = 1;
    for (int i = 0; i < n; ++i) add(s[i] - 'a', i + 1);
  }

  void clear()
  {
    memset(t, 0, sizeof(t));
  }
};

//============================SAM=======================
```

5. KMP // 看**

```
//============================KMP=======================

const int KMPmaxn = 100005;

struct KMP
{
  int fail[KMPmaxn];

  void create(string s)
  {
    fail[0] = -1;
    int n = s.length();
    for (int i = 1, j = -1; i < n; ++i)
    {
      while (j != -1 && s[j + 1] != s[i]) j = fail[j];
      if (s[j + 1] == s[i]) ++j;
      fail[i] = j;
    }
  }

  void clear()
```

```
  {
    memset(fail, 0, sizeof(fail));
  }
};

//============================KMP=======================
```

6. AC_Auto_Maton // AC自动机

```
//============================AC_AutoMaton==============

const int ACAMmaxn = 500005;
const int ACAMmaxm = 30;

struct ACAM_Node
{
  int ch[ACAMmaxm], fail;
  int num;
};

struct AC_Auto_Maton
{
  ACAM_Node t[ACAMmaxn];
  int tot, root;
  queue<int> q;
  char base;

  void create(char ba)
  {
    root = tot = 2;
    base = ba;
  }

  void add(string s)
  {
    int n = s.length();
    for (int now = root, i = 0; i < n; ++i)
    {
      if (t[now].ch[s[i] - base]) now = t[now].ch[s[i] - base];
      else t[now].ch[s[i] - base] = ++tot, now = tot;
      if (i == n - 1) ++t[now].num;
    }
  }

  void get_fail()
  {
    q.push(root);
    while (!q.empty())
    {
      int now = q.front();
      q.pop();
      for (int i = 0; i < 26; ++i)
      {
        int to = t[now].ch[i];
        if (!to) continue;
        if (now == root) t[to].fail = root;
        else
        {
          int k;
```

```
          for (k = t[now].fail; k != root && !t[k].ch[i]; k = t[k].fail);
          if (t[k].ch[i]) k = t[k].ch[i];
          t[to].fail = k;
        }
        q.push(to);
      }
    }
  }

  void clear()
  {
    memset(t, 0, sizeof(t));
    tot = 0;
  }
};

//============================AC_AutoMaton==============
```

7. Splay // Splay 平衡树

```
//============================Splay=====================

const int SPLAYmaxn = 200005;
const int SPLAYinf = 100000000;

struct Splay_Node
{
  int l, r, fa, v, sum;
};

struct Splay
{
  Splay_Node t[SPLAYmaxn];
  int root, tot;

  void clear(int a)
  {
    t[a].l = t[a].r = t[a].fa = t[a].v = t[a].sum = 0;
  }

  void create()
  {
    clear(1), clear(2);
    root = 1, tot = 2;
    t[1].v = -SPLAYinf;
    t[2].v = SPLAYinf;
    t[1].r = t[1].sum = 2;
    t[2].fa = t[2].sum = 1;
  }

  void up(int now)
  {
    t[now].sum = t[t[now].l].sum + t[t[now].r].sum + 1;
  }

  void left(int now)
  {
    int fa = t[now].fa;
    t[now].fa = t[fa].fa;
```

```
  if (t[t[fa].fa].l == fa) t[t[fa].fa].l = now;
  if (t[t[fa].fa].r == fa) t[t[fa].fa].r = now;
  t[fa].fa = now;
  t[fa].r = t[now].l;
  t[t[now].l].fa = fa;
  t[now].l = fa;
  up(fa);
}

void right(int now)
{
  int fa = t[now].fa;
  t[now].fa = t[fa].fa;
  if (t[t[fa].fa].l == fa) t[t[fa].fa].l = now;
  if (t[t[fa].fa].r == fa) t[t[fa].fa].r = now;
  t[fa].fa = now;
  t[fa].l = t[now].r;
  t[t[now].r].fa = fa;
  t[now].r = fa;
  up(fa);
}

void splay(int now, int FA = 0)
{
  while (t[now].fa != FA)
  {
    int fa = t[now].fa;
    if (t[fa].fa == FA)
      if (t[fa].l == now) right(now);
      else left(now);
    else
      if (t[t[fa].fa].l == fa)
        if (t[fa].l == now) right(fa), right(now);
        else left(now), right(now);
      else
        if (t[fa].l == now) right(now), left(now);
        else left(fa), left(now);
  }
  up(now);
  if (!FA) root = now;
}

int lower_bound(int v)
{
  int ans = 0, la = 0;
  for (int now = root; now;)
  {
    la = now;
    if (t[now].v >= v) ans = now, now = t[now].l;
    else now = t[now].r;
  }
  splay(la);
  return ans;
}

void insert(int v)
{
  for (int now = root;;)
```

```
  {
    ++t[now].sum;
    if (t[now].v >= v)
      if (t[now].l) now = t[now].l;
      else
      {
        t[now].l = ++tot;
        clear(tot);
        t[tot].sum = 1;
        t[tot].fa = now;
        t[tot].v = v;
        splay(tot);
        return;
      }
    else
      if (t[now].r) now = t[now].r;
      else
      {
        t[now].r = ++tot;
        clear(tot);
        t[tot].sum = 1;
        t[tot].fa = now;
        t[tot].v = v;
        splay(tot);
        return;
      }
  }
}

int get_lower(int a)
{
  splay(a);
  for (a = t[a].l; t[a].r; a = t[a].r);
  return a;
}

int get_upper(int a)
{
  splay(a);
  for (a = t[a].r; t[a].l; a = t[a].l);
  return a;
}

int get_rank(int a)
{
  splay(a);
  return t[t[a].l].sum;
}

void del(int l, int r)
{
  l = get_lower(l);
  r = get_upper(r);
  splay(l);
  splay(r, l);
  t[r].l = 0;
  up(r);
  up(l);
```

```
  }

  int get_kth(int k)
  {
    ++k;
    for (int now = root;;)
    {
      if (t[t[now].l].sum == k - 1)
        return now;
      if (t[t[now].l].sum >= k) now = t[now].l;
      else k -= t[t[now].l].sum + 1, now = t[now].r;
    }
  }
};

//============================Splay=====================
```

8. Link_Cut_Tree // 动态树

```
//============================LinkCutTree===============

const int LCTmaxn = 1000005;

struct LCT_Node
{
  int l, r, v, fa, lazy_swap, lazy_sum, sum, min, max, min_v, max_v, cnt;
  bool root;
};

struct Link_Cut_Tree
{
  private:

  LCT_Node t[LCTmaxn];
  int to[LCTmaxn], tot;
  queue<int> q;

  int mi(int a, int b) { return (a && (t[a].v < t[b].v || !b)) ? a : b; }
  int mx(int a, int b) { return (a && (t[a].v > t[b].v || !b)) ? a : b; }

  int mi2(int a, int a_v, int b, int b_v)  { return (a && (a_v < b_v || !b)) ? a :
b; }
  int mx2(int a, int a_v, int b, int b_v)  { return (a && (a_v > b_v || !b)) ? a :
b; }
  int miv2(int a, int a_v, int b, int b_v) { return (a && (a_v < b_v || !b)) ? a_v :
b_v; }
  int mxv2(int a, int a_v, int b, int b_v) { return (a && (a_v > b_v || !b)) ? a_v :
b_v; }

  void down(int now)
  {
    if (!now) return;
    if (t[now].lazy_swap)
    {
      swap(t[now].l, t[now].r);
      mdf_swap(t[now].l);
      mdf_swap(t[now].r);
      t[now].lazy_swap = 0;
    }
```

```
    if (t[now].lazy_sum)
    {
      int k = t[now].lazy_sum;
      t[now].lazy_sum = 0;
      t[now].sum += t[now].cnt * k;
      t[now].min_v += k;
      t[now].max_v += k;
      t[now].v += k;
      mdf_sum(t[now].l, k);
      mdf_sum(t[now].r, k);
    }
  }

  void up(int now)
  {
    if (!now) return;
    down(now), down(t[now].l), down(t[now].r);
    t[now].sum = t[t[now].l].sum + t[t[now].r].sum + t[now].v;
    t[now].cnt = t[t[now].l].cnt + t[t[now].r].cnt + 1;
    int mi = mi2(t[t[now].l].min, t[t[now].l].min_v, t[t[now].r].min,
t[t[now].r].min_v),
        miv = miv2(t[t[now].l].min, t[t[now].l].min_v, t[t[now].r].min,
t[t[now].r].min_v),
        mx = mx2(t[t[now].l].max, t[t[now].l].max_v, t[t[now].r].max,
t[t[now].r].max_v),
        mxv = mxv2(t[t[now].l].max, t[t[now].l].max_v, t[t[now].r].max,
t[t[now].r].max_v);
    t[now].min   =  mi2(now, t[now].v, mi, miv);
    t[now].min_v = miv2(now, t[now].v, mi, miv);
    t[now].max   =  mx2(now, t[now].v, mx, mxv);
    t[now].max_v = mxv2(now, t[now].v, mx, mxv);
  }

  void mdf_swap(int now)
  {
    if (!now) return;
    t[now].lazy_swap ^= 1;
  }

  void mdf_sum(int now, int v)
  {
    if (!now) return;
    t[now].lazy_sum += v;
  }

  void left(int now)
  {
    int fa = t[now].fa;
    t[now].fa = t[fa].fa;
    if (t[t[fa].fa].l == fa) t[t[fa].fa].l = now;
    if (t[t[fa].fa].r == fa) t[t[fa].fa].r = now;
    t[fa].fa = now;
    t[fa].r = t[now].l;
    t[t[now].l].fa = fa;
    t[now].l = fa;
    t[now].root = t[fa].root;
    t[fa].root = false;
    up(fa);
```

```
  }

  void right(int now)
  {
    int fa = t[now].fa;
    t[now].fa = t[fa].fa;
    if (t[t[fa].fa].l == fa) t[t[fa].fa].l = now;
    if (t[t[fa].fa].r == fa) t[t[fa].fa].r = now;
    t[fa].fa = now;
    t[fa].l = t[now].r;
    t[t[now].r].fa = fa;
    t[now].r = fa;
    t[now].root = t[fa].root;
    t[fa].root = false;
    up(fa);
  }

  void splay(int now)
  {
    if (!now) return;
    down(now);
    while (!t[now].root)
    {
      int fa = t[now].fa;
      if (t[fa].root)
      {
        down(fa), down(now);
        if (t[fa].l == now) right(now);
        else left(now);
      }
      else
      {
        down(t[fa].fa), down(fa), down(now);
        if (t[t[fa].fa].l == fa)
          if (t[fa].l == now) right(fa), right(now);
          else left(now), right(now);
        else
          if (t[fa].l == now) right(now), left(now);
          else left(fa), left(now);
      }
    }
    up(now);
  }

  void bfs(int now, Link_Table &L, int a[])
  {
    to[now] = ++tot;
    q.push(now);
    while (!q.empty())
    {
      int now = q.front();
      q.pop();
      int p = to[now];
      t[p].root = true, t[p].v = t[p].sum = t[p].min_v = t[p].max_v = a[now], t[p].min
= t[p].max = p, t[p].cnt = 1;
      for (int i = L.be(now); i; i = L.next(i))
      {
        int T = L.to(i);
```

```
      if (to[T]) continue;
      to[T] = ++tot;
      t[tot].fa = p;
      q.push(T);
    }
  }
}

void cut_imag(int now)
{
  if (!now) return;
  int fa = t[now].fa;
  if (!fa) return;
  down(fa);
  if (t[fa].l == now) t[fa].l = 0;
  if (t[fa].r == now) t[fa].r = 0;
  t[now].root = true;
  up(fa);
}

void access(int now)
{
  if (!now) return;
  splay(now);
  while (t[now].fa)
  {
    int fa = t[now].fa;
    splay(fa);
    cut_imag(t[fa].r);
    t[fa].r = now;
    t[now].root = false;
    up(fa);
    splay(now);
  }
  cut_imag(t[now].r);
}

int get_root(int now)
{
  for (splay(now); t[now].l; now = t[now].l, down(now));
  //splay(now);
  return now;
}

int get_fa(int now)
{
  for (splay(now), now = t[now].l, down(now); t[now].r; now = t[now].r, down(now));
  //splay(now);
  return now;
}

int lca(int u, int v)
{
  access(u);
  splay(v);
  while (t[v].fa)
  {
    int fa = t[v].fa;
```

```
    splay(fa);
    cut_imag(t[fa].r);
    t[fa].r = v;
    t[v].root = false;
    up(fa);
    v = fa;
  }
  return v;
}

void mk_root(int now)
{
  if (!now) return;
  access(now);
  mdf_swap(now);
  splay(now);
}

public:

void make_root(int now)
{
  now = to[now];
  mk_root(now);
}

void link(int u, int v)
{
  u = to[u], v = to[v];
  mk_root(u);
  mk_root(v);
  splay(v);
  splay(u);
  t[v].fa = u;
  access(u);
}

void cut(int u, int v)
{
  u = to[u], v = to[v];
  int c = lca(u, v);
  if (c == v) swap(u, v);
  access(v);
  splay(u);
  cut_imag(v);
  t[v].fa = 0;
}

int query_sum(int u, int v)
{
  u = to[u], v = to[v];
  int c = lca(u, v), ans = 0;
  access(u);
  splay(c);
  down(t[c].r);
  ans += t[t[c].r].sum;
  access(v);
  splay(c);
```

```
  down(t[c].r);
  ans += t[t[c].r].sum + t[c].v;
  return ans;
}

int query_max(int u, int v)
{
  u = to[u], v = to[v];
  int c = lca(u, v), ans = 0, ans_v = 0;
  access(u);
  splay(c);
  down(t[c].r);
  ans = mx2(t[t[c].r].max, t[t[c].r].max_v, c, t[c].v);
  ans_v = mxv2(t[t[c].r].max, t[t[c].r].max_v, c, t[c].v);
  access(v);
  splay(c);
  down(t[c].r);
  ans = mx2(ans, ans_v, t[t[c].r].max, t[t[c].r].max_v);
  return ans;
}

int query_min(int u, int v)
{
  u = to[u], v = to[v];
  int c = lca(u, v), ans = 0, ans_v = 0;
  access(u);
  splay(c);
  down(t[c].r);
  ans = mi2(t[t[c].r].min, t[t[c].r].min_v, c, t[c].v);
  ans_v = miv2(t[t[c].r].min, t[t[c].r].min_v, c, t[c].v);
  access(v);
  splay(c);
  down(t[c].r);
  ans = mi2(ans, ans_v, t[t[c].r].min, t[t[c].r].min_v);
  return ans;
}

int root(int u)
{
  u = to[u];
  access(u);
  return get_root(u);
}

void modify_sum(int u, int v, int value)
{
  u = to[u], v = to[v];
  mk_root(u);
  access(v);
  mdf_sum(v, value);
}

void create(int n, Link_Table &L, int a[])
{
  for (int i = 1; i <= n; ++i)
    if (!to[i])
      bfs(i, L, a);
}
```

```
  void refresh(int now)
  {
    now = to[now];
    splay(now);
  }

  void clear()
  {
    memset(t, 0, sizeof(t));
    memset(to, 0, sizeof(to));
    tot = 0;
  }

  int get_v(int now)
  {
    splay(now);
    return t[now].v;
  }

  int to_num(int now)
  {
    return to[now];
  }
};

//============================LinkCutTree===============
```

9. Segment_Tree // 线段树

```
//============================SegmentTree===============

const int STmaxn = 400005;
const int STinf = 0x7fffffff;

struct ST_node
{
  int min, max, sum, lazy_sum, lazy_is_b, lazy_is;
  ST_node(int _min = 0, int _max = 0, int _sum = 0,
          int _lazy_sum = 0, int _lazy_is_b = 0, int _lazy_is = 0)
          : min(_min), max(_max), sum(_sum), lazy_sum(_lazy_sum),
            lazy_is_b(_lazy_is_b), lazy_is(_lazy_is) {};
};

struct Segment_Tree
{
  private:
  ST_node t[STmaxn];
  int L, R, num[STmaxn];

  void is(int now, int v)
  {
    if (t[now].lazy_is_b) t[now].lazy_is = v;
    else
    {
      t[now].lazy_sum = 0;
      t[now].lazy_is_b = 1;
      t[now].lazy_is = v;
    }
```

```
  t[now].min = t[now].max = v;
  t[now].sum = num[now] * v;
}

void sum(int now, int v)
{
  if (t[now].lazy_is_b) t[now].lazy_is += v;
  else t[now].lazy_sum += v;
  t[now].max += v;
  t[now].min += v;
  t[now].sum += num[now] * v;
}

void lazy(int now)
{
  if (now * 2 + 1 >= STmaxn) return;
  int l = now * 2, r = now * 2 + 1;
  if (t[now].lazy_sum)
  {
    sum(l, t[now].lazy_sum);
    sum(r, t[now].lazy_sum);
    t[now].lazy_sum = 0;
  }
  if (t[now].lazy_is_b)
  {
    is(l, t[now].lazy_is);
    is(r, t[now].lazy_is);
    t[now].lazy_is_b = 0;
  }
}

void up(int now)
{
  if (now * 2 + 1 >= STmaxn) return;
  int l = now * 2, r = now * 2 + 1;
  lazy(l), lazy(r);
  t[now].min = min(t[l].min, t[r].min);
  t[now].max = max(t[l].max, t[r].max);
  t[now].sum = t[l].sum + t[r].sum;
}

void mk_tree(int l, int r, int now, int a[])
{
  num[now] = r - l + 1;
  if (l == r)
  {
    t[now] = ST_node(a[l], a[l], a[l]);
    return;
  }
  int mid = (l + r) / 2;
  mk_tree(l, mid, now * 2, a), mk_tree(mid + 1, r, now * 2 + 1, a);
  up(now);
}

void my_is(int l, int r, int now, int lf, int rt, int v)
{
  lazy(now);
  if (l >= lf && r <= rt)
```

```
  {
    is(now, v);
    return;
  }
  int mid = (l + r) / 2;
  if (lf <= mid) my_is(l, mid, now * 2, lf, rt, v);
  if (rt >= mid + 1) my_is(mid + 1, r, now * 2 + 1, lf, rt, v);
  up(now);
}

void my_sum(int l, int r, int now, int lf, int rt, int v)
{
  lazy(now);
  if (l >= lf && r <= rt)
  {
    sum(now, v);
    return;
  }
  int mid = (l + r) / 2;
  if (lf <= mid) my_sum(l, mid, now * 2, lf, rt, v);
  if (rt >= mid + 1) my_sum(mid + 1, r, now * 2 + 1, lf, rt, v);
  up(now);
}

int qy_max(int l, int r, int now, int lf, int rt)
{
  lazy(now);
  if (l >= lf && r <= rt) return t[now].max;
  int mid = (l + r) / 2, ans = -STinf;
  if (lf <= mid) ans = max(ans, qy_max(l, mid, now * 2, lf, rt));
  if (rt >= mid + 1) ans = max(ans, qy_max(mid + 1, r, now * 2 + 1, lf, rt));
  return ans;
}

int qy_min(int l, int r, int now, int lf, int rt)
{
  lazy(now);
  if (l >= lf && r <= rt) return t[now].min;
  int mid = (l + r) / 2, ans = STinf;
  if (lf <= mid) ans = min(ans, qy_min(l, mid, now * 2, lf, rt));
  if (rt >= mid + 1) ans = min(ans, qy_min(mid + 1, r, now * 2 + 1, lf, rt));
  return ans;
}

int qy_sum(int l, int r, int now, int lf, int rt)
{
  lazy(now);
  if (l >= lf && r <= rt) return t[now].sum;
  int mid = (l + r) / 2, ans = 0;
  if (lf <= mid) ans += qy_sum(l, mid, now * 2, lf, rt);
  if (rt >= mid + 1) ans += qy_sum(mid + 1, r, now * 2 + 1, lf, rt);
  return ans;
}

public:
void modify_is(int l, int r, int v)
{
  my_is(L, R, 1, l, r, v);
```

```
  }

  void modify_sum(int l, int r, int v)
  {
    my_sum(L, R, 1, l, r, v);
  }

  int query_max(int l, int r)
  {
    return qy_max(L, R, 1, l, r);
  }

  int query_min(int l, int r)
  {
    return qy_min(L, R, 1, l, r);
  }

  int query_sum(int l, int r)
  {
    return qy_sum(L, R, 1, l, r);
  }

  void create(int l, int r, int a[])
  {
    L = l;
    R = r;
    mk_tree(L, R, 1, a);
  }

  void clear()
  {
    L = R = 0;
    memset(t, 0, sizeof(t));
  }
};

//============================SegmentTree===============
```

10. Tree_Chain_Division // **树链剖分**

```
//============================TreeChainDivision=========

const int TCDmaxn = 100005;

struct Tree_Chain_Division
{
  private:
  Segment_Tree t;
  int last[TCDmaxn], d[TCDmaxn], to[TCDmaxn], sum[TCDmaxn], fa[TCDmaxn], p, tot,
a[TCDmaxn], n;
  queue<par> q;
  stack<int> s;

  void build(Link_Table_V &L)
  {
    q.push(par(1, 0));
    d[1] = 1;
    while (!q.empty())
    {
```

```
    par now = q.front();
    q.pop();
    s.push(now.a);
    for (int i = L.be(now.a); i; i = L.next(i))
    {
      int to = L.to(i);
      if (to == now.b) continue;
      d[to] = d[now.a] + 1;
      fa[to] = now.a;
      q.push(par(to, now.a));
    }
  }
  while (!s.empty())
  {
    int now = s.top();
    s.pop();
    for (int i = L.be(now); i; i = L.next(i))
    {
      int to = L.to(i);
      if (d[to] == d[now] - 1) continue;
      sum[now] += sum[to];
    }
    ++sum[now];
  }
  q.push(par(1, 1));
  a[to[1] = 1] = 0;
  last[1] = 1;
  while (!q.empty())
  {
    par now = q.front();
    q.pop();
    if (sum[now.a] == 1) continue;
    int mx = 0;
    ++now.b;
    for (int i = L.be(now.a); i; i = L.next(i))
    {
      int to = L.to(i);
      if (d[to] == d[now.a] - 1) continue;
      if (sum[L.to(mx)] < sum[to]) mx = i;
    }
    a[to[L.to(mx)] = now.b] = L.v(mx);
    q.push(par(L.to(mx), now.b));
    now.b += sum[L.to(mx)];
    last[L.to(mx)] = last[now.a];
    for (int i = L.be(now.a); i; i = L.next(i))
    {
      int _to = L.to(i);
      if (i == mx || d[_to] == d[now.a] - 1) continue;
      a[to[_to] = now.b] = L.v(i);
      q.push(par(_to, now.b));
      now.b += sum[_to];
      last[_to] = _to;
    }
  }
}

void build(Link_Table &L, int v[])
{
```

```
q.push(par(1, 0));
d[1] = 1;
while (!q.empty())
{
  par now = q.front();
  q.pop();
  s.push(now.a);
  for (int i = L.be(now.a); i; i = L.next(i))
  {
    int to = L.to(i);
    if (to == now.b) continue;
    d[to] = d[now.a] + 1;
    fa[to] = now.a;
    q.push(par(to, now.a));
  }
}
while (!s.empty())
{
  int now = s.top();
  s.pop();
  for (int i = L.be(now); i; i = L.next(i))
  {
    int to = L.to(i);
    if (d[to] == d[now] - 1) continue;
    sum[now] += sum[to];
  }
  ++sum[now];
}
q.push(par(1, 1));
a[to[1] = 1] = v[1];
last[1] = 1;
while (!q.empty())
{
  par now = q.front();
  q.pop();
  if (sum[now.a] == 1) continue;
  int mx = 0;
  ++now.b;
  for (int i = L.be(now.a); i; i = L.next(i))
  {
    int to = L.to(i);
    if (d[to] == d[now.a] - 1) continue;
    if (sum[L.to(mx)] < sum[to]) mx = i;
  }
  a[to[L.to(mx)] = now.b] = v[L.to(mx)];
  q.push(par(L.to(mx), now.b));
  now.b += sum[L.to(mx)];
  last[L.to(mx)] = last[now.a];
  for (int i = L.be(now.a); i; i = L.next(i))
  {
    int _to = L.to(i);
    if (i == mx || d[_to] == d[now.a] - 1) continue;
    a[to[_to] = now.b] = v[_to];
    q.push(par(_to, now.b));
    now.b += sum[_to];
    last[_to] = _to;
  }
}
```

```
  }

  int qy_sum(int u, int num)
  {
    if (!num) return 0;
    if (d[u] - d[last[u]] + 1 >= num) return t.query_sum(to[u] - num + 1, to[u]);
    else return qy_sum(fa[last[u]], num - (d[u] - d[last[u]] + 1)) +
t.query_sum(to[last[u]], to[u]);
  }

  int qy_max(int u, int num)
  {
    if (!num) return 0;
    if (d[u] - d[last[u]] + 1 >= num) return t.query_max(to[u] - num + 1, to[u]);
    else return max(qy_max(fa[last[u]], num - (d[u] - d[last[u]] + 1)),
t.query_max(to[last[u]], to[u]));
  }

  int qy_min(int u, int num)
  {
    if (!num) return 0;
    if (d[u] - d[last[u]] + 1 >= num) return t.query_min(to[u] - num + 1, to[u]);
    else return min(qy_min(fa[last[u]], num - (d[u] - d[last[u]] + 1)),
t.query_min(to[last[u]], to[u]));
  }

  void my_sum(int u, int num, int va)
  {
    if (!num) return;
    if (d[u] - d[last[u]] + 1 >= num) t.modify_sum(to[u] - num + 1, to[u], va);
    else my_sum(fa[last[u]], num - (d[u] - d[last[u]] + 1), va),
t.modify_sum(to[last[u]], to[u], va);
  }

  void my_is(int u, int num, int va)
  {
    if (!num) return;
    if (d[u] - d[last[u]] + 1 >= num) t.modify_is(to[u] - num + 1, to[u], va);
    else my_is(fa[last[u]], num - (d[u] - d[last[u]] + 1), va),
t.modify_is(to[last[u]], to[u], va);
  }

  public:

  int lca(int u, int v)
  {
    while (last[u] != last[v])
    {
      if (d[last[u]] < d[last[v]]) swap(u, v);
      u = fa[last[u]];
    }
    if (d[u] > d[v]) swap(u, v);
    return u;
  }

  int query_sum(int u, int v)
  {
    int c = lca(u, v);
```

```
    if (p)
      return qy_sum(u, d[u] - d[c]) + qy_sum(v, d[v] - d[c]);
    else
      return qy_sum(u, d[u] - d[c]) + qy_sum(v, d[v] - d[c] + 1);
  }

  int query_max(int u, int v)
  {
    int c = lca(u, v);
    if (p)
      return max(qy_max(u, d[u] - d[c]), qy_max(v, d[v] - d[c]));
    else
      return max(qy_max(u, d[u] - d[c]), qy_max(v, d[v] - d[c] + 1));
  }

  int query_min(int u, int v)
  {
    int c = lca(u, v);
    if (p)
      return min(qy_min(u, d[u] - d[c]), qy_min(v, d[v] - d[c]));
    else
      return min(qy_min(u, d[u] - d[c]), qy_min(v, d[v] - d[c] + 1));
  }

  void modify_sum(int u, int v, int va)
  {
    int c = lca(u, v);
    if (p)
      my_sum(u, d[u] - d[c], va), my_sum(v, d[v] - d[c], va);
    else
      my_sum(u, d[u] - d[c], va), my_sum(v, d[v] - d[c] + 1, va);
  }

  void modify_is(int u, int v, int va)
  {
    int c = lca(u, v);
    if (p)
      my_is(u, d[u] - d[c], va), my_is(v, d[v] - d[c], va);
    else
      my_is(u, d[u] - d[c], va), my_is(v, d[v] - d[c] + 1, va);
  }

  void create(int N, Link_Table_V &L)
  {
    p = 1;
    n = N;
    build(L);
    t.create(1, n, a);
  }

  void create(int N, Link_Table &L, int a[])
  {
    p = 0;
    n = N;
    build(L, a);
    t.create(1, n, a);
  }
};
```

```
//============================TreeChainDivision=========
```

11. KD_Tree // kd树

```
//============================KDTree====================

const int KDTmaxp = 5;
const int KDTmaxn = 100005;
const int KDTinf = 0x7fffffff;
int P;

struct KDT_Point
{
  int xy[KDTmaxp], num, dist;
};

struct KDT_Node
{
  KDT_Point p;
  int max[KDTmaxp], min[KDTmaxp], l, r;
  KDT_Node(KDT_Point k)
  {
    l = r = 0;
    p = k;
    for (int i = 0; i < KDTmaxp; ++i)
      max[i] = min[i] = k.xy[i];
  }
  KDT_Node() {}
};

bool operator<(KDT_Point a, KDT_Point b)
{
    return a.xy[P] < b.xy[P];
}

struct KD_Tree
{
  private:

  KDT_Node t[KDTmaxn];
  int n, tot, root, KDTp;

  int sqr(int a) { return a * a; }

  void up(int now)
  {
    for (int i = 0; i < KDTp; ++i)
    {
      if (t[now].l)
        t[now].max[i] = max(t[now].max[i], t[t[now].l].max[i]),
        t[now].min[i] = min(t[now].min[i], t[t[now].l].min[i]);
      if (t[now].r)
        t[now].max[i] = max(t[now].max[i], t[t[now].r].max[i]),
        t[now].min[i] = min(t[now].min[i], t[t[now].r].min[i]);
    }
  }

  int build(int l, int r, KDT_Point K[], int p)
```

```
{
  if (l > r) return 0;
  int now = ++tot;
  P = p;
  sort(K + l, K + r + 1);
  int mid = (l + r) / 2;
  t[now] = KDT_Point(K[mid]);
  t[now].l = build(l, mid - 1, K, (p + 1) % KDTp);
  t[now].r = build(mid + 1, r, K, (p + 1) % KDTp);
  up(now);
  return now;
}

void ins(int now, KDT_Point k, int p)
{
  if (t[now].p.xy[p] > k.xy[p])
  {
    if (!t[now].l)
    {
      t[now].l = ++tot;
      t[tot] = KDT_Point(k);
    }
    else ins(t[now].l, k, (p + 1) % KDTp);
  }
  else
  {
    if (!t[now].r)
    {
      t[now].r = ++tot;
      t[tot] = KDT_Point(k);
    }
    else ins(t[now].r, k, (p + 1) % KDTp);
  }
  up(now);
}

void re_min(int now, KDT_Point &a, KDT_Point ans[], int k)
{
  int i = k;
  int di = dist(t[now].p, a);
  t[now].p.dist = di;
  for (; i >= 1 && ans[i].dist >= di; --i);
  if (i == k) return;
  ++i;
  for (int j = k; j > i; --j) ans[j] = ans[j - 1];
  ans[i] = t[now].p;
}

void qy_min(int now, KDT_Point &a, KDT_Point ans[], int k, int p)
{
  if (!now) return;
  KDT_Point P;
  for (int i = 0; i < KDTp; ++i)
    P.xy[i] = (a.xy[i] <= t[now].max[i] && a.xy[i] >= t[now].min[i]) ? a.xy[i]
              : (abs(t[now].max[i] - a.xy[i]) < abs(t[now].min[i] - a.xy[i])
                 ? t[now].max[i] : t[now].min[i]);
  if (dist(P, a) > ans[k].dist) return;
  re_min(now, a, ans, k);
```

```
    if (a.xy[p] > t[now].p.xy[p])
    {
      qy_min(t[now].r, a, ans, k, (p + 1) % KDTp);
      qy_min(t[now].l, a, ans, k, (p + 1) % KDTp);
    }
    else
    {
      qy_min(t[now].l, a, ans, k, (p + 1) % KDTp);
      qy_min(t[now].r, a, ans, k, (p + 1) % KDTp);
    }
  }

  void re_max(int now, KDT_Point &a, KDT_Point ans[], int k)
  {
    int i = k;
    int di = dist(t[now].p, a);
    t[now].p.dist = di;
    for (; i >= 1 && (ans[i].dist < di || (ans[i].dist == di && ans[i].num >
t[now].p.num)); --i);
    if (i == k) return;
    ++i;
    for (int j = k; j > i; --j) ans[j] = ans[j - 1];
    ans[i] = t[now].p;
  }

  void qy_max(int now, KDT_Point &a, KDT_Point ans[], int k, int p)
  {
    if (!now) return;
    KDT_Point P;
    for (int i = 0; i < KDTp; ++i)
      P.xy[i] = abs(t[now].max[i] - a.xy[i]) > abs(t[now].min[i] - a.xy[i])
                ? t[now].max[i] : t[now].min[i];
    if (dist(P, a) < ans[k].dist) return;
    re_max(now, a, ans, k);
    if (a.xy[p] < t[now].p.xy[p])
    {
      qy_max(t[now].r, a, ans, k, (p + 1) % KDTp);
      qy_max(t[now].l, a, ans, k, (p + 1) % KDTp);
    }
    else
    {
      qy_max(t[now].l, a, ans, k, (p + 1) % KDTp);
      qy_max(t[now].r, a, ans, k, (p + 1) % KDTp);
    }
  }

  public:

  void create(int N, KDT_Point K[], int p)
  {
    n = N;
    KDTp = p;
    root = build(0, n - 1, K, 0);
  }

  void insert(KDT_Point k)
  {
    ins(root, k, 0);
```

```
  }

  void query_max(KDT_Point &a, KDT_Point ans[], int k)
  {
    for (int i = 0; i <= k; ++i)
      ans[i].dist = -KDTinf;
    qy_max(root, a, ans, k, 0);
  }

  void query_min(KDT_Point &a, KDT_Point ans[], int k)
  {
    for (int i = 0; i <= k; ++i)
      ans[i].dist = KDTinf;
    qy_min(root, a, ans, k, 0);
  }

  int dist(KDT_Point a, KDT_Point b)
  {
    int ans = 0;
    for (int i = 0; i < KDTp; ++i)
      ans += sqr(a.xy[i] - b.xy[i]);
    return ans;
  }

  void clear()
  {
    n = tot = root = KDTp = 0;
    memset(t, 0, sizeof(t));
  }

};

//============================KDTree====================
```

12. Network_Flow // 最大流

```
//============================NetworkFlow===============

const int NFmaxn = 1005;
const int NFmaxm = 1005;
const int NFinf = 0x7fffffff;

struct NF_Line
{
  int to, next, v, opt;
};

struct Network_Flow
{
  NF_Line li[NFmaxm];
  int be[NFmaxn], l, s, t, n, num[NFmaxn], note[NFmaxn];

  void makeline(int fr, int to, int v)
  {
    ++l;
    li[l].next = be[fr];
    be[fr] = l;
    li[l].to = to;
    li[l].v = v;
```

```
  li[l].opt = l + 1;

  ++l;
  li[l].next = be[to];
  be[to] = l;
  li[l].to = fr;
  li[l].v = 0;
  li[l].opt = l - 1;
}

void create(int N)
{
  n = N;
}

int sap(int now, int maxf)
{
  if (now == t) return maxf;
  int mi = n, tot = 0;
  for (int i = be[now]; i; i = li[i].next)
  {
    int to = li[i].to;
    if (li[i].v && note[to] + 1 == note[now])
    {
      int k = sap(to, min(maxf - tot, li[i].v));
      li[i].v -= k;
      tot += k;
      li[li[i].opt].v += k;
      if (note[s] >= n || tot == maxf) return tot;
    }
    if (li[i].v) mi = min(mi, note[to]);
  }
  if (!tot)
  {
    if (!--num[note[now]])
    {
      note[s] = n;
      return 0;
    }
    ++num[note[now] = mi + 1];
  }
  return tot;
}

int query(int S, int T)
{
  s = S, t = T;
  memset(num, 0, sizeof(num));
  memset(note, 0, sizeof(note));
  num[0] = n;
  int ans = 0;
  while (note[s] < n) ans += sap(s, NFinf);
  return ans;
}

void clear()
{
  l = s = t = n = 0;
```

```
    memset(be, 0, sizeof(be));
    memset(num, 0, sizeof(num));
    memset(note, 0, sizeof(note));
  }
};

//============================NetworkFlow===============
```

13. Network_Cost_Flow_Spfa // **最小费用最大流** spfa**版**

```
//============================NetworkCostFlowSpfa=======

const int NCFSmaxn = 10005;
const int NCFSmaxm = 100005;
const int NCFSinf_ = 0x7f;
const int NCFSinf  = 0x7f7f7f7f;

struct NCFS_Line
{
  int fr, to, next, v, c, opt;
};

struct Network_Cost_Flow_Spfa
{
  NCFS_Line li[NCFSmaxm];
  int be[NCFSmaxn], l, s, t, n, dist[NCFSmaxn], fa[NCFSmaxn], b[NCFSmaxn];
  deque<int> q;

  void makeline(int fr, int to, int v, int c)
  {
    ++l;
    li[l].next = be[fr];
    be[fr] = l;
    li[l].fr = fr;
    li[l].to = to;
    li[l].v = v;
    li[l].c = c;
    li[l].opt = l + 1;

    ++l;
    li[l].next = be[to];
    be[to] = l;
    li[l].fr = to;
    li[l].to = fr;
    li[l].v = 0;
    li[l].c = -c;
    li[l].opt = l - 1;
  }

  void create(int N)
  {
    n = N;
  }

  void clear()
  {
    l = s = t = n = 0;
    memset(be, 0, sizeof(be));
  }
```

```
  bool spfa()
  {
    memset(dist, NCFSinf_, sizeof(dist));
    memset(fa, 0, sizeof(fa));
    dist[t] = 0;
    b[t] = 1;
    q.push_back(t);
    while (!q.empty())
    {
      int now = q.front();
      q.pop_front();
      for (int i = be[now]; i; i = li[i].next)
      {
        int to = li[i].to;
        if (!li[li[i].opt].v || dist[to] <= dist[now] - li[i].c) continue;
        dist[to] = dist[now] - li[i].c;
        fa[to] = i;
        if (!b[to])
        {
          b[to] = 1;
          if (!q.empty() && dist[to] < dist[q.front()]) q.push_front(to);
          else q.push_back(to);
        }
      }
      b[now] = 0;
    }
    return dist[s] != NCFSinf;
  }
  par query(int S, int T)
  {
    par ans;
    ans.a = ans.b = 0;
    s = S, t = T;
    while (spfa())
    {
      int mi = NCFSinf;
      for (int i = s; i != t; i = li[fa[i]].fr)
        mi = min(mi, li[li[fa[i]].opt].v);
      for (int i = s; i != t; i = li[fa[i]].fr)
      {
        li[fa[i]].v += mi;
        li[li[fa[i]].opt].v -= mi;
        ans.b -= mi * li[fa[i]].c;
      }
      ans.a += mi;
    }
    return ans;
  }
};

//============================NetworkCostFlowSpfa=======
```

14. Network_Cost_Flow_Zkw // **最小费用最大流** zkw**版**

```
//============================NetworkCostFlowZkw========

const int NCFZmaxn = 10005;
const int NCFZmaxm = 100005;
```

```
const int NCFZinf_ = 0x7f;
const int NCFZinf  = 0x7f7f7f7f;

struct NCFZ_Line
{
  int fr, to, next, v, c, opt;
};

struct Network_Cost_Flow_Zkw
{
  NCFZ_Line li[NCFZmaxm];
  int be[NCFZmaxn], l, s, t, dist[NCFZmaxn], b[NCFZmaxn];
  deque<int> q;

  void makeline(int fr, int to, int v, int c)
  {
    ++l;
    li[l].next = be[fr];
    be[fr] = l;
    li[l].fr = fr;
    li[l].to = to;
    li[l].v = v;
    li[l].c = c;
    li[l].opt = l + 1;

    ++l;
    li[l].next = be[to];
    be[to] = l;
    li[l].fr = to;
    li[l].to = fr;
    li[l].v = 0;
    li[l].c = -c;
    li[l].opt = l - 1;
  }

  void create()
  {
  }

  void clear()
  {
    l = s = t = 0;
    memset(be, 0, sizeof(be));
    memset(b, 0, sizeof(b));
  }

  bool spfa()
  {
    memset(dist, NCFZinf_, sizeof(dist));
    memset(b, 0, sizeof(b));
    dist[t] = 0;
    b[t] = 1;
    q.push_back(t);
    while (!q.empty())
    {
      int now = q.front();
      q.pop_front();
      for (int i = be[now]; i; i = li[i].next)
```

```
      {
        int to = li[i].to;
        if (!li[li[i].opt].v || dist[to] <= dist[now] - li[i].c) continue;
        dist[to] = dist[now] - li[i].c;
        if (!b[to])
        {
          b[to] = 1;
          if (!q.empty() && dist[to] < dist[q.front()]) q.push_front(to);
          else q.push_back(to);
        }
      }
      b[now] = 0;
    }
    return dist[s] != NCFZinf;
  }

  int sap(int now, int maxf)
  {
    if (now == t) return maxf;
    int tot = 0;
    b[now] = 1;
    for (int i = be[now]; i; i = li[i].next)
    {
      int to = li[i].to;
      if (!b[to] && li[i].v && dist[to] == dist[now] - li[i].c)
      {
        int k = sap(to, min(maxf - tot, li[i].v));
        li[i].v -= k;
        li[li[i].opt].v += k;
        tot += k;
      }
    }
    return tot;
  }

  par query(int S, int T)
  {
    par ans;
    ans.a = ans.b = 0;
    s = S, t = T;
    while (spfa())
      while (int k = sap(s, NCFZinf))
      {
        memset(b, 0, sizeof(b));
        ans.a += k;
        ans.b += k * dist[s];
      }
    return ans;
  }
};

//============================NetworkCostFlowZkw========
```

15. Network_Flow_Up_Down // 上下界流

```
//============================NetworkFlowUpDown=========

const int NFUDinf  = 0x7f7f7f7f;
```

```
struct NFUD_Line
{
  int fr, to, next, vu, vd, c, opt;
};

struct Network_Flow_Up_Down
{
  Network_Flow NF;
  int s, t, ss, tt, li, fi;

  void makeline(int fr, int to, int vd, int vu)
  {
    NF.makeline(fr, to, vu - vd);
    NF.makeline(ss, to, vd);
    NF.makeline(fr, tt, vd);
    fi += vd;
  }

  void create(int n, int S, int T, int SS, int TT)
  {
    s = S, t = T, ss = SS, tt = TT;
    NF.create(n);
    NF.makeline(t, s, NFUDinf);
    li = NF.l;
  }

  int query()
  {
    int ans = 0, p;
    p = NF.query(ss, tt);
    if (p != fi) return -1;
    p = NF.query(s, t);
    ans += p;
    return ans;
  }

  void clear()
  {
    s = t = ss = tt = li = 0;
    NF.clear();
  }
};

//============================NetworkFlowUpDown=========
```

16. Network_Cost_Flow_Up_Down // 上下界费用流

```
//============================NetworkCostFlowUpDown=====

const int NCFUDinf  = 0x7f7f7f7f;

struct NCFUD_Line
{
  int fr, to, next, vu, vd, c, opt;
};

struct Network_Cost_Flow_Up_Down
{
  Network_Cost_Flow_Zkw NCFZ;
```

```
  int s, t, ss, tt, li, mo, fi;

  void makeline(int fr, int to, int vd, int vu, int c = 0)
  {
    NCFZ.makeline(fr, to, vu - vd, c);
    NCFZ.makeline(ss, to, vd, 0);
    NCFZ.makeline(fr, tt, vd, 0);
    fi += vd;
    mo += vd * c;
  }

  void create(int S, int T, int SS, int TT)
  {
    s = S, t = T, ss = SS, tt = TT;
    NCFZ.create();
    NCFZ.makeline(t, s, NCFUDinf, 0);
    li = NCFZ.l;
  }

  par query()
  {
    par ans, p;
    p = NCFZ.query(ss, tt);
    if (p.a != fi) return par(-1, -1);
    ans.b = mo + p.b;
    ans.a = 0;
    p = NCFZ.query(s, t);
    ans.a += p.a, ans.b += p.b;
    return ans;
  }

  void clear()
  {
    s = t = ss = tt = li = mo = 0;
    NCFZ.clear();
  }
};

//============================NetworkCostFlowUpDown=====
```

17. Mergeable_Tree // **可并堆**

```
//============================MergeableTree=============

const int MHmaxn = 100005;

struct MHnode
{
  int l, r, dis, v;
};

struct MergeableHeap  //big first
{
  MHnode t[MHmaxn];
  int tot;

  void create()
  {
```

```
  }

  int merge(int a, int b)
  {
    if (!a) return b;
    if (!b) return a;
    if (t[a].v < t[b].v) swap(a, b);
    t[a].r = merge(t[a].r, b);
    if (t[t[a].l].dis < t[t[a].r].dis) swap(t[a].l, t[a].r);
    t[a].dis = t[t[a].r].dis + 1;
    return a;
  }

  int make_node(int v)
  {
    ++tot;
    t[tot].l = t[tot].r = t[tot].dis = 0;
    t[tot].v = v;
    return tot;
  }

  int pop(int root)
  {
    return merge(t[root].l, t[root].r);
  }

  int top(int root)
  {
    return t[root].v;
  }

  void clear()
  {
    tot = 0;
  }
};

//============================MergeableTree=============
```

18. Hash_Map // hash

```
//============================HashMap===================

const int HMmaxn = 1000000;

struct HM_Member_V
{
  int v;
};

struct HM_Member
{
  int n, a[20];
  HM_Member_V v;
};

struct HM_Line
{
  int next;
```

```
  HM_Member to;
};

struct Hash_Map
{
  static const int MOD = 19961107;
  static const int MOD2 = 1234567891;
  HM_Line li[HMmaxn];
  int be[MOD], l;

  void create()
  {

  }

  int hash(const HM_Member& a)
  {
    int ans = a.n;
    for (int i = 0; i < a.n; ++i)
      ans = (((ll)ans) * MOD2 + a.a[i]) % MOD;
    return ans;
  }

  void makeline(int fr, const HM_Member& to)
  {
    ++l;
    li[l].next = be[fr];
    be[fr] = l;
    li[l].to = to;
  }

  bool same(const HM_Member& a, const HM_Member& b)
  {
    bool br = a.n == b.n;
    for (int i = 0; i < a.n; ++i)
      if (a.a[i] != b.a[i])
      {
        br = false;
        break;
      }
    return br;
  }

  bool check(const HM_Member& a)
  {
    int h = hash(a);
    for (int i = be[h]; i; i = li[i].next)
      if (same(li[i].to, a))
        return true;
    return false;
  }

  HM_Member_V get(const HM_Member& a)
  {
    int h = hash(a);
    for (int i = be[h]; i; i = li[i].next)
      if (same(li[i].to, a))
        return li[i].to.v;
```

```
    HM_Member_V k;
    return k;
  }

  void insert(const HM_Member& a)
  {
    int h = hash(a);
    if (check(a))
    {
      for (int i = be[h]; i; i = li[i].next)
        if (same(li[i].to, a))
          {
            li[i].to.v = a.v;
            break;
          }
    }
    else makeline(h, a);
  }

  void clear()
  {
    l = 0;
    memset(be, 0, sizeof(be));
  }
};

//============================HashMap===================
```

19. Geometry_Base // 几何_基本

```
//============================Geometry_Base=============

struct Geometry_Base
{
       public:

       static const double eps = 1e-8;
       double pi;

       struct point
       {
              double x, y;

              point() {}
              point(double _x, double _y) : x(_x), y(_y) {}

              point operator-(point a) { return point(x - a.x, y - a.y); }
              point operator+(point a) { return point(x + a.x, y + a.y); }
              point operator*(double a) { return point(x * a, y * a); }
              point operator/(double a) { return point(x / a, y / a); }
              double operator&(point a) { return x * a.y - y * a.x; }
              double operator|(point a) { return x * a.x + y * a.y; }

              bool operator==(point a) { return !cmp(x, a.x) && !cmp(y, a.y); }
              bool operator!=(point a) { return !(!cmp(x, a.x) && !cmp(y, a.y)); }
              bool operator<(point a) const { return cmp(x, a.x) == -1 || (cmp(x, a.x)
== 0 && cmp(y, a.y) == -1); }

              point operator-=(point a) { x -= a.x, y -= a.y; return *this; }
```

```
		point operator+=(point a) { x += a.x, y += a.y; return *this; }
		point operator*=(double a) { x *= a, y *= a; return *this; }
		point operator/=(double a) { x /= a, y /= a; return *this; }
	};

	struct segment
	{
		point a, b;
		segment() {};
		segment(point _a, point _b) : a(_a), b(_b) {};
		bool operator==(segment A) { return (a == A.a && b == A.b) || (b == A.a
&& a == A.b); }
		bool operator!=(segment A) { return !((a == A.a && b == A.b) || (b == A.a
&& a == A.b)); }
	};

	Geometry_Base() { pi = acos(-1.); }
	void create() {}
	double sqr(double a) { return a * a; }

	point rotate(point a, double b) {return point(a.x * cos(b) - a.y * sin(b), a.x
* sin(b) + a.y * cos(b)); }
	double dist(point a, point b) { return sqrt(sqr(a.x - b.x) + sqr(a.y - b.y)); }
	static int cmp(double a, double b) { return abs(a - b) < eps ? 0 : (a < b ? -1 :
1); } // (0 =) (-1 <) (1 >)
	int pos(point a, point b) { return cmp(a & b, 0); } // (0 =) (-1 right) (1 left)
	int pos(segment a, segment b) { return pos(a.b - a.a, b.b - b.a); } // (0 =) (-1
right) (1 left)
	int pos(segment a, point b) { return pos(a, segment(a.a, b)); } // (0 =) (-1
right) (1 left)
	bool init(segment a, point b)    {     return !cmp(dist(a.a, a.b), dist(a.a, b)
+ dist(a.b, b));    }
	segment get_vertical(segment a, point b) { return segment(b, b + rotate(a.b -
a.a, pi / 2)); }
	point get_foot(segment a, point b) { return cross(a, get_vertical(a, b));}
	point get_mid(segment a) { return (a.a + a.b) / 2; }
	double dist(segment a, point b)  {     return dist(get_foot(a, b), b); }
	double dist2(segment a, point b) { point p = get_foot(a, b); return init(a, p) ?
dist(p, b) : min(dist(a.a, b), dist(a.b, b)); }

	int check(segment a, segment b)
	{
		if (pos(a, b) == 0)
		{
			if (pos(a, b.a)) return 0;
			if (!cmp(a.a.x, a.b.x))
			{
				if (a.a.y > a.b.y) swap(a.a, a.b);
				if (b.a.y > b.b.y) swap(b.a, b.b);
				if (a.a.y > b.a.y) swap(a, b);
				return cmp(a.b.y, b.a.y) >= 0 ? -1 : 0;
			}
			else
			{
				if (a.a.x > a.b.x) swap(a.a, a.b);
				if (b.a.x > b.b.x) swap(b.a, b.b);
				if (a.a.x > b.a.x) swap(a, b);
				return cmp(a.b.x, b.a.x) >= 0 ? -1 : 0;
```

```
                }
            }
            return pos(a, b.a) * pos(a, b.b) <= 0 && pos(b, a.a) * pos(b, a.b) <= 0 ?
1 : 0;
        }

        double get_k(segment a)
        {
            if (!cmp(a.a.x, a.b.x)) return 0;
            return (a.b.y - a.a.y) / (a.b.x - a.a.x);
        }

        double get_b(segment a)
        {
            if (!cmp(a.a.x, a.b.x)) return 0;
            return a.a.y - a.a.x * get_k(a);
        }

        point cross(segment a, segment b)
        {
            if (!pos(a, b)) return point(0, 0);
            if (!cmp(b.a.x, b.b.x)) swap(a, b);
            if (!cmp(a.a.x, a.b.x))
            {
                double k = (b.b.y - b.a.y) / (b.b.x - b.a.x),
                                b_ = b.a.y - b.a.x * k;
                return point(a.a.x, k * a.a.x + b_);
            }
            else
            {
                double k1 = (a.b.y - a.a.y) / (a.b.x - a.a.x),
                                k2 = (b.b.y - b.a.y) / (b.b.x - b.a.x),
                                b1 = a.a.y - a.a.x * k1,
                                b2 = b.a.y - b.a.x * k2,
                                x  = (b2 - b1) / (k1 - k2),
                                y  = k1 * x + b1;
                return point(x, y);
            }
        }
};
typedef Geometry_Base::point gbp;
typedef Geometry_Base::segment gbs;

//============================Geometry_Base=============
```

20. Geometry_Polygon // 几何_多边形

```
//============================Geometry_Polygon==========

struct Geometry_Polygon
{
        static const int GPmaxn = 1005;
        Geometry_Base gb;
        int n;
        gbp a[GPmaxn], mx, t[GPmaxn + 10];
        void create(int _n, vector<gbp> _a)
        {
            n = _n;
            mx = _a[0];
```

```
	for (int i = 0; i < n; ++i)
	{
		a[i] = _a[i];
		mx.x = max(mx.x, a[i].x);
		mx.y = max(mx.y, a[i].y);
	}
}

void clear()
{
	n = 0;
}

double get_s()
{
	double ans = a[n - 1] & a[0];
	for (int i = 1; i < n; ++i)
		ans += a[i - 1] & a[i];
	return abs(ans);
}

int check(gbp p) //0外部 -1边界 1内部
{
	if (gb.init(gbs(a[n - 1], a[0]), p)) return -1;
	for (int i = 1; i < n; ++i)
		if (gb.init(gbs(a[i - 1], a[i]), p))
			return -1;
	gbs s = gbs(p, mx + gbp(rand(), rand()));
	int ans = gb.check(s, gbs(a[n - 1], a[0])) ? 1 : 0;
	for (int i = 1; i < n; ++i)
		ans += gb.check(s, gbs(a[i - 1], a[i])) ? 1 : 0;
	return (ans & 1) ? 1 : 0;
}

double get_in_len(gbs s)
{
	int tot = 0;
	t[tot++] = s.a, t[tot++] = s.b;
	if (gb.check(s, gbs(a[n - 1], a[0])) == 1)
		t[tot++] = gb.cross(s, gbs(a[n - 1], a[0]));

	if (gb.init(s, a[0])) t[tot++] = a[0];
	for (int i = 1; i < n; ++i)
	{
		if (gb.init(s, a[i])) t[tot++] = a[i];
		if (gb.check(s, gbs(a[i - 1], a[i])) == 1)
			t[tot++] = gb.cross(s, gbs(a[i - 1], a[i]));
	}
	sort(t, t + tot);
	tot = unique(t, t + tot) - t;
	double ans = 0;
	for (int i = 0; i < tot - 1; ++i)
		if (check(gb.get_mid(gbs(t[i], t[i + 1]))))
			ans += gb.dist(t[i], t[i + 1]);
	return ans;
}

vector<gbp> get_convex_hull()
```

```
	{
		vector<gbp> ans;
		sort(a, a + n);
		ans.push_back(a[0]);
		for (int i = 1; i < n; ++i)
		{
			while (ans.size() > 1 && gb.pos(ans.back() - ans[ans.size() - 2],
a[i] - ans[ans.size() - 2]) >= 0)
				ans.pop_back();
			ans.push_back(a[i]);
		}
		int p = ans.size();
		for (int i = n - 2; i >= 0; --i)
		{
			while (ans.size() > p + 1 && gb.pos(ans.back() - ans[ans.size() -
2], a[i] - ans[ans.size() - 2]) >= 0)
				ans.pop_back();
			ans.push_back(a[i]);
		}
		ans.pop_back();
		return ans;
	}
};

//============================Geometry_Polygon==========
```

21. Geometry_Round // 几何_圆

```
//============================Geometry_Round============

struct Geometry_Round
{
	gbp o;
	double r;
	Geometry_Base gb;

	void create(gbp _o, double _r)
	{
		o = _o;
		r = _r;
	};

	int check(gbp p) //里1 边界-1 外0
	{
		int k = gb.cmp(gb.dist(p, o), r);
		if (k == -1) return 1;
		else
		if (k == 0) return -1;
		else return 0;
	}

	int check2(gbs s)
	{
		vector<gbp> v;
		v = cross(s);
		int ans = 0;
		for (int i = 0; i < (int)v.size(); ++i)
			if (gb.init(s, v[i]))
				++ans;
```

```
        return ans;
    }

    int check(gbs s)
    {
        int k = check(gb.get_foot(s, o));
        if (k == -1) return 1;
        else
        if (k == 0) return 0;
        else return 2;
    }

    vector<gbp> cross(gbs s)
    {
        vector<gbp> v;
        int c = check(s);
        if (c == 0) return v;
        if (c == 1)
        {
            v.push_back(gb.get_foot(s, o));
            return v;
        }
        s.a -= o, s.b -= o;
        if (!gb.cmp(s.a.x, s.b.x))
        {
            v.push_back(gbp(s.a.x, sqrt(r * r - s.a.x * s.a.x)) + o);
            v.push_back(gbp(s.a.x, -sqrt(r * r - s.a.x * s.a.x)) + o);
            return v;
        }
        else
        {
            double k = gb.get_k(s), b = gb.get_b(s),
                                oo = 1 + k * k, p = 2 * k * b, q = b * b - r
* r,
                                delta = sqrt(p * p - 4 * oo * q);
            double x = (-p + delta) / 2 * oo, y = k * x + b;
            v.push_back(gbp(x, y) + o);
            x = (-p - delta) / 2 * oo, y = k * x + b;
            v.push_back(gbp(x, y) + o);
            return v;
        }
        return v;
    }
};

//============================Geometry_Round============
```

22. Shortest_Path // 最短路

```
//============================Shortest_Path=============

const int SPmaxn = 50000;

struct Shortest_Path
{
    struct member
    {
        int n, v;
        member(int a, int b) : n(a), v(b){}
```

```
	};

	int dist[SPmaxn], b[SPmaxn];
	priority_queue<member> q;

	int query(Link_Table_V &l, int s, int t)
	{
		memset(dist, 0x3f, sizeof(dist));
		memset(b, 0, sizeof(b));
		dist[s] = 0;
		while (!q.empty()) q.pop();
		q.push(member(s, 0));
		for (;;)
		{
			int k = -1;
			while (!q.empty())
			{
				k = q.top().n;
				if (!b[k]) break;
				k = -1;
				q.pop();
			}
			if (k == -1) break;
			if (k == t) return dist[t];
			b[k] = 1;
			for (int i = l.be(k); i; i = l.next(i))
			{
				int to = l.to(i), v = l.v(i);
				if (b[to]) continue;
				if (dist[k] + v >= dist[to]) continue;
				dist[to] = dist[k] + v;
				q.push(member(to, dist[to]));
			}
		}
		return -1;
	}
};

bool operator<(Shortest_Path::member a, Shortest_Path::member b) { return a.v > b.v; }

//============================Shortest_Path=============
```

23. High_Num // 高精

```
//============================HighNum===================

struct High_Num
{
	static const int HNmaxn = 30;
	static const int cut = 10000;
	static const int cut_n = 4;

	bool nag;
	int a[HNmaxn], n;

	inline void create(int t)
	{
		n = 0;
		memset(a, 0, sizeof(a));
```

```
	if (t < 0) nag = true, t = -t;
	else nag = false;
	while (t)
	{
		a[n++] = t % cut;
		t /= cut;
	}
}

inline void create(string t)
{
	int fr = 0, to = t.length() - 1;
	if (t[0] == '-') nag = true, ++fr;
	for (n = 0; to >= fr; ++n, to -= cut_n)
		for (int i = max(fr, to - cut_n + 1); i <= to; ++i)
			a[n] = a[n] * 10 + t[i] - '0';
}

inline High_Num(int t = 0)
{
	create(t);
}

inline bool bigger(const High_Num& a, const High_Num& b)
{
	if (a.n != b.n) return a.n > b.n;
	for (int i = a.n - 1; i >= 0; --i)
	{
		if (a.a[i] < b.a[i])
			return false;
		if (a.a[i] > b.a[i])
			return true;
	}
	return false;
}

inline bool smaller(const High_Num& a, const High_Num& b)
{
	if (a.n != b.n) return a.n < b.n;
	for (int i = a.n - 1; i >= 0; --i)
	{
		if (a.a[i] < b.a[i])
			return true;
		if (a.a[i] > b.a[i])
			return false;
	}
	return false;
}

inline bool operator==(const High_Num& b)
{
	if (n != b.n || nag != b.nag) return false;
	for (int i = 0; i < n; ++i)
		if (a[i] != b.a[i])
			return false;
	return true;
}
```

```
	inline bool operator!=(const High_Num& b)
	{
		return !((*this) == b);
	}

	inline bool operator<(const High_Num& b) { return (nag != b.nag) ? nag : (nag ?
bigger(*this, b) : smaller(*this, b)); }
	inline bool operator>(const High_Num& b) { return (nag != b.nag) ? b.nag :
(nag ? smaller(*this, b) : bigger(*this, b)); }
	inline bool operator<=(const High_Num& b) { return *this < b || *this == b; }
	inline bool operator>=(const High_Num& b) { return *this > b || *this == b; }

	inline void get_mid(const High_Num& a, High_Num& ans)
	{
		ans.clear();
		for (int i = a.n - 1, t = 0; i >= 0; --i)
		{
			ans.a[i] = (a.a[i] + t) / 2;
			t = (a.a[i] % 2) * cut;
		}
		ans.n = a.n;
		while (ans.n && !ans.a[ans.n - 1]) --ans.n;
	}

	inline void plus(const High_Num& a, const High_Num& b, High_Num& ans)
	{
		ans.clear();
		ans.n = max(a.n, b.n);
		for (int i = 0; i < ans.n; ++i)
		{
			ans.a[i] += a.a[i] + b.a[i];
			ans.a[i + 1] = ans.a[i] / cut;
			ans.a[i] %= cut;
		}
		if (ans.a[ans.n]) ++ans.n;
	}

	inline void minus(const High_Num& a, const High_Num& b, High_Num& ans)
	{
		if (smaller(a, b))
		{
			minus(b, a, ans);
			ans.nag = true;
			return;
		}
		ans.clear();
		ans.n = a.n;
		for (int i = 0; i < ans.n; ++i)
		{
			ans.a[i] += a.a[i] - b.a[i];
			if (ans.a[i] < 0)
				ans.a[i + 1] -= 1, ans.a[i] += cut;
		}
		while (ans.n && !ans.a[ans.n - 1]) --ans.n;
	}

	inline void multiply(const High_Num& a, const High_Num& b, High_Num& ans)
	{
```

```
	ans.clear();
	for (int i = 0; i < a.n; ++i)
		for (int j = 0; j < b.n; ++j)
		{
			ans.a[i + j] += a.a[i] * b.a[j];
			ans.a[i + j + 1] += ans.a[i + j] / cut;
			ans.a[i + j] %= cut;
		}
	ans.n = a.n + b.n + 1;
	while (ans.n && !ans.a[ans.n - 1]) --ans.n;
}

inline void divide(const High_Num& a, const High_Num& b, High_Num& ans)
{
	ans.clear();
	High_Num l = 0, r = 0, mid;
	r = a;
	r = r + 1;
	while (l + 1 < r)
	{
		get_mid(l + r, mid);
		if (mid * b > a) r = mid;
		else l = mid;
	}
	ans = l;
}

inline void divide(const High_Num& a, int b, High_Num& ans)
{
	ans.clear();
	for (int i = a.n - 1, last = 0; i >= 0; --i)
		ans.a[i] = (a.a[i] + last) / b,
		last = ((a.a[i] + last) % b) * a.cut;
	ans.n = a.n;
	while (ans.n && !ans.a[ans.n - 1]) --ans.n;
}

inline High_Num operator+(const High_Num& t)
{
	High_Num ans;
	if (nag == t.nag)
	{
		plus(*this, t, ans);
		if (nag)
			ans.nag = !ans.nag;
	}
	else
	{
		if (nag) minus(t, *this, ans);
		else minus(*this, t, ans);
	}
	return ans;
}

inline High_Num operator-(const High_Num& t)
{
	High_Num ans;
	if (nag != t.nag)
```

```
		{
			plus(*this, t, ans);
			if (nag)
				ans.nag = !ans.nag;
		}
		else
		{
			if (nag) minus(t, *this, ans);
			else minus(*this, t, ans);
		}
		return ans;
	}

	inline High_Num operator*(const High_Num& t)
	{
		High_Num ans;
		multiply(*this, t, ans);
		if (nag != t.nag && ans.n) ans.nag = !ans.nag;
		return ans;
	}

	inline High_Num operator/(const High_Num& t)
	{
		if (!n || !t.n) return High_Num(0);
		High_Num ans;
		divide(*this, t, ans);
		if (nag != t.nag)   ans.nag = !ans.nag;
		return ans;
	}

	inline High_Num operator/(int t)
	{
		if (!n || !t) return High_Num(0);
		High_Num ans;
		divide(*this, t, ans);
		if (nag != (t < 0)) ans.nag = !ans.nag;
		return ans;
	}

	inline High_Num operator%(const High_Num& t)
	{
		return (*this) - (((*this) / t) * t);
	}

	inline High_Num operator%(int t)
	{
		return (*this) - (((*this) / t) * t);
	}

	inline High_Num operator+=(const High_Num& t) { (*this) = (*this) + t; return
*this;}
	inline High_Num operator-=(const High_Num& t) { (*this) = (*this) - t; return
*this;}
	inline High_Num operator*=(const High_Num& t) { (*this) = (*this) * t; return
*this;}
	inline High_Num operator/=(const High_Num& t) { (*this) = (*this) / t; return
*this;}
```

```
	inline High_Num operator%=(const High_Num& t) { (*this) = (*this) % t; return
*this;}

	inline bool operator==(const int t){	High_Num t2 = t; return (*this) == t2; }
	inline bool operator!=(const int t){	High_Num t2 = t; return (*this) != t2; }
	inline bool operator<(const int t) {	High_Num t2 = t; return (*this) < t2; }
	inline bool operator<=(const int t){	High_Num t2 = t; return (*this) <= t2; }
	inline bool operator>(const int t)	{	High_Num t2 = t; return (*this) >
t2; }
	inline bool operator>=(const int t){	High_Num t2 = t; return (*this) >= t2; }

	inline High_Num operator+(const int t) {	High_Num t2 = t; return (*this) +
t2; }
	inline High_Num operator-(const int t) {	High_Num t2 = t; return (*this) -
t2; }
	inline High_Num operator*(const int t)  {	High_Num t2 = t; return (*this) *
t2; }

	inline High_Num operator+=(const int t) {	High_Num t2 = t; (*this) =
(*this) + t2; return *this;}
	inline High_Num operator-=(const int t) {	High_Num t2 = t; (*this) =
(*this) - t2; return *this;}
	inline High_Num operator*=(const int t) {	High_Num t2 = t; (*this) =
(*this) * t2; return *this;}
	inline High_Num operator/=(int t) {	(*this) = (*this) / t; return *this;}
	inline High_Num operator%=(int t) {	(*this) = (*this) % t; return *this;}

	inline void print()
	{
		if (!n)
		{
			printf("0");
			return;
		}
		if (nag) printf("-");
		printf("%d", a[n - 1]);
		for (int i = n - 2; i >= 0; --i)
			printf("%.04d", a[i]);
	}

	inline void clear()
	{
		n = 0;
		nag = false;
		memset(a, 0, sizeof(a));
	}

	inline int to_int()
	{
		int k = 0;
		for (int i = 0; i < n; ++i) k = (k * cut) + a[i];
		if (nag) k *= -1;
		return k;
	}
};

//============================HighNum====================
```

24. Discretization // **离散化**

```
//============================Discretization============

template<typename T>
struct Discretization
{
    static const int Dmaxn = 1005;
    T t[Dmaxn];
    int sau(T *a, int n)
    {
        sort(a, a + n, less<T>());
        return unique(a, a + n, equal_to<T>()) - a;
    }
    void query(T *a, int n, int *ans)
    {
        for (int i = 0; i < n; ++i)
            t[i] = a[i];
        sort(t, t + n, less<T>());
        int m = unique(t, t + n, equal_to<T>()) - t;
        for (int i = 0; i < n; ++i)
            ans[i] = lower_bound(t, t + m, a[i], less<T>()) - t;
    }
    template<typename _compare, typename __compare>
    int sau(T *a, int n, _compare _less, __compare equal)
    {
        sort(a, a + n, _less);
        return unique(a, a + n, equal) - a;
    }
    template<typename _compare, typename __compare>
    void query(T *a, int n, int *ans, _compare _less = less<T>(), __compare equal =
equal_to<T>())
    {
        for (int i = 0; i < n; ++i)
            t[i] = a[i];
        sort(t, t + n, _less);
        int m = unique(t, t + n, equal) - t;
        for (int i = 0; i < n; ++i)
            ans[i] = lower_bound(t, t + m, a[i], _less) - t;
    }
};

//============================Discretization============
```

25. Tarjan // tarjan

```
//============================Tarjan====================

struct Tarjan
{
       static const int Tmaxn = 200005;
       static const int Tmaxm = 200005;
       int low[Tmaxn], dfn[Tmaxn], tot, t[Tmaxm];

       int tarjan(int now, int fa, Link_Table &L, bool *b, int tp)
       {
              dfn[now] = ++tot;
              low[now] = dfn[now];
              int sum = 0, lfa = -1, mi = 0x7fffffff;
```

```
	for (int i = L.be(now); i; i = L.next(i))
	{
		int to = L.to(i);
		if (to == fa)
		{
			lfa = i;
			continue;
		}
		if (!dfn[to])
		{
			int t = tarjan(to, now, L, b, tp);
			low[now] = min(low[now], low[to]);
			if (tp)
				if (low[to] > dfn[now]) b[i] = b[t] = 1;
			++sum;
			if (low[to] >= dfn[now] && fa != -1) b[now] = 1;
			mi = min(mi, low[to]);
		}
		else low[now] = min(low[now], dfn[to]);
	}
	if (!tp && sum && fa == -1)
		b[now] = sum > 1 ? 1 : 0;
	return lfa;
}

void clear()
{
	memset(low, 0, sizeof(low));
	memset(dfn, 0, sizeof(dfn));
	tot = 0;
}

void get_cut_node(Link_Table &L, int n, bool *b)
{
	clear();
	for (int i = 0; i < n; ++i)
		b[i] = 0;
	for (int i = 0; i < n; ++i)
		if (!dfn[i])
			tarjan(i, -1, L, b, 0);
}

void get_cut_edge(Link_Table &L, int n, bool *b)
{
	clear();
	for (int i = 0; i < n; ++i)
		b[i] = 0;
	for (int i = 0; i < n; ++i)
		if (!dfn[i])
			tarjan(i, -1, L, b, 1);
	for (int i = 0; i < n; ++i)
	{
		for (int j = L.be(i); j; j = L.next(j))
			++t[L.to(j)];
		for (int j = L.be(i); j; j = L.next(j))
			if (t[L.to(j)] > 1)
				b[j] = 0;
		for (int j = L.be(i); j; j = L.next(j))
```

```
                        --t[L.to(j)];
            }
        }
};

//============================Tarjan====================
```

26. Other // par point range

```
//============================other====================

struct par
{
  int a, b;
  par(int _a = 0, int _b = 0) : a(_a), b(_b) {}
};

struct point
{
  int x, y;
  point(int _x = 0, int _y = 0) : x(_x), y(_y) {}
};

struct range
{
  int l, r;
  range(int _l = 0, int _r = 0) : l(_l), r(_r) {}
};

//============================other====================
```