1. 数据结构

1.1. Splay 树

```
#define keyTree (ch[ ch[root][1] ][0])

struct SplayTree{

int sz[maxn], n;
int ch[maxn][2];
int pre[maxn];
int root , top1 , top2;
int ss[maxn] , que[maxn];
bool rev[maxn];

inline void Rotate(int x,int f) {
  int y = pre[x];
  push_down(y);
  push_down(x);
  ch[y][!f] = ch[x][f];
  pre[ ch[x][f] ] = y;
  pre[x] = pre[y];
  if(pre[x]) ch[ pre[y] ][ ch[pre[y]][1] == y ] = x;
  ch[x][f] = y;
  pre[y] = x;
  push_up(y);
}

inline void Splay(int x,int goal) {
  push_down(x);
  while(pre[x] != goal) {
    if(pre[pre[x]] == goal) {
      Rotate(x , ch[pre[x]][0] == x);
    } else {
      int y = pre[x] , z = pre[y];
      int f = (ch[z][0] == y);
      if(ch[y][f] == x) {
        Rotate(x , !f) , Rotate(x , f);
      } else {
        Rotate(y , f) , Rotate(x , f);
      }
    }
  }
  push_up(x);
  if(goal == 0) root = x;
}

//k == 0,1,2...
inline void RotateTo(int k,int goal) {//  k 位的数转到 goal 下边
  int x = root;
  push_down(x);
  while(sz[ ch[x][0] ] != k) {
    if(k < sz[ ch[x][0] ]) {
      x = ch[x][0];
    } else {
      k -= (sz[ ch[x][0] ] + 1);
      x = ch[x][1];
    }
    push_down(x);
  }
  Splay(x,goal);
}

/* add 完后要设置 pre! */
inline void NewNode(int &x, int c) {
  x = ++top1;
  ch[x][0] = ch[x][1] = pre[x] = rev[x] = 0;
  sz[x] = 1;

  val[x] = sum[x] = c;
  left[x] = right[x] = maxi[x] = c;
  rev[x] = same[x] = false;
}

inline void reverse(int x)
{
    swap(ch[x][0], ch[x][1]);
    swap(left[x], right[x]);
    rev[x] = !rev[x];
}

inline void push_down(int x) {
  if(rev[x]) {
      if (ch[x][0]) reverse(ch[x][0]);
      if (ch[x][1]) reverse(ch[x][1]);
        rev[x] = false;
  }

}

inline void push_up(int x) {
}

inline void makeTree(int &x,int l,int r,int f) {
  if(l > r) return ;
  int m = (l + r)>>1;
  NewNode(x, num[m]);

  makeTree(ch[x][0] , l , m - 1 , x);
  makeTree(ch[x][1] , m + 1 , r , x);
  pre[x] = f;
  push_up(x);
}

inline void init(int n) {

  root = top1 = top2 = 0;
  NewNode(root , -0x3f3f3f3f);
  NewNode(ch[root][1] , -0x3f3f3f3f);
  pre[top1] = root;
  sz[root] = 2;

  makeTree(keyTree , 0 , n-1 , ch[root][1]);
  push_up(ch[root][1]);
  push_up(root);
}
```

```
}
```

1.2. 轻重边树链剖分

```
struct HeavyLightDecomp {

int roots, total;
vector<pair<int,int> > e[maxn];
int father[maxn], sz[maxn], child[maxn],
    pre[maxn], depth[maxn], pos[maxn], cost[maxn];
SegTree st;

void init() {
    for (int i = 0;i < n;++i) e[i].clear();
}

void buildTree(int u, int fa) {
    father[u] = fa;
    sz[u] = 1;
    child[u] = -1;
    int nowmax = 0;
    vector<pair<int,int> > &v = e[u];
    for (int i = 0; i < v.size(); ++i)
        if (v[i].first != fa) {
            cost[v[i].first] = v[i].second;
            buildTree(v[i].first, u);
            if (sz[v[i].first] > nowmax) {
                nowmax = sz[v[i].first];
                child[u] = v[i].first;
            }
            sz[u] += sz[v[i].first];
        }
}

void go(int u, int fa, int last, int dep, int value) {
    if (last == -1)
        last = u;
    pre[u] = last;
    st.val[total] = value;
    pos[u] = total++;
    depth[u] = dep;
    if (child[u] != -1)
        go(child[u], u, last, dep, cost[child[u]]);

    vector<pair<int,int> > &v = e[u];
    for (int i = 0; i < v.size(); ++i)
        if (v[i].first != fa && v[i].first != child[u])
            go(v[i].first, u, -1, dep+1, v[i].second);
}

void buildHeavyLightDecomp() {
    total = 0;
    buildTree(0,-1);
    go(0,-1,-1,0,maxint);
    st.build(0,n-1,1);
}

void modify(int u, int v, int cost) {
    if (father[u] == v)
        swap(u,v);
    st.modify(0,n-1,pos[v],1,cost);
}

int getMax(int u, int v) {
    if (depth[u] > depth[v])
        swap(u, v);
    int ret = -maxint;
    while (depth[v] > depth[u]) {
        ret = max(ret, st.getMax(0,n-1,pos[pre[v]],pos[v],1));
        v = father[pre[v]];
    }
    while (pre[u] != pre[v]) {
        ret = max(ret, st.getMax(0,n-1,pos[pre[v]],pos[v],1));
        ret = max(ret, st.getMax(0,n-1,pos[pre[u]],pos[u],1));
        v = father[pre[v]];
        u = father[pre[u]];
    }
    int l = min(pos[u],pos[v])+1, r = max(pos[u],pos[v]);
    if (l <= r) ret = max(ret, st.getMax(0,n-1,l,r,1));
    return ret;
}

};
```

1.3. RMQST

```
void construct() {
    int power = 1;
    while ((1 << power) < size) {
        ++power;
    }
    maxPw = power + 1;
    min = new int[maxPw][size];
    log = new int[size + 1];
    log[0] = 0;
    for (int i = 1; i <= size; ++i) {
        log[i] = 0;
        while ((1 << (log[i] + 1)) < i) {
            ++log[i];
        }
    }

    for (int i = 0; i < size; ++i) {
        min[0][i] = orig[i];
    }

    for (int i = 1; i < maxPw; ++i) {
        for (int j = 0; j < size; ++j) {
            min[i][j] = min[i - 1][j];
            if (j + (1 << (i - 1)) < size) {
                min[i][j] = Math.min(min[i][j], min[i - 1][j + (1 << (
                    i - 1))]);
            }
        }
    }
}
```

```
@Override
public int getMinValue(int left, int right) {
    if (right < left) {
        return Integer.MAX_VALUE;
    }
    int step = log[right - left + 1];
    return Math.min(min[step][left], min[step][right - (1 << step) +
        1]);
}
```

1.4. KD 树

```
int n, sortKey;

struct NODE {
    int ID;
    long long x[maxk], dis;
    bool operator < (const NODE &a) const {
        return x[sortKey] < a.x[sortKey];
    }
} tree[maxn], op, res, ans[maxn], Beg[maxn];

struct KDTree {

int Dim;
bool use[maxn];
int split[maxn];
void clear() { memset(use, false, sizeof(n+3)); }
template<class Tp> Tp sqr(Tp x) { return x*x; }
void build(int L, int R, int cut) {
    if (L > R) return;
    int mid = (L+R)/2;
    split[mid] = cut;
    sortKey = split[mid];
    nth_element(tree+L, tree+mid, tree+R+1);
    build(L, mid-1, (cut+1)%Dim);
    build(mid+1, R, (cut+1)%Dim);
}
void query(int L, int R) {
    if (L > R) return;
    int mid = (L+R)/2;
    long long dis = 0;
    for (int i = 0; i < Dim; i++) dis += sqr(op.x[i]-tree[mid].x[i]);
    if (!use[tree[mid].ID] && dis < op.dis) {
        op.dis = dis;
        res = tree[mid];
    }
    long long Rad = sqr(op.x[split[mid]]-tree[mid].x[split[mid]]);
    if (op.x[split[mid]] < tree[mid].x[split[mid]]) {
        query(L, mid-1);
        if (Rad <= op.dis) query(mid+1, R);
    }
    else {
        query(mid+1, R);
        if (Rad <= op.dis) query(L, mid-1);
    }
}

} KD;
```

1.5. 笛卡尔树线性构造

```
void build() {
    last = 1;
    sort(keys.begin(), keys.end());
    for (int i = 0; i < n; ++i) {
        while (S.size() && value[S.top()] > keys[i].y)
            S.pop();
        int x = newNode(keys[i].x, keys[i].y);
        father[x] = S.top();
        ch[x][0] = ch[father[x]][1];
        ch[father[x]][1] = x;
        if (ch[x][0]) father[ch[x][0]] = x;
        S.push(x);

        id[x] = keys[i].z;
    }
}
```

1.6. 函数式线段树

```
struct Node {
    int l, r, cl, cr, val;
    Node(int tl, int tr, int tcl, int tcr, int tval = 0) :
        l(tl), r(tr), cl(tcl), cr(tcr), val(tval){}
    Node() {}
} tree[MAXN];

int last;

int getNode() { return last++;}

int makeNode(int left, int right, int cl, int cr, int x) {
    tree[x] = Node(left, right, cl, cr);
    tree[x].val = tree[cl].val + tree[cr].val;
    return x;
}

int build(int left, int right, int val, int x) {
    tree[x].l = left;
    tree[x].r = right;
    if (left == right) {
        tree[x].val = val;
        return x;
    }

    int mid = (left + right) >> 1;
    tree[x].cl = build(left, mid, val, getNode());
    tree[x].cr = build(mid+1, right, val, getNode());
    tree[x].val = tree[tree[x].cl].val + tree[tree[x].cr].val;

    return x;
}

int add(int pos, int val, int x) {
    if (tree[x].l == tree[x].r)
```

```
        return build(tree[x].l, tree[x].r, tree[x].val + val, getNode
            ());
    int mid = tree[tree[x].cl].r;
    if (pos <= mid)
        return makeNode(tree[x].l, tree[x].r, add(pos, val, tree[x].cl
            ), tree[x].cr, getNode());
    return makeNode(tree[x].l, tree[x].r, tree[x].cl, add(pos, val,
        tree[x].cr), getNode());
}

build(0, total, 0, getNode());
```

2. 字符串

2.1. AC 自动机

```
struct node {
    int next[26];
    int suf,count;
    bool dan;
} tree[maxn];

inline int tran(char s){
    return s - 'a';
}

void insert(char *s,int len) {
    int now = 1;
    for (int i = 0;i < len;++i)
        if (tree[now].next[tran(s[i])])
            now = tree[now].next[tran(s[i])];
        else {
            tree[now].next[tran(s[i])] = last;
            now = last++;
            memset(&tree[now],0,sizeof(node));
        }
    ++tree[now].count;
    tree[now].dan = true;
}

void buildTrie() {
    int front = 0,rear = 0;
    queue[rear++] = 1;
    while (front < rear) {
        int now = queue[front++];
        tree[now].dan = tree[now].dan || tree[tree[now].suf].dan;
        for (int i = 0;i < 26;++i)
            if (tree[now].next[i]) {
                queue[rear++] = tree[now].next[i];
                tree[tree[now].next[i]].suf = tree[tree[now].suf].next
                    [i];
            }
            else
                tree[now].next[i] = tree[tree[now].suf].next[i];
    }
}

memset(&tree[0],0,sizeof(node));
memset(&tree[1],0,sizeof(node));
for (int i = 0;i < 26;++i)
    tree[0].next[i] = 1;
last = 2;
```

2.2. 后缀数组

```
int sa[MAXN],height[MAXN],rank[MAXN];
int wa[MAXN],wb[MAXN],wc[MAXN],wd[MAXN];

bool cmp(int *r,int a,int b,int l,int n) {
    int la = r[a],lb = r[b],ra,rb;
    ra = a+l < n? r[a+l] : -1;
    rb = b+l < n? r[b+l] : -1;
    return (la == lb) && (ra == rb);
}

void makesa(char *r,int *sa,int n,int m) {
    int *x = wa,*y = wb;
    for (int i = 0;i < m;++i) wc[i] = 0;
    for (int i = 0;i < n;++i) ++wc[x[i] = r[i]];
    for (int i = 1;i < m;++i) wc[i] += wc[i-1];
    for (int i = n-1;i >= 0;--i) sa[--wc[x[i]]] = i;
    for (int tot = 0,p = 1;tot+1 < n;p <<= 1,m = tot+1) {
        tot = 0;
        for (int i = n-p;i < n;++i) y[tot++] = i;
        for (int i = 0;i < n;++i) if (sa[i] >= p) y[tot++] = sa[i] - p
            ;
        for (int i = 0;i < n;++i) wd[i] = x[y[i]];
        for (int i = 0;i < m;++i) wc[i] = 0;
        for (int i = 0;i < n;++i) ++wc[wd[i]];
        for (int i = 1;i < m;++i) wc[i] += wc[i-1];
        for (int i = n-1;i >= 0;--i) sa[--wc[wd[i]]] = y[i];
        int *t = x; x = y; y = t;
        x[sa[0]] = tot = 0;
        for (int i = 1;i < n;++i)
            x[sa[i]] = cmp(y,sa[i-1],sa[i],p,n) ? tot : ++tot;
    }
}

void makeheight(char *r,int *sa,int *height,int n)
{
    for (int i = 0;i < n;++i) rank[sa[i]] = i;
    height[0] = 0;
    for (int i = 0;i < n;++i) {
        if (!rank[i]) continue;
        if (!i) height[rank[i]] = 0;
        else height[rank[i]] = height[rank[i-1]] - 1;
        if (height[rank[i]] < 0) height[rank[i]] = 0;
        for (;r[i+height[rank[i]]] == r[sa[rank[i]-1] + height[rank[i
            ]]];++height[rank[i]]);
    }
}

makesa(data,sa,n,200);
makeheight(data,sa,height,n);
```

2.3. 树型后缀数组

```
// 0 .. alphaSize - 1
```

```
void makesa(int N, int alphaSize) {
    for (int i = 1; i <= N; ++i) fa[i][0] = father[i];
    for (int j = 1; j < MAXD; ++j) {
        for (int i = 1; i <= N; ++i) {
            fa[i][j] = fa[fa[i][j - 1]][j - 1];
        }
    }
    for (int i = 1; i <= N; ++i) rank[i][0] = ch[i] + 1;
    //alphaSize
    static int total[MAXN];
    static int order[MAXN];
    static int stepsa[MAXN];
    for (int j = 1; j < MAXD; ++j) {
        for (int i = 0; i <= alphaSize; ++i) total[i] = 0;
        for (int i = 1; i <= N; ++i) ++total[rank[fa[i][j - 1]][j -
            1]];
        for (int i = 1; i <= alphaSize; ++i) total[i] += total[i - 1];
        for (int i = 1; i <= N; ++i) order[total[rank[fa[i][j - 1]][j
            - 1]]--] = i;
        for (int i = 0; i <= alphaSize; ++i) total[i] = 0;
        for (int i = 1; i <= N; ++i) ++total[rank[i][j - 1]];
        for (int i = 1; i <= alphaSize; ++i) total[i] += total[i - 1];
        for (int i = N; i >= 1; --i) stepsa[total[rank[order[i]][j -
            1]]--] = order[i];
        int total = 0;
        for (int i = 1; i <= N; ++i) {
            if (i == 1 || rank[stepsa[i]][j - 1] != rank[stepsa[i -
                1]][j - 1]
                || rank[fa[stepsa[i]][j - 1]][j - 1] != rank[fa[stepsa
                    [i - 1]][j - 1]][j - 1]) {
                ++total;
            }
            rank[stepsa[i]][j] = total;
        }
        alphaSize = total;
    }
}

int lcp(int x, int y) {
    int ret = 0;
    for (int k = MAXD - 1; k >= 0; --k) {
        if (fa[x][k] != 0 && fa[y][k] != 0 && rank[x][k] == rank[y][k
            ]) {
            ret += 1 << k;
            x = fa[x][k];
            y = fa[y][k];
        }
    }
    if (x > 0 && y > 0 && ch[x] == ch[y]) ++ret;
    return ret;
}
```

2.4. 后缀自动机

空间至少要开 2-3 倍以上!
Max(Parent(s)) = Min(s) -1。
Right(str) 表示 str 在母串 S 中所有出现的结束位置集合。

```
/*
Suffix Automaton
如果有多个串要按前缀树的 bfs 序列构造
*/

struct SAM
{
    struct Node
    {
        int suf, ch[26];
        int maxi, dan;
        Node() : suf(-1), maxi(0), dan(0)
        {
            memset(ch, -1, sizeof(ch));
        }
    };

    SAM()
    {
        init();
    }

    void init()
    {
        total = 1;
        tree[0] = Node();
        root = last = 0;
    }

    int extend(int w, int rear = -1)
    {
        int p = (rear == -1 ? last : rear);
        int np = total++;
        tree[np].maxi = tree[p].maxi + 1;
        tree[np].dan = 1;
        while (-1 != p && -1 == tree[p].ch[w])
            tree[p].ch[w] = np, p = tree[p].suf;
        if (-1 == p)
            tree[np].suf = root;
        else
        {
            int q = tree[p].ch[w];
            if (tree[p].maxi+1 == tree[q].maxi)
                tree[np].suf = q;
            else
            {
                int nq = total++;
                memcpy(tree[nq].ch, tree[q].ch, sizeof(tree[q].ch));
                tree[nq].maxi = tree[p].maxi + 1;
                tree[nq].suf = tree[q].suf;
                tree[q].suf = nq;
                tree[np].suf = nq;
                while (p != -1 && tree[p].ch[w] == q)
                    tree[p].ch[w] = nq, p = tree[p].suf;
            }
        }
        last = np;
        return last;
    }
    int root, last;
    Node tree[MAXN];
```

```
    int total;

    //extended
    int sz[MAXN], in[MAXN];
    queue<int> Q;
    void calcSZ()
    {
        while (Q.size()) Q.pop();
        REP(i, total) in[i] = 0;
        REP(i, total) if (tree[i].suf != -1)
            in[tree[i].suf]++;
        REP(i, total) sz[i] = tree[i].dan;
        REP(i, total) if (!in[i])
            Q.push(i);
        while (Q.size())
        {
            int now = Q.front();
            Q.pop();
            if (-1 != tree[now].suf)
            {
                sz[tree[now].suf] += sz[now];
                --in[tree[now].suf];
                if (!in[tree[now].suf])
                    Q.push(tree[now].suf);
            }
        }

    }
};
```

2.5. Z 算法

```
//next数组指串 T 的后缀与自身的最长公共前缀
//lcp指 S 的后缀与 T 的最长公共前缀，既 KMP 的加强版。
//扩展 KMP 可以算出 S 以 i 结尾的串能和子串匹配的最长长度。
//这里的 next 指函数 next[i] = |T 与 T(i,m) 的最长公共前缀 |，这里2<i<m。
void next_function(char S[], int n, int next[]){
    next[0] = n;
    int j = 0, k = 1;
    while(1+j<n && S[j] == S[1+j]) j++;
    next[1] = j;
    for(int i=2; i<n; ++i) {
        int len = k + next[k] - 1;
        int l = next[i - k];
        if(l < len - i + 1) next[i] = l;
        else {
            int j = max(0, len - i + 1);
            while(S[i+j] == S[j] && i + j < n) j++;
            next[i] = j;
            k = i;
        }
    }
}
void extend_kmp(char S[], int n, char T[], int m, int next[], int lcp
    []) {
    next_function(T, m, next);
    int j = 0, k = 0;
    while(j<min(n, m) && T[j] == S[j]) ++j;
    lcp[0] = j;
    for(int i=1; i<n; ++i) {
        int len = k + lcp[k] - 1;
        int l = next[i - k];
        if(l < len - i + 1) lcp[i] = l;
        else {
            int j = max(0, len - i + 1);
            while(i+j<n && j<m && S[i+j]==T[j]) j++;
            lcp[i] = j;
            k = i;
        }
    }
}
```

2.6. Manecher 算法

```
//str 为插入字符后的串，在串首加入个没有出现的字符如‘¥’，防止比较的时候指
    针小于 0
//所以实际 str 是从下标 1 开始的
//aba = #a#b#a#
//aa = #a#a#
int p[maxn];
char str[maxn];
int manacher(char *s, int len) {
    str[0] = '$';
    for (int i = 0;i < len;++i) {
        str[i*2+1] = '#'; str[i*2+2] = s[i];
    }
    str[len*2+1] = '#'; str[len*2+2] = 0;
    int n = len * 2 + 1;
    int i,mx=0,id;
    for(i=1;i<=n;i++) {
        if(mx>i) p[i]=min(p[2*id-i],mx-i);
        else p[i]=1;
        for(;str[i+p[i]]==str[i-p[i]];p[i]++);
        if(p[i]+i>mx) {
            mx=p[i]+i; id=i;
        }
    }
    int ret = 0;
    for (int i = 1;i <= n;++i) ret = max(ret, p[i] - 1);
    return ret;
}
```

2.7. 最小表示

```
int go(char a[],int len) {
  int i = 0,j = 1,k = 0;
  while (i < len && j < len && k < len) {
    int cmp = a[(j+k)%len]-a[(i+k)%len];
    if (cmp == 0) k++;
    else {
      if (cmp > 0) j += k+1;
      else i += k+1;
      if (i == j) j++;
      k = 0;
```

```
    }
  }
  return min(i,j);
}
```

3. 图论算法

3.1. Dinic 最大流

```
const int N = 100000*2;
const int M = 100000*2;

int n, m, x[N], y[N];

int edgeCount, firstEdge[N], to[M], capacity[M], nextEdge[M],
    currentEdge[N];
int source, target, flow, pre[N], sign;

void addEdge(int u, int v, int w) {
    to[edgeCount] = v;
    capacity[edgeCount] = w;
    nextEdge[edgeCount] = firstEdge[u];
    firstEdge[u] = edgeCount ++;
}

void insert(int u, int v, int w) {
    addEdge(u, v, w);
    addEdge(v, u, 0);
}

int level[N], queue[N];

bool bfs(int s, int t) {
    memset(level, -1, sizeof(level)); //fix
    sign=t;
    level[t] = 0;
    int tail = 0;
    queue[tail ++] = t;
    int head = 0;
    while (head != tail && level[s] == -1) {
        int v = queue[head ++];
        for (int iter = firstEdge[v]; iter != -1; iter = nextEdge[iter
            ]) {
            if (capacity[iter ^ 1] > 0 && level[to[iter]] == -1) {
                level[to[iter]] = level[v] + 1;
                queue[tail ++] = to[iter];
            }
        }
    }
    return level[s] != -1;
}

inline void push() {
    int delta=INT_MAX,u,p;
    for (u=target;u!=source;u=to[p]){
        p=pre[u];
        delta=min(delta,capacity[p]);
        p^=1;
    }

    for (u=target;u!=source;u=to[p]){
        p=pre[u];
        capacity[p]-=delta;
        if (!capacity[p]){
            sign=to[p^1];
        }
        capacity[p^=1]+=delta;
    }
    flow+=delta;
}

void dfs(int u) {
    if (u == target) {
        push();
        return;
    }
    for (int &iter = currentEdge[u]; iter != -1; iter = nextEdge[iter
        ]) {
        if (capacity[iter] > 0 && level[u] == level[to[iter]] + 1) {
            pre[to[iter]]=iter;
            dfs(to[iter]);
            if (level[sign]>level[u]) return;
            sign=target;
        }
    }
    level[u]=-1;
}

void initNetwork(int nodes) {
    n = nodes;
    edgeCount = 0;
    for (int i = 0; i <= n; ++i)
        firstEdge[i] = -1;
}

int maxFlow(int s, int t) {
    source = s;
    target = t;

    flow=0;
    while (bfs(source, target)) {
        for (int i = 0; i < n; ++ i) {
            currentEdge[i] = firstEdge[i];
        }
        dfs(source);
    }

    return flow;
}
```

3.2. 经典费用流

```
const int maxn=2000;
const int maxint = 1<<28;

void get_min(int &a,int b){ if(b<a) a=b; }
void get_max(int &a,int b){ if(b>a) a=b; }
```

```
int NEXT(int a,int b){ return a%b; }

struct Graph{

struct Adj{
    int v,f,c,w,b;
    Adj(int _v,int _c,int _w,int _b):v(_v),f(0),c(_c),w(_w),b(_b){}
}*st[maxn];
vector<Adj> adj[maxn];
int n;
void clear(){
    for(int i=0;i<n;i++){
        adj[i].clear();
    }
    n=0;
}

void insert(int u,int v,int c,int w,int d=0){
    get_max(n,max(u,v)+1);
    adj[u].push_back(Adj(v,c,w,adj[v].size()));
    adj[v].push_back(Adj(u,0,-w,adj[u].size()-1));
    if(d){
        adj[v].push_back(Adj(u,c,w,adj[u].size()));
        adj[u].push_back(Adj(v,0,-w,adj[v].size()-1));
    }
}
pair<int,int> mcmf(int S,int T){
    int d;
    int flow=0,cost=0;
    while((d=bell(S,T))){
        flow+=d;
        for(int v=T;v!=S;v=adj[st[v]->v][st[v]->b].v){
            cost+=st[v]->w*d;
            st[v]->f+=d;
            adj[st[v]->v][st[v]->b].f-=d;
        }
    }
    return make_pair(flow,cost);
}
int bell(int S,int T){
    int d[maxn],bfs[maxn],hash[maxn];
    fill(hash,hash+n,0);
    fill(d,d+n,maxint);
    hash[S]=1;d[S]=0;bfs[0]=S;
    for(int s=0,t=1;s!=t;hash[bfs[s]]=0,s=NEXT(s+1,n)){
        int v=bfs[s];
        for(vector<Adj>::iterator it=adj[v].begin();it!=adj[v].end();
            it++){
            if(it->f < it->c&&d[v]+it->w < d[it->v]){
                d[it->v]=d[v]+it->w;
                st[it->v]=&(*it);
                if(hash[it->v]==0){
                    hash[it->v]=1;
                    bfs[t]=it->v;
                    t=NEXT(t+1,n);
                }
            }
        }
    }
    if(d[T]==maxint){
        return 0;
    }
    int ans=maxint;
    for(int v=T;v!=S;v=adj[st[v]->v][st[v]->b].v){
        get_min(ans,st[v]->c - st[v]->f);
    }
    return ans;
}

}G;
```

3.3. 改进版费用流

```
#define NMax 2000
#define MMax 100000
#define OPT(_) (epool+(((_)-epool)^1))

struct edge{
    int e,c,f;
    edge *next;
}epool[MMax+MMax],*etop;
int N,ret;
edge *E[NMax];
int dist[NMax];
char vi[NMax];
int dfs(int a,int m){
    if (a==N-1)return m;
    vi[a]=1;
    int l=m;
    for (edge *p=E[a];p && l;p=p->next)if (p->f && !vi[p->e] && dist[p
        ->e]==dist[a]+p->c){
        int t=dfs(p->e,l>p->f?p->f:l);
        ret+=t*p->c,p->f-=t,OPT(p)->f+=t,l-=t;
    }
    return m-l;
}
int extend(int source, int target){
    static int queue[NMax+1];
    static char inq[NMax];
    static edge *fa[NMax];
    for (int i=0;i<N;i++)dist[i]=1000000000,inq[i]=0;
    dist[queue[0]=source]=0;inq[source]=1;fa[source]=NULL;
    int head=0,bot=1/*,alpha=1000000000*/;
    while (head!=bot){int x=queue[head];head=(head==NMax?0:head+1);
        inq[x]=0;
        for (edge *p=E[x];p;p=p->next)if (p->f && dist[p->e]>dist[x]+p
            ->c){
            dist[p->e]=dist[x]+p->c;fa[p->e]=OPT(p);
            if (!inq[p->e])inq[queue[bot]=p->e]=1,bot=(bot==NMax?0:bot
                +1);
        }
    }
    if (dist[target]==1000000000)return 0;
    //for (edge *p=fa[N-1];p;p=fa[p->e])if (alpha>OPT(p)->f)alpha=OPT(
        p)->f;
    //for (edge *p=fa[N-1];p;p=fa[p->e])p->f+=alpha,OPT(p)->f-=alpha,
        ret+=OPT(p)->c*alpha;
    do memset(vi,0,N*sizeof(vi[0]));
```

```
    while (dfs(source,1000000000));
    return 1;
}

void insert(int x, int y, int u, int c) {
    etop->e=y;etop->c=c;etop->f=u;etop->next=E[x];E[x]=etop++;
    etop->e=x;etop->c=-c;etop->f=0;etop->next=E[y];E[y]=etop++;
}

etop=epool;

for (int i=0;i< n + m + 2;i++)E[i]=NULL;
N = n + m + 2;
    insert(i, j + n, 1, 0);
ret=0;
while (extend(s, t));
cost = ret;
```

3.4. KM 算法

```
int mat[V][V];
int lx[V], ly[V], link[V], slack[V];
bool sx[V], sy[V];
int n,m;
bool find(int v) {
    sx[v]=true;
    for (int i=1;i<=n;i++)
    if ( !sy[i] ) {
        if ( lx[v]+ly[i]==mat[v][i] ) {
            sy[i]=true;
            if ( link[i]==0 || find( link[i] ) ) {
                link[i] = v;
                return true;
            }
        }
        else
            slack[v] = min(slack[v],lx[v]+ly[i]-mat[v][i]);
    }
    return false;
}
int KM() {
    for (int i=1;i<=n;i++) {
        ly[i] = 0;
        lx[i] = -INFI;
        for (int j=1;j<=n;j++)
            lx[i] = max(lx[i],mat[i][j]);
    }
    memset(link,0,sizeof(link));
    for (int t=1;t<=n;t++) {
        while (1) {
            memset(sx,0,sizeof(sx));
            memset(sy,0,sizeof(sy));
            memset(slack,0x5f,sizeof(slack));
            if ( find(t) ) break;
            int delta = INFI;
            for (int i=1;i<=n;i++)
            if ( sx[i] )
                delta = min(delta,slack[i]);
            for (int i=1;i<=n;i++) {
                if ( sx[i] ) lx[i] -= delta;
                if ( sy[i] ) ly[i] += delta;
            }
        }
    }
    int rec=0;
    for (int i=1;i<=n;i++)
        rec += lx[i]+ly[i];
    return rec;
}
```

3.5. 有向图强连通分支

```
void dfs(int v) {
  dfn[v] = low[v] = ++sign;
  instack[v] = true;
  stack[++top] = v;
  for (int i=0;i<adj[v].size();i++) {
    int u = adj[v][i];
    if (!dfn[u]) {
      dfs(u);
      low[v] = min(low[v], low[u]);
    } else if (instack[u])
      low[v] = min(low[v], low[u]);
  }
  if (dfn[v] == low[v]) {
    NP++;
    do {
      instack[stack[top]] = false;
      now[stack[top]] = NP;
      top--;
    } while (stack[top+1] != v);
  }
}
void SCC() {
  NP = top = sign = 0;
  memset(dfn,0,sizeof(dfn));
  memset(instack,0,sizeof(instack));
  memset(stack,0,sizeof(stack));
  for (int i=1;i<=n;i++)
  if (!dfn[i])
    dfs(i);
}
```

3.6. 一般图最大匹配

```
int g[250][250], match[250], inque[250], finish, que[250], head, tail,
    father[250], n, base[250], inblossom[250], ans;
int pop()
{
    return que[head++];
}
int push(int i)
{
    que[++tail] = i;
    inque[i] = 1;
    return 0;
}
```

```
int findancestor(int u, int v)
{
    int inpath[250];
    for (int i = 1;i <= n;i++) inpath[i] = 0;
    while (u)
    {
        u = base[u];
        inpath[u] = 1;
        u = father[match[u]];
    }
    while (v)
    {
        v = base[v];
        if (inpath[v]) return v;
        v = father[match[v]];
    }
}
void reset(int u, int anc)
{
    while (u != anc)
    {
        int v = match[u];
        inblossom[base[v]] = 1;
        inblossom[base[u]] = 1;
        v = father[v];
        if (base[v] != anc) father[v] = match[u];
        u = v;
    }
}
void contract(int u, int v)
{
    int anc = findancestor(u, v);
    for (int i = 1;i <= n;i++) inblossom[i] = 0;
    reset(u, anc);
    reset(v, anc);
    if (base[u] != anc) father[u] = v;
    if (base[v] != anc) father[v] = u;
    for (int i = 1;i <= n;i++)
        if (inblossom[base[i]])
        {
            base[i] = anc;
            if (!inque[i]) push(i);
        }
}
void findaugment(int start)
{
    for (int i = 1;i <= n;i++)
    {
        father[i] = 0;
        inque[i] = 0;
        base[i] = i;
    }
    head = 1; tail = 1; que[1] = start; inque[start] = 1;
    while (head <= tail)
    {
        int u = pop();
        for (int v = 1;v <= n;v++)
            if (g[u][v] && base[v] != base[u] && match[v] != u)
            {
                if (v == start || (match[v] && father[match[v]])) //
                    out-point
                {
                    contract(u, v);
                }
                else
                if (father[v] == 0) // not in-point
                    if (match[v])
                    {
                        push(match[v]);
                        father[v] = u;
                    }else
                    {
                        father[v] = u;
                        finish = v;
                        return;
                    }
            }
    }
}
void augment()
{
    int u = finish, v, w;
    while (u)
    {
        v = father[u];
        w = match[v];
        match[u] = v;
        match[v] = u;
        u = w;
    }
}
int main()
{
    cin >> n;
    for (int i = 1;i <= n;i++)
    {
        match[i] = 0;
        for (int j = 1;j <= n;j++)
            g[i][j] = 0;
    }
    int kk, ll;
    while (cin >> kk >> ll)
    {
        g[kk][ll] = 1;
        g[ll][kk] = 1;
    }
    ans = 0;
    for (int i = 1;i <= n;i++)
        if (!match[i])
        {
            finish = 0;
            findaugment(i);
            if (finish) {augment(); ans += 2;}
        }
    cout << ans << endl;
    for (int i = 1;i <= n;i++)
        if (match[i])
        {
            cout << i << "␣" << match[i] << endl;
            match[match[i]] = 0;
```

```
        }
}
```

3.7. 无向图边双连通分支

```
void DFS(int v) {
    dfn[v] = low[v] = ++sign;
    stack[++top] = v;
    for (int e=head[v];e!=-1;e=next[e])
    if (!edge[e]) {
        int u = pot[e];
        edge[e]=true;
        edge[e^1]=true;
        if (dfn[u] == 0) {
            DFS(u);
            low[v] = min(low[v],low[u]);
        }
        else
            low[v] = min(low[v],dfn[u]);
    }
    if (low[v] == dfn[v]) {
        NP++;
        do {
            wh[stack[top--]] = NP;
        }while (stack[top+1] != v);
    }
}
bool SCC() {
    memset(dfn,0,sizeof(dfn));
    memset(edge,0,sizeof(edge));
    sign = NP = top = 0;
    for (int i=1;i<=n;i++)
    if (dfn[i] == 0)
        DFS(i);
}
```

3.8. 无向图点双连通分支

```
pair<int,int> stack[MAXN];
void dfs(int v,int fa)
{
    dfn[v] = low[v] = ++sign;
    int son = 0;
    ge[v] = false;

    for (int i=0;i<adj[v].size();i++)
    if (adj[v][i] != fa) {
        int u = adj[v][i];
        if (dfn[u] == 0) {
            stack[++top] = make_pair(v,u);
            dfs(u, v);
            low[v] = min(low[v], low[u]);
            if (low[u] >= dfn[v])    {
                block[++bnum].clear();
                do {
                    int l = stack[top].first;
                    int r = stack[top].second;
                    if (now[l] != bnum) {
                        block[bnum].push_back(l);
                        now[l] = bnum;
                    }
                    if (now[r] != bnum) {
                        block[bnum].push_back(r);
                        now[r] = bnum;
                    }
                    top--;
                } while (!(stack[top+1].first == v && stack[top+1].
                    second == u));
                if (v != 1) ge[v] = true;
            }
            if (v == 1)
                if (++son > 1) ge[v] = true;
        }else
            low[v] = min(low[v], dfn[u]);
    }
}
```

3.9. 无向图全局最小割

```
int prim(int n) {
    best = INFI;
    for (int i=1;i<=n;i++) node[i] = i;
    while (n > 1) {
        int S = node[1];
        for (int i=1;i<=n;i++)       //初始化 dis {
            dis[ node[i] ] = mat[S][node[i]];
            use[i] = false;
        }
        use[1] = 1;
        int maxv = 1, prev;
        for (int run=1;run<n;run++) { //做最大生成树
            int maxd = -INFI;
            prev = maxv;
            for (int i=1;i<=n;i++)  //找最大值
                if (!use[i] && dis[node[i]] > maxd) {
                    maxd = dis[node[i]];
                    maxv = i;
                }
            use[maxv] = true;
            for (int i=1;i<=n;i++)   //更新 dis 值
                if (!use[i])
                  dis[ node[i] ] += mat[node[maxv]][node[i]];
        }
        best = min(best, dis[node[maxv]]);
        for (int i=1;i<=n;i++) { //合并 maxv, prev
            mat[node[i]][node[prev]] += mat[node[maxv]][node[i]];
            mat[node[prev]][node[i]] = mat[node[i]][node[prev]];
        }
        node[maxv] = node[n--]; //删除掉 maxv 这个点，直接用 node[n] 覆盖
    }
    return best;
}
```

3.10. 支配集

```
//vertex from [[[1..n]]]
```

```
const int MAXN = 200007;
list<int> e[MAXN], pred[MAXN];
int head[MAXN], next[MAXN];
int parent[MAXN], ancestor[MAXN], size[MAXN], child[MAXN];
int label[MAXN], semi[MAXN], vertex[MAXN], dom[MAXN];
int n;

void dfs(int v) {
    semi[v] = ++n;
    vertex[n] = label[v] = v;
    ancestor[v] = child[v] = 0;
    size[v] = 1;
    for (list<int>::iterator i = e[v].begin(); i != e[v].end(); ++i) {
        if (!semi[*i]) {
            parent[*i] = v;
            dfs(*i);
        }
        pred[*i].push_back(v);
    }
}

void compress(int v) {
    if (ancestor[ancestor[v]] != 0) {
        compress(ancestor[v]);
        if (semi[label[ancestor[v]]] < semi[label[v]])
            label[v] = label[ancestor[v]];
        ancestor[v] = ancestor[ancestor[v]];
    }
}

int eval(int v) {
    if (ancestor[v] == 0) {
        return v;
    }

    compress(v);
    return label[v];
}

void link(int v, int w) {
    ancestor[w] = v;
}

void initNetwork(int n) {
    for (int i = 0; i <= n; ++i) {
        e[i].clear();
        pred[i].clear();
        // bucket[i].clear();
        ancestor[i] = dom[i] = semi[i] = label[i] = vertex[i] = 0;
        head[i] = -1;
    }
}

void domi(int s) {
    n = 0;
    dfs(s);
    size[0] = label[0] = semi[0] = 0;
    for (int i = n; i >= 2; --i) {
        int w = vertex[i];
        for (std::list<int>::iterator i = pred[w].begin(); i != pred[w
            ].end(); ++i) {
            int u = eval(*i);
            if (semi[u] < semi[w])
                semi[w] = semi[u];
        }
        next[w] = head[vertex[semi[w]]];
        head[vertex[semi[w]]] = w;
        link(parent[w], w);
        while (head[parent[w]] != -1) {
            int v = head[parent[w]];
            head[parent[w]] = next[v];
            int u = eval(v);
            if (semi[u] < semi[v]) {
                dom[v] = u;
            } else {
                dom[v] = parent[w];
            }
        }
    }

    for (int i = 2; i <= n; ++i) {
        int w = vertex[i];
        if (dom[w] != vertex[semi[w]]) {
            dom[w] = dom[dom[w]];
        }
    }

    dom[s] = 0;
}
```

3.11. 树的分治

```
void calcSize(int u, int fa) {
    sz[u] = 1;
    int n = e[u].size();
    vector<pair<int,int> > &v = e[u];
    for (int i = 0;i < n; ++i)
        if (!vis[v[i].first] && v[i].first != fa) {
            calcSize(v[i].first, u);
            sz[u] += sz[v[i].first];
        }
}

int findCenter(int u, int fa, int size) {
    int maxpart = size - sz[u], n = e[u].size();
    vector<pair<int,int> > &v = e[u];
    for (int i = 0; i < n; ++i)
        if (!vis[v[i].first] && v[i].first != fa)
            maxpart = max(maxpart, sz[v[i].first]);
    if (maxpart * 2 <= size) return u;
    for (int t, i = 0; i < n; ++i)
        if (!vis[v[i].first] && v[i].first != fa) {
            t = findCenter(v[i].first,u,size);
            if (t != -1) return t;
        }
    return -1;
}

int getCenter(int u) {
    calcSize(u, -1);
    return findCenter(u,-1,sz[u]);
```

```
}

int solve(int u) {
    u = getCenter(u);
    vis[u] = true;
    //solve subproblem
    for (int i = 0; i < n; ++i)
        if (!vis[v[i].first])
            ret += solve(v[i].first);
    return ret;
}
```

3.12. 稳定婚姻匹配问题

```
struct person {
    int rank, ID;//当前匹配异性的排名和 ID
    int opp[maxn];//男人关于某个排名的异性 ID 女人关于某个异性的排名
} Man[maxn], Woman[maxn];

void StableMatch() {
    queue<int> zz;
    for (int i = 1; i <= n; i++) zz.push(i), use[i] = true;
    for (int i = 1; i <= n; i++) Man[i].rank = 1, Woman[i].rank = -1;
    while (zz.size()) {
        int x = zz.front();
        use[x] = false;
        zz.pop();
        int ID = Man[x].opp[Man[x].rank];
        while (Woman[ID].rank != -1 && Woman[ID].opp[x] > Woman[ID].
            rank) ID = Man[x].opp[++Man[x].rank];
        Man[x].ID = ID;
        if (Woman[ID].rank != -1 && !use[Woman[ID].ID]) use[Woman[ID].
            ID] = true, zz.push(Woman[ID].ID);
        Woman[ID].ID = x;
        Woman[ID].rank = Woman[ID].opp[x];
    }
}
```

3.13. 2SAT 构造解

```

void topsort()
{
    for (int i = 1; i <= Blocks; i++) cnt[i].clear();
    memset(fin, 0, sizeof(fin));
    for (int x = 0; x < 2*n; x++)
        for (int i = 0; i < edg[x].size(); i++)
        {
            int xx = edg[x][i];
            if (block[x] != block[xx])
            {
                fin[block[x]]++;
                cnt[block[xx]].push_back(block[x]);
            }
        }
    for (int x = 0; x < 2*n; x++) opp[block[x]] = block[x^1], opp[
        block[x^1]] = block[x];
    queue<int> zz;
    for (int i = 1; i <= Blocks; i++)
        if (fin[i] == 0) zz.push(i);
    memset(color, 0, sizeof(color));
    while (zz.size())
    {
        int x = zz.front();
        zz.pop();
        if (!color[x]) color[x] = 1, color[opp[x]] = 2;
        for (int i = 0; i < cnt[x].size(); i++)
        {
            int xx = cnt[x][i];
            if (--fin[xx] == 0) zz.push(xx);
        }
    }
}
```

4. 数学算法

4.1. 等差数列

```
class Arithmetic {
    /*
        getSum(N, A, B, M):
          sum{(A+Bk)/M | A >= 0, B >= 0, 0 <= k < N}
     */
public:
    static unsigned long long getSum(unsigned long long n, unsigned
        long long a, unsigned long long b, unsigned long long m) {
        if (b == 0) {
            return n * (a / m);
        }
        if (a >= m) {
            return n * (a / m) + getSum(n, a % m, b, m);
        }
        if (b >= m) {
            return n * (n - 1) / 2 * (b / m) + getSum(n, a, b % m, m);
        }
        return getSum((a + b * n) / m, (a + b * n) % m, m, b);
    }

    /*
    h(y) = a*y + b (mod m)
    You are given x,n,c,d and are curious how many of the hash values
        h(x),h(x+1),...,h(x+n) land in
    the interval [c,d].

    sum{(x-c)/m - (x-d-1)/m} % 2^64
    */

    static long long hash(long long a, long long b, long long x, long
        long n, long long c, long long d, long long m) {
        long long a0 = (b + x * a) % m;
        long long delta = (d + m) / m * m;
        unsigned long long ans = getSum(n + 1, a0 + delta - c, a, m);
        ans -= getSum(n + 1, a0 + delta - d - 1, a, m);
        return (long long)ans;
    }

};
```

4.2. 快速傅里叶变换

```
typedef complex<double > C;

void FFT(const C a[],C c[],int n,bool inv) {
    int k = int(log(1.*n)/log(2.0)+eps);
    for (int i = 0, r = (1<<k)-1;i < n;c[i++]=a[r])
        for (int t = 1<<k-1;t && !((r^=t)&t);t>>= 1);
    for (int s = 1, m = 1;s <= k;m <<= 1, s++) {
        C w0(cos(pi/(inv?-m:m)),sin(pi/(inv?-m:m))),w,u,v;
        for (int i = 0,j;i < n;i += m<<1)
            for (w = 1,j = 0;j < m;j++, w *= w0)
                u = w * c[i+j+m], v = c[i+j], c[i+j] += u,
                 c[i+j+m] = v-u;
    }
    if (inv) for (int i = 0;i < n;++i) c[i] /= n;
}
```

4.3. 线性筛法

```
int prime[V], pnum;
bool isp[V];
void getprime(int n) {
    pnum = 0;
  for (int i=2;i<n;i++) {
        if(!isp[i])
      prime[pnum++] = i;
     for(int j=0;(j<pnum && i*prime[j]<n);j++) {
            isp[i * prime[j]] = true;
            if(i % prime[j] == 0) break;
        }
    }
}
```

4.4. 辛普森数值积分

```
int sgn(double x) { return x > 1e-6 ? 1 : x < -1e-6 ? -1 : 0; }
double sps(double l, double r){
    return (f(l) + f(r) + f((l+r)/2)*4)/6 * (r - l);
}
double sps2(double l, double r, int dep){
    double cur = sps(l, r), mid = (l + r)/2;
    double y = sps(l, mid) + sps(mid, r);
    if(sgn(cur-y) == 0 && dep > 9) return cur;
    return sps2(l, mid, dep+1) + sps2(mid, r, dep+1);
}
```

4.5. Sigma i^k

```
void pre() {
    stirl[1][1] = 1;
    for (int i = 2; i < MAXN; ++i) {
        stirl[i][1] = stirl[i][i] = 1;
        for (int j = 2; j < i;  ++j)
            stirl[i][j] = (((LL)j*stirl[i-1][j])%MOD + stirl[i-1][j
                -1])%MOD;
    }

    for (int i = 1; i < MAXN ; ++i) inv[i] = POW(i, MOD-2);
}

int sum_cal(int n, int k) {
    int res = 0, X = n+1;
    int tmp;

    for (int i = 1; i <= k; ++i) {
        X = ((LL)X * (n-i+1))%MOD;
        tmp = ((LL)stirl[k][i] * inv[i+1])%MOD;
        tmp = ((LL)X * tmp)%MOD;
        res = (res+tmp)%MOD;
    }

    return res;
}
```

4.6. 中国剩余定理及拓展欧几里得算法

```
LL mulmod( LL a, LL b, LL c);
int powmod( LL a, int b, int c);
void ext_gcd(LL a, LL b, LL &x, LL &y){
  if (!b) { x = 1; y = 0; return;}
  ext_gcd( b, a % b, y, x);
  y -= a / b * x;
}
LL CRT(int b[], int w[], int k) {
  int i;
  LL x, y, a = 0, m, n = 1;
  for (i = 0; i < k; i++) n *= w[i];
  for (i = 0; i < k; i++) {
    m = n / w[i];
    ext_gcd(w[i], m, x, y);
    a = (a + mulmod( mulmod(y , m, n), b[i], n) ) % n;
  }
  return a < 0 ? a + n : a;
}

private long[] exgcd(long x, long y) {
    long a0 = 1, a1 = 0, b0 = 0, b1 = 1, t;
    while (y != 0) {
        t = a0 - x / y * a1; a0 = a1; a1 = t;
        t = b0 - x / y * b1; b0 = b1; b1 = t;
        t = x % y; x = y; y = t;
    }

    if (x < 0) {
        a0 = -a0;
        b0 = -b0;
        x = -x;
    }

    return new long[]{a0, b0, x};
}
```

4.7. 解同余方程组

解同余方程（及同余方程组）
input: a*x % n = b
output: x 的最小解或 -1 表示无解

```
int solve_equ(int a,int b,int n) {
  ll x,y;
  int d = ex_gcd(a,n,x,y);
  if (b % d != 0) return -1;
  x = x * b/d;
  x = (x % (n/d) + (n/d)) % (n/d);
  return x;
}
long long multi_mod_equ(long long m[],long long a[],int n) {
  if (n == 0) return 0;
  long long a1,a2,m1,m2,delta, d, c, x, y;
  m1 = m[0];
  a1 = a[0];
  for (int i=1;i<n;i++) {
    m2 = m[i];
    a2 = a[i];
    delta = a1 - a2;
    d = ex_gcd(m1,m2,x,y);
    if (delta % d == 0) {
      long long tmp = m1 / d;
      y = (delta/d*y %tmp + tmp)%tmp;
      c = y*m2 + a2;
      a1 = c;
      m1 = lcm(m1,m2);
    } else
      return -1;
  }
  return a1;
}
```

4.8. 求逆元

```
rng_58神奇的算逆元的模板(MOD为素数时)
    inv[1] = 1;
    for(i=2;i<MOD;i++)
      inv[i] = (MOD - MOD/i) * inv[(int)(MOD%i)] % MOD;
1.对于 b,MOD 不互质，没有逆元时求模数:)
1假如 b 事先已知，那么一开始就设 MOD=MOD*b 即可)
2还有就是不断 +n
    while(a%MOD != 0) a += MOD;
      a = a/MOD;
```

4.9. Nim 积

```
long long Nim_Multi(long long x, long long y)
{
    if (x < y) return Nim_Multi(y, x);
    if (x < 2) return sg[x][y]; //sg[2][2]的表
    long long m = 2;
    for (int i = 0; x >= (1ll << (1<<i)); i++) m = 1ll << (1<<i);
    long long p = x/m, q = x%m, s = y/m, t = y%m;
    long long c1 = Nim_Multi(p, s);
    long long c2 = Nim_Multi(p, t)^Nim_Multi(q, s);
    long long c3 = Nim_Multi(q, t);
    return (m*(c1^c2))^c3^Nim_Multi(m/2, c1);
}
```

4.10. 高斯消元

高斯消元法解方程组 (整数 + 浮点数版 + 同余版)
input: 方程组矩阵 mat[][] var - 变量数, equ - 方程数
output: 有唯一解时返回解集或无解 -1

```
long long Gauss(long long a[][MAXN],int equ,int var) {
  int r, c;
  for (r=0,c=0;r<equ && c < var;r++,c++)// 枚举当前处理的行.  {
    // 浮点数模板时，找到该 c 列元素绝对值最大的那行与第 k 行交换.(为了在除
       法时减小误差)
    int maxr = r;
    for (int i=r+1;i<equ;i++)
      if (abs(a[i][c]) > abs(a[maxr][c]))
        maxr = i;

    if (maxr != r) { // 与第 r 行交换.
      for (int j=r;j<var+1;j++)
        swap(a[r][j],a[maxr][j]);
    }

    // 说明该 c 列第 r 行以下全是 0 了，则处理当前行的下一列.
      if (a[r][c] == 0) {
      r--;
      continue;
    }
    for (int i=r+1;i<equ;i++) { // 枚举要删去的行
      if (a[i][c] != 0) {
        // 整数版
        long long ta = a[r][c]; long long tb = a[i][c];
        for (int j=c;j<var+1;j++)
          a[i][j] = a[i][j] * ta - a[r][j] * tb;
      }
    }
  }
  // 1. 无解的情况：化简的增广阵中存在 (0, 0, ..., a) 这样的行 (a != 0).
  for (int i=r;i<equ;i++) {
    if (a[i][c] != 0) return -1;
  }
  if (r < var)    // 自由变元有var - 个r.
    return var - r;
  // 3. 唯一解的情况：在 var * (var + 1) 的增广阵中形成严格的上三角阵，计算
     出 Xn-1, Xn-2..x0
  for (int i=var-1;i>=0;i--) {
    int tmp = a[i][var];
    for (int j=i+1;j<var;j++) {
      if (a[i][j] != 0) tmp -= a[i][j] * x[j];
    }
    //浮点或整数方程组
```

```
    if (tmp % a[i][i] != 0) return -2; // 说明有浮点数解，但无整数解.
    x[i] = tmp / a[i][i];
    //模方程组
    while (tmp % a[i][i]) tmp += MOD;
    x[i] = tmp / a[i][i];
  }
  return 0;
}
```

4.11. RHO

大数分解 + millar 裸奔
将一个 long long 级别的大数分解质因数 (复杂度 $n^{\frac{1}{4}}$)
millar 裸奔是利用 $a^{n-1}\%n == 1$ 为素数 (否则合数)
input: x
output: fac[] - 因子的 vector
为素数时直接返回该素数
为 1 时返回 fac 为空

```
long long pow(long long a,long long x,long long n) {
  if ( x==0 ) return 1;
  long long mid = pow(a,x/2,n);
  if ( mid==0 ) return 0;
  long long ans=mul(mid,mid,n);
  if ( ans==1 && mid != 1 && mid!=n-1 ) return 0;
  if ( x&1 ) ans = mul(ans,a,n);
  return ans;
}
bool millar_rabin(long long n) {
  if ( n<=1 ) return 0;
  if ( n==2 ) return 1;
  for (int i=0;i<5;i++) {
    long long a = rand() % (n-2) + 2;
    if ( pow(a,n-1,n) != 1) return 0;
  }
  return 1;
}
long long pollard_rho(long long n,long long c) {
  long long i=1, x=rand() % (n-1)+1, y=x , k=2,d;
  while ( 1 ) {
    i++;
    x = ( mul(x,x,n) - c + n) % n;
    long long d=gcd((y-x+n)%n,n);
    if ( d>1 && d<n ) return d;
    if ( x==y ) return n;
    if ( i==k ) {
      y = x;
      k <<= 1;
    }
  }
}
void getfac(long long n) {
  if (n <= 1) return;
  if ( millar_rabin(n) ) {
    fac.push_back(n);
  } else {
    long long k;
    do {
      k = pollard_rho(n,rand()%(n-1)+1);
    } while ( k>=n );
    getfac(k);
    getfac(n/k);
  }
}
```

4.12. x ／ m % mod，mod 比较小时的处理方法

```
mod = mod * mod
if (m % 10007 != 0) {
  ans = ans * QuickPower(m, MOD-2) % 10007;
} else
  ans = ans * QuickPower(m/10007, MOD-2) % MOD / 10007;
```

5. 计算几何

2D Geometry
3D Computing Geometry
3D Convex Hull
简单多边形面积并
圆面积并和交，利用扫描线求的。
平面最近点对, 分治法
最小覆盖矩形，旋转卡壳法的经典应用
最小覆盖圆，随机增量法。
圆和简单多边形的交
将三维点集投影到一个平面上，并求投影面积
NlogN 半平面交，（该版本还没加排序, 因为给的直线本身有序）

```
double cpr(const pt &o,const pt &a,const pt &b) { return (a.x-o.x)*(b.
    y-o.y) - (a.y-o.y)*(b.x-o.x); }
double dpr(const pt &a,const pt &b,const pt &c) { return (b.x-a.x)*(c.
    x-a.x) + (b.y-a.y)*(c.y-a.y); }
//直线线段非严格相交，包括交在端点等，严格相交是<=0
bool line_seg_cross(pt a,pt b,pt c,pt d) // a,b : Line    c,d :
    Segment {
    return sgn(det(c-a,b-a)) * sgn(det(d-a,b-a)) <= 0;
}
//无法处理整条直线重合的情况
//直线 直线相交，相交 1. 平行 0. 重合 -1.
int line_line_cross(pt a,pt b,pt c,pt d) {
    if (sgn(det(b-a,d-c)) != 0) return 1;
    return sgn(det(b-a,d-a)) ? 0 : -1;
}
//线段 线段非严格相交 ,
bool cross(pt a,pt b,pt c,pt d) {
    int d1 = sgn(det(a-c,d-c)),
        d2 = sgn(det(b-c,d-c)),
        d3 = sgn(det(c-a,b-a)),
        d4 = sgn(det(d-a,b-a));
    if (d1*d2 == -1 && d3*d4 == -1)
        return true;
    // 下面是用来判断非严格相交部分。
```

```
    if (!d1 && sgn(dot(a-c,a-d)) <= 0) return true; //在a[c,d上]
    if (!d2 && sgn(dot(b-c,b-d)) <= 0) return true;
    if (!d3 && sgn(dot(c-a,c-b)) <= 0) return true;
    if (!d4 && sgn(dot(d-a,d-b)) <= 0) return true;
    return false;
}
//求直线 (a->b) (c->d) 交点
pt cross_point(pt a,pt b,pt c,pt d) {
    double radio = det(c-a,b-a) / det(b-a,d-c);
    return c + (d-c) * radio;
}
//点到x 直线 (a->b)的距离
double point_line_dis(pt x,pt a,pt b) {
    return fabs(det(a-x,b-x) / dis(a,b));
}
//点到x 线段 (a->b)的距离
double point_seg_dis(pt x,pt a,pt b) {
    if (sgn(dot(b-a,x-a)) * sgn(dot(b-a,x-b)) < 0)
        return point_line_dis(x, a, b);
    return min(dis(a,x),dis(b,x));
}
//线段 (c->d) 到线段  (a->b) 的距离
double seg_seg_dis(pt a,pt b,pt c,pt d) {
    return min(min(point_seg_dis(a,c,d),point_seg_dis(b,c,d)),min(
        point_seg_dis(c,a,b),point_seg_dis(d,a,b)));
}
pt rotate(double ang) {
    return pt(x*cos(ang) - y*sin(ang), x*sin(ang) + y*cos(ang));
}
//计算圆与圆的交点保证圆与圆有交点圆心不重合,, by isun
void intersection_circle_circle(pt c1, double r1, pt c2, double r2, pt
     &p1, pt &p2) {
    double d2 = (c1.x - c2.x) * (c1.x - c2.x) + (c1.y - c2.y) * (c1.y
        - c2.y);
    double cos = (r1 * r1 + d2 - r2 * r2) / (2 * r1 * sqrt(d2));
    pt v1 = (c2 - c1) / dis(c1, c2), v2 = pt(-v1.y, v1.x) * (r1 * sqrt
        (1 - cos * cos));
    pt X = c1 + v1 * (r1 * cos);
    p1 = X + v2;
    p2 = X - v2;
}
// 球和直线相交
bool intersection_line_circle(pt p, pt p1, double &x1, double &x2)
    const {
    // line:    p(t) = p + dir * t
    // sphere:  (p - o)^2 = r*r
    //          (p - o + dir * t) ^ 2 = r*r
    //dir^2 * t^2 + 2*dir*(p-o)*t + (p-o)^2 == r*r;
    pt dir = p1 - p;
    float c2 = dot(dir, dir);
    float c1 = 2 * dot(dir, p-o);
    float c0 = dot(p-o, p-o) - r*r;
    float delta = c1*c1 - 4*c2*c0;
    if (delta < -eps)
        return false;
    delta = fabs(delta);
    // closest intersection point
    double x1 = (-c1 - sqrt(delta)) / (2*c2);
    double x2 = (-c1 + sqrt(delta)) / (2*c2);
}
// 自己写的圆的内外公切线模板，其实非常简单就是算出角度再旋转，注意相交时
    没有内公切线，返回第一个圆上的切点。
void circle_circle(pt a,pt b,double r1,double r2,pt &t0,pt &t1,pt &t2,
    pt &t3) {
    double ang = acos( (r1-r2)/dis(a, b) );
    double ang1 = acos( (r1+r2) / dis(a,b) );
    pt p1 = a + ((b-a).unit(r1));
    t0 = rotate(p1, a, -ang);
    t1 = rotate(p1, a, -ang1);
    t2 = rotate(p1, a, ang1);
    t3 = rotate(p1, a, ang);
}
//三维几何
struct pt3 {
    pt3 operator * (pt3 p){return pt3(y*p.z-z*p.y, z*p.x-x*p.z, x*p.y-
        y*p.x);} //叉
        乘
    double operator ^ (pt3 p){return x*p.x+y*p.y+z*p.z;} //点乘
};
pt det(pt a, pt b) { return pt(a.y*b.z-a.z*b.y, a.z*b.x-a.x*b.z,  a.x*
    b.y-a.y*b.x); }  //叉
    乘
double dot(pt a, pt b) { return a.x*b.x + a.y*b.y + a.z*b.z; }
//点到平面距离
double ptoplane(pt3 p, pt3 s1, pt3 s2, pt3 s3) {
    pt3 norm = (s2 - s1) * (s3 - s1);
    return fabs(norm ^ (p - s1)) / vlen(norm);
}
//点到直线距离
double ptoline(pt3 p, pt3 l1, pt3 l2){
    return vlen((p-l1)*(l2-l1)) / dis(l1, l2);
}
//直线到直线距离
double linetoline(pt3 u1, pt3 u2, pt3 v1, pt3 v2){
    pt3 n = (u1 - u2) * (v1 - v2);
    return fabs((u1 - v1) ^ n) / vlen(n);
}
//判点是否在空间三角形上包括边界三点共线无意义,,
bool dot_intri_in(pt3 p, pt3 s1, pt3 s2, pt3 s3)
{
    return zero(vlen((s1-s2)*(s1-s3))-vlen((p-s1)*(p-s2))-vlen((p-s2)
        *(p-s3))-vlen((p-s3)*(p-s1)));
}
//判点是否在空间三角形上不包括边界三点共线无意义,,
bool dot_intri_ex(pt3 p, pt3 s1, pt3 s2, pt3 s3)
{
    return dot_intri_in(p,s1,s2,s3)&&
    vlen((p-s1)*(p-s2))>eps&&vlen((p-s2)*(p-s3))>eps&&vlen((p-s3)*(p-
        s1))>eps;
}
//计算直线与平面交点注意事先判断是否平行并保证三点不共线,,!
//线段和空间三角形交点请另外判断
pt3 intersection(pt3 l1, pt3 l2, pt3 s1, pt3 s2, pt3 s3)
```

```
{
    pt3 norm = (s1 - s2) * (s2 - s3);
    double t = (norm ^ (s1 - l1)) / (norm ^ (l2 - l1));
    return l1 + (l2 - l1) * t;
}
//判断直线是否穿过空间三角形abs1s2s3
bool line_throughtri(pt3 a, pt3 b, pt3 s1, pt3 s2, pt3 s3)
{
    pt3 norm = (s2 - s1) * (s3 - s1);
    if (((a - s1)^norm) * ((b - s1)^norm) > 0 || fabs((a - b)^norm) <
        eps)
        return 0;
    pt3 X = intersection(a, b, s1, s2, s3);
    return dot_intri_ex(X, s1, s2, s3);
}
//点绕过原点的向量旋转顺时针角后得到的点pvA
pt3 rotate(pt3 p, pt3 v, double A)
{
    double len = sqrt(v.x*v.x+v.y*v.y+v.z*v.z);
    double x = v.x/len, y = v.y/len, z = v.z/len;
    double M[][3] =
    {
        cos(A)+(1-cos(A))*x*x, (1-cos(A))*x*y-sin(A)*z, (1-cos(A))*x*z
            +sin(A)*y,
        (1-cos(A))*y*x+sin(A)*z, cos(A)+(1-cos(A))*y*y, (1-cos(A))*y*z
            -sin(A)*x,
        (1-cos(A))*z*x-sin(A)*y, (1-cos(A))*z*y+sin(A)*x, cos(A)+(1-
            cos(A))*z*z
    };
    return pt3 (p.x * M[0][0] + p.y * M[1][0] + p.z * M[2][0],
                p.x * M[0][1] + p.y * M[1][1] + p.z * M[2][1],
                p.x * M[0][2] + p.y * M[1][2] + p.z * M[2][2]);
}
//取平面法向量
pt3 pvec(pt3 s1, pt3 s2, pt3 s3){
    return (s1 - s2) * (s2 - s3);
}
//判点是否在线段上包括端点,
bool dot_online_in(pt3 p, pt3 l1, pt3 l2) {
    return sgn(vlen(det(p - l1, p - l2))) == 0 && sgn(dot(p-l1, l2-l1)
         * dot(p-l2,l2-l1)) <= 0;
}
//判两点在线段同侧点在线段上返回不共面无意义,0,
bool same_side(pt3 p1, pt3 p2, pt3 l1, pt3 l2) {
    return sgn(dot(det(l1 - l2,p1 - l2), det(l1 - l2,p2 - l2))) >= 0;
}
//判两点在平面同侧点在平面上返回,0
bool same_side(pt3 p1, pt3 p2, pt3 s1, pt3 s2, pt3 s3){
    return (pvec(s1,s2,s3) ^ (p1-s1)) * (pvec(s1,s2,s3) ^ (p2 - s1)) >
         eps;
}
//判两直线平行
int parallel(pt3 u1, pt3 u2, pt3 v1, pt3 v2) {
    return vlen((u1 - u2) * (v1 - v2)) < eps;
}
//判两线段相交包括端点和部分重合,
int intersect_in(pt3 u1, pt3 u2, pt3 v1, pt3 v2) {
    if (!dots_onplane(u1, u2, v1, v2)) // 四点共面
        return 0;
    if (!dots_inline(u1, u2, v1) || !dots_inline(u1, u2, v2))
        return !same_side(u1, u2, v1, v2) && !same_side(v1, v2, u1, u2
            );
    return  dot_online_in(u1, v1, v2) || dot_online_in(u2, v1, v2) ||
            dot_online_in(v1, u1, u2) || dot_online_in(v2,u1,u2);
}
//判两线段相交不包括端点和部分重合,
int intersect_ex(pt3 u1, pt3 u2, pt3 v1, pt3 v2){
    return  dots_onplane(u1, u2, v1, v2) &&
            opposite_side(u1, u2, v1, v2) && opposite_side(v1, v2, u1,
                 u2);
}
//判线段与空间三角形相交包括交于边界和部分,()包含
int intersect_in(pt3 l1, pt3 l2, pt3 s1, pt3 s2, pt3 s3){
    return  !same_side(l1, l2, s1, s2, s3) &&
            !same_side(s1, s2, l1, l2, s3) &&
            !same_side(s2, s3, l1, l2, s1) &&
            !same_side(s3, s1, l1, l2, s2);
}
//计算两平面交线注意事先判断是否平行并保证三点不共线,,!
void intersection_plane(pt3 u1, pt3 u2, pt3 u3, pt3 v1, pt3 v2, pt3 v3
    , pt3 &a, pt3 &b){
    a = parallel(v1,v2,u1,u2,u3)?intersection_plane_line(v2,v3,u1,u2,
        u3):intersection_plane_line(v1,v2,u1,u2,u3);
    b = parallel(v3,v1,u1,u2,u3)?intersection_plane_line(v2,v3,u1,u2,
        u3):intersection_plane_line(v3,v1,u1,u2,u3);
}
//直线和平面夹角值sin
double angle_sin(pt3 l1, pt3 l2, pt3 s1, pt3 s2, pt3 s3){
    return ((l1-l2)^pvec(s1,s2,s3)) / vlen(l1-l2) / vlen(pvec(s1,s2,s3
        ));
}
struct _3DCH {
    struct fac{
        int a, b, c;     //表示凸包一个面上三个点的编号
        bool ok;         //表示该面是否属于最终凸包中的面
    };
    int n;  //初始点数
    pt P[MAXV]; //初始点
    int cnt;     //凸包表面的三角形数
    fac F[MAXV*8]; //凸包表面的三角形
    int to[MAXV][MAXV];

    double vlen(pt a){return sqrt(a.x*a.x+a.y*a.y+a.z*a.z);}     //向量
        长度
    double area(pt a, pt b, pt c){return vlen((b-a)*(c-a));}     //三角
        形面积*2
    double volume(pt a, pt b, pt c, pt d){return (b-a)*(c-a)^(d-a);}
          //四面体有向体
        积*6
    //正: 点在面同向
    double ptof(pt &p, fac &f){
        pt m = P[f.b]-P[f.a], n = P[f.c]-P[f.a], t = p-P[f.a];
        return (m * n) ^ t;
    }
    void deal(int p, int a, int b){
```

```
        int f = to[a][b];
        fac add;
        if (F[f].ok){
            if (ptof(P[p], F[f]) > eps)
                dfs(p, f);
            else{
                add.a = b, add.b = a, add.c = p, add.ok = 1;
                to[p][b] = to[a][p] = to[b][a] = cnt;
                F[cnt++] = add;
            }
        }
    }
    void dfs(int p, int cur){
        F[cur].ok = 0;
        deal(p, F[cur].b, F[cur].a);
        deal(p, F[cur].c, F[cur].b);
        deal(p, F[cur].a, F[cur].c);
    }
    bool same(int s, int t) {
        pt &a = P[F[s].a], &b = P[F[s].b], &c = P[F[s].c];
        return fabs(volume(a, b, c, P[F[t].a])) < eps && fabs(volume(a
            , b, c, P[F[t].b])) < eps && fabs(volume(a, b, c, P[F[t].c
            ])) < eps;
    }
    //构建三维凸包
    void construct(){
        cnt = 0;
        if (n < 4)
            return;
        /*此段是为了保证前四个点不公面，若已保证，可去掉****************/
        bool sb = 1;
        //使前两点不公点
        for (int i = 1; i < n; i++){
            if (vlen(P[0] - P[i]) > eps){
                swap(P[1], P[i]);
                sb = 0;
                break;
            }
        }
        if (sb)return;
        sb = 1;
        //使前三点不公线
        for (int i = 2; i < n; i++){
            if (vlen((P[0] - P[1]) * (P[1] - P[i])) > eps){
                swap(P[2], P[i]);
                sb = 0;
                break;
            }
        }
        if (sb)return;
        sb = 1;
        //使前四点不共面
        for (int i = 3; i < n; i++){
            if (fabs((P[0] - P[1]) * (P[1] - P[2]) ^ (P[0] - P[i])) >
                eps){
                swap(P[3], P[i]);
                sb = 0;
                break;
            }
        }
        if (sb)return;
        /*此段是为了保证前四个点不公面****************/
        fac add;
        for (int i = 0; i < 4; i++){
            add.a = (i+1)%4, add.b = (i+2)%4, add.c = (i+3)%4, add.ok
                = 1;
            if (ptof(P[i], add) > 0)
                swap(add.b, add.c);
            to[add.a][add.b] = to[add.b][add.c] = to[add.c][add.a] =
                cnt;
            F[cnt++] = add;
        }
        for (int i = 4; i < n; i++){
            for (int j = 0; j < cnt; j++){
                if (F[j].ok && ptof(P[i], F[j]) > eps){
                    dfs(i, j);
                    break;
                }
            }
        }
        int tmp = cnt;
        cnt = 0;
        for (int i = 0; i < tmp; i++){
            if (F[i].ok){
                F[cnt++] = F[i];
            }
        }
    }
    //表面积
    double area(){
        double ret = 0.0;
        for (int i = 0; i < cnt; i++){
            ret += area(P[F[i].a], P[F[i].b], P[F[i].c]);
        }
        return ret / 2.0;
    }
    //体积
    double volume(){
        pt O(0, 0, 0);
        double ret = 0.0;
        for (int i = 0; i < cnt; i++){
            ret += volume(O, P[F[i].a], P[F[i].b], P[F[i].c]);
        }
        return fabs(ret / 6.0);
    }
    //表面三角形数
    int facetCnt_tri(){
        return cnt;
    }
    //表面多边形数
    int facetCnt(){
        int ans = 0;
        for (int i = 0; i < cnt; i++){
            bool nb = 1;
            for (int j = 0; j < i; j++){
                if (same(i, j)){
                    nb = 0;
                    break;
                }
```

```
            }
            ans += nb;
        }
        return ans;
    }
};
/*常用几何算法**********************************************/
//半平面交：向量a->的左手边b O(N^2)
void half_plane_its(pt p[],int &n,pt a,pt b,pt tmp[]) {
    int tot = 0;
    for (int i = 0;i < n;++i) {
        int now = sgn(det(b-a,p[i]-a)),
            next = sgn(det(b-a,p[(i+1)%n]-a));
        if (now >= 0)
            tmp[tot++] = p[i];
        if (now * next < 0)
            tmp[tot++] = cross_point(a,b,p[i],p[(i+1)%n]);
    }
    n = tot;
    for (int i = 0;i < n;++i)
        p[i] = tmp[i];
}
//创建点集p的凸包，非严格形式，可以有中间点，输出点[]
//注意：原始点集序列将被排序，因为是传指针
//对于无中间点的严格形式凸包，改sgn() <= 即可0
void convex_hull(pt p[],int n,pt s[],int &top) {
    top = 0;
    sort(p,p+n, [](const pt &a, const pt &b) {
        if (sgn(a.y-b.y) != 0)
            return sgn(a.y-b.y) < 0;
        return sgn(a.x-b.x) < 0;
    });
    for (int i = 0;i < n;i++) {
        while (top > 1 && sgn(det(s[top-1] - s[top-2],p[i] - s[top-2])
            ) < 0) top--;
        s[top++] = p[i];
    }
    int mid = top;
    for (int i = n-2;i >= 0;i--) {
        while (top > mid && sgn(det(s[top-1] - s[top-2],p[i] - s[top
            -2])) < 0) top--;
        s[top++] = p[i];
    }
    top--;
}
//多边形重心
pt barycenter(int n, pt *p) {
    pt ret(0, 0), t;
    double t1 = 0, t2;
    for (int i = 1; i < n - 1; i++)
    {
        if (fabs(t2 = cpr(p[i+1], p[0], p[i])) > eps)
        {
            t.x = (p[0].x + p[i].x + p[i+1].x) /3.0;
            t.y = (p[0].y + p[i].y + p[i+1].y) /3.0;
            ret.x += t.x*t2;
            ret.y += t.y*t2;
            t1 += t2;
        }
    }
    if (fabs(t1) > eps)
        ret.x /= t1, ret.y /= t1;
    return ret;
}三角形外心：垂直平分线的交点三角形内心：角平分线的交点（角平分线是

A -> unit(B-A) + unit(C-A))三角形费马点：模拟退火，从重心往四个方向选代价
    最小的拓展

//简单多边形面积并
struct pt
{
    double x, y;
    pt(){}
    pt(double _x, double _y):x(_x), y(_y){}
    pt operator - (const pt p1){return pt(x - p1.x, y - p1.y);}
    pt operator + (const pt p1){return pt(x + p1.x, y + p1.y);}
    pt operator * (double s){return pt(x * s, y * s);}
    pt operator / (double s){return pt(x / s, y / s);}
    bool operator < (const pt p1)const{return y < p1.y-eps || y < p1.y
        +eps && x < p1.x;}
};
double cpr(pt a, pt b, pt c) { return (b.x-a.x)*(c.y-a.y)-(b.y-a.y)*(c
    .x-a.x); }
double cpr(pt a, pt b) {return a.x*b.y-a.y*b.x;}
double dpr(pt a, pt b, pt c) { return (b.x-a.x)*(c.x-a.x)+(b.y-a.y)*(c
    .y-a.y); }
inline double dpr(pt a, pt b) { return a.x*b.x+a.y*b.y; }
pt its(const pt &a, const pt &b, const pt &c, const pt &d)
{
    pt ret = a;
    double t =  ((c.x - a.x)*(d.y - c.y) - (c.y - a.y)*(d.x - c.x))/
                ((b.x - a.x)*(d.y - c.y) - (b.y - a.y)*(d.x - c.x));
    ret.x += (b.x - a.x) * t;
    ret.y += (b.y - a.y) * t;
    return ret;
}
pair<double, int> e[510];
int cnt;
inline void insert(pt &s, pt &t, pt X, int inc)
{
    double ratio = SGN(t.x - s.x) ? (X.x - s.x) / (t.x - s.x) : (X.y -
        s.y) / (t.y - s.y);
    if (ratio > 1.0)ratio = 1.0;
    if (ratio < 0.0)ratio = 0.0;
    e[cnt++] = make_pair(ratio, inc);
}
double poly_union(vector<vector <pt> > &p) {
    double ans = 0.0;
    int cp0, cp1, cp2, cp3;

    for (int i = 0; i < p.size(); i++) {
        for (int k = 0; k < p[i].size(); k++) {
            pt &s = p[i][k], &t = p[i][(k + 1) % p[i].size()];
            if (fabs(cpr(s, t)) < eps)continue;
```

```
            cnt = 0;
            e[cnt++] = make_pair(0.0, 1);
            e[cnt++] = make_pair(1.0, -1);
            for (int j = 0; j < p.size(); j++) if (i != j) {
                for (int l = 0; l < p[j].size(); l++) {
                    pt &a = p[j][l], &b = p[j][(l + 1) % p[j].size()];
                    cp0 = SGN(cpr(s, t, p[j][(l + p[j].size() - 1) % p
                        [j].size()]));
                    cp1 = SGN(cpr(s, t, a));
                    cp2 = SGN(cpr(s, t, b));
                    if (cp1 * cp2 < 0)
                        insert(s, t, its(s, t, a, b), -cp2);
                    else if (!cp1 && cp0 * cp2 < 0)
                        insert(s, t, a, -cp2);
                    else if (!cp1 && !cp2) {
                        cp3 = SGN(cpr(s, t, p[j][(l + 2) % p[j].size()
                            ]));
                        int dp = SGN(dpr(t - s, b - a));
                        if (dp && cp0)insert(s, t, a, dp > 0 ? cp0 * (
                            j > i ^ cp0 < 0) : -(cp0 < 0));
                        if (dp && cp3)insert(s, t, b, dp > 0 ? -cp3 *
                            (j > i ^ cp3 < 0) : cp3 < 0);
                    }
                }
            }
            sort(e, e + cnt);
            int acc = 0;
            double total = 0.0, last;
            for (int j = 0; j < cnt; j++) {
                if (acc == 1)
                    total += e[j].first - last;
                acc += e[j].second;
                last = e[j].first;
            }
            ans += cpr(s, t) * total;
        }
    }
    return fabs(ans) * 0.5;
}
//圆面积并 和交，利用扫描线求的。
int n;
pt o[1010];
double r[1010];
pair<double, int> e[2010];
int cnt;
double ans[1010];
//0:包含ab    1:包含ba 相交 2: 相离    3:
inline int rlt(int a, int b) {
    double d = dis(o[a], o[b]), d1 = SGN(d - r[a] + r[b]), d2 = SGN(d
        - r[b] + r[a]);
    if (d1 < 0 || !d1 && (d > eps || a > b))return 0;
    if (d2 < 0 || !d2 && (d > eps || a < b))return 1;
    return d < r[a] + r[b] - eps ? 2 : 3;
}
inline double arcArea(pt &o, double r, double ang1, double ang2) {
    pt a(o.x + r * cos(ang1), o.y + r * sin(ang1));
    pt b(o.x + r * cos(ang2), o.y + r * sin(ang2));
    double dif = ang2 - ang1;
    return (cpr(a, b) + (dif - sin(dif)) * r * r) * 0.5;
}
void circleUnion(pt o[], double r[])
{
    double last, center, d2, ang, angX, angY;
    pt X, Y;

    for (int i = 0; i < n; i++)
    if (r[i] > eps)
    {
        int acc = 0;
        cnt = 0;
        e[cnt++] = make_pair(-PI, 1);
        e[cnt++] = make_pair(PI, -1);
        for (int j = 0; j < n; j++)
        if (i != j && r[j] > eps)
        {
            int rel = rlt(i, j);
            if (rel == 1)
            {
                e[cnt++] = make_pair(-PI, 1);
                e[cnt++] = make_pair(PI, -1);
            } else if (rel == 2)
            {
                center = atan2(o[j].y - o[i].y, o[j].x - o[i].x);
                d2 = (o[i].x - o[j].x) * (o[i].x - o[j].x) + (o[i].y -
                    o[j].y) * (o[i].y - o[j].y);
                ang = acos((r[i] * r[i] + d2 - r[j] * r[j]) / (2 * r[i
                    ] * sqrt(d2)));
                angX = center + ang;
                angY = center - ang;
                if (angX > PI)angX -= 2*PI;
                if (angY < -PI)angY += 2*PI;
                if (angX < angY) acc++;
                e[cnt++] = make_pair(angY, 1);
                e[cnt++] = make_pair(angX, -1);
            }
        }
        sort(e, e + cnt);
        last = -PI;
        for (int j = 0; j < cnt; j++)
        {
            double tmp = arcArea(o[i], r[i], last, e[j].first);
            ans[acc] += tmp;
            ans[acc - 1] -= tmp;
            acc += e[j].second;
            last = e[j].first;
        }
    }
}
// 分治法求平面最近点对
bool CompareByX(const pt &a, const pt &b) {
    return sgn(a.x - b.x) < 0 || (sgn(a.x - b.x) == 0 && sgn(a.y - b.y
        ) < 0);
}
bool CompareByY(const pt &a, const pt &b) {
    return sgn(a.y - b.y) < 0 || (sgn(a.y - b.y) == 0 && sgn(a.x - b.x
        ) < 0);
```

```
}
int n;
vector<pt> p, merge_backup;
vector<pt> Left, right;
double closest_pair_point_distance(int l, int r)
{
    if (l >= r)
        return INFI;
    int mid = (l + r) / 2;
    double mid_x = p[mid].x;
    double min_left = closest_pair_point_distance(l, mid);
    double min_right = closest_pair_point_distance(mid+1, r);
    double min_all = min(min_left, min_right);
    Left.clear();
    right.clear();
    for (int i = l; i <= mid; i++)
    if (p[i].x > mid_x - min_all - eps)
        Left.push_back(p[i]);

    for (int i = mid+1; i <= r; i++)
    if (p[i].x < mid_x + min_all + eps)
        right.push_back(p[i]);

    sort(Left.begin(), Left.end(), CompareByY);
    sort(right.begin(), right.end(), CompareByY);

    double ret = min_all;
    int j1 = 0;
    for (int i = 0; i < Left.size(); i++) {
        // j1 is the last one in the box
        while (j1 < right.size() && right[j1].y - Left[i].y < -min_all
            +eps) j1++;
        assert(j1 == right.size() || right[j1].y - Left[i].y > -
            min_all+eps);
        for (int j = j1; j < right.size(); j++) {
            if (right[j].y - Left[i].y > min_all) break;
            if (Left[i].tag != right[j].tag)
                ret = min(ret, dis(Left[i], right[j]));
        }
    }
    return ret;
}
int main() {
    sort(p.begin(), p.end(), CompareByX);
    printf("%.3f\n", closest_pair_point_distance(0, 2*n-1));
}
//最小覆盖矩形，旋转卡壳法的经典应用
void min_rectangle(pt p[],int n,double &area,double &perimeter) {
    int l = 1 ,r = 1, u = 1;
    for (int i=0;i<n;i++) {
        pt edge = (p[(i+1)%n] - p[i]).unit();
        while (dot(edge, p[r%n]-p[i]) < dot(edge, p[(r+1)%n]-p[i])) r
            ++;
        while (u < r || det(edge, p[u%n]-p[i]) < det(edge, p[(u+1)%n]-
            p[i])) u++;
        while (l < u || dot(edge, p[l%n]-p[i]) > dot(edge, p[(l+1)%n]-
            p[i])) l++;
        double width = dot(edge, p[r%n]-p[i]) - dot(edge, p[l%n]-p[i])
            ;
        double height = point_line_dis(p[u%n], p[i], p[(i+1)%n]);
        area = min(area, width * height);
        perimeter = min(perimeter, (width + height) * 2);
    }
}
//最小覆盖圆，随机增量法。
void minCoverCircle(pt p[]) {
    pt o = p[0]; double r = 0;
    for (int i = 0;i < n;++i)
        if (dblcmp(dis(o,p[i]) - r) > 0) {
            o = p[i],r = 0;
            for (int j = 0;j < i;++j)
                if (dblcmp(dis(o,p[j]) - r) > 0) {
                    o = (p[i] + p[j]) * 0.5;
                    r = dis(p[i],p[j]) * 0.5;
                    for (int k = 0;k < j;++k)
                        if (dblcmp(dis(o,p[k]) - r) > 0) {
                            o = circumcenter(p[i],p[j],p[k]); //三角形
                                外心
                            r = dis(p[i],o);
                        }
                }
        }
}
// 圆和简单多边形的交:
// 直接保留所有的点，交点和原来多边形的点，然后枚举保留下的点间的线段的中
    点与圆心距离判断是圆外两点还是圆内两点。
//（没有两个点穿过圆，因为所有交点都被保留了。。）
const int V = 100;
int n;
pt p[V], s[V * 4];
pt o(0, 0);
double R;
bool on_line(pt o,pt a,pt b) {
    return dot(o-a,b-a) * dot(o-b,b-a) < -eps;
}
void add_point(pt &a,pt &b,double r,pt s[],int &top) {
    s[top++] = a;
    pt p1, p2;
    double D = point_line_dis(o, a, b);
    if (D < r+eps) {
        intersection_line_circle(o, R, a, b, p1, p2);
        if (dis(a, p1) > dis(a, p2)) swap(p1, p2);
        if (on_line(p1, a, b)) s[top++] = p1;
        if (D < R-eps && on_line(p2, a, b))
            s[top++] = p2;
    }
}
double angle(pt a,pt b) {
    double ang = b.ang() - a.ang();
    if (ang > PI) ang -= 2*PI;
    if (ang < -PI) ang += 2*PI;
    return ang;
}
double area(pt &a,pt &b) {
```

```
    if (dis(o, (a+b)/2.0) < R-eps)
        return det(a, b) * 0.5;
    else
        return angle(a, b);
}
bool intersection_circle_polygon(pt p[], int n) {
    if (!iscounter(p))
        reverse(p, p+n);
    for (int i=0;i<n;i++)
        add_point(p[i], p[i+1], R, s, top);
    s[top] = s[0];
    double ret = 0;
    for (int i=0;i<top;i++)
        ret += area(s[i], s[i+1]);
    printf("%.2f\n",fabs(ret));
}
/*
NlogN 半平面交，(该版本还没加排序，因为给的直线本身有序)
sgn332
    1. 将直线用极角（斜率）排序。(nlogn边保持一致右转的顺序即可，起点无所谓,)
    2. 用双端队列维护交出来的凸包。
    3. 维护完凸包后，再进行的两个循环不懂。while
    4. 用减掉重复的点（这样判是否交集为空就很方便了，不要用面积，面积判精度较低，可能本来不为空都变成空了）unique
*/
struct Line {
    pt a, b;
    double k, c;
    Line() {}
    Line(const pt& _a,const pt& _b) : a(_a), b(_b) {}
    void toEqu() {
        k = atan2(b.y-a.y, b.x-a.x);
        c = det(a, b) / dis(a, b);
        //k = atan2(a.y - b.y, a.x - b.x);
        //if (sgn(a.x-b.x) != 0) c = det(a, b) / fabs(a.x - b.x);
        //else c = det(a, b) / fabs(a.y - b.y);
    }
    bool operator < (const Line& a) const {
        return sgn(k-a.k) < 0 || (sgn(k-a.k) == 0 && c < a.c);
    }
};
bool parallel(Line l, Line r) {
    return sgn(det(l.b-l.a, r.b-r.a)) == 0;
}
void HalfPlaneIntersection(Line line[], int n, pt poly[], int& cnt) {
    cnt = 0;
    if (n <= 2) return;
    for (int i = 0; i < n; ++i)
        line[i].toEqu();
    sort(line, line+n);
    int now = 1;
    for (int i = 1; i < n; ++i)
    if (sgn(line[i].k - line[i-1].k) != 0) {
        line[now++] = line[i];
    }
    n = now;
    int h = 0, t = 0;
    Q[t++] = line[0];
    Q[t++] = line[1];
    for (int i = 2; i < n; ++i) {
        if (parallel(Q[h], Q[h+1]) || parallel(Q[t-2],Q[t-1])) return;
        pt &a = line[i].a, &b = line[i].b;
        while (h < t-1 && sgn(cpr(a, b, cross_point(Q[t-2], Q[t-1]))) < 0) t--;
        while (h < t-1 && sgn(cpr(a, b, cross_point(Q[h], Q[h+1]))) < 0) h++;
        Q[t++] = line[i];
    }
    while (h < t-1 && sgn(cpr(Q[h].a, Q[h].b, cross_point(Q[t-2], Q[t-1]))) < 0) t--;
    while (h < t-1 && sgn(cpr(Q[t].a, Q[t].b, cross_point(Q[h], Q[h+1]))) < 0) h++;
    for (int i = h; i < t-1; ++i)
        poly[cnt++] = cross_point(Q[i], Q[i+1]);
    if (h < t)
        poly[cnt++] = cross_point(Q[h], Q[t-1]);
    cnt = unique(poly, poly + cnt) - poly;
}
/* 将三维点集投影到一个平面上，并求投影面积方法：将目标平面旋转到平面，XOY（旋转即平面法向量
norm 绕 det(norm,Z)轴三维旋转 至轴，注意已经就是轴或ZnormZ-轴的情况）Z然后直接求面积，注意判断点在投影点上方
 */
int main() {
    double a, b, c, d;
    while (cin >> a >> b >> c >> d && !(a == 0 && b == 0 && c == 0 && d == 0)) {
        cin >> n;
        p.resize(n);
        REP(i, n)
            p[i].read();
        o.read();
        plane.norm = pt3(a, b, c);
        if (sgn(a) != 0) {
            plane.p0 = pt3(d/a, 0, 0);
        } else if (sgn(b) != 0) {
            plane.p0 = pt3(0, d/b, 0);
        } else if (sgn(c) != 0) {
            plane.p0 = pt3(0, 0, d/c);
        }
        if (((o - plane.p0)^plane.norm) < 0) {
            plane.norm = -plane.norm;
        }
        REP(i, n)
            p[i] = p[i] - plane.p0;
        o = o - plane.p0;

        pt3 z_axis = pt3(0,0,1);
        pt3 axis = plane.norm * z_axis;

        bool already_z_axis = false;
        if (sgn(vlen(axis)) == 0) {
            if ((plane.norm^z_axis) < 0) {
                axis = pt3(1.0, 0.0, 0.0);
```

```
            } else {
                already_z_axis = true;
            }
        }
        double ang = -Angle(plane.norm, z_axis);
        if (!already_z_axis) {
            REP(i, n)
                p[i] = rotate(p[i], axis, ang);
            o = rotate(o, axis, ang);
        }
        bool has_upper = false, has_lower = false;
        s.clear();
        REP(i, n) {
            if (sgn(p[i].z - o.z) >= 0) {
                has_upper = true;
            } else {
                has_lower = true;
                pt tmp;
                double ratio = (o.z-p[i].z) / o.z;
                tmp.x = o.x + (p[i].x-o.x) / ratio;
                tmp.y = o.y + (p[i].y-o.y) / ratio;
                s.push_back(tmp);
            }
        }
        if (has_upper) {
            if (!has_lower)
                puts("0.00");
            else
                puts("Infi");
        } else {
            int top = 0;
            convex_hull(s, s.size(), convex, top);
            printf("%.2f\n", area(convex, top));
        }
    }
}

//SweepLine
bool cmp(const Point &a, const Point &b) {
  bool aup = (a.y > 0 || a.y == 0 && a.x > 0);
  bool bup = (b.y > 0 || b.y == 0 && b.x > 0);
  if (aup ^ bup) return bup;
  return det(a, b) > 0;
}

Point p[MAXN];
Point pSum[MAXN];

void solve() {
  int n;
  cin >> n;
  REP(i, n) p[i].read();
  if (n < 4) {
    cout << 0 << endl;
    return;
  }
  long long area = 0, bigArea = 0;
  REP(o, n) {
    vector<Point> s;
    REP(i, n) if (i != o) {
      s.PB(p[i] - p[o]);
    }
    sort(ALL(s), cmp);

    REP(i, n - 1) s.PB(s[i]);
    pSum[0] = Point(0, 0);
    for (int i = 1; i <= s.size(); ++i) {
      pSum[i] = pSum[i - 1] + s[i - 1];
    }
    int left = 0;
    while (left + 1 < n - 1 && det(s[0], s[left + 1]) > 0) {
      ++left;
    }

    int right = left + 1;
    Point lsum, rsum, asum;
    for (int i = 1, r = right; i <= left; ++i) {
      while (r < n - 1 && det(s[i], s[r]) > 0) ++r;
      lsum = lsum + s[i] * (n - 1 - r);
      rsum = rsum + pSum[n - 1] - pSum[r];
      asum = asum + s[i];
    }

    for (int i = 0; i < n - 1; ++i) {
      area = (area + det(s[i], lsum)) % MOD;
      area = (area + det(rsum, s[i])) % MOD;
      bigArea = (bigArea + det(s[i], asum * (n + i - 1 - right))) %
          MOD;
      if (i + 1 == n - 1) break;

      asum = asum - s[i + 1];
      //=号很重要!
      while (right < n + i && det(s[i + 1], s[right]) >= 0) {
        asum = asum + s[right];
        ++right;
      }

      if (left != i) lsum = lsum - s[i + 1] * (n - 1 + i - right);
      if (left != i) lsum = lsum + pSum[left + 1] - pSum[i + 2];
      if (left != i) rsum = rsum - pSum[n + i - 1] + pSum[right];
      if (left != i) rsum = rsum + s[i] * (left - i - 1);

      left = right - 1;
    }

     // cout << "area = " << area << " bigArea = " << bigArea << endl;
  }

  //bigArea = 2 * s1 + 1 * s2
  //area = 4 * s1 + 1 * s2

  long long s1 = (area - bigArea) * powMod(2, MOD - 2) % MOD;
  long long s2 = (bigArea - 2 * s1) % MOD;

  long long ans = (s1 + s2) % MOD;
  ans = (ans + MOD) % MOD;

  cout << ans << endl;
```

```
}
```

6. 常用公式集

Theorem 1. Dilworth 定理

偏序集的两个定理：

定理 1 令（X,≤）是一个有限偏序集，并令 r 是其最大链的大小。则 X 可以被划分成 r 个但不能再少的反链。

其对偶定理称为 Dilworth 定理：

定理 2 令（X,≤）是一个有限偏序集，并令 m 是反链的最大的大小。则 X 可以被划分成 m 个但不能再少的链。

说白了就是链的最少划分数 = 反链的最长长度

Theorem 2. 生成树计数

$sigma(C(Blocks-2,d1-1,d2-1..)*v1^{d1}*v2^{d2}*...) = v1v2v3..*(v1+v2+v3+...)^{Blocks-2}$

Theorem 3. MatrixTree 定理

无向图 G 的生成树个数等于度数矩阵减去邻接矩阵的任意 N-1 阶行列式。

Theorem 4. Prufer 编码

把一棵 n 个节点并且带编号的无向树与一个 n-2 长度的数组建立双射。规则如下：

1. 将树中与编号最小的叶节点相连的节点编号加入数组。
2. 删去编号最小的叶节点。
3. 重复第 1 个操作。直到树中只剩两个节点结束操作。

Theorem 5. 组合数求模

$n! = [1*2*4*5*7*8*\cdots*16*17*19]*(3*6*9*12*15*18) = [1*2*4*5*7*8*\cdots*16*17*19]*3^6(1*2*3*4*5*6)$

Theorem 6. 最长反链构造

左端点在覆盖集中且右端点不在的节点集合。

Theorem 7. N 皇后构造

n*n 的棋盘，放 n 个皇后，互不攻击

一、当 n%6!=2 或 n%6!=3 时，有一个解为：

2,4,6,8,...,n,1,3,5,7,...,n-1 (n 为偶数)

2,4,6,8,...,n-1,1,3,5,7,...,n (n 为奇数)

（上面序列第 i 个数为 ai，表示在第 i 行 ai 列放一个皇后；... 省略的序列中，相邻两数以 2 递增。下同）

二、当 n%6==2 或 n%6==3 时，（当 n 为偶数,k=n/2；当 n 为奇数,k=(n-1)/2)

k,k+2,k+4,...,n,1,2,4,...,k-2,k+3,k+5,...,n-1,1,3,5,...,k+1 (k 为偶数,n 为偶数)

k,k+2,k+4,...,n-1,2,4,...,k-2,k+3,k+5,...,n-2,1,3,5,...,k+1,n (k 为偶数,n 为奇数)

k,k+2,k+4,...,n-1,1,3,5,...,k-2,k+3,...,n,2,4,...,k+1 (k 为奇数,n 为偶数)

k,k+2,k+4,...,n-2,1,3,5,...,k-2,k+3,...,n-1,2,4,...,k+1,n (k 为奇数,n 为奇数)

Theorem 8. 环形计数

$\frac{1}{n}\sum_{n|x} F(\frac{n}{x})\varphi(x)$

Theorem 9. 特殊计数序列

1.Catalan 数

$$\frac{1}{n+1}\binom{2n}{n}$$

2.Catalan 数，p * q 的矩阵 ($p \geq q$)

$$\frac{p-q+1}{q+1}\binom{p+q}{q}$$

3. 一般形式 Catalan 数，对角线向上平移 k 格

$$\binom{n+m}{n} - \binom{n+m}{n+k+1}$$

4.Narayana 数

Narayana numbers N(n, k), is the number of expressions containing n pairs of parentheses which are correctly matched and which contain k distinct nestings. For instance, N(4, 2) = 6 as with four pairs of parentheses six sequences can be created which each contain two times the sub-pattern '()':

$$()((()))(())(())((()))(((())))((()))(())((()))()$$

$$N(n,k) = \frac{1}{n}\binom{n}{k}\binom{n}{k-1}$$

Theorem 10. 四边形不等式

对于$w(x,i+1)+w(x+1,i) \geq w(x,i)+w(x+1,i+1)$

有$pred(i-1,j) \leq pred(i,j) \leq pred(i,j+1)$

亦可用性质直接证明

Theorem 11. $\sum i^k$系数递推公式

```
a[0][1]=fraction(1,1);
for (i=1;i<=30;i++){
    for (j=1;j<=i+1;j++) a[i][j]=C[i+1][j];
    for (k=0;k<i;k++)
        for (j=0;j<=i+1;j++)
            a[i][j]=a[i][j]-C[i+1][k]*a[k][j];
    for (j=1;j<=i+1;j++) a[i][j]=a[i][j]/C[i+1][1];
}
```