# "Basic Algorithms" for Our ICPC

Edited by Veracity

version 1.1 updated on 17/10/2019

## Introduction

≡ ω ≡ ¡ Hola !

As one of the teams participating in ICPC, team Veracity considers editing a series of practical algorithms instrumental. Our captain edited *Basic Algorithms for Entrance Examination of Provincial Delegation* for himself and the name stuck. One thousand people can tell one thousand kinds of ICPC, that is, these pieces of notes (let's call it "BAI" briefly) are edited for ourselves initially. What should be pointed out is, part of, maybe most of BAI, is written in Chinese up to present. If you have something to do with BAI, please read license carefully. Please contact with us if it is necessary.

Teammates are shown following:

TooYoungTooSimp, lj020, Nonad

此致

敬礼!

### Abstract

to be filled...

## Ad Hoc

### Fastdiv

开优化的话，普通的mod会被优化成同样的东西）

```
struct FastDiv {
  FastDiv() {}
  FastDiv(u64 n) : m(n) {
    s = std::__lg(n - 1);
    x = ((__uint128_t(1) << (s + 64)) + n - 1) / n;
  }
  friend u64 operator / (u64 n, FastDiv d) {
    return (__uint128_t(n) * d.x >> d.s) >> 64;
  }
  friend u64 operator % (u64 n, FastDiv d) { return n - n / d * d.m; }
  u64 m, x; int s;
} mod;
```

## Fastmul

加加加

```
long long qm(long long a,long long b,long long mod)
{
    long long ans=1;
    for(;b;b>>=1,a=(a+a)%mod)if(b&1)ans=(ans+a)%mod;
    return ans;
}
```

## Fastinput

奇妙能力

```
inline char nc() {
    static char buf[100000], * p1 = buf, * p2 = buf;
    return p1 == p2 && (p2 = (p1 = buf) + fread(buf, 1, 100000, stdin), p1 ==
p2) ? EOF : *p1++;
}
inline int rd() {
    char ch = nc(); int sum = 0;
    while (!(ch >= '0' && ch <= '9'))ch = nc();
    while (ch >= '0' && ch <= '9')sum = sum * 10 + ch - 48, ch = nc();
    return sum;
}
//非负数
```

HDU6396

TYTS's multifunctional

```
inline char getchar(int)
{
```

```
    static char buf[64 << 20], *S = buf, *T = buf;
    if (S == T) T = fread(S = buf, 1, 64 << 20, stdin) + S;
    return S == T ? EOF : *S++;
}
template <typename T>
inline typename enable_if<is_integral<T>::value>::type read(T &x)
{
    int ch = x = 0, f = 1;
    while (!isdigit(ch = getchar(0))) if(ch == '-') f = -1;
    for (; isdigit(ch); ch = getchar(0)) x = x * 10 + ch - '0';
    x *= f;
}
```

# Data Structures

## 第k大元素

hdu4006

## 块状链表

hdu5193

## 堆heap

Here will be placed an article about pb_ds

## 跳跃表Skip List

hdu5266

## 线段树Segment Tree

一维线段树hdu1540

二维线段树poj1195

最大子段和hdu6638

## 周长并

hdu1828

## LCA

hdu2586

## 莫队算法

hdu5145

## 树状数组Binary Index Tree

hdu2852

N维树状数组hdu3584

## 平衡树二叉树balanced binary tree

poj2828

## 二叉搜索树Binary Search Tree

hdu3791

## Treap树

hdu4099

poj2985

静动态建树

## 伸展树splay tree

hdu1890/3726/4453

poj2892

## 笛卡尔树

hdu4095

## 划分树

hdu4417 查询区间第k大

## 表达式树

hdu1805

## RMQ range minimum/maximum query

hdu3183

## 左偏堆

hdu1512

## 可并堆

hdu1512

## 主席树

zoj2112

查询区间多少个不同的数

静态区间第k大poj2104

树上路径点权第k大

动态第k大

## 树链剖分

hdu3966

## KD树 K-demension tree

hdu4347

k近邻、求出最近的k个点

## 替罪羊树 ScapeGoat Tree

poj1442

### 动态KD树

hdu5992

结合了KD树和替罪羊树

### 树套树

hdu5412

## Complete Search(Iterative/Recursive)

### 子集生成

hdu1584

### 三分搜索Ternary Search

hdu2899

### 双向广搜

## Divide and Conquer

## Greedy

## Dynamic Programming

# Graph

## Tarjan（有向图强连通分量）

> **定义**：在有向图 G 中，如果两个顶点 u, v 间存在一条路径 u 到 v 的路径且也存在一条 v 到 u 的路径，则称这两个顶点 u;,v 是**强连通的 (strongly connected)**。如果有向图 G 的每两个顶点都强连通，称 G 是一个强连通图。有向非强连通图的极大强连通子图，称为**强连通分量 (strongly connected components)**。 若将有向图中的强连通分量都缩为一个点，则原图会形成一个 DAG（有向无环图）
>
> **极大强连通子图**： G 是一个极大强连通子图当且仅当 G 是一个强连通子图且不存在另一个强连通子图 G' 使得 G 是 G' 的真子集

> 定义 dfn(u) 为结点 u 搜索的次序编号，给出函数 low(u) 使得
> low(u) = min
> {
> dfn(u),
> low(v), (u, v) 为树枝边， u 为 v 的父结点
> dfn(v), (u, v) 为后向边或指向栈中结点的横叉边
> }
> 当结点 u 的搜索过程结束后，若 dfn(u) = low(u)，则以 u 为根的搜索子树上所有还在栈中的结点是一个强连通分量。

```
void tarjan(int u)
{
    dfn[u] = low[u] = ++idx;
    st[top++] = u;
    for (Edge cur : G[u])
        if (!dfn[cur.to])
            tarjan(cur.to),
            low[u] = min(low[u], low[cur.to]);
        else if (!scc[cur.to])
            low[u] = min(low[u], dfn[cur.to]);
    if (dfn[u] == low[u] && ++cnt)
        do scc[st[--top]] = cnt;
        while (st[top] != u);
 }
```

#### POJ2186/BZOJ1051

有关系A→B与B→C，则有关系A→C，求全图中有多少点被其它所有点指向

> tarjan强连通分量求缩点重构图，出度为0的点若只有一个则输出其代表强连通分量的大小，否则无解

```
#include <cstdio>
inline int min(int a, int b) { return a < b ? a : b; }
int head[10010], next[50010], to[50010], ecnt;
int dfn[10010], low[10010], stk[10010], scc[10010], top, idx, scccnt;
bool instk[10010];
int deg[10010];
inline void addEdge(int f, int t)
```

```
{
    ecnt++;
    next[ecnt] = head[f];
    head[f] = ecnt;
    to[ecnt] = t;
}
void tarjan(int x)
{
    dfn[x] = low[x] = ++idx;
    instk[stk[top++] = x] = true;
    for (int cur = head[x]; cur; cur = next[cur])
        if (!dfn[to[cur]])
            tarjan(to[cur]), low[x] = min(low[x], low[to[cur]]);
        else if (instk[to[cur]])
            low[x] = min(low[x], dfn[to[cur]]);
    if (dfn[x] == low[x])
    {
        scccnt++;
         do
        {
            top--;
            scc[stk[top]] = scccnt;
            instk[stk[top]] = false;
        } while (stk[top] != x);
    }
}
int main()
{
    int n, m;
    scanf("%d%d", &n, &m);
    for (int i = 0, x, y; i < m; i++)
    {
        scanf("%d%d", &x, &y);
        addEdge(x, y);
    }
    for (int i = 1; i <= n; i++)
        if (!dfn[i])
            tarjan(i);
    for (int i = 1; i <= n; i++)
        for (int cur = head[i]; cur; cur = next[cur])
            if (scc[i] != scc[to[cur]])
                deg[scc[i]]++;
    int zcnt = 0, id = 0;
    for (int i = 1; i <= scccnt; i++)
        if (deg[i] == 0)
            zcnt++, id = i;
    if (zcnt != 1)
        putchar('0');
    else
    {
        int ans = 0;
        for (int i = 1; i <= n; i++)
            if (scc[i] == id)
                ans++;
        printf("%d", ans);
    }
    return 0;
}
```

### POJ1236

> 强联通分量缩点求出度为0和入度为0的分量个数

```
#include <cstdio>
inline int min(int a, int b) { return a < b ? a : b; }
const int maxn = 110, maxm = 10100;
int head[maxn], next[maxm], to[maxm], ecnt, f[maxn], g[maxn];
inline void addEdge(int f, int t)
{
    ecnt++;
    next[ecnt] = head[f];
    head[f] = ecnt;
    to[ecnt] = t;
 }
 int dfn[maxn], low[maxn], stk[maxn], scc[maxn], scccnt, top, idx;
 void tarjan(int x)
 {
    dfn[x] = low[x] = ++idx;
    stk[top++] = x;
    for (int i = head[x]; i; i = next[i])
        if (!dfn[to[i]])
            tarjan(to[i]), low[x] = min(low[x], low[to[i]]);
        else if (!scc[to[i]])
            low[x] = min(low[x], dfn[to[i]]);
    if (dfn[x] == low[x])
    {
        scccnt++;
        do
            scc[stk[--top]] = scccnt;
        while (stk[top] != x);
    }
 }
 int main()
 {
    int n;
    scanf("%d", &n);
    for (int i = 1, x; i <= n; i++)
        for (scanf("%d", &x); x; scanf("%d", &x))
            addEdge(i, x);
    for (int i = 1; i <= n; i++)
        if (!dfn[i]) tarjan(i);
            for (int i = 1; i <= n; i++)
                for (int j = head[i]; j; j = next[j])
                    if (scc[i] != scc[to[j]])
                        f[scc[i]]++, g[scc[to[j]]]++;
    int ans1 = 0, ans2 = 0;
    if (scccnt == 1)
        printf("1\n0");
    else
    {
        for (int i = 1; i <= scccnt; i++)
            ans1 += f[i] == 0, ans2 += g[i] == 0;
        printf("%d\n%d", ans2, ans1 > ans2 ? ans1 : ans2);
    }
```

```
    return 0;
}
```

## 图的割点、桥与双连通分量

> **定义**:点连通度与边连通度 在一个无向连通图中，如果有一个顶点集合 V，删除顶点集合 V，以及与 V 中顶点相连（至少有一端在 V 中）的所有边后，原图不连通，就称这个点集 V 为**割点集合**。
>
> 一个图的点连通度的定义为：最小割点集合中的顶点数。
>
> 类似的，如果有一个边集合，删除这个边集合以后，原图不连通，就称这个点集为**割边集合**。
>
> **双连通图、割点与桥**
>
> 如果一个无向连通图的点连通度大于 1，则称该图是**点双连通的 (point biconnected)**，简称双连通或重连通。一个图有割点，当且仅当这个图的点连通度为 1，则割点集合的唯一元素被称为割点 (cut point)，又叫关节点 (articulation point)。一个图可能有多个割点。
>
> 如果一个无向连通图的边连通度大于 1，则称该图是**边双连通的 (edge biconnected)**，简称双连通或重连通。一个图有桥，当且仅当这个图的边连通度为 1，则割边集合的唯一元素被称为**桥 (bridge)**，又叫关节边(articulation edge)。一个图可能有多个桥。
>
> 可以看出，点双连通与边双连通都可以简称为双连通，它们之间是有着某种联系的，下文中提到的双连通，均既可指点双连通，又可指边双连通。（但这并不意味着它们等价）
>
> 双连通分量（分支）：在图 G 的所有子图 G' 中，如果 G' 是双连通的，则称 G' 为双连通子图。如果一个
>
> 双连通子图 G' 它不是任何一个双连通子图的真子集，则 G' 为极大双连通子图。双连通分量 (biconnected component)，或重连通分量，就是图的极大双连通子图。特殊的，点双连通分量又叫做块。

> Tarjan 算法 给出函数 low(u) 使得
> low(u) = min
> {
> dfn(u),
> low(v), (u, v) 为树枝边 (父子边)
> dfn(v) (u, v) 为后向边 (返祖边) 等价于 dfn(v) < dfn(u) 且 v 不为 u 的父亲结点
> }

```
//tarjan - BCC
 void tarjan(int u, int p)
 {
    dfn[u] = low[u] = ++idx;
    for (int e = head[u]; e; e = next[e])
        if (!dfn[to[e]])
            tarjan(to[e], u), low[u] = min(low[u], low[to[e]]);
        else if (to[e] != p)
            low[u] = min(low[u], dfn[to[e]]);
 }
```

**POJ3177**

> 将一张有桥图通过加边变成双连通图，至少要加$\frac{leaf+1}{2}$条边

```
#include <cstdio>
inline int min(int a, int b) { return a < b ? a : b; }
int head[5010], to[20010], next[20010], ecnt, map[5010][5010];
int dfn[5010], low[5010], idx, cnt[5010];
void addEdge(int f, int t)
{
    ecnt++;
    next[ecnt] = head[f];
    head[f] = ecnt;
    to[ecnt] = t;
 }
 void tarjan(int u, int p)
 {
    dfn[u] = low[u] = ++idx;
    for (int e = head[u]; e; e = next[e])
        if (!dfn[to[e]])
            tarjan(to[e], u), low[u] = min(low[u], low[to[e]]);
        else if (to[e] != p)
            low[u] = min(low[u], dfn[to[e]]);
 }
 int main()
 {
    int n, m;
    scanf("%d%d", &n, &m);
    for (int i = 1, x, y; i <= m; i++)
    {
        scanf("%d%d", &x, &y);
        if (!map[x][y])
        {
            addEdge(x, y);
            addEdge(y, x);
            map[x][y] = map[y][x] = true;
        }
    }
    tarjan(1, 0);
    for (int i = 1; i <= n; i++)
        for (int e = head[i]; e; e = next[e])
            if (low[to[e]] != low[i])
                cnt[low[i]]++;
    int ans = 0;
```

```
    for (int i = 1; i <= n; i++)
        ans += cnt[i] == 1;
        printf("%d", (ans + 1) >> 1);
    return 0;
 }
```

### POJ1523

> 求割点与删除这个点之后有多少个连通分量

```
#include <cstdio>
#include <cctype>
#include <cstring>
#define clz(X) memset(X, 0, sizeof(X))
inline int max(int a, int b) { return a > b ? a : b; }
inline int min(int a, int b) { return a < b ? a : b; }
inline void read(int &x)
{
    int ch = x = 0;
    while (!isdigit(ch = getchar()));
    for (; isdigit(ch); ch = getchar())
    x = x * 10 + ch - '0';
}
int map[1010][1010], range;
int dfn[1010], low[1010], idx;
int son, subnet[1010];
void tarjan(int u)
{
    dfn[u] = low[u] = ++idx;
    for (int v = 1; v <= range; v++)
        if (map[u][v])
            if (!dfn[v])
            {
                tarjan(v);
                low[u] = min(low[u], low[v]);
                if (low[v] >= dfn[u])
                (u == 1 ? son : subnet[u])++;
            }
            else
                low[u] = min(low[u], dfn[v]);
}
int main(   )
{
    int x, y, T = 0;
    while (read(x), x)
    {
        clz(map), clz(dfn), clz(low), clz(subnet), son = idx = 0;
        read(y);
        map[x][y] = map[y][x] = 1;
        range = max(x, y);
        while (read(x), x)
        {
            read(y);
            map[x][y] = map[y][x] = 1;
            range = max(range, max(x, y));
        }
```

```
        printf("Network #%d\n", ++T);
        tarjan(1);
        bool flag = false;
        if (son > 1) subnet[1] = son - 1;
        for (int i = 1; i <= range; i++)
            if (subnet[i])
                printf(" SPF node %d leaves %d subnets\n", i, subnet[i] + 1),
flag = true;
        if (!flag)
            puts(" No SPF nodes");
        putchar('\n');
    }
    return 0;
}
```

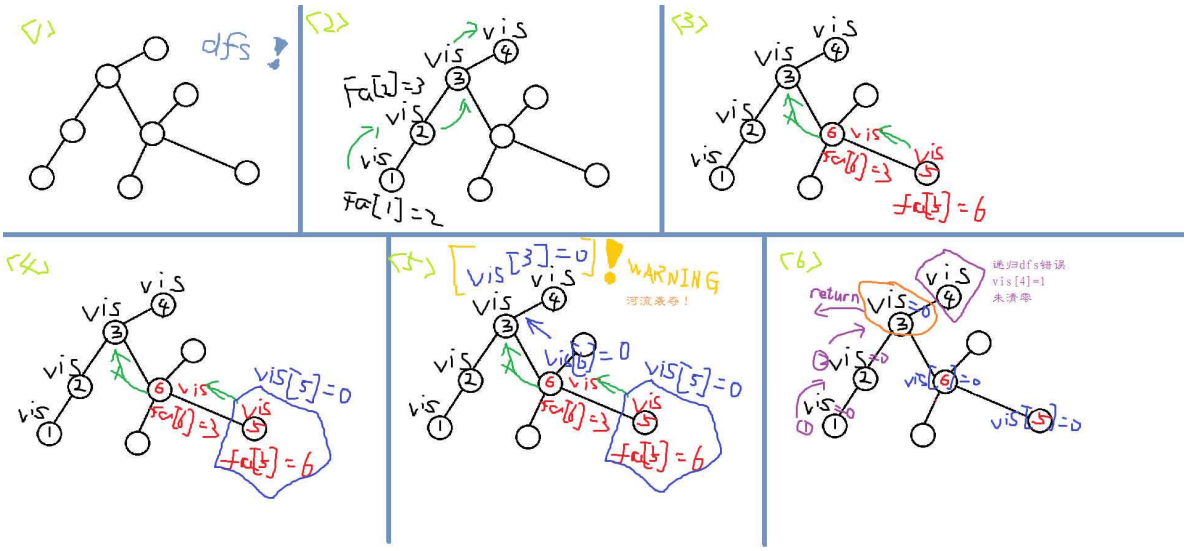
**POJ2942**

*disappeared into the sky...*

_(:3」∠)_

## 2-SAT

*190806 lj020发现图论中的河流袭夺现象*




# Mathematics

### 数论Number Theory

# GCD、LCM

## 素数判断

poj2689

## 素数生成Prime Number Generation

hdu4548

```
void pri()
{
    int co=0;
    for(int i=2;i<=n;i++){
        if(!notp[i])isp[++co]=i;
        for(int j=1;j<=co&&i*isp[j]<=n;j++){
            notp[i*isp[j]]=1;
            if(i%isp[j]==0)break;
        }
    }
}
```

## 分解质因数

hdu6287

## 欧拉定理Euler Totient Theorem

hdu1395

## 扩展欧几里得Extended Euclid

hdu1576

> 一个崭新的py板子

```
def ex_gcd(a: int, b: int):
    if b == 0:
        return 1, 0, a
    x, y, g = ex_gcd(b, a % b)
    return y, x - int(a / b) * y, g
```

## 逆元Inverse

## 费马小定理Fermat's Theorem

hdu4704

## 随机素数测试和大数分解

poj1811

## 高斯消元Gaussian elimination

hdu5755

bzoj1013

## 模线性方程组

hdu3797

## 佩尔方程Pell's equation

hdu3292

## 整数拆分Integer Factorization

hdu4651/4658

## 求$A^B$的约数之和对MOD取模

poj1845

## BSGS算法Baby-Step Giant-Step

北上广深x算法

## 斐波那契数列取模

hdu1021

（循环节）

## 原根

hdu4992

## 快速数论变换(FNT/NTT)

hdu4656卷积取模

## 线性丢番图方程Linear Diophantine Equations

## 模运算Modulus Arithmetic

## 卢卡斯定理Lucas Theorem

hdu5226

## 中国剩余定理Chinese Remainder Theorem

hdu3430

## 威尔逊定理Wilson Theorem

hdu5391

## 米勒-罗宾随机素性测试Miller-Rabin Primality Testing

hdu4910

## 完全数Perfect Numbers

hdu 2683

## 哥德巴赫猜想Goldbach Conjecture

hdu1397

## 连分数Continued fraction

hdu4188

### 概率Probability

## 基本概率和条件概率Basic Probability and Conditional Probability

hdu1204

## 随机变量Random variables

hdu1145

## 概率生成函数Probability Generating Functions

## 期望Expectation

hdu5984

代价a，成功概率p，代价期望a/p

## 概率分布Probability Distribution

poj3716

Binomial, Poisson, Normal, Bernoulli

### 组合数学Counting

Special numbers: Stirling, Fibonacci, Catalan, Eulerian, Harmonic, Bernoulli

## 错排公式

n个不同元素排列的方法数为n!，那么恰有k个元素错排的方法数为$\binom{k}{n} D_k$

有$\sum_{k=0}^{n} \binom{k}{n} D_k = n!$

根据二项式反演得到$D_n = n! \sum_{k=0}^{n} (-1)^{n-k} \frac{1}{(n-k)!}$

## 容斥原理Inclusion Exclusion

hdu2204

## 鸽巢原理（抽屉原理）Pigeonhole principle

hdu1205

## 乘法原理

hdu5525

## Stirling数（与Stirling公式）

hdu4372

> 第一类斯特林数表示将n个不同元素构成m个圆排列的数目，根据正负有有无符号之分

递推式$s(n+1,2) = s(n,1) + s(n,2) \cdot n$

边界条件$\binom{0}{0} = 1$

**性质**:

1. $s(n,1) = (n-1)!$
2. $s(n,2) = (n-1)! \times \sum_{i=1}^{n-1} \frac{1}{i}$
3. $\sum_{i=0}^{n} s(n,k) = n!$

> 第二类斯特林数表示将n个数分成k组，组内无序，每组没有区别

递推式$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} * k$

边界条件同上

通项公式：$S(n,m) = \frac{1}{m!}\sum_{k=0}^{m}(-1)^k\binom{k}{m}(m-k)^n$

卷积形式：$\binom{n}{m} = \sum_{k=0}^{m}\frac{(-1)^k}{k!}\frac{(m-k)^n}{(m-k)!}$可用FFT在$O(mlog_2m)$的时间算出$\binom{n}{1}\dots\binom{n}{m}$

转化幂：$x^n = \sum_{k=1}^{n}\binom{n}{k}x^k$

两类斯特林数可互相转化：$\sum_{k=0}^{n}S(n,k)s(k,m) = \sum_{k=0}^{n}s(n,k)S(k,m)$

斯特林公式hdu1018

取阶乘近似值$n! \approx \sqrt{2\pi n}(\frac{n}{e})^n(1+\frac{1}{12n}+\frac{1}{288n^2}+o(\frac{1}{n^3}))$

求n较大时$\prod_{i=1}^{n}(2i-1)$的位数

可变换为$\frac{(2n)!}{2\times4\times6\times\dots\times(2n)} = \frac{(2n)!}{2^n n!}$

## Catalan数

hdu5673

## 斐波那契数列

hdu1316

## Polya计数Polya Counting

hdu3547

## 莫比乌斯反演Mobius inversion

hdu5382

## 母函数Generating function

hdu2082/2065 普通型、指数型

## 调和级数Harmonic Number

poj1003

## 幻方Magic Square

hdu3927

## N皇后

hdu2563

### 线性代数Linear Algebra

## 矩阵变换Matrix Transformations

hdu5671

## 矩阵的行列式、秩和逆Determinant,Rank and Inverse Of Matrix

hdu5852

## 线性方程组的求解Solving System Of Linear Equations

## 矩阵求幂Matrix Exponentiation

hdu1757

## 特征值和特征向量Eigenvalues And Eigen vector

## Roots of a polynomial

hdu1296

$$f(x) = (x-a)g(x) + r$$

$$\begin{aligned}
f(x) =& a_0 + a_1x^1 + a_2x^2 + \ldots + a_nx^n \\
g(x) =& b_0 + b_1x^1 + \ldots + b_{n-1}x^{n-1} \\
(x-a)g(x) =& b_0x^1 + b_1x^2 + \ldots + b_{n-1}x^n \\
& - (ab_0 + ab_1x^1 + \ldots + ab_{n-1}x^{n-1}) \\
=& -ab_0 + (b_0 - ab_1)x^1 + \ldots + (b_{n-2} - ab_{n-1}x^{n-1}) + b_{n-1}x^n \\
b_{n-1} =& a_n \\
b_{n-2} - ab_{n-1} =& a_{n-1} \\
& \vdots \\
b_0 - ab_1 =& a_1 \\
r - ab_0 =& a_0
\end{aligned}$$

## 拉格朗日插值Lagrange Interpolation

hdu6253

## min_25筛

积性函数前缀和

## 线性基

hdu3949

### 组合游戏（博弈论）Game Theory

## 尼姆游戏Nim game

hdu2176

## P-position、N-position

## 图游戏与SG函数sprague-Grundy

hdu3023

## Hackenbush游戏

hdu3197

## 威佐夫游戏Wythoff's game

hdu2177

### 群论Group Theory

## 伯恩赛德引理Burnside's lemma

hdu4633

## 波利亚定理Polya's Theorem

hdu3547

## 拉格朗日定理

### 计算方法

## 快速傅里叶变换(FFT)

hdu4609

## 迭代法

hdu3809

## 三分法

hdu2899

## 定积分计算

hdu1071

## 自适应simpson积分

hdu1724

## 线性基（≈基底）

(hdu 6579)

```
struct LB
{
    ll p[63];
    void init() { memset(p, 0, sizeof(p)); }
    bool ins(ll x)
    {
        for (int i = 62; i >= 0; i--)
            if (x >> i & 1)
            {
                if (!p[i])
                {
                    p[i] = x;
                    break;
                }
                x ^= p[i];
            }
        return x;
    }
};
```

求交...

求并...

## 十进制矩阵快速幂

2019nowcoder多校B

超长二阶线性递推，$x_n = a * x_{n-1} + b * x_{n-2}$

```
#include <bits/stdc++.h>
using namespace std;
#define CRP(t, x) const t &x
#define OPL(t, x) bool operator<(CRP(t, x)) const
#define FIL(x, v) memset(x, v, sizeof(x))
```

```
#define CLR(x) FIL(x, 0)
#define NE1(x) FIL(x, -1)
#define INF(x) FIL(x, 0x3f)
typedef long long ll, i64;
ll mod;
struct Mat
{
    ll a[2][2];
    void clear()
    {
        ll *const p = (ll *)a;
        *p = 0;
        *(p + 1) = 0;
        *(p + 2) = 0;
        *(p + 3) = 0;
    }
    void init()
    {
        ll *const p = (ll *)a;
        *p = 1;
        *(p + 1) = 0;
        *(p + 2) = 0;
        *(p + 3) = 1;
    }
};
Mat mat_mul(CRP(Mat, lhs), CRP(Mat, rhs))
{
    Mat v;
    v.clear();
    auto a = lhs.a, b = rhs.a;
    auto c = v.a;
    c[0][0] = (a[0][0] * b[0][0] + a[0][1] * b[1][0]) % mod;
    c[0][1] = (a[0][0] * b[0][1] + a[0][1] * b[1][1]) % mod;
    c[1][0] = (a[1][0] * b[0][0] + a[1][1] * b[1][0]) % mod;
    c[1][1] = (a[1][0] * b[0][1] + a[1][1] * b[1][1]) % mod;
    return v;
}
Mat mat_fpow(Mat a, ll b)
{
    Mat r;
    r.init();
    for (; b; b >>= 1, a = mat_mul(a, a))
        if (b & 1)
            r = mat_mul(r, a);
    return r;
}
Mat mat_fpow_10(Mat a, int *rb, int len)
{
    Mat r;
    r.init();
    for (int i = 0; i < len; i++, a = mat_fpow(a, 10))
        r = mat_mul(r, mat_fpow(a, rb[i]));
    return r;
}
const int N = 1e6 + 50;
char s[N];
int res[N];
int main()
```

```
{
    ll x0, x1, a, b;
    scanf("%lld%lld%lld%lld%s%lld", &x0, &x1, &a, &b, s, &mod);
    int len = strlen(s);
    for (int i = 0; i < len; i++) res[len - i - 1] = s[i] - '0';
    Mat ini;
    ini.a[0][0] = (a * x1 + b * x0) % mod;
    ini.a[1][0] = x1;
    ini.a[0][1] = x1;
    ini.a[1][1] = x0;
    Mat fac;
    fac.a[0][0] = a;
    fac.a[1][0] = b;
    fac.a[0][1] = 1;
    fac.a[1][1] = 0;
    auto re = mat_mul(ini, mat_fpow_10(fac, res, len));
    printf("%lld", re.a[1][1]);
    return 0;
}
```

## 外观数列（康威常数）

hdu4148

首项为1，之后的每一项都是对前一项的描述

1,11,21,1211,111221,312211.....

当项数趋近无穷大时，相邻两数的长度之比接近一个固定的常数约为1.303577

4永远不出现

## 整数拆分

将自然数n拆分为若干数的和，若干数的积为m，求m的最大值

将n拆出尽可能多的3，若剩余2或0则结束，若剩余1则与一个3组合为两个2

## 阿贝尔变换（阿贝尔部分求和公式）

给定两个数列$a_n b_n$

a数列的前n项和为$S_n$，且首项为0

则有$\sum_{k=1}^{n} a_k b_k = S_n b_n + \sum_{k=1}^{n-1} S_k (b_k - b_{k+1})$

## 二项式反演

> 设F(n)和f(n)是定义在非负整数集上的两个函数，并且满足

> $F(n)=\sum_{k=0}^{n}\binom{k}{n}f(k)$
>
> 那么得到$f(n)=\sum_{k=0}^{n}(-1)^{n-k}\binom{k}{n}F(k)$

首先要知道$\sum_{k=i}^{n}(-1)^{n-k}\binom{k}{n}\binom{i}{k}=\begin{cases}0 & \text{i<n}\\ 1 & \text{i=n}\end{cases}$

## 马青公式（计算π）

$\pi=16arctan\frac{1}{5}-4arctan\frac{1}{239}$

由于$arctan(x)=x(1-\frac{x^2}{3}+\frac{x^4}{5}-\frac{x^6}{7}+\dots)$

设$m=x^2$

$g(m)=1-\frac{1}{3}m+\frac{1}{5}m^2-\frac{1}{7}m^3+\dots$

可用秦九韶算法求之，用FFT提高乘法运算效率

## 最小二乘法

$a\sum_{i=1}^{n}x_i^2+b\sum_{i=1}^{n}x_i=\sum_{i=1}^{n}x_iy_i$

$a\sum_{i=1}^{n}x_i+nb=\sum_{i=1}^{n}y_i$

联立解出a和b，解唯一。若方程数较多，可用高斯消元

## 自守数

一个k位的自然数n，若平方后得到的最后k位与原数相同，则n为自守数

$n^2\equiv n\pmod{10^k}$

一位数 1、5、6

一位数以上的自守数

1. 个位数为5，形式为$n\equiv 1\pmod{2^k},n\equiv 0\pmod{5^k}$
2. 个位数为6，形式为$n\equiv 0\pmod{2^k},n\equiv 1\pmod{5^k}$

两个位数相同的自守数之和为$10^k+1$

一个数为自守数当且仅当它为另一个自守数的后缀

n位(1位数除外)的自守数仅有两个，其位数包括前导零

一个k+1位的自然数F（k+1）可由F（k）求得

1. 对于个位数为5的自守数，求F(k)的平方，取最后k+1位，若第k+1位为零，则多取一位
2. 对于个位数为6的自守数，求F(k)的平方，取最后k+1位，把第k+1位的数用10减之代替，若第k+1位是零，则多取一位进行操作

## 法里数列（0-1内所有既约真分数))

(hdu 6584)

## Stern-Brocot树（生成0-1之间的所有真分数）

(hdu 6584)

## 艾森斯坦因判别法（判断多项式可约与否）

给定一个n次本原多项式（多项式所有系数的gcd为1）$f(x)=a_0+a_1x+a_2x^2+\ldots+a_nx^n$, 并且这是一个整系数多项式，若存在一个素数p，使得

1. $p\nmid a_n$
2. $p\mid a_i, i\in[0,n-1]$
3. $p^2\nmid a_0$

那么$f(x)$为不可约多项式

## 实系数多项式因式分解定理

每个次数不小于1的实系数多项式在实数域上都可以唯一地分解成一次因式与二次不可约因式的乘积

# String Processing

## KMP（线性时间匹配字符串）

> **字符串**： s[1...n], |s| = n
> **子串**： s[i ...j] = s[i]s[i + 1] · · · [j]
> **前缀**： pre(s;, x) = s[1 ... x], **后缀**： suf(s, x) = s[n-x + 1 ... n]
> 若 0 ≤ r ≤ |s|, pre(s, r) = suf(s, r), 就称 pre(s, r) 是 s 的 **border**。
> KMP 算法的第一步主要做这么一件事：在 O(n) 时间求出数组 next[1 ... n], 其中 next[i] 表示前缀 s[1 ... i]的最大 border 长度。于是可以知道 s 的所有 border 长度为 next[n], next[next[n]] · · ·我想这是显然的，于是在这里不加证明地给出。
> 第二步就是匹配，如果失配了就把模式串的当前位置指针 i 跳到 next[i] 处然后继续匹配，然后就好了

```
//genNext
 for (int i = 1, j = -1; i < len; i++)
 {
    while (~j && str[j + 1] != str[i]) j = next[j];
    if (str[j + 1] == str[i]) j++;
    next[i] = j;
 }
//Find
 for (int i = 0, j = -1; i < len; i++)
 {
    while (~j && t[j + 1] != s[i]) j = next[j];
```

```
    if (t[j + 1] == s[i]) j++;
    if (j == len - 1) ans++, j = next[j];
 }
```

### POJ2406

每个字符串最小周期的个数

> next数组的奇妙性质

```
#include <cstdio>
#include <cstring>
char str[1 << 20 | 1];
int next[1 << 20 | 1];
int len;
int main()
{
    next[0] = -1;
    while (scanf("%s", str))
    {
        if (str[0] == '.') break;
        len = strlen(str);
        for (int i = 0, j = -1; i < len;)
            (~j && str[i] != str[j]) ? j = next[j] : next[++i] = ++j;
        printf("%d\n", len % (len - next[len]) == 0 ? len / (len - next[len]) :
1);
    }
    return 0;
 }
```

### CF526D

> 给定一个串 TT，对它的每一个前缀能否写成 A+B+A+B+...+B+AA+B+A+B+...+B+A 的形式（kk 个 AA，k+1k+1 个 BB，均可为空串）

k个AB1个A)

```
#include <cstdio>
#include <cstring>
char s[1000010];
int next[1000010], n, k, len;
int main()
{
    scanf("%d%d%s", &n, &k, s);
    next[0] = -1;
    len = strlen(s);
    for (int i = 1; i < len; ++i)
    {
        int j;
        for (j = next[i - 1]; j != -1 && s[j + 1] != s[i]; j = next[j]);
            if (s[j + 1] == s[i]) j++;
            next[i] = j;
    }
    for (int i = 0; i < len; ++i)
```

```
    {
        int p = i + 1, q = p / (i - next[i]);
        putchar(((p % (i - next[i]) == 0) ? (q / k >= q % k ? '1' : '0') : (q /
k > q % k? '1' : '0')) );
    }
    return 0;
 }
```

## 扩展KMP（和后缀的最长公共前缀）

## 回文树

(hdu 6599)

> 1.求串S前缀0~i内本质不同回文串的个数（两个串长度不同或者长度相同且至少有一个字符不同便是本质不同）

> 2.求串S内每一个本质不同回文串出现的次数

> 3.求串S内回文串的个数（其实就是1和2结合起来）

> 4.求以下标i结尾的回文串的个数

快乐$O(n)$建树，不是n个字符串哦

**APIO2014 Palindromes**

> s 的一个子串的存在值为这个子串在 s 中出现的次数乘以这个子串的长度，求最大存在值

```
#include <bits/stdc++.h>
using namespace std;
const int N = 300010;
int last, cnt, ch[N][26], fail[N], len[N], num[N];
char s[N];
int find(int i, int x)
{
    while (s[i - len[x] - 1] != s[i]) x = fail[x];
    return x;
}
int main()
{
    long long ans = 0;
    scanf("%s", s);
    fail[0] = 1;
    fail[1] = 1;
    len[1] = -1;
    cnt = 1;
    for (int i = 0; s[i]; i++)
    {
        int j = find(i, last);
        if (ch[j][s[i] - 'a'])
            last = ch[j][s[i] - 'a'];
```

```
        else
        {
            len[last = ++cnt] = len[j] + 2;
            fail[last] = ch[find(i, fail[j])][s[i] - 'a'];
            ch[j][s[i] - 'a'] = last;
        }
        num[last]++;
    }
    for (int i = cnt; i >= 0; i--)
        ans = max(ans, 1ll * num[i] * len[i]), num[fail[i]] += num[i];
    printf("%lld\n", ans);
    return 0;
}
```

## Trie

> 字典树，也称 Trie、字母树，指的是某个字符串集合对应的有根树。树的每条边上对应有恰好一个字符，每个顶点代表从根到该节点的路径所对应的字符串（将所有经过的边上的字符按顺序连接起来）

```
 struct node
{
    node *trans[26];
    int cnt;
};
void insert(node *n, char *str)
{
    for (; *str; n = n->trans[*str - '0'])
        if (n->trans[*str - '0'] == 0)
            n->trans[*str - '0'] = new_node();
    n->cnt++;
 }
```

### POJ3630

> 若插入过程中，有某个经过的节点带有串结尾标记，则之前插入的某个串是当前串的前缀

```
#include <cstdio>
#include <cstring>
struct node
{
    node *trans[10];
    bool is_end;
} nodes[100010];
node *root;
int cnt;
node *new_node() { return &nodes[cnt++]; }
char buf[11];
bool try_insert(node *n, char *str)
 {
    if (n->is_end) return false;
```

```
    if (*str == '\0')
    {
        for (int i = 0; i < 10; i++)
            if (n->trans[i])
                return false;
         n->is_end = true;
         return true;
    }
    if (n->trans[*str - '0'] == 0) n->trans[*str - '0'] = new_node();
    return try_insert(n->trans[*str - '0'], str + 1);
}
int main()
{
    int t, n;
    scanf("%d", &t);
    while (t--)
    {
        memset(nodes, 0, sizeof(nodes));
        cnt = 0;
        root = new_node();
        scanf("%d", &n);
        bool flag = true;
        while (n--)
        {
            scanf("%s", buf);
            if (flag) flag = try_insert(root, buf);
        }
        puts(flag ? "YES" : "NO");
    }
    return 0;
}
```

**POJ2945**

> n 个基因片段，每个长度为 m，输出 n 行表示重复出现 i 次 (1 ≤ i ≤ n)的基因片段的个数

```
#include <cstdio>
#include <cstring>
struct node
{
node *trans[4];
int cnt;
} nodes[400010];
int tot;
node *root;
 inline node *new_node() { return &nodes[tot++]; }
 void try_insert(node *n, char *str)
 {
    if (*str == '\0')
        n->cnt++;
    else
    {
        if (n->trans[*str - '0'] == 0) n->trans[*str - '0'] = new_node();
        try_insert(n->trans[*str - '0'], str + 1);
```

```
    }
}
char f[1 << 8 | 1];
int ans[20010];
int main()
{
    f['A'] = '0', f['C'] = '1', f['G'] = '2', f['T'] = '3';
    int n, m;
    char buf[22];
    while (~scanf("%d%d", &n, &m) && (n + m))
    {
        memset(ans, 0, sizeof(ans));
        memset(nodes, 0, sizeof(nodes));
        tot = 0;
        root = new_node();
        for (int i = 0; i < n; i++)
        {
            scanf("%s", buf);
            for (int j = 0; j < m; j++)
                buf[j] = f[buf[j]];
            try_insert(root, buf);
        }
        for (int i = 0; i < tot; i++) ans[nodes[i].cnt]++;
        for (int i = 1; i <= n; i++)
        printf("%d\n", ans[i]);
    }
    return 0;
}
```

## Aho-Corasick Automaton（多模式字符串匹配）

> Trie上的KMP，其中next数组变成了fail指针，功能相同

```
//buildFail
void buildFail()
{
    int h = 0, t = 0;
    root->fail = &virt;
    que[t++] = root;
    while (h < t)
    {
        node *cur = que[h++];
        for (int i = 0; i < 26; i++)
        {
            node *f = cur->fail;
            while (f->trans[i] == 0) f = f->fail;
            f = f->trans[i];
            if (cur->trans[i])
                (que[t++] = cur->trans[i])->fail = f;
            else
                cur->trans[i] = f;
        }
    }
```

```
}
```

### HDU2222

> AC 自动机模板题，注意统计答案时，每个节点只能统计一次不要重复统计。

```
#include <cstdio>
#include <cstring>
struct node
{
    node *trans[26], *fail;
    int cnt;
} nodes[500010], virt;
node *que[500010];
int tot;
 node *root;
 node *new_node() { return &nodes[tot++]; }
 void insert(char *str)
 {
     node *cur = root;
    for (; *str; str++)
    {
        if (cur->trans[*str - 'a'] == 0)
            cur->trans[*str - 'a'] = new_node();
        cur = cur->trans[*str - 'a'];
    }
    cur->cnt++;
 }
 void buildFail()
 {
    int h = 0, t = 0;
    root->fail = &virt;
    que[t++] = root;
    while (h < t)
    {
        node *cur = que[h++];
        for (int i = 0; i < 26; i++)
        {
            node *f = cur->fail;
            while (f->trans[i] == 0) f = f->fail;
            f = f->trans[i];
            if (cur->trans[i])
                (que[t++] = cur->trans[i])->fail = f;
            else
                cur->trans[i] = f;
        }
    }
 }
 char buf[1000010];
 int vis[500010];
 int main()
 {
    memset(vis, -1, sizeof(vis));
    int T, n;
```

```
    scanf("%d", &T);
    while (T--)
    {
        memset(nodes, 0, sizeof(nodes));
        tot = 0, root = new_node();
        for (int i = 0; i < 26; i++) virt.trans[i] = root;
        scanf("%d", &n);
        for (int i = 0; i < n; i++)
        {
            scanf("%s", buf);
            insert(buf);
        }
        buildFail();
        scanf("%s", buf);
        node *cur = root, *tmp;
        int ans = 0;
        for (char *ch = buf; *ch; ch++)
        {
            tmp = cur = cur->trans[*ch - 'a'];
            while (tmp != &virt && vis[tmp - nodes] != T)
            {
                vis[tmp - nodes] = T;
                ans += tmp->cnt;
                tmp = tmp->fail;
            }
        }
        printf("%d\n", ans);
    }
    return 0;
}
```

## Manacher（回文串）

**POJ3974**

```
#include <cstdio>
#include <cstring>
inline int min(int a, int b) { return a < b ? a : b; }
inline int max(int a, int b) { return a > b ? a : b; }
char buf[1000100], str[2000200];
int R[2000200], T;
//str.size=R.size=N<<1
int main()
{
    while (scanf("%s", buf), buf[0] != 'E')
    {
        int m = int(strlen(buf)), n = 0;
        str[n++] = '!';
        str[n++] = '#';
        for (int i = 0; i < m; i++)
            str[n++] = buf[i], str[n++] = '#';
        str[n++] = '#';
        str[n++] = '?';
        int p = 0, mx = 0, ans = 0;
        for (int i = 1; i < n; i++)
```

```
        {
            R[i] = mx > i ? min(R[2 * p - i], mx - i) : 1;
            while (str[i + R[i]] == str[i - R[i]]) R[i]++;
            if (R[i] + i > mx)
                mx = i + R[p = i];
            ans = max(ans, R[i]);
        }
        printf("Case %d: %d\n", ++T, ans - 1);
    }
    return 0;
}
```

### 后缀自动机

### 后缀数组

### 后缀树

## Computational Geometry

```
int relation(Point p,Line l){    //点和向量关系    1:左侧    2:右侧    3:在线上
     int c=dcmp(cross(p-l.s,l.e-l.s));
     if(c<0)    return 1;
     else if(c>0)    return 2;
     else    return 3;
 }

 Point projection(Point p,Line a){        //点在直线上的投影
     return a.s+(((a.e-a.s)*dot(a.e-a.s,p-a.s))/(a.e-a.s).len2());
 }

 Point symmetry(Point p,Line a){            //点关于直线的对称点
     Point q=projection(p,a);
     return Point(2*q.x-p.x,2*q.y-p.y);
 }
 //0810了解用 fromDD-BOND
```

### 费马点

三角形无超过120°的内角，三角形内的一点P与三顶点距离之和最小，P为费马点

爬山处理

(POJ 2420)

```
#include <vector>
#include <cstdio>
#include <limits>
#include <cmath>
using namespace std;
const double eps = 1e-8;
int dx[] = {1, -1, 0, 0};
int dy[] = {0, 0, 1, -1};
struct Point
{
    double x, y;
} v[200];
inline double sqr(double x) { return x * x; }
inline double dis(const Point &lhs, const Point &rhs)
{
    return sqrt(sqr(lhs.x - rhs.x) + sqr(lhs.y - rhs.y));
}
inline double sum(int n, const Point &p)
{
    double ret = 0;
    for (int i = 0; i < n; i++)
        ret += dis(v[i], p);
    return ret;
}
int main()
{
    int n;
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
        scanf("%lf%lf", &v[i].x, &v[i].y);
    Point s = v[0];
    double ans = numeric_limits<double>::max() / 2;
    for (int t = 100; t > eps; t *= 0.98) //also t--;
    {
        bool flag = true;
        while (flag)
        {
            flag = false;
            for (int i = 0; i < 4; i++)
            {
                Point p{s.x + dx[i] * t, s.y + dy[i] * t};
                double cur = sum(n, p);
                if (ans > cur)
                {
                    ans = cur;
                    s = p;
                    flag = true;
                }
            }
        }
    }
    printf("%.0f",ans);
```

```
    return 0;
}
```

## 最大空凸包

一个内部没有给定点的最大凸多边形

```
#define mem(a, b) memset(a, b, sizeof(a))
#define pi acos(-1)
using namespace std;
typedef long long ll;
const int  inf = 0x3f3f3f3f;
const int maxn = 105;

struct Point{
    int x, y;
    Point(){};
    Point(int x, int y):x(x), y(y){};
    Point operator + (const Point &a){
        return Point(x+a.x, y+a.y);
    }
    Point operator - (const Point &a){
        return Point(x-a.x, y-a.y);
    }
    int operator * (const Point &a){
        return x*a.y - y*a.x;
    }
    int len() const {
        return x*x+y*y;
    }
    bool const operator < (const Point &a){
        if((*this)*a > 0 || (*this)*a == 0 && len() < a.len()){
            return 1;
        }
        return 0;
    }
}point1[maxn], point2[maxn];
int dp[maxn][maxn];

int jud(int m){
    int ans = 0;
    mem(dp, 0);
    for(int i = 2; i <= m; i++){
        int now = i-1;
        while(now >= 1 && point2[i]*point2[now] == 0){
            now--;
        }
        int flag = 0;
        if(now == i-1){
            flag = 1;
        }
        while(now >= 1){
            int S = point2[now]*point2[i], k = now-1;
```

```
            while(k >= 1 && (point2[now]-point2[i])*(point2[k]-point2[now]) > 0)
{
                k--;
            }
            if(k >= 1){
                s += dp[now][k];
            }
            if(flag){
                dp[i][now] = s;
            }
            ans = max(ans, s);
            now = k;
        }
        if(!flag){
            continue;
        }
        for(int j = 1; j <= i-1; j++){
            dp[i][j] = max(dp[i][j], dp[i][j-1]);
        }
    }
    return ans;
}

int main(){
    int t;
    scanf("%d", &t);
    while(t--){
        int n;
        scanf("%d", &n);
        for(int i = 1; i <= n; i++){
            scanf("%d %d", &point1[i].x, &point1[i].y);
        }
        int ans = 0;
        for(int i = 1; i <= n; i++){
            int m = 0;
            for(int j = 1; j <= n; j++){
                if(point1[j].y > point1[i].y || point1[j].y == point1[i].y &&
point1[j].x >= point1[i].x){
                    point2[++m] = point1[j] - point1[i];
                }
            }
            sort(point2+1, point2+m+1);
            ans = max(ans, jud(m));
        }
        printf("%.1lf\n", ans/2.0);
    }

}
```

## 求两个多边形是否相交

(hdu 6590)

## 两多边形相交面积

nowcoder2019国庆派对day7 三角形和矩形

```
#include <bits/stdc++.h>

using namespace std;
const double eps = 1e-8;
const int maxisn = 20;

int dcmp(double x) {
    if (x > eps) return 1;
    return x < -eps ? -1 : 0;
}

inline double Sqr(double x) {
    return x * x;
}

struct Point {
    double x, y;

    Point() { x = y = 0; }

    Point(double x, double y) : x(x), y(y) {};

    friend Point operator+(const Point &a, const Point &b) {
        return Point(a.x + b.x, a.y + b.y);
    }

    friend Point operator-(const Point &a, const Point &b) {
        return Point(a.x - b.x, a.y - b.y);
    }

    friend bool operator==(const Point &a, const Point &b) {
        return dcmp(a.x - b.x) == 0 && dcmp(a.y - b.y) == 0;
    }

    friend Point operator*(const Point &a, const double &b) {
        return Point(a.x * b, a.y * b);
    }

    friend Point operator*(const double &a, const Point &b) {
        return Point(a * b.x, a * b.y);
    }

    friend Point operator/(const Point &a, const double &b) {
        return Point(a.x / b, a.y / b);
    }

    friend bool operator<(const Point &a, const Point &b) {
        return a.x < b.x || (a.x == b.x && a.y < b.y);
    }

    inline double dot(const Point &b) const {
```

```
        return x * b.x + y * b.y;
    }

    inline double cross(const Point &b, const Point &c) const {
        return (b.x - x) * (c.y - y) - (c.x - x) * (b.y - y);
    }

};

Point LineCross(const Point &a, const Point &b, const Point &c, const Point &d)
{
    double u = a.cross(b, c), v = b.cross(a, d);
    return Point((c.x * v + d.x * u) / (u + v), (c.y * v + d.y * u) / (u + v));
}

double PolygonArea(Point p[], int n) {
    if (n < 3) return 0.0;
    double s = p[0].y * (p[n - 1].x - p[1].x);
    p[n] = p[0];
    for (int i = 1; i < n; i++) {
        s += p[i].y * (p[i - 1].x - p[i + 1].x);
    }
    return fabs(s * 0.5);
}

double CPIA(Point a[], Point b[], int na, int nb) {
    Point p[maxisn], temp[maxisn];
    int i, j, tn, sflag, eflag;
    a[na] = a[0], b[nb] = b[0];
    memcpy(p, b, sizeof(Point) * (nb + 1));
    for (i = 0; i < na && nb > 2; ++i) {
        sflag = dcmp(a[i].cross(a[i + 1], p[0]));
        for (j = tn = 0; j < nb; ++j, sflag = eflag) {
            if (sflag >= 0) temp[tn++] = p[j];
            eflag = dcmp(a[i].cross(a[i + 1], p[j + 1]));
            if ((sflag ^ eflag) == -2)
                temp[tn++] = LineCross(a[i], a[i + 1], p[j], p[j + 1]);
        }
        memcpy(p, temp, sizeof(Point) * tn);
        nb = tn, p[nb] = p[0];
    }
    if (nb < 3) return 0.0;
    return PolygonArea(p, nb);
}

double SPIA(Point a[], Point b[], int na, int nb) {
    int i, j;
    Point t1[4], t2[4];
    double res = 0.0, if_clock_t1, if_clock_t2;
    a[na] = t1[0] = a[0];
    b[nb] = t2[0] = b[0];
    for (i = 2; i < na; i++) {
        t1[1] = a[i - 1], t1[2] = a[i];
        if_clock_t1 = dcmp(t1[0].cross(t1[1], t1[2]));
        if (if_clock_t1 < 0) swap(t1[1], t1[2]);
        for (j = 2; j < nb; j++) {
            t2[1] = b[j - 1], t2[2] = b[j];
            if_clock_t2 = dcmp(t2[0].cross(t2[1], t2[2]));
```

```
            if (if_clock_t2 < 0) swap(t2[1], t2[2]);
            res += CPIA(t1, t2, 3, 3) * if_clock_t1 * if_clock_t2;
        }
    }
    return res;
}

Point aa[5], bb[5];

int main() {
    double x1, y1, x2, y2, x3, y3, x4, y4;
    while (~scanf("%lf%lf%lf%lf%lf%lf%lf%lf", &x1, &y1, &x2, &y2, &x3, &y3, &x4,
&y4)) {
        aa[0] = {x1, y1};
        aa[1] = {x1, y2};
        aa[2] = {x2, y1};
        bb[0] = {x3, y3};
        bb[1] = {x3, y4};
        bb[2] = {x4, y4};
        bb[3] = {x4, y3};
        printf("%.8lf\n", abs(SPIA(aa, bb, 3, 4)));
    }
    return 0;
}
```

## 半平面交

## 半平面并

# Some Harder/Rare Problem

I doubt that whether here will be filled...

DA☆ZE~