**STL 标准模版库**

**pair**

Example，想要定义一个对象表示一个平面坐标点，可以：

pair<double, double> p;

cin >> p.first >> p.second;

定义了pair上的六个比较运算符：<、>、<=、>=、==、!=，其规则是先比较first，first相等时再比较second。

make\_pair需要两个参数，分别为元素对的首元素和尾元素。

**set**

set的基本操作：

s.begin() // 返回指向第一个元素的迭代器

s.clear() // 清除所有元素

s.count() // 返回某个值元素的个数

s.empty() // 如果集合为空，返回true(真）

s.end() // 返回指向最后一个元素之后的迭代器，不是最后一个元素

s.equal\_range() // 返回集合中与给定值相等的上下限的两个迭代器

s.erase() // 删除集合中的元素

s.find() // 返回一个指向被查找到元素的迭代器

s.get\_allocator() // 返回集合的分配器

s.insert() // 在集合中插入元素

s.lower\_bound() // 返回指向大于（或等于）某值的第一个元素的迭代器

s.key\_comp() // 返回一个用于元素间值比较的函数

s.max\_size() // 返回集合能容纳的元素的最大限值

s.rbegin() // 返回指向集合中最后一个元素的反向迭代器

s.rend() // 返回指向集合中第一个元素的反向迭代器

s.size() // 集合中元素的数目

s.swap() // 交换两个集合变量

s.upper\_bound() // 返回大于某个值元素的迭代器

s.value\_comp() // 返回一个用于比较元素间的值的函数

**vector**

vector的基本操作：

s[i] // 直接以下标方式访问容器中的元素

s.front() // 返回首元素

s.back() // 返回尾元素

s.push\_back(x) // 向表尾插入元素x

s.size() // 返回表长

s.empty() // 表为空时，返回真，否则返回假

s.pop\_back() // 删除表尾元素

s.begin() // 返回指向首元素的随机存取迭代器

s.end() // 返回指向尾元素的下一个位置的随机存取迭代器

s.insert(it, val) // 向迭代器it指向的元素前插入新元素val

s.insert(it, n, val)// 向迭代器it指向的元素前插入n个新元素val

s.insert(it, first, last) // 将由迭代器first和last所指定的序列[first, last)插入到迭代器it指向的元素前面

s.erase(it) // 删除由迭代器it所指向的元素

s.erase(first, last)// 删除由迭代器first和last所指定的序列[first, last)

s.reserve(n) // 预分配缓冲空间，使存储空间至少可容纳n个元素

s.resize(n) // 改变序列长度，超出的元素将会全部被删除，如果序列需要扩展（原空间小于n），元素默认值将填满扩展出的空间

s.resize(n, val) // 改变序列长度，超出的元素将会全部被删除，如果序列需要扩展（原空间小于n），val将填满扩展出的空间

s.clear() // 删除容器中的所有元素

s.swap(v) // 将s与另一个vector对象进行交换

**string**

int find(string s1, int i) 返回从下标i开始，出现子串s1时的下标，若无返回string::npos

构造函数：

string(const char \*s);    //用c字符串s初始化  
string(int n,char c);     //用n个字符c初始化

**子串：**

string substr(int pos = 0,int n = npos) const;//返回pos开始的n个字符组成的字符串

**stack**

stack的基本操作有：

s.push(x); // 入栈

s.pop(); // 出栈

s.top(); // 访问栈顶

s.empty(); // 当栈空时，返回true

s.size(); // 访问栈中元素个数

**queue**

queue的基本操作：

q.push(x); // 入队列

q.pop(); // 出队列

q.front(); // 访问队首元素

q.back(); // 访问队尾元素

q.empty(); // 判断队列是否为空

q.size(); // 访问队列中的元素个数

**priority\_queue**

priority\_queue<int> q;

priority\_queue<pair<int, int> > qq; //注意在两个尖括号之间一定要留空格，防止误判 priority\_queue<int, vector<int>, greater<int> > qqq;// 定义小的先出队列

priority\_queue的基本操作：

q.empty() // 如果队列为空，则返回true，否则返回false

q.size() // 返回队列中元素的个数

q.pop() // 删除队首元素，但不返回其值

q.top() // 返回具有最高优先级的元素值，但不删除该元素

q.push(item) // 在基于优先级的适当位置插入新元素

**map**

map的基本操作：

/\* 向map中插入元素 \*/

m[key] = value; // [key]操作是map很有特色的操作,如果在map中存在键值为key的元素对, 则返回该元素对的值域部分,否则将会创建一个键值为key的元素对,值域为默认值。所以可以用该操作向map中插入元素对或修改已经存在的元素对的值域部分。

m.insert(make\_pair(key, value)); // 也可以直接调用insert方法插入元素对,insert操作会返回一个pair,当map中没有与key相匹配的键值时,其first是指向插入元素对的迭代器,其second为true;若map中已经存在与key相等的键值时,其first是指向该元素对的迭代器,second为false。

/\* 查找元素 \*/

int i = m[key]; // 要注意的是,当与该键值相匹配的元素对不存在时,会创建键值为key（当另一个元素是整形时，m[key]=0）的元素对。

map<string, int>::iterator it = m.find(key); // 如果map中存在与key相匹配的键值时,find操作将返回指向该元素对的迭代器,否则,返回的迭代器等于map的end()(参见vector中提到的begin()和end()操作)。

/\* 删除元素 \*/

m.erase(key); // 删除与指定key键值相匹配的元素对,并返回被删除的元素的个数。

m.erase(it); // 删除由迭代器it所指定的元素对,并返回指向下一个元素对的迭代器。

/\* 其他操作 \*/

m.size(); // 返回元素个数

m.empty(); // 判断是否为空

m.clear(); // 清空所有元素

**algorithm**

1、for\_each遍历容器

for\_each(L.begin(), L.end(), visit);

int visit(int v) // 遍历算子函数 { cout << v << " "; return 1; }

1. min\_element/max\_element找出容器中的最小/最大值
2. sort对容器进行排序

**Number 数论**

**分解质因数法**

参考：   
《合数相关》

/\*

\* 分解质因数法求解，getFactor(n)函数见《合数相关》

\*/

int main(int argc, const char \* argv[])

{

// ...

getFactors(n);

int ret = n;

for (int i = 0; i < fatCnt; i++)

{

ret = (int)(ret / factor[i][0] \* (factor[i][0] - 1));

}

return 0;

}

**筛法欧拉函数**

const int MAXN = 100;

int phi[MAXN + 2];

int main(int argc, const char \* argv[])

{

for (int i = 1; i <= MAXN; i++)

{

phi[i] = i;

}

for (int i = 2; i <= MAXN; i += 2)

{

phi[i] /= 2;

}

for (int i = 3; i <= MAXN; i += 2)

{

if (phi[i] == i)

{

for (int j = i; j <= MAXN; j += i)

{

phi[j] = phi[j] / i \* (i - 1);

}

}

}

return 0;

}

**欧拉函数单独求解**

/\*

\* 单独求解的本质是公式的应用

\*/

unsigned euler(unsigned x)

{

unsigned i, res = x; // unsigned == unsigned int

for (i = 2; i < (int)sqrt(x \* 1.0) + 1; i++)

{

if (!(x % i))

{

res = res / i \* (i - 1);

while (!(x % i))

{

x /= i; // 保证i一定是素数

}

}

}

if (x > 1)

{

res = res / x \* (x - 1);

}

return res;

}

**欧拉函数线性筛**

/\*

\* 同时得到欧拉函数和素数表

\*/

const int MAXN = 10000000;

bool check[MAXN + 10];

int phi[MAXN + 10];

int prime[MAXN + 10];

int tot; // 素数个数

void phi\_and\_prime\_table(int N)

{

memset(check, false, sizeof(check));

phi[1] = 1;

tot = 0;

for (int i = 2; i <= N; i++)

{

if (!check[i])

{

prime[tot++] = i;

phi[i] = i - 1;

}

for (int j = 0; j < tot; j++)

{

if (i \* prime[j] > N)

{

break;

}

check[i \* prime[j]] = true;

if (i % prime[j] == 0)

{

phi[i \* prime[j]] = phi[i] \* prime[j];

break;

}

else

{

phi[i \* prime[j]] = phi[i] \* (prime[j] - 1);

}

}

}

return ;

}

**扩展GCD**

/\*

\* 求x，y使得gcd(a, b) = a \* x + b \* y;

\*/

int extgcd(int a, int b, int &x, int &y)

{

if (b == 0)

{

x = 1;

y = 0;

return a;

}

int d = extgcd(b, a % b, x, y);

int t = x;

x = y;

y = t - a / b \* y;

return d;

}

**线性方程组（高斯消元）**

**列主元**

/\*

\* 列主元gauss消去求解a[][] \* x[] = b[]

\* 返回是否有唯一解，若有解在b[]中

\*/

#define fabs(x) ((x) > 0 ? (x) : (-x))

#define eps 1e-10

const int MAXN = 100;

int gaussCpivot(int n, double a[][MAXN], double b[])

{

int i, j, k, row = 0;

double MAXP, temp;

for (k = 0; k < n; k++)

{

for (MAXP = 0, i = k; i < n; i++)

{

if (fabs(a[i][k]) > fabs(MAXP))

{

MAXP = a[row = i][k];

}

}

if (fabs(MAXP) < eps)

{

return 0;

}

if (row != k)

{

for (j = k; j < n; j++)

{

temp = a[k][j];

a[k][j] = a[row][j];

a[row][j] = temp;

temp = b[k];

b[k] = b[row];

b[row] = temp;

}

}

for (j = k + 1; j < n; j++)

{

a[k][j] /= MAXP;

for (i = k + 1; i < n; i++)

{

a[i][j] -= a[i][k] \* a[k][j];

}

}

b[k] /= MAXP;

for (i = n - 1; i >= 0; i--)

{

for (j = i + 1; j < n; j++)

{

b[i] -= a[i][j] \* b[j];

}

}

}

return 1;

}

**全主元**

/\*

\* 全主元gauss消去解a[][] \* x[] = b[]

\* 返回是否有唯一解，若有解在b[]中

\*/

#define fabs(x) ((x) > 0 ? (x) : (-x))

#define eps 1e-10

const int MAXN = 100;

int gaussTpivot(int n, double a[][MAXN], double b[])

{

int i, j, k, row = 0, col = 0, index[MAXN];

double MAXP, temp;

for (i = 0; i < n; i++)

{

index[i] = i;

}

for (k = 0; k < n; k++)

{

for (MAXP = 0, i = k; i < n; i++)

{

for (j = k; j < n; j++)

{

if (fabs(a[i][j] > fabs(MAXP)))

{

MAXP = a[row = i][col = j];

}

}

}

if (fabs(MAXP) < eps)

{

return 0;

}

if (col != k)

{

for (i = 0; i < n; i++)

{

temp = a[i][col];

a[i][col] = a[i][k];

a[i][k] = temp;

}

j = index[col];

index[col] = index[k];

index[k] = j;

}

if (row != k)

{

for (j = k; j < n; j++)

{

temp = a[k][j];

a[k][j] = a[row][j];

a[row][j] = temp;

}

temp = b[k];

b[k] = b[row];

b[row] = temp;

}

for (j = k + 1; j < n; j++)

{

a[k][j] /= MAXP;

for (i = k + 1; i < n; i++)

{

a[i][j] -= a[i][k] \* a[k][j];

}

}

b[k] /= MAXP;

for (i = k + 1; i < n; i++)

{

b[i] -= b[k] \* a[i][k];

}

}

for (i = n - 1; i >= 0; i--)

{

for (j = i + 1; j < n; j++)

{

b[i] -= a[i][j] \* b[j];

}

}

for (k = 0; k < n; k++)

{

a[0][index[k]] = b[k];

}

for (k = 0; k < n; k++)

{

b[k] = a[0][k];

}

return 1;

}

**高斯消元（自由变元，一类开关问题，位运算操作）**

// 高斯消元法求方程组的解

const int MAXN = 300;// 有equ个方程，var个变元。增广矩阵行数为equ，列数为var＋1，分别为0到var

int equ, var;

int a[MAXN][MAXN]; // 增广矩阵

int x[MAXN]; // 解集

int free\_x[MAXN]; // 用来存储自由变元（多解枚举自由变元可以使用）

int free\_num; // 自由变元的个数

// 返回值为－1表示无解，为0是唯一解，否则返回自由变元个数

int Gauss()

{

int max\_r, col, k;

free\_num = 0;

for (k = 0, col = 0; k < equ && col < var; k++, col++)

{

max\_r = k;

for (int i = k + 1; i < equ; i++)

{

if (abs(a[i][col]) > abs(a[max\_r][col]))

{

max\_r = i;

}

}

if (a[max\_r][col] == 0)

{

k--;

free\_x[free\_num++] = col; // 这是自由变元

continue;

}

if (max\_r != k)

{

for (int j = col; j < var + 1; j++)

{

swap(a[k][j], a[max\_r][j]);

}

}

for (int i = k + 1; i < equ; i++)

{

if (a[i][col] != 0)

{

for (int j = col; j < var + 1; j++)

{

a[i][j] ^= a[k][j];

}

}

}

}

for (int i = k; i < equ; i++)

{

if (a[i][col] != 0)

{

return -1; // 无解

}

}

if (k < var)

{

return var - k; // 自由变元个数

}

// 唯一解，回代

for (int i = var - 1; i >= 0; i--)

{

x[i] = a[i][var];

for (int j = i + 1; j < var; j++)

{

x[i] ^= (a[i][j] && x[j]);

}

}

return 0;

}

**模线性方程**

/\*

\* 模线性方程 a \* x = b (% n)

\*/

void modeq(int a, int b, int n)

{

int e, i, d, x, y;

d = extgcd(b, a % b, x, y);

if (b % d > 0)

{

cout << "No answer!\n";

}

else

{

e = (x \* (b / d)) % n;

for (i = 0; i < d; i++)

{

cout << i + 1 << "-th ans:" << (e + i \* (n / d)) % n << '\n';

}

}

return ;

}

**模线性方程组（互质）**

/\*

\* 模线性方程组

\* a = B[1](% W[1]); a = B[2](% W[2]); ... a = B[k](% W[k]);

\* 其中W，B已知，W[i] > 0且W[i]与W[j]互质，求a（中国剩余定理）

\*/

int china(int b[], int w[], int k)

{

int i, d, x, y, m, a = 0, n = 1;

for (i = 0; i < k; i++)

{

n \*= w[i]; // 注意不能overflow

}

for (i = 0; i < k; i++)

{

m = n / w[i];

d = extgcd(w[i], m, x, y);

a = (a + y \* m \* b[i]) % n;

}

if (a > 0)

{

return a;

}

else

{

return (a + n);

}

}

**模线性方程组（不要求互质）**

typedef long long ll;

const int MAXN = 11;

int n, m;

int a[MAXN], b[MAXN];

int main(int argc, const char \* argv[])

{

int T;

cin >> T;

while (T--)

{

cin >> n >> m;

for (int i = 0; i < m; i++)

{

cin >> a[i];

}

for (int i = 0; i < m; i++)

{

cin >> b[i];

}

ll ax = a[0], bx = b[0], x, y;

int flag = 0;

for (int i = 1; i < m; i++)

{

ll d = extgcd(ax, a[i], x, y);

if ((b[i] - bx) % d != 0)

{

flag = 1; // 无整数解

break;

}

ll tmp = a[i] / d;

x = x \* (b[i] - bx) / d; // 约分

x = (x % tmp + tmp) % tmp;

bx = bx + ax \* x;

ax = ax \* tmp; // ax = ax \* a[i] / d

}

if (flag == 1 || n < bx)

{

puts("0");

}

else

{

ll ans = (n - bx) / ax + 1;

if (bx == 0)

{

ans--;

}

printf("%lld\n", ans);

}

}

return 0;

}

**判断小于MAXN的数是不是素数**

/\*

\* 素数筛选，判断小于MAXN的数是不是素数

\* notprime是一张表，false表示是素数，true表示不是

\*/

const int MAXN = 1000010;

bool notprime[MAXN];

void init()

{

memset(notprime, false, sizeof(notprime));

notprime[0] = notprime[1] = true;

for (int i = 2; i < MAXN; i++)

{

if (!notprime[i])

{

if (i > MAXN / i) // 阻止后边i \* i溢出（或者i,j用long long)

{

continue;

}

// 直接从i \* i开始就可以，小于i倍的已经筛选过了

for (int j = i \* i; j < MAXN; j += i)

{

notprime[j] = true;

}

}

}

}

**查找出小于等于MAXN的素数（生成连续素数表）**

/\*

\* 素数筛选，查找出小于等于MAXN的素数

\* prime[0]存素数的个数

\*/

const int MAXN = 100000;

int prime[MAXN + 1];

void getPrime()

{

memset(prime, 0, sizeof(prime));

for (int i = 2; i <= MAXN; i++)

{

if (!prime[i])

{

prime[++prime[0]] = i;

}

for (int j = 1; j <= prime[0] && prime[j] <= MAXN / i; j++)

{

prime[prime[j] \* i] = 1;

if (i % prime[j] == 0)

{

break;

}

}

}

}

**随机素数测试**

/\*

\* 随机素数测试（伪素数原理）

\* CALL: bool res = miller(n);

\* 快速测试n是否满足素数的“必要”条件，出错概率极低

\* 对于任意奇数n > 2和正整数s，算法出错概率≤2^(-s)

\*/

int witness(int a, int n)

{

int x, d = 1;

int i = ceil(log(n - 1.0) / log(2.0)) - 1;

for (; i >= 0; i--)

{

x = d;

d = (d \* d) % n;

if (d == 1 && x != 1 && x != n - 1)

{

return 1;

}

if (((n - 1) & (1 << i)) > 0)

{

d = (d \* a) % n;

}

}

return (d == 1 ? 0 : 1);

}

int miller(int n, int s = 50)

{

if (n == 2) // 质数返回1

{

return 1;

}

if (n % 2 == 0) // 偶数返回0

{

return 0;

}

int j, a;

for (j = 0; j < a; j++)

{

a = rand() \* (n - 2) / RAND\_MAX + 1;

// rand()只能随机产生[0, RAND\_MAX)内的整数

// 而且这个RAND\_MAX只有32768直接%n的话是永远

// 也产生不了[RAND\_MAX, n)之间的数

if (witness(a, n))

{

return 0;

}

}

return 1;

}

**大数素数测试**

#define MAXL 4

#define M10 1000000000

#define Z10 9

const int zero[MAXL - 1] = {0};

struct bnum

{

int data[MAXL]; // 断成每截9个长度

// 读取字符串并转存

void read()

{

memset(data, 0, sizeof(data));

char buf[32];

scanf("%s", buf);

int len = (int)strlen(buf);

int i = 0, k;

while (len >= Z10)

{

for (k = len - Z10; k < len; ++k)

{

data[i] = data[i] \* 10 + buf[k] - '0';

}

++i;

len -= Z10;

}

if (len > 0)

{

for (k = 0; k < len; ++k)

{

data[i] = data[i] \* 10 + buf[k] - '0';

}

}

}

bool operator == (const bnum &x)

{

return memcmp(data, x.data, sizeof(data)) == 0;

}

bnum & operator = (const int x)

{

memset(data, 0, sizeof(data));

data[0] = x;

return \*this;

}

bnum operator + (const bnum &x)

{

int i, carry = 0;

bnum ans;

for (i = 0; i < MAXL; ++i)

{

ans.data[i] = data[i] + x.data[i] + carry;

carry = ans.data[i] / M10;

ans.data[i] %= M10;

}

return ans;

}

bnum operator - (const bnum &x)

{

int i, carry = 0;

bnum ans;

for (i = 0; i < MAXL; ++i)

{

ans.data[i] = data[i] - x.data[i] - carry;

if (ans.data[i] < 0)

{

ans.data[i] += M10;

carry = 1;

}

else

{

carry = 0;

}

}

return ans;

}

// assume \*this < x \* 2

bnum operator % (const bnum &x)

{

int i;

for (i = MAXL - 1; i >= 0; --i)

{

if (data[i] < x.data[i])

{

return \*this;

}

else if (data[i] > x.data[i])

{

break;

}

}

return ((\*this) - x);

}

bnum & div2()

{

int i, carry = 0, tmp;

for (i = MAXL - 1; i >= 0; --i)

{

tmp = data[i] & 1;

data[i] = (data[i] + carry) >> 1;

carry = tmp \* M10;

}

return \*this;

}

bool is\_odd()

{

return (data[0] & 1) == 1;

}

bool is\_zero()

{

for (int i = 0; i < MAXL; ++i)

{

if (data[i])

{

return false;

}

}

return true;

}

};

void mulmod(bnum &a0, bnum &b0, bnum &p, bnum &ans)

{

bnum tmp = a0, b = b0;

ans = 0;

while (!b.is\_zero())

{

if (b.is\_odd())

{

ans = (ans + tmp) % p;

}

tmp = (tmp + tmp) % p;

b.div2();

}

}

void powmod(bnum &a0, bnum &b0, bnum &p, bnum &ans)

{

bnum tmp = a0, b = b0;

ans = 1;

while (!b.is\_zero())

{

if (b.is\_odd())

{

mulmod(ans, tmp, p, ans);

}

mulmod(tmp, tmp, p, tmp);

b.div2();

}

}

bool MillerRabinTest(bnum &p, int iter)

{

int i, small = 0, j, d = 0;

for (i = 1; i < MAXL; ++i)

{

if (p.data[i])

{

break;

}

}

if (i == MAXL)

{

// small integer test

if (p.data[0] < 2)

{

return false;

}

if (p.data[0] == 2)

{

return true;

}

small = 1;

}

if (!p.is\_odd())

{

return false; // even number

}

bnum a, s, m, one, pd1;

one = 1;

s = pd1 = p - one;

while (!s.is\_odd())

{

s.div2();

++d;

}

for (i = 0; i < iter; ++i)

{

a = rand();

if (small)

{

a.data[0] = a.data[0] % (p.data[0] - 1) + 1;

}

else

{

a.data[1] = a.data[0] / M10;

a.data[0] %= M10;

}

if (a == one)

{

continue;

}

powmod(a, s, p, m);

for (j = 0; j < d && !(m == one) && !(m == pd1); ++j)

{

mulmod(m, m, p, m);

}

if (!(m == pd1) && j > 0)

{

return false;

}

}

return true;

}

int main()

{

bnum x;

x.read();

puts(MillerRabinTest(x, 5) ? "Yes" : "No");

return 0;

}

**合数分解**

/\*

\* 合数的分解需要先进行素数的筛选

\* factor[i][0]存放分解的素数

\* factor[i][1]存放对应素数出现的次数

\* fatCnt存放合数分解出的素数个数(相同的素数只算一次)

\*/

const int MAXN = 10000;

int prime[MAXN + 1];

// 获取素数

void getPrime()

{

memset(prime, 0, sizeof(prime));

for (int i = 2; i <= MAXN; i++)

{

if (!prime[i])

{

prime[++prime[0]] = i;

}

for (int j = 1; j <= prime[0] && prime[j] <= MAXN / i; j++)

{

prime[prime[j] \* i] = 1;

if (i % prime[j] == 0)

{

break;

}

}

}

return ;

}

long long factor[100][2];

int fatCnt;

// 合数分解

int getFactors(long long x)

{

fatCnt = 0;

long long tmp = x;

for (int i = 1; prime[i] <= tmp / prime[i]; i++)

{

factor[fatCnt][1] = 0;

if (tmp % prime[i] == 0)

{

factor[fatCnt][0] = prime[i];

while (tmp % prime[i] == 0)

{

factor[fatCnt][1]++;

tmp /= prime[i];

}

fatCnt++;

}

}

if (tmp != 1)

{

factor[fatCnt][0] = tmp;

factor[fatCnt++][1] = 1;

}

return fatCnt;

}

**组合数学相关**

**定理**

**One**

{1, 2, … n}的r组合a1, a2, … ar出现在所有r组合中的字典序位置编号, C(n, m)表示n中取m的组合数   
index = C(n, r) - C(n - a1, r) - C(n - a2, r - 1) - … - C(n - ar, 1)

**Two**

k \* C(n, k) = n \* C(n - 1, k - 1);   
C(n, 0) + C(n, 2) + … = C(n, 1) + C(n, 3) + …   
1 \* C(n, 1) + 2 \* C(n, 2) + … + n \* C(n, n) = n \* 2^(n - 1)

**Three · Catalan数**

C\_n = C(2 \* n, n) / (n + 1)   
C\_n = (4 \* n - 2) / (n + 1) \* C\_n - 1   
C\_1 = 1

**Four · Stirling数 · 1**

s(p, k)是将p个物体排成k个非空的循环排列的方法数(或者: 把p个人排成k个非空圆圈的方法数)。   
s(p, k) = (p - 1) \* s(p - 1, k) + s(p - 1, k - 1);

**Five · Stirling数 · 2**

S(p, k) = k \* S(p - 1, k) + S(p - 1, k - 1).   
S(p, 0) = 0, (p >= 1);   
S(p, p) = 1, (p >= 0);   
且有 S(p, 1) = 1, (p >= 1);   
S(p, 2) = 2^(p - 1) - 1, (p >= 2);   
S(p, p - 1) = C(p, 2);

**Six · Bell数**

B\_p = S(p, 0) + S(p, 1) + … + S(p, p)   
B\_p = C(p - 1, 0) \* B\_0 + C(p - 1, 1) \* B\_1 + … + C(p - 1, p - 1) \* B\_(p - 1)

**组合数C(n, r)**

int com(int n, int r) // return C(n, r)

{

if (n - r > r)

{

r = n - r; // C(n, r) = C(n, n - r)

}

int i, j, s = 1;

for (i = 0, j = 1; i < r; ++i)

{

s \*= (n - i);

for (; j <= r && s % j == 0; ++j)

{

s /= j;

}

}

return s;

}

**组合数C(a, b) (预处理)**

typedef long long ll;

const ll MOD = 1e9 + 7; // 必须为质数才管用

const ll MAXN = 1e5 + 3;

ll fac[MAXN]; // 阶乘

ll inv[MAXN]; // 阶乘的逆元

ll QPow(ll x, ll n)

{

ll ret = 1;

ll tmp = x % MOD;

while (n)

{

if (n & 1)

{

ret = (ret \* tmp) % MOD;

}

tmp = tmp \* tmp % MOD;

n >>= 1;

}

return ret;

}

void init()

{

fac[0] = 1;

for (int i = 1; i < MAXN; i++)

{

fac[i] = fac[i - 1] \* i % MOD;

}

inv[MAXN - 1] = QPow(fac[MAXN - 1], MOD - 2);

for (int i = MAXN - 2; i >= 0; i--)

{

inv[i] = inv[i + 1] \* (i + 1) % MOD;

}

}

ll C(ll a, ll b)

{

if (b > a)

{

return 0;

}

if (b == 0)

{

return 1;

}

return fac[a] \* inv[b] % MOD \* inv[a - b] % MOD;

}

**集合划分问题**

/\*

\* n元集合分划为k类的方案数记为S(n, k),称为第二类Stirling数。

\* 如{A,B,C}可以划分{{A}, {B}, {C}}, {{A, B}, {C}}, {{B, C}, {A}}, {{A, C}, {B}}, {{A, B, C}}。

\* 即一个集合可以划分为不同集合(1...n个)的种类数

\* CALL: compute(N); 每当输入一个n,输出B[n]

\*/

const int N = 2001;

int data[N][N], B[N];

void NGetM(int m, int n) // m 个数 n 个集合

{

// data[i][j]: i个数分成j个集合

int min, i, j;

data[0][0] = 1;

for (i = 1; i <= m; i++)

{

data[i][0] = 0;

}

for (i = 0; i <= m; i++)

{

data[i][i + 1] = 0;

}

for (i = 1; i <= m; i++)

{

if (i < n)

{

min = i;

}

else

{

min = n;

}

for (j = 1; j <= min; j++)

{

data[i][j] = (j \* data[i - 1][j] + data[i - 1][j - 1]);

}

}

return ;

}

void compute(int m)

{

// b[i]: Bell数

NGetM(m, m);

memset(B, 0, sizeof(B));

int i, j;

for (i=1; i <= m; i++)

{

for (j = 0; j <= i; j++)

{

B[i] += data[i][j];

}

}

return ;

}

**卢卡斯定理（从(1, 1)到(n, m)的走法，机器人走方格问题）**

#define MOD 1000000007

typedef long long LL;

LL quickPower(LL a, LL b)

{

LL ans = 1;

a %= MOD;

while (b)

{

if (b & 1)

{

ans = ans \* a % MOD;

}

b >>= 1;

a = a \* a % MOD;

}

return ans;

}

LL c(LL n, LL m)

{

if (m > n)

{

return 0;

}

LL ans = 1;

for (int i = 1; i <= m; i++)

{

LL a = (n + i - m) % MOD;

LL b = i % MOD;

ans = ans \* (a \* quickPower(b, MOD - 2) % MOD) % MOD;

}

return ans;

}

LL lucas(LL n, LL m)

{

if (m == 0)

{

return 1;

}

return c(n % MOD, m % MOD) \* lucas(n / MOD, m / MOD) % MOD;

}

int main(int argc, const char \* argv[])

{

LL n, m;

while (~scanf("%lld %lld", &n, &m))

{

LL max, min;

max = n + m - 3;

min = m - 1;

printf("%lld\n", lucas(max - 1, min - 1));

}

return 0;

}

**Polya计数**

/\*

\* c种颜色的珠子，组成长为s的项链，项链没有方向和起始位置

\*/

int gcd(int a, int b)

{

return b ? gcd(b, a % b) : a;

}

int main(int argc, const char \* argv[])

{

int c, s;

while (cin >> c >> s)

{

int k;

long long p[64];

p[0] = 1; // power of c

for (k = 0; k < s; k++)

{

p[k + 1] = p[k] \* c;

}

// reflection part

long long count = s & 1 ? s \* p[s / 2 + 1] : (s / 2) \* (p[s / 2] + p[s / 2 + 1]);

// rotation part

for (k = 1 ; k <= s ; k++)

{

count += p[gcd(k, s)];

count /= 2 \* s;

}

cout << count << '\n';

}

return 0;

}

**最大1矩阵**

const int N = 1000;

bool a[N][N];

int Run(const int &m, const int &n) // a[1...m][1...n]

{ // O(m\*n)

int i, j, k, l, r, max=0;

int col[N];

for (j = 1; j <= n; j++)

{

if (a[1][j] == 0 )

{

col[j] = 0;

}

else

{

for (k = 2; k <= m && a[k][j] == 1; k++);

col[j] = k - 1;

}

}

for (i = 1; i <= m; i++)

{

if (i > 1)

{

for (j = 1; j <= n; j++)

{

if (a[i][j] == 0)

{

col[j] = 0;

}

else

{

if (a[i - 1][j] == 0)

{

for (k = i + 1; k <= m && a[k][j] == 1; k++);

col[j] = k-1;

}

}

}

}

for (j = 1; j <= n; j++)

{

if (col[j] >= i)

{

for (l = j - 1; l > 0 && col[l] >= col[j]; --l);

l++;

for (r = j + 1; r <= n && col[r] >= col[j]; ++r);

r--;

int res = (r - l + 1) \* (col[j] - i + 1);

if( res > max )

{

max = res;

}

}

}

}

return max;

}

**约瑟夫环问题**

/\*

\* n个人(编号 1...n),先去掉第m个数,然后从m+1个开始报1,

\* 报到k的退出,剩下的人继续从1开始报数.求胜利者的编号.

\*/

int main(int argc, const char \* argv[])

{

int n, k, m;

while (cin >> n >> k >> m, n || k || m)

{

int i, d, s = 0;

for (i = 2; i <= n; i++)

{

s = (s + k) % i;

}

k = k % n;

if (k == 0)

{

k = n;

}

d = (s + 1) + (m - k);

if (d >= 1 && d <= n)

{

cout << d << '\n';

}

else if (d < 1)

{

cout << n + d << '\n';

}

else if (d > n)

{

cout << d % n << '\n';

}

}

return 0;

}

**函数图像法**

/\*

\* n 个人数到 k 出列，最后剩下的人编号

\*/

unsigned long long n, k;

int main()

{

cin >> n >> k;

long long y = k % 2;

long long x = 2, t = 0;

long long z1 = y, z2 = x;

while (x <= n)

{

z1 = y;

z2 = x;

t = (x - y) / (k - 1);

if (t == 0)

{

t++;

}

y = y + t \* k - ((y + t \* k) / (x + t)) \* (x + t);

x += t;

}

cout << (z1 + (n - z2) \* k) % n + 1 << endl;

return 0;

}

**博弈论**

**Bash**

#define \_MAX 10000

int a[\_MAX];

int b[\_MAX];

int bash(int N, int K)

{

if (N % (K + 1) == 0)

{

return 2;

}

return 1;

}

int main()

{

int T;

scanf("%d", &T);

for (int i = 0; i < T; i++)

{

scanf("%d%d", a + i, b + i);

}

for (int i = 0; i < T; i++)

{

if (bash(a[i], b[i]) == 1)

{

printf("A\n");

}

else

{

printf("B\n");

}

}

return 0;

}

**Nim**

int main(int argc, const char \* argv[])

{

int N, stone, tag = 0;

scanf("%d", &N);

while (N--)

{

scanf("%d", &stone);

tag ^= stone;

}

//tag为0则为后手赢，否则为先手赢

printf("%c\n", tag == 0 ? 'B' : 'A');

return 0;

}

**追赶法解周期性方程**

/\*

\* 周期性方程定义(n = 5)

\* |a\_1 b\_1 c\_1 d\_1 e\_1| = x\_1 --- 1

\* |e\_2 a\_2 b\_2 c\_2 d\_2| = x\_2 --- 2

\* |d\_2 e\_2 a\_2 b\_2 c\_2| = x\_3 --- 3

\* |c\_4 d\_2 e\_2 a\_4 b\_4| = x\_4 --- 4

\* |b\_5 c\_5 d\_5 e\_5 a\_5| = x\_5 --- 5

\* 输入： a[], b[], c[], x[]

\* 输出： 求解结果x在x[]中

\*/

const int MAXN = 1000;

int a[MAXN];

int b[MAXN];

int c[MAXN];

int x[MAXN];

void run()

{

c[0] /= b[0];

a[0] /= b[0];

x[0] /= b[0];

for (int i = 1; i < MAXN - 1; i++)

{

double temp = b[i] - a[i] \* c[i - 1];

c[i] /= temp;

x[i] = (x[i] - a[i] \* x[i - 1]) / temp;

a[i] = -a[i] \* a[i - 1] / temp;

}

a[MAXN - 2] = -a[MAXN - 2] - c[MAXN - 2];

for (int i = MAXN - 3; i >= 0; i--)

{

a[i] = -a[i] - c[i] \* a[i + 1];

x[i] -= c[i] \* x[i + 1];

}

x[MAXN - 1] -= (c[MAXN - 1] \* x[0] + a[MAXN - 1] \* x[MAXN - 2]);

x[MAXN - 1] /= (c[MAXN - 1] \* a[0] + a[MAXN - 1] \* a[MAXN - 2] + b[MAXN - 1]);

for (int i = MAXN - 2; i >= 0; i --)

{

x[i] += a[i] \* x[MAXN - 1];

}

return ;

}

**阶乘**

**阶乘最后非零位**

/\*

\* 阶乘最后非零位 复杂度O(nlongn)

\* 返回改为，n以字符串方式传入

\*/

#define MAXN 10000

const int mod[20] = {1, 1, 2, 6, 4, 2, 2, 4, 2, 8, 4, 4, 8, 4, 6, 8, 8, 6, 8, 2};

int lastDigit(char \*buf)

{

int len = (int)strlen(buf);

int a[MAXN], i, c, ret = 1;

if (len == 1)

{

return mod[buf[0] - '0'];

}

for (i = 0; i < len; i++)

{

a[i] = buf[len - 1 - i] - '0';

}

for (; len; len -= !a[len - 1])

{

ret = ret \* mod[a[1] % 2 \* 10 + a[0]] % 5;

for (c = 0, i = len - 1; i >= 0; i--)

{

c = c \* 10 + a[i];

a[i] = c / 5;

c %= 5;

}

}

return ret + ret % 2 \* 5;

}

**n的阶乘的长度**

#define PI 3.1415926

int main()

{

int n, a;

while (~scanf(“%d", &n))

{

a = (int)((0.5 \* log(2 \* PI \* n) + n \* log(n) - n) / log(10));

printf("%d\n", a + 1);

}

return 0;

}

**求逆元**

**扩展欧几里得法**

/\*

\* 扩展欧几里得法（求ax + by = gcd）

\*/

// 返回d = gcd(a, b);和对应于等式ax + by = d中的x、y

long long extendGcd(long long a, long long b, long long &x, long long &y)

{

if (a == 0 && b == 0)

{

return -1; // 无最大公约数

}

if (b == 0)

{

x = 1;

y = 0;

return a;

}

long long d = extendGcd(b, a % b, y, x);

y -= a / b \* x;

return d;

}

// 求逆元 ax = 1(mod n)

long long modReverse(long long a, long long n)

{

long long x, y;

long long d = extendGcd(a, n, x, y);

if (d == 1)

{

return (x % n + n) % n;

}

else

{

return -1; // 无逆元

}

}

**简洁写法**

/\*

\* 简洁写法I

\* 只能求a < m的情况，且a与m互质

\* 求ax = 1(mod m)的x值，即逆元(0 < a < m)

\*/

long long inv(long long a, long long m)

{

if (a == 1)

{

return 1;

}

return inv(m % a, m) \* (m - m / a) % m;

}

**欧拉函数法**

/\*

\* 欧拉函数法

\* mod为素数，而且a和m互质

\*/

// 快速幂取模

long long powM(long long a, long long b, long long m)

{

long long tmp = 1;

if (b == 0)

{

return 1;

}

if (b == 1)

{

return a % m;

}

tmp = powM(a, a >> 1, m);

tmp = tmp \* tmp % m;

if (b & 1)

{

tmp = tmp \* a % m;

}

return tmp;

}

long long inv(long long a, long long m)

{

return powM(a, m - 2, m);

}

**欧拉函数法（求阶乘逆元）**

typedef long long ll;

const ll MOD = 1e9 + 7; // 必须为质数才管用

const ll MAXN = 1e5 + 3;

ll fac[MAXN]; // 阶乘

ll inv[MAXN]; // 阶乘的逆元

ll QPow(ll x, ll n)

{

ll ret = 1;

ll tmp = x % MOD;

while (n)

{

if (n & 1)

{

ret = (ret \* tmp) % MOD;

}

tmp = tmp \* tmp % MOD;

n >>= 1;

}

return ret;

}

void init()

{

fac[0] = 1;

for (int i = 1; i < MAXN; i++)

{

fac[i] = fac[i - 1] \* i % MOD;

}

inv[MAXN - 1] = QPow(fac[MAXN - 1], MOD - 2);

for (int i = MAXN - 2; i >= 0; i--)

{

inv[i] = inv[i + 1] \* (i + 1) % MOD;

}

}

**FFT**

const double PI = acos(-1.0);

// 复数结构体

struct Complex

{

double x, y; // 实部和虚部 x + yi

Complex(double \_x = 0.0, double \_y = 0.0)

{

x = \_x;

y = \_y;

}

Complex operator - (const Complex &b) const

{

return Complex(x - b.x, y - b.y);

}

Complex operator + (const Complex &b) const

{

return Complex(x + b.x, y + b.y);

}

Complex operator \* (const Complex &b) const

{

return Complex(x \* b.x - y \* b.y, x \* b.y + y \* b.x);

}

};

// 进行FFT和IFFT前的反转变换

// 位置i和（i二进制反转后的位置）互换

// len必须去2的幂

void change(Complex y[], int len)

{

int i, j, k;

for (i = 1, j = len / 2; i < len - 1; i++)

{

if (i < j)

{

swap(y[i], y[j]);

}

// 交换护卫小标反转的元素，i < j保证交换一次

// i做正常的+1，j左反转类型的+1，始终保持i和j是反转的

k = len / 2;

while (j >= k)

{

j -= k;

k /= 2;

}

if (j < k)

{

j += k;

}

}

return ;

}

// FFT

// len必须为2 ^ k形式

// on == 1时是DFT，on == -1时是IDFT

void fft(Complex y[], int len, int on)

{

change(y, len);

for (int h = 2; h <= len; h <<= 1)

{

Complex wn(cos(-on \* 2 \* PI / h), sin(-on \* 2 \* PI / h));

for (int j = 0; j < len; j += h)

{

Complex w(1, 0);

for (int k = j; k < j + h / 2; k++)

{

Complex u = y[k];

Complex t = w \* y[k + h / 2];

y[k] = u + t;

y[k + h / 2] = u - t;

w = w \* wn;

}

}

}

if (on == -1)

{

for (int i = 0; i < len; i++)

{

y[i].x /= len;

}

}

}

**FWT**

/\*

\* FWT(快速沃尔什变化)-Xor

\* MOD:1e9 + 7, INV\_2:2关于MOD的逆元

\* N:2的整次幂(不够就向上取整)

\*/

typedef long long ll;

const int MOD = 1e9 + 7;

const int INV\_2 = 5e8 + 4;

inline void FWT(int c[], int N, int tf\_utf) // tf\_utf 1:tf; 0:utf

{

for (int i = 1; i < N; i <<= 1)

{

int tmp = i << 1;

for (int j = 0; j < N; j += tmp)

{

for (int k = 0; k < i; k++)

{

int x = c[j + k], y = c[j + k + i];

if (tf\_utf)

{

c[j + k] = x + y;

if (c[j + k] >= MOD)

{

c[j + k] -= MOD;

}

c[j + k + i] = x - y;

if (c[j + k + i] < 0)

{

c[j + k + i] += MOD;

}

}

else

{

c[j + k] = (ll)(x + y) \* INV\_2 % MOD;

c[j + k + i] = (ll)(x - y + MOD) \* INV\_2 % MOD;

}

}

}

}

}

**整数划分（五边形定理）**

// 划分元素可重复任意次

#define f(x) (((x) \* (3 \* (x) - 1)) >> 1)

#define g(x) (((x) \* (3 \* (x) + 1)) >> 1)

const int MAXN = 1e5 + 10;

const int MOD = 1e9 + 7;

int n, ans[MAXN];

int main()

{

scanf("%d", &n);

ans[0] = 1;

for (int i = 1; i <= n; ++i)

{

for (int j = 1; f(j) <= i; ++j)

{

if (j & 1)

{

ans[i] = (ans[i] + ans[i - f(j)]) % MOD;

}

else

{

ans[i] = (ans[i] - ans[i - f(j)] + MOD) % MOD;

}

}

for (int j = 1; g(j) <= i; ++j)

{

if (j & 1)

{

ans[i] = (ans[i] + ans[i - g(j)]) % MOD;

}

else

{

ans[i] = (ans[i] - ans[i - g(j)] + MOD) % MOD;

}

}

}

printf("%d\n", ans[n]);

return 0;

}

**整数划分（五边形定理拓展）**

// 问一个数n能被拆分成多少种情况

// 且要求拆分元素重复次数不能≥k

const int MOD = 1e9 + 7;

const int MAXN = 1e5 + 10;

int ans[MAXN];

// 此函数求ans[]效率比上一个代码段中求ans[]效率高很多

void init()

{

memset(ans, 0, sizeof(ans));

ans[0] = 1;

for (int i = 1; i < MAXN; ++i)

{

ans[i] = 0;

for (int j = 1; ; j++)

{

int tmp = (3 \* j - 1) \* j / 2;

if (tmp > i)

{

break;

}

int tmp\_ = ans[i - tmp];

if (tmp + j <= i)

{

tmp\_ = (tmp\_ + ans[i - tmp - j]) % MOD;

}

if (j & 1)

{

ans[i] = (ans[i] + tmp\_) % MOD;

}

else

{

ans[i] = (ans[i] - tmp\_ + MOD) % MOD;

}

}

}

return ;

}

int solve(int n, int k)

{

int res = ans[n];

for (int i = 1; ; i++)

{

int tmp = k \* i \* (3 \* i - 1) / 2;

if (tmp > n)

{

break;

}

int tmp\_ = ans[n - tmp];

if (tmp + i \* k <= n)

{

tmp\_ = (tmp\_ + ans[n - tmp - i \* k]) % MOD;

}

if (i & 1)

{

res = (res - tmp\_ + MOD) % MOD;

}

else

{

res = (res + tmp\_) % MOD;

}

}

return res;

}

int main(int argc, const char \* argv[])

{

init();

int T, n, k;

cin >> T;

while (T--)

{

cin >> n >> k;

cout << solve(n, k) << '\n';

}

return 0;

}

**A^B约数之和对MOD取模**

int MOD(int a, int b)

{

return a - a / b \* b;

}

int miu(int n)

{

int cnt, k = 0;

for (int i = 2; i \* i <= n; i++)

{

if (MOD(n, i))

{

continue;

}

cnt = 0;

k++;

while (MOD(n, i) == 0)

{

n /= i;

cnt++;

}

if (cnt >= 2)

{

return 0;

}

}

if (n != 1)

{

k++;

}

return MOD(k, 2) ? -1 : 1;

}

**莫比乌斯反演**

**莫比乌斯反演公式**

/\*

\* 莫比乌斯反演公式

\* 线性筛法求解积性函数（莫比乌斯函数）

\*/

const int MAXN = 1000000;

bool check[MAXN + 10];

int prime[MAXN + 10];

int mu[MAXN + 10];

void Moblus()

{

memset(check, false, sizeof(check));

mu[1] = 1;

int tot = 0;

for (int i = 2; i <= MAXN; i++)

{

if (!check[i])

{

prime[tot++] = i;

mu[i] = -1;

}

for (int j = 0; j < tot; j++)

{

if (i \* prime[j] > MAXN)

{

break;

}

check[i \* prime[j]] = true;

if (i % prime[j] == 0)

{

mu[i \* prime[j]] = 0;

break;

}

else

{

mu[i \* prime[j]] = -mu[i];

}

}

}

}

**单独求解**

int MOD(int a, int b)

{

return a - a / b \* b;

}

int miu(int n)

{

int cnt, k = 0;

for (int i = 2; i \* i <= n; i++)

{

if (MOD(n, i))

{

continue;

}

cnt = 0;

k++;

while (MOD(n, i) == 0)

{

n /= i;

cnt++;

}

if (cnt >= 2)

{

return 0;

}

}

if (n != 1)

{

k++;

}

return MOD(k, 2) ? -1 : 1;

}

**Baby-Step Giant-Step**

/\*

\* baby\_step giant \_step

\* a^x = b(mod n) n不要求是素数

\* 求解上式0 ≤ x < n的解

\*/

#define MOD 76543

int hs[MOD];

int head[MOD];

int \_next[MOD];

int id[MOD];

int top;

void insert(int x, int y)

{

int k = x % MOD;

hs[top] = x;

id[top] = y;

\_next[top] = head[k];

head[k] = top++;

return ;

}

int find(int x)

{

int k = x % MOD;

for (int i = head[k]; i != -1; i = \_next[i])

{

if (hs[i] == x)

{

return id[i];

}

}

return -1;

}

long long BSGS(int a, int b, int n)

{

memset(head, -1, sizeof(head));

top = 1;

if (b == 1)

{

return 0;

}

int m = (int)sqrt(n \* 1.0), j;

long long x = 1, p = 1;

for (int i = 0; i < m; i++, p = p \* a % n)

{

insert(p \* b % n, i);

}

for (long long i = m; ; i++)

{

if ((j = find(x = x \* p % n)) != -1)

{

return i - j;

}

if (i > n)

{

break;

}

}

return -1;

}

**自适应simpson积分**

const double eps = 1e-6; // 积分精度

// 被积函数

double F(double x)

{

double ans;

// 被积函数

// ...

// ans = x \* exp(x); // 椭圆为例

return ans;

}

// 三点simpson法，这里要求F是一个全局函数

double simpson(double a, double b)

{

double c = a + (b - a) / 2;

return (F(a) + 4 \* F(c) + F(b)) \* (b - a) / 6;

}

// 自适应simpson公式（递归过程），已知整个区间[a, b]上的三点simpson指A

double asr(double a, double b, double eps, double A)

{

double c = a + (b - a) / 2;

double L = simpson(a, c), R = simpson(c, b);

if (fabs(L + R - A) <= 15 \* eps)

{

return L + R + (L + R - A) / 15.0;

}

return asr(a, c, eps / 2, L) + asr(c, b, eps / 2, R);

}

// 自适应simpson公式（主过程）

double asr(double a, double b, double eps)

{

return asr(a, b, eps, simpson(a, b));

}

int main(int argc, const char \* argv[])

{

// std::cout << asr(1, 2, eps) << '\n';

return 0;

}

**多项式求根（牛顿法）**

/\*

\* 牛顿法解多项式的根

\* 输入:多项式系数c[],多项式度数n,求在[a,b]间的根

\* 输出:根 要求保证[a,b]间有根

\*/

double fabs(double x)

{

return (x < 0) ? -x : x;

}

double f(int m, double c[], double x)

{

int i;

double p = c[m];

for (i = m; i > 0; i--)

{

p = p \* x + c[i - 1];

}

return p;

}

int newton(double x0, double \*r, double c[], double cp[], int n, double a, double b, double eps)

{

int MAX\_ITERATION = 1000;

int i = 1;

double x1, x2, fp, eps2 = eps / 10.0;

x1 = x0;

while (i < MAX\_ITERATION)

{

x2 = f(n, c, x1);

fp = f(n - 1, cp, x1);

if ((fabs(fp) < 0.000000001) && (fabs(x2) > 1.0))

{

return 0;

}

x2 = x1 - x2 / fp;

if (fabs(x1 - x2) < eps2)

{

if (x2 < a || x2 > b)

{

return 0;

}

\*r = x2;

return 1;

}

x1 = x2;

i++;

}

return 0;

}

double Polynomial\_Root(double c[], int n, double a, double b, double eps)

{

double \*cp;

int i;

double root;

cp = (double \*)calloc(n, sizeof(double));

for (i = n - 1; i >= 0; i--)

{

cp[i] = (i + 1) \* c[i + 1];

}

if (a > b)

{

root = a;

a = b;

b = root;

}

if ((!newton(a, &root, c, cp, n, a, b, eps)) && (!newton(b, &root, c, cp, n, a, b, eps)))

{

newton((a + b) \* 0.5, &root, c, cp, n, a, b, eps);

}

free(cp);

if (fabs(root) < eps)

{

return fabs(root);

}

else

return root;

}

**星期问题**

基姆拉尔森公式：

W = (D + 2 \* M + 3 \* (M + 1) \ 5 + Y + Y \ 4 - Y \ 100 + Y \ 400) Mod 7

基姆拉尔森公式的计算结果是0，1，2，3，4，5，6 七种可能；   
结果的对应关系：   
0：星期一   
1：星期二   
2：星期三   
3：星期四   
4：星期五   
5：星期六   
6：星期日

/\*

\* 已知1752年9月3日是Sunday，并且日期控制在1700年2月28日后

\*/

char name[][15] = { "monday", "tuesday", "wednesday", "thursday", "friday", "saturday", "sunday"};

int main()

{

int d, m, y, a;

printf("Day: ");

scanf("%d", &d);

printf("Month: ");

scanf("%d", &m);

printf("Year: ");

scanf("%d", &y);

// 1月2月当作前一年的13,14月

if (m == 1 || m == 2)

{

m += 12;

y--;

}

// 判断是否在1752年9月3日之前,实际上合并在一起倒更加省事

if ((y < 1752) || (y == 1752 && m < 9) || (y == 1752 && m == 9 && d < 3))

{

// 因为日期控制在1700年2月28日后，所以不用考虑整百年是否是闰年

a = (d + 2 \* m + 3 \* (m + 1) / 5 + y + y / 4 + 5) % 7;

}

else

{

// 这里需要考虑整百年是否是闰年的情况

a = (d + 2 \* m + 3 \* (m + 1) / 5 + y + y / 4 - y / 100 + y / 400) % 7; // 实际上这个可以当做公式背下来

}

printf("it's a %s\n", name[a]);

return 0;

}

**斐波那契数列**

/\*

\* 求斐波那契数列第N项，模MOD

\*/

#define mod(a, m) ((a) % (m) + (m)) % (m)

const int MOD = 1e9 + 9;

struct MATRIX

{

long long a[2][2];

};

MATRIX a;

long long f[2];

void ANS\_Cf(MATRIX a)

{

f[0] = mod(a.a[0][0] + a.a[1][0], MOD);

f[1] = mod(a.a[0][1] + a.a[1][1], MOD);

return ;

}

MATRIX MATRIX\_Cf(MATRIX a, MATRIX b)

{

MATRIX ans;

int k;

for (int i = 0; i < 2; i++)

{

for (int j = 0; j < 2; j++)

{

ans.a[i][j] = 0;

k = 0;

while (k < 2)

{

ans.a[i][j] += a.a[k][i] \* b.a[j][k];

ans.a[i][j] = mod(ans.a[i][j], MOD);

++k;

}

}

}

return ans;

}

MATRIX MATRIX\_Pow(MATRIX a, long long n)

{

MATRIX ans;

ans.a[0][0] = 1;

ans.a[1][1] = 1;

ans.a[0][1] = 0;

ans.a[1][0] = 0;

while (n)

{

if (n & 1)

{

ans = MATRIX\_Cf(ans, a);

}

n = n >> 1;

a = MATRIX\_Cf(a, a);

}

return ans;

}

int main()

{

long long n;

while (cin >> n)

{

if (n == 1)

{

cout << '1' << '\n';

continue;

}

a.a[0][0] = a.a[0][1] = a.a[1][0] = 1;

a.a[1][1] = 0;

a = MATRIX\_Pow(a, n - 2);

ANS\_Cf(a);

cout << f[0] << '\n';

}

return 0;

}

**1/n循环节长度**

/\*

\* 求1/i的循环节长度的最大值，i<=n

\*/

const int MAXN = 1005;

int res[MAXN]; // 循环节长度

int main()

{

memset(res, 0, sizeof(res));

int i, temp, j, n;

for (temp = 1; temp <= 1000; temp++)

{

i = temp;

while (i % 2 == 0)

{

i /= 2;

}

while (i % 5 == 0)

{

i /= 5;

}

n = 1;

for (j = 1; j <= i; j++)

{

n \*= 10;

n %= i;

if (n == 1)

{

res[temp] = j;

break;

}

}

}

int max\_re;

while (cin >> n)

{

max\_re = 1;

for (i = 1; i <= n; i++)

{

if (res[i] > res[max\_re])

{

max\_re = i;

}

}

cout << max\_re << endl;

}

return 0;

}

**矩阵乘法**

/\*

\* 矩阵乘法 n\*n矩阵乘法

\*/

#define MAXN 111

#define mod(x) ((x) % MOD)

#define MOD 1000000007

#define LL long long

int n;

struct mat

{

int m[MAXN][MAXN];

};

// 矩阵乘法

mat operator \* (mat a, mat &b)

{

mat ret;

memset(ret.m, 0, sizeof(ret.m));

for (int k = 0; k < n; k++)

{

for (int i = 0; i < n; i++)

{

if (a.m[i][k])

{

for (int j = 0; j < n; j++)

{

ret.m[i][j] = mod(ret.m[i][j] + (LL)a.m[i][k] \* b.m[k][j]);

}

}

}

}

return ret;

}

矩阵乘法 + 判等

/\*

\* AB == C ???

\*/

struct Matrix

{

Type mat[MAXN][MAXN];

int n, m;

Matrix()

{

n = m = MAXN;

memset(mat, 0, sizeof(mat));

}

Matrix(const Matrix &a)

{

set\_size(a.n, a.m);

memcpy(mat, a.mat, sizeof(a.mat));

}

Matrix & operator = (const Matrix &a)

{

set\_size(a.n, a.m);

memcpy(mat, a.mat, sizeof(a.mat));

return \*this;

}

void set\_size(int row, int column)

{

n = row;

m = column;

}

friend Matrix operator \* (const Matrix &a, const Matrix &b)

{

Matrix ret;

ret.set\_size(a.n, b.m);

for (int i = 0; i < a.n; ++i)

{

for (int k = 0; k < a.m; ++k)

{

if (a.mat[i][k])

{

for (int j = 0; j < b.m; ++j)

{

if (b.mat[k][j])

{

ret.mat[i][j] = ret.mat[i][j] + a.mat[i][k] \* b.mat[k][j];

}

}

}

}

}

return ret;

}

friend bool operator == (const Matrix &a, const Matrix &b)

{

if (a.n != b.n || a.m != b.m)

{

return false;

}

for (int i = 0; i < a.n; ++i)

{

for (int j = 0; j < a.m; ++j)

{

if (a.mat[i][j] != b.mat[i][j])

{

return false;

}

}

}

return true;

}

};

/\*

\* 矩阵快速幂 n\*n矩阵的x次幂

\*/

#define MAXN 111

#define mod(x) ((x) % MOD)

#define MOD 1000000007

#define LL long long

int n;

struct mat

{

int m[MAXN][MAXN];

} unit; // 单元矩阵

// 矩阵乘法

mat operator \* (mat a, mat &b)

{

mat ret;

memset(ret.m, 0, sizeof(ret.m));

for (int k = 0; k < n; k++)

{

for (int i = 0; i < n; i++)

{

if (a.m[i][k])

{

for (int j = 0; j < n; j++)

{

ret.m[i][j] = mod(ret.m[i][j] + (LL)a.m[i][k] \* b.m[k][j]);

}

}

}

}

return ret;

}

void init\_unit()

{

for (int i = 0; i < MAXN; i++)

{

unit.m[i][i] = 1;

}

return ;

}

mat pow\_mat(mat a, LL n)

{

mat ret = unit;

while (n)

{

if (n & 1)

{

// n--;

ret = ret \* a;

}

n >>= 1;

a = a \* a;

}

return ret;

}

int main()

{

LL x;

init\_unit();

while (cin >> n >> x)

{

mat a;

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

cin >> a.m[i][j];

}

}

a = pow\_mat(a, x); // a矩阵的x次幂

// 输出矩阵

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

if (j + 1 == n)

{

cout << a.m[i][j] << endl;

}

else

{

cout << a.m[i][j] << " ";

}

}

}

}

return 0;

}

**容斥**

const int MAXN = 1111;

int n;

double ans;

double p[MAXN];

void dfs(int x, int tot, double sum) // dfs(1, 0, ?)

{

if (x == n + 1)

{

if (sum == 0.0)

{

return ;

}

if (tot & 1)

{

ans += 1 / sum; // 公式随意变

}

else

{

ans -= 1 / sum;

}

return ;

}

dfs(x + 1, tot, sum);

dfs(x + 1, tot + 1, sum + p[x]);

}

**母函数**

/\*

\* 母函数

\* c1是保存各项质量砝码可以组合的数目

\* c2是中间量，保存每一次的情况

\*/

const int MAXN = 1e4 + 10;

int n;

int c1[MAXN];

int c2[MAXN];

int main()

{

while (cin >> n)

{

for (int i = 0; i <= n; ++i)

{

c1[i] = 1;

c2[i] = 0;

}

for (int i = 2; i <= n; ++i)

{

for (int j = 0; j <= n; ++j)

{

for (int k = 0; k + j <= n; k += i)

{

c2[j + k] += c1[j];

}

}

for (int j = 0; j <= n; ++j)

{

c1[j] = c2[j];

c2[j] = 0;

}

}

cout << c1[n] << endl;

}

return 0;

}

**String 字符串**

**编辑距离**

编辑距离，又称Levenshtein距离（也叫做Edit Distance），是指两个字串之间，由一个转成另一个所需的最少编辑操作次数。许可的编辑操作包括将一个字符替换成另一个字符，插入一个字符，删除一个字符。

#include <iostream>

#include <cstring>

using namespace std;

typedef long long LL;

const int N = 1e3 + 5;

int T, cas = 0;int n, m;int dp[N][N];char s[N], t[N];

int main()

{

while (scanf("%s%s", s, t) != EOF)

{

int n = (int)strlen(s), m = (int)strlen(t);

for (int i = 0; i <= n; i++)

{

dp[i][0] = i;

}

for (int i = 0; i <= m; i++)

{

dp[0][i] = i;

}

for (int i = 1; i <= n; i++)

{

for (int j = 1; j <= m; j++)

{

dp[i][j] = min(dp[i - 1][j], dp[i][j - 1]) + 1;

dp[i][j] = min(dp[i][j], dp[i - 1][j - 1] + (s[i - 1] != t[j - 1]));

}

}

printf("%d\n", dp[n][m]);

}

return 0;

}

**strstr**

/\*

\* strstr函数

\* 功能：在串中查找指定字符串的第一次出现

\* 用法：char \*strstr(char \*strOne, char \*strTwo);

\* 据说strstr函数和KMP的算法效率差不多

\*/

int main(int argc, const char \* argv[])

{

char strOne[] = "Borland International";

char strTwo[] = "nation";

char \*ptr;

ptr = strstr(strOne, strTwo);

std::cout << ptr << '\n';

return 0;

}

**BM算法改进的算法：Sunday Algorithm**

void SUNDAY(char \*text, char \*patt)

{

size\_t temp[256];

size\_t \*shift = temp;

size\_t i, patt\_size = strlen(patt), text\_size = strlen(text);

cout << "size : " << patt\_size << endl;

for(i = 0; i < 256; i++)

{

\*(shift+i) = patt\_size + 1;

}

for(i = 0; i < patt\_size; i++)

{

\*(shift + (unsigned char)(\*(patt+i))) = patt\_size-i; // shift['s']=6步,shitf['e']=5以此类推

}

size\_t limit = text\_size - patt\_size + 1;

for(i = 0; i < limit; i += shift[text[i + patt\_size]])

{

if(text[i] == \*patt)

{

char \*match\_text = text + i + 1;

size\_t match\_size = 1;

do // 输出所有匹配的位置

{

if(match\_size == patt\_size)

{

cout << "the NO. is " << i << endl;

}

}

while((\*match\_text++) == patt[match\_size++]);

}

}

cout << endl;

}

int main(void)

{

char text[100] = "substring searching algorithm search";

char patt[10] = "search";

SUNDAY(text, patt);

return 0;

}

**字符串**HASH

/\*

\* 字符串 Hash

\* 注意：mod选择足够大的质数（至少大于字符串个数）

\*/

unsigned int hashA(char \*url, int mod)

{

unsigned int n = 0;

char \*b = (char \*)&n;

for (int i = 0; url[i]; i++)

{

b[i % 4] ^= url[i];

}

return n % mod;

}

unsigned int hashB(char \*url, int mod)

{

unsigned int h = 0;

unsigned int g;

while (\*url)

{

h = (h << 4) + \*url++;

g = h & 0xF0000000;

if (g)

{

h ^= (g >> 24);

}

h &= ~g;

}

return h % mod;

}

unsigned int hashC(char \*p, int prime = 25013)

{

unsigned int h = 0;

unsigned int g;

for (; \*p; p++)

{

h = (h << 4) + \*p;

g = h & 0xF0000000;

if (g)

{

h ^= (g >> 24);

h ^= g;

}

}

return h % prime;

}

**Graph 图论**

Floyd算法 邻接矩阵形式

/\*

\* Floyd算法，求从任意节点i到任意节点j的最短路径

\* cost[][]:初始化为INF（cost[i][i]：初始化为0）

\* lowcost[][]:最短路径，path[][]:最短路径（无限制）

\*/

const int MAXN = 100;

int cost[MAXN][MAXN];

int lowcost[MAXN][MAXN];

int path[MAXN][MAXN];

void Floyd(int n)

{

memcpy(lowcost, cost, sizeof(cost));

memset(path, -1, sizeof(path));

for (int k = 0; k < n; k++)

{

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

if (lowcost[i][j] > (lowcost[i][k] + lowcost[k][j]))

{

lowcost[i][j] = lowcost[i][k] + lowcost[k][j];

path[i][j] = k;

}

}

}

}

return ;

}

Floyd算法 点权 + 路径限制

/\*

\* Floyd算法，求从任意节点i到任意节点j的最短路径

\* cost[][]:初始化为INF（cost[i][i]：初始化为0）

\* val[]:点权，lowcost[][]:除起点、终点外的点权之和+最短路径

\* path[][]:路径限制，要求字典序最小的路径，下标1~N

\*/

const int MAXN = 110;

const int INF = 0x1f1f1f1f;

int val[MAXN]; // 点权

int cost[MAXN][MAXN];

int lowcost[MAXN][MAXN];

int path[MAXN][MAXN]; // i~j路径中的第一个结点

void Floyd(int n)

{

memcpy(lowcost, cost, sizeof(cost));

for (int i = 0; i <= n; i++)

{

for (int j = 0; j <= n; j++)

{

path[i][j] = j;

}

}

for (int k = 1; k <= n; k++)

{

for (int i = 1; i <= n; i++)

{

for (int j = 1; j <= n; j++)

{

int temp = lowcost[i][k] + lowcost[k][j] + val[k];

if (lowcost[i][j] > temp)

{

lowcost[i][j] = temp;

path[i][j] = path[i][k];

}

else if (lowcost[i][j] == temp && path[i][j] > path[i][k])

{

path[i][j] = path[i][k];

}

}

}

}

return ;

}

第K短路

Dijkstra

/\*

\* Dijkstra变形，可以证明每个点经过的次数为小于等于K，

\* 所有Dijkstra的数组dist由一维变为二维，记录经过该点

\* 1次、2次......k次的最小值

\* 输出dist[n - 1][k]即可

\*/

int g[1010][1010];

int n, m, x;

const int INF = 0x3f3f3f3f;

int vis[1010];

int dist[1010][20];

int main(int argc, const char \* argv[])

{

while (cin >> n >> m >> x)

{

//初始化

memset(g, 0x3f, sizeof(g));

memset(dist, 0x3f, sizeof(dist));

memset(vis, 0, sizeof(vis));

for (int i = 0; i < m; i++)

{

int p, q, r;

cin >> p >> q >> r;

if (r < g[p][q])

{

g[p][q] = r;

}

}

dist[1][0] = 0;

dist[0][0] = INF;

while (1)

{

int k = 0;

for (int i = 1; i <= n; i++)

{

if (vis[i] < x && dist[i][vis[i]] < dist[k][0])

{

k = i;

}

}

if (k == 0)

{

break;

}

if (k == n && vis[n] == x - 1)

{

break;

}

for (int i = 1; i <= n; i++)

{

if (vis[i] < x && dist[k][vis[k]] + g[k][i] < dist[i][x])

{

dist[i][x] = dist[k][vis[k]] + g[k][i];

for (int j = x; j > 0; j--)

{

if (dist[i][j] < dist[i][j - 1])

{

swap(dist[i][j], dist[i][j - 1]);

}

}

}

}

vis[k]++;

}

if (dist[n][x - 1] < INF)

{

cout << dist[n][x - 1] << endl;

}

else

{

cout << -1 << endl;

}

}

return 0;

}

A\*

/\*

\* A\* 估价函数 fi为到当前点走过的路经长度，hi为该点到终点的长度

\* gi = hi + fi

\*/

int n, m, x, ct;

int g[1010][1010];

int gr[1010][1010];

int dist[1010];

int vis[1010];

const int INF = 0x3f3f3f3f;

struct node

{

int id;

int fi;

int gi;

friend bool operator < (node a, node b)

{

if (a.gi == b.gi)

{

return a.fi > b.fi;

}

return a.gi > b.gi;

}

} s[20000010];

int init()

{

memset(dist, 0x3f, sizeof(dist));

for (int i = 0; i <= n; i++)

{

vis[i] = 1;

}

dist[n - 1] = 0;

for (int i = 0; i < n; i++)

{

int k = n;

for (int j = 0; j < n; j++)

{

if (vis[j] && dist[j] < dist[k])

{

k = j;

}

}

if (k == n)

{

break;

}

vis[k] = 0;

for (int j = 0; j < n; j++)

{

if (vis[j] && dist[k] + gr[k][j] < dist[j])

{

dist[j] = dist[k] + gr[k][j];

}

}

}

return 1;

}

int solve()

{

if (dist[0] == INF)

{

return -1;

}

ct = 0;

s[ct].id = 0;

s[ct].fi = 0;

s[ct++].gi = dist[0];

int cnt = 0;

while (ct)

{

int id = s[0].id;

int fi = s[0].fi;

if (id == n - 1)

{

cnt++;

}

if (cnt == x)

{

return fi;

}

pop\_heap(s, s + ct);

ct--;

for (int j = 0; j < n; j++)

{

if (g[id][j] < INF)

{

s[ct].id = j;

s[ct].fi = fi + g[id][j];

s[ct].gi = s[ct].fi + dist[j];

ct++;

push\_heap(s, s + ct);

}

}

}

return -1;

}

int main()

{

while (cin >> n >> m >> x)

{

memset(g, 0x3f, sizeof(g));

memset(gr, 0x3f, sizeof(gr));

for (int i = 0; i < n; i++)

{

int p, q, r;

cin >> p >> q >> r;

p--;

q--;

g[p][q] = g[p][q] <= r ? g[p][q] : r;

gr[q][p] = gr[q][p] <= r ? gr[q][p] : r;

}

init();

cout << solve() << endl;

}

return 0;

}

**欧拉回路**

每条边只经过一次，而且回到起点

无向图：

连通（不考虑度为0的点），每个顶点度数都为偶数。

/\*

\* SGU 101

\*/

struct Edge

{

int to;

int next;

int index;

int dir;

bool flag;

} edge[220];

int head[10]; //前驱int tot;

void init()

{

memset(head, -1, sizeof((head)));

tot = 0;

}

void addEdge(int u, int v, int index)

{

edge[tot].to = v;

edge[tot].next = head[u];

edge[tot].index = index;

edge[tot].dir = 0;

edge[tot].flag = false;

head[u] = tot++;

edge[tot].to = u;

edge[tot].next = head[v];

edge[tot].index = index;

edge[tot].dir = 1;

edge[tot].flag = false;

head[v] = tot++;

return ;

}

int du[10];std::vector<int>ans;

void dfs(int u)

{

for (int i = head[u]; i != -1; i = edge[i].next)

{

if (!edge[i].flag)

{

edge[i].flag = true;

edge[i ^ 1].flag = true;

dfs(edge[i].to);

ans.push\_back(i); //容器尾部插入i

}

}

return ;

}

int main()

{

//freopen("in.txt", "r", stdin);

//freopen("out.txt", "w", stdout);

int n;

while (std::cin >> n)

{

init();

int u, v;

memset(du, 0, sizeof(du));

for (int i = 1; i <= n; i++)

{

std::cin >> u >> v;

addEdge(u, v, i);

du[u]++;

du[v]++;

}

int s = -1;

int cnt = 0;

for (int i = 0; i <= 6; i++)

{

if (du[i] & 1)

{

cnt++;

s = i;

}

if (du[i] > 0 && s == -1)

{

s = i;

}

}

if (cnt != 0 && cnt != 2)

{

std::cout << "No solution" << '\n';

continue;

}

ans.clear();

dfs(s);

if (ans.size() != n)

{

std::cout << "No solution" << '\n';

continue;

}

for (int i = 0; i < ans.size(); i++)

{

printf("%d ", edge[ans[i]].index);

if (edge[ans[i]].dir == 0)

{

std::cout << "-" << '\n';

}

else

{

std::cout << "+" << '\n';

}

}

}

return 0;

}

**有向图：**

基图连通（把边当成无向边，同样不考虑度为0的点），每个顶点出度等于入度。

**欧拉路径**

每条边只经过一次，不要求回到起点

无向图：

连通（不考虑度为0的点），每个顶点度数都为偶数或者仅有两个点的度数为奇数。

/\*

\* O(E)

\* INIT:adj[][]置为图的邻接表；cnt[a]为a点的邻接点数

\* CALL:alpath(0); 注意：不要有自向边

\*/

const int V = 10000;int adj[V][V];int idx[V][V];int cnt[V];int stk[V];int top = 0;

int path(int v)

{

for (int w; cnt[v] > 0; v = w)

{

stk[top++] = v;

w = adj[v][--cnt[v]];

adj[w][idx[w][v]] = adj[w][--cnt[w]];

//处理的是无向图——边是双向边，删除v->w后，还要处理删除w->v

}

return v;

}

void elpath(int b, int n)

{

int i, j;

for (i = 0; i < n; i++)

{

for (j = 0; j < cnt[i]; j++)

{

idx[i][adj[i][j]] = j;

}

}

std::cout << b;

for (top = 0; path(b) == b && top != 0; )

{

b = stk[--top];

std::cout << '-' << b;

}

std::cout << std::endl;

}

有向图：

基图连通（把边当成无向边，同样不考虑度为0的点），每个顶点出度等于入度或者有且仅有一个点的出度比入度多1，有且仅有一个点的出度比入度少1，其余的出度等于入度。

/\*

\* POJ 2337

\* 给出n个小写字母组成的单词，要求将n个单词连接起来。使得前一个单词的最后一个字母和

\* 后一个单词的第一个字母相同。输出字典序最小解

\*/

struct Edge

{

int to;

int next;

int index;

bool flag;

}edge[2010];

int head[30];int tot;

void init()

{

tot = 0;

memset(head, -1, sizeof(head));

}

void addEdge(int u, int v, int index)

{

edge[tot].to = v;

edge[tot].next = head[u];

edge[tot].index = index;

edge[tot].flag = false;

head[u] = tot++;

return ;

}

std::string str[1010];int in[30];int out[30];int cnt;int ans[1010];

void dfs(int u)

{

for (int i = head[u]; i != -1; i = edge[i].next)

{

if (!edge[i].flag)

{

edge[i].flag = true;

dfs(edge[i].to);

ans[cnt++] = edge[i].index;

}

}

return ;

}

int main(int argc, const char \* argv[])

{// freopen("in.txt", "r", stdin);// freopen("out.txt", "w", stdout);

int T, n;

std::cin >> T;

while (T--)

{

std::cin >> n;

for (int i = 0; i < n; i++)

{

std::cin >> str[i];

}

std::sort(str, str + n); //要输出字典序最小的解，先按照字典序排序

init();

memset(in, 0, sizeof(in));

memset(out, 0, sizeof(out));

int start = 100;

for (int i = n - 1; i >= 0; i--) //字典序大的先加入

{

int u = str[i][0] - 'a';

int v = str[i][str[i].length() - 1] - 'a';

addEdge(u, v, i);

out[u]++;

in[v]++;

if (n < start)

{

start = u;

}

if (v < start)

{

start = v;

}

}

int ccOne = 0;

int ccTwo = 0;

for (int i = 0; i < 26; i++)

{

if (out[i] - in[i] == 1)

{

ccOne++;

start = 1; //如果有一个出度比入度大1的点，就从这个点出发，否则从最小的点出发

}

else if (out[i] - in[i] == -1)

{

ccTwo++;

}

else if (out[i] - in[i] != 0)

{

ccOne = 3;

}

}

if (!((ccOne == 0 && ccTwo == 0) || (ccOne == 1 && ccTwo == 1)))

{

std::cout << "\*\*\*" << '\n';

continue;

}

cnt = 0;

dfs(start);

if (cnt != n) //判断是否连通

{

std::cout << "\*\*\*" << '\n';

continue;

}

for (int i = cnt - 1; i >= 0; i--)

{

std::cout << str[ans[i]];

if (i > 0)

{

std::cout << '.';

}

else

{

std::cout << '\n';

}

}

}

return 0;

}

**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,删除这个顶点集合,以及这个集合中所有顶点相关联的边以后,原图变成多个连通块,就称这个点集为割点集合。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,原图变成多个连通块,就称这个点集为割边集合。一个图的边连通度的定义为,最小割边集合中的边数。

**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)。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)。

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。

**双连通分支**

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。双连通分支(biconnected component),或重连通分支, 就是图的极大双连通子图。特殊的,点双连通分支又叫做块。

**求割点与桥**

该算法是R.Tarjan发明的。对图深度优先搜索,定义DFS(u)为u在搜索树(以下简称为树)中被遍历到的次序号。定义Low(u)为u或u的子树中能通过非父子边追溯到的最早的节点,即DFS序号最小的节点。根据定义,则有:Low(u)=Min{DFS(u)DFS(v)(u,v)为后向边(返祖边)等价于DFS(v) < DFS(u)且v不为u的父亲节点Low(v)(u,v)为树枝边(父子边)}一个顶点u是割点,当且仅当满足(1)或(2)(1)u为树根,且u有多于一个子树。(2)u不为树根,且满足存在(u,v)为树枝边(或称父子边,即u为v在搜索树中的父亲),使得 DFS(u) <= Low(v)。一条无向边(u,v)是桥,当且仅当(u,v)为树枝边,且满足DFS(u) < Low(v)。

**求双连通分支**

下面要分开讨论点双连通分支与边双连通分支的求法。   
对于点双连通分支,实际上在求割点的过程中就能顺便把每个点双连通分支求出。建立一个栈,存储当前双连通分支,在搜索图时,每找到一条树枝边或后向边(非横叉边),就把这条边加入栈中。如果遇到某时满 足DFS(u) <= Low(v),说明u是一个割点,同时把边从栈顶一个个取出,直到遇到了边(u,v),取出的这些边与其关联的点,组成一个点双连通分支。割点可以属于多个点双连通分支,其余点和每条边只属于且属于一个点双连通分支。对于边双连通分支,求法更为简单。只需在求出所有的桥以后,把桥边删除,原图变成了多个连通块,则每个连通块就是一个边双连通分支。桥不属于任何一个边双连通分支,其余的边和每个顶点都属于且只属于一个边双连通分支。

**构造双连通图**

一个有桥的连通图,如何把它通过加边变成边双连通图?   
方法为首先求出所有的桥,然后删除这些桥边, 剩下的每个连通块都是一个双连通子图。把每个双连通子图收缩为一个顶点,再把桥边加回来,最后的这个图一定是一棵树,边连通度为1。统计出树中度为1的节点的个数,即为叶节点的个数,记为leaf。则至少在树上添加(leaf + 1) / 2条边,就能使树达到边二连通,所以至少添加的边数就是(leaf + 1) / 2。具体方法为,首先把两个最近公共祖先最远的两个叶节点之间连接一条边,这样可以把这两个点到祖先的路径上所有点收缩到一起,因为一个形成的环一定是双连通的。然后再找两个最近公共祖先最远的两个叶节点,这样一对一对找完,恰好是(leaf + 1) / 2 次,把所有点收缩到了一起。

无向图找桥

/\*

\* 无向图找桥

\* INIT: edge[][]邻接矩阵；vis[],pre[],ans[],bridge置0；

\* CALL: dfs(0, -1, 1, n);

\*/

const int V = 1010;

int bridge; //桥int edge[V][V];int ans[V];int pre[V];int vis[V];

void dfs(int cur, int father, int dep, int n)

{

//vertex: 0 ~ n - 1

if (bridge)

{

return ;

}

vis[cur] = 1;

pre[cur] = ans[cur] = dep;

for (int i = 0; i < n; i++)

{

if (edge[cur][i])

{

if (i != father && 1 == vis[i])

{

if (pre[i] < ans[cur])

{

ans[cur] = pre[i]; //back edge

}

}

if (0 == vis[i]) //tree edge

{

dfs(i, cur, dep + 1, n);

if (bridge)

{

return ;

}

if (ans[i] < ans[cur])

{

ans[cur] = ans[i];

}

if (ans[i] > pre[cur])

{

bridge = 1;

return ;

}

}

}

}

vis[cur] = 2;

return ;

}

LCA

**DFS + ST在线算法**

const int MAXN = 10010;int rmq[2 \* MAXN]; // rmq数组,就是欧拉序列对应的深度序列

struct ST

{

int mm[2 \* MAXN];

int dp[2 \* MAXN][20]; // 最小值对应的下标

void init(int n)

{

mm[0] = -1;

for (int i = 1; i <= n; i++)

{

mm[i] = ((i & (i - 1)) == 0) ? mm[i - 1] + 1 : mm[i - 1];

dp[i][0] = i;

}

for (int j = 1; j <= mm[n]; j++)

{

for (int i = 1; i + (1 << j) - 1 <= n; i++)

{

dp[i][j] = rmq[dp[i][j - 1]] < rmq[dp[i + (1 << (j - 1))][j - 1]] ? dp[i][j - 1] : dp[i + (1 << (j - 1))][j - 1];

}

}

}

int query(int a,int b) // 查询[a,b]之间最小值的下标

{

if (a > b)

{

swap(a, b);

}

int k = mm[b - a + 1];

return rmq[dp[a][k]] <= rmq[dp[b - (1 << k) + 1][k]] ? dp[a][k] : dp[b - (1 << k) + 1][k];

}

};

// 边的结构体定义

struct Edge

{

int to, next;

};

Edge edge[MAXN \* 2];

int tot, head[MAXN];int F[MAXN \* 2]; // 欧拉序列,就是dfs遍历的顺序,长度为2\*n-1,下标从1开始int P[MAXN]; // P[i]表示点i在F中第一次出现的位置int cnt;

ST st;

void init()

{

tot = 0;

memset(head, -1, sizeof(head));

}

void addedge(int u, int v) // 加边,无向边需要加两次

{

edge[tot].to = v;

edge[tot].next = head[u];

head[u] = tot++;

}

void dfs(int u, int pre, int dep)

{

F[++cnt] = u;

rmq[cnt] = dep;

P[u] = cnt;

for (int i = head[u]; i != -1; i = edge[i].next)

{

int v = edge[i].to;

if (v == pre)

{

continue;

}

dfs(v, u, dep + 1);

F[++cnt] = u;

rmq[cnt] = dep;

}

}

void LCA\_init(int root, int node\_num) // 查询LCA前的初始化

{

cnt = 0;

dfs(root, root, 0);

st.init(2 \* node\_num - 1);

}

int query\_lca(int u, int v) // 查询u,v的lca编号

{

return F[st.query(P[u], P[v])];

}

bool flag[MAXN];

int main()

{

int T;

int N;

int u, v;

scanf("%d", &T);

while(T--)

{

scanf("%d", &N);

init();

memset(flag, false, sizeof(flag));

for (int i = 1; i < N; i++)

{

scanf("%d%d", &u, &v);

addedge(u, v);

addedge(v, u);

flag[v] = true;

}

int root;

for (int i = 1; i <= N; i++)

{

if (!flag[i])

{

root = i;

break;

}

}

LCA\_init(root, N);

scanf("%d%d", &u, &v);

printf("%d\n", query\_lca(u, v));

}

return 0;

}

**Tarjan离线算法**

/\*

\* 给出一颗有向树，Q个查询

\* 输出查询结果中每个点出现次数

\* 复杂度O(n + Q);

\*/const int MAXN = 1010;const int MAXQ = 500010; // 查询数的最大值

// 并查集部分int F[MAXN]; // 需要初始化为-1

int find(int x)

{

if (F[x] == -1)

{

return x;

}

return F[x] = find(F[x]);

}

void bing(int u, int v)

{

int t1 = find(u);

int t2 = find(v);

if (t1 != t2)

{

F[t1] = t2;

}

}

bool vis[MAXN]; // 访问标记int ancestor[MAXN]; // 祖先struct Edge

{

int to, next;

} edge[MAXN \* 2];int head[MAXN],tot;

void addedge(int u, int v)

{

edge[tot].to = v;

edge[tot].next = head[u];

head[u] = tot++;

}

struct Query

{

int q, next;

int index; // 查询编号

} query[MAXQ \* 2];

int answer[MAXQ]; // 存储最后的查询结果,下标0~Q-1int h[MAXQ];int tt;int Q;

void add\_query(int u, int v, int index)

{

query[tt].q = v;

query[tt].next = h[u];

query[tt].index = index;

h[u] = tt++;

query[tt].q = u;

query[tt].next = h[v];

query[tt].index = index;

h[v] = tt++;

}

void init()

{

tot = 0;

memset(head, -1, sizeof(head));

tt = 0;

memset(h, -1, sizeof(h));

memset(vis, false, sizeof(vis));

memset(F, -1, sizeof(F));

memset(ancestor, 0, sizeof(ancestor));

}

void LCA(int u)

{

ancestor[u] = u;

vis[u] = true;

for (int i = head[u]; i != -1; i = edge[i].next)

{

int v = edge[i].to;

if (vis[v])

{

continue;

}

LCA(v);

bing(u, v);

ancestor[find(u)] = u;

}

for (int i = h[u]; i != -1; i = query[i].next)

{

int v = query[i].q;

if (vis[v])

{

answer[query[i].index] = ancestor[find(v)];

}

}

}

bool flag[MAXN];int Count\_num[MAXN];

int main()

{

int n;

int u, v, k;

while (scanf("%d", &n) == 1)

{

init();

memset(flag, false, sizeof(flag));

for (int i = 1; i <= n; i++)

{

scanf("%d:(%d)", &u, &k);

while (k--)

{

scanf("%d", &v);

flag[v] = true;

addedge(u,v);

addedge(v,u);

}

}

scanf("%d", &Q);

for (int i = 0; i < Q; i++)

{

char ch;

cin >> ch;

scanf("%d %d)", &u, &v);

add\_query(u, v, i);

}

int root;

for (int i = 1; i <= n; i++)

{

if (!flag[i])

{

root = i;

break;

}

}

LCA(root);

memset(Count\_num, 0, sizeof(Count\_num));

for (int i = 0; i < Q; i++)

{

Count\_num[answer[i]]++;

}

for (int i = 1; i <= n; i++)

{

if (Count\_num[i] > 0)

{

printf("%d:%d\n", i, Count\_num[i]);

}

}

}

return 0;

}

**倍增法**

/\*

\* LCA在线算法(倍增法)

\*/const int MAXN = 10010;const int DEG = 20;

struct Edge

{

int to, next;

} edge[MAXN \* 2];

int head[MAXN], tot;void addedge(int u, int v)

{

edge[tot].to = v;

edge[tot].next = head[u];

head[u] = tot++;

}

void init()

{

tot = 0;

memset(head, -1, sizeof(head));

}

int fa[MAXN][DEG]; // fa[i][j]表示结点i的第2^j个祖先int deg[MAXN]; // 深度数组

void BFS(int root)

{

queue<int>que;

deg[root] = 0;

fa[root][0] = root;

que.push(root);

while (!que.empty())

{

int tmp = que.front();

que.pop();

for (int i = 1; i < DEG; i++)

{

fa[tmp][i] = fa[fa[tmp][i - 1]][i - 1];

}

for (int i = head[tmp]; i != -1; i = edge[i].next)

{

int v = edge[i].to;

if (v == fa[tmp][0])

{

continue;

}

deg[v] = deg[tmp] + 1;

fa[v][0] = tmp;

que.push(v);

}

}

}

int LCA(int u, int v)

{

if (deg[u] > deg[v])

{

swap(u, v);

}

int hu = deg[u], hv = deg[v];

int tu = u, tv = v;

for (int det = hv-hu, i = 0; det ; det >>= 1, i++)

{

if (det & 1)

{

tv = fa[tv][i];

}

}

if (tu == tv)

{

return tu;

}

for (int i = DEG - 1; i >= 0; i--)

{

if (fa[tu][i] == fa[tv][i])

{

continue;

}

tu = fa[tu][i];

tv = fa[tv][i];

}

return fa[tu][0];

}

bool flag[MAXN];

int main()

{

int T;

int n;

int u, v;

scanf("%d", &T);

while(T--)

{

scanf("%d", &n);

init();

memset(flag, false, sizeof(flag));

for (int i = 1; i < n; i++)

{

scanf("%d%d", &u, &v);

addedge(u, v);

addedge(v, u);

flag[v] = true;

}

int root;

for (int i = 1; i <= n; i++)

{

if (!flag[i])

{

root = i;

break;

}

}

BFS(root);

scanf("%d%d", &u, &v);

printf("%d\n", LCA(u, v));

}

return 0;

}

**生成树计数**

Matrix-Tree 定理(Kirchhoff 矩阵-树定理)   
1、G 的度数矩阵 D[G]是一个 n\*n 的矩阵,并且满足:当 i≠j 时,dij=0;当 i=j 时,dij 等于 vi 的度数。   
2、G 的邻接矩阵 A[G]也是一个 n\*n 的矩阵, 并且满足:如果 vi、vj 之间有边直接相连,则 aij=1,否则   
为 0。   
我们定义 G 的 Kirchhoff 矩阵(也称为拉普拉斯算子)C[G]为 C[G]=D[G]-A[G],则 Matrix-Tree 定理可以   
描述为:G 的所有不同的生成树的个数等于其 Kirchhoff 矩阵 C[G]任何一个 n-1 阶主子式的行列式的绝对   
值。所谓 n-1 阶主子式,就是对于 r(1≤r≤n),将 C[G]的第 r 行、第 r 列同时去掉后得到的新矩阵,用 Cr[G]   
表示。

**求生成树计数部分代码,计数对10007取模**

// 求生成树计数部分代码,计数对10007取模

const int MOD = 10007;

int INV[MOD];

// 求ax = 1(mod m)的x值,就是逆元(0<a<m)

long long inv(long long a, long long m)

{

if (a == 1)

{

return 1;

}

return inv(m % a, m) \* (m - m / a) % m;

}

struct Matrix

{

int mat[330][330];

void init()

{

memset(mat, 0, sizeof(mat));

}

int det(int n) // 求行列式的值模上MOD,需要使用逆元

{

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

mat[i][j] = (mat[i][j] % MOD + MOD) % MOD;

}

}

int res = 1;

for (int i = 0; i < n; i++)

{

for (int j = i; j < n; j++)

{

if (mat[j][i] != 0)

{

for (int k = i; k < n; k++)

{

swap(mat[i][k], mat[j][k]);

}

if (i != j)

{

res = (-res + MOD) % MOD;

}

break;

}

}

if (mat[i][i] == 0)

{

res = -1; // 不存在(也就是行列式值为0)

break;

}

for (int j = i + 1; j < n; j++)

{

//int mut = (mat[j][i]\*INV[mat[i][i]])%MOD;//打表逆元

int mut = (mat[j][i] \* inv(mat[i][i], MOD)) % MOD;

for (int k = i; k < n; k++)

{

mat[j][k] = (mat[j][k] - (mat[i][k] \* mut) % MOD + MOD) % MOD;

}

}

res = (res \* mat[i][i]) % MOD;

}

return res;

}

};

int main()

{

Matrix ret;

ret.init();

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

if (i != j && g[i][j])

{

ret.mat[i][j] = -1;

ret.mat[i][i]++;

}

}

}

printf("%d\n", ret.det(n - 1));

return 0;

}

**计算生成树个数,不取模**

const double eps = 1e-8;

const int MAXN = 110;

int sgn(double x)

{

if (fabs(x) < eps)

{

return 0;

}

if (x < 0)

{

return -1;

}

else

{

return 1;

}

}

double b[MAXN][MAXN];double det(double a[][MAXN], int n)

{

int i, j, k, sign = 0;

double ret = 1;

for (i = 0; i < n; i++)

{

for (j = 0; j < n; j++)

{

b[i][j] = a[i][j];

}

}

for (i = 0; i < n; i++)

{

if (sgn(b[i][i]) == 0)

{

for (j = i + 1; j < n; j++)

{

if (sgn(b[j][i]) != 0)

{

break;

}

}

if (j == n)

{

return 0;

}

for (k = i; k < n; k++)

{

swap(b[i][k], b[j][k]);

}

sign++;

}

ret \*= b[i][i];

for (k = i + 1; k < n; k++)

{

b[i][k] /= b[i][i];

}

for (j = i+1; j < n; j++)

{

for (k = i+1; k < n; k++)

{

b[j][k] -= b[j][i] \* b[i][k];

}

}

}

if (sign & 1)

{

ret = -ret;

}

return ret;

}

double a[MAXN][MAXN];int g[MAXN][MAXN];

int main()

{

int T;

int n, m;

int u, v;

scanf("%d", &T);

while (T--)

{

scanf("%d%d", &n, &m);

memset(g, 0, sizeof(g));

while (m--)

{

scanf("%d%d", &u, &v);

u--;

v--;

g[u][v] = g[v][u] = 1;

}

memset(a, 0, sizeof(a));

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

if (i != j && g[i][j])

{

a[i][i]++;

a[i][j] = -1;

}

}

}

double ans = det(a, n - 1);

printf("%.0lf\n", ans);

}

return 0;

}

有向图最小树形图

/\*

\* 有向图最小树形图

\* INIT: eg置为边表；res置为0；cp[i]置为i；

\* CALL: dirTree(root, nv, ne); res是结果

\*/

#define typec int // type of res

const typec V = 1010;

const typec E = 10010;

const typec inf = 0x3f3f3f3f; // max of res

typec res, dis[V];

int to[V], cp[V], tag[V];

struct Edge

{

int u, v;

typec c;

} eg[E];

int iroot(int i)

{

if (cp[i] == i)

{

return i;

}

return cp[i] = iroot(cp[i]);

}

int dirTree(int root, int nv, int ne) // root:树根

{

// vertex:0~n-1

int i, j, k, circle = 0;

memset(tag, -1, sizeof(tag));

memset(to, -1, sizeof(to));

for (i = 0; i < nv; i++)

{

dis[i] = inf;

}

for (j = 0; j < ne; j++)

{

i = iroot(eg[j].u);

k = iroot(eg[j].v);

if (k != i && dis[k] > eg[j].c)

{

dis[k] = eg[j].c;

to[k] = i;

}

}

to[root] = -1;

dis[root] = 0;

tag[root] = root;

for (i = 0; i < nv; i++)

{

if (cp[i] == i && -1 == tag[i])

{

j = i;

for (; j != -1 && tag[j] == -1; j = to[j])

{

tag[j] = i;

if (j == -1)

{

return 0;

}

if (tag[j] == i)

{

circle = 1;

tag[j] = -2;

for (k = to[j]; k != j; k = to[k])

{

tag[k] = -2;

}

}

}

}

}

if (circle)

{

for (j = 0; j < ne; j++)

{

i = iroot(eg[j].u);

k = iroot(eg[j].v);

if (k != i && tag[k] == -2)

{

eg[j].c -= dis[k];

}

}

for (i = 0; i < nv; i++)

{

if (tag[i] == -2)

{

res += dis[i];

tag[i] = 0;

for (j = to[i]; j != i; j = to[j])

{

res += dis[j];

cp[j] = i;

tag[j] = 0;

}

}

}

if (0 == dirTree(root, nv, ne))

{

return 0;

}

}

else

{

for (i = 0; i < nv; i++)

{

if (cp[i] == i)

{

res += dis[i];

}

}

}

return 1; // 若返回0代表原图不连通

}

有向图的强连通分量

/\*

\* Tarjan算法

\* 复杂度O(N+M)

\*/

const int MAXN = 20010; // 点数

const int MAXM = 50010; // 边数

struct Edge

{

int to, next;

}edge[MAXM];

int head[MAXN], tot;

int Low[MAXN], DFN[MAXN], Stack[MAXN], Belong[MAXN]; // Belong数组的值是1~scc

int Index, top;

int scc; // 强连通分量的个数

bool Instack[MAXN];

int num[MAXN]; // 各个强连通分量包含点的个数,数组编号1~scc

// num数组不一定需要,结合实际情况

void addedge(int u, int v)

{

edge[tot].to = v;

edge[tot].next = head[u];

head[u] = tot++;

return ;

}

void Tarjan(int u)

{

int v;

Low[u] = DFN[u] = ++Index;

Stack[top++] = u;

Instack[u] = true;

for (int i = head[u]; i != -1; i = edge[i].next)

{

v = edge[i].to;

if (!DFN[v])

{

Tarjan(v);

if (Low[u] > Low[v])

{

Low[u] = Low[v];

}

}

else if (Instack[v] && Low[u] > DFN[v])

{

Low[u] = DFN[v];

}

}

if (Low[u] == DFN[u])

{

scc++;

do

{

v = Stack[--top];

Instack[v] = false;

Belong[v] = scc; num[scc]++;

}

while (v != u);

}

return ;

}

void solve(int N)

{

memset(DFN, 0, sizeof(DFN));

memset(Instack, false, sizeof(Instack));

memset(num, 0, sizeof(num));

Index = scc = top = 0;

for (int i = 1; i <= N; i++)

{

if (!DFN[i])

{

Tarjan(i);

}

}

return ;

}

void init()

{

tot = 0;

memset(head, -1, sizeof(head));

return ;

}

点双连通分支

去掉桥,其余的连通分支就是边双连通分支了。一个有桥的连通图要变成边双连通图的话,把双连通子图 收缩为一个点,形成一颗树。需要加的边为(leaf+1)/2 (leaf 为叶子结点个数)

参考题目链接：

POJ 3177 Redundant Paths

给定一个连通的无向图 G,至少要添加几条边,才能使其变为双连通图。

const int MAXN = 5010; // 点数

const int MAXM = 20010; // 边数,因为是无向图,所以这个值要\*2

struct Edge

{

int to, next;

bool cut; // 是否是桥标记

}edge[MAXM];

int head[MAXN], tot;

int Low[MAXN], DFN[MAXN], Stack[MAXN], Belong[MAXN]; //Belong数组的值是1~block

int Index,top;

int block; // 边双连通块数

bool Instack[MAXN];

int bridge; // 桥的数目

void addedge(int u, int v)

{

edge[tot].to = v;

edge[tot].next = head[u];

edge[tot].cut=false;

head[u] = tot++;

return ;

}

void Tarjan(int u, int pre)

{

int v;

Low[u] = DFN[u] = ++Index;

Stack[top++] = u;

Instack[u] = true;

for (int i = head[u]; i != -1; i = edge[i].next)

{

v = edge[i].to;

if (v == pre)

{

continue;

}

if (!DFN[v])

{

Tarjan(v, u);

if (Low[u] > Low[v])

{

Low[u] = Low[v];

}

if (Low[v] > DFN[u])

{

bridge++;

edge[i].cut = true;

edge[i^1].cut = true;

}

}

else if (Instack[v] && Low[u] > DFN[v])

{

Low[u] = DFN[v];

}

}

if (Low[u] == DFN[u])

{

block++;

do

{

v = Stack[--top]; Instack[v] = false;

Belong[v] = block;

}

while (v != u);

}

return ;

}

void init()

{

tot = 0;

memset(head, -1, sizeof(head));

return ;

}

int du[MAXN]; // 缩点后形成树,每个点的度数

void solve(int n)

{

memset(DFN, 0, sizeof(DFN));

memset(Instack, false, sizeof(Instack));

Index = top = block = 0;

Tarjan(1,0);

int ans = 0;

memset(du, 0, sizeof(du));

for (int i = 1; i <= n; i++)

{

for (int j = head[i]; j != -1; j = edge[j].next)

{

if (edge[j].cut)

{

du[Belong[i]]++;

}

}

}

for (int i = 1; i <= block; i++)

{

if(du[i]==1)

{

ans++;

}

}

// 找叶子结点的个数ans,构造边双连通图需要加边(ans+1)/2

printf("%d\n", (ans + 1) / 2);

}

int main()

{

int n, m;

int u, v;

while (scanf("%d%d", &n, &m) == 2)

{

init();

while (m--)

{

scanf("%d%d",&u,&v);

addedge(u,v);

addedge(v,u);

}

solve(n);

}

return 0;

}

边双连通分支

对于点双连通分支,实际上在求割点的过程中就能顺便把每个点双连通分支求出。建立一个栈,存储 当前双连通分支,在搜索图时,每找到一条树枝边或后向边(非横叉边),就把这条边加入栈中。如果遇到某时满足 DFS(u)<=Low(v),说明u是一个割点,同时把边从栈顶一个个取出,直到遇到了边(u,v), 取出的这些边与其关联的点,组成一个点双连通分支。割点可以属于多个点双连通分支,其余点和每条边只属于且属于一个点双连通分支。

参考题目链接：

POJ 2942 Knights of the Round Table

奇圈,二分图判断的染色法,求点双连通分支

/\*

\* POJ 2942 Knights of the Round Table

\* 亚瑟王要在圆桌上召开骑士会议,为了不引发骑士之间的冲突,

\* 并且能够让会议的议题有令人满意的结果,每次开会前都必须对出席会议的骑士有如下要求:

\* 1、 相互憎恨的两个骑士不能坐在直接相邻的2个位置;

\* 2、 出席会议的骑士数必须是奇数,这是为了让投票表决议题时都能有结果。

\* 注意:1、所给出的憎恨关系一定是双向的,不存在单向憎恨关系。

\* 2、由于是圆桌会议,则每个出席的骑士身边必定刚好有2个骑士。

\* 即每个骑士的座位两边都必定各有一个骑士。

\* 3、一个骑士无法开会,就是说至少有3个骑士才可能开会。

\* 首先根据给出的互相憎恨的图中得到补图。

\* 然后就相当于找出不能形成奇圈的点。

\* 利用下面两个定理:

\* (1)如果一个双连通分量内的某些顶点在一个奇圈中(即双连通分量含有奇圈), 那么这个双连通分量的其他顶点也在某个奇圈中;

\* (2)如果一个双连通分量含有奇圈,则他必定不是一个二分图。反过来也成立,这是一个充要条件。

\* 所以本题的做法,就是对补图求点双连通分量。然后对于求得的点双连通分量,使用染色法判断是不是二分图,不是二分图,这个双连通分量的点是可以存在的

\*/

const int MAXN = 1010;

const int MAXM = 2000010;

struct Edge

{

int to, next;

} edge[MAXM];

int head[MAXN], tot;

int Low[MAXN], DFN[MAXN], Stack[MAXN], Belong[MAXN];

int Index,top;

int block; // 点双连通分量的个数

bool Instack[MAXN];

bool can[MAXN];

bool ok[MAXN]; // 标记

int tmp[MAXN]; // 暂时存储双连通分量中的点

int cc; // tmp的计数

int color[MAXN];// 染色

void addedge(int u, int v)

{

edge[tot].to = v;

edge[tot].next = head[u];

head[u] = tot++;

return ;

}

bool dfs(int u, int col) // 染色判断二分图

{

color[u] = col;

for (int i = head[u]; i != -1; i = edge[i].next)

{

int v = edge[i].to;

if (!ok[v])

{

continue;

}

if (color[v] != -1)

{

if (color[v]==col)

{

return false;

}

continue;

}

if (!dfs(v,!col))

{

return false;

}

}

return true;

}

void Tarjan(int u, int pre)

{

int v;

Low[u] = DFN[u] = ++Index;

Stack[top++] = u;

Instack[u] = true;

for (int i = head[u]; i != -1; i = edge[i].next)

{

v = edge[i].to;

if (v == pre)

{

continue;

}

if (!DFN[v])

{

Tarjan(v, u);

if (Low[u] > Low[v])

{

Low[u] = Low[v];

}

if (Low[v] >= DFN[u])

{

block++;

int vn;

cc = 0;

memset(ok, false, sizeof(ok));

do

{

vn = Stack[--top];

Belong[vn] = block;

Instack[vn] = false;

ok[vn] = true;

tmp[cc++] = vn;

}

while (vn!=v);

ok[u] = 1;

memset(color, -1, sizeof(color));

if (!dfs(u,0))

{

can[u] = true;

while (cc--)

{

can[tmp[cc]] = true;

}

}

}

}

else if (Instack[v] && Low[u] > DFN[v])

{

Low[u] = DFN[v];

}

}

}

void solve(int n)

{

memset(DFN, 0, sizeof(DFN));

memset(Instack, false, sizeof(Instack));

Index = block = top = 0;

memset(can, false, sizeof(can));

for (int i = 1; i <= n; i++)

{

if (!DFN[i])

{

Tarjan(i, -1);

}

}

int ans = n;

for (int i = 1; i <= n; i++)

{

if(can[i])

{

ans--;

}

}

printf("%d\n", ans);

}

void init()

{

tot = 0;

memset(head, -1, sizeof(head));

}

int g[MAXN][MAXN];

int main()

{

int n, m;

int u, v;

while (scanf("%d%d", &n, &m) == 2)

{

if (n == 0 && m == 0)

{

break;

}

init();

memset(g, 0, sizeof(g));

while (m--)

{

scanf("%d%d", &u, &v);

g[u][v] = g[v][u] = 1;

}

for (int i = 1; i <= n; i++)

{

for (int j = 1; j <= n; j++)

{

if(i != j && g[i][j] == 0)

{

addedge(i, j);

}

}

}

solve(n);

}

return 0;

}

弦图判断

/\*

\* 弦图判断

\* INIT: g[][]置为邻接矩阵;

\* CALL: mcs(n); peo(n);

\* 第一步: 给节点编号 mcs(n)

\* 设已编号的节点集合为A, 未编号的节点集合为B

\* 开始时A为空, B包含所有节点.

\* for num=n-1 downto 0 do {

\* 在B中找节点x, 使与x相邻的在A集合中的节点数最多,

\* 将x编号为num，并从B移入A。

\* }

\* 第二部：检查peo(n)

\* for num=0 to n-1 do {

\* 对编号为num的点x，设所有编号>num且与x相邻的点集为C

\* 在C中找出编号最小的节点y，

\* 若C中存在x != y，使得y与x之间无边，则此图不是弦图。

\* }

\* 检查完毕, 则此图是弦图.

\*/

const int V = 10010;

int g[V][V], order[V], inv[V], tag[V];

void mcs(int n)

{

int i, j, k;

memset(tag, 0, sizeof(tag));

memset(order, -1, sizeof(order));

for (i = n - 1; i >= 0; i--)

{ // vertex:0~n-1

for (j = 0; order[j] >= 0; j++);

for (k = j + 1; k < n; k++)

{

if (order[k] < 0 && tag[k] > tag[j])

{

j = k;

}

}

order[j] = i, inv[i] = j;

for (k = 0; k < n; k++)

{

if (g[j][k])

{

tag[k]++;

}

}

}

return ;

}

int peo(int n)

{

int i, j, k, w, min;

for (i = n - 2; i >= 0; i--)

{

j = inv[i], w = -1, min = n;

for (k = 0; k < n; k++)

{

if (g[j][k] && order[k] > order[j] && order[k] < min)

{

min = order[k], w=k;

}

}

if (w < 0)

{

continue;

}

for (k = 0; k < n; k++)

{

if (g[j][k] && order[k] > order[w] && !g[k][w])

{

return 0; // no

}

}

}

return 1; // yes

}

弦图的PERFECT ELIMINATION点排列

/\*

\* INIT: g[][]置为邻接矩阵;

\* CALL: cardinality(n);tag[i]为排列中第i个点的标号;

\*/

const int V = 10010;

int tag[V], g[V][V], deg[V], vis[V];

void cardinality(int n)

{

int i, j, k;

memset(deg, 0, sizeof(deg));

memset(vis, 0, sizeof(vis));

for (i = n - 1; i >= 0; i--)

{

for (j = 0, k = -1; j < n; j++)

{

if (0 == vis[j])

{

if (k == -1 || deg[j] > deg[k])

{

k = j;

}

}

}

vis[k] = 1, tag[i] = k;

for (j = 0; j<n; j++)

{

if (0 == vis[j] && g[k][j])

{

deg[j]++;

}

}

}

return ;

}

稳定婚姻问题

/\*

\* 稳定婚姻问题O(n^2)

\*/

const int N = 1001;

struct People

{

bool state;

int opp, tag;

int list[N]; // man使用

int priority[N]; // woman使用，有必要的话可以和list合并，以节省空间

void Init()

{

state = tag = 0;

}

} man[N], woman[N];

struct R

{

int opp;

int own;

} requst[N];

int n;

void Input();

void Output();

void stableMatching();

int main()

{

Input();

stableMatching();

Output();

return 0;

}

void Input()

{

scanf("%d\n", &n);

int i, j, ch;

for (i = 0; i < n; ++i)

{

man[i].Init();

for(j = 0; j < n; ++j)

{ // 按照man的意愿递减排序

scanf("%d", &ch);

man[i].list[j] = ch - 1;

}

}

for (i = 0; i < n; ++i)

{

woman[i].Init();

for (j = 0; j < n; ++j)

{ // 按照woman的意愿递减排序,但是，存储方法与man不同

scanf("%d", &ch);

woman[i].priority[ch - 1] = j;

}

}

return ;

}

void stableMatching()

{

int k;

for (k = 0; k < n; ++k)

{

int i, id = 0;

for (i = 0; i < n; ++i)

{

if (man[i].state == 0)

{

requst[id].opp = man[i].list[man[i].tag];

requst[id].own = i;

man[i].tag += 1;

++id;

}

}

if (id == 0)

{

break;

}

for (i = 0; i < id; i++)

{

if (woman[requst[i].opp].state == 0)

{

woman[requst[i].opp].opp = requst[i].opp;

woman[requst[i].opp].state = 1;

man[requst[i].own].state = 1;

man[requst[i].own].opp = requst[i].opp;

}

else

{

if (woman[requst[i].opp].priority[woman[requst[i].opp].opp] >woman[requst[i].opp].priority[requst[i].own])

{

man[woman[requst[i].opp].opp].state = 0;

woman[requst[i].opp].opp = requst[i].own;

man[requst[i].own].state = 1;

man[requst[i].own].opp = requst[i].opp;

}

}

}

}

return ;

}

void Output()

{

for (int i = 0; i < n; i++)

{

printf("%d\n", man[i].opp + 1);

}

return ;

}

无向图连通分支

/\*

\* 无向图连通分支(dfs/bfs邻接阵)

\* DFS / BFS / 并查集

\* 返回分支数,id返回1.分支数的值

\* 传入图的大小n和邻接阵mat,不相邻点边权0

\*/

#define MAXN 100

void search(int n, int mat[][MAXN], int\* dfn, int\* low, int now, int& cnt, int& tag, int\* id, int\* st, int& sp)

{

int i, j;

dfn[st[sp++]=now] = low[now] = ++cnt;

for (i = 0; i < n; i++)

{

if (mat[now][i])

{

if (!dfn[i])

{

search(n, mat, dfn, low, i, cnt, tag, id, st, sp);

if (low[i] < low[now])

{

low[now]=low[i];

}

}

else if (dfn[i] < dfn[now])

{

for (j = 0; j < sp && st[j] != i; j++)

{

if (j < cnt && dfn[i] < low[now])

{

low[now] = dfn[i];

}

}

}

}

}

if (low[now] == dfn[now])

{

for (tag++; st[sp] != now; id[st[--sp]] = tag);

}

}

int find\_components(int n, int mat[][MAXN], int\* id)

{

int ret = 0, i, cnt, sp, st[MAXN], dfn[MAXN], low[MAXN];

for (i = 0; i < n; dfn[i++] = 0);

for (sp = cnt = i = 0; i < n; i++)

{

if (!dfn[i])

{

search(n, mat, dfn, low, i, cnt, ret, id, st, sp);

}

}

return ret;

}

Floyd求最小环

令e(u, v)表示u和v之间的连边，令min(u, v)表示删除u和v之间的连边之后u和v之间的最短路, 最小环则是min(u, v) + e(u, v). 时间复杂度是 O(EV^2).

一个环中的最大结点为k(编号最大), 与他相连的两个点为i, j, 这个环的最短长度为g[i][k]+g[k][j]+i到j的路径中所有结点编号都小于k的最短路径长度. 根据floyd的原理, 在最外层循环做了k-1次之后, dist[i][j]则代表了i到j的路径中所有结点编号都小于k的最短路径综上所述,该算法一定能找到图中最小环。

const int INF = 0x3f3f3f3f;

const int MAXN = 110;

int n, m; // n:节点个数, m:边的个数

int g[MAXN][MAXN]; // 无向图

int dist[MAXN][MAXN]; // 最短路径

int r[MAXN][MAXN]; // r[i][j]: i到j的最短路径的第一步

int out[MAXN], ct; // 记录最小环

int solve(int i, int j, int k)

{ // 记录最小环

ct = 0;

while (j != i)

{

out[ct++] = j;

j = r[i][j];

}

out[ct++] = i;

out[ct++] = k;

return 0;

}

int main()

{

while (scanf("%d%d", &n, &m) != EOF)

{

int i, j, k;

for (i = 0; i < n; i++)

{

for (j = 0; j < n; j++)

{

g[i][j] = INF;

r[i][j] = i;

}

}

for (i = 0; i < m; i++)

{

int x, y, l;

scanf("%d%d%d", &x, &y, &l);

--x;

--y;

if (l < g[x][y])

{

g[x][y] = g[y][x] = l;

}

}

memmove(dist, g, sizeof(dist));

int Min = INF; // 最小环

for (k = 0; k < n; k++)

{ // Floyd

for (i = 0; i < k; i++) // 一个环中的最大结点为k(编号最大)

{

if (g[k][i] < INF)

{

for (j = i + 1; j < k; j++)

{

if (dist[i][j] < INF && g[k][j] < INF && Min > dist[i][j] + g[k][i] + g[k][j])

{

Min = dist[i][j] + g[k][i] + g[k][j];

solve(i, j, k); // 记录最小环

}

}

}

}

for (i = 0; i < n; i++)

{

if (dist[i][k] < INF)

{

for (j = 0; j < n; j++)

{

if (dist[k][j] < INF && dist[i][j] > dist[i][k]+dist[k][j])

{

dist[i][j] = dist[i][k] + dist[k][j];

r[i][j] = r[k][j];

}

}

}

}

}

if (Min < INF)

{

for (ct--; ct >= 0; ct--)

{

printf("%d", out[ct] + 1);

if (ct)

{

printf(" ");

}

}

}

else

{

printf("No solution.");

}

printf("\n");

}

return 0;

}

2-SAT

/\*

\* 2-sat 问题

\* N个集团,每个集团2个人,现在要想选出尽量多的人,

\* 且每个集团只能选出一个人。如果两人有矛盾,他们不能同时被选中

\* 问最多能选出多少人

\*/

const int MAXN = 3010;

int n, m;

int g[3010][3010], ct[3010], f[3010];

int x[3010], y[3010];

int prev[MAXN], low[MAXN], stk[MAXN], sc[MAXN];

int cnt[MAXN];

int cnt0, ptr, cnt1;

void dfs(int w)

{

int min(0);

prev[w] = cnt0++;

low[w] = prev[w];

min = low[w];

stk[ptr++] = w;

for (int i = 0; i < ct[w]; ++i)

{

int t = g[w][i];

if (prev[t] == -1)

{

dfs(t);

}

if (low[t] < min)

{

min = low[t];

}

}

if (min < low[w])

{

low[w] = min;

return ;

}

do

{

int v = stk[--ptr];

sc[v] = cnt1;

low[v] = MAXN;

} while(stk[ptr] != w);

++cnt1;

return ;

}

void Tarjan(int N)

{ // 传入N为点数,结果保存在sc数组中,同一标号的点在同一个强连通分量内,

// 强连通分量数为cnt1

cnt0 = cnt1 = ptr = 0;

int i;

for (i = 0; i < N; ++i)

{

prev[i] = low[i] = -1;

}

for (i = 0; i < N; ++i)

{

if (prev[i] == -1)

{

dfs(i);

}

}

return ;

}

int solve()

{

Tarjan(n);

for (int i = 0; i < n; i++)

{

if (sc[i] == sc[f[i]])

{

return 0;

}

}

return 1;

}

int check(int Mid)

{

for (int i = 0; i < n; i++)

{

ct[i] = 0;

}

for (int i = 0; i < Mid; i++)

{

g[f[x[i]]][ct[f[x[i]]]++] = y[i];

g[f[y[i]]][ct[f[y[i]]]++] = x[i];

}

return solve();

}

int main()

{

while (scanf("%d%d", &n, &m) != EOF && n + m)

{

for (int i = 0; i < n; i++)

{

int p, q;

scanf("%d%d", &p, &q);

f[p] = q, f[q] = p;

}

for (int i = 0; i < m; i++)

{

scanf("%d%d", &x[i], &y[i]);

}

n \*= 2;

int Min = 0, Max = m + 1;

while (Min + 1 < Max)

{

int Mid = (Min + Max) / 2;

if (check(Mid))

{

Min = Mid;

}

else

{

Max = Mid;

}

}

printf("%d\n", Min);

}

return 0;

}

树的重心

typedef long long ll;

typedef pair<int, int> pll;

const int INF = 0x3f3f3f3f;

const int MAXN = 100000 + 10;

int n;

/\* 树的重心

\* 初始化 vis[] son[] 为 0

\* 初始化 sz 为 INF

\*/

int zx, sz;

int son[MAXN], vis[MAXN];

vector<pll> edge[MAXN];

void init()

{

for (int i = 1; i <= n; i++)

{

edge[i].clear();

}

memset(vis, 0, sizeof(vis));

sz = INF;

zx = -1;

}

void dfs(int r)

{

vis[r] = 1;

son[r] = 0;

int tmp = 0;

for (int i = 0; i < edge[r].size(); i++)

{

int v = edge[r][i].second;

if (!vis[v])

{

dfs(v);

son[r] += son[v] + 1;

tmp = max(tmp, son[v] + 1);

}

}

tmp = max(tmp, n - son[r] - 1);

if (tmp < sz)

{

zx = r;

sz = tmp;

}

}

Network 网络流

二分图匹配

匈牙利算法

邻接矩阵+DFS

/\*

\* 初始化:g[][]两边顶点的划分情况

\* 建立g[i][j]表示i->j的有向边就可以了,是左边向右边的匹配

\* g没有边相连则初始化为0

\* uN是匹配左边的顶点数,vN是匹配右边的顶点数

\* 调用:res=hungary();输出最大匹配数

\* 优点:适用于稠密图,DFS找增广路,实现简洁易于理解

\* 时间复杂度:O(VE)

\*/

//顶点编号从0开始的

const int MAXN = 510;

int uN, vN; // u,v的数目,使用前面必须赋值

int g[MAXN][MAXN]; // 邻接矩阵

int linker[MAXN];

bool used[MAXN];

bool dfs(int u)

{

for (int v = 0; v < vN; v++)

{

if (g[u][v] && !used[v])

{

used[v] = true;

if (linker[v] == -1 || dfs(linker[v]))

{

linker[v] = u;

return true;

}

}

}

return false;

}

int hungary()

{

int res = 0;

memset(linker, -1, sizeof(linker));

for (int u = 0; u < uN; u++)

{

memset(used, false, sizeof(used));

if (dfs(u))

{

res++;

}

}

return res;

}

邻接表+DFS

/\*

\* 使用前用init()进行初始化,给uN赋值

\* 加边使用函数addedge(u,v)

\*/

const int MAXN = 5010; // 点数的最大值

const int MAXM = 50010; // 边数的最大值

struct Edge

{

int to, next;

} edge[MAXM];

int head[MAXN], tot;

void init()

{

tot = 0;

memset(head, -1, sizeof(head));

return ;

}

void addedge(int u, int v)

{

edge[tot].to = v;

edge[tot].next = head[u];

head[u] = tot++;

return ;

}

int linker[MAXN];

bool used[MAXN];

int uN;

bool dfs(int u)

{

for (int i = head[u]; i != -1; i = edge[i].next)

{

int v = edge[i].to;

if (!used[v])

{

used[v] = true;

if (linker[v] == -1 || dfs(linker[v]))

{

linker[v] = u;

return true;

}

}

}

return false;

}

int hungary()

{

int res = 0;

memset(linker, -1, sizeof(linker));

for (int u = 0; u < uN; u++) // 点的编号0~uN-1

{

memset(used, false, sizeof(used));

if (dfs(u))

{

res++;

}

}

return res;

}

Hopcroft-Carp算法

邻接矩阵+DFS

/\*

\* INIT: g[][]邻接矩阵；

\* CALL: res = MaxMatch(); Nx, Ny要初始化！！！

\* 时间复杂度: O(V^0.5 \* E)

\*/

const int MAXN = 3001;

const int INF = 1 << 28;

int g[MAXN][MAXN], Mx[MAXN], My[MAXN], Nx, Ny;

int dx[MAXN], dy[MAXN], dis;

bool vst[MAXN];

bool searchP()

{

queue<int> Q;

dis = INF;

memset(dx, -1, sizeof(dx));

memset(dy, -1, sizeof(dy));

for (int i = 0; i < Nx; i++)

{

if (Mx[i] == -1)

{

Q.push(i); dx[i] = 0;

}

}

while (!Q.empty())

{

int u = Q.front();

Q.pop();

if (dx[u] > dis)

{

break;

}

for (int v = 0; v < Ny; v++)

{

if (g[u][v] && dy[v] == -1)

{

dy[v] = dx[u]+1;

if (My[v] == -1)

{

dis = dy[v];

}

else

{

dx[My[v]] = dy[v] + 1;

Q.push(My[v]);

}

}

}

}

return dis != INF;

}

bool DFS(int u)

{

for (int v = 0; v < Ny; v++)

{

if (!vst[v] && g[u][v] && dy[v] == dx[u] + 1)

{

vst[v] = 1;

if (My[v] != -1 && dy[v] == dis)

{

continue;

}

if (My[v] == -1 || DFS(My[v]))

{

My[v] = u; Mx[u] = v;

return 1;

}

}

}

return 0;

}

int MaxMatch()

{

int res = 0;

memset(Mx, -1, sizeof(Mx));

memset(My, -1, sizeof(My));

while (searchP())

{

memset(vst, 0, sizeof(vst));

for (int i = 0; i < Nx; i++)

{

if (Mx[i] == -1 && DFS(i))

{

res++;

}

}

}

return res;

}

邻接表+DFS

/\*

\* 复杂度O(sqrt(n)\*E)

\* 邻接表存图,vector实现

\* vector先初始化,然后假如边

\* uN为左端的顶点数,使用前赋值(点编号0开始)

\*/

const int MAXN = 3000;

const int INF = 0x3f3f3f3f;

vector<int>G[MAXN];

int uN;

int Mx[MAXN], My[MAXN];

int dx[MAXN], dy[MAXN];

int dis;

bool used[MAXN];

bool SearchP()

{

queue<int>Q;

dis = INF;

memset(dx, -1, sizeof(dx));

memset(dy, -1, sizeof(dy));

for (int i = 0 ; i < uN; i++)

{

if(Mx[i] == -1)

{

Q.push(i);

dx[i] = 0;

}

}

while (!Q.empty())

{

int u = Q.front();

Q.pop();

if (dx[u] > dis)

{

break;

}

int sz = (int)G[u].size();

for (int i = 0; i < sz; i++)

{

int v = G[u][i];

if (dy[v] == -1)

{

dy[v] = dx[u] + 1;

if (My[v] == -1)

{

dis = dy[v];

}

else

{

dx[My[v]] = dy[v] + 1;

Q.push(My[v]);

}

}

}

}

return dis != INF;

}

bool DFS(int u)

{

int sz = (int)G[u].size();

for (int i = 0; i < sz; i++)

{

int v = G[u][i];

if (!used[v] && dy[v] == dx[u] + 1)

{

used[v] = true;

if (My[v] != -1 && dy[v] == dis)

{

continue;

}

if (My[v] == -1 || DFS(My[v]))

{

My[v] = u;

Mx[u] = v;

return true;

}

}

}

return false;

}

int MaxMatch()

{

int res = 0;

memset(Mx, -1, sizeof(Mx));

memset(My, -1, sizeof(My));

while (SearchP())

{

memset(used, false, sizeof(used));

for (int i = 0; i < uN; i++)

{

if(Mx[i] == -1 && DFS(i))

{

res++;

}

}

}

return res;

}

二分图最佳匹配

Kuhn Munkras算法

/\*

\* 邻接距阵形式,复杂度O(m\*m\*n) 返回最佳匹配值,传入二分图大小m,n

\* 邻接距阵mat,表示权,match1,match2返回一个最佳匹配,未匹配顶点

\* match值为-1,一定注意m<=n,否则循环无法终止,最小权匹配可将权值

\* 取相反数

\* 初始化:for (i = 0; i < MAXN; ++i)

\* for (j = 0; j < MAXN ; ++j)

\* mat[i][j] = -inf;

\* 对于存在的边:mat[i][j] = val ; // 注意,不能有负值

\*/

#define MAXN 310

#define inf 1000000000

#define \_clr(x) memset(x, -1, sizeof(int) \* MAXN)

int kuhn\_munkras(int m, int n, int mat[][MAXN], int \*match\_1, int \*match\_2)

{

int s[MAXN], t[MAXN], l\_1[MAXN], l\_2[MAXN];

int p, q, ret = 0;

int i, j, k;

for (i = 0; i < m; i++)

{

for (l\_1[i] = -inf, j = 0; j < n; j++)

{

l\_1[i] = mat[i][j] > l\_1[i] ? mat[i][j] : l\_1[i];

}

if (l\_1[i] == -inf)

{

return -1; // 无结果

}

}

for (i = 0; i < n; l\_2[i++] = 0);

for (\_clr(match\_1), \_clr(match\_2), i = 0; i < m; i++)

{

for (\_clr(t), s[p = q = 0] = i; p <= q && match\_1[i] < 0; p++)

{

for (k = s[p], j = 0; j < n && match\_1[i] < 0; p++)

{

if (l\_1[k] + l\_2[j] == mat[k][j] && t[j] < 0)

{

s[++q] = match\_2[j], t[j] = k;

if (s[q] < 0)

{

for (p = j; p >= 0; j = p)

{

match\_2[j] = k = t[j];

p = match\_1[k];

match\_1[k] = j;

}

}

}

}

}

if (match\_1[i] < 0)

{

for (i--, p = inf, k = 0; k <= q; k++)

{

for (j = 0; j < n; j++)

{

if (t[j] < 0 && l\_1[s[k]] + l\_2[j] - mat[s[k]][j] < p)

{

p = l\_1[s[k]] + l\_2[j] - mat[s[k]][j];

}

}

}

for (j = 0; j < n; l\_2[j] += t[j] < 0 ? 0 : p, j++);

for (k = 0; k <= q; l\_1[s[k++]] -= p);

}

}

for (i = 0; i < m; i++)

{ // if处理无匹配的情况!!

if (match\_1[i] < 0) // ???

{

return -1;

}

if (mat[i][match\_1[i]] <= -inf) // ???

{

return -1;

}

ret += mat[i][match\_1[i]];

}

return ret;

}

二分图多重匹配

const int MAXN = 1010;

const int MAXM = 510;

int uN, vN;

int g[MAXN][MAXM];

int linker[MAXM][MAXN];

bool used[MAXM];

int num[MAXM]; // 右边最大的匹配数

bool dfs(int u)

{

for (int v = 0; v < vN; v++)

{

if (g[u][v] && !used[v])

{

used[v] = true;

if (linker[v][0] < num[v])

{

linker[v][++linker[v][0]] = u;

return true;

}

for (int i = 1; i <= num[0]; i++)

{

if (dfs(linker[v][i]))

{

linker[v][i] = u;

return true;

}

}

}

}

return false;

}

int hungary()

{

int res = 0;

for (int i = 0; i < vN; i++)

{

linker[i][0] = 0;

}

for (int u = 0; u < uN; u++)

{

memset(used, false, sizeof(used));

if (dfs(u))

{

res++;

}

}

return res;

}

无向图最小割

/\*

\* INIT: 初始化邻接矩阵g[][]

\* CALL: res = mincut(n);

\* 注: Stoer-Wagner Minimum Cut;

\* 找边的最小集合，若其被删去则图变得不连通（我们把这种形式称为最小割问题）

\*/

#define typec int // type of res

const typec inf = 0x3f3f3f3f; // max of res

const typec maxw = 1000; // maximum edge weight

const typec V = 10010;

typec g[V][V], w[V];

int a[V], v[V], na[V];

typec minCut(int n)

{

int i, j, pv, zj;

typec best = maxw \* n \* n;

for (i = 0; i < n; i++)

{

v[i] = i; // vertex: 0 ~ n-1

}

while (n > 1)

{

for (a[v[0]] = 1, i = 1; i < n; i++)

{

a[v[i]] = 0;

na[i - 1] = i;

w[i] = g[v[0]][v[i]];

}

for (pv = v[0], i = 1; i < n; i++)

{

for (zj = -1, j = 1; j < n; j++)

{

if (!a[v[j]] && (zj < 0 || w[j] > w[zj]))

{

zj = j;

}

}

a[v[zj]] = 1;

if (i == n - 1)

{

if (best > w[zj])

{

best = w[zj];

}

for (i = 0; i < n; i++)

{

g[v[i]][pv] = g[pv][v[i]] += g[v[zj]][v[i]];

}

v[zj] = v[--n];

break;

}

pv = v[zj];

for (j = 1; j < n; j++)

{

if(!a[v[j]])

{

w[j] += g[v[zj]][v[j]];

}

}

}

}

return best;

}

有上下界的最小（最大）流

/\*

\* 有上下界的最小(最大)流

\* INIT: up[][]为容量上界; low[][]为容量下界;

\* CALL: mf = limitflow(n,src,sink); flow[][]为流量分配;

\* 另附: 循环流问题

\* 描述: 无源无汇的网络N,设N是具有基础有向图D=(V,A)的网络.

\* l和c分别为容量下界和容量上界. 如果定义在A上的函数

\* f满足: f(v, V) = f(V, v). V中任意顶点v,

\* l(a)<=f(a)<=c(a),则称f为网络N的循环流.

\* 解法: 添加一个源s和汇t,对于每个下限容量l不为0的边(u, v),

\* 将其下限去掉,上限改为c-l,增加两条边(u, t),(s, v),

\* 容量均为l.原网络存在循环流等价于新网络最大流是满流.

\*/

const int inf = 0x3f3f3f3f;

const int N = 1010;

int up[N][N], low[N][N], flow[N][N];

int pv[N], que[N], d[N];

void maxflow(int n, int src, int sink)

{

// BFS增广, O(E \* maxflow)

int p, q, t, i, j;

do

{

for (i = 0; i < n; pv[i++] = 0);

pv[t = src] = src + 1;

d[t] = inf;

for (p = q = 0; p <= q && !pv[sink]; t = que[p++])

{

for (i = 0; i < n; i++)

{

if (!pv[i] && up[t][i] && (j = up[t][i] - flow[t][i]) > 0)

{

pv[que[q++] = i] = +t + 1, d[i] = d[t] < j ? d[t] : j;

}

else if (!pv[i] && up[i][t] && (j = flow[i][t]) > 0)

{

pv[que[q++] = i] = -t - 1, d[i] = d[t] < j ? d[t] : j;

}

}

}

for (i = sink; pv[i] && i != src;)

{

if (pv[i] > 0)

{

flow[pv[i] - 1][i] += d[sink], i = pv[i] - 1;

}

else

{

flow[i][-pv[i] - 1] -= d[sink], i = -pv[i] - 1;

}

}

}

while (pv[sink]);

return ;

}

int limitflow(int n, int src, int sink)

{

int i, j, sk, ks;

if (src == sink)

{

return inf;

}

up[n][n + 1] = up[n + 1][n] = up[n][n] = up[n + 1][n + 1] = 0;

for (i = 0; i < n; i++)

{

up[n][i] = up[i][n] = up[n+1][i] = up[i][n+1] = 0;

for (j = 0; j < n; j++)

{

up[i][j] -= low[i][j];

up[n][i] += low[j][i];

up[i][n + 1] += low[i][j];

}

}

sk = up[src][sink];

ks = up[sink][src];

up[src][sink] = up[sink][src] = inf;

maxflow(n + 2, n, n + 1);

for (i = 0; i < n; i++)

{

if (flow[n][i] < up[n][i])

{

return -1;

}

}

flow[src][sink] = flow[sink][src] = 0;

up[src][sink] = sk;

up[sink][src] = ks; // !min:src<-sink; max:src->sink;

maxflow(n, sink, src);

for (i = 0; i < n; i++)

{

for (j = 0; j < n; j++)

{

up[i][j] += low[i][j];

flow[i][j] += low[i][j];

}

}

for (j = i = 0; i < n; j += flow[src][i++]);

return j;

}

最佳边割集

#define MAXN 100

#define inf 1000000000

int max\_flow(int n, int mat[][MAXN], int source, int sink)

{

int v[MAXN], c[MAXN], p[MAXN], ret = 0, i, j;

for (;;)

{

for (i = 0; i < n; i++)

{

v[i] = c[i] = 0;

}

for (c[source] = inf; ;)

{

for (j = -1, i = 0; i < n; i++)

{

if (!v[i] && c[i] && (j == -1 || c[i] > c[j]))

{

j = i;

}

}

if (j < 0)

{

return ret;

}

if (j == sink)

{

break;

}

for (v[j] = 1, i = 0; i < n; i++)

{

if (mat[j][i] > c[i] && c[j] > c[i])

{

c[i] = mat[j][i] < c[j] ? mat[j][i] : c[j], p[i] = j;

}

}

}

for (ret += j = c[i = sink]; i != source; i = p[i])

{

mat[p[i]][i] -= j, mat[i][p[i]] += j;

}

}

}

int best\_edge\_cut(int n, int mat[][MAXN], int source, int sink, int set[][2], int &mincost)

{

int m0[MAXN][MAXN], m[MAXN][MAXN], i, j, k, l, ret = 0, last;

if (source == sink)

{

return -1;

}

for (i = 0; i < n; i++)

{

for (j = 0; j < n; j++)

{

m0[i][j] = mat[i][j];

}

}

for (i = 0; i < n; i++)

{

for (j = 0; j < n; j++)

{

m[i][j] = m0[i][j];

}

}

mincost = last = max\_flow(n, m, source, sink);

for (k = 0; k < n && last; k++)

{

for (l = 0; l < n && last; l++)

{

if (m0[k][l])

{

for (i = 0; i < n + n; i++)

{

for (j = 0; j < n + n; j++)

{

m[i][j] = m0[i][j];

}

}

m[k][l] = 0;

if (max\_flow(n, m, source, sink) == last - mat[k][l])

{

set[ret][0] = k;

set[ret++][1] = l;

m0[k][l] = 0;

last -= mat[k][l];

}

}

}

}

return ret;

}

最佳点割集

#define MAXN 100

#define inf 1000000000

int max\_flow(int n, int mat[][MAXN], int source, int sink)

{

int v[MAXN], c[MAXN], p[MAXN], ret = 0, i, j;

for (;;)

{

for (i = 0; i < n; i++)

{

v[i] = c[i] = 0;

}

for (c[source] = inf; ;)

{

for (j = -1, i = 0; i < n; i++)

{

if (!v[i] && c[i] && (j == -1 || c[i] > c[j]))

{

j = i;

}

}

if (j < 0)

{

return ret;

}

if (j == sink)

{

break;

}

for (v[j] = 1, i = 0; i < n; i++)

{

if (mat[j][i] > c[i] && c[j] > c[i])

{

c[i] = mat[j][i] < c[j] ? mat[j][i] : c[j], p[i] = j;

}

}

}

for (ret += j = c[i = sink]; i != source; i = p[i])

{

mat[p[i]][i] -= j, mat[i][p[i]] += j;

}

}

}

int best\_vertex\_cut(int n, int mat[][MAXN], int \*cost, int source, int sink, int \*set, int &mincost)

{

int m0[MAXN][MAXN], m[MAXN][MAXN], i, j, k, ret = 0, last;

if (source == sink || mat[source][sink])

{

return -1;

}

for (i = 0; i < n + n; i++)

{

for (j = 0; j < n + n; j++)

{

m0[i][j] = 0;

}

}

for (i = 0; i < n; i++)

{

for (j = 0; j < n; j++)

{

if (mat[i][j])

{

m0[i][n + j] = inf;

}

}

}

for (i = 0; i < n; i++)

{

m0[n + i][i] = cost[i];

}

for (i = 0; i < n + n; i++)

{

for (j = 0; j < n + n; j++)

{

m[i][j] = m0[i][j];

}

}

mincost = last = max\_flow(n + n, m, source, n + sink);

for (k = 0; k < n && last; k++)

{

if (k != source && k != sink)

{

for (i = 0; i < n + n; i++)

{

for (j = 0; j < n + n; j++)

{

m[i][j] = m0[i][j];

}

}

m[n + k][k] = 0;

if (max\_flow(n + n, m, source, n + sink) == last - cost[k])

{

set[ret++] = k;

m0[n + k][k] = 0;

last -= cost[k];

}

}

}

return ret;

}

最小边割集

#define MAXN 100

#define inf 1000000000

int max\_flow(int n, int mat[][MAXN], int source, int sink)

{

int v[MAXN], c[MAXN], p[MAXN], ret = 0, i, j;

for (;;)

{

for (i = 0; i < n; i++)

{

v[i] = c[i] = 0;

}

for (c[source] = inf; ;)

{

for (j = -1, i = 0; i < n; i++)

{

if (!v[i] && c[i] && (j == -1 || c[i] > c[j]))

{

j = i;

}

}

if (j < 0)

{

return ret;

}

if (j == sink)

{

break;

}

for (v[j] = 1, i = 0; i < n; i++)

{

if (mat[j][i] > c[i] && c[j] > c[i])

{

c[i] = mat[j][i] < c[j] ? mat[j][i] : c[j], p[i] = j;

}

}

}

for (ret += j = c[i = sink]; i != source; i = p[i])

{

mat[p[i]][i] -= j, mat[i][p[i]] += j;

}

}

}

int min\_edge\_cut(int n, int mat[][MAXN], int source, int sink, int set[][2])

{

int m0[MAXN][MAXN], m[MAXN][MAXN], i, j, k, l, ret = 0, last;

if (source == sink)

{

return -1;

}

for (i = 0; i < n; i++)

{

for (j = 0; j < n; j++)

{

m0[i][j] = (mat[i][j] != 0);

}

}

for (i = 0; i < n; i++)

{

for (j = 0; j < n; j++)

{

m[i][j] = m0[i][j];

}

}

last = max\_flow(n, m, source, sink);

for (k = 0; k < n && last; k++)

{

for (l = 0; l < n && last; l++)

{

if (m0[k][l])

{

for (i = 0; i < n + n; i++)

{

for (j = 0; j < n + n; j++)

{

m[i][j] = m0[i][j];

}

}

m[k][l] = 0;

if (max\_flow(n, m, source, sink) < last)

{

set[ret][0] = k;

set[ret++][1] = l;

m0[k][l] = 0;

last--;

}

}

}

}

return ret;

}

最小点割集

/\*

\* 最小点割集(点连通度)

\*/

#define MAXN 100

#define inf 1000000000

int max\_flow(int n, int mat[][MAXN], int source, int sink)

{

int v[MAXN], c[MAXN], p[MAXN], ret = 0, i, j;

for (; ;)

{

for (i = 0; i < n; i++)

{

v[i] = c[i] = 0;

}

for (c[source] = inf; ;)

{

for (j = -1, i = 0; i < n; i++)

{

if (!v[i] && c[i] && (j == -1 || c[i] > c[j]))

{

j = i;

}

}

if (j < 0)

{

return ret;

}

if (j == sink)

{

break;

}

for (v[j] = 1, i = 0; i < n; i++)

{

if (mat[j][i] > c[i] && c[j] > c[i])

{

c[i] = mat[j][i] < c[j] ? mat[j][i] : c[j], p[i] = j;

}

}

}

for (ret += j = c[i = sink]; i != source; i = p[i])

{

mat[p[i]][i] -= j, mat[i][p[i]] += j;

}

}

}

int min\_vertex\_cut(int n, int mat[][MAXN], int source, int sink, int \*set)

{

int m0[MAXN][MAXN], m[MAXN][MAXN], i, j, k, ret = 0, last;

if (source == sink || mat[source][sink])

{

return -1;

}

for (i = 0; i < n + n; i++)

{

for (j = 0; j < n + n; j++)

{

m0[i][j] = 0;

}

}

for (i = 0; i < n; i++)

{

for (j = 0; j < n; j++)

{

if (mat[i][j])

{

m0[i][n + j] = inf;

}

}

}

for (i = 0; i < n; i++)

{

m0[n + i][i]=1;

}

for (i = 0; i < n + n; i++)

{

for (j = 0; j < n + n; j++)

{

m[i][j] = m0[i][j];

}

}

last = max\_flow(n + n, m, source, n + sink);

for (k = 0; k < n && last; k++)

{

if (k != source && k != sink)

{

for (i = 0; i < n + n; i++)

{

for (j = 0; j < n + n; j++)

{

m[i][j] = m0[i][j];

}

}

m[n+k][k] = 0;

if (max\_flow(n + n, m, source, n + sink) < last)

{

set[ret++] = k;

m0[n+k][k] = 0;

last--;

}

}

}

return ret;

}

最小路径覆盖

最小路径覆盖O(n^3)路径覆盖:就是在图中找一些路经,使之覆盖了图中的所有顶点,且任何一个顶点有且只有一条路径与之关联。

最小路径覆盖:就是找出最少的路径条数,使之成为P的一个路径覆盖。

路径覆盖与二分图匹配的关系:最小路径覆盖=|P|-最大匹配数;其中最大匹配数的求法是把P中的每个顶点pi分成两个顶点pi’与pi”,如果在p中存在一条pi到pj的边,那么在二分图P’中就有一条连接pi’与pj”的有向边(求二分图匹配时必须是单向边);这里pi’就是p中pi的出边,pj”就是p中pj的一条入边;

有向图: 最小路径覆盖=|P|-最大匹配数;

无向图: 最小路径覆盖=|P|-最大匹配数/2;

最小点集覆盖

结论:一个二分图中的最大匹配数等于这个图中的最小点覆盖数。

Structure 数据结构

Treap

long long gcd(long long a, long long b)

{

if (b == 0)

{

return a;

}

else

{

return gcd(b, a % b);

}

}

const int MAXN = 300010;

int num[MAXN], st[MAXN];

struct Treap

{

int tot1;

int s[MAXN], tot2; // 内存池和容量

int ch[MAXN][2];

int key[MAXN], size[MAXN];

int sum0[MAXN], sum1[MAXN];

int status[MAXN];

void Init()

{

tot1 = tot2 = 0;

size[0] = 0;

ch[0][0] = ch[0][1] = 0;

sum0[0] = sum1[0] = 0;

return ;

}

bool random(double p)

{

return (double)rand() / RAND\_MAX < p;

}

int newnode(int val, int \_status)

{

int r;

if (tot2)

{

r = s[tot2--];

}

else

{

r = ++tot1;

}

size[r] = 1;

key[r] = val;

status[r] = \_status;

ch[r][0] = ch[r][1] = 0;

sum0[r] = sum1[r] = 0; // 需要push\_up

return r;

}

void del(int r)

{

if (!r)

{

return ;

}

s[++tot2] = r;

del(ch[r][0]);

del(ch[r][1]);

return ;

}

void push\_up(int r)

{

int lson = ch[r][0], rson = ch[r][1];

size[r] = size[lson] + size[rson] + 1;

sum0[r] = (int)gcd(sum0[lson], sum0[rson]);

sum1[r] = (int)gcd(sum1[lson], sum1[rson]);

if (status[r] == 0)

{

sum0[r] = (int)gcd(sum0[r], key[r]);

}

else

{

sum1[r] = (int)gcd(sum1[r],key[r]);

}

return ;

}

void merge(int &p, int x, int y)

{

if (!x || !y)

{

p = x | y;

}

else if (random((double)size[x] / (size[x] + size[y])))

{

merge(ch[x][1], ch[x][1], y);

push\_up(p = x);

}

else

{

merge(ch[y][0], x, ch[y][0]);

push\_up(p = y);

}

return ;

}

void split(int p, int &x, int &y, int k)

{

if (!k)

{

x = 0;

y = p;

return ;

}

if (size[ch[p][0]] >= k)

{

y = p;

split(ch[p][0], x, ch[y][0], k);

push\_up(y);

}

else

{

x = p;

split(ch[p][1], ch[x][1], y, k - size[ch[p][0]] - 1);

push\_up(x);

}

return ;

}

void build(int &p, int l, int r)

{

if (l > r)

{

return ;

}

int mid = (l + r) / 2;

p = newnode(num[mid], st[mid]);

build(ch[p][0], l, mid - 1);

build(ch[p][1], mid + 1, r);

push\_up(p);

return ;

}

void debug(int root)

{

if (root == 0)

{

return ;

}

printf("%d 左儿子:%d 右儿子: %d size = %d key = %d\n", root, ch[root][0], ch[root][1], size[root], key[root]);

debug(ch[root][0]);

debug(ch[root][1]);

}

};

Treap T;

char op[10];

int main()

{

// freopen("in.txt", "r", stdin);

// freopen("out.txt", "w", stdout);

int n, q;

while (scanf("%d%d", &n, &q) == 2)

{

int root = 0;

T.Init();

for (int i = 1; i <= n; i++)

{

scanf("%d%d", &num[i], &st[i]);

}

T.build(root, 1, n);

while (q--)

{

scanf("%s", op);

if (op[0] == 'Q')

{

int l, r, s;

scanf("%d%d%d", &l, &r, &s);

int x, y, z;

T.split(root, x, z, r);

T.split(x, x, y, l - 1);

if (s == 0)

{

printf("%d\n", T.sum0[y] == 0 ? -1 : T.sum0[y]);

}

else

{

printf("%d\n", T.sum1[y] == 0 ? -1 : T.sum1[y]);

}

T.merge(x, x, y);

T.merge(root, x, z);

}

else if (op[0] == 'I')

{

int v, s, loc;

scanf("%d%d%d", &loc, &v, &s);

int x, y;

T.split(root, x, y, loc);

T.merge(x, x, T.newnode(v,s));

T.merge(root, x, y);

}

else if (op[0] == 'D')

{

int loc;

scanf("%d", &loc);

int x, y, z;

T.split(root, x, z, loc);

T.split(x, x, y, loc - 1);

T.del(y);

T.merge(root, x, z);

}

else if(op[0] == 'R')

{

int loc;

scanf("%d", &loc);

int x, y, z;

T.split(root, x, z, loc);

T.split(x, x, y, loc - 1);

T.status[y] = 1 - T.status[y];

T.push\_up(y);

T.merge(x, x, y);

T.merge(root, x, z);

}

else

{

int loc, v;

scanf("%d%d", &loc, &v);

int x, y, z;

T.split(root, x, z, loc);

T.split(x, x, y, loc - 1);

T.key[y] = v;

T.push\_up(y);

T.merge(x, x, y);

T.merge(root, x, z);

}

}

}

return 0;

}

二分套二分

/\*

\* 二分套二分

\* 数组A同数组B组合乘积，二分查找第K大

\*/

typedef long long ll;

const int MAXN = 5e4 + 10;

ll N, K;

ll A[MAXN];

ll B[MAXN];

// 查找小于x的元素个数

ll check(ll x)

{

ll j = N, ans = 0;

for (int i = 1; i <= N; i++)

{

for (; j > 0;)

{

if (A[i] \* B[j] > x)

{

j--;

}

else

{

break;

}

}

ans += j;

}

return ans;

}

int main(int argc, const char \* argv[])

{

cin >> N >> K;

for (int i = 1; i <= N; i++)

{

scanf("%lld %lld", A + i, B + i);

}

sort(A + 1, A + N + 1);

sort(B + 1, B + N + 1);

ll ans = 0;

ll key = N \* N - K + 1;

ll low = A[1] \* B[1]; // 初始最小值

ll high = A[N] \* A[N]; // 初始最大值

while (high - low > 1)

{

ll mid = (low + high) >> 1;

if (check(mid) >= key)

{

ans = mid;

high = mid;

}

else

{

low = mid;

}

}

cout << ans << '\n';

return 0;

}

归并排序求逆序数

/\*

\* 也可以用树状数组做

\* a[0...n-1] cnt=0; call: MergeSort(0, n)

\*/

const int N = 1010;

int a[N];

int c[N];

int cnt = 0;

void MergeSort(int l, int r)

{

int mid, i, j, tmp;

if (r > l + 1)

{

mid = (l + r) / 2;

MergeSort(l, mid);

MergeSort(mid, r);

tmp = l;

for (i = l, j = mid; i < mid && j < r;)

{

if (a[i] > a[j])

{

c[tmp++] = a[j++];

cnt += mid - i;

}

else

{

c[tmp++] = a[i++];

}

}

if (j < r)

{

for (; j < r; ++j)

{

c[tmp++] = a[j];

}

}

else

{

for (; i < mid; ++i)

{

c[tmp++]=a[i];

}

}

for (i = l; i < r; ++i)

{

a[i] = c[i];

}

}

return ;

}

取第k个元素

/\*

\* 取第k个元素

\* k = 0 ... n - 1,平均复杂度O(n) 注意a[]中的顺序被改变

\*/

#define \_cp(a,b) ((a) < (b))

typedef int elem\_t;

elem\_t kth\_element(int n, elem\_t \*a, int k)

{ // a[0 ... n-1]

elem\_t t, key;

int l = 0, r = n - 1, i, j;

while (l < r)

{

for (key = a[((i = l - 1) + (j = r + 1)) >> 1]; i < j;)

{

for (j--; \_cp(key, a[j]); j--);

for (i++; \_cp(a[i], key); i++);

if (i < j)

{

t = a[i], a[i] = a[j], a[j] = t;

}

}

if (k > j)

{

l = j + 1;

}

else

{

r = j;

}

}

return a[k];

}

最长有序子序列

/\*

\* 递增（默认）

\* 递减

\* 非递增

\* 非递减 (1)>= && < (2)< (3)>=

\*/

const int MAXN = 1001;

int a[MAXN], f[MAXN], d[MAXN]; // d[i] 用于记录 a[0...i] 以 a[i] 结尾的最大长度

int bsearch(const int \*f, int size, const int &a)

{

int l = 0, r = size - 1;

while (l <= r)

{

int mid = (l + r) / 2;

if (a > f[mid - 1] && a <= f[mid]) // (1)

{

return mid;

}

else if (a < f[mid])

{

r = mid - 1;

}

else

{

l = mid + 1;

}

}

return -1;

}

int LIS(const int \*a, const int &n)

{

int i, j, size = 1;

f[0] = a[0];

d[0] = 1;

for (i = 1; i < n; ++i)

{

if (a[i] <= f[0]) // (2)

{

j = 0;

}

else if (a[i] > f[size - 1]) // (3)

{

j = size++;

}

else

{

j = bsearch(f, size, a[i]);

}

f[j] = a[i];

d[i] = j + 1;

}

return size;

}

int main()

{

int i, n;

while (scanf("%d", &n) != EOF)

{

for (i = 0; i < n; ++i)

{

scanf("%d", &a[i]);

}

printf("%d\n", LIS(a, n)); // 求最大递增 / 上升子序列(如果为最大非降子序列,只需把上面的注释部分给与替换)

}

return 0;

}

最长公共子序列

const int N = 1010;

int a[N][N];

int LCS(const char \*s1, const char \*s2)

{ // s1:0...m, s2:0...n

int m = (int)strlen(s1), n = (int)strlen(s2);

int i, j;

a[0][0] = 0;

for (i = 1; i <= m; ++i)

{

a[i][0] = 0;

}

for (i = 1; i <= n; ++i)

{

a[0][i] = 0;

}

for (i = 1; i <= m; ++i)

{

for (j = 1; j <= n; ++j)

{

if (s1[i - 1] == s2[j - 1])

{

a[i][j] = a[i - 1][j - 1] + 1;

}

else if (a[i - 1][j] > a[i][j - 1])

{

a[i][j]= a[i - 1][j];

}

else

{

a[i][j] = a[i][j - 1];

}

}

}

return a[m][n];

}

背包相关

const int MAXN = 101;

const int SIZE = 50001;

int dp[SIZE];

int volume[MAXN], value[MAXN], c[MAXN];

int n, v; // 总物品数，背包容量

// 01背包

void ZeroOnepark(int val, int vol)

{

for (int j = v ; j >= vol; j--)

{

dp[j] = max(dp[j], dp[j - vol] + val);

}

}

// 完全背包

void Completepark(int val, int vol)

{

for (int j = vol; j <= v; j++)

{

dp[j] = max(dp[j], dp[j - vol] + val);

}

}

// 多重背包

void Multiplepark(int val, int vol, int amount)

{

if (vol \* amount >= v)

{

Completepark(val, vol);

}

else

{

int k = 1;

while (k < amount)

{

ZeroOnepark(k \* val, k \* vol);

amount -= k;

k <<= 1;

}

if (amount > 0)

{

ZeroOnepark(amount \* val, amount \* vol);

}

}

}

int main()

{

while (cin >> n >> v)

{

for (int i = 1 ; i <= n ; i++)

{

cin >> volume[i] >> value[i] >> c[i]; // 费用，价值，数量

}

memset(dp, 0, sizeof(dp));

for (int i = 1; i <= n; i++)

{

Multiplepark(value[i], volume[i], c[i]);

}

cout << dp[v] << endl;

}

return 0;

}

使序列有序的最少交换次数

交换相邻两数

如果只是交换相邻两数，那么最少交换次数为该序列的逆序数。

交换任意区间

/\*

\* 默认目标映射关系是 key 1 => val 1 …… key n => val n

\* 如果序列不是 1~n 可以通过 map 建立新的目标映射关系

\* 交换任意区间的本质是改变了元素的后继，故建立元素与其初始状态后继的映射关系

\*/

const int MAXN = 30;

int n;

int vis[MAXN];

int A[MAXN], B[MAXN];

int getMinSwaps()

{

memset(vis, 0, sizeof(vis));

for (int i = 1; i <= n; i++)

{

B[A[i]] = A[i % n + 1];

}

for (int i = 1; i <= n; i++)

{

B[i] = (B[i] - 2 + n) % n + 1;

}

int cnt = n;

for (int i = 1; i <= n; i++)

{

if (vis[i])

{

continue;

}

vis[i] = 1;

cnt--;

for (int j = B[i]; j != i; j = B[j])

{

vis[j] = 1;

}

}

return cnt;

}

int main()

{

cin >> n;

for (int i = 1; i <= n; i++)

{

cin >> A[i];

}

int res = getMinSwaps();

cout << res << '\n';

return 0;

}

Geometry 计算几何

//平面几何模板整理

//在原来的内容上进行了一些修改

#include <cstdio>

#include <algorithm>

#include <queue>

#include <cstring>

#include <string>

#include <cmath>

#include <iostream>

using namespace std;

const int maxn=10050;

const double eps=1e-8;//精度范围

//\*\*\*\*\*\*\*\*\*平面点，线，简单图形的表示\*\*\*\*\*\*\*\*\*

struct point{

//平面点的表示，坐标值为实数，适用于对时间要求不太高的情况

//优点是可以进行精确计算，在大部分情况下都是使用这种表示法表示平面上的点

//缺点是运算起来可能会比较慢

double x,y;

point(double x0=0,double y0=0){

x=x0;y=y0;

}

};

struct point2{

int x,y;

//如果仅仅是进行定性判断或者是不涉及实数计算的算法（如求凸包），可以使用以下平面点的表示法

//注意如果涉及到计算的话，最好还是使用坐标值为实数的点表示法

point2(int x0=0,int y0=0){

x=x0;y=y0;

}

};

struct line{

//待定系数法表示直线，形如ax+by+c=0可以表示平面上的一条直线

//该表示法可以方便利用解析几何的知识解决问题

//用这种表示法需要记住一些公式，例如两直线交点坐标公式，下面有写

double a,b,c;

line(double a0=0,double b0=0,double c0=0){

a=a0;b=b0;c=c0;

}

};

struct line2{

//两点式表示一条直线或线段

//实际上很多时候不用刻意使用结构体，使用两个point数组亦可表示

point p1,p2;

line2(point \_p1,point \_p2){

p1=\_p1;p2=\_p2;

}

};

struct circle{

//点径式表示一个圆

//其实有时也不用刻意使用结构体表示圆

point o;

double r;

};

typedef point vctor;//将point另命名为向量，这样一个点坐标就可表示为一条由坐标原点指向该点坐标的向量

//任意多边形可以使用一组按照顺时针或逆时针给出的点序列表示

//\*\*\*\*\*\*\*\*\*点，向量的运算符的重载\*\*\*\*\*\*\*\*\*

vctor operator + (vctor a,vctor b){

//平面两向量相加

return vctor(a.x+b.x,a.y+b.y);

}

vctor operator - (vctor a,vctor b){

//平面两向量相减

return vctor(a.x-b.x,a.y-b.y);

}

vctor operator \* (vctor a,double k){

//向量的数乘，乘以一个实数

return vctor(a.x\*k,a.y\*k);

}

vctor operator / (vctor a,double k){

//向量的数乘，除以一个实数

return vctor(a.x/k,a.y/k);

}

bool operator == (point a,point b){

//判断两点是否为同一点

return a.x==b.x&&a.y==b.y;

}

point p[maxn],res[maxn],d[maxn],tmp[maxn],q;

int cnt,n,m,r;

//\*\*\*\*\*\*\*\*\*点，线的基本操作\*\*\*\*\*\*\*\*\*

int sig(double x){

//返回实数x的符号，如果x非常接近精确度的范围，那么x=0

//否则x>0即为正数，返回1，x<0返回-1

return fabs(x)<eps?0:(x>0)?1:-1;

}

double dot(point a,point b,point c){

//求向量ab与向量ac的点积

//就是高中数学的点积，所有高中学过的点积知识均适用

//以下情况可能会用到点积的运算

//1.判断夹角是锐角，直角还是钝角，判垂直

//2.求两向量的夹角

//3.求点到直线的最短距离

//具体应用下面会写到

return (b.x-a.x)\*(c.x-a.x)+(b.y-a.y)\*(c.y-a.y);

}

double cross(point a,point b,point c){

//求向量ab与向量ac的叉积

//平面向量ab与向量ac的叉积定义为(sig)\*|ab|\*|ac|\*sin(theta)

//sig表示符号，由ab与ac的位置关系决定

//如果ab在ac的顺时针方向，那么sig=1，ab在ac逆时针方向时sig=-1，ab与ac共线时sig=0，整个叉积为0

//sin(theta)表示两向量夹角的正弦

//叉积的几何意义为以两向量为邻边组成的平行四边形的有向面积

//注意是有向面积，意味着可能会出现值为负的面积，除了定性判断或者计算多边形的面积，一般加上绝对值

//以下情况可能会用到叉积的运算

//1.判断一个点在一个向量的顺时针方向还是逆时针方向

//2.判断折线的折向

//3.求三角形面积，求多边形面积，求点到直线的最短距离

//4.几乎所有几何算法，如极角排序，凸包，半平面交，旋转卡壳

//具体应用下面会写到

return (b.x-a.x)\*(c.y-a.y)-(b.y-a.y)\*(c.x-a.x);

}

point rotate(point p,double theta){

//将一个点向逆时针旋转theta弧度，返回这个点

double x=p.x\*cos(theta)-p.y\*sin(theta);

double y=p.x\*sin(theta)+p.y\*cos(theta);

return point(x,y);

}

point midpoint(point a,point b){

//中点坐标公式，其实不需要写函数，因为已经重载了运算符

return (a+b)/2;

}

double dist(point a,point b){

//求两点距离

return sqrt((a.x-b.x)\*(a.x-b.x)+(a.y-b.y)\*(a.y-b.y));

}

double dist2(point a,point b){

//返回两点距离的平方，有时可以方便计算用

return (a.x-b.x)\*(a.x-b.x)+(a.y-b.y)\*(a.y-b.y);

}

double dist\_point\_to\_line(point p1,point p2,point p3){

//求点p3到线段p1p2的最短距离，解释如下

if(dot(p1,p2,p3)<0) return dist(p1,p3);

//如果向量p1p3和向量p1p2形成钝角，那么最短距离当然是向量p1p3的模

if(dot(p2,p1,p3)<0) return dist(p2,p3);

//如果向量p2p3和向量p2p1形成钝角，那么最短距离当然是向量p2p3的模

return fabs(cross(p1,p2,p3))/dist(p1,p2);

//其余情况，我们让p1,p2,p3组成一个三角形

//那么三角形面积除以线段p1p2的长就是三角形的高，也就是p3到线段p1p2的（最短）距离了

}

double area(point p[],int n){

//求多边形的面积，转化为求三角形的面积求解

//不断以原点(0,0)，p[i]和p[i+1](i=1,2,...,n)作三角形，求出这些三角形的（有向）面积加起来

//这样求出来的一系列三角形（有向）面积有正有负，但是没关系

//把所有正的面积减去负的面积，剩下的就是多边形的面积了

//多边形的顶点必须以顺时针或逆时针给出

p[n+1]=p[1];

double s=0;

for(int i=1;i<=n;i++) s+=cross(point(0,0),p[i],p[i+1]);//s为面积

return fabs(s)/2.0;

}

double angle(point a,point b,point c){

//求向量ab和向量ac的夹角

//定性判断只需判断点积符号即可

return acos(dot(a,b,c)/(dist(a,b)\*(dist(a,c))));

}

bool iscross(point p1,point p2,point p3,point p4){

//判断两线段ab与cd是否相交，两个实验：快排实验和跨立实验

if(min(p1.x,p2.x)>max(p3.x,p4.x)||min(p3.x,p4.x)>max(p1.x,p2.x)

||min(p1.y,p2.y)>max(p3.y,p4.y)||min(p3.y,p4.y)>max(p1.y,p2.y)) return 0;

//快排实验：如果一条线段两端点的横（纵）坐标均大于另一条线段两端点的横（纵）坐标

//那么这两条线段不可能相交，返回false

double r1=cross(p1,p2,p3);

double r2=cross(p1,p2,p4);

double r3=cross(p3,p4,p1);

double r4=cross(p3,p4,p2);

if(sig(r1\*r2)<=0&&sig(r3\*r4)<=0) return 1;

return 0;

//跨立实验：如果两条线段相交，那么他们肯定互相跨越彼此

//我们先判断一条线段的两个端点是否在另一条线段的两侧

//然后再对另一条线段这么做，如果至少有一条线段不能跨越另一条线段，返回false

//否则返回true

}

point cross\_point(line l1,line l2){

//求两直线l1,l2交点坐标，前提条件为两直线相交，使用解方程法

//x=(b1\*c2-b2\*c1)/(a1\*b2-a2\*b1)

//y=(c1\*a2-c2\*a1)/(a1\*b2-a2\*b1)

double a1=l1.a,b1=l1.b,c1=l1.c;

double a2=l2.a,b2=l2.b,c2=l2.c;

double x=(b1\*c2-b2\*c1)/(a1\*b2-a2\*b1);

double y=(c1\*a2-c2\*a1)/(a1\*b2-a2\*b1);

return point(x,y);

}

point cross\_point2(point p1,point p2,point p3,point p4){

//求线段p1p2和p3p4的交点坐标，前提条件为两线段相交

//方法同求直线交点坐标，只是直线方程的系数需要自己去求

double a1=p1.y-p2.y;

double b1=p2.x-p1.x;

double c1=p1.x\*p2.y-p2.x\*p1.y;

double a2=p3.y-p4.y;

double b2=p4.x-p3.x;

double c2=p3.x\*p4.y-p4.x\*p3.y;

double x=(b1\*c2-b2\*c1)/(a1\*b2-a2\*b1);

double y=(c1\*a2-c2\*a1)/(a1\*b2-a2\*b1);

return point(x,y);

}

//\*\*\*\*\*\*\*\*\*极角排序\*\*\*\*\*\*\*\*\*

//对于平面上的所有点，我们取一个点作为原点，以平行于x轴坐标轴且向右的射线为极轴

//在这个极坐标平面上，任一点的角度定义为极轴向逆时针旋转的角度

//将所有点的坐标按照角度从小到大排序，就将所有点进行了极角排序

//也可以使用atan2(y,x)函数，该函数返回一个弧度，取值范围为(-pi,pi]

//它返回的是向量(x,y)与坐标轴形成的夹角。

int cmp(point a,point b){

//极角排序

return atan2(a.y,a.x)<atan2(b.y,b.x);

}

//\*\*\*\*\*\*\*\*\*凸包算法\*\*\*\*\*\*\*\*\*

//给定平面上的一些点，要求一个多边形，使得平面上的所有点或者在这个多边形上，或者在多边形内

//简单地说就是求一个多边形把所有平面上所有点围起来

//有四种算法：卷包裹法，Greham扫描法，Jarvis步进法，完美凸包算法

//这里仅给出时间复杂度较好的Greham扫描法，算法复杂度为O(n\*logn)

int cmp2(point a,point b){

//求凸包前必须先将所有点进行坐标排序，

//排序之后，所有点按照纵坐标从下到上的顺序排序，如果纵坐标相同则按横坐标从左到右的顺序排

return a.y<b.y||a.y==b.y&&a.x<b.x;

}

int convex(int n,point p[]){

//该函数传入平面上点的个数和所有点的坐标（已排序）

//返回凸包上点的个数，同时求出的凸包按逆时针顺序存在数组res[]内

int now=1;

for(int i=1;i<=n;i++){

while(now>2&&sig(cross(res[now-2],p[i],res[now-1]))>0) now--;

//res[]数组相当于一个栈

//首先凸包是一个多边形，至少要有三个点，故now>2

//当res[now-2],p[i],res[now-1]三点按顺序组成的折线的折向向右时，说明res[now-1]不满足条件

//因为凸包的点要按逆时针给出，而此时三点是顺时针顺序

//故将res[now-1]弹出，继续循环直到res[now-2],p[i],res[now-1]组成的折线折向向左

//此时三点是逆时针顺序，故res[]中压入p[i]

res[now++]=p[i];

//循环到满足条件的点时，压入栈中

}

//此时已求出半个凸包（从最左下方的点到最右上方的点的一个逆时针凸包）

//但凸包应该是一个闭合多边形，故还要从右上方到左下方来一遍

int k=now;

for(int i=n-1;i>=1;i--){

while(now>k&&sig(cross(res[now-2],p[i],res[now-1]))>0) now--;

res[now++]=p[i];

}

//此时求出的凸包会多出一个左下角的点，删掉这个点

if(n>1) now--;

return now-1;

}

//\*\*\*\*\*\*\*\*\*半平面交\*\*\*\*\*\*\*\*\*

//如果我们规定直线的一侧（例如右侧）为有效区域（半平面）

//那么平面上的多条直线组合起来就可以划出一块区域

//这块区域称为平面交，求平面交的算法叫做半平面交算法

//直白的说，半平面交就是将一块区域不断地用一条直线进行切割然后扔掉不要的部分

//当用直线不断切割平面时，被切割的区域可能会不断的产生新的点，也可能会有原来在平面交上的点被删掉

//因此这个算法就是对一个点序列进行不断的删除和添加操作

//需要用两个数组d[]和tmp[]，先将d[]求平面交的结果放到tmp[]里，再将tmp[]复制给d[]

//由于篇幅所限，可能有些难理解，如果不好理解就不必深入，直接CV代码也可以

void init(int n){

//初始化平面，向d[]中放入按顺时针给出的点序列

//点序列可以是给定的多边形，如该函数就传入了多边形的顶点数

//也可以是无限大平面（此时d[]中放入的是四个无穷大的点(inf,inf),(inf,-inf),(-inf,-inf),(-inf,inf)）

for(int i=1;i<=n;i++)

d[i]=p[i];

d[n+1]=d[1];d[0]=d[n];

//由于切割时可能涉及到当前点的前一个点或后一个点的操作

//故在d[0]和d[n+1]补充上d[1]的前一个点和d[n]的后一个点

m=n;//m为全局变量，表示当前平面交区域的顶点数

}

void cut(point a,point b){

//用向量ab切割平面以得到一个新的半平面交，规定有效区域为ab右边

//由于这种规定，最后得到的平面交总是顺时针的顶点序列

//m为平面交的顶点的数量

//tmp[]为临时数组，存放平面交的，最后再复制给d[]

int cnt=1;//cnt为新的半平面交的顶点个数，最后再复制给m

for(int i=1;i<=m;i++){

//判断所有点是否在新的半平面交内，然后对这些点进行一些操作

double r=cross(a,b,d[i]);

if(sig(r)<=0) tmp[cnt++]=d[i];

//如果r<=0说明该点在平面交内，直接放进tmp[]即可

else{

double r1=cross(a,b,d[i-1]);

if(sig(r1)<0) tmp[cnt++]=cross\_point2(a,b,d[i],d[i-1]);

//由于r>=0，故d[i]在新的平面交外，而这个点的前一个点有可能在新的平面交内

//这里我们需要判断一下这个点的前一个点是否在平面交内

//如果是，把这个点和这个点的前一个点作为一条线段

//然后求这条线段和ab的交点，这个交点一定在新的半平面交上，放入tmp[]

double r2=cross(a,b,d[i+1]);

if(sig(r2)<0) tmp[cnt++]=cross\_point2(a,b,d[i],d[i+1]);

//同样d[i]的后一个点有可能在新的半平面交内，像上述一样操作

}

}

//接下来就是对cnt和tmp[]的复制操作

m=cnt-1;

for(int i=1;i<=m;i++)

d[i]=tmp[i];

d[m+1]=d[1];d[0]=d[m];

}

int getcut(int n){

//p[]中存入了用于求交的直线，用相邻的两个点坐标表示一条直线

init(n);//先初始化，将所有点放入d[]中

for(int i=1;i<=n;i+=2){

cut(p[i],p[i+1]);//用给出的直线不断的切割

}

return m;//返回半平面交的顶点数

}

//\*\*\*\*\*\*\*\*\*旋转卡壳\*\*\*\*\*\*\*\*\*

//求凸多边形上距离最远的一对顶点，复杂度为O(n)，一般结合凸包算法计算

//首先选取凸多边形的一条边，一般是p1p2

//然后从p3开始，查找p3,p4,...,pn

//以p1p2为底的三角形p1p2pi(i=3,4,...,n)的面积是一个单峰函数，也就是说一定有一个最大值

//假设查找到pk时三角形面积最大

//那么三角形p1p2pk的两条边p1pk和p2pk中，(p1,pk)和(p2,pk)中就可能有一对点是最远的一对顶点

//用一个ans记录一下两对点距离较大的那一对点的距离

//然后再以p2p3为底，从原来的pk接着往下找

//由于面积是单峰函数，故一定又可以找到可能为最远顶点对的两对点，用这两对点中距离较大的维护ans

//然后再以p3p4为底……一直到以pnp1为底时结束，查找了一圈

//得到的ans就是凸多边形距离最远的一对顶点的距离

//由于篇幅所限，不好理解的话，直接照着CV

//如果多边形顶点个数比较少，暴力O(n^2)也可以

double caliper(int n){

//p[]中逆时针给出了凸多边形的顶点，n为凸多边形的顶点数

//返回距离最远的一对顶点的距离

int now=2;

//now表示当前查找的点的下标

double ans=0;

for(int i=1;i<=n;i++){

while(sig(cross(p[i],p[i+1],p[now+1])-cross(p[i],p[i+1],p[now]))>0) now=now%n+1;

//如果当前面积大于上一个面积，说明单峰函数还没达到极值，我们让now=now+1

//直到当前面积小于等于上一个面积为止，说明三角形的面积已达最大

//注意取余，查找到最后一个点的下一个时，即回到第一个点

double d1=dist(p[i],p[now]);

double d2=dist(p[i+1],p[now]);

ans=max(ans,max(d1,d2));//ans维护最大距离

}

return ans;

}

//\*\*\*\*\*\*\*\*\*pick定理\*\*\*\*\*\*\*\*\*

//皮克定理是一个计算点阵中顶点在格点上的多边形面积公式，该公式可以表示为2S=2a+b-2

//a表示多边形内部的点数，b表示多边形边上的点数，S表示多边形的面积

//计算多边形内的点的个数，只需将公式变形为a=(2S-b+2)/2

//设两点p1(x1,y1),p2(x2,y2)

//则线段p1p2的点数=gcd(abs(x1-x2),abs(y1-y2))（注意只计算了线段两个端点中的一个）

//以下是一个计算格点三角形内部点的算法

int gcd(int x,int y){

//求最大公约数

return y==0?x:gcd(y,x%y);

}

//这里需要使用坐标为整数的向量的叉积

int cross(point2 a,point2 b,point2 c){

return (b.x-a.x)\*(c.y-a.y)-(c.x-a.x)\*(b.y-a.y);

}

int pick(point2 p1,point2 p2,point2 p3){

//返回格点三角形内部的点数

//由于是格点三角形，故坐标均为整数

int s=abs(cross(p1,p2,p3));//求出三角形面积的2倍

int b1=gcd(abs(p1.x-p2.x),abs(p1.y-p2.y));

int b2=gcd(abs(p2.x-p3.x),abs(p2.y-p3.y));

int b3=gcd(abs(p3.x-p1.x),abs(p3.y-p1.y));

//求出三边上的格点数

int b=b1+b2+b3;

return (s-b+2)/2;

}

//\*\*\*\*\*\*\*\*\*最小圆覆盖\*\*\*\*\*\*\*\*\*

//最小圆覆盖指给定平面上的一些点，求作一个圆使得平面上的所有点均在这个圆内，并且圆的面积最小

//可以发现，这个最小圆至少应该过平面上的两个点

//我我们先用p[1]和p[i]做圆，再从2到i-1判断是否有点不在这个圆上

//如果都在，则说明已经找到覆盖1到i-1的圆

//如果没有都在，假设我们找到第一个不在这个圆上的点为p[j]

//于是我们用两个已知点p[j]与p[i]去找覆盖1到j-1的最小覆盖圆

//而对于两个已知点p[j]与p[i]求最小覆盖圆

//只要从1到j-1中，第k个点求过p[k],p[j],p[i]三个点的圆

//再判断k+1到j-1是否都在圆上，若都在，说明找到圆

//若有不在的，则再用新的点p[k]更新圆即可

//由于篇幅所限，不必深入，直接CV即可

point func(point p1,point p2,point p3){

//这个函数返回的是三角形的外心，也就是对三角形其中两条边作中垂线得到的交点

line l1=(p1.x-p2.x,p1.y-p2.y,(dist2(point(0,0),p1)-dist2(point(0,0),p2))/2);

line l2=(p2.x-p3.x,p2.y-p3.y,(dist2(point(0,0),p2)-dist2(point(0,0),p3))/2);

return cross\_point(l1,l2);

}

void circlecover(){

q=p[1];r=0;

//初始化圆心为p[1]

for(int i=2;i<=n;i++){

//先确定过p[i]的圆

if(dist(q,p[i])-r>eps){

q=p[i];r=0;

for(int j=1;j<=i-1;j++){

//寻找不在当前圆覆盖上的点p[j]，确定过两点的圆

if(dist(q,p[j])-r>eps){

q=(p[i]+p[j])/2;r=dist(q,p[j]);

for(int k=1;k<=j-1;k++){

//寻找不在当前圆覆盖上的点p[k]，确定过两三点的圆

if(dist(q,p[k])-r>eps){

q=func(p[i],p[j],p[k]);//求外心，外心即为三角形外接圆的圆心

r=dist(q,p[k]);

}

}

}

}

}

}

}

int main(){

}

Other 其他

数据类型的取值范围

数据类型 取值范围

char -128 ~ 127 (1 Byte，大约3位)

short -32768 ~ 32767 (2 Bytes，大约五位)

unsigned short 0 ~ 65536 (2 Bytes，大约五位)

int -2147483648 ~ 2147483647 (4 Bytes，大约十位)

unsigned int 0 ~ 4294967295 (4 Bytes，大约十位)

long == int

long long -9223372036854775808 ~ 9223372036854775807 (8 Bytes，大约十九位)

unsigned long long 0 ~ 18446744073709551615（大约二十位）

\_\_int64 == long long

unsigned \_\_int64 == unsigned long long

double 1.7 \* 10^308 (8 Bytes)

适用于正整数

template <class T>

inline void scan\_d(T &ret)

{

char c;

ret = 0;

while ((c = getchar()) < '0' || c > '9');

while (c >= '0' && c <= '9')

{

ret = ret \* 10 + (c - '0'), c = getchar();

}

}

适用于正负整数

template <class T>

inline bool scan\_d(T &ret)

{

char c;

int sgn;

if (c = getchar(), c == EOF)

{

return 0; //EOF

}

while (c != '-' && (c < '0' || c > '9'))

{

c = getchar();

}

sgn = (c == '-') ? -1 : 1;

ret = (c == '-') ? 0 : (c - '0');

while (c = getchar(), c >= '0' && c <= '9')

{

ret = ret \* 10 + (c - '0');

}

ret \*= sgn;

return 1;

}

template <class T>

inline void print\_d(T x)

{

if (x > 9)

{

print\_d(x / 10);

}

putchar(x % 10 + '0');

}

仅适合纯数字输入输出

int Scan()

{ // 输入外挂

int res = 0, flag = 0;

char ch;

if ((ch = getchar()) == '-')

{

flag = 1;

}

else if(ch >= '0' && ch <= '9')

{

res = ch - '0';

}

while ((ch = getchar()) >= '0' && ch <= '9')

{

res = res \* 10 + (ch - '0');

}

return flag ? -res : res;

}

void Out(int a)

{ // 输出外挂

if (a < 0)

{

putchar('-');

a = -a;

}

if (a >= 10)

{

Out(a / 10);

}

putchar(a % 10 + '0');

}

int main()

{

int T, n;

scanf ("%d", &T);

while (T--)

{

n = Scan();

Out(n);

printf("\n");

}

return 0;

}

适用于正负数(int,long long,float,double)

template <class T>

bool scan\_d(T &ret)

{

char c;

int sgn;

T bit = 0.1;

if (c=getchar(), c==EOF)

{

return 0;

}

while (c! = '-' && c != '.' && (c < '0' || c > '9'))

{

c = getchar();

}

sgn = (c == '-') ? -1 : 1;

ret = (c == '-') ? 0 : (c - '0');

while (c = getchar(), c >= '0' && c <= '9')

{

ret = ret \* 10 + (c - '0');

}

if (c == ' ' || c == '\n')

{

ret \*= sgn;

return 1;

}

while (c = getchar(), c >= '0' && c <= '9')

{

ret += (c - '0') \* bit, bit /= 10;

}

ret \*= sgn;

return 1;

}

template <class T>

inline void print\_d(int x)

{

if (x > 9)

{

print\_d(x / 10);

}

putchar(x % 10 + '0');

}

strtok和sscanf结合输入

/\*

\* 空格作为分隔输入,读取一行的整数

\*/

gets(buf);

int v;

char \*p = strtok(but, " ");

while (p)

{

sscanf(p, "%d", &v);

p = strtok(NULL," ");

}

积累的题目

//网络赛遇见的树形dp

#include <bits/stdc++.h>

using namespace std;

#define maxn 100005

int dp[maxn][4];

int val[maxn];

int ans;

int fl;

struct node

{

int to,l;

node(int to,int l):to(to),l(l){}

};

vector<node> g[maxn];

int vis[maxn];

void dfs(int a)

{

vis[a]=1;

for(int i=0;i<g[a].size();i++)

{

int b = g[a][i].to;

if(vis[b])continue;

fl=b;

dp[b][0] = max(dp[b][0],val[a]-val[b]-g[a][i].l);//从a到b

dp[b][1] = max(dp[b][1],max(dp[a][0],dp[a][1]) + val[a] - val[b] - g[a][i].l);//从a之前到b

dp[b][2] = max(dp[b][2],val[b]-val[a]-g[a][i].l);//从b到a

dp[b][3] = max(dp[b][3],max(dp[a][2],dp[a][3]) + val[b] - val[a] - g[a][i].l);//从b之前到a

ans = max(ans,max(max(dp[b][0],dp[b][1]),max(dp[b][2],dp[b][3])));

dfs(b);

}

}

int main()

{

int t;cin>>t;

while(t--)

{

ans=0;

int n;

scanf("%d",&n);

for(int i=1;i<=n;i++)

{

g[i].clear();

vis[i]=0;

scanf("%d",&val[i]);

dp[i][0]=dp[i][1]=dp[i][2]=dp[i][3]=0;

}

int a,b,l;

for(int i=1;i<n;i++)

{

scanf("%d%d%d",&a,&b,&l);

g[a].push\_back(node(b,l));

g[b].push\_back(node(a,l));

}

for(int i=1;i<=n;i++)

{

if(g[i].size()==1)

{

dfs(i);

break;

}

}

cout<<ans<<endl;

}

return 0;

}

//南宁网络赛遇见的最长不下降子序列nlogn

#include <bits/stdc++.h>

using namespace std;

#define maxn 2000006

int a[maxn];

int d[maxn];

int cnt=1;

int main()

{

int b;

while(scanf("%d",&b)!=EOF)

{

if(b>=10000)

{

b-=10000;

for(int i=1;i<=5;i++)

a[cnt++]=b;

}

else if(b>=0)

{

a[cnt++]=b;

}

}

if(cnt==1) return cout<<0<<endl,0;

d[1]=a[1];

int len=1;

for(int i=2;i<cnt;i++)

{

if (a[i]>=d[len]) d[++len]=a[i];

else

{

int j=upper\_bound(d+1,d+len+1,a[i])-d;

d[j]=a[i];

}

}

printf("%d\n",len);

return 0;

}

//省赛pk遇见的 判断多边形是否相交

#include <cstdio>

#include<algorithm>

#include<iostream>

using namespace std;

typedef long long ll;

struct Point{

ll x,y;

Point(ll x=0,ll y=0):x(x),y(y) {}

void in(){

cin>>x>>y;

}

};

typedef Point Vector;

inline Point operator + (Point a, Point b) { return Point(b.x + a.x, b.y + a.y); }

inline Point operator - (Point a, Point b) { return Point(a.x - b.x, a.y - b.y); }

inline bool operator < (const Point &a, const Point &b) { return a.x == b.x ? a.y < b.y : a.x < b.x; }

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

inline ll dot(Point a, Point b) { return a.x \* b.x + a.y \* b.y; }//点积

inline ll cross(Point a, Point b) { return a.x \* b.y - a.y \* b.x; }//叉积

bool OnSegment(Point p,Point a1,Point a2){

return p==a1||p==a2||cross(a1-p,a2-p)==0&&dot(a1-p,a2-p)<0;

}

bool SegmenProper(Point a1,Point a2,Point b1,Point b2){

ll c1=cross(a2-a1,b1-a1),c2=cross(a2-a1,b2-a1);

ll c3=cross(b2-b1,a1-b1),c4=cross(b2-b1,a2-b1);

bool flag=c1\*c2<0&&c3\*c4<0;

if(OnSegment(a1,b1,b2)||OnSegment(a2,b1,b2)) flag=true;

if(OnSegment(b1,a1,a2)||OnSegment(b2,a1,a2)) flag=true;

return flag;

}

bool ispointinpolygon(Point\* p, ll n, Point c) {

bool flag = 0;

for (ll i = 0, j = n - 1; i < n; j = i++) {

if (((p[i].y > c.y) != (p[j].y > c.y)) && (1.0\*c.x < 1.0\*(p[j].x - p[i].x) \* (c.y - p[i].y) / (p[j].y - p[i].y) + 1.0\*p[i].x))

flag = !flag;

}

return flag;

}

int main() {

int ca;

scanf("%d", &ca);

while (ca--) {

Point A[5],B[5];

for(ll i=0;i<3;i++)

A[i].in();

for(ll i=0;i<3;i++)

B[i].in();

if(cross(A[1]-A[0],A[2]-A[0])<0) swap(A[1],A[2]);

if(cross(B[1]-B[0],B[2]-B[0])<0) swap(B[1],B[2]);

A[3]=A[0];B[3]=B[0];

bool flag=false;

for(ll i=0;i<3;i++){

for(ll j=0;j<3;j++){

if(SegmenProper(A[i],A[i+1],B[j],B[j+1])) flag=true;

}

}

if(flag) printf("intersect\n");

else{

for(ll i=0;i<3;i++){

if(ispointinpolygon(B,3,A[i])) flag=true;

if(ispointinpolygon(A,3,B[i])) flag=true;

}

if(flag) printf("contain\n");

else printf("disjoint\n");

}

}

return 0;

}

PICK 定理

给定顶点坐标均是整点（或正方形格点）的简单多边形，皮克定理说明了其面积 S 和内部格点数目 n、边上格点数目 s 的关系：S = n +s/2+1

//cj自己写的半平面交nlgn

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <cmath>

using namespace std;

#define N 20010

struct Point{

double x,y;

Point(){}

Point(double a,double b):x(a),y(b){}

void f(double &a,double &b)

{

a=x;b=y;

}

void input()

{

scanf("%lf%lf",&x,&y);

}

void output()

{

printf("%.1f %.1f\n",x,y);

}

}P[N];

struct Line{

Point s,t;

double angle;

Line(){}

Line(Point a,Point b):s(a),t(b){}

void get\_angle()

{

angle=atan2(t.y-s.y,t.x-s.x);

}

}L[N];

int n,m;

int que[N];

const double eps=1e-8;

int sig(double x)

{

if(x>eps) return 1;

if(x<-eps) return -1;

return 0;

}

double cross(Point p0,Point p1,Point p2)

{

return (p1.x-p0.x)\*(p2.y-p0.y)-(p2.x-p0.x)\*(p1.y-p0.y);

}

bool cmp(Line l1,Line l2)

{

int t=sig(l1.angle-l2.angle);

if(t>0) return true;

else if(t<0) return false;

else if(sig(cross(l1.s,l1.t,l2.s))>0) return true;

else return false;

}

Point intersect(Line l1,Line l2)

{

double u=cross(l2.s,l2.t,l1.s),v=cross(l2.t,l2.s,l1.t);

Point p;

p.x=(u\*l1.t.x+v\*l1.s.x)/(u+v);

p.y=(u\*l1.t.y+v\*l1.s.y)/(u+v);

return p;

}

Line Move(Line l,double r)//将直线向内推进r

{

double k,dx,dy,tx,ty;

dx=l.t.x-l.s.x;

dy=l.t.y-l.s.y;

k=r/sqrt(dx\*dx+dy\*dy);

tx=k\*dy;ty=-k\*dx;

l.s.x+=tx;l.s.y+=ty;

l.t.x+=tx;l.t.y+=ty;

return l;

}

bool judge(Line l,Line l1,Line l2)

{

Point p=intersect(l1,l2);

if(sig(cross(l.s,l.t,p))>0) return true;

else return false;

}

void solve(Line L[],int n)

{

sort(L+1,L+n+1,cmp);

int cnt=1;

for(int i=2;i<=n;++i)

if(sig(L[i].angle-L[i-1].angle)!=0)

L[++cnt]=L[i];

n=cnt;

int head=1,tail=2;

que[1]=1;que[2]=2;

for(int i=3;i<=n;++i)

{

while(head<tail&&judge(L[i],L[que[tail]],L[que[tail-1]])) --tail;

while(head<tail&&judge(L[i],L[que[head]],L[que[head+1]])) ++head;

que[++tail]=i;

}

while(head<tail&&judge(L[que[head]],L[que[tail]],L[que[tail-1]])) --tail;

while(head<tail&&judge(L[que[tail]],L[que[head]],L[que[head+1]])) ++head;

m=0;

if(head==tail) return;

que[tail+1]=que[head];

for(int i=head;i<=tail;++i)

P[++m]=intersect(L[que[i]],L[que[i+1]]);

}

double area(Point P[],int n)

{

P[n+1]=P[1];

double ans=0;

for(int i=1;i<=n;++i)

ans+=cross(P[1],P[i],P[i+1]);

return -ans/2;

}

int main()

{

while(scanf("%d",&n)!=EOF)

{

L[1]=Line(Point(0,0),Point(0,10000));L[1].get\_angle();

L[2]=Line(Point(0,10000),Point(10000,10000));L[2].get\_angle();

L[3]=Line(Point(10000,10000),Point(10000,0));L[3].get\_angle();

L[4]=Line(Point(10000,0),Point(0,0));L[4].get\_angle();

for(int i=5;i<=n+4;++i)

{

L[i].t.input();L[i].s.input();L[i].get\_angle();

}

solve(L,n+4);

printf("%.1f\n",area(P,m)+eps);

}

return 0;

}

//cj的多边形面积交与并

/\*

\* 多边形的交，多边形的边一定是要按逆时针方向给出

\* 还要判断是凸包还是凹包，调用相应的函数

\* 面积并，只要和面积减去交即可

\*/

#include <bits/stdc++.h>

using namespace std;

const int maxn = //300;

const double eps = 1e-8;

int dcmp(double x)

{

if(x > eps) return 1;

return x < -eps ? -1 : 0;

}

struct Point

{

double x, y;

};

double cross(Point a,Point b,Point c) ///叉积

{

return (a.x-c.x)\*(b.y-c.y)-(b.x-c.x)\*(a.y-c.y);

}

Point intersection(Point a,Point b,Point c,Point d)

{

Point p = a;

double t =((a.x-c.x)\*(c.y-d.y)-(a.y-c.y)\*(c.x-d.x))/((a.x-b.x)\*(c.y-d.y)-(a.y-b.y)\*(c.x-d.x));

p.x +=(b.x-a.x)\*t;

p.y +=(b.y-a.y)\*t;

return p;

}

//计算多边形面积

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)//ConvexPolygonIntersectArea

{

Point p[20], tmp[20];

int tn, sflag, eflag;

a[na] = a[0], b[nb] = b[0];

memcpy(p,b,sizeof(Point)\*(nb + 1));

for(int i = 0; i < na && nb > 2; i++)

{

sflag = dcmp(cross(a[i + 1], p[0],a[i]));

for(int j = tn = 0; j < nb; j++, sflag = eflag)

{

if(sflag>=0) tmp[tn++] = p[j];

eflag = dcmp(cross(a[i + 1], p[j + 1],a[i]));

if((sflag ^ eflag) == -2)

tmp[tn++] = intersection(a[i], a[i + 1], p[j], p[j + 1]); ///求交点

}

memcpy(p, tmp, 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)///SimplePolygonIntersectArea 调用此函数

{

int i, j;

Point t1[4], t2[4];

double res = 0, num1, num2;

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];

num1 = dcmp(cross(t1[1], t1[2],t1[0]));

if(num1 < 0) swap(t1[1], t1[2]);

for(j = 2; j < nb; j++)

{

t2[1] = b[j - 1], t2[2] = b[j];

num2 = dcmp(cross(t2[1], t2[2],t2[0]));

if(num2 < 0) swap(t2[1], t2[2]);

res += CPIA(t1, t2, 3, 3) \* num1 \* num2;

}

}

return res;

}

Point p1[maxn], p2[maxn];

int n1, n2;

int main()

{

while(cin>>n1>>n2)

{

for(int i = 0; i < n1; i++) scanf("%lf%lf", &p1[i].x, &p1[i].y);

for(int i = 0; i < n2; i++) scanf("%lf%lf", &p2[i].x, &p2[i].y);

double Area = SPIA(p1, p2, n1, n2);

}

return 0;

}

//多边形重心

#include <cstdiO>

#include <cstring>

#include <algorithm>

#include <cmath>

using namespace std;

#define N 1000010

struct Point{

double x,y;

Point(){}

Point(double \_x,double \_y):x(\_x),y(\_y){}

void input()

{

scanf("%lf%lf",&x,&y);

}

}P[N];

const double eps=1e-8;

int n;

double cross(Point p0,Point p1,Point p2)

{

return (p1.x-p0.x)\*(p2.y-p0.y)-(p2.x-p0.x)\*(p1.y-p0.y);

}

//求多边形重心的思路把多边形划分为三角形，算出三角形的重心

//三角形的质量都集中在重心，可以把三角形看成一个点

//然后就变成了求一些离散的质点的重心，套用重心公式即可

Point center(Point P[],int n)

{

P[n]=P[0];

double area,sum\_area=0,sum\_x=0,sum\_y=0;

for(int i=0;i<n;++i)

{

area=cross(P[0],P[i],P[i+1])/2;

sum\_area+=area;

sum\_x+=(P[0].x+P[i].x+P[i+1].x)/3\*area;

sum\_y+=(P[0].y+P[i].y+P[i+1].y)/3\*area;

}

return Point(sum\_x/sum\_area,sum\_y/sum\_area);

}

int main()

{

int ca;

scanf("%d",&ca);

while(ca--)

{

scanf("%d",&n);

for(int i=0;i<n;++i) P[i].input();

Point p;

p=center(P,n);

printf("%.2f %.2f\n",p.x+eps,p.y+eps);

}

return 0;

}

//矩形面积并

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <map>

using namespace std;

#define lson node<<1,s,mid

#define rson node<<1|1,mid+1,t

#define lnode node<<1

#define rnode node<<1|1

#define M(x) memset(x,0,sizeof(x));

#define N 110

struct rec{

int l,r,w;

double x1,x2,h;

rec(){}

rec(double a,double b,double c,int d)

{

x1=a;x2=b;h=c;w=d;

}

bool operator<(const rec &a)const

{

return (h<a.h)||(h==a.h)&&(w>a.w);

}

}a[N\*2];

int change[N\*8];

double seg[N\*8],b[N\*2],fx[N\*2];

int n;

map<double,int>mp;

void pushup(int node,int s,int t)

{

if(change[node])

seg[node]=fx[t+1]-fx[s];

else if(s!=t)

seg[node]=seg[lnode]+seg[rnode];

else seg[node]=0;

}

void updata(int node,int s,int t,int l,int r,int w)

{

if(l<=s&&t<=r)

{

change[node]+=w;

pushup(node,s,t);

}

else

{

int mid=s+t>>1;

if(l<=mid) updata(lson,l,r,w);

if(mid<r) updata(rson,l,r,w);

pushup(node,s,t);

}

}

int main()

{

int ca=0;

double x1,x2,y1,y2;

M(seg);M(change);

while(scanf("%d",&n),n)

{

for(int i=0;i<n;++i)

{

scanf("%lf%lf%lf%lf",&x1,&y1,&x2,&y2);

b[i<<1]=x1;b[i<<1|1]=x2;

a[i<<1]=rec(x1,x2,y1,1);

a[i<<1|1]=rec(x1,x2,y2,-1);

}

sort(b,b+2\*n);

int tot=1;

mp.clear();

mp[b[0]]=1;fx[1]=b[0];

for(int i=1;i<2\*n;++i)

if(b[i]!=b[i+1])

{

mp[b[i]]=++tot;

fx[tot]=b[i];

}

for(int i=0;i<2\*n;++i)

a[i].l=mp[a[i].x1],a[i].r=mp[a[i].x2];

sort(a,a+2\*n);

double ans=0;

for(int i=0;i<2\*n;++i)

{

updata(1,1,tot-1,a[i].l,a[i].r-1,a[i].w);

ans+=seg[1]\*(a[i+1].h-a[i].h);

}

printf("Test case #%d\nTotal explored area: %.2f\n\n",++ca,ans);

}

return 0;

}

//求矩形周长

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

#define lson node<<1,s,mid

#define rson node<<1|1,mid+1,t

#define lnode node<<1

#define rnode node<<1|1

#define M(x) memset(x,0,sizeof(x));

#define N 20010

struct rec{

int x1,x2,h,w;

rec(){}

rec(int a,int b,int c,int d)

{

x1=a;x2=b;h=c;w=d;

}

bool operator<(const rec &a)const

{

return (h<a.h)||(h==a.h)&&(w>a.w);

}

}a[N];

int seg[N\*4],change[N\*4],sum[N\*4];

bool lseg[N\*4],rseg[N\*4];

int n;

void pushup(int node,int s,int t)

{

if(change[node])

{

seg[node]=t-s+1;

sum[node]=2;

lseg[node]=rseg[node]=1;

}

else

{

if(s!=t)

{

seg[node]=seg[lnode]+seg[rnode];

lseg[node]=lseg[lnode];

rseg[node]=rseg[rnode];

sum[node]=sum[lnode]+sum[rnode];

if(rseg[lnode]&&lseg[rnode])

sum[node]-=2;

}

else seg[node]=lseg[node]=rseg[node]=sum[node]=0;

}

}

void updata(int node,int s,int t,int l,int r,int w)

{

if(l<=s&&t<=r)

{

change[node]+=w;

pushup(node,s,t);

}

else

{

int mid=s+t>>1;

if(l<=mid) updata(lson,l,r,w);

if(mid<r) updata(rson,l,r,w);

pushup(node,s,t);

}

}

int main()

{

int s,t,x1,x2,y1,y2;

M(seg);M(lseg);M(rseg);M(change);M(sum);

while(scanf("%d",&n)!=EOF)

{

s=N;t=-N;

for(int i=0;i<n;++i)

{

scanf("%d%d%d%d",&x1,&y1,&x2,&y2);

a[i<<1]=rec(x1,x2,y1,1);

a[i<<1|1]=rec(x1,x2,y2,-1);

s=min(s,x1);t=max(t,x2);

}

sort(a,a+2\*n);

int ans=0,last=0;

//printf("ok\n");

for(int i=0;i<2\*n;++i)

{

updata(1,s,t,a[i].x1,a[i].x2-1,a[i].w);

ans+=abs(seg[1]-last)+sum[1]\*(a[i+1].h-a[i].h);

last=seg[1];

}

printf("%d\n",ans);

}

}

//求两圆面积交

#include <iostream>

#include <cmath>

#include <algorithm>

#include <iomanip>

using namespace std;

typedef long double ld;

const ld pi=acosl(-1.0L);

ld area(ld x,ld y,ld r,ld xx,ld yy,ld rr)

{

if(r>rr) swap(r,rr);

ld d=sqrtl((x-xx)\*(x-xx)+(y-yy)\*(y-yy));

if(d>=r+rr) return 0;

if(d<=rr-r) return pi\*r\*r;

ld k=acosl((r\*r+d\*d-rr\*rr)/(d\*r\*2));

ld kk=acosl((rr\*rr+d\*d-r\*r)/(d\*rr\*2));

return r\*r\*k+rr\*rr\*kk-sinl(k)\*(r\*r+d\*d-rr\*rr)/(2\*d)\*r-sinl(kk)\*(rr\*rr+d\*d-r\*r)/(2\*d)\*rr;

}

int main()

{

ld x,x0,y,y0,r,r0;

cin>>x>>y>>r>>x0>>y0>>r0;

cout.precision(20);

cout<<fixed<<area(x,y,r,x0,y0,r0)<<endl;

return 0;

}

/\*

Sample test(s)

input

0 0 4

6 0 4

output

7.25298806364175601379

input

0 0 5

11 0 5

output

0.00000000000000000000

\*/

//圆的面积并

/\*

两个剪枝。

首先把所有被内含的圆删掉。

其次我们可以一段一段的求simpson值，累加。

因为中间可能会有空的部分。

\*/

#include<math.h>

#include<stdio.h>

#include<string.h>

#include<algorithm>

#define N 1100

#define pr pair<double,double>

#define Fabs(x) ((x)>0?(x):-(x))

using namespace std;

const double pi=acos(-1.0);

const double EPS=1e-13;

const double INF=1e100;

struct Point

{

int x,y;

friend double dis(Point a,Point b)

{

return sqrt((double)(a.x-b.x)\*(a.x-b.x)+(a.y-b.y)\*(a.y-b.y));

}

};

struct Circle

{

Point p;

int r;

void read(){scanf("%d%d%d",&p.x,&p.y,&r);}

friend bool operator <(Circle a,Circle b)

{

if(a.p.x-a.r<b.p.x-a.r)

return a.p.x+a.r<b.p.x+a.r;

return a.p.x-a.r<b.p.x-a.r;

}

pr f(double x)

{

if(r<=fabs(p.x-x)) return pr(0,0);

double t=r\*r-(p.x-x)\*(p.x-x);

t=sqrt(t);

return pr(p.y-t,p.y+t);

}

}O[N];

bool ban[N];

pr p[N];

int n;

double Cut(double x)

{

double ret=0,last=-INF;

int cnt=0;

for(int i=1;i<=n;i++)

{

p[++cnt]=O[i].f(x);

if(p[cnt]==pr(0,0))

cnt--;

}

sort(p+1,p+cnt+1);

for(int i=1;i<=cnt;i++)

{

if(p[i].first>last)

ret+=p[i].second-p[i].first,last=p[i].second;

else if(p[i].second>last)

ret+=p[i].second-last,last=p[i].second;

}

return ret;

}

double Simpson(double l,double r,double mid,double Cl,double Cr,double Cm)

{

double tCl=Cut((l+mid)/2),tCr=Cut((mid+r)/2);

double ans=(r-l)\*(Cl+Cr+4\*Cm)/6,lans=(mid-l)\*(Cl+Cm+4\*tCl)/6,rans=(r-mid)\*(Cr+Cm+4\*tCr)/6;

if(Fabs(lans+rans-ans)<EPS)

return ans;

else

return Simpson(l,mid,(l+mid)/2,Cl,Cm,tCl)+Simpson(mid,r,(mid+r)/2,Cm,Cr,tCr);

}

int main()

{

int i,j,k;

double l,r;

scanf("%d",&n);

l=INF,r=-INF;

for(i=1;i<=n;i++)

{

O[i].read();

l=min(l,(double)O[i].p.x-O[i].r);

r=max(r,(double)O[i].p.x+O[i].r);

}

sort(O+1,O+n+1);

for(i=1;i<=n;i++)

{

if(ban[i]) continue;

for(j=i+1;j<=n;j++)

{

if(ban[j]) continue;

if(dis(O[i].p,O[j].p)+O[j].r<=O[i].r)

ban[j]=1;

}

}

for(i=1;i<=n;i++)

{

if(ban[i])

{

swap(ban[i],ban[n]);

swap(O[i--],O[n--]);

}

}

printf("%.3lf\n",Simpson(l,r,(l+r)/2,0,0,Cut((l+r)/2)));

return 0;

}

//圆与多边形面积交

#include<algorithm>

#include<cstdio>

#include<cmath>

#include<cstring>

#include<iostream>

using namespace std;

struct Point{

double x,y;

Point(){}

Point(double x0,double y0):x(x0),y(y0){}

}p[200005];

struct Line{

Point s,e;

Line(){}

Line(Point s0,Point e0):s(s0),e(e0){}

};

int n;

double R;

const double eps=1e-8;

const double Pi=acos(-1);

double sgn(double x){

if (x>eps) return 1.0;

if (x<-eps) return -1.0;

return 0;

}

Point operator \*(Point p1,double x){

return Point(p1.x\*x,p1.y\*x);

}

Point operator /(Point p1,double x){

return Point(p1.x/x,p1.y/x);

}

double operator /(Point p1,Point p2){

return p1.x\*p2.x+p1.y\*p2.y;

}

double operator \*(Point p1,Point p2){

return p1.x\*p2.y-p1.y\*p2.x;

}

Point operator +(Point p1,Point p2){

return Point(p1.x+p2.x,p1.y+p2.y);

}

Point operator -(Point p1,Point p2){

return Point(p1.x-p2.x,p1.y-p2.y);

}

double dis(Point p1){

return sqrt(p1.x\*p1.x+p1.y\*p1.y);

}

double dis(Point p1,Point p2){

return dis(Point(p1.x-p2.x,p1.y-p2.y));

}

double sqr(double x){

return x\*x;

}

double dist\_line(Line p){

double A,B,C,dist;

A=p.s.y-p.e.y;

B=p.s.x-p.e.x;

C=p.s.x\*p.e.y-p.s.y\*p.e.x;

dist=fabs(C)/sqrt(sqr(A)+sqr(B));

return dist;

}

double get\_cos(double a,double b,double c){

return (b\*b+c\*c-a\*a)/(2\*b\*c);

}

double get\_angle(Point p1,Point p2){

if (!sgn(dis(p1))||!sgn(dis(p2))) return 0.0;

double A,B,C;

A=dis(p1);

B=dis(p2);

C=dis(p1,p2);

if (C<=eps) return 0.0;

return acos(get\_cos(C,A,B));

}

Point get\_point(Point p){

double T=sqr(p.x)+sqr(p.y);

return Point(sgn(p.x)\*sqrt(sqr(p.x)/T),sgn(p.y)\*sqrt(sqr(p.y)/T));

}

double S(Point p1,Point p2,Point p3){

return fabs((p2-p1)\*(p3-p1))/2;

}

double work(Point p1,Point p2,Point O){

p1=p1-O;p2=p2-O;

O=Point(0,0);

double f=sgn(p1\*p2),res=0;

if (!sgn(f)||!sgn(dis(p1))||!sgn(dis(p2))) return 0.0;

double l=dist\_line(Line(p1,p2));

double a=dis(p1);

double b=dis(p2);

double c=dis(p1,p2);

if (a<=R&&b<=R){

return fabs(p1\*p2)/2.0\*f;

}

if (a>=R&&b>=R&&l>=R){

double ang=get\_angle(p1,p2);

return fabs((ang/(2.0))\*(R\*R))\*f;

}

if (a>=R&&b>=R&&l<=R&&(get\_cos(a,b,c)<=0||get\_cos(b,a,c)<=0)){

double ang=get\_angle(p1,p2);

return fabs((ang/(2.0))\*(R\*R))\*f;

}

if (a>=R&&b>=R&&l<=R&&(get\_cos(a,b,c)>0&&get\_cos(b,a,c)>0)){

double dist=dist\_line(Line(p1,p2));

double len=sqrt(sqr(R)-sqr(dist))\*2.0;

double ang1=get\_angle(p1,p2);

double cos2=get\_cos(len,R,R);

res+=fabs(len\*dist/2.0);

double ang2=ang1-acos(cos2);

res+=fabs((ang2/(2))\*(R\*R));

return res\*f;

}

if ((a>=R&&b<R)||(a<R&&b>=R)){

if (b>a) swap(a,b),swap(p1,p2);

double T=sqr(p1.x-p2.x)+sqr(p1.y-p2.y);

Point u=Point(sgn(p1.x-p2.x)\*sqrt(sqr(p1.x-p2.x)/T),sgn(p1.y-p2.y)\*sqrt(sqr(p1.y-p2.y)/T));

double dist=dist\_line(Line(p1,p2));

double len=sqrt(R\*R-dist\*dist);

double len2=sqrt(sqr(dis(p2))-sqr(dist));

if (fabs(dis(p2+u\*len2)-dist)<=eps) len+=len2;

else len-=len2;

Point p=p2+u\*len;

res+=S(O,p2,p);

double ang=get\_angle(p1,p);

res+=fabs((ang/2.0)\*R\*R);

return res\*f;

}

return 0;

}

int main(){

Point O=Point(0,0);

while (scanf("%lf",&R)!=EOF){

scanf("%d",&n);

for (int i=1;i<=n;i++)//顺时针逆时针均可！！！

scanf("%lf%lf",&p[i].x,&p[i].y);

p[n+1]=p[1];

double ans=0;

for (int i=1;i<=n;i++)

ans+=work(p[i],p[i+1],O);

ans=fabs(ans);

printf("%.2f\n",ans);

}

}

//最近点对

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <cmath>

using namespace std;

#define N 100010

const double inf=1e20;

struct Point{

double x,y;

}a[N];

int tmp[N];

int n;

bool cmpxy(Point &p1,Point &p2)

{

return (p1.x<p2.x)||(p1.x==p2.x&&p1.y<p2.y);

}

bool cmpy(int i,int j)

{

return a[i].y<a[j].y;

}

double dis(Point &p1,Point &p2)

{

return sqrt((p1.x-p2.x)\*(p1.x-p2.x)+(p1.y-p2.y)\*(p1.y-p2.y));

}

double closest\_pair(int l,int r)

{

double d=inf;

if(l==r) return d;

if(l+1==r) return dis(a[l],a[r]);

int mid=l+r>>1;

double d1=closest\_pair(l,mid);

double d2=closest\_pair(mid+1,r);

d=min(d1,d2);

int k=0;

for(int i=l;i<=r;++i)

if(fabs(a[i].x-a[mid].x)<=d)

tmp[++k]=i;

sort(tmp+1,tmp+k+1,cmpy);

for(int i=1;i<k;++i)

for(int j=i+1;j<=k&&a[tmp[j]].y-a[tmp[i]].y<d;++j)

d=min(d,dis(a[tmp[i]],a[tmp[j]]));

return d;

}

int main()

{

while(scanf("%d",&n),n)

{

for(int i=1;i<=n;++i)

scanf("%lf%lf",&a[i].x,&a[i].y);

sort(a+1,a+n+1,cmpxy);

double ans=closest\_pair(1,n);

if(ans<=10000) printf("%.4f\n",ans);

else printf("INFINITY\n");

}

return 0;

}

//最小圆覆盖

#include <cstdio>

#include <cmath>

#include <cstring>

#include <algorithm>

using namespace std;

#define N 510

const double eps=1e-8;

struct Point{

double x,y;

}a[N];

int n;

double dis(Point p1,Point p2)

{

return sqrt((p1.x-p2.x)\*(p1.x-p2.x)+(p1.y-p2.y)\*(p1.y-p2.y));

}

Point circumcenter(Point p1,Point p2,Point p3)//求三角形外心

{

Point p;

double a1=p2.x-p1.x,b1=p2.y-p1.y,c1=(a1\*a1+b1\*b1)/2;

double a2=p3.x-p1.x,b2=p3.y-p1.y,c2=(a2\*a2+b2\*b2)/2;

double d=a1\*b2-a2\*b1;

p.x=p1.x+(c1\*b2-c2\*b1)/d;

p.y=p1.y+(a1\*c2-a2\*c1)/d;

return p;

}

void min\_cover\_circle(Point p[],int n,Point &c,double &r)

{

random\_shuffle(a+1,a+n+1);//随机化序列

c=a[1];r=0;

for(int i=2;i<=n;++i)

if(dis(p[i],c)>r+eps)//第一个点

{

c=p[i];r=0;

for(int j=1;j<i;++j)

if(dis(p[j],c)>r+eps)//第二个点

{

c.x=(p[i].x+p[j].x)/2;

c.y=(p[i].y+p[j].y)/2;

r=dis(p[i],c);

for(int k=1;k<j;++k)

if(dis(p[k],c)>r+eps)//第三个点

{

c=circumcenter(p[i],p[j],p[k]);

r=dis(p[i],c);

}

}

}

}

int main()

{

Point c;

double r;

while(scanf("%d",&n),n)

{

for(int i=1;i<=n;++i)

scanf("%lf%lf",&a[i].x,&a[i].y);

min\_cover\_circle(a,n,c,r);

printf("%.2f %.2f %.2f\n",c.x,c.y,r);

}

}

//求两凸包最近点对距离

/\*

---------------

点集编号从0开始

---------------

\*/

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <cmath>

using namespace std;

#define N 10010

struct Point{

double x,y;

Point(){}

Point(double \_x,double \_y):x(\_x),y(\_y){}

void input()

{

scanf("%lf%lf",&x,&y);

}

}P[N],Q[N],b[N];

int n,m;

const double eps=1e-8;

const double inf=1e20;

double cross(Point p0,Point p1,Point p2)

{

return (p1.x-p0.x)\*(p2.y-p0.y)-(p2.x-p0.x)\*(p1.y-p0.y);

}

double dot(Point p0,Point p1,Point p2)

{

return (p1.x-p0.x)\*(p2.x-p0.x)+(p1.y-p0.y)\*(p2.y-p0.y);

}

double area(Point a[],int n)

{

double ans=0;

a[n]=a[0];

for(int i=0;i<n;++i)

ans+=cross(a[0],a[i],a[i+1]);

return ans/2.0;

}

void Reverse(Point a[],int n)

{

for(int i=0;i<n;++i) b[i]=a[i];

for(int i=0;i<n;++i) a[i]=b[n-i-1];

}

double dis(Point p1,Point p2)

{

return sqrt((p1.x-p2.x)\*(p1.x-p2.x)+(p1.y-p2.y)\*(p1.y-p2.y));

}

//计算p3到线段p1,p2的最短距离

double get\_dis(Point p1,Point p2,Point p3)

{

if(dis(p1,p2)<eps) return dis(p1,p3);

if(dot(p1,p2,p3)<-eps) return dis(p1,p3);

if(dot(p2,p1,p3)<-eps) return dis(p2,p3);

return fabs(cross(p1,p2,p3))/dis(p1,p2);

}

//计算线段p1,p2到线段p3,p4的最短距离

double get\_dis(Point p1,Point p2,Point p3,Point p4)

{

return min(get\_dis(p1,p2,p3),get\_dis(p1,p2,p4));

}

//旋转卡壳

double solve(Point P[],Point Q[],int n,int m)

{

int yP=0,yQ=0;

//找出P中y最小的点和Q中y最大的点

for(int i=0;i<n;++i)

if(P[i].y<P[yP].y) yP=i;

for(int i=0;i<m;++i)

if(Q[i].y>Q[yQ].y) yQ=i;

P[n]=P[0];Q[m]=Q[0];

double tmp,ans=inf;

for(int i=0;i<n;++i)

{

while(tmp=cross(P[yP+1],Q[yQ+1],P[yP])-cross(P[yP+1],Q[yQ],P[yP])>eps)

yQ=(yQ+1)%m;

if(tmp<-eps) ans=min(ans,get\_dis(P[yP],P[yP+1],Q[yQ]));

else ans=min(ans,get\_dis(P[yP],P[yP+1],Q[yQ],Q[yQ+1]));

yP=(yP+1)%n;

}

return ans;

}

int main()

{

while(scanf("%d%d",&n,&m),n)

{

for(int i=0;i<n;++i) P[i].input();

for(int i=0;i<m;++i) Q[i].input();

//将P和Q按顺时针排列

if(area(P,n)<-eps) Reverse(P,n);

if(area(Q,m)<-eps) Reverse(Q,m);

//求出分别绕两个多边形旋转的最小值

printf("%.5f\n",min(solve(P,Q,n,m),solve(Q,P,m,n)));

}

return 0;

}

//求凸包的直径

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <cmath>

using namespace std;

#define N 50010

struct Point{

double x,y;

Point(){}

Point(double \_x,double \_y):x(\_x),y(\_y){}

bool operator<(const Point &a)const

{

return x<a.x||x==a.x&&y<a.y;

}

void input()

{

scanf("%lf%lf",&x,&y);

}

}P[N],sta[N];

int n;

const double eps=1e-8;

double cross(Point p0,Point p1,Point p2)

{

return (p1.x-p0.x)\*(p2.y-p0.y)-(p2.x-p0.x)\*(p1.y-p0.y);

}

double area(Point P[],int n)

{

double ans=0;

P[n]=P[0];

for(int i=0;i<n;++i)

ans+=cross(P[0],P[i],P[i+1]);

return -ans/2;

}

void Reverse(Point P[],int n)

{

for(int i=0;i<(n+1)/2;++i)

swap(P[i],P[n-i-1]);

}

double dis2(Point p1,Point p2)

{

return (p1.x-p2.x)\*(p1.x-p2.x)+(p1.y-p2.y)\*(p1.y-p2.y);

}

double solve(Point P[],int n)

{

int i1=0,i2=0;

P[n]=P[0];

for(int i=1;i<n;++i)

{

if(P[i].y>P[i1].y) i1=i;

if(P[i].y<P[i2].y) i2=i;

}

double ans=0,tmp;

for(int i=0;i<n;++i)

{

while(tmp=cross(P[i1],P[i2],P[i1+1])-cross(P[i1],P[i2+1],P[i1+1])<-eps)

i2=(i2+1)%n;

ans=max(ans,max(dis2(P[i1],P[i2]),dis2(P[i1+1],P[i2+1])));

i1=(i1+1)%n;

}

return ans;

}

void Graham()

{

sort(P,P+n);

int top=1;

sta[0]=P[0];sta[1]=P[1];

for(int i=2;i<n;++i)

{

while(top&&cross(sta[top-1],sta[top],P[i])>-eps) --top;

sta[++top]=P[i];

}

int k=top;

for(int i=n-2;i>=0;--i)

{

while(top>k&&cross(sta[top-1],sta[top],P[i])>-eps) --top;

sta[++top]=P[i];

}

if(top) top--;

for(int i=0;i<=top;++i) P[i]=sta[i];

n=top+1;

}

int main()

{

while(scanf("%d",&n)!=EOF)

{

for(int i=0;i<n;++i) P[i].input();

if(area(P,n)<0) Reverse(P,n);

Graham();

printf("%.0f\n",solve(P,n));

}

return 0;

}

//求凸包上最大三角形面积

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <cmath>

#include <set>

#include <utility>

using namespace std;

#define N 50010

struct Point{

double x,y;

Point(){}

Point(double \_x,double \_y):x(\_x),y(\_y){}

bool operator<(const Point &a)const

{

return x<a.x||x==a.x&&y<a.y;

}

void input()

{

scanf("%lf%lf",&x,&y);

}

}P[N],sta[N];

int n;

bool f[N];

const double eps=1e-8;

double cross(Point p0,Point p1,Point p2)

{

return (p1.x-p0.x)\*(p2.y-p0.y)-(p2.x-p0.x)\*(p1.y-p0.y);

}

double solve()

{

if(n<3) return 0;

double ans=0;

int i=0,j,k;

P[n]=P[0];

for(int i=0;i<n;++i)

{

j=(i+1)%n;k=(j+1)%n;

while(i!=k)

{

while(j!=k&&cross(P[i],P[k],P[j])-cross(P[i],P[k],P[j+1])<-eps)

j=(j+1)%n;

ans=max(ans,cross(P[i],P[k],P[j])/2);

k=(k+1)%n;

}

}

return ans;

}

void Graham()

{

sort(P,P+n);

int top=1;

sta[0]=P[0];sta[1]=P[1];

for(int i=2;i<n;++i)

{

while(top&&cross(sta[top-1],sta[top],P[i])>-eps) --top;

sta[++top]=P[i];

}

int k=top;

for(int i=n-2;i>=0;--i)

{

while(top>k&&cross(sta[top-1],sta[top],P[i])>-eps) --top;

sta[++top]=P[i];

}

if(top) --top;

for(int i=0;i<=top;++i) P[i]=sta[i];

n=top+1;

}

int main()

{

//freopen("in.txt","r",stdin);

//freopen("out.txt","w",stdout);

while(scanf("%d",&n),n!=-1)

{

for(int i=0;i<n;++i) P[i].input();

Graham();

printf("%.2f\n",solve());

}

return 0;

}

//最小矩形覆盖

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <cmath>

using namespace std;

#define N 50010

const double eps=1e-12;

struct Point{

double x,y,angle;

Point(){}

Point(double \_x,double \_y):x(\_x),y(\_y){}

void input()

{

scanf("%lf%lf",&x,&y);

}

void output()

{

printf("%.5f %.5f\n",x+eps,y+eps);

}

void f(double &a,double &b)

{

a=x;b=y;

}

bool operator<(const Point &a)const

{

return y<a.y||y==a.y&&x<a.x;

}

Point operator-(const Point &a)const

{

return Point(x-a.x,y-a.y);

}

double operator\*(const Point &a)const

{

return x\*a.x+y\*a.y;

}

bool operator!=(const Point &a)const

{

return x!=a.x||y!=a.y;

}

Point operator+(const Point &a)const

{

return Point(x+a.x,y+a.y);

}

}P[N],sta[N],Q[5];

//Q数组记录最小矩形的四个点

int n;

const double inf=1e20;

double cross(Point p0,Point p1,Point p2)

{

return (p1.x-p0.x)\*(p2.y-p0.y)-(p2.x-p0.x)\*(p1.y-p0.y);

}

void Graham()

{

sort(P,P+n);

int cnt=0;

for(int i=1;i<n;++i)

if(P[i]!=P[i-1]) P[++cnt]=P[i];

int top=1;

sta[0]=P[0];sta[1]=P[1];

for(int i=2;i<=cnt;++i)

{

while(top&&cross(sta[top-1],sta[top],P[i])>-eps) --top;

sta[++top]=P[i];

}

int k=top;

for(int i=cnt-1;i>=0;--i)

{

while(top>k&&cross(sta[top-1],sta[top],P[i])>-eps) --top;

sta[++top]=P[i];

}

if(top) --top;

for(int i=0;i<=top;++i) P[i]=sta[i];

n=top+1;

}

double dis(Point p1,Point p2)

{

return sqrt((p1.x-p2.x)\*(p1.x-p2.x)+(p1.y-p2.y)\*(p1.y-p2.y));

}

double dis\_lp(Point p1,Point p2,Point p3)

{

return fabs(cross(p1,p2,p3))/dis(p1,p2);

}

Point intersect(Point p1,Point p2,Point p3)//过点p3与向量p1p2垂直的向量与直线p1p2的交点

{

double x1,x2,x3,y1,y2,y3,d,dx,dy,a1,b1,c1,a2,b2,c2;

p1.f(x1,y1);p2.f(x2,y2);p3.f(x3,y3);

a1=x2-x1;b1=y2-y1;c1=x3\*(x2-x1)+y3\*(y2-y1);

a2=y2-y1;b2=x1-x2;c2=x1\*y2-x2\*y1;

d=a1\*b2-a2\*b1;dx=c1\*b2-c2\*b1;dy=a1\*c2-a2\*c1;

return Point(dx/d,dy/d);

}

double solve()

{

P[n]=P[0];

int i1=0,i2=0;

for(int i=1;i<n;++i)

{

if(P[i].y<P[i1].y) i1=i;

if(P[i].y>P[i2].y) i2=i;

}

int i3=i1+1,i4=i2+1;

double ans=inf,a,h;

for(int i=0;i<n;++i)

{

while(cross(P[i1],P[i2],P[i1+1])-cross(P[i1],P[i2+1],P[i1+1])<-eps)

i2=(i2+1)%n;

h=dis\_lp(P[i1],P[i1+1],P[i2]);

while((P[i1+1]-P[i1])\*(P[i3+1]-P[i3])>eps) i3=(i3+1)%n;

while((P[i1+1]-P[i1])\*(P[i4+1]-P[i4])<-eps) i4=(i4+1)%n;

a=fabs((P[i1+1]-P[i1])\*(P[i3]-P[i4]))/dis(P[i1+1],P[i1]);

if(ans>a\*h)

{

ans=a\*h;

Q[0]=intersect(P[i1],P[i1+1],P[i3]);

Q[1]=intersect(P[i1],P[i1+1],P[i4]);

Point p=P[i2]+(P[i1]-P[i1+1]);

Q[2]=intersect(P[i2],p,P[i3]);

Q[3]=intersect(P[i2],p,P[i4]);

}

i1=(i1+1)%n;

}

return ans;

}

bool cmp(Point p1,Point p2)

{

return p1.angle<p2.angle;

}

int main()

{

scanf("%d",&n);

for(int i=0;i<n;++i) P[i].input();

Graham();

if(n==1)

{

printf("%.5f\n",0);

for(int i=1;i<=4;++i)

P[0].output();

}

else if(n==2)

{

printf("%.5f\n",0);

sort(P,P+1);

P[0].output();P[0].output();

P[1].output();P[1].output();

}

else

{

printf("%.5f\n",solve()+eps);

sort(Q,Q+4);

for(int i=1;i<4;++i)

Q[i].angle=atan2(Q[i].y-Q[0].y,Q[i].x-Q[0].x);

sort(Q+1,Q+4,cmp);

for(int i=0;i<4;++i)

Q[i].output();

}

}

//CDQ分治-动态求矩形内数字和

/\*

你有一个N\*N的棋盘，每个格子内有一个整数，初始时的时候全部为0，现在需要维护两种操作：

1 x y A 1<=x,y<=N，A是正整数，将格子x,y里的数字加上A

2 x1 y1 x2 y2，1<=x1<= x2<=N，1<=y1<= y2<=N，输出x1 y1 x2 y2这个矩形内的数字和

3 无 终止程序

\*/

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

#define N 500010

#define M 200010

struct rec{

int x,y,t,k,w,be;

}a[M\*4],tmp[M\*4];

int c[N],ans[M];

int n,m;

void add(int x,int w)

{

for(;x<=n;x+=x&-x) c[x]+=w;

}

int getsum(int x)

{

int ans=0;

for(;x;x-=x&-x) ans+=c[x];

return ans;

}

bool cmp(const rec &a,const rec &b)

{

return a.y<b.y||a.y==b.y&&a.k<b.k;

}

void solve(int l,int r)

{

if(l==r) return;

int mid=l+r>>1;

for(int i=l;i<=r;++i)

if(a[i].t<=mid&&a[i].k==1) add(a[i].x,a[i].w);

else if(a[i].t>mid&&a[i].k==2) ans[a[i].be]+=getsum(a[i].x)\*a[i].w;

for(int i=l;i<=r;++i)

if(a[i].t<=mid&&a[i].k==1) add(a[i].x,-a[i].w);

int ll=l,rr=mid+1;

for(int i=l;i<=r;++i)

if(a[i].t<=mid) tmp[ll++]=a[i];

else tmp[rr++]=a[i];

for(int i=l;i<=r;++i) a[i]=tmp[i];

solve(l,mid);solve(mid+1,r);

}

int main()

{

scanf("%d",&n);

int tot=0,k,x1,y1,x2,y2,w;

while(true)

{

scanf("%d",&k);

if(k==1)

{

scanf("%d%d%d",&x1,&y1,&w);

a[++tot].x=x1;a[tot].y=y1;a[tot].t=tot;a[tot].k=k;a[tot].w=w;

}

else if(k==2)

{

scanf("%d%d%d%d",&x1,&y1,&x2,&y2);

a[++tot].x=x2;a[tot].y=y2;a[tot].t=tot;a[tot].k=2;a[tot].w=1;a[tot].be=++m;

a[++tot].x=x2;a[tot].y=y1-1;a[tot].t=tot;a[tot].k=2;a[tot].w=-1;a[tot].be=m;

a[++tot].x=x1-1;a[tot].y=y2;a[tot].t=tot;a[tot].k=2;a[tot].w=-1;a[tot].be=m;

a[++tot].x=x1-1;a[tot].y=y1-1;a[tot].t=tot;a[tot].k=2;a[tot].w=1;a[tot].be=m;

}

else break;

}

sort(a+1,a+tot+1,cmp);

solve(1,tot);

for(int i=1;i<=m;++i) printf("%d\n",ans[i]);

return 0;

}

//CDQ分治-动态求矩形内点的数量

/\*

1 x y z 添加一个点

2 x1 y1 z1 x2 y2 z2 查找长方体内点的数量

CDQ分治套CDQ分治套树状数组

第一重CDQ分治计算左边的修改对右边的询问的影响，

第二重CDQ分治处理三维偏序问题

\*/

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

#define N 50010

struct rec{

int k,x,y,z,w,id;

rec(){}

rec(int k,int x,int y,int z,int w,int id):

k(k),x(x),y(y),z(z),w(w),id(id){}

bool operator==(const rec &a)

{

return x==a.x&&y==a.y&&z==a.z;

}

}q[N\*8],qq[N\*8],tmp[N\*8];

int ans[N],tree[N\*2],a[N\*2],b[N\*2],c[N\*2],kind[N];

int n,ta,tb,tc;

bool cmp1(rec &a,rec &b)

{

if(a==b) return a.id<b.id;

return a.x<b.x||a.x==b.x&&a.y<b.y||a.x==b.x&&a.y==b.y&&a.z<b.z;

}

bool cmp2(rec &a,rec &b)

{

return a.y<b.y;

}

inline void add(int x,int w)

{

for(;x<=tc;x+=x&-x) tree[x]+=w;

}

inline int getsum(int x)

{

int ans=0;

for(;x;x-=x&-x) ans+=tree[x];

return ans;

}

void cdq(int l,int r)

{

if(l==r) return;

int mid=l+r>>1;

for(int i=l;i<=r;++i) tmp[i]=qq[i];

sort(qq+l,qq+mid+1,cmp2);

sort(qq+mid+1,qq+r+1,cmp2);

int ll=l;

for(int i=mid+1;i<=r;++i)

{

while(ll<=mid&&qq[ll].y<=qq[i].y)

{

if(qq[ll].k==1) add(qq[ll].z,1);

++ll;

}

if(qq[i].k==2) ans[qq[i].id]+=getsum(qq[i].z)\*qq[i].w;

}

for(int i=l;i<ll;++i)

if(qq[i].k==1) add(qq[i].z,-1);

for(int i=l;i<=r;++i) qq[i]=tmp[i];

cdq(l,mid);cdq(mid+1,r);

}

void solve(int l,int r)

{

if(l==r) return;

int mid=l+r>>1,n=0;

for(int i=l;i<=mid;++i)

if(q[i].k==1) qq[++n]=q[i];

for(int i=mid+1;i<=r;++i)

if(q[i].k==2) qq[++n]=q[i];

if(n)

{

sort(qq+1,qq+n+1,cmp1);

cdq(1,n);

}

solve(l,mid);solve(mid+1,r);

}

int main()

{

int ca;

scanf("%d",&ca);

while(ca--)

{

scanf("%d",&n);

int nn=0,k,a1,b1,c1,a2,b2,c2;

ta=tb=tc=0;

for(int i=1;i<=n;++i)

{

scanf("%d%d%d%d",&k,&a1,&b1,&c1);

kind[i]=k;

if(k==1)

{

q[++nn]=rec(k,a1,b1,c1,1,i);

c[++tc]=c1;

}

else

{

scanf("%d%d%d",&a2,&b2,&c2);

--a1;--b1;--c1;

c[++tc]=c1;c[++tc]=c2;

q[++nn]=rec(k,a2,b2,c2,1,i);

q[++nn]=rec(k,a1,b2,c2,-1,i);

q[++nn]=rec(k,a2,b1,c2,-1,i);

q[++nn]=rec(k,a2,b2,c1,-1,i);

q[++nn]=rec(k,a1,b1,c2,1,i);

q[++nn]=rec(k,a1,b2,c1,1,i);

q[++nn]=rec(k,a2,b1,c1,1,i);

q[++nn]=rec(k,a1,b1,c1,-1,i);

}

}

sort(c+1,c+tc+1);tc=unique(c+1,c+tc+1)-c-1;

for(int i=1;i<=nn;++i)

q[i].z=lower\_bound(c+1,c+tc+1,q[i].z)-c;

memset(ans,0,sizeof(ans));

solve(1,nn);

for(int i=1;i<=n;++i)

if(kind[i]==2) printf("%d\n",ans[i]);

}

return 0;

}

//树分治-treedp计数

/\*

给定一棵n个节点的树，树上的权值要么为0，要么为1，

如果一条路径上的0和1数量相同且路径上存在一个点（称为休息点）（非路径的端点），

在这个点的左边路径上0和1的数量相等，右边路径上的0和1的数量也相等，

则称这条路径符合条件，问符合条件的路有多少条。

解：先把树上的权值0变成-1，

在计数的时候，运用treedp的思想，

在当前根中，

la[x][0/1]表示当前根中以前访问的子树到根的路径权值为x的没有/有休息点的节点数量

g[x][0/1]表示当前根中现在访问的节点的子树到根的路径权值为x的没有/有休息点的节点数量

点的编号为1~n

\*/

#include<iostream>

#include<stdio.h>

#include<algorithm>

#include<string.h>

#include<vector>

#define N 100010

typedef long long ll;

using namespace std;

struct node{

int x,y;

node(int xx,int yy)

{

x=xx,y=yy;

}

};

vector<node> lin[N];

vector<int> d;

int sz[N],f[N];//sz[x]-->x的树大小，f[x]-->x最大子树的节点数；

int n;

int rt;

int vis[N],dis[N],g[N\*2][2],la[N\*2][2],s[N\*2];

ll ans,sum;

int size;

int aa,bb,cc,md;

void getrt(int x,int fa)//利用\*sz,\*f求重心

{

sz[x]=1;

f[x]=0;

for(int i=0;i<lin[x].size();i++)

{

int u=lin[x][i].x;

if(vis[u]||u==fa) continue;

getrt(u,x);

sz[x]+=sz[u];

f[x]=max(sz[u],f[x]);

}

f[x]=max(f[x],size-sz[x]);//! x最大子树的节点数=max(与此子树大小－f[x],f[x])

if(f[x]<f[rt]) rt=x;

}

void getdis(int x,int fa)//一遍dfs求距离＋求重心的一点预处理sz[x],求子树大小size

{

sz[x]=1;

for(int i=0;i<lin[x].size();i++)

{

int u=lin[x][i].x;

if(vis[u]||u==fa) continue;

getdis(u,x);

sz[x]+=sz[u];

}

}

void dfs(int x,int fa,int w)//求出g[][]

{

//printf("%d %d ",x,w);

md=max(md,max(w,-w));

if(s[w+N]) ++g[w+N][1];

else ++g[w+N][0];

//printf("%d %d !!\n",g[w+N][0],g[w+N][1]);

++s[w+N];

int y;

for(int i=0;i<lin[x].size();++i)

{

y=lin[x][i].x;

if(vis[y]||y==fa) continue;

dfs(y,x,w+lin[x][i].y);

}

--s[w+N];

}

ll cal(int x)

{

//printf("%d ----------\n",x);

int y;

ll tmp=0;

int mdd=0;

for(int i=0;i<lin[x].size();++i)

{

y=lin[x][i].x;

if(vis[y]) continue;

md=0;

dfs(y,x,lin[x][i].y);

mdd=max(mdd,md);

tmp+=g[N][1]+(ll)g[N][0]\*la[N][0];

for(int j=-md;j<=md;++j)

{

tmp+=(ll)g[N-j][0]\*la[N+j][1]+(ll)g[N-j][1]\*la[N+j][0]+(ll)g[N-j][1]\*la[N+j][1];

//printf("%d %d %d %d %d %d ??\n",y,j,g[N+j][0],g[N+j][1],la[N+j][0],la[N+j][1]);

}

for(int j=N-md;j<=N+md;++j)

{

la[j][1]+=g[j][1];

la[j][0]+=g[j][0];

g[j][1]=g[j][0]=0;

}

}

for(int i=N-mdd;i<=N+mdd;++i)

la[i][0]=la[i][1]=0;

//printf("%d ?\n",tmp);

return tmp;

}

void solve(int x)

{

ans+=cal(x);

vis[x]=1;

for(int i=0;i<lin[x].size();i++)

{

int u=lin[x][i].x;

if(vis[u]) continue;

getdis(u,0);

size=sz[u];//!!! getdis中已经处理子树的全大小

getrt(u,rt=0);

solve(rt);

}

}

int main()

{

while(scanf("%d",&n)!=EOF)

{

for(int i=1;i<=n;i++)

{

vis[i]=0;

lin[i].clear();

}

for(int i=1;i<n;i++)

{

scanf("%d%d%d",&aa,&bb,&cc);

if(cc==0) cc=-1;

lin[aa].push\_back(node(bb,cc));

lin[bb].push\_back(node(aa,cc));

}

ans=0;

f[0]=n+1;size=n;

getrt(1,rt=0);

solve(rt);

printf("%lld\n",ans);

}

}

/\*

求树上距离小于K的点对的数量，

对于一个节点，先算出以它为根的子树的符合条件的点对的数量

再减去以它儿子为根的子树的符合条件的点对的数量，

求点对数量时用双指针

点的编号为1~n

\*/

#include<iostream>

#include<stdio.h>

#include<algorithm>

#include<string.h>

#include<vector>

#define N 10005

using namespace std;

struct node{

int x,y;

node(int xx,int yy)

{

x=xx,y=yy;

}

};

vector<node> lin[N];

vector<int> d;

int sz[N],f[N];//sz[x]-->x的树大小，f[x]-->x最大子树的节点数；

int n;

int rt;

int vis[N],dis[N];

int K;

int ans,size;

int aa,bb,cc;

void getrt(int x,int fa)//利用\*sz,\*f求重心

{

sz[x]=1;

f[x]=0;

for(int i=0;i<lin[x].size();i++)

{

int u=lin[x][i].x;

if(vis[u]||u==fa) continue;

getrt(u,x);

sz[x]+=sz[u];

f[x]=max(sz[u],f[x]);

}

f[x]=max(f[x],size-sz[x]);//! x最大子树的节点数=max(与此子树大小－f[x],f[x])

if(f[x]<f[rt]) rt=x;

}

void getdis(int x,int fa)//一遍dfs求距离＋求重心的一点预处理sz[x],求子树大小size

{

sz[x]=1;

d.push\_back(dis[x]);

for(int i=0;i<lin[x].size();i++)

{

int u=lin[x][i].x;

if(vis[u]||u==fa) continue;

dis[u]=dis[x]+lin[x][i].y;

getdis(u,x);

sz[x]+=sz[u];

}

}

int cal(int x,int y)

{

int ret=0;

d.clear();

dis[x]=y;

getdis(x,0);

sort(d.begin(),d.end());

int l=0;

int r=d.size()-1;

while(l<r)

{

while(d[l]+d[r]>K&l<r) r--;

ret+=r-l;

l++;

}

return ret;

}

void solve(int x)

{

//cout<<x<<endl;

ans+=cal(x,0);

vis[x]=1;

for(int i=0;i<lin[x].size();i++)

{

int u=lin[x][i].x;

if(vis[u]) continue;

ans-=cal(u,lin[x][i].y);

f[0]=size=sz[u];//!!! getdis中已经处理子树的全大小

getrt(u,rt=0);

solve(rt);

}

}

int main()

{

while(scanf("%d%d",&n,&K))

{

if(!n&&!K) return 0;

for(int i=1;i<=n;i++)

{

vis[i]=0;

lin[i].clear();

}

for(int i=1;i<n;i++)

{

scanf("%d%d%d",&aa,&bb,&cc);

lin[aa].push\_back(node(bb,cc));

lin[bb].push\_back(node(aa,cc));

}

ans=0;

f[0]=size=n;

getrt(1,rt=0);

solve(rt);

printf("%d\n",ans);

}

}

//hash找相同连续子串

//2017.1.7组队赛遇见的字符串处理题

//给两个串S和T，长度4000，找出各自最长的连续子串，使得两个子串的组成字母完全相同

//TL:10S

//思路：暴力找出S所有子串的hash值，扔进set，从大到小枚举ans，在T中找是否存在

/\*需要注意的是，每当unordered\_set（hash表实现的set）内部进行一次扩容或者缩容，

都需要对表中的数据重新计算，也就是说，扩容或者缩容的时间复杂度至少为O(N)\*/

#include <bits/stdc++.h>

using namespace std;

typedef long long LL;

typedef unsigned long long ull;

char s[4008], t[4008];

LL a[4008][28]={0}, b[4008][28]={0};

int slen, tlen;

int t1[4008], t2[4008];

unordered\_set<ull> S;

bool check(int x)

{

for(int i=1; i+x-1<=tlen; ++i)//检查长度为x的子串的hash值是否出现在S中

{

ull r = 2333;

for(int j=0; j<26; ++j) r += (r<<4)+b[i+x-1][j]-b[i-1][j];

if(S.count(r)) return true;

}

return false;

}

int main()

{

scanf("%s%s",s+1,t+1);

slen = strlen(s+1);

tlen = strlen(t+1);

for(int i=1; i<=slen; ++i)//遍历整个字符串

{

for(int j=0; j<26; ++j)//遍历所有字母,当前长度的字母个数等于上一长度的个数

a[i][j] = a[i-1][j];

++a[i][s[i]-'a']; //当前下标的字母数+1

}

for(int i=1; i<=tlen; ++i)

{

for(int j=0; j<26; ++j) b[i][j] = b[i-1][j];

++b[i][t[i]-'a'];

}

for(int i=1; i<=slen; ++i)//子串的起始位置

{

for(int j=i; j<=slen; ++j)//子串从起始位置到结束位置的遍历

{

ull r = 2333;//r为种子

for(int k=0; k<26; ++k) r += (r<<4)+a[j][k]-a[i-1][k];//遍历所有字母

S.insert(r);

//插入子串hash值

}

}

int ans = min(slen, tlen);

while(ans)

{

if(check(ans)) break;

--ans;

}

printf("%d\n",ans);

return 0;

}

strstr是一种函数，从字符串str1中查找是否有字符串str2，如果有，从str2的位置起，返回str1的指针，否则返回null

strstr（str1，str2）

//hash长度相同的

#include<bits/stdc++.h>

using namespace std;

const int maxn = 100010;

typedef unsigned long long ull;

char s[maxn];

ull nbase [maxn];

ull hs[maxn];

ull h[maxn];

ull seed = 31;

int main()

{

int n, m;

nbase[1] = seed;

for(int i = 2; i <= maxn; ++i)

{

nbase[i] = nbase[i-1]\*seed;

}

while(scanf("%d%d", &n, &m) == 2)

{

scanf("%s", s+1);

int len = strlen(s+1);

hs[len+1] = 0;

for(int i = len; i > 0; --i)

{

hs[i] = hs[i+1]\*seed+s[i]-'a';

}

for(int i = 1; i+m-1 <= len; ++i)

{

h[i] = hs[i]-hs[i+m]\*nbase[m];

}

int ans = 0;

for(int i = 1; i <= m && i+n\*m-1 <= len; ++i)

{

map<ull, int> mp;

for(int j = 0; j < n; ++j)

{

ull th = h[i+j\*m];

mp[th]++;

}

if(mp.size() == n)

ans++;

for(int j = n; i+((j+1)\*m)-1 <= len; ++j)

{

ull th = h[i+(j-n)\*m];

mp[th]--;

if(!mp[th]) mp.erase(th);

th = h[i+j\*m];

mp[th]++;

if(mp.size() == n)

ans++;

}

}

printf("%d\n", ans);

}

return 0;

}

//ac自动机动态内存版

#include<bits/stdc++.h>

using namespace std;

#define maxn 500005

char s[1000006];

struct node

{

int cnt;

node \*fail;

node \*child[26];

node()

{

cnt=0;

fail = NULL;

memset(child,0,sizeof(child));

}

}\*que[maxn];

void insert(node \*head,char \*keyword)

{

int len = strlen(keyword);

node \*p = head;

for(int i=0;i<len;i++)

{

int id = keyword[i] -'a';

if(p->child[id]==NULL)

{

p->child[id] = new node;

}

p = p->child[id];

}

p->cnt++;

}

void buildAC(node \*root)

{

int head=0,tail=0;

que[tail++] = root;

while(tail!=head)

{

node \*p = que[head++];

for(int i=0;i<26;i++)

{

if(p->child[i]!=NULL)

{

if(p==root)

{

p->child[i]->fail = root;

}

else

{

node \*temp = p->fail;

while(temp!=NULL)

{

if(temp->child[i] !=NULL)

{

p->child[i]->fail = temp->child[i];

break;

}

temp = temp->fail;

}

if(temp==NULL)

{

p->child[i]->fail = root;

}

}

que[tail++] = p->child[i];

}

}

}

}

int query(node \*root)

{

int ret = 0;

int len = strlen(s);

node \*p = root;

for(int i=0;i<len;i++)

{

int id = s[i]-'a';

while(p!=root && p->child[id] == NULL)

{

p = p->fail;

}

p = p->child[id];

if(p==NULL)

{

p = root;

}

node \*temp = p;

while(temp != root && temp->cnt != -1)

{

ret += temp->cnt;

temp->cnt = -1;

temp = temp->fail;

}

}

return ret;

}

void deletenode(node \*u)

{

if(u==NULL)return;

for(int i=0;i<26;i++)

{

deletenode(u->child[i]);

}

delete u;

}

int main()

{

char keyword[55];

int ca;cin>>ca;

node \*root = NULL;

int n;

while(ca--)

{

scanf("%d",&n);

root = new node;

for(int i=0;i<n;i++)

{

scanf("%s",keyword);

insert(root,keyword);

}

buildAC(root);

scanf("%s",s);

printf("%d\n",query(root));

deletenode(root);

}

return 0;

}

//静态内存版

#include<bits/stdc++.h>

using namespace std;

struct node

{

int count;//尾部标记

node\* fail;//指向其他字符串

node\* child[26];

node()

{

count=0;

fail=NULL;

memset(child,0,sizeof(child));

}

};

char s[1000006];

node arr[500005];

node\* root;

int k;

void insert(char \*s)

{

int len = strlen(s);

node\*p = root;

for(int i=0;i<len;i++)

{

int m = s[i]-'a';

if(p->child[m]==NULL)

{

p->child[m]= &arr[k++];

}

p=p->child[m];

}

p->count++;

}

void getfail()

{

queue<node\*>q;

q.push(root);

while(!q.empty())

{

node\* p = q.front();

q.pop();

for(int i=0;i<26;i++)

{

if(p->child[i]!=NULL)

{

q.push(p->child[i]);

node\* t = p->fail;

while(t)

{

if(t->child[i]!=NULL)

{

p->child[i]->fail = t->child[i];

break;

}

t = t->fail;

}

if(t==NULL)

p->child[i]->fail = root;

}

}

}

}

int acsearch(char \*s)

{

node\*p = root;

int len = strlen(s);

int ret = 0;

for(int i=0;i<len;i++)

{

int m = s[i]-'a';

while(p!=NULL && p->child[m]==NULL)

p = p->fail;

if(p==NULL)

{

p=root;continue;

}

node\* t = p->child[m];

p = p->child[m];

while(t!=NULL && t->count!=-1)

{

if(t->count)

{

ret += t->count;

t->count = -1;

}

t = t->fail;

}

}

return ret;

}

int main()

{

int n;cin>>n;

while(n--)

{

root = &arr[0];

int m;

k=1;

cin>>m;

for(int i=0;i<m;i++)

{

scanf("%s",s);

insert(s);

}

getfail();

scanf("%s",s);

printf("%d\n",acsearch(s));

for(int i=0;i<k;i++)

{

arr[i].count=0;

arr[i].fail=NULL;

for(int j=0;j<26;j++)

{

arr[i].child[j]=NULL;

}

}

}

return 0;

}

//线段树单点修改，区间求和

#define M 50005

#define lson l,m,rt<<1

#define rson m+1,r,rt<<1|1

/\*left,right,root,middle\*/

int sum[M<<2];

inline void PushPlus(int rt)

{

sum[rt] = sum[rt<<1] + sum[rt<<1|1];

}

void Build(int l, int r, int rt)

{

if(l == r)

{

scanf("%d", &sum[rt]);

return ;

}

int m = ( l + r )>>1;

Build(lson);

Build(rson);

PushPlus(rt);

}

void Updata(int p, int add, int l, int r, int rt)

{

if( l == r )

{

sum[rt] += add;

return ;

}

int m = ( l + r ) >> 1;

if(p <= m)

Updata(p, add, lson);

else

Updata(p, add, rson);

PushPlus(rt);

}

int Query(int L,int R,int l,int r,int rt)

{

if( L <= l && r <= R )

{

return sum[rt];

}

int m = ( l + r ) >> 1;

int ans=0;

if(L<=m )

ans+=Query(L,R,lson);

if(R>m)

ans+=Query(L,R,rson);

return ans;

}

int main()

{

int T, n, a, b;

scanf("%d",&T);

for( int i = 1; i <= T; ++i )

{

printf("Case %d:\n",i);

scanf("%d",&n);

Build(1,n,1);

char op[10];

while( scanf("%s",op) &&op[0]!='E' )

{

scanf("%d %d", &a, &b);

if(op[0] == 'Q')

printf("%d\n",Query(a,b,1,n,1));

else if(op[0] == 'S')

Updata(a,-b,1,n,1);

else

Updata(a,b,1,n,1);

}

}

return 0;

}

//线段树区间修改，区间求和

#include<iostream>

#include<cstdio>

using namespace std;

#define lson l,mid,rt<<1

#define rson mid+1,r,rt<<1|1

#define intmid int mid = (l+r)>>1

#define maxn 100555

typedef long long ll;

int n,q;

ll sum[maxn<<2],cur[maxn<<2];

void push\_up(int rt)

{

sum[rt] = sum[rt<<1] + sum[rt<<1|1];

}

void push\_down(int l,int r,int rt)

{

if(cur[rt])

{

intmid;

cur[rt<<1] += cur[rt];

cur[rt<<1|1]+= cur[rt];

sum[rt<<1] += (mid-l+1)\*cur[rt];

sum[rt<<1|1]+= (r-mid)\*cur[rt];

cur[rt]=0;

}

}

void build(int l,int r,int rt)

{

intmid;

cur[rt]=0;

if(l==r)

{

scanf("%lld",&sum[rt]);

return ;

}

push\_down(l,r,rt);

build(lson);

build(rson);

push\_up(rt);

}

void update(int ll,int rr,int val,int l,int r,int rt)

{

intmid;

if( ll<=l && rr>=r )

{

cur[rt]+=val;

sum[rt]+=(r-l+1)\*val;

return ;

}

push\_down(l,r,rt);

if(ll<=mid)

update(ll,rr,val,lson);

if(rr>mid)

update(ll,rr,val,rson);

push\_up(rt);

}

ll query(int ll,int rr,int l,int r,int rt)

{

long long ret=0;

if(ll<=l && rr>=r)

{

return sum[rt];

}

push\_down(l,r,rt);

intmid;

if(ll<=mid) ret += query(ll,rr,lson);

if(rr>mid) ret += query(ll,rr,rson);

return ret;

}

int main()

{

cin>>n>>q;

build(1,n,1);

char s[2];

int a,b,c;

while(q--)

{

scanf("%s%d%d",s,&a,&b);

if(s[0]=='C')

{

scanf("%d",&c);

update(a,b,c,1,n,1);

}

else

printf("%lld\n",query(a,b,1,n,1));

// printf("%lld%lld%lld",sum[17],sum[9],sum[10]);

}

return 0;

}

//树状数组板子

#include<bits/stdc++.h>

using namespace std;

#define maxn 1000060

int a[maxn],c[maxn];

int lowbit(int x)

{

return x&(-x);

}

int n;

int sum(int i)

{

int s=0;

while(i)

{

s+=c[i];

i-=lowbit(i);

}

return s;

}

void add(int i,int v)

{

while(i<=n)

{

c[i]+=v;

i+=lowbit(i);

}

}

int main()

{

int a,b;

while(scanf("%d",&n)!=EOF,n)

{

memset(c,0,sizeof(c));

for(int i=0;i<n;i++)

{

scanf("%d%d",&a,&b);

add(a,1);

add(b+1,-1);

}

for(int i=1;i<n;i++)

printf("%d ",sum(i));

printf("%d\n",sum(n));

}

return 0;

}

//二维树状数组板子

#define maxn 2005

#define lowbit(i) ((i)&(-i))

int op,n,x,y,k;

int c[maxn][maxn];

int sum(int x,int y)

{

int s=0;

for(int i=x;i;i-=lowbit(i))

{

for(int j=y;j;j-=lowbit(j))

{

s+=c[i][j];

}

}

return s;

}

void add(int x,int y,int v)

{

for(int i=x;i<=n;i+=lowbit(i))

{

for(int j=y;j<=n;j+=lowbit(j))

{

c[i][j]+=v;

}

}

}

int main()

{

while(scanf("%d%d",&op,&n)!=EOF)

{

memset(c,0,sizeof(c));

while(1)

{

scanf("%d",&op);

if(op==1)

{

scanf("%d%d%d",&x,&y,&k);

add(x+1,y+1,k);

}

else if(op==2)

{

int l,r,t,d;

scanf("%d%d%d%d",&l,&d,&r,&t);

l++,r++,t++,d++;

printf("%d\n",sum(r,t)-sum(r,d-1)-sum(l-1,t)+sum(l-1,d-1));

}

else break;

}

}

return 0;

}

//建树的动态申请空间和静态申请空间

//动态

#define ll long long

int t,n,a,ok;

char s[100];

struct trie

{

trie \*child[11];

bool qwq;

trie()

{

qwq=false;

memset(child,0,sizeof(child));

}

};

trie \*root;

trie \*now,\*next1;

void insert(char \*s)

{

now=root;

int len = strlen(s);

for(int i=0;i<len;i++)

{

if(now->qwq==true)ok=0;

int m=s[i]-'0';

if(now->child[m]!=NULL)

{

now = now->child[m];

}

else

{

next1 = new trie;

now->child[m] = next1;

now = next1;

}

if(i==len-1)

{

now->qwq=true;

}

}

for(int i=0;i<10;i++)

{

if(now->child[i]!=NULL)

ok=0;

}

}

void removetrie(trie\* u)

{

if(u==NULL) return ;

for(int i=0;i<10;i++)

removetrie(u->child[i]);

delete u;

}

int main()

{

cin>>t;

while(t--)

{

root = new trie;

ok=1;

cin>>n;

for(int i=0;i<n;i++)

{

scanf("%s",s);

insert(s);

}

cout<<((ok)?"YES\n":"NO\n");

removetrie(root);

}

return 0;

}

//静态

#define ll long long

int t,n,a,ok;

int cnt;

char s[100];

struct trie

{

trie \*child[11];

bool qwq;

void init()

{

qwq=false;

memset(child,0,sizeof(child));

}

}p[100005];

inline trie\* newtrie()

{

p[cnt].init();

return &p[cnt++];

}

trie \*root = NULL;

trie \*now,\*next1;

void insert(char \*s)

{

now = root;

next1 = NULL;

int len = strlen(s);

for(int i=0;i<len && ok;i++)

{

if(now->qwq==true){ok=0;return;}

int m=s[i]-'0';

if(now->child[m]!=NULL)

{

now = now->child[m];

}

else

{

next1 = newtrie();

now->child[m] = next1;

now = next1;

}

if(i==len-1)

{

now->qwq=true;

}

}

for(int i=0;i<10;i++)

{

if(now->child[i]!=NULL)

{ok=0;return ;}

}

}

int main()

{

cin>>t;

while(t--)

{

cnt=0;

root = newtrie();

ok=1;

cin>>n;

for(int i=0;i<n;i++)

{

scanf("%s",s);

if(ok)insert(s);

}

cout<<((ok)?"YES\n":"NO\n");

}

return 0;

}

//由前序遍历和中序遍历求后序遍历

#define maxn 3000

stack<int> tree;

int n,a1[maxn],a2[maxn];

void bt(int l1,int r1,int l2,int r2)

{

tree.push(a1[l1]);

int i,j;

for(i=l2;i<r2;i++)

if(a1[l1]==a2[i])break;

j = l1 + (i-l2) +1;

if(j<=r1 && i<r2)

bt(j,r1,i+1,r2);

if(l1+1<=j-1 && l2<=i-1)

bt(l1+1,j-1,l2,i-1);

}

int main()

{

while(cin>>n)

{

for(int i=0;i<n;i++)scanf("%d",a1+i);

for(int i=0;i<n;i++)scanf("%d",a2+i);

bt(0,n-1,0,n-1);

while(!tree.empty())

{

printf("%d",tree.top());

tree.pop();

if(!tree.empty())

printf(" ");

}

printf("\n");

}

return 0;

}

//并查集

int find(int a)

{

return (a==f[a])?a:f[a]=find(f[a]);

}

void check(int a,int b)

{

int fa=find(a);

int fb=find(b);

if(fa!=fb)

{

f[fa]=fb;

}

}

1.find函数根据情况选择要不要更新父亲

2.check函数根据情况，是把儿子相连，还是把最大的父亲相连

3.一个集合内有多种状态时，用val值区分

//卡特兰数对1e9+7取模

#include<iostream>

#include<cstdio>

#include<cstring>

using namespace std;

typedef \_\_int64 ll;

const int mod = 1e9+7;

const int maxn = 300;

ll fac[maxn] = {1};

void init()

{

for(int i = 1; i < maxn; i++)

fac[i] = fac[i-1]\*i%mod;

}

ll qmod(ll x, ll p)

{

ll ans = 1;

while(p)

{

if(p%2) ans = ans\*x%mod;

x = x\*x%mod;

p /= 2;

}

return ans;

}

int main()

{

init();

ll n;

while(scanf("%I64d",&n)!=EOF&&n>=0)

{

printf("%I64d\n", fac[2\*n]\*qmod(fac[n+1], mod-2)%mod\*qmod(fac[n], mod-2)%mod);

}

return 0;

}

//三角形内心

内心是角平分线的交点,到三边距离相等.

设:在三角形ABC中,三顶点的坐标为：A(x1,y1),B(x2,y2),C(x3,y3) BC=a,CA=b,AB=c

内心为M （X,Y）则有aMA+bMB+cMC=0（三个向量）

MA=（X1-X,Y1-Y）

MB=(X2-X,Y2-Y)

MC=(X3-X,Y3-Y)

则：a(X1-X)+b(X2-X)+c(X3-X)=0,a(Y1-Y)+b(Y2-Y)+c(Y3-Y)=0

∴X=（aX1+bX2+cX3)/(a+b+c),Y=（aY1+bY2+cY3)/(a+b+c)

∴M（（aX1+bX2+cX3)/(a+b+c),（aY1+bY2+cY3)/(a+b+c)）

//辛普森积分

#include<stdio.h>

#include<string.h>

#include<math.h>

#define eps 0.0000001

double r1,r2;

double F(double x)

{

return 8\*sqrt((r1\*r1-x\*x)\*(r2\*r2-x\*x));

}

double cal(double l, double r)

{

return (r-l)/6.0\*(F(r)+4.0\*F((r+l)/2.0)+F(l));

}

double simpson(double l,double r)

{

double m=(l+r)/2.0;

double fl=cal(l,m),fr=cal(m,r);

if(fabs(fl+fr-cal(l,r))<eps) return fr+fl;

else return simpson(l,m)+simpson(m,r);

}

int main()

{

freopen("twocyl.in","r",stdin);

freopen("twocyl.out","w",stdout);

scanf("%lf%lf",&r1,&r2);

double minr;

if(r1<r2) minr=r1;

else minr=r2;

printf("%.6f\n",simpson(0,minr));

}

//JAVA大数类用法

在Java中有两个类BigInteger和BigDecimal分别表示大整数类和大浮点数类，至于两个类的对象能表示最大范围不清楚，理论上能够表示无线大的数，只要计算机内存足够大。

这两个类都在java.math.\*包中，因此每次必须在开头处引用该包。

Ⅰ基本函数：

1.valueOf(parament); 将参数转换为制定的类型

比如 int a=3;

BigInteger b=BigInteger.valueOf(a);

则b=3;

String s=”12345”;

BigInteger c=BigInteger.valueOf(s);

则c=12345；

2.add(); 大整数相加

BigInteger a=new BigInteger(“23”);

BigInteger b=new BigInteger(“34”);

a. add(b);

3.subtract(); 相减

4.multiply(); 相乘

5.divide(); 相除取整

6.remainder(); 取余

7.pow(); a.pow(b)=a^b

8.gcd(); 最大公约数

9.abs(); 绝对值

10.negate(); 取反数

11.mod(); a.mod(b)=a%b=a.remainder(b);

12.max(); min();

13.punlic int comareTo();

14.boolean equals(); 是否相等

15.BigInteger构造函数：

一般用到以下两种：

BigInteger(String val);

将指定字符串转换为十进制表示形式；

BigInteger(String val,int radix);

将指定基数的 BigInteger 的字符串表示形式转换为 BigInteger

Ⅱ.基本常量：

A=BigInteger.ONE 1

B=BigInteger.TEN 10

C=BigInteger.ZERO 0

//产生大量测试数据

下面是随机产生int型数据的代码

[cpp] view plain copy

#include<iostream>

#include<ctime>

#include<cstdlib>

#include<cstdio>

#include<fstream>

#define digit 10

using namespace std;

string LLMax="9223372036854775807";

string IntMax="2147483647";

string RandomInt()

{

int n=rand()%digit+1;//接下来生成n位数

string s;

if(n==digit)//特殊处理，因为int型要在[-2^31,2^31-1]之内

{

s+=rand()%2+1+'0';

for(int i=1;i<n;i++)

{

int temp=rand()%10;

while(temp>IntMax[i]-'0') temp=rand()%10;

s+=temp+'0';

}

return s;

}

if(n==1) s+=rand()%10+'0';//

else s+=rand()%9+1+'0';//第一位为1-9之间的数

for(int i=2;i<=n;i++) s+=rand()%10+'0';//随机产生第2-n位上的数字

if(rand()%2==1) s='-'+s;//产生负数

return s;

}

int main()

{

int m=0;

ofstream in("int\_in.txt");

srand((unsigned)time(NULL));

for(int i=0;i<100;i++)

in<<RandomInt()<<endl;

return 0;

}

同理可以产生 long long型的

代码如下

[cpp] view plain copy

#include<iostream>

#include<ctime>

#include<cstdlib>

#include<cstdio>

#define digit 19

using namespace std;

string LLMax="9223372036854775807";

string IntMax="2147483647";

string RandomLL()

{

int n=rand()%digit+1;//接下来生成n位数

string s;

if(n==digit)//特殊处理，因为int型要在[-2^63,2^63-1]之内

{

s+=rand()%9+1+'0';

for(int i=1;i<n;i++)

{

int temp=rand()%10;

while(temp>LLMax[i]-'0') temp=rand()%10;

s+=temp+'0';

}

return s;

}

if(n==1) s+=rand()%10+'0';//

else s+=rand()%9+1+'0';//第一位为1-9之间的数

for(int i=2;i<=n;i++) s+=rand()%10+'0';//随机产生第2-n位上的数字

if(rand()%2==1) s='-'+s;//产生负数

return s;

}

int main()

{

int m=0;

srand((unsigned)time(NULL));

for(int i=0;i<100;i++)

cout<<RandomLL()<<endl;

return 0;

}

产生大整数

[cpp] view plain copy

#include<iostream>

#include<ctime>

#include<cstdlib>

#include<cstdio>

#include<fstream>

#define digit 100

using namespace std;

string LLMax="9223372036854775807";

string IntMax="2147483647";

string RandomBigInteger()

{

int n=rand()%digit+1;//接下来生成n位数

string s;

if(n==1) s+=rand()%10+'0';//

else s+=rand()%9+1+'0';//第一位为1-9之间的数

for(int i=2;i<=n;i++) s+=rand()%10+'0';//随机产生第2-n位上的数字

//if(rand()%2==1) s='-'+s;//产生负数

return s;

}

int main()

{

int m=0;

ofstream in("in.txt");

srand((unsigned)time(NULL));

for(int i=0;i<100;i++)

in<<RandomBigInteger()<<endl;

return 0;

}

产生字符串

[cpp] view plain copy

#include<iostream>

#include<ctime>

#include<cstdlib>

#include<cstdio>

#define digit 30

using namespace std;

string LLMax="9223372036854775807";

string IntMax="2147483647";

string SmellAlphabet="abcdefghijklmnopqrstuvwxyz";

string UpperAlphabet="ABCDEFGHIJKLMNOPQRSTUVWXYZ";

string RandomString()

{

int n=rand()%digit+1;//接下来生成n位字符串

string s;

for(int i=0;i<n;i++) s+=SmellAlphabet[rand()%26];//随机产生字符串

return s;

}

int main()

{

int m=0;

srand((unsigned)time(NULL));

for(int i=0;i<100;i++)

cout<<RandomString()<<endl;

return 0;

}

//最大空凸包

#include <iostream>

#include <cmath>

#include <cstdio>

#include <algorithm>

using namespace std;

typedef double type\_p;

const double eps = 1e-6;

const int maxn = 510;

double dp[maxn][maxn];

inline double eq(double x, double y)

{

return fabs(x-y)<eps;

}

inline int eq(int x, int y)

{

return x==y;

}

struct point

{

type\_p x,y;

};

type\_p xmult(point a, point b, point o)

{

return (a.x-o.x)\*(o.y-b.y)-(a.y-o.y)\*(o.x-b.x);//b at ao left if negative, at right if positive

}

type\_p dist(point a, point b)

{

return (a.x-b.x)\*(a.x-b.x)+(a.y-b.y)\*(a.y-b.y);

}

point o;

bool cmp\_angle(point a,point b)

{

if(eq(xmult(a,b,o),0.0))

{

return dist(a,o)<dist(b,o);

}

return xmult(a,o,b)>0;

}

/\*

Input: p: Point set

pn: size of the point set

Output: the area of the largest empty convex

\*/

double empty\_convex(point \*p, int pn)

{

double ans=0;

for(int i=0; i<pn; i++)

{

for(int j=0; j<pn; j++)

{

dp[i][j]=0;

}

}

for(int i=0; i<pn; i++)

{

int j = i-1;

while(j>=0 && eq(xmult(p[i], p[j], o),0.0))j--;//coline

bool flag= j==i-1;

while(j>=0)

{

int k = j-1;

while(k >= 0 && xmult(p[i],p[k],p[j])>0)k--;

double area = fabs(xmult(p[i],p[j],o))/2;

if(k >= 0)area+=dp[j][k];

if(flag) dp[i][j]=area;

ans=max(ans,area);

j=k;

}

if(flag)

{

for(int j=1; j<i; j++)

{

dp[i][j] = max(dp[i][j],dp[i][j-1]);

}

}

}

return ans;

}

double largest\_empty\_convex(point \*p, int pn)

{

point data[maxn];

double ans=0;

for(int i=0; i<pn; i++)

{

o=p[i];

int dn=0;

for(int j=0; j<pn; j++)

{

if(p[j].y>o.y||(p[j].y==o.y&&p[j].x>=o.x))

{

data[dn++]=p[j];

}

}

sort(data, data+dn, cmp\_angle);

ans=max(ans, empty\_convex(data, dn));

}

return ans;

}

int main()

{

#ifndef ONLINE\_JUDGE

freopen("in.txt","r",stdin);

#endif // ONLINE\_JUDGE

point p[110];

int t;

scanf("%d",&t);

while(t--)

{

int pn;

scanf("%d",&pn);

for(int i=0; i<pn; i++)

{

scanf("%lf%lf",&p[i].x,&p[i].y);

}

printf("%.1f\n",largest\_empty\_convex(p,pn));

}

return 0;

}