目录

[Data Structure 4](#_Toc304021426)

[Big Integer 4](#_Toc304021427)

[Suffix Index Tree 12](#_Toc304021428)

[String Hash 13](#_Toc304021429)

[Hash Map 14](#_Toc304021430)

[Leftist Tree 15](#_Toc304021431)

[Partition Tree 19](#_Toc304021432)

[Size Balanced Tree 22](#_Toc304021433)

[Splay 25](#_Toc304021434)

[Dynamic Tree 29](#_Toc304021435)

[Graph 33](#_Toc304021436)

[Binary Match 33](#_Toc304021437)

[cut bridge 34](#_Toc304021438)

[Dijkstra 36](#_Toc304021439)

[Dinic 38](#_Toc304021440)

[KM 41](#_Toc304021441)

[Normal Match 42](#_Toc304021442)

[Sap 44](#_Toc304021443)

[Sw 46](#_Toc304021444)

[TreeMST 47](#_Toc304021445)

[Pick 49](#_Toc304021446)

[Search 49](#_Toc304021447)

[Dancing Links 49](#_Toc304021448)

[Dancing Links Mix 53](#_Toc304021449)

[Maximum Clique 57](#_Toc304021450)

[Geometry 59](#_Toc304021451)

[Util 59](#_Toc304021452)

[Lines 63](#_Toc304021453)

[distance from point to line 63](#_Toc304021454)

[point on line 63](#_Toc304021455)

[Relation about point and line 63](#_Toc304021456)

[points in the same side of a line 63](#_Toc304021457)

[point on line 64](#_Toc304021458)

[Symmetry point from a line 64](#_Toc304021459)

[line intersect 64](#_Toc304021460)

[segment intersect 64](#_Toc304021461)

[intersection point 65](#_Toc304021462)

[nearest point to line 65](#_Toc304021463)

[nearest point to segment 65](#_Toc304021464)

[minimum distance between two segments 66](#_Toc304021465)

[inclination between two lines, range [0,PI] 66](#_Toc304021466)

[Horizontal line intersects segment 66](#_Toc304021467)

[Area 67](#_Toc304021468)

[Triangle 67](#_Toc304021469)

[Circle 67](#_Toc304021470)

[Polygon 67](#_Toc304021471)

[Triangle 68](#_Toc304021472)

[Coefficient 68](#_Toc304021473)

[Center 68](#_Toc304021474)

[Polygon 68](#_Toc304021475)

[Check anti-clockwise 68](#_Toc304021476)

[Check Convex 69](#_Toc304021477)

[Point inside polygon 69](#_Toc304021478)

[Segment inside polygon 70](#_Toc304021479)

[Convex Hull 71](#_Toc304021480)

[Gravity center of polygon 72](#_Toc304021481)

[Circles 72](#_Toc304021482)

[Common area between two circles 72](#_Toc304021483)

[Circle intersection 73](#_Toc304021484)

[Segment intersects circle 74](#_Toc304021485)

[Common area of circles and polygon 75](#_Toc304021486)

[Minimum Circle 75](#_Toc304021487)

[Ball Center 76](#_Toc304021488)

[Cricles’ union area 77](#_Toc304021489)

[Cut 78](#_Toc304021490)

[Nearest point pair 83](#_Toc304021491)

[3D Convex Hull 84](#_Toc304021492)

[Sweep line 88](#_Toc304021493)

[Math 92](#_Toc304021494)

[Gauss 92](#_Toc304021495)

[Math relative 93](#_Toc304021496)

[RMQ 95](#_Toc304021497)

[LRS 96](#_Toc304021498)

[Gcd 97](#_Toc304021499)

[linear equations 97](#_Toc304021500)

[simple continued fraction expansion 98](#_Toc304021501)

[Number theory 99](#_Toc304021502)

[Miller rabin 99](#_Toc304021503)

[Other 102](#_Toc304021504)

[Head files 102](#_Toc304021505)

[Java 104](#_Toc304021506)

[String 105](#_Toc304021507)

[KMP 105](#_Toc304021508)

[Extended KMP 106](#_Toc304021509)

[Palindrome 107](#_Toc304021510)

[Cyclic String minimum expression 107](#_Toc304021511)

[AC Machine 108](#_Toc304021512)

[Suffix array da 110](#_Toc304021513)

[Suffix array dc3 113](#_Toc304021514)

# Data Structure

## Big Integer

#include <cstdio>

#include <iostream>

#include <cstring>

#include <cstdlib>

#include <algorithm>

using namespace std;

const int DIGIT\_COUNT = 1000;

class BigInt

{

private:

int len;

int digit[1200];

bool flag;

void fix();

public:

BigInt();

BigInt(const BigInt&);

BigInt(const int&);

BigInt(const char\*);

virtual ~BigInt();

bool operator<(const BigInt&) const;

bool operator<=(const BigInt&) const;

bool operator>(const BigInt&) const;

bool operator>=(const BigInt&) const;

bool operator==(const BigInt&) const;

bool operator!=(const BigInt&) const;

BigInt& operator=(const BigInt&);

friend BigInt operator+(const BigInt&, const BigInt&);

friend BigInt operator-(const BigInt&);

friend BigInt operator-(const BigInt&, const BigInt&);

friend BigInt operator\*(const BigInt&, const int&);

friend BigInt operator\*(const int&, const BigInt&);

friend BigInt operator\*(const BigInt&, const BigInt&);

friend BigInt operator/(const BigInt&, const int&);

friend int operator%(const BigInt&, const int&);

void show();

};

BigInt::BigInt()

{

memset(digit, 0, sizeof(digit));

len = 1;

flag = true;

}

BigInt::BigInt(const char\* s)

{

int slen = strlen(s);

int i = slen - 1, ed = 0;

memset(digit, 0, sizeof(digit));

len = 0;

if (s[ed] == '-')

{

ed++;

flag = false;

}

else

{

flag = true;

}

while (i >= ed)

{

digit[len] = 0;

int tp = 1;

for (int j = 0; tp < DIGIT\_COUNT && i >= ed; j++, i--)

{

digit[len] = digit[len] + (s[i] - '0') \* tp;

tp \*= 10;

}

len++;

}

}

BigInt::BigInt(const int& n)

{

memset(digit, 0, sizeof(digit));

if (n < 0)

{

flag = false;

digit[0] = -n;

}

else

{

flag = true;

digit[0] = n;

}

len = 1;

fix();

}

BigInt::BigInt(const BigInt& big)

{

memset(digit, 0, sizeof(digit));

len = big.len;

flag = big.flag;

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

{

digit[i] = big.digit[i];

}

}

BigInt::~BigInt()

{

}

void BigInt::fix()

{

while (len > 1 && !digit[len - 1])

{

len--;

}

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

{

digit[i + 1] += digit[i] / DIGIT\_COUNT;

digit[i] %= DIGIT\_COUNT;

}

while (digit[len])

{

digit[len + 1] = digit[len] / DIGIT\_COUNT;

digit[len] %= DIGIT\_COUNT;

len++;

}

}

bool BigInt::operator<(const BigInt& big) const

{

if (flag != big.flag) return flag < big.flag;

if (flag)

{

if (len != big.len)

{

return len < big.len;

}

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

{

if (digit[i] != big.digit[i])

{

return digit[i] < big.digit[i];

}

}

}

else

{

if (len != big.len)

{

return len > big.len;

}

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

{

if (digit[i] != big.digit[i])

{

return digit[i] > big.digit[i];

}

}

}

return false;

}

bool BigInt::operator<=(const BigInt& big) const

{

return \*this < big || \*this == big;

}

bool BigInt::operator>(const BigInt& big) const

{

return !(\*this <= big);

}

bool BigInt::operator>=(const BigInt& big) const

{

return !(\*this < big);

}

bool BigInt::operator==(const BigInt& big) const

{

if (flag != big.flag || len != big.len) return false;

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

if (digit[i] != big.digit[i]) return false;

return true;

}

bool BigInt::operator!=(const BigInt& big) const

{

return !(\*this == big);

}

BigInt& BigInt::operator=(const BigInt& big)

{

memset(digit, 0, sizeof(digit));

len = big.len;

flag = big.flag;

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

{

digit[i] = big.digit[i];

}

return \*this;

}

BigInt operator+(const BigInt& a, const BigInt& b)

{

BigInt ret;

if (a.flag != b.flag)

{

if (a.flag) return a - b;

else return b - a;

}

ret.len = max(a.len, b.len);

ret.flag = a.flag;

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

{

if (i < a.len) ret.digit[i] += a.digit[i];

if (i < b.len) ret.digit[i] += b.digit[i];

}

ret.fix();

return ret;

}

BigInt operator-(const BigInt& big)

{

BigInt ret = big;

ret.flag ^= 1;

return ret;

}

BigInt operator-(const BigInt& a, const BigInt& b)

{

BigInt c = a;

BigInt d = b;

BigInt ret;

if (c < 0)

{

c = -c;

}

if (d < 0)

{

d = -d;

}

if (a.flag != b.flag)

{

if (a.flag)

{

return c + d;

}

else

{

return -(c + d);

}

}

else

{

if (a.flag)

{

if (c >= d)

{

ret = a;

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

{

while (ret.digit[i] < b.digit[i])

{

ret.digit[i] += DIGIT\_COUNT;

ret.digit[i + 1]--;

}

ret.digit[i] -= b.digit[i];

}

for (int i = b.len; i < ret.len; i++)

{

while (ret.digit[i] < 0)

{

ret.digit[i] += DIGIT\_COUNT;

ret.digit[i + 1]--;

}

}

ret.fix();

return ret;

}

else

{

return -(b - a);

}

}

else

{

return d - c;

}

}

return ret;

}

BigInt operator\*(const BigInt& a, const int& \_b)

{

int b = \_b;

if (b == 0)

{

return BigInt(0);

}

BigInt ret = a;

//ret.show();

if (b < 0)

{

ret.flag ^= 1;

b = -b;

}

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

{

ret.digit[i] \*= b;

}

ret.fix();

return ret;

}

BigInt operator\*(const int& a, const BigInt& b)

{

return b \* a;

}

BigInt operator\*(const BigInt& a, const BigInt& b)

{

BigInt ret;

ret.len = a.len + b.len - 1;

ret.flag = (a.flag == b.flag);

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

{

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

{

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

}

}

ret.fix();

return ret;

}

BigInt operator/(const BigInt& a, const int& b)

{

//if b\*b>2^31 set the digit to LL

BigInt ret = a;

if (b < 0)

{

b = -b;

ret.flag ^= 1;

}

for (int i = ret.len - 1; i > 0; i--)

{

ret.digit[i - 1] += ret.digit[i] % b \* DIGIT\_COUNT;

ret.digit[i] /= b;

}

ret.digit[0] /= b;

ret.fix();

return ret;

}

int operator%(const BigInt& a, const int& b)

{

//if b\*b>2^31 set the digit to LL

BigInt ret = a;

for (int i = ret.len - 1; i > 0; i--)

{

ret.digit[i - 1] += ret.digit[i] % b \* DIGIT\_COUNT;

ret.digit[i] /= b;

}

return ret.digit[0] % b;

}

void BigInt::show()

{

if (!flag) printf("-");

printf("%d", digit[len - 1]);

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

{

printf("%03d", digit[i]);

}

printf("\n");

}

## Suffix Index Tree

int find\_kth(int k)

{

int id,i;

id = 0;

for(i=20;i>=0;i--)

{

id |= 1<<i;

if( id<mx && tr[id]<k )

k -= tr[id];

else

id ^= 1<<i;

}

return id+1;

}



init(tr[0], a, c);

init(tr[0], b,-c);

init(tr[1], a, c\*a);

init(tr[1], b,-c\*b);

return sum[x] + (x+1)\*find(tr[0],x) - find(tr[1],x);

## String Hash

#include <cstdio>

#include <cstring>

#include <cstdlib>

using namespace std;

const int MAX = 100000;

const int mod = 100007;

struct hbox

{

char st[20];

hbox\* s;

} hash[MAX], \*h[mod], \*cur, \*pt[MAX];

int cnt[mod];

unsigned int BKDHash(char\* str)

{

unsigned int seed = 131;

unsigned int hash = 0;

while (\*str)

hash = hash \* seed + (\*str++);

return (hash & 0x7FFFFFFF) % mod;

}

void init()

{

cur = hash;

memset(h, 0, sizeof(h));

memset(pt, 0, sizeof(pt));

}

int getId(char\* str)

{

int d = BKDHash(str);

hbox\* p = h[d];

while (p)

{

if (strcmp(str, p->st) == 0) return p - hash;

p = p->s;

}

strcpy(cur->st, str);

cur->s = h[d];

h[d] = cur++;

pt[cur - hash - 1] = h[d];

return cur - hash - 1;

}

## Hash Map

#include <cstdio>

#include <utility>

#include <ext/hash\_map>

using namespace std;

using namespace \_\_gnu\_cxx;

namespace \_\_gnu\_cxx

{

struct Equal

{

bool operator()(const int& a, const int& b) const

{

return a == b;

}

};

struct Hash

{

size\_t operator()(const int& a) const

{

return a % 100007;

}

};

}

hash\_map<int, int, Hash, Equal> mp;

hash\_map<int, int, Hash, Equal>::const\_iterator it;

/\*

it=mp.find(x);

it!=mp.end();

...

\*/

## Leftist Tree

#include <cstdio>

#include <cstring>

#include <cstdlib>

#include <algorithm>

using namespace std;

const int MAX = 10000;

struct Node

{

Node\* left;

Node\* right;

Node\* parent;

int dist;

size\_t val;

} node[MAX], \*root[MAX];

int st[MAX], top, K;

void init(Node\*& root)

{

K = 0;

top = 0;

root = NULL;

}

Node\* New(size\_t x)

{

Node\* ret;

if (top)

{

ret = &node[st[--top]];

if (ret->left != NULL)

{

st[top++] = ret->left - node;

}

if (ret->right != NULL)

{

st[top++] = ret->right - node;

}

}

else

{

ret = &node[K++];

}

ret->left = ret->right = NULL;

ret->parent = NULL;

ret->dist = 0;

ret->val = x;

return ret;

}

void Delete(Node\* x)

{

st[top++] = x - node;

}

Node\* Find(Node\* x)

{

//the same as Union\_Find Set

if (x->parent != NULL)

{

x->parent = Find(x->parent);

}

return x->parent;

}

Node\* merge(Node\* x, Node\* y)

{

if (x == NULL)

{

return y;

}

if (y == NULL)

{

return x;

}

if (x->val > y->val) //min\_heap

{

swap(x, y);

}

x->right = merge(x->right, y);

x->right->parent = x;

if (x->left == NULL || x->right != NULL && x->left->dist < x->right->dist)

{

swap(x->left, x->right);

}

if (x->right == NULL)

{

x->dist = 0;

}

else

{

x->dist = x->right->dist + 1;

}

return x;

}

Node\* insert(Node\*& root, size\_t v)

{

root = merge(root, New(v));

return root;

}

Node\* insert(Node\*& root, Node\*& v)

{

root = merge(root, v);

return root;

}

size\_t min(Node\* root)

{

return root->val;

}

Node\* pop(Node\*& root)

{

Node\* l = root->left;

Node\* r = root->right;

if (l != NULL)

{

l->parent = NULL;

}

if (r != NULL)

{

r->parent = NULL;

}

root->left = NULL;

root->right = NULL;

Delete(root);

root = merge(l, r);

return root;

}

void Del(Node\* x)

{

Node\* q = x->parent;

Node\* p = merge(x->left, x->right);

p->parent = q;

if (q != NULL && q->left == x)

{

q->left = p;

}

else if (q != NULL && q->right == x)

{

q->right = p;

}

while (q != NULL)

{

if (q->left->dist < q->right->dist)

{

swap(q->left, q->right);

}

if (q->left->dist == q->right->dist + 1)

{

return;

}

q->dist++;

p = q;

q = q->parent;

}

}

int main()

{

char s[100];

int id, x;

for (int i = 0; i < 10; i++)

{

init(root[i]);

}

while (~scanf("%s%d%d", s, &id, &x))

{

if (s[0] == 'A')

{

insert(root[id], x);

}

else if (s[0] == 'M')

{

printf("%d\n", min(root[id]));

pop(root[id]);

}

else if (s[0] == 'U')

{

insert(root[id], root[x]);

}

}

return 0;

}

## Partition Tree

#include <cstdio>

#include <cstdlib>

#include <cstring>

#include <algorithm>

using namespace std;

const int MAX = 100010;

struct Node

{

int l, r;

} seg[4 \* MAX];

int sorted[MAX], tr[20][MAX], cot[20][MAX];

void init(int k, int d, int l, int r)

{

seg[k].l = l;

seg[k].r = r;

if (l == r)

{

return;

}

int mid = (seg[k].l + seg[k].r) >> 1;

int lsame = mid - l + 1;

for (int i = l; i <= r; i++)

{

if (tr[d][i] < sorted[mid])

{

lsame--;

}

}

int lp = l;

int rp = mid + 1;

int cnt = 0;

for (int i = l; i <= r; i++)

{

if (i == l)

{

cot[d][i] = 0;

}

else

{

cot[d][i] = cot[d][i - 1];

}

if (tr[d][i] < sorted[mid])

{

cot[d][i]++;

tr[d + 1][lp++] = tr[d][i];

}

else if (tr[d][i] > sorted[mid])

{

tr[d + 1][rp++] = tr[d][i];

}

else

{

if (cnt < lsame)

{

cnt++;

cot[d][i]++;

tr[d + 1][lp++] = tr[d][i];

}

else

{

tr[d + 1][rp++] = tr[d][i];

}

}

}

init(k << 1, d + 1, l, mid);

init(k << 1 | 1, d + 1, mid + 1, r);

}

int read(int k, int d, int l, int r, int rank)

{

if (l == r)

{

return tr[d][l];

}

int nl, nr;

if (l == seg[k].l)

{

nl = 0;

nr = cot[d][r];

}

else

{

nl = cot[d][l - 1];

nr = cot[d][r] - cot[d][l - 1];

}

if (nr >= rank)

{

return read(k << 1, d + 1, seg[k].l + nl, seg[k].l + nl + nr - 1, rank);

}

else

{

int mid = (seg[k].l + seg[k].r) >> 1;

int newl = l - seg[k].l - nl;

int newr = r - l - nr + 1;

return read(k << 1 | 1, d + 1, mid + newl + 1, mid + newl + newr, rank - nr);

}

}

int main()

{

int t, n, m;

int x, y, k;

scanf("%d", &t);

while (t--)

{

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

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

{

scanf("%d", &tr[0][i]);

sorted[i] = tr[0][i];

}

sort(sorted, sorted + n);

init(1, 0, 0, n - 1);

while (m--)

{

scanf("%d%d%d", &x, &y, &k);

printf("%d\n", read(1, 0, x - 1, y - 1, k));

}

}

return 0;

}

## Size Balanced Tree

#include <cstdio>

#include <cstring>

#include <cmath>

#include <cstdlib>

#include <algorithm>

using namespace std;

typedef int T;

const int MAX = 500050;

struct Node

{

T key;

int size;

Node\* c[2];

} memo[MAX], \*cur, \*nil, \*root;

Node\* New(T v)

{

cur->key = v, cur->size = 1;

cur->c[0] = cur->c[1] = nil;

return cur++;

}

struct Sbt

{

void init()

{

nil = cur = memo;

root = nil = New(-1);

nil->size = 0;

}

void rotate(Node\*& t, int f)

{

Node\* k = t->c[f ^ 1];

t->c[f ^ 1] = k->c[f], k->c[f] = t;

k->size = t->size, t->size = t->c[0]->size + t->c[1]->size + 1;

t = k;

}

void keep(Node\*& t, int f)

{

if (t == nil) return; //TLE

else if (t->c[f]->c[f]->size > t->c[f ^ 1]->size) rotate(t, f ^ 1);

else if (t->c[f]->c[f ^ 1]->size > t->c[f ^ 1]->size) rotate(t->c[f], f), rotate(t, f ^ 1);

else return;

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

keep(t->c[i], i);

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

keep(t, i);

}

void insert(Node\*& t, T v)

{

if (t == nil) t = New(v);

else t->size++, insert(t->c[v >= t->key], v), keep(t, v >= t->key);

}

Node\* del(Node\*& t, T v)

{

Node\* p;

if (t == nil) return nil;

t->size--;

if (v == t->key || t->c[v > t->key] == nil)

{

if (t->c[0] != nil && t->c[1] != nil) p = del(t->c[0], v + 1), t->key = p->key;

else p = t, t = t->c[t->c[0] == nil];

return p;

}

else return del(t->c[v > t->key], v);

}

Node\* getMinMax(Node\* t, int f)

{

while (t->c[f] != nil)

t = t->c[f];

return t;

}

Node\* pre(Node\* t, T v)

{

Node\* ret = nil;

while (t != nil)

if (t->key >= v) t = t->c[0];

else ret = t, t = t->c[1];

return ret;

}

Node\* suc(Node\* t, T v)

{

Node\* ret = nil;

while (t != nil)

if (t->key <= v) t = t->c[1];

else ret = t, t = t->c[0];

return ret;

}

Node\* select(Node\* t, int k)

{

if (k <= 0 || k > t->size) return nil;

int tmp;

while ((tmp = t->c[0]->size + 1) != k)

if (tmp < k) k -= tmp, t = t->c[1];

else t = t->c[0];

return t;

}

int getRank(Node\* t, T v)

{

int ret = 0;

while (t != nil)

if (t->key < v) ret += t->c[0]->size + 1, t = t->c[1];

else t = t->c[0];

return ret;

}

int count(Node\* t, T v)

{

return getRank(t, v + 1) - getRank(t, v);

}

bool find(Node\* t, T v)

{

while (t != nil)

if (t->key == v) return true;

else t = t->c[v > t->key];

return false;

}

} tr;

## Splay

#include <cstdio>

#include <cstring>

#include <cmath>

#include <cstdlib>

#include <algorithm>

using namespace std;

typedef int T;

const int MAX = 12000;

const int oo = 0x3f3f3f3f;

struct Node

{

T key;

int size;

bool rev, same;

Node \*c[2], \*p;

} node[MAX], \*q[MAX], \*st[MAX], \*nil, \*root;

int top1, top2;

Node\* New(T v)

{

Node\* p;

if (top2) p = st[--top2];

else p = &node[top1++];

p->key = v, p->size = 1, p->rev = p->same = false;

p->c[0] = p->c[1] = p->p = nil;

return p;

}

int num[MAX];

struct Splay

{

void init()

{

top1 = top2 = 0;

nil = node;

nil = New(-oo), nil->size = 0;

root = New(-oo), root->c[1] = New(-oo);

root->c[1]->p = root, update(root);

}

void rotate(Node\* x, int f)

{

Node\* y = x->p;

pushdown(y), pushdown(x);

y->c[f ^ 1] = x->c[f], x->p = y->p;

if (x->c[f] != nil) x->c[f]->p = y;

if (y->p != nil) y->p->c[y->p->c[1] == y] = x;

x->c[f] = y, y->p = x;

update(y);

}

void splay(Node\* x, Node\* f)

{

pushdown(x);

while (x->p != f)

{

if (x->p->p == f) rotate(x, x->p->c[0] == x);

else

{

Node\* y = x->p;

int t = (y->p->c[0] == y);

if (y->c[t] == x) rotate(x, t ^ 1), rotate(x, t);

else rotate(y, t), rotate(x, t);

}

}

update(x);

if (f == nil) root = x;

}

void reverse(Node\* x)

{

if (x == nil) return;

swap(x->c[0], x->c[1]), x->rev ^= 1;

}

void make\_same(Node\* x, T v)

{

if (x == nil) return;

x->same = true, x->rev = false, x->key = v;

}

void select(int k, Node\* f)

{

Node\* x = root;

int tmp;

pushdown(x);

while ((tmp = x->c[0]->size) != k)

{

if (k < tmp) x = x->c[0];

else x = x->c[1], k -= tmp + 1;

pushdown(x);

}

splay(x, f);

}

void clear(Node\* x)

{

int f = 0, b = 0;

if (x == nil) return;

pushdown(x), q[b++] = x;

while (f != b)

{

st[top2++] = q[f];

if (q[f]->c[0] != nil) q[b++] = q[f]->c[0];

if (q[f]->c[1] != nil) q[b++] = q[f]->c[1];

f++;

}

}

Node\* pre(int v)

{

Node\* t = root;

Node\* ret = nil;

while (t != nil)

if (t->key >= v) t = t->c[0];

else ret = t, t = t->c[1];

return ret;

}

Node\* suc(int v)

{

Node\* t = root;

Node\* ret = nil;

while (t != nil)

if (t->key <= v) t = t->c[1];

else ret = t, t = t->c[0];

return ret;

}

void insert(int pos, int n)

{

select(pos, nil);

select(pos + 1, root);

root->c[1]->c[0] = make\_tree(0, n - 1, root->c[1]);

splay(root->c[1]->c[0], nil);

}

void pushdown(Node\* x)

{

if (x == nil) return;

if (x->same) make\_same(x->c[0], x->key), make\_same(x->c[1], x->key), x->same = x->rev = false;

else if (x->rev) reverse(x->c[0]), reverse(x->c[1]), x->rev = false;

}

void update(Node\* x)

{

if (x == nil) return;

x->size = x->c[0]->size + x->c[1]->size + 1;

}

Node\* make\_tree(int l, int r, Node\* f)

{

if (l > r) return nil;

int mid = (l + r) >> 1;

Node\* p = New(num[mid]);

p->c[0] = make\_tree(l, mid - 1, p);

p->c[1] = make\_tree(mid + 1, r, p);

p->p = f;

update(p);

return p;

}

Node\* getMaxMin(Node\* x, int f)

{

while (x->c[f] != nil)

x = x->c[f];

return x;

}

Node\* joint(Node\* x, Node\* y)

{

if (x == y) return x;

if (x == nil) return y;

if (y == nil) return x;

splay(x = getMaxMin(x, 1), nil);

splay(y = getMaxMin(y, 0), nil);

if (x->p != nil) return y; //in the same tree

x->c[1] = y, y->p = x;

splay(y, nil);

}

Node\* split(Node\* t)

{

if (t == nil) return nil;

splay(t, nil);

t->c[0]->p = t->c[1]->p = nil;

Node\* ret = joint(t->c[0], t->c[1]);

t->c[0] = t->c[1] = nil;

clear(t);

return ret;

}

} tr;

## Dynamic Tree

#include <cstdio>

#include <cstring>

#include <cstdlib>

#include <algorithm>

using namespace std;

const int MAX = 30005;

const int oo = 0x3f3f3f3f;

int n;

struct Node

{

Node \*c[2], \*p;

int key, s;

bool rev;

} memo[MAX], \*nil, \*pt[MAX], \*st[MAX];

struct Splay

{

void init(Node\* x)

{

x->c[0] = x->c[1] = x->p = nil;

x->key = x->s = 0;

x->rev = false;

}

void init()

{

pt[0] = nil = &memo[0];

// nil->size=0;

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

pt[i] = &memo[i], init(pt[i]);

}

void rotate(Node\* x, int f)

{

Node \*y = x->p, \*z = y->p;

x->p = z, y->p = x;

if (z != nil) //can not conbine

{

if (y == z->c[0]) z->c[0] = x;

else if (y == z->c[1]) z->c[1] = x;

}

y->c[f ^ 1] = x->c[f];

if (y->c[f ^ 1] != nil) y->c[f ^ 1]->p = y;

x->c[f] = y;

update(y);

update(x);

}

void splay(Node \*x)

{

int top = 1;

st[0] = x;

for (Node \*q = x; !isroot(q);)

st[top++] = (q = q->p);

while (top)

pushdown(st[--top]);

while (!isroot(x))

{

Node \*y = x->p;

if (isroot(y)) rotate(x, y->c[0] == x);

else

{

int t = (y == y->p->c[0]);

if (x == y->c[t]) rotate(x, t ^ 1), rotate(x, t);

else rotate(y, t), rotate(x, t);

}

}

}

void pushdown(Node\* x)

{

if (x == nil) return;

if (x->rev)

{

reverse(x->c[0]);

reverse(x->c[1]);

x->rev = false;

}

}

void reverse(Node\* x)

{

if (x != nil)

{

swap(x->c[0], x->c[1]);

x->rev ^= 1;

}

}

void update(Node\* x)

{

if (x == nil) return;

x->s = x->c[0]->s + x->key + x->c[1]->s;

}

bool isroot(Node\* x)

{

return x->p == nil || (x->p->c[0] != x && x->p->c[1] != x);

}

Node\* expose(Node\* x)

{

// return the root

Node\* y;

for (y = nil; x != nil; x = x->p)

splay(x), x->c[1] = y, update(y = x);

return y;

}

void set(Node\* x, int v)

{

x->key = v, splay(x);

}

Node\* getRoot(Node\* x)

{

return head(expose(x));

}

Node\* head(Node\* x)

{

if (x == nil) return nil;

while (x->c[0] != nil)

pushdown(x), x = x->c[0];

splay(x);

return x;

}

void getPath(Node\* x, Node\* y)

{

//path: v => u => u->c[1]

//to calculate edge's weight, u can not be included

Node\* ry = head(expose(y)), \*rx = head(expose(x));

if (rx != ry) puts("impossible");

else

{

for (Node\* u = y, \*v = nil; u != nil; u = u->p)

{

splay(u);

if (u->p == nil)

{

printf("%d\n", v->s + u->key + u->c[1]->s);

return;

}

u->c[1] = v, update(v = u);

}

}

}

void setRoot(Node\* x)

{

reverse(expose(x));

}

bool merge(Node\* x, Node\* y)

{

// y is x's father in a rooted tree

Node\* ry = head(expose(y)), \*rx = head(expose(x));

if (rx == ry) return false;

else

{

setRoot(x);

splay(x), x->p = y;

return true;

}

}

void cut(Node\* x)

{

splay(x);

if (x->c[0] != nil) x->c[0]->p = x->p, x->p = x->c[0] = nil;

else x->p = nil;

}

Node\* LCA(Node\* x, Node\* y)

{

Node \*rx = head(expose(x));

Node \*ey = expose(y), \*ry = head(ey);

if (rx == ry) return ey;

else return nil;

}

} tr;

# Graph

## Binary Match

#include <cstdio>

#include <cstring>

#include <cstdio>

#include <cstdlib>

using namespace std;

const int MAX = 200;

int n1, n2, m;

int pre[MAX];

bool visit[MAX];

bool conn[MAX][MAX];

void init()

{

memset(conn, 0, sizeof(conn));

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

}

bool dfs(int a)

{

for (int i = 0; i < n2; i++)

{

if (conn[a][i] == 1 && !visit[i])

{

visit[i] = true;

if (pre[i] == -1 || dfs(pre[i]))

{

pre[i] = a;

return true;

}

}

}

return false;

}

int BinaryMatch()

{

int ans = 0;

for (int i = 0; i < n1; i++)

{

memset(visit, 0, sizeof(visit));

if (dfs(i)) ans++;

}

return ans;

}

## cut bridge

#include<cstdio>

#include<cstring>

#include<cmath>

#include<algorithm>

#include<cstdlib>

using namespace std;

struct Edge{int num,ne;}e[1000002];

struct Bridge{int x,y;}br[1000002];

int vis[10002],K,bn,Root,D[10002],An[10002],p[10002];

bool isc[10002];

void dfs(int x,int fa,int deep)

{

vis[x]=1;D[x]=deep;

An[x]=deep;int Tot=0;

for(int i=p[x];i!=-1;i=e[i].ne)

{

if(e[i].num!=fa&&vis[e[i].num]==1)

An[x]=min(An[x],D[e[i].num]);

if(vis[e[i].num]==0)

{

dfs(e[i].num,x,deep+1);

Tot++;An[x]=min(An[x],An[e[i].num]);

if((x==Root&&Tot>1)||(x!=Root&&An[e[i].num]>=D[x]))

isc[x]=1;

if(An[e[i].num]>D[x])

{br[bn].x=x;br[bn].y=e[i].num;bn++;}

}

}

vis[x]=2;

}

int i,n,m,x,y;

int main()

{

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

{

for(i=1;i<=n;i++)p[i]=-1;K=0;

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

{

scanf("%d%d",&x,&y);

e[K].num=y;e[K].ne=p[x];p[x]=K++;

e[K].num=x;e[K].ne=p[y];p[y]=K++;

}

for(i=1;i<=n;i++){vis[i]=0;isc[i]=0;}

Root=1;bn=0;dfs(1,-1,1);

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

if(isc[i]==1)printf("%d ",i);printf("\n");

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

printf("%d->%d ",br[i].x,br[i].y);printf("\n");

}

}

## Dijkstra

#include <cstdio>

#include <cstring>

#include <cmath>

#include <cstdlib>

#include <algorithm>

using namespace std;

const int V = 1200;

const int E = 100005;

const int oo = 0x3f3f3f3f;

struct Edge

{

int to, w, id;

Edge\* next;

} memo[E], \*cur, \*g[V], \*pree[V];

int q[V], d[V], pre[V], n, m;

int h[V], pos[V], K;

int s, t;

inline void init()

{

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

g[i] = NULL;

cur = memo;

}

inline void add(int u, int v, int w, int id)

{

cur->to = v;

cur->w = w;

cur->id = id;

cur->next = g[u];

g[u] = cur++;

}

inline void sink(int k)

{

while (k <= K)

{

int idx = k, ls = k << 1, rs = k << 1 | 1;

if (ls <= K && d[h[ls]] < d[h[idx]]) idx = ls;

if (rs <= K && d[h[rs]] < d[h[idx]]) idx = rs;

if (idx == k) break;

else

{

swap(h[idx], h[k]);

pos[h[idx]] = idx, pos[h[k]] = k;

k = idx;

}

}

}

inline int getMin()

{

int ret = h[1];

pos[h[1]] = -1;

swap(h[1], h[K]);

pos[h[1]] = 1;

K--;

sink(1);

return ret;

}

inline void swim(int k)

{

while (k > 1)

{

int p = k >> 1;

if (d[h[k]] < d[h[p]])

{

swap(h[k], h[p]);

pos[h[k]] = k;

pos[h[p]] = p;

k >>= 1;

}

else break;

}

}

inline void add(int k)

{

h[++K] = k;

pos[k] = K;

swim(K);

}

inline int dijkstra(const int& id = -1)

{

int u, v;

K = 0;

fill(d + 1, d + n + 1, oo);

fill(pos + 1, pos + n + 1, -1);

d[s] = 0;

add(s);

while (K)

{

u = getMin();

if (u == t) break;

for (Edge\* it = g[u]; it; it = it->next)

{

v = it->to;

if (it->id == id) continue;

if (d[v] - d[u] > it->w)

{

d[v] = d[u] + it->w;

if (id == -1)

{

pre[v] = u;

pree[v] = it;

}

if (~pos[v]) swim(pos[v]);

else add(v);

}

}

}

return d[t];

}

## Dinic

#include <cstdio>

#include <cstring>

#include <cstdlib>

#include <algorithm>

using namespace std;

const int oo=0x3f3f3f3f;

const int V=500;

const int E=V\*V;

struct Edge{

int num,cap,ne;

}e[E];

struct Vert{

int l,o,e;

}v[V];

int K;

int N,s,t; //number of vertix, source and sink

int q[E]; //queue

void init(){

for(int i=0;i<N;i++){

v[i].e=v[i].l=v[i].o=-1;

}

K=0;

}

void add(int a,int b,int cap){

e[K].num=b;e[K].cap=cap;

e[K].ne=v[a].e;v[a].e=K++;

e[K].num=a;e[K].cap=0;

e[K].ne=v[b].e;v[b].e=K++;

}

int dinic(){

int ret=0;

int f,b; //front pointer and back pointer.

e[K].num=s;

K++;

while(true){

for(int u=0;u<N;u++){

v[u].l=-1;

}

v[s].l=f=0;

v[s].o=v[s].e;

b=0;

q[b++]=s;

while(f<b){

int u=q[f++];

for(int i=v[u].e;i!=-1;i=e[i].ne){

int j=e[i].num;

if(v[j].l==-1&&e[i].cap>0){

v[j].o=v[j].e;

v[j].l=v[u].l+1;

q[b++]=j;

}

}

}

if(v[t].l==-1){

break;

}

b=0;

q[b++]=K-1;

while(b){

int u=e[q[b-1]].num;

if(u==t){

f=oo;

for(int i=1;i<b;i++){

if(e[q[i]].cap<f){

f=e[q[i]].cap;

u=i;

}

}

for(int i=1;i<b;i++){

e[q[i]].cap-=f;

e[q[i]^1].cap+=f;

}

ret+=f;

b=u;

}else{

int i;

for(i=v[u].o;i!=-1;i=e[i].ne){

if(v[e[i].num].l<0||e[i].cap<1){

continue;

}

if(v[e[i].num].l==v[u].l+1){

break;

}

}

if((v[u].o=i)!=-1){

q[b++]=i;

}else{

b--;

v[u].l=-1;

}

}

}

}

return ret;

}

## KM

#include<cstdio>

#include<cstring>

#include<cstring>

#include<cmath>

#include<algorithm>

using namespace std;

const int V=410;

int w[V][V],lx[V],ly[V],mat[V];

bool sx[V],sy[V];

int N,M;

bool path(int u)

{

sx[u]=true;

for(int v=0;v<M;v++)

if(!sy[v]&&lx[u]+ly[v]==w[u][v])

{

sy[v]=true;

if(mat[v]==-1||path(mat[v]))

{mat[v]=u;return true;}

}

return false;

}

const int oo=1000000000;

int KM()

{

int i,j;

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

{

lx[i]=-oo;

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

lx[i]=max(lx[i],w[i][j]);

}

for(i=0;i<M;i++)ly[i]=0;

for(i=0;i<M;i++)mat[i]=-1;

for(int u=0;u<N;u++)

while(1)

{

for(i=0;i<N;i++)sx[i]=0;

for(i=0;i<M;i++)sy[i]=0;

if(path(u))break;

int dx=oo;

for(i=0;i<N;i++)if(sx[i])

for(j=0;j<M;j++)if(!sy[j])

dx=min(dx,lx[i]+ly[j]-w[i][j]);

for(i=0;i<N;i++)if(sx[i])lx[i]-=dx;

for(i=0;i<M;i++)if(sy[i])ly[i]+=dx;

}

int ret=0;

for(i=0;i<N;i++)ret+=lx[i];

for(i=0;i<M;i++)ret+=ly[i];

return ret;

}

## Normal Match

#include <stdio.h>

#include <string.h>

#include <algorithm>

#include <vector>

#define maxn 300

#define maxm 90010

using namespace std;

int match[maxn]; //标记是否匹配

int st[maxn],aim[maxm],nxt[maxm],ln; //边表

int q[maxn]; //bfs队列

int level[maxn]; //离根深度的奇偶性

vector<int> ar[maxn]; //存每个点到根的路径

vector<int> a; //找到的一条增广路

int n;

void init()

{

for(int i=0;i<n;i++)st[i]=-1;ln=0;

}

void in\_edge(int x,int y){

aim[ln]=y;

nxt[ln]=st[x];

st[x]=ln++;

}

int lca(int p,int q){ //求p和q的最近公共祖先

int ret=0;

while (ret<ar[p].size() && ret<ar[q].size() && ar[p][ret]==ar[q][ret]) ret++;

return ret-1;

}

int FindAlterRoad(int sp){

int qn=1;

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

level[q[0]=sp]=1;

ar[sp].clear();

ar[sp].push\_back(sp);

for (int p=0;p<qn;p++){

int x=q[p];

for (int i=st[x];i!=-1;i=nxt[i]){

int u=aim[i];

if (match[u]==u) continue;

if (level[u]==-1){ //u是未访问的点

if (match[u]==-1){ //u是未匹配的,找到增广路

a=ar[x];

a.push\_back(u);

return 1;

}else{ //u是已匹配的点

int v=match[u];

if (level[v]!=-1) continue;

ar[v]=ar[x];

ar[v].push\_back(u);

ar[v].push\_back(v);

level[u]=0;

level[v]=1;

q[qn++]=v;

}

}else

if (level[u]==1){ //u和x同为偶点.形成花

int root=lca(u,x);

vector<int> tmp=ar[x];

for (int i=ar[u].size()-1;i>root;i--){

int y=ar[u][i];

tmp.push\_back(y);

if (level[y]==0){

level[y]=1;

ar[y]=tmp;

level[y]=1;

q[qn++]=y;

}

}

tmp=ar[u];

for (int i=ar[x].size()-1;i>root;i--){

int y=ar[x][i];

tmp.push\_back(y);

if (level[y]==0){

level[y]=1;

ar[y]=tmp;

level[y]=1;

q[qn++]=y;

}

}

}

}

}

return 0;

}

int MaximumMatch(){

int ret=0; //最大匹配数

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

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

if (match[i]==-1)

if (FindAlterRoad(i)){

for (int i=0;i<a.size();i+=2){

int u=a[i],v=a[i+1];

match[u]=v;

match[v]=u;

}

ret++;

}else match[i]=i;

return ret;

}

## Sap

#include<cstdio>

#include<cstring>

#include<cstdlib>

#include<cmath>

#include<algorithm>

using namespace std;

const int V=220;

const int En=200000;

const int oo=0x3f3f3f3f;

struct Edge{int num,ne,c;}e[En];

int d[V],p[V],pre[V],pree[V],low[V];

int gap[V],cur[V];

int N,K,st,ed;

void add(int x,int y,int c)

{

e[K].num=y;e[K].c=c;

e[K].ne=p[x];p[x]=K++;

e[K].num=x;e[K].c=0;

e[K].ne=p[y];p[y]=K++;

}

int sap()

{

int ret=0;

bool fail;

memset(low,0,sizeof(low));

memset(gap,0,sizeof(gap));

memset(d,0,sizeof(d));

for(int i=0;i<N;i++)cur[i]=p[i];

low[st]=oo;gap[0]=N;int u=st;

while(d[st]<N)

{

fail=true;

for(int i=cur[u];i!=-1;i=e[i].ne)

{

int v=e[i].num;cur[u]=i;

if(e[i].c&&d[u]==d[v]+1)

{

pre[v]=u;pree[v]=i;

low[v]=min(low[u],e[i].c);u=v;

if(u==ed)

{

do

{

e[pree[u]].c-=low[ed];

e[pree[u]^1].c+=low[ed];

u=pre[u];

}while(u!=st);

ret+=low[ed];

}

fail=false;break;

}

}

if(fail)

{

gap[d[u]]--;

if(!gap[d[u]])return ret;

d[u]=N;

for(int i=p[u];i!=-1;i=e[i].ne)

if(e[i].c)d[u]=min(d[u],d[e[i].num]+1);

gap[d[u]]++;cur[u]=p[u];

if(u!=st)u=pre[u];

}

}

return ret;

}

## Sw

using namespace std;

#define inf 100000000

bool visit[502],com[502];

int map[502][502],W[502],s,t;

int maxadj(int N,int V)

{

int CUT;

memset(visit,0,sizeof(visit));

memset(W,0,sizeof(W));

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

{

int Num=0,Max=-inf;

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

if(!com[j]&&!visit[j]&&W[j]>Max){Max=W[j];Num=j;}

visit[Num]=true;s=t;t=Num;CUT=W[t];

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

if(!com[j]&&!visit[j])W[j]+=map[Num][j];

}

return CUT;

}

int stoer(int V)

{

int Mincut=inf;int N=V;

memset(com,0,sizeof(com));

for(int i=0;i<V-1;i++)

{

int Cut;s=0,t=0;

Cut=maxadj(N,V);N--;

if(Cut<Mincut)Mincut=Cut;

com[t]=true;

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

if(!com[j])

{map[j][s]+=map[j][t];map[s][j]+=map[t][j];}

}

return Mincut;

}

## TreeMST

#include<cstdio>

#include<cstring>

#include<cstdlib>

#include<cmath>

#include<algorithm>

using namespace std;

const int V=1200;

const int En=2100000;

struct Elf{int u,v,len;}b[En];

const int oo=1000000000;

int ret;

int N,M,Root;//点数，边数，根，默认从0开始

int id[V],pre[V],cnt,vis[V];

int in[V];

bool TreeMST()

{

ret=0;

int i,u,v;

while(1)

{

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

in[i]=oo;

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

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

{

u=b[i].u;

v=b[i].v;

if(b[i].len<in[v]&&u!=v)

{

pre[v]=u;

in[v]=b[i].len;

}

}

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

{

if(i==Root)continue;

if(pre[i]==-1)return false;

}

in[Root]=0;

cnt=0;

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

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

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

{

ret+=in[i];v=i;

while(vis[v]!=i&&id[v]==-1&&v!=Root)

{vis[v]=i;v=pre[v];}

if(v!=Root&&id[v]==-1)

{

for(u=pre[v];u!=v;u=pre[u])

id[u]=cnt;

id[v]=cnt++;

}

}

if(cnt==0)return true;

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

if(id[i]==-1)id[i]=cnt++;

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

{

v=b[i].v;

b[i].u=id[b[i].u];

b[i].v=id[b[i].v];

if(b[i].u!=b[i].v)

b[i].len-=in[v];

}

N=cnt;

Root=id[Root];

}

return true;

}

## Pick

皮克公式:f[n,m]=n-m/2+1

其中,n为多边形的面积,m为多边形边上的点数,f[n,m]为多边形内的点数。

# Search

## Dancing Links

#include <cstdio>

#include <cstring>

#include <cmath>

#include <cstdlib>

#include <algorithm>

using namespace std;

typedef long long ll;

const int MAX = 1200;

const int oo = 0x3f3f3f3f;

const double PI = acos(-1.0);

const double eps = 1e-8;

struct Node

{

Node \*l, \*r, \*d, \*u;

int row, col;

} memo[MAX \* MAX], \*cur, \*hr[MAX], \*hc[MAX];

int cnt[MAX], st[MAX], ans, nC, nR;

/\*

\* Exact cover

\*/

void removeColumn(Node\* c)

{

c->r->l = c->l;

c->l->r = c->r;

for (Node\* i = c->d; i != c; i = i->d)

for (Node\* j = i->r; j != i; j = j->r)

{

j->d->u = j->u;

j->u->d = j->d;

cnt[j->col]--;

}

}

void resumeColumn(Node\* c)

{

for (Node\* i = c->u; i != c; i = i->u)

for (Node\* j = i->l; j != i; j = j->l)

{

j->u->d = j;

j->d->u = j;

cnt[j->col]++;

}

c->r->l = c;

c->l->r = c;

}

bool dfsExactly(const int& k)

{

if (hc[0]->r == hc[0])

{

//TODO deal with the solution

printf("%d", k);

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

printf(" %d", st[i]);

puts("");

return true;

}

int s = oo;

Node\* c = 0;

for (Node\* i = hc[0]->r; i != hc[0]; i = i->r)

if (cnt[i->col] < s) s = cnt[i->col], c = i;

removeColumn(c);

for (Node\* i = c->d; i != c; i = i->d)

{

//TODO push down the stack

st[k] = i->row;

for (Node\* j = i->r; j != i; j = j->r)

removeColumn(hc[j->col]);

if (dfsExactly(k + 1)) return true;

for (Node\* j = i->l; j != i; j = j->l)

resumeColumn(hc[j->col]);

}

resumeColumn(c);

return false;

}

Node\* New(int r, int c)

{

cur->l = cur->r = cur->u = cur->d = cur;

cur->row = r, cur->col = c;

return cur++;

}

void init(int r, int c)

{

nR = r;

nC = c;

cur = memo;

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

hc[i] = New(0, i);

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

hc[i]->r = hc[i + 1];

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

hc[i]->l = hc[i - 1];

hc[0]->l = hc[nC], hc[nC]->r = hc[0];

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

cnt[i] = 0;

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

hr[i] = NULL;

}

void add(const int& r, const int& c)

{

Node\* p = New(r, c);

cnt[c]++;

p->u = hc[c];

p->d = hc[c]->d;

if (!hr[r]) hr[r] = p;

p->l = hr[r], p->r = hr[r]->r;

p->r->l = p->l->r = p->u->d = p->d->u = p;

}

/\*

\* Multiple cover

\*/

void removeNode(Node\* c)

{

for (Node\* i = c->d; i != c; i = i->d)

{

i->r->l = i->l, i->l->r = i->r;

cnt[i->col]--;

}

}

void resumeNode(Node\* c)

{

for (Node\* i = c->u; i != c; i = i->u)

{

i->r->l = i->l->r = i;

cnt[i->col]++;

}

}

int F()

{

bool vis[MAX] = { 0 };

int ret = 0;

while (1)

{

int s = oo;

Node\* c = NULL;

for (Node\* i = hc[0]->r; i != hc[0]; i = i->r)

if (!vis[i->col] && cnt[i->col] < s)

{

s = cnt[i->col];

c = i;

if (s <= 1) break;

}

if (!c) break;

ret++;

vis[c->col] = true;

for (Node\* j = c->d; j != c; j = j->d)

for (Node\* k = j->r; k != j; k = k->r)

vis[k->col] = true;

}

return ret;

}

bool dfsMult(const int& k)

{

if (k + F() > ans) return false;

if (hc[0]->r == hc[0]) return true;

int s = oo;

Node\* c = 0;

for (Node\* i = hc[0]->r; i != hc[0]; i = i->r)

if (cnt[i->col] < s)

{

s = cnt[i->col];

c = i;

if (cnt[i->col] <= 1) break;

}

for (Node\* i = c->d; i != c; i = i->d)

{

removeNode(i);

for (Node\* j = i->r; j != i; j = j->r)

removeNode(j);

if (dfsMult(k + 1)) return true;

for (Node\* j = i->l; j != i; j = j->l)

resumeNode(j);

resumeNode(i);

}

return false;

}

## Dancing Links Mix

#include <cstdio>

#include <cstring>

#include <cmath>

#include <ctime>

#include <cstdlib>

#include <algorithm>

using namespace std;

typedef long long ll;

const int MAX = 80;

const int oo = 0x3f3f3f3f;

struct Node

{

Node \*l, \*r, \*d, \*u;

int row, col;

} memo[MAX \* MAX], \*cur, \*hr[MAX], \*hc[MAX];

int cnt[MAX], st[MAX], nK, ans, nC, nR;

bool mp[MAX][MAX];

int lst[MAX][MAX][2];

Node\* New(int r, int c)

{

cur->l = cur->r = cur->u = cur->d = cur;

cur->row = r, cur->col = c;

return cur++;

}

void init(int r, int c)

{

nR = r;

nC = c;

cur = memo;

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

hc[i] = New(0, i);

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

hc[i]->r = hc[i + 1];

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

hc[i]->l = hc[i - 1];

hc[0]->l = hc[nC], hc[nC]->r = hc[0];

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

cnt[i] = 0;

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

{

hr[i] = NULL;

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

mp[i][j] = false;

}

}

void add(const int& r, const int& c)

{

if (mp[r][c]) return;

mp[r][c] = true;

Node\* p = New(r, c);

cnt[c]++;

p->u = hc[c];

p->d = hc[c]->d;

if (!hr[r]) hr[r] = p;

p->l = hr[r], p->r = hr[r]->r;

p->r->l = p->l->r = p->u->d = p->d->u = p;

}

void remove(Node\* c)

{

if (c->col <= nK)

{

for (Node\* i = c->d; i != c; i = i->d)

{

i->r->l = i->l, i->l->r = i->r;

}

}

else

{

c->l->r = c->r, c->r->l = c->l;

for (Node\* i = c->d; i != c; i = i->d)

for (Node\* j = i->r; j != i; j = j->r)

j->d->u = j->u, j->u->d = j->d, cnt[j->col]--;

}

}

void resume(Node\* c)

{

if (c->col <= nK)

{

for (Node\* i = c->u; i != c; i = i->u)

i->r->l = i->l->r = i;

}

else

{

for (Node\* i = c->u; i != c; i = i->u)

for (Node\* j = i->l; j != i; j = j->l)

j->d->u = j->u->d = j, cnt[j->col]++;

c->l->r = c->r->l = c;

}

}

int F()

{

bool vis[MAX] = { 0 };

int ret = 0;

while (1)

{

int s = oo;

Node\* c = NULL;

for (Node\* i = hc[0]->r; i->col <= nK && i != hc[0]; i = i->r)

if (!vis[i->col] && cnt[i->col] < s)

{

s = cnt[i->col];

c = i;

if (s <= 1) break;

}

if (!c) break;

ret++;

vis[c->col] = true;

for (Node\* j = c->d; j != c; j = j->d)

for (Node\* k = j->r; k != j; k = k->r)

vis[k->col] = true;

}

return ret;

}

bool dfs(const int& k)

{

if (k + F() > ans) return false;

if (hc[0]->r == hc[0] || hc[0]->r->col > nK) return true;

int s = oo;

Node\* c = NULL;

for (Node\* i = hc[0]->r; i != hc[0] && i->col <= nK; i = i->r)

if (cnt[i->col] < s)

{

s = cnt[i->col];

c = i;

if (cnt[i->col] <= 1) break;

}

for (Node\* i = c->d; i != c; i = i->d)

{

remove(i);

for (Node\* j = i->r; j != i; j = j->r)

if (j->col <= nK) remove(j);

for (Node\* j = i->r; j != i; j = j->r)

if (j->col > nK) remove(hc[j->col]);

if (dfs(k + 1)) return true;

for (Node\* j = i->l; j != i; j = j->l)

if (j->col > nK) resume(hc[j->col]);

for (Node\* j = i->l; j != i; j = j->l)

if (j->col <= nK) resume(j);

resume(i);

}

return false;

}

int n, m[MAX];

int doit()

{

ans = 0;

while (!dfs(0))

ans++;

return ans;

}

## Maximum Clique

#include <cstdio>

#include <cstring>

#include <cstdlib>

#include <algorithm>

using namespace std;

const int MAX = 100;

const int oo = 0x3f3f3f3f;

int n;

int dist[MAX][MAX];

bool g[MAX][MAX];

int list[MAX][MAX], degree[MAX], behide[MAX];

int found, curmax;

void sortdegree()

{

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

{

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

if (degree[j] < degree[k]) k = j;

if (k != i)

{

swap(degree[i], degree[k]);

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

swap(g[i][l], g[k][l]);

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

swap(g[l][i], g[l][k]);

}

}

}

void dfs(int d)

{

if (d - 1 > curmax)

{

found = 1;

return;

}

int i, j;

for (i = 1; i < list[d - 1][0] - curmax + d; i++)

if (!found && d + behide[list[d - 1][i] + 1] > curmax && (list[d - 1][0] == i || d + behide[list[d - 1][i + 1]] > curmax))

{

for (j = i + 1, list[d][0] = 0; j <= list[d - 1][0]; j++)

if (g[list[d - 1][j]][list[d - 1][i]]) list[d][++list[d][0]] = list[d - 1][j];

if (list[d][0] == 0 || d + behide[list[d][1]] > curmax) dfs(d + 1);

}

}

void solve()

{

sortdegree();

behide[n + 1] = 0;

behide[n] = 1;

for (int j, i = n - 1; i > 0; i--)

{

curmax = behide[i + 1];

found = list[1][0] = 0;

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

if (g[j][i]) list[1][++list[1][0]] = j;

dfs(2);

behide[i] = curmax + found;

}

}

int check(int v) //表示最长距离不超过v的团的顶点个数

{

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

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

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

if (dist[i][j] >= v) g[i][j] = true;

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

{

degree[i] = 0;

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

degree[i] += g[i][j];

}

solve();

return behide[1];

}

int main()

{

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

{

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

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

scanf("%d", &dist[i][j]);

printf("%d\n", check(1));

}

return 0;

}

# Geometry

## Util

#include <cstdio>

#include <cstring>

#include <cstdlib>

#include <cmath>

#include <vector>

#include <algorithm>

using namespace std;

const int MAX = 1200;

const double oo = 1e100;

const double eps = 1e-8;

const double PI = acos(-1.0);

const int LEFT = 0;

const int RIGHT = 1;

const int ONLINE = 2;

struct Point

{

double x, y;

Point()

{

}

Point(double x, double y) :

x(x), y(y)

{

}

Point operator+(const Point& p) const

{

return Point(x + p.x, y + p.y);

}

Point operator-(const Point& p) const

{

return Point(x - p.x, y - p.y);

}

double operator\*(const Point& p) const

{

return x \* p.y - y \* p.x;

}

double operator&(const Point& p) const

{

return x \* p.x + y \* p.y;

}

bool operator==(const Point& p) const

{

return sgn(x - p.x) == 0 && sgn(y - p.y) == 0;

}

bool operator!=(const Point& p) const

{

return sgn(x - p.x) || sgn(y - p.y);

}

Point operator\*(const double& d) const

{

return Point(x \* d, y \* d);

}

Point operator/(const double& d) const

{

return Point(x / d, y / d);

}

double norm() const

{

return hypot(x, y);

}

Point rotate(double t) const

{

return Point(x \* cos(t) - y \* sin(t), x \* sin(t) + y \* cos(t));

}

bool operator<(const Point& p) const

{

if (dbcmp(y, p.y)) return y < p.y;

else return x < p.x;

}

};

int xmult(const Point& p1, const Point& p2, const Point& p);

struct Line

{

Point a, b;

double k, c; //k = dy/dx c = constant value)

Line()

{

}

Line(const Point& a, const Point& b) :

a(a), b(b)

{

}

bool operator<(const Line& l) const

{

if (dbcmp(k, l.k)) return k < l.k;

else return c < l.c;

}

void set()

{

k = atan2(a.y - b.y, a.x - b.x);

if (dbcmp(a.x, b.x)) c = (a \* b) / fabs(a.x - b.x);

else c = (a \* b) / fabs(a.y - b.y);

}

};

struct Circle

{

Point o;

double r;

Circle()

{

}

Circle(const Point& o, const double& r) :

o(o), r(r)

{

}

};

typedef vector<Point> Points;

typedef vector<Point> Polygon;

int sgn(double x)

{

return x < -eps ? -1 : x > eps;

}

int dbcmp(double x, double y)

{

return sgn(x - y);

}

double sqr(double x)

{

return x \* x;

}

bool is\_int(double x)

{

return sgn(x - floor(x + eps)) == 0;

}

double fix(double x)

{

if (sgn(x)) return x;

else return 0;

}

int xmult(const Point& p1, const Point& p2, const Point& p)

{

return sgn((p1 - p) \* (p2 - p));

}

## Lines

### distance from point to line

double Distance(const Point& p, const Line& l)

{

return fabs((p - l.a) \* (l.b - l.a)) / (l.b - l.a).norm();

}

### point on line

bool on\_line(const Point& p, const Line& l)

{

return sgn((p - l.a) \* (l.b - l.a)) == 0;

}

### Relation about point and line

int relation(const Point& p, const Line& l)

{

double rel = sgn((l.b - l.a) \* (p - l.a));

if (rel == 0) return ONLINE;

else return rel > 0 ? LEFT : RIGHT;

}

### points in the same side of a line

bool sameside(const Point& a, const Point& b, const Line& l)

{

double m1 = (a - l.a) \* (l.b - l.a);

double m2 = (b - l.a) \* (l.b - l.a);

return sgn(m1 \* m2) > 0;

}

### point on line

bool on\_seg(const Point& p, const Line& l)

{

return on\_line(p, l) && sgn((p.x - l.a.x) \* (p.x - l.b.x)) <= 0

&& sgn((p.y - l.a.y) \* (p.y - l.b.y)) <= 0;

}

### Symmetry point from a line

Point symPoint(const Point& p, const Line& l)

{

Point ret;

double a = l.b.x - l.a.x;

double b = l.b.y - l.a.y;

double t = ((p.x - l.a.x) \* a + (p.y - l.a.y) \* b) / (sqr(a) + sqr(b));

ret.x = 2 \* l.a.x + 2 \* a \* t - p.x;

ret.y = 2 \* l.a.y + 2 \* b \* t - p.y;

return ret;

}

### line intersect

bool line\_ins(const Line& l1, const Line& l2)

{

return sgn((l1.a - l1.b) \* (l2.a - l2.b)) != 0;

}

### segment intersect

bool seg\_ins(const Line& l1, const Line& l2)

{

if (on\_seg(l1.a, l2) || on\_seg(l1.b, l2)) return true;

if (on\_seg(l2.a, l1) || on\_seg(l2.b, l1)) return true;

return xmult(l1.b, l2.a, l1.a) \* xmult(l1.b, l2.b, l1.a) < 0

&& xmult(l2.b, l1.a, l2.a) \* xmult(l2.b, l1.b, l2.a) < 0;

}

### intersection point

Point ins\_point(const Line& l1, const Line& l2)

{

double u = (l1.b - l1.a) \* (l2.a - l1.a);

double v = (l1.a - l1.b) \* (l2.b - l1.b);

return (l2.a \* v + l2.b \* u) / (u + v);

}

### nearest point to line

Point pointToLine(const Point& p, const Line& l)

{

double a = l.b.x - l.a.x;

double b = l.b.y - l.a.y;

double t = ((p.x - l.a.x) \* a + (p.y - l.a.y) \* b) / (sqr(a) + sqr(b));

return l.a + Point(a, b) \* t;

}

### nearest point to segment

Point pointToSeg(const Point& p, const Line& l)

{

double a = l.b.x - l.a.x;

double b = l.b.y - l.a.y;

double t = ((p.x - l.a.x) \* a + (p.y - l.a.y) \* b) / (sqr(a) + sqr(b));

if (sgn(t) >= 0 && dbcmp(t, 1) <= 0)

{

return l.a + Point(a, b) \* t;

}

else if ((p - l.a).norm() < (p - l.b).norm())

{

return l.a;

}

else

{

return l.b;

}

}

### minimum distance between two segments

double minDistance(const Line& l1, const Line& l2)

{

double d1, d2, d3, d4;

if (seg\_ins(l1, l2))

{

return 0;

}

else

{

d1 = (pointToSeg(l1.a, l2) - l1.a).norm();

d2 = (pointToSeg(l1.b, l2) - l1.b).norm();

d3 = (pointToSeg(l2.a, l1) - l2.a).norm();

d4 = (pointToSeg(l2.b, l1) - l2.b).norm();

return min(min(d1, d2), min(d3, d4));

}

}

### inclination between two lines, range [0,PI]

double inclination(const Line& l1, const Line& l2)

{

Point u = l1.b - l1.a;

Point v = l2.b - l2.a;

return acos((u & v) / (u.norm() \* v.norm()));

}

### Horizontal line intersects segment

int count\_cross(const Point& p0, const Point& p1, const Point& p2)

{

//判断线段p1-p2与从p0横着向右画出的一条射线的交点个数

//同侧的两种情况

if (dbcmp(p1.y, p0.y) > 0 && dbcmp(p2.y, p0.y) > 0) return 0;

if (dbcmp(p1.y, p0.y) < 0 && dbcmp(p2.y, p0.y) < 0) return 0;

//正好通过

if (dbcmp(p1.y, p0.y) == 0 && dbcmp(p2.y, p0.y) == 0) return 0;

//通过其中一点

if (dbcmp(p1.y, p0.y) == 0) if (dbcmp(p1.x, p0.x) > 0 && dbcmp(p2.y, p0.y) < 0) return 1;

else return 0;

else if (dbcmp(p2.y, p0.y) == 0) if (dbcmp(p2.x, p0.x) > 0 && dbcmp(p1.y, p0.y) < 0) return 1;

else return 0;

//p1和p2同在p0的同侧

if (dbcmp(p1.x, p0.x) <= 0 && dbcmp(p2.x, p0.x) <= 0) return 0;

if (dbcmp(p1.x, p0.x) > 0 && dbcmp(p2.x, p0.x) > 0) return 1;

//需要求交点的情况

if (dbcmp(p2.y, p1.y) > 0) if (dbcmp((p2.x - p1.x) \* (p0.y - p1.y), (p0.x - p1.x) \* (p2.y - p1.y)) > 0) return 1;

else return 0;

else if (dbcmp((p2.x - p1.x) \* (p0.y - p1.y), (p0.x - p1.x) \* (p2.y - p1.y)) < 0) return 1;

else return 0;

}

## Area

### Triangle

double area(const double& a, const double& b, const double& c)

{

double p = (a + b + c) / 2.0;

return sqrt(p \* (p - a) \* (p - b) \* (p - c));

}

double area(const Point& a, const Point& b, const Point& c)

{

return (b - a) \* (c - a); //directed

}

### Circle

double area(const Circle& c)

{

return PI \* sqr(c.r);

}

### Polygon

double area(const Polygon& poly)

{

int n = poly.size();

double ret = 0;

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

{

ret += poly[i] \* poly[(i + 1) % n];

}

return ret / 2.0;

}

## Triangle

### Coefficient

void Coefficient(const Line& L, double & A, double & B, double & C)

{

//返回直线的三个系数

A = L.b.y - L.a.y;

B = L.a.x - L.b.x;

C = L.b.x \* L.a.y - L.a.x \* L.b.y;

}

### Center

void center(const Point \*p)

{

double a1,a2,b1,b2,c1,c2;

a1 = (p[0].x - p[1].x) \* 2;

a2 = (p[1].x - p[2].x) \* 2;

b1 = (p[0].y - p[1].y) \* 2;

b2 = (p[1].y - p[2].y) \* 2;

c1 = sqr(p[0].x) - sqr(p[1].x) + sqr(p[0].y) - sqr(p[1].y);

c2 = sqr(p[1].x) - sqr(p[2].x) + sqr(p[1].y) - sqr(p[2].y);

o.x = (c1\*b2 - c2\*b1) / (a1\*b2 - a2\*b1);

o.y = (a1\*c2 - a2\*c1) / (a1\*b2 - a2\*b1);

r = (o-p[0]).norm();

}

## Polygon

### Check anti-clockwise

bool iscounter(const Polygon& poly)

{

int n = poly.size();

double ret = 0;

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

ret += poly[i] \* poly[(i + 1) % n];

return sgn(ret) > 0;

}

### Check Convex

bool isConvex(const Polygon& poly)

{

// 判断多边形poly是否是凸的

Line l;

int n = poly.size();

if (n < 3) return false;

l.a = poly[0];

l.b = poly[1];

int rel = relation(poly[2], l);

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

{

l.a = poly[i];

l.b = poly[(i + 1) % n];

if (relation(poly[(i + 2) % n], l) != rel) return false;

}

return true;

}

### Point inside polygon

int insidePolygon(const Polygon& poly, Point p)

{

// 判断点p是否在简单多边形poly内, 多边形可以是凸的或凹的

// -1 - inside

// 0 - boarder

// 1 - outside

int rel;

Line ray, side;

int n = poly.size();

rel = 0;

ray.a = p;

ray.b.y = p.y;

ray.b.x = -oo;

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

{

side.a = poly[i];

side.b = poly[(i + 1) % n];

if (on\_seg(p, side))

{

return 0;

}

// 如果side平行x轴则不作考虑

if (dbcmp(side.a.y, side.b.y) == 0)

{

continue;

}

if (on\_seg(side.a, ray))

{

if (dbcmp(side.a.y, side.b.y) > 0) rel++;

}

else if (on\_seg(side.b, ray))

{

if (dbcmp(side.b.y, side.a.y) > 0) rel++;

}

else if (seg\_ins(ray, side))

{

rel++;

}

}

return ((rel % 2 == 1) ? -1 : 1);

}

### Segment inside polygon

bool InsidePolygon(const Polygon& poly, Line L)

{

// 判断线段是否在多边形内 (线段的点可能在多边形上)

bool ret;

Points pt;

Point p;

Line side;

ret = ((insidePolygon(poly, L.a) != 1) && (insidePolygon(poly, L.b) != 1));

if (!ret) return false;

int n = poly.size();

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

{

side.a = poly[i];

side.b = poly[(i + 1) % n];

if (on\_seg(L.a, side)) pt.push\_back(L.a);

else if (on\_seg(L.b, side)) pt.push\_back(L.b);

else if (on\_seg(side.a, L)) pt.push\_back(side.a);

else if (on\_seg(side.b, L)) pt.push\_back(side.b);

else if (seg\_ins(side, L)) return false;

}

// 对交点进行排序

sort(pt.begin(), pt.end());

for (int i = 1; i < (int) pt.size(); i++)

{

if (pt[i - 1] != pt[i])

{

p.x = (pt[i - 1].x + pt[i].x) / 2.0;

p.y = (pt[i - 1].y + pt[i].y) / 2.0;

if (insidePolygon(poly, p) == 1)

{

return false;

}

}

}

return true;

}

### Convex Hull

void graham()

{

int top, n = 50;

Point p[55], st[55];

top = 0;

sort(p, p + n);

st[top++] = p[0];

st[top++] = p[1];

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

{

while (top >= 2 && xmult(p[i], st[top - 1], st[top - 2]) >= 0)

top--;

st[top++] = p[i];

}

int tmp = top;

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

{

while (top != tmp && xmult(p[i], st[top - 1], st[top - 2]) >= 0)

top--;

st[top++] = p[i];

}

top--;

}

### Gravity center of polygon

void center()

{

int i,j,k;

Point a,b,c;

double area,tmp;

area = 0;

a = p[0];

co = Point(0, 0);

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

{

b = p[i-1];

c = p[i];

tmp = (b-a) \* (c-a);

co = co + (a+b+c)\*tmp;

area += tmp;

}

co = co / area / 3.0;

}

## Circles

### Common area between two circles

double CommonArea(const Circle & A, const Circle & B)

{

double s = 0.0;

const Circle & M = (A.r > B.r) ? A : B;

const Circle & N = (A.r > B.r) ? B : A;

double D = (M.o - N.o).norm();

if ((D < M.r + N.r) && (D > M.r - N.r))

{

double cosM = (M.r \* M.r + D \* D - N.r \* N.r) / (2.0 \* M.r \* D);

double cosN = (N.r \* N.r + D \* D - M.r \* M.r) / (2.0 \* N.r \* D);

double alpha = 2.0 \* acos(cosM);

double beta = 2.0 \* acos(cosN);

double TM = 0.5 \* M.r \* M.r \* sin(alpha);

double TN = 0.5 \* N.r \* N.r \* sin(beta);

double FM = (alpha / (2.0 \* PI)) \* area(M);

double FN = (beta / (2.0 \* PI)) \* area(N);

s = FM + FN - TM - TN;

}

else if (D <= M.r - N.r)

{

s = area(N);

}

return s;

}

### Circle intersection

int circle\_intersect(const Circle& A, const Circle& B, Point &p1, Point &p2)

{

//判断圆的交点

//0 包含

//1 相离

//2 内切

//3 外切

//4 相交

if (A.o.x == B.o.x && A.o.y == B.o.y) return 0;

double dd = (A.o - B.o).norm();

if (dbcmp(A.r + B.r, dd) < 0) return 1;

double k, a, b, d, aa, bb, cc, c, del;

k = A.r;

a = B.o.x - A.o.x;

b = B.o.y - A.o.y;

c = B.r;

d = sqr(c) - sqr(k) - sqr(a) - sqr(b);

aa = 4 \* sqr(a) + 4 \* sqr(b);

bb = 4 \* b \* d;

cc = sqr(d) - 4 \* sqr(a) \* sqr(k);

del = sqr(bb) - 4 \* aa \* cc;

if (del < 0) return 0;

del = sqrt(del);

p1.y = (-bb + del) / 2 / aa;

p2.y = (-bb - del) / 2 / aa;

if (sgn(a) == 0)

{

p1.x = sqrt(sqr(k) - sqr(p1.y));

p2.x = -p1.x;

}

else

{

p1.x = (2 \* b \* p1.y + d) / -2 / a;

p2.x = (2 \* b \* p2.y + d) / -2 / a;

}

p1.x += A.o.x;

p1.y += A.o.y;

p2.x += A.o.x;

p2.y += A.o.y;

if (sgn(p1.y - p2.y) == 0)

{

if (sgn(A.r + B.r - dd) == 0) return 3;

if (sgn(dd - (max(A.r, B.r) - min(A.r, B.r))) == 0) return 2;

}

return 4;

}

### Segment intersects circle

bool inCircle(const Point& p, const Circle& c)

{

return dbcmp((p - c.o).norm(), c.r) <= 0;

}

bool SegCir(const Line& l, const Circle& c)

{

// check whether segment intersects circle or not

if (inCircle(l.a, c) || inCircle(l.b, c))

{

return true;

}

if (dbcmp(Distance(c.o, l), c.r) > 0)

{

return false;

}

Point dir = l.b - l.a;

swap(dir.x, dir.y);

dir.x \*= -1;

Line line(c.o, c.o + dir);

return xmult(line.a, l.a, line.b) \* xmult(line.a, l.b, line.b) <= 0;

}

### Common area of circles and polygon

void add(const Point& u, const Point& v, const double& r, Point\* p, int& n) {

//cirlce(0,0) x-=xo...

double a = sqr(v.x - u.x) + sqr(v.y - u.y);

double b = 2 \* ((v.x - u.x) \* u.x + (v.y - u.y) \* u.y);

double c = sqr(u.x) + sqr(u.y) - r \* r;

double d = sqr(b) - 4 \* a \* c;

p[n++] = u;

if (sgn(d) < 0) {

return;

}

d = sqrt(d);

double t1 = (-b + d) / (2 \* a);

double t2 = (-b - d) / (2 \* a);

if (t1 > t2) {

swap(t1, t2);

}

if (sgn(t1) > 0 && dbcmp(t1, 1) < 0) {

p[n++] = u + (v - u) \* t1;

}

if (sgn(t2) > 0 && dbcmp(t2, 1) < 0 && dbcmp(t2, t1)) {

p[n++] = u + (v - u) \* t2;

}

}

double area(const Point& u, const Point& v, const double& r) {

if (dbcmp(hypot((u.x + v.x) / 2.0, (u.y + v.y) / 2.0), r) < 0) {

return 0.5 \* (u \* v);

} else {

double t = atan2(v.y, v.x) - atan2(u.y, u.x);

while (t > PI)

t -= 2 \* PI;

while (t < -PI)

t += 2 \* PI;

return 0.5 \* sqr(r) \* t;

}

}

### Minimum Circle

void min\_cir(int n,cir c)

{ //最小包围圆，主代码

c.pn++;

For(i, n)

if( fcmp( (p[i]-o).norm(), r) > 0 )

{

c.p[ c.pn-1 ] = p[i];

maintain(c);

if( c.pn < 3 )

min\_cir(i, c);

}

}

### Ball Center

void ball(const Point \*p)

{ //空间四点求球心

double a[3],b[3],c[3],d[3];

int i;

For(i, 3)

{

a[i] = 2 \* (p[i].x - p[i+1].x);

b[i] = 2 \* (p[i].y - p[i+1].y);

c[i] = 2 \* (p[i].z - p[i+1].z);

d[i] = sqr( p[i].norm() ) - sqr( p[i+1].norm() );

}

o.x = det(d, b, c) / det(a, b, c);

o.y = det(a, d, c) / det(a, b, c);

o.z = det(a, b, d) / det(a, b, c);

r = (o-p[0]).norm();

}

void circle(const Point \*p)

{ //空间三点外接球球心

double a[3],b[3],c[3],d[3];

int i;

For(i, 2)

{

a[i] = 2 \* (p[i].x - p[i+1].x);

b[i] = 2 \* (p[i].y - p[i+1].y);

c[i] = 2 \* (p[i].z - p[i+1].z);

d[i] = sqr( p[i].norm() ) - sqr( p[i+1].norm() );

}

a[2] = p[0].y \* (p[2].z - p[1].z)

+ p[0].z \* (p[1].y - p[2].y)

+ p[1].z\*p[2].y - p[1].y\*p[2].z;

b[2] = p[0].x \* (p[1].z - p[2].z)

+ p[0].z \*(p[2].x - p[1].x)

+ p[1].x\*p[2].z - p[1].z\*p[2].x;

c[2] = p[0].y \* (p[1].x - p[2].x)

+ p[0].x \* (p[2].y - p[1].y)

+ p[1].y\*p[2].x - p[1].x\*p[2].y;

d[2] = p[0].x\*p[1].z\*p[2].y

+ p[0].y\*p[1].x\*p[2].z

+ p[0].z\*p[1].y\*p[2].x

- p[0].x\*p[1].y\*p[2].z

- p[0].y\*p[1].z\*p[2].x

- p[0].z\*p[1].x\*p[2].y ;

o.x = det(d, b, c) / det(a, b, c);

o.y = det(a, d, c) / det(a, b, c);

o.z = det(a, b, d) / det(a, b, c);

r = (o-p[0]).norm();

}

### Cricles’ union area

double cir\_union()

{ /// 圆面积并

int i, j, k, up;

double ans = 0;

double dis, ang, da, low, high;

Point o, a, b;

For(i, cn)

{

up = en = 0;

e[en++] = event(0, 1);

e[en++] = event(pi\_2, -1);

o = c[i];

For(j, cn)

{

dis = (o-c[j]).norm();

if( fcmp(dis, o.r + c[j].r) >= 0 )

continue;

if( fcmp(dis, fabs(o.r-c[j].r)) <= 0 )

{ if( i > j )

goto over;

continue;

}

ang = (c[j]-o).ang();

da = acos( ( sqr(o.r) + sqr(dis) - sqr(c[j].r) )

/ ( 2 \* o.r \* dis ) );

low = def(ang - da);

high = def(ang + da);

e[en++] = event(low, 1);

e[en++] = event(high, -1);

if( fcmp(low, high) > 0 )

up++;

}

sort(e, e+en);

up += e[0].tp;

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

{

if( 1 == up )

{

a = o.to(e[j-1].ang);

b = o.to(e[j].ang);

ang = e[j].ang - e[j-1].ang;

ans += sqr(o.r) \* (ang - sin(ang))

+ a \* b ;

}

up += e[j].tp;

}

over:;

}

return ans / 2.0;

}

## Cut

Line l[MAX], q[MAX];

bool add(double a, double b, double c, vector<Line>& l)

{

//set half-panel ax + by < c

if (sgn(c))

{

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

{

if (sgn(c) < 0)

{

return false;

}

}

else if (sgn(b))

{

Point p1(0, c / b);

Point p2(1, (c - a) / b);

if (sgn(c / b) > 0)

{

if (c < 0)

{

l.push\_back(Line(p1, p2));

}

else

{

l.push\_back(Line(p2, p1));

}

}

else

{

if (c > 0)

{

l.push\_back(Line(p1, p2));

}

else

{

l.push\_back(Line(p2, p1));

}

}

}

else

{

Point p1(c / a, 0);

Point p2(c / a, 1);

if (sgn(c / a) > 0)

{

if (c > 0)

{

l.push\_back(Line(p1, p2));

}

else

{

l.push\_back(Line(p2, p1));

}

}

else

{

if (c < 0)

{

l.push\_back(Line(p1, p2));

}

else

{

l.push\_back(Line(p2, p1));

}

}

}

}

else

{

if (sgn(a))

{

Point p1(0, 0);

Point p2(-b / a, 1);

if (a > 0)

{

l.push\_back(Line(p1, p2));

}

else

{

l.push\_back(Line(p2, p1));

}

}

else

{

Point p1(0, 0);

Point p2(-1, 0);

if (sgn(b))

{

if (b > 0)

{

l.push\_back(Line(p1, p2));

}

else

{

l.push\_back(Line(p2, p1));

}

}

else

{

return false;

}

}

}

return true;

}

Polygon run(int n)

{

//Time: O(NlogN)

int f, b;

Polygon ret;

//the line vector must be anti-clockwise

//cut the right side and the left side gets left

if (n < 3)

{

return ret;

}

b = 1;

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

{

l[i].set();

}

sort(l, l + n);

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

{

if (dbcmp(l[i].k, l[i - 1].k))

{

l[b++] = l[i];

}

}

n = b;

f = b = 0;

q[b] = l[0];

q[++b] = l[1];

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

{

if (!line\_ins(q[b], q[b - 1]) || !line\_ins(q[f], q[f + 1]))

{

return ret;

}

while (f != b && xmult(l[i].b, ins\_point(q[b], q[b - 1]), l[i].a) < 0)

{

b--;

}

while (f != b && xmult(l[i].b, ins\_point(q[f], q[f + 1]), l[i].a) < 0)

{

f++;

}

q[++b] = l[i];

}

while (f != b && xmult(q[f].b, ins\_point(q[b], q[b - 1]), q[f].a) < 0)

{

b--;

}

while (f != b && xmult(q[b].b, ins\_point(q[f], q[f + 1]), q[b].a) < 0)

{

f++;

}

if (b <= f + 1)

{

return ret;

}

for (int i = f; i < b; i++)

{

ret.push\_back(ins\_point(q[i], q[i + 1]));

}

if (f < b + 1)

{

ret.push\_back(ins\_point(q[f], q[b]));

}

Polygon::iterator it = unique(ret.begin(), ret.end());

ret.erase(it, ret.end());

if (\*ret.begin() == ret.back())

{

ret.pop\_back();

}

return ret;

}

vector<Point> cut(const vector<Point>& vt, const Line& l)

{

//Time: O(N ^ 2)

vector < Point > ret[3];

Line side;

Point p;

int n, cur, pre;

ret[LEFT].clear();

ret[RIGHT].clear();

ret[ONLINE].clear();

n = vt.size();

if (n == 0) return vt;

pre = cur = relation(vt[0], l);

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

{

cur = relation(vt[(i + 1) % n], l);

if (cur == pre)

{

ret[cur].push\_back(vt[(i + 1) % n]);

}

else

{

side.a = vt[i];

side.b = vt[(i + 1) % n];

p = ins\_point(side, l);

ret[pre].push\_back(p);

ret[cur].push\_back(p);

ret[cur].push\_back(vt[(i + 1) % n]);

pre = cur;

}

}

return ret[LEFT];

}

## Nearest point pair

bool cmp1(const Point& a, const Point& b)

{

return a.x < b.x;

}

bool cmp2(const Point& a, const Point& b)

{

return a.y < b.y;

}

Point point[MAX];

Point s1[MAX], s2[MAX];

double dfs(int l, int r)

{

int mid, top1, top2, t;

double d1, d2, d, m;

if (l >= r) return oo;

if (l + 1 == r) return (point[l] - point[r]).norm();

mid = (l + r) >> 1;

d1 = dfs(l, mid);

d2 = dfs(mid + 1, r);

if (d1 < d2) d = d1;

else d = d2;

//conbine

m = (point[mid].y + point[mid + 1].y) / 2;

top1 = top2 = 0;

for (int i = mid; i >= l; i--)

if (point[i].y >= m - d) s1[top1++] = point[i];

else break;

for (int i = mid + 1; i <= r; i++)

if (point[i].y <= m + d) s2[top2++] = point[i];

else break;

sort(s1, s1 + top1, cmp1);

sort(s2, s2 + top2, cmp1);

for (int tmp = 0, i = 0; i < top1; i++)

{

while (tmp >= 0 && s2[tmp].x >= s1[i].x - d)

tmp--;

tmp++;

for (t = tmp; s2[t].x <= s1[i].x + d && t < top2; t++)

if ((s1[i] - s2[t]).norm() < d) d = (s1[i] - s2[t]).norm();

tmp = t - 1;

}

return d;

}

## 3D Convex Hull

#include <cstdio>

#include <cmath>

#include <algorithm>

#define eps 1e-7

#define MAXV 305

using namespace std;

//三维点

struct pt {

double x, y, z;

pt() {

}

pt(double \_x, double \_y, double \_z) :

x(\_x), y(\_y), z(\_z) {

}

pt operator -(const pt p1) {

return pt(x - p1.x, y - p1.y, z - p1.z);

}

pt operator \*(pt p) {

return pt(y \* p.z - z \* p.y, z \* p.x - x \* p.z, x \* p.y - y \* p.x);

} //叉乘

double operator ^(pt p) {

return x \* p.x + y \* p.y + z \* p.z;

} //点乘

};

struct \_3DCH {

struct fac {

int a, b, c; //表示凸包一个面上三个点的编号

bool ok; //表示该面是否属于最终凸包中的面

};

int n; //初始点数

pt P[MAXV]; //初始点

int cnt; //凸包表面的三角形数

fac F[MAXV \* 4]; //凸包表面的三角形

int to[MAXV][MAXV];

double vlen(pt a) {

return sqrt(a.x \* a.x + a.y \* a.y + a.z \* a.z);

} //向量长度

double area(pt a, pt b, pt c) {

return vlen((b - a) \* (c - a));

} //三角形面积\*2

double volume(pt a, pt b, pt c, pt d) {

return (b - a) \* (c - a) ^ (d - a);

} //四面体有向体积\*6

//正：点在面同向

double ptof(pt &p, fac & f) {

pt m = P[f.b] - P[f.a], n = P[f.c] - P[f.a], t = p - P[f.a];

return (m \* n) ^ t;

}

void deal(int p, int a, int b) {

int f = to[a][b];

fac add;

if (F[f].ok) {

if (ptof(P[p], F[f]) > eps)

dfs(p, f);

else {

add.a = b, add.b = a, add.c = p, add.ok = 1;

to[p][b] = to[a][p] = to[b][a] = cnt;

F[cnt++] = add;

}

}

}

void dfs(int p, int cur) {

F[cur].ok = 0;

deal(p, F[cur].b, F[cur].a);

deal(p, F[cur].c, F[cur].b);

deal(p, F[cur].a, F[cur].c);

}

bool same(int s, int t) {

pt &a = P[F[s].a], &b = P[F[s].b], &c = P[F[s].c];

return fabs(volume(a, b, c, P[F[t].a])) < eps && fabs(

volume(a, b, c, P[F[t].b])) < eps && fabs(

volume(a, b, c, P[F[t].c])) < eps;

}

//构建三维凸包

void construct() {

if (n < 4)

return;

//使前三点不公线

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

if (vlen((P[0] - P[1]) \* (P[1] - P[i])) > eps) {

swap(P[2], P[i]);

break;

}

}

//使前四点不共面

for (int i = 3; i < n; i++) {

if (fabs((P[0] - P[1]) \* (P[1] - P[2]) ^ (P[0] - P[i])) > eps) {

swap(P[3], P[i]);

break;

}

}

cnt = 0;

fac add;

for (int i = 0; i < 4; i++) {

add.a = (i + 1) % 4, add.b = (i + 2) % 4, add.c = (i + 3) % 4, add.ok

= 1;

if (ptof(P[i], add) > 0)

swap(add.b, add.c);

to[add.a][add.b] = to[add.b][add.c] = to[add.c][add.a] = cnt;

F[cnt++] = add;

}

for (int i = 4; i < n; i++) {

for (int j = 0; j < cnt; j++) {

if (F[j].ok && ptof(P[i], F[j]) > eps) {

dfs(i, j);

break;

}

}

}

int tmp = cnt;

cnt = 0;

for (int i = 0; i < tmp; i++) {

if (F[i].ok) {

F[cnt++] = F[i];

}

}

}

//表面积

double area() {

double ret = 0.0;

for (int i = 0; i < cnt; i++) {

ret += area(P[F[i].a], P[F[i].b], P[F[i].c]);

}

return ret / 2.0;

}

//体积

double volume() {

pt O(0, 0, 0);

double ret = 0.0;

for (int i = 0; i < cnt; i++) {

ret += volume(O, P[F[i].a], P[F[i].b], P[F[i].c]);

}

return fabs(ret / 6.0);

}

//表面三角形数

int facetCnt\_tri() {

return cnt;

}

//表面多边形数，即面的个数

int facetCnt() {

int ans = 0;

for (int i = 0; i < cnt; i++) {

bool nb = 1;

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

if (same(i, j)) {

nb = 0;

break;

}

}

ans += nb;

}

return ans;

}

};

int sample() {

\_3DCH hull;

while (~scanf("%d", &hull.n)) {

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

scanf("%lf%lf%lf", &hull.P[i].x, &hull.P[i].y, &hull.P[i].z);

hull.construct();

printf("%d\n", hull.facetCnt());

}

return 0;

}

## Sweep line

#include <cstdio>

#include <cstring>

#include <cstdlib>

#include <cmath>

#include <set>

#include <algorithm>

using namespace std;

const int MAX = 50005;

const double eps = 1e-8;

int sgn(const double& x)

{

return x < -eps ? -1 : x > eps;

}

int dbcmp(const double& x, const double& y)

{

return sgn(x - y);

}

double sqr(const double& x)

{

return x \* x;

}

enum

{

down, up

};

double tx; //now x

struct Circle

{

double x, y, r;

int ret;

void init()

{

scanf("%lf%lf%lf", &x, &y, &r);

}

double getY(const int& side)

{

double dy = sqrt(sqr(r) - sqr(tx - x));

if (side == up) return y + dy;

else return y - dy;

}

} c[MAX];

struct Node

{

int id;

int side;

Node()

{

}

Node(const int& id, const int& side) :

id(id), side(side)

{

}

bool operator<(const Node& node) const

{

double y1 = c[id].getY(side);

double y2 = c[node.id].getY(node.side);

if (dbcmp(y1, y2)) return y1 < y2;

return side < node.side;

}

};

set<Node> st;

typedef set<Node>::iterator Ptr;

struct Event

{

double x, y;

int id;

bool st; //false - in true - out

Event()

{

}

Event(const double& x, const double& y, const int& id, const bool& st) :

x(x), y(y), id(id), st(st)

{

}

bool operator<(const Event& e) const

{

if (dbcmp(x, e.x)) return x < e.x;

else return y < e.y;

}

} e[MAX \* 2];

int n, ne;

int run()

{

int ret = 1;

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

{

tx = e[i].x;

if (e[i].st) st.erase(Node(e[i].id, down)), st.erase(Node(e[i].id, up));

else

{

Ptr it = st.insert(Node(e[i].id, down)).first;

Ptr l = it, r = it;

if (l-- == st.begin() || ++r == st.end()) c[e[i].id].ret = 1;

else

{

if (r->id == l->id) c[it->id].ret = c[r->id].ret + 1;

else c[it->id].ret = max(c[l->id].ret, c[r->id].ret);

}

st.insert(Node(e[i].id, up));

}

}

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

ret = max(ret, c[i].ret);

return ret;

}

int main()

{

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

{

ne = 0, st.clear();

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

{

c[i].init();

e[ne++] = Event(c[i].x - c[i].r, c[i].y, i, 0);

e[ne++] = Event(c[i].x + c[i].r, c[i].y, i, 1);

}

sort(e, e + ne);

printf("%d\n", run());

}

return 0;

}

# Math

## Gauss

double Gauss()

{

double ans[N];

MEM(ans);

int i,j,k,r,c;

double tp;

for(r=c=0; r<an && c<an; r++,c++)

{

int R = r;

for(i=r+1; i<an; i++)

if( fabs(a[i][c]) > fabs(a[r][c]) )

swap(a[i], a[r]);

if( fabs(a[r][c]) < eps )

{

--r;

continue;

}

for(i=r+1; i<an; i++)

{

tp = a[i][c] / a[r][c];

for(j=c; j<=an; j++)

a[i][j] -= tp \* a[r][j];

}

}

for(i=r; i<an; i++)

if( fabs(a[i][an]) > eps )

return -1;

for(r--,c--; r>=0 && c>=0; r--,c--)

{

while( fabs(a[r][c]) < eps )

c--;

for(j=c+1; j<an; j++)

a[r][an] -= a[r][j] \* ans[j];

ans[c] = a[r][an] / a[r][c];

}

return ans[0];

}

## Math relative

组合数学，果断打表，节约时间

C(m, n)=C(m-1, n)+C(m-1, n-1) /// n 选 m

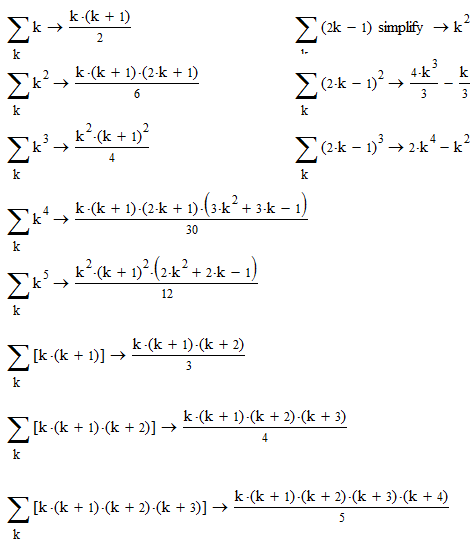
卡特兰数: 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786,

令h(1)＝1,h(0)=1，catalan数满足：h(n)= h(0)\*h(n-1)+ ... + h(n-1)h(0)

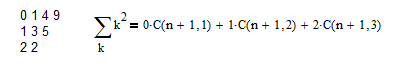
另类递归式：h(n)=((4\*n-2)/(n+1))\*h(n-1);

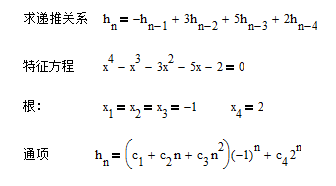
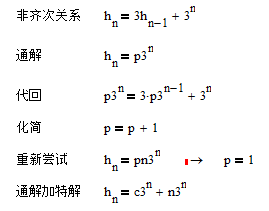
该递推关系的解为：h(n)=C(2n,n)/(n+1)

错排公式：D(n) = n!(1/2! - 1/3! + ... + (-1)^n/n!) = (n-1)(D(n-2) - D(n-1))









第二类Stirling数表示将n个元素划分为k个集合的方案数

S( n , k ) = S( n-1, k-1 ) + k\*S( n-1, k )

Bell 数：n个元素划分为集合的文字总数 S(n,k) 求和，同时：

B[1~10] = 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975

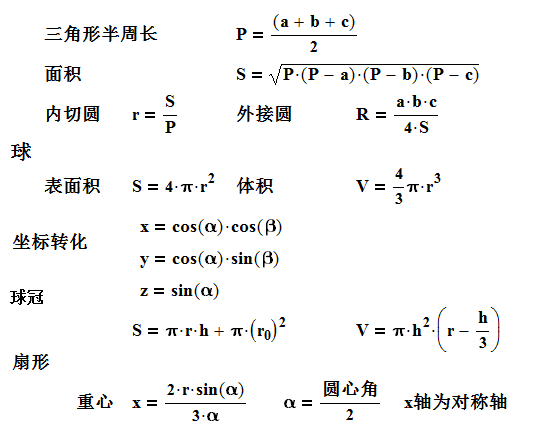


fibonacci number

封闭曲线(如一般位置上的圆)分平面的最大区域数的序列为:

2, 4, 8, 14, 22, ...,

递推公式是:fn=fn-1+2(n-1)=n2-n+2



## RMQ

void RMQ(int rmq[][N],int n)

{

int i,j,k;

// to[0] = to[1] = 0;

// for(i=1;i\*2<N;i++)

// to[i\*2] = to[i\*2+1] = to[i]+1;

int rn = floor( log(n\*2.0) / log(2.0) );

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

For(j, n+2 - (1<<(i-1)) )

rmq[i][j] = MIN( rmq[i-1][j]

, rmq[ i-1 ][ j+ (1<<(i-1)) ] );

}

/// if(a>b) swap

int askRMQ(int rmq[][N],int a,int b) /// [a,b]闭区间

{

int rq = floor( log(b-a+0.0) / log(2.0) );

// int rq = to[ b-a ];

return MIN( rmq[rq][a] , rmq[ rq ][ b+1 - (1<<rq) ] );

}

## LRS

int trans[N][M];

int pre[N];

int mx[N];

int LRS(int now,int p)

{

if( p == pre[now] )

return mx[now];

while( pre[now] != pre[p] )

p = pre[ p ];

return MIN(mx[now], mx[p]);

}

int def\_lrs(int now,int p)

{

return 0==p? 0:(1+LRS(now,p-1));

}

int ask\_lrs(char \*s,int sn)

{

int i, now, p;

char c;

MEME(trans);

MEM(mx); s--;

pre[0] = -1;

for(now=i=1; i<=sn; now=i)

{

c = s[i] - 'a';

trans[i][c] = i + 1;

for(p=pre[i++]; -1!=p&&-1==trans[p][c]; )

{

trans[p][c] = i;

p = pre[ now = p ];

}

pre[i] = (-1==p? 0:trans[p][c]);

mx[i] = def\_lrs(now, pre[i]);

}

}

## Gcd

//扩展Euclid 求解gcd(a,b)=ax+by -->x,y;

// x += k\*b/gcd(a,b); y+= k \* a /gcd(a,b);

ll e\_gcd(ll a,ll b,ll& x,ll& y)

{

if (!b)

{

x=1; y=0;

return a;

}

ll ret = e\_gcd(b, a%b, y, x);

y -= a/b\*x ;

return ret;

}

## linear equations

void gcd\_ex(ll a, ll b, ll &x, ll &y, ll &d) {

if (b == 0) {

d = a;

x = 1;

y = 0;

return;

}

gcd\_ex(b, a % b, x, y, d);

ll t = x;

x = y;

y = t - (a / b) \* y;

}//扩展欧几里得算法

ll mod\_mult(ll a, ll b, ll n) //计算a\*b mod n，支持负数

{

if (b == 0) return 0;

if (b == 1) return a % n;

return (mod\_mult((2 \* a) % n, b / 2, n)+(a \* (b % 2)) % n) % n;

}

int China(ll a[], ll b[], int n) //a 为取模得到的结果，b 为模数，n为方程个数，从0 存起

{

ll x, d, y;

int i;

if (a[0] <= 0) a[0] += b[0];

for (i = 0; i < n - 1; i++)

{

gcd\_ex(b[i], b[i + 1], x, y, d);

if ((a[i + 1] - a[i]) % d != 0) return 0;

//无解的情况

b[i + 1] = b[i] / d \* b[i + 1];

//a[i+1]=(x\*(a[i+1]-a[i])/d\*b[i]+a[i])%(b[i]/d\*b[i+1]);

//各数较小时直接用它计算

a[i + 1] = mod\_mult((a[i + 1] - a[i]) / d, x, b[i + 1]);

a[i + 1] = mod\_mult(a[i + 1], b[i], b[i + 1]);

a[i + 1] = (a[i + 1] + a[i]) % b[i + 1];

if (a[i + 1] <= 0) a[i + 1] += b[i + 1];

}

return 1; //有解，存在a[n-1]中

}

ll get\_num()

{

ll x, m1, c, k, y1, y2, g, i;

x = a[0];

m1 = m[0];

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

{

c = a[i] - x;

g = e\_gcd(m1, m[i], y1, y2);

if( 0 != c%g )

return -1;

k = m[i]/g;

y1 = mod(y1, k);

y1 = mod(y1 \* (c/g), k);

x += y1 \* m1;

m1 \*= m[i] / g;

x %= m1;

}

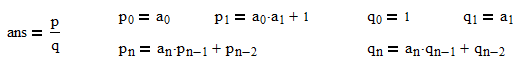
if( 0==x )

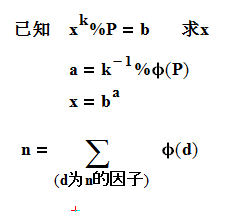
x = m1;

return x;

}

## simple continued fraction expansion



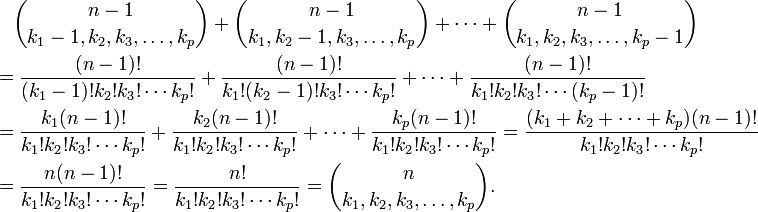


## Number theory

正整数n 可以表示为两个整数的平方和，当且仅当n 的每个4k+3形式的素因子在n 的素幂分解形式中为偶次方。

对于正整数m，令S为所有小于m且与m互质的正整数的集合，P为S中所有元素的乘积。则P ≡ ±1 (mod m)，且P ≡ -1 (mod m)当且仅当m有原根。

E:\备份\金山快盘\C代码\备注\模板\3383c025f8fd85f137d71c88ee43a97d.pngE:\备份\金山快盘\C代码\备注\模板\22ef220153cdbf501fb3b8400f813217.png

E:\备份\金山快盘\C代码\备注\模板\76eb540033111b8ad04e73e5dd4b21d2.pngE:\备份\金山快盘\C代码\备注\模板\a58c5958da6a29c8a3ebab34ed3a4510.png

## Miller rabin

#include<iostream>

#include<ctime>

#include<cstdlib>

#include<cmath>

#include<algorithm>

#define MAX (pow(2.0, 60))

//标记最大值

#define C 240

#define TIME 12

//Miller 测试次数

using namespace std;

typedef long long ll;

ll MIN;

ll gcd(ll a, ll b)

//计算a 和b 的最大公约数

{

if (b == 0)

return a;

return gcd(b, a % b);

}

ll mod\_mult(ll a, ll b, ll n) //计算a\*b mod n

{

if (b == 0) return 0;

if (b == 1) return a % n;

return (mod\_mult((2 \* a) % n, b / 2, n)+(a \* (b % 2)) % n) % n;

}

ll mod\_exp(ll a, ll b, ll n)

//计算(a^b) mod n

{

ll d = 1;

a = a % n;

while (b >= 1) {

if (b & 1)

- 3 -

d = mod\_mult(d, a, n);

a = mod\_mult(a, a, n);

b = b >> 1;

}

return d;

}

bool Wintess(ll a, ll n) //以a 为基对n进行Miller 测试并实现二次探测

{

ll m, x, y;

int i, j = 0;

m = n - 1;

while (m % 2 == 0) //

计算(n - 1) = m \* (2^j)中的j 和m, j = 0时m = n - 1, 不断的除以2 直至n 为奇数

{

m = m >> 1;

j++;

}

x = mod\_exp(a, m, n);

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

y = mod\_exp(x, 2, n);

if ((y == 1) && (x != 1) && (x != n - 1)) //二次探测

return true;

//返回true时,n是合数

x = y;

}

if (y != 1)

return true;

return false;

}

bool miller\_rabin(int times, ll n) //对n进行s次的Miller测试

{

ll a;

int i;

if (n == 1)

return false;

if (n == 2)

return true;

if (n % 2 == 0)

return false;

srand(time(NULL));

for (i = 1; i <= times; i++) {

a = rand() % (n - 1) + 1;

if (Wintess(a, n))

return false;

}

return true;

}

ll Pollard(ll n, int c) //对n进行因字分解,找出n的一个因子,注意该因子不一定是最小的

{

ll i, k, x, y, d;

srand(time(NULL));

i = 1;

- 4 -

k = 2;

x = rand() % n;

y = x;

while (true) {

i++;

x = (mod\_mult(x, x, n) + c) % n;

d = gcd(y - x, n);

if (d > 1 && d < n)

return d;

if (y == x)

//该数已经出现过,直接返回即可

return n;

if (i == k) {

y = x;

k = k << 1;

}

}

}

void get\_small(ll n, int c) //找出最小的素数因子

{

ll m;

if (n == 1)

return;

if (miller\_rabin(TIME, n)) //判断是否为素数

{

if (n < MIN)

MIN = n;

return;

}

m = n;

while (m == n) //找出n

的一个因子

m = Pollard(n, c--);

get\_small(m, c); //二分查找

get\_small(n / m, c);

}

# Other

## Head files

# pragma comment (linker,"/STACK:16777216")

# define For(i,a) for((i)=0;i<(a);(i)++)

# define MAX(x,y) ((x)>(y)? (x):(y))

# define MIN(x,y) ((x)<(y)? (x):(y))

# define MEM(a) (memset((a),0,sizeof(a)))

# define pb(a) push\_back(a)

typedef long long ll ;

typedef unsigned long long ull ;

typedef istringstream Iss ;

typedef Vs::iterator Vsi ;

template<class T> inline void checkmin(T &a,T b){if(a>b) a=b;}

template<class T> inline void checkmax(T &a,T b){if(a<b) a=b;}

make\_heap pop\_heap push\_heap sort\_heap

aq[an++] = dn;

push\_heap(aq, aq+an, cmp\_a);

pop\_heap(aq, aq+(an--), cmp\_a);

生成字典序函数：next() 已达到最大后返回 false ，并重置最小。

next\_permutation(a,a+n) , prev\_permutation(a,a+n);

random\_shuffle(a, a+n);

判断矩形相交

max(Xa1,Xb1) <= min(Xa2,Xb2)

max(Ya1,Yb1) <= min(Ya2,Yb2)

priority\_queue<ll> a;

priority\_queue< ll,vector<ll>, greater<ll> > b;

Ubuntu 设置

系统 -> 首选项 -> 外观 -> 自定义 -> 颜色 -> 输入法 -> 背景 -> RGB: 199 237 204

文本编辑器 -> 编辑 -> 首选项 ->

-> 查看 -> 启动自动换行..不将单词拆分两行

-> 显示行号, 突出当前行, 突出显示匹配括号

皮克定理——简单多边形

面积：A、内部格点数：i、边上格点数目：b、满足：A = i + b/2 - 1。

C++ Sets

集合(Set)是一种包含已排序对象的关联容器

clear() 清除所有元素

count() 返回某个值元素的个数

erase() 删除集合中的元素

find() 返回一个指向被查找到元素的迭代器

insert() 在集合中插入元素

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

upper\_bound() 返回大于某个值元素的迭代器

线性逆元预处理：(P%i) \* x = 1 (mod P)

int dayofweek(int y, int m, int d) /\* 0 = Sunday \*/

{

int t[] = {0, 3, 2, 5, 0, 3, 5, 1, 4, 6, 2, 4};

y -= m < 3;

return (y + y/4 - y/100 + y/400 + t[m-1] + d) % 7;

}

C++ -> using namespace stdext;

G++

# include <hash\_map>

# include <hash\_set>

using namespace \_\_gnu\_cxx;

namespace \_\_gnu\_cxx

{

template<> struct hash<const string>

{

size\_t operator()(const string& s) const

{ return hash<const char\*>()( s.c\_str() ); }

};

template<> struct hash<string>

{

size\_t operator()(const string& s) const

{ return hash<const char\*>()( s.c\_str() ); }

};

}

hash\_map<string,int> mp;

hash\_set<string,int> ms;

System.out.println( "start: "+new java.sql.Timestamp(System.currentTimeMillis()));

cout << fixed << setprecision(4);

## Java

import java.io.\*;

import java.util.\*;

import java.math.\*;

public class Main {

public static void main(String args[]) throws FileNotFoundException,

IOException {

Scanner sc = new Scanner(new FileReader("a.in"));

PrintWriter pw = new PrintWriter(new FileWriter("a.out"));

int c, ci, i;

c = sc.nextInt();

for (ci = 1; ci <= c; ++ci) {

pw.println("Case #" + ci + ": easy for output");

}

pw.close();

sc.close();

}

}

5. 进制转换

java很强大的一个功能。

函数：

String st = Integer.toString( num, base);

// 把num当做10进制的数转成base进制的st(base <= 35).

int num = Integer.parseInt(st, base);

// 把st当做base进制，转成10进制的int

BigInter m = new BigInteger(st, base);

// st是字符串，base是st的进制.

// 这里0指一位数字，#指除0以外的数字(如果是0，则不显示),四舍五入.

DecimalFormat fd = new DecimalFormat("#.00#");

DecimalFormat gd = new DecimalFormat("0.000");

System.out.println("x =" + fd.format(x));

System.out.println("x =" + gd.format(x));

# String

## KMP

void kmp()

{

int j,k;

next[0] = -1;

j = -1;k = 0;

while( s[k] )

if( -1==j || s[j]==s[k] )

{//修正为 if(s[i]==s[k]) next[i] = next[k]; else

j++,k++;

if( s[ j ] == s[k] )

next[k] = next[j];

else

next[k] = j;

}

else

j = next[j];

sn = k;

}

int count()

{

int i,j,k;

int cnt = 0;

i = j = 0;

while( t[ i ] )

{

if(-1==j || t[i] == s[j] )

i++,j++;

else

j = next[j];

if( j==sn )

j=next[j],cnt++;

}

return cnt;

}

## Extended KMP

void e\_kmp(char \*s,char \*t,int \*has,int \*e\_has)

{

int sp,p,mx,tn;

for(sp=p=mx=0; s[p]>0; p++)

{

if( mx == p || p+e\_has[p-sp]>=mx )

{

for(tn = mx-p; s[mx]==t[tn]; tn++)

mx++;

has[ sp=p ] = mx - p;

if( mx==p )

sp = mx = p+1;

}

else

has[ p ] = e\_has[ p-sp ];

}

}

int main()

{

gets(s);gets(t);

t[tn] = -1;

e\_has[0] = tn;

e\_kmp(t+1, t, e\_has+1, e\_has);

e\_kmp(s, t, has, e\_has);

}

## Palindrome

void pk()

{

int i,j,k,mx,id;

for(mx=p[0]=0,i=1; i<up; i++)

{

if( mx > i )

p[i] = min(p[id\*2-i], mx-i);

else

p[i] = 1;

while( str[i+p[i]] == str[i-p[i]] )

p[i]++;

if( p[i] + i > mx )

{

mx = p[i] + i;

id = i;

}

}

}

## Cyclic String minimum expression

int pk(char \*s,int len,int flag) /// 字符串最小表示法

{

int i,j,k,t;

i = 0;

j = 1;

k = 0;

while( i<len && j<len && k < len )

{

t = s[i+k] - s[j+k];

if( 0 == t )

k++;

else

{

if( t\*flag > 0 )

{

i += k+1;

t = s[i] - s[j]; ///优化

if( t\*flag < 0 )

j = i+1;

}

else

{

j += k+1;

t = s[i] - s[j]; ///优化

if( t\*flag > 0 )

i = j+1;

}

if( i==j )

j++;

k = 0;

}

}

return MIN(i,j);

}

/// flag 1 min, -1 max

## AC Machine

struct tree

{ tree \*next[26],\*fail;

int end;

tree(){ memset(this,0,sz(tree)); }

};

tree tr[N\*50],vd;

tree \*Q[N\*50],\*root;

char s[M];

void build()

{

int n,i,j,k;

tree \*p;

root = tr + (trn++);

while( n-- )

{

p = root;

for(i=0; s[i]; i++)

{ if( NULL == p->next[ s[i]-'a' ] )

{

tr[trn] = vd;

p->next[ s[i]-'a' ] = tr + (trn++);

}

p = p->next[ s[i]-'a' ] ;

}

p->end++;

}

}

void pre()

{

int i,j,k,f,b;

tree \*p,\*q;

f = b = 0;

Q[b++] = root;

while( f < b )

{

p = Q[f++];

For(i,26)

if( NULL != p->next[i] )

{

Q[b++] = p->next[i];

if( root == p )

{

p->next[i]->fail = root;

continue;

}

q = p->fail;

while( NULL != q )

if( NULL != q->next[i] )

{

p->next[i]->fail = q->next[i];

break;

}

else

q = q->fail;

if( NULL == q )

p->next[i]->fail = root;

}

}

}

int work()

{

int i,j,k;

int ans = 0;

tree \*q,\*p = root;

for(i=0; s[i]; i++)

{

while(root != p && NULL == p->next[ s[i]-'a' ] )

p = p->fail;

p = p->next[ s[i]-'a' ];

if( NULL == p )

p = root;

q = p;

while( root != q && -1 != q->end )

{ ans += q->end;

q->end = -1;

q = q->fail;

}

}

return ans;

}

## Suffix array da

char s[N]; /// 长度+1，对于非字符串，加一个小于最小值的元素，

int sa[N]; /// 倍增算法，结果 下标 1-n，第 i 大的是 sa[i]

int rk[N]; /// 第 i 位开始的后缀，的排名为 rk[i]

int wa[N], wb[N], wv[N], rmq[20][N];

int sn, to[N];

bool cmp(int \*y,int a,int b,int L)

{

return y[a]==y[b] && y[a+L]==y[b+L];

}

void da(char \*s,int \*sa,int len,int dn)

{

int i,j,p;

int \*x,\*y,\*t;

x = wa; y = wb;

For(i, dn) rk[i] = 0;

For(i, len) rk[ x[i]=s[i] ]++;

For(i, dn) rk[i+1] += rk[i];

for(i=len-1; i>=0; i--) sa[ --rk[ x[i] ] ] = i;

for( j=1,p=1; p<len; j\*=2,dn=p)

{

For(p, j) y[p] = len-j + p;

For(i, len) if(sa[i] >= j) y[p++] = sa[i] - j;

For(i, len) wv[i] = x[ y[i] ];

For(i, dn) rk[i] = 0;

For(i, len) rk[ wv[i] ]++;

For(i, dn) rk[ i+1 ] += rk[i];

for(i=len-1; i>=0; i--) sa[ --rk[ wv[i] ] ] = y[i];

swap(x,y); x[ sa[0] ] = 0;

for(p =i =1; i<len; i++)

{

p += ! cmp(y, sa[i], sa[i-1], j) ;

x[ sa[i] ] = p-1;

}

}

}

void find\_height(char \*s,int \*sa,int len)

{

int \*h = rmq[0]; /// sa中为 i 的点，与 i-1 的最长公共前缀

int i,j,k = 0;

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

rk[ sa[i] ] = i;

For(i,3)

h[len+i] = 0;

For(i, len)

{

if( k>0 )

k--;

j = sa[ rk[i]-1 ];

while( s[i+k] == s[j+k] )

k++;

h[ rk[i] ] = k;

//printf("h %d = %d\n",rk[i],k);

}

}

void RMQ(int n)

{

int i,j,k;

int rn = floor( log(n\*2.0) / log(2.0) );

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

For(j, n+2 - (1<<(i-1)) )

rmq[i][j] = MIN( rmq[i-1][j] , rmq[ i-1 ][ j+ (1<<(i-1)) ] );

}

int askRMQ(int a,int b) /// [a,b]闭区间

{

//int rq = floor( log(b-a+0.0) / log(2.0) );

int rq = to[ b-a ];

return MIN( rmq[rq][a] , rmq[ rq ][ b+1 - (1<<rq) ] );

}

void PT(char \*s,int \*sa)

{

int i,sn;

sn = strlen(s);

For(i,sn)

puts(s+sa[i+1]);

puts("");

For(i,sn)

printf("rank %d = %d\n",i,rk[i]);

}

int lcp(int a,int b,int len)

{

if( a==b )

return len-a;

a = rk[a];

b = rk[b];

if( a>b )

swap(a,b);

return askRMQ(a+1,b);

}

void pre\_log()

{ int i;

to[0] = to[1] = 0;

for(i=1;i\*2<N;i++)

to[i\*2] = to[i\*2+1] = to[i]+1;

}

int main()

{

int T,\_=0;

pre\_log();

while( ~scanf("%s",s) )

{

sn = strlen(s);

da(s, sa, sn+1, 128);

find\_height(s, sa, sn);

RMQ(sn);

PT(s,sa);

scanf("%d",&T);

while( T-- )

{ int a,b;

scanf("%d%d",&a,&b);

a--,b--; /// 求原串的 a b 开始的后缀的公共前缀

printf("lcp = %d\n",lcp(a,b,sn));

}

}

return 0;

}

// For(i, len)

// sa[i]=i;

// sort(sa, sa+len, sacmp);

// for(p =i =1; i<len; i++)

// {

// p += (s[ sa[i-1] ] != s[ sa[i] ]);

// x[ sa[i] ] = p-1;

// }

// x[ sa[0] ] = 0;

// 注意此时也应传入正确dn。

## Suffix array dc3

/// 诡异的 DC3 数组内存为字符串 3 倍

# define F(x) ( (x)/3 + ((x)%3==1? 0:tb) )

# define G(x) ( (x)<tb? (x)\*3+1 : ((x)-tb)\*3+2 )

int s3[3\*N],sa[3\*N];

int c0(int \*r,int a,int b)

{ return r[a]==r[b] && r[a+1]==r[b+1] && r[a+2]==r[b+2] ; }

int c12(int k,int \*r,int a,int b)

{ if(k==2) return r[a]<r[b] || r[a]==r[b] && c12(1, r, a+1, b+1);

else return r[a]<r[b] || r[a]==r[b] && wv[a+1]<wv[b+1] ;}

void sort(int \*r,int \*a,int \*b,int n,int m)

{ int i;

For(i, n) wv[i] = r[ a[i] ];

For(i, m) rk[i] = 0;

For(i, n) rk[ wv[i] ]++;

For(i, m) rk[i] += rk[i-1];

for(i=n-1; i>=0; i--) b[ --rk[ wv[i]]] = a[i]; }

void dc3(int \*r,int \*sa,int n,int m)

{ /// \*r 必须为 int

int i, j, p, ta=0, tb=(n+1)/3, tbc=0;

int \*rn = r+n, \*san = sa+n;

r[n] = r[n+1] = 0;

For(i, n) if(i%3!=0) wa[tbc++]=i;

sort(r+2, wa, wb, tbc, m);

sort(r+1, wb, wa, tbc, m);

sort(r , wa, wb, tbc, m);

for(rn[ F(wb[0]) ]=0,i=p=1; i<tbc; i++)

rn[ F(wb[i]) ] = c0(r,wb[i-1],wb[i])? p-1:p++;

if(p<tbc) dc3(rn, san, tbc, p);

else For(i, tbc) san[ rn[i]] = i;

For(i, tbc) if(san[i]<tb) wb[ta++] = san[i]\*3;

if(n%3==1) wb[ta++] = n-1;

sort(r, wb, wa, ta, m);

For(i, tbc) wv[ wb[i]= G(san[i]) ] = i;

for(i=j=p=0; i<ta && j<tbc; p++)

sa[p] = c12(wb[j]%3, r, wa[i], wb[j])? wa[i++]:wb[j++];

for(; i<ta; p++) sa[p] = wa[i++];

for(; j<tbc; p++) sa[p] = wb[j++];

}