**Vimrc**

set showcmd,nu,cindent,autoindent,ts=4,sw=4,mouse=a

map <F12> ggVG"+y map <F2> :vsp %<.in <CR>

autocmd filetype cpp nmap <F9> :w<CR>:make %< <CR>

autocmd filetype cpp nmap <F7> :w<CR>:!g++ % -o %< -O2 -Wall <CR>

autocmd filetype cpp nmap <F8> :!time ./%< < %<.in <CR>

autocmd filetype java :compiler javac

autocmd filetype java nmap <F9> :w<CR>:make % <CR>

autocmd filetype java nmap <F8> :!time java %< < %<.in <CR>

**Suffix Array**

const int MaxLen = 100000 + 5, MaxKey = MaxLen;

int Len;

int SA[MaxLen], tmp[MaxLen], height[MaxLen];

int a[MaxLen \* 2], rank[MaxLen \* 2], tmp2[MaxLen \* 2];

int hash[MaxKey];

void RadixSort(int\* key) {

memset(hash, 0, sizeof(hash));

for (int i = 1; i <= Len; ++i) hash[key[i]]++;

for (int i = 1; i < MaxKey; ++i) hash[i] += hash[i - 1];

for (int i = Len; i >= 1; --i) SA[hash[key[tmp[i]]]--] = tmp[i];

}

void GetSA() {

if (Len == 1) {SA[1] = rank[1] = 1; return;}

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

memcpy(rank, a, sizeof(rank)); RadixSort(rank);

for (int k = 1; k < Len; k <<= 1) {

RadixSort(rank + k); memcpy(tmp, SA, sizeof(tmp));

RadixSort(rank); memcpy(tmp2, rank, sizeof(tmp2));

for (int i = 1, p = 1; i <= Len; ++i) {

rank[SA[i]] = p;

if (tmp2[SA[i]] != tmp2[SA[i + 1]]

|| tmp2[SA[i] + k] != tmp2[SA[i + 1] + k]) p++;

}

}

}

void GetHeight() {

int i, j, p = 0; height[1] = 0;

for (i = 1; i <= Len; ++i) {

if (p) p--; if (rank[i] == 1) continue;

j = SA[rank[i] - 1];

while (a[i + p] == a[j + p]) p++;

height[rank[i]] = p;

}

}

**DLX**

const int MaxN = 1005, MaxM = 1005, MaxC = 105, Total = MaxN \* MaxC;

const int oo = 0x7fffffff;

#define Lab(a, b) (a \* M + b)

int N, M, h, ans, buf;

int lx[MaxN][MaxC], ly[MaxM][MaxC], plx[MaxN], ply[MaxM], O[MaxN];

int L[Total], R[Total], U[Total], D[Total], C[Total], row[Total];

void Init() {

buf = M;

memset(plx, 0, sizeof(plx));

memset(ply, 0, sizeof(ply));

for (int x = 1; x <= N; ++x) {

int c, y; scanf("%d", &c);

while (c--) {

scanf("%d", &y);

lx[x][plx[x]++] = ly[y][ply[y]++] = ++buf;

C[buf] = y; row[buf] = x;

}

}

int a, b;

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

for (int k = 0; k < plx[x]; ++k) {

a = lx[x][k]; b = lx[x][(k + 1) % plx[x]];

R[a] = b; L[b] = a;

}

for (int y = 0; y <= M; ++y) {

R[y] = (y + 1) % (M + 1); L[(y + 1) % (M + 1)] = y;

}

for (int y = 1; y <= M; ++y) {

for (int k = 0; k < ply[y] - 1; ++k) {

a = ly[y][k]; b = ly[y][k + 1];

D[a] = b; U[b] = a;

}

if (ply[y]) {

U[ly[y][0]] = y; D[y] = ly[y][0];

D[ly[y][ply[y] - 1]] = y; U[y] = ly[y][ply[y] - 1];

}

else D[y] = U[y] = y;

}

}

void remove(int c) {

L[R[c]] = L[c]; R[L[c]] = R[c];

for (int x = D[c]; x != c; x = D[x])

for (int y = R[x]; y != x; y = R[y])

U[D[y]] = U[y], D[U[y]] = D[y], ply[C[y]]--;

}

void resume(int c) {

for (int x = U[c]; x != c; x = U[x])

for (int y = L[x]; y != x; y = L[y])

U[D[y]] = y, D[U[y]] = y, ply[C[y]]++;

L[R[c]] = c; R[L[c]] = c;

}

bool Dfs(int k) {

if (R[h] == h) {ans = k - 1; return true;}

int s = oo, c;

for (int y = R[h]; y != h; y = R[y])

if (ply[y] < s) s = ply[y], c = y;

remove(c);

for (int x = D[c]; x != c; x = D[x]) {

O[k] = row[x];

for (int y = R[x]; y != x; y = R[y]) remove(C[y]);

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

for (int y = L[x]; y != x; y = L[y]) resume(C[y]);

}

resume(c); return false;

}

void Solve() {

h = ans = 0;

if (Dfs(1)) {

printf("%d", ans); for (int i = 1; i <= ans; ++i) printf(" %d", O[i]);

} else puts("NO");

}

int main() {Init(); Solve();}

**ExtendedGcd**

int ExtendedGcd(int a, int b) {

if (b == 0) {x = 1; y = 0; return a;}

int d = ExtendedGcd(b, a % b);

int t = x; x = y; y = t - a / b \* y;

return d;

}

**ExtendedKMP**

const int MaxN = 100000, MaxM = 100000;

int Next[MaxM], Extend[MaxN];

void GetNext(char\* s) {

int k = 0, a = 1, j;

while (s[k] == s[k + 1]) k++; Next[1] = k;

for (k = 2; s[k]; ++k) {

if (k + Next[k - a] < a + Next[a]) {Next[k] = Next[k - a]; continue;}

j = max(a + Next[a] - k, 0);

while (s[k + j] && s[k + j] == s[j]) j++;

Next[k] = j; a = k;

}

}

void GetExtend(char\* s1, char\* s2) {

int k = 0, a = 0, j;

while (s1[k] && s2[k] && s1[k] == s2[k]) k++; Extend[0] = k;

for (k = 1; s1[k]; ++k) {

if (k + Next[k - a] < a + Extend[a]) {Extend[k] = Next[k - a]; continue;}

j = max(a + Extend[a] - k, 0);

while (s1[k + j] && s2[j] && s1[k + j] == s2[j]) j++;

Extend[k] = j; a = k;

}

}

void ExtendedKMP(char\* s1, char\* s2) {GetNext(s2); GetExtend(s1, s2);}

**SAP**

const int MaxNode = 10000 + 5, MaxE = 100000 + 5;

const int oo = 0x7fffffff;

int S, T, Num;

int d[MaxNode], v[MaxNode];

struct etp {

int t, c; etp \*nxt, \*p;

etp() {}

etp(int \_t, int \_c, etp\* \_nxt): t(\_t), c(\_c), nxt(\_nxt) {}

void\* operator new(size\_t, void\* p) {return p;}

}\*e[MaxNode], buf[MaxE], \*cbuf = buf;

void Me(int a, int b, int cab, int cba) {

e[a] = new(cbuf++)etp(b, cab, e[a]);

e[b] = new(cbuf++)etp(a, cba, e[b]);

(cbuf - 2)->p = cbuf-1; (cbuf - 1)->p = cbuf-2;

}

int aug(int now, int Max) {

if (now == T) return Max;

int Rest = Max, dmin = Num, dt;

for (etp\* u = e[now]; u; u = u->nxt) if (u->c) {

if (Rest && d[u->t] + 1 == d[now]) {

dt = aug(u->t, min(Rest, u->c));

Rest -= dt; u->c -= dt; u->p->c += dt;

if (d[S] >= Num) return Max - Rest;

} if (u->c) dmin = min(d[u->t] + 1, dmin);

}

if (Max == Rest) if (!--v[d[now]]) d[S] = Num; else ++v[d[now] = dmin];

return Max - Rest;

}

int main() {

memset(e, 0, sizeof(e));

memset(d, 0, sizeof(d)); memset(v, 0, sizeof(v)); v[0] = Num;

int ans = 0; while (d[S] < Num) ans += aug(S, oo);

return 0;

}

**MCMF\_SPFA**

const int MaxNode = 10000 + 5, MaxE = 100000 + 5;

const int oo = 0x7fffffff;

int ans, d[MaxNode], pre[MaxNode], S, T, Num;

bool vis[MaxNode];

queue<int> Q;

struct etp {

int t, c, v; etp \*nxt, \*p;

etp() {}

etp(int \_t, int \_c, int \_v, etp\* \_nxt): t(\_t), c(\_c), v(\_v), nxt(\_nxt) {}

void\* operator new(size\_t, void\* p) {return p;}

}\*e[MaxNode], buf[MaxE], \*cbuf = buf, \*r[MaxNode];

void Me(int a, int b, int cab, int cba, int vab, int vba) {

e[a] = new(cbuf++)etp(b, cab, vab, e[a]);

e[b] = new(cbuf++)etp(a, cba, vba, e[b]);

(cbuf - 2)->p = cbuf-1; (cbuf - 1)->p = cbuf-2;

}

bool aug() {

memset(vis, false, sizeof(vis)); vis[S] = true;

for (int i = 0; i < Num; ++i) d[i] = oo; d[S] = 0;

Q.push(S); int o, Max = oo, Cost = 0;

while (!Q.empty()) {

o = Q.front(); Q.pop(); vis[o] = false;

for (etp\* u = e[o]; u; u = u->nxt)

if (u->c && d[o] + u->v < d[u->t]) {

d[u->t] = d[o] + u->v; pre[u->t] = o; r[u->t] = u;

if (!vis[u->t]) {vis[u->t] = true; Q.push(u->t);}

}

}

if (d[T] == oo) return false;

for (int i = T; i != S; i = pre[i]) Max = min(Max, r[i]->c);

for (int i = T; i != S; i = pre[i]) {

Cost += r[i]->v; r[i]->c -= Max; r[i]->p->c += Max;

} ans += Cost \* Max; return true;

}

int main() {

ans = 0; while (aug()) {}

return 0;

}

**AC\_Automation**

const int MaxNode = 100000 + 5;

struct Node {

int t; Node \*f, \*c[26];

void\* operator new(size\_t, void\* p) {return p;}

Node\* rz(Node\* fa) {t = 0; f = fa; return memset(c, 0, sizeof(c)), this;}

}trie[MaxNode], \*cbuf = trie, \*root = (new(cbuf++)Node)->rz(NULL);

queue<Node\*> Q;

void Ins(Node\* now, char\* s) {

for (int i = 0; s[i]; ++i) {

if (now->c[s[i] - 'a']) now = now->c[s[i] - 'a'];

else now = now->c[s[i] - 'a'] = (new(cbuf++)Node)->rz(now);

} now->t++;

}

void GetNext() {

Q.push(root); Node \*u, \*v;

while (!Q.empty()) {

u = Q.front(); Q.pop();

for (int i = 0; i < 26; ++i) if (u->c[i]) {

Q.push(u->c[i]); if (u == root) continue;

v = u->f; while (v != root && !v->c[i]) v = v->f;

if (v->c[i]) u->c[i]->f = v->c[i]; else u->c[i]->f = root;

}

}

}

int Match(Node\* now, char\* s) {

int ret = 0; Node\* d;

for (int i = 0; s[i]; ++i) {

while (now != root && !now->c[s[i] - 'a']) now = now->f;

if (now->c[s[i] - 'a']) now = now->c[s[i] - 'a'];

for (d = now; d != root && d->t != -1; d = d->f)

ret += d->t, d->t = -1;

} return ret;

}

**LinkCutSplay**

const int MaxNode = 30000 + 5;

#define nonRoot(x) ((x)->fa->c[0] == (x) || (x)->fa->c[1] == (x))

#define nonNull(x) ((x)->c[0] != (x) && (x)->c[1] != (x))

void\* stk[MaxNode];

struct Node {

int key, sz, rev; Node \*fa, \*c[2];

Node(): sz(0) {

fa = c[0] = c[1] = this;

}

Node(int \_key, Node\* \_fa, Node\* \_c0, Node\* \_c1): key(\_key), rev(0) {

fa = \_fa; c[0] = \_c0; c[1] = \_c1;

}

void rz() {

sz = c[0]->sz + c[1]->sz + 1;

c[0]->fa = c[1]->fa = this;

}

void flip() {

if (nonNull(this)) {

swap(c[0], c[1]);

rev ^= 1;

}

}

void pd() {

if (rev) {

c[0]->flip(); c[1]->flip();

rev = 0;

}

}

void zig(bool d) {

Node \*x = fa, \*y = x->fa;

x->c[d] = c[!d]; x->rz();

c[!d] = x; x->fa = this;

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

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

fa = y;

}

void splay() {

int top = 0; stk[top++] = this;

for (Node\* v = this; nonRoot(v); v = v->fa) stk[top++] = v->fa;

for (int i = top - 1; i >= 0; --i) ((Node\*)stk[i])->pd();

while (nonRoot(this)) {

bool d = fa->c[1] == this;

if (!nonRoot(fa)) {zig(d); break;}

bool dd = fa->fa->c[1] == fa;

d != dd? (zig(d), zig(dd)): (fa->zig(dd), zig(d));

} rz();

}

}Tnull, \*null = &Tnull, buf[MaxNode], \*cbuf;

struct Splay {

Node \*root;

Splay(int x) {

root = new(cbuf++)Node(x, null, null, null);

root->rz();

}

void select(int k) {

Node\* now = root;

while (now->pd(), now->c[0]->sz != k) {

if (now->c[0]->sz > k) now = now->c[0];

else {k -= now->c[0]->sz + 1; now = now->c[1];}

} now->splay(); root = now;

}

void search(int x) {

Node\* now = root;

while (now->pd(), now->c[x > now->key] != null) {

now = now->c[x > now->key];

} now->splay();root = now;

if (x > root->key) select(root->c[0]->sz + 1);

}

void ins(int x) {

root->pd(); root = new(cbuf++)Node(x, null, root->c[0], root);

root->c[1]->c[0] = null; root->c[1]->rz(); root->rz();

}

void del() {

root->pd(); Node\* oldroot = root;

root = root->c[1]; root->fa = null; select(0);

root->pd(); root->c[0] = oldroot->c[0]; root->rz();

}

};

struct LinkCut {

void access(Node\* u) {

for (Node\* v = null; u != null; u = u->fa) {

u->splay(); u->c[1] = v; (v = u)->rz();

}

}

Node\* getRoot(Node\* u) {

access(u); u->splay();

while (u->c[0] != null) {

u->pd(); u = u->c[0];

} u->splay(); return u;

}

void setRoot(Node\* u) {

access(u); u->splay(); u->flip();

}

void join(Node\* u, Node\* v) {

setRoot(u); u->fa = v; access(u);

}

void cut(Node\* u) {

access(u); u->splay();

u->c[0]->fa = null; u->c[0] = null; u->rz();

}

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

Node\* res; access(x);

for (Node \*u = y, \*v = null; u != null; u = u->fa) {

if (u->splay(), u->fa == null) {v->fa = u; res = u;}

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

} return res;

}

void modify(Node\* u, int x) {

access(u); u->splay(); u->key = x; u->rz();

}

};

**2D Geometry**

**Point::**

void makeLine(const Point &t, double &A, double &B, double &C) const { /\* (A, B) - leftside, Ax + By + C > 0 \*/

A = y - t.y; B = t.x - x; C = det(t);}

**Line::**

void load(double A, double B, double C) { /\* (A, B) - leftside, Ax + By + C > 0 \*/

if (fabs(A) < eps) a = Point(0, -C / B);

else a = Point(-C / A, 0);

b = a + Point(B, -A).normalize();

}

Point getProjection(const Point &t) const {

return a + (b - a) \* ((b - a).dot(t - a) / b.dis2(a));

}

**Circle**

Point intersect(const Point &a, const Point &b) const { /\* ensure normally intersected, returned point closer to a \*/

double x = Sqrt(Sqr(r) - Sqr(Line(a, b).disToLine(o)));

return Line(a, b).getProjection(o) - (b - a).normalize() \* x;

}

Point intersect(const Circle &t) const { /\* ensure normally intersected, especially not coincide, returned point on the left side of (o, t.o) \*/

double d = o.dis(t.o);

double x = 0.5 \* ((Sqr(r) - Sqr(t.r)) / d + d);

double h = Sqrt(Sqr(r) - Sqr(x));

return (o \* (d - x) + t.o \* x) / d + (t.o - o).rotate().normalize() \* h;

}

Point tangent(const Point &t) const { /\* return the tagent point from t on the left side of (t, o), ensure not strictly inside the circle \*/

return Circle(t, o).intersect(\*this);

}

Line inTangent(const Circle &t) const { /\* return the tagent in the left direction of (o, t.o), ensure strictly away from each other \*/

Point p = o + (t.o - o) / (r + t.r) \* r;

return Line(p, t.tangent(p));

}

Line outTangent(const Circle &t) const { /\* return the tagent in the ***unknown*** side of (o, t.o), ensure one not normally contains another \*/

if (fabs(r - t.r) < eps) {

Point d = (t.o - o).normalize().rotate() \* r;

return Line(o + d, t.o + d);

}

if (r > t.r + eps) return t.outTangent(\*this);

Point p = Circle(t.o, t.r - r).tangent(o);

Point d = (p - t.o).normalize() \* r;

return Line(o + d, p + d);

}

double interArea(const Circle &t) const {

if (to(t)) return 0;

if (in(t)) return area(); if (t.in(\*this)) return t.area();

double a1 = 2 \* Acos((Sqr(r) + o.dis2(t.o) - Sqr(t.r)) / (2 \* r \* o.dis(t.o)));

double a2 = 2 \* Acos((Sqr(t.r) + o.dis2(t.o) - Sqr(r)) / (2 \* t.r \* o.dis(t.o)));

return fabs(Sqr(r) \* (a1 - sin(a1)) / 2 + Sqr(t.r) \* (a2 - sin(a2)) / 2);

}

double interArea(const Point &t1, const Point &t2) const {

Point tmp[4], cx; int top = 0; tmp[top++] = t1;

if (Line(t1, t2).disToLine(o) < r - eps) {

if (Line(t1, t2).strictlyContains(cx = intersect(t1, t2)))

tmp[top++] = cx;

if (Line(t1, t2).strictlyContains(cx = intersect(t2, t1)))

tmp[top++] = cx;

}

tmp[top++] = t2; double res = 0;

for (int i = 0; i + 1 < top; ++i) {

if (tmp[i].dis(o) > eps && tmp[i + 1].dis(o) > eps &&

(tmp[i].dis(o) > r + eps || tmp[i + 1].dis(o) > r + eps))

res += ((tmp[i + 1] - o) / (tmp[i] - o)).getAng() \* r \* r;

else res += (tmp[i] - o).det(tmp[i + 1] - o);

} return 0.5 \* res;

}

double interArea(Point\* pts, int N) const {

double res = 0;

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

if (i == N - 1) res += interArea(pts[i], pts[0]);

else res += interArea(pts[i], pts[i + 1]);

return fabs(res);

}

**namespace TriangleCenter**

Point circumcenter(Point a, Point b, Point c) {

b = b - a, c = c - a;

double d = b.det(c) \* 2;

Point p = Point(b.norm2(), c.norm2());

return a + Point(p.det(Point(b.y, c.y)), Point(b.x, c.x).det(p))/ d;

}

Point incenter(const Point &a, const Point &b, const Point &c) {

double A = b.dis(c), B = a.dis(c), C = a.dis(b);

return (a \* A + b \* B + c \* C) / (A + B + C);

}

Point orthocenter(const Point &a, const Point &b, const Point &c) {

return a + b + c - circumcenter(a, b, c) \* 2;

}

**namespace HalfPlaneIntersection**

bool cmpRange(const Line &v1, const Line &v2) {

if (fabs(v1.ang - v2.ang) < eps)

return (v2.b - v2.a).det(v1.b - v2.a) > eps;

else return v1.ang < v2.ang;

}

bool judgeIn(const Line &v, const Line &t1, const Line &t2) {

Point cp = t1.intersect(t2);

return (v.b - v.a).det(cp - v.a) < -eps;

}

int getKernel(Line\* \_seg, int N, Point\* pts, double xmin, double ymin, double xmax, double ymax) {

Line \*seg = new Line[N + 5], \*deq = new Line[N + 5];

for (int i = 0; i < N; ++i) seg[i] = \_seg[i];

seg[N++] = Line(Point(xmin, ymin), Point(xmax, ymin));

seg[N++] = Line(Point(xmax, ymin), Point(xmax, ymax));

seg[N++] = Line(Point(xmax, ymax), Point(xmin, ymax));

seg[N++] = Line(Point(xmin, ymax), Point(xmin, ymin));

for (int i = 0; i < N; ++i) seg[i].getAng();

sort(seg, seg + N, cmpRange);

int tN = 1;

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

if (fabs(seg[i].ang - seg[i - 1].ang) > eps) seg[tN++] = seg[i];

N = tN;

int bot = 0, top = 0;

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

while (top - bot >= 2 && judgeIn(seg[i], deq[top - 1], deq[top - 2])) top--;

while (top - bot >= 2 && judgeIn(seg[i], deq[bot], deq[bot + 1])) bot++;

if (top - bot == 1 && deq[bot].det(seg[i]) < eps) return 0;

deq[top++] = seg[i];

while (top - bot >= 3 && judgeIn(deq[bot], deq[top - 1], deq[top - 2])) top--;

while (top - bot >= 3 && judgeIn(deq[top - 1], deq[bot], deq[bot + 1])) bot++;

}

deq[top] = deq[bot]; int num = 0;

for (int i = bot; i < top; ++i) pts[num++] = deq[i].intersect(deq[i + 1]);

delete[] seg; delete[] deq;

return num;

}

**namespace MinCircleCover**

Circle minCircle(Point\* \_pts, int N) {

Point\* pts = new Point[N];

for (int i = 0; i < N; ++i) pts[i] = \_pts[i];

random\_shuffle(pts, pts + N);

Circle res(Point(0, 0), 0);

for (int i = 0; i < N; ++i) if (!res.contains(pts[i])) {

res = Circle(pts[i], 0);

for (int j = 0; j < i; ++j) if (!res.contains(pts[j])) {

res = Circle(pts[i], pts[j]);

for (int k = 0; k < j; ++k) if (!res.contains(pts[k]))

res = Circle(pts[i], pts[j], pts[k]);

}

}

delete[] pts;

return res;

}

**namespace CirclesArea**

struct Event {

double a; int cnt;

Event() {}

Event(double \_a, int \_cnt): a(\_a), cnt(\_cnt) {}

bool operator < (const Event &t) const {

return a < t.a;

}

};

Point polar(double a, double r) {

return Point(cos(a), sin(a)) \* r;

}

void add(Event\* events, int& top, double a, int t) {

events[top++] = Event(a, t);

}

void addPair(Event\* events, int& top, double a, double b) {

add(events, top, a, 1); add(events, top, b, -1);

}

double normalize(double x) {

if (x < -pi) x += 2 \* pi;

if (x >= pi) x -= 2 \* pi;

return x;

}

double calcArea(Circle\* circles, int N, double\* ans) { /\* return area coverd more than K times, stored in ans[K] \*/

/\* when only consider K == 0 or K == N - 1, remove the useless circles first to get significant speedup \*/

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

Event\* events = new Event[N \* 4 + 10];

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

Circle &a = circles[i]; int top = 0, counter = 0;

add(events, top, -pi, 0); add(events, top, pi, 0);

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

Circle &b = circles[j];

if (a == b) {if (i < j) counter++;}

else {

if (a.o.dis(b.o) < fabs(b.r - a.r) + eps) {if (a.r < b.r - eps) counter++;}

else if (a.o.dis(b.o) < a.r + b.r - eps) {

double d2 = a.o.dis2(b.o);

double ag = (b.o - a.o).getAng();

double tg = acos((a.r \* a.r + d2 - b.r \* b.r) / (2 \* a.r \* Sqrt(d2)));

double x = normalize(ag - tg), y = normalize(ag + tg);

if (x < y) addPair(events, top, x, y);

else {addPair(events, top, x, pi); addPair(events, top, -pi, y);}

}

}

}

sort(events, events + top);

counter += events[0].cnt;

for (int j = 1; j < top; ++j) { /\* when only consider K, add ``if (counter == K)``, to get very good speedup \*/

double delta = events[j].a - events[j - 1].a;

if (delta < eps) {

counter += events[j].cnt;

continue;

}

ans[counter] += 0.5 \* Sqr(a.r) \* (delta - sin(delta));

Point p = a.o + polar(events[j - 1].a, a.r);

Point q = a.o + polar(events[j].a, a.r);

ans[counter] += 0.5 \* p.det(q);

counter += events[j].cnt;

}

}

delete[] events;

}

**三维凸包**

#define SIZE(X) (int(X.size()))

#define PI 3.14159265358979323846264338327950288

const double eps = 1e-8;

inline int Sign(double x) {

return x < -eps ? -1 : (x > eps ? 1 : 0);

}

inline double Sqrt(double x) {

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

}

struct Point {

double x, y, z;

Point() {

x = y = z = 0;

}

Point(double x, double y, double z): x(x), y(y), z(z) {

}

bool operator <(const Point &p) const {

return x < p.x || x == p.x && y < p.y || x == p.x && y == p.y && z < p.z;

}

bool operator ==(const Point &p) const {

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

}

Point cross(const Point &p) const {

return Point(y \* p.z - z \* p.y, z \* p.x - x \* p.z, x \* p.y - y \* p.x);

}

double dot(const Point &p) const {

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

}

double norm() {

return dot(\*this);

}

double length() {

return Sqrt(norm());

}

};

int mark[1005][1005];

Point info[1005];

int n, cnt;

double mix(const Point &a, const Point &b, const Point &c) {

return a.dot(b.cross(c));

}

double area(int a, int b, int c) {

return ((info[b] - info[a]).cross(info[c] - info[a])).length();

}

double volume(int a, int b, int c, int d) {

return mix(info[b] - info[a], info[c] - info[a], info[d] - info[a]);

}

struct Face {

int a, b, c;

Face() {}

Face(int a, int b, int c): a(a), b(b), c(c) {}

int &operator [](int k) {

if (k == 0) return a;

if (k == 1) return b;

return c;

}

};

vector <Face> face;

inline void insert(int a, int b, int c) {

face.push\_back(Face(a, b, c));

}

void add(int v) {

vector <Face> tmp;

int a, b, c;

cnt++;

for (int i = 0; i < SIZE(face); i++) {

a = face[i][0];

b = face[i][1];

c = face[i][2];

if (Sign(volume(v, a, b, c)) < 0)

mark[a][b] = mark[b][a] = mark[b][c] = mark[c][b] = mark[c][a] = mark[a][c] = cnt;

else

tmp.push\_back(face[i]);

}

face = tmp;

for (int i = 0; i < SIZE(tmp); i++) {

a = face[i][0];

b = face[i][1];

c = face[i][2];

if (mark[a][b] == cnt) insert(b, a, v);

if (mark[b][c] == cnt) insert(c, b, v);

if (mark[c][a] == cnt) insert(a, c, v);

}

}

int Find() {

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

Point ndir = (info[0] - info[i]).cross(info[1] - info[i]);

if (ndir == Point()) continue;

swap(info[i], info[2]);

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

if (Sign(volume(0, 1, 2, j)) != 0) {

swap(info[j], info[3]);

insert(0, 1, 2);

insert(0, 2, 1);

return 1;

}

}

return 0;

}

int main() {

for (; scanf("%d", &n) == 1; ) {

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

info[i].Input();

sort(info, info + n);

n = unique(info, info + n) - info;

face.clear();

random\_shuffle(info, info + n);

if (Find()) {

memset(mark, 0, sizeof(mark));

cnt = 0;

for (int i = 3; i < n; i++) add(i);

vector<Point> Ndir;

for (int i = 0; i < SIZE(face); ++i) {

Point p = (info[face[i][0]] - info[face[i][1]]).cross(info[face[i][2]] - info[face[i][1]]);

p = p / p.length();

Ndir.push\_back(p);

}

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

int ans = unique(Ndir.begin(), Ndir.end()) - Ndir.begin();

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

} else {

printf("1\n");

}

}

}

**三维凸包求重心**

double calcDist(const Point &p, int a, int b, int c) {

return fabs(mix(info[a] - p, info[b] - p, info[c] - p) / area(a, b, c));

}

//compute the minimal distance of center of any faces

double findDist() {

//compute center of mass

double totalWeight = 0;

Point center(.0, .0, .0);

Point first = info[face[0][0]];

for (int i = 0; i < SIZE(face); ++i) {

Point p = (info[face[i][0]] + info[face[i][1]] + info[face[i][2]] + first) \* .25;

double weight = mix(info[face[i][0]] - first, info[face[i][1]] - first, info[face[i][2]] - first);

totalWeight += weight;

center = center + p \* weight;

}

center = center / totalWeight;

//compute distance

double res = 1e100;

for (int i = 0; i < SIZE(face); ++i) {

res = min(res, calcDist(center, face[i][0], face[i][1], face[i][2]));

}

return res;

}

**最优匹配**

int n,b[MAXN],dx[MAXN],dy[MAXN],slack[MAXN],a[MAXN][MAXN];

bool f[MAXN],g[MAXN];

bool hungary(int x)

{

if (!x)

return(true);

f[x]=true;

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

{

if (g[i])

continue;

int t=dx[x]+dy[i]-a[x][i];

if (!t)

{

g[i]=true;

if (hungary(b[i]))

{

b[i]=x;

return(true);

}

}

else if (t<slack[i])

slack[i]=t;

}

return(false);

}

int main()

{

memset(dx,0,sizeof(dx));

memset(dy,0,sizeof(dy));

scanf("%d",&n);

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

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

{

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

if (a[i][j]>dx[i])

dx[i]=a[i][j];

}

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

{

memset(slack,63,sizeof(slack));

memset(f,0,sizeof(f));

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

while (!hungary(i))

{

int d=inf;

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

if (!g[i] && slack[i]<d)

d=slack[i];

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

{

if (f[i])

dx[i]-=d;

if (g[i])

dy[i]+=d;

}

memset(f,0,sizeof(f));

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

}

}

}

**带花树**

#define maxn 301

vector<int> link[maxn];

int n;

int match[maxn];

int Queue[maxn], head, tail;

int pred[maxn], base[maxn];

bool InQueue[maxn], InBlossom[maxn];

int start, finish;

int newbase;

void push(int u) {

Queue[tail++] = u; InQueue[u] = true;

}

int pop() {

return Queue[head++];

}

int FindCommonAncestor(int u, int v) {

bool InPath[maxn];

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

InPath[i] = 0;

while(true) {

u = base[u];

InPath[u] = true;

if(u == start) break;

u = pred[match[u]];

}

while(true) {

v = base[v];

if(InPath[v]) break;

v = pred[match[v]];

}

return v;

}

void ResetTrace(int u) {

int v;

while(base[u] != newbase) {

v = match[u];

InBlossom[base[u]] = InBlossom[base[v]] = true;

u = pred[v];

if(base[u] != newbase) pred[u] = v;

}

}

void BlossomContract(int u, int v) {

newbase = FindCommonAncestor(u, v);

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

InBlossom[i] = 0;

ResetTrace(u); ResetTrace(v);

if(base[u] != newbase) pred[u] = v;

if(base[v] != newbase) pred[v] = u;

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

if(InBlossom[base[i]]) {

base[i] = newbase;

if(!InQueue[i]) push(i);

}

}

bool FindAugmentingPath(int u) {

bool found = false;

for(int i = 0; i < n; ++i) pred[i] = -1, base[i] = i;

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

InQueue[i] = 0;

start = u; finish = -1;

head = tail = 0;

push(start);

while(head < tail) {

int u = pop();

for(int i = link[u].size() - 1; i >= 0; i--) {

int v = link[u][i];

if(base[u] != base[v] && match[u] != v)

if(v == start || (match[v] >= 0 && pred[match[v]] >= 0))

BlossomContract(u, v);

else if(pred[v] == -1) {

pred[v] = u;

if(match[v] >= 0)

push(match[v]);

else {

finish = v;

return true;

}

}

}

}

return found;

}

void AugmentPath() {

int u, v, w;

u = finish;

while(u >= 0) {

v = pred[u];

w = match[v];

match[v] = u;

match[u] = v;

u = w;

}

}

void FindMaxMatching() {

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

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

if(match[i] == -1)

if(FindAugmentingPath(i))

AugmentPath();

}

**最小树形图**

#define INF 99999999

#define min( a, b ) ( (a)< (b)?(a): (b) )

struct point {

double x;

double y;

}p[200];

int pre[200];//记录该节点的前驱

double graph[200][200], ans;//图数组和结果

bool visit[110], circle[110];//visit记录该点有没有被访问过，circle记录改点是不是在一个圈里

int n, m, root;//顶点数+边数+根节点标号

void dfs( int t ) {//一个深度优先搜索，搜索出一个最大的联通空间

int i;

visit[t]= true;

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

if( !visit[i] && graph[t][i]!= INF )

dfs( i );

}

}

bool check() {//这个函数用来检查最小树形图是否存在，即如果存在，那么一遍dfs后，应该可以遍历到所有的节点

memset( visit, false, sizeof(visit) );

dfs( root );

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

if( !visit[i] )

return false;

}

return true;

}

double dist( int i, int j ) {

return sqrt( (p[i].x-p[j].x)\*(p[i].x-p[j].x)+(p[i].y-p[j].y)\*(p[i].y-p[j].y) );

}

int exist\_circle() {//判断图中是不是存在有向圈

int i;

int j;

root= 1; pre[root]= root;

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

if( !circle[i] && i!= root ) {

pre[i]= i; graph[i][i]= INF;

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

if( !circle[j] && graph[j][i]< graph[pre[i]][i] )

pre[i]= j;

}

}

}//这个for循环负责找出所有非根节点的前驱节点

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

if( circle[i] )

continue;

memset( visit, false, sizeof(visit) );

int j= i;

while( !visit[j] ) {

visit[j]= true;

j= pre[j];

}

if( j== root )

continue;

return j;

}//找圈过程，最后返回值是圈中的一个点

return -1;//如果没有圈，返回-1

}

void update( int t ) {//缩圈之后更新数据

int i;

int j;

ans+= graph[pre[t]][t];

for(i=pre[t]; i!= t; i= pre[i] ) {

ans+= graph[pre[i]][i];

circle[i]= true;

}//首先把圈里的边权全部加起来，并且留出t节点，作为外部接口

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

if( !circle[i] && graph[i][t]!= INF )

graph[i][t]-= graph[pre[t]][t];

//上面这个for循环的作用是对t节点做更新操作，为什么要单独做？你可以看看线面这个循环的跳出条件。

for(j= pre[t]; j!= t; j= pre[j] )

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

if( circle[i] )

continue;

if( graph[i][j]!= INF )

graph[i][t]= min( graph[i][t], graph[i][j]- graph[pre[j]][j] );

graph[t][i]= min( graph[j][i], graph[t][i] );

}

//这个循环对圈中的其他顶点进行更新

}

void solve() {

int j;

memset( circle, false, sizeof(circle) );

while( ( j= exist\_circle() )!= -1 )

update( j );

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

if( j!= root && !circle[j] )

ans+= graph[pre[j]][j];

printf("%.2f\n", ans );

}

int main() {

int i;

while( scanf("%d%d",&n,&m)!= EOF ) {

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

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

graph[i][j]= INF;

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

scanf("%lf%lf",&p[i].x, &p[i].y );

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

int a, b;

scanf("%d%d",&a,&b);

graph[a][b]= dist( a, b );

}

root= 1;

ans= 0;

if( !check() )

printf("poor snoopy\n");

else

solve();

}

return 0;

}

**最大团搜索算法**

Int g[][]为图的邻接矩阵。

MC(V)表示点集V的最大团

令Si={vi, vi+1, ..., vn}, mc[i]表示MC(Si)

倒着算mc[i]，那么显然MC(V)=mc[1]

此外有mc[i]=mc[i+1] or mc[i]=mc[i+1]+1

void init(){

int i, j;

for (i=1; i<=n; ++i) for (j=1; j<=n; ++j) scanf("%d", &g[i][j]);

}

void dfs(int size){

int i, j, k;

if (len[size]==0) {

if (size>ans) {

ans=size; found=true;

}

return;

}

for (k=0; k<len[size] && !found; ++k) {

if (size+len[size]-k<=ans) break;

i=list[size][k];

if (size+mc[i]<=ans) break;

for (j=k+1, len[size+1]=0; j<len[size]; ++j)

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

dfs(size+1);

}

}

void work(){

int i, j;

mc[n]=ans=1;

for (i=n-1; i; --i) {

found=false;

len[1]=0;

for (j=i+1; j<=n; ++j) if (g[i][j]) list[1][len[1]++]=j;

dfs(1);

mc[i]=ans;

}

}

void print(){

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

}

**极大团的计数**

Bool g[][] 为图的邻接矩阵，图点的标号由1至n。

【代码】

void dfs(int size){

int i, j, k, t, cnt, best = 0;

bool bb;

if (ne[size]==ce[size]){

if (ce[size]==0) ++ans;

return;

}

for (t=0, i=1; i<=ne[size]; ++i) {

for (cnt=0, j=ne[size]+1; j<=ce[size]; ++j)

if (!g[list[size][i]][list[size][j]]) ++cnt;

if (t==0 || cnt<best) t=i, best=cnt;

}

if (t && best<=0) return;

for (k=ne[size]+1; k<=ce[size]; ++k) {

if (t>0){

for (i=k; i<=ce[size]; ++i) if (!g[list[size][t]][list[size][i]]) break;

swap(list[size][k], list[size][i]);

}

i=list[size][k];

ne[size+1]=ce[size+1]=0;

for (j=1; j<k; ++j)if (g[i][list[size][j]]) list[size+1][++ne[size+1]]=list[size][j];

for (ce[size+1]=ne[size+1], j=k+1; j<=ce[size]; ++j)

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

dfs(size+1);

++ne[size];

--best;

for (j=k+1, cnt=0; j<=ce[size]; ++j) if (!g[i][list[size][j]]) ++cnt;

if (t==0 || cnt<best) t=k, best=cnt;

if (t && best<=0) break;

}

}

void work(){

int i;

ne[0]=0; ce[0]=0;

for (i=1; i<=n; ++i) list[0][++ce[0]]=i;

ans=0;

dfs(0);

}

**二次剩余**

int power(int a, int b, const int MODE) {

if (b == 0) return 1;

int t = power(a, b / 2, MODE);

t = (t \* t) % MODE;

if (b & 1) t = (t \* a) % MODE;

return t;

}

void calcH(int &t, int &h, const int p) {

int tmp = p - 1;

for (t = 0; (tmp & 1) == 0; tmp /= 2) t++;

h = tmp;

}

// solve equation x^2 mod p = a

bool solve(int a, int p, int &x, int &y) {

srand(19920225);

if (p == 2) {

x = y = 1;

return true;

}

int p2 = p / 2;

int tmp = power(a, p2, p);

if (tmp == p - 1) return false;

if ((p + 1) % 4 == 0) {

x = power(a, (p + 1) / 4, p);

y = p - x;

return true;

} else {

int t, h, b, pb;

calcH(t, h, p);

if (t >= 2) {

do {

b = rand() % (p - 2) + 2;

}

while (power(b, p / 2, p) != p - 1);

pb = power(b, h, p);

}

int s = power(a, h / 2, p);

for (int step = 2; step <= t; step++) {

int ss = (((s \* s) % p) \* a) % p;

for (int i = 0; i < t - step; i++) ss = (ss \* ss) % p;

if (ss + 1 == p) s = (s \* pb) % p;

pb = (pb \* pb) % p;

}

x = (s \* a) % p;

y = p - x;

}

return true;

}

**无向图全局最小割**

#define initSet(n,Arr) for(int i=0;i<n;++i)Arr[i]=i;

#define MAX 1<<30;

int graph[600][600];

// Stoer-Wagner Algorithm

int globalMinCut(int n){

// A is A set for Stoer-Wagner Algorithm

bool\* A=new bool[n];

// V is vertex index

int\* V=new int[n];

int\* W=new int[n];

initSet(n,V);

int best=MAX;

while(n>1){

//the most tightly connected vertex.

int maxj=1;

// initialize set A and other vertex's weight

A[V[0]] = true;

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

A[V[i]]=false;

W[i]=graph[V[0]][V[i]];

if(W[i]>W[maxj])

maxj=i;

}

// find a min-cut

int prev=0,buf=n;

while(--buf){

// add it to A

A[V[maxj]]=true;

if(buf==1){

// update min cut

best=min(best,W[maxj]);

// merge prev and last vertex

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

graph[V[k]][V[prev]]=(graph[V[prev]][V[k]]

+=graph[V[maxj]][V[k]]);

V[maxj]=V[--n];

}

prev=maxj;

maxj=-1;

// update the weights

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

if(!A[V[j]]){

W[j]+=graph[V[prev]][V[j]];

if(maxj<0 || W[j]>W[maxj])

maxj=j;

}

}

}

delete[] A;

delete[] V;

delete[] W;

return best;

}

int main(){

// n - vertex number

// m - edge number

int n,m;

while(scanf("%d %d",&n,&m)==2){

memset(graph,0,sizeof(graph)/sizeof(bool));

// v-w is an edge with c weight

int v,w,c;

while(m--){

scanf("%d %d %d",&v,&w,&c);

graph[v][w]+=c;

graph[w][v]+=c;

}

// output min cut

printf("%d\n",globalMinCut(n));

}

}

**字符串最小表示**

std::string find(std::string s) {

int i, j, k, l;

int N = s.length();

s += s;

for (i = 0, j = 1; j < N; ) {

for (k = 0; k < N && s[i + k] == s[j + k]; k ++);

if (k >= N) break;

if (s[i + k] < s[j + k]) {

j += k + 1;

} else {

l = i + k;

i = j;

j = max(l, j) + 1;

}

}

return s.substr(i, N);

}

**素数判定，大整数分解**

int strong\_pseudo\_primetest(long long n,int base) {

long long n2=n-1,res;

int s; s=0;

while(n2%2==0) n2>>=1,s++;

res=powmod(base,n2,n);

if((res==1)||(res==n-1)) return 1;

s--;

while(s>=0) {

res=mulmod(res,res,n);

if(res==n-1) return 1;

s--;

}

return 0; // n is not a strong pseudo prime

}

int isprime(long long n) {

if(n<2) return 0; if(n<4) return 1;

if(strong\_pseudo\_primetest(n,2)==0) return 0;

if(strong\_pseudo\_primetest(n,3)==0) return 0;

if(n<1373653LL) return 1;

if(strong\_pseudo\_primetest(n,5)==0) return 0;

if(n<25326001LL) return 1;

if(strong\_pseudo\_primetest(n,7)==0) return 0;

if(n==3215031751LL) return 0;

if(n<25000000000LL) return 1;

if(strong\_pseudo\_primetest(n,11)==0) return 0;

if(n<2152302898747LL) return 1;

if(strong\_pseudo\_primetest(n,13)==0) return 0;

if(n<3474749660383LL) return 1;

if(strong\_pseudo\_primetest(n,17)==0) return 0;

if(n<341550071728321LL) return 1;

if(strong\_pseudo\_primetest(n,19)==0) return 0;

if(strong\_pseudo\_primetest(n,23)==0) return 0;

if(strong\_pseudo\_primetest(n,29)==0) return 0;

if(strong\_pseudo\_primetest(n,31)==0) return 0;

if(strong\_pseudo\_primetest(n,37)==0) return 0;

return 1;

}

**Pollard-Rho**

inline LL pollardRho(LL n,LL c) {

//return a non-trival factor of n, otherwise return n

//if (n-1==0) while(1);

LL x,y;x=y=rand()%(n-1)+1;

LL head=1,tail=2;

while (1) {

x=mod\_mul(x,x,n);

x+=c;

if (x>=n) x-=n;

if (x==y) return n;

LL d=\_\_gcd(myAbs(x-y),n);

if (d>1 && d<n) return d;

if ((++head)==tail){

y=x;

tail<<=1;

}

}

}

inline void factor(LL n)//factorize n {

if (n<=1) return;

if (isPrime(n)){

if (N>100) while (1);

fac[N++]=n;

return;

}

//if (n-1==0) while(1);

LL p=n;

while (p>=n) p=pollardRho(n,rand()%(n-1)+1);

factor(n/p);

factor(p);

}

**O(p)求1..p-1的逆元**

void solve (int m) {

int inv[m];

inv[1] = 1;

for (int i = 2; i < m; ++ i) {

inv[i] = ((long long)(m - m / i) \* inv[m % i]) % m;

}

}

**经纬度转向量：(latitude纬度， longitude经度)**

x1=cos(lati1)\*sin(longi1),y1=cos(lati1)\*cos(longi1),z1=sin(lati1);

x2=cos(lati2)\*sin(longi2),y2=cos(lati2)\*cos(longi2),z2=sin(lati2);

**积分表**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  | | | | | |  |
|  | |  | | | | | |  |
|  | |  | | | | | |  |
|  | |  | | | | | |  |
|  | |  | | | | | |  |
|  | | | | | |  | | |
|  | | | | | |  | | |
|  | | | | | |  | | |
|  | | | | | |  | | |
|  | | | | | |  | | |
|  | | | | | |  | | |
|  | | | | | | | | |
|  | | | |  | | | | |
|  | | | |  | | | | |
|  | | | |  | | | | |
|  | | | | | | | | |
|  | | | |  | | | | |
|  | | | |  | | | | |
|  | | | |  | | | | |
|  | | | | | | | | |
|  | | | | |  | | | |
|  | | | | | | | | |
|  | | | | | | | | |
|  | | | | | | | |  |
|  | | | | | | | |
|  |  | | | | | | |  |
|  |  | | | | | | |  |
|  | | |  | | | | | |
|  | |  | | | | |  | |
|  | | |  | | | | | |
|  | | |  | | | | | |
|  | | |  | | | | | |

**有根树的计数**

令

于是，n+1个结点的有根树的总数为

附：

**无根树的计数**

当n是奇数时，则有 种不同的无根树。

当n是偶数时，则有这么多种不同的无根树。

**生成树的计数**

完全图的生成树个数

任意图的生成树个数： 生成树计数行列式tab[i][i] = Di，Di为i的度数tab[i][j] = −k, k为i和j之间的边数。任去一行一列之后的行列式。

**牛顿迭代开根**

double SQRT(n) {

x0 = n + 1;

for (int i = 1; i < 100; ++i) {

x1 = x0 - (x0\*x0 - n) / (2 \* x0);

x0 = x1;

}

Return x0;

}

**中国剩余定理**

设m1,m2,...,mk是两两互素的正整数，对于任意的正整数a1,a2,a3,..,ak

同余方程组：

x≡a1 （mod m1）

x≡a2 （mod m2）

...

x≡ak （mod mk)

必有解，

且解可写为

x≡M1N1a1+MkNkak+....MkNkak (mod m)

其中

m=m1m2m3....mk

Mi=m/mi,(1<=i<=k)

Nj为Mj在模mj下的逆元。

**组合公式**

错排：