**Suffix Array**

#include <bits/stdc++.h>

using namespace std;

#define N 200100

#define LOGN 50

//Suffix Array + LCP

struct suffix {

int index;

int rank[2];

} suffixes[N];

int ind[N], suffixArr[N], lcp[N], invSuff[N];

int cmp(struct suffix a, struct suffix b) {

if (a.rank[0] == b.rank[0])

return a.rank[1] < b.rank[1];

return a.rank[0] < b.rank[0];

}

void buildSuffixArray(string txt, int n) {

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

suffixes[i].index = i;

suffixes[i].rank[0] = txt[i];

suffixes[i].rank[1] = ((i+1) < n)? (txt[i + 1]): -1;

}

sort(suffixes, suffixes+n, cmp);

for (int k = 4; k < 2\*n; k = k\*2) {

int rank = 0;

int prev\_rank = suffixes[0].rank[0];

suffixes[0].rank[0] = rank;

ind[suffixes[0].index] = 0;

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

if (suffixes[i].rank[0] == prev\_rank && suffixes[i].rank[1] == suffixes[i-1].rank[1]) {

prev\_rank = suffixes[i].rank[0];

suffixes[i].rank[0] = rank;

}

else {

prev\_rank = suffixes[i].rank[0];

suffixes[i].rank[0] = ++rank;

}

ind[suffixes[i].index] = i;

}

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

int nextindex = suffixes[i].index + k/2;

suffixes[i].rank[1] = (nextindex < n)? suffixes[ind[nextindex]].rank[0]: -1;

}

sort(suffixes, suffixes+n, cmp);

}

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

suffixArr[i] = suffixes[i].index;

}

void kasai(string txt, int n) {

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

invSuff[suffixArr[i]] = i;

int k = 0;

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

if (invSuff[i] == n-1) {

k = 0;

continue;

}

int j = suffixArr[invSuff[i]+1];

while (i+k<n && j+k<n && txt[i+k]==txt[j+k] && txt[i+k] != ' ' && txt[j+k] != ' ')

k++;

lcp[invSuff[i]] = k;

if (k > 0) k--;

}

}

//RMQ

int rmq[N][LOGN], LOG[N];

void processRMQ() {

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

LOG[i] = ret += (i > 1 && !(i&(i-1)));

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

rmq[i][0] = i;

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

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

if (lcp[rmq[i][j - 1]] < lcp[rmq[i + (1 << (j - 1))][j - 1]])

rmq[i][j] = rmq[i][j - 1];

else

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

}

int RMQquery(int i, int j) {

int k = LOG[j - i + 1];

return min(lcp[rmq[i][k]], lcp[rmq[j - (1 << k) + 1][k]]);

}

**Persistent Segtree**

#include <bits/stdc++.h>

using namespace std;

#define N 200100

struct Node {

int acum;

Node \*left, \*right;

Node(int acum, Node \*left, Node \*right) : acum(acum), left(left), right(right) {}

Node\* update(int ini, int fim, int pos, int val);

};

Node \*null = new Node(0, NULL, NULL);

Node\* Node::update(int ini, int fim, int pos, int val) {

if (ini > pos || fim < pos) return this;

if (ini == fim)

return new Node(this->acum + val, null, null);

int meio = (ini + fim) / 2;

return new Node(this->acum + val, this->left->update(ini, meio, pos, val), this->right->update(meio+1, fim, pos, val));

}

int persQuery(int ini, int fim, int start, int end, Node \*atual) {

if (start > fim || end < ini)

return 0;

if (ini >= start && fim <= end)

return atual->acum;

int meio = (ini + fim) / 2;

return persQuery(ini, meio, start, end, atual->left) + persQuery(meio+1, fim, start, end, atual->right);

}

Node \*root[N];

**Treap**

#include <bits/stdc++.h>

using namespace std;

typedef struct item \* pitem;

struct item {

int prior, value, cnt, acum;

bool rev;

pitem l, r;

item(int v) {

prior = rand(); value = v; cnt = 1;

l = r = NULL;

rev = false; acum = 0;

}

};

int cnt (pitem it) {

return it ? it->cnt : 0;

}

void upd\_cnt (pitem it) {

if (it)

it->cnt = cnt(it->l) + cnt(it->r) + 1;

}

void push (pitem it) {

if (it && it->rev) {

it->rev = false;

swap (it->l, it->r);

if (it->l) it->l->rev ^= true;

if (it->r) it->r->rev ^= true;

}

if (it && it->acum) {

if (it->l) it->l->acum += it->acum;

if (it->r) it->r->acum += it->acum;

it->value += it->acum;

it->value %= 26;

it->acum = 0;

}

}

void merge (pitem & t, pitem l, pitem r) {

push (l);

push (r);

if (!l || !r)

t = l ? l : r;

else if (l->prior > r->prior)

merge (l->r, l->r, r), t = l;

else

merge (r->l, l, r->l), t = r;

upd\_cnt (t);

}

void split (pitem t, pitem & l, pitem & r, int key, int add = 0) {

if (!t)

return void( l = r = 0 );

push (t);

int cur\_key = add + cnt(t->l);

if (key <= cur\_key)

split (t->l, l, t->l, key, add), r = t;

else

split (t->r, t->r, r, key, add + 1 + cnt(t->l)), l = t;

upd\_cnt (t);

}

void reverse (pitem t, int l, int r) {

pitem t1, t2, t3;

split (t, t1, t2, l);

split (t2, t2, t3, r-l+1);

t2->rev ^= true;

merge (t, t1, t2);

merge (t, t, t3);

}

void output (pitem t) {

if (!t) return;

push (t);

output (t->l);

printf ("%c", t->value + 'a');

output (t->r);

}

void increase(pitem t, int l, int r) {

pitem t1, t2, t3;

split (t, t1, t2, l);

split (t2, t2, t3, r-l+1);

t2->acum++;

merge (t, t1, t2);

merge (t, t, t3);

}

void change(pitem t, int i, int j, int k, int l) {

pitem t1, t2, t3, t4, t5;

split (t, t1, t2, i);

split (t2, t2, t3, j-i+1);

split (t3, t3, t4, k-j-1);

split (t4, t4, t5, l-k+1);

merge(t, t1, t4);

merge(t, t, t3);

merge(t, t, t2);

merge(t, t, t5);

}

int t, n, a, b, c, d;

string s;

int main() {

scanf("%d", &t);

for (int tc = 0; tc < t; tc++) {

cin >> s >> n;

pitem root = NULL;

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

merge(root, root, new item(s[i] - 'a'));

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

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

change(root, a-1, b-1, c-1, d-1);

reverse(root, a-1, a-1+d-c);

reverse(root, a-1+d-b, d-1);

increase(root, a-1, a-1+d-c);

increase(root, a-1+d-b, d-1);

}

output(root);

printf("\n");

}

return 0;

}

**Centroid Dec.**

#include <bits/stdc++.h>

using namespace std;

#define N 100100

vector < int > graph[N];

struct CentroidDec {

int root, visited[N], siz[N], layer[N], parent[N];

vector < int > centroidTree[N];

void init() {

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

visited[i] = siz[i] = layer[i] = parent[i] = 0;

centroidTree[i].clear();

}

}

int getCentroid(int u, int p = 0) {

siz[u] = 1;

for (int v : graph[u])

if (v != p && !visited[v]) {

getCentroid(v, u);

siz[u] += siz[v];

}

if (p) return 0;

int par = 0, aux = u, nxt = 0;

while (1) {

for (int v : graph[aux])

if (!visited[v] && v != par && siz[v] > siz[u] / 2)

nxt = v;

if (!nxt) return aux;

else { par = aux; aux = nxt; nxt = 0; }

}

}

void buildTree(int u = 0) {

if (u == 0) {

u = root = getCentroid(1);

visited[u] = 1; layer[u] = 1;

}

for (int v : graph[u])

if (!visited[v]) {

int x = getCentroid(v);

visited[x] = 1; layer[x] = layer[u] + 1; parent[x] = u;

centroidTree[u].push\_back(x);

centroidTree[x].push\_back(u);

buildTree(x);

}

}

} centroid;

**Aho-Corasick**

#include <bits/stdc++.h>

using namespace std;

#define N 110

#define ALP\_SIZ 26

struct Automaton {

int nodes, fail[N], mask[N];

int child[N][ALP\_SIZ];

queue < int > q;

int newnode() {

mask[nodes] = 0;

memset(child[nodes], 0, sizeof(child[nodes]));

return nodes++;

}

void clear() {

nodes = 0;

newnode();

}

void insert(string s, int id) {

int atual = 0;

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

int c = s[i] - 'a';

if (!child[atual][c])

child[atual][c] = newnode();

atual = child[atual][c];

}

mask[atual] |= 1 << id;

}

void getFails() {

for (int i = 0; i < ALP\_SIZ; i++)

if (child[0][i])

fail[child[0][i]] = 0, q.push(child[0][i]);

while (!q.empty()) {

int u = q.front(); q.pop();

for (int i = 0; i < ALP\_SIZ; i++) {

int v = child[u][i];

if (!v) { child[u][i] = child[fail[u]][i]; continue; }

q.push(v);

int j = fail[u];

while (j && !child[j][i]) j = fail[j];

fail[v] = child[j][i], mask[v] |= mask[fail[v]];

}

}

}

} AC;

**Complete FFT**

#include <bits/stdc++.h>

using namespace std;

#define MOD 7340033

#define ll long long

#define PI acos(-1)

typedef unsigned uint;

template<typename Double>

struct FFT {

typedef complex<Double> doublex;

vector<doublex> fft(vector<doublex> y, bool invert = false) {

const int N = y.size();

assert(N == (N&-N));

vector<int> rev(N);

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

rev[i] = (rev[i >> 1] >> 1) | (i & 1 ? N >> 1 : 0);

if (rev[i] < i) swap(y[i], y[rev[i]]);

}

vector<doublex> rootni(N / 2);

for (int n = 2; n <= N; n \*= 2) {

const doublex rootn = polar((Double)1.0, (invert ? +1.0 : -1.0)\*2.0\*Double(PI) / n);

rootni[0] = 1.0;

for (int i = 1; i < n / 2; i++) rootni[i] = rootni[i - 1] \* rootn;

for (int left = 0; left != N; left += n) {

const int mid = left + n / 2;

for (int i = 0; i < n / 2; i++) {

const doublex temp = rootni[i] \* y[mid + i];

y[mid + i] = y[left + i] - temp;

y[left + i] += temp;

}

}

}

if (invert) for (auto &v : y) v /= (doublex)N;

return move(y);

}

uint nextpow2(uint v) {

if (!v) return 1;

v = 2 \* v - 1;

uint u = 1;

while (v) {

u <<= 1;

v >>= 1;

}

return u;

}

vector<doublex> convolution(vector<doublex> a, vector<doublex> b) {

const int n = max((int)a.size() + (int)b.size() - 1, 0), n2 = nextpow2(n);

a.resize(n2);

b.resize(n2);

vector<doublex> fa = fft(move(a)), fb = fft(move(b)), &fc = fa;

for (int i = 0; i < n2; i++) fc[i] = fc[i] \* fb[i];

vector<doublex> c = fft(move(fc), true);

c.resize(n);

return move(c);

}

vector<int> polymult(const vector<int> &a, const vector<int> &b, int mod, int npart = 2) {

if (a.empty() || b.empty()) return{};

const int div = pow(mod + 1, 1.0 / npart) + 4, n = nextpow2(a.size() + b.size() - 1);

vector<vector<doublex>> splita(npart, vector<doublex>(n)), splitb(npart, vector<doublex>(n));

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

int v = a[i];

for (int j = 0; j < npart; j++) splita[j][i] = v%div, v /= div;

}

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

int v = b[i];

for (int j = 0; j < npart; j++) splitb[j][i] = v%div, v /= div;

}

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

splita[i] = fft(move(splita[i]));

splitb[i] = fft(move(splitb[i]));

}

vector<int> result(a.size() + b.size() - 1);

vector<doublex> res(n);

for (int i = 0; i < npart; i++) for (int j = 0; j < npart; j++) {

int multby = 1;

for (int k = 0; k < i + j; k++) multby = (ll)multby\*div%mod;

for (int k = 0; k < n; k++) res[k] = splita[i][k] \* splitb[j][k];

res = fft(move(res), true);

for (int k = 0; k < (int)result.size(); k++) result[k] = (result[k] + (ll)round(res[k].real()) % mod\*multby) % mod;

}

return move(result);

}

vector<doublex> cyclic\_convolution(vector<doublex> a, vector<doublex> b) {

assert(a.size() == b.size());

int n = a.size();

vector<doublex> c = convolution(move(a), move(b));

for (int i = c.size() - 1; i >= n; --i)c[i - n] += c[i];

c.resize(n);

return move(c);

}

};

**3D Geometry Example**

#include <bits/stdc++.h>

using namespace std;

#define EPS 1e-9

int t, a, b, c;

struct point {

double x, y, z;

point() { x = y = z = 0.0; }

point(double x, double y, double z) : x(x), y(y), z(z) {}

} shipa[4], shipb[4];

double dist(point a, point b) {

return sqrt((a.x - b.x)\*(a.x - b.x) + (a.y - b.y)\*(a.y - b.y) + (a.z - b.z)\*(a.z - b.z));

}

struct vec {

double x, y, z;

vec(double x, double y, double z) : x(x), y(y), z(z) {}

};

vec toVec(point a, point b) {

return vec(b.x - a.x, b.y - a.y, b.z - a.z);

}

vec scale(vec v, double s) {

return vec(v.x \* s, v.y \* s, v.z \* s);

}

point translate(point p, vec v) {

return point(p.x + v.x, p.y + v.y, p.z + v.z);

}

vec add(vec p, vec v) {

return vec(p.x + v.x, p.y + v.y, p.z + v.z);

}

double dot(vec a, vec b) { return a.x \* b.x + a.y \* b.y + a.z \* b.z; }

double norm\_sq(vec v) { return v.x \* v.x + v.y \* v.y + v.z \* v.z; }

double distToLine(point p, point a, point b) {

vec ap = toVec(a, p), ab = toVec(a, b);

double u = dot(ap, ab) / norm\_sq(ab);

point c = translate(a, scale(ab, u));

return dist(p, c);

}

double distToLineS(point p, point a, point b) {

vec ap = toVec(a, p), ab = toVec(a, b);

double u = dot(ap, ab) / norm\_sq(ab);

if (u < 0.0) return dist(p, a);

if (u > 1.0) return dist(p, b);

return distToLine(p, a, b);

}

vec norm(vec a, vec b) {

return vec(a.y \* b.z - a.z \* b.y, a.z \* b.x - a.x \* b.z, a.x \* b.y - a.y \* b.x);

}

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

return sqrt(norm\_sq(norm(toVec(a, b), toVec(a, c)))) / 2.0;

}

double inside(point p, point a, point b, point c) {

vec ab = toVec(a, b), ac = toVec(a, c);

vec n = norm(ab, ac);

n = scale(n, 1.0 / sqrt(norm\_sq(n)));

double d = -a.x\*n.x - a.y\*n.y - a.z\*n.z;

double dst = dot(n, vec(p.x, p.y, p.z)) + d;

point pj = translate(p, scale(n, -dot(toVec(a, p), n)));

double area1 = area(a, b, c), area2 = 0.0;

area2 += area(a, b, pj);

area2 += area(b, c, pj);

area2 += area(c, a, pj);

if (fabs(area1 - area2) < EPS) return fabs(dst);

return -1.0;

}

double distseg(point p1, point p2, point p3, point p4) {

vec u = toVec(p1, p2);

vec v = toVec(p3, p4);

vec w = toVec(p3, p1);

double a = dot(u, u);

double b = dot(u, v);

double c = dot(v, v);

double d = dot(u, w);

double e = dot(v, w);

double D = a\*c - b\*b;

double sc, sN, sD = D;

double tc, tN, tD = D;

if (D < EPS) {

sN = 0.0; sD = 1.0; tN = e; tD = c;

}

else {

sN = (b\*e - c\*d);

tN = (a\*e - b\*d);

if (sN < 0.0) {

sN = 0.0;

tN = e;

tD = c;

}

else if (sN > sD) {

sN = sD;

tN = e + b;

tD = c;

}

}

if (tN < 0.0) {

tN = 0.0;

if (-d < 0.0) sN = 0.0;

else if (-d > a) sN = sD;

else {

sN = -d;

sD = a;

}

}

else if (tN > tD) {

tN = tD;

if ((-d + b) < 0.0) sN = 0;

else if ((-d + b) > a) sN = sD;

else {

sN = (-d + b);

sD = a;

}

}

sc = (abs(sN) < EPS ? 0.0 : sN / sD);

tc = (abs(tN) < EPS ? 0.0 : tN / tD);

vec dP = add(w, scale(u, sc));

dP = add(dP, scale(v, -tc));

return sqrt(norm\_sq(dP));

}

int main() {

scanf("%d", &t);

while (t--) {

for (int i = 0; i < 4; i++) {

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

shipa[i] = point(a, b, c);

}

for (int i = 0; i < 4; i++) {

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

shipb[i] = point(a, b, c);

}

double ans = DBL\_MAX;

for (int i = 0; i < 4; i++) {

for (int j = 0; j < 4; j++)

for (int k = j+1; k < 4; k++)

ans = min(ans, distToLineS(shipa[i], shipb[j], shipb[k]));

for (int j = 0; j < 4; j++)

for (int k = j+1; k < 4; k++)

for (int l = k+1; l < 4; l++) {

double d = inside(shipa[i], shipb[j], shipb[k], shipb[l]);

if (d > 0.0) ans = min(ans, d);

}

}

for (int i = 0; i < 4; i++) {

for (int j = 0; j < 4; j++)

for (int k = j+1; k < 4; k++)

ans = min(ans, distToLineS(shipb[i], shipa[j], shipa[k]));

for (int j = 0; j < 4; j++)

for (int k = j+1; k < 4; k++)

for (int l = k+1; l < 4; l++) {

double d = inside(shipb[i], shipa[j], shipa[k], shipa[l]);

if (d > 0.0) ans = min(ans, d);

}

}

for (int i = 0; i < 4; i++)

for (int j = i+1; j < 4; j++)

for (int k = 0; k < 4; k++)

for (int l = k+1; l < 4; l++)

ans = min(ans, distseg(shipa[i], shipa[j], shipb[k], shipb[l]));

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

}

return 0;

}