[Graph 4](#_Toc217999331)

[Base Element 4](#_Toc217999332)

[Connected Component, Cut Edge, Cut Vertex 4](#_Toc217999333)

[Cut Vertex, Biconnected-O( E ) 4](#_Toc217999334)

[Cut Edge-O( E ) 5](#_Toc217999335)

[Strognly Connected Component-O( E ) 5](#_Toc217999336)

[Shortest Path 6](#_Toc217999337)

[Dijestra-O( E Log( E ) ) 6](#_Toc217999338)

[Bellman Ford-O( E V ) 6](#_Toc217999339)

[Johnson-[Pretreatment O (V E),Intermediate processing O (V E log V),After-treatment O (V^2)] 7](#_Toc217999340)

[K-th Shortest Path-O (k + E + V log V) 7](#_Toc217999341)

[Floyd-Warshal-O( V ^ 3 ) 8](#_Toc217999342)

[Counting Paths-O(t \* V ^3) 8](#_Toc217999343)

[Spanning Tree 9](#_Toc217999344)

[Prim-O( E log V ) 9](#_Toc217999345)

[Kruskal-O( E log E ) 9](#_Toc217999346)

[Network Flow 9](#_Toc217999347)

[Bipartite Matching, Vertex Cover-O( E V ) 9](#_Toc217999348)

[Ford Fulkerson-O(E \* MAXFLOW) 11](#_Toc217999349)

[Dinic-O( V ^ 3 ) 11](#_Toc217999350)

[Minimum Cut in undirected graph (Stoer-Wagner)-O( V \* MAXFLOW ) 12](#_Toc217999351)

[Weighted Matching-O (V ^ 3) 12](#_Toc217999352)

[Weighted Matching-O (EV^2) 13](#_Toc217999353)

[Min Cost Max Flow-O( E log E \* MAXFLOW ) 15](#_Toc217999354)

[Tours 16](#_Toc217999355)

[Directed Euler Tour-O( E ) 16](#_Toc217999356)

[Undirected Euler Tour-O( E ) 17](#_Toc217999357)

[Shortest Hamilton road-O( ( V ^ 2 ) \* ( 2 ^ V ) ) 17](#_Toc217999358)

[Tree Algorithms 17](#_Toc217999359)

[Offline Smallest common ancestor-O( E \* A( V ) ) 17](#_Toc217999360)

[Other 18](#_Toc217999361)

[Topological Sort-O( E ) 18](#_Toc217999362)

[Geometry 18](#_Toc217999363)

[Base Geometry(typedef's, ccw, cross, dot) 18](#_Toc217999364)

[Angle(vectorAngle, angle, rotate, angleInclusion) 19](#_Toc217999365)

[Line(Segment), Point Intersection Detection 20](#_Toc217999366)

[Line(Segment), Point Distane and Intersection and Other 20](#_Toc217999367)

[Line Creation 21](#_Toc217999368)

[General Polygon 22](#_Toc217999369)

[Area of Polygon-O( V ) 22](#_Toc217999370)

[Point Inclusion-O( V ) 22](#_Toc217999371)

[Shrink Polygon-O( V ) 22](#_Toc217999372)

[Triangle (make\_triangle, triangle\_contains, triAreaFromMedians) 22](#_Toc217999373)

[Convex Polygon 23](#_Toc217999374)

[Convex Hull-O( V log V ) 23](#_Toc217999375)

[Cutting of Convex Polygon-O( V ) 24](#_Toc217999376)

[Intersection of Convex Polygons-O( V1 + V2 ) 25](#_Toc217999377)

[Diameter of Convex Polygon (Most Distant Points)-O( V ) 26](#_Toc217999378)

[Point Inclusion Convex-O( log V ) 26](#_Toc217999379)

[Circle (incircle, three\_point\_circle, circle\_circle\_intersect, line and circle, greatCircle) 27](#_Toc217999380)

[Data Structures 30](#_Toc217999381)

[Point Location-[pretreatment O( V ^ 2 ), query: O( log V )] 30](#_Toc217999382)

[Kd-Tree-[Insert: O( log N ), Search: O( log N )] 31](#_Toc217999383)

[Additional Topics 31](#_Toc217999384)

[Closest Pair-[Generally: O( N log N ), Wortst Case(Vertically Line) : O(N ^ 2)] 31](#_Toc217999385)

[Base Element of 3D Geomety 31](#_Toc217999386)

[3D Rotation and Other 32](#_Toc217999387)

[Data Structures 33](#_Toc217999388)

[AVL Tree-[All O( log N ) ] 33](#_Toc217999389)

[Set-Disjoint-[find, union O(log(n)), make\_set O(1)] 34](#_Toc217999390)

[Interval Tree-[Insert: O( log N ), Find: O( log N )] 34](#_Toc217999391)

[Binary Indexed Tree-[Insert: O( log V ), Find: O( log V )] 35](#_Toc217999392)

[RMQ-[pretreatment: O( N log N ), query: O( 1 )] 36](#_Toc217999393)

[Segment Tree-[Insert: O( log N ), Find: O( log N )] 37](#_Toc217999394)

[Strings 38](#_Toc217999395)

[String Searching 38](#_Toc217999396)

[Shift And-O( L1 + L2 ) 38](#_Toc217999397)

[Knuth Morris Pratt-O( L1 + L2 ) 38](#_Toc217999398)

[String-Related Data Structures 38](#_Toc217999399)

[Aho-Corasick-O(Patterns\_length + sigma(Texts\_length)) 38](#_Toc217999400)

[Trie-O(n) 39](#_Toc217999401)

[Suffix Tree 39](#_Toc217999402)

[Suffix Array-[Matching naive O (m log n),Mamber-Myers's O (m + log n)] 42](#_Toc217999403)

[Additional Topics 44](#_Toc217999404)

[Longest Palinrome-O(n) 44](#_Toc217999405)

[Longest increase substring-O (n log n) 44](#_Toc217999406)

[Longest common subsequence-O ((n+r) log n) 44](#_Toc217999407)

[Math 45](#_Toc217999408)

[Big Integer 45](#_Toc217999409)

[Rational number 47](#_Toc217999410)

[Rational number 47](#_Toc217999411)

[Stern-Brocot 48](#_Toc217999412)

[Number Theory 48](#_Toc217999413)

[GCD, Extended-GCD-O( log ( a + b ) ) 48](#_Toc217999414)

[Modular Inverse-O( log( a ) ) 48](#_Toc217999415)

[Linear Congruences Equations-O(n \* log(n)) 48](#_Toc217999416)

[Linear Diophantine Equation Solver-O( log N ) 49](#_Toc217999417)

[Modular Linear Equation Solver-O(log n) 49](#_Toc217999418)

[Mod Power-O( log k ) 49](#_Toc217999419)

[Jacobi Sign-O(loga) 49](#_Toc217999420)

[Sqrt Mode-O(log k) 50](#_Toc217999421)

[Euler Phi Function-O( sqrt( N ) ) 50](#_Toc217999422)

[Prime Detection 50](#_Toc217999423)

[Algebra 52](#_Toc217999424)

[Matrix Operations 52](#_Toc217999425)

[LU disassembly( Linear Equition Solver )-O( N ^ 3 ) 53](#_Toc217999426)

[Fast Fourier transform-O( N log N ) 56](#_Toc217999427)

[Simplex Method 56](#_Toc217999428)

# Graph

## Base Element

typedef int Weight;

struct Edge {

int src and dst;

Weight weight;

Edge (int src, int dst and Weight weight):

src (src), dst (dst), weight (weight) {}

};

bool operator < (const Edge &e and const Edge &f) {

return e.weight! = f.weight? e.weight > f.weight: //!! INVERSE!!

e.src! = f.src? e.src < f.src: e.dst < f.dst;

}

typedef vector<Edge> Edges;

typedef vector<Edges> Graph;

typedef vector<Weight> Array;

typedef vector<Array> Matrix;

## Connected Component, Cut Edge, Cut Vertex

### Cut Vertex, Biconnected-O( E )

struct UndirectionalCompare {

bool operator () (const Edge& e, const Edge& f) const {

if (min (e.src , e.dst)! = min (f.src , f.dst))

return min (e.src , e.dst) < min (f.src , f.dst);

return max (e.src , e.dst) < max (f.src , f.dst);

}

};

typedef set<Edge, UndirectionalCompare> Edgeset;

void visit (const Graph &g, int v, int u,

vector<int>& art, vector<Edgeset>& bcomp,

stack<Edge>& S, vector<int>& num, vector<int>& low, int& time) {

low[v] = num[v] = ++time;

FOR (e, g[v]) {

int w = e->dst;

if (num[w] < num[v]) S.push (\*e); // for bcomps

if (num[w] == 0) {

visit (g, w, v, art, bcomp, S, num, low, time);

low[v] = min (low[v], low[w]);

if ((num[v] == 1 && num[w]! = 2) || // for arts

(num[v]! = 1 && low[w] >= num[v])) art.push\_back(v);

if (low[w] >= num[v]) { // for bcomps

bcomp.push\_back (Edgeset());

while (1) {

Edge f = S.top (); S.pop ();

bcomp.back () .insert (f);

if (f.src == v && f.dst == w) break;

}

}

} else low[v] = min (low[v], num[w]);

}

}

void articulationPoint (const Graph& g,

vector<int>& art, vector<Edgeset>& bcomp) {

const int n = g.size ();

vector<int> low (n), num (n);

stack<Edge> S;

REP(u, n) if (num[u] == 0) {

int time = 0;

visit (g, u, -1, art, bcomp, S, num, low, time);

}

}

### Cut Edge-O( E )

#define NODES 100

int n, graph[NODES][NODES], dfi[NODES], t, low[NODES];

bool mark[NODES];

vector<pair<int, int> > cut;

void dfsMatrix(int v, int p)

{

mark[v] = true;

dfi[v] = t++;

low[v] = dfi[v];

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

if (i != p && graph[v][i] && mark[i] && dfi[i] < low[v])

low[v] = dfi[i];

else if (graph[v][i] && !mark[i])

{

dfsMatrix(i, v);

if (low[i] < low[v])

low[v] = low[i];

if (low[i] == dfi[i])

cut.push\_back(make\_pair(v, i));

}

}

void cutEdgeMatrix()

{

t = 0;

memset(mark, false, n \* sizeof(bool));

cut = vector<pair<int, int> >();

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

if (!mark[i])

dfsMatrix(i, -1);

}

### Strognly Connected Component-O( E )

const int max\_n = 20000 + 10 ;

vector < int > g[max\_n] ;

vector < int > bg[max\_n];

vector < int > new\_graph[max\_n] ;

int cc\_index[max\_n];

bool mark[max\_n];

int order[max\_n] , size = 0;

int n , m ;

void dfs ( int x ){

mark[x] = 1;

for ( int i=0 ; i<g[x].size() ; i++ ){

if ( mark[g[x][i]] == 0 )

dfs ( g[x][i] ) ;

}

order[n-++size] = x ;

}

void bdfs ( int x , int index ){

cc\_index[x] = index ;

for ( int i=0 ; i<bg[x].size() ; i++ )

if ( cc\_index[bg[x][i]] == -1 )

bdfs ( bg[x][i] , index ) ;

}

Usage:

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

g[i].clear() , bg[i].clear() , new\_graph[i].clear();

while ( m -- ){

int a , b ;

-- a , -- b ;

g[a].push\_back ( b ) ;

bg[b].push\_back ( a ) ;

}

memset ( mark , 0 , sizeof mark ) ;

size = 0 ;

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

if ( mark[i] == 0 )

dfs ( i ) ;

memset ( cc\_index , -1 , sizeof cc\_index ) ;

int cc = 0 ; // number of scc

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

int index = order[i] ;

if ( cc\_index[index] == -1 ){

bdfs ( index , cc++ );

}

}

## Shortest Path

### Dijestra-O( E Log( E ) )

void shortestPath (const Graph &g and int s,

vector<Weight> &dist and vector<int> &prev) {

int n = g.size ();

dist.assign (n, INF); dist [s] = 0;

prev.assign (n, -1);

priority\_queue<Edge> Q; // “e < f” <=> “e.weight > f.weight”

for (Q.push (Edge (- 2, s, 0)); ! Q.empty (); ) {

Edge e = Q.front (); Q.pop ();

if (prev[e.dst]! = -1) continue;

prev[e.dst] = e.src;

FOR (f , g[e.dst]) {

if (dist[f->dst] > e.weight+f->weight) {

dist[f->dst] = e.weight+f->weight;

Q.push (Edge (f->src, f->dst , e.weight+f->weight));

}

}

}

}

vector<int> buildPath (const vector<int> &prev , int t) {

vector<int> path;

for (int u = t; u >= 0; u = prev[u])

path.push\_back (u);

reverse (path.begin (), path.end ());

return path;

}

### Bellman Ford-O( E V )

bool shortestPath (const Graph g and int s,

vector<Weight> &dist) vector<int> &prev) {

int n = g.size ();

dist.assign (n , INF+INF); dist[0] = 0;

prev.assign (n, -2);

bool negative\_cycle = false;

REP (k , n) REP (i, n) FOR (e , g [i]) {

if (dist[e->dst] > dist[e->src] + e->weight) {

dist[e->dst] = dist[e->src] + e->weight;

prev[e->dst] = e->src;

if (k == n-1) {

dist[e->dst] = - INF;

negative\_cycle = true;

}

}

}

return !negative\_cycle;

}

vector<int> buildPath (const vector<int> &prev , int t) {

vector<int> path;

for (int u = t; u >= 0; u = prev[u])

path.push\_back(u);

reverse (path.begin(), path.end());

return path;

}

### Johnson-[Pretreatment O (V E),Intermediate processing O (V E log V),After-treatment O (V^2)]

bool shortestPath (const Graph &g,

Matrix &dist and vector<vector<int> > &prev) {

int n = g.size ();

Array h(n+1);

REP (k , n) REP (i, n) FOR (e , g [i]) {

if (h[e->dst] > h[e->src] + e->weight) {

h[e->dst] = h[e->src] + e->weight;

if (k == n-1) return false; // negative cycle

}

}

dist.assign (n , Array (n , INF));

prev.assign (n , vector<int> (n, -2));

REP (s , n) {

priority\_queue<Edge> Q;

Q.push (Edge (s , s, 0));

while (! Q.empty()) {

Edge e = Q.top(); Q.pop();

if (prev[s][e.dst]! = -2) continue;

prev[s][e.dst] = e.src;

FOR (f , g [e.dst]) {

if (dist[s][f->dst] > e.weight + f->weight) {

dist[s][f->dst] = e.weight + f->weight;

Q.push (Edge (f->src, f->dst , e.weight + f->weight));

}

}

}

REP (u , n) dist[s][u] += h[u] - h[s];

}

}

vector<int> buildPath (const vector< vector<int> >& prev, int s , int t) {

vector<int> path;

for (int u = t; u >= 0; u = prev[s][u])

path.push\_back (u);

reverse (ALL (path));

return path;

}

### K-th Shortest Path-O (k + E + V log V)

Weight k\_shortestPath (const Graph &g, int s, int t , int k) {

const int n = g.size ();

Graph h (n); // make reverse graph

REP (u , n) FOR (e , g [u])

h[e->dst].push\_back (Edge (e->dst, e->src , e->weight));

vector<Weight> d (n , INF); d[t] = 0; // make potential

vector<int> p (n, -1); // using backward dijkstra

priority\_queue<Edge> Q; Q.push (Edge (t , t, 0));

while (! Q.empty ()) {

Edge e = Q.top (); Q.pop ();

if (p[e.dst] >= 0) continue;

p[e.dst] = e.src;

FOR (f , h[e.dst]) if (d[f->dst] > e.weight + f->weight) {

d[f->dst] = e.weight + f->weight;

Q.push (Edge (f->src, f->dst , e.weight + f->weight));

}

}

int l = 0; // forward dijkstra-like method

priority\_queue<Edge> R; R.push (Edge (- 1, s, 0));

while (! R.empty ()) {

Edge e = R.top (); R.pop ();

if (e.dst == t && ++l == k) return e.weight + d [s];

FOR (f , g [e.dst])

R.push (Edge (f->src, f->dst , e.weight+f->weight-d [f->src] +d [f->dst]));

}

return -1; // not found

}

// simpler method

Weight k\_shortestPath (const Graph &g, int s, int t , int k) {

const int n = g.size ();

vector<Weight> dist[n];

priority\_queue<Edge> Q; Q.push (Edge (- 1, s, 0));

while (! Q.empty ()) {

Edge e = Q.top (); Q.top ();

if (dist[e.dst].size () >= k) continue;

dist[e.dst].push\_back(e.weight);

FOR (f , g[e.dst]) Q.push (Edge (f->src, f->dst and f->weight+e.weight));

}

}

### Floyd-Warshal-O( V ^ 3 )

void shortestPath (const Matrix &g,

Matrix &dist , vector< vector<int> > &inter) {

int n = g.size ();

dist = g;

inter.assign (n , vector<int> (n, - 1));

REP (k , n) REP (i, n) REP (j , n) {

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

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

inter[i][j] = k;

}

}

}

void buildPath (const vector< vector<int> > &inter,

int s, int t , vector<int> &path) {

int u = inter[s][t];

if (u < 0) path.push\_back (s);

else buildPath (inter, s, u , path), buildPath (inter, u, s , path);

}

vector<int> buildPath (

const vector< vector<int> > &inter, int s , int t) {

vector<int> path;

buildPath (inter, s, t, path);

path.push\_back (t);

return path;

}

### Counting Paths-O(t \* V ^3)

A[t][i][j] be the number of paths from *i* to *j* with exactly *t* steps.

*Among all the paths from i to j that takes exactly k steps, which one is of the shortest length?*

int BG=1000000000;

// to avoid overflow in addition, do not use 2^31-1

A[1][i][j] = r[i][j]; p[1][i][j]=j;

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

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

{

A[t][i][j]=BG; p[t][i][j]=-1;

for(k=0; k<n; k++) if(A[1][i][k]<BG && A[t-1][k][j]<BG)

if(A[1][i][k]+A[t-1][k][j] < A[t][i][j])

{

A[t][i][j] = A[1][i][k]+A[t-1][k][j];

p[t][i][j] = k;

}

}

// to output the best path from a to b with t steps:

void output(int a, int b, int t)

{

while(t)

{

cout<<a<<" ";

a = p[t][a][b];

t--;

}

cout<<b<<endl;

}

## Spanning Tree

### Prim-O( E log V )

pair<Weight , Edges> minimumSpanningTree (const Graph &g, int r = 0) {

int n = g.size();

Edges T;

Weight total = 0;

vector<bool> visited(n);

priority\_queue<Edge> Q;

Q.push (Edge (- 1, r, 0));

while (!Q.empty()) {

Edge e = Q.top(); Q.pop ();

if (visited[e.dst]) continue;

T.push\_back (e);

total += e.weight;

visited[e.dst] = true;

FOR (f , g[e.dst]) if (!visited[f->dst]) Q.push (\*f);

}

return pair<Weight , Edges> (total , T);

}

### Kruskal-O( E log E )

pair<Weight , Edges> minimumSpanningForest (const Graph &g) {

int n = g.size ();

UnionFind uf (n);

priority\_queue<Edge> Q;

REP (u , n) FOR (e , g[u]) if (u < e->dst) Q.push (\*e);

Weight total = 0;

Edges F;

while (F.size() < n-1 && !Q.empty ()) {

Edge e = Q.top(); Q.pop();

if (uf.unionSet (e.src , e.dst)) {

F.push\_back (e);

total += e.weight;

}

}

return pair<Weight , Edges> (total , F);

}

## Network Flow

### Bipartite Matching, Vertex Cover-O( E V )

#define M 128

#define N 128

bool graph[M][N];

bool seen[N];

int matchL[M], matchR[N];

int n, m;

bool bpm( int u )

{

for( int v = 0; v < n; v++ ) if( graph[u][v] )

{

if( seen[v] ) continue;

seen[v] = true;

if( matchR[v] < 0 || bpm( matchR[v] ) )

{

matchL[u] = v;

matchR[v] = u;

return true;

}

}

return false;

}

vector<int> vertex\_cover()

{

// Comment : Vertices on the left side (n side) are labeled like this : m+i where i is the index

set<int> s, t, um; // um = UnMarked

vector<int> vc;

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

if(matchL[i]==-1)

s.insert(i), um.insert(i);

while( um.size() )

{

int v = \*(um.begin());

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

if( graph[v][i] && matchL[v]!=i)

{

t.insert(i);

if( s.find(matchR[i]) == s.end())

s.insert(matchR[i]), um.insert(matchR[i]);

}

um.erase(v);

}

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

if( s.find(i) == s.end() )

vc.push\_back(i);

for(set<int>::iterator i = t.begin(); i != t.end(); i++)

vc.push\_back((\*i) + m);

return vc;

}

int main()

{

// Read input and populate graph[][]

// Set m, n

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

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

int cnt = 0;

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

{

memset( seen, 0, sizeof( seen ) );

if( bpm( i ) ) cnt++;

}

vector<int> vc = vertex\_cover();

// cnt contains the number of happy pigeons

// matchL[i] contains the hole of pigeon i or -1 if pigeon i is unhappy

// matchR[j] contains the pigeon in hole j or -1 if hole j is empty

// vc contains the Vertex Cover

return 0;

}

### Ford Fulkerson-O(E \* MAXFLOW)

int cap[NN][NN];

int fnet[NN][NN];

int q[NN], qf, qb, prev[NN];

int fordFulkerson( int n, int s, int t )

{

memset( fnet, 0, sizeof( fnet ) );

int flow = 0;

while( true )

{

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

qf = qb = 0;

prev[q[qb++] = s] = -2;

while( qb > qf && prev[t] == -1 )

for( int u = q[qf++], v = 0; v < n; v++ )

if( prev[v] == -1 && fnet[u][v] - fnet[v][u] < cap[u][v] )

prev[q[qb++] = v] = u;

if( prev[t] == -1 ) break;

int bot = 0x7FFFFFFF;

for( int v = t, u = prev[v]; u >= 0; v = u, u = prev[v] )

bot <?= cap[u][v] - fnet[u][v] + fnet[v][u];

for( int v = t, u = prev[v]; u >= 0; v = u, u = prev[v] )

fnet[u][v] += bot;

flow += bot;

}

return flow;

}

int main()

{

memset( cap, 0, sizeof( cap ) );

int numVertices = 100;

// ... fill up cap with existing edges.

// if the edge u->v has capacity 6, set cap[u][v] = 6.

cout << fordFulkerson( numVertices, s, t ) << endl;

return 0;

}

### Dinic-O( V ^ 3 )

/\* prev contains the minimum cut. If prev[v] == -1, then v is not

\* reachable from s; otherwise, it is reachable.

\* RUNNING TIME: O(n^3)

\*/

#define NN 1024

int cap[NN][NN], deg[NN], adj[NN][NN];

int q[NN], prev[NN];

int dinic( int n, int s, int t )

{

int flow = 0;

while( true )

{

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

int qf = 0, qb = 0;

prev[q[qb++] = s] = -2;

while( qb > qf && prev[t] == -1 )

for( int u = q[qf++], i = 0, v; i < deg[u]; i++ )

if( prev[v = adj[u][i]] == -1 && cap[u][v] )

prev[q[qb++] = v] = u;

if( prev[t] == -1 ) break;

for( int z = 0; z < n; z++ ) if( cap[z][t] && prev[z] != -1 )

{

int bot = cap[z][t];

for( int v = z, u = prev[v]; u >= 0; v = u, u = prev[v] )

bot <?= cap[u][v];

if( !bot ) continue;

cap[z][t] -= bot;

cap[t][z] += bot;

for( int v = z, u = prev[v]; u >= 0; v = u, u = prev[v] )

{

cap[u][v] -= bot;

cap[v][u] += bot;

}

flow += bot;

}

}

return flow;

}

int main()

{

memset( cap, 0, sizeof( cap ) );

int n, s, t, m;

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

while( m-- )

{

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

cap[u][v] = c;

}

memset( deg, 0, sizeof( deg ) );

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

for( int v = 0; v < n; v++ ) if( cap[u][v] || cap[v][u] )

adj[u][deg[u]++] = v;

printf( "%d\n", dinic( n, s, t ) );

return 0;

}

### Minimum Cut in undirected graph (Stoer-Wagner)-O( V \* MAXFLOW )

Weight minimumCut (const Graph &g) {

int n = g.size ();

vector< vector<Weight> > h (n , vector<Weight> (n)); // make adj. matrix

REP (u , n) FOR (e , g[u]) h[e->src][e->dst] += e->weight;

vector<int> V (n); REP (u , n) V[u] = u;

Weight cut = INF;

for (int m = n; m > 1; m--) {

vector<Weight> ws (m , 0);

int u, v;

Weight w;

REP (k, m) {

u = v; v = max\_element (ws.begin (), ws.end ()) - ws.begin ();

w = ws[v]; ws[v] = -1;

REP (i , m) if (ws[i] >= 0) ws[i] += h[V[v]][V[i]];

}

REP (i , m) {

h[V[i]][V[u]] += h[V[i]][V[v]];

h[V[u]][V[i]] += h[V[v]][V[i]];

}

V.erase (V.begin() + v);

cut = min (cut , w);

}

return cut;

}

### Weighted Matching-O (V ^ 3)

#define inf 1000000000

#define NN 200

int n, m, weight[NN][NN];

int x[NN], y[NN];// X[i] = j means i-th vertex of left matched to j-th vertex of rigth

int hungarian() {

int p, q;

vector<int> fx(n, inf), fy(n, 0);

memset(x, -1, sizeof x);

memset(y, -1, sizeof y);

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

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

fx[i] = max(fx[i], weight[i][j]);

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

vector<int> t(n, -1), s(n+1, i);

for (p = q = 0; p <= q && x[i] < 0; ++p)

for (int k = s[p], j = 0; j < n && x[i] < 0; ++j)

if (fx[k] + fy[j] == weight[k][j] && t[j] < 0) {

s[++q] = y[j], t[j] = k;

if (s[q] < 0)

for (p = j; p >= 0; j = p)

y[j] = k = t[j], p = x[k], x[k] = j;

}

if (x[i] < 0) {

int d = inf;

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

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

if (t[j] < 0) d = min(d, fx [s[k]] + fy[j] - weight[s[k]][j]);

for (int j = 0; j < n; ++j) fy[j] += (t[j] < 0? 0: d);

for (int k = 0; k <= q; ++k) fx[s[k]] -= d;

} else ++i;

}

int ret = 0;

for (int i = 0; i < n; ++i) ret += weight[i][x[i]];

return ret;

}

### Weighted Matching-O (EV^2)

#pragma comment(linker, "/STACK:50000000") //if MAXSIZE > 100 then use it

#define MAXSIZE 100

#define INF 1000000000

int match[MAXSIZE];

bool dfs(int v, int n, int adj[MAXSIZE][MAXSIZE], int mark[], int s[], int t[])

{

int i;

s[v] = 1;

mark[v] = 1;

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

if (adj[v][i])

if (t[i] = 1, match[i] == -1 || (!mark[match[i]] &&

dfs(match[i], n, adj, mark, s, t)))

return match[i] = v, true;

return false;

}

bool matching(int n, int adj[MAXSIZE][MAXSIZE], int s[], int t[], int sa[])

{

int i;

int max = 0;

int mark[MAXSIZE];

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

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

if (!sa[i] && !mark[i] && dfs(i, n, adj, mark, s, t))

{

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

sa[i] = 1;

}

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

if (!sa[i])

return false;

return true;

}

void weighted(int n, int m, int weight[MAXSIZE][MAXSIZE])

{

int i, j;

int size = 0;

int sa[MAXSIZE], s[MAXSIZE], t[MAXSIZE];

int cover[2][MAXSIZE];

int adj[MAXSIZE][MAXSIZE];

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

memset(sa, 0, sizeof(sa));

memset(adj, 0, sizeof(adj));

if (m > n)

n = m;

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

{

int index = 0;

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

if (weight[i][index] < weight[i][j])

index = j;

cover[1][i] = 0;

cover[0][i] = weight[i][index];

adj[i][index] = 1;

}

while (!matching(n, adj, s, t, sa))

{

memset(s, 0, sizeof(s));

memset(t, 0, sizeof(t));

matching(n, adj, s, t, sa);

int min = INF;

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

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

if (s[i] && !t[j] && !adj[i][j])

if (cover[0][i] + cover[1][j] - weight[i][j] < min)

min = cover[0][i] + cover[1][j] - weight[i][j];

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

if (s[i])

cover[0][i] -= min;

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

if (t[i])

cover[1][i] += min;

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

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

if ((s[i] && !t[j]) || (!s[i] && t[j]))

if ((cover[0][i] + cover[1][j]-weight[i][j])==0)

adj[i][j] = 1;

else

adj[i][j] = 0;

}

}

main()

{

int i, j, tn;

int n, m, sum;

int weight[MAXSIZE][MAXSIZE];

for (cin >> tn; tn--;)

{

memset(weight, 0, sizeof weight);

cin >> n >> m;

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

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

cin >> weight[i][j];

weighted(n, m, weight);

sum = 0;

for (i = 0; i < max(n, m); i++)

{

if (i < m && match[i] < n)

{

cout << match[i] << ' ' << i << endl;

sum += weight[match[i]][i];

}

}

cout << sum << endl;

}

return 0;

}

### Min Cost Max Flow-O( E log E \* MAXFLOW )

/\* fnet contains the flow network. Careful: both fnet[u][v] and

\* fnet[v][u] could be positive. Take the difference.

\* COMPLEXITY: O(m\*log(m)\*flow <? n\*m\*log(m)\*fcost)

\*\*/

#include <iostream>

using namespace std;

#define NN 1024

int cap[NN][NN];

int cost[NN][NN];

int fnet[NN][NN], adj[NN][NN], deg[NN];

int par[NN], d[NN], q[NN], inq[NN], qs;

int pi[NN];

#define CLR(a, x) memset( a, x, sizeof( a ) )

#define Inf (INT\_MAX/2)

#define BUBL { \

t = q[i]; q[i] = q[j]; q[j] = t; \

t = inq[q[i]]; inq[q[i]] = inq[q[j]]; inq[q[j]] = t; }

#define Pot(u,v) (d[u] + pi[u] - pi[v])

bool dijkstra( int n, int s, int t )

{

CLR( d, 0x3F );

CLR( par, -1 );

CLR( inq, -1 );

d[s] = qs = 0;

inq[q[qs++] = s] = 0;

par[s] = n;

while( qs )

{

int u = q[0]; inq[u] = -1;

q[0] = q[--qs];

if( qs ) inq[q[0]] = 0;

for( int i = 0, j = 2\*i + 1, t; j < qs; i = j, j = 2\*i + 1 )

{

if( j + 1 < qs && d[q[j + 1]] < d[q[j]] ) j++;

if( d[q[j]] >= d[q[i]] ) break;

BUBL;

}

for( int k = 0, v = adj[u][k]; k < deg[u]; v = adj[u][++k] )

{

if( fnet[v][u] && d[v] > Pot(u,v) - cost[v][u] )

d[v] = Pot(u,v) - cost[v][par[v] = u];

if( fnet[u][v] < cap[u][v] && d[v] > Pot(u,v) + cost[u][v] )

d[v] = Pot(u,v) + cost[par[v] = u][v];

if( par[v] == u )

{

if( inq[v] < 0 ) { inq[q[qs] = v] = qs; qs++; }

for( int i = inq[v], j = ( i - 1 )/2, t;

d[q[i]] < d[q[j]]; i = j, j = ( i - 1 )/2 )

BUBL;

}

}

}

for( int i = 0; i < n; i++ ) if( pi[i] < Inf ) pi[i] += d[i];

return par[t] >= 0;

}

#undef Pot

int mcmf4( int n, int s, int t, int &fcost )

{

CLR( deg, 0 );

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

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

if( cap[i][j] || cap[j][i] ) adj[i][deg[i]++] = j;

CLR( fnet, 0 );

CLR( pi, 0 );

int flow = fcost = 0;

while( dijkstra( n, s, t ) )

{

int bot = INT\_MAX;

for( int v = t, u = par[v]; v != s; u = par[v = u] )

bot <?= fnet[v][u] ? fnet[v][u] : ( cap[u][v] - fnet[u][v] );

for( int v = t, u = par[v]; v != s; u = par[v = u] )

if( fnet[v][u] ) { fnet[v][u] -= bot; fcost -= bot \* cost[v][u]; }

else { fnet[u][v] += bot; fcost += bot \* cost[u][v]; }

flow += bot;

}

return flow;

}

int main()

{

int numV;

cin >> numV;

memset( cap, 0, sizeof( cap ) );

int m, a, b, c, cp;

int s, t;

cin >> m;

cin >> s >> t;

// fill up cap with existing capacities.

// if the edge u->v has capacity 6, set cap[u][v] = 6.

// for each cap[u][v] > 0, set cost[u][v] to the

// cost per unit of flow along the edge i->v

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

cin >> a >> b >> cp >> c;

cost[a][b] = c; // cost[b][a] = c;

cap[a][b] = cp; // cap[b][a] = cp;

}

int fcost;

int flow = mcmf3( numV, s, t, fcost );

cout << "flow: " << flow << endl;

cout << "cost: " << fcost << endl;

return 0;

}

## Tours

### Directed Euler Tour-O( E )

void visit (Graph& g, int a , vector<int>& path) {

while (!g[a].empty()){

int b = g[a].back().dst;

g[a].pop\_back();

visit (g, b, path);

}

path.push\_back (a);

}

bool eulerPath (Graph g, int s , vector<int> &path) {

int n = g.size(), m = 0;

vector<int> deg (n);

REP (u , n) {

m += g[u].size();

FOR (e , g[u]) --deg[e->dst]; // in-deg

deg[u] += g[u].size(); // out-deg

}

int k = n - count (ALL (deg), 0);

if (k == 0 || (k == 2 && deg[s] == 1)) {

path.clear();

visit (g, s , path);

reverse (ALL (path));

return path.size () == m + 1;

}

return false;

}

### Undirected Euler Tour-O( E )

void visit(const Graph &g, vector< vector<int> > &adj, int s, vector<int> &path) {

FOR (e , g[s]) if (adj[e->src][e->dst]) {

--adj[e->src][e->dst];

--adj[e->dst][e->src];

visit(g, adj, e->dst , path);

}

path.push\_back(s);

}

bool eulerPath (const Graph &g, int s , vector<int> &path) {

int n = g.size();

int odd = 0, m = 0;

REP (i, n) {

if (g[i].size() % 2 == 1) ++odd;

m += g[i].size();

}

m/= 2;

if (odd == 0 || (odd == 2 && g[s].size() % 2 == 0)) {

vector< vector<int> > adj (n , vector<int> (n));

REP (u , n) FOR (e , g[u]) ++adj[e->src][e->dst];

path.clear ();

visit (g, adj, s, path);

reverse (ALL (path));

return path.size() == m + 1;

}

return false;

}

### Shortest Hamilton road-O( ( V ^ 2 ) \* ( 2 ^ V ) )

const int M = 20;

Weight best[1<<M][M];

int prev[1<<M][M];

void buildPath (int S, int i, vector<int> &path) {

if (!S) return;

buildPath (S ^ (1<<i), prev[S][i], path);

path.push\_back(i);

}

Weight shortestHamiltonPath (Matrix w, int s, vector<int> &path) {

int N = 1 << n;

REP (S, N) REP (i, n) best[S][i] = INF;

best[1<<s][s] = 0;

REP (S, N) REP (i, n) if (S & (1<<i)) REP (j, n)

if (best[S | (1<<j)][j] > best[S][i] + w[i][j])

best[S | (1<<j)][j] = best[S][i] + w[i][j],

prev[S | (1<<j)][j] = i;

int t = min\_element(best[N-1], best[N-1] + n) - best[N-1];

path.clear(); buildPath (N-1, t, path);

return best[N-1][t];

}

## Tree Algorithms

### Offline Smallest common ancestor-O( E \* A( V ) )

struct Query {

int u, v, w;

Query (int u, int v): u(u), v(v), w(- 1) {}

};

void visit (const Graph &g, int u, int w,

vector<Query> &qs, vector<int> &color,

vector<int> &ancestor, UnionFind &uf) {

ancestor[uf.root(u)] = u;

FOR (e, g[u]) if (e->dst! = w) {

visit (g, e->dst, u, qs, color, ancestor, uf);

uf.unionSet (e->src, e->dst);

ancestor[uf.root(u)] = u;

}

color[u] = 1;

FOR (q, qs) {

int w = (q->v == u ? q->u : q->u == u ? q->v : -1);

if (w >= 0 && color[w]) q->w = ancestor[uf.root(w)];

}

}

void leastCommonAncestor (const Graph &g, int r, vector<Query> &qs) {

UnionFind uf(g.size());

vector<int> color(g.size()), ancestor(g.size());

visit (g, r, -1, qs, color, ancestor, uf);

}

## Other

### Topological Sort-O( E )

bool visit (const Graph &g, int v, vector<int> &order, vector<int> &color) {

color[v] = 1;

FOR (e, g[v]) {

if (color[e->dst] == 2) continue;

if (color[e->dst] == 1) return false;

if (!visit(g, e->dst, order, color)) return false;

}

order.push\_back(v); color[v] = 2;

return true;

}

bool topologicalSort(const Graph &g, vector<int> &order) {

int n = g.size();

vector<int> color(n);

REP (u, n) if (!color[u] && !visit(g, u, order, color))

return false;

reverse (ALL (order));

return true;

}

# Geometry

## Base Geometry(typedef's, ccw, cross, dot)

const double EPS = 1e-8;

const double INF = 1e12;

typedef complex<double> P; //point

namespace std {

bool operator < (const P& a, const P& b) {

return real(a) != real(b) ? real(a) < real(b) : imag(a) < imag(b);

}

}

double cross (const P& a, const P& b) {

return imag (conj(a) \*b);

}

double dot (const P& a, const P& b) {

return real (conj(a) \*b);

}

struct L: public vector<P> { //line

L (const P &a, const P &b) {

push\_back(a); push\_back(b);

}

};

typedef vector<P> G;

typedef vector<P> polygon;

struct C { // circle

P p; double r;

C (const P &p, double r) : p(p), r(r) {}

};

int ccw (P a, P b, P c) {

b -= a; c -= a;

if (cross (b, c) > 0) return +1; // counter clockwise

if (cross (b, c) < 0) return -1; // clockwise

if (dot (b, c) < 0) return +2; // c--a--b ON line

if (norm (b) < norm (c)) return -2; // a--b--c ON line

return 0;

}

double cross(double ax, double ay, double bx, double by)

{

return ax \* by – ay \* bx;

}

double dot(double ax, double ay, double bx, double by)

{

return ax \* bx + ay \* by;

}

double ang(double ax, double ay, double bx, double by)

{

return atan2(cross(ax,ay,bx,by), dot(ax,ay,bx,by));

}

## Angle(vectorAngle, angle, rotate, angleInclusion)

inline double vectorAngle(double x, double y)

{

double r = atan2(y, x);

/\*

(-Pi < r <= Pi): the following 'if' statement should be excluded

(0 <= r < 2 \* Pi): the following 'if' statement should be included

\*/

if (less(r, 0))

r += 2 \* PI;

return r;

}

inline double angle(double x1, double y1, double x2, double y2, double x3, double y3)

/\*

angle (x1,y1)-(x2,y2)-(x3,y3)

\*/

{

double r = vectorAngle(x3 - x2, y3 - y2) - vectorAngle(x1 - x2, y1 - y2);

/\*

(-Pi < r <= Pi): the following 'if' statement should be excluded

(0 <= r < 2 \* Pi): the following 'if' statement should be included

\*/

if (less(r, 0))

r += 2 \* PI;

return r;

}

void rotate(double &x, double &y, double angle)

/\*

rotate (x, y) around (0, 0)

angle should be in radians

\*/

{

double tx = x \* cos(angle) - y \* sin(angle);

double ty = x \* sin(angle) + y \* cos(angle);

x = tx;

y = ty;

}

bool angleInclusion(double xo, double yo, double xa, double ya, double xb, double yb, double xp, double yp)

/\*

checks to see if we encounter point 'p' while sweeping angle 'a-o-b' from 'a' to 'b'

\*/

{

double ab = multiply(xo, yo, xa, ya, xb, yb);

double ap = multiply(xo, yo, xa, ya, xp, yp);

double bp = multiply(xo, yo, xb, yb, xp, yp);

return (greaterEqual(ab, 0) && greaterEqual(ap, 0) && lessEqual(bp, 0)) ||

(less(ab, 0) && (greaterEqual(ap, 0) || lessEqual(bp, 0)));

}

## Line(Segment), Point Intersection Detection

bool intersectLL (const L &l, const L &m) {

return abs (cross(l[1] - l[0], m[1] - m[0])) > EPS || // non-parallel

abs (cross(l[1] - l[0], m[0] - l[0])) < EPS; // same line

}

bool intersectLS (const L &l, const L &s) {

return cross (l[1] - l[0], s[0] - l[0]) \* // s [0] is left of l

cross (l[1] - l[0], s[1] - l[0]) < EPS; // s [1] is right of l

}

bool intersectLP (const L &l, const P &p) {

return abs (cross (l[1] – p, l[0] - p)) < EPS;

}

bool intersectSS (const L &s, const L &t) {

return ccw (s[0], s[1], t[0]) \* ccw (s[0], s[1], t[1]) <= 0 &&

ccw (t[0], t[1], s[0]) \* ccw (t[0], t[1], s[1]) <= 0;

}

bool intersectSP (const L &s, const P &p) {

return abs (s[0] - p) + abs (s[1] - p) - abs (s[1] - s[0]) < EPS; // triangle inequality

}

## Line(Segment), Point Distane and Intersection and Other

P projection (const L &l, const P &p) {

double t = dot (p-l[0], l[0] - l[1]) / norm (l [0] - l [1]);

return l[0] + t \* (l[0] - l[1]);

}

P reflection (const L &l, const P &p) {

return p + 2 \* (projection (l, p) - p);

}

double distancePP(double x1, double y1, double x2, double y2)

{

return sqrt((x2 - x1) \* (x2 - x1) + (y2 - y1) \* (y2 - y1));

}

double distanceLP (const L &l, const P &p) {

return abs (p - projection (l, p));

}

double distanceLL (const L &l, const L &m) {

return intersectLL (l, m) ? 0 : distanceLP (l, m[0]);

}

double distanceLS (const L &l, const L &s) {

if (intersectLS (l, s)) return 0;

return min (distanceLP (l, s[0]), distanceLP (l, s[1]));

}

double distanceSP (const L &s, const P &p) {

const P r = projection (s, p);

if (intersectSP (s, r)) return abs (r - p);

return min (abs (s[0] - p), abs (s[1] - p));

}

double distanceSS (const L &s, const L &t) {

if (intersectSS (s, t)) return 0;

return min (min (distanceSP (s, t[0]), distanceSP (s, t[1])),

min (distanceSP (t, s[0]), distanceSP (t, s[1])));

}

P crosspoint (const L &l, const L &m) {

double A = cross (l[1] - l[0], m[1] - m[0]);

double B = cross (l[1] - l[0], l[1] - m[0]);

if (abs (A) < EPS && abs (B) < EPS) return m[0]; // same line

//if (abs (A) < EPS) assert (false); //!!! PRECONDITION NOT SATISFIED!!!//parallel

return m[0] + B/A \* (m[1] - m[0]);

}

int segmentsIntersectionPoint2(double x1, double y1, double x2, double y2, double x3, double y3, double x4, double y4, double &x5, double &y5)

/\*

return values:

7 important bits of the result value are:

bit 0: 1 : segments are intersecting at (x5, y5)

0 : segments are not intersecting

bit 1: 1 : containing lines are parallel

0 : containing lines are not parallel

bit 2: 1 : containing lines are coincident

0 : containing lines are not coincident

bit 3: 1 : containing lines are intersecting on

the extension of segment 1-2 at (x5, y5)

0 : containing lines are not intersecting

on the extension of segment 1-2

bit 4: 1 : containing lines are intersecting on

the extension of segment 2-1 at (x5, y5)

0 : containing lines are not intersecting

on the extension of segment 2-1

bit 5: 1 : containing lines are intersecting on

the extension of segment 3-4 at (x5, y5)

0 : containing lines are not intersecting

on the extension of segment 3-4

bit 6: 1 : containing lines are intersecting on

the extension of segment 4-3 at (x5, y5)

0 : containing lines are not intersecting

on the extension of segment 4-3

\*/

{

double rnum = (y1 - y3) \* (x4 - x3) - (x1 - x3) \* (y4 - y3);

double den = (x2 - x1) \* (y4 - y3) - (y2 - y1) \* (x4 - x3);

double snum = (y1 - y3) \* (x2 - x1) - (x1 - x3) \* (y2 - y1);

int res = 0;

if (equal(den, 0))

{

res |= 2;

if (equal(rnum, 0)) res |= 4;

return res;

}

double r = rnum / den;

double s = snum / den;

x5 = x1 + r \* (x2 - x1);

y5 = y1 + r \* (y2 - y1);

if (greater(r, 1)) res |= 8;

if (less(r, 0)) res |= 16;

if (greater(s, 1)) res |= 32;

if (less(s, 0)) res |= 64;

return (res ? res : 1);

}

## Line Creation

inline void segmentToLine(double x1, double y1, double x2, double y2,

double &a, double &b, double &c)

{

a = y2 - y1;

b = x1 - x2;

c = -y1 \* b - x1 \* a;

}

inline void lineThroughPoint(double x, double y, double m, double &a, double &b, double &c)

{

a = -m;

b = 1;

c = m \* x - y;

}

## General Polygon

### Area of Polygon-O( V )

number area2 (const polygon& P) {

number A = 0;

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

A += cross (curr (P, i), next (P, i));

return A;

}

### Point Inclusion-O( V )

#define curr (P, i) P[i]

#define next (P, i) P[(i+1) % P.size()]

enum {OUT, ON, IN};

int contains (const polygon& P, const point& p) {

bool in = false;

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

point a = curr (P, i) – p, b = next (P, i) - p;

if (imag (a) > imag (b)) swap (a, b);

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

if (cross (a, b) < 0) in = !in;

if (cross (a, b) == 0 && dot (a, b) <= 0) return ON;

}

return in ? IN : OUT;

}

### Shrink Polygon-O( V )

polygon shrink\_polygon(const polygon &P, number len) { // use sufficiently small len

polygon res;

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

point a = prev(P, i), b = curr(P, i), c = next(P, i);

point u = (b - a) / abs(b - a);

double th = arg((c - b) / u) \* 0.5;

res.push\_back (b + u & point (-sin(th), cos(th)) \* len / cos(th));

}

return res;

}

### Triangle (make\_triangle, triangle\_contains, triAreaFromMedians)

typedef vector<point> triangle;

triangle make\_triangle (const point& a, const point& b, const point& c) {

triangle ret (3);

ret[0] = a; ret[1] = b; ret[2] = c;

return ret;

}

bool triangle\_contains (const triangle& tri, const point& p) {

return ccw (tri[0], tri[1], p) >= 0 &&

ccw (tri[1], tri[2], p) >= 0 &&

ccw (tri[2], tri[0], p) >= 0;

}

/\* Given the lengths of the 3 medians of a triangle,

\* returns the triangle's area, or -1 if it impossible.

\* WARNING: Deal with the case of zero area carefully.

\* #include <math.h>

\* FIELD TESTING:

\* - Valladolid 10347: Medians

\*\*/

double triAreaFromMedians( double ma, double mb, double mc )

{

double x = 0.5 \* ( ma + mb + mc );

double a = x \* ( x - ma ) \* ( x - mb ) \* ( x - mc );

if( a < 0.0 ) return -1.0;

else return sqrt( a ) \* 4.0 / 3.0;

}

## Convex Polygon

### Convex Hull-O( V log V )

#define curr (P, i) P[(i) % P.size()]

#define next (P, i) P[(i+1) % P.size()]

#define prev (P, i) P[(i+P.size() - 1) % P.size ()]

bool isconvex (const polygon &P) {

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

if (ccw (prev (P, i), curr (P, i), next (P, i)) > 0) return false;

return true;

}

vector<point> convex\_hull (vector<point> ps) {

int n = ps.size (), k = 0;

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

vector<point> ch (2\*n);

for (int i = 0; i < n; ch[k++] = ps[i++]) // lower-hull

while (k >= 2 && ccw (ch[k-2], ch[k-1], ps[i]) <= 0) --k;

for (int i = n-2, t = k+1; i >= 0; ch[k++] = ps[i--]) // upper-hull

while (k >= t && ccw (ch[k-2], ch[k-1], ps[i]) <= 0) --k;

ch.resize (k-1);

return ch;

}

//Another Vesion of Convex Hull

struct CompCHGS

{

int m;

double \*x, \*y;

CompCHGS(int m, double \*x, double \*y): m(m), x(x), y(y) {}

bool operator()(const int i1, const int i2) const

{

return (greater((x[i1] - x[m]) \* (y[i2] - y[m]), (x[i2] - x[m]) \* (y[i1] - y[m])) ||

/\*

'greater' : counter-clockwise

'less' : clockwise

\*/

(equal((x[i1] - x[m]) \* (y[i2] - y[m]), (x[i2] - x[m]) \* (y[i1] - y[m])) &&

greaterEqual(x[i1], min(x[i2], x[m])) && lessEqual(x[i1], max(x[i2], x[m])) &&

greaterEqual(y[i1], min(y[i2], y[m])) && lessEqual(y[i1], max(y[i2], y[m]))));

}

};

void convexHullGS(int n, double x[], double y[], vector<int> &ch)

/\*

Convex Hull finding, Graham Scan algorithm, O(n\*log(n))

\*/

{

/\*

note the array size

\*/

int sorted[1000], i, m = 0;

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

{

sorted[i] = i;

if (less(y[i], y[m]) || (equal(y[i], y[m]) && less(x[i], x[m])))

m = i;

}

sorted[0] = m;

sorted[m] = 0;

/\*

the angles 1 to n - 1 should be sorted in ascending order. for each two equal angles,

the angle with enpoint nearer to origin should become first.

\*/

sort(sorted + 1, sorted + n, CompCHGS(m, x, y));

/\*

after the sort, order of the last equal angles should be reversed.

\*/

for (i = n - 1; i > 1 &&

equal((x[sorted[i]] - x[m]) \* (y[sorted[i - 1]] - y[m]),

(x[sorted[i - 1]] - x[m]) \* (y[sorted[i]] - y[m])); i--);

if (i > 1) reverse(sorted + i, sorted + n);

ch = vector<int>();

ch.push\_back(sorted[0]);

if (n > 1)

{

ch.push\_back(sorted[1]);

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

{

while (ch.size() > 1 &&

greaterEqual(multiply(x[sorted[i]], y[sorted[i]], x[ch.back()], y[ch.back()],

x[ch[ch.size() - 2]], y[ch[ch.size() - 2]]), 0))

/\*

'greater' : counter-clockwise, maximum number of points

'greaterEqual' : counter-clockwise, minimum number of points

'less' : clockwise, maximum number of points

'lessEqual' : clockwise, minimum number of points

\*/

ch.pop\_back();

ch.push\_back(sorted[i]);

}

/\*

maximum number of points : exclude the following 'if' statement

minimum number of points : include the following 'if' statement

\*/

if (ch.size() > 2 && equal(multiply(x[ch.front()], y[ch.front()], x[ch.back()],

y[ch.back()], x[ch[ch.size() - 2]], y[ch[ch.size() - 2]]), 0))

ch.pop\_back();

}

}

### Cutting of Convex Polygon-O( V )

//! CAUTION! As for number above rational number

#define curr (P, i) P[i]

#define next (P, i) P[(i+1) % P.size()]

polygon convex\_cut (const polygon& P, const line& l) {

polygon Q;

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

point A = curr (P, i), B = next (P, i);

if (ccw (l[0], l[1], A) != -1) Q.push\_back (A);

if (ccw (l[0], l[1], A) \* ccw (l[0], l[1], B) < 0)

Q.push\_back (crosspoint (line (A, B), l));

}

return Q;

}

//Another version of polycut

void cutPoly(list<Pt> &poly, Pt a, Pt b)

{

list<Pt> result;

if(poly.size() == 0)

return;

if(poly.size() == 1)

{

if(leftTurn(a, b, poly.back()))

result.push\_back(poly.back());

poly = result;

return;

}

#define LI list<Pt>::iterator

LI last = poly.end(); last--;

bool lastin = leftTurn(a, b, \*last);

for(LI it = poly.begin(); it != poly.end(); it++)

{

bool thisin = leftTurn(a, b, \*it);

if(!thisin && lastin || thisin && !lastin)

{

Pt r;

lineIntersect(a, b, \*last, \*it, r);

result.push\_back(r);

}

if(thisin)

result.push\_back(\*it);

last = it;

lastin = leftTurn(a, b, \*last);

}

poly.clear();

Pt Last;

for(LI it = result.begin(); it != result.end(); it++)

{

if(!(it != result.begin() && Equal(\*it, Last)))

poly.push\_back(\*it);

Last = \*it;

}

if(poly.size() > 1)

{

if(Equal(poly.front(), poly.back()))

poly.pop\_back();

}

}

### Intersection of Convex Polygons-O( V1 + V2 )

bool intersect\_1pt (const point& a, const point& b,

const point& c, const point& d, point &r) {

number D = cross (b – a, d - c);

if (EQ (D, 0)) return false;

number t = cross (c - a, d - c)/D;

number s = - cross (a - c, b - a)/D;

r = a + t \* (b - a);

return GE (t, 0) && LE (t, 1) && GE (s, 0) && LE (s, 1);

}

polygon convex\_intersect (const polygon &P, const polygon &Q) {

const int n = P.size (), m = Q.size ();

int a = 0, b = 0, aa = 0, ba = 0;

enum {Pin, Qin, Unknown} in = Unknown;

polygon R;

do {

int a1 = (a+n-1) % n, b1 = (b+m-1) % m;

number C = cross (P[a] - P[a1], Q[b] - Q[b1]);

number A = cross (P[a1] - Q[b], P[a] - Q[b]);

number B = cross (Q[b1] - P[a], Q[b] - P[a]);

point r;

if (intersect\_1pt (P[a1], P[a], Q[b1], Q[b], r)) {

if (in == Unknown) aa = ba = 0;

R.push\_back (r);

in = B > 0 ? Pin : A > 0 ? Qin : in;

}

if (C == 0 && B == 0 && A == 0) {

if (in == Pin) {b = (b + 1) % m; ++ba; }

else {a = (a + 1) % m; ++aa; }

} else if (C >= 0) {

if (A > 0) {if (in == Pin) R.push\_back (P[a]); a = (a+1) %n; ++aa; }

else {if (in == Qin) R.push\_back (Q[b]); b = (b+1) %m; ++ba; }

} else {

if (B > 0) {if (in == Qin) R.push\_back (Q[b]); b = (b+1) %m; ++ba; }

else {if (in == Pin) R.push\_back (P[a]); a = (a+1) %n; ++aa; }

}

} while ((aa < n || ba < m) && aa < 2\*n && ba < 2\*m);

if (in == Unknown) {

if (convex\_contains (Q, P [0])) return P;

if (convex\_contains (P, Q [0])) return Q;

}

return R;

}

### Diameter of Convex Polygon (Most Distant Points)-O( V )

#define curr (P, i) P[i]

#define next (P, i) P[(i+1) % P.size ()]

#define diff (P, i) (next (P, i) - curr (P, i))

number convex\_diameter (const polygon &pt) {

const int n = pt.size ();

int is = 0, js = 0;

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

if (imag (pt [i]) > imag (pt [is])) is = i;

if (imag (pt [i]) < imag (pt [js])) js = i;

}

number maxd = norm (pt [is] - pt [js]);

int i, maxi, j, maxj;

i = maxi = is;

j = maxj = js;

do {

if (cross (diff (pt, i), diff (pt, j)) >= 0) j = (j+1) % n;

else i = (i+1) % n;

if (norm (pt[i] - pt[j]) > maxd) {

maxd = norm (pt[i] - pt[j]);

maxi = i; maxj = j;

}

} while (i != is || j != js);

return maxd; /\* farthest pair is (maxi and maxj). \*/

}

### Point Inclusion Convex-O( log V )

enum {OUT, ON, IN};

int convex\_contains (const polygon &P, const point &p) {

const int n = P.size();

point g = (P[0] + P[n/3] + P[2\*n/3])/3.0; // inner-point

int a = 0, b = n;

while (a+1 < b) {// invariant: c is in fan g-P [a] - P [b]

int c = (a + b)/2;

if (cross (P[a] – g, P[c] - g) > 0) {// angle < 180 deg

if (cross (P[a] – g, p-g) > 0 && cross (P[c] – g, p-g) < 0) b = c;

else a = c;

} else {

if (cross (P[a] – g, p-g) < 0 && cross (P[c] – g, p-g) > 0) a = c;

else b = c;

}

}

b %= n;

if (cross (P[a] – p, P[b] - p) < 0) return 0;

if (cross (P[a] – p, P[b] - p) > 0) return 2;

return 1;

}

//Another Version of convexPolygonInclusion

int convexPolygonInclusion(int n, double x[], double y[], double xp, double yp)

/\*

the polygon should have at least three points which are not on the same line.

\*/

/\*

return values:

0: (xp, yp) is outside the polygon

1: (xp, yp) is inside the polygon

2: (xp, yp) is on the polygon side

\*/

{

double xm, ym;

int i;

/\*

the points are checked to be in counter-clockwise order

\*/

if (polygonOrientation(n, x, y) > 0)

{

reverse(x, x + n);

reverse(y, y + n);

}

/\*

find an internal point

\*/

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

if (!equal(multiply(x[0], y[0], x[1], y[1], x[i], y[i]), 0))

{

xm = (x[0] + x[1] + x[i]) / 3;

ym = (y[0] + y[1] + y[i]) / 3;

break;

}

/\*

NOTE: as you see, this algorithm is of O(log(n)) if the above

O(n) computations are done before!

\*/

int f = 0, l = n;

while (f < l - 1)

{

int m = (f + l) / 2;

if (angleInclusion(xm, ym, x[f], y[f], x[m], y[m], xp, yp))

l = m;

else

f = m;

}

double r = multiply(x[f], y[f], x[l % n], y[l % n], xp, yp);

return (equal(r, 0) ? 2 : greater(r, 0));

}

## Circle (incircle, three\_point\_circle, circle\_circle\_intersect, line and circle, greatCircle)

bool incircle (point a, point b, point c, point p) {

a -= p; b -= p; c -= p;

return norm (a) \* cross (b, c)

+ norm (b) \* cross (c, a)

+ norm (c) \* cross (a, b) >= 0; // <: inside, = cocircular, > outside

}

point three\_point\_circle (const point& a, const point& b, const point& c) {

point x = 1.0/conj(b - a), y = 1.0/conj(c - a);

return (y-x)/(conj(x) \* y – x \* conj(y)) + a;

}

pair<point, point> circle\_circle\_intersect (

const point& c1, const double& r1, const point& c2, const double& r2) {

point A = conj(c2-c1), B = (r2\*r2-r1\*r1-(c2-c1)\*conj(c2-c1)), C = r1\*r1\* (c2-c1);

point D = B\*B-4.0\*A\*C;

point z1 = (-B+sqrt(D))/(2.0\*A)+c1, z2 = (-B-sqrt(D))/(2.0\*A)+c1;

return pair<point, point> (z1, z2);

}

int circleLineIntersect(double x, double y, double r, double a, double b, double c)

/\*

return values:

0: no intersection

1: one intersection

2: two intersections

\*/

{

double delta;

if (equal(b, 0))

delta = a \* a \* r \* r - (a \* x + c) \* (a \* x + c);

else

delta = b \* b \* ((a \* a + b \* b) \* r \* r - (a \* x + b \* y + c) \* (a \* x + b \* y + c));

return (less(delta, 0) ? 0 : (equal(delta, 0) ? 1 : 2));

}

int circlesIntersectionPoints(double x1,double y1,double r1,double x2,double y2,double r2,

double &xp1,double &yp1,double &xp2,double &yp2)

{

double a=atan2(y2 - y1, x2 - x1);

double s=hypot(x1 - x2, y1 - y2);

if(s > r1 + r2 + eps)return 0;

double b=acos((r1 \* r1 + s \* s - r2 \* r2) / (2 \* r1 \* s));

if(abs((r1 \* r1 + s \* s - r2 \* r2) / (2 \* r1 \* s))>1)//necessary for some situation due to precision

b=(r1 \* r1 + s \* s - r2 \* r2)>=0?0:PI;

xp1=cos(a+b)\*r1+x1;

xp2=cos(a-b)\*r1+x1;

yp1=sin(a+b)\*r1+y1;

yp2=sin(a-b)\*r1+y1;

if(abs(s-r1-r2)<eps||abs(s-abs(r1-r2))<eps)

return 1;

if(s<abs(r1-r2)-eps)return 0;

return 2;

}

bool circleThroughPoints(double x1, double y1, double x2, double y2, double x3, double y3,

double &x, double &y, double &r)

/\*

result values:

false : the points are on a same extension

true : circle is (x, y, r)

\*/

{

double d = 2 \* (y1 \* x3 + y2 \* x1 - y2 \* x3 - y1 \* x2 - y3 \* x1 + y3 \* x2);

if (equal(d, 0))

return false;

x = (y2 \* x1 \* x1 - y3 \* x1 \* x1 - y2 \* y2 \* y1 + y3 \* y3 \* y1 +

x2 \* x2 \* y3 + y1 \* y1 \* y2 + x3 \* x3 \* y1 - y3 \* y3 \* y2 -

x3 \* x3 \* y2 - x2 \* x2 \* y1 + y2 \* y2 \* y3 - y1 \* y1 \* y3) / d;

y = (x1 \* x1 \* x3 + y1 \* y1 \* x3 + x2 \* x2 \* x1 - x2 \* x2 \* x3 +

y2 \* y2 \* x1 - y2 \* y2 \* x3 - x1 \* x1 \* x2 - y1 \* y1 \* x2 -

x3 \* x3 \* x1 + x3 \* x3 \* x2 - y3 \* y3 \* x1 + y3 \* y3 \* x2) / d;

r = sqrt((x1 - x) \* (x1 - x) + (y1 - y) \* (y1 - y));

return true;

}

int circlesIntersect(double x1, double y1, double r1,

double x2, double y2, double r2)

/\*

return values:

0: no intersection

1: one intersection

2: two intersections

3: the circles are the same

\*/

{

if (equal(x1, x2) && equal(y1, y2) && equal(r1, r2))

return 3;

double d = distance(x1, y1, x2, y2);

if (greater(d, r1 + r2) || less(d, fabs(r1 - r2)))

return 0;

if (equal(d, r1 + r2)||equal(d, abs(r1 - r2))

return 1;

return 2;

}

/\* Computes the center of a circle containing the 2 given

\* points. The circle has the given radius. The returned

\* center is never to the right of the vector

\* (x1, y1)-->(x2, y2).

\* If this is possible, returns true and passes the center

\* through the ctr array. Otherwise, returns false.

\* #include <math.h>

\* FIELD TESTING:

\* - Valladolid 10136: Chocolate Chip Cookies

\*\*/

bool circle2ptsRad( double x1, double y1, double x2, double y2, double r, double ctr[2] )

{

double d2 = ( x1 - x2 ) \* ( x1 - x2 ) + ( y1 - y2 ) \* ( y1 - y2 );

double det = r \* r / d2 - 0.25;

if( det < 0.0 ) return false;

double h = sqrt( det );

ctr[0] = ( x1 + x2 ) \* 0.5 + ( y1 - y2 ) \* h;

ctr[1] = ( y1 + y2 ) \* 0.5 + ( x2 - x1 ) \* h;

return true;

}

/\* Given two pairs of (latitude, longitude), returns the

\* great circle distance between them.

\* FIELD TESTING

\* - Valladolid 535: Globetrotter

\*\*/

double greatCircle( double laa, double loa, double lab, double lob )

{

double PI = acos( -1.0 ), R = 6378.0;

double u[3] = { cos( laa ) \* sin( loa ), cos( laa ) \* cos( loa ), sin( laa ) };

double v[3] = { cos( lab ) \* sin( lob ), cos( lab ) \* cos( lob ), sin( lab ) };

double dot = u[0]\*v[0] + u[1]\*v[1] + u[2]\*v[2];

bool flip = false;

if( dot < 0.0 )

{

flip = true;

for( int i = 0; i < 3; i++ ) v[i] = -v[i];

}

double cr[3] = { u[1]\*v[2] - u[2]\*v[1], u[2]\*v[0] - u[0]\*v[2], u[0]\*v[1] - u[1]\*v[0] };

double theta = asin( sqrt( cr[0]\*cr[0] + cr[1]\*cr[1] + cr[2]\*cr[2] ) );

double len = theta \* R;

if( flip ) len = PI \* R - len;

return len;

}

## Data Structures

### Point Location-[pretreatment O( V ^ 2 ), query: O( log V )]

#define curr(P,i) P[i]

#define next(P,i) P[(i+1)%P.size()]

#define prev(P,i) P[(i+P.size()-1)%P.size()]

struct point\_location {

vector<polygon> regions;

point\_location(const vector<polygon> &regions) :

regions(regions) { compile(); }

struct cut {

double x, y, a;

int d;

cut(double x, double y, double a, int d) :

x(x), y(y), a(a), d(d) { }

bool operator < (const cut &c) const {

return y != c.y ? y < c.y : a != c.a ? a < c.a : d > c.d;

}

bool operator == (const cut &c) const {

return y == c.y && a == c.a;

}

};

vector<double> xs;

vector< vector<cut> > ys;

void compile() {

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

for (int j = 0; j < regions[i].size(); ++j)

xs.push\_back(real(regions[i][j]));

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

xs.erase(unique(xs.begin(), xs.end()), xs.end());

ys.resize(xs.size());

for (int k = 0; k < xs.size(); ++k) {

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

for (int j = 0; j < regions[i].size(); ++j) {

segment seg( curr(regions[i],j), next(regions[i],j) );

int dominant = i;

if (real(seg[0]) > real(seg[1])) {

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

dominant = -1;

}

if (xs[k] < real(seg[0]) || real(seg[1]) <= xs[k]) continue;

double a = imag(seg[1]-seg[0])/real(seg[1]-seg[0]);

double y = imag(seg[0]) + a \* (xs[k]-real(seg[0]));

ys[k].push\_back( cut(xs[k], y, a, dominant) );

}

}

sort(ys[k].begin(), ys[k].end());

ys[k].erase(unique(ys[k].begin(), ys[k].end()), ys[k].end());

}

}

struct at\_x {

double X;

at\_x(double X) : X(X) { }

bool operator () (double y, const cut &c) const {

return y < c.y + c.a \* (X - c.x);

}

};

int locate(const point &p) {

double x = real(p), y = imag(p);

int i = distance(xs.begin(),

upper\_bound(xs.begin(), xs.end(), x)) - 1;

if (i < 0 || i >= xs.size()) return -1;

int j = distance(ys[i].begin(),

upper\_bound(ys[i].begin(), ys[i].end(), y, at\_x(x))) - 1;

if (j < 0 || j >= ys[i].size()) return -1;

return ys[i][j].d;

}

};

### Kd-Tree-[Insert: O( log N ), Search: O( log N )]

struct kdtree {

struct node {

point p;

node \*l, \*r;

node(const point &p)

: p(p), l(NULL), r(NULL) { }

} \*root;

kdtree() : root(NULL) { }

#define compare(d, p, q) (d ? real(p) < real(q) : imag(p) < imag(q))

void insert(const point &p) {

root = insert(root, 0, p);

}

node \*insert(node \*t, int d, const point &p) {

if (t == NULL) return new node(p);

if (compare(d,p,t->p)) t->l = insert(t->l, !d, p);

else t->r = insert(t->r, !d, p);

return t;

}

template <class OUT>

void search(const point &ld, const point &ru, OUT out) {

search(root, 0, ld, ru, out);

}

template <class OUT>

void search(node \*t, int d, const point &ld, const point &ru, OUT out) {

if (t == NULL) return;

const point &p = t->p;

if (real(ld) <= real(p) && real(p) <= real(ru) &&

imag(ld) <= imag(p) && imag(p) <= imag(ru)) \*out++ = p;

if (!compare(d,p,ld)) search(t->l, !d, ld, ru, out);

if (!compare(d,ru,p)) search(t->r, !d, ld, ru, out);

}

};

## Additional Topics

### Closest Pair-[Generally: O( N log N ), Wortst Case(Vertically Line) : O(N ^ 2)]

pair<P, P> closestPair (vector<P> p) {

int n = p.size (), s = 0, t = 1, m = 2, S[n]; S[0] = 0, S[1] = 1;

sort (ALL (p)); // “p < q” <=> “p.x < q.x”

double d = norm (p[s] - p[t]);

for (int i = 2; i < n; S[m++] = i++) REP (j, m) {

if (norm (p[S[j]] - p[i]) <d) d = norm (p[s = S[j]] - p[t = i]);

if (real (p[S[j]]) < real (p[i]) - d) S[j--] = S[--m];

}

return make\_pair (p[s], p[t]);

}

### Base Element of 3D Geomety

#include <valarray>

typedef double number;

typedef valarray<number> point;

const int dim = 3;

number dot (const point& a, const point& b) {

return (a \* b) .sum ();

}

point cross (const point& a, const point& b) {

return a.cshift (+1) \* b.cshift (-1)

- a.cshift (-1) \* b.cshift (+1);

}

number dist2 (const point& a, const point& b) {

return dot (a-b, a-b);

}

### 3D Rotation and Other

----------------------------- Rotation Matrices --------------------------

Where c = cos (theta), s = sin (theta), t = 1-cos (theta), and <X,Y,Z> is the unit vector representing the arbitary axis

1. Left handed about arbitrary axis:

tX^2+c tXY-sZ tXZ+sY 0

tXY+sZ tY^2+c tYZ-sX 0

tXZ-sY tYZ+sX tZ^2+c 0

0 0 0 1

2. Right handed about arbitrary axis:

tX^2+c tXY+sZ tXZ-sY 0

tXY-sZ tY^2+c tYZ+sX 0

tXZ+sY tYZ-sX tZ^2+c 0

0 0 0 1

3. About X Axis

1 0 0 0

0 c -s 0

0 s c 0

0 0 0 1

4. About Y Axis

c 0 s 0

0 1 0 0

-s 0 c 0

0 0 0 1

5. About Z Axis

c -s 0 0

s c 0 0

0 0 1 0

0 0 0 1

----------------------------- Eclipse Properties --------------------------

Equation = x^2 / a^2 + y^2 / b^2 = 1

Area = pi \* a \* b

Area of sector with angle theta = theta \* a \* b \* 0.5

Perimeter = pi \* a \* b \* (1 + h / 4 + h^2 / 64 + h^3 / 256 + ...)

where h = ((a-b) / (a+b))^2

Parametric Equation = (a \* cos(theta), b \* sin(theta))

Polar Equation: r = (a \* b) / sqrt(a^2 \* sin2(theta) + b^2 \* cos2(theta))

--------------------------- Hyperbola Properties -------------------------

Equation = x^2 / a^2 - y^2 / b^2 = 1

Paramteric Equation = (a \* sec(theta), b \* tan(theta))

Another Paramteric Equation = (a \* cosh(theta), b \* sinh(theta)) Which gives only one branch

Area of a sector = a \* b \* theta \* 0.5

Polar Equation: r = (a \* b) / sqrt(a^2 \* sin2(theta) - b^2 \* cos2(theta))

====================== AREA EQUATION OF LATICE POLYGON ====================

B / 2 + I - 1 = A

# Data Structures

## AVL Tree-[All O( log N ) ]

template <class T>

struct avl\_tree {

struct node {

T key;

int size, height;

node \*child[2];

node (const T &key): key (key), size (1), height (1) {

child[0] = child[1] = 0; }

} \*root;

typedef node \*pointer;

avl\_tree () {root = NULL; }

pointer find (const T &key) {return find (root, key); }

node \*find (node \*t and const T &key) {

if (t == NULL) return NULL;

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

else if (key < t->key) return find (t->child[0], key);

else return find (t->child[1], key);

}

void insert (const T &key) {root = insert (root, new node (key)); }

node \*insert (node \*t, node \*x) {

if (t == NULL) return x;

if (x->key <= t->key) t->child[0] = insert (t->child[0], x);

else t->child[1] = insert (t->child[1], x);

t->size += 1;

return balance (t);

}

void erase (const T &key) {root = erase (root, key); }

node \*erase (node \*t, const T &x) {

if (t == NULL) return NULL;

if (x == t->key) {

return move\_down (t->child[0], t->child[1]);

} else {

if (x < t->key) t->child[0] = erase (t->child[0], x);

else t->child[1] = erase (t->child[1], x);

t->size -= 1;

return balance (t);

}

}

node \*move\_down (node \*t, node \*rhs) {

if (t == NULL) return rhs;

t->child[1] = move\_down (t->child[1], rhs);

return balance (t);

}

#define sz (t) (t ? t->size : 0)

#define ht (t) (t ? t->height : 0)

node \*rotate (node \*t, int l, int r) {

node \*s = t->child[r];

t->child[r] = s->child[l];

s->child[l] = balance (t);

if (t) t->size = sz (t->child[0]) + sz (t->child[1]) + 1;

if (s) s->size = sz (s->child[0]) + sz (s->child[1]) + 1;

return balance (s);

}

node \*balance (node \*t) {

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

if (ht (t->child[!i]) - ht (t->child[i]) < -1) {

if (ht(t->child[i]->child[!i]) - ht(t->child[i]->child[i]) > 0)

t->child[i] = rotate (t->child[i], i, !i);

return rotate (t, !i, i);

}

}

if (t) t->height = max (ht (t->child[0]), ht (t->child[1])) + 1;

if (t) t->size = sz (t->child[0]) + sz (t->child[1]) + 1;

return t;

}

pointer rank (int k) const {return rank (root, k); }

pointer rank (node \*t, int k) const {

if (!t) return NULL;

int m = sz (t->child[0]);

if (k < m) return rank (t->child[0], k);

if (k == m) return t;

if (k > m) return rank (t->child[1], k - m - 1);

}

};

## Set-Disjoint-[find, union O(log(n)), make\_set O(1)]

struct UnionFind {

vector<int> data;

UnionFind (int size): data (size, -1) {}

bool unionSet (int x, int y) {

x = root (x); y = root (y);

if (x != y) {

if (data[y] < data[x]) swap (x, y);

data[x] += data[y]; data[y] = x;

}

return x != y;

}

bool findSet (int x, int y) {

return root (x) == root (y);

}

int root (int x) {

return data[x] < 0 ? x : data[x] = root (data[x]);

}

int size (int x) {

return - data[root (x)];

}

};

## Interval Tree-[Insert: O( log N ), Find: O( log N )]

typedef int position;

typedef int contents;

struct interval {

position low, high;

interval(position low, position high) :

low(low), high(high) { }

};

struct interval\_tree {

vector<position> pos;

struct node {

vector<contents> values;

position B, E, M;

node \*left, \*right;

} \*root;

template <class IN>

interval\_tree(IN begin, IN end) : pos(begin, end) {

root = build\_tree(0, pos.size()-1);

}

~interval\_tree() { release(root); }

node \*build\_tree(int i, int j) {

int m = (i+j)/2;

node \*t = new node;

t->B = pos[i]; t->E = pos[j]; t->M = pos[m];

t->left = (i+1 < j ? build\_tree(i, m) : NULL);

t->right = (i+1 < j ? build\_tree(m, j) : NULL);

return t;

}

void insert(const interval& I, contents c) { insert(root, I, c); }

void insert(node \*v, const interval& I, contents c) {

if (I.low <= v->B && v->E <= I.high) {

v->values.push\_back( c );

} else {

if (I.low < v->M) insert(v->left , I, c);

if (I.high > v->M) insert(v->right, I, c);

}

}

template <class OUT>

void query(position p, OUT out) { query(root, p, out); }

template <class OUT>

void query(node \*t, position p, OUT out) {

if (!t || p < t->B || p >= t->E) return;

copy(t->values.begin(), t->values.end(), out);

if (p < t->M) query(t->left, p, out); //half-opened section

else query(t->right, p, out);

}

void release(node \*t) {

if (t->left) release(t->left);

if (t->right) release(t->right);

delete t;

}

};

## Binary Indexed Tree-[Insert: O( log V ), Find: O( log V )]

template <class T>

struct fenwick\_tree {

vector<T> x;

fenwick\_tree (int n) : x (n, 0) {}

T sum (int i, int j) {

if (i == 0) {

T S = 0;

for (j; j >= 0; j = (j & (j + 1)) - 1) S += x[j];

return S;

} else return sum (0, j) - sum (0, i-1);

}

void add (int k, T a) {

for (; k < x.size (); k |= k+1) x[k] += a;

}

};

MaxVal - maximum value which will have non-zero frequency  
  f[i] - frequency of value with index **i**, **i** = **1** .. MaxVal   
  c[i] - cumulative frequency for index **i** (f[1] + f[2] + ... + f[i])  
  tree[i] - sum of frequencies stored in **BIT** with index **i** (latter will be described what index means); sometimes we will write

int read(int idx){

int sum = 0;

while (idx > 0){

sum += tree[idx];

idx -= (idx & -idx);

}

return sum;

}

void update(int idx ,int val){

while (idx <= MaxVal){

tree[idx] += val;

idx += (idx & -idx);

}

}

2D:

1. set dot at (x , y)
2. remove dot from (x , y)
3. count number of dots in rectangle (0 , 0), (x , y) - where (0 , 0) if down-left corner, (x , y) is up-right corner and sides are parallel to x-axis and y-axis.

void update(int x , int y , int val){

while (x <= max\_x){

updatey(x , y , val);

// this function should update array tree[x]

x += (x & -x);

}

}

void updatey(int x , int y , int val){

while (y <= max\_y){

tree[x][y] += val;

y += (y & -y);

}

}

void update(int x , int y , int val){

int y1;

while (x <= max\_x){

y1 = y;

while (y1 <= max\_y){

tree[x][y1] += val;

y1 += (y1 & -y1);

}

x += (x & -x);

}

}

## RMQ-[pretreatment: O( N log N ), query: O( 1 )]

void process2(int M[MAXN][LOGMAXN], int A[MAXN], int N)

{

int i, j;

//initialize M for the intervals with length 1

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

M[i][0] = i;

//compute values from smaller to bigger intervals

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

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

if (A[M[i][j - 1]] < A[M[i + (1 << (j - 1))][j - 1]])

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

else

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

}

Let **k = [log(j - i + 1)]**. For computing **RMQA(i, j)** we can use the following formula:

C:\Documents and Settings\ACM\Desktop\RMQ_005.gif

//Another RMQ

int \*buildRMQ (int \*a and int n) {

int logn = 1;

for (int k = 1; k < n; k \*= 2) ++logn;

int \*r = new int [n \* logn];

int \*b = r; copy (a, a+n and b);

for (int k = 1; k < n; k \*= 2) {

copy (b, b+n and b+n); b += n;

REP (i, n-k) b [i] = min (b [i], b [i+k]);

}

return r;

}

int minimum (int x, int y, int \*rmq and int n) {

int z = y - x, k = 0, e = 1, s; // y - Maximum k it becomes x >= e = 2^k

s = ((z & 0xffff0000)! = 0) << 4; z >>= s; e <<= s; k |= s;

s = ((z & 0x0000ff00)! = 0) << 3; z >>= s; e <<= s; k |= s;

s = ((z & 0x000000f0)! = 0) << 2; z >>= s; e <<= s; k |= s;

s = ((z & 0x0000000c)! = 0) << 1; z >>= s; e <<= s; k |= s;

s = ((z & 0x00000002)! = 0) << 0; z >>= s; e <<= s; k |= s;

return min (rmq [x+n\*k], rmq [y+n\*k-e+1]);

}

## Segment Tree-[Insert: O( log N ), Find: O( log N )]

// M is the array that hold information related to b->e segment

// A is the array from which we gather information

// to construct segment tree call initialize( 0, 0, N - 1, M, A )

void initialize(int node, int b, int e, int M[MAXIND], int A[MAXN])

{

if (b == e)

M[node] = b;

else

{

//compute the values in the left and right subtrees

initialize(2 \* node, b, (b + e) / 2, M, A);

initialize(2 \* node + 1, (b + e) / 2 + 1, e, M, A);

//Do what ever you want with the result of subtrees

//Take a look at the following example for more information

if (A[M[2 \* node]] <= A[M[2 \* node + 1]])

M[node] = M[2 \* node];

else

M[node] = M[2 \* node + 1];

}

}

// to collect information related to i->j segment

// call query( 0, 0, N - 1, M, A, i, j )

int query(int node, int b, int e, int M[MAXIND], int A[MAXN], int i, int j) {

//if the current interval doesn't intersect

//the query interval return -1

if (i > e || j < b)

return -1;

//if the current interval is included in

//the query interval return M[node]

if (b >= i && e <= j)

return M[node];

//compute required value in the

//left and right part of the interval

int p1 = query(2 \* node, b, (b + e) / 2, M, A, i, j);

int p2 = query(2 \* node + 1, (b + e) / 2 + 1, e, M, A, i, j);

//Do what ever you want with the result of subtrees

//Take a look at the following example for more information

if (p1 == -1)

return M[node] = p2;

if (p2 == -1)

return M[node] = p1;

if (A[p1] <= A[p2])

return M[node] = p1;

return M[node] = p2;

}

# Strings

## String Searching

### Shift And-O( L1 + L2 )

//Construction O (m)

//Search O (n + m)

int match (char \*t, char \*p) {

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

int M[0x100]; fill(M, M+0x100, 0);

int count = 0;

for (int i = 0; i < m; ++i) M[p[i]] |= (1 << i);

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

S = ((S << 1) | 1) & M[t[i]];

if (S & (1 << (m-1))) {

++count; // match at t[i-m+1… i]

}

}

return count;

}

### Knuth Morris Pratt-O( L1 + L2 )

int \*buildFail (char \*p) {

int m = strlen (p);

int \*fail = new int [m+1];

int j = fail [0] = -1;

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

while (j >= 0 && p[j]! = p[i-1]) j = fail[j];

fail [i] = ++j;

}

return fail;

}

int match (char \*t, char \*p, int \*fail) {

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

int count = 0;

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

while (k >= 0 && p [k]! = t [i]) k = fail [k];

if (++k >= m) {

++count; // match at t [i-m+1. i]

k = fail [k];

}

}

return count;

}

## String-Related Data Structures

### Aho-Corasick-O(Patterns\_length + sigma(Texts\_length))

struct PMA {

PMA \*next [0x100]; // next [0] is for fail

vector<int> accept;

PMA () {fill (next, next+0x100, (PMA\*) 0); }

};

PMA \*buildPMA (char \*p [], int size) {

PMA \*root = new PMA;

for (int i = 0; i < size; ++i) {// make trie

PMA \*t = root;

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

char c = p [i] [j];

if (t->next [c] == NULL) t->next [c] = new PMA;

t = t->next [c];

}

t->accept.push\_back (i);

}

queue<PMA\*> Q; // make failure link using bfs

for (int c = 'a'; c <= 'z'; ++c) {

if (root->next [c]) {

root->next [c] - >next [0] = root;

Q.push (root->next [c]);

} else root->next [c] = root;

}

while (! Q.empty ()) {

PMA \*t = Q.front (); Q.pop ();

for (int c = 'a'; c <= 'z'; ++c) {

if (t->next [c]) {

Q.push (t->next[c]);

PMA \*r = t->next[0];

while (! r->next[c]) r = r->next [0];

t->next[c]->next[0] = r->next [c];

}

}

}

return root;

}

// after a call to match i-th element of result will be true if i-th

// pattern is found in t

int match (char \*t, PMA \*v, int \*result) {

int n = strlen (t);

int count = 0;

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

char c = t[i];

while (!v->next[c]) v = v->next[0];

v = v->next[c];

for (int j = 0; j < v->accept.size(); ++j)

result[v->accept[j]]++;

}

}

### Trie-O(n)

struct Trie {

int value;

Trie \*next [0x100];

Trie () {fill (next and next+0x100, (Trie\*) 0); }

};

Trie \*find (char \*t, Trie \*r) {

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

char c = t[i];

if (!r->next[c]) r->next[c] = new Trie;

r = r->next[c];

}

return r;

}

### Suffix Tree

class Suffix {

public :

int origin\_node;

int first\_char\_index;

int last\_char\_index;

Suffix( int node, int start, int stop )

: origin\_node( node ),

first\_char\_index( start ),

last\_char\_index( stop ){};

int Explicit(){ return first\_char\_index > last\_char\_index; }

int Implicit(){ return last\_char\_index >= first\_char\_index; }

void Canonize();

};

class Edge {

public :

int first\_char\_index;

int last\_char\_index;

int end\_node;

int start\_node;

void Insert();

void Remove();

Edge();

Edge( int init\_first\_char\_index,

int init\_last\_char\_index,

int parent\_node );

int SplitEdge( Suffix &s );

static Edge Find( int node, int c );

static int Hash( int node, int c );

};

class Node {

public :

int suffix\_node;

Node() { suffix\_node = -1; }

static int Count;

};

const int MAX\_LENGTH = 100010;

const int HASH\_TABLE\_SIZE = 100021; //A prime roughly 10% larger

Edge Edges[ HASH\_TABLE\_SIZE ];

int Node::Count = 1;

Node Nodes[ MAX\_LENGTH \* 2 ];

char T[ MAX\_LENGTH ];

int N;

Edge::Edge()

{

start\_node = -1;

}

Edge::Edge( int init\_first, int init\_last, int parent\_node )

{

first\_char\_index = init\_first;

last\_char\_index = init\_last;

start\_node = parent\_node;

end\_node = Node::Count++;

}

int Edge::Hash( int node, int c )

{

return ( ( node << 8 ) + c ) % HASH\_TABLE\_SIZE;

}

void Edge::Insert()

{

int i = Hash( start\_node, T[ first\_char\_index ] );

while ( Edges[ i ].start\_node != -1 )

i = ++i % HASH\_TABLE\_SIZE;

Edges[ i ] = \*this;

}

void Edge::Remove()

{

int i = Hash( start\_node, T[ first\_char\_index ] );

while ( Edges[ i ].start\_node != start\_node ||

Edges[ i ].first\_char\_index != first\_char\_index )

i = ++i % HASH\_TABLE\_SIZE;

for ( ; ; ) {

Edges[ i ].start\_node = -1;

int j = i;

for ( ; ; ) {

i = ++i % HASH\_TABLE\_SIZE;

if ( Edges[ i ].start\_node == -1 )

return;

int r = Hash( Edges[ i ].start\_node, T[ Edges[ i ].first\_char\_index ] );

if ( i >= r && r > j )

continue;

if ( r > j && j > i )

continue;

if ( j > i && i >= r )

continue;

break;

}

Edges[ j ] = Edges[ i ];

}

}

Edge Edge::Find( int node, int c )

{

int i = Hash( node, c );

for ( ; ; ) {

if ( Edges[ i ].start\_node == node )

if ( c == T[ Edges[ i ].first\_char\_index ] )

return Edges[ i ];

if ( Edges[ i ].start\_node == -1 )

return Edges[ i ];

i = ++i % HASH\_TABLE\_SIZE;

}

}

int Edge::SplitEdge( Suffix &s )

{

Remove();

Edge \*new\_edge =

new Edge( first\_char\_index,

first\_char\_index + s.last\_char\_index - s.first\_char\_index,

s.origin\_node );

new\_edge->Insert();

Nodes[ new\_edge->end\_node ].suffix\_node = s.origin\_node;

first\_char\_index += s.last\_char\_index - s.first\_char\_index + 1;

start\_node = new\_edge->end\_node;

Insert();

return new\_edge->end\_node;

}

void Suffix::Canonize()

{

if ( !Explicit() ) {

Edge edge = Edge::Find( origin\_node, T[ first\_char\_index ] );

int edge\_span = edge.last\_char\_index - edge.first\_char\_index;

while ( edge\_span <= ( last\_char\_index - first\_char\_index ) ) {

first\_char\_index = first\_char\_index + edge\_span + 1;

origin\_node = edge.end\_node;

if ( first\_char\_index <= last\_char\_index ) {

edge = Edge::Find( edge.end\_node, T[ first\_char\_index ] );

edge\_span = edge.last\_char\_index - edge.first\_char\_index;

};

}

}

}

void AddPrefix( Suffix &active, int last\_char\_index )

{

int parent\_node;

int last\_parent\_node = -1;

for ( ; ; ) {

Edge edge;

parent\_node = active.origin\_node;

if ( active.Explicit() ) {

edge = Edge::Find( active.origin\_node, T[ last\_char\_index ] );

if ( edge.start\_node != -1 )

break;

} else { //implicit node, a little more complicated

edge = Edge::Find( active.origin\_node, T[ active.first\_char\_index ] );

int span = active.last\_char\_index - active.first\_char\_index;

if ( T[ edge.first\_char\_index + span + 1 ] == T[ last\_char\_index ] )

break;

parent\_node = edge.SplitEdge( active );

}

Edge \*new\_edge = new Edge( last\_char\_index, N, parent\_node );

new\_edge->Insert();

if ( last\_parent\_node > 0 )

Nodes[ last\_parent\_node ].suffix\_node = parent\_node;

last\_parent\_node = parent\_node;

if ( active.origin\_node == 0 )

active.first\_char\_index++;

else

active.origin\_node = Nodes[ active.origin\_node ].suffix\_node;

active.Canonize();

}

if ( last\_parent\_node > 0 )

Nodes[ last\_parent\_node ].suffix\_node = parent\_node;

active.last\_char\_index++; //Now the endpoint is the next active point

active.Canonize();

};

bool isPresent(char\* s, int len)

{

int start\_node = 0;

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

{

Edge edge = Edge::Find( start\_node, s[i] );

if(edge.start\_node == -1)

return 0;

for(int j = edge.first\_char\_index; j != edge.last\_char\_index; j++)

if(i >= len)

return true;

else if(s[i++] != T[j])

return false;

start\_node = edge.end\_node;

}

return true;

}

int main()

{

int t, q;cin >> t;

cin.getline(T, MAX\_LENGTH - 1);

for(; t--;)

{

cin.getline(T, MAX\_LENGTH - 2);

N = strlen(T) - 1;

Suffix active(0, 0, -1);

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

AddPrefix(active, i);

cin >> q;

char ha[1002];

cin.getline(ha, 1001);

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

{

cin.getline(ha, 1001);

if(isPresent(ha, strlen(ha)))

cout << "y" << endl;

else

cout << "n" << endl;

}

}

return 0;

};

### Suffix Array-[Matching naive O (m log n),Mamber-Myers's O (m + log n)]

struct SAComp {

const int h and \*g;

SAComp (int h and int\* g): h (h), g (g) {}

bool operator () (int a and int b) {

return a == b? false: g [a]! = g [b]? g [a] < g [b]: g [a+h] < g [b+h];

}

};

int \*buildSA (char\* t and int n) {

int g [n+1], b [n+1], \*v = new int [n+1];

REP (i, n+1) v [i] = i, g [i] = t [i];

b [0] = 0; b [n] = 0;

sort (v, v+n+1 and SAComp (0, g));

for (int h = 1; b [n]! = n; h \*= 2) {

SAComp comp (h and g);

sort (v, v+n+1 and comp);

REP (i, n) b [i+1] = b [i] + comp (v [i], v [i+1]);

REP (i, n+1) g [v [i]] = b [i];

}

return v;

}

// Naive matching O (m log n)

int find (char \*t, int n, char \*p and int m, int \*sa) {

int a = 0, b = n;

while (a < b) {

int c = (a + b)/2;

if (strncmp (t+sa [c], p and m) < 0) a = c+1; else b = c;

}

return strncmp (t+sa [a], p, m) == 0? sa [a]: -1;

}

// Kasai-Lee-Arimura-Arikawa-Park's simple LCP computation: O (n)

int \*buildLCP (char \*t, int n and int \*a) {

int h = 0, b [n+1], \*lcp = new int [n+1];

REP (i, n+1) b [a [i]] = i;

REP (i, n+1) {

if (b [i]) {

for (int j = a [b [i] - 1]; j+h<n && i+h<n && t [j+h] == t [i+h]; ++h);

lcp [b [i]] = h;

} else lcp [b [i]] = -1;

if (h > 0) --h;

}

return lcp;

}

// call RMQ = buildRMQ (lcp and n+1)

int \*buildRMQ (int \*a and int n) {

int logn = 1;

for (int k = 1; k < n; k \*= 2) ++logn;

int \*r = new int [n \* logn];

int \*b = r; copy (a, a+n and b);

for (int k = 1; k < n; k \*= 2) {

copy (b, b+n and b+n); b += n;

REP (i, n-k) b [i] = min (b [i], b [i+k]);

}

return r;

}

// inner LCP computation with RMQ: O (1)

int minimum (int x, int y, int \*rmq and int n) {

int z = y - x, k = 0, e = 1, s; // y - Maximum k it becomes x >= e = 2^k

s = ((z & 0xffff0000)! = 0) << 4; z >>= s; e <<= s; k |= s;

s = ((z & 0x0000ff00)! = 0) << 3; z >>= s; e <<= s; k |= s;

s = ((z & 0x000000f0)! = 0) << 2; z >>= s; e <<= s; k |= s;

s = ((z & 0x0000000c)! = 0) << 1; z >>= s; e <<= s; k |= s;

s = ((z & 0x00000002)! = 0) << 0; z >>= s; e <<= s; k |= s;

return min (rmq [x+n\*k], rmq [y+n\*k-e+1]);

}

// outer LCP computation: O (m - o)

int computeLCP (char \*t, int n, char \*p and int m, int o and int k) {

int i = o;

for (; i < m && k+i < n && p [i] == t [k+i]; ++i);

return i;

}

// Mamber-Myers's O (m + log n) string matching with LCP/RMQ

#define COMP (h and k) (h == m || (k+h<n && p [h] <t [k+h]))

int find (char \*t, int n, char \*p and int m, int \*sa and int \*rmq) {

int l = 0, lh = 0, r = n and rh = computeLCP (t, n+1 and p, m and 0, sa [n]);

if (! COMP (rh and sa [r])) return -1;

for (int k = (l+r) /2; l+1 < r; k = (l+r) /2) {

int A = minimum (l+1, k, rmq and n+1), B = minimum (k+1, r, rmq and n+1);

if (A >= B) {

if (lh < A) l = k;

else if (lh > A) r = k and rh = A;

else {

int i = computeLCP (t, n+1 and p, m, A and sa [k]);

if (COMP (i, sa [k])) r = k, rh = i; else l = k and lh = i;

}

} else {

if (rh < B) r = k;

else if (rh > B) l = k and lh = B;

else {

int i = computeLCP (t, n+1 and p, m, B and sa [k]);

if (COMP (i, sa [k])) r = k, rh = i; else l = k and lh = i;

}

}

}

return rh == m? sa [r]: -1;

}

## Additional Topics

### Longest Palinrome-O(n)

int longest\_palindrome (char \*text and int n) {

int rad [2\*n], i, j and k;

for (i = 0, j = 0; i < 2\*n; i += k and j = max (j-k, 0)) {

while (i-j >= 0 && i+j+1 < 2\*n && text [(i-j) /2] == text [(i+j+1) /2]) ++j;

rad [i] = j;

for (k = 1; i-k >= 0 && rad [i] - k >= 0 && rad [i-k]! = rad [i] - k; ++k)

rad [i+k] = min (rad [i-k], rad [i] - k);

}

return \*max\_element (rad and rad+2\*n); // ret. centre of the longest palindrome

}

### Longest increase substring-O (n log n)

const int inf = 99999999;

#define index\_of (as and x) ¥

distance (as.begin (), lower\_bound (as.begin (), as.end (), x))

vector<int> lis\_fast (const vector<int>& a) {

const int n = a.size ();

vector<int> A (n and inf);

vector<int> id (n);

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

id [i] = index\_of (A and a [i]);

A [id [i]] = a [i];

}

int m = \*max\_element (id.begin (), id.end ());

vector<int> b (m+1);

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

if (id [i] == m) b [m--] = a [i];

return b;

}

### Longest common subsequence-O ((n+r) log n)

struct node {

int value;

node \*next;

node (int value and node \*next): value (value), next (next) {}

};

#define index\_of (as and x) ¥

distance (as.begin (), lower\_bound (as.begin (), as.end (), x))

const int inf = 99999999;

vector<int> lcs\_hs (const vector<int> &a and const vector<int> &b) {

const int n = a.size (), m = b.size ();

map< int and vector<int> > M;

for (int j = m-1; j >= 0; --j)

M [b [j]] .push\_back (j);

vector<int> xs (n+1 and inf); xs [0] = - inf;

vector< node\* > link (n+1);

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

if (M.count (a [i])) {

vector<int> ys = M [a [i]];

for (int j = 0; j < ys.size (); ++j) {

int k = index\_of (xs and ys [j]);

xs [k] = ys [j];

link [k] = new node (b [ys [j]], link [k-1]);

}

}

}

int l = index\_of (xs and inf-1) - 1;

for (node \*p = link [l]; p; p = p->next)

c.push\_back (p->value);

reverse (c.begin (), c.end ());

return c;

}

# Math

## Big Integer

const int64 D = 31;

const int64 N = 40;

const int64 base = (1ll << D);

const int64 bitmask = base - 1;

struct BigInt{

int64 digits[N];

int cnt;

BigInt(int64 n = 0) {

cnt = 0;

memset(digits, 0, sizeof digits);

for(int i = 0; i < N && n; n >>= D) {

digits[i] = n & bitmask;

cnt = ++i;

}

}

BigInt(string s) {

\*this = BigInt(0);

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

\*this = \*this \* 10 + s[i] - '0';

}

void operator=(const BigInt& b) {

memcpy(digits, b.digits, sizeof digits);

cnt = b.cnt;

}

BigInt operator+(const BigInt& b) const {

int64 r = 0;

BigInt result;

for(int i = 0; i < maxi(cnt, b.cnt) || r; i ++) {

r += digits[i] + b.digits[i];

result.digits[i] = r & bitmask;

r >>= D;

result.cnt = i + 1;

}

return result;

}

BigInt operator-(const BigInt& b) const {

BigInt result;

int64 r = 0;

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

if((result.digits[i] = digits[i] - b.digits[i] + r) < 0)

result.digits[i] += base, r = -1;

else

r = 0;

if(result.digits[i]) result.cnt = i+1;

}

return result;

}

BigInt operator\*(int64 n) const {

int64 r = 0;

BigInt result;

for(int i = 0; (i < cnt || r) && n; i ++) {

r += n \* digits[i];

result.digits[i] = r & bitmask;

r >>= D;

result.cnt = i + 1;

}

return result;

}

BigInt operator\*(const BigInt& b) const {

BigInt result;

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

result = result \* base + b \* digits[i];

return result;

}

void shiftLeft() {

int r = 0;

for(int i = 0; i < cnt+1 && i < N; i ++) {

int nr = 0;

if(digits[i] & (1 << (D-1)))

nr = 1;

digits[i] = (digits[i] << 1) & bitmask;

digits[i] += r;

r = nr;

if(digits[i]) cnt = i + 1;

}

}

BigInt operator/(const BigInt& b) const {

BigInt result, row;

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

int64 digit = 0;

for(int j = D-1; j >= 0; j --) {

row.shiftLeft();

row = row + ((digits[i] & (1 << j)) != 0);

digit <<= 1;

if(row >= b) row = row - b, digit += 1;

}

result = result \* base + digit;

}

return result;

}

BigInt operator%(const BigInt& b) const {

return (\*this) - (\*this / b) \* b;

}

bool operator<(const BigInt& b) const {

if(cnt != b.cnt)

return cnt < b.cnt;

else

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

if(digits[i] != b.digits[i])

return digits[i] < b.digits[i];

return false;

}

bool operator==(const BigInt& b) const{ return !(\*this < b) && !(b < \*this); }

bool operator<=(const BigInt& b) const{ return !(b < \*this); }

bool operator>(const BigInt& b) const{ return !(\*this <= b); }

bool operator>=(const BigInt& b) const{ return !(\*this < b); }

bool operator!=(const BigInt& b) const{ return !(\*this == b); }

string str() const {

string result = "0";

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

string temp;

int64 r = 0;

for(int j = result.length()-1; j >= 0 || r; j --) {

if( j >= 0 ) r += (result[j] - '0') \* base;

temp += char('0' + r % 10);

r /= 10;

}

result = "";

r = digits[i];

for(int j = 0; j < temp.length() || r; j ++) {

if( j < temp.length() ) r += (temp[j] - '0');

result += char('0' + r % 10);

r /= 10;

}

reverse(result.begin(), result.end());

}

return result;

}

};

## Rational number

### Rational number

typedef long long Integer;

Integer gcd (Integer a and Integer b) {return a > 0? gcd (b % a and a): b; }

struct rational {

Integer p and q;

void normalize () {// keep q positive

if (q < 0) p \*= -1, q \*= -1;

Integer d = gcd (p < 0? - p: p, q);

if (d == 0) p = 0, q = 1;

else p/= d and q/= d;

}

rational (Integer p and Integer q = 1): p (p) and q (q) {

normalize ();

}

rational &operator += (const rational &a) {

p = a.q \* p + a.p \* q; q = a.q \* q; normalize ();

return \*this;

}

rational &operator - = (const rational &a) {

p = a.q \* p - a.p \* q; q = a.q \* q; normalize ();

return \*this;

}

rational &operator \*= (const rational &a) {

p \*= a.p; q \*= a.q; normalize ();

return \*this;

}

rational &operator/= (const rational &a) {

p \*= a.q; q \*= a.p; normalize ();

return \*this;

}

rational &operator - () {

p \*= -1;

return \*this;

}

};

rational operator + (const rational &a and const rational &b) {

return rational (a) += b;

}

rational operator \* (const rational &a and const rational &b) {

return rational (a) \*= b;

}

rational operator - (const rational &a and const rational &b) {

return rational (a) - = b;

}

rational operator/(const rational &a and const rational &b) {

return rational (a)/= b;

}

bool operator < (const rational &a and const rational &b) {// avoid overflow

return (long double) a.p \* b.q < (long double) a.q \* b.p;

}

bool operator <= (const rational &a and const rational &b) {

return! (b < a);

}

bool operator > (const rational &a and const rational &b) {

return b < a;

}

bool operator >= (const rational &a and const rational &b) {

return! (a < b);

}

bool operator == (const rational &a and const rational &b) {

return! (a < b) &&! (b < a);

}

bool operator! = (const rational &a and const rational &b) {

return (a < b) || (b < a);

}

### Stern-Brocot

// Stern-Brocot Tree for enumerating rationals

// Enumerating all irreducible rationals ascending order,

// whose sum of N and D is atmost B

void sternBrocot (Int B and Int pl = 0, Int ql = 1,

Int PR = 1, Int qr = 0) {

Int pm = pl + PR and qm = ql + qr;

if (pm + qm > B) return;

sternBrocot (B, pl, ql, pm and qm); // [pl/ql and pm/qm]

cout << pm << “/” << qm << endl;

sternBrocot (B, pm, qm, PR and qr); // [pm/qm and pr/qr]

}

## Number Theory

### GCD, Extended-GCD-O( log ( a + b ) )

Int gcd (Int a, Int b) {

return b != 0? gcd (b, a % b): a;

}

Int lcm (Int a and Int b) {

return a \* b/gcd (a, b);

}

// a x + b y = gcd (a and b)

Int extgcd(Int a, Int b, Int &x, Int &y) {

Int g = a; x = 1; y = 0;

if (b != 0) g = extgcd(b, a % b, y, x), y -= (a/b) \* x;

return g;

}

### Modular Inverse-O( log( a ) )

Int invMod (Int a, Int m) {

Int x, y;

if (extgcd (a, m, x, y) == 1) return (x + m) % m;

else return 0; // unsolvable

}

### Linear Congruences Equations-O(n \* log(n))

//It solves a[i] \* x == b[i] (mod m[i]) (i = 0,…, n-1)

// x + k \* M will be the solution if a solutions exists

bool linearCongruences (const vector<Int> &a,

const vector<Int> &b,

const vector<Int> &m,

Int &x, Int &M) {

int n = a.size ();

x = 0; M = 1;

REP (i, n) {

Int a\_ = a [i] % M, b\_ = b [i] - a [i] \* x, m\_ = m [i];

Int y, t and g = extgcd (a\_, m\_, y, t);

if (b\_ % g) return false;

b\_/= g; m\_/= g;

x += M \* (y \* b\_ % m\_);

M \*= m\_;

}

x = (x + M) % M;

return true;

}

### Linear Diophantine Equation Solver-O( log N )

/\*Solves integer equations of the form ax + by = c

\* solutions will be of the form:

\* x = t.x + k \* b / t.d,

\* y = t.y - k \* a / t.d;\*/

template< class Int >

Triple< Int > ldioph( Int a, Int b, Int c )

{

Triple< Int > t = egcd( a, b );

if( c % t.d ) return Triple< Int >( 0, 0, 0 );

t.x \*= c / t.d; t.y \*= c / t.d;

return t;

}

### Modular Linear Equation Solver-O(log n)

/\*solves ax = b (mod n)

\* Returns the vector of solutions, all smaller

\* than n and sorted in increasing order. The vector is

\* empty if there are no solutions. \*/

template< class Int >

vector< Int > msolve( Int a, Int b, Int n )

{

if( n < 0 ) n = -n;

Triple< Int > t = egcd( a, n );

vector< Int > r;

if( b % t.d ) return r;

Int x = ( b / t.d \* t.x ) % n;

if( x < Int( 0 ) ) x += n;

for( Int i = 0; i < t.d; i++ )

r.push\_back( ( x + i \* n / t.d ) % n );

return r;

}

### Mod Power-O( log k )

Int powMod (Int x, Int k and Int m) {

if (k == 0) return 1;

if (k % 2 == 0) return powMod (x\*x % m, k/2, m);

else return x\*powMod (x, k-1, m) % m;

}

### Jacobi Sign-O(loga)

// Jacobi Symbol (m/n),

// m, n >= 0, n: odd

//

// (m/n) == 1 <=> x^2 == m (mod n) solvable

// == -1 <=>… unsolvable

#define NEGPOW (e) ((e) % 2? -1: 1)

Int jacobi (Int a and Int m) {

if (a == 0) return m == 1? 1: 0;

if (a % 2) return NEGPOW ((a-1) & (m-1) /4) \*jacobi (m%a and a);

else return NEGPOW ((m\*m-1) /8) \*jacobi (a/2, m);

}

### Sqrt Mode-O(log k)

// x^2 == a (mod p)

Int sqrtMod (Int n, Int p) {

Int S, Q and W, i and m = invMod (n, p);

for (Q = p - 1, S = 0; Q % 2 == 0; Q/= 2, ++S);

do {W = rand () % p; } while (W == 0 || jacobi (W, p)! = -1);

for (Int R = powMod (n, (Q+1) /2, p), V = powMod (W, Q, p); ;) {

Int z = R & R \* m % p;

for (i = 0; i < S && z % p! = 1; z \*= z, ++i);

if (i == 0) return R;

R = (R \* powMod (V and 1 << (S-i-1) and p)) % p;

}

}

### Euler Phi Function-O( sqrt( N ) )

// n (1-1/p1)… (1-1/pn)

Int eulerPhi (Int n) {

if (n == 0) return 0;

Int ans = n;

for (Int x = 2; x\*x <= n; ++x) {

if (n % x == 0) {

ans - = ans/x;

while (n % x == 0) n/= x;

}

}

if (n > 1) ans - = ans/n;

return ans;

}

// LookUp Version

const int N = 1000000;

Int eulerPhi (Int n) {

static int lookup = 0, p [N], f [N];

if (! lookup) {

REP (i, N) p [i] = 1, f [i] = i;

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

if (p [i]) {

f [i] - = f [i]/i;

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

p [j] = 0, f [j] - = f [j]/i;

}

}

lookup = 1;

}

return f [n];

}

### Prime Detection

#### Sieve of Eratosthenes

vector<int> sieve\_of\_eratosthenes (int n) {

vector<int> primes (n);

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

primes [i] = i;

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

if (primes [i])

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

primes [j] = 0;

return primes;

}

#### Sieve of Atkin

void sieve\_of\_atkin () {

int n;

for (int z = 1; z <= 5; z += 4) {

for (int y = z; y <= sqrtN; y += 6) {

for (int x = 1; x <= sqrtN && (n = 4\*x\*x+y\*y) <= N; ++x)

isprime [n] =! isprime [n];

for (int x = y+1; x <= sqrtN && (n = 3\*x\*x-y\*y) <= N; x += 2)

isprime [n] =! isprime [n];

}

}

for (int z = 2; z <= 4; z += 2) {

for (int y = z; y <= sqrtN; y += 6) {

for (int x = 1; x <= sqrtN && (n = 3\*x\*x+y\*y) <= N; x += 2)

isprime [n] =! isprime [n];

for (int x = y+1; x <= sqrtN && (n = 3\*x\*x-y\*y) <= N; x += 2)

isprime [n] =! isprime [n];

}

}

for (int y = 3; y <= sqrtN; y += 6) {

for (int z = 1; z <= 2; ++z) {

for (int x = z; x <= sqrtN && (n = 4\*x\*x+y\*y) <= N; x += 3)

isprime [n] =! isprime [n];

}

}

for (int n = 5; n <= sqrtN; ++n)

if (isprime [n])

for (int k = n\*n; k <= N; k+=n\*n)

isprime [k] = false;

isprime [2] = isprime [3] = true;

}

void sieve\_of\_atkin () {

int n;

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

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

n = 4\*x\*x + y\*y;

if (n <= N && (n % 12 == 1 || n % 12 == 5))

isprime [n] =! isprime [n];

n = 3\*x\*x + y\*y;

if (n <= N && n % 12 == 7)

isprime [n] =! isprime [n];

n = 3\*x\*x - y\*y;

if (x > y && n <= N && n % 12 == 11)

isprime [n] =! isprime [n];

}

}

for (int n = 5; n <= sqrtN; ++n)

if (isprime [n])

for (int k = n\*n; k <= N; k+=n\*n)

isprime [k] = false;

isprime [2] = isprime [3] = true;

}

#### Sieve of Yarin

// Super fast & Memory-tight Sieve by Yarin

#define MAXSIEVE 100000000 // All prime numbers up to this

#define MAXSIEVEHALF (MAXSIEVE/2)

#define MAXSQRT 5000 // sqrt(MAXSIEVE)/2

char a[MAXSIEVE/16+2];

#define isprime(n) (a[(n)>>4]&(1<<(((n)>>1)&7))) // Works when n is odd

//have to check for even numbers

void sieve()

{

int i,j;

memset(a,255,sizeof(a));

a[0]=0xFE;

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

if (a[i>>3]&(1<<(i&7)))

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

a[j>>3]&=~(1<<(j&7));

}

#### Stochastic

//Frequency of the loop is 200 times vis-a-vis most numbers.

bool suspect (Int a, int s, Int d and Int n) {

Int x = powMod (a, d and n);

if (x == 1) return true;

for (int r = 0; r < s; ++r) {

if (x == n - 1) return true;

x = x \* x % n;

}

return false;

}

// {2,7,61, - 1} is for n < 4759123141 (= 2^32)

// {2,3,5,7,11,13,17,19,23, - 1} is for n < 10^16 (at least)

bool isPrime (Int n) {

if (n <= 1 || (n > 2 && n % 2 == 0)) return false;

int test [] = {2,3,5,7,11,13,17,19,23, - 1};

Int d = n - 1, s = 0;

while (d % 2 == 0) ++s, d/= 2;

for (int i = 0; test [i] < n && test [i]! = -1; ++i)

if (! suspect (test [i], s, d and n)) return false;

return true;

}

## Algebra

### Matrix Operations

typedef double number;

const number eps = 1e-8;

typedef vector<number> array;

typedef vector<array> matrix;

// O (n)

matrix identity (int n) {

matrix A (n and array (n));

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

return A;

}

// O (n)

number inner\_product (const array &a and const array &b) {

number ans = 0;

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

ans += a [i] & b [i];

return ans;

}

// O (n^2)

array mul (const matrix &A and const array &x) {

array y (A.size ());

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

for (int j = 0; j < A [0] .size (); ++j)

y [i] = A [i] [j] \* x [j];

return y;

}

// O (n^3)

matrix mul (const matrix &A and const matrix &B) {

matrix C (A.size (), array (B [0] .size ()));

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

for (int j = 0; j < C [i] .size (); ++j)

for (int k = 0; k < A [i] .size (); ++k)

C [i] [j] += A [i] [k] & B [k] [j];

return C;

}

// O (n^3 log e)

matrix pow (const matrix &A and int e) {

return e == 0? identity (A.size ()) :

e % 2 == 0? pow (mul (A, A) and e/2): mul (A and pow (A and e-1));

}

// O (n^3)

number det (matrix A) {

const int n = A.size ();

number D = 1;

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

int pivot = i;

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

if (abs (A [j] [i]) > abs (A [pivot] [i])) pivot = j;

swap (A [pivot], A [i]);

D \*= A [i] [i] & (i! = pivot? -1: 1);

if (abs (A [i] [i]) < eps) break;

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

for (int k = n-1; k >= i; --k)

A [j] [k] - = A [i] [k] \* A [j] [i]/A [i] [i];

}

return D;

}

// O (n)

number tr (const matrix &A) {

number ans = 0;

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

ans += A [i] [i];

return ans;

}

// O (n^3).

int rank (matrix A) {

const int n = A.size (), m = A [0] .size ();

int r = 0;

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

int pivot = r;

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

if (abs (A [j] [i]) > abs (A [pivot] [i])) pivot = j;

swap (A [pivot], A [r]);

if (abs (A [r] [i]) < eps) continue;

for (int k = m-1; k >= i; --k)

A [r] [k]/= A [r] [i];

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

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

A [j] [k] - = A [r] [k] \* A [j] [i];

++r;

}

return r;

}

### LU disassembly( Linear Equition Solver )-O( N ^ 3 )

#define number double

#define array vector< number >

#define matrix vector< vector< number > >

struct LUinfo {

vector<number> value;

vector<int> index;

};

// A = matrice zarayeb, B = matrice javabha

// O (n^3), Gaussian forward elimination

LUinfo LU\_decomposition ( matrix A ) {

const int n = A.size ();

LUinfo data;

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

int pivot = i;

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

if (abs (A [j] [i]) > abs (A [pivot] [i])) pivot = j;

swap (A [pivot], A [i]);

data.index.push\_back (pivot);

// if A [i] [i] == 0, LU decomposition failed.

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

A [j] [i]/= A [i] [i];

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

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

data.value.push\_back (A [j] [i]);

}

}

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

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

data.value.push\_back (A [i] [j]);

data.value.push\_back (A [i] [i]);

}

return data;

}

// O (n^2) Gaussian backward substitution

array LU\_backsubstitution (const LUinfo &data , array b) {

const int n = b.size ();

int k = 0;

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

swap (b [data.index [i]], b [i]);

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

b [j] -= b [i] \* data.value[k++];

}

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

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

b [i] -= b [j] \* data.value[k++];

b [i] /= data.value[k++];

}

return b;

}

/\*Another Linear Equation Solver

\* \* Solves a system of linear equations using Gauss Method.

\* \* You should :

\* a) put equation matrix in mat[0..n - 1][0..n - 1]

\* b) put array of answers in mat[0..n - 1][n]

\* c) call solve(n)

\*

\* \* Solve(n) returns :

\* a) 0 : if system has a unique answer

\* b) 1 : if system has infinite answers

\* c) 2 : if system has no answers

\*

\* \* It also fills mark[0..n - 1] with above flags (0, 1, 2) to

\* indicate that the system had unique/infinite/no answer(s)

\* for parameter[0..n - 1]

\*

\* \* If the system has a unique answer for parameter[i] then after

\* calling solve(n) the answer for parameter[i] equals:

\* (mat[i][n] / mat[i][i])

\*

\*-----------------------------------------------------------------\*/

#include <fstream>

#include <cmath>

#include <cstring>

using namespace std;

const double epsillon = 1e-6

;

double mat[100][101];

int mark[100];

int ZeroRow(int r, int n)

{

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

if (fabs(mat[r][i]) >= epsillon)

return 0;

return 1;

}

void AddRow(double a, int source, int target, int n)

{

for (int i = 0; i < n + 1; i++)

mat[target][i] += a \* mat[source][i];

}

void ChangeRow(int source, int target, int n)

{

double temp;

for (int i = 0; i < n + 1; i++)

{

temp = mat[target][i];

mat[target][i] = mat[source][i];

mat[source][i] = temp;

}

}

int Solve(int n)

{

int i, j;

int flag = 0;

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

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

{

j = i;

for (j = i; (fabs(mat[j][i]) < epsillon) && (j < n); j++);

if ((j == n) && mark[i-1])

j = i - 1;

if (j == n)

{

flag = 1;

mark[i] = 1;

}

if ((j != i) && (j != n))

ChangeRow(j, i, n);

if (j != n)

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

if (j != i)

AddRow((-mat[j][i] / mat[i][i]), i, j, n);

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

if (fabs(mat[j][i]) < epsillon)

mat[j][i] = 0;

}

if (flag == 1)

{

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

if (ZeroRow(i, n))

{

if ((fabs(mat[i][n]) >= epsillon))

{

mark[i]++;

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

if(!mark[m] && mat[m][i])

mark[m] = 2;

flag = 2;

}

else

{

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

if (!mark[m] && mat[m][i])

mark[m] = 1;

}

}

return flag;

}

return 0;

}

int main()

{

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

mat[0][0] = 1; mat[0][1] = 1; mat[0][2] = 0; mat[0][3] = 1;mat[0][4] = 0;

mat[1][0] = 1; mat[1][1] = 2; mat[1][2] = 0; mat[1][3] = 2;

mat[1][4] = 0;

mat[2][0] = 1; mat[2][1] = 3; mat[2][2] = 0; mat[2][3] = 3;

mat[2][4] = 0;

int n = 3;

int res = Solve(n);

int flag = 0;

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

{

switch (mark[i])

{

case 0: cout << "X(" << i << ") = " << mat[i][n] / mat[i][i] << endl; break;

case 1: cout << "X(" << i << ") has infinite Answers . . ." << endl;

flag >?= 1; break;

case 2: cout << "X(" << i << ") has no answers . . ." << endl;

flag >?= 2; break;

}

}

cout << "So the equation has ";

switch (flag)

{ case 0: cout << "a Unique answer" << endl; break;

case 1: cout << "infinite answers" << endl; break;

case 3: cout << "no answers" << endl; break;

}

return 0;

}

### Fast Fourier transform-O( N log N )

/\*

int n

n = A ^ 2

double theta

Angle of rotation. At the time of sequential conversion 2\*PI/n is appointed,

at the time of adverse change exchanging -2\*PI/n is appointed.

Complex a []

The arrangement which it converts. The result is converted with in place.

In case of adverse change exchanging

the result furthermore it is necessary 1/n-tuple to do.

\*/

const double PI = 4.0\*atan (1.0);

typedef complex<double> Complex;

const Complex I (0, 1);

void fft (int n, double theta, Complex a[]) {

for (int m = n; m >= 2; m >>= 1) {

int mh = m >> 1;

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

Complex w = exp (i\*theta\*I);

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

int k = j + mh;

Complex x = a [j] - a [k];

a [j] += a [k];

a [k] = w \* x;

}

}

theta \*= 2;

}

int i = 0;

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

for (int k = n >> 1; k > (i ^= k); k >>= 1);

if (j < i) swap (a [i], a [j]);

}

}

### 

### Simplex Method

// UVA 10498, Happiness

#include <iostream>

#include <cstdio>

#include <cmath>

#include <vector>

using namespace std;

typedef vector<double> array;

typedef vector<array> matrix;

const double EPS = 1e-8;

enum {OPTIMAL, UNBOUNDED, NOSOLUTION and UNKNOWN};

struct two\_stage\_simplex {

int N, M, st;

matrix a;

vector<int> s;

two\_stage\_simplex (const matrix &A, const array &b and const array &c)

: N (A.size ())M (A [0] .size ())a (N+2 and array (M+N+1))s (N+2), st (UNKNOWN) {

for (int j = 0; j < M; ++j) a [N+1] [j] = c [j]; // make simplex table

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

for (int j = 0; j < M; ++j) a [i+1] [j] = A [i] [j];

for (int i = 0; i < N; ++i) a [i+1] [M+N] = b [i]; // add helper table

for (int i = 0; i < N; ++i) a [0] [i+M] = 1;

for (int i = 0; i < N; ++i) a [i+1] [i+M] = 1;

for (int i = 0; i < N; ++i) s [i+1] = i+M;

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

for (int j = 0; j <= N+M; ++j) a [0] [j] += a [i] [j];

st = solve ();

}

int status () const {return st; }

double solution () const {return - a [0] [M]; }

double solution (array &x) const {

x.resize (M and 0);

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

x [s [i+1]] = a [i+1] .back ();

return - a [0] [M];

}

int solve () {

M += N; N += 1;

solve\_sub (); // solve stage one

if (solution () > EPS) return NOSOLUTION;

N - = 1; M - = N;

swap (a [0], a.back ()); a.pop\_back (); // modify table

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

swap (a [i] [M], a [i] .back ());

a [i] .resize (M+1);

}

return solve\_sub (); // solve stage two

}

int solve\_sub () {

int p and q;

while (1) {

//print ();

for (q = 0; q <= M && a [0] [q] >= - EPS; ++q);

for (p = 0; p <= N && a [p] [q] <= EPS; ++p);

if (q >= M || p > N) break;

for (int i = p+1; i <= N; ++i) // bland's care for cyclation

if (a [i] [q] > EPS)

if (a [i] [M] /a [i] [q] < a [p] [M] /a [p] [q] ||

(a [i] [M] /a [i] [q] == a [p] [M] /a [p] [q] && s [i] < s [q])) p = i;

pivot (p and q);

}

if (q >= M) return OPTIMAL;

else return UNBOUNDED;

}

void pivot (int p and int q) {

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

for (int k = M; k >= 0; --k)

if (j! = p && k! = q)

a [j] [k] - = a [p] [k] \*a [j] [q] /a [p] [q];

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

if (j! = p) a [j] [q] = 0;

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

if (k! = q) a [p] [k] = a [p] [k] /a [p] [q];

a [p] [q] = 1.0;

s [p] = q;

}

};

int main () {

for (int n, m; cin >> n >> m; ) {

array c (n+m), b (m);

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

cin >> c [i], c [i] \*= -1;

matrix A (m, array (n+m));

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

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

cin >> A [i] [j];

A [i] [n+i] = 1;

cin >> b [i];

}

two\_stage\_simplex tss (A and b, c);

double ans = - tss.solution () \* m;

printf (“Nasa can spend %.0f taka.¥n” and ans + 0.5 - EPS);

}

}

# Some More

## Base Element

#include <iostream>

#include <vector>

using namespace std;

#define REP (i, n) for (int i=0; i< (int) n; ++i)

#define FOR (i, c) for (\_\_typeof ((c) .begin ())i= (c) .begin (); i! = (c) .end (); ++i)

#define ALL (c) (c) .begin (), (c) .end ()

## Double Exact Comparison

#include <cmath>

using namespace std;

#define EPS 1e-9

double inline les( double a, double b ) { //a < b

return a + EPS < b;

}

double inline lesQ( double a, double b ) { // a <= b

return a - EPS < b;

}

double inline gre( double a, double b ) { // a > b

return a - EPS > b;

}

double inline greQ( double a, double b ) { // a >= b

return a + EPS > b;

}

double inline eq( double a, double b ) { // a == b

return fabs( a - b ) < EPS;

}

## 2-Sat

#include<iostream>

#include<vector>

using namespace std;

//fill the v array

//e.g. to push (p v !q) use the following code:

// v[VAR(p)].push\_back( NOT( VAR(q) ) )

// v[NOT( VAR(q) )].push\_back( VAR(p) )

//the result will be in color array

#define VAR(X) (X << 1)

#define NOT(X) (X ^ 1)

#define CVAR(X,Y) (VAR(X) | (Y))z

#define COL(X) (X & 1)

#define NVAR 400

int n;

vector<int> v[2 \* NVAR];

int color[2 \* NVAR];

int bc[2 \* NVAR];

bool dfs( int a, int col ) {

color[a] = col;

int num = CVAR( a, col );

for( int i = 0; i < v[num].size(); i++ ) {

int adj = v[num][i] >> 1;

int ncol = NOT( COL( v[num][i] ) );

if( ( color[adj] == -1 && !dfs( adj, ncol ) ) ||

( color[adj] != -1 && color[adj] != ncol ) ) {

color[a] = -1;

return false;

}

}

return true;

}

bool twosat() {

memset( color, -1, sizeof color );

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

if( color[i] == -1 ){

memcpy(bc, color, sizeof color);

if( !dfs( i, 0 )){

memcpy(color, bc, sizeof color);

if(!dfs( i, 1 ))

return false;

}

}

}

return true;

}

## Ternary Search

min = a;

max = b;

while(max - min > epsilon){

g = min + (max-min)/3;

h = min + 2\*(max-min)/3;

if(f(g) < f(h))

max = h;

else

min = g;

}

return (max+min)/2;

## Binary Search ();

// returns the index if the num is found in arr[li]..[ri],else -1

int bSearch(int arr[],int li,int ri,int num){

while(li<=ri){

if( num == arr[(li+ri)/2] ){

return (li+ri)/2;

}

if(num < arr[(li+ri)/2]){

ri=(li+ri)/2-1;

}

else{

li=(li+ri)/2+1;

}

}

return -1;

}

// finds lowest value of func which is greater than or equal to val

int l = 0, r = MAX - 1;

while( l < r ) {

int mid = ( l + r ) / 2;

if( func( mid ) >= val )

r = mid;

else

l = mid + 1;

}

// finds greatest value of func which is lower than or equal to val

int l = 0, r = MAX - 1;

while( l < r ) {

int mid = ( l + r + 1 ) / 2;

if( func( mid ) <= val )

l = mid;

else

r = mid - 1;

}

دوست های صميمی .... کار های قديمی ..... اينجا ما همينيم .......