FFT Extended + OP:

#define MAXN 1048576 /// 2 \* MAX at least

int len,last=-1,step=0,rev[MAXN];long long C[MAXN],D[MAXN],P[MAXN],Q[MAXN];

struct complx{

long double real, img;

inline complx(){real = img = 0.0;}

inline complx conjugate(){return complx(real, -img);}

inline complx(long double x){real = x, img = 0.0;}

inline complx(long double x, long double y){real = x, img = y;}

inline complx operator +(complx x){return complx(real+x.real,img+x.img);}

inline complx operator -(complx x){return complx(real-x.real,img-x.img);}

inline complx operator \*(complx x){

return complx((real\*x.real)-(img\*x.img),(real\*x.img)+(img\*x.real));}

} u[MAXN], v[MAXN], f[MAXN], g[MAXN], dp[MAXN], inv[MAXN];

void build(int& a, long long\* A, int& b, long long\* B){

while (a > 1 && A[a - 1] == 0) a--;

while (b > 1 && B[b - 1] == 0) b--;

len = 1<<(32-\_\_builtin\_clz(a+b)-(\_\_builtin\_popcount(a+b)==1));

for (int i = a; i < len; i++) A[i] = 0;

for (int i = b; i < len; i++) B[i] = 0;

if (!step++){

dp[1] = inv[1] = complx(1);

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

double theta = (2.0 \* acos(0.0)) / (1 << i);

complx mul = complx(cos(theta), sin(theta));

complx inv\_mul = complx(cos(-theta), sin(-theta));

int lim = 1 << i;

for (int j = lim >> 1; j < lim; j++){

dp[2 \* j] = dp[j], inv[2 \* j] = inv[j];

inv[2\*j+1]=inv[j]\*inv\_mul,dp[2\*j+1]=dp[j]\*mul;}}}

if (last != len){

last = len;

int bit = (32-\_\_builtin\_clz(len)-(\_\_builtin\_popcount(len)==1));

for (int i=0;i<len;i++) rev[i]=(rev[i>>1]>>1)+((i&1)<<(bit-1));}

}

void transform(complx \*in, complx \*out, complx\* ar){

for (int i = 0; i < len; i++) out[i] = in[rev[i]];

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

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

for (int j = 0; j < k; j++){

complx z = out[i + j + k] \* ar[j + k];

out[i + j + k] = out[i + j] - z, out[i+j]=out[i+j]+z;}}

}}

int multiply(int a,LL\* A, int b,LL\* B){/// Returns multiplication in A

build(a, A, b, B);

for (int i = 0; i < len; i++) u[i] = complx(A[i], B[i]);

transform(u, f, dp);

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

int j = (len - 1) & (len - i);

u[i] = (f[j]\*f[j]-f[i].conjugate()\*f[i].conjugate())\*complx(0,-0.25/len);

}

transform(u, f, dp);

for (int i = 0; i < len; i++) A[i] = f[i].real + 0.5;

return a + b - 1;}

int mod\_multiply(int a, long long\* A, int b, long long\* B, int mod){

build(a, A, b, B);

int flag = equals(a, A, b, B);

for (int i = 0; i < len; i++) A[i] %= mod, B[i] %= mod;

for (int i = 0; i < len; i++) u[i] = complx(A[i] & 32767, A[i] >> 15);

for (int i = 0; i < len; i++) v[i] = complx(B[i] & 32767, B[i] >> 15);

transform(u, f, dp);

for (int i = 0; i < len; i++) g[i] = f[i];

if (!flag) transform(v, g, dp);

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

int j = (len - 1) & (len - i);

complx c1 = f[j].conjugate(), c2 = g[j].conjugate();

complx a1=(f[i]+c1)\*complx(0.5,0),complx a2=(f[i]-c1)\*complx(0,-0.5);

complx b1=(g[i]+c2)\*complx(0.5/len,0),complx b2=(g[i]-c2)\*complx(0,-0.5/len);

v[j] = a1 \* b2 + a2 \* b1,u[j] = a1 \* b1 + a2 \* b2 \* complx(0, 1);

}

transform(u, f, dp);transform(v, g, dp);long long x, y, z;

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

x = f[i].real + 0.5, y = g[i].real + 0.5, z = f[i].img + 0.5;

A[i] = (x + ((y % mod) << 15) + ((z % mod) << 30)) % mod;}

return a + b - 1;}

/// Multiplies two polynomials where intermediate and final values fits in LL

int long\_multiply(int a, long long\* A, int b, long long\* B){

int mod1 = 1.5e9;int mod2 = mod1 + 1;

for (int i = 0; i < a; i++) C[i] = A[i];

for (int i = 0; i < b; i++) D[i] = B[i];

mod\_multiply(a, A, b, B, mod1);mod\_multiply(a, C, b, D, mod2);

for (int i=0;i<len;i++){A[i]=A[i]+(C[i]-A[i]+(LL)mod2)\*(LL)mod1%mod2\*mod1;}

return a + b - 1;}

int build\_convolution(int n, long long\* A, long long\* B){

int i, m, d = 0;

for (i = 0; i < n; i++) Q[i] = Q[i + n] = B[i];

for (i = 0; i < n; i++) P[i] = A[i], P[i + n] = 0;

n \*= 2, m = 1 << (32 - \_\_builtin\_clz(n)-(\_\_builtin\_popcount(n)==1));

for (i = n; i < m; i++) P[i] = Q[i] = 0;

return n;}

void convolution(int n,LL\* A,LL\* B){/// Returns circular conv of A and B in A

int len = build\_convolution(n, A, B); /// A and B must be of equal size

multiply(len, P, len, Q); /// c0 = a0b0 + a1b4 + a2b3 + a3b2 + a4b1

for (int i=0;i<n;i++) A[i]=P[i+n];/// c1 = a0b1 + a1b0 + a2b4 + a3b3 + a4b2

} /// c4 = a0b4 + a1b3 + a2b2 + a3b1 + a4b0

void mod\_convolution(int n, long long\* A, long long\* B, int mod){

int len = build\_convolution(n, A, B);mod\_multiply(len, P, len, Q, mod);

for (int i = 0; i < n; i++) A[i] = P[i + n];}

void long\_convolution(int n, long long\* A, long long\* B){

int len = build\_convolution(n, A, B);long\_multiply(len, P, len, Q);

for (int i = 0; i < n; i++) A[i] = P[i + n];}

///Hamming distance vector of B with every substring of length |pattern| in str

vector <int> hamming\_distance(const char\* str, const char\* pattern){

int n = strlen(str), m = strlen(pattern);

for (int i = 0; i < n; i++) P[i] = Q[i] = 0;

for (int i = 0; i < n; i++) P[i] = str[i] == '1' ? 1 : -1;

for (int i=0,j=m-1;j>=0;i++,j--)Q[i]=pattern[j]=='1'?1:-1;

vector<int>res;///str="01111000010011111111110010001101000100011110101111"

fft::multiply(n, P, m, Q); /// pattern = "1001101001101110101101000"

for (int i=0;(i+m)<=n;i++){res.push\_back(m-((P[i+m-1]+m)>>1));}

return res;}/// Sum of values in hamming distance vector = 321

Chinese Remainder Theorem:

LL CRT(int n,LL\* ar,LL\* mods){

int i, j;long long x, y, res = 0, M = 1;

for (i = 0; i < n; i++) M \*= mods[i];

for (i = 0; i < n; i++){ /// mods must be pairwise co-prime

x = M / mods[i],y = mod\_inverse(x, mods[i]);

res = (res + (((x \* ar[i]) % M) \* y)) % M;}

return res;}/// unique solution x modulo M(product of mods)for which x%mods[i]=ar[i]

Prime Counting Function:

#define MAXN 100 #define MAXM 100010 #define MAXP 666666 #define MAX 10000010

long long dp[MAXN][MAXM];

void init(){sieve(); /// must call init()

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

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

if (!n) dp[n][m] = m;

else dp[n][m] = dp[n - 1][m] - dp[n - 1][m / primes[n - 1]];}}}

long long phi(long long m,int n){///nums<=m not divisible by first m primes

if (n == 0) return m;if (primes[n - 1] >= m) return 1;

if (m<MAXM&&n<MAXN) return dp[n][m]; /// counter[i]=num of primes<=i

if ((long long)primes[n-1]\*primes[n-1]>=m&&m<MAX) return counter[m]-n+1;

if ((long long)primes[n - 1] \* primes[n - 1] \* primes[n - 1]>=m&&m<MAX){

int lim = counter[(int)sqrt(m + 0.95)];

long long res = counter[m] - (((lim + n - 2) \* (lim - n + 1 ))>>1);

for (int i = n; i < lim; i++) res += counter[m / primes[i]];

return res;}

return phi(m, n - 1) - phi(m / primes[n - 1], n - 1);}

long long Lehmer(long long m){ /// number of primes <= m

if (m < MAX) return counter[m];

long long w, res = 0;int i, a, s, c, x, y;

s = sqrt(0.9 + m), y = c = cbrt(0.9+m),a=counter[y],res=phi(m,a)+a-1;

for (i=a;primes[i]<=s;i++) res=res-Lehmer(m/primes[i])+Lehmer(primes[i])-1;

return res;}

Gaussin elimination on Band Matrix:

#define MAXM 22 #define MAXN 10002 /// Complexity: O(n\*m^3),n=no of rows, m=no of cols

#define cell\_num(i, j) (m \* (n - (i)) - (j) - 1)

#define valid(i, j) ((i) >= 0 && (i) < n && (j) >= 0 && (j) < m)

const int dx[] = {1, 0, -1, 0};const int dy[] = {0, 1, 0, -1};int n, m;

double band[MAXN \* MAXM][2 \* MAXM + 1];

double rhs[MAXN\*MAXM],aux[MAXN\*MAXM];///rhs=RH side cnstants before gauss,and sol after

void init(int n, int m){

int i, j, q = n \* m, d = 2 \* m + 1;clr(rhs);

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

for (j = 0; j < d; j++){band[i][j] = 0.0;}

band[i][m] = 1.0;}}

/// adding p to gauss\_matrix[r][c] (implicit) where r = cell\_num[i][j] and c = cell\_num[k][l]

void add(int i, int j, int k, int l, double p){

int u = cell\_num(i, j), v = cell\_num(k, l) - u + m;

band[u][v] += p;}/// gauss\_matrix doesnt exist,the coordinates converted to band matrix

void gauss(int n, int m){

double x, y;int i, j, k, l, d, u, v, q = n \* m, r = 2 \* m + 1;

for (i = 0; i < n; i++){ /// Forward Elimination

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

d = i \* m + j;x = band[d][m], rhs[d] /= x;

for (k = 0; k <= m && (d + k) < q; k++) band[d][m + k] /= x;

for (l = 1; l <= m && (d + l) < q; l++){

x = band[d + l][m - l], rhs[d + l] -= (x \* rhs[d]);

for (k = 0; k <= m && (d + k) < q; k++){

band[d + l][m - l + k] -= (x \* band[d][m + k]);

}}

}}

for (i = 0; i < q; i++) aux[i] = rhs[i], rhs[i] = 0.0;

for (i = 0; i < n; i++){ /// Backward substitution

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

x = 0.0, u = cell\_num(i, j);

for (v = 0; v < r; v++) x += (band[u][v] \* rhs[v + u - m]);

rhs[u] = aux[u] - x;}}}

int main(){scanf("%d %d", &n, &m);init(n, m);return 0;}

Java + python:

from fractions import Fraction ||| sys.stdin = open("input.txt")

hashmap = {Fraction(-111111, 777777) : 666} ||| n, k = input().split()

dp = [[Fraction(0, 1) for col in range(102)] for row in range(102)]

from decimal import \* ||| Decimal(n).sqrt() ||| getcontext().prec = 17000

Random rand = new Random(); rand.nextInt(r - l + 1); Integer.bitCount(x);

res=res.add(A[i][k].multiply(B[k][j])).setScale(SCALE,BigDecimal.ROUND\_HALF\_UP);///SCALE=100

java.awt.\*; java.awt.geom.\*; java.awt.Polygon.\*; java.math.\*; java.lang.\*;java.lang.Object.\*;

pbstt Line2D[]ar=new Line2D[25];ar[i]=new Line2D.Double(x1,y1,x2,y2);ar[i].intersectsLine(ar[j])

TreeMap dis=new TreeMap();Queue<String>Q=new LinkedList<String>();Q.add(src);dis.put(src,0);

Q.poll();int d=(int)dis.get(cur);Polygon[]poly=new Polygon[50];int[]u=poly[i].xpoints;

poly[i].addPoints(x,y);if (poly[i].contains(x[k],y[k]));int n = poly[i].npoints;

Line2D line = new Line2D.Double(u[l], v[l], u[k], v[k]);line.ptSegDist(x,y);

Miller Rabin:

bitset <MAX> flag;long double op = 0.0;int p = 0, prime[78777];

const int base[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022};

void init(){if (!flag[2]) Sieve();}

inline long long mul(long long a, long long b, long long MOD){

if ((MOD < 3037000500LL)) return ((a \* b) % MOD);

long double res = a;res \*= b;long long c = (long long)(res \* op);

a \*= b;a -= c \* MOD;if (a >= MOD) a -= MOD;if (a < 0) a += MOD;

return a;}

inline long long expo(long long x, long long n, long long m){

long long res = 1;

while (n){if (n & 1) res = mul(res, x, m);x = mul(x, x, m);n >>= 1;}

return (res % m);}

inline bool miller\_rabin(long long p){

if (p < MAX) return flag[p];if ((p + 1) & 1) return false;

for (int i = 1; i < 10; i++){if (!(p % prime[i])) return false;}

long long a, m, x, s = p - 1, y = p - 1;

op = (long double)1 / p, s = s >> \_\_builtin\_ctzll(s);

for (int i = 0; i < 7; i++) {

x = s, a = (base[i] % y) + 1;m = expo(a, x, p);

while ((x != y) && (m != 1) && (m != y)) m = mul(m, m, p), x <<= 1;

if ((m != y) && !(x & 1)) return false;}

return true;}

Thomas Algorithm:

struct equation{/// Equation of the form: (x\_prev\*l)+(x\_cur\*p)+(x\_next\*r)=rhs

LD l, p, r, rhs;equation(){}

equation(LD l, LD p, LD r, LD rhs = 0.0):l(l), p(p), r(r), rhs(rhs){}};

vector <LD> thomas\_algorithm(int n, vector <struct equation> ar){ /// O(n)

ar[0].r = ar[0].r / ar[0].p;ar[0].rhs = ar[0].rhs / ar[0].p;

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

LD v = 1.0 / (ar[i].p - ar[i].l \* ar[i - 1].r);ar[i].r = ar[i].r \* v;

ar[i].rhs = (ar[i].rhs - ar[i].l \* ar[i - 1].rhs) \* v;}

for (int i = n - 2; i >= 0; i--) ar[i].rhs = ar[i].rhs - ar[i].r \* ar[i + 1].rhs;

vector <LD> res;for (int i = 0; i < n; i++) res.push\_back(ar[i].rhs);return res;}

2SAT:

vector<int> adj[MAX], rev[MAX];///1 based

int n,m,l,dfs\_t[MAX],order[MAX],par[MAX],visited[MAX];/// MAX=2\*MAX

int neg(int x){return((x)<=n?(x+n):(x-n));}

void init(int nodes){n=nodes,m=nodes\*2;for(int i=0;i<MAX;i++)adj[i].clear(),rev[i].clear();}

void add\_implication(int a,int b){/// a->b

if(a<0) a=n-a;if(b<0) b=n-b;

adj[a].push\_back(b);rev[b].push\_back(a);}

void add\_or(int a,int b){add\_implication(-a,b);add\_implication(-b,a);}

void add\_xor(int a,int b){add\_or(a, b);add\_or(-a,-b);}

void add\_and(int a,int b){add\_or(a, b);add\_or(a,-b);add\_or(-a,b);}

void force\_true(int x){if(x <0) x = n - x;add\_implication(neg(x), x);}

void force\_false(int x){if(x <0) x = n - x;add\_implication(x, neg(x));}

void topsort(int i){

visited[i]=true;

for(int j =0; j < rev[i].size(); j++){if(!visited[rev[i][j]]) topsort(rev[i][j]);}

dfs\_t[i]=++l;}

void dfs(int i,int p){

par[i]= p, visited[i]=true;

for(int j =0; j < adj[i].size(); j++){if(!visited[adj[i][j]]) dfs(adj[i][j], p);}}

void build(){

clr(visited);

for(int i = m, l =0; i >=1; i--){

if(!visited[i]) topsort(i);

order[dfs\_t[i]]= i;

}clr(visited);

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

if(!visited[order[i]])dfs(order[i],order[i]);}}

bool satisfy(vector<int>& res){/// returns nodes set to true in res

build();clr(visited);res.clear();

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

int x = order[i];if(par[x]== par[neg(x)])returnfalse;

if(!visited[par[x]]){visited[par[x]]=true;visited[par[neg(x)]]=false;}

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

if(visited[par[i]]) res.push\_back(i);}

returntrue;}

Aho Corasick:

#define LOG 19#define LET 26#define MAX 300010

struct aho\_corasick{

int id, edge[256];vector<longlong> counter;

vector<string>dictionary;vector<map<char,int>> trie;vector<int>leaf,fail,dp[LET];

int node(){

leaf.push\_back(0);counter.push\_back(0);trie.push\_back(map<char,int>());

return id++;}

int size(){return dictionary.size();}

void clear(){

trie.clear(), dictionary.clear(),fail.clear(), leaf.clear(), counter.clear();

for(int i =0; i < LET; i++) dp[i].clear();

id =0, node();

for(int i='a';i<='z';i++) edge[i]= i-'a';/// modify?

}

aho\_corasick(){clear();}

inlinevoid insert(constchar\* str){

int j, x, cur =0;

for(j =0; str[j]!=0; j++){

x = edge[str[j]];

if(!trie[cur].count(x)){int next\_node = node();trie[cur][x]= next\_node;}

cur = trie[cur][x];}

leaf[cur]++;dictionary.push\_back(str);}

inlinevoid build(){/// remember to call build

vector<pair<int, pair<int,int>>> Q;

fail.resize(id,0), Q.push\_back({0,{0,0}});

for(int i =0; i < LET; i++) dp[i].resize(id,0);

for(int i =0; i < id; i++){for(int j =0; j < LET; j++){dp[j][i]= i;}}

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

char c = Q[i].second.second;int u = Q[i].first, p = Q[i].second.first;

for(auto&it:trie[u])Q.push\_back({it.second,{u,it.first}});

if(u){

int f = fail[p];

while(f &&!trie[f].count(c)) f = fail[f];

if(!trie[f].count(c)|| trie[f][c]== u) fail[u]=0;

else fail[u]= trie[f][c];

counter[u]= leaf[u]+ counter[fail[u]];

for(int j=0;j<LET;j++){if(u&&!trie[u].count(j))dp[j][u]=dp[j][fail[u]];}

}}}

int next(int cur,char ch){

int x = edge[ch];cur = dp[x][cur];

if(trie[cur].count(x)) cur = trie[cur][x];

return cur;}

longlong count(constchar\* str){

longlong res =0;/// total occurrence of all words in str

for(int j =0,cur=0;str[j]&&id>1;j++){cur=next(cur,str[j]),res+=counter[cur];}

return res;}

};

struct dynamic\_aho{/// dynamic aho corasick in N log N

aho\_corasick ar[LOG];

dynamic\_aho(){for(int i =0; i < LOG; i++) ar[i].clear();}

inlinevoid insert(constchar\* str){

int i, k =0;

for(k =0; k < LOG && ar[k].size(); k++){}

for(i =0, ar[k].insert(str); i < k; i++){

for(auto s: ar[i].dictionary){ar[k].insert(s.c\_str());}

ar[i].clear();}

ar[k].build();}

LL count(constchar\* str){LL res=0;for(int i=0;i<LOG;i++)res+=ar[i].count(str);return res;}};

Algorith X:

#define LIM 100010#define MAXR 100010#define MAXC 100010

namespace dlx{///Rows satisfy cols, select rows st cols satisfied exactly once, call init first

int row[LIM],col[LIM],L[LIM],R[LIM],U[LIM],D[LIM];

int n, idx, len, selected\_rows[MAXR], column\_count[MAXC];

void init(int ncolumn){/// ncolumn = total number of columns(1 based)

clr(column\_count), n = ncolumn, idx = n +1;

for(int i =0; i <= n; i++) U[i]= D[i]= i, L[i]= i -1, R[i]= i +1,L[0]=n,R[n]=0;}

void addrow(int r,vector<int>&columns){/// r = 1 base index of row

int i, c, l = columns.size(), first = idx;

for(i =0; i < l; i++){/// vector contains index of columns for row r

c=columns[i], L[idx]=idx-1, R[idx]=idx+1, D[idx]=c, U[idx]=U[c];

D[U[c]]= idx, U[c]= idx, row[idx]= r, col[idx]= c, column\_count[c]++, idx++;

}/// column\_count[c] = number of rows satisfying column c

R[idx -1]= first, L[first]= idx -1;}

inlinevoid remove(int c){/// remove column c

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

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

for(int j = R[i]; j != i; j = R[j]){column\_count[col[j]]--;U[D[j]]= U[j], D[U[j]]= D[j];}}}

inlinevoid restore(int c){/// restores column c

for(int i = U[c]; i != c; i = U[i]){

for(int j = L[i]; j != i; j = L[j]){

column\_count[col[j]]++;U[D[j]]= j, D[U[j]]= j;L[R[c]]= c, R[L[c]]= c;}}}

bool algorithmX(int depth){

if(R[0]==0){len = depth;returntrue;}

int i, j, c = R[0], flag =0;

for(i=R[0];i!=0;i=R[i]){/// select col deterministically

if(column\_count[i]< column\_count[c]) c = i;}

remove(c);

for(i = D[c]; i != c &&!flag; i = D[i]){

selected\_rows[depth]= row[i];

for(j = R[i]; j != i; j = R[j]) remove(col[j]);

flag|=algorithmX(depth+1);/// select rows with rnd\_shuffle?

for(j = L[i]; j != i; j = L[j]) restore(col[j]);}

restore(c);

return flag;}

bool exact\_cover(vector<int>& rows){

rows.clear();/// Returns subset of rows in vector rows

if(!algorithmX(0))returnfalse;

for(int i=0; i<len; i++) rows.push\_back(selected\_rows[i]);

returntrue;

}}

Alpha Beta pruning:

///alpha = maximum score maximizing player (p=0) is assured so far

///beta = minimum score minimizing player (p=1) is assured so far

int backtrack(node cur,int i,int p,int alpha,int beta){

if(i == n)return evaluate(cur);

vector<struct node>ar=transition(cur,deck[i],p);

for(int j =0; j < ar.size(); j++){

int x = backtrack(ar[j], i +1, p ^1, alpha, beta);

if(p==0&&x>alpha) alpha=x;/// First player maximizes

if(p==1&&x<beta) beta=x;/// Second player minimizes

if(alpha >= beta)break;}

return(p ==0)? alpha : beta;}

int main(){backtrack(start,0,0,-INF, INF);}

Barlekamp Massey:

#define MOD 1000000007/// MOD prime and > 2

structMatrix{/// namespace bbl

int n, ar[MAXN][MAXN];

Matrix(){}

Matrix(int dim,int diag =0){clr(ar);n = dim;for(int i =0; i < n; i++) ar[i][i]= diag;}

Matrixoperator\*(constMatrix& mat)const{

Matrix res(n);

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

for(int j =0; j < mat.n; j++){

unsignedlonglong x =0;

for(int k=0;k<n;k++)x=(x+(ULL)ar[i][k]\*mat.ar[k][j])%MOD;

res.ar[i][j]= x;}}

return res;}

Matrixoperator^(longlong n)const{

Matrix x =\*this, res =Matrix(this->n,1);

while(n){if(n &1){res = res \* x;}n = n >>1, x = x \* x;}

return res;}

};

int mod\_inverse(int x){

int u =1, v =0, t =0, a = x, b = MOD;

while(b){t = a/b, a -=(t\*b), u -=(t\*v);swap(a, b), swap(u, v);}

return(u + MOD)% MOD;}

int convolution(constint\* A,constint\* B,int n){

unsignedlonglong res =0;

for(int i=0;i<n;i++)res=(res+(ULL)A[i]\*B[i])%MOD;

return res % MOD;}

int berlekamp\_massey(vector<int> S,vector<int>& C){

assert(S.size()&&(S.size()%2)==0);/// O(n^2)

int n = S.size();

vector<int> T, B(n +1,0);

C.assign(n +1,0), reverse(S.begin(), S.end());

int i, j, k, d, x, l =0, m =1, b =1, deg =0;

for(i =0, C[0]=1, B[0]=1; i < n; i++){

d = S[n - i -1];

if(l >0) d =(d+convolution(&C[1],&S[n-i],l))%MOD;

if(d ==0) m++;

else{

if(l\*2<=i) T.assign(C.begin(),C.begin()+l+1);

x =(((longlong)mod\_inverse(b)\*(MOD-d))%MOD+MOD)%MOD;

for(j =0; j <= deg; j++){

C[m+j]=(C[m+j]+(unsignedlonglong)x\*B[j])%MOD;}

if(l \*2<= i){B.swap(T);deg = B.size()-1, b = d, m =1, l = i - l +1;}

else m++;

}}

C.resize(l +1);

return l;}

longlonginterpolate(vector<int>& recurrence,longlong n){

int len = recurrence.size();vector<int> polynomial;

if(n < len)return recurrence[n];///nth term of recurrence,recurrence and n both 0 base

int l = berlekamp\_massey(recurrence, polynomial);

reverse(polynomial.begin(), polynomial.begin()+ l +1);

structMatrix mat =Matrix(l);

for(int i=1;i<mat.n;i++)mat.ar[i][i -1]=1;

for(int i=0;i<mat.n;i++)mat.ar[0][i]=MOD-polynomial[l-i -1];

longlong res =0; mat = mat ^(n - len +1);

for(int i =0; i < mat.n; i++)res=(res+(longlong)mat.ar[0][i]\*recurrence[len-i-1])%MOD;

return res;}

Bitstring LCS:

bool flag[MAX];char A[MAX], B[MAX], S[2][MAX], flag[MAX];

int lcs(char\* A,char\* B){

int i, j, k, v, n, m, res =0;unsignedlonglong x, t, q, y, mask[128]={0};

for(n =0, clr(flag); A[n]; n++) S[0][n]= A[n];

for(m =0; B[m]; m++) S[1][m]= B[m];

for(i =0;(i \*64)< m; i++){

for(k=0,clr(mask);k<64&&(i\*64+k)< m; k++){mask[S[1][i \*64+ k]]|=(1ULL<< k);}

for(j =0, x =0; j < n; j++){

t = mask[S[0][j]]&~x,x|=t,v=flag[j];q = x -(t <<1)- v, y =(q &~x)|t;

flag[j]= y >>63, x &=~(y <<1);

if(v) x &=~1ULL;}

res += \_\_builtin\_popcountll(x);}

return res;}

DiscreteLogarithm:

int extended\_gcd(int a,int b,int& x,int& y){/// Bezout's identity,ax+by=gcd(a,b)

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

int g = extended\_gcd(b, a % b, y, x);

y -=((a / b)\* x);

return g;}

int discrete\_log(int g,int h,int p){/// returns smallest x st (g^x)%p=h,-1 if none

if(h >= p)return-1;

if((g % p)==0){

if(h==1)return0;/// -1 if only positive req

elsereturn-1;}

int i, c, x, y, z, r, m, counter =0;

longlong v =1, d =1, mul =1, temp =1% p;

for(int i =0; i <100; i++){

if(temp == h)return i;

temp =(temp \* g)% p;}

while((v = \_\_gcd(g, p))>1){

if(h % v)return-1;

h/=v,p/=v,d=(d\*(g/v))%p,counter++;}

m = ceil(sqrtl(p))+1;tr1::unordered\_map <int,int> mp;

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

if(!mp[mul]) mp[mul]= i +1;

mul =(mul \* g)% p;}

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

z = extended\_gcd(d, p, x, y),c=p/z;r =((((longlong)x\*h)/z)%p+p)%p;

if(mp[r])return((i\*m)+mp[r]+counter-1);

d =(d \* mul)%p;}

return-1;}

FaulhabersFormula:

void generate(){

for(int i=0; i<MAX;i++) inv[i]=expo(i,MOD-2);

clr(S), S[0][0]=1;

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

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

S[i][j]=((longlong)S[i-1][j]\*j+S[i-1][j-1])%MOD;}}}

int faulhaber(longlong n,int k){

n %= MOD;if(!k)return n;

longlong res =0, p =1;

for(int j =0; j <= k; j++){

p =(p \*(n +1- j))% MOD;

res=(res+(((S[k][j]\*p)%MOD)\*inv[j+1]))%MOD;}

return(res % MOD);}

2D BIT:

structFenwick2D{longlong n, m, tree[4][MAX][MAX];

Fenwick2D(){}Fenwick2D(int a,int b){clr(tree), n = a, m = b;}

void update(int p,int q,longlong v){/// Add v from upper-left[p,q] to lower-right[n,n]

if((p<0)||(q<0)||(p>n)||(q>m))return;

int i = p, c = p -1, d = q -1;

while(i <= n){

int j = q;

while(j <= m){

tree[0][i][j]+=v,tree[1][i][j]+=v\*d;tree[2][i][j]+=v\*c,tree[3][i][j]+=v\*c\*d;

j +=(j &(-j));}

i +=(i &(-i));}}

/// Sum from upper-left[p,q] to lower-right[n,n]

longlong query(int p,int q){

int i = p, j;

longlong x=0,y=0,z=0,c,d,res;

while(i){

j = q, c = d =0;

while(j){

c+=tree[0][i][j],d+=tree[1][i][j];y+=tree[2][i][j],z+=tree[3][i][j];

j^=(j &(-j));}

i^=(i &(-i)),x+=((c \* q)- d);}

return(x \* p)-(y \* q)+ z;}

/// Add v from upper-left [i,j] to lower-right [k,l]

void update(int i,int j,int k,int l,longlong v){

update(i,j,v),update(k+1,j,-v);update(k+1,l+1,v),update(i,l+1,-v);}

longlong query(int i,int j,int k,int l){

if(i > k || j > l)return0;/// Sum from upper-left [i,j] to lower-right [k,l]

return query(k,l)-query(i-1,l)+query(i-1,j-1)-query(k,j-1);}};

GamblersRuin:

LDB gamblers\_ruin(int n1,int n2,LDB p,LDB q){///First=n1 coins,Second=n2 loser give 1 to winner

if(fabs(p-q)<=1e-9)return(LDB)n2/(n1+n2);///First wins with prob p second with q,p+q=1

LDB x =1.0-expo(q/p,n2);/// Returns prob of 1st winning, lose when coin=0

LDB y =1.0-expo(q/p,n1+n2);/// double bigmod

return(x/y);}

GaussianElimination:

#define EPS 1e-9#define MAX 512

int gauss(int n,int m,double ar[MAX][MAX],vector<double>& res){

res.assign(m,0);/// n=no of equations,m=no of var, all 0 based, ar[i][m]=rhs

vector<int> pos(m,-1);int i, j, k, l, p, free\_var =0;

for(j =0, i =0; j < m && i < n; j++){

for(k = i, p = i; k < n; k++){

if(abs(ar[k][j])>abs(ar[p][j])) p=k;}

if(abs(ar[p][j])> EPS){

for(l=j, pos[j]= i;l<=m;l++)swap(ar[p][l],ar[i][l]);

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

if(k != i){

double x=ar[k][j]/ar[i][j];

for(l=j;l<=m;l++)ar[k][l]-=ar[i][l]\*x;}

}

i++;

}}

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

if(pos[i]==-1) free\_var++;else res[i]=ar[pos[i]][m]/ar[pos[i]][i];}

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

double val =0.0;for(j =0; j < m; j++) val+=res[j]\*ar[i][j];

if(abs(val-ar[i][m])>EPS)return-1;

}

return free\_var;}

GreedyCoinChange:

constlonglong n =10, INF =0x7(15F);

constint coins[]={0,1,5,10,20,50,100,200,500,1000,2000};///1 based

longlong res, counter[44], sum[44];///counter[i]=no of coins type i

void backtrack(int i,longlong p,longlong c){///Complexity 2^n

if(p==0&&c<res)res=c;///Change min to max here

if(i==0||p<=0)return;

longlong k, x =0;

if((p - sum[i -1])> x) x = p - sum[i -1];

k =(x / coins[i])+(x % coins[i]!=0);

if(k<=counter[i]) backtrack(i-1,p-k\*coins[i],c+k);

if(++k<=counter[i])backtrack(i-1,p-k\*coins[i],c+k);/// Do this 5-10 i WA

/// Min coins required to make s from coin values and count of coins

longlong solve(longlong s){/// -1 if no solution

for(int i=1;i<=n;i++)sum[i]=sum[i-1]+coins[i]\*counter[i];

res = INF;backtrack(n, s,0);

return(res==INF)?-1:res;}

Hashing:

for(int i =1; i <= n; i++) H[i]=(H[i-1]\*b+str[i-1]+13)%m;

for(int i = n; i >=1; i--) R[i]=(R[i+1]\*b+str[i-1]+13)%m;

inlineint range\_hash(int l,int r){return(m+H[r+1]-((longlong)P[r-l+1]\*H[l]%m))%m;}

inlineint reverse\_hash(int l,int r){;return(m+R[l+1]-((longlong)P[r-l+1]\*R[r+2]%m))%m;}

while(entry[i]==id&&hashtable[i]!=x)i++;/// hashtable

HopcroftKarp:

namespace hc{///Hopcroft karp in O(m \* sqrt(n)), 0 based index for graphs

vector<int> adj[MAX];

int n,L[MAX],R[MAX],Q[MAX],len[MAX],dis[MAX],parent[MAX],visited[MAX];

void init(int nodes){/// Nof vertex in left set,or max(left,right)

n = nodes, clr(len);for(int i =0; i < MAX; i++) adj[i].clear();}

void add\_edge(int u,int v){len[u]++; adj[u].push\_back(v);}

bool dfs(int i){

for(int j =0; j < len[i]; j++){

int x = adj[i][j];

if(L[x]==-1||(parent[L[x]]==i)){

if(L[x]==-1|| dfs(L[x])){L[x]= i, R[i]= x;returntrue;}}

}

returnfalse;}

bool bfs(){

clr(visited);

int i, j, x, d, f =0, l =0;

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

if(R[i]==-1){visited[i]=true;Q[l++]= i, dis[i]=0;}}

while(f < l){

for(j =0,i=Q[f++]; j < len[i]; j++){

x = adj[i][j], d = L[x];

if(d ==-1)returntrue;

elseif(!visited[d]){Q[l++]= d;parent[d]=i,visited[d]=1,dis[d]=dis[i]+1;}

}}

returnfalse;}

int hopcroft\_karp(){

int res =0;

memset(L,-1,sizeof(L));memset(R,-1,sizeof(R));

while(bfs()){for(int i =0; i < n; i++){if(R[i]==-1&& dfs(i)) res++;}}

return res;}}

HungarianAlgorithm:

#define MAXIMIZE +1#define MINIMIZE -1

bool visited[MAX];/// namespace wm

int U[MAX],V[MAX],P[MAX],way[MAX],minv[MAX],match[MAX],ar[MAX][MAX];

int hungarian(int n,int m,int mat[MAX][MAX],int flag){/// n = nof row and m = nof col in 1 base

clr(U), clr(V), clr(P), clr(ar), clr(way);

for(int i =1; i <= n; i++){///inf=INT\_MAX

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

ar[i][j]= mat[i][j];

if(flag == MAXIMIZE) ar[i][j]=-ar[i][j];}}

if(n > m) m = n;

int i, j, a, b, c, d, r, w;

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

P[0]= i, b =0;

for(j=0;j<=m;j++)minv[j]=inf,visited[j]=0;

do{

visited[b]=1,a=P[b],d=0,w=inf;

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

if(!visited[j]){

r = ar[a][j]- U[a]- V[j];

if(r<minv[j])minv[j]=r,way[j]=b;

if(minv[j]<w)w=minv[j],d=j;}

}

for(j =0; j <= m; j++){if(visited[j])U[P[j]]+=w,V[j]-=w;else minv[j]-= w;}

b = d;

}while(P[b]!=0);

do{

d = way[b], P[b]= P[d], b = d;

}while(b !=0);

}

for(j=1;j<=m;j++) match[P[j]]=j;/// match[i] contains the col row i is matched to

return(flag== MINIMIZE)?-V[0]:V[0];}

IDA Star:

int ida(int g,int lim,int l,int last,int idx){///last=last move,don't go there!

int h=heuristic(l);if(h==0)return g+h///goal reached

int f = g + h;if(f > lim)return f;}

int solve(int l){

int lim = heuristic(l);

while(1){

int nlim = ida(0, lim, l, l,0);

if(nlim <= lim)return nlim;else lim = nlim;

}return-1;}

InverseFactorialLinear:

for(i=MAX-2;i>=0;i--)inv[i]=((LL)inv[i+1]\*(i+1))%MOD;

for(i=2,inv[1]=1;i<MAX;i++){///inv of 1 to n not factorial

inv[i]=MOD-((MOD/i)\*(LL)inv[MOD%i])%MOD;

if(inv[i]<0) inv[i]+= MOD;}

Josephus:

LL josephus(LL n,LL k,LL m){

if(n ==1)return0;

return(josephus(n-1,k)+k)% n;}

LL josephus(LL n,LL k,LL m){///Starts from 1 and every kth people dies, O(k log(n))

m = n - m;if(k <=1)return n - m;

LL i = m;

while(i < n){

LL r =(i - m + k -2)/(k -1);

if((i + r)> n) r = n - i;

elseif(!r) r =1;

i+=r,m=(m+(r\*k))%i;}

return m +1;}

Knights tour in InfiniteChessBoard:

int knight\_move(int x,int y){/// Min moves from(x,y) to(0,0) in chessboard

int a, b, z, c, d;/// board is non-negative

x = abs(x), y = abs(y);

if(x < y) a = x, x = y, y = a;

if(x ==2&& y ==2)return4;if(x ==1&& y ==0)return3;

if(y ==0||(y <<1)< x){

c=y&1,a=x-(c<<1),b=a&3;

return((a-b)>>1)+b+c;}

else{

d = x -((x - y)>>1);z =((d%3)!=0),c=(x-y)&1;

return((d/3)\*2)+c+(z\*2\*(1-c));}}

LagrangePolynomialInterpolation:

import sys import math

val =[0,1,2,3,4,5,6]

out =[1,1,2,4,8,16,31]

def lagrange(n):

res =0

for i in range(len(val)):

x = out[i],y=1

for j in range(len(val)):

if(i != j):

x \*=(n - val[j])

y \*=(val[i]- val[j])

res +=(x // y)

return res

def main():

print (lagrange(20))

if \_\_name\_\_ =='\_\_main\_\_':

main()

MaximumMatching(EdmondsBlossom):

constint MOD =1073750017;

bool adj[MAX][MAX];int n, ar[MAX][MAX];

int rank(int n){

longlong inv;int i,j,k,u,v,x,r=0,T[MAX];

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

for(k = r;k<n&&!ar[k][j];k++){}

if(k == n)continue;

inv = expo(ar[k][j], MOD -2);

for(i =0; i < n; i++){x=ar[k][i],ar[k][i]=ar[r][i];ar[r][i]=(inv \* x)% MOD;}

for(u = r +1; u < n; u++){

if(!ar[u][j])continue;

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

if(!ar[r][v])continue;

ar[u][v]=ar[u][v]-(((LL)ar[r][v]\*ar[u][j])%MOD);

if(ar[u][v]<0)ar[u][v]+=MOD;

}}

r++;}

return r;}

int solve(int n){

srand(time(0));clr(ar);

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

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

if(!adj[i][j])continue;

unsignedint x =(rand()<<15)^rand();

x=(x %(MOD-1))+1;ar[i][j]= x, ar[j][i]= MOD - x;}}

return(rank(n)>>1);}

Maximum XOR subset:

#define bitlen(x)((x)==0?(0):(64-\_\_builtin\_clzll(x)))

LL solve(int n, LL\* ar){/// select subset st xor is max

vector<LL> v[64]; LL m, x, res =0;

for(int i=0;i<n;i++)v[bitlen(ar[i])].push\_back(ar[i]);

for(int i =63; i >0; i--){

int l = v[i].size();

if(l==0)continue;

m = v[i][0],res = max(res, res ^ m);

for(int j =1; j < l; j++){

x = m ^ v[i][j];

if(x) v[bitlen(x)].push\_back(x);}

v[i].clear();

}return res;}

MinimumLexicographicRotation:

int minlex(char\* str){///Returns the 0 based index

int i,j,k,n,len,x,y;

len=n=strlen(str),n<<=1,i=0,j=1,k=0;

while((i+k)<n&&(j+k)<n){

x=i+k>=len?str[i+k-len]:str[i+k];

y=j+k>=len?str[j+k-len]:str[j+k];

if(x == y) k++;

elseif(x < y){j +=++k, k =0;if(i >= j) j = i +1;}

else{i +=++k, k =0;if(j >= i) i = j +1;}

}return(i < j)? i : j;}

MobiusFunction:

for(i =1, mu[1]=1; i < MAX; i++)for(j=i+i;j<MAX;j+=i)mu[j]-=mu[i];

OutputCompression:

int base =0, mp[256];char digit[256], str[256], temp[256];

void generate(){

for(int i =32; i <127; i++){

if(i==32||i==4||i==39||i==44||i==92)continue;

digit[base]= i,mp[i]= base,base++;}

digit[base]=0;}

void encode(char\* str,int v){

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

do{temp[k++]=digit[v%base];v/=base;}while(v);

for(i=k-1,str[k]=0;i>=0; i--)str[l++]=temp[i];}

int decode(char\* str){

int i, v =0;

for(i=0;str[i];i++)v=v\*base+mp[str[i]];

return v;}

PalindromicTree:

char str[MAX];/// namespace ptree below

char S[MAX];int idx,cur,slen,lps,len[MAX],link[MAX],counter[MAX],trie[MAX][LET];

void init(){///len[i]=len of pal at node i, link[i]=suffix link of node i

cur=0,slen=1,lps=0,idx=2;///link[i]=longest proper suffix-palindrome of i

clr(S),clr(len),clr(link),clr(trie),clr(counter);

S[0]=-1,link[0]=1,len[0]=0,link[1]=0,len[1]=-1;}

inlineint nextlink(int cur){///e.g, i = "aba", c = 'd', trie[i][c] = "dabad"

while(S[slen-len[cur]-2]!=S[slen-1])cur=link[cur];

return cur;}

inlinebool insert(char ch){///Returns true if a new distinct pal appears after adding ch

S[slen++]= ch, cur = nextlink(cur);

int c=ch-'a',flag=trie[cur][c];///Change here for diff char

if(!flag){len[idx]=len[cur]+2,link[idx]=trie[nextlink(link[cur])][c];trie[cur][c]=idx++;}

cur = trie[cur][c];counter[cur]++;///count of palindrome substring cur in string

lps = max(lps, len[cur]);

return!flag;}

///IMPORTANT:do this in main to update count of all nodes

for(int i = idx; i >=0; i--) counter[link[i]]+= counter[i];

PollardRho:

/// first make n odd, then prime factors are in ar after call

publicvoid rhoPollard(BigInteger n){

if(n.compareTo(BigInteger.valueOf(1))==0)return;

if(n.isProbablePrime(100)){ar[len++]= n;return;}

Random r =newRandom();

BigInteger x=BigInteger.valueOf((r.nextLong())).mod(n),y=x,d=newBigInteger("1");

BigInteger c=BigInteger.valueOf((r.nextLong())).mod(n);

while(d.compareTo(BigInteger.valueOf(1))==0){

x = f(x, n, c), y = f(f(y, n, c), n, c);

BigInteger z =newBigInteger("0");

if(x.compareTo(y)>0) z = x.subtract(y);

else z = y.subtract(x);

d = gcd(z, n);}/// Write BigInteger gcd

BigInteger z = n.divide(d);

if(d.compareTo(BigInteger.valueOf(1))!=0)rhoPollard(d);

if(z.compareTo(BigInteger.valueOf(1))!=0)rhoPollard(z);}

RadixSort:

#define CHUNK 8

unsignedint bucket[1<<CHUNK],temp[MAX];

void radix\_sort(unsignedint\* ar,int n){

constint mask =(1<< CHUNK)-1;

for(int i =0; i <32; i += CHUNK){

clr(bucket);

for(int j=0; j<n;j++)bucket[(ar[j]>>i)&mask]++;

for(int j=1;j<=mask;j++)bucket[j]+=bucket[j -1];

for(j = n -1; j >=0; j--){temp[--bucket[(ar[j]>>i)&mask]]=ar[j];}

for(j =0; j < n; j++)ar[j]=temp[j];}}

Rope:

#include<ext/pb\_ds/assoc\_container.hpp>#include<ext/pb\_ds/tree\_policy.hpp>

#include<ext/rope>usingnamespace \_\_gnu\_cxx;usingnamespace \_\_gnu\_pbds;

tree<LL,null\_type,less<LL>,rb\_tree\_tag,tree\_order\_statistics\_node\_update>T;

Simplex:

#define EPS 1e-9#define MAXV 4010#define MAXC 4010

#define MINIMIZE -1#define MAXIMIZE +1#define EQUAL 0#define LESSEQ -1#define GREATEQ 1

#define FEASIBLE +1#define INFEASIBLE -1#define UNBOUNDED 666

/\*\* Constraints and vars are 1 base,if var can be negative,replace var with diff of 2 new var

If ABS(X)<=M in constraints,Replace with X<=M and -X<=M

Fractional LP: max/min

3x1 + 2x2 + 4x3 + 6

-------------------

3x1 + 3x2 + 2x3 + 5

s.t. 2x1 + 3x2 + 5x3 ≥ 23

3x2 + 5x2 + 4x3 <= 30

x1, x2, x3 ≥ 0

Replace with: max/min

3y1 + 2y2 + 4y3 + 6t

s.t. 3y1 + 3y2 + 2y3 + 5t = 1

2y1 + 3y2 + 5y3 - 23t ≥ 0

3y1 + 5y2 + 4y3- 30t <= 0

y1, y2, y3, t ≥ 0

\*\*\*/

double ar[MAXC][MAXV],val[MAXV],rhs[MAXC];

int n, m, flag,adj[MAXV],idx[MAXV],down[MAXV],link[MAXC];

void init(int nvar,double func[],int min\_or\_max){

m =0, n = nvar, flag = min\_or\_max;

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

for(int i =1; i <= n; i++) ar[0][i]= func[i]\* flag;}

inlinevoid add\_constraint(double var[],double lim,int flag){

flag \*=-1;

if(flag ==0){

rhs[++m]= lim;for(int i =1; i <= n; i++) ar[m][i]= var[i];

rhs[++m]=-lim;for(int i =1; i <= n; i++) ar[m][i]=-var[i];}

else{rhs[++m]= lim \* flag;for(int i =1; i <= n; i++) ar[m][i]= var[i]\* flag;}}

void pivot(int x,int y,double& res){

int i, j, len =0;double v = ar[x][y];

swap(link[x], down[y]), rhs[x]/= v, ar[x][y]=1;

for(j =1; j <= n; j++){ar[x][j]/= v;if(abs(ar[x][j])> EPS) adj[len++]= j;}

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

if(abs(ar[i][y])> EPS && i != x){

rhs[i]-= ar[i][y]\* rhs[x], v = ar[i][y], ar[i][y]=0;

for(j =0; j < len; j++) ar[i][adj[j]]-=(v \* ar[x][adj[j]]);

}}

res +=(ar[0][y]\* rhs[x]), v = ar[0][y], ar[0][y]=0;

for(j =0; j < len; j++) ar[0][adj[j]]-=(v \* ar[x][adj[j]]);}

int solve(double& res){

int i, j, x, y;double u, v, mn, mx;

for(i =1; i <= n; i++) down[i]= i;

for(i =1; i <= m; i++) link[i]= i + n;

while(1){

x =0, y =0, mn =-EPS;

for(i =1; i <= m; i++){if(rhs[i]< mn) mn = rhs[i], x = i;}

if(x ==0)break;

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

if(ar[x][i]<-EPS){y = i;if(rand()&1)break;}}

if(y ==0)return INFEASIBLE;

pivot(x, y, res);}

while(1){/// phase 2

x =0, y =0, mx = EPS;

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

if(ar[0][i]> mx) mx = ar[0][i], y = i;}

if(y ==0)break;

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

if(ar[i][y]> EPS){u = rhs[i]/ ar[i][y];if(x ==0|| u < v) x = i, v = u;}}

if(x ==0)return UNBOUNDED;

pivot(x, y, res);}

res \*= flag;

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

if(link[i]<= n) idx[link[i]]= i;}

for(int i =1; i <= n; i++) val[i]= rhs[idx[i]];

return FEASIBLE;}

SparseTable:

for(i=2,lg[0]=lg[1]=0;i<MAX;i++)lg[i]=lg[i>>1]+1;

int query(int i,int j){int k=lg[j-i];int x=dp[k][i],y=dp[k][j-(1<<k)+1];return min(x, y);}

Treap:

int pool\_index =0;

struct node{

node \*l,\*r,\*parent;int key, subtree, priority;

inline node(){l = r =0, parent =0;}

inline node(int val){l = r =0, parent =0;subtree =1, key = val;priority =(rand()<<16)^ rand();}

inlinevoid update(){

subtree =1;

if(l){l->parent=this,subtree+=l->subtree;}

if(r){r->parent=this,subtree+=r->subtree;}}

} pool[MAXN];

structTreap{

struct node\* root;

inlineint size(node\*&cur){if(cur) cur->update();return(cur ? cur->subtree :0);}

inlineint size(){return(root?root->subtree:0);}

inlinevoid merge(node\*&cur, node\* l, node\* r){

if(!l ||!r) cur = l ? l : r;

elseif(l->priority>r->priority) merge(l->r,l->r,r),cur=l;

else merge(r->l,l,r->l),cur=r;

if(cur) cur->update();}

void split(node\* cur, node\*&l, node\*&r,int key){

l =0, r =0;

if(!cur)return;

if(cur->key < key){l = cur, split(l->r, l->r, r, key);l->update();}

else{r = cur, split(r->l, l, r->l, key);r->update();}}

void split\_index(node\* cur, node\*&l, node\*&r,int index){

l =0, r =0;

if(!cur)return;

if(size(cur->l)< index){

l = cur;split\_index(l->r, l->r, r,index-size(cur->l)-1);

if(l) l->update();}

else{r = cur, split\_index(r->l, l, r->l, index);if(r) r->update();}}

Treap(){root =0,pool\_index=0;}

inlinevoid insert(int key){

node\* l,\*r;split(root, l, r, key);

pool[pool\_index]= node(key);

merge(root, l,&pool[pool\_index++]);merge(root, root, r);}

inlinebool erase(int key){

node \*l,\*r,\*m;split(root, l, r, key), split\_index(r, m, r,1);

bool res =(m && m->key == key);

if(!res) merge(r, m, r);

merge(root, l, r);

return res;}

inlineint rank(node\* cur){/// rank of node in the treap

int res =1+ size(cur->l);

while(cur->parent){

if(cur->parent->r==cur)res+=(size(cur->parent->l)+1);

cur = cur->parent;}

return res;}

inlineint kth(int k){/// kth smallest element 1 based

if((k <1)||(k > size()))return-1;

node \*l,\*r,\*m;

split\_index(root, l, r, k);split\_index(l, l, m, l->subtree -1);

int res = m->key;

merge(l, l, m);merge(root, l, r);

return res;}

inlineint count(int key){///count of values<key

node \*l,\*r,\*cur = root; split(root, l, r, key);

int res =(l ? l->subtree :0);

merge(root, l, r);

return res;}};

Trie:

int r, idx, counter[MAX], trie[MAX][LET];

void initialize(){r =0, idx =1, counter[r]=0;

for(int i=0;i<LET;i++)trie[r][i]=0;}

void insert(int x){

if(!trie[r][x]){trie[r][x]=idx, r=idx++;

for(i=0,counter[r]=1;i<LET;i++)trie[r][i]=0;}

else r = trie[r][x], counter[r]++;}

WalshHadamardTransformation:

#define MAX 1048576/// namespace fwht below

constint OR=0,constint AND=1,constint XOR=2;longlong P1[MAX], P2[MAX];

void walsh\_transform(LL\*ar,int n,intflag=XOR){

if(n ==0)return;

int i, m = n /2;

walsh\_transform(ar, m, flag);walsh\_transform(ar + m, m, flag);

for(i=0;i<m;i++){///Dont forget modulo if required

longlong x = ar[i], y = ar[i + m];

if(flag==OR) ar[i]=x,ar[i+m]=x+y;if(flag==AND) ar[i]=x+y,ar[i+m]=y;

if(flag==XOR)ar[i]=x+y,ar[i+m]=x-y;

}}

void inverse\_walsh\_transform(LL\* ar,int n,int flag=XOR){

if(n ==0)return;

int i, m = n /2;

inverse\_walsh\_transform(ar, m, flag);inverse\_walsh\_transform(ar + m, m, flag);

for(i=0;i<m;i++){///Dont forget modulo if required

longlong x = ar[i], y = ar[i + m];

if(flag==OR) ar[i]=x,ar[i+m]=y-x;if(flag==AND) ar[i]=x-y,ar[i+m]=y;

if(flag==XOR)ar[i]=(x+y)>>1,ar[i+m]=(x-y)>>1;///Mod inv if required here

}}

vector<LL> convolution(int n,LL\* A,LL\* B,int flag=XOR){

assert(\_\_builtin\_popcount(n)==1);

for(int i =0; i < n; i++) P1[i]=A[i];

for(int i =0; i < n; i++) P2[i]=B[i];

walsh\_transform(P1, n, flag);walsh\_transform(P2, n, flag);

for(int i =0; i < n; i++) P1[i]= P1[i]\* P2[i];

inverse\_walsh\_transform(P1, n, flag);

returnvector<LL>(P1, P1 + n);}

Z Algorithm:

voidZFunction(char\* str){

int i, x, l =0, r =0;for(i =1; str[i]; i++){

Z[i]= max(0, min(Z[i - l], r - i));

while(str[i+Z[i]]&&str[Z[i]]==str[i+Z[i]])Z[i]++;

if((i+Z[i])>r)l=i,r=i+Z[i];

}

Z[0]= i;}

ArticulationPoint:

void AP(LL u){ dis[u]= low[u]=++T;

LL child =0; visit[u]=1; LL len = g[u].size();

for(i=0;i<len;i++){ v = g[u][i];

if(visit[v]==0){ child++; par[v]= u; AP(v);

low[u]= min(low[u],low[v]);if(par[u]==-1&&child>1)ap[u]=1;

if(par[u]!=-1&&low[v]>=dis[u])ap[u]=1;if(low[v]> dis[u])bridge;}

elseif(v != par[u]){ low[u]= min(low[u],dis[v]);}}}

CentroidDecomposition:

vector<int> g[Max],ctree[Max],cost[Max];int visit[Max],cs;

int centroid,s,n,croot;///ctrd=glbl ctrd; s=szof subtree;croot=ctrd tree root

bool iscentroid[Max];int cpar[Max],clevel[Max];int ctable[Max][20];///ctrd par & ctrd lvl

int dfs(int u){ visit[u]= cs;///find a ctrd in given subtree

int len = sz(g[u]),i,ret =1,mx =0;

Rep(i,len){int v = g[u][i];

if(visit[v]!=cs&&iscentroid[v]==0){

int my = dfs(v); mx = max(mx,my); ret += my;}}

mx = max( mx,s-ret );if(mx \*2<= s)centroid = u;return ret;}

int get\_size(int u){ visit[u]= cs;///size of given subtree

int len = sz(g[u]),i,ret =1;

Rep(i,len){int v = g[u][i];

if(visit[v]!=cs&&iscentroid[v]==0){ ret += get\_size(v);}}

return ret;}

void F(int u,int p){ cs++;s = get\_size(u); cs++;dfs(u);

int cur = centroid;iscentroid[cur]=1;///ctrd dec(cur\_node,parent)

if(p){ cpar[cur]= p; ctree[p].pb(cur);///build centroid tree

clevel[cur]= clevel[p]+1;}

else{ croot = cur; clevel[cur]=0;}

int len = sz(g[cur]),i;

Rep(i,len){int v = g[cur][i];

if( iscentroid[v]==0)F(v,cur);}}

void compute\_table(int u,int d,int p){

visit[u]= cs; ctable[u][ clevel[u]-clevel[p]]= d;

int len = sz(g[u]),i;

Rep(i,len){int v = g[u][i];

if(visit[v]!= cs&&clevel[v]>clevel[p]){

compute\_table(v,d+1,p);///compute\_table(v,d+cost[u][i],p);

}}}

KMP:

void prework(){ lps[0]=0; LL len =0, i =1; LL l = strlen(patt);

while(i < l){if(patt[i]==patt[len]){len++;lps[i]= len;i++;}

else{if(len ==0){ lps[i]=0;i++;}

else{ len = lps[len-1];}}}}

void kmp(){LL i=0,j=0;LL l1=strlen(text);LL l2=strlen(patt);ind =0;

while(i < l1){if(text[i]== patt[j]){i++;j++;}

if(j == l2){ans[ind++]= i - j;j = lps[j-1];}

elseif(text[i]!= patt[j]){if(j ==0)i++;else j = lps[j-1];}}}

Manachar:

void manodd(){int l=0, r=-1;

for(int i=0; i<n;++i){int k=(i>r ?1: min (d1[l+r-i], r-i));

while(i+k<n&&i-k>=0&&s[i+k]==s[i-k])++k; d1[i]= k--;

if(i+k > r)l=i-k, r=i+k;}}

void make(){int k =0; s[k++]='$';

for(int i=0;i<n;i++){s[k++]='#';s[k++]=str[i];}s[k++]='#';s[k]=0;n=k;}

2DGeometry:

struct line {// Creates a line with equation ax + by + c = 0

double a, b, c;point p1, p2;line(){}

line(double \_a,double \_b,double \_c){

a = \_a, b = \_b, c = \_c; assert(!(eq(a,0)&& eq(b,0)));

if(eq(a,0)) p1 = point(0,-c/b), p2 = point(1,-c/b);

elseif(eq(b,0)) p1 = point(-c/a,0), p2 = point(-c/a,1);

else p1 = point(0,-c/b), p2 = point(-c/a,0);}

line( point \_p1,point \_p2 ){p1 = \_p1, p2 = \_p2;

a = p1.y - p2.y; b = p2.x - p1.x; c = p1.x \* p2.y - p2.x \* p1.y;}};

point operator\*(const point &u,const point &v){return point(u.x \* v.x - u.y \* v.y, u.x \* v.y + v.x \* u.y);}

ostream &operator<<(ostream &os,const point &p){ os <<"("<< p.x <<","<< p.y <<")";}

double norm(point u){return sqrt(u.x \* u.x + u.y \* u.y);}

double arg(point u){ assert(u != origin);return atan2(u.y, u.x);}

point polar(double r,double theta){return point(r \* cos(theta), r \* sin(theta));}

double crsp(point u, point v){return u.x \* v.y - u.y \* v.x;}

double smlr\_angle(point l,point m,point r){return abs(remainder(arg(l-m)-arg(r-m),2.0\*pi));}

point rtt(point piv, point u,double theta){return(u - piv)\* polar(1.00, theta)+ piv;}

point projection(point p, point st, point ed){return dotp(ed - st, p - st)/ norm(ed - st)\* unit\_vector(ed - st)+ st;}

point extend(point st, point ed,double len){return ed + unit\_vector(ed-st)\* len;}

point segmentProjection(point p, point st, point ed){

double d = dotp(p - st, ed - st)/ norm(ed - st);

if(d <0)return st;if(d > norm(ed - st)+ eps)return ed;

return st + unit\_vector(ed - st)\* d;}

double distancePointSegment(point p,point st,point ed){return norm(p-segmentProjection(p, st,ed));}

double distancePointLine(point P,point st,point ed){return norm(projection(P,st,ed)-P);}

point reflection(point p, point st, point ed){

point proj = projection(p, st, ed);

if(p != proj)return extend(p, proj, norm(p - proj));

return proj;}

bool collinear(point p, point st, point ed){return fabs(crsp(p - st, ed -st))< eps;}

int lineLineIntersection(point a, point b, point p, point q, point &ret){

if(fabs(crsp(b - a, p - q))< eps){if(collinear(a, p, q)){ ret = a;return inf;}

return0;}

else{ ret = a +(b - a)\* crsp(p - a, p - q)/ crsp(b - a, p - q);

return1;}}

int lineLineIntersection(line L1, line L2, point &ret){return lineLineIntersection(L1.p1, L1.p2, L2.p1, L2.p2, ret);}

bool segmentSegmentIntersection(point a, point b, point p, point q, point &ret){

if( fabs(crsp(b - a, q - p))< eps ){

if( eq(norm(a - p)+ norm(q - a), norm(q - p))){ret = a;returntrue;}

if( eq(norm(b - p)+ norm(q - b), norm(q - p))){ret = b;returntrue;}

returnfalse;}

double dir1 = crsp(b - a, p - a), dir2 = crsp(b - a, q - a);

if((dir1 < eps && dir2 < eps)||(dir1 > eps && dir2 > eps))returnfalse;

dir1 = crsp(q - p, a - p), dir2 = crsp(q - p, b - p);

if((dir1 < eps && dir2 < eps)||(dir1 > eps && dir2 > eps))returnfalse;

return lineLineIntersection(a, b, p, q, ret);

}

point circumCircleCenter(point u, point v, point w){

assert(collinear(u, v, w)==false);

point vw\_mid =(v + w)/2; point uv\_mid =(u + v)/2; point ret;

lineLineIntersection(vw\_mid, rtt(vw\_mid, extend(v, vw\_mid,1), pi/2),

uv\_mid, rtt(uv\_mid, extend(u, uv\_mid,1), pi/2), ret);

return ret;}

double angle(point u, point v){return acos(dotp(u, v)/(norm(u)\* norm(v)));}

bool segSegIntersection(point u, point v, point a, point b){

double ang1 = angle(a - v, u - v);double ang2 = angle(b - v, u - v);

if(ang1 > eps && ang2 > eps)returnfalse;if(ang1 <-eps && ang2 <-eps)returnfalse;

if(fabs(ang1)< eps || fabs(ang2)< eps){

if(fabs(ang1)< eps){if(onSegment(a, u, v))returntrue;}

if(fabs(ang2)< eps){if(onSegment(b, u, v))returntrue;}

returnfalse;}

swap(u, a); swap(v, b);

/\*REPEAT THE ABOVE CODE HERE\*/

returntrue;}

//Intersection - Circle, Line:

inlineboolintersection(circle C,line L,point &p1,point &p2){

if(Distance( C.center, L )> C.r + eps )returnfalse;

double a, b, c, d, x = C.center.x, y = C.center.y;

d = C.r\*C.r - x\*x - y\*y;

if( eq( L.a,0)){

p1.y = p2.y =-L.c / L.b; a =1; b =2\* x;

c = p1.y \* p1.y -2\* p1.y \* y - d; d = b \* b -4\* a \* c;

d = sqrt( fabs (d));

p1.x =( b + d )/(2\* a ); p2.x =( b - d )/(2\* a );}

else{ a = L.a \*L.a + L.b \* L.b;

b =2\*( L.a \* L.a \* y - L.b \* L.c - L.a \* L.b \* x);

c = L.c \* L.c +2\* L.a \* L.c \* x - L.a \* L.a \* d;

d = b \* b -4\* a \* c; d = sqrt( fabs(d));

p1.y =( b + d )/(2\* a ); p2.y =( b - d )/(2\* a );

p1.x =(-L.b \* p1.y -L.c )/ L.a;

p2.x =(-L.b \* p2.y -L.c )/ L.a;}

returntrue;}

//Find Points that are r1 unit away from A, and r2 unit away from B:

inlinebool findpointAr1Br2(point A,double r1,point B,double r2,point &p1,point &p2){

line L; circle C; L.a =2\*(B.x - A.x ); L.b =2\*(B.y - A.y );

L.c = A.x \* A.x + A.y \* A.y - B.x \* B.x - B.y \* B.y + r2 \* r2 - r1 \* r1;

C.center = A; C.r = r1; returnintersection( C, L, p1, p2 );}

//Intersection Area between Two Circles:

inlinedoubleintersectionArea2C( circle C1, circle C2 ){

C2.center.x =Distance( C1.center, C2.center );

C1.center.x = C1.center.y = C2.center.y =0;

if( C1.r < C2.center.x - C2.r + eps )return0;

if(-C1.r + eps > C2.center.x - C2.r )return pi \* C1.r \* C1.r;

if( C1.r + eps > C2.center.x + C2.r )return pi \* C2.r \* C2.r;

double c, CAD, CBD, res; c = C2.center.x;

CAD =2\* acos((C1.r \* C1.r + c \* c - C2.r \* C2.r)/(2\* C1.r \* c));

CBD =2\* acos((C2.r \* C2.r + c \* c - C1.r \* C1.r)/(2\* C2.r \* c));

res=C1.r \* C1.r \*( CAD - sin( CAD ))+ C2.r \* C2.r \*( CBD - sin ( CBD ));

return.5\* res;}

// Checks whether ractangle with sides (a, b) fits into rectangle with sides (c, d)

bool fits(int a,int b,int c,int d ){

double X, Y, L, K,DMax;

if( a < b ) swap( a, b );if( c < d ) swap( c, d );

if( c <= a && d <= b )returntrue;if( d >= b )returnfalse;

X = sqrt( a\*a + b\*b ); Y = sqrt( c\*c + d\*d );

if( Y < b )returntrue;if( Y > X )returnfalse;

L =( b - sqrt( Y\*Y - a\*a))/2;

K =( a - sqrt( Y\*Y - b\*b))/2;DMax= sqrt(L \* L + K \* K);

if( d >=DMax)returnfalse;

returntrue;}

3DGeometry:

point crsp(point u, point v){return point(u.y\*v.z-u.z\*v.y, u.z\*v.x-u.x\*v.z, u.x\*v.y-u.y\*v.x);}

struct plane{

double a, b, c, d;//ax + by + cz + d = 0plane(){;}

plane(point p1, point p2, point p3){

point vtr = crsp(p2 - p1, p3 - p1);

if(norm(vtr)< eps){assert(false);}// doesn't define a plance

a = vtr.x; b = vtr.y; c = vtr.z;

d =-p1.x\*vtr.x -p1.y\*vtr.y - p1.z \* vtr.z;}};

double smlr\_angle(point l, point m, point r){

double d = dotp(l - m, r - m);return acos(d /(norm(l-m)\* norm(r-m)));}

point projection(point p, point st, point ed){return dotp(ed - st, p - st)/ norm(ed - st)\* unit\_vector(ed - st)+ st;}

point extend(point st, point ed,double len){return ed + unit\_vector(ed-st)\* len;}

point rtt(point axis, point p,double theta){

axis = unit\_vector(axis);

return p \* cos(theta)+sin(theta)\*crsp(axis, p)+ axis \*(1-cos(theta))\* dotp(axis, p);}

double pointPlaneDistance(plane P, point q){

return fabs(P.a\*q.x+ P.b\*q.y +P.c\*q.z + P.d)/ sqrt(P.a \* P.a + P.b \* P.b + P.c \* P.c);}

double pointPlaneDistance(point p1, point p2, point p3, point q){return pointPlaneDistance(plane(p1,p2,p3), q);}

bool coplanar(point p1, point p2, point p3, point q){

p2 = p2-p1, p3 = p3-p1, q = q-p1;if( fabs( dotp(q, crsp(p2, p3)))< eps )returntrue;

returnfalse;}

int linePlaneIntersection(point u, point v, point l, point m, point r, point &x){

assert(l != m && m != r && l != r && u != v);

if(coplanar(l, m, r, u)&& coplanar(l, m, r, v))return-1;

l = l - m; r = r - m; u = u - m; v = v - m;

point C = crsp(l, r);double denom = dotp(v - u, C);

if(fabs(denom)< eps)return0;

double alpha =-dotp(C, u)/ denom; x = u +(v - u)\* alpha + m;

return1;}

double angle(point u, point v){return acos(dotp(u, v)/(norm(u)\* norm(v)));}

AdaptiveSimpson:

#define z\_slice\_eps 1e-5#define SIMPSON\_EPS 1e-9#define SIMPSON\_TERMINAL\_EPS 1e-12

double F(double x){return x;}

double single\_simpson(double miny,double maxy){

return(maxy - miny)/6\*(F(miny)+4\* F((miny + maxy)/2.)+ F(maxy));}

double adaptive\_simpson(double miny,double maxy,double c,double eps = SIMPSON\_EPS){

if(maxy - miny < SIMPSON\_TERMINAL\_EPS)return0;

double midy =(miny + maxy)/2;

double a = single\_simpson(miny, midy);double b = single\_simpson(midy, maxy);

if(fabs(a + b - c)<15\* eps)return a + b +(a + b - c)/15.0;

return adaptive\_simpson(miny,midy,a,eps/2.)+ adaptive\_simpson(midy, maxy, b, eps /2.);}

double simpson(double minz,double maxz){

double ans, last, z, temp; ans =0; last = F(minz);

for(z = minz; z<=maxz; z+=z\_slice\_eps){if(z>(minz+maxz)/2) z = z;

temp = F(z+z\_slice\_eps);ans += last +4\*F(z+z\_slice\_eps/2)+ temp; last = temp;}

ans \*= z\_slice\_eps/6;

return ans;}

doubleIntegrate(double x\_st,double x\_ed){return adaptive\_simpson(x\_st, x\_ed, single\_simpson(x\_st, x\_ed));}

Area of Subpolygon:

vector< pii > polygon; LL dp[MAX+5];

LL F(int ed){

if(ed <0)return0;if(dp[ed]!=-1)return dp[ed];

int nxt =(ed +1)% polygon.size();

return dp[ed]=(LL) polygon[ed].xx \* polygon[nxt].yy -(LL) polygon[ed].yy \* polygon[nxt].xx + F(ed -1);}

LL compute(int st,int ed){

LL ret =(LL) polygon[ed].xx \* polygon[st].yy -(LL)polygon[ed].yy \* polygon[st].xx;

if(st <= ed)return abs( ret + F(ed -1)- F(st -1));

return abs( ret + F((int) polygon.size()-1)-(F(st -1)- F(ed -1)));}

ClosestPair:

bool cmp(const point &u,const point &v){if(eq(u.y , v.y))return u.x < v.x;return u.y > v.y;}

// Don't forget to sort all the points before calling closest\_pair and to call unique()

longdouble closest\_pair(point \*P,int st,int ed){

if(st == ed)return numeric\_limits<double>::max();

if(st +1== ed)return abs(P[st]- P[ed]);

int mid =(st+ed)/2, i, j, k, turn;

longdouble soFar = min(closest\_pair(P, st, mid), closest\_pair(P, mid+1, ed));

vector<point>Lt,Rt;

for(i = st; i <= mid; i++)if(abs(P[mid].x - P[i].x)< soFar + eps)Lt.push\_back(P[i]);

for(i = mid+1; i <= ed; i++)if(abs(P[i].x - P[mid].x)< soFar + eps)Rt.push\_back(P[i]);

stable\_sort(Lt.begin(),Lt.end(), cmp); stable\_sort(Rt.begin(),Rt.end(), cmp);

for(i = j =0; i <Lt.size(); i++){

while(j <Rt.size()&&Rt[j].y >Lt[i].y + soFar + eps) j++;

for(k = j; k < min((int)Rt.size(), j +6); k++)// You may increase the bound.

soFar = min(soFar,(longdouble) abs(Lt[i]-Rt[k]));}

return soFar;}

ConvexHull:

#define CW -1#define ACW 1

int direction(pii st, pii ed, pii q){

LL xp =(LL)(ed.xx - st.xx)\*(q.yy - ed.yy)-(LL)(ed.yy - st.yy)\*(q.xx - ed.xx);

if(!xp)return0;if(xp >0)return ACW;return CW;}

int convex\_hull(vector<pii> p,vector<pii>&h){

vector<pii> up, dwn;//constructs upper hull in clockwise order, lower hull in anti-clockwise order

h.clear();sort(p.begin(), p.end()); up.push\_back(p[0]); dwn.push\_back(p[0]);

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

if(direction(p[0], p.back(), p[i])!= CW){

while(up.size()>=2&& direction(up[up.size()-2], up.back(), p[i])!= CW) up.pop\_back();

up.push\_back(p[i]);}

if(direction(p[0], p.back(), p[i])!= ACW){

while(dwn.size()>=2&& direction(dwn[dwn.size()-2], dwn.back(), p[i])!= ACW) dwn.pop\_back();

dwn.push\_back(p[i]);}}

h = dwn;for(int i =(int) up.size()-2; i >=1; i--) h.push\_back(up[i]);

return h.size();}

/\* CONVEX HULL TRICK \*/

structLine{ LL m,c;Line(LL \_m =0, LL \_c =0):m(\_m), c(\_c){};};

structConvexHullTrick{

vector<Line> Q;bool minFlag;//Fast -> Slow -> Slower -> Slowest

ConvexHullTrick(bool flg =false):minFlag(flg){};

LL getX(Line u,Line v){// Fast vrs Slow \*ORDER MATTERS\*

LL difC = v.c - u.c, difM = u.m - v.m, ret = difC/difM;

return(difC % difM)? ret+1:ret;}

bool isBad(Line L1,Line L2,Line L3){

if(minFlag ==false)return(L3.c - L1.c)/(longdouble)(L1.m - L3.m)>(L2.c-L1.c)/(longdouble)(L1.m - L2.m);

elsereturn(L3.c - L1.c)/(longdouble)(L1.m - L3.m)<(L2.c-L1.c)/(longdouble)(L1.m - L2.m);}

void addLine(Line L){//Has to be slower than then the slowest in the Q

while(Q.empty()==false){

if(Q.back().m < L.m) \_\_builtin\_trap();

elseif(minFlag ==false&& Q.back().m == L.m && L.c > Q.back().c) Q.pop\_back();

elseif(minFlag ==true&& Q.back().m == L.m && L.c < Q.back().c) Q.pop\_back();

elseif(Q.back().m == L.m)return;elseif(Q.size()<=1)break;

elseif(isBad(Q[Q.size()-2], Q.back(), L)) Q.pop\_back();elsebreak;}

Q.push\_back(L);}

LL query(LL pos){

int lo =0, hi =(int) Q.size()-1, n = hi, mid; LL L, R;

while(true){

mid =(lo+hi)/2;

if(minFlag){

if(mid ==0) L =-5e18;else L = getX(Q[mid-1], Q[mid]);

if(mid == n) R =5e18;else R = getX(Q[mid], Q[mid+1]);

if(L <= pos && pos < R)return Q[mid].m \* pos + Q[mid].c;

if(pos < L) hi = mid-1;else lo = mid+1;}

else{

if(mid == n) L =-5e18;else L = getX(Q[mid], Q[mid+1]);

if(mid ==0) R =5e18;else R = getX(Q[mid-1], Q[mid]);

if(L <= pos && pos < R)return Q[mid].m \* pos + Q[mid].c;

if(pos < L) lo = mid+1;else hi = mid-1;}}}};

/\* DINITZ \*/

int src, snk, nNode, nEdge, Q[MAXN+5], fin[MAXN+5], pro[MAXN+5], dist[MAXN+5], flow[2\*MAXE+5], cap[2\*MAXE+5], nxt[2\*MAXE+5], to[2\*MAXE+5];

void init(int \_src,int \_snk,int \_n){

src = \_src, snk = \_snk, nNode = \_n, nEdge =0;

memset(fin,-1,sizeof(fin));}

void add\_edge(int u,int v,int c){

to[nEdge]= v, cap[nEdge]= c, flow[nEdge]=0, nxt[nEdge]= fin[u], fin[u]= nEdge++;

to[nEdge]= u, cap[nEdge]= c, flow[nEdge]=0, nxt[nEdge]= fin[v], fin[v]= nEdge++;

assert(nEdge <=2\*MAXE);}

bool bfs(){

int st, en, i, u, v; memset(dist,-1,sizeof(dist)); dist[src]= st = en =0;

Q[en++]= src;

while(st < en){

u = Q[st++];

for(i=fin[u]; i>=0; i=nxt[i]){ v = to[i];

if(flow[i]< cap[i]&& dist[v]==-1){

dist[v]= dist[u]+1;Q[en++]= v;}}}

return dist[snk]!=-1;}

int dfs(int u,int fl){

if(u==snk)return fl;

for(int&e=pro[u], v, df; e>=0; e=nxt[e]){

v = to[e];

if(flow[e]< cap[e]&& dist[v]==dist[u]+1){

df = dfs(v, min(cap[e]-flow[e], fl));

if(df>0){ flow[e]+= df; flow[e^1]-= df;return df;}}}

return0;}

LL dinitz(){

LL ret =0;int df;

while(bfs()){

for(int i=1; i<=nNode; i++) pro[i]= fin[i];

while(true){

df = dfs(src, INF);if(df) ret +=(LL)df;elsebreak;}}

return ret;}

ErdosGaloi:

int deg[MM], n, degSum[MM], ind[MM], minVal[MM];

boolErdosGallai(){// 1 indexed

bool poss =true;int i, sum =0, j, r;

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

if( deg[i]>= n ) poss =false; sum += deg[i];}

if(!poss ||( sum &1)||( n ==1&& deg[1]>0))returnfalse;

sort( deg +1, deg + n +1, greater <int>());

degSum[0]=0; j = n;

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

degSum[i]= degSum[i-1]+ deg[i];for(; j >=1&& deg[j]< i; j--);

ind[i]= j+1;}

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

j = ind[r];

if( j == n+1) minVal[r]=( n - r )\* r;

elseif( j <= r ) minVal[r]= degSum[n]- degSum[r];

else{

minVal[r]= degSum[n]- degSum[j-1]; minVal[r]+=(j-r-1)\*r;}}

for( r =1; r < n; r++)if( degSum[r]>( r \*(r-1)+ minVal[r]))returnfalse;

returntrue;}

ExtendedEuclid:

LL exEuclid(LL a, LL b, LL &x, LL &y){

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

LL nx, ny, g, r;

g = exEuclid(b, a % b, nx, ny); x = ny; y = nx - a / b \* ny;

return g;}

LL \_\_lower\_bound(LL x0, LL dx, LL x){//min step to : get as close as possible to the value x such that x0 > x

LL d = x - x0;

if(d >0&& d % dx){

if(dx >0)return d/dx +1;return d/dx -1;}

return d/dx;}

LL number\_of\_solution(LL A, LL B, LL C, LL x1, LL x2, LL y1, LL y2){// Ax + By + C = 0

LL g, x0, y0, dx, dy, stp, st1, st2, ed1, ed2, st, ed,x , y;

if(A && B){

g = exEuclid(A, B, x0, y0);

if(C % g)return0;

else{ x0 \*=-C/g; y0 \*=-C/g; dx = B/g; dy = A/g;

if(dx <0) dx \*=-1, dy \*=-1;

stp = \_\_lower\_bound(x0, dx,-10000000000000LL);

x0 += dx \* stp; y0 -= dy \* stp;

st1 = \_\_lower\_bound(x0, dx, x1); ed1 = \_\_lower\_bound(x0, dx, x2 +1)-1;

if(y0 >0){ ed2 = \_\_lower\_bound(y0,-dy, y1); st2 = \_\_lower\_bound(y0,-dy, y2 +1)+1;}

else{ st2 = \_\_lower\_bound(y0,-dy, y1); ed2 = \_\_lower\_bound(y0,-dy, y2 +1)-1;}

st = max(st1, st2); ed = min(ed1, ed2);

if(st <= ed)return ed - st +1;

return0;}}

elseif(A){

if( C % A )return0;elseif( x1 <=-C / A &&-C / A <= x2)return y2 - y1 +1;

elsereturn0;}

elseif(B){if(C % B)return0;

if( y1 <=-C / B &&-C / B <= y2)return x2 - x1 +1;

return0;}

else{if(C)return0;return(x2 - x1 +1)\*(y2 - y1 +1);}}

Point in ConvexPloygon:

bool inConvexPoly(vector<pair <int,int>>&P, pair <int,int> q){

pii fix = P[0];int st =1, ed = P.size()-1, mid;

while(ed - st >1){

mid =(st+ed)>>1;if(triArea2(fix, P[mid], q)>0) st = mid;else ed = mid;}

if(triArea2(fix, P[st], q)<0)returnfalse;

if(triArea2(P[st],P[ed], q)<0)returnfalse;if(triArea2(P[ed], fix, q)<0)returnfalse;

returntrue;}

/\* MAx point cover with a circle of Radious R \*/

int maxPointCover(double radius, point pnt[],int n){

int i, j, ret =(bool) n, cnt;vector< pair<double,int>> ep[2];

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

ep[0].clear(); ep[1].clear(); cnt =0;

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

if(pnt[i]== pnt[j]){ cnt++;continue;}

if(abs(pnt[j]- pnt[i])>2\* radius + eps)continue;

double ang = get\_angle(radius, abs(pnt[j]- pnt[i]), radius);

double curr = arg(pnt[j]- pnt[i]);

double seg\_st = remainder(curr - ang,2\*pi);

double seg\_ed = remainder(curr + ang,2\*pi);

if(seg\_st + eps <0&& seg\_ed > eps){

ep[0].pb(mp(0,+1)); ep[0].pb(mp(abs(seg\_st),-1));

ep[1].pb(mp(0,+1)); ep[1].pb(mp(seg\_ed,-1));}

elseif(seg\_st > eps && seg\_ed + eps <0){

ep[0].pb(mp(abs(seg\_ed),+1)); ep[0].pb(mp(pi,-1));

ep[1].pb(mp(seg\_st,+1)); ep[1].pb(mp(pi,-1));}

elseif(seg\_st > eps){ ep[1].pb(mp(seg\_st,+1)); ep[1].pb(mp(seg\_ed,-1));}

else{ ep[0].pb(mp(abs(seg\_st),+1)); ep[0].pb(mp(abs(seg\_ed),-1));}}

cnt += max(getMaxInt(ep[0]), getMaxInt(ep[1])); ret = max(ret, cnt);}

return ret;}

MinCostMaxFlow:

int src, snk, nNode, nEdge, fin[MAXN +5], pre[MAXN +5], dist[MAXN +5], cap[2\*MAXE+5], cost[2\*MAXE+5],Next[2\*MAXE+5], to[2\*MAXE+5], from[2\*MAXE+5];

inlinevoid init(int \_src,int \_snk,int nodes){

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

nNode = nodes, nEdge =0; src = \_src, snk = \_snk;}

inlinevoid addEdge(int u,int v,int \_cost,int \_cap){

from[nEdge]= u, to[nEdge]= v, cap[nEdge]= \_cap, cost[nEdge]= \_cost;

Next[nEdge]= fin[u], fin[u]= nEdge++;

from[nEdge]= v, to[nEdge]= u, cap[nEdge]=0, cost[nEdge]=-(\_cost);

Next[nEdge]= fin[v], fin[v]= nEdge++;assert(nEdge <=2\*MAXE);}

bool bellman(){

int iter, u, v, i;bool flag =true;

memset(dist,0x7f,sizeof(dist)); memset(pre,-1,sizeof(pre));

dist[src]=0;

for(iter =1; iter < nNode && flag; iter++){

flag =false;

for(u =1; u <= nNode; u++){

for(i = fin[u]; i >=0; i =Next[i]){

v = to[i];

if(cap[i]&& dist[v]> dist[u]+ cost[i]){

dist[v]= dist[u]+ cost[i]; pre[v]= i;

flag =true;}}}}

return(dist[snk]< INF);}

int mcmf(int&fcost){

int netflow, i, bot, u; netflow = fcost =0;

while(bellman()){

bot = INF;

for(u = pre[snk]; u >=0; u = pre[from[u]]) bot = min(bot, cap[u]);

for(u = pre[snk]; u >=0; u = pre[from[u]]){

cap[u]-= bot; cap[u^1]+= bot; fcost += bot \* cost[u];}

netflow += bot;}

return netflow;

}

SegmentSegmentIntersection:

inlineboolintersect(pii p1, pii p2, pii p3, pii p4){

int d1, d2, d3, d4;

d1 = order(p3, p4, p1); d2 = order(p3, p4, p2);

d3 = order(p1, p2, p3); d4 = order(p1, p2, p4);

if(((d1 <0&& d2 >0)||(d1 >0&& d2 <0))&&((d3 <0&& d4 >0)||(d3 >0&& d4 <0)))returntrue;

if(!d3 && onsegment(p1, p2, p3))returntrue;if(!d4 && onsegment(p1, p2, p4))returntrue;

if(!d1 && onsegment(p3, p4, p1))returntrue;if(!d2 && onsegment(p3, p4, p2))returntrue;

returnfalse;}

SOS DP:

void sos\_dp(int\*arr, LL dp[],int n){

int k =1, i, pos, mask;while((1<< k)<= n) k++;

for(i =0; i <= n; i++) dp[i]= arr[i];

for(pos =0; pos < k; pos++) for(mask =0; mask <(1<< k); mask++)

if(mask &(1<< pos)) dp[mask]+= dp[mask ^(1<< pos)];}

SuffixArray:

intPlc[MAXLG+5][MAXN+10], stp;int S[MAXN+10];

pair< pii ,int> L[MAXN+10];

voidGenerate\_SA(string str){

int i, j, k, len = str.size(), cur;

for(i =0; i < len; i++)Plc[0][i]= str[i];

for(cur = stp =1;(cur>>1)< len; cur \*=2, stp++){

for(i =0; i < len; i++){L[i].xx.xx =Plc[stp-1][i];

L[i].xx.yy = i+cur < len?Plc[stp-1][i+cur]:-inf;L[i].yy = i;}

sort(L, L+len);

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

if(!i || L[i-1].xx.xx != L[i].xx.xx || L[i-1].xx.yy != L[i].xx.yy )Plc[stp][L[i].yy]= i;

elsePlc[stp][L[i].yy]=Plc[stp][L[i-1].yy];}}

for(i =0; i < len; i++) S[Plc[stp-1][i]]= i; stp--;}

int lcp(int u,int v,int N){// Here N = length of the string \*\*Call Generate\_SA(string)\*\*

int ret =0, k;if(u == v)return N-u;

for(k = stp; k >=0&& u < N && v < N; k--)if(Plc[k][u]==Plc[k][v]) ret +=1<<k, u +=1<<k, v +=1<<k;

return ret;}

Convex Hull for double valued points:

point Firstpoint;

int cmp(const void \*a,const void \*b) {

double x,y; point aa,bb; aa = \*(point \*)a; bb = \*(point \*)b;

x = isleft( Firstpoint, aa, bb ); if( x > eps ) return -1;

else if( x < -eps ) return 1; x = sq\_Distance( Firstpoint, aa );

y = sq\_Distance( Firstpoint, bb ); if( x + eps < y ) return -1;

return 1;}

void ConvexHull( point P[], point C[], int &nP, int &nC ) {

int i, j, pos = 0; // Remove duplicate points if necesary

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

if( P[i].y < P[pos].y || ( eq( P[i].y, P[pos].y ) && P[i].x > P[pos].x + eps ) )

pos = i;

swap( P[pos], P[0] ); Firstpoint = P[0];

qsort( P + 1, nP - 1, sizeof( point ), cmp ); C[0] = P[0]; C[1] = P[1]; i = 2, j = 1;

while( i < nP ) {

if( isleft( C[j-1], C[j], P[i] ) > -eps ) C[++j] = P[i++]; else j--;}

nC = j + 1;}

point to\_plane(point a, point b, point c, point p){

point ydir, xdir, res;

point cp = crsp(b - a, c - a); xdir = (b - a) / norm(b - a);

ydir = crsp(cp, xdir);ydir = ydir / norm(ydir);res.x = dotp(p-a, xdir);res.y = dotp(p-a, ydir);

return res;}

**Misc Geometric Formula:**

|  |  |
| --- | --- |
| **Triangle** | Circum Radius = a\*b\*c/(4\*area)  In Radius = area/s, where s = (a+b+c)/2  length of median to side c = sqrt(2\*(a\*a+b\*b)-c\*c)/2  length of bisector of angle C = sqrt(ab[(a+b)\*(a+b)-c\*c])/(a+b) |
| **Ellipse** | Area = PI\*a\*b  Circumference = 4a \*int(0,PI/2){sqrt(1-(k\*sint)\*(k\*sint))}dt  = 2\*PI\*sqrt((a\*a+b\*b)/2) approx  where k = sqrt((a\*a-b\*b)/a)  = PI\*(3\*(r1+r2)-sqrt[(r1+3\*r2)\*(3\*r1+r2)]) |
| **Spherical cap** | V = (1/3)\*PI\*h\*h\*(3\*r-h)  Surface Area = 2\*PI\*r\*h |
| **Spherical Sector** | V = (2/3)\*PI\*r\*r\*h |
| **Spherical Segment** | V = (1/6)\*PI\*h\*(3\*a\*a+3\*b\*b+h\*h) |
| **Torus** | V = 2\*PI\*PI\*R\*r\*r |
| **Truncated Conic** | V = (1/3)\*PI\*h\*(a\*a+a\*b+b\*b)  Surface Area = PI\*(a+b)\*sqrt(h\*h+(b-a)\*(b-a))  = PI\*(a+b)\*l |
| **Pyramidal frustum** | (1/3)\*h\*(A1+A2+sqrt(A1\*A2)) |
| **Length of an arc** | Sqrt( 1 + (f’(x))^2 ) |
| **Picks Theorem** | A = i + b/2 - 1. |
| **Crsp, dotp** | a x (b + c) = a x b + a x c; a x b = -(b x a);  a x ( b x c ) != (a x b) x c; A.(B+C) = A.B + A.C. |
| **H of Trapizium** | h=2\*tri\_area(a,b,d)/abs(a-c); a || c? |

**Misc Trigonometric Functions and Formulas:**

For a triangle: a/sin(A)=b/sin(B)=c/sin(C) = 2\*r

tan A/2 = +sqrt((1-cos A)/(1+cos A))

= sin A / (1+cos A)

= (1-cos A) / sin A

= cosec A – cot A

sin 3A = 3\*sin A – 4\*sincube A cos 3A = 4\*coscube A – 3\*cos A

tan 3A = (3\*tan A-tancube A)/(1-3\*tansq A)

sin 4A = 4\*sin A\*cos A – 8\*sincube A\*cos A

cos 4A = 8\*cos4 A – 8\*cossq A + 1

[r\*(cost+i\*sint)]p = rp\*(cos pt+i\*sin pt)

**a**cos**x** + **b**sin**x = c,** x = 2nπ + α ± β, where

cosα = a / (sqrt(a^2+b^2)), cosβ = c / (sqrt(a^2+b^2));

2sinAcosB = sin(A+B) + sin(A-B)

2cosAsinB = sin(A+B) - sin(A-B)

2cosAcosB = cos(A-B) + cos(A+B)

2sinAsinB = cos(A-B) – cos(A+B)

sinC + sinD = 2sin[(C+D)/2]cos[(C-D)/2]

sinC - sinD = 2cos[(C+D)/2]sin[(C-D)/2]

cosD + cosC = 2cos[(C+D)/2]cos[(C-D)/2]

cosD - cosC = 2sin[(C+D)/2]sin[(C-D)/2]

**Misc Integration Formula:**

a^x => a^x/ln(a)

1/sqrt(x\*x+a\*a) => ln(x+sqrt(x\*x+a\*a))

1/sqrt(x\*x-a\*a) => ln(x+sqrt(x\*x-a\*a))

1/(x\*sqrt(x\*x+a\*a) => -(1/a)\*ln([a+sqrt(x\*x+a\*a)]/x)

1/(x\*sqrt(a\*a-x\*x) => -(1/a)\*ln([a+sqrt(a\*a-x\*x)]/x)

**Misc Differentiation Formula:**

asin x => 1/sqrt(1-x\*x) acos x => -1/sqrt(1-x\*x)

atan x => 1/(1+x\*x) acot x => -1/(1+x\*x)

asec x => 1/[x\*sqrt(x\*x-1)] acosec x => -1/[x\*sqrt(x\*x-1)]

a^x => a^x\*ln(x) cot x => -cosecsq x

sec x => sec x \* tan x cosec x => -cosec x \* cot x

**Centroid of a 2D polygon:**

As in the calculation of the area above, xN is assumed to be x0, in other words the polygon is closed.



**Misc NT and GT Trivias:**

|  |  |
| --- | --- |
| **Extended Euclid** | x' = x + k\*(b/g) y' = y - k\*(a/g) |
| **Euler Phi** | E(phi(d)): d| n, equals to n  Capture.JPG |
| **Catalan Numbers** | (2n)! / ( (n+1)! n!) |
| **Pythagorian Triple** | (m^2 - n^2, 2mn, m^2+n^2) |
| **Linearity Of exp** | E[X+Y+Z] = E[X] + E[Y] + E[Z] |
| **Stirling Number[1]** | Capture.JPG  number of [permutations](https://en.wikipedia.org/wiki/Permutation) of n elements with k disjoint cycle |
| **Stirling Number[2]** | Capture.JPG  ways to a set of n objects into k non-empty subsets |
| **Bell Number** | ways to partition a set that has exactly n elements 1, 1, 2, 5, 15, 52, 203, 877 |
| **Number of MST** | Capture.JPG |