1. MATH

#include…

// calculate the gcd

ll gcd(ll a, ll b) { return b ? gcd(b, a % b) : a; }

// calculate the lcm

ll lcm(ll a, ll b) { return a / gcd(a, b) \* b; }

// calculate lg2(a)

int lg2(ll a) { return 63 - \_\_builtin\_clzll(a); }

// count the number of 1 bits

int bitcount(ll a) { return \_\_builtin\_popcountll(a); }

// calculate ceil(a / b)

ll ceildiv(ll a, ll b) {

if (b < 0) return ceildiv(-a, -b);

if (a >= 0) return a / b;

return -(ll)(((ull)(-a) + b - 1) / b);

}

// Calculate a \* b mod c (or a ^ b mod c) for 0 <= a, b, c <= 7.2 \* 10^18

// O(1) for modmul, O(logb) for modpow

ull modmul(ull a, ull b, ull M) {

ll ret = a \* b - M \* ull(1.L / M \* a \* b);

return ret + M \* (ret < 0) - M \* (ret >= (ll)M);

}

ull modpow(ull b, ull e, ull mod) {

ull ans = 1;

for (; e; b = modmul(b, b, mod), e /= 2)

if (e & 1) ans = modmul(ans, b, mod);

return ans;

}

// (a ^ b) % m in O(logb) where b is small

ll modpow(ll a, ll b, ll m) {

ll r = 1;

for (a %= m; b; b >>= 1, a = (a \* a) % m) if (b & 1) r = (r \* a) % m;

return r;

}

// return the smallest x > 0 such that a^x === b (mod m), or -1 if no

// such x exists. modLog(a, 1, m) can be used to calculate the order of a

// Time O(sqrt(m))

ll modLog(ll a, ll b, ll m) {

ll n = (ll) sqrt(m) + 1, e = 1, f = 1, j = 1;

unordered\_map<ll, ll> A;

while (j <= n && (e = f = e \* a % m) != b % m)

A[e \* b % m] = j++;

if (e == b % m) return j;

if (\_\_gcd(m, e) == \_\_gcd(m, b))

for (int i = 2; i < n + 2; ++i) if (A.count(e = e \* f % m))

return n \* i - A[e];

return -1;

}

// Find x such that x^2 === a (mod p) (-x gives the other solution)

// Time: O(log^2 (p)) worst case, O(logp) for most p

ll sqrt(ll a, ll p) {

// modpow using in this algorithm is for ll, not for ull

a %= p; if (a < 0) a += p;

if (a == 0) return 0;

assert(modpow(a, (p-1)/2, p) == 1); // else no solution

if (p % 4 == 3) return modpow(a, (p+1)/4, p);

// a^(n+3)/8 or 2^(n+3)/8 \* 2^(n-1)/4 works if p % 8 == 5

ll s = p - 1, n = 2;

int r = 0, m;

while (s % 2 == 0)

++r, s /= 2;

/// find a non-square mod p

while (modpow(n, (p - 1) / 2, p) != p - 1) ++n;

ll x = modpow(a, (s + 1) / 2, p);

ll b = modpow(a, s, p), g = modpow(n, s, p);

for (;; r = m) {

ll t = b;

for (m = 0; m < r && t != 1; ++m)

t = t \* t % p;

if (m == 0) return x;

ll gs = modpow(g, 1LL << (r - m - 1), p);

g = gs \* gs % p;

x = x \* gs % p;

b = b \* g % p;

}

}

// find a pair (c, d) such that ac + bd = gcd(a, b)

pair<ll, ll> extended\_gcd(ll a, ll b) {

if (b == 0) return {1, 0};

auto t = extended\_gcd(b, a % b);

return {t.second, t.first - t.second \* (a / b)};

}

// find x in [0, m) such that ax === gcd(a, m) (mod m)

ll modinverse(ll a, ll m) {

return (extended\_gcd(a, m).first % m + m) % m;

}

// calculate mod inverse in range 1 ~ n

void calc\_range\_modinv(int n, int mod, ll ret[]) {

ret[1] = 1LL;

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

ret[i] = (ll)(mod - mod / i) \* ret[mod % i] % mod;

}

// Find fibonacci in range [0, n]

vector<ll> fibonacci(int n) {

vector<ll>f(n);

f[0] = 0; f[1] = 1;

for (int i = 2; i <= n; ++i) f[i] = f[i - 1] + f[i - 2];

return f;

}

// Find n-th fibonnacci in O(logN), call fib(n)

void multiply(ll F[2][2], ll M[2][2]);

void power(ll F[2][2], ll n);

ll fib(int n) {

ll F[2][2] = {{1LL, 1LL}, {1LL, 0LL}};

if (n == 0)

return 0;

power(F, n - 1);

return F[0][0];

}

void power(ll F[2][2], int n) {

if(n == 0 || n == 1)

return;

ll M[2][2] = {{1LL, 1LL}, {1LL, 0LL}};

power(F, n / 2);

multiply(F, F);

if (n % 2 != 0)

multiply(F, M);

}

void multiply(ll F[2][2], ll M[2][2]) {

int x = F[0][0] \* M[0][0] + F[0][1] \* M[1][0];

int y = F[0][0] \* M[0][1] + F[0][1] \* M[1][1];

int z = F[1][0] \* M[0][0] + F[1][1] \* M[1][0];

int w = F[1][0] \* M[0][1] + F[1][1] \* M[1][1];

F[0][0] = x;

F[0][1] = y;

F[1][0] = z;

F[1][1] = w;

}

// Find nth-fibonacci in O(logN)

map<int, ll> f;

ll fib(int n) {

if (n == 0)

return 0;

if (n == 1 || n == 2)

return (f[n] = 1LL);

if (f[n])

return f[n];

int k = (n & 1)? (n+1)/2 : n/2;

f[n] = (n & 1)? (fib(k)\*fib(k) + fib(k-1)\*fib(k-1))

: (2\*fib(k-1) + fib(k))\*fib(k);

return f[n];

}

// calculate catalan number

vector<ll> catalan(int n) {

vector<ll>cat(n);

cat[0] = 1;

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

return cat;

}

// Calculate A ^ B mod C when A or B very large

// support for calculate when a and b is very large

ll aModM(string s, ll mod) {

ll res = 0;

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

res = (res \* 10 + (s[i] - '0'));

res %= mod;

}

return res;

}

// 'a' very large O(len(a) + log(b))

ll ApowBmodM(string &a, ll b, ll m) {

ll ans = aModM(a, m);

return modpow(ans, b, m);

}

// 'b' very large và 'm' is a prime O(len(b))

ll ApowBmodM(ll a, string &b, ll m) {

ll remB = 0;

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

remB = (remB \* 10 + (b[i] - '0')) % (m - 1);

return modpow(a, remB, m);

}

// both 'a' and 'b' is very large O(len(a) + len(b) + log(y))

ll ApowBmodM(string &a, string &b, ll m) {

ll res = 1;

ll x = aModM(a, m);

ll y = aModM(b, m);

res = modpow(x, y, m);

return (res % m + m) % m;

}

// find two integers x and y such that ax + by = gcd(a, b)

void euclid(ll a, ll b, ll &x, ll &y, ll &d) {

if (b) euclid(b, a % b, y, x, d), y -= x \* (a / b);

else x = 1, y = 0, d = a;

}

// find any solution for a \* x + b \* y = c

bool find\_any\_solution(ll a, ll b, int c, ll &x0, ll &y0, ll &g) {

euclid(abs(a), abs(b), x0, y0, g);

if (c % g) return false;

x0 \*= c / g;

y0 \*= c / g;

if (a < 0) x0 = -x0;

if (b < 0) y0 = -y0;

return true;

}

// All x' and y' is also a solution with any number k

// x' = x + k \* b / gcd

// y' = y - k \* a / gcd

void shift\_solution(ll &x, ll &y, ll a, ll b, ll cnt) {

x += cnt \* b;

y -= cnt \* a;

}

// Find the number of solution in range [minx, maxx], [miny, maxy]

int find\_all\_solutions(ll a, ll b, ll c, ll minx, ll maxx, ll miny, ll maxy) {

ll x, y, g;

if (!find\_any\_solution(a, b, c, x, y, g))

return 0;

a /= g;

b /= g;

int sign\_a = a > 0 ? +1 : -1;

int sign\_b = b > 0 ? +1 : -1;

shift\_solution(x, y, a, b, (minx - x) / b);

if (x < minx)

shift\_solution(x, y, a, b, sign\_b);

if (x > maxx)

return 0;

int lx1 = x;

shift\_solution(x, y, a, b, (maxx - x) / b);

if (x > maxx)

shift\_solution(x, y, a, b, -sign\_b);

int rx1 = x;

shift\_solution(x, y, a, b, -(miny - y) / a);

if (y < miny)

shift\_solution(x, y, a, b, -sign\_a);

if (y > maxy)

return 0;

int lx2 = x;

shift\_solution(x, y, a, b, -(maxy - y) / a);

if (y > maxy)

shift\_solution(x, y, a, b, sign\_a);

int rx2 = x;

if (lx2 > rx2)

swap(lx2, rx2);

int lx = max(lx1, lx2);

int rx = min(rx1, rx2);

if (lx > rx)

return 0;

return (rx - lx) / abs(b) + 1;

}

// Solve:

// t = a mod m1

// t = b mod m2

// ans = t mod lcm(m1, m2)

bool chinese\_remainder(ll a, ll b, ll m1, ll m2, ll &ans, ll &lcm) {

ll x, y, g, c = b - a;

euclid(m1, m2, x, y, g);

if (c % g) return false;

lcm = m1 / g \* m2;

ans = ((a + c / g \* x % (m2 / g) \* m1) % lcm + lcm) % lcm;

return true;

}

// Eratosthenes

// Prime sieve, isprime[i] is true if i is a prime

// O(n \* loglogn)

const int MAX\_PR = 5'000'000;

bitset<MAX\_PR> isprime;

vector<int> eratosthenesSieve(int lim) {

isprime.set(); isprime[0] = isprime[1] = 0;

for (int i = 4; i < lim; i += 2) isprime[i] = 0;

for (int i = 3; i\*i < lim; i += 2) if (isprime[i])

for (int j = i\*i; j < lim; j += i\*2) isprime[j] = 0;

vector<int> pr;

for (int i = 2; i < lim; ++i) if (isprime[i]) pr.push\_back(i);

return pr;

}

// Fast Eratosthenes

const int LIM = 1e6;

bitset<LIM> isPrime;

vector<int> eratosthenes() {

const int S = (int)round(sqrt(LIM)), R = LIM / 2;

vector<int> pr = {2}, sieve(S+1); pr.reserve(int(LIM/log(LIM)\*1.1));

vector<pair<int, int>> cp;

for (int i = 3; i <= S; i += 2) if (!sieve[i]) {

cp.push\_back({i, i \* i / 2});

for (int j = i \* i; j <= S; j += 2 \* i) sieve[j] = 1;

}

for (int L = 1; L <= R; L += S) {

array<bool, S> block{};

for (auto &[p, idx] : cp)

for (int i=idx; i < S+L; idx = (i+=p)) block[i-L] = 1;

for (int i = 0; i < min(S, R - L); ++i)

if (!block[i]) pr.push\_back((L + i) \* 2 + 1);

}

for (int i : pr) isPrime[i] = 1;

return pr;

}

// linear sieve, other version of sieve of eratosthenes in O(n)

vector<int> linear\_sieve(int n) {

vector<int>pr;

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

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

if (lp[i] == 0) {

lp[i] = i;

pr.push\_back(i);

}

for (int j = 0; j < (int)pr.size() && pr[j] <= lp[i] && i \* pr[j] <= n; ++j) {

lp[i \* pr[j]] = pr[j];

}

}

return pr;

}

// Count the number of factor in range 1 ~ n

// If you want to calculate the sum of factor change ++ to += i

// O(n \* logn)

void num\_of\_divisors(int n, ll ret[]) {

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

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

ret[j]++;

}

// euler phi of number in range 1 ~ n

// phi(n) = number of x such that 0 < x < n && gcd(n, x) = 1

// O(n \* loglogn)

void euler\_phi(int n, ll ret[]) {

for (ll i = 0; i <= n; ++i) ret[i] = i & 1 ? i : i / 2;

for (ll i = 3; i <= n; i += 2)

if (ret[i] == i)

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

ret[j] -= ret[j] / i;

}

// Miller-Rabin for primality test. Guaranteed to work for numbers

// up to 7 \* 10^18; for large numbers, use Python and extend A randomly.

// 7 times the complexity a^b mod c

bool isPrime(ull n) {

if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;

ull A[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022},

s = \_\_builtin\_ctzll(n-1), d = n >> s;

for (ull a : A) { // ^ count trailing zeroes

ull p = modpow(a%n, d, n), i = s; // modpow using ull

while (p != 1 && p != n - 1 && a % n && i--)

p = modmul(p, p, n);

if (p != n-1 && i != s) return 0;

}

return 1;

}

// n < 2,047 base = {2};

// n < 1,373,653 base = {2, 3};

// n < 9,080,191 base = {31, 73};

// n < 25,326,001 base = {2, 3, 5};

// n < 3,215,031,751 base = {2, 3, 5, 7};

// n < 4,759,123,141 base = {2, 3, 7, 61};

// n < 1,122,004,669,633 base = {2, 13, 23, 1662803};

// n < 2,152,302,898,747 base = {2, 3, 5, 7, 11};

// n < 3,474,749,660,383 base = {2, 3, 5, 7, 11, 13, 17};

// n < 3,825,123,056,413,051 base = {2, 3, 5, 7, 11, 13, 17, 19, 23};

// n < 318,665,857,834,031,151,167,461 base = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};

// n < 3,317,044,064,679,887,385,961,981 base = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41};

// factor of a number in O(logN)

const int MAXN = 1e6 + 1;

int spf[MAXN];

void sieve() {

spf[1] = 1;

for (ll i = 2; i < MAXN; ++i)

spf[i] = i;

for (ll i = 4; i < MAXN; i += 2)

spf[i] = 2;

for (ll i = 3; i \* i < MAXN; ++i)

if (spf[i] == i)

for (ll j = i \* i; j < MAXN; j += i)

if (spf[j] == j)

spf[j] = i;

}

vector<ll> getFactorization(int x) {

vector<ll>ret;

while(x != 1) {

ret.push\_back(spf[x]);

x = x / spf[x];

}

return ret;

}

// Find the prime factor in O(sqrt(n), can factor up to 9 \* 10^13

// With larger number, use Pollard Rho

vector<int> prime\_factor(int n) {

vector<int>factors;

vector<int>primes = eratosthenesSieve(n);

int ind = 0, pf = primes[0];

while(pf \* pf <= n) {

while(n % pf == 0) n /= pf, factors.push\_back(pf);

pf = primes[++ind];

}

if (n != 1) factors.push\_back(n);

return factors;

}

// Pollard-rho randomized factorization algorithm. Return prime

// factors of a number, in arbitrary order

// time: O(n^(1/4)) use with modpow ull and miller

ull pollard(ull n) {

auto f = [n](ull x) { return modmul(x, x, n) + 1; };

ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;

while (t++ % 40 || \_\_gcd(prd, n) == 1) {

if (x == y) x = ++i, y = f(x);

if ((q = modmul(prd, max(x,y) - min(x,y), n))) prd = q;

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

}

return \_\_gcd(prd, n);

}

vector<ull> factor(ull n) {

if (n == 1) return {};

if (isPrime(n)) return {n};

ull x = pollard(n);

auto l = factor(x), r = factor(n / x);

l.insert(l.end(), r.begin(), r.end());

return l; }

1. STRING

//////////////////////KMP

const int N = 1e6 + 5;

int b[N];

int cnt=0;

//thiết lập đánh dấu xâu b

void kmppre(string &p,int m) {

b[0] = -1;

for (int i = 0, j = -1; i < m; b[++i] = ++j)

while (j >= 0 and p[i] != p[j])

j = b[j];

}

//truy tìm vị trí xâu b xuất hiện xâu trong a

void kmp(string &s,string &p,int n,int m) {

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

while (j >= 0 and s[i] != p[j]) j=b[j];

i++, j++;

if (j == m) {

//vitri: match position i-j

//soluong: cout<<(i-j+1)<<" ";

//cnt++;

j = b[j];

}

}

}

string d,b;

cin>>d;

cin>>b;

int n=(int)d.size();

int m=(int)b.size();

kmppre(b,m);

kmp(d,b,n,m);

cout<<cnt;

////////////////end\_KMP

// Manacher (Longest Palindromic String) - O(n)

int lps[2\*M+5];

int manacher(string &s,int &n) {

string p (2\*n+3, '#');

p[0] = '^';

for (int i = 0; i < n; i++) p[2\*(i+1)] = s[i];

p[2\*n+2] = '$';

int k = 0, r = 0, m = 0;

int l = p.length();

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

int o = 2\*k - i;

lps[i] = (r > i) ? min(r-i, lps[o]) : 0;

while (p[i + 1 + lps[i]] == p[i - 1 - lps[i]]) lps[i]++;

if (i + lps[i] > r) k = i, r = i + lps[i];

m = max(m, lps[i]);

}

return m;

}

int n; cin>>n; string s; cin>>s;

cout<<manacher(s,n);

///////end\_manacher

//tim cac xau lyndon (co thu tu tu` dien nho nhat);

void lyndon(string s) {

int n = (int) s.length();

int i = 0;

while (i < n) {

int j = i + 1, k = i;

while (j < n && s[k] <= s[j]) {

if (s[k] < s[j]) k = i;

else ++k;

++j;

}

while (i <= k) {

cout << s.substr(i, j - k) << ' ';

i += j - k;

}

}

cout << endl;

}

string s;cin>>s;lyndon(s);

////end\_lyndon

////manacher\_select

vector<int> manacher\_odd(string s) {

int n = s.size();

s = "$" + s + "^";

vector<int> p(n + 2);

int l = 0, r = -1;

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

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

while(s[i - p[i]] == s[i + p[i]]) {

p[i]++;

}

if(i + p[i] > r) {

l = i - p[i], r = i + p[i];

}

}

return vector<int>(begin(p) + 1, end(p) - 1);

}

vector<int>temp=manacher\_odd("abacd");

for(auto it:temp) cout<<it<<" ";

////end\_manacher\_select;

// Tính vị trí của xâu xoay vòng có thứ tự từ điển nhỏ nhất của xâu s[]

int minmove(string s) {

int n = (int)s.length();

int x, y, i, j, u, v; // x là vị trí chuỗi nhỏ nhất trước chuỗi y.

for (x = 0, y = 1; y < n; ++ y) {

i = u = x;

j = v = y;

while (s[i] == s[j]) {

++ u; ++ v;

if (++ i == n) i = 0;

if (++ j == n) j = 0;

if (i == x) break; // tất cả kí tự đều bằng nhau

}

if (s[i] <= s[j]) y = v;

else {

x = y;

if (u > y) y = u;

}

}

return x;

}

cout<<minmove(s);

////end\_minimal\_string\_rotation;

//for trie

const ll N = 250000+5;

int trie[N][26], trien = 0; //\*trien\* dai dien cho so node cua cay

int finish\_u[N],fs[N];

//them mot ki tu vao cay

int add(int u, char c){

c-='a';

if (trie[u][c]) return trie[u][c];

return trie[u][c] = ++trien; //ki tu c chua xuat hien thi tang them mot node

}

int main (){

ios\_base::sync\_with\_stdio(false);

cin.tie(0); cout.tie(0);

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

string s;cin>>s;int u = 0;

//them 1 string vao trie

for(char c : s){

u=add(u, c);

}

finish\_u[u]=1;

}

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

for(int j=0;j<26;++j)

if(trie[i][j]) fs[i]=max(fs[i],fs[trie[i][j]]);

fs[i]+=finish\_u[i];

}

cout<<fs[0];

return 0;

}

///end\_trie;

1. GEOMETRY

#define st first

#define nd second

#define pb push\_back

#define cl(x,v) memset((x), (v), sizeof(x))

#define db(x) cerr << #x << " == " << x << endl

#define dbs(x) cerr << x << endl

#define \_ << ", " <<

typedef long long ll;

typedef long double ld;

typedef pair<int,int> pii;

typedef pair<int, pii> piii;

typedef pair<ll,ll> pll;

typedef pair<ll, pll> plll;

typedef vector<int> vi;

typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);

const ll LINF = 0x3f3f3f3f3f3f3f3f;

const int INF = 0x3f3f3f3f, MOD = 1e9+7;

const int N = 1e5+5;

typedef long double type;

//for big coordinates change to long long

//lớn hơn

bool ge(type x, type y) { return x + EPS > y; }

//nhỏ hơn

bool le(type x, type y) { return x - EPS < y; }

//bằng

bool eq(type x, type y) { return ge(x, y) and le(x, y); }

struct point {

type x, y;

point() : x(0), y(0) {}

point(type x, type y) : x(x), y(y) {}

point operator -() { return point(-x, -y); }

point operator +(point p) { return point(x + p.x, y + p.y); }

point operator -(point p) { return point(x - p.x, y - p.y); }

point operator \*(type k) { return point(k\*x, k\*y); }

point operator /(type k) { return point(x/k, y/k); }

//inner product

type operator \*(point p) { return x\*p.x + y\*p.y; }

//cross product

type operator %(point p) { return x\*p.y - y\*p.x; }

bool operator ==(const point &p) const{ return x == p.x and y == p.y; }

bool operator !=(const point &p) const{ return x != p.x or y != p.y; }

bool operator <(const point &p) const { return (x < p.x) or (x == p.x and y < p.y); }

// 0 => cùng hướng

// 1 => p ở bên trái

// - 1 => p ở bên phải

int dir(point o, point p) {

type x = (\*this - o) % (p - o);

return ge(x,0) - le(x,0);

}

bool on\_seg(point p, point q) {

if (this->dir(p, q)) return 0;

return ge(x, min(p.x, q.x)) and le(x, max(p.x, q.x)) and ge(y, min(p.y, q.y)) and le(y, max(p.y, q.y));

}

//ĐỘ DÀI

ld abs() { return sqrt(x\*x + y\*y); }

type abs2() { return x\*x + y\*y; }

//KHOẢNG CÁCH GIỮA 2 ĐIỂM

ld dist(point q) { return (\*this - q).abs(); }

type dist2(point q) { return (\*this - q).abs2(); }

ld arg() { return atan2l(y, x); }

//Hình chiếu của 1 điểm lên vectơ y

point project(point y) { return y \* ((\*this \* y) / (y \* y)); }

//Hình chiếu của 1 điểm lên đường thẳng được tạo bởi 2 điểm x, y

point project(point x, point y) { return x + (\*this - x).project(y-x); }

//Khoảng cách của 1 điểm đến 1 ĐƯỜNG thẳng được tạo bởi 2 điểm x, y

ld dist\_line(point x, point y) { return dist(project(x, y)); }

//Khoảng cách của 1 điểm đến 1 ĐOẠN thẳng được tạo bởi 2 điểm x, y

ld dist\_seg(point x, point y) {

return project(x, y).on\_seg(x, y) ? dist\_line(x, y) : min(dist(x), dist(y));

}

point rotate(ld sin, ld cos) { return point(cos\*x - sin\*y, sin\*x + cos\*y); }

point rotate(ld a) { return rotate(sin(a), cos(a)); }

// xoay xung quanh đối số của vectơ p

point rotate(point p) { return rotate(p.x / p.abs(), p.y / p.abs()); }

};

//HƯỚNG

int direction(point o, point p, point q) { return p.dir(o, q); }

//QUAY ngược CHIỀU KIM ĐỒNG HỒ

point rotate\_ccw90(point p) { return point(-p.y,p.x); }

//QUAY cùng CHIỀU KIM ĐỒNG HỒ

point rotate\_cw90(point p) { return point(p.y,-p.x); }

//for reading purposes avoid using \* and % operators, use the functions below:

type dot(point p, point q) { return p.x\*q.x + p.y\*q.y; }

type cross(point p, point q) { return p.x\*q.y - p.y\*q.x; }

//diện tích tam giác \* 2

type area\_2(point a, point b, point c) { return fabs(cross(a,b) + cross(b,c) + cross(c,a)); }

int angle\_less(const point& a1, const point& b1, const point& a2, const point& b2) {

// góc giữa (a1 và b1) so với góc giữa (a2 và b2)

// 1: lớn hơn

// - 1: nhỏ hơn

// 0: bằng nhau

point p1(dot( a1, b1), abs(cross( a1, b1)));

point p2(dot( a2, b2), abs(cross( a2, b2)));

if(cross(p1, p2) < 0) return 1;

if(cross(p1, p2) > 0) return -1;

return 0;

}

ostream &operator<<(ostream &os, const point &p) {

os << "(" << p.x << "," << p.y << ")";

return os;

}

//lines

//hình chiếu của điểm lên đường thẳng AB

point project\_point\_line(point c, point a, point b) {

ld r = dot(b - a, b - a);

if (fabs(r) < EPS) return a;

return a + (b - a)\*dot(c - a, b - a)/dot(b - a, b - a);

}

//hình chiếu của điểm lên tia AB

point project\_point\_ray(point c, point a, point b) {

ld r = dot(b - a, b - a);

if (fabs(r) < EPS) return a;

r = dot(c - a, b - a) / r;

if (le(r, 0)) return a;

return a + (b - a)\*r;

}

//hình chiếu của điểm lên đoạn thẳng

point project\_point\_segment(point c, point a, point b) {

ld r = dot(b - a, b - a);

if (fabs(r) < EPS) return a;

r = dot(c - a, b - a) / r;

if (le(r, 0)) return a;

if (ge(r, 1)) return b;

return a + (b - a)\*r;

}

//bắt đầu độ dài của hình chiếu

ld distance\_point\_line(point c, point a, point b) {

return c.dist2(project\_point\_line(c, a, b));

}

ld distance\_point\_ray(point c, point a, point b) {

return c.dist2(project\_point\_ray(c, a, b));

}

ld distance\_point\_segment(point c, point a, point b) {

return c.dist2(project\_point\_segment(c, a, b));

}

//kết thúc độ dài của hình chiếu

ld distance\_point\_plane(ld x, ld y, ld z, ld a, ld b, ld c, ld d){

return fabs(a\*x + b\*y + c\*z - d)/sqrt(a\*a + b\*b + c\*c);

}

//kiểm tra ab // cd

bool lines\_parallel(point a, point b, point c, point d) {

return fabs(cross(b - a, d - c)) < EPS;

}

//kiểm tra 4 điểm thẳng hàng

bool lines\_collinear(point a, point b, point c, point d) {

return lines\_parallel(a, b, c, d) && fabs(cross(a-b, a-c)) < EPS && fabs(cross(c-d, c-a)) < EPS;

}

//giao điểm của 2 đường thẳng

point lines\_intersect(point p, point q, point a, point b) {

point r = q - p, s = b - a, c(p % q, a % b);

if (eq(r % s, 0)) return point(LINF, LINF);

return point(point(r.x, s.x) % c, point(r.y, s.y) % c) / (r % s);

}

//kiểm tra 2 tia trùng nhau (LineLineIntersection) trước khi sử dụng hàm này

//giao điểm của 2 đường thẳng

point compute\_line\_intersection(point a, point b, point c, point d) {

b = b - a; d = c - d; c = c - a;

assert(dot(b, b) > EPS && dot(d, d) > EPS);

return a + b\*cross(c, d)/cross(b, d);

}

//kiểm tra 2 đường thẳng trùng nhau

bool line\_line\_intersect(point a, point b, point c, point d) {

if(!lines\_parallel(a, b, c, d)) return true;

if(lines\_collinear(a, b, c, d)) return true;

return false;

}

//hướng a -> b, c -> d

//kiểm tra 2 tia giao nhau

bool ray\_ray\_intersect(point a, point b, point c, point d){

if (a.dist2(c) < EPS || a.dist2(d) < EPS ||

b.dist2(c) < EPS || b.dist2(d) < EPS) return true;

if (lines\_collinear(a, b, c, d)) {

if(ge(dot(b - a, d - c), 0)) return true;

if(ge(dot(a - c, d - c), 0)) return true;

return false;

}

if(!line\_line\_intersect(a, b, c, d)) return false;

point inters = lines\_intersect(a, b, c, d);

if(ge(dot(inters - c, d - c), 0) && ge(dot(inters - a, b - a), 0)) return true;

return false;

}

//kiểm tra đoạn thẳng giao nhau

bool segment\_segment\_intersect(point a, point b, point c, point d) {

if (a.dist2(c) < EPS || a.dist2(d) < EPS ||

b.dist2(c) < EPS || b.dist2(d) < EPS) return true;

int d1, d2, d3, d4;

d1 = direction(a, b, c);

d2 = direction(a, b, d);

d3 = direction(c, d, a);

d4 = direction(c, d, b);

if (d1\*d2 < 0 and d3\*d4 < 0) return 1;

return a.on\_seg(c, d) or b.on\_seg(c, d) or

c.on\_seg(a, b) or d.on\_seg(a, b);

}

//kiểm tra đoạn thẳng giao với đường thẳng

bool segment\_line\_intersect(point a, point b, point c, point d){

if(!line\_line\_intersect(a, b, c, d)) return false;

point inters = lines\_intersect(a, b, c, d);

if(inters.on\_seg(a, b)) return true;

return false;

}

//hướng tia c -> d

//đoạn thẳng giao với tia

bool segment\_ray\_intersect(point a, point b, point c, point d){

if (a.dist2(c) < EPS || a.dist2(d) < EPS ||

b.dist2(c) < EPS || b.dist2(d) < EPS) return true;

if (lines\_collinear(a, b, c, d)) {

if(c.on\_seg(a, b)) return true;

if(ge(dot(d - c, a - c), 0)) return true;

return false;

}

if(!line\_line\_intersect(a, b, c, d)) return false;

point inters = lines\_intersect(a, b, c, d);

if(!inters.on\_seg(a, b)) return false;

if(ge(dot(inters - c, d - c), 0)) return true;

return false;

}

//hướng tia a -> b

//kiểm tra tia giao với đường thẳng

bool ray\_line\_intersect(point a, point b, point c, point d){

if (a.dist2(c) < EPS || a.dist2(d) < EPS ||

b.dist2(c) < EPS || b.dist2(d) < EPS) return true;

if (!line\_line\_intersect(a, b, c, d)) return false;

point inters = lines\_intersect(a, b, c, d);

if(!line\_line\_intersect(a, b, c, d)) return false;

if(ge(dot(inters - a, b - a), 0)) return true;

return false;

}

//khoảng cách giữa đoạn thẳng với đường thẳng

ld distance\_segment\_line(point a, point b, point c, point d){

if(segment\_line\_intersect(a, b, c, d)) return 0;

return min(distance\_point\_line(a, c, d), distance\_point\_line(b, c, d));

}

//khoảng cách giữa đoạn thẳng với tia

ld distance\_segment\_ray(point a, point b, point c, point d){

if(segment\_ray\_intersect(a, b, c, d)) return 0;

ld min1 = distance\_point\_segment(c, a, b);

ld min2 = min(distance\_point\_ray(a, c, d), distance\_point\_ray(b, c, d));

return min(min1, min2);

}

//khoảng cách giữa đoạn thẳng với đoạn thẳng

ld distance\_segment\_segment(point a, point b, point c, point d){

if(segment\_segment\_intersect(a, b, c, d)) return 0;

ld min1 = min(distance\_point\_segment(c, a, b), distance\_point\_segment(d, a, b));

ld min2 = min(distance\_point\_segment(a, c, d), distance\_point\_segment(b, c, d));

return min(min1, min2);

}

//khoảng cách giữa tia với đường thẳng

ld DistanceRayLine(point a, point b, point c, point d){

if(ray\_line\_intersect(a, b, c, d)) return 0;

ld min1 = distance\_point\_line(a, c, d);

return min1;

}

//khoảng cách giữa tia với tia

ld DistanceRayRay(point a, point b, point c, point d){

if(ray\_ray\_intersect(a, b, c, d)) return 0;

ld min1 = min(distance\_point\_ray(c, a, b), distance\_point\_ray(a, c, d));

return min1;

}

//khoảng cách giữa đường thẳng với đường thẳng

ld DistanceLineLine(point a, point b, point c, point d){

if(line\_line\_intersect(a, b, c, d)) return 0;

return distance\_point\_line(a, c, d);

}

//circle

struct circle {

point c;

ld r;

circle() { c = point(); r = 0; }

circle(point \_c, ld \_r) : c(\_c), r(\_r) {}

//tính diện tích

ld area() { return acos(-1.0)\*r\*r; }

//tính độ dài cung tròn ứng với góc rad nào đó

ld chord(ld rad) { return 2\*r\*sin(rad/2.0); }

//diện tích hình quạt ứng với góc rad nào đó

ld sector(ld rad) { return 0.5\*rad\*area()/acos(-1.0); }

//2 đường tròn giao nhau

bool intersects(circle other) {

return le(c.dist(other.c), r + other.r);

}

//điểm nằm trong đường tròn

bool contains(point p) { return le(c.dist(p), r); }

//lấy điểm tiếp tuyến từ 1 điểm p đến đường tròn

pair<point, point> getTangentPoint(point p) {

ld d1 = c.dist(p), theta = asin(r/d1);

point p1 = (c - p).rotate(-theta);

point p2 = (c - p).rotate(theta);

p1 = p1\*(sqrt(d1\*d1 - r\*r)/d1) + p;

p2 = p2\*(sqrt(d1\*d1 - r\*r)/d1) + p;

return make\_pair(p1,p2);

}

};

//đường tròn ngoại tiếp tam giác

circle circumcircle(point a, point b, point c) {

circle ans;

point u = point((b - a).y, -(b - a).x);

point v = point((c - a).y, -(c - a).x);

point n = (c - b)\*0.5;

ld t = cross(u,n)/cross(v,u);

ans.c = ((a + c)\*0.5) + (v\*t);

ans.r = ans.c.dist(a);

return ans;

}

//tìm tâm của đường tròn ngoại tiếp tam giác

point compute\_circle\_center(point a, point b, point c) {

//circumcenter

b = (a + b)/2;

c = (a + c)/2;

return compute\_line\_intersection(b, b + rotate\_cw90(a - b), c, c + rotate\_cw90(a - c));

}

//kiểm tra điểm nằm bên trong đường tròn

// 0 bên trong

// 1 nằm trên đường tròn

// 2 bên ngoài

int inside\_circle(point p, circle c) {

if (fabs(p.dist(c.c) - c.r)<EPS) return 1;

else if (p.dist(c.c) < c.r) return 0;

else return 2;

}

//đường tròn nội tiếp tam giác

circle incircle( point p1, point p2, point p3 ) {

ld m1 = p2.dist(p3);

ld m2 = p1.dist(p3);

ld m3 = p1.dist(p2);

point c = (p1\*m1 + p2\*m2 + p3\*m3)\*(1/(m1 + m2 + m3));

ld s = 0.5\*(m1 + m2 + m3);

ld r = sqrt(s\*(s - m1)\*(s - m2)\*(s - m3))/s;

return circle(c, r);

}

//tìm kiếm đường tròn có bán kính bao quanh các điểm

circle minimum\_circle(vector<point> p) {

random\_shuffle(p.begin(), p.end());

circle C = circle(p[0], 0.0);

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

if (C.contains(p[i])) continue;

C = circle(p[i], 0.0);

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

if (C.contains(p[j])) continue;

C = circle((p[j] + p[i])\*0.5, 0.5\*p[j].dist(p[i]));

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

if (C.contains(p[k])) continue;

C = circumcircle(p[j], p[i], p[k]);

}

}

}

return C;

}

//Với tọa độ của tâm một vòng tròn và bán kính của nó, và phương trình của một đường thẳng

// tìm các giao điểm của đường thẳng với đường tròn

vector<point> circle\_line\_intersection(point a, point b, point c, ld r) {

vector<point> ret;

b = b - a;

a = a - c;

ld A = dot(b, b);

ld B = dot(a, b);

ld C = dot(a, a) - r\*r;

ld D = B\*B - A\*C;

if (D < -EPS) return ret;

ret.push\_back(c + a + b\*(sqrt(D + EPS) - B)/A);

if (D > EPS)

ret.push\_back(c + a + b\*(-B - sqrt(D))/A);

return ret;

}

//tìm tập giao điểm của 2 đường tròn với nhau

vector<point> circle\_circle\_intersection(point a, point b, ld r, ld R) {

vector<point> ret;

ld d = sqrt(a.dist2(b));

if (d > r + R || d + min(r, R) < max(r, R)) return ret;

ld x = (d\*d - R\*R + r\*r)/(2\*d);

ld y = sqrt(r\*r - x\*x);

point v = (b - a)/d;

ret.push\_back(a + v\*x + rotate\_ccw90(v)\*y);

if (y > 0)

ret.push\_back(a + v\*x - rotate\_ccw90(v)\*y);

return ret;

}

//polygon

//Graham scan with bugs, not safe, prefer monotone chain!

point origin;

//radial function for graham scan, for a generic radial sort see radial\_sort.cpp

bool radial(point p, point q) {

int dir = p.dir(origin, q);

return dir > 0 or (!dir and p.on\_seg(origin, q));

}

// Graham Scan O(nlog(n))

vector<point> graham\_hull(vector<point> pts) {

vector<point> ch(pts.size());

point mn = pts[0];

for(point p : pts) if (p.y < mn.y or (p.y == mn.y and p.x < p.y)) mn = p;

origin = mn;

sort(pts.begin(), pts.end(), radial);

int n = 0;

// IF: Convex hull without collinear points

for(point p : pts) {

while (n > 1 and ch[n-1].dir(ch[n-2], p) < 1) n--;

ch[n++] = p;

}

/\* ELSE IF: Convex hull with collinear points

for(point p : pts) {

while (n > 1 and ch[n-1].dir(ch[n-2], p) < 0) n--;

ch[n++] = p;

}

for(int i=pts.size()-1; i >=1; --i)

if (pts[i] != ch[n-1] and !pts[i].dir(pts[0], ch[n-1]))

ch[n++] = pts[i];

// END IF \*/

ch.resize(n);

return ch;

}

//Monotone chain O(nlog(n))

#define REMOVE\_REDUNDANT

#ifdef REMOVE\_REDUNDANT

bool between(const point &a, const point &b, const point &c) {

return (fabs(area\_2(a,b,c)) < EPS && (a.x-b.x)\*(c.x-b.x) <= 0 && (a.y-b.y)\*(c.y-b.y) <= 0);

}

#endif

void monotone\_hull(vector<point> &pts) {

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

pts.erase(unique(pts.begin(), pts.end()), pts.end());

vector<point> up, dn;

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

while (up.size() > 1 && area\_2(up[up.size()-2], up.back(), pts[i]) >= 0) up.pop\_back();

while (dn.size() > 1 && area\_2(dn[dn.size()-2], dn.back(), pts[i]) <= 0) dn.pop\_back();

up.push\_back(pts[i]);

dn.push\_back(pts[i]);

}

pts = dn;

for (int i = (int) up.size() - 2; i >= 1; i--) pts.push\_back(up[i]);

#ifdef REMOVE\_REDUNDANT

if (pts.size() <= 2) return;

dn.clear();

dn.push\_back(pts[0]);

dn.push\_back(pts[1]);

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

if (between(dn[dn.size()-2], dn[dn.size()-1], pts[i])) dn.pop\_back();

dn.push\_back(pts[i]);

}

if (dn.size() >= 3 && between(dn.back(), dn[0], dn[1])) {

dn[0] = dn.back();

dn.pop\_back();

}

pts = dn;

#endif

}

//avoid using long double for comparisons, change type and remove division by 2

// tính diện tích đa giác

ld compute\_signed\_area(const vector<point> &p) {

ld area = 0;

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

int j = (i+1) % p.size();

area += p[i].x\*p[j].y - p[j].x\*p[i].y;

}

return area / 2.0;

}

ld compute\_area(const vector<point> &p) {

return fabs(compute\_signed\_area(p));

}

//tính chu vi

ld compute\_perimeter(vector<point> &p) {

ld per = 0;

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

int j = (i+1) % p.size();

per += p[i].dist(p[j]);

}

return per;

}

//not tested

//tìm trọng tâm của đa giác

point compute\_centroid(vector<point> &p) {

point c(0,0);

ld scale = 6.0 \* compute\_signed\_area(p);

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

int j = (i+1) % p.size();

c = c + (p[i]+p[j])\*(p[i].x\*p[j].y - p[j].x\*p[i].y);

}

return c / scale;

}

//O(n^2)

bool is\_simple(const vector<point> &p) {

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

for (int k = i+1; k < (int)p.size(); k++) {

int j = (i+1) % p.size();

int l = (k+1) % p.size();

if (i == l || j == k) continue;

if (segment\_segment\_intersect(p[i], p[j], p[k], p[l]))

return false;

}

}

return true;

}

//điểm trong tam giác

bool point\_in\_triangle(point a, point b, point c, point cur){

ll s1 = abs(cross(b - a, c - a));

ll s2 = abs(cross(a - cur, b - cur)) + abs(cross(b - cur, c - cur)) + abs(cross(c - cur, a - cur));

return s1 == s2;

}

void sort\_lex\_hull(vector<point> &hull){

int n = hull.size();

//Sort hull by x

int pos = 0;

for(int i = 1; i < n; i++) if(hull[i] < hull[pos]) pos = i;

rotate(hull.begin(), hull.begin() + pos, hull.end());

}

// xác định xem điểm nằm trong hay nằm trên ranh giới của đa giác (O (logn))

// điểm nằm trong đa giác

bool point\_in\_convex\_polygon(vector<point> &hull, point cur){

int n = hull.size();

// Trường hợp góc: trỏ ra ngoài hầu hết các nêm bên trái và bên phải

if(cur.dir(hull[0], hull[1]) != 0 && cur.dir(hull[0], hull[1]) != hull[n - 1].dir(hull[0], hull[1]))

return false;

if(cur.dir(hull[0], hull[n - 1]) != 0 && cur.dir(hull[0], hull[n - 1]) != hull[1].dir(hull[0], hull[n - 1]))

return false;

// Tìm kiếm nhị phân để tìm xem nó nằm giữa

int l = 1, r = n - 1;

while(r - l > 1){

int mid = (l + r)/2;

if(cur.dir(hull[0], hull[mid]) <= 0)l = mid;

else r = mid;

}

return point\_in\_triangle(hull[l], hull[l + 1], hull[0], cur);

}

// xác định xem điểm có nằm trên ranh giới của đa giác hay không (O (N))

bool point\_on\_polygon(vector<point> &p, point q) {

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

if (q.dist2(project\_point\_segment(p[i], p[(i+1)%p.size()], q)) < EPS) return true;

return false;

}

//Shamos - Hoey for test polygon simple in O(nlog(n))

inline bool adj(int a, int b, int n) {return (b == (a + 1)%n or a == (b + 1)%n);}

struct edge{

point ini, fim;

edge(point ini = point(0,0), point fim = point(0,0)) : ini(ini), fim(fim) {}

};

//< here means the edge on the top will be at the begin

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

if (a.ini == b.ini) return direction(a.ini, a.fim, b.fim) < 0;

if (a.ini.x < b.ini.x) return direction(a.ini, a.fim, b.ini) < 0;

return direction(a.ini, b.fim, b.ini) < 0;

}

bool is\_simple\_polygon(const vector<point> &pts){

vector <pair<point, pii>> eve;

vector <pair<edge, int>> edgs;

set <pair<edge, int>> sweep;

int n = (int)pts.size();

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

point l = min(pts[i], pts[(i + 1)%n]);

point r = max(pts[i], pts[(i + 1)%n]);

eve.pb({l, {0, i}});

eve.pb({r, {1, i}});

edgs.pb(make\_pair(edge(l, r), i));

}

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

for(auto e : eve){

if(!e.nd.st){

auto cur = sweep.lower\_bound(edgs[e.nd.nd]);

pair<edge, int> above, below;

if(cur != sweep.end()){

below = \*cur;

if(!adj(below.nd, e.nd.nd, n) and segment\_segment\_intersect(pts[below.nd], pts[(below.nd + 1)%n], pts[e.nd.nd], pts[(e.nd.nd + 1)%n]))

return false;

}

if(cur != sweep.begin()){

above = \*(--cur);

if(!adj(above.nd, e.nd.nd, n) and segment\_segment\_intersect(pts[above.nd], pts[(above.nd + 1)%n], pts[e.nd.nd], pts[(e.nd.nd + 1)%n]))

return false;

}

sweep.insert(edgs[e.nd.nd]);

}

else{

auto below = sweep.upper\_bound(edgs[e.nd.nd]);

auto cur = below, above = --cur;

if(below != sweep.end() and above != sweep.begin()){

--above;

if(!adj(below->nd, above->nd, n) and segment\_segment\_intersect(pts[below->nd], pts[(below->nd + 1)%n], pts[above->nd], pts[(above->nd + 1)%n]))

return false;

}

sweep.erase(cur);

}

}

return true;

}

1. GRAPH
2. DFS(tarjan)

class Graph {

public:

ll n, timeDFS, scc;

vector<vector<ll>> adj;

vector<ll> num, low, tail;

stack<ll> st; //Lưu lại các đỉnh trong thành phần liên thông mạnh

vector<bool> deleted;

Graph(ll \_n) {

n = \_n;

timeDFS = 0;

scc = 0;

adj = vector<vector<ll>> (n + 1);

num = low = tail = vector<ll> (n + 1, 0);

deleted = vector<bool> (n + 1, false);

}

void addEdge(ll u, ll v) {

adj[u].pb(v);

}

void dfs(ll u, vector<vt> &SCC) {

num[u] = low[u] = ++timeDFS;

st.push(u);

for(auto v:adj[u]) {

if(deleted[v]) continue;

if(!num[v]) {

dfs(v, SCC);

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

} else {

low[u] = min(low[u], num[v]);

}

}

if(low[u] == num[u]) {

scc++;

ll v;

vt a;

while(st.top() != u) {

v = st.top();

a.pb(v);

deleted[v] = true;

st.pop();

}

v = st.top();

a.pb(v);

deleted[v] = true;

st.pop();

if(a.size() > 0) SCC.pb(a);

}

}

};

1. BFS // dùng trong bài toán loang, tìm đường đi trong ma trận.

// int moveX[4] = {0, 0, 1, -1};

// int moveY[4] = {1, -1, 0, 0};

// int moveX[8] = {-1, -1, -1, 0, 0, 1, 1, 1};

// int moveY[8] = {-1, 0, 1, -1, 1, -1, 0, 1};

int moveX[8] = {1, 1, 2, 2, -1, -1, -2, -2};

int moveY[8] = {-2, 2, -1, 1, -2, 2, -1, 1};

class Graph {

public:

int n, m;

vector<vector<char>> adj;

vector<vector<bool>> visited;

Graph(int n, int m) {

this -> n = n;

this -> m = m;

adj = vector<vector<char>> (n, vector<char> (m, '0'));

visited = vector<vector<bool>> (n, vector<bool> (m, false));

}

void addEdge() {

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

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

cin >> adj[i][j];

}

}

}

void bfs(int sx, int sy) {

// nc = nf = 0;

queue<pair<int, int>> q;

q.push({sx, sy});

visited[sx][sy] = true;

while(!q.empty()) {

int ux = q.front().x;

int uy = q.front().y;

q.pop();

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

int vx = ux + moveX[i];

int vy = uy + moveY[i];

if(vx > n - 1 || vx < 0) continue;

if(vy > m - 1 || vy < 0) continue;

if(adj[vx][vy] == '0' && !visited[vx][vy]) {

visited[vx][vy] = true;

q.push({vx, vy});

}

}

}

}

};

//dùng trong đảo chiều một số cạnh trong đồ thị

class Graph{

public:

int n;

vector<vector<pair<int, int>>> adj;

vector<int> d;

Graph(int n) {

this -> n = n;

adj = vector<vector<pair<int, int>>> (n + 1);

d = vector<int> (n + 1, 999999999);

}

void addEdge(int u, int v) {

adj[u].pb({0, v});

adj[v].pb({1, u});

}

void bfs(int s) {

deque<int> q;

q.pb(s);

d[s] = 0;

while(!q.empty()) {

int u = q.front();

q.pop\_front();

if(u == n) return;

for(auto edge:adj[u]) {

int v = edge.y;

int w = edge.x;

if(d[v] > d[u] + w) {

d[v] = d[u] + w;

if(w) q.push\_back(v);

else q.push\_front(v);

}

}

}

d[n] = -1;

}

};

1. DIJKSTRA

class Graph{

public:

int n;

vector<vector<p>> adj;

vt trace, dist;

Graph(int n) {

this -> n = n;

adj = vector<vector<p>> (n + 1);

trace.resize(n + 1, -1);

dist.resize(n + 1, INF);

}

void addEdge(ll u, ll v, ll w) {

adj[u].pb({v, w});

}

void dijkstra(int s) {

vector<bool> P(n + 1, false);

dist[s] = 0;

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

int uBest;

ll Max = INF;

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

if(dist[u] < Max && P[u] == false) {

uBest = u;

Max = dist[u];

}

}

int u = uBest;

P[u] = true;

for(auto e:adj[u]) {

ll v = e.x;

ll w = e.y;

if(dist[v] > dist[u] + w) {

dist[v] = dist[u] + w;

trace[v] = u;

}

}

}

}

void dijkstraSparse(int s) {

vector<bool> P(n + 1, false);

dist[s] = 0;

priority\_queue<p, vector<p>, cmp> h;

h.push({s, dist[s]});

while(!h.empty()) {

p tmp = h.top();

h.pop();

ll u = tmp.x;

if(P[u]) continue;

P[u] = true;

for(auto e:adj[u]) {

ll v = e.x;

ll w = e.y;

if(dist[v] > dist[u] + w) {

dist[v] = dist[u] + w;

h.push({v, dist[v]});

trace[v] = u;

}

}

}

}

vt trace\_path(int s, int u) {

if(s != u && trace[u] == -1) return vt(0);

vt path;

while(u != -1) {

path.pb(u);

u = trace[u];

}

reverse(all(trace));

return path;

}

};

1. BELLMAN-FORD

class Graph{

public:

int n;

vector<tp> adj; // danh sach canh

vt trace, dist, negCycle;

Graph(int n) {

this -> n = n;

trace.resize(n + 1, -1);

dist.resize(n + 1, INF);

}

void addEdge(ll u, ll v, ll w) {

adj.pb({u, {v, w}});

}

bool checkNegativeCycle() {

int negStart = -1;

for(auto e:adj) {

ll u = e.fi;

ll v = e.se;

ll w = e.th;

if(dist[u] != INF && dist[v] > dist[u] + w) {

dist[v] = -INF;

trace[v] = u;

negStart = v;

}

}

if(negStart == -1) return false;

// int u = negStart;

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

// u = trace[u];

// }

// negCycle = vt (1, u);

// for(int v = trace[u]; v != u; v = trace[u]) {

// negCycle.pb(v);

// }

// reverse(all(negCycle));

return true;

}

void bellman\_ford(int s) {

dist[s] = 0;

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

for(auto e:adj) {

ll u = e.fi;

ll v = e.se;

ll w = e.th;

if(dist[u] != INF && dist[v] > dist[u] + w) {

dist[v] = dist[u] + w;

trace[v] = u;

}

}

}

}

vt trace\_path (int s, int u) {

if(u != s && trace[u] == -1) return vt(0);

vt path;

while(u != -1) {

path.pb(u);

u = trace[u];

}

reverse(all(path));

return path;

}

};

1. **FLOYD**

class Graph{

int n;

vector<vt> adj, trace, dist;

Graph(int n) {

this -> n = n;

dist = trace = adj = vector<vt> (n + 1, vt(n + 1));

}

void initTrace() {

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

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

cin >> adj[i][j];

trace[i][j] = i;

}

}

}

void floyd() {

initTrace();

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

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

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

if(dist[u][v] > dist[u][k] + dist[k][v]) {

dist[u][v] = dist[u][k] + dist[k][v];

trace[u][v] = trace[k][v];

}

}

}

}

}

vt trace\_path(int u, int v) {

vt path;

while(u != v) {

path.pb(v);

v = trace[u][v];

}

path.pb(u);

reverse(all(path));

return path;

}

};

1. **KRUSKAL**

class DSU {

public:

vector<int> par;

DSU(int n) {

par = vector<int> (n + 1, 0);

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

}

int find(int u) {

if(par[u] == u) return u;

return par[u] = find(par[u]);

}

bool join(int u, int v) {

u = find(u);

v = find(v);

if(u == v) return false;

par[v] = u;

return true;

}

};

bool cmp (tp a, tp b) {

return a.th < b.th;

}

class MST {

public:

int n, totalWeight = 0;

vector<tp> adj;

MST(int n) {

this -> n = n;

adj = vector<tp> (n + 1);

}

void addEdge(int u, int v, int w) {

adj.push\_back({u, {v, w}});

}

void kruskal(vector<tp> &sp) {

sort(all(adj), cmp);

DSU dsu(n);

for(auto e:adj) {

if(!dsu.join(e.fi, e.se)) continue;

sp.pb(e);

totalWeight += e.th;

}

cout << totalWeight;

}

};

1. **PRIM**

class MST {

public:

int n;

vector<vector<pair<int, int>>> adj;

vt dis;

MST(int n) {

this -> n = n;

adj = vector<vector<pair<int, int>>> (n + 1);

dis = vt (n + 1, INF);

}

void addEdge(int u, int v, int w) {

adj[u].pb({v, w});

adj[v].pb({u, w});

}

int prim (int s) {

int ret = 0;

priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> q;

dis[s] = 0;

q.push({0, s});

while(!q.empty()) {

auto top = q.top();

q.pop();

int curDis = top.x;

int u = top.y;

if(curDis != dis[u]) continue;

ret += dis[u]; dis[u] = -INF;

for(auto &e:adj[u]) {

int v = e.x, c = e.y;

if(dis[v] > c) {

dis[v] = c;

q.push({dis[v], v});

}

}

}

return ret;

}

};

1. DATA STRUCT
2. Segment tree bacis  
   ll t[4\*(int)1e5];

ll n, q;

void modify(ll l, ll r, ll value){

for(l += n, r += n; l < r; l >>= 1, r >>= 1){

if(l&1) t[l++] += value;

if(r&1) t[--r] += value;

}

}

ll query(ll pos){

ll res = 0;

for(pos += n; pos > 0; pos >>= 1){

res += t[pos];

}

return res;

}

void push(){

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

t[i<<1] += t[i];

t[i<<1|1] += t[i];

t[i] = 0;

}

}

/\*

void buildTree(){

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

t[i] = t[i<<1] + t[i<<1|1];

}

}

void modify(ll pos, ll value){

for(t[pos += n] = value; pos > 1; pos >>= 1){

t[pos >> 1] = t[pos] + t[pos^1];

}

}

ll query(ll l, ll r){

ll res = 0;

for(l += n, r += n; l < r; l >>= 1, r >>= 1){

if(l&1) res += t[l++];

if(r&1) res += t[--r];

}

return res;

}

\*/

1. Segment tree lazy

#include "bits/stdc++.h"

using namespace std;

// example implementation of sum tree

const int TSIZE = 131072; // always 2^k form && n <= TSIZE

int segtree[TSIZE \* 2], prop[TSIZE \* 2];

int dat[TSIZE]; // data array

void seg\_init(int nod, int l, int r) {

if (l == r) segtree[nod] = dat[l];

else {

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

seg\_init(nod << 1, l, m);

seg\_init(nod << 1 | 1, m + 1, r);

segtree[nod] = segtree[nod << 1] + segtree[nod << 1 | 1];

}

}

void seg\_relax(int nod, int l, int r) {

if (prop[nod] == 0) return;

if (l < r) {

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

segtree[nod << 1] += (m - l + 1) \* prop[nod];

prop[nod << 1] += prop[nod];

segtree[nod << 1 | 1] += (r - m) \* prop[nod];

prop[nod << 1 | 1] += prop[nod];

}

prop[nod] = 0;

}

int seg\_query(int nod, int l, int r, int s, int e) {

if (r < s || e < l) return 0;

if (s <= l && r <= e) return segtree[nod];

seg\_relax(nod, l, r);

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

return seg\_query(nod << 1, l, m, s, e) + seg\_query(nod << 1 | 1, m + 1, r, s, e);

}

void seg\_update(int nod, int l, int r, int s, int e, int val) {

if (r < s || e < l) return;

if (s <= l && r <= e) {

segtree[nod] += (r - l + 1) \* val;

prop[nod] += val;

return;

}

seg\_relax(nod, l, r);

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

seg\_update(nod << 1, l, m, s, e, val);

seg\_update(nod << 1 | 1, m + 1, r, s, e, val);

segtree[nod] = segtree[nod << 1] + segtree[nod << 1 | 1];

}

// usage:

// seg\_update(1, 0, n - 1, qs, qe, val);

// seg\_query(1, 0, n - 1, qs, qe);

1. Nhân ma trận

|  |
| --- |
| struct Matrix { |
|  | vector <vector <type> > data; |
|  |  |
|  | int row() const { return data.size(); } |
|  |  |
|  | int col() const { return data[0].size(); } |
|  |  |
|  | auto & operator [] (int i) { return data[i]; } |
|  |  |
|  | const auto & operator[] (int i) const { return data[i]; } |
|  |  |
|  | Matrix() = default; |
|  |  |
|  | Matrix(int r, int c): data(r, vector <type> (c)) { } |
|  |  |
|  | Matrix(const vector <vector <type> > &d): data(d) { } |
|  |  |
|  | friend ostream & operator << (ostream &out, const Matrix &d) { |
|  | for (auto x : d.data) { |
|  | for (auto y : x) out << y << ' '; |
|  | out << '\n'; |
|  | } |
|  | return out; |
|  | } |
|  |  |
|  | static Matrix identity(long long n) { |
|  | Matrix a = Matrix(n, n); |
|  | while (n--) a[n][n] = 1; |
|  | return a; |
|  | } |
|  |  |
|  | Matrix operator \* (const Matrix &b) { |
|  | Matrix a = \*this; |
|  | assert(a.col() == b.row()); |
|  | Matrix c(a.row(), b.col()); |
|  | for (int i = 0; i < a.row(); ++i) |
|  | for (int j = 0; j < b.col(); ++j) |
|  | for (int k = 0; k < a.col(); ++k){ |
|  | c[i][j] += 1ll \* a[i][k] % mod \* (b[k][j] % mod) % mod; |
|  | c[i][j] %= mod; |
|  | } |
|  | return c; |
|  | } |
|  |  |
|  | Matrix pow(long long exp) { |
|  | assert(row() == col()); |
|  | Matrix base = \*this, ans = identity(row()); |
|  | for (; exp > 0; exp >>= 1, base = base \* base) |
|  | if (exp & 1) ans = ans \* base; |
|  | return ans; |
|  | } |
|  | }; |

1. Fenwick2d

|  |
| --- |
|  |
| class FenwickTree2d{ |
|  | private: |
|  | // Matrix to store the tree |
|  | vector<vector<int>> ft; |
|  |  |
|  | public: |
|  | // Function to get least significant bit |
|  | int LSB(int x){ |
|  | return x & (-x); |
|  | } |
|  |  |
|  | int query(int x, int y){ |
|  | int sum = 0; |
|  | for(int i = x; i > 0; i = i - LSB(i)){ |
|  | for(int j = y; j > 0; j = j - LSB(j)){ |
|  | sum = sum + ft[i][j]; |
|  | } |
|  | } |
|  | return sum; |
|  | } |
|  |  |
|  | int query(int x1, int y1, int x2, int y2){ |
|  | return (query(x2, y2) - query(x1 - 1, y2) - query(x2, y1 - 1) + query(x1 - 1, y1 - 1)); |
|  | } |
|  |  |
|  | void update(int x, int y, int value){ |
|  | // also update matrix[x][y] if needed. |
|  |  |
|  | for(int i = x; i < ft.size(); i = i + LSB(i)){ |
|  | for(int j = y; j < ft[0].size(); j = j + LSB(j)){ |
|  | ft[i][j] += value; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | FenwickTree2d(vector<vector<int>> matrix){ |
|  | int n = matrix.size(); |
|  | // matrix must not be empty. |
|  | int m = matrix[0].size(); |
|  | // Initialize matrix ft |
|  | ft.assign(m + 1, vector<int> (n + 1, 0)); |
|  | for(int i = 0; i < m; ++i){ |
|  | for(int j = 0; j < n; ++j) |
|  | update(i + 1, j + 1, matrix[i][j]); |
|  | } |
|  | } |
|  | }; |
|  |  |

1. Fenwick

|  |
| --- |
| class Fenwick{ |
|  | public: |
|  | ll n; |
|  | vector<ll> fw1, fw2; |
|  | Fenwick(ll n){ |
|  | this->n = n; |
|  | fw1.assign(n + 1, 0); |
|  | fw2.assign(n + 1, 0); |
|  | } |
|  | void updatePoint(vector<ll>& fw, ll u, ll v) { |
|  | ll idx = u; |
|  | while (idx <= n) { |
|  | fw[idx] += v; |
|  | idx += (idx & (-idx)); |
|  | } |
|  | } |
|  |  |
|  | void updateRange(ll l, ll r, ll v) { |
|  | updatePoint(fw1, l, (n - l + 1) \* v); |
|  | updatePoint(fw1, r + 1, -(n - r) \* v); |
|  | updatePoint(fw2, l, v); |
|  | updatePoint(fw2, r + 1, -v); |
|  | } |
|  |  |
|  | ll getSum(vector<ll>& fw, ll u) { |
|  | ll idx = u, ans = 0; |
|  | while (idx > 0) { |
|  | ans += fw[idx]; |
|  | idx -= (idx & (-idx)); |
|  | } |
|  | return ans; |
|  | } |
|  |  |
|  | ll prefixSum(ll u) { |
|  | return getSum(fw1, u) - getSum(fw2, u) \* (n - u); |
|  | } |
|  |  |
|  | ll rangeSum(ll l, ll r) { |
|  | return prefixSum(r) - prefixSum(l - 1); |
|  | } |
|  | void check(){ |
|  | for(auto it:fw1) cout << it << ' '; |
|  | cout << '\n'; |
|  | for(auto it:fw2) cout << it << ' '; |
|  | cout << '\n'; |
|  | } |
|  | }; |