Contenido

[Square Free Numbers 1](#_Toc497716248)

[Kth substring 2](#_Toc497716249)

[DIGIT SUM 2](#_Toc497716250)

[Largest zero submatriz 3](#_Toc497716251)

[Longest Increasing Subsecuence 4](#_Toc497716252)

[Max empty rectangle 4](#_Toc497716253)

[2D Geometry: Minimum bounding circle 5](#_Toc497716254)

[Palindrome Free Numbers 6](#_Toc497716255)

[Sinonimos y antonimos 7](#_Toc497716256)

# Square Free Numbers

#include<bits/stdc++.h>

using namespace std;

long long primesSquared[1000005];

bool B[1000005];

int r;

//recurse(1,1,m) calcula la cantidad de enteros menores o

//iguales que m que tienen un primo cuadrado como factor

//(principio de inclusion-exclusion)

// no hay que preocuparse por muchos calculos porque el

// producto de los 6 primeros primos cuadrados 4\*9\*25\*49\*121\*169 = 901,800,900

long long recurse(int idx, long long soFar,long long k){

long long ret = 0;

if(primesSquared[idx]\*soFar > k)return 0;

ret += (long long)(k/primesSquared[idx]/soFar);

ret += recurse(idx+1,soFar,k);

ret -= recurse(idx+1,soFar\*primesSquared[idx],k);

return ret;

}

long long BS(long long N)

{

long long I=1,F=(long long)1<<32;

long long sol;

while(I<=F)

{

long long piv=(I+F)/2LL;

if(piv-recurse(1,1,piv)>=N)

{

F=piv-1;

if(piv-recurse(1,1,piv)==N)

sol=piv;

}

else

if(piv-recurse(1,1,piv)<N)

I=piv+1;

}

return sol;

}

int main()

{

for(int i=2;i<=1000000;i++)

if(!B[i])

{

primesSquared[++r]=(long long)i\*i;

if((long long)i\*i>1000000LL)

continue;

for(int j=i\*i;j<=1000000;j+=i)

B[j]=1;

}

//si se quiere buscar el nth square free basta con buscar un m

//tal que m-recurse(1,1,m)==n y se puede hacer con busqueda binaria

long long N;

cin >> N;

cout << BS(N) << '\n';

}

# Kth substring

void solve(int I,int F,int len)

{

while(true)

{

int INI=I,FIN=F,sol=I;

// hallar el intervalo a partir de I

//donde el lcp sea >=len

while(INI<=FIN)

{

int piv=(INI+FIN)/2;

if(lcp(I,piv)>=len)

sol=piv,INI=piv+1;

else

FIN=piv-1;

}

// Calcular la cantidad de substring

//del intervalo con longitud >=len

//sum(i,j)cantidad de substrings desde

//sufijo i al j despues de ordenados

int CS=sum(I,sol)-1ll\*(len-1)\*(sol-I+1);

if(CS<K)

{

K-=CS;

I=sol+1;

}

else

if(sol-I+1<K)

{

len++;

K-=1ll\*sol-1ll\*I+1ll;

}

else

{

int pos=I+K-1;

for(int i=SA[pos];i<SA[pos]+len;i++)

cout << s[i];

cout << '\n';

break;

}

}

}

# DIGIT SUM

Cantidad de numeros entre A y B que la suma de sus digitos es S.

llint A, B;

llint pow10[16];

llint memo[16][136];

llint min\_solution = -1;

llint rec( llint prefix, int digits, int sum ) {

if( sum < 0 ) return 0;

llint mini = prefix;

llint maxi = prefix + pow10[digits]-1;

if( mini > B || maxi < A ) return 0;

if( digits == 0 ) {

if( sum > 0 ) return 0;

if( min\_solution == -1 ) min\_solution = prefix;

return 1;

}

bool memoize = (mini >= A && maxi <= B);

if( memoize && memo[digits][sum] != -1 ) return memo[digits][sum];

llint ret = 0;

for( int dig = 0; dig < 10; ++dig )

ret += rec( prefix + dig\*pow10[digits-1], digits-1, sum-dig );

if( memoize ) memo[digits][sum] = ret;

return ret;

}

int main( void ) {

pow10[0] = 1;

for( int i = 1; i <= 15; ++i ) pow10[i] = pow10[i-1] \* 10;

int S;

scanf( "%lld%lld%d", &A, &B, &S );

memset( memo, -1, sizeof memo );

printf( "%lld\n", rec( 0, 15, S ) );

printf( "%lld\n", min\_solution );

return 0;

}

# Largest zero submatriz

#define MAXN 5005

int N,M;

int D[MAXN];

int A[MAXN][MAXN];

int max\_submatr()

{

int h[MAXN], s[MAXN], ptr = 0;

int ret = 0;

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

{

int l=i;

while(ptr>0 && D[i]<h[ptr-1])

{

ret=max(ret,(i-s[ptr-1])\*(h[ptr-1]));

l=s[ptr-1];

ptr--;

}

h[ptr]=D[i];

s[ptr++]=l;

}

while(ptr>0)

{

ret=max(ret,(M-s[ptr-1])\*(h[ptr-1]));

ptr--;

}

return ret;

}

int main()

{

scanf("%d%d",&N,&M);

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

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

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

int sol=0;

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

{

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

if(!A[i][j])

D[j]++;

else

D[j]=0;

sol=max(sol,max\_submatr());

}

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

return 0;

}

# Longest Increasing Subsecuence

(<) --> lower\_bound

(<=) --> upper\_bound

#include <cstdio>

#include <algorithm>

#define RANG 100

using namespace std;

int n, m, up, N[RANG], SOL[RANG], ID[RANG], L[RANG];

void write (int ID) {

if (ID) {

write (L[ID]);

printf ("%d ", N[ID]);

}

}

int main() {

scanf ("%d", &n);

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

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

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

if (SOL[m] <= N[i]) {

SOL[++m] = N[i];

ID[m] = i;

L[i] = ID[m - 1];

}

else {

up = upper\_bound (SOL + 1, SOL + m + 1, N[i]) - SOL;

SOL[up] = N[i];

ID[up] = i;

L[i] = ID[up - 1];

}

}

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

write (ID[m]);

return 0;

}

# Max empty rectangle

Largest rectangle not enclosing points

Description: Given a field of length W by H, and a set of

N points, find one of the largest rectangles

which do not enclose any points

Complexity: O(N^3) where N is the number of points

#define MAXN 100

typedef struct{

double x, y;

} Point;

typedef struct{

double left,right,top,bot;

} Rect;

Rect max\_Rect(Point \*p, int n, double width, double height){

set< double > xset, yset, ys; Rect r;

double old, curr, bot, top, area, best;

set<double>::iterator left, right, y;

xset.insert(0); xset.insert(width);

yset.insert(0); yset.insert(height);

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

xset.insert(p[i].x);

yset.insert(p[i].y);

}

area = -1;

for(left = xset.begin(); left != xset.end(); left++){

for(right = left; right != xset.end(); right++){

ys.clear();

ys.insert(height);

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

if(\*left < p[i].x && p[i].x < \*right) ys.insert(p[i].y);

old = best = 0;

for(y = ys.begin(); y != ys.end(); y++){

if(\*y-old > best){

best = \*y-old;

bot = old;

top = \*y;

}

old = \*y;

}

curr = best\*(\*right-\*left);

if(curr > area){

area = curr;

r.left = \*left;

r.right = \*right;

r.bot = bot;

r.top = top;

}

}

}

return r;

}

int main(){

int i, n; double w, h;

Rect r; Point p[MAXN];

while(scanf(" %lf %lf %d", &w, &h, &n) == 3){

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

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

r = max\_Rect(p, n, w, h);

printf("Largest rectangle: %.2f -> %.2f, %.2f -> %.2f\n",

r.left, r.right, r.bot, r.top);

}

return 0;

}

# 2D Geometry: Minimum bounding circle

Description: Given a set of points, this returns the circle with the minimum area which completely contains all those points

Complexity: O(n^3) worst case, where n is the number of points but on average O(n)

#define EPS 1e-6

#define MAXN 100

#define SQR(x) ((x)\*(x))

typedef struct{

double x, y;

} Point;

typedef struct{

double x, y, r;

} Circle;

int inside(Point p, Circle c){

return SQR(p.x-c.x)+SQR(p.y-c.y) <= SQR(c.r);

}

Circle Circle1(Point p){

Circle c;

c.x = p.x; c.y = p.y; c.r = 0;

return c;

}

Circle Circle2(Point p1, Point p2){

Circle c;

c.x = 0.5\*(p1.x + p2.x);

c.y = 0.5\*(p1.y + p2.y);

c.r = 0.5\*sqrt(SQR(p1.x-p2.x)+SQR(p1.y-p2.y));

return c;

}

Circle Circle3(Point p1, Point p2, Point p3){

Circle res; double a,b,c,d,e,f,g;

a = p2.x - p1.x; b = p2.y - p1.y;

c = p3.x - p1.x; d = p3.y - p1.y;

e = (p2.x + p1.x)\*a + (p2.y + p1.y)\*b;

f = (p3.x + p1.x)\*c + (p3.y + p1.y)\*d;

g = 2.0\*(a\*(p3.y - p2.y) - b\*(p3.x - p2.x));

if (fabs(g) < EPS){

res.x = res.y = res.r = DBL\_MAX;

return res;

}

res.x = (d\*e - b\*f) / g;

res.y = (a\*f - c\*e) / g;

res.r = sqrt(SQR((p1.x-res.x))+SQR((p1.y-res.y)));

return res;

}

Circle min\_circle(Point \*p, int n){

int i, j, k; Point t; Circle c = Circle1(p[0]);

/\* Randomize point array to avoid doctored input - may modify the

limit on the for loop to increase/decrease randomness \*/

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

j = rand() % n;

k = rand() % n;

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

}

for(i = 1; i < n; i++) if(!inside(p[i], c)){ c = Circle1(p[i]);

for(j = 0; j < i; j++) if(!inside(p[j], c)){ c = Circle2(p[i],p[j]);

for(k = 0; k < j; k++) if(!inside(p[k], c)) c = Circle3(p[i],p[j],p[k]);}}

return c;

}

int main(){

Point pts[MAXN]; Circle res; int i, n;

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

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

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

res = min\_circle(pts, n);

printf("(%.3f, %.3f) r = %.3f\n", res.x, res.y, res.r);

}

return 0;

}

# Palindrome Free Numbers

#define l l long long

ll dp [10][10][20][2];

string str ;

// all :

all digits are allowed

ll Calc ( int first , int second , int len , bool a l l )

{

if ( len >= str . length () )

{

return 1;

}

else

{

if ( dp [ first ][ second ][ len ][ all ]==-1)

{

l l help= 0;

if ( all )

{

for ( int n=0; n<=9; n++)

{

if (n!= first && n!= second )

{

help+=Calc ( second , n , len +1, true ) ;

}

}

}

else

{

int limit = str[ len]-’0’ ;

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

{

if (n!= first && n!= second )

{

help+=Calc ( second , n , len +1, true ) ;

}

}

if ( limit != first && limit != second )

{

help+=Calc ( second , limit , len +1, false ) ;

}

}

dp [ first ][ second ][ len ][ all ]= help ;

}

return dp [ first ][ second ][ len ][ all ];

}

}

ll Get ( ll num)

{

if (num<0) return 0;

stringstream ss ;

ss << num;

str = ss.str() ;

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

int first = str[0]-’0’;

ll res =1;

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

{

res+=Calc (n,n ,1 , n!=first);

}

for ( int k=2; k<= str.length () ; k++)

{

for ( int n=1; n<10; n++)

{

res+=Calc (n , n , k , true ) ;

}

}

return res ;

}

int main ()

{

ll a , b ;

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

ll counter = Get (b)-Get (a-1);

printf ( "%lld \n" , counter ) ;

}

# Sinonimos y antonimos

#include<bits/stdc++.h>

using namespace std;

const int MAXN=2e5+5;

int N,M,Q;

map<string,int>MAP;

int S[MAXN],A[MAXN];

void inicializar()

{

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

S[i]=i,A[i]=0;;

}

void join\_set(int seta,int setb)

{

S[setb]=seta;

}

int find\_set(int nod)

{

if(S[nod]==nod)

return S[nod];

return S[nod]=find\_set(S[nod]);

}

bool antonimos(int a,int b)

{

int seta=find\_set(a);

int setb=find\_set(b);

if(A[seta]==setb)

return 1;

return 0;

}

bool sinonimos(int a,int b)

{

int seta=find\_set(a);

int setb=find\_set(b);

if(seta==setb)

return 1;

return 0;

}

void set\_sinonimos(int a,int b)

{

int seta=find\_set(a);

int setb=find\_set(b);

if(!A[seta] && !A[setb])

join\_set(seta,setb);

else

if(A[seta] && A[setb])

join\_set(seta,setb),join\_set(A[seta],A[setb]);

else

if(A[seta])

join\_set(seta,setb);

else

join\_set(setb,seta);

}

void set\_antonimos(int a,int b)

{

int seta=find\_set(a);

int setb=find\_set(b);

if(A[seta] && A[setb])

join\_set(seta,A[setb]),join\_set(A[seta],setb);

else

if(!A[seta] && !A[setb])

A[seta]=setb,A[setb]=seta;

else

if(A[seta])

join\_set(A[seta],setb);

else

join\_set(A[setb],seta);

}

int main()

{

cin.tie(0);

ios\_base::sync\_with\_stdio(0);

cin >> N >> M >> Q;

inicializar();

string s;int c=0;

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

cin >> s,MAP[s]=++c;

string a,b;

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

{

cin >> c >> a >> b;

int ma=MAP[a],mb=MAP[b];

if(c==1)

{

if(antonimos(ma,mb))

{

cout << "NO\n";

continue;

}

else

cout << "YES\n";

set\_sinonimos(ma,mb);

}

else

{

if(sinonimos(ma,mb))

{

cout << "NO\n";

continue;

}

else

cout << "YES\n";

set\_antonimos(ma,mb);

}

}

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

{

cin >> a >> b;

int ma=MAP[a],mb=MAP[b];

if(sinonimos(ma,mb))

cout << "1\n";

else

if(antonimos(ma,mb))

cout << "2\n";

else

cout << "3\n";

}

return 0;

}