Graph

Distancia minima de A a B en matriz

// tipos movimientos que puede hacer en la matriz

**static** **int** *gox*[] = {2, 2,1,-1,-2, -2, -1, 1};

**static** **int** *goy*[] = {1,-1,2, 2, 1, -1 , -2, -2};

**static** **boolean** can(**int** x,**int** y){

**return** x >= 1 && y >= 1 && x <= 8 && y <= 8;

}

**static** **int** minimo(**int** ax, **int** ay, **int** bx, **int** by){

LinkedList<Integer> cola = **new** LinkedList<Integer>();

**boolean** visitado[][] = **new** **boolean**[10][10];

visitado[ax][ay] = **true**;

cola.add(ax);

cola.add(ay);

cola.add(0);

**while**(!cola.isEmpty()){

**int** cx = cola.poll();

**int** cy = cola.poll();

**int** cnivel = cola.poll();

**if**(cx == bx && cy == by)

**return** cnivel;

**for**(**int** i = 0;i < 8;i++){

**int** nexx = cx + *gox*[i];

**int** nexy = cy + *goy*[i];

**if**(*can*(nexx,nexy) && !visitado[nexx][nexy]){

cola.add(nexx);

cola.add(nexy);

cola.add(cnivel + 1);

visitado[nexx][nexy] = **true**;

}

}

}

**return** -1;

}

private static void **insertEdgeMA**(int ini, int end, int w) {

matrizAdy[ini][end] = w;

}

private static void **insertEdgeLA**(int ini, int end, int w) {

if (listaAdy[ini] == null) {

listaAdy[ini] = new Node();

listaAdy[ini].ady = new ArrayList<Edge>();

}

Edge e = new Edge();

e.ini = ini;

e.end = end;

e.weight = w;

listaAdy[ini].ady.add(e);

}

private static void **dfsMA**(int start) {

byte[] estado = new byte[matrizAdy.length];

// 1 preparado

// 2 espera

// 3 procesado

Arrays.fill(estado, (byte) 1);

Deque<Integer> pila = new ArrayDeque<Integer>();

pila.push(start);

estado[start] = 2;

while (!pila.isEmpty()) {

int toProcess = pila.pop();

estado[toProcess] = 3;

// for para poner los vecinos de toProcess en la pila

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

if (matrizAdy[toProcess][i] > 0) {

if (estado[i] == 1) {

pila.push(i);

estado[i] = 2;

}

}

}

// operaciones

}

}

private static void **bfsMA**(int start) {

byte[] estado = new byte[matrizAdy.length];

// 1 preparado

// 2 espera

// 3 procesado

Arrays.fill(estado, (byte) 1);

Deque<Integer> cola = new ArrayDeque<Integer>();

cola.push(start);

estado[start] = 2;

while (!cola.isEmpty()) {

int toProcess = cola.pop();

estado[toProcess] = 3;

// for para poner los vecinos de toProcess en la cola

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

if (matrizAdy[toProcess][i] > 0) {

if (estado[i] == 1) {

cola.addLast(i);

estado[i] = 2;

}

}

}

// aqui se realiza la operacion deseada

// con el nodo actualmente

// visitado

}

}

private static void **dfsLA**(int start) {

byte[] estado = new byte[listaAdy.length];

// 1 preparado

// 2 espera

// 3 procesado

Arrays.fill(estado, (byte) 1);

Deque<Integer> pila = new ArrayDeque<Integer>();

pila.push(start);

estado[start] = 2;

while (!pila.isEmpty()) {

int toProcess = pila.pop();

estado[toProcess] = 3;

for (int i = 0; i < listaAdy[toProcess].ady.size(); i++) {

if (estado[listaAdy[toProcess].ady.get(i).end] == 1) {

pila.push(listaAdy[toProcess].ady.get(i).end);

estado[listaAdy[toProcess].ady.get(i).end] = 2;

}

}

// operaciones

}

}

private static void **bfsLA**(int start) {

byte[] estado = new byte[listaAdy.length];

// 1 preparado

// 2 espera

// 3 procesado

Arrays.fill(estado, (byte) 1);

Deque<Integer> cola = new ArrayDeque<Integer>();

cola.push(start);

estado[start] = 2;

while (!cola.isEmpty()) {

int toProcess = cola.pop();

estado[toProcess] = 3;

for (int i = 0; i < listaAdy[toProcess].ady.size(); i++) {

if (estado[listaAdy[toProcess].ady.get(i).end] == 1) {

cola.addLast(listaAdy[toProcess].ady.get(i).end);

estado[listaAdy[toProcess].ady.get(i).end] = 2;

}

}

// operaciones

}

}

Conjuntos Disjuntos

**public** **class** MisConjustosDisjuntos {

**private** **static** **int**[] *P*;

// private static int[] rank ;

**private** **static** **void** create(**int** x) {

*P*[x] = x;

// rank[x] = 0;

}

**private** **static** **int** find(**int** x) {

// if(x != P[x])

// P[x] = find(P[x]);

**return** *P*[x];

}

**private** **static** **void** merge(**int** x,**int** y){

**int** px = find(x);

**int** py = find(y);

**if**(px>py){

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

**if**(P[i]==py){

P[i]=px;

}

}

}

**else**{

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

**if**(P[i]==px){

P[i]=py;

}

}

}

}

Geometria Computacional

* Cubo:
  + - volumen=lado^3,
    - diagonal=lado\*raiz(3)
* Ortoedro:
  + - volumen=alto\*ancho\*largo ,
    - diagonal=raiz(alto^2+ancho^2+largo^2)
* Prisma:
  + - volumen=areaBase\*altura
* Cono circular:
  + - volumen=(1/3)\*pi\*radio^2\*altura
    - area lateral=pi\*radio\*generatriz
* Esfera:
  + - volumen=(4/3)\*pi\*radio^2
* Octaedro regular:
  + - volumen=(lado^3)/3\*raiz(2)
    - area total=2\*(lado^2)\*raiz(3)
* Tetraedro regular:
  + - volumen=(lado^3)/12\*raiz(2)
    - area total=(lado^2)\*raiz(3)
* Área del cualquier polígono regular
  + - = n \* Apotema^2 \* tan(π/n) n => número de lados
    - = ½ \* n \* Radio^2\* sin(2 × π/n)
    - = ¼ \* n \* Lado^2 / tan(π/n)
* Radio de la circunferencia circunscrita al triangulo
*  s =semiperimetro
* Radio de la circunferencia inscrita al triangulo
*  s =semiperimetro
* Longitud de la mediana
* ta = ½ \* Raiz de (2(b^2+c^2)-a^2)
* Longitud de la altura
* ha = 2/a \* Raiz de (s(s-a)(s-b)(s-c))
* Longitud del angulo bisector
* la = Raiz de (bc[1-(a/(b+c))^2])
* Circunferencia
* longitud del arco =radio\*arcoseno del angulo
* area del sector circular = (1/2)\*longitud del arco \* radio
* area del anillo circular = pi(radio1^2-radio2^2)

Area: **//para calcular el area de un polígono dados sus vértices**

**double** area(){

**double** total = 0.0;

**int** i, j;

**for**(i = 0; i < NumPoints; i++)

{

j = (i+1) % NumPoints;

total += (points[i].x\*points[j].y) –

(points[j].x\*points[i].y);

}

**return** total / 2;}

Teoria de numeros

* the number of digits of an integer *x* in base *b* is log*b*(*x*)*+* 1

GSD

int gcd(int a, int b) { return (b == 0 ? a : gcd(b, a % b)); }

LCM  
int lcm(int a, int b) { return (a \* (b / gcd(a, b))); } // divide before multiply!

public boolean[] **sieve**(int n) { **//TopCoder Es primo si prime[i];**

boolean[] prime=new boolean[n+1];

Arrays.fill(prime,true);

prime[0]=false;

prime[1]=false;

int m=(int) Math.sqrt(n);

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

if (prime[i])

for (int k=i\*i; k<=n; k+=i)

prime[k]=false;

return prime;

}

Si quisiera contar la cantidad de primos menores que n cada vez que ponga a alguien false incremento un contador y luego resto n –contador.

Descomposicion en factores primos:

Primero lleno un arreglo con los primos menores que la Raiz del numero a descomponer mas uno o dos primos por acaso.

private static ArrayList<Long> **factoresPrimos**(Long number) {

ArrayList<Long> fP = new ArrayList<Long>();

try {

for (int i = 0; number > 1; i++) {

while (number % primos.get(i) == 0) {

number = number / primos.get(i));

fP.add(primos.get(i));

}

}

} catch (Exception e) {

fP.add(number);

}

return fP;

}

**ARITMETICA MODULAR**

* (x + y) mod n = ((x mod n) + (y mod n)) mod n
* (x - y) mod n = ((x mod n) - (y mod n) + n) mod n
* xy mod n = (x mod n)(y mod n) mod n

**Fórmula para calcular el número de *Arboles Binarios* que se pueden formar con N nodos:**

2n! / (n! \* n! \* (n+1))

C (N / K) = N! / ((N-K)!\*K!)

P (N / K) = N! / (N-K)!

**Digitos de n!**

int factDigits(int n) {

return (int)(0.5\*Math.log10(2\*n\*Math.PI) + n\*Math.log10(n/Math.E)+1);

}

**EULER TOTIEN FUNCTION**

//EulerTotientFunction(n) es la cantidad de numeros de 1 a n coprimos con n

ll EulerTotientFunction(ll n){

ll ans = n;

for(ll i=2;i\*i<= n;i++){

if(n %i==0)

ans -= ans/i;

while(n%i==0) n/=i;

}

if(n>1) ans -=ans/n;

return ans; }

**EXPONENCIACION MODULAR**

int modulo(long a, long b, long c) {

long x = 1, y = a;

while (b > 0) {

if (b % 2 == 1) {

x = (x \* y) % c;

}

y = (y \* y) % c;

b /= 2;

}

return (int) (x % c);

}

Factorial de n mod m

long long x = 1;  
for (int i = 2; i <= n i++) {  
x = (x\*i)%m;  
}  
cout << x << "\n" ;

if reminder become negative

x = x%m;  
if (x < 0) x += m;

Bit manipulation

* *x* is divisible by 2*k* exactly when *x* & (2*k −* 1) = 0
* the expression *x* | (1 *<< k*) sets the *k*th bit of *x* to one,
* the expression *x*& ~(1 *<< k*) sets the *k*th bit of *x* to zero,
* and the expression *x* ^ (1 *<< k*) inverts the *k*th bit of *x*.
* The formula *x* & (*x −* 1) sets the last one bit of *x* to zero,
* and the formula *x* &*−x* sets all the one bits to zero, except for the last one bit.
* The formula *x* | (*x −* 1) inverts all the bits after the last one bit.

Print the bit representation of an int x

for (int i = 31; i >= 0; i--) {  
if (x&(1<<i) == 1) cout << "1";  
else cout << "0";  
}

Teoria Combinatoria

Algunas sucesiones:

1+2+3+,,,+n = n(n+1)/2

2+4+6+,,,+2n = n(n+1)

1+3+5+,,,+2n-1 = n^2

1^2+2^2+3^2+,,,+n^2+ = n(n+1)(2n+1)/6

1^3+2^3+3^3+,,,+n^3+ = (n(n+1)/2)^2

1^2+3^2+5^2+,,,+2n-1^2 = n(2n-1)(2n+1)/3

1/1\*2+1/2\*3+1/3\*4+,,,+1/n(n+1) = n/(n+1)

1-(1/2^2)+1-(1/3^2)+,,,,+1-(1/n^2) = (n+1)/2n