EBook Gratis

APRENDIZAJE math

Free unaffiliated eBook created from
Stack Overflow contributors.

#math

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: math

It is an unofficial and free math ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official math.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con las matemáticas

Observaciones

Esta sección proporciona una descripción general de qué es matemática y por qué un desarrollador puede querer usarla.

También debe mencionar cualquier materia grande dentro de las matemáticas, y vincular a los temas relacionados. Dado que la Documentación para matemáticas es nueva, es posible que deba crear versiones iniciales de los temas relacionados.

Examples

Instalación o configuración

Instrucciones detalladas sobre cómo configurar o instalar matemáticas.

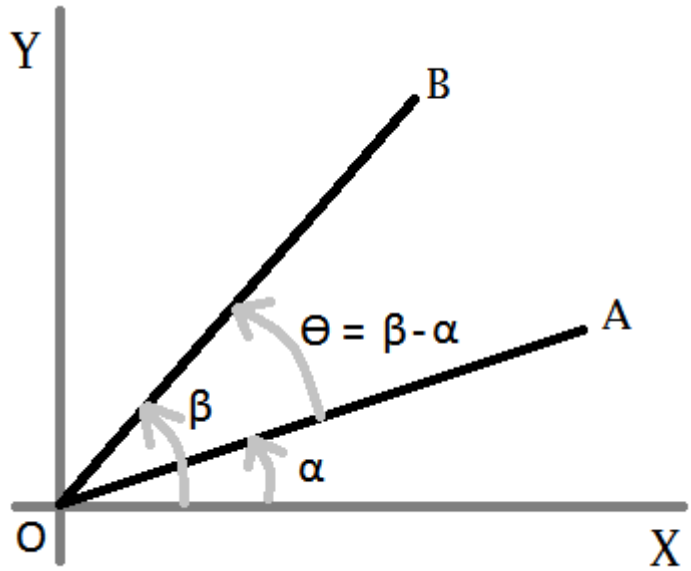
Lea Empezando con las matemáticas en línea:
https://riptutorial.com/es/math/topic/2700/empezando-con-las-matematicas

Capítulo 2: Geometría

Examples

Calcular el ángulo a partir de tres puntos

Primero entendamos el problema, consideremos esta figura



Queremos calcular , donde sabemos *A* , *B* & *O.*

Ahora, si queremos obtener , primero debemos averiguar *α* y *β* . Para cualquier línea recta, sabemos-

```
y = m * x + c
```

Sea *A = (ax, ay)* , *B = (bx, by)* y *O = (buey, oy)* . Así que para la línea *OA* -

```
oy = m1 * ox + c    ⇒ c = oy - m1 * ox    ...(eqn-1)

ay = m1 * ax + c    ⇒ ay = m1 * ax + oy - m1 * ox    [from eqn-1]
                    ⇒ ay = m1 * ax + oy - m1 * ox
                    ⇒ m1 = (ay - oy) / (ax - ox)
                    ⇒ tan α = (ay - oy) / (ax - ox)    [m = slope = tan □]    ...(eqn-2)
```

De la misma manera, para la línea *OB* -

```
tan β = (by - oy) / (bx - ox)    ...(eqn-3)
```

Ahora, necesitamos □ = β - α . En trigonometría tenemos una fórmula

```
tan (β-α) = (tan β + tan α) / (1 - tan β * tan α)    ...(eqn-4)
```

Después de reemplazar el valor de `tan α` (de eqn-2) y `tan b` (de eqn-3) en eqn-4, y aplicar la simplificación obtenemos

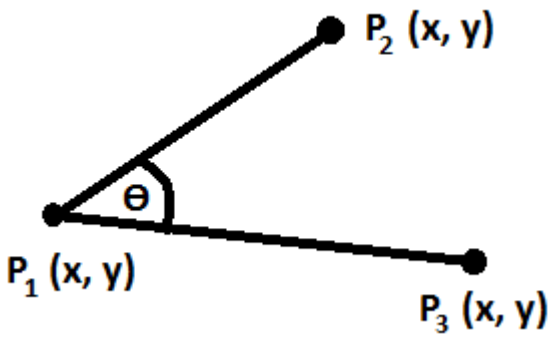
```
tan (β-α) = ( (ax-ox)*(by-oy)+(ay-oy)*(bx-ox) ) / ( (ax-ox)*(bx-ox)-(ay-oy)*(by-oy) )
```

Asi que,

```
 = β-α = tan^(-1) ( ((ax-ox)*(by-oy)+(ay-oy)*(bx-ox)) / ((ax-ox)*(bx-ox)-(ay-oy)*(by-oy)) )
```

¡Eso es!

Ahora, toma la siguiente figura-



Siguiendo C # o, el método Java implementa la teoría anterior

```
    double calculateAngle(double P1X, double P1Y, double P2X, double P2Y,
            double P3X, double P3Y){

        double numerator = P2Y*(P1X-P3X) + P1Y*(P3X-P2X) + P3Y*(P2X-P1X);
        double denominator = (P2X-P1X)*(P1X-P3X) + (P2Y-P1Y)*(P1Y-P3Y);
        double ratio = numerator/denominator;

        double angleRad = Math.Atan(ratio);
        double angleDeg = (angleRad*180)/Math.PI;

        if(angleDeg<0){
            angleDeg = 180+angleDeg;
        }

        return angleDeg;
    }
```

Lea Geometría en línea: https://riptutorial.com/es/math/topic/7653/geometria

Capítulo 3: Número de Fibonacci

Introducción

Los números de Fibonacci son la secuencia entera (OEIS A000045) F (n) que obedece a la siguiente recurrencia:

```
F(n) = F(n-1) + F(n-2)
```

Para F (0) = 0, F (1) = 1, la serie así formada es 0, 1, 1, 2, 3, 5, 8, 13, 21, ...

Los números de Fibonacci aparecen en varias áreas de Matemáticas discretas y Algoritmos.

Examples

Implementación recursiva ingenua

Los números de Fibonacci se usan como un ejemplo muy común para enseñar recursión.

```
fib 0 = 0
fib 1 = 1
fib n = fib (n-1) + fib (n-2)
```

Lea Número de Fibonacci en línea: https://riptutorial.com/es/math/topic/8585/numero-de-fibonacci

Capítulo 4: números primos

Examples

Factorización prima

Ejemplo de implementación de un algoritmo de factorización prima. Un algoritmo de factorización prima encontrará para un número dado `n` una lista de números primos, de modo que si multiplicas esos números primos obtienes `n` . La siguiente implementación agregará `-1` a la lista de factores primos en el caso `n < 0` . Tenga en cuenta que no existe una factorización prima de `0` , por lo que el método a continuación devuelve una lista vacía.

```
List<Integer> primeFactors(int n) {
    List<Integer> factors = new ArrayList<>();
    if (n < 0) {
        factors.add(-1);
        n *= -1;
    }
    for (int i = 2; i <= n / i; ++i) {
        while (n % i == 0) {
            factors.add(i);
            n /= i ;
        }
    }
    if (n > 1) {
        factors.add(n);
    }
    return factors ;
}
```

Primer chequeo

Tenga en cuenta que, por definición, `1` no es un número primo. Para verificar si un número `n` es un número primo, debemos tratar de encontrar un divisor `i` de `n` . Si no podemos, entonces `n` es un primo. Hemos encontrado un divisor cuando `n % i == 0` evalúa como verdadero. Solo necesitamos probar los números impares, ya que solo hay un número primo par, a saber, `2` , que trataremos como un caso especial. Además, solo los números hasta e incluyendo `sqrt(n)` son divisores posibles, porque cuando `n = a * b` entonces al menos uno de los factores es como máximo `sqrt(n)` .

Para verificar si un número es o no un número primo, se puede usar el siguiente algoritmo:

```
boolean isPrime (int n) {
    if (n < 2) {
        return false;
    }
    if (n % 2 == 0) {
        return n == 2;
    }
    for (int i = 3; i*i <= n; i += 2) {
```

```
        if (n % i == 0) {
            return false;
        }
    }
    return true ;
}
```

Primer tamiz

El Tamiz de Eratóstenes genera todos los números primos de 2 a un número dado *n* .

1. Supongamos que todos los números (de 2 a *n*) son primos.
2. Luego toma el primer número primo y elimina todos sus múltiplos.
3. Iterar el paso 2 para la próxima prima. Continuar hasta que todos los números hasta *n* hayan sido marcados.

Pseudocódigo

```
Input: integer n > 1

Let A be an array of Boolean values, indexed by integers 1 to n,
initially all set to true.

for i = 2, 3, 4, ..., not exceeding sqrt(n):
    if A[i] is true:
        for j = 2i, 3i, 4i, 5i, ..., not exceeding n :
            A[j] := false

Output: all i such that A[i] is true.
```

Código C :

```
void PrimeSieve(int n)
{
    int *prime;
    prime = malloc(n * sizeof(int));
        int i;
        for (i = 2; i <= n; i++)
            prime[i] = 1;


    int p;
    for (p = 2; p * p <= n; p++)
    {
        if (prime[p] == 1) // a Prime found
        {
            // mark all multiples of p as not prime.
            for (int i = p * 2; i <= n; i += p)
                prime[i] = 0;
        }
    }

    // print prime numbers
    for (p = 2; p <= n; p++)
        if (prime[p] == 1)
```

```
            printf("%d ", p);
}
```

Lea números primos en línea: https://riptutorial.com/es/math/topic/5558/numeros-primos

Capítulo 5: Sumarios comunes en informática

Examples

La suma de Gauss: 1 + 2 + 3 + ... + n

La suma

    1 + 2 + 3 + ... + n

Simplifica a

    n (n + 1) / 2.

Tenga en cuenta que esta cantidad es $\Theta(n^2)$.

Este atajo surge frecuentemente en el análisis de algoritmos como la ordenación por inserción o la ordenación por selección.

Los números de la forma n (n + 1) / 2 se denominan números triangulares .

Sumas de potencias de dos: 1 + 2 + 4 + 8 + 16 + ...

La suma

    $2^0 + 2^1 + 2^2 + ... + 2^{n-1}$

se simplifica a $2^n - 1$. Esto explica por qué el valor máximo que se puede almacenar en un entero sin signo de 32 bits es $2^{32} - 1$.

Suma de una serie geométrica: r ^ 0 + r ^ 1 + r ^ 2 + ...

La suma de las series geométricas.

    $r^0 + r^1 + r^2 + ... + r^{n-1}$

En el caso donde $r \neq 1$, se simplifica a $(r^n - 1) / (r - 1)$. Si r <1, esta suma está limitada desde arriba por 1 / (1 - r).

Si r = 1, esta suma es rn.

Fencepost Sums

Aquí consideramos sumas de la forma

a + b + a + b + ... a

contra

a + b + a + b + ... b

Para visualizar estas sumas, imagine una sección de cerca que alterne entre postes y rieles. Tres escenarios son posibles.

---

1. Imagina una sección de cerca con postes en cada extremo, conectados por rieles. Los rieles requieren n + 1 postes. A la inversa, los mensajes p están unidos por carriles p-1.

   | - | - | - |

---

2. Imagine una sección de cerca con un poste en un extremo, pero una barra abierta en el otro. Los rieles requieren n postes.

   | - | - | -

   o

   - | - | - |

---

3. Imagina una sección de cerca con rieles abiertos en ambos extremos. Los rieles requieren n-1 postes. Por el contrario, las publicaciones p están unidas por p + 1 carriles.

   - | - | -

Los cálculos como este surgen en situaciones como la disposición de objetos gráficos en los que los tamaños de los objetos deben sumarse y los espacios entre los objetos también deben sumarse. En tales situaciones, es importante tener en cuenta si la intención es tener espacios en cada extremo.

El ancho total de dicha cerca siempre será:

(ancho del poste) x (número de postes) + (ancho del riel) x (número de rieles)

Pero se debe tener cuidado al calcular la cantidad de publicaciones y la cantidad de rieles para evitar el llamado error " *off-by-one"* . Los errores de este tipo son comunes.

Sumas de los números de Fibonacci

Los números de Fibonacci se definen inductivamente como

- $F_0 = 0$
- $F_1 = 1$
- $F_{n+2} = F_n + F_{n+1}$

La suma de los primeros n + 1 números de Fibonacci está dada por

$$F_0 + F_1 + F_2 + ... + F_n = F_{n+2} - 1.$$

Esta suma surge, entre otros lugares, en el análisis de los montones de Fibonacci, donde se utiliza para proporcionar un límite inferior en el número de nodos en cada árbol en el montón.

Sumas de recíprocos: 1/1 + 1/2 + 1/3 + 1/4 + ...

La suma

$$1/1 + 1/2 + 1/3 + 1/4 + ... + 1 / n$$

es igual al nth número armónico , denotado H_n . El nth número armónico obedece a las desigualdades.

$$\ln (n + 1) \leq H_n \leq (\ln n) + 1$$

y por lo tanto $H_n = \Theta (\log n)$. Los números armónicos a menudo surgen en el análisis de algoritmos, con un orden aleatorio aleatorio que es un ejemplo particularmente bueno.

Sumas de cuadrados recíprocos: 1/1 + 1/4 + 1/9 + 1/16 + 1/25 + ...

La suma

$$1/1 + 1/4 + 1/9 + 1/16 + ...$$

hasta el infinito converge a pi $^{2/6,}$ y por lo tanto cualquier suma de la forma

$$1/1 + 1/4 + 1/9 + 1/16 + ... + 1 / n^2$$

es $\Theta (1)$.

Lea Sumarios comunes en informática en línea:
https://riptutorial.com/es/math/topic/3458/sumarios-comunes-en-informatica

Creditos

S. No	Capítulos	Contributors
1	Empezando con las matemáticas	Community
2	Geometría	Minhas Kamal
3	Número de Fibonacci	ABcDexter, Agnishom Chattopadhyay, square1001
4	números primos	ABcDexter, martijnn2008, Teepeemm
5	Sumarios comunes en informática	ABcDexter, cdo256, Everyone_Else, templatetypedef, Wyck