EBook Gratuito

APPRENDIMENTO math

Free unaffiliated eBook created from **Stack Overflow contributors.**

#math

Sommario

Di

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

Capitolo 1: Iniziare con la matematica

Osservazioni

Questa sezione fornisce una panoramica di cosa è la matematica e perché uno sviluppatore potrebbe volerlo utilizzare.

Dovrebbe anche menzionare qualsiasi argomento di grandi dimensioni in matematica e collegarsi agli argomenti correlati. Poiché la documentazione per la matematica è nuova, potrebbe essere necessario creare versioni iniziali di tali argomenti correlati.

Examples

Installazione o configurazione

Istruzioni dettagliate su come installare o installare la matematica.

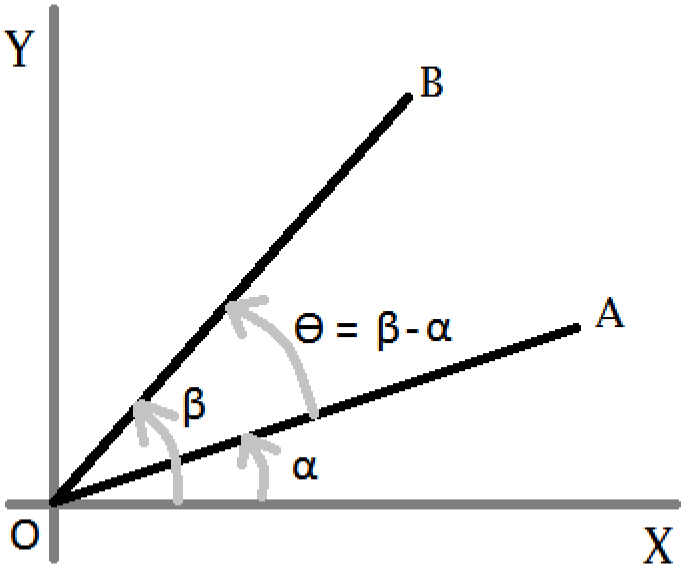
Leggi Iniziare con la matematica online: https://riptutorial.com/it/math/topic/2700/iniziare-con-la-matematica

Capitolo 2: Geometria

Examples

Calcola angolo da tre punti

Prima di tutto, capiamo il problema, considera questa figura



Vogliamo calcolare *Θ* , dove conosciamo *A* , *B* & *O.*

Ora, se vogliamo ottenere *Θ* , dobbiamo prima scoprire *α* e *β* . Per qualsiasi linea retta, sappiamo

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

Sia- *A = (ax, ay)* , *B = (bx, per)* , e *O = (bue, oy)* . Quindi per la linea *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)
```

Allo stesso modo, per *OB di* linea -

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

Ora, abbiamo bisogno di `□ = β - α` . Nella trigonometria abbiamo una formula-

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

Dopo aver sostituito il valore di `tan α` (da eqn-2) e `tan b` (da eqn-3) in eqn-4, e applicando la semplificazione otteniamo-

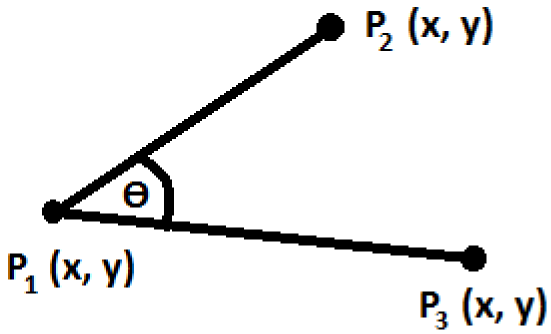
```
tan (β-α) = ( (ax-ox)*(by-oy)+(ay-oy)*(bx-ox) ) / ( (ax-ox)*(bx-ox)-(ay-oy)*(by-oy) )
```

Così,

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

È così!

Ora, prendi la seguente figura:



Seguendo C # o, il metodo Java implementa sopra la teoria-

```
    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;
    }
```

Leggi Geometria online: https://riptutorial.com/it/math/topic/7653/geometria

Capitolo 3: numeri primi

Examples

fattorizzazione in numeri primi

Esempio di implementazione di un algoritmo di fattorizzazione primaria. Un algoritmo di fattorizzazione primo troverà per un dato numero `n` un elenco di numeri primi, in modo tale che se si moltiplicano quei numeri primi si ottiene `n` . L'implementazione seguente aggiungerà `-1` all'elenco dei fattori primi nel caso `n < 0` . Si noti che non esiste una fattorizzazione primaria pari a `0` , quindi il metodo sottostante restituisce una lista vuota.

```
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 ;
}
```

Primo controllo

Si noti che per definizione `1` non è un numero primo. Per verificare se un numero `n` è un numero primo dovremmo cercare di trovare un divisore `i` di `n` . Se non possiamo, allora `n` è un numero primo. Abbiamo trovato un divisore quando `n % i == 0` vero. Abbiamo solo bisogno di provare numeri dispari, dal momento che ce n'è uno solo primo, cioè `2` , che tratteremo come caso speciale. Inoltre, solo i numeri fino a includere `sqrt(n)` sono possibili divisori, perché quando `n = a * b` allora almeno uno dei fattori è al massimo `sqrt(n)` .

Per verificare se un numero è o meno un numero primo è possibile utilizzare il seguente 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 ;
}
```

Setaccio Prime

Il setaccio di Eratostene genera tutti i numeri primi da 2 a un numero dato *n* .

1. Supponiamo che tutti i numeri (da 2 a *n*) siano primi.
2. Quindi prendi il primo numero primo e rimuovi tutti i suoi multipli.
3. Iterate il passaggio 2 per il prossimo primo. Continua fino a quando tutti i numeri fino a *n* sono stati segnati.

pseudocodice:

```
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.
```

Codice 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);
}
```

Leggi numeri primi online: https://riptutorial.com/it/math/topic/5558/numeri-primi

Capitolo 4: Numero di Fibonacci

introduzione

I numeri di Fibonacci sono la sequenza intera (OEIS A000045) F (n) che obbedisce alla seguente ricorrenza:

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

Per F (0) = 0, F (1) = 1, la serie così formata è 0, 1, 1, 2, 3, 5, 8, 13, 21, ...

I numeri di Fibonacci compaiono in diverse aree della Discrete Mathematics and Algorithms.

Examples

Implementazione ricorsiva ingenua

I numeri di Fibonacci sono usati come esempio molto comune per insegnare la ricorsione.

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

Leggi Numero di Fibonacci online: https://riptutorial.com/it/math/topic/8585/numero-di-fibonacci

Capitolo 5: Sommatorie comuni in informatica

Examples

La somma di Gauss: 1 + 2 + 3 + ... + n

La somma

$1 + 2 + 3 + ... + n$

Semplifica a

$n (n + 1) / 2$.

Si noti che questa quantità è $\Theta (n^2)$.

Questo collegamento si verifica frequentemente nell'analisi di algoritmi come l'ordinamento di inserimento o l'ordinamento di selezione.

I numeri della forma $n (n + 1) / 2$ sono chiamati numeri triangolari .

Somma di poteri di due: 1 + 2 + 4 + 8 + 16 + ...

La somma

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

semplifica fino a $2^n - 1$. Questo spiega perché il valore massimo che può essere memorizzato in un numero intero a 32 bit senza segno è $2^{32} - 1$.

Somma di una serie geometrica: r ^ 0 + r ^ 1 + r ^ 2 + ...

La somma delle serie geometriche

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

Nel caso in cui $r \neq 1$, semplifica a $(r^n - 1) / (r - 1)$. Se $r < 1$, questa somma è limitata da $1 / (1 - r)$.

Se $r = 1$, questa somma è rn.

Fencepost Sums

Qui consideriamo le somme del modulo

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

vs.

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

Per visualizzare queste somme, immagina una sezione di recinzione alternata tra pali e binari. Sono possibili tre scenari.

---

1. Immagina una sezione di recinzione con i messaggi alle estremità, collegati da binari. n rail richiedono n + 1 post. Viceversa i p post sono uniti da rails p-1.

   | - | - | - |

---

2. Immagina una sezione di recinzione con un palo a un'estremità, ma a parte un binario aperto. n rail richiedono n post.

   | - | - | -

   o

   - | - | - |

---

3. Immagina una sezione di recinzione con binari aperti alle due estremità. n rail richiedono post n-1. Viceversa, i p post sono uniti da p + 1 binari.

   - | - | -

Calcoli come questo si presentano in situazioni come il layout di oggetti grafici in cui le dimensioni degli oggetti devono essere sommate e gli spazi tra gli oggetti devono essere sommati. In tali situazioni è importante essere consapevoli o meno dell'intenzione di avere spazi a ciascuna estremità.

La larghezza totale di una tale recinzione sarà sempre:

(larghezza del post) x (numero di post) + (larghezza del binario) x (numero di binari)

Ma bisogna essere cauti nel calcolare il numero di post e il numero di binari in modo da evitare un cosiddetto errore " *off-by-one"* . Gli errori di questo tipo sono comuni.

Somma dei numeri di Fibonacci

I numeri di Fibonacci sono definiti induttivamente come

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

La somma dei primi n + 1 numeri di Fibonacci è data da

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

Questa sommatoria si pone, tra l'altro, nell'analisi degli heap di Fibonacci, dove viene utilizzato per fornire un limite inferiore sul numero di nodi in ciascun albero nell'heap.

Somme di reciproci: 1/1 + 1/2 + 1/3 + 1/4 + ...

La somma

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

è uguale all'ennesimo numero armonico , indicato con H_n . L'ennesimo numero armonico obbedisce alle disuguaglianze

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

e quindi $H_n = \Theta(\log n)$. I numeri armonici si presentano spesso nell'analisi degli algoritmi, con il quicksort randomizzato che è un esempio particolarmente piacevole.

Somma dei quadrati reciproci: 1/1 + 1/4 + 1/9 + 1/16 + 1/25 + ...

La somma

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

all'infinito converge a ¸ $^{2/6,}$ e quindi ogni sommatoria della forma

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

è $\Theta(1)$.

Leggi Sommatorie comuni in informatica online: https://riptutorial.com/it/math/topic/3458/sommatorie-comuni-in-informatica

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con la matematica	Community
2	Geometria	Minhas Kamal
3	numeri primi	ABcDexter, martijnn2008, Teepeemm
4	Numero di Fibonacci	ABcDexter, Agnishom Chattopadhyay, square1001
5	Sommatorie comuni in informatica	ABcDexter, cdo256, Everyone_Else, templatetypedef, Wyck