eBook Gratuit

APPRENEZ math

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#math

Table des matières

À propos

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

Chapitre 1: Commencer avec les maths

Remarques

Cette section fournit une vue d'ensemble de ce qu'est le calcul et de la raison pour laquelle un développeur peut vouloir l'utiliser.

Il devrait également mentionner tous les grands sujets en mathématiques, et établir un lien avec les sujets connexes. La documentation pour les mathématiques étant nouvelle, vous devrez peut-être créer des versions initiales de ces rubriques connexes.

Examples

Installation ou configuration

Instructions détaillées sur la configuration ou l'installation des mathématiques.

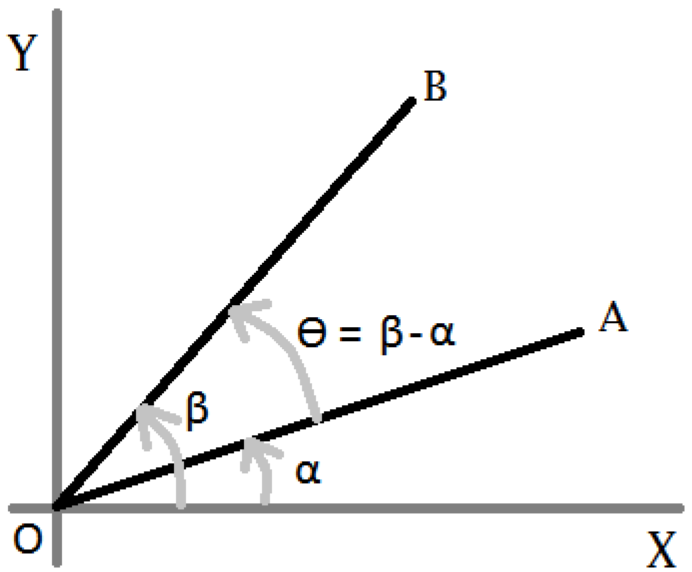
Lire Commencer avec les maths en ligne: https://riptutorial.com/fr/math/topic/2700/commencer-avec-les-maths

Chapitre 2: Géométrie

Examples

Calculer l'angle à partir de trois points

Comprenons d'abord le problème, considérez cette figure.



Nous voulons calculer *Θ,* où nous savons *A, B* & *O.*

Maintenant, si nous voulons obtenir , nous devons d'abord trouver *α* et *β* . Pour toute ligne droite, nous savons-

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

Let- *A = (ax, ay)* , *B = (bx, par)* , et *O = (ox, oy)* . Donc pour la ligne *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 même, pour la ligne *OB* -

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

Maintenant, nous avons besoin de `▯ = β - α` . En trigonométrie, nous avons une formule-

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

Après avoir remplacé la valeur de `tan α` (par eqn-2) et `tan b` (par eqn-3) dans eqn-4 et appliqué la simplification, nous obtenons-

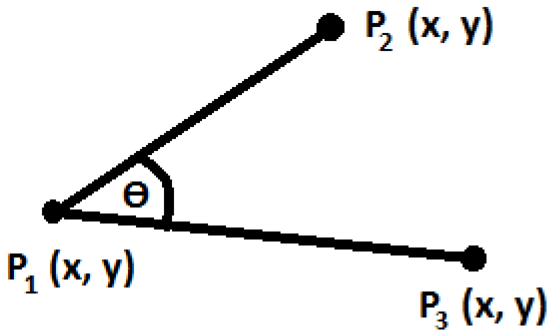
```
tan (β-α) = ( (ax-ox)*(by-oy)+(ay-oy)*(bx-ox) ) / ( (ax-ox)*(bx-ox)-(ay-oy)*(by-oy) )
```

Alors,

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

C'est ça!

Maintenant, prenez la figure suivante:



En suivant C # ou, la méthode Java implémente ci-dessus

```
    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;
    }
```

Lire Géométrie en ligne: https://riptutorial.com/fr/math/topic/7653/geometrie

Chapitre 3: nombres premiers

Examples

Factorisation prime

Exemple d'implémentation d'un algorithme de factorisation premier. Un algorithme de factorisation prime trouvera pour un nombre donné `n` une liste de nombres premiers, de sorte que si vous multipliez ces nombres premiers, vous obtenez `n` . L'implémentation ci-dessous ajoutera `-1` à la liste des facteurs premiers dans le cas `n < 0` . Notez qu'il n'existe pas de factorisation de `0` , donc la méthode ci-dessous renvoie une liste vide.

```
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 ;
}
```

Vérification de prime

Veuillez noter que par définition `1` n'est pas un nombre premier. Pour vérifier si un nombre `n` est un nombre premier, nous devrions essayer de trouver un diviseur `i` de `n` . Si nous ne le pouvons pas, alors `n` est un nombre premier. Nous avons trouvé un diviseur lorsque `n % i == 0` évalué comme vrai. Nous avons seulement besoin d'essayer des nombres impairs, car il n'y a qu'un seul nombre premier, à savoir `2` , que nous traiterons comme un cas particulier. De plus, seuls les nombres allant jusqu'à et incluant `sqrt(n)` sont des diviseurs possibles, car lorsque `n = a * b` au moins un des facteurs est au plus `sqrt(n)` .

Pour vérifier si un nombre est un nombre premier, l'algorithme suivant peut être utilisé:

```
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 ;
}
```

Tamis premier

Le crible d'Eratosthène génère tous les nombres premiers de 2 à un nombre donné *n* .

1. Supposons que tous les nombres (de 2 à *n*) sont premiers.
2. Prenez ensuite le premier nombre premier et supprimez tous ses multiples.
3. Parcourez l'étape 2 pour le premier prime. Continuez jusqu'à ce que tous les chiffres jusqu'à *n* aient été marqués.

Pseudocode:

```
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.
```

Code 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);
}
```

Lire nombres premiers en ligne: https://riptutorial.com/fr/math/topic/5558/nombres-premiers

Chapitre 4: Numéro de Fibonacci

Introduction

Fibonacci Numbers est la séquence entière (OEIS A000045) F (n) qui obéit à la récurrence suivante:

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

Pour F (0) = 0, F (1) = 1, la série ainsi formée est 0, 1, 1, 2, 3, 5, 8, 13, 21, ...

Les nombres de Fibonacci apparaissent dans plusieurs domaines des mathématiques discrètes et des algorithmes.

Examples

Implémentation récursive naïve

Les nombres de Fibonacci sont utilisés comme exemple très courant pour enseigner la récursivité.

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

Lire Numéro de Fibonacci en ligne: https://riptutorial.com/fr/math/topic/8585/numero-de-fibonacci

Chapitre 5: Sommations communes en informatique

Examples

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

La somme

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

Simplifie à

$$n (n + 1) / 2.$$

Notez que cette quantité est $\Theta (n^2)$.

Ce raccourci apparaît fréquemment dans l' analyse d'algorithmes tels que le tri par insertion ou le tri par sélection.

Les nombres de la forme n (n + 1) / 2 sont appelés les nombres triangulaires .

Sommes de Pouvoirs de Deux: 1 + 2 + 4 + 8 + 16 + ...

La somme

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

simplifie à $2^n - 1$. Ceci explique pourquoi la valeur maximale pouvant être stockée dans un entier non signé de 32 bits est $2^{32} - 1$.

Somme d'une série géométrique: r ^ 0 + r ^ 1 + r ^ 2 + ...

La somme des séries géométriques

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

Dans le cas où $r \neq 1$, simplifie à $(r^n - 1) / (r - 1)$. Si $r <1$, cette somme est limitée par le haut par $1 / (1 - r)$.

Si r = 1, cette somme est rn.

Fencepost Sums

Nous considérons ici les sommes de la forme a + b + a + b + ... a

contre.

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

Pour visualiser ces sommes, imaginez une section de clôture alternant poteaux et rails. Trois scénarios sont possibles.

---

1. Imaginez une section de clôture avec des poteaux à chaque extrémité, reliés par des rails. n rails nécessitent n + 1 postes. Inversement, les messages p sont reliés par des rails p-1.

   | - | - | - |

---

2. Imaginez une section de clôture avec un poteau à une extrémité, mais un rail ouvert à l'autre. n rails nécessitent n postes.

   | - | - | -

   ou

   - | - | - |

---

3. Imaginez une section de clôture avec des rails ouverts aux deux extrémités. n rails nécessitent n-1 postes. Inversement, p postes sont rejoints par p + 1 rails.

   - | - | -

De tels calculs se produisent dans des situations telles que la disposition d'objets graphiques où les tailles des objets doivent être additionnées et les espaces entre les objets doivent également être additionnés. Dans de telles situations, il est important de savoir si l'intention est d'avoir des espaces à chaque extrémité.

La largeur totale d'une telle clôture sera toujours:

(largeur du poteau) x (nombre de poteaux) + (largeur du rail) x (nombre de rails)

Cependant, il faut faire preuve de prudence dans le calcul du nombre de messages et du nombre de rails afin d’éviter une erreur dite « *hors-la-une*» . Des erreurs de ce genre sont courantes.

Chiffres de Fibonacci

Les nombres de Fibonacci sont définis par induction comme

- $F_{0} = 0$
- $F_{1} = 1$
- $F_{n+2} = F_{n} + F_{n+1}$

La somme des n + 1 premiers nombres de Fibonacci est donnée par

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

Cette sommation se situe, entre autres, dans l'analyse des tas de Fibonacci, où elle est utilisée pour fournir une limite inférieure au nombre de nœuds dans chaque arbre du tas.

Sommes des réciproques: 1/1 + 1/2 + 1/3 + 1/4 + ...

La sommation

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

est égal au nième nombre harmonique , noté H_n . Le nième nombre harmonique obéit aux inégalités

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

et donc $H_n = \Theta (\log n)$. Les nombres harmoniques apparaissent souvent dans l’analyse des algorithmes, le tri rapide aléatoire étant un exemple particulièrement intéressant.

Sommes de carrés réciproques: 1/1 + 1/4 + 1/9 + 1/16 + 1/25 + ...

La sommation

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

à l' infini converge vers tc $^{2/6,}$ et par conséquent , toute sommation de la forme

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

est Θ (1).

Lire Sommations communes en informatique en ligne: https://riptutorial.com/fr/math/topic/3458/sommations-communes-en-informatique

Crédits

S. No	Chapitres	Contributeurs
1	Commencer avec les maths	Community
2	Géométrie	Minhas Kamal
3	nombres premiers	ABcDexter, martijnn2008, Teepeemm
4	Numéro de Fibonacci	ABcDexter, Agnishom Chattopadhyay, square1001
5	Sommations communes en informatique	ABcDexter, cdo256, Everyone_Else, templatetypedef, Wyck