LEARNING math

#math

Table of Contents

About

Chapter 1: Getting started with math

Remarks

This section provides an overview of what math is, and why a developer might want to use it.

It should also mention any large subjects within math, and link out to the related topics. Since the Documentation for math is new, you may need to create initial versions of those related topics.

Examples

Installation or Setup

Detailed instructions on getting math set up or installed.

Read Getting started with math online: https://riptutorial.com/math/topic/2700/getting-started-with-math

Chapter 2: Common summations in computer science

Examples

Gauss's sum: 1 + 2 + 3 + ... + n

The sum

$$1 + 2 + 3 + \dots + n$$

Simplifies to

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

Notice that this quantity is $\Theta(n^2)$.

This shortcut arises frequently in the analysis of algorithms like insertion sort or selection sort.

Numbers of the form $n(n+1)/2$ are called the triangular numbers.

Sums of Powers of Two: 1 + 2 + 4 + 8 + 16 + ...

The sum

$$2^0 + 2^1 + 2^2 + \dots + 2^{n-1}$$

simplifies to $2^n - 1$. This explains why the maximum value that can be stored in an unsigned 32-bit integer is $2^{32} - 1$.

Sum of a geometric series: r^0 + r^1 + r^2 + ...

The sum of the geometric series

$$r^0 + r^1 + r^2 + \dots + r^{n-1}$$

In the case where $r \neq 1$, simplifies to $(r^n - 1) / (r - 1)$. If $r < 1$, this sum is bounded from above by $1 / (1 - r)$.

If $r = 1$, this sum is rn.

Fencepost Sums

Here we consider sums of the form

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

vs.

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

To visualize these sums, imagine a section of fence alternating between posts and rails. Three scenarios are possible.

---

1. Imagine a section of fence with posts at each end, connected by rails. n rails require n+1 posts. Conversely p posts are joined by p-1 rails.

   |—|—|—|

---

2. Imagine a section of fence with a post at one end, but but an open rail at the other. n rails require n posts.

   |—|—|—

   or

   —|—|—|

---

3. Imagine a section of fence with open rails at both ends. n rails require n-1 posts. Conversely, p posts are joined by p+1 rails.

   —|—|—

Calculations like this arise in situations such as the layout of graphical objects where the sizes of the objects must be summed and the spaces between the objects must also be summed. In such situations it is important to be aware of whether or not the intent is to have spaces at each end.

The total width of such a fence will always be:

(width of post) x (number of posts) + (width of rail) x (number of rails)

But caution must be used in computing the number of posts and number of rails so as to avoid a so-called *off-by-one* error. Mistakes of this kind are common.

Sums of Fibonacci Numbers

The Fibonacci numbers are defined inductively as

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

The sum of the first n+1 Fibonacci numbers is given by

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

This summation arises, among other places, in the analysis of Fibonacci heaps, where it's used to provide a lower bound on the number of nodes in each tree in the heap.

Sums of reciprocals: 1/1 + 1/2 + 1/3 + 1/4 + ...

The summation

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

is equal to the nth harmonic number, denoted H_n. The nth harmonic number obeys the inequalities

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

and therefore $H_n = \Theta(\log n)$. The harmonic numbers often arise in the analysis of algorithms, with randomized quicksort being a particularly nice example.

Sums of reciprocal squares: 1/1 + 1/4 + 1/9 + 1/16 + 1/25 + ...

The summation

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

out to infinity converges to $\pi^2 / 6$, and therefore any summation of the form

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

is $\Theta(1)$.

Read Common summations in computer science online: https://riptutorial.com/math/topic/3458/common-summations-in-computer-science

Chapter 3: Fibonacci Number

Introduction

Fibonacci Numbers is the integer sequence (OEIS A000045) F(n) that obeys the following recurrence:

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

For F(0) = 0, F(1) = 1, the series thus formed is 0, 1, 1, 2, 3, 5, 8, 13, 21, ...

Fibonacci numbers appear in several areas of Discrete Mathematics and Algorithms.

Examples

Naive Recursive Implementation

Fibonacci numbers are used as a very common example for teaching recursion.

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

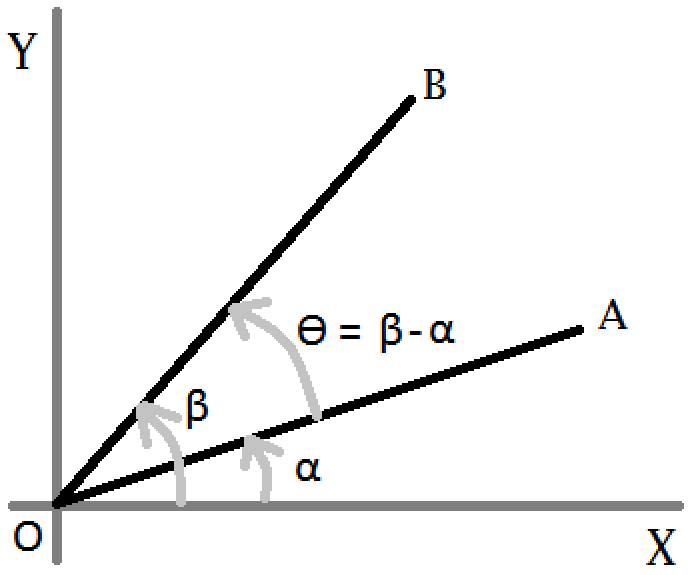
Read Fibonacci Number online: https://riptutorial.com/math/topic/8585/fibonacci-number

Chapter 4: Geometry

Examples

Calculate Angle from Three Points

Lets first understand the problem, consider this figure-



We want to calculate , where we know *A*, *B* & *O*.

Now, if we want to get , we need to find out *α* and *β* first. For any straight line, we know-

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

Let- *A = (ax, ay)*, *B = (bx, by)*, and *O = (ox, oy)*. So for the line *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 ϴ]   ...(eqn-2)
```

In the same way, for line *OB*-

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

Now, we need `ϴ = β - α`. In trigonometry we have a formula-

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

After replacing the value of `tan α` (from eqn-2) and `tan b` (from eqn-3) in eqn-4, and applying simplification we get-

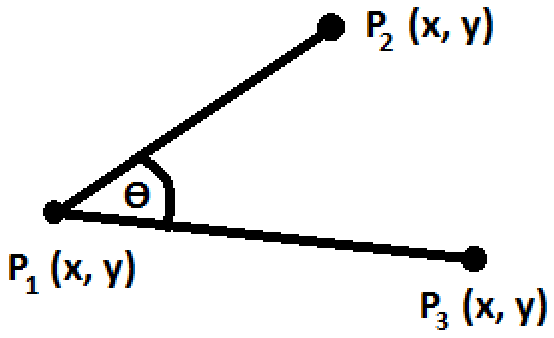
```
tan (β-α) = ( (ax-ox)*(by-oy)+(ay-oy)*(bx-ox) ) / ( (ax-ox)*(bx-ox)-(ay-oy)*(by-oy) )
```

So,

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

That is it!

Now, take the following figure-



Following C# or, Java method implements above theory-

```
    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;
    }
```

Read Geometry online: https://riptutorial.com/math/topic/7653/geometry

Chapter 5: prime numbers

Examples

Prime factorization

Example implementation of a prime factorization algorithm. A prime factorization algorithm will find for a given number `n` a list of primes, such that if you multiply those primes you get `n`. The below implementation will add `-1` to the list of prime factors in case `n < 0`. Note that there exists no prime factorization of `0`, therefore the method below returns an empty list.

```
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 ;
}
```

Prime checking

Please note that by definition `1` is not a prime number. To check if a number `n` is a prime we should try to find a divisor `i` of `n`. If we cannot, then `n` is a prime. We have found a divisor when `n % i == 0` evaluates to true. We only need to try odd numbers, since there's only one even prime, namely `2`, which we'll treat as a special case. Additionally, only the numbers up to and including `sqrt(n)` are possible divisors, because when `n = a * b` then at least one of the factors is at most `sqrt(n)`.

To check whether or not a number is a prime number the following algorithm can be used:

```
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 ;
}
```

Prime sieve

The Sieve of Eratosthenes generates all the primes from 2 to a given number *n*.

1. Assume that all numbers (from 2 to *n*) are prime.
2. Then take the first prime number and removes all of its multiples.
3. Iterate the step 2 for next prime. Continue until all numbers till *n* have been marked.

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.
```

C code:

```
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);
}
```

Read prime numbers online: https://riptutorial.com/math/topic/5558/prime-numbers

Credits

S. No	Chapters	Contributors
1	Getting started with math	Community
2	Common summations in computer science	ABcDexter, cdo256, Everyone_Else, templatetypedef, Wyck
3	Fibonacci Number	ABcDexter, Agnishom Chattopadhyay, square1001
4	Geometry	Minhas Kamal
5	prime numbers	ABcDexter, martijnn2008, Teepeemm