Basic Mathematics

Original Article:

- Mathematics for Topcoders (by dimkadimon)

Sieve of Eratosthenes

Start with a boolean array of size n (maximum value of the prime number you want to find). Initialise all the values to true. Set `prime[0]` and `prime[1]` to false. Iterate for each value and if the `prime[i]` is set to true cross out all its multiples. The resultant array will contain the value true for all the prime numbers.

```
vector<bool> sieve(int n){
  vector<bool> prime(n + 1, true);
  prime[0] = false;
  prime[1] = false;
  for(int i = 2; i <= sqrt(n); ++i){
    if(prime[i] == false)
      continue;
    for(int k = i * i; k <= n; k += i)
      prime[k] = false;
  }
  return prime;
}
```

Euclid's algorithm for computing GCD

The worst case for this algorithm is when the two input numbers are close by and for each iteration the resultant remainder is also close to the numbers. A prime candidate for this are two consecutive fibonacci numbers. The value of the fibonacci numbers can be bounded by exponential expression of the index of the number hence the running time of Euclid's algorithm is `O(logn)`.

```
// assume both a and b cannot be 0
int gcd(int a, int b){
  if(b == 0)
    return a;
  return gcd(b, a % b);
}
```

Similarily, LCM can be given by `a*b / GCD(a, b)`.
Also, euclid's algorithm can be used to solve linear Diophantine equations. These equations have integer coefficients and are of the form:-
`ax + by = c`

Extended Euclid's Theorem

Given two numbers a > 0 and b >= 0. There always exist two integers x and y such that `ax + by = GCD(a,`

`b)` and these numbers can be found in the same time it takes to compute the value of the GCD(a,b).

For a given linear diophantine equation, `ax + by = c` if gcd(a, b) does not divide c then the diophantine equation does not have any (integer) solution. [Proof by contradiction]
Otherwise we can calculate the value of x and y using the extended euclid's theorem. (or Aryabhatia's algorithm)

Intersecting Rectangles

To calculate the intersection between 2 rectangles in 2d space. The first rectangle R1 is given by `(x1, y1), (x2, y2)` and the second rectangle R2 is given by `(x3, y3), (x4, y4)`. Each rectangle is defined by its lower left corner and upper right corner. The intersection R3 is given by `(max(x1, x3), max(y1, y3)), (min(x2, x4), min(y2, y4))`. If max(x1, x3) > min(x2, x4) or max(y1, y3) > min(y2, y4) the intersection doesn't exist.
This can be extended to higher dimensional spaces too.