Maths

League of Programmers

ACA, IIT Kanpur

October 22, 2012

Outline

1. Maths
2. Probability
3. Problems

GCD

- gcd(a, b): greatest integer divides both a and b

GCD

- gcd(a, b): greatest integer divides both a and b
- If $b|a$ then $\gcd(a,b) = b$

GCD

- gcd(a, b): greatest integer divides both a and b
- If $b|a$ then $gcd(a,b) = b$
- Otherwise $a = bt+r$ for some t and r

GCD

- gcd(a, b): greatest integer divides both a and b
- If $b|a$ then gcd(a,b) = b
- Otherwise a = bt+r for some t and r
    - gcd(a,b) = gcd(b,r)

GCD

- gcd(a, b): greatest integer divides both a and b
- If $b|a$ then gcd(a,b) = b
- Otherwise a = bt+r for some t and r
  - gcd(a,b) = gcd(b,r)
  - gcd(a,b) = gcd(b,a%b)

GCD

- gcd(a, b): greatest integer divides both a and b
- If $b|a$ then gcd(a,b) = b
- Otherwise a = bt+r for some t and r
    - gcd(a,b) = gcd(b,r)
    - gcd(a,b) = gcd(b,a%b)
- lcm(a,b) = (a*b)/gcd(a,b)

GCD

- gcd(a, b): greatest integer divides both a and b
- If $b|a$ then gcd(a,b) = b
- Otherwise a = bt+r for some t and r
  - gcd(a,b) = gcd(b,r)
  - gcd(a,b) = gcd(b,a%b)
- lcm(a,b) = (a*b)/gcd(a,b)
- Running time: O (log (a + b))

GCD

Recursive Implementation:

```
int gcd(int a, int b) {
  if (b==0)
    return a;
  else
    return gcd(b,a%b);
}
```

GCD

Iterative Implementation:

```
int gcd(int a, int b) {
  while(b) {
    int r = a % b;
    a = b;
    b = r;
  }
  return a;
}
```

Modular Exponentiation

- Compute a^n in O(log n) time

Modular Exponentiation

- Compute a^n in O(log n) time
- $a^n = 1$, if n=0
  $= a$ if n=1
  $= a^{(n/2)^2}$, if n =even
  $= a^{((n-1)/2)^2}$, if n = odd

Modular Exponentiation

Recursive Implementation:

```
double pow(double a, int n) {
  if(n == 0) return 1;
  if(n == 1) return a;
  double t = pow(a, n/2);
  return t * t * pow(a, n%2);
}
```

Modular Exponentiation

Iterative Implementation:

$a = a_0 + a_1 * 2 + a_2 * 2^2 + ... + a_k * 2^k$

```
int result=1,power=a;
while(!n) {
  if(n&1)
    result*=power;
  power*=power;
  n»=1;
}
```

Fibonacci

Fibonacci

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

Fibonacci

- $F_n = F_{n-1} + F_{n-2}$
- $\left(\begin{smallmatrix} F_n \\ F_{n-1} \end{smallmatrix} = \left[\begin{smallmatrix} 1 & 1 \\ 1 & 0 \end{smallmatrix} \right] * \begin{smallmatrix} F_{n-1} \\ F_{n-2} \end{smallmatrix} \right)$

Fibonacci

- $F_n = F_{n-1} + F_{n-2}$
- $\left(\begin{smallmatrix} F_n \\ F_{n-1} \end{smallmatrix} = \left[\begin{smallmatrix} 1 & 1 \\ 1 & 0 \end{smallmatrix} \right] * \begin{smallmatrix} F_{n-1} \\ F_{n-2} \end{smallmatrix} \right)$
- $\left(\begin{smallmatrix} F_n \\ F_{n-1} \end{smallmatrix} = \left[\begin{smallmatrix} 1 & 1 \\ 1 & 0 \end{smallmatrix} \right]^n * \begin{smallmatrix} F_1 \\ F_0 \end{smallmatrix} \right)$

Fibonacci

- $F_n = F_{n-1} + F_{n-2}$
- $\left(\begin{matrix} F_n \\ F_{n-1} \end{matrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} * \begin{matrix} F_{n-1} \\ F_{n-2} \end{matrix} \right)$
- $\left(\begin{matrix} F_n \\ F_{n-1} \end{matrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n * \begin{matrix} F_1 \\ F_0 \end{matrix} \right)$
- Compute the product in O(lg n) time

Fibonacci

- $F_n = F_{n-1} + F_{n-2}$
- $\left(\begin{array}{c} F_n \\ F_{n-1} \end{array} = \left[\begin{smallmatrix}1 & 1\\ 1 & 0\end{smallmatrix}\right] * \begin{array}{c} F_{n-1} \\ F_{n-2} \end{array} \right)$
- $\left(\begin{array}{c} F_n \\ F_{n-1} \end{array} = \left[\begin{smallmatrix}1 & 1\\ 1 & 0\end{smallmatrix}\right]^n * \begin{array}{c} F_1 \\ F_0 \end{array} \right)$
- Compute the product in O(lg n) time
- Can be extended to support any linear recurrence with constant coefficients

Binomial Coefficients

Binomial Coefficients

- $\binom{n}{k}$ = Number of ways to choose k objects out of n distinguishable objects

Binomial Coefficients

- $\binom{n}{k}$ = Number of ways to choose k objects out of n distinguishable objects
- Computing $\binom{n}{k}$

Binomial Coefficients

- $\binom{n}{k}$ = Number of ways to choose k objects out of n distinguishable objects
- Computing $\binom{n}{k}$
  1. Compute using the following formula

$$\binom{n}{k} = \frac{n(n-1)\ldots(n-k+1)}{k!}$$

Binomial Coefficients

- $\binom{n}{k}$ = Number of ways to choose k objects out of n distinguishable objects
- Computing $\binom{n}{k}$
  1. Compute using the following formula

$$\binom{n}{k} = \frac{n(n-1)\ldots(n-k+1)}{k!}$$

  2. Use Pascal's triangle

Binomial Coefficients

- $\binom{n}{k}$ = Number of ways to choose k objects out of n distinguishable objects
- Computing $\binom{n}{k}$
  1. Compute using the following formula
  $$\binom{n}{k} = \frac{n(n-1)\ldots(n-k+1)}{k!}$$
  2. Use Pascal's triangle
- Cases:

Binomial Coefficients

- $\binom{n}{k}$ = Number of ways to choose k objects out of n distinguishable objects
- Computing $\binom{n}{k}$
  1. Compute using the following formula
  $$\binom{n}{k} = \frac{n(n-1)\ldots(n-k+1)}{k!}$$
  2. Use Pascal's triangle
- Cases:
  1. Both n and k are small
     Use either solution

Binomial Coefficients

- $\binom{n}{k}$ = Number of ways to choose k objects out of n distinguishable objects
- Computing $\binom{n}{k}$
  1. Compute using the following formula
     $$\binom{n}{k} = \frac{n(n-1)\ldots(n-k+1)}{k!}$$
  2. Use Pascal's triangle
- Cases:
  1. Both n and k are small
     Use either solution
  2. n is big, but k or n-k is small
     Use Solution 1 (carefully)

Lucas Theorem

Lucas Theorem

- The Lucas' theorem expresses the remainder of division of the binomial coefficient $\binom{m}{n}$ by a prime number p in terms of the base p expansions of the integers m and n.

Lucas Theorem

- The Lucas' theorem expresses the remainder of division of the binomial coefficient $\binom{m}{n}$ by a prime number p in terms of the base p expansions of the integers m and n.
- For non-negative integers m and n and a prime p, the following congruence relation holds:

$$\binom{m}{n} = \prod_{i=0}^{k} \binom{m_i}{n_i} (mod p)$$

where
$m = m_k p^k + m_{k-1} p^{k-1} + \cdots + m_1 p + m_0$
and
$n = n_k p^k + n_{k-1} p^{k-1} + \cdots + n_1 p + n_0$
are base p expansions of m and n respectively.

Problem

Problem 1

Find the number of strings of length "N" made up of only 3 characters - a, b, c such that "a" occurs at least "min_a" times and at most "max_a" times, "b" occurs at least "min_b" times and at most "max_b" times and "c" occurs at least "min_c" times and at most "max_c" times. Note that all permutations of same string count as 1, so "abc" is same as "bac".
`http://www.spoj.pl/problems/DCEPC702/`

Problem

Problem 2

The main idea is to find a geometrical interpretation for the problem in which we should calculate the number of paths of a special type. More precisely, if we have two points A, B on a plane with integer coordinates, then we will operate only with the shortest paths between A and B that pass only through the lines of the integer grid and that can be done only in horizontal or vertical movements with length equal to 1.

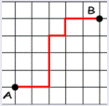

Figure: 1

Problem

Solution

Solution: All paths between A and B have the same length equal to n+m (where n is the difference between x-coordinates and m is the difference between y-coordinates). We can easily calculate the number of all the paths between A and B:

Ans: $\binom{m+n}{n}$ or $\binom{m+n}{m}$

Problem

Problem 3

Let's solve a famous problem using this method. The goal is to find the number of Dyck words with a length of 2n. What is a Dyck word? It's a string consisting only of n X's and n Y's, and matching this criteria: each prefix of this string has more X's than Y's. For example, "XXYY" and "XYXY" are Dyck words, but "XYYX" and "YYXX" are not.

OR

Find the number of ways to arrange n '(' and n ')' brackets such that at each index, the number of '(' is never less than the number of ')'

Problem

Solution

Solution: Total ways: $\binom{2n}{n}$

Incorrect ways: $\binom{2n}{n-1}$

Ans: Catalan number $\frac{1}{n+1}\binom{2n}{n}$

Modular Arithmetic

Modular Arithmetic

- $(x+y) \bmod n = ((x \bmod n) + (y \bmod n)) \bmod n$

Modular Arithmetic

- $(x+y) \bmod n = ((x \bmod n) + (y \bmod n)) \bmod n$
- $(x-y) \bmod n = ((x \bmod n) - (y \bmod n)) \bmod n$

Modular Arithmetic

- (x+y) mod n = ((x mod n) + (y mod n))mod n
- (x-y) mod n = ((x mod n) - (y mod n))mod n
- (x*y) mod n = (x mod n)*(y mod n)mod n

Modular Arithmetic

- (x+y) mod n = ((x mod n) + (y mod n))mod n
- (x-y) mod n = ((x mod n) - (y mod n))mod n
- (x*y) mod n = (x mod n)*(y mod n)mod n
- But, (x/y) mod n = ((x mod n)/(y mod n))mod n, not always true

Modular Arithmetic

- (x+y) mod n = ((x mod n) + (y mod n))mod n
- (x-y) mod n = ((x mod n) - (y mod n))mod n
- (x*y) mod n = (x mod n)*(y mod n)mod n
- But, (x/y) mod n = ((x mod n)/(y mod n))mod n, not always true
- x^y mod n = $(xmodn)^y$ mod n

Multiplicative Inverse

Multiplicative Inverse

- x^{-1} is the inverse of x modulo n if $x * x^{-1} \equiv 1 (mod n)$

Multiplicative Inverse

- x^{-1} is the inverse of x modulo n if $x * x^{-1} \equiv 1 (modn)$
- $5^{-1} \equiv 3 (mod 7)$ because $5 * 3 \equiv 15 \equiv 1 (mod 7)$

Multiplicative Inverse

- x^{-1} is the inverse of x modulo n if $x * x^{-1} \equiv 1(mod n)$
- $5^{-1} \equiv 3(mod 7)$ because $5 * 3 \equiv 15 \equiv 1(mod 7)$
- May not exist (e.g. Inverse of 2 mod 4)

Multiplicative Inverse

- x^{-1} is the inverse of x modulo n if $x * x^{-1} \equiv 1 (mod n)$
- $5^{-1} \equiv 3 (mod 7)$ because $5 * 3 \equiv 15 \equiv 1 (mod 7)$
- May not exist (e.g. Inverse of 2 mod 4)
- Exists iff gcd(x,n) $= 1$

Multiplicative Inverse

- x^{-1} is the inverse of x modulo n if $x * x^{-1} \equiv 1(mod n)$
- $5^{-1} \equiv 3(mod 7)$ because $5 * 3 \equiv 15 \equiv 1(mod 7)$
- May not exist (e.g. Inverse of 2 mod 4)
- Exists iff gcd(x,n) = 1
- gcd(a,b)=ax+by for some integers x, y

Multiplicative Inverse

- x^{-1} is the inverse of x modulo n if $x * x^{-1} \equiv 1(modn)$
- $5^{-1} \equiv 3(mod7)$ because $5 * 3 \equiv 15 \equiv 1(mod7)$
- May not exist (e.g. Inverse of 2 mod 4)
- Exists iff gcd(x,n) = 1
- gcd(a,b)=ax+by for some integers x, y
- If gcd(a,n)=1, then ax + ny = 1 for some x, y

Multiplicative Inverse

- x^{-1} is the inverse of x modulo n if $x * x^{-1} \equiv 1(mod n)$
- $5^{-1} \equiv 3(mod 7)$ because $5 * 3 \equiv 15 \equiv 1(mod 7)$
- May not exist (e.g. Inverse of 2 mod 4)
- Exists iff gcd(x,n) = 1
- gcd(a,b)=ax+by for some integers x, y
- If gcd(a,n)=1, then ax + ny = 1 for some x, y
- Taking modulo n gives $ax \equiv 1(mod n)$

Multiplicative Inverse

- x^{-1} is the inverse of x modulo n if $x * x^{-1} \equiv 1 (mod n)$
- $5^{-1} \equiv 3 (mod 7)$ because $5 * 3 \equiv 15 \equiv 1 (mod 7)$
- May not exist (e.g. Inverse of 2 mod 4)
- Exists iff gcd(x,n) = 1
- gcd(a,b)=ax+by for some integers x, y
- If gcd(a,n)=1, then ax + ny = 1 for some x, y
- Taking modulo n gives $ax \equiv 1 (mod n)$
- Given a,b, Finding x and y, such that ax+by = d is done by **Extended Euclid's algorithm**

Prime Seive

Prime Seive

- Idea: every composite number n has a prime factor $p \leq \sqrt{n}$. So let us assume that all numbers are prime. But if we come across a prime factor of a number, we immediately know that it is not a prime. If there is no prime factor of a number n in the range [2 . . . n-1] then it must be prime.

Prime Seive

Implementation

Generate all primes in range [1..n]

```
For i=1 to n
  prime[i]=1
Prime[1]=0
For i=2 to √n
  if(prime[i])
    for j = i to n/i
      prime[i*j]=0
```

At the end of this step, all numbers i which are prime have prime[i]=1. Others have prime[i]=0.

Prime Number Theorem

Prime Number Theorem

- Prime number Theorem
  Number of primes till n $\sim$ n/logn

Prime Number Theorem

- Prime number Theorem
  Number of primes till n $\sim$ n/logn
- Maximum number of prime factors of n = log n

Euler Torient Function

Euler Torient Function

- $\phi(n)$ = Number of positive integers less than or equal to n which are coprime to n

Euler Torient Function

- $\phi(n)$ = Number of positive integers less than or equal to n which are coprime to n
- $\phi(ab) = \phi(a) * \phi(b)$

Euler Torient Function

- $\phi(n)$ = Number of positive integers less than or equal to n which are coprime to n
- $\phi(ab) = \phi(a) * \phi(b)$
  - For a prime p, $\phi(p) = p - 1$

Euler Torient Function

- $\phi(n)$ = Number of positive integers less than or equal to n which are coprime to n
- $\phi(ab) = \phi(a) * \phi(b)$
  - For a prime p, $\phi(p) = p - 1$
  - For a prime p, $\phi(p^k) = p^k - p^{k-1} = p^k(1 - \frac{1}{p})$

Euler Torient Function

- $\phi(n)$ = Number of positive integers less than or equal to n which are coprime to n
- $\phi(ab) = \phi(a) * \phi(b)$
  - For a prime p, $\phi(p) = p - 1$
  - For a prime p, $\phi(p^k) = p^k - p^{k-1} = p^k(1 - \frac{1}{p})$
- $N = (p_1^{k_1}) * (p_2^{k_2}) * \ldots * (p_t^{k_t})$

Euler Torient Function

- $\phi(n)$ = Number of positive integers less than or equal to n which are coprime to n
- $\phi(ab) = \phi(a) * \phi(b)$
  - For a prime p, $\phi(p) = p - 1$
  - For a prime p, $\phi(p^k) = p^k - p^{k-1} = p^k(1 - \frac{1}{p})$
- $N = (p_1^{k_1}) * (p_2^{k_2}) * \ldots * (p_t^{k_t})$
  - $\phi(n) = p_1^{a_1} p_2^{a_2} .. p_t^{a_t}) = n * (1 - \frac{1}{p_1}) * (1 - \frac{1}{p_2}) * \cdots * (1 - \frac{1}{p_t})$
    $\equiv \phi(n) = p_1^{a_1} p_2^{a_2} .. p_t^{a_t}) = n * \frac{(p_1 - 1) * (p_2 - 1) .. (p_t - 1)}{(p_1 p_2 .. p_t)}$

Euler Torient Function

Seive for $\phi(n)$

Euler Torient Function

Seive for $\phi(n)$

- Run prime sieve once and store result in primes[1..n]

Euler Torient Function

Seive for $\phi(n)$

- Run prime sieve once and store result in primes[1..n]
- ```
  for(i=1 to n) phi[i]=i
  for(i=2 to n)
    if(prime[i])
      for(j=1 to n/i)
        phi[i*j]=phi[i*j]*(i-1)/i
  ```
  ```

Euler Torient Function

Seive for $\phi(n)$

- Run prime sieve once and store result in primes[1..n]
- ```
  for(i=1 to n) phi[i]=i
  for(i=2 to n)
    if(prime[i])
      for(j=1 to n/i)
        phi[i*j]=phi[i*j]*(i-1)/i
  ```
- This algo runs in O(n log log n) time, but we will make improvements to show how you can at times optimize your code.
  ```

Euler Torient Function

Seive for $\phi(n)$

Euler Torient Function

Seive for $\phi(n)$

- Question: Do we really need to generate list of all primes?

Euler Torient Function

Seive for $\phi(n)$

- Question: Do we really need to generate list of all primes?
- Answer : No

Euler Torient Function

Seive for $\phi(n)$

- Question: Do we really need to generate list of all primes?
- Answer : No
-
```
for(i=1 to n) phi[i]=i
for(i=2 to n)
  if(phi[i]==i)
    for(j=i to n;j+=i)
      phi[j] = (phi[j]/i)*(i-1);
```

Fermat's Little Theorem

Fermat's Little Theorem

- If p is a prime number, then for any integer a that is coprime to n, we have $a^p \equiv a (mod p)$

Fermat's Little Theorem

- If p is a prime number, then for any integer a that is coprime to n, we have $a^p \equiv a(modp)$
- This theorem can also be stated as: If p is a prime number and a is coprime to p, then $a^{p-1} \equiv 1(modp)$

Euler's Theorem

Euler's Theorem

- Euler's Theorem is a genaralization for Fermat's little theorem when a and n are co-prime

Euler's Theorem

- Euler's Theorem is a genaralization for Fermat's little theorem when a and n are co-prime
- If $x \equiv y(mod\phi(n))$, then $ax \equiv ay(modn)$.

Euler's Theorem

- Euler's Theorem is a genaralization for Fermat's little theorem when a and n are co-prime
- If $x \equiv y (mod \phi(n))$, then $ax \equiv ay(modn)$.
- $a^{\phi(n)} \equiv 1(modn)$ (actual theorem is a generalization of the above)

Other results

If $n = p_1^{a_1} * p_2^{a_2} * \ldots * p_t^{a_t}$, then

- The number of its positive divisors equals $(a_1 + 1) * (a_2 + 1) * \ldots * (a_t + 1)$

Other results

If $n = p_1^{a_1} * p_2^{a_2} * \ldots * p_t^{a_t}$, then

- The number of its positive divisors equals $(a_1 + 1) * (a_2 + 1) * \ldots * (a_t + 1)$
- Sum of the divisors of n equals $\sum_{d|n} = \prod_{i=1}^{t} \frac{p_i^{m_i+1} - 1}{p_i - 1}$

Miller-Rabin Test (Primality Testing)

Input: n > 1, an odd integer to test for primality.
write n-1 as $2^s d$ by factoring powers of 2 from n-1
repeat for all : a $\in$ [2,n-2]
  If $a^d \neq 1$ mod n and $a^{2^r}.d \neq -1$ mod n for all $r \in [0, s1]$
  then return composite
Return prime

Miller-Rabin Test (Primality Testing)

Input: n $>$ 1, an odd integer to test for primality.
write n-1 as $2^s d$ by factoring powers of 2 from n-1
repeat for all : a $\in$ [2,n-2]
  If $a^d \neq 1$ mod n and $a^{2^r}.d \neq -1$ mod n for all $r \in [0, s1]$
  then return composite
Return prime

Miller-Rabin Test (Primality Testing)

Input: n > 1, an odd integer to test for primality.
write n-1 as $2^s d$ by factoring powers of 2 from n-1
repeat for all : a $\in$ [2,n-2]
  If $a^d \neq 1$ mod n and $a^{2^r}.d \neq -1$ mod n for all $r \in [0, s1]$
  then return composite
Return prime

- if n < 9,080,191, it is enough to test a = 31 and 73;

Miller-Rabin Test (Primality Testing)

Input: n > 1, an odd integer to test for primality.
write n-1 as $2^s d$ by factoring powers of 2 from n-1
repeat for all : a $\in$ [2,n-2]
  If $a^d \neq 1$ mod n and $a^{2^r}.d \neq -1$ mod n for all $r \in [0, s1]$
  then return composite
Return prime

- if n < 9,080,191, it is enough to test a = 31 and 73;
- if n < 4,759,123,141, it is enough to test a = 2, 7, and 61;

Miller-Rabin Test (Primality Testing)

Input: n > 1, an odd integer to test for primality.
write n-1 as $2^s d$ by factoring powers of 2 from n-1
repeat for all : a $\in$ [2,n-2]
If $a^d \neq 1$ mod n and $a^{2^r}.d \neq -1$ mod n for all $r \in [0, s1]$
then return composite
Return prime

- if n < 9,080,191, it is enough to test a = 31 and 73;
- if n < 4,759,123,141, it is enough to test a = 2, 7, and 61;
- if n < 2,152,302,898,747, it is enough to test a = 2, 3, 5, 7, and 11;

Miller-Rabin Test (Primality Testing)

Input: n > 1, an odd integer to test for primality.
write n-1 as $2^s d$ by factoring powers of 2 from n-1
repeat for all : a $\in$ [2,n-2]
  If $a^d \neq 1$ mod n and $a^{2^r}.d \neq -1$ mod n for all $r \in [0, s1]$
  then return composite
Return prime

- if n < 9,080,191, it is enough to test a = 31 and 73;
- if n < 4,759,123,141, it is enough to test a = 2, 7, and 61;
- if n < 2,152,302,898,747, it is enough to test a = 2, 3, 5, 7, and 11;
- if n < 3,474,749,660,383, it is enough to test a = 2, 3, 5, 7, 11, and 13;

Miller-Rabin Test (Primality Testing)

Input: n > 1, an odd integer to test for primality.
write n-1 as $2^s d$ by factoring powers of 2 from n-1
repeat for all : a $\in$ [2,n-2]
  If $a^d \neq 1$ mod n and $a^{2^r}.d \neq -1$ mod n for all $r \in [0, s1]$
  then return composite
Return prime

- if n < 9,080,191, it is enough to test a = 31 and 73;
- if n < 4,759,123,141, it is enough to test a = 2, 7, and 61;
- if n < 2,152,302,898,747, it is enough to test a = 2, 3, 5, 7, and 11;
- if n < 3,474,749,660,383, it is enough to test a = 2, 3, 5, 7, 11, and 13;
- if n < 341,550,071,728,321, it is enough to test a = 2, 3, 5, 7, 11, 13, and 17.

Outline

Mathematical Expectation

Mathematical Expectation

- For a discrete variable X with probability function P(X), the expected value E[X] is given by $\sum_i x_i * P(x_i)$ the summation runs over all the distinct values xi that the variable can take.

Mathematical Expectation

- For a discrete variable X with probability function P(X), the expected value E[X] is given by $\sum_i x_i * P(x_i)$ the summation runs over all the distinct values xi that the variable can take.
- The rule of "linearity of of the expectation" says that E[x1+x2] = E[x1] + E[x2].

Mathematical Expectation

- For a discrete variable X with probability function P(X), the expected value E[X] is given by $\sum_i x_i * P(x_i)$ the summation runs over all the distinct values xi that the variable can take.
- The rule of "linearity of of the expectation" says that E[x1+x2] = E[x1] + E[x2].
- It is important to understand that "expected value" is not same as "most probable value" - rather, it need not even be one of the probable values.

Mathematical Expectation

Example

Mathematical Expectation

Example

- For a six-sided die, the expected number of throws to get each face at least once?

Mathematical Expectation

Example

- For a six-sided die, the expected number of throws to get each face at least once?
- It's $(6/6)+(6/5)+(6/4)+(6/3)+(6/2)+(6/1) = 14.7$.

Mathematical Expectation

Example

- For a six-sided die, the expected number of throws to get each face at least once?
- It's $(6/6)+(6/5)+(6/4)+(6/3)+(6/2)+(6/1) = 14.7$.
- Logic: The chance of rolling a number you haven't yet rolled when you start off is 1, as any number would work. Once you've rolled this number, your chance of rolling a number you haven't yet rolled is 5/6. Continuing in this manner, after you've rolled n different numbers the chance of rolling one you haven't yet rolled is (6-n)/6.

Mathematical Expectation

Example

- For a six-sided die, the expected number of throws to get each face at least once?
- It's $(6/6)+(6/5)+(6/4)+(6/3)+(6/2)+(6/1) = 14.7$.
- Logic: The chance of rolling a number you haven't yet rolled when you start off is 1, as any number would work. Once you've rolled this number, your chance of rolling a number you haven't yet rolled is 5/6. Continuing in this manner, after you've rolled n different numbers the chance of rolling one you haven't yet rolled is (6-n)/6.
- For an n-sided die the expected throws is $(n/n) + (n/(n-1)) + (n/(n-2)) + \ldots + n$.

Other Things to Read

Other Things to Read

- Extended Euclid's algo

Other Things to Read

- Extended Euclid's algo
- Chinese remaindering

Other Things to Read

- Extended Euclid's algo
- Chinese remaindering
- Farey's sequence

Other Things to Read

- Extended Euclid's algo
- Chinese remaindering
- Farey's sequence
- Optimised Sieve, Sieve of Atkins

Other Things to Read

- Extended Euclid's algo
- Chinese remaindering
- Farey's sequence
- Optimised Sieve, Sieve of Atkins
- How to solve Diophantine Equation

Other Things to Read

- Extended Euclid's algo
- Chinese remaindering
- Farey's sequence
- Optimised Sieve, Sieve of Atkins
- How to solve Diophantine Equation
- Pollard Rho factorization

Other Things to Read

- Extended Euclid's algo
- Chinese remaindering
- Farey's sequence
- Optimised Sieve, Sieve of Atkins
- How to solve Diophantine Equation
- Pollard Rho factorization
- Stirling numbers

Other Things to Read

- Extended Euclid's algo
- Chinese remaindering
- Farey's sequence
- Optimised Sieve, Sieve of Atkins
- How to solve Diophantine Equation
- Pollard Rho factorization
- Stirling numbers
- Inclusion-exclusion

Other Things to Read

- Extended Euclid's algo
- Chinese remaindering
- Farey's sequence
- Optimised Sieve, Sieve of Atkins
- How to solve Diophantine Equation
- Pollard Rho factorization
- Stirling numbers
- Inclusion-exclusion
- Gaussean Elimination (Find the determinant of a matrix)

Other Things to Read

- Extended Euclid's algo
- Chinese remaindering
- Farey's sequence
- Optimised Sieve, Sieve of Atkins
- How to solve Diophantine Equation
- Pollard Rho factorization
- Stirling numbers
- Inclusion-exclusion
- Gaussean Elimination (Find the determinant of a matrix)
- Group Theory

Outline

Problems

Links:

1. http://www.spoj.pl/problems/MAIN111/
2. http://www.spoj.pl/problems/NDIVPHI/
3. http://www.spoj.pl/problems/CUBEFR/
4. http://www.spoj.pl/problems/NOSQ/
5. http://www.spoj.pl/problems/UCI2009B/
6. http://www.spoj.pl/problems/SEQ6/
7. http://www.spoj.pl/problems/HAMSTER1/
8. http://www.spoj.pl/problems/MAIN74/
9. http://www.spoj.pl/problems/TUTMRBL/
10. http://www.spoj.pl/problems/FACT0/
11. http://www.spoj.pl/problems/GCD3/
12. http://www.spoj.pl/problems/CRYPTON/
13. http://www.spoj.pl/problems/MAIN12B/
14. http://www.spoj.pl/problems/PLYGRND/

Problems

Added on the contest on VOC `http://ahmed-aly.com/voc/`
Contest ID: 2633
Name: ACA, IITK LOP 04
Author: pnkjjindal