NUMBER THEORY SHORTLIST

Fahim Ahmed

Military Institute of Science and Technology

Dept. of Computer Science and Engineering

Created only for personal use

NOT FOR SALE

Some codes of my own:

1 ALL DIVISORS, NOD of a range, SOD of a range:

```
int mark[100002];
vector<int> divisors[100002];
li Number_of_divisors[100002]; ///NOD
li Sum_of_divisors[100002]; ///SOD
void all_divisors(int n)
{
    int I,j,k;
    int sq = sqrt(n);
    for(I = 1; I <= sq; i++)
    {
        for(j = I; j <= n; j+=i)
        {
            divisors[j].push_back(i);
            Number_of_divisors[j]++;
            Sum_of_divisors[j]+=I;

            k = j/I;
            if(k > sq)
            {
                divisors[j].push_back(k);
                Number_of_divisors[j]++;
                Sum_of_divisors[j]+=k;
            }
        }
    }
}
```

2. INVERSE MODULAS

```
li inv_modulo(li a, li b)
{
        li b0 = b, t, q;
        li x0 = 0, x1 = 1;
        if (b == 1) return 1;
        while (a > 1) {
                q = a / b;
                t = b, b = a % b, a = t;
                t = x0, x0 = x1 - q * x0, x1 = t;
```

```
        }
        if (x1 < 0) x1 += b0;
        return x1;
}
```

3. MODULUSED FACTORIAL

```
li facs[1000001];
void factorial()
{
    li n = 1000000;
    facs[1] = 1;
    facs[0] = 0;
    li fac = 1;
    for(li i = 2; i <= n; i++)
    {
        fac = (fac*i) % MOD;
        facs[i] = fac;
    }
}
```

4. /*METHOD TO FIND THE LAST DIGIT of any N^K: For every N^(4*m+1); the last digit of N will be kept as the N^(4*m+1)'s last digit*/

```
int t,n,k,lastdig,cur;
int main()
{
    rfile("in.txt");
    freopen("out.txt","w",stdout);
    ri(t);
    For(tc,1,t+1)
    {
        ri(n); ri(k);
        if(k == 0)
        {
            printf("1\n");
            continue;
        }
        lastdig = n%10;cur = lastdig;
        int m =(int) (k-1)/4;
        m = 4*m + 1;
```

```
        int trns;
        if(m != 1) trns = (k%m);
        else trns = k-1;
        for(int i = 1; i <= trns; i++)
        {
            cur = cur*lastdig;
            cur%=10;
        }
        printf("%d\n", cur);
    }

}
```

5. REMAINDER OF ANY BIGINTEGER

```
    /*Testing Divisibility of BIGINTEGERS USING INTEGER */
    #include <bits/stdc++.h>
    using namespace std;
    #define li long long int

    int main()
    {
        string integer;
        int T;
        li mod;
        scanf("%d ", &T);
        for(int t = 1; t <= T; t++)
        {
            cin>>integer; scanf("%lld", &mod);
            li res = 0;
            if(integer[0] >= '0' && integer[0] <= '9')
            {
                for(int i = 0; i < integer.size(); i++)
                {
                    res = (res*10 + (integer[i] -  48)) % mod;
                }
            }
            else{
                for(int i = 1; i < integer.size(); i++)
                {
```

```
                res = (res*10 + (integer[i] -  48)) % mod;
            }
        }
        if(res == 0)
        {
            printf("Case %d: divisible\n", t);
        }
        else{
            printf("Case %d: not divisible\n", t);
        }
    }
    return 0;
}
```

6. FAST FIBONACCI:

```
#include <iostream>
#include <unordered_map>
#define li long long int
using namespace std;

const li P = 1000000007;
/* Fast 4ibonacci using cache */
li fib (li n)
{
    static std::unordered_map<li,li> cache;

    if (cache.find(n) == cache.end())
    {
        li f;
        if (n==0)
            f = 0;
        else if (n < 3)
            f = 1;
        else if (n % 2 == 0)
        {
            li k = n/2;
            f = (fib(k) * (2*fib(k+1) – fib(k))) % P;
        }
        else
```

```
            {
                li k = (n-1)/2;
                f = (fib(k+1)*fib(k+1)+ fib(k) * fib(k)) % P;
            }
            if (f<0)
                f += P;

            cache[n] = f;
        }
        return cache.at(n);
    }

    int main ()
    {
        li i ;
        cin >> i;
        cout << i << " : " << fib(i) << endl;
    return 0;
    }
```

7. BITWISE SIV()

```
    int primes[10000000];
    int pr_ctr = 0;
    bitset<10000010> st(0);



    void siv()
    {
        int n = 10000000;
        st.set(0,1);
        st.set(1,1);
        primes[pr_ctr++] = 2;
        for(int i = 4; i <= n; i+=2)
        {
            st.set(i,1);
        }
        int sqn = sqrt(n);
        for(int i = 3; i <= sqn ; i+=2)
```

```
        {
            if(st.test(i) == 0)
            {
                for(int j = i*i; j <= n; j+=2*i)
                {
                    st.set(j,1);
                }
            }
        }
        for(int i = 3; i <= n; i+=2)
        {
            if(st.test(i) == 0)primes[pr_ctr++]=i;
        }
    }
```

8. PRIME FACTORIZATION OF n!

```
    ///PRIME FACTORIAZATION OF n!
    void factFactorize ( int n ) {
        int cntr = 0;
        for ( int i = 0; i < pr_ctr && primes[i] <= n; i++ )
        {
            int x = n;
            int freq = 0;

            while ( x / primes[i] ) {
                freq += x / primes[i];
                x = x / primes[i];
            }
            if(freq == 0)continue;

            cntr++;
            if(cntr > 1) printf(" *");
            printf (" %d (%d)", primes[i], freq );
        }
        printf("\n");
    }
```

9. EULER SIEVE PHI/ RANGED COPRIME COUNTER:

```
/*MUST HAVE SIV WRITTEN ABOVE THIS*/
ui phi[5000001];
bitset<5000001> mark(0);

void sievephi()
{
    int n = 5000000;
    int i,j;
    for(i = 1; i <= n; i++)
    {
        phi[i]=i;
    }

    mark.set(1,1);
    phi[1] = 1;
    for(i = 2; i <= n; i+=2)
    {
        if( i != 2)mark[i]=1;
        phi[i]/=2;
    }

    for(i = 3; i <= n; i+=2)
    {
        if(mark[i]==0)
        {
            phi[i]--;

            for(j = 2*i; j <= n; j+=i)
            {
                mark.set(j,1);
                phi[j] = phi[j]/i * (i-1);
            }
        }
    }
}
```

10. EULER PHI/SINGLE INPUT COPRIME COUNTER:

```
li eulerPhi(li n)
{
    li res = n;
    li sqrtn = sqrt(n);
    for(int i = 0; i < pr_ctr && primes[i] <= sqrtn; i++)
    {
        if(n % primes[i] == 0)
        {
            while(n % primes[i] == 0)
            {
                n/=primes[i];
            }
            sqrtn = sqrt(n);
            res/=primes[i];
            res*=(primes[i]-1);
        }
    }
    if(n != 1)
    {
        res/=n;
        res*=(n-1);
    }
    return res;
}
```

11. NUMBER OF DIVISOR OF A SINGLE INTEGER:

```
int NOD ( int n ) {
    int sqrtn = sqrt ( n );
    int res = 1;
    for ( int i = 0; i < primes.size() && primes[i] <= sqrtn; i++
) {
        if ( n % primes[i] == 0 ) {
            int p = 0; /*Counter for power of prime*/
            while ( n % primes[i] == 0 ) {
                n /= primes[i];
                p++;
            }
            sqrtn = sqrt ( n );
            p++;/*Increase it by one at end*/
```

```
                res *= p; /*Multiply with answer*/
            }
        }
        if ( n != 1 ) {
            res *= 2; /*Remaining prime has power p^1. So multiply
    with 2*/
        }
        return res;
    }
```

12. Sieve Factorization technique, that counts the number of prime factors

```
int factor_ctr[2000001];
int pr_ctr = 0;
pair<int,int> arr[2000001];

///Function to count prime factors of all numbers from 1 to n
///O(nloglog(n))

void siv_factorize()
{
    int n = 2000000;
    for(int i = 2; i <= n; i++)
    {
        if(factor_ctr[i] == 0)
        {
            factor_ctr[i] = 1;
            //proves that 'i' is a prime, so it's only prime factor is
itself
            for(int j = i << 1; j <= n; j+=i)//(i << 1) = 2*i
            {
                int p = j;
                int freq = 0;
                while(p % i == 0)
                {
                    p/=i;
                    freq++;
                }
                factor_ctr[j]+=freq;
```

```
                //adding how many times i (the prime) is in j;
            }
        }
    }
    factor_ctr[1] = 0; //1 has no prime factors
    for(int i = 1; i <= n; i++)
    {
        arr[i] = make_pair(factor_ctr[i],i);
        //storing each number with it's prime_factor counts
        //for sorting factor_ctr is the primary key
        //number itself is the secondary key
    }
    sort(arr+1, arr+n+1); //sorting the numbers on basis of the keys
}
```

---

13. **Eucledian GCD LCM**

```
template <typename t1> t1 gcd(t1 a, t1 b)
{while(!b){a=a%b;swap(a,b);}return a;}
template <typename t1> t1 lcm(t1 a, t1 b) { return a * (b / gcd(a,
b)); }
```

---

14. **SNOD, sum of number of divisors**

1 = {1}, 2 = {1,2}, 3 = {1,3}, 4= {1,2,4}, 5= {1,5}; hence here the SNOD = SUM (Number of divisior) = for(I = 1 to 5){sum+=NOD[i]}

But this is a naïve approach. A better way to do this is:
http://blog.forthright48.com/2015/07/divisor-summatory-function.html

```
int SNOD( int n )
{
        int res = 0;
        int u = sqrt(n) ;
        for ( int i = 1; i <= u; i++ ) {
                res += (int)( n / i ) - i; //Step 1
        }
        res *= 2; //Step 2
        res += u; //Step 3
        return res;
```

```
}
```

15. Prime factorization of a single integer, O((sqrt(n)/(logn))+(logn))

```
vector<int> factors;
void factorize( int n )
{
        int sqrtn = sqrt ( n ) ;
        for ( int i = 0; i < pr_ctr && primes[i] <= sqrtn; i++ )
        {
                if ( n % primes[i] == 0 )
                {
                        while ( n % primes[i] == 0 )
                        {
                                n /= primes[i] ;
                                factors.push_back(primes[i]) ;
                        }
                        sqrtn = sqrt ( n ) ;
                }
        }
        if ( n != 1 )
        {
                factors. push_back(n) ;
        }
}
```

factors would contain the prime factorization of the integer n
in practical case the complexity can be counted as ***O(sqrt(n)+logn)***

16. SUM OF DIVISOR OF A SINGLE INTEGER N,(For SOD of a range of integers see above algorithm 1)

```
# It is to be noted:
```
Maximum sum of LCM of N = sum of divisors of N

```
# if 12 = (3^1)*(2^2) or (2^a)*(3^b) then
SUM OF DIVISOR(12) = (2^0+2^1+2^2)*(3^0+3^1)
                   = (2^0+2^1+……+2^a)*(3^0+……+3^b)
//CODE**********************
int SOD( int n )
```

```
{
        int sum_of_divisors = 0;
        int sqrtn = sqrt (n) ;
        for ( int i = 0; i < pr_ctr && primes[i] <= sqrtn; i++ )
        {
                if ( n % primes[i] == 0 )
                {
                        int primesum = 1;
                        int p_mul=1;
                        while ( n % primes[i] == 0 )
                        {
                                n /= prime[i] ;
                                p_mul*=prime[i];
                                primesum+=p_mul;
                        }
                        sqrtn = sqrt ( n ) ;
                        sum_of_divisors*= primesum;
                }
        }
        if ( n != 1 )
        {
                int primesum = 1 + n;
                sum_of_divisors*=primesum;
        }
        return sum_of_divisors;
}
```

forthright48

Learning Never Ends

Wednesday, July 1, 2015

Euclidean Algorithm - Greatest Common Divisor

Problem

Given two number A and B, find the greatest number that divides both A and B.

What we are trying to find here is the Greatest Common Divisor(GCD) of A and B. What is GC suggests, it the largest number (let that be G), that can divide both A and B. For example, G $GCD(3, 4) = 1, GCD(12, 15) = 3.$

How do we find it?

There are many ways. One way is to list down all the divisors of A and B and then find the larg divisors from those two lists. This is a naive method and takes too much time.

Another approach is to use Euclidean Algorithm, that works on the principle $GCD(a, b) = G$ Since $GCD(b, a\%b)$ is a smaller state, it is easier to find than the original. And of course, we principle on the smaller states repeatedly until the state becomes trivial. The two trivial states f $GCD(a, a) = a$ and $GCD(a, 0) = a$.

Euclidean Algorithm

Recursive Version

The recursive version of GCD is simple and small. Just 4 lines of code are enough to find GCI

```
1  int gcd ( int a, int b ) {
2      if ( b == 0 ) return a;
3      return gcd ( b, a % b );
4  }
```

Iterative Version

It's possible to find GCD without using recursion. This removes the recursion overhead and m faster.

```
1  int gcd ( int a, int b ) {
2      while ( !b ) {
3          a = a % b;
4          swap ( a, b );
```

```
    }
    return a;
}
```

Built-In Version
There is also a builtin function in C++ for finding gcd. You can simply write __gcd(a,b) to find G

Proof

The Euclidean Algorithm works on the principle $GCD(a, b) = GCD(b, a\%b)$. If we can pro
will be no doubt about the algorithm.

Let $g = GCD(a, b)$ and $a = k \times b + r$, where k is a non-negative integer and r is the rema
divides a, g also divides $k \times b + r$. Since g divides b, g also divides $k \times b$. Therefore, g mus
otherwise $k \times b + r$ won't be divisible.

So we proved that g divides b and r.

Now lets say we have $g' = gcd(b, r)$. Since g' divides both b and r, it will divide $k \times b + r$. T
divide a.

Now, can g and g' be two different numbers?

We will prove this using contradiction. Let's say that $g > g'$. We know that g divides both b an
$gcd(b, r)$ be g' when we have a number greater than g' that divides both b and r? So g cann
than g'.

Using the same logic, we find there is a contradiction when $g < g'$. Therefore, the only possib

$\therefore GCD(a, b) = GCD(b, r) = GCD(b, a\%b)$.

Complexity

It's kind of tricky. For now, just look at this answer from StackOverflow. The complexity of Eculi
according to the answer is $O(log_{10}A + log_{10}B)$.

Apparently, there is a whole chapter in Knuth's book TAOCP. I will write a separate post on the
this algorithm when I understand it. For now, just know that it works fast.

Properties

1. Every common divisor of a and b is a divisor of $gcd(a, b)$.
2. The gcd is a commutative function: $gcd(a, b) = gcd(b, a)$.
3. The gcd is an associative function: $gcd(a, gcd(b, c)) = gcd(gcd(a, b), c)$.
4. The gcd of three numbers can be computed as $gcd(a, b, c) = gcd(gcd(a, b), c)$, or in
   way by applying commutativity and associativity. This can be extended to any number c

More properties can be found on Wiki page

Coding Pitfalls

Notice that the algorithm work correctly for non-negative inputs only. Try to find $GCD(4, -2)$ algorithm. The correct answer should be 2. GCD will always be positive. But it returns -2. Even algorithm works correctly only for a non-negative number, it can easily be extended to work w numbers. You need to either send the absolute value of the inputs to the algorithm or use the the return value.

Next, notice that $GCD(0, 0) = 0$. It should be infinity. Also, if you try to work with the return v $GCD(0, 0)$, then you might get RTE due to division by zero.

Try to be careful in the two scenarios above.

Resources

1. Wikipedia - Greatest Common Divisor

Related Problems

1. UVa 11417 - GCD
2. UVa 11827 - Maximum GCD

Posted by Mohammad Samiul Islam

G+1 Recommend this on Google

Labels: GCD, Number Theory

No comments:

Post a Comment

Leave comments for Queries, Bugs and Hugs.

forthright48

Learning Never Ends

Wednesday, July 8, 2015

Lowest Common Multiple of Two Number

Problem

Given two number A and B, find their lowest common multiple (LCM).

We are trying to find the LCM of A and B. What is LCM? It is the smallest **positive** number whi both A and B.

How do we find it?

It is based on the formula that, $LCM(A, B) \times GCD(A, B) = A \times B$. How did we get this discuss it another day. It's not that hard to figure out though. Anyways, from that formula we c $LCM(A, B) = \frac{A \times B}{GCD(A,B)}$.

For example, $LCM(6, 15) = \frac{(6 \times 15)}{GCD(6,15)} = \frac{90}{3} = 30$, $LCM(3, 4) = \frac{3 \times 4}{GCD(3,4)} = \frac{12}{1} = 12$

Code and Pitfalls

Let us try to convert the above equation for LCM to code. If we convert exactly like the equatio something like the following:

```
1  int lcm ( int a, int b ) {
2      return ( a * b ) / gcd ( a, b );
3  }
```

This will work, but for some cases this code will overflow. For example, if we try to find $LCM($ obvious that the LCM is 2^{20}. But notice what happens when we follow the algorithm. We first t with 2^{15}, which results in 2^{35} which is too big to fit in an "int" variable. It overflows, and returns But the LCM itself should fit in the "int" variable easily.

A better way to write the above code is using the following observation.
$LCM(A, B) = \frac{A \times B}{GCD(A,B)} = \frac{A}{GCD(A,B)} \times B$. Since $GCD(A, B)$ divides both A and B, the $\frac{A}{GCD(A,B)}$ will be an integer. This alternate equation avoids overflow since instead of multiplyi reduces A to a smaller number and multiplies the resultant number with B. So, if $LCM(A, B)$ variable, then we will get it without the risk of intermediate products overflowing.

```
int lcm ( int a, int b ) {
    return ( a / gcd ( a, b ) ) * b;
}
```

Related Problems

1. UVa 11388 - GCD LCM **(Analysis)**

Posted by Mohammad Samiul Islam

G+1 Recommend this on Google

Labels: LCM, Number Theory

No comments:

Post a Comment

Leave comments for Queries, Bugs and Hugs.

Enter your comment...

Comment as: Unknown (Goo

Publish Preview

Newer Post Home

Subscribe to: Post Comments (Atom)

Blog Archive

▼ 2015 (35)
► Sep (7)
► Aug (13)
▼ Jul (15)

Labels

Analysis Arithmetic Function Backtrack Big
Complexity Contest D&C Divisors Factorial
Language LCM Logarithm Math M
Number Theory Optimization Prim
Repeated Squaring Sieve SPOJ Theorem UVa

forthright48

Learning Never Ends

Wednesday, July 8, 2015

Primality Test - Naive Methods

Problem

Given a number N, determine if it is a prime.

What is a Prime Number?

A prime number is a positive number greater than 1 which has no positive divisor except for 1 example, 2, 3, 5, 11 are prime numbers. Whereas 4, 6 are non-prime or composite numbers.

O(N) Solution

From the definition, we can easily construct a function that can determine if a given number N Let us call it isPrime() function.

```
1  bool isPrime ( int n ) {
2      if ( n <= 1 ) return false; /*n needs to be greater than 1*/
3      for ( int i = 2; i < n; i++ ) {
4          /*If it is possible to divide n with a number other than 1 and
5          if ( n % i == 0 ) return false;
6      }
7      return true; /*Otherwise, this is prime*/
8  }
```

The code simply iterates over all values between 2 and N-1 and checks if it can divide N. It ha O(N).

We can observe that, when i is greater than $n/2$, it will never divide n. Hence, we can optimi

```
1  bool isPrime ( int n ) {
2      if ( n <= 1 ) return false;
3      for ( int i = 2; i <= n / 2; i++ ) {
4          if ( n % i == 0 ) return false;
5      }
6      return true;
7  }
```

This, however, does not change the complexity.

O($\sqrt{N}$) Solution

We can further optimize the Primality test from the following observation. Let us consider A an $N = A \times B$. Now notice that, when $A = \sqrt{N}$, B is also $\sqrt{N}$. Otherwise, $A \times B$ would not What happens when $A > \sqrt{N}$? In order for $A \times B$ to be equal to N, B must be $< \sqrt{N}$.

So using the following observation, we can claim that if N has a divisor which is $>= \sqrt{N}$, th divisor which is $<= \sqrt{N}$.

For example, 12.
$12 = 1 \times 12$
$12 = 2 \times 6$
$12 = 3 \times 4$
$12 = \sqrt{12} \times \sqrt{12}$

Every divisor $>= \sqrt{N}$ has a complementary divisor which is $<= \sqrt{N}$.

This means that if we fail to find any divisor of N below $\sqrt{N}$ then it is safe to assume we will r above $\sqrt{N}$.

```
1  bool isPrime ( int n ) {
2      if ( n <= 1 ) return false;
3      int sqrtn = sqrt(n);
4      for ( int i = 2; i <= sqrtn; i++ ) {
5          if ( n % i == 0 ) return false;
6      }
7      return true;
8  }
```

Notice that I defined a new variable "sqrtn" instead of simply writing "i <= sqrt(n)" in line 4. That function sqrt(n) has a complexity of O(√N) and writing it inside for loop condition would mean t would get called unnecessarily every time the loop iterates.

Posted by Mohammad Samiul Islam

G+1 Recommend this on Google

Labels: Number Theory, Primality Test, Prime

No comments:

Post a Comment

Leave comments for Queries, Bugs and Hugs.

forthright48

Learning Never Ends

Wednesday, July 8, 2015

Sieve of Eratosthenes - Generating Primes

Problem

Given an integer N, generate all primes less than or equal to N.

Sieve of Eratosthenes - Explanation

Sieve of Eratosthenes is an algorithm that generates all prime up to N. Read this article written Jan on Generating Primes - LightOJ Tutorial. The pdf contains almost all the optimizations of S Eratosthenes.

Code

```
1  vector<int> prime; /*Stores generated primes*/
2  char sieve[SIZE]; /*0 means prime*/
3  void primeSieve ( int n ) {
4      sieve[0] = sieve[1] = 1; /*0 and 1 are not prime*/
5
6      prime.push_back(2); /*Only Even Prime*/
7      for ( int i = 4; i <= n; i += 2 ) sieve[i] = 1; /*Remove multiple
8
9      int sqrtn = sqrt ( n );
10     for ( int i = 3; i <= sqrtn; i += 2 ) {
11         if ( sieve[i] == 0 ) {
12             for ( int j = i * i; j <= n; j += 2 * i ) sieve[j] = 1;
13         }
14     }
15
16     for ( int i = 3; i <= n; i += 2 ) if ( sieve[i] == 0 ) prime.push
17 }
```

Some lines from the code above can be omitted depending on situation. But as a whole, the a us two products, a vector<int> prime which contains all generated primes and a char[] sieve th whether a particular value is prime or not. We will need sieve array for optimizations in other a

Complexity

The algorithm has runtime complexity of $O(Nlog(logN))$

forthright48

Learning Never Ends

Thursday, July 9, 2015

Prime Factorization of an Integer

Problem

Given an integer N, find its prime factorization.

For example, $12 = 2 \times 2 \times 3 = 2^2 \times 3$.

It is somewhat difficult to factorize an integer. If N is small enough, then it is possible to factori
generate by Sieve of Eratosthenes.

Trial Division Method

Suppose we are trying to factorize 980. How would we attempt to factorize it with pen and pap
extract the smallest possible prime factors from it and make it smaller. That is we will try to divi
with prime numbers and keep track which are capable of dividing it.

The first prime we have is 2. Can we divide 980 with 2? Yes. $980 = 2 \times 490$. Now we need t
which is smaller and thus easier.

Again, let's try to factorize 490 with 2. It is possible, so $980 = 2 \times 2 \times 245 = 2^2 \times 245$. It is
extract another 2 from 245 so we move on to next prime.

3 fails to divide 245, so we move on to 5. By repeating this process, we find out that $980 = 2^$

This process is kind of laborious, but we have computers to do our bidding. So for this trial div
we need is a list of primes so that we can try dividing N one by one from that list. How will we g
Sieve of Eratosthenes.

But before we start generating prime, we need to answer another question. Sieve of Eratosthe
generate prime up to a certain number. So up to which value should we generate prime numb
N?

Limit for Sieve of Eratosthenes

Well, if we are trying to factorize N, there is no point in generating primes that are larger than generate up to N and factorize using the list with primes less than or equal to N. But we can observing a special property of prime factors of N.

N can have many prime factors. Some of the prime numbers will be $< \sqrt{N}$ and some $>=$ many primes factors can N have that are greater than $\sqrt{N}$?

Notice that if N can never have two prime factors $> \sqrt{N}$ since, $if\ A > \sqrt{N}, then\ A \times A$

So if we generate primes up to $\sqrt{N}$ and use that list to factorize N then at the end, we will eit number greater than $\sqrt{N}$ which will be a prime for sure.

For example, $N = 42$ and $\sqrt{N} = 6.4 \approx 6$. So let's try to factorize using primes less than or that is using only $[2, 3, 5]$. What do we get? $N = 42 = 2 \times 3 \times 7$. Since 7 can no longer be prime from our list $[2, 3, 5]$ we stop. 7 is our last missing prime factor.

So, in order to factorize N, we need to generate primes up to $\sqrt{N}$ only.

Code for Sieve of Eratosthenes

```
1  vector<int>factors;
2  void factorize( int n ) {
3      int sqrtn = sqrt ( n );
4      for ( int i = 0; i < prime.size() && prime[i] <= sqrtn; i++ ) {
5          if ( n % prime[i] == 0 ) {
6              while ( n % prime[i] == 0 ) {
7                  n /= prime[i];
8                  factors.push_back(prime[i]);
9              }
10             sqrtn = sqrt ( n );
11         }
12     }
13     if ( n != 1 ) {
14         factors.push_back(n);
15     }
16 }
```

Let me explain few things about the code. In line 4, inside the for loop I wrote the following cor prime.size() && prime[i] <= sqrtn". The first condition ensures that we don't go array over-boun understand it, imagine what will happen if we try to factorize a prime number $> \sqrt{N}$. Without you might get RTE.

The second condition ensures we are always trying to factorize using prime less than $\sqrt{N}$.

In line 5, we check if it is possible to divide current N with prime[i]. If so we start a loop in line 6 extracting until we cannot anymore.

In line 10, we are optimizing our code. Since, after extraction of prime factors, N has become need to only factorize with a smaller amount of prime numbers. Due to this single line, the cod really quickly.

In line 13, we are checking if the residual number after factorization is 1 or not. If not, it is the o which is greater than $\sqrt{N}$. So we add it to our prime factor list.

Time Complexity

There are two loops in the code for $factorize()$. The first one is at line 4. This loop runs once $\leq \sqrt{N}$. From "Prime Number Theorem", $\pi(N) \approx \frac{N}{ln(N)}$. So there are approximately $\frac{\sqrt{N}}{ln(\sqrt{N})}$

The second loop at line 6 runs once for every prime factor N has. So it cannot run more than

So the complexity is $O(\frac{\sqrt{N}}{ln(\sqrt{N})} + log_2(N))$. In practice, if N is not a prime then due to line 1 small quickly. So it is much faster than what we calculated.

Optional Optimization

There is an optional optimization we can perform only when it is possible to generate sieve up single statement in line 5.

```
4   for ( int i = 0; i < prime.size() && prime[i] <= sqrtn; i++ ) {
5           if ( sieve[n] == 0 ) break; /*Checks if n is prime or not*/
6           if ( n % prime[i] == 0 ) {
```

The new line added simply checks if the number we are trying to factorize is a prime or not. If i obviously we cannot factorize it anymore. This optimization is highly useful when we already h generated up to N.

Reference

1. forthright48 - Prime Number Theorem - http://forthright48.blogspot.com/2015/07/prime-n theorem.html

Posted by Mohammad Samiul Islam

G+1 Recommend this on Google

Labels: Factorization, Number Theory, Prime, Sieve

No comments:

Post a Comment

Leave comments for Queries, Bugs and Hugs.

forthright48

Learning Never Ends

Saturday, July 11, 2015

Number of Divisors of an Integer

Problem

Given an integer N, find its number of divisors.

For example, 12 has the following divisors $1, 2, 3, 4, 6$ and 12. So its number of divisors is 6.

Number Of Divisors Function: $NOD(N)$ or $\sigma_0(N)$

Number of Divisors of N is also called $\sigma_0(N)$. That's just a fancy way of asking for the numbe
You can read more on it from Wiki. I will refer to Number of Divisors of N as either $\sigma_0(N)$ or 1
both mean same.

Brute Force Method O(N)

Anyways, what is the easiest way to find Number of Divisors (NOD)? We can try to divide N w
from $1 - N$ and keep count of how many of those numbers divide N. This will definitely work
need to do better.

Optimized to O($\sqrt{N}$)

In "Primality Test - Naive Methods", we established that if we can write $N = A \times B$, then one
be $<= \sqrt{N}$.

So using that same idea we can find NOD by simply looping till $\sqrt{N}$ and check if that particula
N. If it doesn't then it is not a divisor and if it does then we found a $\{A, B\}$ pair of divisors. If A
add 2 to result, otherwise we add 1.

Examples

Suppose $N = 24$. We need to loop till $\lfloor \sqrt{24} \rfloor = 4$. We get the following pairs of divisors,
$\{1, 24\}, \{2, 12\}, \{3, 8\}, \{4, 6\}$. So our answer is 8.

Let's try again with $N = 36$. We need to loop till $\lfloor \sqrt{36} \rfloor = 6$. We get the following pairs of d
$\{1, 36\}, \{2, 18\}, \{3, 12\}, \{4, 9\}, \{6, 6\}$. So our answer is 9. Notice that the last pair does n
condition $A \neq B$. So we add one for that pair.

This brings out an interesting observation. NOD of N is always even except for when N is a p
square. Because whenever N is perfect square, $\sqrt{N}$ would be its divisor and it will form a pa

Code Using Loop till $\sqrt{N}$

```
1  int NOD ( int n ) {
2      int res = 0;
3      int sqrtn = sqrt ( n );
4
5      for ( int i = 1; i < sqrtn; i++ ) {
6          if ( n % i == 0 ) res += 2; //Found a divisor pair {A,B}
7      }
8
9      //Need to check if sqrtn divides n
10     if ( n % sqrtn == 0 ) {
11         if ( sqrtn * sqrtn == n ) res++; //If n is perfect square
12         else res += 2; //Not perfect square
13     }
14
15     return res;
16 }
```

We loop from 1 to $\sqrt{N} - 1$ at line 5. Then at line 10, we handle $\sqrt{N}$ seperately.

Using Prime Factorization

It is possible to find NOD from prime factorization of N.

Suppose $N = 12$. Its prime factorization is $2^2 \times 3$. Is it possible to divide 12 with 5? No. Caus
factorization of 12 does not contain 5. We can divide N with primes that appear in factorizatic

Next, can we divide 12 with 2^3? No. The power of 2 in prime factorization of 12 is 2^2. So we ca
with 2^3, since 2^2 is not divisible by 2^3.

So, if we are trying to divide N, that has prime factorization $N = p_1^{a_1} \times p_2^{a_2} \times \ldots p_x^{a_x}$, then the
will have prime factorization of form $D = p_1^{b_1} \times p_2^{b_2} \times \ldots p_x^{b_x}$, where $b_i <= a_i$.

For example, divisor of $12 = 2^2 \times 3$ will have prime factorization, $D = 2^x \times 3^y$, where $x <=$
When $x = 1$ and $y = 1$, $D = 2^1 \times 3^1 = 6$ which is a divisor of 12. If we use a different com
then we get a different divisor. So how many different combination can we use here? We kno
$0 <= x <= 2$ and $0 <= y <= 1$. So for x we can use $\{0, 1, 2\}$ and for y we can use $\{0,$
use $3 \times 2 = 6$ combinations of $\{x, y\}$. And that is our NOD.

So, $if\ N = p_1^{a_1} \times p_2^{a_2} \times \ldots p_x^{a_x},\ then\ D = p_1^{b_1} \times p_2^{b_2} \times \ldots p_x^{b_x}$. We get new divisors for usin
combination for $\{b_1, b_2 \ldots b_x\}$. How many different combinations can we make?
$(a_1 + 1) \times (a_2 + 1) \times \ldots (a_x + 1)$. Therefore, $\sigma_0(N) = (a_1 + 1) \times (a_2 + 1) \times \ldots (a_x +$

So basically, in order to find NOD, all we need to do is factorize N, then take each of its prime
increase them by 1 and finally multiply all of them. Easy right?

Examples

$N = 12 = 2^2 \times 3. NOD(N) = (2 + 1) \times (1 + 1) = 6.$
$N = 15 = 3 \times 5. NOD(N) = (1 + 1) \times (1 + 1) = 4.$

$N = 252 = 2^2 \times 3^2 \times 7. NOD(N) = (2+1) \times (2+1) \times (1+1) = 18.$

Try out this yourself using other small values of N.

Code Using Prime Factorization

The code for NOD is not very different from the one we used for $factorize()$ in Prime Factc Integer. We just need to modify it slightly to suit our needs.

```
int NOD ( int n ) {
    int sqrtn = sqrt ( n );
    int res = 1;
    for ( int i = 0; i < prime.size() && prime[i] <= sqrtn; i++ ) {
        if ( n % prime[i] == 0 ) {
            int p = 0; /*Counter for power of prime*/
            while ( n % prime[i] == 0 ) {
                n /= prime[i];
                p++;
            }
            sqrtn = sqrt ( n );
            p++;/*Increase it by one at end*/
            res *= p; /*Multiply with answer*/
        }
    }
    if ( n != 1 ) {
        res *= 2; /*Remaining prime has power p^1. So multiply with 2
    }
    return res;
}
```

I have highlighted the lines that are different from $factorize()$.

Reference

1. Divisor Function - https://en.wikipedia.org/wiki/Divisor_function

Posted by Mohammad Samiul Islam

G+1 Recommend this on Google

Labels: Divisors, Factorization, Number Theory

No comments:

Post a Comment

Leave comments for Queries, Bugs and Hugs.

forthright48

Learning Never Ends

Monday, July 13, 2015

Highly Composite Numbers

Definition

> A Highly Composite Number (HCN) is a positive integer which has more divisors than ar
> positive integer (Wiki), i.e, if $NOD(N) > NOD(i)$, where $0 < i < N$, then N is HC

For example, 6 is HCN cause $NOD(6) = 4$ which is bigger than $NOD\{1, 2, 3, 4, 5\} = \{1$

Here are the first few HCN: $1, 2, 4, 6, 12, 24, 36, 48, 60, 120, 180, 240$ - A002182

Properties of Highly Composite Numbers

There are two properties of HCN and both of them are related to prime factorization.

Suppose we have a HCN with prime factorization $HCN = p_1^{a_1} \times p_2^{a_2} \ldots \times p_k^{a_k}$, then:

1. **First K Primes**: The prime factorization of HCN will contain the first K consecutive prime
   then we can replace the k^{th} prime in factorization with a smaller prime and still have sar
   example, $2^2 \times 5 = 20$ cannot be a HCN, since we can replace prime factor 5 with 3 an
   $2^2 \times 3 = 12$ which is smaller than 20 and has same NOD.
2. **Non-Increasing Power of Primes**: Power of prime factors of HCN, i.e, $\{a_1, a_2 \ldots a_k\}$ w
   increasing sequence. Why is that? If power $a_j > a_i$, where $i < j$, then we can simply s
   a smaller number with same NOD. For example, $2 \times 3^2 = 18$ cannot be HCN cause w
   of prime 2 and 3 to get $2^2 \times 3 = 12$ which is smaller with same NOD.

Therefore we conclude that Highly Composite Numbers are product of Primorials (Primorials a
Factorials, but instead of natural number we multiply primes. $p_3\# = 2 \times 3 \times 5$ and
$p_5\# = 2 \times 3 \times 5 \times 7 \times 11$)

For example, 180 is HCN and it can be decomposed into $180 = p_3\# \times p_2\# = (2 \times 3 \times 5$

Generating Highly Composite Numbers

Problem: Given an integer N, find the largest HCN which is smaller than or equal to N.

This problem can be solved using backtrack. We just need to ensure that we satisfy the two p

Let's try to solve it for when $N = 10^9$. We know that $p_{10}\# = 6469693230$, so we will only h
$\{2, 3, 5, 7, 11, 13, 17, 19, 23\}$ in HCN factorization.

Now, we will assign power to each of the prime in non increasing order to generate numbers
corresponding NOD. Out of all the numbers generated, we will keep the one with highest NO
tie, the one with smallest value.

```
1  //prime[] is a list of prime.
2  int prime[] = {2, 3, 5, 7, 11, 13, 17, 19, 23 };
3
4  int resNum, resDiv, n;
5  void recur ( int pos, int limit, long long num, int div ) {
6      if ( div > resDiv ) { //Get the number with highest NOD
7          resNum = num;
8          resDiv = div;
9      }
10     else if ( div == resDiv && num < resNum ) { //In case of tie, tak
11         resNum = num;
12     }
13
14     if ( pos == 9 ) return; //End of prime list
15
16     long long p = prime[pos];
17
18     for ( int i = 1; i <= limit; i++ ) {
19         if ( num * p > n ) break;
20         recur ( pos + 1, i, num * p, div * ( i + 1 ) );
21         p *= prime[pos];
22     }
23 }
```

Line 2 contains a prime list. It contains first 9 primes so it will work correctly for $N \leq 10^9$. Line
variables to store result. $resNum$ to hold required value and $resDiv$ to hold its NOD. Line 1
multiplying $prime[pos]^i$ with res is becoming bigger than N or not.

In line 20, we call $recur()$ with $pos + 1$ as we want to work with the next prime in list, i as the
we don't want the next prime to have power greater than the current one. The next two param
adjusted accordingly to maintain value and NOD.

In order to solve the above problem for $N = 10^9$, we have to initiate and call $recur()$ from m
following manner.

```
1  int main () {
2      n = 1000000000;
3      resNum = 0;
4      resDiv = 0;
5      recur ( 0, 30, 1, 1 );
6      printf ( "%d %d\n", resNum, resDiv );
7  }
```

In line 5, we call $recur(0, 40, 1, 1)$. pos is assigned 0 since we want to start with the first prim
$limit$ parameter is set to 30 since $2^{30} > N$.

Reference

forthright48

Learning Never Ends

Tuesday, July 14, 2015

Upper Bound for Number Of Divisors

Given a number N, we can find its number of divisors using prime factorization. But when the n
big, it gets difficult to factorize thus harder to find the number of divisors. So given a number N
at most how many divisors N can have? That is, we want to find the upper bound for NOD.

NOD Upper Bounds

We proceed from loose to tighter bounds.

$$NOD(N) \leq N$$

A number which is greater than N cannot divide N. For $N > 2$, we can further say $NOD(N$
improvement is negligible.

Can we make the limit tighter?

$$NOD(N) \leq \frac{N}{2} + 1$$

A divisor D, such that $\frac{N}{2} < D < N$ cannot divide N, since the result would be less than 2 a
1, a fraction. So possible divisors are $D \leq \frac{N}{2}$ and N.

Better than before by half, but its possible to make it tighter.

$$NOD(N) \leq 2 \times \sqrt{N}$$

If we write $N = A \times B$, where $A \leq B$, then $A \leq \sqrt{N}$. Each of $\{A, B\}$ forms a divisor pair
take any value from $1 \; to \; \sqrt{N}$, so it is possible to form only $\sqrt{N}$ pairs of divisors. Thus, there
$2 \times \sqrt{N}$ divisors for N.

$$NOD(N) \approx 2 \times \sqrt[3]{N}$$

I didn't know about this approximation until I read a blog post on Codeforces. From there I fou
practice we can safely use $\sqrt[3]{N}$ as an upper bound, though I failed to understand the proof. A
approximation has been tested for $N \leq 10^{18}$, which is large enough to be used in programm

Using Highly Composite Numbers

Sometimes we need upper bound for small values of N. For example, in a problem you might upper bound of NOD for $N \leq 10^9$. For such small values of N we could use NOD of largest Number (HCN), which is less than or equal to N, as an upper bound.

Read more about HCN here.

For programming contest, we could memorize values of HCN that comes frequently. Mainly 1 and $103,680$ for $N \leq 10^{18}$.

Application

The upper bound for NOD is needed for complexity analysis.

Reference

1. Codeforces - Upper bound for number of divisors: http://codeforces.com/blog/entry/144
2. forthright48 - Highly Composite Numbers: http://forthright48.blogspot.com/2015/07/highl numbers.html

Posted by Mohammad Samiul Islam

Labels: Complexity, Divisors, Number Theory

G+1 Recommend this on Google

No comments:

Post a Comment

Leave comments for Queries, Bugs and Hugs.

Enter your comment...

Comment as: Unknown (Goo

Publish Preview

forthright48

Learning Never Ends

Thursday, July 16, 2015

Prime Number Theorem

> In number theory, the prime number theorem (PNT) describes the asymptotic distributio
> prime numbers among the positive integers. It formalizes the intuitive idea that primes b
> common as they become larger by precisely quantifying the rate at which this occurs. -

The definition above is hard to understand. To put it simply, PNT provides us with an estimatic
Counting Function.

What is Prime-Counting Function?

This brings out another Wiki definition:

> In mathematics, the prime-counting function is the function counting the number of prim
> less than or equal to some real number x. It is denoted by $\pi(x)$ (this does not refer to t
>). - Wiki

This definition is easy to understand. Prime-Counting Function, $\pi(N)$, gives us number of pri
equal to N. For example, $\pi(10) = 4$ since there are 4 primes, $\{2, 3, 5, 7\}$, which are ≤ 10.

PNT - Estimation for $\pi(N)$

So, as I was saying before, PNT provides us with an estimation for Prime-Counting Function. I

$$\pi(N) \approx \frac{N}{ln(N)}$$

The accuracy of the estimation increase as N becomes larger.

Application

We can use the theorem for complexity analysis.

Reference

1. Wiki - Prime Number Theorem

forthright48

Learning Never Ends

Monday, July 20, 2015

Sum of Divisors of an Integer

Problem

Given an integer N, find the sum of all its divisors.

For example, find Sum Of Divisors (SOD) of 12. 12 has the following divisors, $\{1, 2, 3, 4, 6, 12$
$SOD(12) = 1 + 2 + 3 + 4 + 6 + 12 = 28$.

Sum of Divisor Function: $SOD(N)$ or $\sigma_1(N)$

The Sum of Divisors of N is also called $\sigma_1(N)$. That's just a fancy way of saying Sum of Divis
can read more on it from Wiki. I will refer to Sum of Divisors of N as either $\sigma_1(N)$ or $SOD(N$
mean same.

Inefficient Solutions

Let us first look into some simple, but inefficient, solutions. These solutions are similar to the n
came up for "Number of Divisors of an Integer".

Loop Till N

This is the simplest solution. We loop from 1 to N and add all numbers that divide N.

Loop Till $\sqrt{N}$

If $N = A \times B$, where $A \leq B$, A must be $\leq \sqrt{N}$. Using this fact we can run a loop from 1 t
all numbers that divide N and their complements to result.

```
1  int SOD ( int n ) {
2      int sqrtn = sqrt ( n );
3      int res = 0;
4      for ( int i = 1; i < sqrtn; i++ ) {
5          if ( n % i == 0 ) {
6              res += i; //"i" is a divisor
7              res += n / i; //"n/i" is also a divisor
8          }
9      }
10     if ( n % sqrtn == 0 ) {
11         if ( sqrtn * sqrtn == n ) res += sqrtn; //Perfect Square.
12         else {
```

```
13                  res += sqrtn; //Two different divisor
14                  res += n / sqrtn;
15              }
16          }
17          return res;
18      }
```

At line 10, we handle $sqrtn$ separately cause when N is a perfect square we don't get differe
$\{A, B\}$ pair. For example, for 49 we have a divisor pair 7×7. We need to add 7 only once i

Using Prime Factorization

It is possible to find $SOD(N)$ using the prime factorization of N. Let us see an example for i

$$
\begin{aligned}
&Let\ N = 12, \\
&SOD(12) = 1 + 2 + 3 + 4 + 6 + 12, \\
&SOD(12) = (2^0 \times 3^0) + (2^1 \times 3^0) + (2^0 \times 3^1) + (2^2 \times 3^0) + (2^1 \times 3^1) + (2^2 \times 3^1) \\
&SOD(12) = 2^0(3^0 + 3^1) + 2^1(3^0 + 3^1) + 2^2(3^0 + 3^1), \\
&SOD(12) = (2^0 + 2^1 + 2^2) \times (3^0 + 3^1)
\end{aligned}
$$

This pattern emerges with any value of N. More generally, if $N = p_1^{a_1} \times p_2^{a_2} \times \ldots p_k^{a_k}$, then

$$SOD(N) = (p_1^0 + p_1^1 + p_1^2 \ldots p_1^{a_1}) \times (p_2^0 + p_2^1 + p_2^2 \ldots p_2^{a_2}) \times \ldots (p_k^0 + p_k^1 + p_k^2$$

Using this formula and our code for Prime Factorization, we can write the function $SOD(N)$.
similar to $factorize()$. We only need to modify it in few places.

```
1   int SOD( int n ) {
2       int res = 1;
3       int sqrtn = sqrt ( n );
4       for ( int i = 0; i < prime.size() && prime[i] <= sqrtn; i++ ) {
5           if ( n % prime[i] == 0 ) {
6               int tempSum = 1; //Contains value of (p^0+p^1+...p^a)
7               int p = 1;
8               while ( n % prime[i] == 0 ) {
9                   n /= prime[i];
10                  p *= prime[i];
11                  tempSum += p;
12              }
13              sqrtn = sqrt ( n );
14              res *= tempSum;
15          }
16      }
17      if ( n != 1 ) {
18          res *= ( n + 1 ); //Need to multiply (p^0+p^1)
19      }
20      return res;
21  }
```

For each prime we find in factorization, we need to find the corresponding value of $(p^0 + p^1 +$
we use $tempSum$ in line 6. It contains p^0 initially. Then for each successful division at line 8,
power of p in line 10 and add it to $tempSum$ in line 11. Eventually, we multiply the final sum t

In line 17, we check if we are left with a sole prime remainder. In this case, we know that n is a
simply multiply $p^0 + p^1 = 1 + n$ to the result.

Reference

1. Wiki - Divisor Function - https://en.wikipedia.org/wiki/Divisor_function
2. forthright48 - Number of Divisors of an Integer - http://forthright48.blogspot.com/2015/07 divisors-of-integer.html
3. forthright48 - Prime Factorization - http://forthright48.blogspot.com/2015/07/prime-factori integer.html

Posted by Mohammad Samiul Islam

G+1 Recommend this on Google

Labels: Divisors, Factorization, Number Theory

No comments:

Post a Comment

Leave comments for Queries, Bugs and Hugs.

Enter your comment...

Comment as: Unknown (Goo

Publish Preview

Newer Post Home

Subscribe to: Post Comments (Atom)

Blog Archive

▼ 2015 (35)
- ► Sep (7)
- ► Aug (13)
- ▼ Jul (15)
  - Simple Hyperbolic Diophantine Equation
  - Linear Diophantine Equation

Labels

Analysis Arithmetic Function Backtrack Bi Complexity Contest D&C Divisors Factorial Language LCM Logarithm Math M Number Theory Optimization Prin Repeated Squaring Sieve SPOJ Theorem UV

forthright48

Learning Never Ends

Monday, July 20, 2015

Divisor Summatory Function

Problem

Given an integer N, find the Sum of Number of Divisors from 1 to N. That is, find $\sum_{i=1}^{N} NO$.

For example, find the Sum of Number of Divisors, $SNOD()$, when $N = 5$.
$SNOD(5) = NOD(1) + NOD(2) + NOD(3) + NOD(4) + NOD(5) = 1 + 2 + 2$

Divisor Summatory Function

> "In number theory, the divisor summatory function is a function that is a sum over the di function." - Wiki

Divisor Summatory Function is defined as $D()$ in Wiki, but we will use $SNOD()$ in this article Summatory Function finds Sum of Number of Divisors from 1 to N.

Brute Force Solution - $O(N^2)$ Approach

A simple solution is to run two nested loop which counts all divisors from 1 to N. It is simple b

```
1  int SNOD( int n ) {
2      int res = 0;
3      for ( int i = 1; i <= n; i++ ) { //For each number from 1 - N
4          for ( int j = 1; j <= i; j++ ) { //Find possible divisors of "
5              if ( i % j == 0 ) res++;
6          }
7      }
8      return res;
9  }
```

Using $NOD()$ - $O(N \times \frac{\sqrt{N}}{ln(\sqrt{N})})$ Approach

We can use the code for $NOD()$ to get the number of divisors for each integer from 1 to N i $O(N)$ loop. This will make it faster than the above solution.

```
1  int SNOD( int n ) {
2      int res = 0;
3      for ( int i = 1; i <= n; i++ ) {
```

```
4           res += NOD(i);
5       }
6       return res;
7   }
```

Since $NOD()$ has same complexity as $factorize()$, the complexity for this code is $O(N \times$

Using Divisor List - $O(N)$ Approach

Suppose we are trying to find $SNOD(10)$. Let us write down the divisors for each integer be

$NOD(1) = 1,\ \{1\}$
$NOD(2) = 2,\ \{1, 2\}$
$NOD(3) = 2,\ \{1, 3\}$
$NOD(4) = 3,\ \{1, 2, 4\}$
$NOD(5) = 2,\ \{1, 5\}$
$NOD(6) = 4,\ \{1, 2, 3, 6\}$
$NOD(7) = 2,\ \{1, 7\}$
$NOD(8) = 4,\ \{1, 2, 4, 8\}$
$NOD(9) = 3,\ \{1, 3, 9\}$
$NOD(10) = 4,\ \{1, 2, 5, 10\}$

So $SNOD(10) = 1 + 2 + 2 + 3 + 2 + 4 + 2 + 4 + 3 + 4 = 27$.

Now, look at the divisor list for each of integer between 1 and N. Each divisor in the divisor lis the result. There are 27 divisors in total.

How many times do we find 1 in the divisor lists? It will appear once for every integer it can div can divide $\frac{N}{1}$ numbers. So it appears $\frac{10}{1} = 10$ times.

Repeat for $2\ to\ N$. How many times do we find 2? 2 can divide $\frac{10}{2} = 5$ numbers, namely $\{2,$ will appear 5 times.

By repeating this from 1 to N we can find $SNOD(N)$.

```
1   int SNOD( int n ) {
2       int res = 0;
3       for ( int i = 1; i <= n; i++ ) {
4           res += n / i;
5       }
6       return res;
7   }
```

We removed $NOD()$ from the code in line 4 and replaced it with $\frac{n}{i}$.

Using Divisor Pairs - $O(\sqrt{N})$ Approach

Let us create another list. Instead of making divisor list, we will make "Ordered Divisor Pair" list between 1 to N. A divisor pair of value X is a pair (A, B) such that $A \times B = X$. Ordered pa $A \neq B$, then (A, B) is a different pair than (B, A).

$NOD(1) = 1,\ (1, 1)$
$NOD(2) = 2,\ (1, 2)(2, 1)$

$NOD(3) = 2,\ (1,3)(3,1)$
$NOD(4) = 3,\ (1,4)(2,2)(4,1)$
$NOD(5) = 2,\ (1,5)(5,1)$
$NOD(6) = 4,\ (1,6)(2,3)(3,2)(6,1)$
$NOD(7) = 2,\ (1,7)(7,1)$
$NOD(8) = 4,\ (1,8)(2,4)(4,2)(8,1)$
$NOD(9) = 3,\ (1,9)(3,3)(9,1)$
$NOD(10) = 4,\ (1,10)(2,5)(5,2)(10,1)$

The "Ordered Divisor Pair" list is almost same as "Divisor List". The only difference is that each paired with $B = \frac{X}{A}$.

Now, $NOD(X)$ represents number of ordered divisor pairs (A, B) such that $A \times B = X$. S is same as number of ordered divisor pairs such that $A \times B \leq N$.

How to find Number of Ordered Divisor Pairs $\leq N$?

We can find this by following three steps.

1. Find number of divisor pairs (A, B) where $A < B$

What are the possible values for A? Can it ever be $> \sqrt{N}$? No. If $A > \sqrt{N}$, then with $B > > \sqrt{N}$ and $A \times B$ will be $> N$. So A can only be between 1 and $\lfloor \sqrt{N} \rfloor$.

Let $u = \lfloor \sqrt{N} \rfloor$. Then for each value of A between 1 and u, what are the possible values of

For any value of A, there are $\frac{N}{A}$ possible values of B for which $A \times B$ does not exceed N. I $A = 2$, there are $\frac{10}{2} = 5$ possible values for B, and they are $\{1, 2, 3, 4, 5\}$. Multiplying $A =$ these values produces value $\leq N$. But we need $B > A$. So we omit $\{1, 2\}$ from the list. We $\{3, 4, 5\}$ as B.

So, for any value of A, there will be $\lfloor \frac{N}{A} \rfloor$ possible values of B. The values will be from 1 to $\lfloor \frac{N}{A} \rfloor$ we have to omit values that are $\leq A$. So we can use $\lfloor \frac{N}{A} \rfloor - A$ values for B.

For example, $N = 10$. Then $u = \lfloor \sqrt{10} \rfloor = 3$. When $A = 1$, number of legal values we can $\lfloor \frac{10}{1} \rfloor - 1 = 9$. Namely, B can be one of $\{2, 3, 4, 5, 6, 7, 8, 9, 10\}$.

When $A = 2$, legal values are $\lfloor \frac{10}{2} \rfloor - 2 = 3$. B can be $\{3, 4, 5\}$.
When $A = 3$, legal values are $\lfloor \frac{10}{3} \rfloor - 3 = 0$.

A cannot be bigger than u. So number of (A, B) pairs such that $A \times B \leq 10$ and $A < B$ is

2. Multiply the result with 2

Once we find the number of pairs (A, B) such that $A < B$ and $A \times B \leq N$, we multiply tha That is because we want to find the number of ordered divisor pairs. When $A \neq B$, each pair in "Ordered Divisor Pair" list as (B, A) again.

Once we double the number, our result contains the number of ordered pair (A, B) such tha $A \times B \leq N$.

3. Add the number of pairs (A, B) where $A = B$
We still did not count pairs where $A = B$ and $A \times B \leq N$. How many pairs can there be? S between 1 and u, there cannot be more than u such pairs. The pairs will be $(1, 1), (2, 2) \ldots$

Code for $O(\sqrt{N})$ Approach

```
int SNOD( int n ) {
    int res = 0;
    int u = sqrt(n);
    for ( int i = 1; i <= u; i++ ) {
        res += ( n / i ) - i; //Step 1
    }
    res *= 2; //Step 2
    res += u; //Step 3
    return res;
}
```

Reference

1. forthright48 - Number of Divisors of an Integer - http://forthright48.blogspot.com/2015/07 divisors-of-integer.html
2. Wiki - Divisor Summatory Function - https://en.wikipedia.org/wiki/Divisor_summatory_fur

Posted by Mohammad Samiul Islam

G+1 Recommend this on Google

Labels: Divisors, Factorization, Number Theory

No comments:

Post a Comment

Leave comments for Queries, Bugs and Hugs.

forthright48

Learning Never Ends

Saturday, July 25, 2015

Introduction to Modular Arithmetic

> "In mathematics, modular arithmetic is a system of arithmetic for integers, where numbe around" upon reaching a certain value—the modulus." - Wiki

The clock is a good example for modular arithmetic. Let's say 12'o clock means 0. Then clock following values, $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 0, 1, 2, 3...\}$. Every time clock hits 12, it w Since clock wraps around 12, we say that 12 is the Modulus.

General Concepts

In order to understand modular arithmetic, we need to know the following:

Modulo Operation $\%$

Modular arithmetic deals with Modulo operation. It has the symbol, "$\%$". The expression $A \%$ that when we divide A with B we get the remainder C. In other words, the modulo operation f remainder. For example:

$5 \% 3 = 2$
$15 \% 5 = 0$
$24 \% 30 = 24$
$30 \% 24 = 6$
$4 \% 4 = 0$

Congruence Relation $\equiv$

In mathematics, saying that $A \equiv B \pmod{M}$ means that both A and B has same remainc divided by M. For example,

$5 \equiv 10 \pmod{5}$
$2 \equiv 7 \pmod{5}$
$4 \equiv 10 \pmod{3}$

Range

In modular arithmetic with modulus M, we only work with values ranging from 0 to $M - 1$. All be converted to the corresponding value from the range using Modulo operation. For exampl

Normal Arithmetic	-3	-2	-1	0	1	2	3
Modular Arithmetic	0	1	2	0	1	2	0

Properties of Modular Arithmetic

There are four properties of Modular Arithmetic that we need to learn.

Addition

$$\begin{aligned} &\text{if}, a \equiv b \,(\text{mod m}) \\ &\text{and}, c \equiv d \,(\text{mod m}) \\ &\text{then}, a + c \equiv b + d (\text{mod m}) \end{aligned}$$

Meaning, if we have to find $(a_1 + a_2 + a_3 \ldots + a_n) \,\%\, m$, we can instead just find $((a_1 \,\%\, m) + (a_2 \,\%\, m) + \ldots (a_n \,\%\, m)) \,\%\, m$.

For example, $(13 + 24 + 44) \,\%\, 3 = (1 + 0 + 2) \,\%\, 3 = 3 \,\%\, 3 = 0$.

Subtraction

$$\begin{aligned} &\text{if}, a \equiv b \,(\text{mod m}) \\ &\text{and}, c \equiv d \,(\text{mod m}) \\ &\text{then}, a - c \equiv b - d (\text{mod m}) \end{aligned}$$

Meaning, if we have to find $(a_1 - a_2 - a_3 \ldots - a_n) \,\%\, m$, we can instead just find $((a_1 \,\%\, m) - (a_2 \,\%\, m) - \ldots (a_n \,\%\, m)) \,\%\, m$.

For example, $(-14 - 24 - 44) \,\%\, 3 = (-2 - 0 - 2) \,\%\, 3 = -4 \,\%\, 3 = -1 \,\%\, 3 = 2$.

Multiplication

$$\begin{aligned} &\text{if}, a \equiv b \,(\text{mod m}) \\ &\text{and}, c \equiv d \,(\text{mod m}) \\ &\text{then}, a \times c \equiv b \times d \,(\text{mod m}) \end{aligned}$$

Meaning, if we have to find $(a_1 \times a_2 \times a_3 \ldots \times a_n) \,\%\, m$, we can instead just find $((a_1 \,\%\, m) \times (a_2 \,\%\, m) \times \ldots (a_n \,\%\, m)) \,\%\, m$.

For example, $(14 \times 25 \times 44) \,\%\, 3 = (2 \times 1 \times 2) \,\%\, 3 = 4 \,\%\, 3 = 1 \,\%\, 3 = 2$.

Division

Modular Division does not work same as the above three. We have to look into a separate to
Modular Inverse in order to find $\frac{a}{b} \,\%\, m$. Modular Inverse will be covered in a separate post.

For now just know that, $\frac{a}{b} \not\equiv \frac{a \% m}{b \% m} \pmod{m}$. For example, $\frac{6}{3} \% 3$ should be 2. But using t we get $\frac{0}{0} \% 3$ which is undefined.

Coding Tips

Handling Negative Numbers

In some programming language, when modulo operation is performed on negative numbers negative. For example in C++ we will get $-5 \% 4 = -1$. But we need to convert this into lega convert the result into legal value by adding M.

Therefore, in C++ it is better to keep a check for negative number when doing modulo operati

```
res = a % m;
if ( res < 0 ) res += m;
```

Modulo Operation is Expensive

Modulo Operation is as expensive as division operator. It takes more time to perform division operations than to perform addition and subtraction. So it is better to avoid using modulo oper possible.

One possible situation is when we are adding lots of values. Let's say that we have two variab between 0 to $m - 1$. Then we can write:

```
res = a + b;
if ( res >= m ) res -= m;
```

This technique will not work when we are multiplying two values.

Resource

1. Wiki - Modular Arithmetic - https://en.wikipedia.org/wiki/Modular_arithmetic
2. AoPS - Introduction to Modular Arithmetic - http://www.artofproblemsolving.com/wiki/index.php/Modular_arithmetic/Introduction

Posted by Mohammad Samiul Islam

G+1 Recommend this on Google

Labels: Modular Arithmetic, Number Theory

No comments:

Post a Comment

Leave comments for Queries, Bugs and Hugs.

forthright48

Learning Never Ends

Sunday, July 26, 2015

Extended Euclidean Algorithm

Extended Euclidean Algorithm is an extension of Euclidean Algorithm which finds two things fo

1. It finds the value of $GCD(a, b)$
2. It finds two integers x and y such that, $ax + by = gcd(a, b)$.

The expression $ax + by = gcd(a, b)$ is known as Bezout's identity and the pair (x, y) that s is called Bezout coefficients. We will look into Bezout's identity at the end of this post. For now name.

How It Works

In Euclidean Algorithm we worked with remainders only. In Extended Euclidean Algorithm (ext the quotient and few other extra variables. Suppose we are finding $GCD(240, 46)$. Using Eu the steps for finding GCD would be:

$GCD(240, 46) = GCD(46, 240 \% 46) = GCD(46, 10)$
$GCD(46, 10) = GCD(10, 46 \% 10) = GCD(10, 6)$
$GCD(10, 6) = GCD(6, 10 \% 6) = GCD(6, 4)$
$GCD(6, 4) = GCD(4, 6 \% 4) = GCD(4, 2)$
$GCD(4, 2) = GCD(2, 4 \% 2) = GCD(2, 0)$
$GCD(2, 0) = 2$

Introducing r and q

We will slowly move towards ext_gcd algorithm. Let us add two new variables. r represents re means quotient.

Let r_0 be a and r_1 be b. At each step, we will calculate r_i. Let $q_i = \lfloor \frac{r_{i-2}}{r_{i-1}} \rfloor$. Therefore, $r_i = r$

Then the above steps will look like the following:

Index i	Quotient q_i	Remainder r_i

0		240
1		46
2	$240/46 = 5$	$240 - 5 \times 46 = 10$
3	$46/10 = 4$	$46 - 4 \times 10 = 6$
4	$10/6 = 1$	$10 - 1 \times 6 = 4$
5	$6/4 = 1$	$6 - 1 \times 4 = 2$
6	$4/2 = 2$	$4 - 2 \times 2 = 0$

This table is same as the calculation for Euclidean algorithm except for few extra details. Note before last (index 5) contains the $GCD(a, b)$.

Introducing x and y

We are trying to express $GCD(a, b) = ax + by$. So the variable x and y will hold the coeffic exact we will write each row as terms of a and b, i.e, $r_i = ax_i + by_i$.

Initially, $(x_0, y_0) = (1, 0)$ and $(x_1, y_1) = (0, 1)$. But how do we calculate (x_i, y_i)?

We know that $r_i = r_{i-2} - q_i \times r_{i-1}$. We also claimed that $r_i = ax_i + by_i$. By combining th

$r_i = (ax_{i-2} + by_{i-2}) - q_i \times (ax_{i-1} + by_{i-1})$
$r_i = ax_{i-2} - q_i \times ax_{i-1} + by_{i-2} - q_i \times by_{i-1}$
$r_i = a(x_{i-2} - q_i \times x_{i-1}) + b(y_{i-2} - q_i \times y_{i-1})$
$r_i = ax_i + by_i$
$\therefore x_i = x_{i-2} - q_i \times x_{i-1}$ and $y_i = y_{i-2} - q_i \times y_{i-1}$.

Our new table enhanced with x and y will now look like:

Index i	Quotient q_i	Remainder r_i	x_i	y_i
0		240	1	0
1		46	0	1
2	$240/46 = 5$	$240 - 5 \times 46 = 10$	$1 - 5 \times 0 = 1$	$0 - 5 \times 1 = -$

3	$46/10 = 4$	$46 - 4 \times 10 = 6$	$0 - 4 \times 1 = -4$	$1 - 4 \times -5 = 2$
4	$10/6 = 1$	$10 - 1 \times 6 = 4$	$1 - 1 \times -4 = 5$	$-5 - 1 \times 21 = -$
5	$6/4 = 1$	$6 - 1 \times 4 = 2$	$-4 - 1 \times 5 = -9$	$21 - 1 \times -26 =$
6	$4/2 = 2$	$4 - 2 \times 2 = 0$	$5 - 2 \times -9 = 23$	$-26 - 2 \times 47 = -$

Our answer lies on the line before last. $240 \times -9 + 46 \times 47 = 2$.

So all we need to do now is implement these steps in code.

Code

Even though we will be calculating many rows in ext_gcd algorithm, in order to calculate any r information from previous two rows. So we can save memory by simply storing $r_{i-2}, r_{i-1}, x_{i-2}, x_{i-1}, y_{i-2}, y_{i-1}$.

In our code, $x2$ is x_{i-2}, $x1$ is x_{i-1} and x is x_i. Same style is followed for other variable.

```
1  int ext_gcd ( int A, int B, int *X, int *Y ){
2      int x2, y2, x1, y1, x, y, r2, r1, q, r;
3      x2 = 1; y2 = 0;
4      x1 = 0; y1 = 1;
5      for (r2 = A, r1 = B; r1 != 0; r2 = r1, r1 = r, x2 = x1, y2 = y1,
6          q = r2 / r1;
7          r = r2 % r1;
8          x = x2 - (q * x1);
9          y = y2 - (q * y1);
10     }
11     *X = x2; *Y = y2;
12     return r2;
13 }
```

In line 1 we define the function. The function $ext\_gcd()$ takes 4 parameters. The first two par represents the two integers whose gcd we wish to find. The next two parameter $*X$ and $*Y$ a pointers. Our function returns us $GCD(A, B)$ but we also want to find the coefficients x, y su $ax + by = GCD(A, B)$. So we extract those values using pointers.

In line 2, we declare all necessary variables. We initiate some variables in line 3 and 4. Then a line 5. We initiate few more variables in first section, $r2$ and $r1$. The loop runs till $r1$ becomes section of for loop, we make transitions of variables, such as $r2$ becomes $r1$ and $r1$ gets the

In side the for loop we calculate values needed for current row, q, r, x, y.

When the loop finishes, $r1$ contains 0 and $r2$ is the row before it containing $GCD(A, B)$. So as coefficients.

Complexity

Same as Euclidean Algorithm.

Uniqueness of Solution

Using ext_gcd we found (x, y) pair which satisfies $ax + by = gcd(a, b)$. But is it unique?

No. Once we find a pair (x, y) using ext_gcd, we can generate infinite pairs of Bezout coeffici formula:

$$(x + k\frac{b}{\gcd(a,b)}, y - k\frac{a}{\gcd(a,b)})$$

Using any integer value for k we can generate a different pair of Bezout coefficients (x, y) wh the Bezout's identity. Here is why it works:

$a(x + \frac{kb}{\gcd(a,b)}) + b(y - \frac{ka}{\gcd(a,b)})$
$ax + \frac{kab}{\gcd(a,b)} + by - \frac{kab}{\gcd(a,b)}$
$ax + by$

As you can see above, the terms with k in them cancel each other out. They don't change the $ax + by$ in any way, therefore, there are infinite pairs of Bezout coefficients.

Smallest Positive Integer of form $ax + by$

We showed that it is possible to write $ax + by = gcd(a, b)$, now we need to find a positive nu than $gcd(a, b)$ of form $ax + by$. Is that even possible?

No. $gcd(a, b)$ divides both a and b. Therefore, if a number is of form $ax + by$ it will be divisibl since $ax + by$ is divisible by $gcd(a, b)$. And the smallest positive number which is divisible by $gcd(a, b)$ itself.

So $gcd(a, b)$ is the smallest positive number of the form $ax + by$.

Bezout's Identity

This was mentioned at the beginning of the post. Almost everything related to Bezout's Identit explained. I will still mention them once more for the sake of completeness.

> Bézout's identity (also called Bézout's lemma) is a theorem in the elementary theory of r a and b be nonzero integers and let d be their greatest common divisor. Then there exis and y such that $ax + by = d$
>
> In addition,
>
> - the greatest common divisor d is the smallest positive integer that can be written a $ax + by$
> - every integer of the form ax + by is a multiple of the greatest common divisor d.
>
> -Wiki

Bezout's Lemma simply states that $ax + by = gcd(a, b)$ exists. We need to use Extended E Algorithm to find Bezout's Coefficients.

Coding Pitfalls

Careful about the equation $ax + by = \gcd(a, b)$. Here $\gcd()$ means the result from Euclidea what we mean mathematically. For example, when $a = 4$ and $b = -2$, Extended Euclidean A solution for $4x - 2y = -2$. According to Euclidean algorithm $gcd(4, -2)$ returns -2, thoug sense it should be 2.

Reference

1. forthright48 - Euclidean Algorithm - http://forthright48.blogspot.com/2015/07/euclidean-a greatest.html
2. Wiki - Extended Euclidean algorithm - https://en.wikipedia.org/wiki/Extended_Euclidean_
3. Wiki - Bezout's identity - https://en.wikipedia.org/wiki/B%C3%A9zout%27s_identity

Posted by Mohammad Samiul Islam

Labels: Number Theory

G+1 Recommend this on Google

1 comment:

Zeron May 23, 2016 at 11:27 AM

Good explanatory resource for extended euclid is hard to come by. Thanks :)

Reply

Enter your comment...

Comment as: Unknown (Goo

Publish Preview

Leave comments for Queries, Bugs and Hugs.

Newer Post Home

Subscribe to: Post Comments (Atom)

Blog Archive

▼ 2015 (35)
► Sep (7)

Labels

Analysis Arithmetic Function Backtrack Bi Complexity Contest D&C Divisors Factorial

1430. Crime and Punishment

Time limit: 0.5 second
Memory limit: 64 MB

Background

Petty bureaucrat Victor Thiefton was disposed towards stealing from his childhood. But one thing is to legally privatize national factories, diamond fields and oil derricks at the cost of billions dollars. And another thing is to filch some money from a poor regional budget. Our legislation is very strict. Therefore Victor felt that justice is on the alert just after he extracted his hand from the national pocket. What should he do to escape inevitable punishment?

Mr. Thiefton has once heard that in accordance with the criminal legislation standards he would be condemned to long imprisonment for a theft whereas in case of a peculation he could escape with a suspended sentence only. So if the most part of stolen money is peculated, the duration of imprisonment will be reduced.

Problem

The same evening Mr. Thiefton burst into "MegaApril" superstore and rushed for overflowing storefronts carrying a purse with N stolen dollars. It appeared that unlimited number of high-quality goods and goods at moderate price were on sale in the superstore. High-quality goods cost A dollars per piece, and goods at moderate price cost B dollars per piece. Victor should spend as much stolen money as possible to reduce the duration of imprisonment to a minimum.

Input

The only line contains the integer numbers A, B and N ($1 \le A, B, N \le 2 \cdot 10^9$).

Output

You should output the number of high-quality goods and the number of goods at moderate price, which should be bought to guarantee the minimal duration of imprisonment for Victor. The numbers should be separated by single space. If the problem has several solutions, you should output any of them.

Sample

input	output
8 5 22	2 1

Problem Author: Nikita Rybak, Ilya Grebnov, Dmitry Kovalioff
Problem Source: Timus Top Coders: First Challenge
Tags: number theory (hide tags for unsolved problems)

Difficulty: 496 Printable version Submit solution Discussion (23)
All submissions (7525) All accepted submissions (1043) Solutions rating (792)

© 2000–2016 Timus Online Judge Team. All rights reserved.

TIMUS 1430:

This above problem basically gives you the values of a,b,c.

They want you to find out the value of x,y where

ax + by = c and (x + y) is the maximum.

If for c it cannot be satisfied… find x,y to satisfy the closes value strictly less than c.

If a = 8, b = 5, c = 22 then, for x = 2, y = 1, we find 8*2 + 5*1 = 21, which is the closest possible value to c = 22.

The constraint also is so (x, y) >= 0.

To solve this the first approach that would come to our mind would be Extended euclid's theorem. But guess what? EGCD returns negative value if it needs.

So, we actually cannot always solve this using EGCD. The better idea here is to search linearly for one value.

First see the following code. Then we explain.

```
#include<bits/stdc++.h>
#define LL long long
using namespace std;

const int INF=0x3f3f3f3f;

int main()
{
    //freopen("data.txt","r",stdin);
    LL a,b,n;
    while(cin>>a>>b>>n)
    {
        bool change=false;
        if(a<b)
        {
           change=true; ///WE KEEP A MARKER TO TRACE THE SWAP
           swap(a,b); ///THIS MEANS THAT a,b was swapped so a had the minimum
         }
```

```
        ///We’re searching linearly upto sqrt(n) at most [when a=b]
        ///For every value from i = 0 to sqrt(n); we’ll assign x = i,
        ///Then we’ll find the y corresponding to that x… and we’ll see
        ///the value of (c’ = ax + by) for those value of x and y
        ///the max value of c’ will always be less or equal to c because
        ///we found y using the equation
        ///if z = by then, ax + z = c
        ///so z = c-ax
        ///so y = floor(z/b)
        ///hence y*b <= z
        ///hence ax + by <= c
```

```
            LL x,y,MAX=-1;
            for(LL i=0;i<=min(n/a,b);++i)
            {
               LL j=(n-i*a)/b;
               if(i*a+j*b>MAX)
               {
                   MAX=i*a+j*b;
                   x=i;
                   y=j;
               }
           }
```

```
    //If we had changed a and b, then their partner x and y are to be swapped as well
```

```
              if(change)
              swap(k1,k2);
        }
        return 0;
    }
```

forthright48

Learning Never Ends

Monday, July 27, 2015

Linear Diophantine Equation

Problem

Given the value of integers A, B and C find a pair of integers (x, y) such that it satisfies the e
$Ax + By = C$.

For example, if $A = 2, B = 3$ and $C = 7$, then possible solution of (x, y) for equation $2x +$
$(2, 1)$ or $(5, -1)$.

The problem above is a type of Diophantine problem. In the Diophantine problem, only intege
equation are required. Since $Ax + By = C$ is a linear equation, this problem is a **Linear Dio
Problem** where we have to find a solution for a **Linear Diophantine Equation**.

For now, let us assume that A and B are non-zero integers.

Existence of Solution

Before we jump in to find the solution for the equation, we need to determine whether it even h
example, is there any solution for $2x + 2y = 3$? On the left side we have $2x + 2y$ which is e
what integer value of (x, y) is used and on the right side we have 3 which is odd. This equati
satisfy using integer values.

So how do we determine if the equation has a solution? Suppose $g = gcd(A, B)$. Then Ax
of g. In order to have a valid solution, since left side of the equation is divisible by g, the right s
divisible by g. Therefore, if $g \not| C$, then there is no solution.

Simplifying the Equation

Since both side of equation is divisible by g, i.e, $g \mid \{ (Ax + By), C \}$, we can safely divide b
resulting in a equivalent equation.

Let $a = \frac{A}{g}$, $b = \frac{B}{g}$ and $c = \frac{C}{g}$. Then,

$$(Ax + By = C) \equiv (ax + by = c)$$

After simplification, $gcd(a, b)$ is either 1 or -1. If it is -1, then we need multiply -1 with a, b a $gcd(a, b)$ becomes 1 and the equation remains unchanged. Why did we make the $gcd(a, b)$ find the reason below.

Using Extended Euclidean Algorithm

Recall that in a previous post "Extended Euclidean Algorithm", we learned how to solve the Be $Ax + By = gcd(A, B)$. Can we apply that here in any way?

Yes. Using $\text{ext\_gcd}()$ function, we can find Bezout's coefficient for $ax + by = gcd(a, b)$. Bu solution for $ax + by = c$. Note that $gcd(a, b) = 1$, so when we use $\text{ext\_gcd}()$ we find a solu $ax + by = 1$. Let this solution be (x_1, y_1). We can extend this solution to solve our original p

Since we already have a solution where $ax_1 + by_1 = 1$, multiplying both sides with c gives u $a(x_1c) + b(y_1c) = c$. So our result is $(x, y) = (x_1c, y_1c)$. This is why we had to make sure was 1 and not -1. Otherwise, multiplying c would have resulted $ax + by = -c$ instead.

Summary of Solution

Here is a quick summary of what I described above. We can find solution for Linear Diophanti $Ax + By = C$ in 3 steps:

1. **No Solution**
   First check if solution exists for given equation. Let $g = gcd(A, B)$. If $g \nmid C$ then no sol

2. **Simplify Equation**
   Let $a = \frac{A}{g}, b = \frac{B}{g}$ and $c = \frac{C}{g}$. Then finding solution for $Ax + By = C$ is same as fin $ax + by = c$. In simplified equation, make sure $GCD(a, b)$ is 1. If not, multiply -1 with

3. **Extended Euclidean Algorithm**
   Use $\text{ext\_gcd}()$ to find solution (x_1, y_1) for $ax + by = 1$. Then multiply the solution with for $ax + by = c$, where $x = x_1 \times c, y = y_1 \times c$.

Let us try few examples.

Example 1: $2x + 3y = 7$

Step 1: $g = GCD(2, 3) = 1$. Since 1 divides 7, solution exists.
Step 2: Since g is already 1 there is nothing to simplify.
Step 3: Using $\text{ext\_gcd}()$ we get $(x, y) = (-1, 1)$. But this is for $ax + by = 1$. We need to r solution is $(-7, 7)$.

$2 \times -7 + 3 \times 7 = -14 + 21 = 7$. The solution is correct.

Example 2: $4x + 10y = 8$

Step 1: $g = GCD(4, 10) = 2$. Since 2 divides 8, solution exists.
Step 2: $a = \frac{4}{2}, b = \frac{10}{2}, c = \frac{8}{2}$. We will find solution of $2x + 5y = 4$.
Step 3: Using $\text{ext\_gcd}()$ we get $(x, y) = (-2, 1)$. But this is for $ax + by = 1$. We need to n solution is $(-8, 4)$.

$ax + by = 2 \times -8 + 5 \times 4 = -16 + 20 = 4 = c$.
Also, $Ax + By = 4 \times -8 + 10 \times 4 = -32 + 40 = 8 = C$. The solution is correct. Both a
$Ax + By = C$ are satisfied.

Finding More Solutions

We can now find a possible solution for $Ax + By = C$, but what if we want to find more? Ho
are there? Since the solution for $Ax + By = C$ is derived from Bezout's Identity, there are in

Suppose we found a solution (x, y) for $Ax + By = C$. Then we can find more solutions usi
$(x + k\frac{B}{g}, y - k\frac{A}{g})$, where k is any integer.

Code

Let us convert our idea into code.

```
1  bool linearDiophantine ( int A, int B, int C, int *x, int *y ) {
2      int g = gcd ( A, B );
3      if ( C % g != 0 ) return false; //No Solution
4
5      int a = A / g, b = B / g, c = C / g;
6      ext_gcd( a, b, x, y ); //Solve ax + by = 1
7
8      if ( g < 0 ) { //Make Sure gcd(a,b) = 1
9          a *= -1; b *= -1; c *= -1;
10     }
11
12     *x *= c; *y *= c; //ax + by = c
13     return true; //Solution Exists
14 }
15
16 int main () {
17     int x, y, A = 2, B = 3, C = 5;
18     bool res = linearDiophantine ( A, B, C, &x, &y );
19
20     if ( res == false ) printf ( "No Solution\n" );
21     else {
22         printf ( "One Possible Solution (%d %d) \n", x, y );
23
24         int g = gcd ( A, B );
25
26         int k = 1; //Use different value of k to get different soluti
27         printf ( "Another Possible Solution (%d %d)\n", x + k * ( B /
28     }
29
30     return 0;
31 }
```

$\text{linearDiophantine}()$ function finds a possible solution for equation $Ax + By = C$. It takes
A, B, C defines the coefficients of equation and $*x, *y$ are two pointers that will carry our sol
function will return $true$ if solution exists and $false$ if not.

In line 2 we calculate $gcd(A, B)$ and in line 3 we check if C is divisible by g or not. If not, we r

Next on line 5 we define a, b, c for simplified equation. On line 6 we solve for $ax + by = 1$ us
Then we check if $g < 0$. If so, we multiply -1 with a, b, c to make it positive. Then we multiply
our solution satisfies $ax + by = c$. A solution is found so we return true.

In $\text{main}()$ function, we call $\text{linearDiophantine}()$ using $A = 2, B = 3, C = 5$. In line 22 we solution. In line 27 we print another possible solution using formula for more solutions.

A and B with Value 0

Till now we assumed $\{A, B\}$ have non-zero values. What happens if they have value 0?

When Both $A = B = 0$

When both A and B are zero, the value of $Ax + By$ will always be 0. Therefore, if $C \neq 0$ th solution. Otherwise, any pair of value for (x, y) will act as a solution for the equation.

When A or B is 0

Suppose only A is 0. Then equation $Ax + By = C$ becomes $0x + By = C \equiv By = C$. If B does not divide C then there is no solution. Else solution will be $(x, y) = (\frac{C}{B}, k)$, where k

Using same logic, when B is 0, solution will be $(x, y) = (k, \frac{C}{A})$.

Coding Pitfalls

When we use $gcd(a, b)$ in our code, we mean the result from Euclidean Algorithm, not what w mathematically. $gcd(4, -2)$ is -2 according to Euclidean Algorithm whereas it is 2 in commo

Resources

1. Wiki - Diophantine Equation - https://en.wikipedia.org/wiki/Diophantine_equation
2. forthright48 - Extended Euclidean Algorithm - http://forthright48.blogspot.com/2015/07/e euclidean-algorithm.html

Posted by Mohammad Samiul Islam

G+1 Recommend this on Google

Labels: Number Theory

No comments:

Post a Comment

Leave comments for Queries, Bugs and Hugs.

You are not logged in. Please login at www.codechef.com to post your questions!

algorithm of pouring water.

http://www.codechef.com/problems/POUR1

0

this is a question i found while solving problems from pratice arena on codechef. I saw many solutions but could not understand the logic. On some of the posts i found that DIOPHANTUS has given the solution for this, but could not get any relevant material. Further, the editorial of this question is not provided. please help me in understanding this problem.

diophantus

asked **30 Aug '13, 14:01**
iitr_sonu
87●6●8●10
accept rate: 0%

Follow this question
By Email:
Once you sign in you will be able t
subscribe for any updates here
By RSS:
Answers
Answers and Comments

Tags:
diophantus ×1

Asked: **30 Aug '13, 14:01**
Seen: **8,090 times**
Last updated: **04 Feb, 21:09**

Related questions

6 Answers: oldest newest most voted

6

The water jug problem can be solved using the extended-eucildean algorithm. Extended-euclidean algorithm finds solution for diophantine equations. How does finding solution of diophantine equation solves the water jug problem? Let me demonstrate:

Imagine you have a jug of 5 liters (A) and 3 liters (B). You want to make 1 liter of water with this 2 container. The the equation will be, 5x + 3y = 1 (Ax + By = 1). If we can find a solution this equation then our problem is solved. Apply extended_euclidean algorithm on it and you will find that the result is x = 2 and y = -3.

If we put value of x and y in the equation then we get 5 * 2 + 3 * (-3) = 10 - 9 = 1. The equation is indeed solved. But what does x = 2 and y = -3 mean? It means, we need to fill our A bottle 2 times and empty our B bottle 3 times.

This how it will go:

```
A               B
5 (fill Once )
2               3 ( Transfer 3 liter )
2               0 ( Empty Once )
0               2 ( Transfer 2 liter )
5 (fill Twice )  2
4               3 ( Transfer 1 liter )
4               0 ( Empty Twice )
1               3 ( Transfer 3 liter )
1               0 ( Empty Thrice )
```

So now that you know the meaning behind the solution, how do you find the number of steps? I guess simulate. You know which bottle you need to fill and which to empty, so do it. Just think about it for a while, it should become clear.

There are other solution to water jug problem, such as BFS and DP. You can give those a try too if you want.

Read this article (http://e-maxx.ru/algo/extended_euclid_algorithm to learn how to use extended euclidean algorithm.

link

answered **01 Sep '13, 02:50**
forthright
525●4●12●17
accept rate: 10%

Can you please elaborate the advantage of using Extended Euclidean method instead of the Euclidean GCD method here

nickzuck_007 (17 May '15, 23:53)

0

see this SO link...hope this will help..:)

link

answered **30 Aug '13, 19:44**
kunal361
5.9k●13●32●72
accept rate: 21%

0

The link is not working foe me ??

link

answered **31 Aug '13, 18:07**
majedatwi11


1●1●1●1
accept rate: 0%

0

Hello everyone , I implemented Extended Euclidean Algorithm . But i am not getting how to solve this http://www.codechef.com/problems/POUR1 problem with the help of Extended Euclidean Algorithm.

I got the point explained by @forthright , that we have to find the solution of ax + by = 1. Is it same for this problem i mean do we need to find the value of x and y for ax + by = c ?

Please anybody explain how to solve this problem using Extended Euclidean Algorithm .

Happy Coding!!!

link

answered **22 Sep '13, 00:49**


upendra1234
2.3k●18●30●69
accept rate: 1%

0

can this problem be solved using bfs ????

link

answered **05 Jun '14, 13:06**


codersumit
1●1
accept rate: 0%

Yes of course, this link may help you :

http://stackoverflow.com/questions/17869493/advice-for-pour1-on-spoj

achaitanyasai (05 Jun '14, 14:43)

0

You may find the proof of Arithmetic solution for this problem here :
http://www.iaeng.org/publication/WCE2013/WCE2013_pp145-147.pdf

link

answered **04 Feb, 21:09**


matrixneo18
26●2
accept rate: 0%

Post Your Answer

© 2009, Directi Group. All Rights Reserved.
Powered by OSQA

forthright48

Learning Never Ends

Wednesday, July 29, 2015

Simple Hyperbolic Diophantine Equation

Problem

Given the integers A, B, C, D, find a pair of (x, y) such that it satisfies the equation $Axy +$

For example, $2xy + 5x + 56y = -7$ has a solution $(x, y) = (-27, 64)$.

Hyperbolic equations are usually of form $Ax^2 + By^2 + Cxy + Dx + Ey = F$. Here we do coefficients of x^2 and y^2. That's why we are calling it "Simple".

So how do we tackle the problem? First we will rewrite the equation to make it easier to approa

Rewriting The Equation

$Axy + Bx + Cy = D$
$A^2xy + ABx + ACy = AD$, Multiply A with both sides
$A^2xy + ABx + ACy + BC = AD + BC$, Add BC to both sides
$Ax(Ay + B) + C(Ay + B) = AD + BC$,
$\therefore (Ax + C)(Ay + B) = AD + BC$

Now that we have rewritten the equation, we can see that the problem has two cases. We will case separately.

When $AD + BC = 0$

When $AD + BC = 0$, our equation becomes $(Ax + C)(Ay + B) = 0$. So we need to ha $(Ax + C) = 0$ or $(Ay + B) = 0$ to satisfy the equation.

So we have two possible formation of solutions. When $(Ax + C) = 0$, we have $x = \frac{-C}{A}$ an value. When $(Ay + B) = 0$, we have $y = \frac{-B}{A}$ and x any integer value.

So, $(x, y) = (\frac{-C}{A}, k)$ or $(k, \frac{-B}{A})$, where k is any integer. What if it is not possible to divide $-$ Then there is no solution of that formation.

When $AD + BC \neq 0$

Let $P = AD + BC$. Since $(Ax + C)(Ay + B) = P$, $(Ax + C)$ and $(Ay + B)$ must be c
$d_1, d_2, d_3 \ldots d_n$ be divisors of P. Then for each divisor d_i, if $(Ax + C) = d_i$ then $(Ay + B$
we get P when we multiply them.

$(Ax + C) = d_i$
$Ax = d_i - C$
$\therefore x = \frac{d_i - C}{A}$

$(Ay + B) = \frac{P}{d_i}$
$Ay = \frac{P}{d_i} - B$
$Ay = \frac{P - Bd_i}{d_i}$
$\therefore y = \frac{P - Bd_i}{Ad_i}$

Therefore for each divisor d_i, we have $(x, y) = (\frac{d_i - C}{A}, \frac{P - Bd_i}{Ad_i})$. This solution is valid only if v
values for (x, y).

Careful when calculating divisors of P. In this problem, we have to include the negative diviso
calculation. For example, 4 will have the following divisors $(1, 2, 4, -1, -2, -4)$.

Example

Find solutions for $2xy + 2x + 2y = 4$.

First we will find $P = AD + BC = 2 \times 4 + 2 \times 2 = 12$. 12 has the following divisors:
$(\pm 1, \pm 2, \pm 3, \pm 4, \pm 6, \pm 12)$.

Since $(2x + 2)(2y + 2) = 12$, we get the following:

$d_1 = 1 : x = \frac{1-2}{2} = \frac{-1}{2}, y = \frac{12-2\times1}{2\times1} = \frac{10}{2} = 5$
$d_2 = -1 : x = \frac{-1-2}{2} = \frac{-3}{2}, y = \frac{12-2\times-1}{2\times-1} = \frac{14}{-2} = -7$
$d_3 = 2 : x = \frac{2-2}{2} = 0, y = \frac{12-2\times2}{2\times2} = \frac{8}{4} = 2$
$d_4 = -2 : x = \frac{-2-2}{2} = -2, y = \frac{12-2\times-2}{2\times-2} = \frac{16}{-4} = -4$
$d_5 = 3 : x = \frac{3-2}{2} = \frac{1}{2}, y = \frac{12-2\times3}{2\times3} = \frac{6}{6} = 1$
$d_6 = -3 : x = \frac{-3-2}{2} = \frac{-5}{2}, y = \frac{12-2\times-3}{2\times-3} = \frac{18}{-1} = -18$
$d_7 = 4 : x = \frac{4-2}{2} = 1, y = \frac{12-2\times4}{2\times4} = \frac{4}{8} = \frac{1}{2}$
$d_8 = -4 : x = \frac{-4-2}{2} = -3, y = \frac{12-2\times-4}{2\times-4} = \frac{20}{-8} = \frac{-5}{2}$
$d_9 = 6 : x = \frac{6-2}{2} = 2, y = \frac{12-2\times6}{2\times6} = \frac{0}{12} = 0$
$d_{10} = -6 : x = \frac{-6-2}{2} = -4, y = \frac{12-2\times-6}{2\times-6} = \frac{24}{-12} = -2$
$d_{11} = 12 : x = \frac{12-2}{2} = 5, y = \frac{12-2\times12}{2\times12} = \frac{-12}{24} = \frac{-1}{2}$
$d_{12} = -12 : x = \frac{-12-2}{2} = -7, y = \frac{12-2\times-12}{2\times-12} = \frac{36}{-24} = \frac{-3}{2}$

So the solutions are $(0, 2), (-2, -4), (2, 0), (-4, -2)$.

Code

Let us try to write a function that will count the number of solutions for the given problem abov
infinite solutions, the function will return -1.

```
1  bool isValidSolution ( int a, int b, int c, int p, int div ) {
2      if ( ( ( div - c )% a ) != 0 ) return false; //x = (div - c) / a
3      if ( ( (p-b*div) % (a*div) ) != 0 ) return false;// y = (p-b*div)
4      return true;
5  }
6
7  int hyperbolicDiophantine ( int a, int b, int c, int d ) {
8      int p = a * d + b * c;
9
10     if ( p == 0 ) { //ad + bc = 0
11         if ( -c % a == 0 ) return -1; //Infinite solutions (-c/a, k)
12         else if ( -b % a == 0 ) return -1; //Infinite solutions (k, -
13         else return 0; //No solution
14     }
15     else {
16         int res = 0;
17
18         //For each divisor of p
19         int sqrtn = sqrt ( p ), div;
20         for ( int i = 1; i <= sqrtn; i++ ) {
21             if ( p % i == 0 ) { //i is a divisor
22
23                 //Check if divisors i,-i,p/i,-p/i produces valid solu
24                 if ( isValidSolution(a,b,c,p,i) )res++;
25                 if ( isValidSolution(a,b,c,p,-i) )res++;
26                 if ( p / i != i ) { //Check whether p/i is different
27                     if ( isValidSolution(a,b,c,p,p/i) )res++;
28                     if ( isValidSolution(a,b,c,p,-p/i) )res++;
29                 }
30             }
31         }
32
33         return res;
34     }
35 }
```

We have two functions here. The first function $\text{isValidSolution}()$ takes input a, b, c, p, div a
whether we can get a valid solution (x, y) using div as a divisor of P.

In line 7 we start our function to find number of solutions. In line 10 We check our first case wh
$AD + BC = 0$. If it is possible to form a valid solution, we return -1 in line $11, 12$ otherwise

From line 15 we start case when $AD + BC \neq 0$. For this case we have to find all divisors of
all divisors come in pairs and one part of that pair lies before $\sqrt{P}$. We loop till $\sqrt{P}$ to find f
and process it in line 16. The second part can be found by dividing P by the first part. We pro
parts (if second part is different than first part, line 26) and their negative parts by passing the
$\text{isValidSolution}()$ functions. If they are valid we increase our result by 1.

If we are asked to print the solutions, we could do it by modifying the functions above. As long
understand how the solution works, there should not be any problem in modifying the functior

Resource

1. alpertron - Methods to Solve $Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$

forthright48

Learning Never Ends

Saturday, August 1, 2015

Introduction to Number Systems

Number System is a consistent way of expressing a number. It consists of three parts:

1. The Symbol - It can be digits, letters or any other symbol.
2. The Position - The position of symbols determine their weight.
3. The Base - The total number of different symbols supported.

Number system is often named after the size of its base. For example, the "Decimal Number S
symbols $(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)$. Since there are 10 different symbols, the base is 10.

The Decimal Number System is the standard number system that we use in our everyday life.
system as our reference for other systems. Since we will be looking into lots of different numb
article, from now on I will write numbers in decimal number system as $(number)_{10}$. In genera
is a number with base y.

How Number System Works

As mentioned before, Number System consists of three parts.

Symbols in Number System

In decimal number system, the first few numbers we encounter are $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$. T
symbols. We add a new position at the front of the number and start placing number in those
counting: $8, 9, 10, 11, 12, 13...$

The same concept is used with number system of different base. For example, let us consider
system with 3 different symbols: $(0, 1, 2)$. The first few numbers in this system are: $0, 1, 2$. On
symbols we introduce new positions and get: $10, 11, 12, 20, 21, 22, 100, 101, 102, 110$.

Position of Symbol

In a number, the position of symbols is 0 indexed and ranked from right to left. For example, th
will have the following index for its positions:

Index	4	3	2	1	0

Symbols	5	4	7	1	1

The last digit of the number has position index 0. The index increases as we go left.

The position in a number represents the weight of the symbol in that position. A symbol in a hi values more than one in a lower position. For example in the decimal number system, we kno if we consider 1 in a higher position then $100 > 20$. How is the weight of each position calcul answer is in next section.

Base of Number System

The "Base" of number system determine the weight of each position.

For example, $(1234)_{10} = 1000 + 200 + 30 + 4 = 1 \times 10^3 + 2 \times 10^2 + 3 \times 10^1 + 4 \times$

Here the weight of the symbol in 0_{th} position is 10^0, 1_{st} position is 10^1, 2_{nd} position is 10^2 an 10^3. As the position increases, the weight of position increases by 10. A symbol in n_{th} positio weight.

But this is true only for the decimal number system. The weight is a power of 10 for decimal sy a base of 10. A number system with base B will have weight B^n for n_{th} position.

Binary Number System

Binary Number System is a Number System with Base 2. It has only two symbols: $(0, 1)$. This is used by computers to store information. As programmers, we will frequently work with binar need to understand this system properly.

Since we are used to the Decimal Number System, it might be awkward to work with the binar But we will get used to it pretty soon.

Here are the first few numbers in Binary System.

Decimal	0	1	2	3	4	5	6	7	8	
Binary	0	1	10	11	100	101	110	111	1000	

Binary to Decimal

We will first look into how we can convert a number from binary to the decimal system.

Suppose we want to find the value of $(11001)_2$ in decimal. How do we find it? Remember tha a symbol in n_{th} position has $Base^n$ weight. What is the base in the binary system? 2. So this written as:

$(11001)_2 = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 16 + 8 + 0 + 0 + 1 = (25$

Decimal to Binary

Solve $(19)_{10} = (?)_2$. We need to convert 19 to binary.

A number x in binary system is of form: $x = s_n 2^n + \ldots + s_3 2^3 + s_2 2^2 + s_1 2^1 + s_0 2^0$, wher in i_{th} position. What happens when we find $x \% 2$? Since all position except for 0_{th} position i $x \% 2 = s_0$.

That is, finding the value of $x \% 2$ gives us the last digit of x in the binary system.

Okay, we got the digit in 0_{th} position, how do we find the rest? In order to find the digit in next divide x by 2. What happens if we divide x by 2?

$$\frac{x}{2} = s_n 2^{n-1} + \ldots + s_3 2^2 + s_2 2^1 + s_1 2^0 + \frac{s_0}{2}$$
$$\therefore \lfloor \frac{x}{2} \rfloor = s_n 2^{n-1} + \ldots + s_3 2^2 + s_2 2^1 + s_1 2^0$$

That is, when we divide x by 2, the last digit of x disappears and all digits of x shifts one plac example, if x is $(11011)_2$ then $\frac{x}{2}$ is $(1101)_2$.

In C++, we can easily perform this floor division by simply performing integer division. Once w find the value of $\lfloor \frac{x}{2} \rfloor \% 2$. By repeating this until x becomes 0, we can find all digits.

Let us try to solve the problem $(19)_{10} = (?)_2$.

$$x = 19 : x \% 2 = 1$$
$$x = \lfloor \frac{19}{2} \rfloor = 9 : x \% 2 = 1$$
$$x = \lfloor \frac{9}{2} \rfloor = 4 : x \% 2 = 0$$
$$x = \lfloor \frac{4}{2} \rfloor = 2 : x \% 2 = 0$$
$$x = \lfloor \frac{2}{2} \rfloor = 1 : x \% 2 = 1$$
$$x = \lfloor \frac{1}{2} \rfloor = 0 : \text{end}$$
$$\therefore (19)_{10} = (10011)_2$$

Hexadecimal Number System

Hexadecimal Number System is a Number System with Base 16. It has 16 symbols, $(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F)$. We already saw Decimal to Binary conversion Looking into Decimal-Hexadecimal conversion and vice verse will help us understand the bas better.

Here are the first few numbers in Hexadecimal System.

Decimal	0	1	2	3	4	5	6	7	8
Hexadecimal	0	1	2	3	4	5	6	7	8
Decimal	10	11	12	13	14	15	16	17	18

$Hexadecimal$	A	B	C	D	E	F	10	11	12

In hexadecimal system, (A, B, C, D, E, F) corresponds to the decimal value $(10, 11, 12, 1$

Hexadecimal to Decimal

Suppose we want to find the value of $(1A3F2B)_{16}$ in decimal. Using the same logic as Binar conversion, we can say that:

$$(1A3F2B)_{16} = 1 \times 16^5 + A \times 16^4 + 3 \times 16^3 + F \times 16^2 + 2 \times 16^1 + B \times 16^0$$
$$(1A3F2B)_{16} = 1 \times 16^5 + (10) \times 16^4 + 3 \times 16^3 + (15) \times 16^2 + 2 \times 16^1 + (11) \times 16$$
$$(1A3F2B)_{16} = 1048576 + 10 \times 65536 + 3 \times 4096 + 15 \times 256 + 2 \times 16 + 11 \times 1$$
$$(1A3F2B)_{16} = 1048576 + 655360 + 12288 + 3840 + 32 + 11$$
$$(1A3F2B)_{16} = (1720107)_{10}$$

Decimal to Hexadecimal

Solve $(123456789)_{10} = (?)_{16}$.

A number x in hexadecimal system is of form: $x = s_n 16^n + \ldots + s_3 16^3 + s_2 16^2 + s_1 16^1 +$ is the symbol in i_{th} position.

Just like Binary System, finding the value of $x \% 16$ gives us the last digit of x in hexadecima the rest, we divide it by 16 and find the last digit again.

$$\frac{x}{16} = s_n 16^{n-1} + \ldots + s_3 16^2 + s_2 16^1 + s_1 16^0 + \frac{s_0}{16}$$
$$\therefore \lfloor \frac{x}{16} \rfloor = s_n 16^{n-1} + \ldots + s_3 16^2 + s_2 16^1 + s_1 16^0$$

Let us try to solve the problem $(123456789)_{10} = (?)_{16}$.

$$x = 123456789 : x \% 16 = 5$$
$$x = \lfloor \frac{123456789}{16} \rfloor = 7716049 : x \% 16 = 1$$
$$x = \lfloor \frac{7716049}{16} \rfloor = 482253 : x \% 16 = 13$$
$$x = \lfloor \frac{482253}{16} \rfloor = 30140 : x \% 16 = 12$$
$$x = \lfloor \frac{30140}{16} \rfloor = 1883 : x \% 16 = 11$$
$$x = \lfloor \frac{1883}{16} \rfloor = 117 : x \% 16 = 5$$
$$x = \lfloor \frac{117}{16} \rfloor = 7 : x \% 16 = 7$$
$$x = \lfloor \frac{7}{16} \rfloor = 0 : \text{end}$$
$$\therefore (123456789)_{10} = (75BCD15)_{16}$$

Notice how we replaced the remainder value $11, 12, 13$ with B, C, D. Since the value $(10, 1$ represents (A, B, C, D, E, F) in hexadecimal, we use the proper symbols in the number.

Number System with Base B

We saw how to work with Binary and Hexadecimal number system. We will now generalize th

Suppose there is number x in base B, then $x = d_n d_{n-1} .. d_3 d_2 d_1 d_0$, where $0 \leq d_i < B$.

Base to Decimal

The number x in decimal will be $d_n \times B^n + d_{n-1} \times B^{n-1} + \ldots + d_1 \times B^1 + d_0 \times B^0$.

Here is a code that can convert a number in base B into decimal.

```
1  int baseToDecimal ( string x, int base ) {
2      int res = 0;
3      int len = x.length();
4
5      int coef = 1; ///initially base^0
6      for ( int i = len - 1; i >= 0; i-- ) { ///Start from reverse
7          res += (x[i]-'0') * coef;
8          coef *= base; //increase power of base
9      }
10     return res;
11 }
```

The number x here is a string. That is because x can have symbols that are not digits, for ex
. We find out the length of the number in line 3. Then we iterate over the number from last digi
We start from back where the coefficient of the digit is $base^0 = 1$. Every time we change posi
the coefficient by $base$. This ensures we multiply d_n with $base^n$.

If we want we can convert the number to decimal from the front of the number, that is from firs

Suppose we want to make the number $d_1 d_2 d_3$ in base B. We can start with the digit d_1 only.
move it to left of its position by multiply it by B, $d_1 B^0 \times B = d_1 B^1$. Then if we add d_2 with it,
$d_1 B^1 + d_2 = d_1 d_2$. We multiply it by B again and then add d_3. It finally becomes $d_1 d_2 d_3$.

For example, $(123)_{10} = (1 \times 10 + 2) \times 10 + 3$.

Using this idea, we can write the following code:

```
1  int baseToDecimalAlternate ( string x, int base ) {
2      int res = 0;
3      int len = x.length();
4
5      for ( int i = 0; i < len; i++ ) {
6          res = ( res * base ) + (x[i]-'0');
7      }
8      return res;
9  }
```

Here we iterate the number x from 0 to $len - 1$ in line 5. Line 6 is what I explained above.

The alternate method is shorter and sometimes makes a problem easier to solve. Knowing bo
us approach problems from different directions.

Decimal To Base

$x \% B$ will give us the last digit of x in base B. We can get the remaining digits by repeatedly
and taking its last digit until x becomes 0.

$x = x : d_0 = x \% B$
$x = \lfloor \frac{x}{B} \rfloor : d_1 = x \% B$
...

Here is how we will do it in code:

```
1  //A list of symbol. Depending on base and number system, this list ca
2  char symbol[] = {'0','1','2','3','4','5','6','7','8','9','A','B','C',
3  string decimalToBase ( int x, int base ) {
4      string res = "";
5
6      while ( x ) {
7          int r = x % base; //Find the last digit
8          res = res + symbol[r];//Change the integer value to symbol an
9          x /= base;//Remove the last digit
10     }
11     if ( res == "" ) res = symbol[0];//If res is empty, that means x
12     reverse ( res.begin(), res.end());//We found the digits in revers
13     return res;
14 }
```

In line 2 we have a symbol list. Since we are converting to a different base, it will have symbol
from 0 to $B-1$. We keep on finding the last digit of x until it becomes 0 in the loop at line 6.
found in line 7. We append it to the result in line 8 and we divide x by B in line 9. In line 11 we
empty. If it is then x was 0 from the beginning, so we append 0 to res. In line 12 we reverse t
res since we generated the digits in reverse order, i.e, from last digit to the first digit.

Resource

1. tibasicdev.wikidot - Binary, Hexadecimal and Octal number system

Posted by Mohammad Samiul Islam

G+1 Recommend this on Google

Labels: Binary, Number Theory

No comments:

Post a Comment

Leave comments for Queries, Bugs and Hugs.

forthright48

Learning Never Ends

Monday, August 3, 2015

UVa 11388 - GCD LCM

Problem

Problem Link - UVa 11388 - GCD LCM

Given two positive integers (G, L), we have to find a pair of integers (a, b) such that $gcd(a,$ $\text{lcm}(a, b) = L$. If there are multiple such pairs, we have to find the pair where a is minimum. needs to be positive.

Solution

We don't know the value of (a, b) yet. Here is how I approached the problem.

Value of a

What we know that $gcd(a, b) = G$. So, G divides a and b. Therefore, a is a multiple of G. W make sure that a is as small as possible. So, what is the smallest positive number that can be The answer is G itself.

$$\therefore a = G$$

Existence of Solution

Next, we know that $lcm(a, b)$ is the smallest positive number which is divisible by both a and and $a \mid lcm(a, b)$, it follows that $a = G$ should divide $lcm(a, b) = L$. If L is not divisible by solution exists.

$$\therefore \text{if } (G \not| L), \text{no solution exists}$$

Value of b

The value of b can be derived in the following way. We know that:

$gcd(a, b) \times lcm(a, b) = a \times b$
$G \times L = G \times b$
$b = \frac{G \times L}{G}$

$\therefore b = L.$

Summary

1. $a = G$
2. If $G \not| L$, no solution exists
3. $b = L$
4. $\therefore (a, b) = (G, L)$

Code

Here is the code in C++

```
#include <bits/stdc++.h>

int main () {
    int kase;
    scanf ( "%d", &kase ); //Input number of case

    while ( kase-- ) {
        int G, L;
        scanf ( "%d %d", &G, &L ); //Take input

        int a, b; //Need to find their value

        a = G;

        if ( L % G != 0 ) {
            printf ( "-1\n" ); //No Solution
            continue;
        }

        b = L;

        printf ( "%d %d\n", a, b );
    }

    return 0;
}
```

Posted by Mohammad Samiul Islam

G+1 Recommend this on Google

Labels: Analysis, GCD, LCM, Number Theory, UVa

No comments:

Post a Comment

Leave comments for Queries, Bugs and Hugs.

forthright48

Learning Never Ends

Tuesday, August 4, 2015

UVa 10407 - Simple division

Problem

Problem Link - UVa 10407 - Simple division

Given an array of numbers, find the largest number d such that, when elements of the array a
they leave the same remainder

Solution

We will be using our knowledge about Congruence Relation and GCD to solve this problem. E
to make sure that we understand the problem properly.

Wrong Approach

At first, one might think that d is the gcd of array elements. Surely, dividing elements of the arr
leave same remainder 0, but that doesn't necessarily mean it is the largest possible answer.

For example, suppose the array is $A = \{2, 8, 14, 26\}$. Then what will be the gcd of array A?
$gcd(2, 8, 14, 26) = 2$. Dividing elements of A with 2 leaves us 0. So is this our answer? No.

For array A, there exists a number greater than 2 that leaves the same remainder. And that n
we divide each element with 3, it leaves us 2 as the remainder.

The problem did not ask us to find the number that will leave 0 as remainder. It asked us to fin
number d that leaves the same remainder. So for array A the $gcd(A)$ is not the answer. Is 3
The answer is given below in Example.

Using Congruence Relation

Let us rephrase the problem using congruence relation. Suppose there is an array with eleme
$A = \{a, b, c\}$. We need to find the largest number d that leaves the same remainder for eac

Let us consider only $\{a, b\}$ for now. We need to find d such that it leaves the same remainde
and b. Meaning

$$a \equiv b \pmod{d}$$
$$a - b \equiv 0 \pmod{d}$$

What does this mean? It means, if d leaves the same remainder when it divides a and b, ther remainder when dividing $a - b$.

Using same logic, we can say that $b - c \equiv 0 \pmod{d}$. Therefore, if we find a number that c $b - c$, then that number will leave the same remainder when dividing $\{a, b, c\}$.

This idea can be extended for any number of elements in the array. So d is such a number th remainder when it divides the difference of adjacent elements in the array.

But wait, there are multiple numbers that can divide the difference of adjacent elements. Whic take? Since we want the largest value of d, we will take the largest divisor that can divide all th adjacent numbers. There is only one divisor that can divide all the adjacent differences, and tl $gcd((a - b), (b - c))$.

Therefore, if we have an array $A = \{a_1, a_2, a_3 \dots a_n\}$, then $d = gcd(a_2 - a_1, a_3 - a_2, \dots$

Careful about negative value of $gcd()$. Make sure you take the absolute value of $gcd()$.

Example

Let us get back to the example we were looking into before. Finding d for the array $A = \{2, 8$ know that d will divide the difference of adjacent elements completely. So let us make another contain the difference of adjacent elements., $B = \{8 - 2, 14 - 8, 26 - 14\} = \{6, 6, 12\}$. T $d = gcd(6, 6, 12) = 6$.

Summary

1. Let the array of elements be $A = \{a, b, c, d \dots\}$.
2. $res \neq gcd(a, b, c, d \dots)$.
3. $res = gcd(a - b, b - c, c - d, \dots)$.

Code

```
#include <bits/stdc++.h>
using namespace std;

typedef long long vlong;

vlong gcd ( vlong a, vlong b ) {
    while ( b ) {
        a = a % b;
        swap ( a, b );
    }
    return a;
}

vlong arr[1010];

int main () {

    while ( scanf ( "%d", &arr[0] ) != EOF ) {
```

```
19          if ( arr[0] == 0 ) break; //End of test case
20
21          //A new test case has started
22          int cur = 1;
23
24          //Take input
25          while ( 1 ) {
26              scanf ( "%lld", &arr[cur] );
27              if ( arr[cur] == 0 ) break;
28              else cur++;
29          }
30
31          vlong g = 0; //Start with 0 since gcd(0,x) = x.
32          for ( int i = 1; i < cur; i++ ) {
33              int dif = arr[i] - arr[i-1]; //Calculate difference
34              g = gcd ( g, dif ); //Find gcd() of differences
35          }
36
37          if ( g < 0 ) g *= -1; //In case gcd() comes out negative
38          printf ( "%lld\n", g );
39      }
40
41      return 0;
42  }
```

Posted by Mohammad Samiul Islam

G+1 Recommend this on Google

Labels: Analysis, GCD, Modular Arithmetic, Number Theory, UVa

No comments:

Post a Comment

Leave comments for Queries, Bugs and Hugs.

Enter your comment...

Comment as: Unknown (Goo ▾

Publish Preview

forthright48

Learning Never Ends

Saturday, August 8, 2015

Number of Digits of Factorial

Problem

Given an integer N, find number of digits in $N!$.

For example, for $N = 3$, number of digits in $N! = 3! = 3 \times 2 \times 1 = 6$ is 1. For $N = 5$, nun
$5! = 120$ is 3.

Brute Force Solution

The first solution that pops into mind is to calculate $N!$ and count how many digits it has. A po
look like the following:

```
int factorialDigit ( int n ) {
    long long fact = 1;
    for ( int i = 2; i <= n; i++ ) {
        fact *= i;
    }

    int res = 0; ///Number of digit of n!
    while ( fact ) { /// Loop until fact becomes 0
        res++;
        fact /= 10; ///Remove last digit
    }

    return res;
}
```

This code works, but only for $N \leq 20$. Once N crosses 20, it no longer fits in a "long long" va

Since factorial of $N > 20$ overflows $long\ long$, how about we use "Big Integer" to store the v

Brute Force Solution with Big Integer

If you don't know what Big Integer is, then know that it is a class for arbitrary large integer. C++
this class so we will have to manually implement it if we want to use C++ to solve problems inv
integers. Or you could use a programming language that supports this class, such as Java.

I will write a post on Big Integer someday, but for now you could just use the Big Int class writte Nekkanti from here.

Multiplication of a Big Integer and an integer takes $O(\text{number of digits of Big Integer})$. As when calculating $N!$, the number of digits of $N!$ increases by 1 at each step, it will take $O(N$ compute $N!$. But obviously $N!$ does not increase by 1 digit at each step (for e.g, multiply by by 2 digits), so worst time complexity is worse than $O(N^2)$.

Solution Using Logarithm

Logarithm of a number is connected to its number of digits, which might not be apparent. Wha Logarithm of a number x, in base b, is a real number y such that $x = b^y$. For example:

$$log_{10}1234 = 3.0913151597 \text{ and } 10^{3.0913151597} = 1234$$

In logarithms, base of the number is important. Since we want number of digits of $N!$ in decim with base 10.

Number of Digit of an Integer

First, we will apply the logarithm idea on an integer. So where is the connection of logarithm w Look at the following logarithm values:

$log_{10}(x) = y$
$log_{10}(1) = 0$
$log_{10}(10) = 1$
$log_{10}(100) = 2$
$log_{10}(1000) = 3$
$log_{10}(10000) = 4$

As the value of x increases, value of y also increases. Every time we multiple x by 10, value 1. That is, every time number of digit increases, value of y increases. From this table, we can about log of other values.

If $log_{10}(100)$ is 2, and $log_{10}(1000)$ is 3, then for all value of x where $100 < x < 1000$, valu between $2 < y < 3$. Let us try this out.

$log_{10}(100) = 2$
$log_{10}(150) = 2.17609125906$
$log_{10}(500) = 2.69897000434$
$log_{10}(999) = 2.99956548823$

Now note that, for every $100 \leq x < 1000$, value of y is $2 \leq y < 3$. Can you see some relati of y and number of digits of x?

Yes. If the value of y is of form $2.XXX$, then x has $2 + 1 = 3$ digits.

$$\therefore \text{number of digits of } x = \lfloor log_{10}(x) \rfloor + 1$$

Be careful, $\lfloor log_{10}(x) \rfloor + 1$ is not same as $\lfloor log_{10}(x) + 1 \rfloor$. Try it out with $100, 1000, 10000$. the log value **before** we add 1.

```
1 int numberDigit ( int n ) {
```

```
    int wrongAnswer = log10(n) + 1; ///This is wrong.
    int rightAnswer = ( (int) log10(n) ) + 1; ///This is right.
    return rightAnswer;
}
```

In line 3, we type cast $log10(n)$ to integer. This has same action as $floor()$ function. Also nc $log10()$ function instead of $log()$ function. Unlike our calculators, in C++ $log()$ has base 2.

Extending To Factorial

So how do we extend this idea to $N!$?

Let $x = log_{10}(N!)$. Then our answer will be $res = \lfloor x \rfloor + 1$. So all we need to do is find valu

$x = log_{10}(N!)$
$x = log_{10}(1 \times 2 \times 3 \times \ldots \times N)$
$\therefore x = log_{10}(1) + log_{10}(2) + log_{10}(3) + \ldots + log_{10}(N)$ This is using the law $log_{10}(ab) = log_{10}(a) + log_{10}(b)$

So in order to calculate $x = log_{10}(N!)$, we don't have to calculate value of $N!$. We can simp of all numbers from 1 to N. This can be achieved in $O(N)$.

```
int factorialDigit ( int n ) {
    double x = 0;
    for ( int i = 1; i <= n; i++ ) {
        x += log10 ( i );
    }
    int res = ( (int) x ) + 1;
    return res;
}
```

Digits of $N!$ in Different Base

Now what if we want to find how many digits $N!$ has if we convert $N!$ to some other base.

For example, how many digits $3!$ has in binary number system with base 2? We know that $(6$ $3!$ has 3 digits in base 2 number system.

Can we use logarithms to solve this problem too? Yes.

$$\text{number of digits of x in base B} = log_B(x)$$

All we need to do is change the base of our log and it will find number of digits in that base.

But, how do we change base in our code? We can only use log with base 2 and 10 in C++. F use the following law to change base of logartihm from B to C.

$$log_B(x) = \frac{log_C(x)}{log_C(B)}$$

So in C++, we will use $C = 2$ or $C = 10$ to find value of $log_B(x)$.

```
int factorialDigitExtended ( int n, int base ) {
    double x = 0;
    for ( int i = 1; i <= n; i++ ) {
        x += log10 ( i ) / log10(base); ///Base Conversion
```

```
    }
    int res = ( (int) x ) + 1;
    return res;
}
```

Posted by Mohammad Samiul Islam

G+1 Recommend this on Google

Labels: Big Int, Factorial, Logarithm, Number Theory

3 comments:

Anirudha Paul February 22, 2016 at 10:15 PM

Nice article ... helped me a lot :-)

Reply

Fahad Ahmed July 3, 2016 at 11:23 PM

Thanks a lot for the article... Completely understood :)

Reply

Sifatul Islam July 5, 2016 at 3:41 PM

Very Useful Article.... It will be Great Help to me if I find more article like this.

Reply

Enter your comment...

Comment as: Unknown (Goo

Publish Preview

Leave comments for Queries, Bugs and Hugs.

Newer Post Home

Subscribe to: Post Comments (Atom)

Blog Archive

▼ 2015 (35)

Labels

Analysis Arithmetic Function Backtrack Big Complexity Contest D&C Divisors Factorial

forthright48

Learning Never Ends

Monday, August 10, 2015

Prime Factorization of Factorial

Problem

Given a positive integer N, find the prime factorization of $N!$.

For example, $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120 = 2^3 \times 3 \times 5$.

Brute Force Solution

A possible solution is to calculate the value of $x = N!$ and then prime factorize x. But calcula tedious. We cannot fit $N!$ where $N > 20$ in a long long variable. We will need to use Big Inte would make things slow. I will soon write a blog post on Big Integer, until then know that using would take more than N^2 steps to calculate $N!$.

Is there a better way?

Limits on Prime

Before we move on to the solution, let us first decide the limit on prime. In order to factorize x generate prime numbers. But up to which value? Should we generate all primes less than $\sqrt{x}$

Even for a small value of N like 100, x can be huge with over 100 digits in it, thus, $\sqrt{x}$ will als Generating so many primes is not feasible. Using Sieve of Eratosthenes we could generate p which is nowhere near $\sqrt{100!}$.

Note that $N! = N \times (N-1) \times (N-2) \times \ldots \times 2 \times 1$. That is, $N!$ is a product of numbers only. Now, can there be any prime greater than N that can divide $N!$?

Suppose there is a number A and we factorized it. It is trivial to realize that all its prime factors or equal to A. So in $N!$, which is the product of numbers less than N, if we decompose all th their prime factors, then they will reduce to primes less than or equal to N.

For example, $6! = 6 \times 5 \times 4 \times 3 \times 2 \times 1 = (2 \times 3) \times 5 \times 2^2 \times 3 \times 2 = 2^4 \times 3^2 \times 5$.

So the prime factors of $N!$ will be less than or equal to N. Generating primes till $\sqrt{N!}$ is not r

just need to generate all primes less than or equal to N.

Prime Factorization

Now that we know the limit for primes, we are ready to begin factorizing the factorial. There is r way to achieve this. We will see three of them and discuss which one is best.

First - Linear Loop From 1 to N

We know that $N! = N \times (N-1) \times (N-2) \times \ldots \times 2 \times 1$. So we could simply factorize e 1 to N and add to a global array that tracks the frequency of primes. Using the code for $fact$ here, we could write a solution like below.

```
1  vector<int> prime;
2  int primeFactor[SIZE]; ///Size should be as big as N
3
4  void factorize( int n ) {
5      int sqrtn = sqrt ( n );
6      for ( int i = 0; i < prime.size() && prime[i] <= sqrtn; i++ ) {
7          if ( n % prime[i] == 0 ) {
8              while ( n % prime[i] == 0 ) {
9                  n /= prime[i];
10                 primeFactor[ prime[i] ]++; ///Increment global primeF
11             }
12             sqrtn = sqrt ( n );
13         }
14     }
15     if ( n != 1 ) {
16         primeFactor[n]++;
17     }
18 }
19
20 void factFactorize ( int n ) {
21     for ( int i = 2; i <= n; i++ ) {
22         factorize( i );
23     }
24
25     ///Now We can print the factorization
26     for ( int i = 0; i < prime.size(); i++ ) {
27         printf ( "%d^%d\n", prime[i], primeFactor[ prime[i] ] );
28     }
29 }
```

We pass the value of N to $factFactorize()$ in line 20, and it calculates the frequency of ea starts a loop from 2 to N and factorizes each of them. In line 4 we have the $factorize()$ func bit in line 10 and $16$to suit our need. When those lines find a prime factor, they increase the fr prime in the $primeFactor$ array.

It is simple and straight forward, but takes $O(N \times factorize())$ amount of time. We can do

Second - Summation of Each Prime Frequency

Instead of factorizing from 1 to N, how about we just find out how many times each prime occ them. If p_1 occurs a_1 times, p_2 occurs a_2 times $\ldots$ p_x occurs a_x times, then $N! = p_1^{a_1} \times p_2^{a_2}$

That sounds nice, but how do we find the frequency of prime factors in $N!$. Let us just focus o factor, for example 2, and find out how many times it occurs in $N!$. We will extend this idea to

Let $N = 12$. How many times does 2 occur in $12!$? We know that $12! = 12 \times 10 \times 9 \times \ldots$ numbers from 1 to 12 has 2 as their prime factors? $\frac{12}{2} = 6$ numbers do and they are $\{2, 4, 6$

we can say that at least 2^6 is a factor of $12!$. But is there more?

Yes. Notice that $4 = 2^2$, so it has an extra 2 in it that we did not count. That means for each n in them as a factor, we need to add 1 to our result. How many number are there which has 2^2 $\frac{12}{4} = 3$ numbers, which are $\{4, 8, 12\}$. So we increase our frequency to $6 + 3 = 9$ and say 2^9 in $12!$. But is that it?

No. $8 = 2^3$ and for each number with 2^3 as factor we add 1 to result. So our result is $9 + \frac{12}{8}$

Do we try with $16 = 2^4$ now? No. $12!$ cannot have any number with factor 2^4 since $\frac{12}{16} = 0.$ that $12!$ has 2^{10} as its factor and no more.

Now, we extend this idea to other primes. What is the frequency of prime factor 3 in $12!$? $\frac{12}{3} + \frac{12}{9} + \frac{12}{27} = 4 + 1 + 0 = 5$. We repeat this for all primes less than equal to 12.

Therefore, we can say that for a given prime p, $N!$ will have p^x as its prime factor where $x = \frac{N}{p} + \frac{N}{p^2} + \frac{N}{p^3} + \ldots$ Until it becomes 0 .

So, using this idea our code will look as the following.

```
1  void factFactorize ( int n ) {
2      for ( int i = 0; i < prime.size() && prime[i] <= n; i++ ) {
3          int p = prime[i];
4          int freq = 0;
5
6          while ( n / p ) {
7              freq += n / p;
8              p *= prime[i];
9          }
10
11         printf ( "%d^%d\n", prime[i], freq ); ///Printing prime^freq
12     }
13 }
```

This code factorizes $N!$ as long as we can generate all primes less than or equal to $N!$. The until $\frac{n}{p}$ becomes 0.

This code has 3 advantages over the "First" code.

1. We don't have to write $factorize()$ code.
2. Using this code, we can find how many times a specific prime p occurs in $N!$ in $O(log_p$ "First" code, we will need to run $O(N)$ loop and add occurrences of p in each number.
3. It has a better complexity for Factorization. Assuming the loop in line 6 runs $log_2(N)$ tin a complexity of $O(Nlog_2(N))$. The code runs faster than this since we only loop over N and at each prime the loop runs only $O(log_p(N))$ times. The "First" code ran with $O(N \times factorize())$ complexity, where $factorize()$ has complexity of $O(\frac{\sqrt{N}}{ln(\sqrt{N})} +$

This idea still has a small flaw. So the next one is better than this one.

Three - Better Code than Two

Suppose, we want to find out how many times 1009 occurs in $9 \times 10^{18}!$. Let us modify the "S write another function that will count the result for us.

```
1  long long factorialPrimePower ( long long n, long long p ) {
2      long long freq = 0;
3      long long cur = p;
4
5      while ( n / cur ) {
6          freq += n / cur;
7          cur *= p;
8      }
9
10     return freq;
11 }
```

If we pass $n = 9 \times 10^{18}$ and $p = 1009$, it will return us 8928571428571439. But this is wro the code overflows resulting in wrong answer. Try it yourself. Print out the value of cur in eac when it overflows.

We could change the condition in line 5 into something like this to solve the situation:

```
1  while ( n / cur > 0 )
```

But this remedy won't work if $cur \times p$ overflows into a positive number. If we want we could u that avoids multiplying two numbers if it crosses a limit, but there is an simpler way.

Note that $\frac{N}{p^3}$ is same as $\frac{\frac{N}{p^2}}{p}$. So instead of saying that $res = \frac{N}{p} + \frac{N}{p^2} \ldots$, we could rewrite it a

$x = N : \ res = res + \frac{x}{p}.$
$x = \frac{N}{p} = \frac{x}{p} : \ res = res + \frac{x}{p}.$
$x = \frac{N}{p^2} = \frac{\frac{N}{p}}{p} = \frac{x}{p} : \ res = res + \frac{x}{p}.$
$\ldots$
$x = 0$

Instead of raising the power of p, we divide the value of N by p at each step. This has the sa

So our code for finding frequency of specific prime should look like following:

```
1  long long factorialPrimePower ( long long n, long long p ) {
2      long long freq = 0;
3      long long x = n;
4
5      while ( x ) {
6          freq += x / p;
7          x = x / p;
8      }
9
10     return freq;
11 }
```

There might still be inputs for which this code will overflow, but chances for that is now lower.. $n = 9 \times 10^{18}$ and $p = 1009$, then this time we get 8928571428571425 as our result.

If we apply this improvement in our $factFactorize()$ function, then it will become:

```
1  void factFactorize ( int n ) {
```

```
    for ( int i = 0; i &lt; prime.size() &amp;&amp; prime[i] &lt;= n;
        int x = n;
        int freq = 0;

        while ( x / prime[i] ) {
            freq += x / prime[i];
            x = x / prime[i];
        }

        printf ( "%d^%d\n", prime[i], freq );
    }
}
```

This code has less chance to overflow so it is better.

Resource

1. forthright48 - Prime Factorization of an Integer

Posted by Mohammad Samiul Islam

G+1 Recommend this on Google

Labels: Factorial, Factorization, Number Theory, Prime

No comments:

Post a Comment

Leave comments for Queries, Bugs and Hugs.

Enter your comment...

Comment as: Unknown (Goo

Publish Preview

Newer Post Home

Subscribe to: Post Comments (Atom)

forthright48

Learning Never Ends

Wednesday, August 12, 2015

Number of Trailing Zeroes of Factorial

Problem

Given a positive integer N, find the number of trailing zero $N!$ has.

For example, $5! = 120$ has 1 trailing zero and $10! = 3628800$ has 2.

Brute Force Solution

Brute Force solution for this problem involves calculating the value of $N!$, which is overkill in th
the section on Brute Force on this article before you continue - [1] Number of Digits of Factoria

The brute force solution for this problem faces same limitation as [1]. We cannot calculate $N!$
using long long variable and using Big Integer is too slow, $O(N^2)$, if we assume number of d
increase by 1 in each step.

Solution Using Factorization

Suppose a there is a number x, with y trailing zeroes at its end, then can't we write that numb
$x = z \times 10^y$?

For example, $x = 12300$ and it has 2 trailing zeroes. Then we can write $x = 123 \times 10^2$.

Therefore, for every factor of 10, x gets a trailing zero. If we can find out number of 10 in $N!$ t
count its number of trailing zeroes.

How do we form a 10? We form a 10 by multiplying 2 and 5. So we need to find out how many
This can be found using the idea for factorizing $N!$. The idea is discussed in [2]Prime Factoriz
We will be using $\text{factorialPrimePower}()$ function from that post.

So all we need to do is call $x = \text{factorialPrimePower}(2)$ and $y = \text{factorialPrimePower}$
frequency of those primes. For every pair of $(2, 5)$ we get one 10 factor. So how many pairs c
have x number of 2's and y number of 5's? We can make $MIN(x, y)$ pairs.

For example, for $10!$ we have $x = \frac{10}{2} + \frac{10}{4} + \frac{10}{8} = 5 + 2 + 1 = 8$ and $y = \frac{10}{5} = 2$. There

trailing zero is $MIN(x, y) = MIN(8, 2) = 2$.

Trailing Zeroes in Different Base

We solved the problem for decimal base. Now what if we want to know how many trailing zerc we convert $N!$ to base B?

For example, how many trailing zero does $10!$ has in base 16? In order to solve this, we neec number system works. Read the post [3]Introduction to Number System to learn more.

Previously we said that for decimal numbers, multiplying them with 10 increase their trailing ze something similar be said for the base 16? Yes. Look at the following values:

$(1)_{16} = 1 \times 16^0$
$(10)_{16} = 1 \times 16^1$
$(100)_{16} = 1 \times 16^2$

Everytime we multiply a number with its base, all its digits shift a place to left and at the end a (a number in base B, if we multiply it by B it gets a trailing zero at its end.

So all we need to do is find out how many B does $N!$ has in it. For base 16, we need to find c times $16 = 2^4$ occurs in $N!$. For that we find out how many times 2 occurs using $x = \text{factorialPrimePower}(2)$ and since we get 16 for each 2^4, we conclude that $N!$ has $\frac{x}{4}$

This is extended for any base. Factorize the base B and find out occurances of each prime. T how many combinations we can make.

For example if base is 60, then $60 = 2^2 \times 3 \times 5$. So we find out $x = \text{factorialPrimePowe}$ $y = \text{factorialPrimePower}(3)$ and $z = \text{factorialPrimePower}(5)$. But then, we see that v so we can't use all the x we calculated, we need to pair them. So it becomes $x = \frac{x}{2}$. Now, us make 60 only $MIN(x, y, z)$ times. So this will be number of trailing zero.

Summary

We find number of trailing zero using the following steps:

1. Factorize the base B
2. If $B = p_1^{a_1} \times p_2^{a_2} \ldots \times p_k^{a_k}$, then find out occurance of $x_i = \text{factorialPrimePower}(p_i$
3. But we can't use x_i directly. In order to create B we will need to combine each p_i into p each x_i by a_i.
4. Number of trailing zero is $MIN(x_1, x_2, \ldots, x_k)$.

Resources

1. forthright48 - Number of Digits of Factorial
2. forthright48 - Prime Factorization of Factorial
3. forthright48 - Introduction to Number System
4. Wikipedia - Trailing Zero

forthright48

Learning Never Ends

Saturday, August 15, 2015

Leading Digits of Factorial

Problem

Given an integer N, find the first K leading digits of $N!$.

For example, for $N = 10$ and $K = 3$, then first 3 leading digits of $10! = 3628800$ is 362.

Finding leading digits uses concepts similar to [1]Number of Trailing Zeroes of Factorial.

Brute Force Solution

Finding the value of $N!$ and then printing the first K digits is a simple but slow solution. Using can calculate value $N!$ up to $N \leq 20$ and using Big Integer we can calculate arbitrary $N!$ bu worse than $O(N^2)$.

Solution Using Logarithm

In [1], we say that a logarithm of value x is y such that $x = 10^y$. For now let us find out leadin x instead of $N!$. We will extend it to cover factorials later.

So, we know that $log_{10}(x) = y$, where y will be some fraction. Let us separate y into its integ part and call them p, q. For example, if $y = 123.456$, then $p = 123$ and $q = 0.456$.

Therefore, we can say that $log_{10}(x) = p + q$. Which means, $x = 10^y = 10^{p+q} = 10^p \times 1($

Now expand the values of 10^p and 10^q. If $A = 10^p$, then A will simply be a power of 10 since To be more exact, A will be 1 with p trailing zeroes. For example, $A = 10^3 = 1000$. What ab

Since q is a fraction which is $0 \leq q < 1$, value of B will be between $10^0 \leq B < 10^1$, that is,

Okay, we got the value of A and B, what now? We know that if we multiply A and B we will g multiply them just yet. Think for a bit what will happen when we multiply a decimal number with it will get a trailing zero, e.g, $3 \times 10 = 30$. But if it is a fraction, its decimal point will shift to righ $23.65 \times 10 = 236.5$. Actually, decimal points shifts for integer numbers too, since integer nu numbers with 0 as fraction, e.g $3 = 3.00$. So in either case multiplying 10 shifts decimal point

So what happens if we multiply, A, which is just 10^p to B? Since A has 10 in it p times, the de will shift to right p times. That is all A does to B is change its decimal point. It does not change any way. Thus, B contains all the leading digits of x.

For example, $log_{10}(5420) = 3.7339993 = 3 + 0.7339993. \therefore B = 10^0.7339993 = 5.42($

So, if we need first K leading digits of x, we just need to multiply B with 10^{K-1} and then thro fraction part. That is $res = \lfloor B \times 10^{K-1} \rfloor$. Why 10^{K-1} not just 10^K? That's because we alre leading digit present in 10^q before shifting it.

Extending to Factorial

It is easy to extend the idea above to $N!$. First we need to find out the value of $y = log_{10}(N!$

$$y = log_{10}(N!)$$
$$y = log_{10}(N \times (N-1) \times (N-2) \times \ldots \times 3 \times 2 \times 1)$$
$$y = log_{10}(N) + log_{10}(N-1) + log_{10}(N-2) + \ldots + log_{10}(2) + log_{10}(1)$$

So we can simply find out the value of y by running a loop from 1 to N and taking its log value

After that we decompose y into p, integer part and q, fraction part. The answer will be $\lfloor 10^q \times$

Code

```
const double eps = 1e-9;

/// Find the first K digits of N!
int leadingDigitFact ( int n, int k ) {
    double fact = 0;

    ///Find log(N!)
    for ( int i = 1; i <= n; i++ ) {
        fact += log10 ( i );
    }

    ///Find the value of q
    double q = fact - floor ( fact+eps );

    double B = pow ( 10, q );

    ///Shift decimal point k-1 times
    for ( int i = 0; i < k - 1; i++ ) {
        B *= 10;
    }

    ///Don't forget to floor it
    return floor(B+eps);
}
```

The code does exactly what we discussed before. But note the eps that we added when floor 12 and 22. This due to precision error when dealing with real numbers in C++. For example, d error sometimes a value which is supposed to be 1, becomes 0.9999999999999. The differe these two values is very small, but if we floor them both, the first one becomes 1 whereas the becomes 0. So in order to avoid this error, when flooring a positive value we add a small num 0.000000001) to the number.

Summary

We need to execute the following steps to find the first K leading digits of a number x (in our):

1. Find the log value of the number whose leading digits we are seeking. $y = log_{10}(x)$.
2. Decompose y into two parts. Integer part p and fraction part q.
3. The answer is $\lfloor 10^q \times 10^{K-1} \rfloor$.

Resource

1. forthright48 - Number of Trailing Zeroes of Factorial

Posted by Mohammad Samiul Islam G+1 Recommend this on Google

Labels: Factorial, Logarithm, Math, Number Theory

4 comments:

Anonymous September 15, 2015 at 11:26 AM

Would have been better if you gave the implementation of code

Reply

Replies

Mohammad Samiul Islam September 15, 2015 at 7:30 PM

I added another comment above the function name in code. Hopefully, that will make find first 4 digits of 100! you just need to call the function: leadingDigitFact(100,4).

taslim uddin June 23, 2016 at 11:21 PM

Vaiya last 3 digit kibabe ber korbo

Mohammad Samiul Islam July 1, 2016 at 5:44 PM

@taslim uddin: Mod the factorial with 1000. It should give you the last 3 digits.

Reply

COME ON CODE ON

A blog about programming and more programming.

Last Non-zero Digit of Factorial

with 26 comments

If you are not familiar with factorial read this.

Question: Find the last non-zero digit of n!

Approach 1
If you think you can calculate the factorial first and then divide out all the zeroes, you can but only to a certain extent, i mean to the point you can calculate n factorial. What if you had to find the last non-zero digit of a very large number whose factorial you cannot calculate.

Approach 2
This is the most popular technique and quite handy too. We know that n! = 1 x 2 x 3 …x (n-1) x n
or n! = (2^a2)(3^a3)(5^a5)(7^a7)…
or n! = (2^a2)(5^a5)(3^a3)(7^a7)…

Now we know that the number of zeroes in n1 is equal to a5. So n! divided by 10^a5 will be equal to factorial without trailing digits. Lets call this n! without trailing zeroes as (n!)' .So, (n!)' = (2^(a2-a5))(3^a3)(7^a7)…

Let L(k) denote the least significant non-zero decimal digit of the integer k. So from above we can infer that
L(n!) = L((n!)') = L[(2^(a2-a5))(3^a3)(7^a7)…]
= L[(3^a3)(7^a7)…]*L[(2^(a2-a5))]

This logic is implemented in the C/C++ function below.

```
1  int last_digit_factorial(int N)
2  {
3      int i,j,ans=1,a2=0,a5=0,a;
4
5      for (i = 1; i <= N; i++)
6      {
7          j = i;
8          //divide i by 2 and 5
9          while (j%2==0)
10         {
```

```
11                  j /= 2;
12                  a2++;
13              }
14              while (j%5==0)
15              {
16                  j/=5;
17                  a5++;
18              }
19
20              ans = (ans*(j%10))%10;
21          }
22
23          a=a2-a5;
24
25          for (i = 1; i <= a; i++)
26              ans=(ans * 2) %10;
27
28          return ans;
29      }
```

Approach 3

Fact : The powers of 2 3 7 and 1 have cyclic last digit.
So we divide all the primes into one of these groups and then take out the power accordingly.

power1[4]={1, 1, 1, 1}
power2[4]={6, 2, 4, 8}
power3[4]={1, 3, 9, 7}
power7[4]={1, 7, 9, 3}
power9[4]={1, 9, 1, 9}

So for example, if we have to find last digit in 6!
6! = 1 * 2 * 3 * 4 * 5 * 6
6! = 2^3 * (3^2) * 10
6!′ = 2^3 * (3^2)

We know the power of 3 has cyclic last digit, so 3^2 last digit is 9 or power3[2 mod 4]
L(6!) = L(6!′) = power2[3 mod 4] * power3[2 mod 4] mod 10
= 32 mod 10 = 2

But why code this method when we have a simpler approach.

Approach 4

Theory : Since we know that only a factor 5 brings in zero and we always have more powers of 2 than of 5, so the value of L(n!) is easily computed recursively as L(L(n)L((n-1)!)) unless n is a multiple of 5, in which case we need more information. As a result, the values of L(n!) come in fixed strings of five, as shown below :

	1	5	10	15	20	25
L(n!)	1264	22428	88682	88682	44846	44846

Thus if we know any value of 5n we can get the value of 5n+j. For example if we know value of L(15!) we know the value of L(16!), L(17!) up-to L(19!). But how to get the value of 5n. If we map the starting values of the L(5n!) we get like 2 8 8 4 4 8 2 2 6. They come in another fixed strings of five. So we need to know L(25n!) to know L((25n+5j)!) for j = 0,1,2,3,4. Continuing in this way if we tabulate the values of L((5^t n)!) we get :

L(n!)	1264	22428	88682	88682	44846	44846
L(5n!)	2884	48226	24668	48226	48226	86442
L(25n!)	4244	82622	82622	28488	46866	64244
L(125n!)	8824	68824	26648	68824	42286	26648
L(625n!)	1264	22428	88682	88682	44846	44846

So we see that the pattern for L(625n!) is same as L(n!). Hence we can conclude that the pattern for L((5^j n)!) is the same as for L((5^(j+4) n)!). So we can use a modulo 4 function to get the next results. Also we can see below that each of the four row have a starting digit as 0 , 2 , 4, 6 and 8.

Result :

k mod 4					
0	06264	22428	44846	66264	88682
1	02884	24668	48226	62884	86442
2	04244	28488	46866	64244	82622
3	08824	26648	42286	68824	84462

This is all we need to get the last digit of n! or L(n!). First we convert the given number n into base 5. So we have
n = d_0 + d_1*5 + d_2*5^2 + … + d_h*5^h
where d_0 is the least significant digit.

Now we enter the above table at the row h (mod 4) in the block whose first digit is 0 (because the coefficient of 5^(h+1) is zero), and determine the digit in the (d_h)th position of this block. Let this digit be denoted by s_h. Then we enter the table at row h-1 (mod 4) in the block that begins with s_h, and determine the digit in the (d_(h-1))th position of this block. Let this digit be denote by s_(h-1). We continue in this way down to s_0, which is the least significant non-zero digit of n!.

Example :
To illustrate, consider the case of the decimal number n=1592. In the base 5 this is n=22332. Now we enter the above table at row k=4=0 (mod 4) in the block beginning with 0, which is 06264. The leading digit of n (in the base 5) is 2, so we check the digit in position 2 of this block to give L((2*5^4)!) = 2. Then we enter the table at row k=3 (mod 4) in the block beginning with 2, which is 26648, to find L((2*5^4 + 2*5^3)!) = 6.

Then in the row k=2 (mod 4), the block beginning with 6 is 64244, and we find L((2*5^4 + 2*5^3 + 3*5^2)!) = 4. From this we know we're in the block 48226 in row k=1 (mod 4), so we have L((2*5^4 + 2*5^3 + 3*5^2 + 3*5)!) = 2. Finally, we enter the row k=0 (mod 4) in block 22428 to find

the result

L(1592!) = L((2*5^4 + 2*5^3 + 3*5^2 + 3*5 + 2)!) = 4

Thus if there are k digits in the base 5 representation of n then we only need k lookups in the table to get the last non-zero digit. Finally a C++ program for the algorithm :

```
#include
using namespace std;

int A[4][5][5] = {
{0,6,2,6,4,2,2,4,2,8,4,4,8,4,6,6,6,2,6,4,8,8,6,8,2},
{0,2,8,8,4,2,4,6,6,8,4,8,2,2,6,6,2,8,8,4,8,6,4,4,2},
{0,4,2,4,4,2,8,4,8,8,4,6,8,6,6,6,4,2,4,4,8,2,6,2,2},
{0,8,8,2,4,2,6,6,4,8,4,2,2,8,6,6,8,8,2,4,8,4,4,6,2}
};

char num[200];
char* dto5(int n)
{
     int b=5;
     int j,l;
     register int i=0;
     do
     {
           j=n%b;
           num[i++]=(j<10) ? (j+'0'): ('A'+j-10);
     }while((n/=b)!=0);

     num[i]='';
     l=strlen(num);
     reverse(num,num+l);
     return num;
}

int last_digit_factorial(int N)
{
    char *num=dto5(N);
    int l = strlen(num),s=0,i;
    for (i=0;i<l;i++)
    {
        s=A[(l-1-i)%4][s/2][num[i]-'0'];
    }
    return s;
}

int main()
{
  int N;

  while (scanf("%d", &N) == 1)
  {
        printf("%d\n", last_digit_factorial(N));
  }
  return 0;
}
```

Can we make it even simpler. Yes.

Let's write n as 5k + r

So,
(n)! = (5k + r)!
= 1·2·3…k.(k+1)…(5k-1)·(5k)·(5k+1)…(5k+r)
(Seperating out multiples of 5)
= {5·10·15…(5k)} · {1·2·3·4·6·7…(5k-4)·(5k-3)·(5k-2)·(5k-1)·(5k+1)…(5k+r)}
(Expanding multiples of 5)
= {5·(2·5)·(3·5)…(k·5)} · {1·2·3·4·6·7…(5k-4)·(5k-3)·(5k-2)·(5k-1)·(5k+1)…(5k+r)}
= 5^k {1·2·3…k} · {1·2·3·4·6·7…(5k-4)·(5k-3)·(5k-2)·(5k-1)·(5k+1)…(5k+r)}
= 5^k · k! · {1·2·3·4·6·7…(5k-4)·(5k-3)·(5k-2)·(5k-1)·(5k+1)…(5k+r)}
= 5^k · k! · product{(5·i+1)·(5·i+2)·(5·i+3)·(5·i+4), i = 0 to k – 1} · (5k+1)…(5k+r)
(Multiplying and dividing by 2^k)
= 5^k · 2^k · k! · product((5i+1)·(5i+2)·(5i+3)·(5i+4)/2, i = 0 to k – 1) · (5k+1)…(5k+r)
= 10^k · k! · product((5i+1)·(5i+2)·(5i+3)·(5i+4)/2, i = 0 to k – 1) · (5k+1)…(5k+r)

Now to find the last digit of n! or L(n), we remove the power of 10 first and move to next type.

So,
L(n) = L(5k+r)
L(n) = L(k! · product((5i+1)·(5i+2)·(5i+3)·(5i+4)/2, i = 0 to k – 1))
L(n) = [L(k) · {product((5i+1)·(5i+2)·(5i+3)·(5i+4)/2, i = 0 to k – 1) mod 10} · {(5k+1)…(5k+r) mod 10 }] mod 10

Now let us first find P = (5i+1)·(5i+2)·(5i+3)·(5i+4)/2 mod 10

To calculate this we will have to take (mod 2) value and (mod 5) value and then using chinese remainder take out (mod 10) value.

Clearly P mod 2 = 0.

Modular inverse of 2 (mod 5) is 3.

So,
P mod 5 = 1·2·3·4·3 mod 5
= 72 mod 5
= 2 mod 5

Using the chinese remainder theorem we get,

P mod 10 = 2 mod 10

Hence,

L(n) = [L(k) · {product((5i+1)·(5i+2)·(5i+3)·(5i+4)/2, i = 0 to k – 1) mod 10} · {(5k+1)…(5k+r) mod 10 }] mod 10
= [L(k) · {product(2, i = 0 to k – 1) mod 10} · L(r)] mod 10
= [L(k) · L(r) · {2^k mod 10}] mod 10
= 2^k · L(k) · L(r) mod 10

Also,
2^k mod 10 = 1 when k = 6 when k = 4i
= 2 when k = 4i+1
= 4 when k = 4i+2
= 8 when k = 4i+3

Hence,

L(0) = L(1) = 1
L(2) = 2
L(3) = 6
L(4) = 4

L(n) = L(5k+r) = 2^k·L(k)·L(r) (mod 10)

And here's a simpler code.

```
#include<iostream>
using namespace std;

int P(int K)
{
        int A[]={6,2,4,8};
        if (K<1) return 1;
        return A[K%4];
}

int L(int N)
{
        int A[]={1,1,2,6,4};
        if (N<5) return A[N];
        return (P(N/5)*L(N/5)*L(N%5))%10;
}

int main()
{
        int N;
        while (scanf("%d", &N) == 1)
                printf("%d\n", L(N));
        return 0;
}
```

NJOY!!!
fR0D

About these ads

Written by fR0DDY

June 20, 2009 at 6:12 PM

Posted in Programming

Tagged with C, code, digit, factorial, last, lookup, non-zero

26 Responses

Subscribe to comments with RSS.

I found the post very useful. . .
Thanks a lot. . .
Hope that a lot of such important posts will be come on later:-). . .

Muhammed

June 28, 2009 at 3:32 PM

Reply
how about finding last k nonzero digits of factorial??

boy

September 26, 2009 at 2:58 PM

Reply
Nice !

But what is this ? : num[i]=”

Besides I think readers will find this link interesting
:http://ipsc.ksp.sk/contests/ipsc1999/real/solutions/d.php

nthrgeek

December 24, 2009 at 8:01 PM

Reply

Actually the dto5 function was a general code to convert from decimal to any base. So the num[i] = ‘ ‘ .

fR0DDY

December 25, 2009 at 1:20 PM

Reply

very very useful.. helped me to get around the soln of spoj prob..

thanks alot….

keep it up

pushkar

August 1, 2010 at 10:45 AM

Reply

Hey buddy,

Thanks for sharing a good information. I appreciate.

Thanks

ranganathg

August 10, 2010 at 10:46 PM

Reply

[…] factorial — The Endeavour posted at The Endeavour. Which naturally leads to Gaurav Kumar‘s Last Non-zero Digit of Factorial posted at COME ON CODE […]

Carnival of Mathematics 69 « JD2718

September 3, 2010 at 9:04 PM

Reply

but why would anyone wanna find the last non-zero digit of a factorial??? where is this applied for good use?

brain

October 1, 2010 at 12:37 PM

Reply

awesome..!!

kush

December 15, 2010 at 6:34 PM

Reply
This is a solution to the Math Monthly problem 11568 (in the April, 2011 issue).

EMR

April 12, 2011 at 7:22 PM

Reply
[…]] . After searching a bit , i found couple of links which are quite good and self explanatory 1] comeoncode 2] mathproblem 3] mathpages [this one contains other nice problems also] . I implemented the […]

Last Non zero digit of factorial « My Weblog

May 24, 2011 at 2:39 AM

Reply
comeoncodeon is a very helpful site, the way of explanation is great.

chetan

June 4, 2011 at 4:27 AM

Reply
great explanation…….

praneeth

June 7, 2011 at 5:03 PM

Reply
So n! divided by 10^a5 will be equal to factorial without trailing digits. Lets call this n! without trailing zeroes as (n!)'

this statement isn't right it is divisible by 10^min(a5,a2)

TDL

July 11, 2011 at 12:53 AM

Reply
Hi TDL,
It is correct, since min(a5,a2) is always equal to a5.

fR0DDY

July 11, 2011 at 9:46 PM

Reply
wat is L(n!)

fdg

November 15, 2011 at 11:48 AM

Reply

It is the last non-zero digit of n factorial.

fR0DDY

November 15, 2011 at 11:57 AM

Reply

[…] 한 자리만 구해도 된다면, 훨씬 빠른 방법이 있습니다. Last Non-zero Digit of Factorial을 읽어 […]

코딩 인터뷰 완전 분석 **210**쪽 **17.3** 변형 문제 풀이 I **dlimpid's blog**

September 5, 2012 at 4:19 PM

Reply

Really great. I didn't come up a close form solution, through it's a logN algorithm too.

Xiaoyi

November 13, 2012 at 8:58 PM

Reply

could you explain why {(5k+1)…(5k+r) mod 10 } = L(r)? As it only stand when k is even.

Xiaoyi

November 13, 2012 at 10:49 PM

Reply

Aha, (2(5k+1)(5k+2)…(5k+r)) mod 10 = 2 * L(r).

Xiaoyi

November 13, 2012 at 11:06 PM

Reply

i dont understand that chinese remainder theorem part…i mean how to know the result of P mod 10 from P mod 2 and P and 5….??..can u pls explain…

jayanth

February 1, 2013 at 6:42 AM

Reply

Probably this will help: http://en.wikipedia.org/wiki/Chinese_remainder_theorem

fR0DDY

February 1, 2013 at 6:44 AM

Reply

thanks….or alternatively we can find P mod 10 in the foll way right…??
we know that [(5i+1)(5i+1)(5i+1)(5i+4)] mod 10 will always give 4…but P is (5i+1)(5i+1)(5i+1)(5i+4)/2….P mod 10 is (4/2)%10 which is (2)mod10 that is equal to 2….and now this gets repeated k times….therefore P mod 10 is nothing but (2^k)mod10…..
and btw amazing post with gr8 explanation….thanks…:)

jayanth

February 5, 2013 at 9:45 PM

sorry i meant (5i+1)(5i+2)(5i+3)(5i+4)…..not (5i+1)(5i+1)(5i+1)(5i+4)…

jayanth

February 5, 2013 at 9:51 PM

[…] musiałem się poddać. Aż w końcu trafiłem na blog fR0DDY'ego, który tłumaczy wszystko jak krowie na rowie. Hmm czyli coś dla mnie. Zaznaczyć trzeba, że […]

Ostatnia niezerowa cyfra silni | Nitcode

February 27, 2015 at 5:00 PM

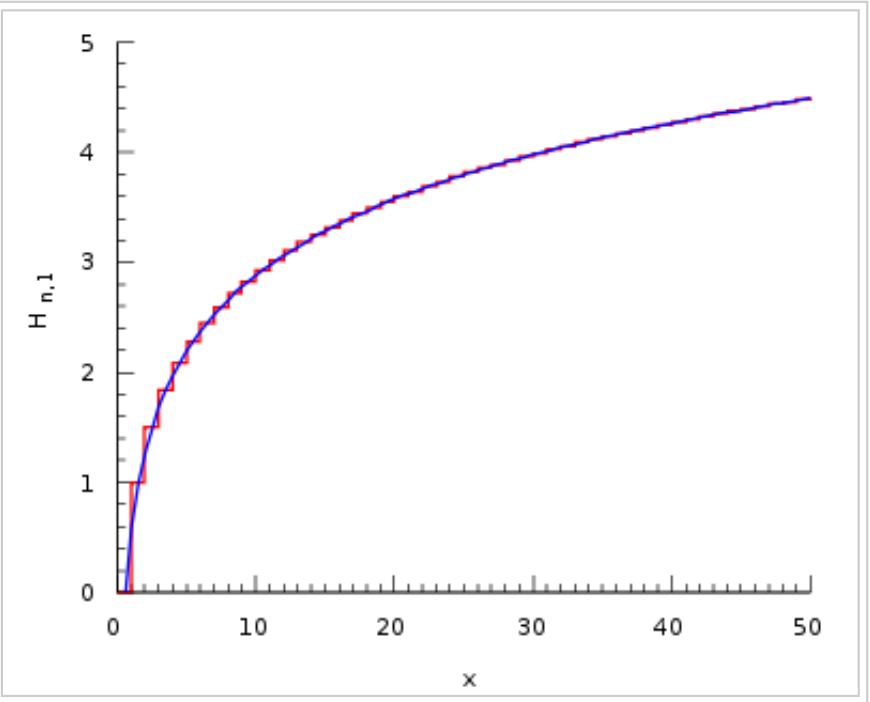
Reply

Create a free website or blog at WordPress.com. The Journalist v1.9 Theme.

Harmonic number

From Wikipedia, the free encyclopedia

In mathematics, the *n*-th **harmonic number** is the sum of the reciprocals of the first *n* natural numbers:



The harmonic number $H_{n,1}$ with $n = \lfloor x \rfloor$ (red line) with its asymptotic limit $\gamma + \ln(x)$ (blue line).

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} = \sum_{k=1}^{n} \frac{1}{k}.$$

This also equals *n* times the inverse of the harmonic mean of these natural numbers.

The numbers *n* such that the numerator of the fully reduced expression for H_n is prime are

> 2, 3, 5, 8, 9, 21, 26, 41, 56, 62, 69, 79, 89, 91, 122, 127, 143, 167, 201, 230, 247, 252, 290, 349, 376, 459, 489, 492, 516, 662, 687, 714, 771, 932, 944, 1061, 1281, 1352, 1489, 1730, 1969, ... (sequence A056903 in OEIS)

Harmonic numbers were studied in antiquity and are important in various branches of number theory. They are sometimes loosely termed harmonic series, are closely related to the Riemann zeta function, and appear in the expressions of various special functions.

The associated harmonic series grows without limit, albeit very slowly, roughly approaching the natural logarithm function.[1]:143 In 1737, Leonhard Euler used the divergence of this series to provide a new proof of the infinity of prime numbers. His work was extended into the complex plane by Bernhard Riemann in 1859, leading directly to the celebrated Riemann hypothesis about the distribution of prime numbers.

When the value of a large quantity of items has a Zipf's law distribution, the total value of the *n* most-valuable items is the *n*-th harmonic number. This leads to a variety of surprising conclusions in the Long Tail and the theory of network value.

Bertrand's postulate entails that, except for the case *n=1*, the harmonic numbers are never integers.[2]

samiul (1).txt

there is a road of 100 m . along the way there are several holes at random point, we need to find a minimum stepsize n <= 100 such that, i can cross the whole way with out falling in any of the holes.

soln:

```
for(int i=1; i <= 100; i++)
{
  for(int j =i; j <= 100; j+=i)
  {
      if(status[j]==hole) break;

      else if( j+i>100)
      {
        cout<<i<<endl;break;
      }
   }
}
```

complexity: O(nlogn)

Contents

Identities involving harmonic numbers

By definition, the harmonic numbers satisfy the recurrence relation

$$H_n = H_{n-1} + \frac{1}{n}.$$

They also satisfy the series identity

$$\sum_{k=1}^{n} H_k = (n+1)H_{n+1} - (n+1).$$

The harmonic numbers are connected to the Stirling numbers of the first kind:

$$H_n = \frac{1}{n!}\begin{bmatrix} n+1 \\ 2 \end{bmatrix}.$$

The functions

$$f_n(x) = \frac{x^n}{n!}(\log x - H_n)$$

satisfy the property

$$f_n'(x) = f_{n-1}(x).$$

In particular

$$f_1(x) = x(\log x - 1)$$

is a primitive of the logarithmic function.

Identities involving π

There are several infinite summations involving harmonic numbers and powers of π:[3]

$$\sum_{n=1}^{\infty} \frac{H_n}{n \cdot 2^n} = \frac{1}{12}\pi^2;$$

$$\sum_{n=1}^{\infty} \frac{H_n^2}{(n+1)^2} = \frac{11}{360}\pi^4;$$

$$\sum_{n=1}^{\infty} \frac{H_n^2}{n^2} = \frac{17}{360}\pi^4;$$

$$\sum_{n=1}^{\infty} \frac{H_n}{n^3} = \frac{1}{72}\pi^4;$$

Calculation

An integral representation given by Euler[4] is

$$H_n = \int_0^1 \frac{1-x^n}{1-x}\,dx.$$

The equality above is obvious by the simple algebraic identity

$$\frac{1-x^n}{1-x} = 1 + x + \cdots + x^{n-1}.$$

Using the simple integral transform $x = 1-u$, an elegant combinatorial expression for H_n is

$$\begin{aligned}
H_n &= \int_0^1 \frac{1-x^n}{1-x}\,dx \\
&= -\int_1^0 \frac{1-(1-u)^n}{u}\,du \\
&= \int_0^1 \frac{1-(1-u)^n}{u}\,du \\
&= \int_0^1 \left[\sum_{k=1}^{n}(-1)^{k-1}\binom{n}{k}u^{k-1}\right]du \\
&= \sum_{k=1}^{n}(-1)^{k-1}\binom{n}{k}\int_0^1 u^{k-1}\,du \\
&= \sum_{k=1}^{n}(-1)^{k-1}\frac{1}{k}\binom{n}{k}.
\end{aligned}$$

The same representation can be produced by using the third Retkes identity by setting $x_1 = 1, \ldots, x_n = n$ and using the fact that $\Pi_k(1,\ldots,n) = (-1)^{n-k}(k-1)!(n-k)!$

$$H_n = H_{n,1} = \sum_{k=1}^{n}\frac{1}{k} = (-1)^{n-1}n!\sum_{k=1}^{n}\frac{1}{k^2\Pi_k(1,\ldots,n)} = \sum_{k=1}^{n}(-1)^{k-1}\frac{1}{k}\binom{n}{k}.$$

The *n*th harmonic number is about as large as the natural logarithm of *n*. The reason is that the sum is approximated by the integral

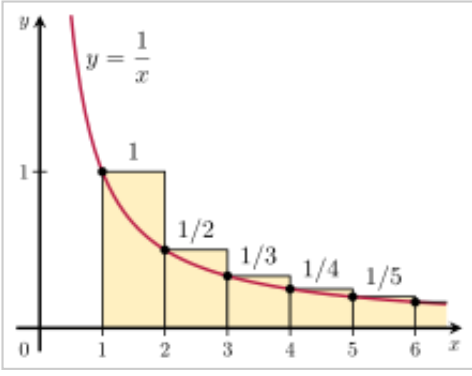
$$\int_1^n \frac{1}{x}\,dx$$

whose value is ln(*n*).

The values of the sequence H_n - ln(*n*) decrease monotonically towards the limit

$$\lim_{n\to\infty}(H_n - \ln n) = \gamma,$$

where γ ≈ 0.5772156649 is the Euler–Mascheroni constant. The corresponding asymptotic expansion as $n \to \infty$ is



Graph demonstrating a connection between harmonic numbers and the natural logarithm. The harmonic number H_n can be interpreted as a Riemann sum of the integral:

$$\int_1^{n+1}\frac{1}{x}dx = \ln(n+1)$$

$$H_n \sim \ln n + \gamma + \frac{1}{2n} - \sum_{k=1}^{\infty}\frac{B_{2k}}{2kn^{2k}} = \ln n + \gamma + \frac{1}{2n} - \frac{1}{12n^2} + \frac{1}{120n^4} - \cdots,$$

where B_k are the Bernoulli numbers.

Special values for fractional arguments

There are the following special analytic values for fractional arguments between 0 and 1, given by the integral

$$H_\alpha = \int_0^1 \frac{1-x^\alpha}{1-x}\,dx\,.$$

More values may be generated from the recurrence relation

$$H_\alpha = H_{\alpha-1} + \frac{1}{\alpha}\,,$$

or from the reflection relation

$$H_{1-\alpha} - H_\alpha = \pi\cot(\pi\alpha) - \frac{1}{\alpha} + \frac{1}{1-\alpha}\,.$$

For example:

$$\begin{aligned}
H_{\frac{3}{4}} &= \tfrac{4}{3} - 3\ln 2 + \tfrac{\pi}{2} \\
H_{\frac{2}{3}} &= \tfrac{3}{2}(1-\ln 3) + \sqrt{3}\tfrac{\pi}{6} \\
H_{\frac{1}{2}} &= 2 - 2\ln 2 \\
H_{\frac{1}{3}} &= 3 - \tfrac{\pi}{2\sqrt{3}} - \tfrac{3}{2}\ln 3 \\
H_{\frac{1}{4}} &= 4 - \tfrac{\pi}{2} - 3\ln 2 \\
H_{\frac{1}{6}} &= 6 - \tfrac{\pi}{2}\sqrt{3} - 2\ln 2 - \tfrac{3}{2}\ln 3 \\
H_{\frac{1}{8}} &= 8 - \tfrac{\pi}{2} - 4\ln 2 - \tfrac{1}{\sqrt{2}}\left\{\pi + \ln\left(2+\sqrt{2}\right) - \ln\left(2-\sqrt{2}\right)\right\} \\
H_{\frac{1}{12}} &= 12 - 3\left(\ln 2 + \tfrac{\ln 3}{2}\right) - \pi\left(1+\tfrac{\sqrt{3}}{2}\right) + 2\sqrt{3}\ln\left(\sqrt{2-\sqrt{3}}\right)
\end{aligned}$$

For positive integers p and q with $p < q$, we have:

$$H_{\frac{p}{q}} = \frac{q}{p} + 2\sum_{k=1}^{\lfloor\frac{q-1}{2}\rfloor} \cos(\frac{2\pi pk}{q})\ln(\sin(\frac{\pi k}{q})) - \frac{\pi}{2}\cot(\frac{\pi p}{q}) - \ln(2q)$$

For every $x > 0$, integer or not, we have:

$$H_x = x\sum_{k=1}^{\infty}\frac{1}{k(x+k)}\,.$$

Based on this, it can be shown that:

$$\int_0^1 H_x\,dx = \gamma\,,$$

where γ is the Euler–Mascheroni constant or, more generally, for every n we have:

$$\int_0^n H_x\,dx = n\gamma + \ln\left(n!\right).$$

Generating functions

A generating function for the harmonic numbers is

$$\sum_{n=1}^{\infty} z^n H_n = \frac{-\ln(1-z)}{1-z},$$

where ln(*z*) is the natural logarithm. An exponential generating function is

$$\sum_{n=1}^{\infty} \frac{z^n}{n!} H_n = -e^z \sum_{k=1}^{\infty} \frac{1}{k} \frac{(-z)^k}{k!} = e^z \mathrm{Ein}(z)$$

where Ein(*z*) is the entire exponential integral. Note that

$$\mathrm{Ein}(z) = \mathrm{E}_1(z) + \gamma + \ln z = \Gamma(0,z) + \gamma + \ln z$$

where Γ(0, *z*) is the incomplete gamma function.

Applications

The harmonic numbers appear in several calculation formulas, such as the digamma function

$$\psi(n) = H_{n-1} - \gamma.$$

This relation is also frequently used to define the extension of the harmonic numbers to non-integer *n*. The harmonic numbers are also frequently used to define γ, using the limit introduced in the previous section, although

$$\gamma = \lim_{n\to\infty} \left(H_n - \ln\left(n + \frac{1}{2}\right)\right)$$

converges more quickly.

In 2002, Jeffrey Lagarias proved[5] that the Riemann hypothesis is equivalent to the statement that

$$\sigma(n) \leq H_n + \ln(H_n) e^{H_n},$$

is true for every integer $n \geq 1$ with strict inequality if $n > 1$; here σ(*n*) denotes the sum of the divisors of *n*.

The eigenvalues of the nonlocal problem

$$\lambda\phi(x) = \int_{-1}^{1} \frac{\phi(x) - \phi(y)}{|x-y|} dy$$

are given by $\lambda = 2H_n$, where by convention, $H_0 = 0$.

Generalization

Generalized harmonic numbers

The **generalized harmonic number** of order *n* of *m* is given by

$$H_{n,m} = \sum_{k=1}^{n} \frac{1}{k^m}.$$

The limit as *n* tends to infinity exists if *m* > 1.

Other notations occasionally used include

$$H_{n,m} = H_n^{(m)} = H_m(n).$$

The special case of *m* = 0 gives $H_{n,0} = n$

The special case of *m* = 1 is simply called a harmonic number and is frequently written without the superscript, as

$$H_n = \sum_{k=1}^{n} \frac{1}{k}.$$

Smallest natural number *k* such that k^n does not divide the denominator of generalized harmonic number *H*(*k*, *n*) nor the denominator of alternating generalized harmonic number *H'*(*k*, *n*) are

> 77, 20, 94556602, 42, 444, 20, 104, 42, 76, 20, 77, 110, 3504, 20, 903, 42, 1107, 20, 104, 42, 77, 20, 2948, 110, 136, 20, 76, 42, 903, 20, 77, 42, 268, 20, 7004, 110, 1752, 20, 19203, 42, 77, 20, 104, 42, 76, 20, 370, 110, 1107, 20, ... (sequence A128670 in OEIS)

In the limit of $n \to \infty$, the generalized harmonic number converges to the Riemann zeta function

$$\lim_{n\to\infty} H_{n,m} = \zeta(m).$$

The related sum $\sum_{k=1}^{n} k^m$ occurs in the study of Bernoulli numbers; the harmonic numbers also appear in the study of Stirling numbers.

Some integrals of generalized harmonic are

$$\int_0^a H_{x,2}\,dx = a\frac{\pi^2}{6} - H_a$$

and

$$\int_0^a H_{x,3}\,dx = aA - \frac{1}{2}H_{a,2}\,,$$ where *A* is the Apéry's constant, i.e. ζ(3).

and

$$\sum_{k=1}^{n} H_{k,m} = (n+1)H_{n,m} - H_{n,m-1} \quad \text{for } m \geq 0$$

Every generalized harmonic number of order m can be written as a function of harmonic of order m-1 using:

$$H_{n,m} = \sum_{k=1}^{n-1} \frac{H_{k,m-1}}{k(k+1)} + \frac{H_{n,m-1}}{n} \quad \text{for example:}$$
$$H_{4,3} = \frac{H_{1,2}}{1 \cdot 2} + \frac{H_{2,2}}{2 \cdot 3} + \frac{H_{3,2}}{3 \cdot 4} + \frac{H_{4,2}}{4}$$

A generating function for the generalized harmonic numbers is

$$\sum_{n=1}^{\infty} z^n H_{n,m} = \frac{\mathrm{Li}_m(z)}{1-z},$$

where $\mathrm{Li}_m(z)$ is the polylogarithm, and $|z| < 1$. The generating function given above for $m = 1$ is a special case of this formula.

Fractional argument for generalized harmonic numbers can be introduced as follows:

For every $p, q > 0$ integer, and $m > 1$ integer or not, we have from polygamma functions:

$$H_{q/p,m} = \zeta(m) - p^m \sum_{k=1}^{\infty} \frac{1}{(q+pk)^m}$$

where $\zeta(m)$ is the Riemann zeta function. The relevant recurrence relation is:

$$H_{a,m} = H_{a-1,m} + \frac{1}{a^m}$$

Some special values are:

$H_{\frac{1}{4},2} = 16 - 8G - \frac{5}{6}\pi^2$ where G is the Catalan's constant
$H_{\frac{1}{2},2} = 4 - \frac{\pi^2}{3}$
$H_{\frac{3}{4},2} = 8G + \frac{16}{9} - \frac{5}{6}\pi^2$
$H_{\frac{1}{4},3} = 64 - 27\zeta(3) - \pi^3$
$H_{\frac{1}{2},3} = 8 - 6\zeta(3)$
$H_{\frac{3}{4},3} = \left(\frac{4}{3}\right)^3 - 27\zeta(3) + \pi^3$

Multiplication formulas

The multiplication theorem applies to harmonic numbers. Using polygamma functions, we obtain

$$H_{2x} = \frac{1}{2}\left(H_x + H_{x-\frac{1}{2}}\right) + \ln 2$$

$$H_{3x} = \frac{1}{3}\left(H_x + H_{x-\frac{1}{3}} + H_{x-\frac{2}{3}}\right) + \ln 3,$$

or, more generally,

$$H_{nx} = \frac{1}{n}\left(H_x + H_{x-\frac{1}{n}} + H_{x-\frac{2}{n}} + \cdots + H_{x-\frac{n-1}{n}}\right) + \ln n.$$

For generalized harmonic numbers, we have

$$H_{2x,2} = \frac{1}{2}\left(\zeta(2) + \frac{1}{2}\left(H_{x,2} + H_{x-\frac{1}{2},2}\right)\right)$$
$$H_{3x,2} = \frac{1}{9}\left(6\zeta(2) + H_{x,2} + H_{x-\frac{1}{3},2} + H_{x-\frac{2}{3},2}\right),$$

where $\zeta(n)$ is the Riemann zeta function.

Generalization to the complex plane

Euler's integral formula for the harmonic numbers follows from the integral identity

$$\int_a^1 \frac{1-x^s}{1-x}\,dx = -\sum_{k=1}^{\infty} \frac{1}{k}\binom{s}{k}(a-1)^k,$$

which holds for general complex-valued *s*, for the suitably extended binomial coefficients. By choosing $a = 0$, this formula gives both an integral and a series representation for a function that interpolates the harmonic numbers and extends a definition to the complex plane. This integral relation is easily derived by manipulating the Newton series

$$\sum_{k=0}^{\infty} \binom{s}{k}(-x)^k = (1-x)^s,$$

which is just the Newton's generalized binomial theorem. The interpolating function is in fact the digamma function

$$H_s = \psi(s+1) + \gamma = \int_0^1 \frac{1-x^s}{1-x}\,dx,$$

where $\psi(x)$ is the digamma, and γ is the Euler-Mascheroni constant. The integration process may be repeated to obtain

$$H_{s,2} = -\sum_{k=1}^{\infty} \frac{(-1)^k}{k}\binom{s}{k} H_k.$$

Relation to the Riemann zeta function

Some derivatives of fractional harmonic numbers are given by:

$$\frac{d^n H_x}{dx^n} = (-1)^{n+1} n! \left[\zeta(n+1) - H_{x,n+1}\right]$$
$$\frac{d^n H_{x,2}}{dx^n} = (-1)^{n+1} (n+1)! \left[\zeta(n+2) - H_{x,n+2}\right]$$
$$\frac{d^n H_{x,3}}{dx^n} = (-1)^{n+1} \frac{1}{2} (n+2)! \left[\zeta(n+3) - H_{x,n+3}\right].$$

And using Maclaurin series, we have for $x < 1$:

$$H_x = \sum_{n=1}^{\infty} (-1)^{n+1} x^n \zeta(n+1)$$
$$H_{x,2} = \sum_{n=1}^{\infty} (-1)^{n+1} (n+1) x^n \zeta(n+2)$$
$$H_{x,3} = \frac{1}{2} \sum_{n=1}^{\infty} (-1)^{n+1} (n+1)(n+2) x^n \zeta(n+3).$$

For fractional arguments between 0 and 1, and for $a > 1$:

$$H_{\frac{1}{a}} = \frac{1}{a}\left(\zeta(2) - \frac{1}{a}\zeta(3) + \frac{1}{a^2}\zeta(4) - \frac{1}{a^3}\zeta(5) + \cdots\right)$$
$$H_{\frac{1}{a},2} = \frac{1}{a}\left(2\zeta(3) - \frac{3}{a}\zeta(4) + \frac{4}{a^2}\zeta(5) - \frac{5}{a^3}\zeta(6) + \cdots\right)$$
$$H_{\frac{1}{a},3} = \frac{1}{2a}\left(2 \cdot 3\zeta(4) - \frac{3 \cdot 4}{a}\zeta(5) + \frac{4 \cdot 5}{a^2}\zeta(6) - \frac{5 \cdot 6}{a^3}\zeta(7) + \cdots\right).$$

Hyperharmonic numbers

The next generalization was discussed by J. H. Conway and R. K. Guy in their 1995 book *The Book of Numbers*.[1]:258 Let

$$H_n^{(0)} = \frac{1}{n}.$$

Then the nth hyperharmonic number of order r ($r>0$) is defined recursively as

$$H_n^{(r)} = \sum_{k=1}^{n} H_k^{(r-1)}.$$

In special, $H_n = H_n^{(1)}$.

See also

- Watterson estimator
- Tajima's D
- Coupon collector's problem
- Jeep problem
- Riemann zeta function

Notes

1. John H., Conway; Richard K., Guy (1995). *The book of numbers*. Copernicus.
2. Ronald L., Graham; Donald E., Knuth; Oren, Patashnik (1994). *Concrete Mathematics*. Addison-Wesley.
3. Sondow, Jonathan and Weisstein, Eric W. "Harmonic Number." From MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/HarmonicNumber.html
4. Sandifer, C. Edward (2007), *How Euler Did It*, MAA Spectrum, Mathematical Association of America, p. 206, ISBN 9780883855638.
5. Jeffrey Lagarias (2002). "An Elementary Problem Equivalent to the Riemann Hypothesis". *Amer. Math. Monthly* **109**: 534–543. arXiv:math.NT/0008177. doi:10.2307/2695443.

References

- Arthur T. Benjamin, Gregory O. Preston, Jennifer J. Quinn (2002). "A Stirling Encounter with Harmonic Numbers" (PDF). *Mathematics Magazine* **75** (2): 95–103. doi:10.2307/3219141.
- Donald Knuth (1997). "Section 1.2.7: Harmonic Numbers". *The Art of Computer Programming*. Volume 1: *Fundamental Algorithms* (Third ed.). Addison-Wesley. pp. 75–79. ISBN 0-201-89683-4.
- Ed Sandifer, *How Euler Did It — Estimating the Basel problem (http://www.maa.org/editorial/euler/How%20Euler%20Did%20It%2002%20Estimating%20the%20Basel%20Problem.pdf)* (2003)
- Paule, Peter; Schneider, Carsten (2003). "Computer Proofs of a New Family of Harmonic Number Identities" (PDF). *Adv. in Appl. Math.* **31** (2): 359–378. doi:10.1016/s0196-8858(03)00016-2.
- Wenchang Chu (2004). "A Binomial Coefficient Identity Associated with Beukers' Conjecture on Apery Numbers" (PDF). *The Electronic Journal of Combinatorics* **11**: N15.
- Ayhan Dil, István Mező (2008). "A Symmetric Algorithm for Hyperharmonic and Fibonacci Numbers". *Applied Mathematics and Computation* **206** (2): 942–951. doi:10.1016/j.amc.2008.10.013.
- Zoltán Retkes (2008). "An extension of the Hermite–Hadamard Inequality". *Acta Sci. Math. (Szeged)* **74**: 95–106.

External links

- Weisstein, Eric W., "Harmonic Number" (http://mathworld.wolfram.com/HarmonicNumber.html), *MathWorld*.

This article incorporates material from Harmonic number on PlanetMath, which is licensed under the Creative Commons Attribution/Share-Alike License.

Retrieved from "https://en.wikipedia.org/w/index.php?title=Harmonic_number&oldid=706329176"

Categories: Number theory

- This page was last modified on 22 February 2016, at 19:00.
- Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy. Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.

StackExchange ▾ sign up log in tour help ▾

Mathematics Stack Exchange is a question and answer site for people studying math at any level and professionals in related fields. It's 100% free, no registration required.

Sign up

Here's how it works:

Anybody can ask a question

Anybody can answer

The best answers are voted up and rise to the top

How to count lattice points on a line.

How can we count the number of lattice point on a line, given that the endpoints of the lines are themselves lattice points? I really can't think of how counting lattice points would work, so please provide me some intuition on how lattice points can be counted. Also, what is the relation of the x-distance and y-distance being coprime to existence of lattice points?

(integer-lattices)

asked Jan 5 '14 at 16:25
shaurya gupta
1,951 ● 2 ■ 10 ▲ 31

1 Start by moving everything so that one lattice point is at the origin. The other is at (a, b). You're hoping to find n and k such that (n, k) is on the line from $(0, 0)$ to (a, b) and $0 \leq n \leq a$. Being on the line means that $bn = ak$. Write both k and n as multiples of $d = gcd(k, n)$ and you should be on your way... – John Hughes Jan 5 '14 at 16:35

1 Answer

For the line from (a, b) to (c, d) the number of such points is

$$\gcd(c - a, d - b) + 1.$$

Especially, if the x and y distances are coprime, only the endpoints are lattice points.

answered Jan 5 '14 at 16:34
Hagen von Eitzen
182k ● 13 ■ 176 ▲ 338

Thanks for the formula...But can you please explain it?? – shaurya gupta Jan 5 '14 at 16:36

3 The equation of the line is

$$y = \frac{d - b}{c - a}(x - a) + b$$

Now define $c - a = \alpha \cdot k$ and $d - b = \beta \cdot k$ where k is the greatest common divisor (gcd) of $c - a$ and $d - b$. Then

$$y = \frac{\beta}{\alpha}(x - a) + b$$

holds for integers x and y if and only if $(x - a)$ is a multiple of α. Since $a \leq x \leq c$ we must have $0 \leq (x - a) \leq c - a = \alpha \cdot k$ so there are $k + 1$ multiples of α in this interval. Thus there will be $k + 1$ lattice points. – String Jan 5 '14 at 16:52

Well the question here was to give you the last nonzero digit of LCM of 1 to N… to do that.. we simply counted the number of primes they had in N. It's kind of

LCM FROM 1 to N

like this:

```
#include <bits/stdc++.h>
using namespace std;
#define li long long int
#define rl(n) scanf("%lld", &n)
#define rll(m,n) scanf("%lld %lld", &m, &n)
#define rlll(m,n,o) scanf("%lld %lld %lld", &m, &n, &o)
#define ri(n) scanf("%d", &n)
#define rc(n) scanf("%c", &n)
#define rs(n) gets(s)
#define rst(n) getline(cin,n)
#define rfile(a) freopen(a, "r", stdin)
#define pi acos(-1.00)
#define pb push_back
#define mp make_pair
#define Pr printf
#define For(i,a,b) for(int i = a; i < b; i++)
#define MOD 1000003
#define eps 1e-9
#define ru(n) scanf("%llu",&n)
#define ruuu(m,n,o) scanf("%llu %llu %llu", &m, &n, &o)
#define ui unsigned long long int

int primes[1000001];
int pr_ctr = 0;
bitset<1003000> st(0);

void siv()
{
    int n = 1000000;
    st.set(0,1);
    st.set(1,1);
    primes[pr_ctr++] = 2;
    for(int i = 4; i <= n; i+=2)
    {
        st.set(i,1);
    }
    int sqn = sqrt(n);
    for(int i = 3; i <= sqn ; i+=2)
    {
```

```
        if(st.test(i) == 0)
        {
            for(int j = i*i; j <= n; j+=2*i)
            {
                st.set(j,1);
            }
        }
    }
    for(int i = 3; i <= n; i+=2)
    {
        if(st.test(i) == 0)primes[pr_ctr++]=i;
    }
}

int main()
{
    siv();
    int n, ans, cntr_2, cntr_5;
    while (true)
    {
        ri(n); if(n == 0)break;
        ans = 1; cntr_2 = cntr_5 = 0;
        for (int i = 2; i <= n; i *= 2) cntr_2++;
        for (int i = 5; i <= n; i *= 5) cntr_5++;
        int loops = abs(cntr_2-cntr_5);
        for (int i = 0; i < loops; i++) ans = (ans * 2) % 10;
        for (int i = 0; i < pr_ctr && primes[i] <= n; i++)
        {
            if(primes[i] == 2 || primes[i] == 5) continue;
            for (li j = primes[i]; j <= n; j *= primes[i]) ans = (ans * primes[i]) % 10;
        }
        printf("%d\n", ans);
    }
    return 0;
}
```

Basically what we did here was to count the number of 2’s and 5’s at first. Now the common of 2 and 5’s would all result in 10’s resulting in making trailing zeroes. So we eliminated those 2’s and 5’s. Then ofcourse there will be some 2’s remaining. We multiplied those, and also multiplied any other primes as many times as they went inside n.. except we never multiplied 5. As it would give zeros in the trail.

forthright48

Learning Never Ends

Monday, August 17, 2015

Modular Exponentiation

Problem

Given three positive integers B, P and M, find the value of $B^P \% M$.

For example, $B = 2$, $P = 5$ and $M = 7$, then $B^P \% M = 2^5 \% 7 = 32 \% 7 = 4$.

This problem is known as [1]Modular Exponentiation. In order to solve this problem, we need knowledge about [2]Modular Arithmetic.

Brute Force - Linear Solution $O(P)$

As usual, we first start off with a naive solution. We can calculate the value of $B^P \% M$ in $O($ running a loop from 1 to P and multiplying B to result.

```
1  int bigmodLinear ( int b, int p, int m ) {
2      int res = 1 % m;
3      b = b % m;
4      for ( int i = 1; i <= p; i++ ) {
5          res = ( res * b ) % m;
6      }
7      return res;
8  }
```

A simple solution which will work as long as the value of P is small enough. But for large value example, $P > 10^9$, this code will time out. Also note that in line 2, I wrote $res = 1 \% m$. Som write $res = 1$. This will produce wrong result when the value of P is 0 and value of M is 1.

Divide and Conquer Approach - $O(log_2(P))$

According to [3]Wiki,

> A divide and conquer algorithm works by recursively breaking down a problem into two sub-problems of the same (or related) type (divide), until these become simple enough directly (conquer).

That is, instead of solving the big problem $B^P \% M$, we will solve smaller problems of similar them to get answer to the big problem.

So how do we apply this D&C (Divide and Conquer) idea?

Suppose we are trying to find the value of B^P. Then three thing can happen.

1. **Value of P is 0 (Base Case)**: B^0 is 1. This will be the base case of our recursion functi reach this state, we no longer divide the problem into smaller parts.
2. **Value of P is Even**: Since P is even, we can say that $B^P = B^{\frac{P}{2}} \times B^{\frac{P}{2}}$. For example, $3^6 = 3^3 \times 3^3$. Therefore, instead of calculating the value of $x = B^P$, if we find the valu then we can get the value of x as $x = y \times y$.
3. **Value of P is Odd**: In this case we can simply say that $B^P = B \times B^{P-1}$.

Using these three states, we are can formulate a D&C code.

```
1  int bigmodRecur ( int b, int p, int m ) {
2      if ( p == 0 ) return 1%m; ///Base Case
3
4      if ( p % 2 == 0 ) { ///p is even
5          int y = bigmodRecur ( b, p / 2, m );
6          return ( y * y ) % m; ///b^p = y * y
7      }
8      else {
9          ///b^p = b * b^(p-1)
10         return ( b * bigmodRecur ( b, p - 1, m ) ) % m;
11     }
12 }
```

In line 2 we have the base case. We return $1 \% m$ in case value of m is 1. Next on line 4 we or not. If it is even, we calculate $A^{\frac{P}{2}}$ and return after squaring it. In line 8, we handle the case

At each step we either divide P by 2 or subtract 1 from P and then divide it by 2. Therefore, tl most $2 \times log_2(P)$ steps in D&C approach. Hence complexity is $O(log_2(P))$.

A Better Solution

A better solution for this problem exists. By using Repeated Squaring algorithm, we can find th Exponentiation faster than D&C. Repeated Squaring Algorithm has the same complexity as D iterative, thus does not suffer from recursion overhead.

We need to know about bitwise operations before we learn Repeated Squaring algorithm. Sin cover this topic yet, I won't write about this approach now. Hopefully, once we cover bitwise o write another post

Resource

1. Wikipedia - Modular Exponentiation
2. forthright48 - Introduction to Modular Arithmetic
3. Wikipedia - Divide and conquer algorithms
4. forthright48 - Introduction to Number Systems

Posted by Mohammad Samiul Islam

G+1 Recommend this on Google

forthright48

Learning Never Ends

Thursday, August 20, 2015

Contest Analysis: IUT 6th National ICT Fest 201 12830 - 12839

I participated in this contest last year. Our team "NSU Shinobis" managed to solve 5 and place failed to find the rank list). The IUT 7th ICT National Fest 2015 is going to be held next month, would try to up solve the remaining problems. I tried to compile detailed hints for the problems

UVa 12830 - A Football Stadium

Complexity: $O(N^3)$
Category: Loop, DP, Kadane's Algorithm

Given N points, we have to find the largest area of the rectangle which can fit such that there the rectangle. Points on boundaries of the rectangle are allowed.

Now imagine a rectangle which has no point inside it and also no point on its boundary. Now f boundary of that rectangle. Since there is no point on that boundary, we can extend it up until limit. Same can be said for the bottom, left and right boundary. That is, the largest rectangle w boundary aligned with the limits or points.

Now let us fix the top and lower boundary of the rectangle. How many possible values can the boundary can take values of y-coordinate from one of the N points or the limits 0 and W. Us loops, we can fix them.

Next we need to fix the left and right boundary. Lets us sort the points according to x first and y. Next we iterate over the points. Initially set the left boundary of rectangle aligned to y-axis, t the left boundary of Sand Kingdom.

Now, for each point, if the y coordinate of the point is above or below or on the boundary we f ignore them, cause they don't cause any problem. But if they fall inside, then we need to proc put the right boundary aligned to the x coordinate of this point and calculate this rectangle.

But it's not over. We shift the left boundary over to current right boundary and continue proces points.

This is a classical problem of Kadane's Algorithm. With two nested loops to fix upper and lowe another loop inside iterating over N points makes the complexity $O(N^3)$.

$O(N^2)$ solution is possible, by constructing a histogram for each row ($O(N)$) and the calcu histogram in $O(N)$. But that's not necessary here.

Code: http://ideone.com/FjvTN2

UVa 12831 - Bob the Builder

Complexity: $O(\sqrt{V}E)$
Category: Max Flow, Bipartite Matching, Dinic's Algorithm, Path Cover

Another classical problem, though we had a hard time solving this one. Mainly cause we didn a path cover problem which can be solved using Max Flow.

Take each number provided and generate all child. Consider each number a node and if it pc B from A, then add a directed edge from A to B. When you are done, you will have DAG.

Now split all the nodes in two. We are going to create a bipartite graph now. On the left side, w original nodes and on the right side we will have the copy we got by splitting the nodes. Now, between A and B, add edge in the bipartite graph from A on the left side to B on the right sic

Find the matching (or maximum flow) of this graph by running Dinic's Algorithm. The answer $\text{Total Nodes} - \text{Matching}$.

Code: http://ideone.com/U1VPNB

UVa 12832 - Chicken Lover

Complexity: $O(M)$
Category: Expected Value, Probability, DP, Combination

If a shop can make N different items, and in a single day prepares K items from those N ite many different sets of menus ($total$) can they make? $total = C_K^N$. Now, if they decide to ma day for sure, how many sets of menus ($chicken$) can they make now? $chicken = C_{K-1}^{N-1}$. probability P that if I visit a shop I will get to eat chicken? $P = \frac{chicken}{total}$. And what is the probat eat chicken? $Q = 1 - P$.

So now I know the probability of eating chicken for each shop. How do we find the expected v it using dynamic programming.

The states of dp only be the shop number. $dp(x)$ will give me the expected number of chicke visit all shops starting from x. Our result will be $dp(1)$.

At each state, I have P_i probability that I will eat chicken and Q probability that I will not. So re will be:

$$dp(pos) = P \times (1 + dp(pos + 1)) + Q \times dp(pos + 1)$$
$$dp(pos) = P + P \times dp(pos + 1) + Q \times dp(pos + 1)$$
$$dp(pos) = P + dp(pos + 1) \times (P + Q)$$

$dp(pos) = P + dp(pos + 1)$.

In order to print the result in $\frac{A}{B}$ form, we need to avoid $double$ and use integer arithmetic in al implemented my own fraction class for that purpose.

Code: http://ideone.com/mGKwuL

UVa 12833 - Daily Potato

Complexity: $O(26 \times N)$
Category: String, Manacher's Algorithm

For each query, we have to count the number of palindromes which are substring of S, starts given C and has exactly X occurrences of C in it. Since it deals with palindrome, perhaps it h do with Manacher's Algorithm?

With Manacher's Algorithm, we can find out the maximum length of palindrome in $O(N)$. But can actually generate an array M which gives us, for each center in the extended string E, (extended becomes ^#a#b#a#$) the maximum length of palindrome with that particular c we use this knowledge to solve our problem?

Let us consider the character $'a'$ only. We can easily extend it for other characters by repeati process 26 times.

Suppose we are working with center x. It has a palindrom from it with length y. Therfore, in ex manacher, the palindrome starts from $x - y$ and ends in $x + y$. Now, how many times does palindrome? Using Cumulative Sum it is possible to answer in $O(1)$. Let that occurance be $f = \#\text{ of times 'a' occurs in palindrome with center x}$. Let us mark this in another arra mark it like $\text{freq[f]}++$, meaning we have $freq[f]$ palindromes where $'a'$ occurs f times. But palindrome does not start and end with $'a'$? Simple, we keep on throwing the leading and trai it starts and ends with $'a'$ and it will still have f occurances of $'a'$ in it.

So we repeat this for all possible center. Now, if the query is find number of palindromes that s with $'a'$ and $'a'$ occurs exactly X times, how do we solve it?

First of all, our result will contain $res = freq[X]$ in it. What's more, our result will also contain $res += freq[X + 2] + freq[X + 4] + freq[X + 6] + \ldots$. Why is that? Take any palindrom more than X occurances of $'a'$. Since they start and end with $'a'$, we can just throw them out palindrome and reduce the occurance of $'a'$ in it by 2. After that, we keep on trimming down h that palindrome until we reach $'a'$ again. That is, a palindrome with Y occurrences of $'a'$ can to palindrome with $Y - 2, Y - 4, Y - 6, \ldots$ occurrences of $'a'$.

Instead of getting the result $res = freq[X] + freq[X + 2] + freq[X + 4] + \ldots$ we can jus sum again to find it in $O(1)$ here. Just find cumulative sum of alternate terms.

Code: http://ideone.com/brZKXL

UVa 12834 - Extreme Terror

Complexity: $O(N\ logN)$
Category: Adhoc

This was the easiest problem. Each shop gives me some money ($income$) and then for that s give Godfather some cut ($expense$). So for each shop I get $profit = income - expense$. profit for each shop and then sort them. I can now skip K shops. I will of course skip shops fo negative as long as I am allowed to skip.

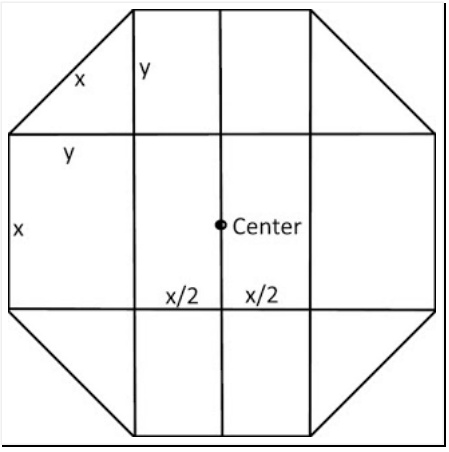
Code: http://ideone.com/s3iHGz

UVa 12835 - Fitting Pipes Again

Complexity: $O(N!\ N^2)$

Category: Geometry, Trigonometry, Permutation, Packing Problem

It's a packing problem. At first glance it seems like a tough problem but it's not. Let us first defi first.



This is the polygon. Each polygon has two horizontal lines through it. We will call them low and side of the polygon has length of x. The radius of the polygon is $r = \frac{x}{2} + y$.

We are given height of each polygon. But first we will need to calculate the value of x and y. \ values using trigonometry.

$$y^2 + y^2 = x^2$$
$$2y^2 = x^2$$
$$x = \sqrt{2y^2}$$

We also know that $h = y + x + y$. From that we can derive:

$$y + x + y = h$$
$$2y + x = h$$
$$2y + \sqrt{2y^2} = h$$
$$2y + y\sqrt{2} = h$$
$$y(2 + sqrt2) = h$$
$$y = \frac{h}{2} + \sqrt{2}$$

With the above, we can find the value of $y = \frac{h}{2} + \sqrt{2}$ and $x = \sqrt{2y^2}$ But why did we find th

Okay, what happens when we put to polygon side by side? In order to solve this problem we
find the distance between the center of two polygon A and B.

Now if two polygons are of same height and they are placed side by side, then the difference
centers will be $d = r_a + r_b$. What happens when two polygons of arbitrary height come besi

3 things can happen when two polygon A and B are placed side by side. A is on left of B.

1. Height of bottom line of A is higher than height of top line of B. In this case, A is so big
inside the radius of A.
2. Height of bottom line of B is higher than height of top line of A. In this case A is so sma
inside the radius of B.
3. Nobody can slide inside each others radius. So $d = r_a + r_b$.

We need to calculate the value of d for step 1 and 2. That can also be easily done using trigo
draw some diagram yourself.

So we used x and y to find the value of d between two polygon. How do we use this to find th
of box?

Suppose we are trying to pack the polygons in some order. Which order? We don't know whi
us minimum width so we try them all. There will be $N!$ order.

So for each order, what will be the minimum of width. First lets take the first polygon, and put it
box such that it touches the left side of the box. We will calculate the center of each polygon r
side of the box. The first box will have center at r_0.

Now take the second box. First imagine there is no polygon in the box. There where will be th
second polygon? Ar r_1. Now, since there is a polygon, we will try to put it beside that one. Wh
center now? $r_0 + d$. We will take the maximum possible center.
Repeat this for the third polygon. Empty box, beside first polygon, beside second polygon. Ta
again. Repeat this for all polygon.

From all polygon, find the one with maximum center position. Add radius of that polygon to it's
width.

Take the minimum width from all permutation.

Code: http://ideone.com/kphGx5

UVa 12836 - Gain Battle Power

Complexity: $O(N^2)$
Category: Interval DP, LIS, Knuth Optimization

First we need to calculate the power of each deatheater. We can do that by calculating LIS fro
and then adding them.

Once we calculate the power, we run an interval dp between 1 and N. The dp will have state
Inside the dp a loop between start and end will run, choosing different cut sections. We will tak
minimum value.

But this results in $O(N^3)$ solution. Using Knuth's optimization we can reduce it to $O(N^2)$.

Code: http://ideone.com/XD6hZP

UVa 12837 - Hasmot Ali Professor

Complexity: $O(100 \times |S|^2)$
Category: String, Trie, Data Structure

We will create two trie trees. The first one will contain all the queries. Take a query and conca
special way. Take the first string of the query and add a '$\#$' and then reverse the second stri
and attach it to result. That is, if we have abc and pqr as query, then special string will be abc
special strings for each query in the first tries.

Now, let us process the main string S. We will take each of its suffix and insert into the second
inserting the suffixes, each time a new node is created, we can say that we found a unique su

Each time we find a unique substring we will process it further. Take the unique substring, and
generate all possible special strings (the one we made using query strings) with the first and
the unique string. We don't need to take more than 10 characters from each end.

For each of those special string we made from unique substring, we will query the first trie with
node where it ends. We will add one to that node number in a global array.

Once we finish processing all nodes in the second trie, we will simply traverse the first trie acc
query and print the result found in that node.

Code: http://ideone.com/fsYMxI

UVa 12838 - Identity Redemption

Complexity: Unknown
Category: Matching on General Graph

I didn't manage to solve this yet. It looks like matching on general graph.

UVa 12839 - Judge in Queue

Complexity: $O(N\ logN)$
Category: Data Structure, Heap

We want to minimize the waiting time for each person. So first we will sort the people in desce
we will create a priority queue, which will contain all the service center along with the informati
that service center will be free. Initially all service centers are free.

Now, we take each person and take out the service center that will get free at the earliest time
to originally wait x minutes and it took y minutes for the service to get free again. So the pers

$x + y$ minutes. We insert the service again inside the heap and update its free time by the tim one person.

Repeat and keep track of the highest time a person had to wait.

Code: http://ideone.com/KVHnTX

I hope the details are clear enough for everyone to understand. Let me know if you find any n

Posted by Mohammad Samiul Islam

Labels: Analysis, Contest, UVa

G+1 Recommend this on Google

No comments:

Post a Comment

Leave comments for Queries, Bugs and Hugs.

Enter your comment...

Comment as: Unknown (Goo

Publish Preview

Newer Post Home

Subscribe to: Post Comments (Atom)

Blog Archive

▼ 2015 (35)

Labels

Analysis Arithmetic Function Backtrack Bi Complexity Contest D&C Divisors Factorial

```
/**********Template Starts Here**********/
#include <bits/stdc++.h>

#define pb push_back
#define ff first
#define ss second
#define MP make_pair
#define FOR(i,x,y) for(vlong i = (x) ; i <= (y) ; ++i)
#define ROF(i,x,y) for(vlong i = (y) ; i >= (x) ; --i)
#define UNIQUE(V) (V).erase(unique((V).begin(),(V).end()),(V).end())
#define MIN(a,b) ((a)<(b)?(a):(b))
#define MAX(a,b) ((a)>(b)?(a):(b))
#define ALL(x) (x).begin(),(x).end()

using namespace std;

typedef long long vlong;
typedef pair < int, int > pii;
typedef vector<pii> vii;
typedef vector<int> vi;

/**********Template Ends Here**********/

vector<pii> p;

int main () {
    #ifdef forthright48
    freopen ( "input.txt", "r", stdin );
    //freopen ( "output.txt", "w", stdout );
    #endif // forthright48

    int kase;
    scanf ( "%d", &kase );

    int cnt = 0;

    while ( kase-- ) {
        int l, w;
        scanf ( "%d %d", &l, &w );

        int n;
        scanf ( "%d", &n );

        p.clear();

        vi height; ///Will list all possible heights for top and bottom boundary

        FOR(i,0,n-1) {
            int a, b;
            scanf ( "%d %d", &a, &b );
            p.pb ( MP(a,b) ); ///Insert point
            height.pb ( b ); ///Insert possible boundary
        }

        height.pb ( 0 ); ///Don't forget the Sand Kingdom Boundaries
        height.pb ( w );
```

14, 0.21s, 0.31s

```
57.
58.          sort ( ALL(height) ); ///Make the heights sorted and unique
59.          UNIQUE(height);
60.
61.          int m = height.size();
62.
63.          p.pb ( MP(0,0) ); ///Add dummy points to indicate left and right wall of sand
      kingdom
64.          p.pb ( MP(l,0) );
65.
66.          n = p.size(); ///Update N
67.
68.          sort ( ALL(p) ); ///Sort all points
69.
70.          vlong res = 0;
71.
72.          FOR(i,0,m-1) { ///Set lower boundary of rectangle
73.              FOR(j,i+1,m-1) { ///Set upper boundary of rectangle
74.                  int d = height[i], u = height[j];
75.
76.                  int cur = 0; ///Current point which is acting as left boundary of rec
     tangle
77.
78.                  p[0].ss = d; ///Make sure first and last point, which are Sand Kingdo
     m's wall don't get ignored.
79.                  p[n-1].ss = d;
80.
81.                  FOR ( k, 1, n - 1 ) {
82.                      if ( p[k].ss < d || p[k].ss > u ) continue; ///Not inside rectang
     le
83.
84.                      int width = p[k].ff - p[cur].ff; ///Inside rectangle. Calculate w
     idth
85.
86.                      res = MAX(res,width*(u-d)); ///Update result
87.
88.                      if ( p[k].ss > d && p[k].ss < u ) { ///If strictly inside, update
       left wall.
89.                          cur = k;
90.                      }
91.                  }
92.              }
93.          }
94.
95.          printf ("Case %d: %lld\n", ++cnt, res );
96.
97.      }
98.
99.      return 0;
100.  }
101.
```

Time limit exceeded #stdin #stdout 5s 200064KB

comments (0)

```
/**********Template Starts Here**********/
#include <bits/stdc++.h>

#define pb push_back
#define ff first
#define ss second
#define MP make_pair
#define FOR(i,x,y) for(vlong i = (x) ; i <= (y) ; ++i)
#define ROF(i,x,y) for(vlong i = (y) ; i >= (x) ; --i)
#define CLR(x,y) memset(x,y,sizeof(x))
#define UNIQUE(V) (V).erase(unique((V).begin(),(V).end()),(V).end())
#define MIN(a,b) ((a)<(b)?(a):(b))
#define MAX(a,b) ((a)>(b)?(a):(b))
#define ALL(x) (x).begin(),(x).end()
#define SZ(x) ((vlong)(x).size())

using namespace std;

typedef long long vlong;
typedef vector<int> vi;

const vlong inf = 2147383647;

/**********Template Ends Here**********/

struct node {
    int x, y, next, cap, cost;
};

#define NODE 20010
#define EDGE 5000010

///Flow Template
struct FLOW {
    int source, sink;

    int head[NODE];
    void clear() {
        e = 0;
        CLR(head,-1);
    }

    node edge[EDGE]; int e;
    void addEdge ( int u, int v, int cap ) {
        edge[e].x = u; edge[e].y = v; edge[e].cap = cap;
        edge[e].next = head[u]; head[u] = e; e++;
        edge[e].x = v; edge[e].y = u; edge[e].cap = 0;
        edge[e].next = head[v]; head[v] = e; e++;
    }

    int vis[NODE], q[NODE], now[NODE];
    bool bfs ( ) {
        memset ( vis, -1, sizeof vis );
        vis[source] = 0;
        int ini = 0, qend = 0;
        q[qend++] = source;
```

```
57.
58.          while ( ini < qend && vis[sink] == -1 ) {
59.              int s = q[ini++];
60.              int i;
61.              for (i=head[s];i!=-1;i= edge[i].next){
62.                  int t = edge[i].y;
63.                  if ( vis[t] == -1 && edge[i].cap){
64.                      vis[t] = vis[s] + 1;
65.                      q[qend++] = t;
66.                  }
67.              }
68.          }
69.          if ( vis[sink] != -1 ) return true;
70.          else return false;
71.      }
72.      int dfs ( int s, int f ) {
73.          if ( f == 0 ) return 0;
74.          if ( s == sink ) return f;
75.          for ( int &i=now[s];i!=-1;i=edge[i].next){
76.              int t = edge[i].y;
77.              if ( vis[s] + 1 != vis[t] ) continue;
78.              int pushed=dfs(t,MIN(f,edge[i].cap));
79.              if ( pushed ) {
80.                  edge[i].cap -= pushed;
81.                  edge[i^1].cap += pushed;
82.                  return pushed;
83.              }
84.          }
85.          return 0;
86.      }
87.
88.      int maxFlow ( int limit, int flow ) {
89.          int res = 0;
90.          while ( 1 ) {
91.              if ( flow == 0 ) break;
92.              if ( bfs () == false ) break;
93.              int i;
94.              for ( i=0;i<=limit;i++)now[i]= head[i];
95.              while (int pushed=dfs(source,flow ) ) {
96.                  res += pushed; ///Can overflow depending on Max Flow
97.                  flow -= pushed;
98.              }
99.          }
100.         return res;
101.     }
102. }graph;
103.
104. int vis[10010], vv = 1;
105. int qqq[10010]; ///Queue for BFS
106. int total = 0;
107.
108. ///Generate all children
109. void bfs ( int s, int lim ) {
110.     int qh, qt;
111.     qh = qt = 0;
112.     qqq[qt++] = s;
```

```
113.
114.      vis[s] = vv;
115.
116.      while ( qh < qt ) {
117.          s = qqq[qh++];
118.
119.          FOR(i,0,15) {
120.              if ( s & ( 1 << i ) ) {
121.                  int t = s + ( 1 << i ); ///Generate Child
122.                  if ( t <= lim ) {
123.                      if ( vis[t] != vv ) { ///Has not been generated before
124.                          vis[t] = vv;
125.                          qqq[qt++] = t;
126.                      }
127.                  }
128.              }
129.          }
130.      }
131.  }
132.
133.  int mp[10010];
134.
135.  int main () {
136.      #ifdef forthright48
137.      freopen ( "input.txt", "r", stdin );
138.      //freopen ( "output.txt", "w", stdout );
139.      #endif // forthright48
140.
141.      int kase;
142.      scanf ( "%d", &kase );
143.
144.      int cnt = 0;
145.
146.      while ( kase-- ) {
147.          int n;
148.          scanf ( "%d", &n );
149.
150.          int l;
151.          scanf ( "%d", &l );
152.
153.          graph.clear(); ///Clear Graph
154.
155.          vv++; ///Clear visit array
156.          FOR(i,0,n-1) {
157.              int seed;
158.              scanf ( "%d", &seed );
159.              bfs ( seed, l ); ///Generate all children of seed
160.          }
161.
162.          vi child;
163.          FOR(i,1,l) {
164.              if ( vis[i] == vv ) child.pb ( i ); ///Collect all child in one place
165.          }
166.
167.          int cur = 1;
```

```
168.          ///Map all children to new nodes. This is necessary to lower node number in F
      low
169.          FOR(i,0,SZ(child)-1) {
170.              mp[child[i]] = cur++;
171.          }
172.
173.          int node = child.size();
174.          graph.source = 0;
175.          graph.sink = node + node + 1; ///Assign source and sink
176.
177.          FOR(i,1,node) {
178.              graph.addEdge(0,i,1); ///Add edge from source to left side
179.              graph.addEdge(i+node,node + node + 1,1); ///Add edge from right side to s
      ink
180.          }
181.
182.          FOR(c,0,SZ(child)-1) {
183.              int s = child[c];
184.              FOR(i,0,15) {
185.                  if ( s & ( 1 << i ) ) {
186.                      int t = s + ( 1 << i );
187.                      if ( t <= l ) {
188.                          graph.addEdge ( c+1, node + mp[t], 1 ); ///Add edge between l
      eft and right side
189.                      }
190.                  }
191.              }
192.          }
193.
194.          int m = graph.maxFlow(node+node+1,inf); ///Find flow
195.
196.          printf ( "Case %d: %d\n", ++cnt, SZ(child) - m ); ///Result is total - flow
197.      }
198.
199.      return 0;
200.  }
201.
```

Runtime error #stdin #stdout 0s 101504KB

comments (0)

stdin copy

```
Standard input is empty
```

stdout

```
Standard output is empty
```

```
/**********Template Starts Here**********/
#include <bits/stdc++.h>

#define pb push_back
#define nl puts ("")
#define sp printf ( " " )
#define phl printf ( "hello\n" )
#define ff first
#define ss second
#define POPCOUNT __builtin_popcountll
#define RIGHTMOST __builtin_ctzll
#define LEFTMOST(x) (63-__builtin_clzll((x)))
#define MP make_pair
#define FOR(i,x,y) for(vlong i = (x) ; i <= (y) ; ++i)
#define ROF(i,x,y) for(vlong i = (y) ; i >= (x) ; --i)
#define CLR(x,y) memset(x,y,sizeof(x))
#define UNIQUE(V) (V).erase(unique((V).begin(),(V).end()),(V).end())
#define MIN(a,b) ((a)<(b)?(a):(b))
#define MAX(a,b) ((a)>(b)?(a):(b))
#define NUMDIGIT(x,y) (((vlong)(log10((x))/log10((y))))+1)
#define SQ(x) ((x)*(x))
#define ABS(x) ((x)<0?-(x):(x))
#define FABS(x) ((x)+eps<0?-(x):(x))
#define ALL(x) (x).begin(),(x).end()
#define LCM(x,y) (((x)/gcd((x),(y)))*(y))

using namespace std;

typedef long long vlong;
typedef unsigned long long uvlong;
typedef pair < int, int > pii;
typedef pair < vlong, vlong > pll;
typedef vector<pii> vii;
typedef vector<int> vi;


inline vlong gcd ( vlong a, vlong b ) {
    a = ABS ( a ); b = ABS ( b );
    while ( b ) { a = a % b; swap ( a, b ); } return a;
}

/**********Template Ends Here**********/
///Custom Fraction Class
class Fraction {
    vlong numerator, denominator;

    void simplify() {
        vlong g = gcd ( numerator, denominator );
        numerator /= g;
        denominator /= g;
        if ( denominator < 0 ) {
            numerator *= -1;
            denominator *= -1;
        }
    }

```

```
public:
    Fraction () {
        numerator = 0;
        denominator = 1;
    }
    Fraction ( vlong a, vlong b ) {
        numerator = a;
        denominator = b;
        simplify();
    }
    Fraction ( vlong x ) {
        numerator = x;
        denominator = 1;
    }
    void operator = ( Fraction b ) {
        numerator = b.numerator;
        denominator = b.denominator;
    }

    Fraction operator + ( Fraction b ) { ///Addition
        Fraction res;
        res.denominator = ABS( LCM(denominator,b.denominator) );
        res.numerator = (res.denominator/denominator)*numerator +
(res.denominator/b.denominator)*b.numerator;
        res.simplify();
        return res;
    }

    void print() {
        printf ( "%lld/%lld",numerator, denominator );
    }
};

///Calculate combinations
vlong nck[25][25];
void calcNCK(int n) {
    nck[0][0] = 1;
    FOR(i,1,n) {
        FOR(j,0,n) {
            if ( j == 0 ) nck[i][j] = 1;
            else nck[i][j] = nck[i-1][j-1] + nck[i-1][j];
        }
    }
}
void precal() {
    calcNCK(20);
}

int menu[10010], item[10010];

///DP to find expected value
Fraction dp ( int pos ) {
    if ( pos < 0 ) return Fraction(0ll);

    Fraction res;

```

```
112.      vlong total = nck[ menu[pos] ][ item[pos] ];
113.      vlong chicken = nck[ menu[pos]-1 ][ item[pos]-1 ];
114.
115.      Fraction p ( chicken, total ), q ( total - chicken, total ); ///Calculate probabi
      lities.
116.
117.      res = p + dp ( pos - 1 );
118.
119.      return res;
120.  }
121.
122.  int main () {
123.      precal();
124.      #ifdef forthright48
125.      freopen ( "input.txt", "r", stdin );
126.      //freopen ( "output.txt", "w", stdout );
127.      #endif // forthright48
128.
129.      int kase, cnt = 0;
130.      scanf ( "%d", &kase );
131.
132.      while ( kase-- ) {
133.          int m;
134.          scanf ( "%d", &m );
135.
136.          FOR(i,0,m-1) {
137.              scanf ( "%d %d", &menu[i], &item[i] );
138.          }
139.
140.          Fraction res = dp ( m - 1 );
141.
142.          printf ( "Case %d: ", ++cnt ); res.print(); nl;
143.      }
144.
145.      return 0;
146.  }
147.
```

Time limit exceeded #stdin #stdout 5s 3496KB

comments (0)

stdin copy

```
Standard input is empty
```

stdout

```
Standard output is empty
```

```
#include <bits/stdc++.h>

#define pb push_back
#define FOR(i,x,y) for(vlong i = (x) ; i <= (y) ; ++i)
#define ROF(i,x,y) for(vlong i = (y) ; i >= (x) ; --i)
#define CLR(x,y) memset(x,y,sizeof(x))
#define MIN(a,b) ((a)<(b)?(a):(b))
#define MAX(a,b) ((a)>(b)?(a):(b))
#define ALL(x) (x).begin(),(x).end()
#define LCM(x,y) (((x)/gcd((x),(y)))*(y))
#define SZ(x) ((vlong)(x).size())

using namespace std;

typedef long long vlong;

#define MANALEN 200000

/*
1. First assign string to manacher, arr[]
2. Call init
3. Call Preprocess to convert string
4. Call calc to calculate array p[]
*/

char arr[MANALEN+10], brr[MANALEN+10];
int p[MANALEN+10];
int cum[26][MANALEN+10];
int fr[26][MANALEN+10];

struct MANACHER {
    int an, bn, mxpalin, mxcenter;

    void init() {
        //Assign string to arr
        an = strlen ( arr );
        CLR(cum,0);
        CLR(fr,0);
    }

    //aba -> ^#a#b#a#$
    void preprocess() { ///Converts string
        int cur = 0;
        brr[cur++] = '^';
        FOR(i,0,an-1) {
            brr[cur++] = '#';
            brr[cur++] = arr[i];
        }
        brr[cur++] = '#';
        brr[cur++] = '$';
        brr[cur] = 0;
        bn = cur;
    }

    void calc() { ///Calculates Manacher Array
        int c = 0, r = 0;
```

```
        FOR(i,1,bn-2) {
            int mi = c - (i-c);

            p[i] = (r > i )? MIN ( r - i, p[mi] ): 0;

            while ( brr[i+p[i]+1] == brr[i-p[i]-1] ) {
                p[i]++;
            }

            if ( i + p[i] > r ) {
                r = i + p[i];
                c = i;
            }
        }

        mxpalin = 0; mxcenter = 0;
        FOR(i,1,bn-2) {
            if ( p[i] > mxpalin ) {
                mxpalin = p[i];
                mxcenter = i;
            }
        }

    }

    ///Build cumulative array for each letter
    ///cum[0][x] gives me number of 'a' from 0 to x.
    void extra() {
        FOR(c,0,25){
            cum[c][0] = 0;
            FOR(joker,1,bn-1){
                int cur = 0;
                if ( brr[joker] >= 'a' && brr[joker] <= 'z' ) {
                    if ( brr[joker] - 'a' == c ) cur = 1;
                }
                cum[c][joker] = cum[c][joker-1] + cur;
            }
        }
    }

    ///Calculate for each center, how many "c" it has in it.
    ///If it has x, then mark[x]++; Meaning, I have a center which has x number of c.
    void moreExtra() {
        FOR(c,0,25){
            FOR(joker,1,bn-2) {
                int palin = p[joker];
                ///Now find how many palindrome exist in the palindrome with center j
oker
                int st = joker - p[joker]; ///RTE not possible. st >= 1
                int en = joker + p[joker]; ///en <= bn-2

                ///st-1 >= 0, cause brr[0] is not part of any palindrome with center
 >= 1.

                if ( st - 1 < 0 ) while(1);

```

```
                int freq = cum[c][en] - cum[c][st-1];
                //debug ( joker, brr[joker], st, en, freq, c );
                fr[c][freq]++;
            }
        }

        ///Calculate cumulate such that fr[c][x] has sum of fr[c][x] + fr[c][x+2] + f
r[c][x+4] ...
        FOR(c,0,25){
            fr[c][bn-1] = 0;
            fr[c][bn] = 0;
            ROF(joker,0,bn-2) {
                fr[c][joker] += fr[c][joker+2];
            }
        }
    }

}mana;


char buf[100000+10];

int main () {
    int kase, cnt = 0;
    scanf ( "%d", &kase );

    while ( kase-- ) {
        int n;
        scanf ( "%d", &n );
        scanf ( "%s", arr );

        mana.init();
        mana.preprocess();
        mana.calc();

        mana.extra();
        mana.moreExtra();

        int q;
        scanf ( "%d", &q );
        scanf ( "%s", buf );

        printf ( "Case %d:\n", ++cnt );
        FOR(iron,0,q-1) {
            int x;
            scanf ( "%d", &x );

            int c = buf[iron] - 'a';

            if ( x <= 0 || x > n ) {
                printf ( "0\n" );
                continue;
            }

            int res = 0;
            res = fr[c][x];
```

```
166.
167.                printf ( "%d\n", res );
168.            }
169.        }
170.
171.        return 0;
172.    }
173.
```

Runtime error #stdin #stdout 0.05s 45312KB

comments (0)

stdin copy

```
Standard input is empty
```

stdout copy

```
#include <bits/stdc++.h>

#define pb push_back
#define nl puts ("")
#define sp printf ( " " )
#define phl printf ( "hello\n" )
#define ff first
#define ss second
#define POPCOUNT __builtin_popcountll
#define RIGHTMOST __builtin_ctzll
#define LEFTMOST(x) (63-__builtin_clzll((x)))
#define MP make_pair
#define FOR(i,x,y) for(vlong i = (x) ; i <= (y) ; ++i)
#define ROF(i,x,y) for(vlong i = (y) ; i >= (x) ; --i)
#define CLR(x,y) memset(x,y,sizeof(x))
#define UNIQUE(V) (V).erase(unique((V).begin(),(V).end()),(V).end())
#define MIN(a,b) ((a)<(b)?(a):(b))
#define MAX(a,b) ((a)>(b)?(a):(b))
#define NUMDIGIT(x,y) (((vlong)(log10((x))/log10((y))))+1)
#define SQ(x) ((x)*(x))
#define ABS(x) ((x)<0?-(x):(x))
#define FABS(x) ((x)+eps<0?-(x):(x))
#define ALL(x) (x).begin(),(x).end()
#define LCM(x,y) (((x)/gcd((x),(y)))*(y))
#define SZ(x) ((vlong)(x).size())
#define NORM(x) if(x>=mod)x-=mod;

using namespace std;

typedef long long vlong;

int arr[1000010];

int main () {

    int kase;
    scanf ( "%d", &kase );

    int cnt = 0;
    while ( kase-- ) {
        int n, k;
        scanf ( "%d %d", &n, &k );

        FOR(i,0,n-1) {
            scanf ( "%d", &arr[i] );
            arr[i] *= -1;
        }

        FOR(i,0,n-1) {
            int t;
            scanf ( "%d", &t );
            arr[i] += t;
        }

        ///arr[i] contains ( what that shop gives shamsu - what shamsu gives to godfa
ther)
```

```
56.
57.         vlong res = 0;
58.         sort ( arr, arr + n ); ///Now, losses come first and profit comes later.
59.
60.         FOR(i,0,n-1) {
61.             if ( arr[i] < 0 ) { ///This is a loss
62.                 if ( k ) { ///I can still skip loss
63.                     k--;
64.                     continue;
65.                 }
66.             }
67.             res += arr[i];
68.         }
69.
70.         printf ( "Case %d: ", ++cnt );
71.         if ( res <= 0 ) printf ( "No Profit\n" );
72.         else printf ( "%lld\n", res );
73.     }
74.
75.     return 0;
76. }
77.
```

Runtime error #stdin #stdout 0s 7360KB

comments (0)

stdin copy

```
Standard input is empty
```

stdout

```
Standard output is empty
```

```
/**********Template Starts Here**********/
#include <bits/stdc++.h>

#define pb push_back
#define nl puts ("")
#define sp printf ( " " )
#define phl printf ( "hello\n" )
#define ff first
#define ss second
#define POPCOUNT __builtin_popcountll
#define RIGHTMOST __builtin_ctzll
#define LEFTMOST(x) (63-__builtin_clzll((x)))
#define MP make_pair
#define FOR(i,x,y) for(vlong i = (x) ; i <= (y) ; ++i)
#define ROF(i,x,y) for(vlong i = (y) ; i >= (x) ; --i)
#define CLR(x,y) memset(x,y,sizeof(x))
#define UNIQUE(V) (V).erase(unique((V).begin(),(V).end()),(V).end())
#define MIN(a,b) ((a)<(b)?(a):(b))
#define MAX(a,b) ((a)>(b)?(a):(b))
#define NUMDIGIT(x,y) (((vlong)(log10((x))/log10((y))))+1)
#define SQ(x) ((x)*(x))
#define ABS(x) ((x)<0?-(x):(x))
#define FABS(x) ((x)+eps<0?-(x):(x))
#define ALL(x) (x).begin(),(x).end()
#define LCM(x,y) (((x)/gcd((x),(y)))*(y))
#define SZ(x) ((vlong)(x).size())
#define NORM(x) if(x>=mod)x-=mod;

using namespace std;

typedef long long vlong;
typedef vector<int> vi;

const vlong inf = 2147383647;
const double eps = 1e-11;

struct polygon {
    vlong h;
    double x, y;

    void init ( vlong _h ) {
        h = _h;
        calcY();
        calcX();
    }
    void calcY (){
        y = h / ( 2 + sqrt ( 2 ) );
    }
    void calcX() {
        x = sqrt ( 2 * y * y );
    }
}pol[10];

vi arr;

double center[10];
```

```
57.
58.  int main () {
59.      int kase, cnt = 0;
60.      scanf ( "%d", &kase );
61.
62.      while ( kase-- ) {
63.          int n;
64.          scanf ( "%d", &n );
65.
66.          arr.clear();
67.
68.          double mxh = -1;
69.          FOR(i,0,n-1){
70.              arr.pb ( i );
71.              int t;
72.              scanf ( "%d", &t );
73.              pol[i].init(t);
74.              if ( t > mxh ) mxh = t;
75.          }
76.
77.          double res = inf * inf;
78.
79.          do {
80.              double curWidth = 0;
81.              curWidth = pol[arr[0]].y * 2 + pol[arr[0]].x;
82.              center[0] = curWidth / 2;
83.
84.              FOR(iron,1,n-1){ ///Try to fit this polygon as close as possible
85.                  ///Try to push center of this pol near a pol, then check for conflict
86.                  int b = arr[iron];
87.                  center[iron] = ( pol[b].y * 2 + pol[b].x ) / 2; ///Empty
88.                  FOR(joker,0,iron-1){
89.                      int a = arr[joker];
90.                      if ( pol[a].h > pol[b].h + eps && pol[a].y > pol[b].y + pol[b].x
     + eps ) {
91.                          ///A is bigger than B
92.                          double d = 2 * pol[b].y + 1.5 * pol[b].x;
93.                          double probableCenter = center[joker] + pol[a].x / 2 + d;
94.
95.                          if ( probableCenter > center[iron] + eps ) {
96.                              center[iron] = probableCenter;
97.                          }
98.                      }
99.                      else if ( pol[b].h > pol[a].h + eps && pol[b].y > pol[a].y + pol[a]
     .x + eps ) {
100.                         ///B is bigger than A
101.
102.                         double d = pol[a].x / 2 + pol[a].y + pol[a].x + pol[a].y + po
     l[b].x / 2;
103.                         double probableCenter = center[joker] + d;
104.
105.                         if ( probableCenter > center[iron] + eps ) {
106.                             center[iron] = probableCenter;
107.                         }
108.                     }
109.                     else {
```

```
                        double d = pol[a].x / 2 + pol[a].y + pol[b].y + pol[b].x / 2;
                        double probableCenter = center[joker] + d;
                        if ( probableCenter > center[iron] + eps ) {
                            center[iron] = probableCenter;
                        }
                    }
                }
                double tempWidth = center[iron] + pol[b].x / 2 + pol[b].y;
                if ( tempWidth > curWidth + eps ) curWidth = tempWidth;
            }

            double vol = 8 * mxh * curWidth;

            if ( vol + eps < res ) res = vol;
        }while ( next_permutation ( ALL(arr) ) );

        printf ( "Case %d: %.10lf\n", ++cnt, res + eps );
    }

    return 0;
}
```

Runtime error #stdin #stdout 0s 3456KB

comments (0)

stdin copy

```
Standard input is empty
```

stdout

```
Standard output is empty
```

Sphere Research Labs (http://sphere-research.com?utm_campaign=permanent&utm_medium=footer&utm_source=ideor
Sphere Engine™ (http://sphere-engine.com/?utm_campaign=permanent&utm_medium=footer&utm_sourc
home (/) terms of use (/terms) api (/sphere-engine) language faq (/faq) credits (/credits) feedback
(/ideone/tools/bug/form/1/link/kphgx5/compiler/44) desktop mobile (/switch/mobile/l2twaed4nq=

```
/**********Template Starts Here**********/
#include <bits/stdc++.h>

#define pb push_back
#define nl puts ("")
#define sp printf ( " " )
#define phl printf ( "hello\n" )
#define ff first
#define ss second
#define POPCOUNT __builtin_popcountll
#define RIGHTMOST __builtin_ctzll
#define LEFTMOST(x) (63-__builtin_clzll((x)))
#define MP make_pair
#define FOR(i,x,y) for(vlong i = (x) ; i <= (y) ; ++i)
#define ROF(i,x,y) for(vlong i = (y) ; i >= (x) ; --i)
#define CLR(x,y) memset(x,y,sizeof(x))
#define UNIQUE(V) (V).erase(unique((V).begin(),(V).end()),(V).end())
#define MIN(a,b) ((a)<(b)?(a):(b))
#define MAX(a,b) ((a)>(b)?(a):(b))
#define NUMDIGIT(x,y) (((vlong)(log10((x))/log10((y))))+1)
#define SQ(x) ((x)*(x))
#define ABS(x) ((x)<0?-(x):(x))
#define FABS(x) ((x)+eps<0?-(x):(x))
#define ALL(x) (x).begin(),(x).end()
#define LCM(x,y) (((x)/gcd((x),(y)))*(y))
#define SZ(x) ((vlong)(x).size())
#define NORM(x) if(x>=mod)x-=mod;

using namespace std;

typedef long long vlong;

const vlong inf = 2147383647;

/**********Template Ends Here**********/
int arr[1010];

vlong LIS[1010], LDS[1010], pwr[1010], cum[1010];

vlong memo[1010][1010];
int knuth[1010][1010], cc = 1;
int done[1010][1010];

vlong dp ( int st, int en ) {
    if ( st == en ) {
        knuth[st][en] = st;
        return 0;
    }
    if ( done[st][en] == cc ) return memo[st][en];

    vlong res = inf * inf;

    dp ( st, en - 1 ); ///First calculate this
    dp ( st + 1, en ); ///First calculate this

    int kst = knuth[st][en-1]; ///Knuth's Optimization
```

```
    int ken = knuth[st+1][en]; ///Knuth's Optimization
    if ( ken == en ) ken--;

    FOR ( iron, kst, ken ) {
        vlong cost = cum[en] - cum[st-1];
        vlong tres = cost + dp ( st, iron ) + dp ( iron + 1, en );
        if ( tres <= res ) {
            res = tres;
            knuth[st][en] = iron;
        }
    }

    done[st][en] = cc;
    return memo[st][en] = res;
}


int main () {
    #ifdef forthright48
    freopen ( "input.txt", "r", stdin );
    //freopen ( "output.txt", "w", stdout );
    #endif // forthright48

    int kase, cnt = 0;
    scanf ( "%d", &kase );

    while ( kase-- ) {
        int n;
        scanf ( "%d", &n );

        FOR(iron,0,n-1){
            scanf ( "%d", &arr[iron] );
        }

        FOR(iron,0,n-1){
            LIS[iron] = 1;
            LDS[iron] = 1;
        }


        ///Find LIS from front
        FOR(iron,0,n-1){
            FOR(joker,iron+1,n-1){
                if ( arr[iron] < arr[joker] && LIS[iron] + 1 > LIS[joker] ) {
                    LIS[joker] = LIS[iron] + 1;
                }
            }
        }
        ///Find LIS from back
        ROF(iron,0,n-1){
            ROF(joker,0,iron-1){
                if ( arr[iron] < arr[joker] && LDS[iron] + 1 > LDS[joker] ) {
                    LDS[joker] = LDS[iron] + 1;
                }
            }
        }
```

```
113.
114.            ///Calculate PWR. Made it 1-indexed
115.            FOR(iron,0,n-1){
116.                pwr[iron+1] = LIS[iron] + LDS[iron] - 1;
117.                cum[iron+1] = pwr[iron+1] + cum[iron];
118.            }
119.
120.            ///Now calculate DP
121.            cc++;
122.            vlong res = dp ( 1, n );
123.
124.            printf ( "Case %d: %lld\n", ++cnt, res );
125.        }
126.
127.        return 0;
128.    }
129.
```

Time limit exceeded #stdin #stdout 5s 19432KB

comments (0)

stdin copy

```
Standard input is empty
```

stdout

```
Standard output is empty
```

```
/**********Template Starts Here**********/
#include <bits/stdc++.h>

#define pb push_back
#define nl puts ("")
#define sp printf ( " " )
#define phl printf ( "hello\n" )
#define ff first
#define ss second
#define POPCOUNT __builtin_popcountll
#define RIGHTMOST __builtin_ctzll
#define LEFTMOST(x) (63-__builtin_clzll((x)))
#define MP make_pair
#define FOR(i,x,y) for(vlong i = (x) ; i <= (y) ; ++i)
#define ROF(i,x,y) for(vlong i = (y) ; i >= (x) ; --i)
#define CLR(x,y) memset(x,y,sizeof(x))
#define UNIQUE(V) (V).erase(unique((V).begin(),(V).end()),(V).end())
#define MIN(a,b) ((a)<(b)?(a):(b))
#define MAX(a,b) ((a)>(b)?(a):(b))
#define NUMDIGIT(x,y) (((vlong)(log10((x))/log10((y))))+1)
#define SQ(x) ((x)*(x))
#define ABS(x) ((x)<0?-(x):(x))
#define FABS(x) ((x)+eps<0?-(x):(x))
#define ALL(x) (x).begin(),(x).end()
#define LCM(x,y) (((x)/gcd((x),(y)))*(y))
#define SZ(x) ((vlong)(x).size())
#define NORM(x) if(x>=mod)x-=mod;

using namespace std;

typedef long long vlong;

/**********Template Ends Here**********/
#define SIZE 6000000
#define RSIZE 6000000

char buf[1010];

struct node {
    int child[27];

    void clear () {
        CLR(child,-1);
    }
}tnQ[SIZE+10], tnS[RSIZE+10];
int freeIndexQ, freeIndexS;

void clearQ () {
    freeIndexQ = 1;
    tnQ[0].clear();
}
void clearS() {
    freeIndexS = 1;
    tnS[0].clear();
}

```

```
int res[SIZE+10]; ///Okay

char luffy[100];

///Inserts into the trie for query
int insertQ ( ){
    int cur = 0;
    int len = strlen(luffy);

    FOR(i,0,len-1) {
        char c = luffy[i];
        if ( c == '#' ) c = 26; ///Okay
        else c -= 'a';

        if ( tnQ[cur].child[c] == -1 ) { ///No child
            tnQ[freeIndexQ].clear(); ///Clear it first
            tnQ[cur].child[c] = freeIndexQ++;
        }

        cur = tnQ[cur].child[c];
    }
    return cur;
}

char zoro[100];
///Update each end of query string in query trie
void queryQ ( ){
    int cur = 0;
    int len = strlen ( zoro );

    FOR(i,0,len-1) {
        char c = zoro[i];
        if ( c == '#' ) c = 26;
        else c -= 'a';

        if ( tnQ[cur].child[c] == -1 ) {
            return;
        }
        cur = tnQ[cur].child[c];
    }
    res[cur]++;

}

///Create all possible query strings from unique substring
void queryAll ( int st, int en ) {
    int len = en - st;
    int l = MIN(len,10);

    int zind = 0;
    FOR(i,0,l-1){
        zoro[zind++] = buf[st+i]; ///First put the prefix

        int nzind = zind; ///Now start a new index of suffix

        zoro[nzind++] = '#'; ///First of # to separate them
```

```
113.          FOR(j,0,l-1){
114.              zoro[nzind++] = buf[en-1-j];
115.              zoro[nzind] = 0; ///A possible query is finished
116.              queryQ();
117.          }
118.      }
119.  }
120.
121.  ///Insert suffix into second trie for main string
122.  void insertS ( int st, int len ){
123.      int cur = 0;
124.
125.      FOR(i,0,len-1) {
126.          char c = buf[st+i];
127.          c -= 'a';
128.
129.          bool newNode = false;
130.          if ( tnS[cur].child[c] == -1 ) {
131.              tnS[freeIndexS].clear();
132.              tnS[cur].child[c] = freeIndexS++;
133.              newNode = true;
134.          }
135.
136.          cur = tnS[cur].child[c];
137.
138.          if ( newNode ) { ///New node found
139.              queryAll ( st, st+i+1 );
140.          }
141.      }
142.  }
143.
144.  char query[50010][25];
145.
146.  int main () {
147.
148.      int kase, cnt = 0;
149.      scanf ( "%d", &kase );
150.
151.      while ( kase-- ) {
152.          CLR(res,0); ///clear res to 0
153.          clearQ(); ///Clear trie for query
154.          clearS(); ///Clear trie for String
155.
156.          scanf ( "%s", buf ); ///Take main string in buf
157.
158.          int q;
159.          scanf ( "%d", &q ); ///Number of query
160.
161.          FOR(i,0,q-1) {
162.              scanf ( "%s", luffy ); ///Take first string of in luffy
163.
164.              int luf = 0;
165.              luf = strlen ( luffy );
166.
167.              luffy[luf] = '#'; ///Attach a # at end
168.              luf++;
```

```
169.
170.              scanf ( "%s", luffy + luf  ); ///take input from end of luffy
171.              reverse ( luffy + luf, luffy + strlen ( luffy ) ); ///reverse as needed.
172.
173.              insertQ ();
174.
175.              strcpy ( &query[i][0], luffy ); ///Keep a copy of query to process later
176.
177.          }
178.
179.          int len = strlen ( buf );
180.
181.          FOR(i,0,len-1) {
182.              insertS( i, len - i ); ///Start and length
183.          }
184.
185.          printf ( "Case %d:\n", ++cnt );
186.          FOR(i,0,q-1){
187.              strcpy ( luffy, &query[i][0] );
188.              int t = insertQ ( );
189.              printf ( "%d\n", res[t] );
190.          }
191.      }
192.
193.      return 0;
194.  }
195.
```

Runtime error #stdin #stdout 0s 148KB comments (0)

stdin copy

```
Standard input is empty
```

stdout

```
Standard output is empty
```

```
/**********Template Starts Here**********/
#include <bits/stdc++.h>

#define pb push_back
#define nl puts ("")
#define sp printf ( " " )
#define phl printf ( "hello\n" )
#define ff first
#define ss second
#define POPCOUNT __builtin_popcountll
#define RIGHTMOST __builtin_ctzll
#define LEFTMOST(x) (63-__builtin_clzll((x)))
#define MP make_pair
#define FOR(i,x,y) for(vlong i = (x) ; i <= (y) ; ++i)
#define ROF(i,x,y) for(vlong i = (y) ; i >= (x) ; --i)
#define CLR(x,y) memset(x,y,sizeof(x))
#define UNIQUE(V) (V).erase(unique((V).begin(),(V).end()),(V).end())
#define MIN(a,b) ((a)<(b)?(a):(b))
#define MAX(a,b) ((a)>(b)?(a):(b))
#define NUMDIGIT(x,y) (((vlong)(log10((x))/log10((y))))+1)
#define SQ(x) ((x)*(x))
#define ABS(x) ((x)<0?-(x):(x))
#define FABS(x) ((x)+eps<0?-(x):(x))
#define ALL(x) (x).begin(),(x).end()
#define LCM(x,y) (((x)/gcd((x),(y)))*(y))
#define SZ(x) ((vlong)(x).size())
#define NORM(x) if(x>=mod)x-=mod;

using namespace std;

typedef long long vlong;

/**********Template Ends Here**********/

vlong arr[100010];
vlong serve[10100];

struct node {
    int id;
    vlong cur;

    node ( int a, vlong b ) {
        id = a;
        cur = b;
    }
};
class compare {
public:
    bool operator () ( const node &a, node &b ) {
        if ( a.cur < b.cur ) return false;
        else if ( a.cur == b.cur && serve[a.id] < serve[b.id] ) return false;
        else return true;
    }
};

int main () {
```

```
57.
58.       int kase;
59.       scanf ( "%d", &kase );
60.
61.       int cnt = 0;
62.
63.       while ( kase-- ) {
64.           int n, m;
65.           scanf ( "%d %d", &n, &m );
66.
67.           FOR(i,0,n-1) {
68.               scanf ( "%lld", &arr[i] );
69.           }
70.
71.           sort ( arr, arr + n, greater<vlong>() ); ///Sort in reverse order
72.
73.           priority_queue<node,vector<node>,compare> pq;
74.           FOR(i,0,m-1){
75.               scanf ( "%d", &serve[i] );
76.               pq.push ( node(i,0) );
77.           }
78.
79.           vlong res = 0;
80.           FOR(i,0,n-1){
81.               node temp = pq.top(); pq.pop();
82.
83.               vlong need = arr[i] + temp.cur;
84.               res = MAX(res,need);
85.
86.               pq.push ( node ( temp.id, temp.cur + serve[temp.id] ) );
87.           }
88.
89.           printf ( "Case %d: %lld\n", ++cnt, res );
90.       }
91.
92.       return 0;
93.   }
94.
```

Runtime error #stdin #stdout 0s 4276KB

comments (0)

stdin copy

```
Standard input is empty
```

stdout

```
Standard output is empty
```

Sphere Research Labs (http://sphere-research.com?utm_campaign=permanent&utm_medium=footer&utm_source=ideor
Sphere Engine™ (http://sphere-engine.com/?utm_campaign=permanent&utm_medium=footer&utm_sourc
home (/) terms of use (/terms) api (/sphere-engine) language faq (/faq) credits (/credits) feedback
(/ideone/tools/bug/form/1/link/kvhntx/compiler/44) desktop mobile (/switch/mobile/l0twsg5uwa=

forthright48

Learning Never Ends

Tuesday, August 25, 2015

Contest Analysis: BUET Inter-University Program Contest - 2011, UVa 12424 - 12432

Last week (2015-08-21) we practiced with this set. Some of the problems didn't have proper c caused some headaches. This contest originally followed Google Code Jam format with two s small and large) for each problem. Each of the sub-problems had a different constraint on the uploaded it online, they removed both constraints from some problems, making things confus guessing the constraints when solving them.

During contest time, we managed to solve B, E, and H. I was stuck with F. Should have mo were far easier than F. Bad decision from my end.

UVa 12424 - Answering Queries on a Tree

Complexity: $O(logN)$
Category: Segment Tree, DFS, LCA

At first it seemed like a problem on Heavy Light Decomposition. But after a moment I realized without HLD.

Run a dfs from any node and make the tree rooted tree. Now consider the color X. For every color X, the edge from its parent will have cost 1. Since the root can also have color X, we ne dummy node 0 and make that root. Now, run another dfs and calculate the distance from the node. We repeat this for all 10 color and build 10 trees. Let us call tree with color of node X, T

Now, whenever we have an update to change node u from color X to color Y, it means that u in tree T_X will now have a cost 0. Since its cost decreased, the distance from root of all the subtree of u will be lowered by 1. We can apply this change using preorder traversal dfs + se the opposite for T_Y. Increase the value of all nodes under subtree u in T_Y.

And for query u and v, find the distance between these two nodes in each of the 10 trees. Th $O(logN)$ using LCA + Segment Tree.

Code: http://ideone.com/UEU9J4

UVa 12425 - Best Friend

Complexity: $O(\sqrt{N} + \sqrt[3]{N} \times \text{Complexity of } \phi(N))$
Category: Number Theory, Euler Phi, GCD

So for given integers N and X, we have to find out how many number I are there such that g

In order to calculate $gcd(N, I) \leq X$, I first need to be able to calculate how many numbers a $gcd(N, I) = Y$. I started with a small value of Y first. So the first thing I asked myself, how m there such that $gcd(N, I) = 1$? The answer is $\phi(N)$. Then I asked how many are there suc $gcd(N, I) = 2$?

This took a moment, but I soon realized, if $gcd(N, I)$ is equal 2, then if I divide both N and I $gcd(\frac{N}{2}, \frac{I}{2})$ will be equal to 1. Now, all I had to calculate how many I are there such that $gcd($ answer is $\phi(\frac{N}{2})$.

Therefore, there are $\phi(\frac{N}{Y})$ number of I such that $gcd(N, I) = Y$.

Great. Now we can calculate number of I such that $gcd(N, I) = Y$. Now all we need to do i all number of I for which $gcd(N, I) = Y$ and $Y \leq X$. GCD of N and I can be as high as 1 loop won't do.

Next I thought, what are the possible values of $g = gcd(N, I)$? Since g is a number that divi must be a divisor of N. So I simply calculated all the divisors of N using a $\sqrt{N}$ loop. Next for calculated $\phi(\frac{N}{d})$ which is the number of ways to chose I such that $gcd(N, I) = d$.

Now, for each query X, all I needed to do was find sum of $\phi(\frac{N}{d})$, where d is a divisor of N a all values of $\phi(\frac{N}{d})$ was precalculated, using cumalitive sum I answered it in $O(1)$.

Code: http://ideone.com/WrIH0C

UVa 12426 - Counting Triangles

Complexity: $O(N^2\ logN)$
Category: Two Pointer, Geo, Binary Search

We need to choose three points of a convex hull such that the area does not exceed K.

First, we need to know, for two points i and j, $j > i$, what is the point m such that (i, j, m) fo triangle possible. This can be done by selecting two points with two loops and then using tern used two pointer technique to find m for every pair of (i, j) in $O(N)$.

After that, for ever pair of (i, j) I used binary search twice. Once on points $[j + 1, m]$ and on $[m + 1, n]$. Using binary search I found the point x which gives the highest area that does n Then I added $x - j - 1$ in first BS and $n - 1 - m$ in second case.

Careful about not adding a degenerate triangle. Hanlde the edge cases properly.

Code: http://ideone.com/JtJjt9

UVa 12427 - Donkey of the Sultan

Complexity: $O(1)$
Category: Game Theory, Combinatorics, Nim Game, Bogus Nim

This game can be converted to into a game of pile. Let the distance between the left boundar be x and the distance between diamond and silver coin be y. Now, instead of playing on the play with two piles of height x (first pile) and y (second pile).

Moving gold coin left is same as removing one coin from the first pile. Moving diamond coin le increasing second pile by one. Moving silver coin left is same as removing one coin from the s Moving gold and silver coin left is same as removing one coin from both first and second pile.

Now, note that is a state of (x, y) is losing and I try to increase the second pile by one, my op simply take one from second pile and put me back to same situation. That is, increasing the s effect in this game. This is a "Bogus" move. We will ignore this move.

Next I drew a 5×5 table and calculated the Win or Lose state of each possible (x, y). I foun both x and y are even, the first person loses. That is the second person, me, wins the game v both are even.

What are the possible values of x? x can only be between a_1 and a_2. Find number of even r range. Let this be r_1. Next, y can be between c_1 and c_2. Let the number of even numbers in The distance between gold and diamond has no effect in this game. So we can put any amou

Therefore, number of possible combination is $r_1 \times r_2 \times (b_2 - b_1 + 1)$.

Code: http://ideone.com/6q2LZT

UVa 12428 - Enemy at the Gates

Complexity: $O(\sqrt{M})$
Category: Adhoc

This was the easiest problem in the set. In this problem, we are given N nodes and M edges out how many leaves can this graph have if we maximize leaves.

The problem says the graph will be connected. So I created a star-shaped graph with it using Let the center node be 0 and remaining nodes be the numbers from 1 to $N - 1$. This gives n edges. Then I checked if I still have more edges left? If I do, then I need to sacrifice a critical e

The first edge that gets sacrifice is a special case. This is because, no matter which two node will reduce the number of critical edge by 2. There is no helping it. So let's connect 1 and 2 an edge by 1 and critical edge by 2.

Then if we still have more edges left, then we need to sacrifice another critical length. From nc step only one critical edge will be removed. Also, this time we can add 2 edges to the graph b critical edge. Connect an edge between $(3, 1)$ and $(3, 2)$. Reduce M by 2 and critical edge l

Next is node 4. We can add 3 edges by removing 1 critical edge now. We need to keep on d edges finish.

I guess it is possible to solve the problem in $O(1)$ but I didn't bother with it. Number of test cas
anyway.

Code: http://ideone.com/X3bWyZ

UVa 12429 - Finding Magic Triplets

Complexity: $O(N\ logN)$
Category: Binary Indexed Tree, Number Theory

We have to find the number of triplets such that $a + b^2 \equiv c^3$ and $a \leq b \leq c$. Here is what I d

Iterate over c from N to 1 using a loop. Let the loop iterator be i. Now for each i, we do the fo

1. Find $x = (i \times i \times i)\ \%\ k$ and store x in a BIT. BIT stores all occurrences of c and by it
   order we make sure that all c in BIT is greater or equal to i.

2. Let $b = (i \times i)\ \%\ k$. This is the current b we are working with. In BIT we have c which a
   we need is to find possible values of a.

3. Now, take any value of c from BIT. For this c and current b, how many ways can we cho
   $a + b \equiv c$. Notice that a can be anything from 1 to k, or 0 to $k - 1$. Therefore, no matte
   value of c we can chose 1 number from 0 to $k - 1$ so that $a + b \equiv c$.

   Let $y = \frac{i}{k}$. This means we have y segments, in which value of a spans from 0 to $k - 1$
   each possible c we take from BIT. Let $total$ be number of c we have inserted till now. S
   $total \times y$ to $result$.

4. But it's not over yet. What about the values of a that we did not use. To be more exact, I
   $r > 0$, then we have unused values of a. We need to use these.

   But we have to be careful. We have used up values of a from 1 to $y \times k$. So the remain
   to r. 0 is not included.

   Since we don't have all values from 0 to $k - 1$, we are not able to handle all possible va
   How many can we handle exactly?

   $a + b \equiv c$
   $a \equiv c - b$

   But we need a can only be between 1 and r.

   $1 \leq c - b \leq r$
   $\therefore 1 + b \leq c \leq r + b$

   Find out how many c are there with values between $1 + b$ and $r + b$ from BIT. Let this v
   each of those c we can use one value of a between 1 to r. So add z to $result$.

And that's it. Be careful when handling edge cases in BIT.

Code: http://ideone.com/iMBUmU

UVa 12430 - Grand Wedding

Complexity: $O(N\ logN)$
Category: DFS, Bicoloring, Binary Search

Constraint was missing. I assumed $N <= 10^6$.

We have to remove some edges and then assign guards such that no edge is left unguarded guards in both ends. We have to output the maximize the value of the edges that we remove.

Binary search over the value of edges which we need to remove. Now, let's say we removed value greater or equal to x. How do we decide that a valid assignment of guard is possible in

Suppose all the nodes in the graph are colored white. If we assign guard in a node, then we c two white nodes cannot be adjacent cause that would mean the edge is left unguarded. Two cannot be adjacent otherwise guards will chat with each other. That is both ends of an edge c color in the graph. This is possible when the graph has no cycle of odd length. If a graph has bicolorable. So all we need to do is check if the graph is bicolorable.

Now, two special cases. If we can bicolor the graph without removing any edge, then answer remove all edges then answer is -1. Why? Cause in the problem it is said we can remove **pr** edges. The full set is not a proper subset.

Code: http://ideone.com/R6lkje

UVa 12431 - Happy 10/9 Day

Complexity: $O(\ (logN)^2\ )$
Category: Divide and Conquor, Modular Exponentiation, Repeated Squaring

We have to find the value of:

$$(d \times b^0 + d \times b^1 + d \times b^2 + \ldots + d \times b^{n-1})\ \%\ M$$
$$(d(b^0 + b^1 + b^2 + \ldots + b^{n-1}))\ \%\ M.$$

Therefore, the problem boils down to finding the value of $(b^0 + b^1 + b^2 + \ldots + b^{n-1})\ \%\ M$. V the formula of geometric progression since M and $b - 1$ may not be coprime. So what do we divide and conquor. How? Let me show you an example.

Suppose $f(n) = (b^0 + b^1 + b^2 + \ldots + b^n)$. Let us find $f(5)$.
$f(5) = b^0 + b^1 + b^2 + b^3 + b^4 + b^5$
$f(5) = (b^0 + b^1 + b^2) + b^3(b^0 + b^1 + b^2)$
$f(5) = f(2) + b^3 \times f(2)$
$f(5) = f(2) \times (1 + b^3)$.

From this we can design a D&C in following way:

$\text{n is odd: } f(n) = f(\frac{n}{2}) \times (1 + b^{n/2+1})$
$\text{n is even: } f(n) = f(n-1) + b^n$

$f(0) = 1$

Just apply modular arithmatic with the whole things. Also note that M can be as larg as 10^{12}, overflow if they are multiplied directly, even after making them small by modulus M. So use re metho to multiply them.

Code: http://ideone.com/Fn6vzW

UVa 12432 - Inked Carpets

Complexity:
Category:

I didn't try this problem yet. I guess this was the stopper.

It was a good set. I espcially enjoyed solving B, D and F.

Posted by Mohammad Samiul Islam

G+1 Recommend this on Google

Labels: Analysis, Contest, UVa

1 comment:

ম্যাট্রিক্স.কোড March 9, 2016 at 12:02 PM

I have solved the first problem in the list using different approach. segment tree but without laz
My code : Code

Reply

Enter your comment...

Comment as: Unknown (Goo

Publish Preview

Leave comments for Queries, Bugs and Hugs.

Newer Post Home

Subscribe to: Post Comments (Atom)

```
/**********Template Starts Here**********/
#include <bits/stdc++.h>

#define pb push_back
#define nl puts ("")
#define sp printf ( " " )
#define phl printf ( "hello\n" )
#define ff first
#define ss second
#define POPCOUNT __builtin_popcountll
#define RIGHTMOST __builtin_ctzll
#define LEFTMOST(x) (63-__builtin_clzll((x)))
#define MP make_pair
#define FOR(i,x,y) for(vlong i = (x) ; i <= (y) ; ++i)
#define ROF(i,x,y) for(vlong i = (y) ; i >= (x) ; --i)
#define CLR(x,y) memset(x,y,sizeof(x))
#define UNIQUE(V) (V).erase(unique((V).begin(),(V).end()),(V).end())
#define MIN(a,b) ((a)<(b)?(a):(b))
#define MAX(a,b) ((a)>(b)?(a):(b))
#define NUMDIGIT(x,y) (((vlong)(log10((x))/log10((y))))+1)
#define SQ(x) ((x)*(x))
#define ABS(x) ((x)<0?-(x):(x))
#define FABS(x) ((x)+eps<0?-(x):(x))
#define ALL(x) (x).begin(),(x).end()
#define LCM(x,y) (((x)/gcd((x),(y)))*(y))
#define SZ(x) ((vlong)(x).size())
#define NORM(x) if(x>=mod)x-=mod;

using namespace std;

typedef long long vlong;
typedef unsigned long long uvlong;
typedef pair < int, int > pii;
typedef pair < vlong, vlong > pll;
typedef vector<pii> vii;
typedef vector<int> vi;

const vlong inf = 2147383647;

/**********Template Ends Here**********/

#define LCANODE 100010
#define LCADEPTH 20
vi adj[LCANODE];
struct LCA{
    private:
    int tab[LCANODE][LCADEPTH], lev[LCANODE], vis[LCANODE], q[LCANODE], cc, N, M, roo
t;

    public:
    int par[LCANODE];

    private:
    void bfs ( int s ) {
        vis[s] = cc;
        lev[s] = 0;
```

```
        par[s] = -1; ///Set par[root] = -1

        int qh = 0, qt = 0;
        q[qt++] = s;
        int t;
        while ( qh < qt ) {
            s = q[qh++];
            FOR(i,0,SZ(adj[s])-1){
                t = adj[s][i];
                if ( vis[t] != cc ) {
                    par[t] = s;
                    vis[t] = cc;
                    lev[t] = lev[s] + 1;
                    q[qt++] = t;
                }
            }
        }
    }

    void calculate (){
        int i,j,p;
        for (i=0;i<=N;i++){
           tab[i][0]=par[i];
        }

        for(i=1;i<=M;i++){
            for(j=0;j<=N;j++){
                p=tab[j][i-1];
                if(p==-1) tab[j][i]=-1;
                else tab[j][i]=tab[p][i-1];
            }
        }
    }

    public:
    void clear () {
        cc++;
        FOR(i,0,N) adj[i].clear();
    }

    LCA() {
        cc = 1;
    }

    void setVar ( int n, int r ) {
        N = n;

        M = 0;
        int temp = n+1;
        while ( temp ) {
            M++;
            temp /= 2;
        }
        root = r;
        clear();
    }
```

```
    void precal () {
        bfs ( root );
        calculate();
    }
    int findLCA(int s,int t){
        int dif,pos,i;
        if(lev[s]!=lev[t] ) {
            if(lev[s]>lev[t]) swap( s, t );
            dif=lev[t]-lev[s];
            while(dif){
                pos=RIGHTMOST(dif);
                t=tab[t][pos];
                dif-=1<<pos;
            }
        }
        if (s==t)return s;
        for(i=M;i>=0;i--){
            if(tab[s][i]!=tab[t][i]){
                s=tab[s][i];
                t=tab[t][i];
            }
        }
        return tab[s][0];
    }
}lca;

int child[100010];
int pretrav[100010], pre = 0;
int preStart[100010];
void dfs ( int s, int p ) {
    child[s] = 1;
    preStart[s] = pre;
    pretrav[pre++] = s;

    FOR(i,0,SZ(adj[s])-1){
        int t = adj[s][i];
        if ( t == p ) continue;
        dfs ( t, s );
        child[s] += child[t];
    }
}

int color[100010], arr[11][100010];

void dfs2 ( int s, int p, int d, int c ) {
    arr[c][s] = d;

    FOR(i,0,SZ(adj[s])-1){
        int t = adj[s][i];
        if ( t == p ) continue;
        int cost = 0;
        if ( color[t] == c ) cost = 1;

        dfs2 ( t, s, d + cost, c );
    }
}
```

```
168.
169.  struct node {
170.      int val;
171.      int lazy;
172.  }tn[11][4*100000];
173.
174.  void build ( int col, int t, int b, int e ) {
175.      int m = ( b + e ) / 2;
176.      int u = t * 2;
177.      int v = u + 1;
178.
179.      tn[col][t].lazy = 0; ///lazyClear();
180.
181.      if ( b == e ) {
182.          tn[col][t].val = arr[col][pretrav[b]];
183.          return;
184.      }
185.
186.      build( col, u, b, m );
187.      build( col, v, m + 1, e );
188.  }
189.
190.  int qb, qe, qv;
191.  void lazyPropagate ( node &p, node &u, node &v ) {
192.      u.lazy += p.lazy;
193.      v.lazy += p.lazy;
194.  }
195.
196.  void update ( int col, int t, int b, int e ) {
197.      int m = ( b + e ) / 2;
198.      int u = t * 2;
199.      int v = u + 1;
200.
201.      if ( qb <= b && e <= qe ) {
202.          tn[col][t].lazy += qv;///lazyUpdate();
203.          tn[col][t].val += tn[col][t].lazy;///calculate();
204.          if ( b != e ) lazyPropagate ( tn[col][t], tn[col][u], tn[col][v] );
205.          tn[col][t].lazy = 0; ///LazyClear
206.          return;
207.      }
208.      if ( qe < b || e < qb ) {
209.          tn[col][t].val += tn[col][t].lazy;///calculate();
210.          if ( b != e ) lazyPropagate ( tn[col][t], tn[col][u], tn[col][v] );
211.          tn[col][t].lazy = 0; ///LazyClear
212.          return;
213.      }
214.
215.      lazyPropagate ( tn[col][t], tn[col][u], tn[col][v] );
216.      tn[col][t].lazy = 0; ///LazyClear
217.
218.      update( col, u, b, m );
219.      update( col, v, m + 1, e );
220.
221.  }
222.
223.  int qvv[11];
```

```
224.
225.  void query ( int t, int b, int e ) {
226.      int m = ( b + e ) / 2;
227.      int u = t * 2;
228.      int v = u + 1;
229.
230.      if ( qb <= b && e <= qe ) {
231.          FOR(i,1,10) tn[i][t].val += tn[i][t].lazy;///calculate();
232.          if ( b != e ) {
233.              FOR(i,1,10) lazyPropagate ( tn[i][t], tn[i][u], tn[i][v] );
234.          }
235.          FOR(i,1,10) tn[i][t].lazy = 0; ///LazyClear
236.          FOR(i,1,10) qvv[i] = tn[i][t].val;
237.          return;
238.      }
239.      if ( qe < b || e < qb ) {
240.          FOR(i,1,10) tn[i][t].val += tn[i][t].lazy;///calculate();
241.          if ( b != e ) {
242.              FOR(i,1,10) lazyPropagate ( tn[i][t], tn[i][u], tn[i][v] );
243.          }
244.          FOR(i,1,10) tn[i][t].lazy = 0; ///LazyClear
245.          return;
246.      }
247.
248.      FOR(i,1,10) lazyPropagate ( tn[i][t], tn[i][u], tn[i][v] );
249.      FOR(i,1,10) tn[i][t].lazy = 0; ///LazyClear
250.
251.      query( u, b, m );
252.      query( v, m + 1, e );
253.
254.  }
255.
256.  int main () {
257.
258.      int kase;
259.      scanf ( "%d", &kase );
260.
261.      while ( kase-- ) {
262.          int n, m;
263.          scanf ( "%d %d", &n, &m );
264.
265.          lca.setVar( n, 0 );
266.
267.          CLR(color,0);
268.          FOR(i,1,n){
269.              int c;
270.              scanf ( "%d", &c );
271.              color[i] = c; ///Set color
272.          }
273.
274.          FOR(i,0,n-2){
275.              int a, b;
276.              scanf ( "%d %d", &a, &b );
277.              adj[a].pb ( b );
278.              adj[b].pb ( a );
279.          }
```

```
280.          adj[0].pb ( 1 ); ///Dummy node
281.
282.          lca.precal();
283.
284.          pre = 0;
285.          dfs ( 0, -1 ); ///Calculate child, starting position in pre-traversal
286.
287.          ///Calculate distance in each color tree
288.          FOR(i,1,10){
289.              dfs2 ( 0, -1, 0, i );
290.          }
291.
292.
293.          ///Preparations complete. Start segment tree
294.          FOR(i,1,10) {
295.              build ( i, 1, 0, n );
296.          }
297.
298.          while ( m-- ) {
299.              int t;
300.              scanf ( "%d", &t );
301.
302.              if ( t == 0 ) {
303.                  int u, c;
304.                  scanf ( "%d %d", &u, &c );
305.
306.                  ///Change color from color[u] to c
307.                  int p = color[u];
308.                  ///Update the subtree of u of color p with -1
309.                  qb = preStart[u]; qe = qb + child[u] - 1; qv = -1;
310.                  update ( p, 1, 0, n );
311.
312.                  ///Update the subtree of u of color c with +1
313.                  qv = 1;
314.                  update ( c, 1, 0, n );
315.                  color[u] = c;
316.              }
317.              else {
318.                  int u, v;
319.                  scanf ( "%d %d", &u, &v );
320.
321.                  ///Find distance in each color tree
322.                  int res = 0;
323.
324.                  int tres[11];
325.                  CLR(tres,0);
326.
327.                  qb = preStart[u]; qe = qb;
328.                  query( 1, 0, n );
329.
330.                  FOR(i,1,10) tres[i] = qvv[i];
331.
332.
333.                  qb = preStart[v]; qe = qb;
334.                  query( 1, 0, n );
335.
```

```
336.                 FOR(i,1,10) tres[i] += qvv[i];
337.
338.                 int par = lca.findLCA( u, v );
339.
340.                 qb = preStart[par]; qe = qb;
341.                 query( 1, 0, n );
342.                 FOR(i,1,10) tres[i] -= 2 * qvv[i];
343.
344.                 FOR(i,1,10){
345.                     int col = color[par];
346.                     if ( col == i ) tres[i]++;
347.                 }
348.
349.                 FOR(i,1,10) res = MAX(res,tres[i]);
350.
351.                 printf ( "%d\n", res );
352.             }
353.         }
354.     }
355.
356.     return 0;
357. }
358.
```

Runtime error #stdin #stdout 0s 54248KB

comments (0)

stdin copy

```
Standard input is empty
```

stdout

```
Standard output is empty
```

Sphere Research Labs (http://sphere-research.com?utm_campaign=permanent&utm_medium=footer&utm_source=ideor
Sphere Engine™ (http://sphere-engine.com/?utm_campaign=permanent&utm_medium=footer&utm_sourc
home (/) terms of use (/terms) api (/sphere-engine) language faq (/faq) credits (/credits) feedback
(/ideone/tools/bug/form/1/link/ueu9j4/compiler/44) desktop mobile (/switch/mobile/l1vfvtlkna==

```
/**********Template Starts Here**********/
#include <bits/stdc++.h>

#define pb push_back
#define nl puts ("")
#define sp printf ( " " )
#define phl printf ( "hello\n" )
#define ff first
#define ss second
#define POPCOUNT __builtin_popcountll
#define RIGHTMOST __builtin_ctzll
#define LEFTMOST(x) (63-__builtin_clzll((x)))
#define MP make_pair
#define FOR(i,x,y) for(int i = (x) ; i <= (y) ; ++i)
#define ROF(i,x,y) for(int i = (y) ; i >= (x) ; --i)
#define CLR(x,y) memset(x,y,sizeof(x))
#define UNIQUE(V) (V).erase(unique((V).begin(),(V).end()),(V).end())
#define MIN(a,b) ((a)<(b)?(a):(b))
#define MAX(a,b) ((a)>(b)?(a):(b))
#define NUMDIGIT(x,y) (((int)(log10((x))/log10((y))))+1)
#define SQ(x) ((x)*(x))
#define ABS(x) ((x)<0?-(x):(x))
#define FABS(x) ((x)+eps<0?-(x):(x))
#define ALL(x) (x).begin(),(x).end()
#define LCM(x,y) (((x)/gcd((x),(y)))*(y))
#define SZ(x) ((int)(x).size())

using namespace std;

typedef long long vlong;
typedef unsigned long long uvlong;
typedef pair < int, int > pii;
typedef pair < vlong, vlong > pll;
typedef vector<pii> vii;
typedef vector<int> vi;

const vlong inf = 2147383647;
const double pi = 2 * acos ( 0.0 );
const double eps = 1e-9;

/**********Template Ends Here**********/
vector<int> prime;
char stat[1000100];

///Generate primes till n
void sieve( int n ) {
    prime.pb ( 2 );
    int sqrtn = sqrt ( n );

    for ( int i = 3; i <= sqrtn; i += 2 ) {
        if ( stat[i] == 0 ) {
            for ( int j = i * i; j <= n; j += 2 * i ) stat[j] = 1;
        }
    }
    for ( int i = 3; i <= n; i += 2 ) if ( stat[i] == 0 ) prime.pb ( i );
}
```

```
57.
58.  ///Calculate phi(n)
59.  vlong calcPhi ( vlong n ) {
60.      vlong res = n;
61.      int sqrtn = sqrt ( n );
62.      for ( int i = 0; i < prime.size() && prime[i] <= sqrtn; i++ ) {
63.          if ( n % prime[i] == 0 ) {
64.              while ( (n % prime[i]) == 0 ) {
65.                  n /= prime[i];
66.              }
67.              res /= prime[i];
68.              res *= prime[i] - 1;
69.          }
70.      }
71.      if ( n != 1 ) {
72.          res /= n;
73.          res *= n - 1;
74.      }
75.
76.      return res;
77.  }
78.
79.  vector < vlong > ddd, phi;
80.
81.  int main () {
82.      sieve ( 1000000 );
83.
84.      int kase, cnt = 0;
85.      scanf ( "%d", &kase );
86.
87.      while ( kase-- ) {
88.          vlong n, q;
89.          scanf ( "%lld %lld", &n, &q );
90.
91.          ///Find all divisors of n
92.          ddd.clear();
93.          int sqrtn = sqrt ( n ) + eps;
94.          FOR(i,1,sqrtn) {
95.              if ( ( n % i ) == 0 ) {
96.                  ddd.pb ( i );
97.                  if ( n / i != i ) ddd.pb ( n / i );
98.              }
99.          }
100.
101.         sort ( ALL(ddd) );
102.
103.         phi.clear();
104.         FOR(i,0,SZ(ddd)-1){
105.             phi.pb ( calcPhi( n / ddd[i] ) );
106.             if ( i ) phi[i] += phi[i-1]; ///Make it cumulative array of phi value
107.         }
108.
109.         printf ( "Case %d\n", ++cnt );
110.         while ( q-- ) {
111.             vlong x;
112.             scanf ( "%lld", &x );
```

```
113.
114.                if ( x >= n ) { ///All
115.                    printf ( "%lld\n", n );
116.                    continue;
117.                }
118.                if ( x < 1 ) { ///None
119.                    printf ( "%lld\n", 0 );
120.                    continue;
121.                }
122.
123.                int pos = upper_bound ( ALL(ddd), x ) - ddd.begin();
124.
125.                pos--;
126.
127.                printf ( "%lld\n", phi[pos] );
128.            }
129.        }
130.
131.        return 0;
132.    }
133.
```

Time limit exceeded #stdin #stdout 5s 4952KB

comments (0)

stdin copy

```
Standard input is empty
```

stdout

```
Standard output is empty
```

Sphere Research Labs (http://sphere-research.com?utm_campaign=permanent&utm_medium=footer&utm_source=ideor
Sphere Engine™ (http://sphere-engine.com/?utm_campaign=permanent&utm_medium=footer&utm_sourc
home (/) terms of use (/terms) api (/sphere-engine) language faq (/faq) credits (/credits) feedback
(/ideone/tools/bug/form/1/link/wrih0c/compiler/44) desktop mobile (/switch/mobile/l1dysugwqw=

```
/***********Template Starts Here***********/
#include <bits/stdc++.h>

#define pb push_back
#define nl puts ("")
#define sp printf ( " " )
#define phl printf ( "hello\n" )
#define ff first
#define ss second
#define POPCOUNT __builtin_popcountll
#define RIGHTMOST __builtin_ctzll
#define LEFTMOST(x) (63-__builtin_clzll((x)))
#define MP make_pair
#define FOR(i,x,y) for(vlong i = (x) ; i <= (y) ; ++i)
#define ROF(i,x,y) for(vlong i = (y) ; i >= (x) ; --i)
#define CLR(x,y) memset(x,y,sizeof(x))
#define UNIQUE(V) (V).erase(unique((V).begin(),(V).end()),
(V).end())
#define MIN(a,b) ((a)<(b)?(a):(b))
#define MAX(a,b) ((a)>(b)?(a):(b))
#define NUMDIGIT(x,y) (((vlong)(log10((x))/log10((y))))+1)
#define SQ(x) ((x)*(x))
#define ABS(x) ((x)<0?-(x):(x))
#define FABS(x) ((x)+eps<0?-(x):(x))
#define ALL(x) (x).begin(),(x).end()
#define LCM(x,y) (((x)/gcd((x),(y)))*(y))
#define SZ(x) ((vlong)(x).size())
#define NORM(x) if(x>=mod)x-=mod;

using namespace std;

typedef long long vlong;
typedef unsigned long long uvlong;
typedef pair < int, int > pii;
typedef pair < vlong, vlong > pll;
typedef vector<pii> vii;
typedef vector<int> vi;

const vlong inf = 2147383647;
```

```
39. const double pi = 2 * acos ( 0.0 );
40. const double eps = 1e-9;
41.
42.
43. /***********Template Ends Here***********/
44. struct {
45.     int n;
46. }_g;
47. pll point[1010];
48.
49. vlong getArea ( int a, int b, int c ) {
50.     if ( b >= _g.n ) b -= _g.n;
51.     if ( c >= _g.n ) c -= _g.n;
52.     vlong res = point[a].ff * point[b].ss + point[b].ff *
    point[c].ss + point[c].ff * point[a].ss;
53.     res -= point[a].ff * point[c].ss + point[b].ff * point[a].ss
    + point[c].ff * point[b].ss;
54.     return ABS(res);
55. }
56.
57. int mxlimit[1010][1010];
58.
59. int binarySearch ( int i, int j, int m , int n, vlong k ) {
60.
61.     ///First search between  (j + 1) - m
62.     int low = j + 1, high = m;
63.
64.     ///Upper Bound
65.     while ( low <= high ) {
66.         int mid = ( low + high ) / 2;
67.
68.         vlong a = getArea( i, j, mid );
69.
70.         if ( a <= k ) {
71.             low = mid + 1;
72.         }
73.         else {
74.             high = mid - 1;
75.         }
```

```
    }

    int res = ( low - j - 1 );

    ///Now search between m+1 - (n-1)
    m = MAX(m+1,j+1);

    if ( m >= n ) return res; ///No more segment

    low = m; high = n - 1;

    ///Lower Bound
    while ( low <= high ) {
        int mid = ( low + high ) / 2;

        vlong a = getArea( i, j, mid );

        if ( a <= k ) {
            high = mid - 1;
        }
        else {
            low = mid + 1;
        }
    }

    res += ( n - 1 ) - low + 1;

    return res;
}

int main () {
    #ifdef forthright48
    freopen ( "input.txt", "r", stdin );
    //freopen ( "output.txt", "w", stdout );
    #endif // forthright48

    int kase;
    scanf ( "%d", &kase );

```

```
    while ( kase-- ) {
        int n;
        vlong lim;
        scanf ( "%d %lld", &n, &lim );
        lim *= 2;
        _g.n = n;

        FOR(i,0,n-1){
            scanf ( "%lld %lld", &point[i].ff, &point[i].ss );
        }

        if ( n <= 2 ) {
            printf ( "0\n" );
            continue;
        }


        ///Find the point which gives maximum area for given points (i,j)
        FOR(i,0,n-1){
            int s = i + 1; ///Initiate two pointer
            vlong prev = 0;
            FOR(f,i+1,n-1){
                prev = getArea( i, f, s );
                ///Make s go forward as much as possible
                while ( 1 ) {
                    if ( s + 1 >= n ) break; ///Not possible to go forward
                    vlong cur = getArea( i, f, s + 1 );
                    if ( cur >= prev ) {
                        prev = cur;
                        s++;
                    }
                    else break;
                }

                mxlimit[i][f] = s;
            }
        }
```

```
152.
153.        vlong res = 0;
154.
155.        ///Binary Search
156.        FOR(i,0,n-3){
157.            FOR(j,i+1,n-2){
158.                int m = mxlimit[i][j];
159.                res += binarySearch ( i, j, m, n, lim );
160.            }
161.        }
162.
163.        printf ( "%lld\n", res );
164.    }
165.
166.    return 0;
167. }
```

```
/**********Template Starts Here**********/
#include <bits/stdc++.h>

#define pb push_back
#define nl puts ("")
#define sp printf ( " " )
#define phl printf ( "hello\n" )
#define ff first
#define ss second
#define POPCOUNT __builtin_popcountll
#define RIGHTMOST __builtin_ctzll
#define LEFTMOST(x) (63-__builtin_clzll((x)))
#define MP make_pair
#define FOR(i,x,y) for(vlong i = (x) ; i <= (y) ; ++i)
#define ROF(i,x,y) for(vlong i = (y) ; i >= (x) ; --i)
#define CLR(x,y) memset(x,y,sizeof(x))
#define UNIQUE(V) (V).erase(unique((V).begin(),(V).end()),(V).end())
#define MIN(a,b) ((a)<(b)?(a):(b))
#define MAX(a,b) ((a)>(b)?(a):(b))
#define NUMDIGIT(x,y) (((vlong)(log10((x))/log10((y))))+1)
#define SQ(x) ((x)*(x))
#define ABS(x) ((x)<0?-(x):(x))
#define FABS(x) ((x)+eps<0?-(x):(x))
#define ALL(x) (x).begin(),(x).end()
#define LCM(x,y) (((x)/gcd((x),(y)))*(y))
#define SZ(x) ((vlong)(x).size())
#define NORM(x) if(x>=mod)x-=mod;

using namespace std;

typedef long long vlong;
typedef unsigned long long uvlong;
typedef pair < vlong, vlong > pll;
typedef vector<pll> vll;
typedef vector<vlong> vl;

const vlong inf = 2147383647;
const double pi = 2 * acos ( 0.0 );
const double eps = 1e-9;

#ifdef forthright48
      #include <ctime>
      clock_t tStart = clock();
      #define debug(args...) {dbg,args; cerr<<endl;}
      #define timeStamp printf("Execution Time: %.2fs\n", (double)(clock() - tStart)/C
LOCKS_PER_SEC)
      #define bug printf("%d\n",__LINE__);

#else
    #define debug(args...)  // Just strip off all debug tokens
    #define timeStamp
#endif

struct debugger{
    template<typename T> debugger& operator , (const T& v){
        cerr<<v<<" ";
```

```
56.           return *this;
57.       }
58.   }dbg;
59.
60.   //int knightDir[8][2] = { {-2,1},{-1,2},{1,2},{2,1},{2,-1},{-1,-2},{1,-2},{-2,-1} };
61.   //int dir4[4][2] = {{-1,0},{0,1},{1,0},{0,-1}};
62.
63.   inline vlong gcd ( vlong a, vlong b ) {
64.       a = ABS ( a ); b = ABS ( b );
65.       while ( b ) { a = a % b; swap ( a, b ); } return a;
66.   }
67.
68.   vlong ext_gcd ( vlong A, vlong B, vlong *X, vlong *Y ){
69.       vlong x2, y2, x1, y1, x, y, r2, r1, q, r;
70.       x2 = 1; y2 = 0;
71.       x1 = 0; y1 = 1;
72.       for (r2 = A, r1 = B; r1 != 0; r2 = r1, r1 = r, x2 = x1, y2 = y1, x1 = x, y1 = y )
      {
73.           q = r2 / r1;
74.           r = r2 % r1;
75.           x = x2 - (q * x1);
76.           y = y2 - (q * y1);
77.       }
78.       *X = x2; *Y = y2;
79.       return r2;
80.   }
81.
82.   inline vlong modInv ( vlong a, vlong m ) {
83.       vlong x, y;
84.       ext_gcd( a, m, &x, &y );
85.       if ( x < 0 ) x += m; //modInv is never negative
86.       return x;
87.   }
88.
89.   inline vlong power ( vlong a, vlong p ) {
90.       vlong res = 1, x = a;
91.       while ( p ) {
92.           if ( p & 1 ) res = ( res * x );
93.           x = ( x * x ); p >>= 1;
94.       }
95.       return res;
96.   }
97.
98.   inline vlong bigmod ( vlong a, vlong p, vlong m ) {
99.       vlong res = 1 % m, x = a % m;
100.      while ( p ) {
101.          if ( p & 1 ) res = ( res * x ) % m;
102.          x = ( x * x ) % m; p >>= 1;
103.      }
104.      return res;
105.  }
106.
107.  /**********Template Ends Here**********/
108.
109.  vlong process() {
110.      vlong a1, a2, b1, b2, c1, c2;
```

```
111.        scanf ( "%lld %lld %lld %lld %lld %lld", &a1, &a2, &b1, &b2, &c1, &c2 );
112.
113.        if ( a1 & 1 ) a1++;
114.        if ( a2 & 1 ) a2--;
115.        if ( a1 > a2 ) return 0;
116.
117.        vlong f = (a2-a1)/2 + 1;
118.
119.        if ( c1 & 1 ) c1++;
120.        if ( c2 & 1 ) c2--;
121.        if ( c1 > c2 ) return 0;
122.        vlong s = (c2-c1)/2 + 1;
123.
124.        return f * s * (b2-b1+1);
125.    }
126.
127.    int main () {
128.        #ifdef forthright48
129.        freopen ( "input.txt", "r", stdin );
130.        //freopen ( "output.txt", "w", stdout );
131.        #endif // forthright48
132.
133.        int kase, cnt = 0;
134.        scanf ( "%d", &kase );
135.
136.        while ( kase-- ) {
137.            vlong res = process();
138.            printf ( "Case %d: %lld\n", ++cnt, res );
139.        }
140.
141.        return 0;
142.    }
```

Runtime error #stdin #stdout 0s 3464KB

comments (0)

stdin copy

```
Standard input is empty
```

stdout copy

```
/**********Template Starts Here**********/
#include <bits/stdc++.h>

#define pb push_back
#define nl puts ("")
#define sp printf ( " " )
#define phl printf ( "hello\n" )
#define ff first
#define ss second
#define POPCOUNT __builtin_popcountll
#define RIGHTMOST __builtin_ctzll
#define LEFTMOST(x) (63-__builtin_clzll((x)))
#define MP make_pair
#define FOR(i,x,y) for(vlong i = (x) ; i <= (y) ; ++i)
#define ROF(i,x,y) for(vlong i = (y) ; i >= (x) ; --i)
#define CLR(x,y) memset(x,y,sizeof(x))
#define UNIQUE(V) (V).erase(unique((V).begin(),(V).end()),(V).end())
#define MIN(a,b) ((a)<(b)?(a):(b))
#define MAX(a,b) ((a)>(b)?(a):(b))
#define NUMDIGIT(x,y) (((vlong)(log10((x))/log10((y))))+1)
#define SQ(x) ((x)*(x))
#define ABS(x) ((x)<0?-(x):(x))
#define FABS(x) ((x)+eps<0?-(x):(x))
#define ALL(x) (x).begin(),(x).end()
#define LCM(x,y) (((x)/gcd((x),(y)))*(y))
#define SZ(x) ((vlong)(x).size())
#define NORM(x) if(x>=mod)x-=mod;

using namespace std;

typedef long long vlong;

/**********Template Ends Here**********/

int main () {

    int kase;
    scanf ( "%d", &kase );

    while ( kase-- ) {
        vlong n, m;
        scanf ( "%lld %lld", &n, &m );

        int critical = n - 1;
        m -= n - 1;

        int k = 1;
        while ( m > 0 ) {

            ///Remove edges
            m -= k;
            if ( k == 1 ) { ///Special case
                critical -= 2;
            }
            else {
                critical -= 1;
```

```
57.             }
58.
59.             k++;
60.         }
61.
62.         printf ( "%d\n", critical );
63.     }
64.
65.     return 0;
66. }
67.
```

Time limit exceeded #stdin #stdout 5s 3456KB

comments (0)

stdin copy

```
Standard input is empty
```

stdout

```
Standard output is empty
```

```
#include <bits/stdc++.h>

#define pb push_back
#define nl puts ("")
#define sp printf ( " " )
#define phl printf ( "hello\n" )
#define ff first
#define ss second
#define POPCOUNT __builtin_popcountll
#define RIGHTMOST __builtin_ctzll
#define LEFTMOST(x) (63-__builtin_clzll((x)))
#define MP make_pair
#define FOR(i,x,y) for(vlong i = (x) ; i <= (y) ; ++i)
#define ROF(i,x,y) for(vlong i = (y) ; i >= (x) ; --i)
#define CLR(x,y) memset(x,y,sizeof(x))
#define UNIQUE(V) (V).erase(unique((V).begin(),(V).end()),(V).end())
#define MIN(a,b) ((a)<(b)?(a):(b))
#define MAX(a,b) ((a)>(b)?(a):(b))
#define NUMDIGIT(x,y) (((vlong)(log10((x))/log10((y))))+1)
#define SQ(x) ((x)*(x))
#define ABS(x) ((x)<0?-(x):(x))
#define FABS(x) ((x)+eps<0?-(x):(x))
#define ALL(x) (x).begin(),(x).end()
#define LCM(x,y) (((x)/gcd((x),(y)))*(y))
#define SZ(x) ((vlong)(x).size())
#define NORM(x) if(x>=mod)x-=mod;

using namespace std;

typedef long long vlong;

#define SIZE 100001

int bit[SIZE+10];

void bitClear( int n ) {
    FOR(i,1,n+2) {
        bit[i] = 0;
    }
}
void bitInsert ( int from, int what ) {
    from++; ///Handle 0

    for ( int i = from; i <= SIZE; i += i & (-i) ) {
        bit[i] += what;
    }
}

int bitQuery ( int at ) {
    at++; ///Handle 0

    int res = 0;
    for ( int i = at; i > 0; i -= i & -i ) {
        res += bit[i];
    }
    return res;
```

```
}


int main () {

    int kase, cnt = 0;
    scanf ( "%d", &kase );

    while ( kase-- ) {
        int n, k;
        scanf ( "%d %d", &n, &k );

        bitClear(k); ///Clear BIT

        vlong total = 0;
        vlong res = 0;
        ROF(i,1,n) {
            total++; ///Total number of "c" available

            int x = ( i * i * i ) % k;
            bitInsert( x, 1 ); ///Insert another "c"

            res += ( i / k ) * total; ///This many "a" segment can handle any "c"


            int r = i % k;
            int b = ( i * i ) % k;

            if ( r > 0 ) { ///These parts of "a" were not used

                int from = 1 + b; ///c >= 1 + b
                int to = r + b; ///c <= r + b

                if ( from >= k ) { ///In case both from and to exceed k
                    from -= k;
                    to -= k;
                }

                int seg = 0;
                if ( to < k ) { ///Continuous
                    seg = bitQuery(to);
                    seg -= bitQuery(from-1);
                }
                else { ///Wraps around
                    seg = bitQuery(k-1); ///from - end
                    seg -= bitQuery(from-1);

                    seg += bitQuery(to%k); ///0 - to
                }
                res += seg;
            }
        }

        printf ( "Case %d: %lld\n", ++cnt, res );
    }
```

```
113.
114.        return 0;
115.    }
116.
```

Runtime error #stdin #stdout 0s 3800KB

comments (0)

stdin copy

```
Standard input is empty
```

stdout

```
Standard output is empty
```

```
1.  /**********Template Starts Here**********/
2.  #include <bits/stdc++.h>
3.
4.  #define pb push_back
5.  #define nl puts ("")
6.  #define sp printf ( " " )
7.  #define phl printf ( "hello\n" )
8.  #define ff first
9.  #define ss second
10. #define POPCOUNT __builtin_popcountll
11. #define RIGHTMOST __builtin_ctzll
12. #define LEFTMOST(x) (63-__builtin_clzll((x)))
13. #define MP make_pair
14. #define FOR(i,x,y) for(int i = (x) ; i <= (y) ; ++i)
15. #define ROF(i,x,y) for(int i = (y) ; i >= (x) ; --i)
16. #define CLR(x,y) memset(x,y,sizeof(x))
17. #define UNIQUE(V) (V).erase(unique((V).begin(),(V).end()),(V).end())
18. #define MIN(a,b) ((a)<(b)?(a):(b))
19. #define MAX(a,b) ((a)>(b)?(a):(b))
20. #define NUMDIGIT(x,y) (((int)(log10((x))/log10((y))))+1)
21. #define SQ(x) ((x)*(x))
22. #define ABS(x) ((x)<0?-(x):(x))
23. #define FABS(x) ((x)+eps<0?-(x):(x))
24. #define ALL(x) (x).begin(),(x).end()
25. #define LCM(x,y) (((x)/gcd((x),(y)))*(y))
26. #define SZ(x) ((int)(x).size())
27.
28. using namespace std;
29.
30. typedef long long vlong;
31. typedef unsigned long long uvlong;
32. typedef pair < int, int > pii;
33. typedef pair < vlong, vlong > pll;
34. typedef vector<pii> vii;
35. typedef vector<int> vi;
36.
37. const vlong inf = 2147383647;
38. /**********Template Ends Here**********/
39.
40. vii adj[1000100];
41. int ign;
42. int vis[1000010], col[1000010], vv;
43.
44. int n;
45. bool bicolor;
46.
47. ///Bicolor the graph
48. int dfs ( int s, int p ) {
49.     vis[s] = vv;
50.
51.     FOR(i,0,SZ(adj[s])-1) {
52.         pii temp = adj[s][i];
53.         int t = temp.ff;
54.         int w = temp.ss;
55.
56.         if ( w >= ign ) continue;
```

```
57.
58.          if ( t == p ) continue;
59.
60.          if ( vis[t] != vv ) {
61.              col[t] = 1 - col[s];
62.              dfs ( t, s );
63.          }
64.          else {
65.              if ( col[t] == col[s] ) {
66.                  bicolor = false;
67.              }
68.          }
69.      }
70.  }
71.
72.  ///Check if graph is bipartite
73.  bool bipartite ( int len ) {
74.      ign = len;
75.      vv++;
76.
77.      bicolor = true;
78.      FOR(i,1,n) {
79.          if ( vis[i] != vv ) {
80.              col[i] = 0; ///White
81.              dfs ( i, -1 );
82.              if ( bicolor == false ) return false;
83.          }
84.      }
85.
86.      return true;
87.  }
88.
89.  int main () {
90.      int kase;
91.      scanf ( "%d", &kase );
92.
93.      while ( kase-- ) {
94.          int m;
95.          scanf ( "%d %d", &n, &m );
96.
97.          FOR(i,1,n) adj[i].clear();
98.
99.          int mx = 0, mn = inf;
100.         FOR(i,0,m-1) {
101.             int a, b, c;
102.             scanf ( "%d %d %d", &a, &b, &c );
103.
104.             adj[a].pb ( MP(b,c) );
105.             adj[b].pb ( MP(a,c) );
106.
107.             mx = MAX(mx,c);
108.             mn = MIN(mn,c);
109.         }
110.
111.         bool res = bipartite( mx + 1 ); ///Keep all the edges
112.         if ( res ) {
```

```
113.                printf ( "%d\n", 0 );
114.                continue;
115.            }
116.
117.            int low = 1, high = mx;
118.
119.            while ( low <= high ) {
120.                int mid = ( low + high ) / 2;
121.
122.                res = bipartite ( mid );
123.
124.                if ( res ) {
125.                    low = mid + 1;
126.                }
127.                else high = mid - 1;
128.            }
129.
130.            if ( low - 1 == mn ) low = 0; ///Need to remove all edges. So answer is -1
131.            printf ( "%d\n", low - 1 );
132.
133.        }
134.
135.        return 0;
136.    }
137.
```

Success #stdin #stdout 0s 22944KB

comments (0)

stdin copy

```
Standard input is empty
```

stdout

```
Standard output is empty
```

```
/**********Template Starts Here**********/
#include <bits/stdc++.h>

#define pb push_back
#define nl puts ("")
#define sp printf ( " " )
#define phl printf ( "hello\n" )
#define ff first
#define ss second
#define POPCOUNT __builtin_popcountll
#define RIGHTMOST __builtin_ctzll
#define LEFTMOST(x) (63-__builtin_clzll((x)))
#define MP make_pair
#define FOR(i,x,y) for(int i = (x) ; i <= (y) ; ++i)
#define ROF(i,x,y) for(int i = (y) ; i >= (x) ; --i)
#define CLR(x,y) memset(x,y,sizeof(x))
#define UNIQUE(V) (V).erase(unique((V).begin(),(V).end()),(V).end())
#define MIN(a,b) ((a)<(b)?(a):(b))
#define MAX(a,b) ((a)>(b)?(a):(b))
#define NUMDIGIT(x,y) (((int)(log10((x))/log10((y))))+1)
#define SQ(x) ((x)*(x))
#define ABS(x) ((x)<0?-(x):(x))
#define FABS(x) ((x)+eps<0?-(x):(x))
#define ALL(x) (x).begin(),(x).end()
#define LCM(x,y) (((x)/gcd((x),(y)))*(y))
#define SZ(x) ((int)(x).size())

using namespace std;

typedef long long vlong;
typedef unsigned long long uvlong;
typedef pair < int, int > pii;
typedef pair < vlong, vlong > pll;
typedef vector<pii> vii;
typedef vector<int> vi;

/**********Template Ends Here**********/
vlong b, m;

///find (a*b)%m without overflowing
vlong mulMod ( vlong a, vlong b ) {
    vlong res = 0;
    vlong x = a;
    while ( b ) {
        if ( b & 1 ) {
            res = res + x;
            if ( res >= m ) res -= m;
        }
        x = ( x + x );
        if ( x >= m ) x -= m;
        b >>= 1;
    }

    return res;
}

```

```
vlong bigmodExtended ( vlong a, vlong p ) {
    vlong res = 1 % m, x = a % m;
    while ( p ) {
        if ( p & 1 ) res = mulMod( res, x );
        x = mulMod( x , x ); p >>= 1;
    }
    return res;
}

vlong divMod ( vlong n ) {
    if ( n == 1 ) {
        return 1 % m;
    }

    if ( n & 1 ) {
        vlong res = bigmodExtended( b, n-1 ) + divMod( n - 1 );
        if ( res >= m ) res -= m;

        return res;
    }
    else {
        vlong res = divMod(n/2);
        vlong temp = 1 + bigmodExtended(b,n/2);
        if ( temp >= m ) temp -= m;

        if ( temp == 0 || res == 0 ) return 0;
        return mulMod ( res, temp );
    }
}

int main () {
    int kase, cnt = 0;
    scanf ( "%d", &kase );

    while ( kase-- ) {
        vlong n, d;
        scanf ( "%lld %lld %lld %lld", &n, &b, &d, &m );

        vlong res = divMod ( n );

        res = ( res * d );
        res %= m;

        printf ( "Case %d: %lld\n", ++cnt, res );
    }

    return 0;
}
```

14, 0.15s, 0.23s

Runtime error #stdin #stdout 0s 3456KB

comments (0)

forthright48

Learning Never Ends

Thursday, August 27, 2015

SPOJ LCMSUM - LCM Sum

Problem

Problem Link - SPOJ LCMSUM

Given n, calculate the sum $LCM(1,n) + LCM(2,n) + \ldots + LCM(n,n)$, where $LCM($
Least Common Multiple of the integers i and n.

Solution

I recently solved this problem and found the solution really interesting. You can find the formu
problem on OEIS. I will show you the derivation here.

In order to solve this problem, you need to know about Euler Phi Function, finding Divisor usin
some properties of LCM and GCD.

Define SUM

Let us define a variable SUM which we need to find.

$SUM = LCM(1,n) + LCM(2,n) + \ldots + LCM(n,n)$ (take $LCM(n,n)$ to the other s
$SUM - LCM(n,n) = LCM(1,n) + LCM(2,n) + \ldots + LCM(n-1,n)$

We know that $LCM(n,n) = n$.

$SUM - n = LCM(1,n) + LCM(2,n) + \ldots + LCM(n-1,n)$ $(eq1)$

Reverse and Add

$SUM - n = LCM(1,n) + LCM(2,n) + \ldots + LCM(n-1,n)$ (Reverse $eq1$ to get eq
$SUM - n = LCM(n-1,n) + LCM(n-2,n) + \ldots + LCM(1,n)$ ($eq2$)

No what will we get if we add $eq1$ with $eq2$? Well, we need to do more work to find that.

Sum of $LCM(a,n) + LCM(n-a,n)$

$x = LCM(a,n) + LCM(n-a,n)$

$x = \frac{an}{gcd(a,n)} + \frac{(n-a)n}{gcd(n-a,n)}$ $(eq3)$

Arghh. Now we need to prove that $gcd(a, n)$ is equal to $gcd(n - a, n)$.

If c divides a and b, then, c will divide $a + b$ and $a - b$. This is common property of division. S
$g = gcd(a, n)$ divides a and n, then it will also divide $n - a$. Hence, $gcd(a, n) = gcd(n -$

So $eq3$ becomes:

$x = \frac{an}{gcd(a,n)} + \frac{(n-a)n}{gcd(a,n)}$
$x = \frac{an+n^2-an}{gcd(a,n)}$.
$x = \frac{n^2}{gcd(a,n)}$.

Now, we can continue adding $eq1$ with $eq2$.

Reverse and Add Continued

$SUM - n = LCM(1, n) + LCM(2, n) + \ldots + LCM(n - 1, n)$ $(eq1)$
$SUM - n = LCM(n - 1, n) + LCM(n - 2, n) + \ldots + LCM(1, n)$ ($eq2$ Add them)
$2(SUM - n) = \frac{n^2}{gcd(1,n)} + \frac{n^2}{gcd(2,n)} + \ldots \frac{n^2}{gcd(n-1,n)}$
$2(SUM - n) = \sum_{i=1}^{n-1} \frac{n^2}{gcd(i,n)}$
$2(SUM - n) = n \sum_{i=1}^{n-1} \frac{n}{gcd(i,n)}$ (take n common)

Group Similar GCD

What are the possible values of $g = gcd(i, n)$? Since g must divide n, g needs to be a divisc
list the possible values of $gcd(i, n)$ by finding the divisors of n.

Let $Z = n \sum_{i=1}^{n-1} \frac{n}{gcd(i,n)}$. Now, every time $gcd(i, n) = d$, where d is a divisor of n, $n \times \frac{n}{d}$ i
Therefore, we just need to find, for each divisor d, how many times $n \times \frac{n}{d}$ is added to Z.

How many values i can we put such that $gcd(i, n) = d$? There are $\phi(\frac{n}{d})$ possible values of
$gcd(i, n) = d$. Therefore:

$2(SUM - n) = n \sum_{i=1}^{n-1} \frac{n}{gcd(i,n)}$
$2(SUM - n) = n \sum_{d \mid n, d \neq n} \phi(\frac{n}{d}) \times \frac{n}{d}$ (but, $\frac{n}{d}$ is also a divisor of n)
$2(SUM - n) = n \sum_{d \mid n, d \neq 1} \phi(d) \times d$ (when $d = 1$, we get $\phi(1) \times 1$)
$2(SUM - n) = n(\sum_{d \mid n} (\phi(d) \times d) - 1)$
$2(SUM - n) = n \sum_{d \mid n} (\phi(d) \times d) - n$
$2SUM - 2n + n = n \sum_{d \mid n} \phi(d) \times d$
$2SUM - n = n \sum_{d \mid n} \phi(d) \times d$

$$2SUM = n(\sum_{d \mid n} \phi(d) \times d) + n$$

$$2SUM = n(\sum_{d \mid n} (\phi(d) \times d) + 1)$$

$$\therefore SUM = \frac{n}{2}(\sum_{d \mid n} (\phi(d) \times d) + 1)$$

Using this formula we can solve the problem.

Code

```
#include <bits/stdc++.h>

#define FOR(i,x,y) for(vlong i = (x) ; i <= (y) ; ++i)

using namespace std;
typedef long long vlong;

vlong res[1000010];
vlong phi[1000010];

void precal( int n ) {
    ///Calculate phi from 1 to n using sieve
    FOR(i,1,n) phi[i] = i;
    FOR(i,2,n) {
        if ( phi[i] == i ) {
            for ( int j = i; j <= n; j += i ) {
                phi[j] /= i;
                phi[j] *= i - 1;
            }
        }
    }

    ///Calculate partial result using sieve
    ///For each divisor d of n, add phi(d)*d to result array
    FOR(i,1,n){
        for ( int j = i; j <= n; j += i ) {
            res[j] += ( i * phi[i] );
        }
    }
}

int main () {
    precal( 1000000 );

    int kase;
    scanf ( "%d", &kase );

    while ( kase-- ) {
        vlong n;
        scanf ( "%lld", &n );

        ///We already have partial result in res[n]
        vlong ans = res[n] + 1;
        ans *= n;
        ans /= 2;

        printf ( "%lld\n", ans );
    }

```

```
50          return 0;
51      }
```

We nee to precalculate partial result using a sieve for all values of N. Precalculation has a co $O(N\ lnN)$. After pre-calculation, for each N we can answer in $O(1)$.

Reference

1. OEIS - A051193

Posted by Mohammad Samiul Islam G+1 Recommend this on Google

Labels: Analysis, Divisors, GCD, LCM, Number Theory, Sieve, SPOJ

8 comments:

Shubham Aggarwal October 26, 2015 at 8:03 PM

good explanation..!!!
thanks man.

Reply

Unknown November 7, 2015 at 1:36 PM

Can you please explain how did you calculated phi() ?

Reply

Replies

Mohammad Samiul Islam November 12, 2015 at 3:22 PM

I used Sieve to generate all phi() in O(N). First initiate phi[x] = x for all values from 1 t prime p, and multiply (p-1)/p to all multiples of p between 1 and N. This way, you w value at the end.

Paras Avkirkar November 14, 2015 at 1:01 AM

Thank you

Shubham Aggarwal January 13, 2016 at 2:00 PM

@Mohammad samiul islam can you please elaborate why do you multiple it and first then multiplication .

Mohammad Samiul Islam January 16, 2016 at 4:31 PM

@Shubham: Are you talking about line 17 and 18? If I multiply first and then divide possibility that phi[j] will overflow. Dividing first and then multiplying reduces risk of over

forthright48

Learning Never Ends

Thursday, August 27, 2015

Bitwise Operators

We know about arithmetic operators. The operators $+, -, /$ and $\times$ adds, subtracts, divides a respectively. We also have another operator $\%$ which finds the modulus.

Today, we going to look at 6 more operators called "Bitwise Operators". Why are they called " Operators"? That's because they work using the binary numerals (bits, which are the individu binary number) of their operands. Why do we have such operators? That's because in compu information is stored as strings of bits, that is, binary numbers. Having operators that work dire pretty useful.

We need to have a good idea how Binary Number System works in order to understand how work. Read more on number system in [1]Introduction to Number Systems. Use the $decimal$ function from [1] to convert the decimal numbers to binary and see how they are affected.

Bitwise AND ($\&$) Operator

The $\&$ operator is a binary operator. It takes two operands and returns single integer as the re it affects the bits.

$0 \ \& \ 0 = 0$
$0 \ \& \ 1 = 0$
$1 \ \& \ 0 = 0$
$1 \ \& \ 1 = 1$

It takes two bits and returns another bit. The $\&$ operator will take two bits a, b and return 1 onl are 1. Otherwise, it will return 0.

But that's only for bits. What happens when we perform $\&$ operation on two integer number?

For example, what is the result of $A \ \& \ B$ when $A = 12$ and $B = 10$?

Since $\&$ operator works on bits of binary numbers we have to convert A and B to binary num

$A = (12)_{10} = (1100)_2$

$B = (10)_{10} = (1010)_2$

We know how to perform $\&$ on individual bits, but how do we perform $\&$ operation on strings take each position of the string and perform and operations using bits of that position.

$$\begin{array}{r} 1100 \\ 1010\ \& \\ \hline 1000 \end{array}$$

Therefore, $12\ \&\ 10 = 8$.

In C++, it's equivalent code is:

```
1 printf ("%d\n", 12 & 10 );
```

Bitwise OR ($|$) Operator

The $|$ operator is also a binary operator. It takes two operands and returns single integer as th how it affects the bits:

$0 \mid 0 = 0$
$0 \mid 1 = 1$
$1 \mid 0 = 1$
$1 \mid 1 = 1$

The $|$ operator takes two bits a, b and return 1 if a **OR** b is 1. Therefore, it return 0 only when b 0.

What is the value of $A \mid B$ if $A = 12$ and $B = 10$? Same as before, convert them into binary apply $|$ operator on both bits of each position.

$$\begin{array}{r} 1100 \\ 1010\ \mid \\ \hline 1110 \end{array}$$

Therefore $12 \mid 10 = 14$.

```
1 printf ( "%d\n", 12 | 10 );
```

Bitwise XOR ($\wedge$) Operator

Another binary operator that takes two integers as operands and returns integer. Here is how

$0 \wedge 0 = 0$
$0 \wedge 1 = 1$
$1 \wedge 0 = 1$
$1 \wedge 1 = 0$

XOR stands for Exclusive-OR. This operator returns 1 only when both operand bits are not sa returns 0.

What is the value of $A \wedge B$ if $A = 12$ and $B = 10$?

$$\begin{array}{r} 1100 \\ 1010 \wedge \\ \hline 0110 \end{array}$$

Therefore, $12 \wedge 10 = 6$.

In mathematics, XOR is represented with $\oplus$, but I used $\wedge$ cause in C++ XOR is performed with

```
printf ( "%d\n", 12 ^ 10 );
```

Bitwise Negation ($\sim$) Operator

This is a unary operator. It works on a single integer and flips all its bits. Here is how it affects i

$\sim 0 = 1$
$\sim 1 = 0$

What is the value of $\sim A$ if $A = 12$?

$\sim (1100)_2 = (0011)_2 = (3)_{10}$

But this will not work in code cause 12 in C++ is not 1100, it is $0000...1100$. Each integer is 3
when each of the bits of the integer is flipped it becomes $11111...0011$. If you don't take unsi
value will even come out negative.

```
1 printf ( "%d\n", ~12 );
```

Bitwise Left Shift ($<<$) Operator

The left shift operator is a binary operator. It takes two integers a and b and shifts the bits of a
times and adds b zeroes to the end of a in its binary system.

For example, $(13)_{10} << 3 = (1101)_2 << 3 = (110100)_2$.

Shifting the bits of a number A left once is same as multiplying it by 2. Shifting it left three time
multiplying the number by 2^3.

Therefore, the value of $A << B = A \times 2^B$.

```
1 printf ( "%d\n", 1 << 3 );
```

Bitwise Right Shift ($>>$) Operator

The $>>$ Operator does opposite of $<<$ operator. It takes two integer a and b and shifts the b
RIGHT b times. The rightmost b bits are lost and b zeroes are added to the left end.

For example, $(13)_{10} >> 3 = (1101)_2 >> 3 = (1)_2$.

Shifting the bits of a number A right once is same as dividing it by 2. Shifting it right three time

dividing the number by 2^3.

Therefore, the value of $A >> B = \lfloor \frac{A}{2^B} \rfloor$.

```
1 printf ( "%d\n", 31 >> 3 );
```

Tips and Tricks

1. When using $<<$ operator, careful about overflow. If $A << B$ does not fit into int, mak cast A into $long\ long$. Typecasting B into $long\ long$ does not work.
2. $A \ \& \ B \leq MIN(A, B)$
3. $A \mid B \geq MAX(A, B)$

That's all about bitwise operations. These operators will come useful during "Bits Manipulation it on next post.

Resource

1. forthright48 - Introduction to Number Systems
2. Wiki - Bitwise Operation

Posted by Mohammad Samiul Islam

G+1 Recommend this on Google

Labels: Binary, Language

No comments:

Post a Comment

Leave comments for Queries, Bugs and Hugs.

forthright48

Learning Never Ends

Monday, August 31, 2015

Bit Manipulation

On this post, we will look into how to use "Bitwise Operators" to manipulate bits of a number. B
a useful tool that can sometimes perform complicated tasks in a simple manner.

We will work on integer values and assume each number has 32 bits in them. If a number ha
digit, then assume that it is in binary form. All position of bits is 0 indexed.

Since binary values can only be 0 or 1, we sometimes refer them like light bulbs. A bit being "(
value 0 and being "On" means it has value 1. Bit 1 is also referred as "Set" and bit 0 as "Rese

We need to have a strong grasp of how AND, OR, Negation, XOR and Shift Operators work tc
bit manipulation works. If you have forgotten about them, then make sure you revise.

Checking Bit at Position X

Given a number N, find the value of its bit at position X. For example, if $N = 12 = (1100)_2$
value of bit is 1.

So how do we find it? We can find this value in three steps:

1. Let, $mask = 1 << X$
2. Let, $res = N \ \& \ mask$
3. If res is 0, then bit is 0 at that position, else bit is 1.

Let us see it in action with the example above, $N = 12$ and $X = 2$.

1100
$\underline{0100}$ & ($1 << 2$ is just 1 shifted to left twice)
0100

Now, what happened here? In step 1, when we left shifted 1 X times, we created a number n
bit 1 only at position X. This is a useful application of left shift operator. Using this, we can mo
we want.

Next, at step 2, we find the result of performing AND operator between N and $mask$. Since m positions except X, the result will have 0 in all places other than X. That's because performin always give 0. Now, what about position X. Will the result at position X have 0 too? That dep N at position X. Since, the X bit in the mask is 1, the only way result will have 0 at that positi that position.

So, when all position of res is 0, that is res has value 0, we can say that N has 0 bit in positi doesn't.

Set Bit at Position X

Given a value N, turn the bit at position X of N to 1. For example, $N = 12 = 1100$ and X become $13 = 1101$.

This can be done in two steps:

1. Let, $mask = 1 << X$
2. $N = N \mid mask$

$$\begin{array}{l} 1100 \\ \underline{0001} \mid \\ 1101 \end{array}$$

In step 1, we shift a 1 bit to position X. Then we perform OR between N and $mask$. Since m places except X, in result all those places remain unchanged. But $mask$ has 1 in position X, position X will be 1.

Reset Bit at Position X

Given a value N, turn the bit at position X of N to 0. For example, $N = 12 = 1100$ and X become $4 = 0100$.

This cane be done in three steps:

1. Let, $mask = 1 << X$
2. $mask = \sim mask$
3. $N = N \ \& \ mask$

$$\begin{array}{l} 1100 \\ \underline{0111} \ \& \text{ (We first got } 1000 \text{ from step } 1. \text{ Then used negation from step 2.)} \\ 0100 \end{array}$$

In step 1 we move 1 bit to position X. This gives us a number with 1 in position X and 0 in all Next we negate the $mask$. This flips all bits of $mask$. So now we have 0 in position X and 1 i Now when we perform AND between $mask$ and N, it forces the bit at the X position to 0 and intact.

Toggle Bit at Position X

Given a value N, toggle the bit at position X, i.e, if bit at X is 0 then make it 1 and if it is alrea

This can be done in two steps:

1. Let, $mask = 1 << X$
2. $N = N \wedge mask$

First we shift bit 1 to our desired position X in step 1. When XOR is performed between a bit a remains unchanged. But when XOR performed between a bit and 1, it toggles. So all position during step 2, since all position in mask except X is 0.

Coding Tips

When coding these in C++, make sure you use lots of brackets. Bitwise operators have lower $==$ and $!=$ operator which often causes trouble. See here.

What would be the output of this code:

```
1 if ( 0 & 6 != 7 ) printf ( "True\n" );
2 else printf ( "False\n" );
```

You might expect things to follow this order: $0 \,\&\, 6 = 0$ and $0 != 7$, so "True" will be output. E precedence of operators, the output comes out "False". First $6 != 7$ is checked. This is true, Next $0 \,\&\, 1$ is performed which is 0.

The best way to avoid these kind of mishaps is to use brackets whenever you are using bitwis correct way to write the code above is:

```
1 if ( (0 & 6) != 7 ) printf ( "True\n" );
2 else printf ( "False\n" );
```

This prints "True" which is our desired result.

Conclusion

These are the basics of bit manipulation and by no means the end of it. There are lots of othe to be seen, such as "Check Odd or Even", "Check Power of Two", "Right Most Bit". We will loo other day.

Reference

1. forthright48 - Bitwise Operators
2. CPPReference - C++ Operator Precedence

Posted by Mohammad Samiul Islam

G+1 Recommend this on Google

Labels: Binary, Bitwise, Optimization

No comments:

Post a Comment

Leave comments for Queries, Bugs and Hugs.

forthright48

Learning Never Ends

Friday, September 4, 2015

Euler Totient or Phi Function

I have been meaning to write a post on Euler Phi for a while now, but I have been struggling w
heard it required Chinese Remainder Theorem, so I have been pushing this until I covered CR
found that CRT is not required and it can be proved much more easily. In fact, the proof is so s
elegant that after reading it I went ahead and played MineCraft for 5 hours to celebrate.

Problem

Given an integer N, how many numbers less than or equal N are there such that they are co
number X is coprime to N if $gcd(X, N) = 1$.

For example, if $N = 10$, then there are 4 numbers, namely $\{1, 3, 7, 9\}$, which are coprime to

This problem can be solved using Euler Phi Function, $\phi()$. Here is the definition from Wiki:

> In number theory, Euler's totient function (or Euler's phi function), denoted as φ(n) or ϕ(n
> arithmetic function that counts the positive integers less than or equal to n that are relati
> n. - Wiki

That's exactly what we need to find in order to solve the problem above. So, how does Euler F

Euler Phi Function

Before we go into its proof, let us first see the end result. Here is the formula using which we c
of the $\phi()$ function. If we are finding Euler Phi of $N = p_1^{a_1} p_2^{a_2} \dots p_k^{a_k}$, then:

$$\phi(n) = n \times \frac{p_1 - 1}{p_1} \times \frac{p_2 - 1}{p_2} \dots \times \frac{p_k - 1}{p_k}$$

If you want you can skip the proof and just use the formula above to solve problems. That's w
doing all these years. But I highly recommend that you read and try to understand the proof. I
am sure someday the proof will help you out in an unexpected way.

Proof of Euler Phi Function

Even though the proof is simple, it has many steps. We will go step by step, and slowly you wi proof is unfolding in front of your eyes.

Base Case - $\phi(1)$

First, the base case. Phi function counts the number of **positive** numbers less than N that ar The keyword here is positive. Since the smallest positive number is 1, we will start with this.

$\phi(1) = 1$, since 1 itself is the only number which is coprime to it.

When n is a Prime - $\phi(p)$

Next, we will consider the case when $n = p$. Here p is any prime number. When n is prime, it numbers less than n. Therefore, $\phi(n) = \phi(p) = p - 1$.

When n is Power of Prime - $\phi(p^a)$

Next, we will consider n where n is a power of a single prime. In this case, how many number coprime to it? Instead of counting that, we will count the inverse. How many numbers are ther coprime.

Since, $n = p^a$, we can be sure that $gcd(p, n) \neq 1$. Since both n and p are divisible by p. Th following numbers which are divisible by p are not coprime to n, $\{p, 2p, 3p \ldots . p^2, (p+1)p$ $\ldots (p^2)p, (p^2+1)p \ldots (p^{a-1})p\}$. There are exactly $\frac{p^a}{p} = p^{a-1}$ numbers which are divisible are $n - p^{a-1}$ numbers which are coprime to n.

Hence, $\phi(n) = \phi(p^a) = n - \frac{n}{p} = p^a - \frac{p^a}{p} = p^a(1 - \frac{1}{p}) = p^a \times (\frac{p-1}{p})$

It's starting to look like the equation above, right?

Assuming $\phi()$ is Multiplicative - $\phi(m \times n)$

This step is the most important step in the proof. This step claims that Euler Phi function is a m function. What does this mean? It means, if m and n are coprime, then $\phi(m \times n) = \phi(m)$ Functions that satisfy this condition are called Multiplicative Functions.

So how do we prove that Euler Phi is multiplicative and how does Euler Phi being multiplicative

We will prove multiplicity of Euler Phi Function in the next section. In this section, we will assun multiplicative and see how it helps us calculating Euler Phi.

Let the prime factorization of n be $p_1^{a_1} p_2^{a_2} \ldots p_k^{a_k}$. Now, obviously p_i nad p_j are coprime to ea funtion is multiplicative, we can simply rewrite the function as:

$\phi(n) = \phi(p_1^{a_1} p_2^{a_2} \ldots p_k^{a_k})$
$\phi(n) = \phi(p_1^{a_1}) \times \phi(p_2^{a_2}) \ldots \times \phi(p_k^{a_k})$.

We can already calculate $\phi(p^a) = p^a \times \frac{p-1}{p}$. So our equationg becomes:

$\phi(n) = \phi(p_1^{a_1}) \times \phi(p_2^{a_2}) \ldots \times \phi(p_k^{a_k})$
$\phi(n) = p_1^{a_1} \times \frac{p_1-1}{p_1} \times p_2^{a_2} \times \frac{p_2-1}{p_2} \ldots \times p_k^{a_k} \times \frac{p_k-1}{p_k}$

$$\phi(n) = (p_1^{a_1} \times p_2^{a_2} \ldots \times p_k^{a_k}) \times \frac{p_1-1}{p_1} \times \frac{p_2-1}{p_2} \ldots \times \frac{p_k-1}{p_k}$$

$$\therefore \phi(n) = n \times \frac{p_1-1}{p_1} \times \frac{p_2-1}{p_2} \ldots \times \frac{p_k-1}{p_k}$$

This is what we have been trying to prove. This equation was derived by assuming that Euler multiplicative. So all we need to do now is prove Euler Phi Function is multiplicative and we are

Proof for Multiplicity of Euler Phi Function

We are trying to prove the following theorem:

> **Theorem** 1: If m and n are coprime, then $\phi(m \times n) = \phi(m) \times \phi(n)$

But in order to prove Theorem 1, we will need to prove few other theorems first.

Theorem Related to Arithmetic Progression

> **Theorem** 2: In an arithmetic progression with difference of m, if we take n terms and fin modulo by n, and if n and m are coprimes, then we will get the numbers from 0 to $n -$ order.

Umm, looks like Theorem 2 is packed with too much information. Let me break it down.

Suppose you have an arithmetic progression (AP). Now, every arithmetic progression has two element and a common difference. That is, arithmetic progressions are of the form $a + kb$ wh starting element, b is the common difference and k is any number.

So take any arithmetic progression that has a common difference of m. Then take n consecu progression. So which terms will they be?

$\{a + 0m, a + 1m, a + 2m, a + 3m \ldots a + (n-1)m\}$ There are exactly n terms in this li

Next, find their modulus by n. That is, find the remainder of each term after dividing by n.

Now, Theorem 2 claims that, if m and n are coprime, then the list will contain a permutation o

For example, let us try with $a = 1$, $m = 7$ and $n = 3$. So the 3 terms in the list will be $(1, 8, 1$ modulus of each element, we get $(1, 2, 0)$.

I hope that now it's clear what the Theorem claims. Now we will look into the proof.

Proof of Theorem 2
What we need to prove is that the list after modulo operation has a permutation of numbers fr That means, all the numbers from 0 to $n-1$ occurs in the list exactly once. There are three s Also, remember that m and n are coprime.

1. **There are exactly n elements in the list.**
   Well, since we took n terms from the AP, this is obvious.
2. **Each element of the list has value between 0 to $n-1$**
   We performed modulo operations on each element by n. So this is also obvious.

3. **No remainder has the same value as another.**
   Since there are n values, and each value is between 0 to $n-1$, if we can prove that ea
   unique in the list, then our work is done.

   Suppose there are two numbers which have the same remainder. That means $a+pm$
   remainder as $a+qm$, where p and q are two integer numbers such that $0 \leq p < q \leq$

   Therefore, $(a+qm)-(a+pm) \equiv 0 \ (mod\, n)$
   $(a+qm-a-pm) \equiv 0 \ (mod\, n)$
   $m(q-p) \equiv 0 \ (mod\, n)$

   That means, n divides $m(q-p)$. But this is impossible. n and m are coprime and $q-$
   n. So it is not possible for two numbers to have the same remainder.

Theorem Related to Remainder

Theorem 3: If a number x is coprime to y, then $(x \ \% \ y)$ will also be coprime to y.

The proof for this theorem is simple. Suppose x and y are coprime. Now, we can rewrite x as
where k is the quotient and r is the remainder.

Theorem 3 claims that y and r are coprime. What happens if this claim is false? Suppose the
That means there is a number $d > 1$ which divides both y and r. But then, d also divides ky
$d > 1$ divides r, y and x, which is impossible cause y and x are coprime. There is no numbe
that can divide both of them. Hence, there is a contradiction. Hence, the theorem is proved.

Proof for Multiplicity of Euler Phi Function Continued

We are now ready to tackle proving Theorem 1.

Suppose you have two numbers m and n, and they are coprime. We want to show that
$\phi(m \times n) = \phi(m) \times \phi(n)$.

What done $\phi(m \times n)$ gives us? It gives us the count of numbers which are coprime to mn. I
coprime to mn, then it is also coprime to m and n separately. So basically, we need to count
positive numbers less than or equal to mn which are coprime to both m and n.

Now, let us build a table of with n rows and m columns. Therefore, the table will look like the f

1	2	3	...	m
1+m	2+m	3+m	...	2m
1+2m	2+2m	3+2m	...	3m
...	...	...	...	...

1 + (n-1)m	2 + (n-1)m	3 + (n-1)m	...	mn

Now, notice that each column is an arithmetic progression with n terms and has common diffe
m and n are coprime. This is exactly the same situation as Theorem 2.

Now, how many numbers in each column are coprime to n? Using theorem 2, we know that e
a permutation of numbers from 0 to $n-1$. Using theorem 3, we know what if the remainder c
coprime to n then the number itself will also be coprime. So, how many numbers between 0 to
coprime to n? We can consider 0 to be same as n (cause this is modular arithmetic), so it bo
many numbers between 1 to n is coprime to n? Euler Phi Function calculates this values.

So, there are exactly $\phi(n)$ numbers which are coprime to n in each column.

We need to find numbers that coprime to **both** n and m. So, we cannot take $\phi(n)$ elements f
column, cause those elements may not be coprime to m. How do we decide which columns v
taking?

Notice that, if we find the modulus of elements of the table by m, then each row has remainde
$m-1$ occuring exactly once. If we consider 0 to be m, then each row has values between 1
table becomes something like this:

1	2	3	...	m
1	2	3	...	m
1	2	3	...	m
...	...	...	...	...
1	2	3	...	m

So, how many columns are there which are coprime to m? There are $\phi(m)$ columns which a

Now we just need to combine the two results from above. There are exactly $\phi(m)$ columns w
to m and in each column there are $\phi(n)$ values which are coprime to n. Therefore, there are
elements which are coprime to both m and n.

$$\therefore \phi(m) \times \phi(n) = \phi(m \times n)$$

Code

Since we have to factorize n in order to calculate $\phi(n)$, we can modify our $factorize()$ funct
"Prime Factorization of Integer Number" to handle Euler Phi.

```
1  int eulerPhi ( int n ) {
2      int res = n;
```

```
3       int sqrtn = sqrt ( n );
4       for ( int i = 0; i < prime.size() && prime[i] <= sqrtn; i++ ) {
5           if ( n % prime[i] == 0 ) {
6               while ( n % prime[i] == 0 ) {
7                   n /= prime[i];
8               }
9               sqrtn = sqrt ( n );
10              res /= prime[i];
11              res *= prime[i] - 1;
12          }
13      }
14      if ( n != 1 ) {
15          res /= n;
16          res *= n - 1;
17      }
18      return res;
19  }
```

I highlighted the lines that are different from $factorize()$ function. Notice that in line 10 divide multiplying in line 11. This is an optimization that lowers the risk of overflowing.

Conclusion

That was a long post with lots of theorems and equations, but hopefully, they were easy to un though Theorem 2 and 3 were used as lemmas to prove Theorem 1, they both are important

Leave a comment if you face any difficulty in understanding the post.

Reference

1. Wiki - Euler Totient Function

Posted by Mohammad Samiul Islam G+1 Recommend this on Google

Labels: Arithmetic Function, Modular Arithmetic, Number Theory, Proof

2 comments:

MH RIYAD December 4, 2015 at 11:57 AM

excellent

Reply

Shadab Anwar July 18, 2016 at 4:49 PM

We can also do it with the modification of sieve of eratosthenes.

Reply

forthright48

Learning Never Ends

Monday, September 7, 2015

Segmented Sieve of Eratosthenes

Problem

Given two integers A and B, find number of primes inside the range of A and B inclusive. He
$1 \leq A \leq B \leq 10^{12}$ and $B - A \leq 10^5$.

For example, $A = 11$ and $B = 19$, then answer is 4 since there are 4 primes within that rang

If limits of A and B were small enough ($\leq 10^8$), then we could solve this problem using the
here limits are huge, so we don't have enough memory or time to run normal sieve. But note t
$B - A \leq 10^5$. So even though we don't have memory/time to run sieve from 1 to N, we hav
memory/time to cover A to B.

A to B is a segment, and we are going to modify our algorithm for Sieve of Eratosthenes to c
Hence, the modified algorithm is called Segmented Sieve of Eratosthenes.

Make sure you fully understand how "Normal" Sieve of Eratosthenes works.

How Normal Sieve Works

First, let us see the steps for "Normal" sieve. In order to make things simpler, we will be lookin
unoptimized sieve. You can implement your own optimizations later.

Suppose we want to find all primes between 1 to N.

1. First we define a new variable $sqrtn = \sqrt{N}$.
2. We take all primes less than $sqrtn$.
3. For each prime p, we repeat the following steps.
    1. We start from $j = p \times p$.
    2. If $j \leq N$, we mark sieve at position j to be not prime.
       Else, we break out of the loop.
    3. We increase the value of j by p. And go back to step step 2 of this loop.
4. All positions in the sieve that are not marked are prime.

This is how the basic sieve works. We will now modify it to work on segments.

How Segmented Sieve Works

We will perform the same steps as normal sieve but just slightly modified.

Generate Primes Less Than $\sqrt{N}$

In the segmented sieve, what is he largest limit possible? 10^{12}. So let $N = 10^{12}$

First of all, in the normal sieve we worked with primes less than $\sqrt{N}$ only. So, if we had to run , we would have required only primes less than $\sqrt{N} = 10^6$. So in order to run sieve on a seg to N, we won't require primes greater than $\sqrt{N}$.

So, using normal sieve we will first generate all primes less than $\sqrt{N} = 10^6$.

Run on Segment

Okay, now we can start our "Segmented" Sieve. We want to find primes between A and B.

1. If A is equal to 1, then increase A by 1. That is, make $A = 2$. Since 1 is not a prime, thi change our answer.
2. Define a new variable $sqrtn = \sqrt{B}$.
3. Declare a new array of size $dif = \text{maximum difference of } (B - A) + 1$. Since it is problem that $B - A \leq 10^5$, $dif = 10^5 + 1$ for this problem.

   Let the array be called arr. This array has index from 0 to $dif - 1$. Here $arr[0]$ repres A, $arr[1]$ represents $A + 1$ and so on.
4. Now, we will be working with all primes less than $sqrtn$. These primes are already gene normal sieve.
5. For each prime p, we will repeat the following steps:
   1. We start from $j = p \times p$.
   2. But initial value of $j = p^2$ might be less than A. We want to mark positions betwee So we will need to shift j inside the segment.

      So, if $j < A$, then $j = ceil(\frac{A}{p}) \times p = \frac{A+p-1}{p} \times p$. This line makes j the smalles which is bigger than A.
   3. If $j \leq B$, we mark sieve at position j to be not prime.
      Else, we break out of the loop.

      But when marking, remember that our array arr is shifted by A positions. $arr[0]$ i A of normal sieve. So, we will mark position $j - A$ of arr as not prime.
   4. Increase the value of j by p. Repeat loop from step 3.
6. All positions in arr which has not been marked are prime.

Step 1 is important. Since we only mark multiples of prime as not prime in the pseducode abo no prime factor never gets marked. So we handle it by increassing value of A by 1 when $A =$

Code

If we convert the above pseducode into C++, then it becomes something like this:

```
1  int arr[SIZE];
2
3  ///Returns number of primes between segment [a,b]
4  int segmentedSieve ( int a, int b ) {
5      if ( a == 1 ) a++;
6
7      int sqrtn = sqrt ( b );
8
9      memset ( arr, 0, sizeof arr ); ///Make all index of arr 0.
10
11     for ( int i = 0; i < prime.size() && prime[i] <= sqrtn; i++ ) {
12         int p = prime[i];
13         int j = p * p;
14
15         ///If j is smaller than a, then shift it inside of segment [a
16         if ( j < a ) j = ( ( a + p - 1 ) / p ) * p;
17
18         for ( ; j <= b; j += p ) {
19             arr[j-a] = 1; ///mark them as not prime
20         }
21     }
22
23     int res = 0;
24     for ( int i = a; i <= b; i++ ) {
25         ///If it is not marked, then it is a prime
26         if ( arr[i-a] == 0 ) res++;
27     }
28     return res;
29 }
```

In line 1 we declare an array of $SIZE$. This array needs to be as large as maximum differenc $B - A + 1$. Next in line 4 we declare a function that finds number of primes between a and b is same as the psedocode above.

It is possible to optimize it further (both for speed and memory) but in expense of clarity. I am understand the core concept behind this algorithm, then they will have no problem tweaking t their needs.

Conclusion

I first learned about Segmented Sieve from blog of +Zobayer Hasan. You can have a look at t wasn't really good at bit manipulation back then, so it looked really scary. Later I realized it's n looks. Hopefully, you guys feel the same.

Leave a comment if you face any difficulty in understanding the post.

Reference

1. forthright48 - Sieve of Eratosthenes
2. zobayer - Segmented Sieve

Related Problems

1. SPOJ PRIME1 - Prime Generator
2. LOJ 1197 - Help Hanzo

forthright48

Learning Never Ends

Thursday, September 17, 2015

Euler's Theorem and Fermat's Little Theorem

We will be looking into two theorems at the same time today, Fermat's Little Theorem and Eul
Euler's Theorem is just a generalized version of Fermat's Little Theorem, so they are quite sim
We will focus on Euler's Theorem and its proof. Later we will use Euler's Theorem to prove Fe
Theorem.

Euler's Theorem

Theorem - Euler's Theorem states that, if a and n are coprime, then $a^{\phi(n)} \equiv 1 (\mod n$

Here $\phi(n)$ is Euler Phi Function. Read more about Phi Function on this post - Euler Totient or

Proof

Let us consider a set $A = \{b_1, b_2, b_3 \ldots, b_{\phi(n)}\} \pmod n$, where b_i is coprime to n and dist
are $\phi(n)$ elements which are coprime to n, A contains $\phi(n)$ integers.

Now, consider the set $B = \{ab_1, ab_2, ab_3 \ldots . ab_{\phi(n)}\} \pmod n$. That is, B is simply set A v
multiplied a with each element. Let a be coprime to n.

Lemma - Set A and set B contains the same integers.

We can prove the above lemma in three steps.

1. **A and B has the same number of elements**
   Since B is simply every element of A multiplied with a, it contains the same number of e
   This is obvious.
2. **Every integer in B is coprime to n**
   An integer in B is of form $a \times b_i$. We know that both b_i and a are coprime to n, so ab_i i
   n.
3. **B contains distinct integers only**
   Suppose B does not contain distinct integers, then it would mean that there is such a b_i

   $ab_i \equiv ab_j \pmod n$
   $b_i \equiv b_j \pmod n$

> But this is not possible since all elements of A are distinct, that is, b_i is never equal to b_j contains distinct elements.

With these three steps, we claim that, since B has the same number of elements as A which coprime to n, it has same elements as A.

Now, we can easily prove Euler's Theorem.

$$ab_1 \times ab_2 \times ab_3 \ldots \times ab_{\phi(n)} \equiv b_1 \times b_2 \times b_3 \ldots \times b_{\phi(n)} \pmod{n}$$
$$a^{\phi(n)} \times b_1 \times b_2 \times b_3 \ldots \times b_{\phi(n)} \equiv b_1 \times b_2 \times b_3 \ldots \times b_{\phi(n)} \pmod{n}$$
$$\therefore a^{\phi(n)} \equiv 1 \pmod{n}$$

Fermat's Little Theorem

Fermat's Little Theorem is just a special case of Euler's Theorem.

> **Theorem** - Fermat's Little Theorem states that, if a and p are coprime and p is a prime, t
> $a^{p-1} \equiv 1 \pmod{p}$ - Wikipedia

As you can see, Fermat's Little Theorem is just a special case of Euler's Theorem. In Euler's T worked with any pair of value for a and n where they are coprime, here n just needs to be pri

We can use Euler's Theorem to prove Fermat's Little Theorem.

Let a and p be coprime and p be prime, then using Euler's Theorem we can say that:

$a^{\phi(p)} \equiv 1 \pmod{p}$ (But we know that for any prime p, $\phi(p) = p - 1$)
$a^{p-1} \equiv 1 \pmod{p}$

Conclusion

Both theorems have various applications. Finding Modular Inverse is a popular application of It can also be used to reduce the cost of modular exponentiation. Fermat's Little Theorem is u Primality Test.

There are more applications but I think it's better to learn them as we go. Hopefully, I will be ab separate posts for each of the applications.

Reference

1. Wiki - Euler's Theorem
2. forthright48 - Euler Totient or Phi Function
3. Wiki - Fermat's Little Theorem

Posted by Mohammad Samiul Islam

G+1 Recommend this on Google

Labels: Modular Arithmetic, Number Theory, Proof, Theorem

forthright48

Learning Never Ends

Monday, September 21, 2015

Repeated Squaring Method for Modular Exponer

Previously on Modular Exponentiation we learned about Divide and Conquer approach to finc $B^P \% M$. In that article, which is recursive. I also mentioned about an iterative algorithm that value in same complexity, only faster due to the absence of recursion overhead. We will be lo faster algorithm on this post today.

Make sure you know about Bit Manipulation before proceeding.

Problem

Given three positive integers B, P and M, find the value of $B^P \% M$.

For example, $B = 2$, $P = 5$ and $M = 7$, then $B^P \% M = 2^5 \% 7 = 32 \% 7 = 4$.

Repeated Squaring Method

Repeated Squaring Method (RSM) takes the advantage of the fact that $A^x \times A^y = A^{x+y}$.

Now, we know that any number can be written as the sum of powers of 2. Just convert the nu Number System. Now for each position i for which binary number has 1 in it, add 2^i to the sur

For example, $(19)_{10} = (10011)_2 = (2^4 + 2^1 + 2^0)_{10} = (16 + 2 + 1)_{10}$

Therefore, in equation B^P, we can decompose P to the sum of powers of 2.

So let's say, $P = 19$, then $B^{19} = B^{2^4+2^1+2^0} = B^{16+2+1} = B^{16} \times B^2 \times B^1$.

This is the main concept for repeated squaring method. We decompose the value P to binary each position i (we start from 0 and loop till the highest position at binary form of P) for which in i_{th} position, we multiply B^{2^i} to result.

Code

Here is the code for RSM. The Explanation is below the code.

```
1  int bigmod ( int b, int p, int m ) {
2      int res = 1 % m, x = b % m;
3      while ( p ) {
4          if ( p & 1 ) res = ( res * x ) % m;
5          x = ( x * x ) % m;
6          p >>= 1;
7      }
8      return res;
9  }
```

Explanation

At line 1, we have the parameters. We simply send the value of B, P and M to this function, the required result.

At line 2, we initiate some variables. res is the variable that will hold our result. It contains the We will multiply b^{2^i} with result to find b^p. x is temporary variable that initially contains the value

Now, from line 3 the loop starts. This loop runs until p becomes 0. Huh? Why is that? Keep rea

At line 4 we first check whether the first bit of p is on or not. If it is on, then it means that we hav to our result. x contains that value, so we multiply x to res.

Now line 5 and 6 are crucial to the algorithm. Right now, x contains the value of b^{2^0} and we a the 0_{th} position of p at each step. We need to update our variables such that they keep worki other than 0.

First, we update the value of x. x contains the value of b^{2^i}. On next iteration, we will be workir $i+1$. So we need to update x to hold $b^{2^{i+1}}$.

$$b^{2^{i+1}} = b^{2^i \times 2^1} = b^{2^i \times 2} = b^{2^i + 2^i} = b^{2^i} \times b^{2^i} = x \times x.$$

Hence, new value of x is $x \times x$. We make this update at line 5.

Now, at each step we are checking the 0_{th} position of p. But next we need to check the 1_{st} p binary. Instead of checking 1_{st} position of p, what we can do is shift p 1 time towards right. No position of p is same as checking 1_{st} position. We do this update at line 6.

These two lines ensures that our algorithm works on each iteration. When p becomes 0, it me more bit to check, so the loop ends.

Complexity

Since there cannot be more than $log_2(P)$ bits in P, the loop at line 3 runs at most $log_2(P)$ ti complexity is $log_2(P)$.

Conclusion

RSM is significantly faster than D&C approach due to lack of recursion overhead. Hence, I alw method when I have to find Modular Exponentiation.

The code may seem a little confusing, so feel free to ask questions.

When I first got my hands on this code, I had no idea how it worked. I found it in a forum with a

Approach to Modular Exponentiation". Since then I have been using this code.

Resources

1. forthrigth48 - Modular Exponentiation
2. forthright48 - Bit Manipulation
3. algorithmist - Repeated Squaring
4. Wiki - Exponentiation by Squaring

Posted by Mohammad Samiul Islam G+1 Recommend this on Google

Labels: Binary, Bitwise, Modular Arithmetic, Repeated Squaring

No comments:

Post a Comment

Leave comments for Queries, Bugs and Hugs.

Enter your comment...

Comment as: Unknown (Goo

Publish Preview

Newer Post Home

Subscribe to: Post Comments (Atom)

Blog Archive

▼ 2015 (35)
▼ Sep (7)
Modular Inverse from 1 to N

Labels

Analysis Arithmetic Function Backtrack Bi
Complexity Contest D&C Divisors Factorial
Language LCM Logarithm Math M

forthright48

Learning Never Ends

Wednesday, September 23, 2015

Modular Multiplicative Inverse

Problem

Given value of A and M, find the value of X such that $AX \equiv 1 \pmod{M}$.

For example, if $A = 2$ and $M = 3$, then $X = 2$, since $2 \times 2 = 4 \equiv 1 \pmod{3}$.

We can rewrite the above equation to this:

$$AX \equiv 1 \pmod{M}$$
$$X \equiv \frac{1}{A} \pmod{M}$$
$$X \equiv A^{-1} \pmod{M}$$

Hence, the value X is known as Modular Multiplicative Inverse of A with respect to M.

How to Find Modular Inverse?

First we have to determine whether Modular Inverse even exists for given A and M before w the solution. Modular Inverse doesn't exist for every pair of given value.

Existence of Modular Inverse

Modular Inverse of A with respect to M, that is, $X = A^{-1} \pmod{M}$ exists, if and only if A a coprime.

Why is that?

$$AX \equiv 1 \pmod{M}$$
$$AX - 1 \equiv 0 \pmod{M}$$

Therefore, M divides $AX - 1$. Since M divides $AX - 1$, then a divisor of M will also divide suppose, A and M are not coprime. Let D be a number greater than 1 which divides both A will divide $AX - 1$. Since D already divides A, D must divide 1. But this is not possible. Ther equation is unsolvable when A and M are not coprime.

From here on, we will assume that A and M are coprime unless state otherwise.

Using Fermat's Little Theorem

Recall Fermat's Little Theorem from a previous post, "Euler's Theorem and Fermat's Little The that, if A and M are coprime and M is a prime, then, $A^{M-1} \equiv 1 (\mod M)$. We can use this the modular inverse.

$A^{M-1} \equiv 1 (\mod M)$ (Divide both side by A)
$A^{M-2} \equiv \frac{1}{A} (\mod M)$
$A^{M-2} \equiv A^{-}1 (\mod M)$

Therefore, when M is prime, we can find modular inverse by calculating the value of A^{M-2}. I calculate this? Using Modular Exponentiation.

This is the easiest method, but it doesn't work for non-prime M. But no worries since we have find the inverse.

Using Euler's Theorem

It is possible to use Euler's Theorem to find the modular inverse. We know that:

$A^{\phi(M)} \equiv 1 (\mod M)$
$\therefore A^{\phi(M)-1} \equiv A^{-1} (\mod M)$

This process works for any M as long as it's coprime to A, but it is rarely used since we have Phi value of M which requires more processing. There is an easier way.

Using Extended Euclidean Algorithm

We are trying to solve the congruence, $AX \equiv 1 (\mod M)$. We can convert this to an equatic

$AX \equiv 1 (\mod M)$
$AX + MY = 1$

Here, both X and Y are unknown. This is a linear equation and we want to find integer solutic means, this is a Linear Diophantine Equation.

Linear Diophantine Equation can be solved using Extended Euclidean Algorithm. Just pass e value of A and M and it will provide you with values of X and Y. We don't need Y so we car we simply take the mod value of X as the inverse value of A.

Code

A and M need to be coprime. Otherwise, no solution exists. The following codes do not chec coprime. The checking is left of the readers to implement.

When M is Prime

We will use Fermat's Little Theorem here. Just call the $bigmod()$ function from where you ne

```
1 int x = bigmod( a, m - 2, m ); ///(ax)%m = 1
```

Here x is the modular inverse of a which is passed to $bigmod()$ function.

When M is not Prime

For this, we have to use a new function.

```
int modInv ( int a, int m ) {
    int x, y;
    ext_gcd( a, m, &x, &y );

    ///Process x so that it is between 0 and m-1
    x %= m;
    if ( x < 0 ) x += m;
    return x;
}
```

I wrote this function since after using $\text{ext\_gcd}()$ we need to process x so that it's value is bet
$M-1$. Instead of doing that manually, I decided to write a function.

So, if we want to find the modular inverse of A with respect to M, then the result will be $X =$

Complexity

Repeated Squaring method has a complexity of $O(logP)$, so the first code has complexity of
whereas Extended Euclidean has complexity of $O(log_{10}A + log_{10}B)$ so second code has c
$O(log_{10}A + log_{10}M)$.

Why Do We Need Modular Inverse?

We need Modular Inverse to handle division during Modular Arithmetic. Suppose we are tryin
of the following equations:

$\frac{4}{2}\ \%\ 3$ - This is simple. We just simplify the equation and apply normal modular operation. Tha
$\frac{4}{2}\ \%\ 3 = 2\ \%\ 3 = 2$.

Then what happens when we try to do same with $\frac{12}{9}\ \%\ 5$? First we simply. $\frac{12}{9}\ \%\ 5 = \frac{4}{3}\ \%\ 5$.
facing an irreducible fraction. Should we simply perform the modular operation with numerator
denominator? That doesn't help since both of them are smaller than 5.

This is where Modular Inverse comes to the rescue. Let us solve the equation $X \equiv 3^{-1}\ (\text{mo}$
find the value of X? We will see that on the later part of the post. For now, just assume that w
of X.

Now, we can rewrite the above equation in the following manner:

$$\frac{12}{9}\ \%\ 5$$
$$\frac{4}{3}\ \%\ 5$$
$$(4 \times 3^{-1})\ \%\ 5$$
$$((4\ \%\ 5) \times (3^{-1}\ \%\ 5))\ \%\ 5$$
$$\therefore 4X\ \%\ 5$$

So, now we can easily find the value of $\frac{A}{B}\ \%\ M$ by simply calculating the value of $(A \times B^{-1}$

Conclusion

Modular Inverse is a small topic but look at the amount of background knowledge it requires t
Euler's Theorem, Euler Phi, Modular Exponentiation, Linear Diophantine Equation, Extended
Algorithm and other small bits of information. We covered them all before, so we can proceed

Hopefully, you understood how Modular Inverse works. If not, make sure to revise the articles
"Reference" section below.

Reference

1. Wiki - Modular Multiplicative Inverse
2. forthright48 - Euler's Theorem and Fermat's Little Theorem
3. forthright48 - Modular Exponentiation
4. forthright48 - Euler Phi
5. forthright48 - Linear Diophantine Equation
6. forthright48 - Extended Euclidean Algorithm

Posted by Mohammad Samiul Islam

G+1 Recommend this on Google

Labels: Modular Arithmetic, Number Theory

No comments:

Post a Comment

Leave comments for Queries, Bugs and Hugs.

Enter your comment...

Comment as: Unknown (Goo

Publish Preview

Newer Post Home

forthright48

Learning Never Ends

Saturday, September 26, 2015

Euler Phi Extension and Divisor Sum Theorem

Previously we learned about Euler Phi Function. Today we are going to look at two theorems r
Phi that frequently appears in CPPS. I am not sure whether these theorems have any official r
made them up. These allow easy references so I will be using these names from now on.

Euler Phi Extension Theorem

> **Theorem**: Given a number N, let d be a divisor of N. Then the number of pairs $\{a, N\}$
> $1 \leq a \leq N$ and $gcd(a, N) = d$, is $\phi(\frac{N}{d})$.

Proof

We will prove the theorem using Euler Phi Function and Arithmetic notion.

We need to find the numbe of pairs $\{a, N\}$ such that $gcd(a, N) = d$, where $1 \leq a \leq N$.

Both a and N are divisible by d and d is the GCD. So, if we divide both a and N by d, then th
have any common divisor.

$gcd(\frac{a}{d}, \frac{N}{d}) = 1,$ where $1 \leq a \leq N$.

We know that the possible values of a lie in range $1 \leq a \leq N$. What about the possible valu
lie between $1 \leq \frac{a}{d} \leq \frac{N}{d}$ otherwise a will cross its limit.

Therefore, $gcd(a, N) = d$, where $1 \leq a \leq N$ is same as, $gcd(\frac{a}{d}, \frac{N}{d}) = 1$, where $1 \leq \frac{a}{d} \leq$

So all we need to do is find the value of $gcd(\frac{a}{d}, \frac{N}{d}) = 1$, where $1 \leq \frac{a}{d} \leq \frac{N}{d}$.

Let $N' = \frac{N}{d}$ and $a' = \frac{a}{d}$. How many pairs of $\{a', N'\}$ are there such that $gcd(a', N') = 1$
$1 \leq a' \leq N'$? Isn't this what Euler Phi Function finds? The answer is $\phi(N') = \phi(\frac{N}{d})$.

Euler Phi Divisor Sum Theorem

Theorem: For a given integer N, the sum of Euler Phi of each of the divisors of N equa

$$N = \sum_{d|N} \phi(d)$$

Proof

The proof is simple. I have broken down the proof in the following chunks for the ease of unde

Forming Array A

Imagine, we the following fractions in a list:

$$\frac{1}{N}, \frac{2}{N}, \frac{3}{N} \cdots \frac{N}{N}$$

Not very hard to imagine right? Let us convert this into an array of pairs. So now, we have the A:

$$A = [\{1, N\}, \{2, N\}, \{3, N\} \ldots \{N, N\}]$$

So we have an array of form $\{a, N\}$, where a is between 1 and N. There are exactly N eler

Finding GCD of Pairs

Next, we find the GCD of each pair, g. What are the possible values of g? Since g must divide must be a divisor of N. Therefore, we can conclude that, GCD of pair $\{a, N\}$ will be one of th

Let the divisors of N be the following: $d_1, d_2, d_3 \ldots d_r$. So these are the only possible GCD.

Forming Parititions

Next, we form partitions P_i. Let us put all pairs which have $gcd(a, N) = d_i$ to partition P_i. Th have R partitions, where R is the number of divisor of N. Note that each pair will belong to or since a pair has a unique GCD. Therefore,

$$N = \sum_{i=1}^{R} P_i$$

Size of Each Paritition

How many elements does partition P_i contain? P_i has all the pairs $\{a, N\}$ such that $gcd(a,$ $1 \leq a \leq N$. Using Euler Phi Extension Theorem from above, this value is $\phi(\frac{N}{d_i})$.

Wrapping it Up

We are almost done with the proof. Since $N = \sum_{i=1}^{R} P_i$, we can now write:

$$N = \sum_{i=1}^{R} P_i = \sum_{i=1}^{R} \phi(\frac{N}{d_i})$$

But d_i is just a divisor of N. So we can simplify and write:

$$N = \sum_{d|N} \phi(\frac{N}{d}) = \sum_{d|N} \phi(d)$$

Conclusion

These theorems may look so simple that you might think they are useless. Especially Euler Pl Theorem, $N = \sum_{d|N} \phi(d)$. How is this useful at all? Hopefully, we will see one of its applicat

Reference

1. forthright48 - Euler Totient or Phi Function
2. Wiki - Divisor Sum

Posted by Mohammad Samiul Islam

G+1 Recommend this on Google

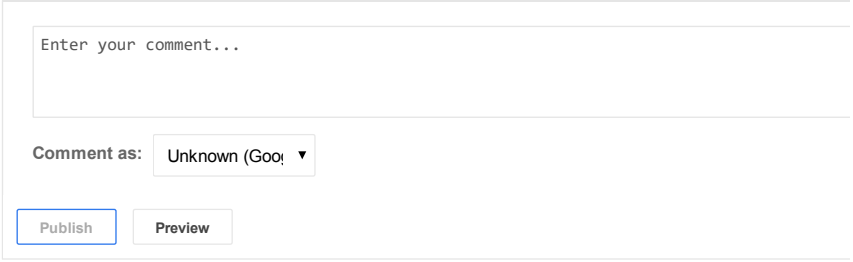
Labels: GCD, Number Theory, Proof, Theorem

1 comment:

MH RIYAD December 4, 2015 at 12:28 PM

plz give a link of a problem belong to the the theorem

Reply



Leave comments for Queries, Bugs and Hugs.

Newer Post Home

forthright48

Learning Never Ends

Tuesday, September 29, 2015

Modular Inverse from 1 to N

We already learned how to find Modular Inverse for a particular number in a previous post, "M Multiplicative Inverse". Today we will look into finding Modular Inverse in a bulk.

Problem

Given N and M ($N < M$ and M is prime), find modular inverse of all numbers between 1 to M.

Since M is prime and N is less than M, we can be sure that Modular Inverse exists for all nu Cause prime numbers are coprime to all numbers less than them.

We will look into two methods. Later one is better than the first one.

$O(NlogM)$ Solution

Using Fermat's little theorem, we can easily find Modular Inverse for a particular number.

$A^{-1} \% M = bigmod(A, M-2, M)$, where $bigmod()$ is a function from the post "Repeat Method for Modular Exponentiation". The function has complexity of $O(logM)$. Since we are inverse for all numbers from 1 to N, we can find them in $O(NlogM)$ complexity by running a

```
int inv[SIZE]; ///inv[x] contains value of (x^-1 % m)
for ( int i = 1; i <= n; i++ ) {
    inv[i] = bigmod ( i, m - 2, m );
}
```

But it's possible to do better.

$O(N)$ Solution

This solution is derived using some clever manipulation of Modular Arithmetic.

Suppose we are trying to find the modular inverse for a number a, $a < M$, with respect to M by a. This will be the starting point.

$M = Q \times a + r$, (where Q is the quotient and r is the remainder)
$M = \lfloor \frac{M}{a} \rfloor \times a + (M \% a)$

Now take modulo M on both sides.

$0 \equiv \lfloor \frac{M}{a} \rfloor \times a + (M \% a) \pmod{M}$
$(M \% a) \equiv - \lfloor \frac{M}{a} \rfloor \times a \pmod{M}$

Now divide both side by $a \times (M \% a)$.

$$\frac{M \% a}{a \times (M \% a)} \equiv \frac{- \lfloor \frac{M}{a} \rfloor \times a}{a \times (M \% a)} \pmod{M}$$

$$\therefore a^{-1} \equiv - \lfloor \frac{M}{a} \rfloor \times (M \% a)^{-1} \pmod{M}$$

The formula establishes a recurrence relation. The formula says that, in order to find the modu
we need to find the modular inverse of $b = M \% a$ first.

Since $b = M \% a$, we can say that its value lies between 0 and $a - 1$. But, a and M are cop
never fully divide M. Hence we can ignore the possibility that b will be 0. So possible values c
and $a - 1$.

Therefore, if we have all modular inverse from 1 to $a - 1$ already calculated, then we can find
inverse of a in $O(1)$.

Code

We can now formulate our code.

```
1  int inv[SIZE];
2  inv[1] = 1;
3  for ( int i = 2; i <= n; i++ ) {
4      inv[i] = (-(m/a) * inv[m%a] ) % m;
5      inv[i] = inv[i] + m;
6  }
```

In line 2, we set the base case. Modular inverse of 1 is always 1. Then we start calculating inv
When $i = 2$, all modular inverse from 1 to $i - 1 = 1$ is already calculated in array $inv[]$. So
in $O(1)$ using the formula above at line 4.

At line 5, we make sure the modular inverse is non-negative.

Next, when $i = 3$, all modular inverse from 1 to $i - 1 = 2$ is already calculated. This is proce
we reach N.

Since we calculated each inverse in $O(1)$, the complexity of this code is $O(N)$.

Conclusion

I saw this code first time on CodeChef forum. I didn't know how it worked back then. I added it and have been using it since then. Recently, while searching over the net for resources on Pc algorithm, I stumbled on an article from Come On Code On which had the explanation. Thank been looking for the proof.

Reference

1. forthright48 - Modular Multiplicative Inverse
2. forthright48 - Repeated Squaring Method for Modular Exponentiation
3. Come On Code On - Modular Multiplicative Inverse

Posted by Mohammad Samiul Islam G+1 Recommend this on Google

Labels: Modular Arithmetic, Number Theory, Proof

3 comments:

Shubham Aggarwal January 31, 2016 at 12:25 AM

thanks Mohammad Samiul Islam for great article .

Reply

Shubham Aggarwal January 31, 2016 at 12:49 AM

https://www.hackerrank.com/contests/worldcodesprint/challenges/colorful-ornaments/editorial editorial why are you not using O(n) approach for calculating modinv ?

Reply

Replies

Mohammad Samiul Islam January 31, 2016 at 10:56 AM

I did. Maybe you got confused seeing modInv() function in my template. That's just the use that function in the code. Look inside the precal() function and you will see that I c array and used that all over the place :)

Reply

Enter your comment...

Comment as: Unknown (Goo

Publish Preview

Wolfram MathWorld
Built with Mathematica Technology
the web's most extensive mathematics resource

Algebra
Applied Mathematics
Calculus and Analysis
Discrete Mathematics
Foundations of Mathematics
Geometry
History and Terminology
Number Theory
Probability and Statistics
Recreational Mathematics
Topology

Alphabetical Index
Interactive Entries
Random Entry
New in *MathWorld*

MathWorld Classroom

About *MathWorld*
Contribute to *MathWorld*
Send a Message to the Team

MathWorld Book

Wolfram Web Resources »

13,594 entries
Last updated: Thu Aug 18 2016

Created, developed, and nurtured by Eric Weisstein at Wolfram Research

Number Theory > Prime Numbers > Prime Representations >
Foundations of Mathematics > Mathematical Problems > Unsolved Problems >
Foundations of Mathematics > Mathematical Problems > Prize Problems >
More...

Goldbach Conjecture

DOWNLOAD Wolfram Notebook
CONTRIBUTE To this Entry

Goldbach's original conjecture (sometimes called the "ternary" Goldbach conjecture), writte to Euler, states "at least it seems that every number that is greater than 2 is the sum of three Dickson 2005, p. 421). Note that here Goldbach considered the number 1 to be a prime, a longer followed. As re-expressed by Euler, an equivalent form of this conjecture (called the Goldbach conjecture) asserts that all positive even integers ≥ 4 can be expressed as the s primes (p, q) such that $p+q=2n$ for n a positive integer are sometimes called a Goldbac Silva).

According to Hardy (1999, p. 19), "It is comparatively easy to make clever guesses; indeed 'Goldbach's Theorem,' which have never been proved and which any fool could have gues offered a $1000000 prize to anyone who proved Goldbach's conjecture between March 20 2002, but the prize went unclaimed and the conjecture remains open.

Schnirelman (1939) proved that every even number can be written as the sum of not more (Dunham 1990), which seems a rather far cry from a proof for *two* primes! Pogorzelski (197 the Goldbach conjecture, but his proof is not generally accepted (Shanks 1985). The follow bounds n such that the strong Goldbach conjecture has been shown to be true for numbers

```
int minimum_prime(int n)
{
if(n == 2)return 1;
else if(n == 3) return 1;
else if(n < 2) return 0;
else if(n % 2 ==0 && n>2)
{ return 2;}
else if(is_prime(n))
{return 1;}
else if(is_prime(n-2))
{return 2;}
else if(n % 2 == 1)
{return 3;}
}
```

bound	reference
1×10^4	Desboves 1885
1×10^5	Pipping 1938
1×10^8	Stein and Stein 1965ab
2×10^{10}	Granville et al. 1989
4×10^{11}	Sinisalo 1993
1×10^{14}	Deshouillers et al. 1998
4×10^{14}	Richstein 1999, 2001
2×10^{16}	Oliveira e Silva (Mar. 24, 2003)
6×10^{16}	Oliveira e Silva (Oct. 3, 2003)
2×10^{17}	Oliveira e Silva (Feb. 5, 2005)
3×10^{17}	Oliveira e Silva (Dec. 30, 2005)
12×10^{17}	Oliveira e Silva (Jul. 14, 2008)
4×10^{18}	Oliveira e Silva (Apr. 2012)

The conjecture that all odd numbers ≥ 9 are the sum of three odd primes is called the "wea Vinogradov (1937ab, 1954) proved that every sufficiently large odd number is the sum of th p. 66; Guy 1994), and Estermann (1938) proved that almost all even numbers are the sums Vinogradov's original "sufficiently large" $N \geq 3^{3^{15}} \approx e^{e^{16.573}} \approx 3.25\times10^{6846168}$ was subsec $e^{e^{11.503}} \approx 3.33\times10^{43000}$ by Chen and Wang (1989). Chen (1973, 1978) also showed that a numbers are the sum of a prime and the product of at most two primes (Guy 1994, Courant

A stronger version of the weak conjecture, namely that every odd number ≥ 7 can be expr prime plus twice a prime is known as Levy's conjecture.

An equivalent statement of the Goldbach conjecture is that for every positive integer m, the such that

$$\phi(p) + \phi(q) = 2m,$$

where $\phi(x)$ is the totient function (e.g., Havil 2003, p. 115; Guy 2004, p. 160). (This follows $\phi(p) = p - 1$ for p prime.) Erdős and Moser have considered dropping the restriction that equation as a possibly easier way of determining if such numbers always exist (Guy 1994,

Other variants of the Goldbach conjecture include the statements that every even number primes, and every integer > 17 the sum of exactly three distinct primes.

Let $R(n)$ be the number of representations of an even number n as the sum of two primes. Goldbach conjecture states that

$$R(n) \sim 2\Pi_2 \prod_{\substack{k=2 \\ p_k \mid n}} \frac{p_k - 1}{p_k - 2} \int_2^n \frac{dx}{(\ln x)^2},$$

where Π_2 is the twin primes constant (Halberstam and Richert 1974).

SEE ALSO:
Chen's Theorem, de Polignac's Conjecture, Goldbach Number, Goldbach Partition, Levy's Partition, Schnirelmann's Theorem, Untouchable Number, Waring's Prime Number Conjec

REFERENCES:

Ball, W. W. R. and Coxeter, H. S. M. *Mathematical Recreations and Essays, 13th ed.* New York: Dov

Caldwell, C. K. "Prime Links++." http://primes.utm.edu/links/theory/conjectures/Goldbach/.

Chen, J. R. "On the Representation of a Large Even Integer as the Sum of a Prime and the Product o *Sinica* **16**, 157-176, 1973.

Chen, J. R. "On the Representation of a Large Even Integer as the Sum of a Prime and the Product o *Sci. Sinica* **21**, 421-430, 1978.

Chen, J. R. and Wang, T.-Z. "On the Goldbach Problem." *Acta Math. Sinica* **32**, 702-718, 1989.

Courant, R. and Robbins, H. *What Is Mathematics?: An Elementary Approach to Ideas and Methods,* Oxford University Press, pp. 30-31, 1996.

Deshouillers, J.-M.; te Riele, H. J. J.; and Saouter, Y. "New Experimental Results Concerning The Go *Algorithmic Number Theory: Proceedings of the 3rd International Symposium (ANTS-III) held at Reed 21-25, 1998* (Ed. J. P. Buhler). Berlin: Springer-Verlag, pp. 204-215, 1998.

Devlin, K. *Mathematics: The New Golden Age, rev. ed.* New York: Columbia University Press, 1999.

Dickson, L. E. "Goldbach's Empirical Theorem: Every Integer is a Sum of Two Primes." In *History of Vol. 1: Divisibility and Primality.* New York: Dover, pp. 421-424, 2005.

Doxiadis, A. *Uncle Petros and Goldbach's Conjecture.* Faber & Faber, 2001.

Dunham, W. *Journey through Genius: The Great Theorems of Mathematics.* New York: Wiley, p. 83,

Estermann, T. "On Goldbach's Problem: Proof that Almost All Even Positive Integers are Sums of Tw *Math. Soc. Ser. 2* **44**, 307-314, 1938.

Faber and Faber. "$1,000,000 Challenge to Prove Goldbach's Conjecture." Archived at http://web.archive.org/web/20020803035741/www.faber.co.uk/faber/million_dollar.asp.

Goldbach, C. Letter to L. Euler, June 7, 1742.

Granville, A.; van der Lune, J.; and te Riele, H. J. J. "Checking the Goldbach Conjecture on a Vector *Theory and Applications: Proceedings of the NATO Advanced Study Institute held in Banff, Alberta, A* R. A. Mollin). Dordrecht, Netherlands: Kluwer, pp. 423-433, 1989.

Guy, R. K. "Goldbach's Conjecture." §C1 in *Unsolved Problems in Number Theory, 2nd ed.* New Yor 107, 1994.

Guy, R. K. *Unsolved Problems in Number Theory, 3rd ed.* New York: Springer-Verlag, 2004.

Halberstam, H. and Richert, H.-E. *Sieve Methods.* New York: Academic Press, 1974.

Hardy, G. H. *Ramanujan: Twelve Lectures on Subjects Suggested by His Life and Work, 3rd ed.* Nev

Hardy, G. H. and Littlewood, J. E. "Some Problems of 'Partitio Numerorum.' III. On the Expression of Primes." *Acta Math.* **44**, 1-70, 1923.

Hardy, G. H. and Littlewood, J. E. "Some Problems of Partitio Numerorum (V): A Further Contribution Problem." *Proc. London Math. Soc. Ser. 2* **22**, 46-56, 1924.

Hardy, G. H. and Wright, E. M. *An Introduction to the Theory of Numbers, 5th ed.* Oxford, England: C

Havil, J. *Gamma: Exploring Euler's Constant.* Princeton, NJ: Princeton University Press, 2003.

Nagell, T. *Introduction to Number Theory.* New York: Wiley, p. 66, 1951.

Oliveira e Silva, T. "Goldbach Conjecture Verification." http://www.ieeta.pt/~tos/goldbach.html.

Oliveira e Silva, T. "Verification of the Goldbach Conjecture Up to 2*10^16." Mar. 24, 2003a. http://listserv.nodak.edu/scripts/wa.exe?A2=ind0303&L=nmbrthry&P=2394.

Oliveira e Silva, T. "Verification of the Goldbach Conjecture Up to 6×10^{16}." Oct. 3, 2003b. http://listserv.nodak.edu/scripts/wa.exe?A2=ind0310&L=nmbrthry&P=168.

Oliveira e Silva, T. "New Goldbach Conjecture Verification Limit." Feb. 5, 2005a. http://listserv.nodak.e
A1=ind0502&L=nmbrthry#9.

Oliveira e Silva, T. "Goldbach Conjecture Verification." Dec. 30, 2005b. http://listserv.nodak.edu/cgi-b
A2=ind0512&L=nmbrthry&T=0&P=3233.

Peterson, I. "Prime Conjecture Verified to New Heights." *Sci. News* **158**, 103, Aug. 12, 2000.

Pipping, N. "Die Goldbachsche Vermutung und der Goldbach-Vinogradovsche Satz." *Acta. Acad. Abo*
1938.

Pogorzelski, H. A. "Goldbach Conjecture." *J. reine angew. Math.* **292**, 1-12, 1977.

Richstein, J. "Verifying the Goldbach Conjecture up to $4\cdot 10^{14}$." Presented at *Canadian Number The*
Winnipeg/Canada June 20-24, 1999.

Richstein, J. "Verifying the Goldbach Conjecture up to $4\cdot 10^{14}$." *Math. Comput.* **70**, 1745-1750, 2001.

Schnirelman, L. G. *Uspekhi Math. Nauk* **6**, 3-8, 1939.

Shanks, D. *Solved and Unsolved Problems in Number Theory, 4th ed.* New York: Chelsea, pp. 30-31

Sinisalo, M. K. "Checking the Goldbach Conjecture up to $4\cdot 10^{11}$." *Math. Comput.* **61**, 931-934, 1993.

Stein, M. L. and Stein, P. R. "New Experimental Results on the Goldbach Conjecture." *Math. Mag.* **38**,

Stein, M. L. and Stein, P. R. "Experimental Results on Additive 2 Bases." *BIT* **38**, 427-434, 1965b.

Vinogradov, I. M. "Representation of an Odd Number as a Sum of Three Primes." *Comptes rendus (L*
Sciences de l'U.R.S.S. **15**, 169-172, 1937a.

Vinogradov, I. "Some Theorems Concerning the Theory of Primes." *Recueil Math.* **2**, 179-195, 1937b.

Vinogradov, I. M. *The Method of Trigonometrical Sums in the Theory of Numbers.* London: Interscien

Woon, M. S. C. "On Partitions of Goldbach's Conjecture" 4 Oct 2000. http://arxiv.org/abs/math.GM/0(

Wang, Y. *Goldbach Conjecture.* Singapore: World Scientific, 1984.

CITE THIS AS:

Weisstein, Eric W. "Goldbach Conjecture." From *MathWorld*--A Wolfram Web Resource.
http://mathworld.wolfram.com/GoldbachConjecture.html

Wolfram Web Resources

Mathematica »
The #1 tool for creating Demonstrations and anything technical.

Wolfram|Alpha »
Explore anything with the first computational knowledge engine.

Wolfram De
Explore tho
across scie
engineering
finance, soc

Computerbasedmath.org »
Join the initiative for modernizing math education.

Online Integral Calculator »
Solve integrals with Wolfram|Alpha.

Step-by-ste
Walk throug
by-step from
help you try

Wolfram Problem Generator »
Unlimited random practice problems and answers with built-in Step-by-step solutions. Practice online or make a printable study sheet.

Wolfram Education Portal »
Collection of teaching and learning tools built by Wolfram education experts: dynamic textbook, lesson plans, widgets, interactive Demonstrations, and more.

Wolfram La
Knowledge
everyone.

Contact the *MathWorld* Team

© 1999-2016 Wolfra

Wilson's Theorem

In number theory, **Wilson's Theorem** states that if integer $p > 1$, then $(p-1)! + 1$ is divisible by p if and only if p is prime. It was stated by John Wilson. The French mathematician Lagrange proved it in 1771.

Contents

Proofs

Suppose first that p is composite. Then p has a factor $d > 1$ that is less than or equal to $p-1$. Then d divides $(p-1)!$, so d does not divide $(p-1)! + 1$. Therefore p does not divide $(p-1)! + 1$.

Two proofs of the converse are provided: an elementary one that rests close to basic principles of modular arithmetic, and an elegant method that relies on more powerful algebraic tools.

Elementary proof

Suppose p is a prime. Then each of the integers $1, \ldots, p-1$ has an inverse modulo p. (Indeed, if one such integer a does not have an inverse, then for some distinct b and c modulo p, $ab \equiv ac \pmod{p}$, so that $a(b-c)$ is a multiple of p, when p does not divide a or $b-c$–a contradiction.) This inverse is unique, and each number is the inverse of its inverse. If one integer a is its own inverse, then

$$0 \equiv a^2 - 1 \equiv (a-1)(a+1) \pmod{p},$$

so that $a \equiv 1$ or $a \equiv p-1$. Thus we can partition the set $\{2, \ldots, p-2\}$ into pairs $\{a, b\}$ such that $ab \equiv 1 \pmod{p}$. It follows that $(p-1)$ is the product of these pairs times $1 \cdot (-1)$. Since the product of each pair is conguent to 1 modulo p, we have

$$(p-1)! \equiv 1 \cdot 1 \cdot (-1) \equiv -1 \pmod{p},$$

as desired. ■

Algebraic proof

Let p be a prime. Consider the field of integers modulo p. By Fermat's Little Theorem, every nonzero element of this field is a root of the polynomial

$$P(x) = x^p - 1.$$

Since this field has only $p-1$ nonzero elements, it follows that

$$x^p - 1 = \prod_{r=1}^{p-1} (x - r).$$

Now, either $p = 2$, in which case $a \equiv -a \pmod{2}$ for any integer a, or $p-1$ is even. In either case, $(-1)^{p-1} \equiv 1 \pmod{p}$, so that

$$x^p - 1 = \prod_{r=1}^{p-1}(x - r) = \prod_{r=1}^{p-1}(-x + r).$$

If we set x equal to 0, the theorem follows. ■

Problems

Introductory

- Let a be an integer such that $\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{23} = \frac{a}{23!}$. Find the remainder when a is divided by 13.

Advanced

- If p is a prime greater than 2, define $p = 2q + 1$. Prove that $(q!)^2 + (-1)^q$ is divisible by p. <url>Forum/viewtopic.php?&t=21733 Solution</url>
- Let p be a prime number such that dividing p by 4 leaves the remainder 1. Show that there is an integer n such that $n^2 + 1$ is divisible by p.

See also

- Number theory
- Modular arithmetic
- Fermat's Little Theorem
- Factorial
- Wilson Prime

Retrieved from "http://artofproblemsolving.com/wiki/index.php?title=Wilson%27s_Theorem&oldid=76419"
Category: Number Theory

Copyright © 2016 Art of Problem Solving