GCD

Euclidean algorithm (finding gcd)

In C++, one can use __gcd(a,b) to find the gcd of a and b.

- gcd(a,b)=gcd(a-b,b) if a>=b
  =gcd(a-2*b,b) if a-b>=b
  =
  =
  =gcd(a%b,b)

Note: gcd(a,0)=a

Code for finding GCD

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

   return gcd(b,a%b);

}
```

Time Complexity: O(log(min(a,b)) or O(logN) if a and b are of order N

0<=a%b<=b-1 (Basic property of modulo)-> (1)
a%b<=a-b // a>=b ----------------------------> (2)
Adding (1) and (2):
2*(a%b)<=a-1
=> (a%b)<=(a-1)/2
=> (a%b)<=a/2
In each gcd call, the value get reduced by atleast ½ , so logarithmic time complexity.

- lcm(a,b)=(a*b)/gcd(a,b)

Extended Euclidean Algo

gcd(a,b)=gcd(a%b,b)=gcd(b,a%b)

```
a*x+b*y=gcd(a,b)-->(1)
b*x1+(a%b)*y1=gcd(b,a%b)
```

```
=>b*x1+(a%b)*y1=gcd(a,b)
=>b*x1+(a-(a/b)*b)*y1=gcd(a,b)
=>b*x1+a*y1-((a/b)*b)*y1=gcd(a,b)
=>a*y1+b*(x1-(a/b)*y1)=gcd(a,b)--(2)

x=y1
y=x1-(a/b)*y1

// {x,y} s.t a*x+b*y=gcd(a,b)  (a,b)-> known (x,y)
// a=3 , b=2  => 3*x+2*y=1  => (1,-1)

// find(3,2)=> return {1,-1}

// (3*x)%5=1

// a,b--> gcd(a,b)=1
// a*x+b*y=1 ->{x,y}
// (a*x)%b+(b*y)%b=1
// (a*x)%b=1

// x is the modulo inverse of a (mod b)


// (a%b)<=a/2

// a=13  b=10
// find(10,3)
pair<int,int> find(int a,int b){
    // base case : b=0
    // a*x+b*y=1
    // a*x+b*y=a
    // {1,0}

    if(b==0)
    return make_pair(1,0);

  pair<int,int> temp=find(b,a%b);  // {x1,y1}
  int x1=temp.first;
  int y1=temp.second;
```

```
  return make_pair(y1,x1-(a/b)*y1);

}
```

Link for Reference:
https://www.geeksforgeeks.org/euclidean-algorithms-basic-and-extended/

Problem 1:[Make Them Equal]

```
// Check if the ans=0 ,1 or 2
// ans will be atmost 2
// if all characters are equal to given character ans=0
// if choosing some position , you can change every character to given
character ans is 1
// else ans is 2

// for checking for ans to be 1 , you can either do a O(nlogn) or O(n) checkup
// In this code, it's O(nlogn)
// For O(n) , just check if there exists any position greater than n/2 which
has character equal to given character if so, just choosing that pos will make
everything equal

#include <bits/stdc++.h>
#define ll long long int
using namespace std;
bool allEqual(string &s,char c){
    for(auto e:s){
        if(e!=c)
        return false;
    }
    return true;
}
int main(){
 ll t;
 cin>>t;
 while(t--){
     ll n;
     cin>>n;
     char c;
     cin>>c;
     string s;
     cin>>s;

   //  s='*'+s;
```

```
    ll ans=0;
    vector<ll>res;

    if(!allEqual(s,c)){
        // ans=1?
        bool f=false;

        for(ll i=2;i<=n;i++){
            bool flag=true;

            for(ll j=i;j<=n;j+=i){
                if(s[j-1]!=c){
                    flag=false;
                    //x=j;
                    break;
                }
            }
            if(flag){
                f=true;
                res.push_back(i);
                break;
            }
        }

        if(!f){
            res.push_back(n);
            res.push_back(n-1);
        }
    }

    ans=res.size();
    cout<<ans<<endl;

    if(ans==0)
    continue;

    for(auto e:res)
    cout<<e<<" ";

    cout<<endl;
 }
}
```

Modular GCD

```
A,B,N

 gcd(A^N+B^N,|a-b|)%(10000000007)
 A,B,N<=10^12

 gcd((a^n+b^n),|a-b|) -> WA


 gcd(a,b)%c!=gcd(a%c,b%c)  ->WA  // don't do this ever xD

gcd(a,b)=gcd(a%b,b)

gcd((a^n+b^n)%(|a-b|),|a-b|)

dif=|a-b|
a=10^12
b=2


0<(a^n+b^n)%dif  <dif
=((a^n)%dif+(b^n)%dif)%dif

// Binary (a^n)%dif -> O(log(n)) -> O(log(10^12))



// (a*b)%mod

// a^(b^c)

ll mul_pow_mod(ll a,ll b,ll mod){
  ll res=0 ;
```

```
  while(b>0){
      if(b%2)
      res=(res+a)%mod;    // res<10^12  a<10^12  res+a<2*10^12
      b/=2;
      a=(a+a)%mod;       // 10^9
  }
  return res;
}

                  10^12    *  10^12
// (res*a)%mod =((res%mod)*(a%mod))%mod

// a*b =a+a+a+a+.... b times O(b)
  //     =  O(logb)

// a^b % mod

ll pow_mod(ll a,ll b,ll mod){
  ll res=1 ;

  while(b>0){
      if(b%2)
      res=mul_pow_mod(res,a,mod);   // res<10^12  a<10^12  10^24
      b/=2;
      a=mul_pow_mod(a,a,mod);      // 10^9
  }
  return res;
}
gcd(x,y)  x,y<=10^12
```

Code:

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

ll add_mod(ll a, ll b, ll mod)
```

```
{
    ll res = 0;

    while (b > 0)
    {
        if (b % 2)
            res = (res + a) % mod;
        a = (a + a) % mod;
        b /= 2;
    }

    return res;
}

ll pow_mod(ll a, ll b, ll mod)
{
    ll res = 1;

    while (b > 0)
    {
        if (b % 2)
            res = add_mod(res,a,mod);
        a = add_mod(a, a,mod);
        b /= 2;
    }

    return res;
}

int main()
{
    ll t;
    cin >> t;
    while (t--)
    {
        ll a, b, n;
        cin >> a >> b >> n;
        ll ans = -1;
        // gcd(a^n+b^n,a-b);
```

```
        if (a < b)
            swap(a, b);

        // (a^n)%g  where g<=10^12

        ll dif = a - b;

        if (dif == 0)
        {
            ans = pow_mod(a, n, 1000000007);
            ans += pow_mod(b, n, 1000000007);
            ans %= 1000000007;
        }
        else
        {
            ll x = pow_mod(a, n, dif);
            x += pow_mod(b, n, dif);
            x %= dif;
            ans = __gcd(x, dif);

            ans %= 1000000007;
        }

        cout << ans << endl;
    }
    return 0;
}
```

EULER'S TOTIENT FUNCTION

n -> 1, 2, 3 ,4 5, ... n
such that they are co prime with n
coprime -> gcd(x,y) =1 then x,y are coprime

phi (5) -> 1, 2 ,3 ,4 (4 is the answer)

phi (1) -> 1 (1 is the answer)

Naive approach-> nlogn complexity

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

int ans=0;

for ( int i=1 ; i<= n ;i++){ // n log (n)
    if(gcd(i,n)==1){
        ans++;
    }
}
cout<<ans;
```

phi(2)=1
phi(3)=2
phi(5)=4
phi(7)=6
phi(11)=10

phi(n) =n-1 // if n is prime

Ex- phi(7) = 7-1 =6
1,2,3,4,5,6 -> these are co prime to 7

phi (n^x) = ? // where n is prime and x is positive integer

phi (n^x) = n^x - no. of integers not coprime to n^x

phi (n^x) = n^x - n^x/ n

3 ^ 5 -> 3, 6, 9, 12, 15, 18..........

3^5/3 total such numbers

phi (n^x) = n^(x-1) * (n - 1) // where n is prime

an arithematic function f(x) is called multiplicative if
->f(n*m) = f(n) * f(m)

->phi is an multiplicative arithmetic function but n and m should be co prime for the above property to hold

phi (x) = p1^ c1 * p2^c2 .... pk^ck

30-> 2^1 * 5^1 * 3^1

phi (x) = phi(p1^ c1) * phi(p2^c2) .... phi(pk^ck)

phi(x) = p1^ (c1-1) * (p1-1) *............pk^ (ck-1) * (pk-1)

phi (x) = x * (p1-1)/p1 * (p2-1)/p2 ....(pk-1)/pk //

phi (15) = 15 * (3-1)/3 * (5-1)/5

Complexity reduced to O(sqrt(n))

Euler's Theorem

x ^ phi(n) <-congruent to-> 1 (mod n) // x and n are coprime

x = 4 , n =165 // gcd(4,165) =1

phi(165) = 80
phi(165) = phi(3*5*11) = phi(3) *phi(5) *phi(11) = 2* 4*10 =80

4^80 mod 165 =1

x ^ phi(n) <-congruent to-> 1 (mod n) // x and n are coprime

if n is prime then what will the expression reduce to?

It reduces to fermat's little theorem (as phi(n) is n-1 when n is prime)

x ^ n-1 <-congruent to-> 1 (mod n) // x and n are coprime // n is prime