18011019 - Feyza Şahin

X^(Y/Z) ÇÖZÜMÜ ALGORİTMASI

ALGORİTMANIN MANTIĞI:

Kodda, dışarıdan kullanıcının girdiği x, y ve z tam sayı değişkenlerine ek olarak bir de [x^(1/z) = kök] ⇒ [x = kök^z] denklemini sağlayan 'kök' adında bir değişkenimiz mevcut. Burada kullanılan denklemin mantığını küçük bir örnek üzerinden görelim: 8^(1/3) = 2. Yani aslında x^(1/z)=kök denkleminde her tarafın z. kuvvetini alıyoruz ve gerçek kök değerini ararken artık x=kök^z denkleminden yola çıkarak kök değerini, kök^z x'e eşit mi diye kontrol ederek değiştiriyoruz.

Bu algoritmada yürütülmesi gereken iki işlem var: y değişkeninin uygulanma süreci olan üs alma işlemi ve z değişkeninin uygulanma süreci olan kök bulma işlemi. Bu çözümde önce kök bulma işlemini uygulayıp sonrasında çıkan değerin üssünü almayı tercih ettim. Nedeni, önce üs alınırsa -x, y ve tam sayılar olduğu için- x^y değerinin kodun çalıştıramayacağı kadar büyük olma ihtimali var.

Kök değerini bulmak için aslında en basit mantıkla, bir while döngüsünde 1'den başlayıp birer artan bir değişken ile bu sayıların z üslerinin x'e eşit olup olmaması kontrol edilebilirdi. Eşit olana kadar değişken 1 artırılırdı, eşit olduğu takdirde döngüden çıkılırdı. Fakat bu iş oldukça uzun bir yol olduğundan zaman açısından verimsiz bir yoldur.

Bir sayının tüm çarpanları aynı olacak şekilde bir işlem düşünelim: 2*2*2*2=16. Aynı sonuca 4*4=16 şeklinde çarpanlar büyütülerek fakat adedi azaltılarak da ulaşılabilir. Buradan şu sonuca varabiliriz: aynı sonucu veren eş çarpanların değerleri ve adetleri ters orantılıdır. Girilen x ve z değerleri arasındaki ilişki de bu şekildedir.

Bu durumu algoritmayı geliştirebilmek açısından düşünürsek: örneğin sürekli 256^(⅛) gibi 'kök' değişkeni (bu soru için 2) çok küçük çıkacak şekilde sayılar girilseydi yukarıda bahsedilen while döngüsünü küçük sayılardan başlatmak da her zaman için daha verimli olabilirdi çünkü döngüde kontrol etmeye 1'den başladığımızı düşünürsek -hemen 2. döngüde- 2^8'in 256'ya eşit olduğunu görebilecektik. Fakat bu z değeri 2 olarak da girilebilirdi (256^(½) durumu). Bu durum için ise döngüde kontrol etmeye büyük sayılardan başlamamız mantıklı olacaktı.

İşte bu durum tamamen kullanıcıya bağlı olup girilen x ve z değerlerini önceden kestiremediğimiz için geneline uygulanabilecek kısa bir yol bulamadım. Fakat kontrol yolunu az da olsa pratikleştirebilmek adına şöyle bir mantık ürettim: kök^z x'e eşit mi diye bakacağımız ve buna göre 'kök' değeriyle oynayacağımız kontroller için referans noktasını, köke yaklaşık olabileceğini düşündüğüm z'nin, kendi üssünü ayarladım. Tabi bu durumda 3 farklı durum için 3 de farklı karmaşıklık değeri incelemiş olacağız.

x

↙ ↓ ↘

z^z = x ⇒ kök = z   z^z < x   z^z > x ⇒ Bisection

↓

kök = z;

kök^z !< x olana kadar kök = kök * z

↓

Bisection

Algoritmayı 3 büyük if kontrolünden oluşturup ilk if kontrolünde z^z'nin x'e eşit olup olmadığına baktım. Diğer iki if kontrollerini de x'in bu değerden büyük ya da küçük olma durumları için geliştirdim.

Eğer z^z=x ise: kök=z'dir.

Karmaşıklık: kök'ün kendisinin z'ye eşit olduğu hemen ilk kontrolde bulunduğu için herhangi bir kontrol sürecine girmek gerekmeyeceğinden direkt x^y değeri hesaplanır. Bu x^y işlemi için ise üs alma fonksiyonunda, x değerini kendisiyle y kere çarptığımızdan dolayı y adet işlem vardır. Yani karmaşıklığımız **Y**'dir.

Eğer z^z>x ise: Kökün 0 ile z arasında olacağı kesin olduğundan burada flag=0 değeriyle kodda oluşturulan bisection_kok fonksiyonuna gidilir.

(Bisection Metodu: Alt ve üst değerlerinin belli olduğu bir aralıkta kök arama metodudur. İlk olarak alt ve üst değerleri toplanıp 2'ye bölünerek bu ortadaki değere 'kök mü?' diye bakılır. Eğer bu orta değeri fonksiyona yerleştirdiğimizde sonuç asıl sonuçtan daha büyük çıkarsa demektir ki kök bu orta değerden daha küçüktür ve bir sonraki adımda kök, alt ve orta değerleri arasında aranmalıdır. Orta değer üst olarak değiştirilir; sonuç daha küçük çıkarsa bu sefer de orta değer alt değer olarak değiştirilir. Her seferinde aralık bu şekilde yarıya indirilerek kökün daha az adımda bulunma ihtimali de artırılmış olur. Karmaşıklık açısından bu yolu değerlendirecek olursak: kök'e kadarki her noktaya 1'den başlanıp bakıldığı takdirde en kötü durum için 'n' durum gerekirdi fakat bu metot ile en fazla $\log_2(n)$ işlemde köke ulaşılmış olur.)

[Bu fonksiyonda köke yaklaşmak için 30 seferlik bir for döngüsü ile nümerik yöntemlerden Bisection metodunu oluşturdum. Metodun başlangıcındaki alt ve üst değerleri de -gerçek köke ulaşana kadar yine kendisini kullanacağımız 'kök' değişkeninin büyüklüğüne göre- "flag" değişkeni ile belirledim:

flag=0: kök^z<x durumu.
(Eğer x negatifse bu sefer kök kesinlikle -z ile 0 arasında olacağından alt_taban olarak -z'nin; ust_taban olarak da 0'ın belirlenmesi gerekirdi fakat bunun yerine kodda olduğu gibi, x'in mutlak değeri baz alınıp kök yine 0 ile z arasında bulunup bu değer (-) ile çarpılırsa da doğru sonuca ulaşılabilir.)

flag=1: kök^z>x durumu.]

Karmaşıklık: Burada aramalarımız Bisection metodu ile olacağından yani kök'ü 0 ile z arasında her seferinde 2 ye bölerek ilerleteceğimiz için işlem sayımız $\log_2(z)$ olacaktır. Bir de sonrasındaki kök^y işlemini katarsak karmaşıklık için **Y*$\log_2$(Z)** değerine ulaşırız.

Eğer z^z<x ise: Buradaki süreç bir while döngüsünden oluşur. En başta kök=z diye tanımlanır ve bu döngü, kök^z < x olduğu sürece, kök değişkeninin üstel bir şekilde artmasıyla devam eder. Yani eğer kök^z < x ise kök değerinin üssü 1 artırılır (kök=kök^2 şeklinde) ve tekrar kontrol edilir. kök^z !< x olduğunda artık döngüden çıkılır ve bir if kontrolü ile kök^z > x mi diye kontrol edilir. Eğer bu şart sağlanmıyorsa demektir ki kök^z = x, yani gerçek kök mevcut olan son kök değeridir. Eğer kök^z > x ise de burada bisection_kok fonksiyonuna, flag=1 değeri ile gidilir. Bu nokta için bisection metoduna göre fonksiyonda alt taban olarak 'kök/z' değeri (bu değer kök^z<x şartını sağlayan kök'ün bir önceki değeridir); üst taban olarak da 'kök' değeri belirlenir (bu kontrolün içine, zaten kök değeri gerçek kökten büyük olduğu için girilmişti). Kök arayışı bisection_kok fonksiyonunda 'kök/z' ve 'kök' arasını yarılayarak sürer ve en son gerçek kök bulunduktan sonra bu değer 'kök' değişkenine eşitlenir.

Aslında bu süreç, 'çift sayıların kökleri daima çift; teklerin kökleriyse daima tek' mantığıyla kök değişkeni ikişer artırılarak da ilerlenebilirdi. Fakat bu durumda gerekenden fazla adım gidilmesi oldukça mümkündü. Çünkü örneğin: 16^(1/2) gibi x için küçük sayılar girildiğinde ilk olarak 2^2<16 şartı sağlandığı için bir sonraki adımda 2+2=4 ⇒ 4^2<16 şartı artık sağlanmayacağı için kök hemen bulunmuş olurdu. Küçük sayılar için değerlendirdiğimizde aslında iki yol arasında kayda değer bir fark olmadığı söylenebilir. Fakat büyük sayılar için, bahsedilen bu durum fazladan işlem demek olabilir.

Örneğin 15 620(1/3) için kök değerine 1. yoldan ulaşmak istersek (3'ün üslerine bakılarak): önce 3 (3^1) için sonra 9 (3^2) için sonra da 27 (3^3) için kök^3<x şartı kontrol edildiğinde 3 işlemde bisection_kok fonksiyonuna ulaştığımızı görebiliriz. (Bu fonksiyona girdikten sonrası zaten yarılayarak arama olduğu için bu fonksiyon bizi hızlı bir şekilde köke götürecektir.) 2. yoldan ulaşmak istersek: önce 3 için sonra 5 için sonra 7 için..(ve 25'e kadar) kök^3<x şartını kontrol etmemiz gerekecekti ve buradan sonrası da eğer hala gerekiyorsa bisection_kok fonksiyonuyla devam edecekti. Bu durum bize 2. yolun fazla işlem istediğini göstermektedir.

1. yolun mantığı kökün bulunduğu aralığı bir an önce bulup hızlı bir arama metodu olan bisection ile aramayı hızlandırmaktır. 2. yolun mantığı ise deneme-yanılma yolu ile yavaşça ilerleyip en son - yine gerekirse- bisection metoduna başvurmaktır.

Karmaşıklık: Buradaki süreç için belirli bir aralık mevcut değil dolayısıyla iteratif ilerleneceğinden kaç adım sonra gerçek köke ulaşabileceğimizi bilemeyiz bu yüzden bu işlem sayısını k olarak belirtirsek karmaşıklık durumumuz **k*Y** olacaktır.

x<0 durumları için -yukarıda flag=0 durumunda belirtildiği üzere- fonksiyonun içerisindeki kök^z'nin x'e eşitlik kontrolleri x'in mutlak değerleriyle yapılarak en son çıkan kök (-) ile çarpılır. Bunun mantığı bir sayının pozitif ve negatif değerlerinin tek köklerinin, birbirilerinin toplama işlemine göre terslerinin olmasıdır.

Üssün (-) olma durumu sonucu çarpma işlemine göre tersine çevirir. Bunun için başta flag_z adında bir değişken kullandım. Eğer z negatif ise flag_z=1, z=-z olarak kontrollere devam edilecek ki negatiflik üs alma işlemi gibi süreçleri etkilemesin.

Kök^y değeri de bulunduktan sonra üssün (-) olup olmadığına flag_z değişkeni ile bakılır: flag_z=1 ise z tekrar (-) ile çarpılır ve net üs işareti için y ve z çarpılır. Eğer bu değer (-) ise sonuç=1/sonuç şeklinde belirlenir; (+) ise sonuç direkt yazdırılır.

Kontrol ya da işlem gerektirmeyen durumlar için en başta bazı kontroller yaptım:

z=0: paydada 0 olamaz, işlem yapılamaz.

y=0:
x=0: 0^0 belirsizdir, işlem yapılamaz.
x!=0: her sayının 0. üssü 1'dir. İşlem yapmaksızın sonuç direkt 1'dir.

x=0: 0'ın her kuvveti 0'dır. İşlem yapmaksızın sonuç direkt 0'dır.

z=1: sonuc direkt x^y'dir.

x=1: 1'in her kuvveti 1'dir. İşlem yapmaksızın sonuç direkt 1'dir.

x<0 ve z çift: negatif bir sayının çift kökü olamaz.

ALTERNATİF OLABİLECEK BİR ÇÖZÜM YOLU

Şöyle bir mantığı daha kullanmak istedim: a^(1/b) ifadesini b=log$_{kok}$(a) şeklinde de yazabiliriz. Bu eşitlikten ln(kok)=ln(a)/b eşitliğini buluruz. Eğer ln(a) değerini bulup bu sonucu b değerine bölersek ln(kok) değerini bulmuş oluruz.

ln(a) değeri için: $\int$(1/x)dx = lnx eşitliğinden yola çıkarak 1/x fonksiyonunun x=1 ve x=a değerleri arasında kalan alanın bize lna değerini verdiğini söyleyebiliriz. Bu sonucu e'nin üssü şeklinde yazdığımızda da kok değerini bulmuş oluruz.

Bu integral sonucunu bulabilmek için Δx=0.000001 değeri ile Trapez yöntemini kullandım. Fakat şunu fark ettim ki gerçek değere ulaşabilmek adına bu değeri çok küçük seçtiğimiz için döngüdeki işlem sayısı oldukça fazla oluyor. Ayriyeten kodun devamını getiremedim çünkü en son yapacağımız işlem olan e'nin üssünü alma işlemindeki üs virgüllü çıkabileceğinden ortaya yeni bir x^(y/z) işlemi çıkıyor. Bu yüzden bu alternatif olabilecek çözüm yolunu gerçekleyemedim.

NOT: Araştırırken, bize direkt sonucu verebilecek fonksiyonlara rastlasam da çözümleri herhangi bir fonksiyon kullanmadan kendim üretmeye çalıştım hocam.

EKRAN ÇIKTILARI:

C:\Users\İahin Ailesi\Desktop\FEYZA\devc\xyz_alg.exe

```
x: 8
y: 2
z: 3
sonuc: 4.000000

Yeni islem icin 1:
Sonlandirmak icin 0: 1


x: 35
y: 5
z: 2
sonuc: 7247.195801

Yeni islem icin 1:
Sonlandirmak icin 0: 1


x: 35
y: 2
z: 5
sonuc: 4.145980

Yeni islem icin 1:
Sonlandirmak icin 0: 1


x: 5
y: 1
z: 2
sonuc: 2.236068

Yeni islem icin 1:
Sonlandirmak icin 0: 1
```

C:\Users\İahin Ailesi\Desktop\FEYZA\devc\xyz_alg.exe

```
x: 5
y: -1
z: 2
sonuc: 0.447214

Yeni islem icin 1:
Sonlandirmak icin 0: 1


x: 5
y: 1
z: -2
sonuc: 0.447214

Yeni islem icin 1:
Sonlandirmak icin 0: 1


x: 5
y: -1
z: -2
sonuc: 2.236068

Yeni islem icin 1:
Sonlandirmak icin 0: 1


x: 200
y: 3
z: 2
sonuc: 2828.427246

Yeni islem icin 1:
Sonlandirmak icin 0: 1
```

C:\Users\İahin Ailesi\Desktop\FEYZA\devc\xyz_alg.exe

```
x: 200
y: -3
z: 2
sonuc: 0.000354

Yeni islem icin 1:
Sonlandirmak icin 0: 1


x: -200
y: 1
z: 4
Negatif bir sayinin cift koku olamaz.

Yeni islem icin 1:
Sonlandirmak icin 0: 1


x: -200
y: 1
z: 3
sonuc: -5.848036

Yeni islem icin 1:
Sonlandirmak icin 0: 1


x: -200
y: 1
z: -3
sonuc: -0.170998

Yeni islem icin 1:
Sonlandirmak icin 0: 1
```

C:\Users\İahin Ailesi\Desktop\FEYZA\devc\xyz_alg.exe

```
x: -55
y: 1
z: 7
sonuc: -1.772651

Yeni islem icin 1:
Sonlandirmak icin 0: 1


x: 0
y: 0
z: 0
Islem yapilamaz!

Yeni islem icin 1:
Sonlandirmak icin 0: 1


x: 0
y: 1
z: 0
Islem yapilamaz!

Yeni islem icin 1:
Sonlandirmak icin 0: 1


x: 5
y: 0
z: 0
Islem yapilamaz!

Yeni islem icin 1:
Sonlandirmak icin 0: 1
```

C:\Users\İahin Ailesi\Desktop\FEYZA\devc\xyz_alg.exe

```
x: 1
y: 0
z: 3
sonuc: 1

Yeni islem icin 1:
Sonlandirmak icin 0: 1


x: 0
y: 2
z: 3
sonuc: 0

Yeni islem icin 1:
Sonlandirmak icin 0: 1


x: 0
y: -2
z: 3
sonuc: 0

Yeni islem icin 1:
Sonlandirmak icin 0: 0

--------------------------------
Process exited after 224.6 seconds with return value 0
Press any key to continue . . .
```