Özyineleme

TODO: tema resmi

$\longrightarrow$ GNU: GNU Not Unix

3.1 Hedefler

- → çözümü, özyinelemeyle basit, aksi takdirde zorlu olan karmaşık problemleri anlamak
- → programların özyineli olarak nasıl formülüze edileceğini anlamak
- → Özyinelemenin üç yasası: anlamak ve uygulamak
- → özyinelemenin yineleme olarak kullanımı
- → bir problemin özyineli formülasyonunu gerçeklemek
- → özyinelemenini bilgisayar sistemiyle gerçeklenmesini sağlamak

3.2 Özyineleme Nedir?

Özyineleme kolayca çözülebilecek yeterince küçük problemlere ulaşıncaya değin, problemi gitgide küçülen altproblemlere bölerek problem çözme yöntemidir.

- → işlevin kendisini çağırması
- → işlevin kendisini çağırdığını istemciye (kullanıcıya) hissettirmeden,
- → probleme hoş/basit/şık bir çözüm sunmak

3.2.1 Sayı Listesinin Toplamını Hesaplamak

- → özyinelemesiz de çözülebilecek basit bir problem
- → [1,3,5,7,9] sayı listesinin toplamını hesaplamak

İlerlemeli (_iterative_) çözüm

→ gerçekleme

```python
def listsum(l):
    sum = 0
    for i in l:
        sum = sum + i
    return sum
```

while – for olmasaydı ne yapardık?

→ döngü imkanınız olmasa ne yapardınız?

→ matematikçi olarak iki parametreli toplama işlevini çağırmakla başlardınız

$$(((1 + 3) + 5) + 7) + 9)$$

veya

$$(1 + (3 + (5 + (7 + 9))))$$

→ herhangi bir özel yapıya gereksinim duymadan çözülebilir,

```
toplam = (1 + (3 + (5 + (7 + 9) ) ) )
toplam = (1 + (3 + (5 + 16 ) ) )
toplam = (1 + (3 + 21 ) )
toplam = (1 + 24 )
toplam = 25
```

özyinelemeye ilk adım

→ Bunu Python da nasıl gerçekleştiririz?

```
listeToplami(sayiListesi) = ilkElemani(sayiListesi) +
                   listeToplami(geriyeKalani(sayiListesi))
```

→ burada listenin ilk elemanı: sayiListesi[0],

→ listenin geriye kalanı: sayiListesi[1:]

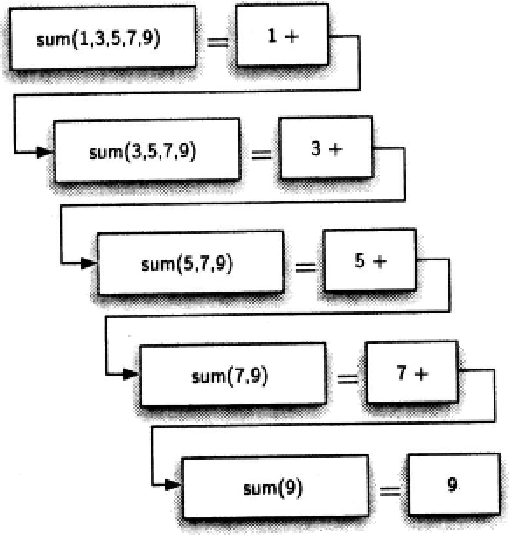
özyinelemeye ilk adım - gerçekleme

- gerçekleme

```
1 def listsum(l):
2     if len(l) == 1:
3         return l[0]
4     else:
5         return l[0] + listsum(l[1:])
```

- satır 2: durma koşulu
- satır 5: işlev kendisini çağırıyor

özyineli çağrı serisi



özyineli çağrı serisi - dönüş değerleri

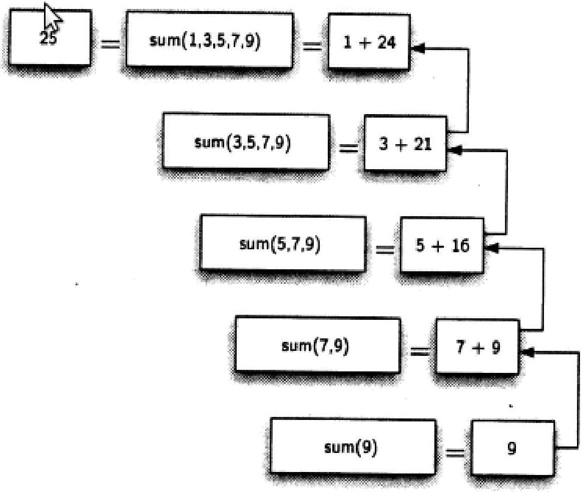


Figure 3.2: Series of Recursive Returns from Adding a List of Numbers

3.2.2 Özyinelemenin Üç Yasası

1. baz durumuna sahip olmalıdır
2. durumunu değiştirerek baz durumuna doğru hareket
3. kendini çağırma

örnek

```
1 def listsum(l):
2     if len(l) == 1:
3         return l[0]
4     else:
5         return l[0] + listsum(l[1:])
```

- → satır 2–3: birinci yasa
- → satır 5: ikinci ve üçüncü
- → baz durum: dizinin bir elemanlı olduğu durum

3.2.3 tamsayıyı herhangi bir tabanlı dizgiye (katar) çevirme

→ ör. 10 tamsayısını, ondalık olarak 10, ikilik olarak 1010 dizgisine

→ daha önce Bölüm 2.3.6'da özyinesiz çözmüştük

algoritma

1. orijinal sayıyı tek haneli sayıya dönüştür
2. _lookup_ ile tek haneyi katara çevir
3. tek haneli dizgileri sonuç dizgisinde birleştir

baz durum

→ baz duruma nasıl yakınsayacağız?

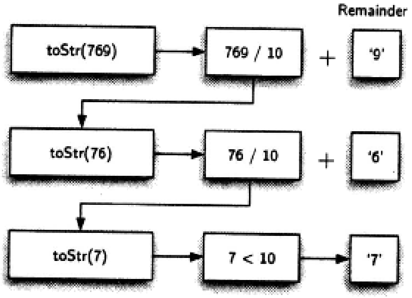


Figure 3.3: Converting an Integer to a String in Base 10

gerçekleme

→ gerçekleme

```
convertString = "0123456789ABCDEF"

def toStr(n,base):
    if n < base:
        return convertString[n]
    else:
        return toStr(n / base,base) + convertString[n%base]
```

ikilik tabana dönüşüm

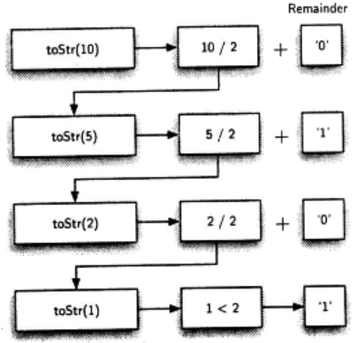


Figure 3.4: Converting the Number 10 to its Base 2 String Representation

yorum

→ sonuç beklenenin (1010) aksine ters sırada (0101)

3.3 Yığıt Frameleri: özyineli gerçekleme

- özyinelemeyle toStr ile elde edilen verileri ardı ardına eklemek yerine,
- yığıta ittiğinizi düşünün

```python
convertString = "0123456789ABCDEF"
rStack = Stack()

def toStr(n,base):
    if n < base:
        rStack.push(convertString[n])
    else:
        rStack.push(convertString[n%base])
        toStr(n / base,base)
```

çevrim sırasında yığıtta dizginin yerleşimi

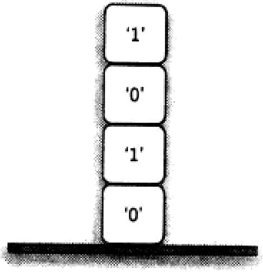


Figure 3.5: Strings Placed on the Stack During Conversion

toStr(10,2) ile üretilen çağrı yığıtı

→ şimdi sıra doğru

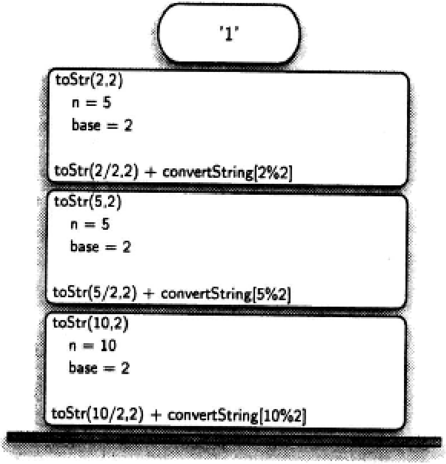


Figure 3.6: Call Stack Generated from `toStr(10,2)`

yığıt frame'i

→ işlevin yerel değişkenlerini işlemek için kullanılır

→ işlevden dönerken, dönüş değeri çağıranın erişebilmesini mümkün kılmak için yığıtın tepesinde bırakılır

→ yığıt frame'i, işlevde kullanılan değişkenlerin erimini (scope) verir.

→ böylelikle aynı işlevin üst üste çağrılmasında (ve hatta kendi kendisini çağırmasında),

→ işlevin değişkenleri yeni erimleriyle (ki yeni çağrılı işleve yerel olarak) oluşturulur

3.4 Karmaşık Özyineleme Problemleri

→ TODO: tema resmi

3.4.1 Hano Kulesi

- → 1883'de Edouard Lucas, Fransız matematikçi
- → üç kule,

İki kısıt,

1. bir anda bir disk hareketi
2. küçüğün üzerine büyüğü gelemez

Oyunda Amaç

- → bir kuleden diğerine diskleri taşımak
- → 64 diskli durumda gerekli hareket sayısı: 2^64 - 1
- → 1 hareket/sn ise 584.942.417.355 yıl.

örnek disk yerleşimi

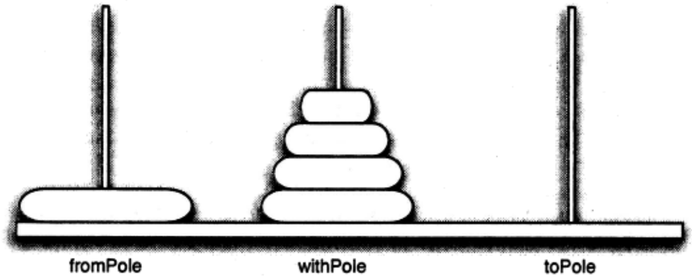


Figure 3.7: An Example Arrangement of Disks for the Tower of Hanoi

çözüme doğru

- → özyineli nasıl çözeriz?
- → baz durum nedir?
- → beş diskimiz var, 1i kule1'de, 4ü kule2'de
- → kule1'dekini kule3'e taşı, ardından kule2'deki 4 diski kule3'e taşı, fakat nasıl?
- → 4 yükseklikli kuleden taşımayı nasıl yapacağımızı bilmiyoruz
- → eğer 3 yükseklikli olanını bilseydik, 3 diski taşıdıktan sonra 4.cüsünü kule3'e alabilirdik
- → Fakat 3 yükseklikli kuleyi nasıl taşıyacağımızı bilmiyoruz

baz durum

→ tek bir diski kule3'e taşımak **kolay** işte baz durum da budur

taşıma işi

height adet diski başlangıç kutbundan (fromPole), hedef kutba (toPole), ara kutbu kullanarak (withPole) taşıma işi,

1. (height-1)'lik kuleyi hedef kutbu kullanarak ara kutba taşı
2. Geriye kalan 1 adet diski hedef kutba taşı (baz durum)
3. (height-1) adet diski başlangıç kutbundan (fromPole), ... (özyineli çağrı)

gerçekleme

→ gerçekleme

```
def moveTower(height,fromPole, toPole, withPole):
    if height >= 1:
        moveTower(height-1,fromPole,withPole,toPole)
        moveDisk(fromPole,toPole)
        moveTower(height-1,withPole,toPole,fromPole)
```

baz durum

→ bu kodda baz durum (moveDisk)) için ne yapılıyor? (hiçbir şey)

```python
def moveDisk(fp,tp):
    print "moving disk from %d to %d\n" % (fp,tp)
```

Ödev: sıra sizde

→ diskler izlenmiyor, hangi kulede hangi disk var?

→ sadece ekrana dökülüyor

→ nasıl bir veri yapısıyla durumu izlenebilir?

→ ipucu: her bir kutub için bir yığıt seçimi

3.4.2 Sierpinski Üçgeni

- demo
- Fraktal parçalanmış ya da kırılmış anlamına gelen Latince fractuuss kelimesinden gelmiştir.
- 1975, Polon matematikçi Benoit Mandelbrot
- Kendi kendini tekrar eden ama sonsuza kadar küçülen şekilleri, kendine benzer bir cisimde cismi oluşturan parçalar ya da bileşenler cismin bütününü inceler.
- Düzensiz ayrıntılar ya da desenler giderek küçülen ölçeklerde yinelenir ve tümüyle soyut nesnelerde sonsuza kadar sürebilir; tam tersi de her parçanın her bir parçası büyütüldüğünde, gene cismin bütününe benzemesi olayıdır.
- Doğada bir çok örneği vardır: bulut, karnıbahar, sahil şeridi vs.

Fractal özellikleri

A fractal often has the following features:[3]

- → It has a fine structure at arbitrarily small scales.
- → It is too irregular to be easily described in traditional Euclidean geometric language.
- → It is self-similar (at least approximately or stochastically).
- → It has a Hausdorff dimension which is greater than its topological dimension (although this requirement is not met by space-filling curves such as the Hilbert curve).[4]
- → It has a simple and recursive definition.

Sierpinski üçgeni

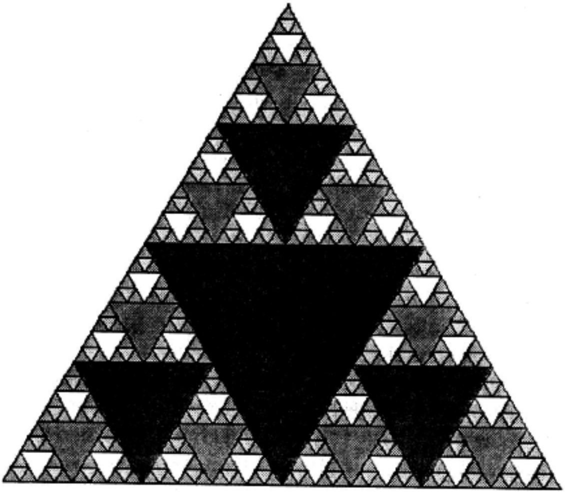


Figure 3.8: The Sierpinski Triangle

Algoritma

En basit fraktal Sierpinski üçgenidir,

1. Büyük bir üçgen çiz
2. bu üçgeni orta noktalarını kullanarak dört yeni üçgene böl
3. ortadakini göz ardı et
4. yukarıdaki üç adımı (1-2-3) üç üçgen için tekrarla (özyineleme)
5. bu işleme sonsuza kadar devam et

sorular

- → Baz durum nedir?
- → fraktalı kaç kez parçalara ayıracağız?
- → fraktalın derecesi
- → özyineli her bir çağrıda değeri bir azalır, sıfır olunca biter

gerçekleme

```python
def sierpinskiT(points,level,win):
    colormap = ['blue','red','green','white',
                'yellow','violet','orange']
    p = Polygon(points)
    p.setFill(colormap[level])
    p.draw(win)
    if level > 0:
        sierpinskiT([points[0],
                getMid(points[0],points[1]),
                getMid(points[0],points[2])],level-1,win)
        sierpinskiT([points[1],
                getMid(points[0],points[1]),
                getMid(points[1],points[2])],level-1,win)
        sierpinskiT([points[2],
                getMid(points[2],points[1]),
                getMid(points[0],points[2])],level-1,win)
```

gerçekleme

gerçekleme (devam)

```
def getMid(p1,p2):
    return Point( ((p1.getX()+p2.getX()) / 2.0),
                  ((p1.getY()+p2.getY()) / 2.0) )

if __name__ == '__main__':
    win = GraphWin('st',500,500)
    win.setCoords(20,-10,80,50)
    myPoints = [Point(25,0),Point(50,43.3),Point(75,0)]
    sierpinskiT(myPoints,6,win)
```

açıklama

- → satır 4-5-6: ana üçgen çizilir
- → satır 8...16: alt üçgenler çizilir

Sierpinski üçgenini oluşturma

en uçtaki boş olan bittiğinde bir üste çık, bir diğer altüçgeni çiz sonra diğerini

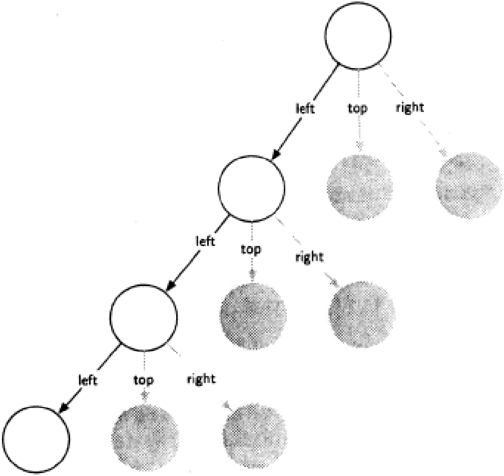


Figure 3.9: Building a Sierpinski Triangle

3.4.3 Şifreleme ve Modül Aritmetiği

Şifreleme istenmeyen kişilerin bilgilerinize erişimini kısıtlamak için kodlama ve kod çözme

```python
def encrypt(m):
    s = 'abcdefghijklmnopqrstuvwxyz'
    n = ''
    for i in m:
        j = (s.find(i)+13)%26
        n = n + s[j]
    return n
```

Caesar Cipher

- → şifre biliminin ilk ürünlerinden
- → mod aritmetiği
- → Caesar Cipher veya rot13: geleneksel sıradan 13 ötede
- → alfabenin sonuna ulaşınca başına sar: mod işleci
- → 26 harf ve rot13 olduğundan simetrik yapı: aynı işlevle kodla ve çöz
- → m= "uryybjbeyg" ise n= "helloworld"

Asimetrikleştirme

13 dışındaki döndürme asimetrikleştirir

```
def decrypt(m,k):
    s = 'abcdefghijklmnopqrstuvwxyz'
    n = ''
    for i in m:
        j = (s.find(i)26-k)%26
        n = n + s[j]
    return n
```

açıklama

- ⟶ ne kadar döneceğini işleve göndeririz
- ⟶ buna key'de denir
- ⟶ SIRA SİZDE: Şifre çözmeyi de siz yapın
- ⟶ key bilinmeden çözülemez!
- ⟶ daha güvenilir olanları da vardır: RSA.

3.4.3.1 Mod Aritmetiği Teoremi

→ n ile bölündüğünde aynı kalanı veren a ve b sayıları

→ **congruent modulo n** veya

→ kısaca a == b (mod n)

teoremler

1. $a == b \pmod{n}$ o zaman $a + c == b + c \pmod{n}$ (her bir c için)
2. $a == b \pmod{n}$ o zaman $ac == bc \pmod{n}$ (her bir c için)
2. $a == b \pmod{n}$ o zaman a^p == b^p (mod n) (her bir p için)

3.4.3.2 Modular Exponentiation

- → 3^1.254.906 sayısının son hanesi nedir?
- → bu sadece hesapsal olarak yoğun bir işlem değil
- → aynı zamanda Python'un maksimum 598.743 sınırına toslamadır

algoritma

1. x^n'i etkin bir şekilde nasıl hesaplarız
2. x^n (mod p)'yi, x^n'i hesaplamadan p'ye bölümünden kalanını hesaplayabilir miyiz?

üstel sayıyı hesaplamadan kalanını bulma

1. result= 1 ilklendir
2. n kez tekrarla
   a. result *= x
   b. result %= p

→ burada x sayısının n.nci üstelinin p'ye bölümünden kalan hesaplanıyor

iyileştirme

→ iyileştirme

$$x^n = \begin{cases} (x \cdot x)^{n/2} & \text{if n is even} \\ (x \cdot x^{n-1}) = x \cdot (x \cdot x)^{\lfloor n/2 \rfloor} & \text{if n is odd} \end{cases}$$

→ burada aşağı yuvarla işlevidir.

→ n çiftse, n/2=aşağıyuvarla(n/2) olur.

gerçekleme

→ baz durum x^0 = 1

```
def modexp(x,n,p):
    if n == 0:
        return 1
    t = (x*x)%p
    tmp = modexp(t,n/2,p)
    if n%2 != 0:
        tmp = (tmp * x) % p
    return tmp
```

→ n çiftse durumunu idare et,

→ tek olduğu durumda sadece tmp = (tmp * x) %p'yi hesapla

3.4.3.3 En Büyük Ortak Bölen ve Multiplicative Inverse

Multiplicative Inverse x mod m'nin çarpmaya göre tersi, çarpılıp modülüne bakıldığında 1 değeri veren a'dır. yani ax== 1 (mod m)

→ örn. x=3, m=7 ve a=5 olsun

$$ax = 5 \,.\, 3 = 15$$

$$ax \bmod m = 15 \bmod 7 = 1$$

→ bu durumda 5 . 3== 1 (mod 7).

→ dolayısıyla 5, 3 mod 7'nin çarpmaya göre tersidir

sorular

1. önceki örnekteki 5 değerini nasıl seçtik?
2. 3 mod 7'nin 5 dışında başka çarpmaya göre tersi var mı?
3. tüm sayılar için bir çarpmaya göre tersi değer var mıdır?

soru 2: çarpmaya göre tersi kaç tane var?

→ gerçekleme

```
>>> for i in range(1,40):
...     if (3*i)%7 == 1:
...             print i
...
5
12
19
26
33
>>>
```

yakın bakış

- birden fazla sayıda tersi var: 5, 12, ...
- ortak özellikleri nedir? n*7 - 2 (3 mod 7'in tersi hesaplanıyordu!)
- $x = 4$, $m = 8$ yaparsak, çıktı üretmez yani tersi olan bir sayı yok!
- bunu baştan anlayabilir miyiz?

Göreceli Asallık

- $\longrightarrow$ çarpmaya göre tersi olup – olmadığını baştan anlamanın yolu
- $\longrightarrow$ x sayısının çarpmaya göre tersinin olabilmesinin koşulu
- $\longrightarrow$ m ve x'in **göreceli asal** olması gerekir!
- $\longrightarrow$ (m, x)'nin göreceli asallığı, obeb(m, x) = 1
- $\longrightarrow$ OBEB hesabı (veya GCD) "Euclid Algoritması"

Euclid algoritması

gcd(a, b) kaba algoritma:

- $\longrightarrow$ tekrarlı olarak, $a < b$ oluncaya değin a'dan b'yi çıkart.
- $\longrightarrow$ $a < b$ olunca rolleri değiştir
- $\longrightarrow$ gcd(a, 0) ise çıkış a'dır (yani gcd çıkışı)

gerçekleme – v1

→ gerçekleme – v1

```python
def gcd(a,b):
    if b == 0:
        return a
    elif a < b:
        return gcd(b,a)
    else:
        return gcd(a-b,b)
```

açıklama

- → b = 0 baz durum
- → a >> b ise program etkin çalışmaz.
- → mod aritmetiğinden yararlanabiliriz
- → son çıkartmanın sonucu (a - b < b iken), gerçekte a % b'dir

gerçekleme – v2

→ gerçekleme – v2

```python
def gcd(a,b):
    if b == 0:
        return a
    else:
        return gcd(b, a % b)
```

açıklama

→ x ve m'nin çarpmaya göre tersi olup-olmadığını bilmenin bir yolu var! (gcd)

→ daha etkin bir şekilde yazabiliriz

→ herhangi bir (x, y) çifti için

  → *hem* gcd(x, y)

  → *hem de* d= gcd(x, y)= a*x + b*y*'deki* (a, b) *çifti hesaplanabilir.*

→ Örn. 1= gcd(3, 7)= -2*3 + 1*7, yani a=-2, b=1

iyileştirme

- → 1= gcd(m, x)= a*m + b*x ve bx= 1 mod m idi,
- → böylece b, x mod m'nin çarpmaya göre tersidir
- → ters bulma problemini, d= gcd(x, y)= a*x + b*y'deki (a, b)'yi bulmaya indirgedik
- → x>=y olmak üzere, ext_gcd() işlevi, (d, a, b) tuple'ı döndürecek

gerçekleme – v3

→ gerçekleme – v3

```
def ext_gcd(x,y):
    if y == 0:
        return(x,1,0)
    else:
        (d,a,b) = ext_gcd(y, x%y)
        return(d,b,a-(x/y)*b)
```

ext_gcd'nin çağro ağacı

→ x=25, y=9

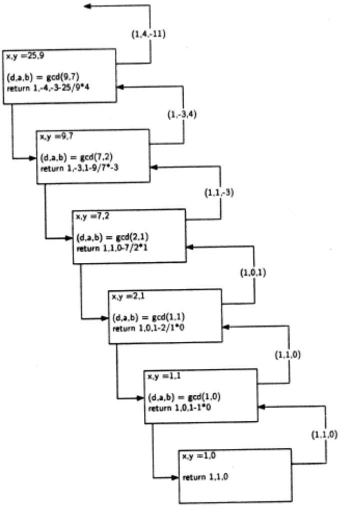


Figure 3.10: Call Tree for Extended GCD Algorithm

açıklama

- y=0 baz durumunda d=x döndürülür (=orijinal Euclid algoritması)
- ek iki değer döndürüyoruz: a=1, b=0
- d= a*x + b*y
- y > 0 ise özyineli olarak (d,a,b) değerlerini
- d= gcd(y, x mod y) ve d= a *y + b (x mod y) şartını sağlayacak şekilde hesapla
- orijinal algoritmada d= gcd(x, y) tamam
- (a, b) ne olacak? (ikisi de tamsayı)

çıkarım

→ (a, b)'yi A, B olarak adlandıralım

$$\mathbf{d = a^* y + b\ (x\ mod\ y)} = a^* y + b\ (x - AşYuv(x/y)^* y) = b^* x + (a - AşYuv(x/y)^* y)^* y$$

$$X\ mod\ y = x - AşYuv(x/y)$$

buradan A= b ve B= a - AşYuv(x/y)*b

3.4.3.4 RSA Algoritması

- tüm araçlar hazır
- açık-anahtar şifreleme algoritmaları arasında en anlaşılır olanı
- Whitfield, Diffie ve Martin tarafından, ve bağımsız olarak Ralph Merkle

Açık anahtar şifreleme

- → anahtar çifti: şifreleme – çözme anahtarı
- → biriyle şifrele, diğeriyle aç
- → özel anahtar – açık anahtar
- → biriyle şifrelenen ancak diğeriyle açılabilir!

RSA

- → RSA gücünü devasa sayılardan alır
- → özel - açık anahtarlar büyük (100-200 hane) asal sayılardan türetilir

algoritma

anahtar çiftini üretmek için

1. iki büyük p ve q asal sayı seç
2. $n = p \times q$ çarpımını hesapla
3. $(p - 1)x(q - 1)$ ile aralarında asal olan rastgele olarak e'yi seç. Yani $gcd(e, (p - 1)x(q - 1)) = 1$
4. d – deşifre anahtarı basit bir şekilde e mod $(p - 1)x(q - 1)$'in modülar tersidir. Bu amaçla Euclid'in genişletilmiş utyarlaması kullanılır

anahtar

- → açık anahtar: e ve n
- → özel anahtar: d
- → n, e ve d bir kez hesaplanınca p ve q'ya gerek kalmaz, gizli kalmalıdır

şifrele - çöz

- $\longrightarrow$ m-mesajını şifrelemek için c= m^e (mod n)
- $\longrightarrow$ c'nin çözülmesi için: m= c^d (mod n)

çıkarım

- $\rightarrow$ d, e (mod n)'nin çarpmaya göre tersi olduğuna göre
- $\rightarrow$ c^d= (m^e)^d (mod n)
- $\rightarrow$ = m^ed (mod n)
- $\rightarrow$ = m^1 (mod n)
- $\rightarrow$ = m (mod n)

ör

- mesaj = “helloworld”
- ASCII karşılıkları
- m = 104 101 108 108 ... 108 100
- RSA şifreleme mekanizması m – mesajını küçük junklara böler

1. *performans,* 1k’*lık metin* --> 2000-3000 *hane.* d’*nin (10 hane) üstünü almakla çok büyük sayılar ortaya çıkar*
2. m <= n *kısıtlamasıyla mesajın* mod n’*de eşsiz bir temsilini mümkün kılıyoruz. İkil veriyle* n’*den daha kçük en büyük ikinin kuvveti seçilir.*

m'nin chunkları

→ ör. p ve q, 5563 ve 8191 seçilirse, n= 5563x8191= 45.566.533 (8 hane), chunk boyutu 7 (=8-1) (n'deki hane sayısının bir eksiği).

→ m= 104 101 108 108 ... 108 100

$$m1 = 1041011 \quad m2 = 0810811 \ \dots \ m5 = 8100$$

e'nin seçilmesi

→ şimdi e değerini seçelim (rastgele) ve gcd algoritmasıyla (p - 1)x(q - 1)=45.552.780'e karşı test et

→ bununla aralarında asal olan e değerini arıyoruz

→ bu örnekte 1.471 iyi sonuç veriyor

d= ext_gcd(45.552.780, 1.471) = *-11.705.609= 45.552.780 - 11.705.609= 33.847.171*

şifrele

→ ilk chunk'ı şifrele

$$c = 1.041.011\hat{}1.471 \pmod{45.566.533} = 28.713.328$$

→ deşifreleyerek veriyi tekrardan elde et

$$m = 28.713.328\hat{}33.847.171 \pmod{45.566.533} = 1.041.011$$

→ tüm chunk'lar benzer biçimde şifrelenir ve çözülür

gerçekleme

→ gerçekleme

```
import random
import string

def gcd(a, b):
    if b == 0:
        return a
    else:
        return gcd(b, a % b)

def ext_gcd(x, y):
    if y == 0:
        return (x, 1, 0)
    else:
        (d, a, b) = ext_gcd(y, x%y)
        return (d, b, a-(x/y)*b)
```

gerçekleme

→ gerçekleme

```
def toChunks(m, chunkSize):
    chunk_m = []
    for ch in m:
        chunk_m.append(ord(ch).__str__())
    chunk_m_str = string.join(chunk_m, sep='')
    chunks = []
    sz = len(chunk_m_str)
    adet = (sz/chunkSize)
    for i in range(1,adet+1):
        iba = (i - 1) * chunkSize
        iso = iba + chunkSize
        chunks.append(chunk_m_str[iba:iso])
    if not (adet*chunkSize) == sz:
        iba = iso
        iso = sz
        chunks.append(chunk_m_str[iba:iso])
    return chunks

def chunksToPlain(m):
    #todo
    return True
```

gerçekleme

→ gerçekleme

```
def RSAgenKeys(p,q):
    n = p * q
    pqminus = (p-1) * (q-1)
    e = int(random.random() * n)
    while gcd(pqminus,e) != 1:
        e = int(random.random() * n)
    d,a,b = ext_gcd(pqminus,e)
    if b < 0:
        d = pqminus+b
    else:
        d = b
    return ((e,d,n))

def RSAencrypt(m,e,n):
    ndigits = len(str(n))
    chunkSize = ndigits - 1
    chunks = toChunks(m,chunkSize)
    encList = []
    for messChunk in chunks:
        print messChunk
        c = modexp(messChunk,e,n)
        encList.append(c)
    return encList

```

açıklama

→ RSAgenKeys: açık ve özel anahtar (p ve q) üretir

→ RSAencrypt: m-mesajını e ve n'yi kullanarak şifreler

→ RSAdecrypt: şifreli metni d ve n'yi kullanarak çözer

demo

```
e, d, n = RSAgenKeys(5563, 8191)
m = 'goodby girl'
c = RSAencrypt(m, e, n)
cm = RSAdecrypt(c, d, n)
```

3.5 Özet

$\longrightarrow$ ...

3.6 Anahtar Kelimeler

→ base case

3.7 Tartışma Soruları

1. aa

3.8 Programlama Exersizleri

1. aa