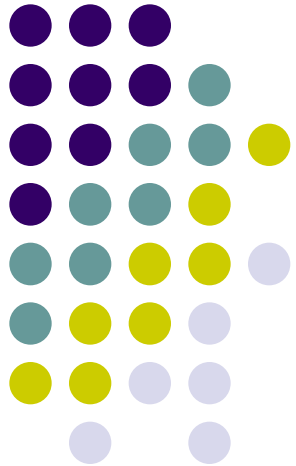
ALGORİTMALAR

BÖLÜM 2-DVM



İÇERİK

- **Recurrence Relations (Özyineleme İlişkileri)**
  - **Exact Solution (Kesin Çözüm)**
    - Forward substitution (İleriye Doğru yerine koyma)
    - Backward substitution (Geriye Doğru Yerine Koyma)
    - Diferansiyel denklemlerde kullanılan yöntemler,
  - **Asymptotic Solution (Asimptotik Çözüm)**
    - Master teoremi

Recurrence Relations

- Genellikler özyineleme bulunan fonksiyonlarda görülür

  Örnekler :

$$f(n) = f(n/2) + 1$$

$$f(0) = 1$$

$$f(n) + f(n-1) - 6f(n-2) = 2^n - 1$$

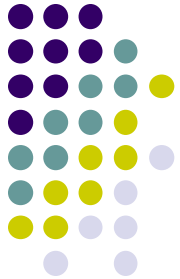
Örnekler

- Array'deki en büyük elemanı belirleme
  - İki özyineli çağırım
- Binary search

Recurrence Relations

- **Recurrence Relations(Özyineleme İlişkileri)**
  - **Exact Solution (Kesin Çözüm)**
    - **Forward substitution (İleriye Doğru yerine koym)**
    - **Backward substitution (Geriye Doğru Yerine Koyma)**
    - Diferansiyel denklemlerde kullanılan yöntemler
  - **Asymptotic Solution**
    - Master theorem

Örnek 1 – Faktöriyel Hesaplama



- Hedef:

  Herhangi bir doğal tamsayı n için **`F(n)=n!`**

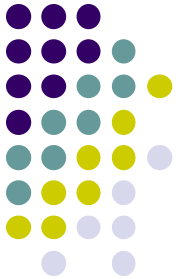
  `Faktöriyel` fonksiyonunu hesapla.

$$n! = n \times (n-1) \times \ldots \times 1 = n \times (n-1)! \quad \text{for} \quad n \geq 1 \quad \text{and} \quad 0! = 1$$

$$F(n) = F(n-1) \times n \quad \text{for} \quad n > 0$$

$$F(0) = 1$$

Faktöriyel Hesaplama



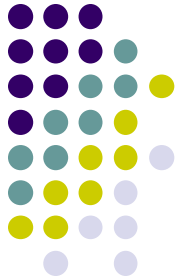
Algoritma :

```
if n=0 return 1
else return F(n-1)*n
```

Analizi:

$$M(n) = M(n-1) + 1 \quad \text{for } n > 0,$$
$$M(0) = 0.$$

Forward Substitution (İleriye Doğru yerine koyma)



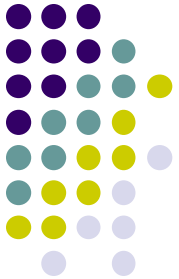
- Initial condition ile başlayın
- Çözüme doğru bir kaç terimi hesaplayın.
  - Önceden bulduğunuz terimleri kullanarak
- Bir desen bulmaya çalışın
- Kapalı formda bir formül bulmaya çalışın
- Bulduğunuz formulü doğrulayın
  - Tümevarım
  - Yerine koyma

Faktörüyel Fonksiyonu

Forward substitution:

$$M(0) = 0$$
$$M(1) = M(0) + 1 = 1$$
$$M(2) = M(1) + 1 = 2$$
$$M(3) = M(2) + 1 = 3$$
$$\vdots$$
$$M(n) = n$$

Faktöriyel Fonksiyonu



(Backward Substituiton) Geriye doğru yerine koyma ile:

$$M(n) = M(n-1) + 1$$

$$M(n) = M(n-2) + 1 + 1$$

$$M(n) = M(n-3) + 1 + 1 + 1$$

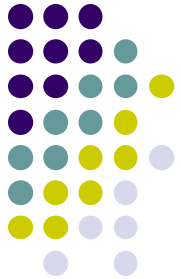
$$\vdots$$

$$M(n) = M(n-i) + i$$

for $n = i$

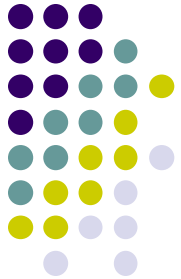
$$M(n) = M(0) + n = n$$

Backward Substitution



- Recurrence denklemi ile başlayın (f(n) ile)
- *f(n-1)* terimini açın
  - Recurrence denklemini kullanarak
- Aynı işlemi birkaç kere tekrarlayın.
  - for *f(n-2), f(n-3)* etc..
- *f(n)* denkleminin formülünün *f(n-i) cinsinden bulmaya çalışın.*
- Desenin doğruluğunu kontrol edin
  - Genellikle tümevarım ile
- n-i'yi initial condition yapan i'yi seçin ve kapalı formülü elde ediniz.

Örnek 2- Towers of Hanoi



Hedef:

n tane diski A çubuğundan C çubuğuna arada B çubuğunu kullanarak yerleştirme

Yaklaşım: (recursive)

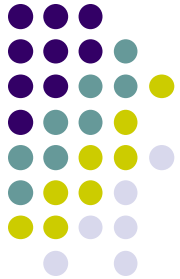
- n-1 tane diski *A* 'dan *B'ye C çubuğunu kullanarak aktarın*
- En büyük diski *A*'dan *C'ye iletin.*
- n-1 tane diski *B* çubuğundan *C'ye A çubuğunu kullanarak iletin*

Toplam hareket sayısı

$T(n) = 2T(n-1) + 1$

$T(1) = 1$

Recursive Algoritmaların Analizi

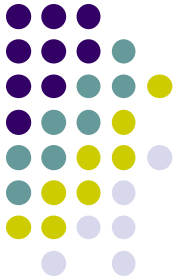


Recursive algoritmalarda Genel Plan

- Girdi boyutunu belirleyecek parametrelere karar verme.
- Algoritmanın temel işlemini (basic operation) belirle.

- Her recursive çağırmada temel işlem kaç defa çalıştırılıyor belirle.
  - Eğer çalıştırma sayısı aynı boyuttaki farklı girdiler de değişiyorsa;
    - worst-case, average-case ve best-case durumları ayrı ayrı incelenmelidir.
  - Temel işlem çağrılmasını sayan bir özyineleme ilişkisi oluşturun.
  - Özyinelemeli çağrımları ve bu çağrımlardaki girdi boyutunu belirleyin.
  - Uygun bir initial condition bulun.
- Özyinelemeyi çözün.
  - En azından girdi büyüdükçe işlem adımlarındaki büyümeyi belirleyin.

Towers of Hanoi



$$T(n) = 2T(n-1) + 1$$
$$T(1) = 1$$

Backward Substituion

$$T(n) = 2T(n-1) + 1$$
$$T(n) = 4\big(T(n-2)\big) + 2 + 1$$
$$T(n) = 4\big(2T(n-3) + 1\big) + 2 + 1$$
$$T(n) = 8T(n-3) + 4 + 2 + 1$$
$$\vdots$$
$$T(n) = 2^i T(n-i) + 2^{i-1} + 2^{i-2} + \ldots + 2^1 + 1 \quad \leftarrow \text{Doğrulama}$$

$$when \qquad i = n-1$$
$$T(n) = 2^{n-1} T(1) + 2^{n-2} + \ldots + 2^1 + 1$$
$$T(n) = 2^n - 1 = \theta(2^n)$$

Example 3: Counting #bits

ALGORITHM $BinRec(n)$

//Input: A positive decimal integer n

//Output: The number of binary digits in n's binary representation

if $n = 1$ **return** 1

else return $BinRec(\lfloor n/2 \rfloor) + 1$

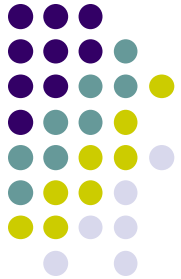
Örnek

$$T(n) = 2T(\sqrt{n}) + 1 \qquad T(2) = 0$$

$$T(n) = 2T(n^{1/2}) + 1$$
$$T(n) = 2\left(2T(n^{1/4}) + 1\right) + 1$$
$$T(n) = 4T((n^{1/4}) + 1 + 2$$
$$T(n) = 8T(n^{1/8}) + 1 + 2 + 3$$
$$\vdots$$
$$T(n) = 2^i T(n^{1/2^i}) + 2^0 + 2^1 + \ldots + 2^{i-1}$$

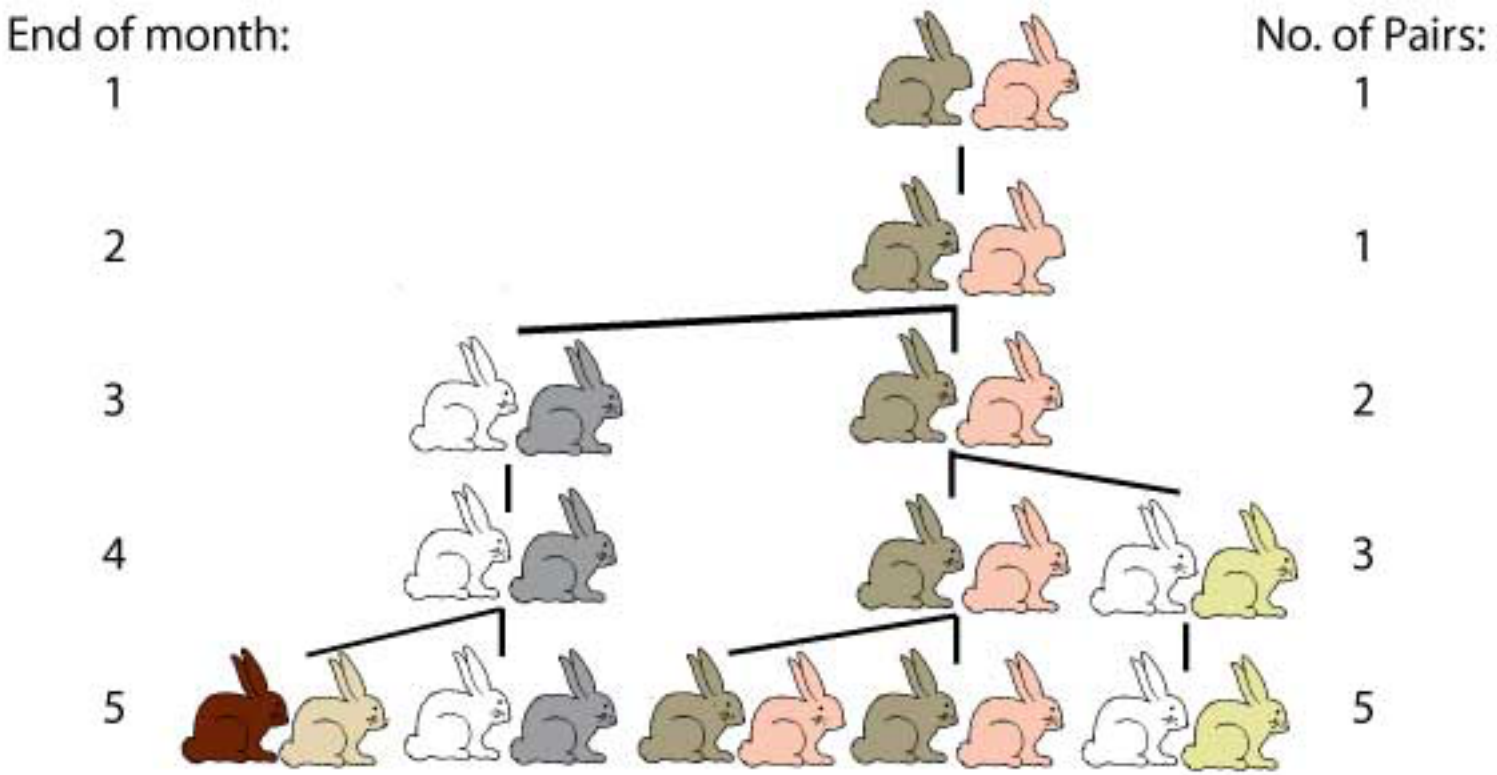
$$n^{1/2^i} = 2 \quad \Rightarrow \quad i = \log\log(n)$$
$$T(n) = 2^0 + \ldots + 2^{\log\log n - 1} = \theta(\log n)$$

ROAD MAP

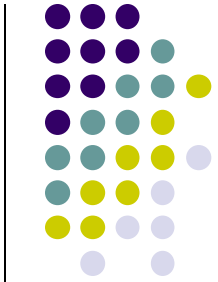


- **Recurrence Relations**
  - **Exact Solution**
    - Forward substitution
    - Backward substitution
    - Methods similar to those used in solving differential equations
  - **Asymptotic Solution**
    - Master theorem

Fibonacci numbers



Fibonacci numbers



The Fibonacci numbers:

0, 1, 1, 2, 3, 5, 8, 13, 21, …

The Fibonacci recurrence:

$F(n) = F(n-1) + F(n-2)$

$F(0) = 0$

$F(1) = 1$

Genel 2^{nd} derece lineer homojen recurrence denklemi

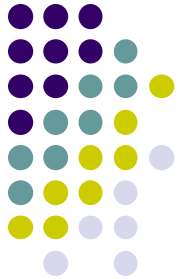
constant coefficients:

$$aX(n) + bX(n-1) + cX(n-2) = 0$$

Fibonacci numbers

$$F(n) = \frac{1}{\sqrt{5}}(\phi^n - \hat{\phi}^n),$$

$$\phi = (1 + \sqrt{5})/2 \approx 1.61803 \qquad \hat{\phi} = -1/\phi \approx -0.61803$$



ALGORITHM $F(n)$

//Computes the nth Fibonacci number recursively by using its definition
//Input: A nonnegative integer n
//Output: The nth Fibonacci number
if $n \leq 1$ **return** n
else return $F(n-1) + F(n-2)$

Fibonacci sayıları

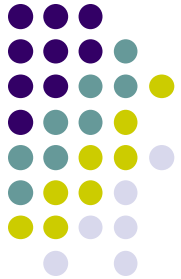
$$A(n) = A(n-1) + A(n-2) + 1 \quad \text{for } n > 1,$$
$$A(0) = 0, \qquad A(1) = 0.$$

$$[A(n) + 1] - [A(n-1) + 1] - [A(n-2) + 1] = 0$$

$$B(n) = A(n) + 1:$$

$$A(n) = B(n) - 1 = F(n+1) - 1 = \frac{1}{\sqrt{5}}(\phi^{n+1} - \hat{\phi}^{n+1}) - 1.$$

Fibonacci sayıları



ALGORITHM *Fib*(n)

//Computes the nth Fibonacci number iteratively by using its definition
//Input: A nonnegative integer n
//Output: The nth Fibonacci number
$F[0] \leftarrow 0;\ F[1] \leftarrow 1$
for $i \leftarrow 2$ **to** n **do**
    $F[i] \leftarrow F[i-1] + F[i-2]$
return $F[n]$