BM 210 Algoritma Analizi (Analysis of Algorithms)

Hazırlayan: M.Ali Akcayol
Gazi Üniversitesi
Bilgisayar Mühendisliği Bölümü

Recurrences

- Özyinelemeli (Recursive) algoritmaların çalışma sürelerinin analizi (örnek divide-and-conquer)

Recurrences

- Recursive algoritmaların çalışma süreleri recurrence ifadelerle tanımlanır
- Bir **recurrence** ifade küçük giriş değerleri için bir fonksiyon tanımlar
- divide-and-conquer algoritmalar için kullanılır

T(n)=
- **denemeyle çözülür, n=1**
- **Alt problem sayısı * T(n/alt problem boyutu)+bölme+birleşim, n>1**

- Örnek: Merge Sort

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

Recurrence ifadelerin çözümü

- Tekrarlı substitution metodu
- Recursion-trees
- Master metod

Tekrarlı Substitution

- Merge sort algoritmasının çalışma süresi (bazı *b* değerleri için $n=2^b$ olduğunu varsayalım).

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2T(n/2) + n & \text{if } n > 1 \end{cases}$$

$$\begin{aligned}
T(n) &= 2\,T(n/2) + n && \text{// substitute} \\
&= 2\,(2\,T(n/4) + n/2) + n && \text{// expand} \\
&= 2^2\,T(n/4) + 2n && \text{// substitute} \\
&= 2^2\,(2\,T(n/8) + n/4) + 2n && \text{// expand} \\
&= 2^3\,T(n/2^3) + 3n && \text{// observe} \\
&= 2^b\,T(n/2^b) + bn \\
&= 2^{lgn}\,T(n/n) + nlgn \\
&= n + n\,lgn
\end{aligned}$$

Tekrarlı Substitution Metodu

- İşlem akışı:
  - Substitute
  - Expand
  - Substitute
  - Expand
  - …
  - Sonuçlar incelenir (Observe) ve *i*-th substitution' dan sonra ifadenin nasıl devam ettiği bulunur

Tekrarlı Substitution

- Merge sort için daha kesin çalışma süresi bulma (bazı *b* değerleri için $n=2^b$ alınırsa).

$$T(n) = \begin{cases} 2 & \text{if } n = 1 \\ 2T(n/2) + 2n + 3 & \text{if } n > 1 \end{cases}$$

$$T(n) = 5n + 2nlgn - 3$$

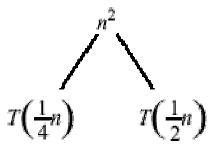
$$T(n) = \begin{cases} 2 & \text{if } n = 1 \\ 2T(n/2) + 2n + 3 & \text{if } n > 1 \end{cases}$$

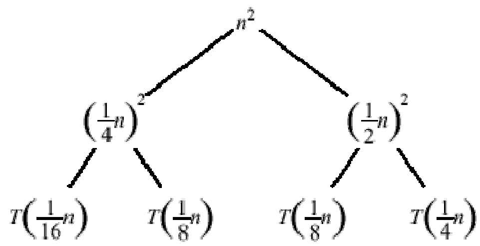
$$\begin{aligned}
T(n) &= 2\,T(n/2) + 2n + 3 \\
&= 2\,(2\,T(n/4) + n + 3) + 2n + 3 \\
&= 2^2\,T(n/4) + 4n + 2\,.\,3 + 3 \\
&= 2^2\,T(n/2^2) + 2^2 n + 2^1\,.\,3 + 2^0.3 \\
&= 2^2\,(2\,T(n/8) + n/2 + 3) + 4n + 2.3 + 3 \\
&= 2^3(T(n/2^3) + 2.3n + (2^2 + 2^1 + 2^0).3 \\
&= 2^b\,T(n/2^b) + 2.bn + 3\sum_{j=0}^{b-1} 2^j \\
&= n\,T(n/n) + 2nlgn + 3(2^b - 1) \\
&= 2n + 2nlgn + 3n - 3 \\
&= 5n + 2nlgn - 3
\end{aligned}$$

Recursion Tree

- Recursion tree görsel olarak iterasyonların çalışmasını gösterir.
  - Recurrence ifadelerde asymtotic çözüm tahmini için iyi bir yoldur

$$T(n) = T(n/4) + T(n/2) + n^2$$





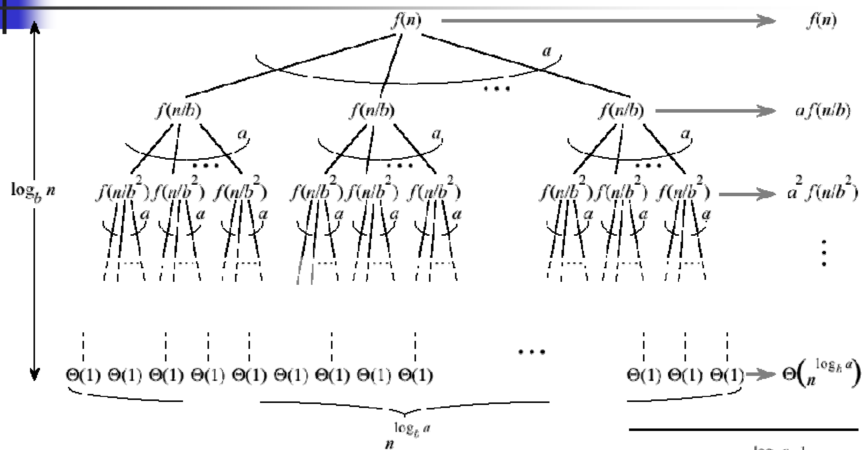
Master Metodu

- Aşağıdaki yapıdaki recurrence ifadelerin çözümü için kullanılır

$$T(n) = aT(n/b) + f(n)$$

- $a \geq 1$ ve $b > 1$, ve f pozitif.
- $T(n)$ bir algoritmanın çalışma süresidir
  - n/b boyutunda a tane alt problem recursive olarak çözülür ve herbiri $T(n/b)$ süresindedir
  - $f(n)$ problemin bölünmesi ve sonuçların birleştirilmesi için geçen süredir. Merge-sort için $T(n) = 2T(n/2) + \Theta(n)$ yazılabilir.

Master Metodu (2)



Total: $\Theta\left(n^{\log_b a}\right) + \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j)$

Problem $\log_b n$ seviyesinde a parçaya bölünür. $a^{\log_b n} = n^{\log_b a}$ adet yaprak vardır.

Master Metodu (3)

- Yaprak sayısı: $a^{\log_b n} = n^{\log_b a}$
- Recurrence ifade döngüyle çalıştırılır ve ağaç açılmış olur

$$
\begin{aligned}
T(n) &= f(n) + aT(n/b) \\
&= f(n) + af(n/b) + a^2T(n/b^2) \\
&= f(n) + af(n/b) + a^2T(n/b^2) + \ldots \\
&\quad + a^{\log_b n-1} f(n/b^{\log_b n-1}) + a^{\log_b n}T(1)
\end{aligned}
$$

Böylece

$$T(n) = \sum_{j=0}^{\log_b n-1} a^j f(n/b^j) + \Theta(n^{\log_b a})$$

- İlk terim bölme / birleştirme süresi (ağacın tüm seviyeleri için toplam)
- İkinci terim 1 büyüklüğündeki $n^{\log_b a}$ adet alt problemin çözümünü ifade eder.

Master Metodu

- Üç durum vardır: $T(n) = aT(n/b) + f(n)$
  - Çalışma süresi ağırlıklı olarak yapraklar tarafından belirlenir
  - Çalışma süresi ağacın tamamına düzgün şekilde dağılmıştır
  - Çalışma süresi ağırlıklı olarak kök tarafından belirlenir
- Recurrence ifadenin çözülmesi için ağırlıklı terim belirlenir
- Her durum için $f(n)$ ile $O(n^{\log_b a})$ karşılaştırılır

Master Metodu (Durum 1)

$T(n) = aT(n/b) + f(n)$

- $f(n) = O(n^{\log_b a - \varepsilon})$ $\varepsilon > 0$ değerleri için
  - $f(n)$ fonksiyonu $n^{\log_b a}$ fonksiyonundan daha yavaş büyür
- **Çalışma yaprak ağırlıklıdır**
  - Recursion-tree seviyelerindeki çalışma süreleri toplamı $O(n^{\log_b a})$
  - Tüm yaprakların çalışma süreleri toplamı $\Theta(n^{\log_b a})$
  - Tüm çalışma süresi yaprakların çalışma süresine eşittir $\Theta(n^{\log_b a})$

Master Metodu (Durum 2)

- $f(n) = \Theta(n^{\log_b a})$ $T(n) = aT(n/b) + f(n)$
  - $f(n)$ ve $n^{\log_b a}$ asymptotic olarak aynıdır
- **Çalışma süresi ağacın tamamına eşit olarak dağılmıştır**

$$T(n) = \Theta(n^{\log_b a} \lg n)$$

Master Metodu (Durum 3)

- $f(n) = \Omega(n^{\log_b a + \varepsilon})$ $\varepsilon > 0$ değerleri için
  - 1. durumun tersidir
  - *f*(*n*) fonksiyonu $n^{\log_b a}$ fonksiyonundan daha hızlı büyür
  - Ayrıca aşağıdaki şartın doğruluğu gerekir
    $\exists c < 1 \text{ and } n_0 > 0 \text{ such that } af(n/b) \le cf(n) \ \forall n > n_0$
- **Çalışma süresi kök ağırlıklıdır**

$$T(n) = \Theta(f(n))$$

Master Metodu Özet

- Verilen bir recurrence ifade için $T(n) = aT(n/b) + f(n)$

1. $f(n) = O\left(n^{\log_b a - \varepsilon}\right)$

   $\Rightarrow T(n) = \Theta\left(n^{\log_b a}\right)$

2. $f(n) = \Theta\left(n^{\log_b a}\right)$

   $\Rightarrow T(n) = \Theta\left(n^{\log_b a} \lg n\right)$

3. $f(n) = \Omega\left(n^{\log_b a + \varepsilon}\right)$ and $af(n/b) \le cf(n)$, for some $c < 1, n > n_0$

   $\Rightarrow T(n) = \Theta\left(f(n)\right)$

İşlem sırası

- Önce *a*, *b*, ve *f*(*n*) verilen recurrence ifadeden elde edilir
- $n^{\log_b a}$ belirlenir
- *f*(*n*) ve $n^{\log_b a}$ fonksiyonları asimtotik olarak karşılaştırılır
- Hangi durum olduğu belirlenir ve ilgili kural uygulanır

Örnek (Merge Sort)

$$T(n) = 2T(n/2) + \Theta(n)$$

$$a = 2,\ b = 2;\ n^{\log_b a} = n^{\log_2 2} = n = \Theta(n)$$

$$f(n) = \Theta(n)$$

$$\Rightarrow \qquad 2:\ T(n) = \Theta\left(n^{\log_b a} \lg n\right) = \Theta\left(n \lg n\right)$$

Örnekler

```
Binary-search(A, p, r, s):
   q←(p+r)/2
   if A[q]=s then return q
   else if A[q]>s then
        Binary-search(A, p, q-1, s)
   else Binary-search(A, q+1, r, s)
```

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

$$a = 1, b = 2;\ n^{\log_2 1} = 1$$

$$f(n) = 1,\ f(n) = \Theta(1)$$

$$\Rightarrow \sim \qquad 2:\ T(n) = \Theta(\lg n)$$

$$T(n) = \Theta(n^{\log_b a} \lg n)$$

$$T(n) = 9T(n/3) + n$$

$$a = 9, b = 3; \qquad \Theta(n^{\log_b a})$$

$$f(n) = n,\ f(n) = O(n^{\log_3 9 - \varepsilon}) \qquad \varepsilon = 1$$

$$\Rightarrow \sim \qquad 1:\ T(n) = \Theta\left(n^2\right)$$

Örnekler

$T(n) = 3T(n/4) + n\lg n$

$a = 3, b = 4;\ n^{\log_4 3} = n^{0.793}$

$f(n) = n\lg n,\ f(n) = \Omega(n^{\log_4 3+\varepsilon}) \qquad \varepsilon \approx 0.2$

$\Rightarrow$ 3:

$af(n/b) = 3(n/4)\lg(n/4) \leq (3/4)n\lg n = cf(n) \qquad c = 3/4$

$T(n) = \Theta(n\lg n)$

Haftalık Ödev

- Quick sort algoritmasının analizini yapınız. Ödev çıktı alınarak ve bir kapak sayfasıyla birlikte teslim edilecek. Kapak sayfası örneği http://w3.gazi.edu.tr/web/akcayol
  adresinde downloads bölümünden elde edilebilir.

```
Quicksort(A,p,r)
if p<r
   then q←Partition(A,p,r)
        Quicksort(A,p,q)
        Quicksort(A, q+1, r)
```

```
Partition(A,p,r)
x ← A[r]
i ← p-1
j ← r+1
while TRUE
  repeat j←j-1
     until A[j] ≤ x
  repeat i←i+1
     until A[i] ≥ x
  if i < j
     then exchange A[i]↔A[j]
     else return j
```