Analisis Algoritma Rekursif dengan Substitusi

Wijayanti N Khotimah

Pendahuluan

- Pada pertemuan sebelumnya anda sudah belajar cara merubah algoritma rekursif ke dalam recurrence.
- Setelah suatu algoritma rekursif dikonversikan ke dalam reccurence, untuk menghitung kompleksitas algoritma tersebut adalah dengan menyelesaikan recurrence nya.

Cara Penyelesaian Recurrence

Secara umum penyelesaian recurrence ada 3 cara yaitu:

1. Substitusi → penyelesaian recurrence menggunakan induksi matematika
2. Pohon Rekursi (recursion tree)→ mengonversi recurrence dalam bentuk tree dimana node-nodenya merepresentasikan cost pada masing-masing level
3. Master Method → mengubah recurrence dalam bentuk $T(n)=aT(n/b)+f(n)$

Substitusi

Contoh: Recurrence dari problem mergesort

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

Penyelesaian:

$T(n) = 2T\left(\frac{n}{2}\right) + \text{n, dimana T}(1) = 1$ ....(1)

Missal: $n=2^k$ dan $k={}^2\log n$ ......(2)

Persamaan (2) dimasukkan ke persamaan (1)

Maka didapat:

$$T(2^k) = 2T(2^{k-1}) + 2^k$$

$$= 2(2T(2^{k-2}) + 2^{k-1}) + 2^k$$

$$= 2^2T(2^{k-2}) + 2.\,2^{k-1} + 2^k$$

$$= 2^2T(2^{k-2}) + 2^k + 2^k$$

$$= 2^2T(2^{k-2}) + 2.\,2^k \text{ .....(3)}$$

Substitusi

Kalau sudah ketemu polanya dibuat polanya dari persamaan (3)

$T(2^k) = 2^i T(2^{k-i}) + i.2^k$ .....(4)

Pada persamaan (1) di dapat $\mathrm{T}(1) = 1$ ekuivalen dengan $\mathrm{T}(2^0) = 1$

Maka dimisalkan *k-i=0, i=k* sehingga persamaan (4) menjadi

$$T(2^k) = 2^k T(2^{k-k}) + k.2^k$$

$$= 2^k T(2^0) + k.2^k$$

$$= 2^k.1 + k.2^k \text{ ......(5)}$$

Persamaan (2) dimasukkan ke dalam persamaan (5) sehingga

$$T(n) = 2^{\log_2 n}.1 + \log_2 n\,.\,2^{\log_2 n}$$

$$= n + n.\log_2 n = \Theta(n \log n)$$

Jadi kompleksitas merge sort adalah ***Θ(nlogn)***

Contoh: algoritma rekursif untuk menghitung *n*!

ALGORITHM $F(n)$

//Computes n! recursively
//Input: A nonnegative integer n
//Output: The value of n!
if $n = 0$ **return** 1
else return $F(n-1) * n$

- Apa yang menjadi ukuran input?
- Apa yang menjadi operasi dasar?
- Berapa kali operasi dasar dieksekusi?
  - Jika jumlah eksekusi operasi dasar dinotasikan dengan M(n), karena F(n) dihitung menggunakan rumus F(n) = F(n – 1) * n, maka M(n) = M(n – 1) + 1.
  - M(n) tidak secara eksplisit merupakan fungsi terhadap *n.*
  - Persamaan untuk M(n) diatas disebut recurrence relations atau recurrences.
  - Solusi: Ubah *recurrences* menjadi fungsi eksplisit terhadap *n*.

Contoh 1: algoritma rekursif untuk menghitung *n*!

ALGORITHM $F(n)$

//Computes $n!$ recursively
//Input: A nonnegative integer n
//Output: The value of $n!$
if $n = 0$ **return** 1
else return $F(n-1) * n$

- Recurrence M(n) = M(n – 1) + 1 memiliki banyak solusi jika tidak diberi kondisi awal, misal:
  - ✓1, 2, 3, 4, 5, ...
  - ✓5, 6, 7, 8, 9, ...
  - ✓dst
- Untuk algoritma pada contoh 1, kondisi awal bisa diambil dari kondisi kapan fungsi rekursi berhenti, sehingga
- Jadi, *recurrence* jumlah iterasi:

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

Solving Recurrences (1)

- Backward Substitutions

$$
\begin{aligned}
M(n) &= M(n-1) + 1 && \text{substitute } M(n-1) = M(n-2) + 1\\
&= [M(n-2) + 1] + 1 = M(n-2) + 2 && \text{substitute } M(n-2) = M(n-3) + 1\\
&= [M(n-3) + 1] + 2 = M(n-3) + 3.
\end{aligned}
$$

- Apa yang menjadi pola selanjutnya?
- Dengan $M(n) = M(n-1) + 1 = \cdots = M(n-i) + i = \cdots = M(n-n) + n = n.$

Jadi kompleksitas untuk menghitung algoritma iteratif adalah T(n)= n

Contoh 2: algoritma rekursif untuk menghitung jumlah digit pada angka biner sebuah bilangan desimal

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

- *Recurrence* untuk total operasi penjumlahan:

  dengan kondisi awal
- Solusi *recurrence* dicari dengan *backward substitution,* dengan $n = 2^k$ untuk memudahkan penghitungan?

[redacted]

$$
\begin{aligned}
A(2^k) &= A(2^{k-1}) + 1 && \text{substitute } A(2^{k-1}) = A(2^{k-2}) + 1 \\
&= [A(2^{k-2}) + 1] + 1 = A(2^{k-2}) + 2 && \text{substitute } A(2^{k-2}) = A(2^{k-3}) + 1 \\
&= [A(2^{k-3}) + 1] + 2 = A(2^{k-3}) + 3 && \dots \\
&\dots \\
&= A(2^{k-i}) + i \\
&\dots \\
&= A(2^{k-k}) + k.
\end{aligned}
$$

$$A(2^k) = A(1) + k = k, \quad n = 2^k \text{ and hence } k = \log_2 n, \quad A(n) = \log_2 n \in \Theta(\log n).$$