Analisa Kompleksitas Algoritma Iteratif

Wijayanti N Khotimah, M.Sc.

Tujuan Perkuliahan

- Mahasiswa mampu melakukan perhitungan kompleksitas algoritma-algoritma iteratif secara langsung
- Mahasiswa mampu melakukan perhitungan kompleksitas algoritma-algoritma iteratif dengan menggunakan basic operation

Cara I: Menghitung Jumlah Eksekusi Setiap Operasi

INSERTION-SORT(A)	*cost*	*times*
for $j = 2$ **to** $A.length$	c_1	n
$\quad key = A[j]$	c_2	$n-1$
$\quad$ // Insert $A[j]$ into the sorted sequence $A[1\,..\,j-1]$.	0	$n-1$
$\quad i = j-1$	c_4	$n-1$
$\quad$ **while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
$\qquad A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
$\qquad i = i-1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
$\quad A[i+1] = key$	c_8	$n-1$

Cara I: Menghitung Jumlah Eksekusi Setiap Operasi (2)

- ✓ Selanjutnya running time diperoleh dengan menjumlahkan seluruh waktu yang dibutuhkan untuk eksekusi algoritma

$$
\begin{aligned}
T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) \\
&\quad + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1) \,.
\end{aligned}
$$

- ✓ Dari contoh di atas tampak bahwa t_j adalah suatu variabel yang nilainya tergantung pada isi dari input.
- ✓ Jika input sudah terurut, maka t_j pada baris ke-5 akan bernilai 1 untuk *j=2,3,..,n*.

Cara I: Menghitung Jumlah Eksekusi Setiap Operasi (3)

- ✓ Kondisi inilah yang disebut ***best case*** dengan running time:

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) \\ &= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8) . \end{aligned}$$

- ✓ Running time tersebut dapat diekspresikan dengan *an+b* (**fungsi linear**)

Cara I: Menghitung Jumlah Eksekusi Setiap Operasi (4)

- Jika isi input dalam kondisi urutan yang terbalik, maka pada baris no 5 kita harus membandingkan seluruh isi dari subarray A[1,...,j-1] sehingga $t_j = j$ untuk j=2,3,4,...,n
- Sehingga running timenya adalah:

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) \\ &\quad + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1) \\ &= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)n \\ &\quad - (c_2 + c_4 + c_5 + c_8)\,. \end{aligned}$$

Cara II: Menghitung Jumlah Eksekusi Basic Operation (2)

- **Basic operation** adalah operasi di dalam algoritma yang paling berpengaruh terhadap running time.

Langkah-langkah menghitung kompleksitas:

1. Tentukan parameter n yang mengindikasikan ukuran input
2. Identifikasi *basic operation* dari algoritma tersebut.
3. Tentukan *worst*, *average*, dan *best* cases untuk input dengan ukuran *n*
4. Tentukan penjumlahan *basic operation* dieksekusi.
5. Sederhanakan penjumlahan tersebut menggunakan Appendix A (pada halaman berikutnya)

APENDIKS A

Important Summation Formulas

1. $\sum_{i=l}^{u} 1 = \underbrace{1 + 1 + \cdots + 1}_{u-l+1 \text{ times}} = u - l + 1$ (l, u are integer limits, $l \le u$); $\sum_{i=1}^{n} 1 = n$

2. $\sum_{i=1}^{n} i = 1 + 2 + \cdots + n = \frac{n(n+1)}{2} \approx \frac{1}{2}n^2$

3. $\sum_{i=1}^{n} i^2 = 1^2 + 2^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6} \approx \frac{1}{3}n^3$

4. $\sum_{i=1}^{n} i^k = 1^k + 2^k + \cdots + n^k \approx \frac{1}{k+1}n^{k+1}$

5. $\sum_{i=0}^{n} a^i = 1 + a + \cdots + a^n = \frac{a^{n+1} - 1}{a - 1}$ $(a \ne 1)$; $\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$

6. $\sum_{i=1}^{n} i2^i = 1 \cdot 2 + 2 \cdot 2^2 + \cdots + n2^n = (n-1)2^{n+1} + 2$

7. $\sum_{i=1}^{n} \frac{1}{i} = 1 + \frac{1}{2} + \cdots + \frac{1}{n} \approx \ln n + \gamma$, where $\gamma \approx 0.5772\ldots$ (Euler's constant)

8. $\sum_{i=1}^{n} \lg i \approx n \lg n$

APENDIKS A

Sum Manipulation Rules

1. $$\sum_{i=l}^{u} c a_i = c \sum_{i=l}^{u} a_i$$

2. $$\sum_{i=l}^{u} (a_i \pm b_i) = \sum_{i=l}^{u} a_i \pm \sum_{i=l}^{u} b_i$$

3. $$\sum_{i=l}^{u} a_i = \sum_{i=l}^{m} a_i + \sum_{i=m+1}^{u} a_i, \text{ where } l \le m < u$$

4. $$\sum_{i=l}^{u} (a_i - a_{i-1}) = a_u - a_{l-1}$$

CONTOH 1: Maximum element

ALGORITHM $MaxElement(A[0..n-1])$

//Determines the value of the largest element in a given array
//Input: An array $A[0..n-1]$ of real numbers
//Output: The value of the largest element in A
$maxval \leftarrow A[0]$
for $i \leftarrow 1$ **to** $n-1$ **do**
  if $A[i] > maxval$
    $maxval \leftarrow A[i]$
return $maxval$

Basic operation

Langkah penyelesaian:

1. Input size: jumlah elemen dalam array A
2. Basic operation: operasi yang sering dieksekusi terdapat di dalam for loop, ada dua operasi yaitu perbandingan dan assignment. Karena perbandingan selalu dieksekusi didalam loop sedangkan assignment tidak maka basic operationnya adalah perbandingan.

Contonh 1 lanjut

- Langkah penyelesaian:

1. Input size: jumlah elemen dalam array A
2. Basic operation: operasi yang sering dieksekusi terdapat di dalam for loop, ada dua operasi yaitu perbandingan dan assignment. Karena perbandingan selalu dieksekusi didalam loop sedangkan assignment tidak maka basic operationnya adalah perbandingan.
3. Pencarian worst, average, dan best case tidak perlu karena operasinya perbandingan akan dilakukan sebanyak n kali untuk semua kondisi

4 $$C(n) = \sum_{i=1}^{n-1} 1.$$

5 $$C(n) = \sum_{i=1}^{n-1} 1 = n - 1 \in \Theta(n).$$

Contoh 2: Element uniqueness problem

ALGORITHM $UniqueElements(A[0..n-1])$
//Determines whether all the elements in a given array are distinct
//Input: An array $A[0..n-1]$
//Output: Returns "true" if all the elements in A are distinct
//   and "false" otherwise
for $i \leftarrow 0$ **to** $n-2$ **do**
    for $j \leftarrow i+1$ **to** $n-1$ **do**
        if $A[i] = A[j]$ **return false**
return true

Penyelesaian:

1. Input size: n (ukuran array)
2. Basic operation: karena loop yang terdalam hanya mengandung satu operasi, yaitu perbandingan, kita menganggapnya sebagai basic operation

Contoh 2 lanjut

Penyelesaian:

1. Input size: n (ukuran array)
2. Basic operation: karena loop yang terdalam hanya mengandung satu operasi, yaitu perbandingan, kita menganggapnya sebagai basic operation
3. Penghitungan kali ini hanya dibatasi pada kondisi worst case, yaitu ketika tidak ada elemen yang sama atau ketika dua elemen terakhir sama

4

$$
\begin{aligned}
C_{worst}(n) &= \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} [(n-1) - (i+1) + 1] = \sum_{i=0}^{n-2} (n-1-i) \\
&= \sum_{i=0}^{n-2} (n-1) - \sum_{i=0}^{n-2} i = (n-1) \sum_{i=0}^{n-2} 1 - \frac{(n-2)(n-1)}{2} \\
&= (n-1)^2 - \frac{(n-2)(n-1)}{2} = \frac{(n-1)n}{2} \approx \frac{1}{2} n^2 \in \Theta(n^2).
\end{aligned}
$$

Contoh 3: Matrix multiplication

ALGORITHM *MatrixMultiplication*$(A[0..n-1, 0..n-1], B[0..n-1, 0..n-1])$

//Multiplies two n-by-n matrices by the definition-based algorithm
//Input: Two n-by-n matrices A and B
//Output: Matrix $C = AB$
for $i \leftarrow 0$ **to** $n-1$ **do**
  for $j \leftarrow 0$ **to** $n-1$ **do**
    $C[i, j] \leftarrow 0.0$
    for $k \leftarrow 0$ **to** $n-1$ **do**
      $C[i, j] \leftarrow C[i, j] + A[i, k] * B[k, j]$
return C

- Penyelesaian:

1. Input size: ukuran matrix (n)
2. Basic operation: dianggap 1, perkalian

$$M(n) = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\sum_{k=0}^{n-1} 1 = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} n = \sum_{i=0}^{n-1} n^2 = n^3.$$

LATIHAN

No 1

Consider the following algorithm.

ALGORITHM *Mystery*(n)

//Input: A nonnegative integer n
$S \leftarrow 0$
for $i \leftarrow 1$ **to** n **do**
    $S \leftarrow S + i * i$
return S

a. What does this algorithm compute?

b. What is its basic operation?

c. How many times is the basic operation executed?

d. What is the efficiency class of this algorithm?

e. Suggest an improvement, or a better algorithm altogether, and indicate its efficiency class. If you cannot do it, try to prove that, in fact, it cannot be done.

No 2

Consider the following algorithm.

ALGORITHM $Secret(A[0..n-1])$

//Input: An array $A[0..n-1]$ of n real numbers
$minval \leftarrow A[0]$; $maxval \leftarrow A[0]$
for $i \leftarrow 1$ **to** $n-1$ **do**
  if $A[i] < minval$
    $minval \leftarrow A[i]$
  if $A[i] > maxval$
    $maxval \leftarrow A[i]$
return $maxval - minval$

Answer questions (a)–(e) of Problem 4 about this algorithm.

No 3

Consider the following algorithm.

ALGORITHM *Enigma*$(A[0..n-1, 0..n-1])$

//Input: A matrix $A[0..n-1, 0..n-1]$ of real numbers

for $i \leftarrow 0$ **to** $n-2$ **do**

  for $j \leftarrow i+1$ **to** $n-1$ **do**

    if $A[i, j] \neq A[j, i]$

      return false

return true

Answer questions (a)–(e) of Problem 4 about this algorithm.

No 4

. Consider the following version of an important algorithm that we will study later in the book.

ALGORITHM $GE(A[0..n-1, 0..n])$

```
//Input: An n × (n + 1) matrix A[0..n − 1, 0..n] of real numbers
for i ← 0 to n − 2 do
    for j ← i + 1 to n − 1 do
        for k ← i to n do
            A[j, k] ← A[j, k] − A[i, k] ∗ A[j, i] / A[i, i]
```

a. Find the time efficiency class of this algorithm.

b. What glaring inefficiency does this pseudocode contain and how can it be eliminated to speed the algorithm up?

No 5

For each of the following pairs of functions, indicate whether the first function of each of the following pairs has a lower, same, or higher order of growth (to within a constant multiple) than the second function.

a. $n(n+1)$ and $2000n^2$ **b.** $100n^2$ and $0.01n^3$

c. $\log_2 n$ and $\ln n$ **d.** $\log_2^2 n$ and $\log_2 n^2$

e. 2^{n-1} and 2^n **f.** $(n-1)!$ and $n!$

No 6

- Tentukan apakah pernyataan berikut benar atau salah

a. $n(n+1)/2 \in O(n^3)$

b. $n(n+1)/2 \in O(n^2)$

c. $n(n+1)/2 \in \Theta(n^3)$

d. $n(n+1)/2 \in \Omega(n)$

No 7

- Bubble sort merupakan algoritma yang terkenal tetapi tidak efisien. Algoritma ini bekerja dengan melakukan swapping pada dua elemen yang berdekatan yang tidak sesuai dengan urutan.

```
BUBBLESORT(A)
1  for i = 1 to A.length − 1
2      for j = A.length downto i + 1
3          if A[j] < A[j − 1]
4              exchange A[j] with A[j − 1]
```

a. Misal A’ adalah outbut dari BubbleSort (A). Untuk menganalisa algoritma bubble sort, selain membuktikan bahwa algoritma ini pasti berhenti dan memenuhi A’[1]<A’[2]<…<A’[n], apalagi yang harus dibuktikan?
b. Nyatakan loop invariant untuk for loop pada baris 2-4
c. Menggunakan termination condition pada loop invariant soal b, nyatakan loop invariant pada for loop baris 1-4
d. Apa worstcase dari algoritma tersebut? Bagaimana jika dibandingkan dengan insertion sort?

No 7

- Correctness of Horner's rule
- Berikut adalah potongan implementasi horner's rule untuk mengevaluasi suatu polinomial:

$$
\begin{aligned}
P(x) &= \sum_{k=0}^{n} a_k x^k \\
&= a_0 + x(a_1 + x(a_2 + \cdots + x(a_{n-1} + x a_n)\cdots)) ,
\end{aligned}
$$

- Jika nilai a_0, a_1, …, a_n serta nilai x sudah diketahui maka:

$y = 0$
for $i = n$ **downto** 0
    $y = a_i + x \cdot y$

[redacted]

a. Nyatakan dengan Θ kompleksitas dari potongan algoritma tersebut
b. Tulis dengan pseudocode untuk mengimplementasikan algoritma naïve polynomial-evaluation yang menghitung setiap term polynomial dari dasar.
c. Bagaimana running timenya dan bagimana jika dibandingkan dengan Horner’s rule?