Algoritma Rekursif

Elsa Elvira Awal, M.Kom

Review Materi Sebelumnya

- Menerapkan lima langkah analisis kompleksitas algoritma untuk sequential search pada contoh array $A = [46, 59, 95, 28, 80]$, dengan mencari elemen tertentu.
- Kita menganalisis tiga scenario: best case, worst case, dan average case.

$$A = [46, 59, 95, 28, 80]$$

- **Menentukan Parameter yang Mengindikasi Ukuran Input**
  - Pada kasus ini, ukuran input adalah panjang array n, di mana $n = 5$
- **Mengidentifikasi Basic Operation Algoritma**
  - Basic operation dalam sequential search adalah perbanding antara elemen dalam array dengan elemen yang dicari, yaitu $A[i] \neq K$

$$A = [46, 59, 95, 28, 80]$$

- **Menentukan Apakah untuk Ukuran Input yang Sama, Banyaknya Eksekusi Basic Operation Bisa Berbeda**
  - Jumlah eksekusi basic operation dapat bervariasi tergantung pada posisi elemen yang dicari dalam array:
    - Best case: jika elemen yang dicari ada di indeks pertama (hanya satu kali perbandingan diperlukan)
    - Worst case: jika elemen berada di indeks terakhir atau tidak ada dalam array (perlu memeriksa semua elemen)
    - Average case: rata-rata elemen ditemukan di tengah atau di posisi acak, dengan jumlah perbandingan sekitar $\frac{n+1}{2}$

$A = [46, 59, 95, 28, 80]$

- **Menentukan Rumus Deret yang Menunjukkan Berapa Kali Basic Operation Dieksekusi**
  - Best case: $T(n) = 1$
  - Worst case: $T(n) = n$
  - Average case: $T(n) = \frac{n+1}{2}$
- **Selesaikan Rumus Deret untuk Menghitung Banyaknya Eksekusi Basic Operation**
  - Dalam notasi Big-O:
    - Best case: $O(1)$
    - Worst case: $O(n)$
    - Average case: $O(n)$

Contoh

- Misalkan kita ingin mencari elemen $K = 46$
- Array: $A = [46, 59, 95, 28, 80]$
- Langkah:
  - $i = 0$, cek apakah $A[0] = 45$. Karena $A[0] = K$, algoritma berhenti dan mengembalikan indeks 0.
- Jumlah Perbandingan: 1 kali
- Kompeksitas Waktu: $O(1)$
- Maka ini merupakan **Best Case**

Contoh

- Misalkan kita ingin mencari elemen $K = 100$, yang tidak ada dalam array.
- Array: $A = [46, 59, 95, 28, 80]$
- Langkah:
  - $i = 0$, cek apakah $A[0] = 100$. Tidak sama.
  - $i = 1$, cek apakah $A[1] = 100$. Tidak sama.
  - $i = 2$, cek apakah $A[2] = 100$. Tidak sama.
  - $i = 3$, cek apakah $A[3] = 100$. Tidak sama.
  - $i = 4$, cek apakah $A[4] = 100$. Tidak sama.
  - Karena $i = 5$ (di luar batas array), algoritma mengembalikan -1 (elemen tidak ditemukan)
- Jumlah Perbandingan: 5 kali.
- Kompleksitas Waktu: $O(n)$. Maka ini merupakan **Worst Case**

Contoh

- Misalkan kita ingin mencari elemen $K = 95$, yang berada di posisi tengah.
- Array: $A = [46, 59, 95, 28, 80]$
- Langkah:
  - $i = 0$,cek apakah $A[0] = 95$. Tidak sama.
  - $i = 1$, cek apakah $A[1] = 95$. Tidak sama.
  - $i = 2$, cek apakah $A[2] = 95$. Sama. Algoritma berhenti dan mengembalikan indeks 2.
- Jumlah Perbandingan: 3 kali.
- Rata-rata Kasus: 3 kali.
- Kompleksitas Waktu: $O(n)$
- Maka ini merupakan **Average Case**

Algoritma Rekursif

- Fungsi yang memanggil dirinya sendiri
- Fungsi F disebut rekursif jika
  - Di bagian badan F ada pemanggilan terhadap F
  - Di bagian F ada pemanggilan fungsi G. dan di badan G ada pemanggilan terhadap F

Algoritma Rekursif

- Fungsi F akan disebut “rekursif” jika di dalam tubuh atau implementasinya, ada instruksi untuk memanggil F lagi. Ini merupakan ciri khas dari fungsi rekursif.

```
def fungsiF():
    ...
    fungsiF()
    ...
```

Algoritma Rekursif

- Fungsi F memanggil fungsi lain, misalnya F, dan kemudian G kembali memanggil F. ini adalah contoh rekursi tidak langsung. Alur pemanggil semacam ini juga dianggap sebagai rekursi, tetapi disebut”rekursi tidak langsung” karena terjadi melalui fungsi perantara (dalam hal ini G)

```
def fungsiF():
    fungsiG()


def fungsiG():
    fungsiF()
```

Algoritma Rekursif

- Ada beberapa jenis permasalahan yang cocok diselesaikan secara rekursif:
  - Dengan menggunakan rekursif kode akan menjadi sederhana dan singkat
  - Versi iteratifnya sangat kompleks
  - Fungsi rekursif umumnya membutuhkan memori lebih banyak
  - Lebih sulit mentrace fungsi rekursif disbanding iteratif

Algoritma Rekurisf

- Tidak semua permasalahan bisa diselesaikan menggunakan rekursif
- Ciri-ciri permasalahan yang dapat diselesaikan oleh fungsi rekursif
  - Memiliki kasus sederhana yang dapat langsung diselesaikan (base case). Contoh: $1! = 1$
  - Kasus bersar dapat diubah menjadi kasus sejenis yang lebih sederhana (recursive case). Contoh: $n! = n \times (n-1)!$
  - Dengan menerapkan recursive case secara berulang maka kasus besar akan mendekati dan sampai pada base case. Contoh: $n! \rightarrow (n-1) \rightarrow (n-2) \rightarrow \cdots 1!$

Contoh Soal 1

- Buatlah funsgi non-rekursif dan fungsi rekursif untuk menghitung $n!$
- Maka $n! = n \times (n-1) \times (n-2) \times \cdots \times 1$. Misal $10! = 10 \times 9 \times 8 \times \cdots \times 1$
- Punya sifat menarik $9! = 9 \times 8!$ Kasus faktorial bisa diubah menjadi kasus faktorial yang lebih sederhana dan memenuhi ciri-ciri permasalahan yang bisa diselesaikan secara rekursif:
  - Base case: $1! = 1$
  - Recursive case: $n! = n \times (n-1)!$
  - Dengan menerapkan recursive case secara berulang. Maka kasus besar akan bisa dicapai base case. Contoh: $9! = 9 \times 8! \rightarrow 8! \times 7! \rightarrow 7! = 7 \times 6! \rightarrow$ dst akan mencapai $1!$

Algoritma Non-Rekursif

Algoritma Faktorial(n)

// Mencari nilai faktorial dari suatu bilangan n

// Input: n (bilangan bulat)

// Output: $n!$

```
nfaktorial ← 1
for i ← 1 to n do
    nfaktorial ← nfaktorial * i
return nfaktorial
```

Algoritma Rekursif

Algoritma nfactorialrekursif (n)

// Mencari nilai faktorial dari bilangan n secara rekursif

// Input: n (bilangan bulat)

// Output: $n!$

```
if n ≤ 1 then
    return 1
else
    return n * nfactorialrekursif(n − 1)
end if
```

Algoritma Rekursif

- Penjelasan:
  - Base Case (if $n \leq 1$ then return 1):
    - Jika $n \leq 1$, maka algoritma akan mengembalikan 1, karena $0!$ dan $1!$ sama-sama bernilai 1.
  - Recursive Case (else return n * nfactorialrekursif$(n-1)$):
    - Jika $n > 1$, algoritma akan memanggil dirinya dengan $n-1$ dan mengalikan dengan n. Ini akan terus berlanjut sampai base case.

Pohon Rekursif

faktorial(5) = 5 * faktorial(4)   pending

faktorial(4) = 4 * faktorial(3)   pending

faktorial(3) = 3 * faktorial(2)   pending

faktorial(2) = 2 * faktorial(1)   pending

faktorial(1)

Pohon Rekursif

faktorial(5) = 5 * faktorial(4) eksekusi → 120

faktorial(4) = 4 * faktorial(3) eksekusi

faktorial(3) = 3 * faktorial(2) eksekusi

faktorial(2) = 2 * faktorial(1) eksekusi

faktorial(1) eksekusi

Contoh Soal 2

- Buatlah fungsi rekursif untuk menghitung nilai $a \times b$ (perkalian)
- Makna $a \times b = b + b + \cdots + b$ (sebanyak a). Misal $3 \times 5 = 5 + 5 + 5$
- Punya sifat menarik $3 \times 5 = (3 - 1) \times 5 + 5$.
- Kasus perkalian bisa diubah menjadi kasus perkalian yang lebih sederhana.
- Memenuhi ciri-ciri permasalahan yang bisa diselesaikan secara rekursif
  - Base case: $1 \times b = b$
  - Recursive case: $a \times b = (a - 1) \times b + b$
  - Dengan menerapkan recursive case secara berulang. Maka kasus besar akan bisa mencapai base case. Contoh: $4 \times 5 \rightarrow 3 \times 5 \rightarrow$ dst akan mencapai 1×5

Algoritma Rekursif Perkalian

Algoritma perkalianrekursif(m, n)

// Mencari nilai perkalian variabel yang diinputkan

// Input: variabel m dan n

// Output: return nilai perkalian m dan n

if $m \leftarrow 1$ then

    return n

else

    return perkalianrekursif $(m-1, n) + n$

end if

Pohon Rekursif

kali $(5, 4)$ = kali $(4, 4) + 4$ pending

kali $(4, 4)$ = kali $(3, 4) + 4$ pending

kali $(3, 4)$ = kali $(2, 4) + 4$ pending

kali $(2, 4)$ = kali $(1, 4) + 4$ pending

kali $(1, 4)$ pending

Pohon Rekursif

kali $(5, 4)$ = kali $(4, 4) + 4$ eksekusi

kali $(4, 4)$ = kali $(3, 4) + 4$ eksekusi

kali $(3, 4)$ = kali $(2, 4) + 4$ eksekusi

kali $(2, 4)$ = kali $(1, 4) + 4$ eksekusi

kali $(1, 4)$ eksekusi

Thank You