

Struktur Data - Algoritma Pencarian (*Searching Algorithms*)

Pencarian (*Searching*) adalah proses menemukan data tertentu, yang dapat berupa satu elemen atau sekumpulan kecil data, di dalam kumpulan data yang sangat besar. Data dapat berbentuk array, linked list, pohon, heap, graf, dan sebagainya. Seiring bertambahnya jumlah data saat ini, tersedia berbagai teknik untuk melakukan pencarian.

Algoritma Pencarian dalam Struktur Data

Berbagai teknik pencarian dapat diterapkan pada struktur data untuk memperoleh data tertentu. Pencarian dikatakan berhasil apabila menghasilkan elemen atau data yang diinginkan. Jika tidak, pencarian dinyatakan tidak berhasil.

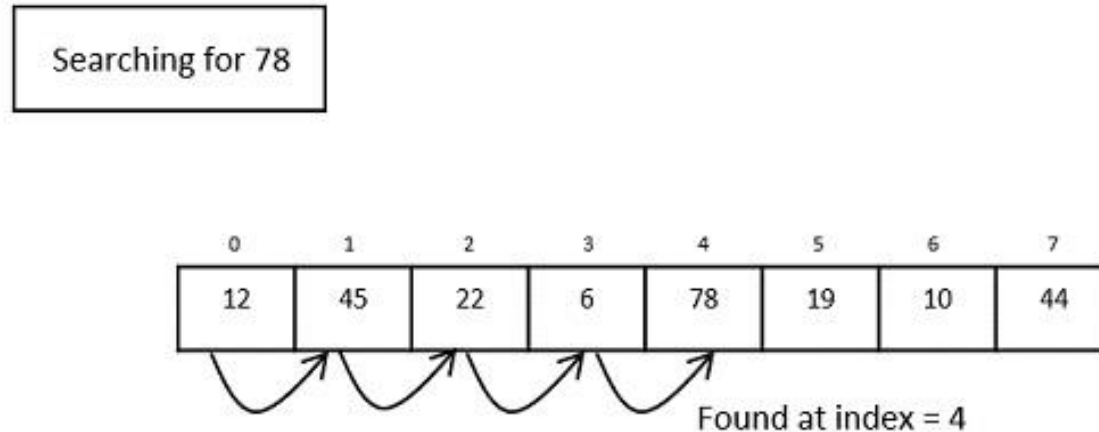
Teknik pencarian ini terbagi menjadi dua kategori, yaitu:

- Pencarian Berurutan (*Sequential Searching*)
- Pencarian Interval (*Interval Searching*)

Pencarian Berurutan (*Sequential Searching*)

Sesuai namanya, pencarian berurutan menelusuri setiap elemen data secara berurutan untuk menemukan data yang diinginkan. Pada jenis pencarian ini, data tidak harus terurut.

Contoh — Linear Search

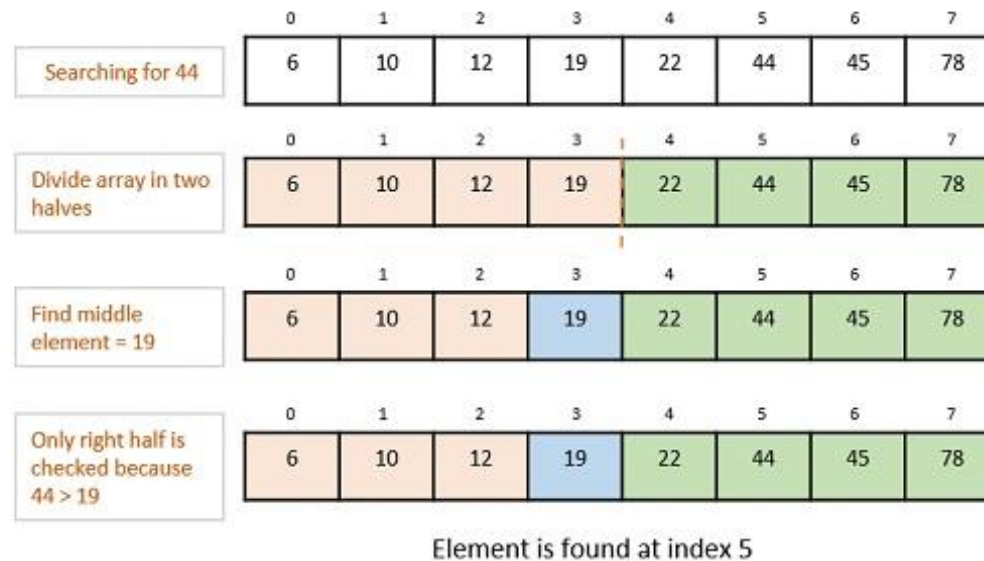


Gambar 1: Proses Linear Search

Pencarian Interval (*Interval Searching*)

Berbeda dengan pencarian berurutan, pencarian interval mengharuskan data sudah terurut. Metode ini biasanya mencari data dalam rentang tertentu, baik dengan membagi data menjadi beberapa bagian maupun melompati indeks untuk menemukan suatu elemen.

Contoh — Binary Search, Jump Search, dan sebagainya.



Gambar 2: Proses Binary Search

Mengevaluasi Algoritma Pencarian

Tidak semua teknik pencarian cocok untuk semua jenis struktur data. Dalam beberapa keadaan, pencarian berurutan lebih sesuai, sedangkan pada keadaan lain pencarian interval lebih sesuai. Teknik-teknik tersebut dievaluasi dengan memeriksa waktu eksekusi yang dibutuhkan setiap metode pada masukan tertentu.

Di sinilah notasi asimtotik digunakan.

Secara singkat, terdapat tiga kasus kompleksitas waktu dalam menjalankan program, yaitu:

- Kasus Terbaik (*Best Case*)
- Kasus Rata-Rata (*Average Case*)
- Kasus Terburuk (*Worst Case*)

Pembahasan terutama berfokus pada kompleksitas waktu kasus terbaik dan kasus terburuk karena kasus rata-rata sulit dihitung. Karena waktu eksekusi bergantung pada banyaknya masukan yang diberikan kepada program, kompleksitas waktu kasus terburuk dianggap paling menggambarkan kinerja suatu algoritma.

Sebagai contoh, kompleksitas waktu kasus terbaik linear search adalah **$O(1)$** , yaitu ketika elemen yang dicari ditemukan pada perulangan pertama. Sementara itu, kompleksitas waktu kasus terburuk adalah **$O(n)$** , ketika program menelusuri seluruh elemen tetapi tetap tidak menemukan elemen yang dicari. Keadaan ini disebut pencarian tidak berhasil. Oleh karena itu, kompleksitas waktu linear search dinyatakan sebagai **$O(n)$** , dengan n sebagai jumlah elemen dalam struktur data masukan.

Banyak metode pencarian digunakan untuk menemukan data dalam berbagai struktur data. Beberapa di antaranya adalah:

- **Linear Search**
- **Binary Search**
- Interpolation Search
- Jump Search
- Hash Table
- Exponential Search
- Sublist search
- Fibonacci Search
- Ubiquitous Binary Search

Kita akan membahas 2 diantaranya yaitu Linear Search dan Binary Search secara lebih rinci.

Algoritma Linear Search

Linear search adalah salah satu jenis algoritma pencarian berurutan. Dalam metode ini, setiap elemen pada array masukan ditelusuri dan dibandingkan dengan nilai kunci yang dicari. Jika ditemukan kecocokan, pencarian dinyatakan berhasil. Jika tidak ditemukan kecocokan, pencarian dinyatakan tidak berhasil dan menghasilkan kompleksitas waktu kasus terburuk.

Sebagai contoh, pada diagram animasi yang ditampilkan, kita mencari elemen 33. Metode linear search mencarinya secara berurutan mulai dari elemen pertama hingga menemukan kecocokan. Hasilnya adalah pencarian yang berhasil.



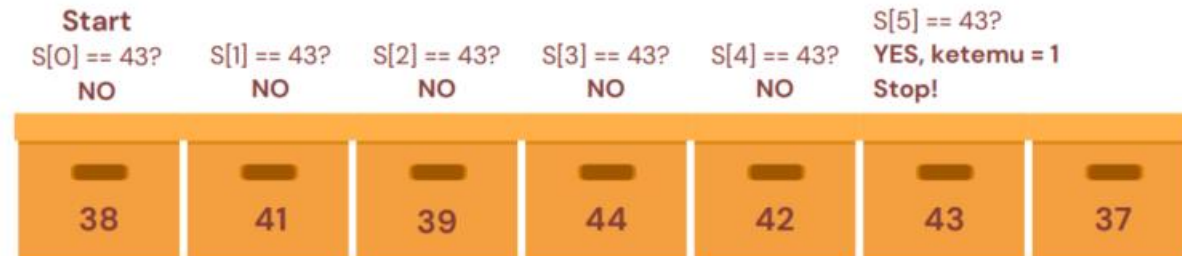
Pada diagram yang sama, jika kita mencari elemen 46, pencarian tidak berhasil karena 46 tidak terdapat dalam data masukan.

Sequential (Linear) Search

Misalnya Anda diminta untuk mencari sepatu dengan ukuran 43 di dalam kotak yang tertutup

Set var **ketemu** = 0 (Asumsi data belum ditemukan)

Pencarian berhenti apabila **ketemu** = 1



Ingat! Anda tidak pernah tau isi setiap lokernya



Sepatu ukuran 43 ditemukan di indeks ke-5

Sequential (Linear) Search

- Misalnya Anda diminta untuk mencari apakah ada **Anton** pada barisan mahasiswa? ada di urutan berapa? Bagaimana caranya?
- Cara paling sederhana adalah dengan melihat wajah mahasiswa satu persatu secara berurutan





JEJAK LINEAR SEARCH

Target 11 pada daftar pesanan kantin

Data: [17, 8, 23, 11, 29]



1. Periksa 17



Indeks 0: belum cocok

2. Periksa 8



Indeks 1: belum cocok

3. Periksa 23



Indeks 2: belum cocok

4. Periksa 11



Indeks 3: cocok, berhenti



Hasil: indeks 3. Jumlah pemeriksaan: 4.

Nilai 29 tidak perlu diperiksa setelah target ditemukan.



Algoritma Linear Search

Algoritma linear search relatif sederhana. Prosesnya dimulai dari indeks paling awal pada array masukan yang akan ditelusuri.

Langkah 1 - Mulai dari indeks 0 pada array masukan. Bandingkan nilai kunci dengan nilai pada indeks 0.

Langkah 2 - Jika nilainya sama dengan kunci, kembalikan posisi tempat nilai tersebut ditemukan.

Langkah 3 - Jika nilainya tidak sama dengan kunci, bandingkan elemen berikutnya dalam array.

Langkah 4 - Ulangi Langkah 3 hingga ditemukan kecocokan. Kembalikan posisi tempat kecocokan ditemukan.

Langkah 5 - Jika pencarian tidak berhasil, tampilkan pesan bahwa elemen tidak terdapat dalam array, lalu keluar dari program.

Pseudocode

```
prosedur linear_search (daftar, nilai)
    untuk setiap elemen dalam daftar
        jika elemen == nilai
            kembalikan lokasi elemen
        akhir jika
    akhir untuk
akhir prosedur
```

Analisis

Linear search menelusuri setiap elemen secara berurutan. Karena itu, kasus terbaik terjadi ketika elemen ditemukan pada perulangan pertama. Kompleksitas waktu kasus terbaik adalah $O(1)$.

Sementara itu, kasus terburuk pada linear search terjadi ketika pencarian tidak berhasil menemukan nilai kunci di dalam array sehingga dilakukan n perulangan. Oleh karena itu, kompleksitas waktu kasus terburuk algoritma linear search adalah $O(n)$.

Contoh

Mari kita lihat langkah-langkah pencarian nilai kunci, misalnya 47, dalam sebuah array menggunakan metode linear search.

0	1	2	3	4	5	6	7
34	10	66	27	47	8	55	78

Langkah 1

Linear search dimulai dari indeks 0. Bandingkan nilai kunci dengan nilai pada indeks 0, yaitu 34.

0	1	2	3	4	5	6	7
34	10	66	27	47	8	55	78

=
47

Namun, 47 tidak sama dengan 34. Karena itu, pencarian beralih ke elemen berikutnya.

Langkah 2

Sekarang, nilai kunci dibandingkan dengan nilai pada indeks 1 dalam array.

0	1	2	3	4	5	6	7
34	10	66	27	47	8	55	78

=
47

Nilai 47 tetap tidak sama dengan 10 sehingga algoritma melanjutkan ke perulangan berikutnya.

Langkah 3

Elemen berikutnya, yaitu 66, dibandingkan dengan 47. Keduanya tidak sama sehingga algoritma membandingkan elemen selanjutnya.

0	1	2	3	4	5	6	7
34	10	66	27	47	8	55	78

=
47

Langkah 4

Sekarang, elemen pada indeks 3, yaitu 27, dibandingkan dengan nilai kunci 47. Keduanya tidak sama sehingga algoritma melanjutkan pemeriksaan ke elemen berikutnya.

0	1	2	3	4	5	6	7
34	10	66	27	47	8	55	78

=
47

Langkah 5

Elemen pada indeks 4, yaitu 47, dibandingkan dengan nilai kunci 47. Keduanya sama. Selanjutnya, posisi tempat 47 berada, yaitu 4, dikembalikan.

0	1	2	3	4	5	6	7
34	10	66	27	47	8	55	78

=
47

Keluaran yang diperoleh adalah: Elemen ditemukan pada indeks 4.

Implementasi

Dalam tutorial ini, program Linear Search ditampilkan dalam empat bahasa pemrograman. Fungsi membandingkan elemen masukan dengan nilai kunci, lalu mengembalikan posisi kunci dalam array atau pesan pencarian tidak berhasil jika kunci tidak terdapat dalam array.

Struktur Data

Python

```
def linear_search(a, n, key):  
    count = 0  
    for i in range(n):  
        if(a[i] == key):  
            print("Elemen ditemukan pada posisi", (i+1))  
            count = count + 1  
    if(count == 0):  
        print("Elemen tidak terdapat dalam array")  
  
a = [14, 56, 77, 32, 84, 9, 10]  
n = len(a)  
key = 32  
linear_search(a, n, key)  
key = 3  
linear_search(a, n, key)
```

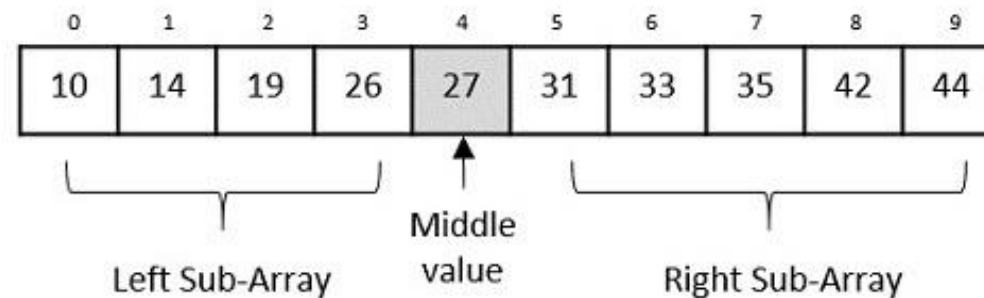
Keluaran

```
Elemen ditemukan pada posisi 4  
Elemen tidak terdapat dalam array
```

Algoritma Binary Search

Binary search adalah algoritma pencarian cepat dengan kompleksitas waktu ($\log n$). Algoritma ini bekerja berdasarkan prinsip divide and conquer karena membagi array menjadi dua sebelum melakukan pencarian. Agar algoritma bekerja dengan benar, kumpulan data harus sudah terurut.

Binary search mencari nilai kunci tertentu dengan membandingkannya dengan elemen tengah pada kumpulan data. Jika cocok, indeks elemen dikembalikan. Namun, jika nilai elemen tengah lebih besar daripada nilai kunci, pencarian dilakukan pada subarray di sebelah kanan elemen tengah. Jika tidak, pencarian dilakukan pada subarray sebelah kiri. Proses ini berlangsung secara rekursif hingga ukuran subarray menjadi nol.





Binary Search

Langkah 1: Urutkan elemen array Sepatu (Asumsikan data sudah terurut atau gunakan algoritma pengurutan)



low = 0 high = SIZE - 1

mid = (low+high)/2

= (0+6)/2 = 3

mid = indeks ke-3



key > sepatu[mid] ? YES

Lanjut pencarian bagian kanan



low = mid+1 = 4 high = SIZE - 1

mid = (low+high)/2

= (4+6)/2 = 5

mid = indeks ke-5



key = sepatu[mid] ? YES

STOP



43

Sepatu ukuran 43
ditemukan di
indeks ke-mid,
yaitu ke-5

**Ingat! Pencarian biner
selalu membagi 2 setiap
kali belum ditemukan**

Algoritma Binary Search

Binary Search merupakan metode pencarian interval yang melakukan pencarian dalam rentang tertentu. Masukannya harus berupa array terurut karena algoritma membagi array menjadi subarray berdasarkan nilai yang lebih besar atau lebih kecil. Algoritma ini mengikuti prosedur berikut:

Langkah 1 - Pilih elemen tengah dalam array dan bandingkan dengan nilai kunci yang dicari. Jika sama, kembalikan posisi elemen tengah.

Langkah 2 - Jika tidak sama, periksa apakah nilai kunci lebih besar atau lebih kecil daripada nilai tengah.

Langkah 3 - Jika nilai kunci lebih besar, lakukan pencarian pada subarray sebelah kanan. Jika nilai kunci lebih kecil daripada nilai tengah, lakukan pencarian pada subarray sebelah kiri.

Langkah 4 - Ulangi Langkah 1, 2, dan 3 secara berulang hingga ukuran subarray menjadi 1.

Langkah 5 - Jika nilai kunci tidak terdapat dalam array, algoritma mengembalikan hasil pencarian tidak berhasil.

Pseudocode algoritma binary search adalah sebagai berikut:

```
Prosedur binary_search
  A ← array terurut
  n ← ukuran array
  x ← nilai yang dicari

  Tetapkan lowerBound = 1
  Tetapkan upperBound = n

  selama x belum ditemukan
    jika upperBound < lowerBound
      KELUAR: x tidak ada.

    tetapkan midPoint = lowerBound + ( upperBound - lowerBound ) / 2

    jika A[midPoint] < x
      tetapkan lowerBound = midPoint + 1

    jika A[midPoint] > x
      tetapkan upperBound = midPoint - 1

    jika A[midPoint] = x
      KELUAR: x ditemukan pada lokasi midPoint
  akhir selama
akhir prosedur
```

Analisis

Karena algoritma binary search melakukan pencarian secara berulang, perhitungan kompleksitas waktunya tidak sesederhana linear search.

Array masukan ditelusuri secara berulang dengan membaginya menjadi beberapa subarray setelah setiap perulangan yang belum menemukan target. Oleh karena itu, relasi rekurensi yang terbentuk berupa fungsi pembagian.

Secara lebih sederhana,

- Pada perulangan pertama, elemen dicari pada seluruh array. Dengan demikian, panjang array = n .
- Pada perulangan kedua, hanya setengah dari array awal yang ditelusuri. Jadi, panjang array = $n/2$.
- Pada perulangan ketiga, setengah dari subarray sebelumnya ditelusuri. Panjang array menjadi $n/4$.
- Demikian pula, pada perulangan ke- i , panjang array menjadi $n/2^i$

Agar pencarian berhasil, setelah perulangan terakhir panjang array harus menjadi 1. Dengan demikian,

$$n/2^i = 1$$

Diperoleh:

$$n = 2^i$$

Dengan menerapkan logaritma pada kedua ruas,

$$\log n = \log 2^i$$

$$\log n = i \cdot \log 2$$

$$i = \log n$$

Kompleksitas waktu algoritma binary search adalah $O(\log n)$

Contoh

Agar binary search dapat bekerja, array yang akan ditelusuri harus sudah terurut. Kita akan mempelajari prosesnya melalui contoh bergambar. Berikut adalah array terurut yang digunakan. Misalkan kita ingin mencari lokasi nilai 31 menggunakan binary search.

0	1	2	3	4	5	6	7	8	9
10	14	19	26	27	31	33	35	42	44

Pertama, kita menentukan indeks tengah array menggunakan rumus berikut:

```
mid = low + (high - low) / 2
```

Perhitungannya adalah $0 + (9 - 0) / 2 = 4$ (nilai bulat dari 4,5). Jadi, indeks tengah array adalah 4.

0	1	2	3	4	5	6	7	8	9
10	14	19	26	27	31	33	35	42	44



Sekarang kita membandingkan nilai pada lokasi 4 dengan nilai yang dicari, yaitu 31. Nilai pada lokasi 4 adalah 27 sehingga tidak cocok. Karena nilai yang dicari lebih besar daripada 27 dan array sudah terurut, kita mengetahui bahwa nilai target berada pada bagian dengan indeks yang lebih tinggi.

Struktur Data

0	1	2	3	4	5	6	7	8	9
10	14	19	26	27	31	33	35	42	44

Kita mengubah low menjadi mid + 1, lalu menghitung nilai mid yang baru.

```
low = mid + 1  
mid = low + (high - low) / 2
```

Nilai mid yang baru adalah 7. Kita membandingkan nilai pada lokasi 7 dengan nilai target 31.

0	1	2	3	4	5	6	7	8	9
10	14	19	26	27	31	33	35	42	44



Nilai pada lokasi 7 tidak cocok, melainkan lebih kecil daripada nilai yang dicari. Jadi, nilai yang dicari harus berada pada bagian dengan indeks yang lebih rendah dari lokasi ini.

0	1	2	3	4	5	6	7	8	9
10	14	19	26	27	31	33	35	42	44

Karena itu, kita menghitung kembali mid. Kali ini nilainya adalah 5.

Struktur Data

0	1	2	3	4	5	6	7	8	9
10	14	19	26	27	31	33	35	42	44



Kita membandingkan nilai pada lokasi 5 dengan nilai target. Keduanya ternyata cocok.

0	1	2	3	4	5	6	7	8	9
10	14	19	26	27	31	33	35	42	44

Kita menyimpulkan bahwa nilai target 31 tersimpan pada lokasi 5.

Binary search membagi dua jumlah elemen yang perlu ditelusuri sehingga jumlah perbandingan yang diperlukan menjadi jauh lebih sedikit.

Implementasi

Binary search adalah algoritma pencarian cepat dengan kompleksitas waktu ($\log n$). Algoritma ini bekerja berdasarkan prinsip divide and conquer. Agar dapat bekerja dengan benar, kumpulan data harus sudah terurut.

Python

```
def binary_search(a, low, high, key):
    mid = (low + high) // 2
    if (low <= high):
        if(a[mid] == key):
            print("Elemen terdapat pada indeks:", mid)
        elif(key < a[mid]):
            binary_search(a, low, mid-1, key)
        elif (a[mid] < key):
            binary_search(a, mid+1, high, key)
    if(low > high):
        print("Pencarian tidak berhasil")

a = [6, 12, 14, 18, 22, 39, 55, 182]
n = len(a)
low = 0
high = n-1
key = 22
binary_search(a, low, high, key)
key = 54
binary_search(a, low, high, key)
```

Keluaran

```
Elemen terdapat pada indeks: 4
Pencarian tidak berhasil
```