

Struktur Data

Prasyarat Data Terurut dan Perbandingan Efisiensi Algoritme Pencarian

Minggu ke-3

Yazid Aufar, M.Kom

Pengantar

Mengapa cara mencari data perlu dipilih?

Bayangkan terdapat daftar NIM mahasiswa dan kita ingin menemukan satu NIM tertentu. Jika daftar belum terurut, kita dapat memeriksa NIM satu per satu. Jika daftar sudah terurut, kita dapat memanfaatkan urutan tersebut untuk mempersempit pencarian.

Kedua cara tersebut dapat menghasilkan data yang sama, tetapi jumlah pemeriksaannya dapat berbeda. Perbedaan ini semakin penting ketika jumlah data bertambah.

Dalam materi ini, digunakan dua algoritme:

Aspek	Linear search	Binary search
Cara kerja	Memeriksa elemen secara berurutan	Memeriksa elemen tengah dan mempersempit rentang
Kondisi data	Boleh terurut atau belum terurut	Harus terurut
Penghentian pencarian	Target ditemukan atau seluruh data selesai diperiksa	Target ditemukan atau rentang pencarian habis

Pembahasan menggunakan **indeks yang dimulai dari 0**. Jadi, elemen pertama berada pada indeks 0, elemen kedua pada indeks 1, dan seterusnya.

PRASYARAT DATA TERURUT

1. Pengertian Data Terurut

Data terurut adalah data yang disusun berdasarkan suatu aturan urutan. Dua bentuk yang umum digunakan adalah:

- **Menaik (*ascending*)**: dari nilai kecil ke nilai besar.
- **Menurun (*descending*)**: dari nilai besar ke nilai kecil.

Contoh:

Kondisi	Data
Terurut menaik	[10, 20, 30, 40, 50]
Terurut menurun	[50, 40, 30, 20, 10]
Belum terurut	[30, 10, 50, 20, 40]

Data terurut tetap dapat memiliki nilai yang sama. Misalnya, [10, 20, 20, 30] merupakan data terurut menaik dalam arti nilainya tidak menurun.

2. Mengapa Linear Search Tidak Memerlukan Data Terurut?

Linear search memeriksa setiap elemen mulai dari awal. Algoritme ini tidak menggunakan hubungan urutan antarangka untuk menentukan elemen berikutnya.

Contoh mencari 20 pada data:

[30, 10, 50, 20, 40]

Langkah	Indeks	Nilai diperiksa	Hasil
1	0	30	Tidak cocok
2	1	10	Tidak cocok
3	2	50	Tidak cocok
4	3	20	Cocok, pencarian berhenti

Target ditemukan pada indeks 3 setelah empat pemeriksaan. Urutan data tidak menghalangi pencarian karena semua elemen sebelum target diperiksa satu per satu.

3. Mengapa Binary Search Memerlukan Data Terurut?

Binary search menggunakan nilai tengah untuk menentukan bagian data yang masih mungkin berisi target.

Pada data terurut menaik:

- Jika target sama dengan nilai tengah, pencarian berhasil.
- Jika target lebih kecil daripada nilai tengah, pencarian dilanjutkan ke kiri.
- Jika target lebih besar daripada nilai tengah, pencarian dilanjutkan ke kanan.

Keputusan membuang separuh rentang hanya dapat dibenarkan jika data sudah terurut.

Contoh:

Data : [10, 20, 30, 40, 50]

Target : 20

Nilai tengah adalah 30. Karena $20 < 30$, target hanya mungkin berada di bagian kiri. Bagian kanan dapat diabaikan karena seluruh nilainya lebih besar daripada atau sama dengan 30.

4. Akibat Menggunakan Binary Search Pada Data Belum Terurut

Perhatikan data berikut:

Data : [30, 10, 50, 20, 40]

Target : 20

Jika digunakan binary search dengan aturan urutan menaik:

Langkah	low	high	mid	Nilai tengah	Keputusan
1	0	4	2	50	Target lebih kecil, cari ke kiri
2	0	1	0	30	Target lebih kecil, cari ke kiri
Berhenti	0	-1	-	-	Rentang habis

Algoritme menyatakan target tidak ditemukan, padahal 20 ada pada indeks 3.

Kesalahan terjadi ketika bagian kanan dibuang pada langkah pertama. Karena data belum terurut, bagian kanan masih mungkin berisi nilai yang lebih kecil daripada nilai tengah.

Binary search pada data belum terurut kadang dapat menemukan target secara kebetulan, tetapi hasilnya tidak dapat diandalkan.

5. Urutan Harus Sesuai Dengan Atribut Pencarian

Perhatikan data mahasiswa berikut:

NIM	Nama
1003	Andi
1001	Budi
1002	Citra

Data tersebut terurut berdasarkan nama, tetapi belum terurut berdasarkan NIM.

Jika pencarian dilakukan menggunakan NIM, binary search harus menggunakan data yang terurut berdasarkan NIM:

NIM	Nama
1001	Budi
1002	Citra
1003	Andi

Jadi, menyatakan bahwa “data sudah terurut” belum cukup. Kita harus mengetahui **berdasarkan atribut apa data tersebut diurutkan.**

6. Bagaimana Jika Data Terurut Menurun?

Binary search tetap dapat digunakan, tetapi arah pencariannya harus disesuaikan.

Perbandingan	Data menaik	Data menurun
Target lebih kecil daripada nilai tengah	Cari ke kiri	Cari ke kanan
Target lebih besar daripada nilai tengah	Cari ke kanan	Cari ke kiri

Implementasi dalam materi ini menggunakan **data terurut menaik**.

PENELUSURAN LANGKAH BINARY SEARCH

Binary search menggunakan tiga variabel utama:

Variabel	Fungsi
low	Indeks awal rentang pencarian
high	Indeks akhir rentang pencarian
mid	Indeks tengah rentang pencarian

Indeks tengah dihitung menggunakan:

$$\text{mid} = \text{low} + (\text{high} - \text{low}) // 2$$

Operator `//` digunakan agar hasil pembagian berupa bilangan bulat.

Untuk data berjumlah n elemen, nilai awalnya adalah:

```
low = 0  
high = n - 1
```

Pencarian berlangsung selama:

```
low <= high
```

Contoh mencari 60:

```
Indeks : 0  1  2  3  4  5  6  
Data   : [10, 20, 30, 40, 50, 60, 70]
```

Langkah	low	high	mid	Nilai tengah	Keputusan
1	0	6	3	40	$60 > 40$, ubah low menjadi 4
2	4	6	5	60	Target ditemukan pada indeks 5

Pencarian memeriksa dua elemen: 40 dan 60.

Pembaruan batas menggunakan $mid + 1$ atau $mid - 1$ karena elemen pada mid sudah diperiksa dan diketahui bukan target.

PERBANDINGAN EFISIENSI PENCARIAN

1. Apa Yang Dimaksud Dengan Efisiensi?

Efisiensi berkaitan dengan sumber daya yang digunakan algoritme untuk menyelesaikan pekerjaan. Dalam pembelajaran ini, efisiensi terutama ditinjau dari:

- Jumlah elemen yang diperiksa.
- Pertumbuhan jumlah langkah ketika ukuran data meningkat.
- Kebutuhan pengolahan awal, khususnya pengurutan.

Aturan pengukuran: satu pemeriksaan berarti satu elemen data diperiksa terhadap target.

Jumlah pemeriksaan elemen bukan jumlah seluruh operasi program. Dalam satu pemeriksaan, program dapat melakukan pengecekan kesamaan, pengecekan lebih kecil, dan pembaruan indeks.

2. Perbandingan pada data yang sama

Gunakan data:

[10, 20, 30, 40, 50, 60, 70]

Dengan linear search yang berhenti saat target ditemukan dan binary search yang membulatkan indeks tengah ke bawah, diperoleh:

Target	Nilai diperiksa oleh linear search	Jumlah	Nilai diperiksa oleh binary search	Jumlah
10	10	1	40, 20, 10	3
40	10, 20, 30, 40	4	40	1
70	10, 20, 30, 40, 50, 60, 70	7	40, 60, 70	3
45	10, 20, 30, 40, 50, 60, 70	7	40, 60, 50	3

Dari tabel tersebut:

1. Linear search lebih sedikit langkahnya ketika target berada di awal.
2. Binary search hanya membutuhkan satu pemeriksaan jika target berada di titik tengah awal.
3. Target yang tidak ada tetap memerlukan proses pencarian sebelum ketidakberadaannya dapat dipastikan.
4. Binary search tidak selalu memiliki langkah lebih sedikit untuk setiap target.

3. Kasus Terbaik Dan Kasus Terburuk

Algoritme	Kasus terbaik	Kasus terburuk
Linear search	Target berada pada elemen pertama	Target berada pada elemen terakhir atau tidak ditemukan
Binary search	Target berada pada elemen tengah pertama	Pencarian mempersempit rentang hingga kedalaman maksimum

Untuk linear search standar, kasus terbaik membutuhkan satu pemeriksaan, sedangkan kasus terburuk membutuhkan n pemeriksaan.

Pada binary search, rentang yang masih perlu diperiksa menyusut sekitar setengah setiap langkah.

4. KOMPLEKSITAS WAKTU

Notasi Big O menjelaskan batas pertumbuhan biaya algoritme ketika ukuran input bertambah. Notasi ini **bukan waktu dalam detik**.

Kasus	Linear search	Binary search pada array terurut
Terbaik	$O(1)$	$O(1)$
Rata-rata*	$O(n)$	$O(\log n)$
Terburuk	$O(n)$	$O(\log n)$

*Untuk pembahasan rata-rata sederhana, diasumsikan target yang ada memiliki peluang yang sama untuk berada di setiap posisi.

[high confidence] Kompleksitas tersebut berlaku untuk implementasi standar yang berhenti ketika target ditemukan, dengan akses elemen array dan perbandingan nilai berbiaya konstan.

Makna $O(1)$

Jumlah langkah pada kasus yang dibahas tidak bertambah seiring ukuran data. Contohnya, linear search menemukan target langsung pada elemen pertama.

Makna $O(n)$

Jumlah langkah tumbuh sebanding dengan jumlah elemen. Jika jumlah data menjadi dua kali lebih besar, jumlah pemeriksaan terburuk linear search juga menjadi dua kali lebih besar.

Makna $O(\log n)$

Jumlah langkah tumbuh lebih lambat karena rentang pencarian terus dibagi dua. Pada binary search, menggandakan ukuran data hanya menambah sekitar satu tingkat pencarian.

5. Pengaruh Pertambahan Ukuran Data

Untuk binary search standar pada $n \geq 1$, jumlah pemeriksaan maksimum adalah:

$$\lfloor \log_2 n \rfloor + 1$$

Tanda $\lfloor \rfloor$ berarti pembulatan ke bawah.

Jumlah data	Maksimum pemeriksaan linear search	Maksimum pemeriksaan binary search
8	8	4
16	16	5
32	32	6
64	64	7
128	128	8
1.024	1.024	11

Tabel ini membandingkan **tahap pencarian pada data yang sudah memenuhi prasyarat**, belum memasukkan biaya pengurutan.

BIAYA PENGURUTAN DAN PEMILIHAN ALGORITME

1. Data Belum Terurut Dan Hanya Dicari Sekali

Linear search dapat langsung dijalankan. Binary search memerlukan pengurutan terlebih dahulu.

Jika pengurutan yang digunakan membutuhkan $O(n \log n)$, maka:

$$\text{Pengurutan} + \text{satu binary search} = O(n \log n) + O(\log n) = O(n \log n)$$

Sementara satu linear search membutuhkan $O(n)$ pada kasus terburuk.

Karena itu, mengurutkan data hanya untuk satu pencarian **tidak otomatis lebih efisien**.

2. Data Dicari Berulang Kali

Jika data relatif tetap, pengurutan cukup dilakukan sekali. Setelah itu, binary search dapat digunakan berulang kali.

Misalkan q adalah jumlah pencarian:

Strategi	Biaya waktu
Melakukan linear search sebanyak q kali	$O(qn)$ pada kasus terburuk
Mengurutkan sekali, lalu binary search sebanyak q kali	$O(n \log n + q \log n)$, dengan pengurutan $O(n \log n)$

Semakin banyak pencarian, semakin besar peluang biaya awal pengurutan terimbangi. Titik peralihannya bergantung pada ukuran data, implementasi, dan kondisi pencarian.

3. Data Sering Berubah

Jika data baru ditambahkan tanpa mempertahankan urutan, prasyarat binary search dapat hilang.

Contoh:

Sebelum ditambah : [10, 20, 30, 40]

Setelah ditambah : [10, 20, 30, 40, 15]

Data perlu disusun kembali atau elemen baru ditempatkan pada posisi yang menjaga urutan. Pemeliharaan urutan tersebut juga memiliki biaya.

4. Panduan Pemilihan

Kondisi	Pilihan yang layak dipertimbangkan	Alasan
Data belum terurut, satu pencarian	Linear search	Langsung digunakan tanpa pengurutan
Array sudah terurut	Binary search	Rentang pencarian cepat menyusut
Data relatif tetap, pencarian berulang	Urutkan sekali lalu binary search	Biaya pengurutan dapat dimanfaatkan untuk banyak pencarian
Data sering berubah tanpa menjaga urutan	Linear search atau evaluasi ulang strategi	Binary search tidak boleh langsung digunakan
Target sering berada di awal	Linear search dapat menguntungkan	Dapat berhenti setelah sedikit pemeriksaan

Implementasi Python

Kedua fungsi berikut mengembalikan:

```
(indeks_target, jumlah_pemeriksaan)
```

Nilai indeks -1 berarti target tidak ditemukan. Nilai tersebut digunakan sebagai **penanda hasil**, bukan untuk mengakses elemen terakhir dalam list Python.

1. Linear search

```
def linear_search(data, target):  
    pemeriksaan = 0  
  
    for indeks in range(len(data)):  
        pemeriksaan += 1  
  
        if data[indeks] == target:  
            return indeks, pemeriksaan  
  
    return -1, pemeriksaan
```

Penjelasan:

- pemeriksaan bertambah ketika satu elemen diperiksa.
- return langsung menghentikan fungsi ketika target ditemukan.
- Jika seluruh elemen sudah diperiksa tanpa kecocokan, fungsi mengembalikan -1.

2. Binary search

```
def binary_search(data, target):  
    # Prasyarat: data sudah terurut menaik.  
    low = 0  
    high = len(data) - 1  
    pemeriksaan = 0  
  
    while low <= high:  
        mid = low + (high - low) // 2  
        pemeriksaan += 1  
  
        if data[mid] == target:  
            return mid, pemeriksaan  
        elif target < data[mid]:  
            high = mid - 1  
        else:  
            low = mid + 1  
  
    return -1, pemeriksaan
```

Fungsi tidak mengurutkan atau memeriksa keterurutan data secara otomatis. Pemanggil fungsi bertanggung jawab memenuhi prasyarat tersebut.

Memeriksa seluruh data untuk memastikan keterurutan membutuhkan $O(n)$. Jika pemeriksaan ini dilakukan pada setiap pemanggilan, biaya keseluruhan pemanggilan tidak lagi hanya $O(\log n)$.

3. Program Perbandingan

Jalankan kode berikut setelah kedua fungsi didefinisikan:

```
data = [10, 20, 30, 40, 50, 60, 70]
daftar_target = [10, 40, 70, 45]

print("Data:", data)

for target in daftar_target:
    indeks_linear, jumlah_linear = linear_search(data, target)
    indeks_binary, jumlah_binary = binary_search(data, target)

    print(f"\nTarget: {target}")
    print(
        f"Linear search: indeks={indeks_linear}, "
        f"pemeriksaan={jumlah_linear}"
    )
    print(
        f"Binary search: indeks={indeks_binary}, "
        f"pemeriksaan={jumlah_binary}"
    )
```

Hasil yang diharapkan:

Data: [10, 20, 30, 40, 50, 60, 70]

Target: 10

Linear search: indeks=0, pemeriksaan=1

Binary search: indeks=0, pemeriksaan=3

Target: 40

Linear search: indeks=3, pemeriksaan=4

Binary search: indeks=3, pemeriksaan=1

Target: 70

Linear search: indeks=6, pemeriksaan=7

Binary search: indeks=6, pemeriksaan=3

Target: 45

Linear search: indeks=-1, pemeriksaan=7

Binary search: indeks=-1, pemeriksaan=3

PENGUJIAN KONDISI KHUSUS

Program perlu diuji pada kondisi yang berbeda, bukan hanya ketika target berhasil ditemukan.

Kondisi	Contoh data	Target	Perilaku yang diharapkan
Data kosong	[]	10	Tidak ditemukan; 0 pemeriksaan
Satu elemen, cocok	[10]	10	Indeks 0; 1 pemeriksaan
Satu elemen, tidak cocok	[10]	20	Tidak ditemukan; 1 pemeriksaan
Target di awal	[10, 20, 30]	10	Ditemukan
Target di akhir	[10, 20, 30]	30	Ditemukan
Target tidak ada	[10, 20, 30]	25	Tidak ditemukan
Ada nilai duplikat	[10, 20, 20, 20, 30]	20	Salah satu indeks yang memuat target ditemukan

Pada data duplikat, linear search di atas mengembalikan **kemunculan pertama**. Binary search standar mengembalikan kecocokan yang ditemui dalam prosesnya, sehingga belum tentu kemunculan pertama.

Perbedaan indeks pada data duplikat tidak otomatis berarti salah. Periksa apakah nilai pada indeks tersebut sama dengan target. Jika tugas menuntut kemunculan pertama, binary search perlu dimodifikasi.

KESALAHAN YANG PERLU DIHINDARI

Kesalahan	Akibat	Perbaikan
Binary search pada data belum terurut	Target yang ada dapat tidak ditemukan	Pastikan data terurut berdasarkan atribut pencarian
Arah kiri dan kanan tertukar	Rentang yang dipilih salah	Sesuaikan aturan dengan urutan menaik atau menurun
Menggunakan / untuk indeks tengah di Python	Menghasilkan bilangan pecahan	Gunakan //
Menggunakan while low < high pada implementasi ini	Satu kandidat terakhir bisa terlewat	Gunakan while low <= high
Memperbarui batas dengan low = mid	Rentang dapat tidak menyusut	Gunakan low = mid + 1
Linear search tidak berhenti setelah cocok	Seluruh elemen tetap diperiksa	Gunakan return untuk pencarian satu kecocokan
Menganggap indeks hasil pengurutan sebagai indeks awal	Posisi asal salah dilaporkan	Simpan identitas atau indeks asal jika diperlukan
Menganggap langkah lebih sedikit pasti berarti waktu terukur lebih singkat	Kesimpulan pengujian terlalu jauh	Bedakan jumlah pemeriksaan dengan waktu eksekusi nyata

Untuk latihan awal, jumlah pemeriksaan lebih mudah dibandingkan secara konsisten daripada waktu eksekusi pada data yang sangat kecil. Pengukuran waktu dipengaruhi perangkat, aktivitas sistem, dan biaya pelaksanaan program.

Praktikum 1 — Memeriksa Prasyarat

Perhatikan data berikut:

```
data_a = [12, 24, 36, 48, 60]
data_b = [36, 12, 60, 24, 48]
data_c = [60, 48, 36, 24, 12]
data_d = [12, 24, 24, 36, 48]
```

Tugas:

1. Tentukan kondisi urutan setiap data.
2. Tentukan data yang dapat langsung digunakan oleh fungsi binary search pada materi ini.
3. Jelaskan mengapa data terurut menurun memerlukan penyesuaian kode.
4. Jelaskan apakah nilai duplikat melanggar prasyarat keterurutan.

Praktikum 2 — Menelusuri Langkah

Gunakan:

```
data = [5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75]
```

Cari target 5, 40, 75, dan 42.

Untuk linear search, catat:

Langkah	Indeks	Nilai diperiksa	Hasil perbandingan
...	...	...	...

Untuk binary search, catat:

Langkah	low	high	mid	Nilai tengah	Keputusan
...	...	...	...	...	...