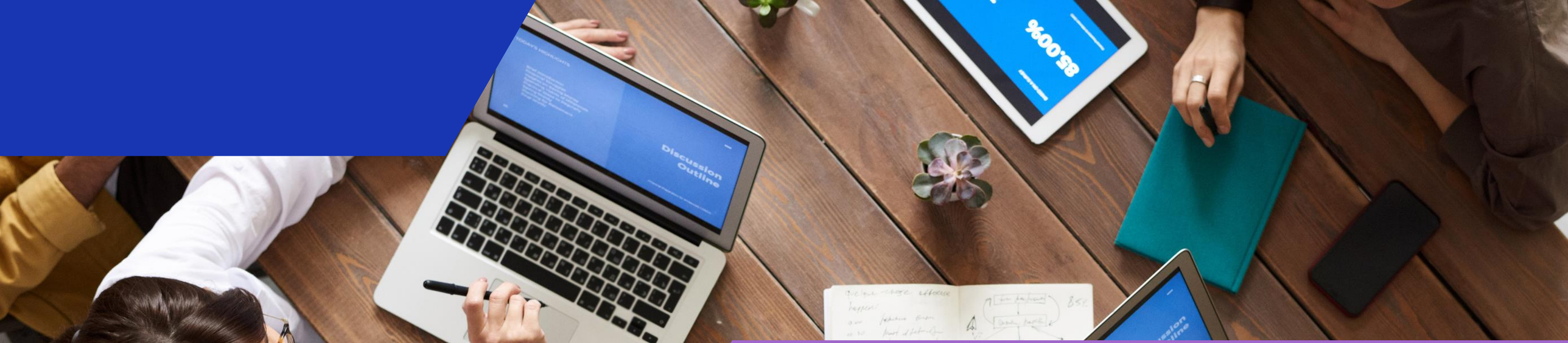


SEARCHING (PENCARIAN)

Pertemuan 12-13

Searching (Pencarian)

- Pencarian (Searching) merupakan proses yang fundamental dalam pemrograman guna menemukan data (nilai) tertentu di dalam sekumpulan data yang bertipe sama.
- Fungsi pencarian itu sendiri adalah untuk memvalidasi (mencocokkan) data.

Sebagai contoh,

- Untuk menghapus atau mengubah sebuah data di dalam sekumpulan nilai, langkah pertama yang harus di tempuh adalah mencari data tersebut, lalu menghapus atau mengubahnya.
- Contoh lain adalah penyisipan data ke dalam kumpulan data, jika data telah ada, maka data tersebut tidak akan disisipkan, selainnya akan disisipkan ke kumpulan data tersebut.

Searching (Pencarian)

Ada beberapa metode mencari data di dalam sekumpulan data yang bertipe sama,yaitu:

- Metode Pencarian Beruntun (Sequential Search)
- Metode Pencarian Bagi dua (Binary Search)

Sequential Search

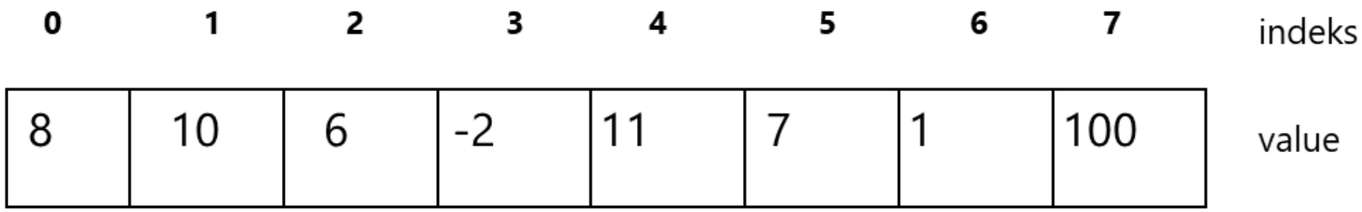
- Metode pencarian beruntun atau linear atau sequential search.
- Adalah suatu teknik pencarian data yang akan menelusuri tiap elemen satu per-satu dari awal sampai akhir.
- Suatu deret data dapat disimpan dalam bentuk array maupun linked list.

Case

- Best case : jika data yang dicari terletak di indeks array terdepan (elemen array pertama) sehingga waktu yang dibutuhkan untuk pencarian data sangat sebentar (minimal).
- Worst case : jika data yang dicari terletak di indeks array terakhir (elemen array terakhir) sehingga waktu yang dibutuhkan untuk pencarian data sangat lama (maksimal).

Case

- Misalnya terdapat array satu dimensi sebagai berikut:



- Kemudian program akan meminta data yang akan dicari, misalnya 6.
- Iterasi :
- 6 = 8 (tidak!)
- 6 = 10 (tidak!)
- 6 = 6 (Ya!) => output : 2 (index)

Algoritma

1. i ← 0
2. ketemu ← false
3. Selama (tidak ketemu) dan (i < N) kerjakan baris 4
4. Jika (Data[i] = key) maka
   ketemu ← true
   jika tidak
   i ← i + 1
5. Jika (ketemu) maka
   i adalah indeks dari data yang dicari
   jika tidak
   data tidak ditemukan

Q & A

- Problem: Apakah cara di atas efisien? Jika datanya ada 10000 dan semua data dipastikan unik?
- Solution: Untuk meningkatkan efisiensi, seharusnya jika data yang dicari sudah ditemukan maka perulangan harus dihentikan!
- Hint: Gunakan break!
- Question: Bagaimana cara menghitung ada berapa data dalam array yang tidak unik, yang nilainya sama dengan data yang dicari oleh user?
- Hint: Gunakan variabel counter yang nilainya akan selalu bertambah jika ada data yang ditemukan!

Binary Search

- Metode ini diterapkan pada sekumpulan data yang sudah terurut (menaik atau menurun).
- Metode ini lebih cepat dibandingkan metode pencarian beruntun. Data yang sudah terurut menjadi syarat mutlak untuk menggunakan metode ini.
- Konsep dasar metode ini adalah membagi 2 jumlah elemennya, dan menentukan apakah data yang berada pada elemen paling tengah bernilai sama, lebih dari atau kurang dari nilai data yang akan dicari.

Binary Search

- Jika bernilai sama, maka langsung data yang dicari ditemukan.
- Jika data di elemen terurut naik, maka jika data yang berada di tengah kurang dari data yang dicari, maka pencarian selanjutnya berkisar di elemen tengah ke kanan, dan begitu seterusnya sampai ketemu atau tidak sama sekali.
- Dan sebaliknya untuk nilai data yang berada di tengah lebih dari data yang dicari, maka pencarian selanjutnya berkisar di elemen tengah ke kiri, dan begitu seterusnya sampai ketemu atau tidak sama sekali. Dan demikian sebaliknya untuk data yang terurut menurun. Dalam hal ini tentukan indeks paling awal dan indeks paling akhir, untuk membagi 2 elemen tersebut.

Algoritma

- Indeks awal = i, dimana nilai i, pada awalnya bernilai 0;
- Indeks akhir = j, dimana nilai j, pada awalnya bernilai sama dengan jumlah elemen.
- Langkah-langkah untuk metode pencarian bagi dua.
- Asumsikan data terurut secara horizontal dari indeks 0 sampai n-1, untuk menggunakan istilah kanan dan kiri.
- Misalkan kumpulan data yang berjumlah n adalah larik L, dan data yang akan dicari adalah X.
- Tentukan nilai indeks awal i = 0 dan indeks akhir j = n-1.

Algoritma

4.Tentukan apakah data terurut menurun atau menaik dengan menggunakan membandingkan apakah elemen paling kiri L[0] lebih dari atau kurang dari elemen paling kanan L[n-1].

- Jika data di elemen paling kiri L[0] > data di elemen paling kanan L[n-1], maka data terurut menurun.
- Jika data di elemen paling kiri L[0] < data di elemen paling kanan L[n-1], maka data terurut menaik.

5.Asumsikan bahwa data terurut menaik (tergantung hasil dari nomor 3).

6.Misalkan variabel k adalah indeks paling tengah, diperoleh dengan rumus: k = (i + j) div 2.

Algoritma

7.Periksa, jika L[k] = X, maka data dicari langsung ketemu di elemen k.
8.Jika nomor 7 tidak terpenuhi, periksa jika L[k] < X, maka pencarian berikutnya dilakukan di sisi kanan indeks k, lakukan proses seperti pada nomor 6, dimana nilai indeks i sekarang sama dengan nilai indeks k sebelumnya.

- i = k.
- k = (i + j) div 2.
- Dan seterusnya sampai nilai X dicari ketemu atau tidak sama sekali.

Algoritma

9.Jika nomor 8 tidak terpenuhi, maka tidak pasti nilai L[k] > X, maka pencarian berikutnya dilakukan di sisi kiri indeks k, lakukan proses seperti pada nomor 6, dimana nilai indeks j sekarang sama dengan nilai indeks k sebelumnya.

- j = k.
- k = (i + j) div 2.
- Dan seterusnya sampai nilai X dicari ketemu atau tidak sama sekali.

10.Jika data terurut menurun, maka tukar kondisi yang ada di nomor 8 dan 9.

Contoh

Diberikan 10 data terurut L[10] = {12,14,15,17,23,25,45,67,68,70}. Cari nilai X = 14 dielemen tersebut.

Solusi :

- Menentukan apakah data terurut menaik atau menurun.
- L[0] = 12
- L[9] = 70
- Karena L[0] < L[9], maka data tersebut terurut menaik.

Contoh

2.Misal indeks paling kiri adalah i = 0 dan indeks paling kanan adalah j = 9, maka indeks tengahnya adalah :

- k = (i + j) div 2
- =(0 + 9) div 2
- = 4.
- Elemen tengah sekarang adalah 4 dengan L[4] = 23.

Contoh

3.Karena data di indeks tengah lebih dari nilai data yang dicari (L[4] > X), maka pencarian berikutnya dilakukan pada sisi kiri indeks k, maka nilai j sekarang sama dengan k, lalu lakukan proses sama seperti nomor 2.

- j = k
- = 4
- k = (i + j) div 2
- = (0 + 4) div 2
- = 2.
- Elemen tengah sekarang adalah 2 dengan L[2] = 15.

Contoh

4.Karena data di indeks tengah lebih dari nilai data yang dicari (L[2] > X), maka pencarian berikutnya dilakukan pada sisi kiri indeks k, maka nilai j sekarang sama dengan k, lalu lakukan proses sama seperti nomor 2.

- j = k
- = 2
- k = (i + j) div 2
- = (0 + 2) div 2
- = 1.
- Elemen tengah sekarang adalah 1 dengan L[1] = 14.

Contoh

key < 30

	bawah							tengah							atas
index	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]
list	5	8	12	15	17	23	26	30	34	38	42	54	64	78	81

key > 15

	bawah			tengah				atas
index	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
list	5	8	12	15	17	23	26	30

key < 23

	bawah		tengah		atas
index	[3]	[4]	[5]	[6]	[7]
list	15	17	23	26	30

key == 26

	bawah	tengah	atas
index	[5]	[6]	[7]
list	23	26	30