Modul 4

Pencarian

Algoritma pencarian, *search algorithm*, adalah salah satu algoritma yang paling sering dijalankan oleh sistem komputer maupun penggunanya. Di modul ini akan kita bahas beberapa algoritma pencarian. Di sini kita akan membahas pencarian pada struktur data satu dimensi yang *iterable*, dan tertarik pada dua kasus: yang elemenya tidak urut dan yang sudah terurutkan.

4.1 Linear Search

Solusi paling mudah untuk pencarian di suatu daftar adalah pencarian lurus, yakni *linear search*/*sequential search*[1]. Cara ini akan mengiterasi sepanjang daftar itu, satu item demi satu item sampai item yang dicari ditemukan atau semua item sudah diperiksa. Di Python, menemukan suatu item di sebuah list –atau apapun yang *iterable*– dapat dilakukan dengan kata kunci `in`:

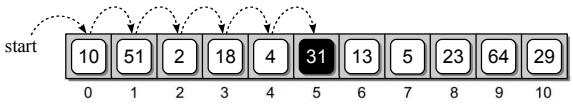
```
if target in arrayTempatYangDicari:
    print("targetnya terdapat di array itu.")
else:
    print("targetnya tidak terdapat di array itu.")
```

Penggunaan katakunci `in` merupakan fasilitas yang memudahkan pencarian, namun itu juga menyembunyikan kerja internalnya. Di level bawahnya, operator `in` ini diimplementasikan menggunakan `linear search`. Misal kita mempunyai array ***tidak urut*** yang didefinisikan memakai list dengan perintah berikut:

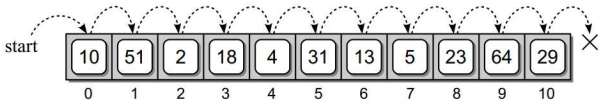
```
A = [10, 51, 2, 18, 4, 31, 13, 5, 23, 64, 29]
```

Untuk menentukan apakah, misal, nilai 31 terdapat pada array itu, pencarian dilakukan dengan melihat elemen pertamanya. Karena di elemen pertama tidak ketemu, pencarian dilanjutkan dengan elemen kedua. Demikian seterusnya sampai angka 31 ditemukan pada elemen keenam.

[1]Di beberapa kesempatan di modul ini, kita menerjamahkan *linear search* secara bebas dengan "pencarian lurus"



(a) Mencari 31



(b) Mencari 8

Gambar 4.1: Pencarian linear pada sebuah array yang tidak urut. (a) Item yang dicari ketemu. (b) Item yang dicari tidak ketemu.

Lihat Gambar 4.1(a)[2]. Bagaimana kalau item yang dicari tidak ada di array itu? Misalnya kita ingin mencari nilai 8 di array itu. Seperti sebelumnya, pemeriksaan dimulai dari elemen pertama, tapi kali ini setiap dan semua item dicocokkan dengan angka 8. Baru ketika elemen terakhir diperiksa, diketahui bahwa tidak ada angka 8 di array itu, seperti diilustrasikan di Gambar 4.1(b).

Ok? K. Nah, program berikut akan mencari `target` di data yang ada di `wadah`. Ketik dan cobalah.

```
1 def cariLurus( wadah, target ):
2     n = len( wadah )
3     for i in range( n ):
4         if wadah[i] == target:
5             return True
6     return False
```

Mari kita coba di Python Shell, seperti ini

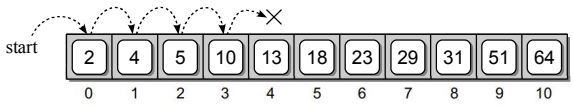
```
>>> A = [10, 51, 2, 18, 4, 31, 13, 5, 23, 64, 29]
>>> cariLurus(A,31)
True
>>> cariLurus(A,8)
False
```

Array yang urut tentu saja bisa juga dicari item-itemnya dengan linear search ini, namun dengan suatu tambahan keuntungan bahwa jika item yang diperiksa sudah 'kelewatan' dibandingkan target yang dicari, maka pencarian bisa dihentikan. Lihat Gambar 4.2.

Pencarian Lurus untuk Objek Buatan Sendiri

Di Modul 2 kita telah belajar bagaimana membuat objek dari class yang kita buat sendiri. Kita juga dapat mencari sesuatu yang spesifik di daftar objek-objek ini. Misal kita mempunyai data mahasiswa yang diinput seperti berikut (ketiklah):

[2]Gambar-gambar di bab ini diambil dari Rance D. Necaise, *Data Structures and Algorithms Using Python*, John Wiley and Sons, 2011.



Gambar 4.2: Linear search pada data yang sudah urut. Di sini kita mencoba mencari angka 8; ketika satu persatu diperiksa dari yang paling kecil dan lalu ketemu angka 10, pencarian dihentikan.

```
c0 = MhsTIF('Ika',10,'Sukoharjo', 240000)
c1 = MhsTIF('Budi',51,'Sragen', 230000)
c2 = MhsTIF('Ahmad',2,'Surakarta', 250000)
c3 = MhsTIF('Chandra',18,'Surakarta', 235000)
c4 = MhsTIF('Eka',4,'Boyolali', 240000)
c5 = MhsTIF('Fandi',31,'Salatiga', 250000)
c6 = MhsTIF('Deni',13,'Klaten', 245000)
c7 = MhsTIF('Galuh',5,'Wonogiri', 245000)
c8 = MhsTIF('Janto',23,'Klaten', 245000)
c9 = MhsTIF('Hasan',64,'Karanganyar', 270000)
c10 = MhsTIF('Khalid',29,'Purwodadi', 265000)
##
## Lalu kita membuat daftar mahasiswa dalam bentuk list seperti ini:
##
Daftar = [c0, c1, c2, c3, c4, c5, c6, c7, c8, c9, c10]
```

Kita akan mencari, sebagai contoh, mahasiswa yang beralamat Klaten. Dan, sebagai tambahan, diminta juga agar kita mencetak semua mahasiswa yang berasal dari Klaten, tidak hanya sekedar 'ketemu' dan 'tidak ketemu'. Bagaimana caranya?

Ketik dan larikan yang berikut ini

```
target = 'Klaten'
for i in Daftar:
    if i.kotaTinggal == target:
        print(i.nama + ' tinggal di ' + target)
```

Pencarian Lurus di Linked-List

Pencarian di linked-list dilakukan dengan mengunjungi satu-persatu elemen yang ada di situ. Kamu bisa mengacu pada pencetakan item di linked list yang dipaparkan di halaman 35.

Mencari nilai yang terkecil pada array yang tidak urut

Alih-alih mencari suatu target dengan nilai tertentu, terkadang kita diminta mencari nilai yang terkecil[3] dari suatu array. Seperti sebelumnya, kita akan melakukan pencarian lurus. Tapi kali ini kita memegang nilai terkecil yang terus diperbandingkan tiap kali kita mengunjungi elemen-elemen di array itu.

Untuk memulai loop ini, kita awalnya menganggap elemen pertama sebagai yang terkecil. Ketika mengunjungi elemen kedua, kita bandingkan nilai terkecil tadi dengan elemen ini. Jika elemen

[3] atau yang terbesar. Idenya sama saja.

ini lebih kecil, maka 'yang terkecil' tadi diubah nilainya. Ini terus dilakukan sampai elemen yang terakhir. Program berikut mengilustrasikannya.

```
1  def cariTerkecil(kumpulan):
2      n = len(kumpulan)
3      # Anggap item pertama adalah yang terkecil
4      terkecil = kumpulan[0]
5      # tentukan apakah item lain lebih kecil
6      for i in range(1,n):
7          if kumpulan[i] < terkecil:
8              terkecil = kumpulan[i]
9
10     return terkecil   #kembalikan yang terkecil
```

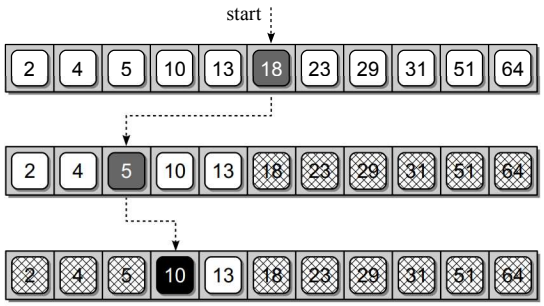
Sekarang,

- Bagaimanakah programnya jika kita ingin mencari mahasiswa (dari class MhsTIF di atas) yang uang sakunya terkecil?
- Bagaimana kalau yang terbesar?
- Bagaimanakah programnya jika kita ingin mencari *semua* mahasiswa yang uang sakunya kurang dari 250 ribu?
- Yang lebih dari 250 ribu?

4.2 Binary search

Jika elemen yang ada di suatu list ***sudah urut***, kita bisa memakai algoritma yang lebih efisien, yakni *binary search*. Algoritma ini bisa dijelaskan seperti di bawah ini — dengan asumsi array-nya sudah urut dari kecil ke besar.

- Dari suatu array yang urut kita ingin mencari suatu item dengan nilai tertentu.
- Ambil nilai di tengah-tengah array itu. Jika itu yang target yang dicari: selesai.
- Bandingkan nilai yang di tengah itu dengan target yang dicari.
  - Jika nilai tengah itu terlalu besar (berarti yang dicari berada di sebelah kirinya), abaikan semua yang di sebelah kanannya dan lalu cari targetnya di array yang tinggal separuh kiri itu.
  - Jika nilai tengah itu terlalu kecil (berarti yang dicari berada di sebelah kanannya), abaikan semua yang sebelah kirinya dan lalu cari targetnya di array yang tinggal separuh kanan itu.



Gambar 4.3: Binary search. Di array yang sudah urut (dari kecil ke besar) di atas kita mencari angka 10, yakni targetnya. Pencarian dimulai dari tengah, dan bertemu angka 18. Angka ini lebih besar dari target, berarti yang dicari berada sebelah kiri, dan yang di sebelah kanan diabaikan. Sekarang ambil yang tengah lagi: ketemu angka 5. Angka ini lebih kecil dari target, jadi abaikan sebelah kirinya dan cari sebelah kanannya. Akhirnya ketemu angka 10.

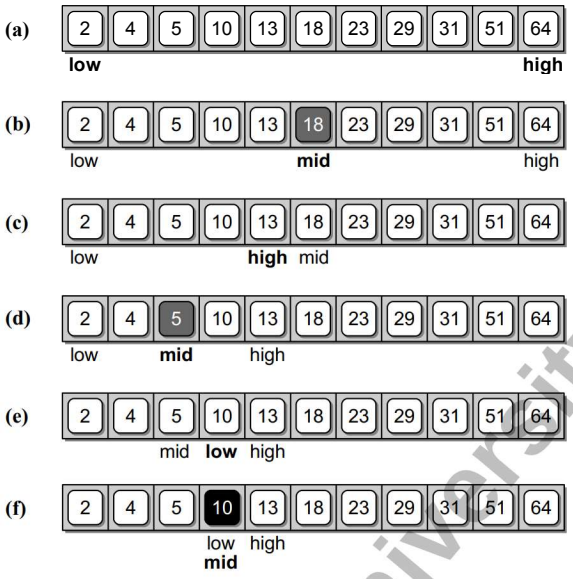
Lihat Gambar 4.3 untuk ilustrasinya. Program di bawah adalah salah satu bentuk implementasi binary search.

```
def binSe(kumpulan, target):
    # Mulai dari seluruh runtutan elemen
    low = 0
    high = len(kumpulan) - 1

    # Secara berulang belah runtutan itu menjadi separuhnya
    #   sampai targetnya ditemukan
    while low <= high:
         # Temukan pertengahan runtut itu
        mid = (high + low) // 2
         # Apakah pertengahannya memuat target?
        if kumpulan[mid] == target:
            return True
         # ataukah targetnya di sebelah kirinya?
        elif target < kumpulan[mid]:
            high = mid - 1
         # ataukah targetnya di sebelah kanannya?
        else:
            low = mid + 1
    # Jika runtutnya tidak bisa dibelah lagi, berarti targetnya tidak ada
    return False
```

Program di atas akan mengembalikan `True` jika targetnya ditemukan dan `False` jika targetnya tidak ditemukan. Gambar 4.4 mengilustrasikan kejadiannya ketika kita mencari angka 10 di array di Gambar 4.3.

Dapatkah kamu mengubah programnya agar dia mengembalikan index-nya kalau targetnya ditemukan, dan mengembalikan `False` kalau targetnya tidak ditemukan?



Gambar 4.4: Langkah-langkah yang dilakukan oleh algoritma *binary search* dalam mencari angka 10: (a) rentang awal elemen-elemennya, (b) menentukan lokasi pertengahannya, (c) mengeliminasi separuh atas, (d) pertengahannya separuh bawah, (e) mengeliminasi perempat bawah, (f) menemukan target.

4.3 Soal-soal untuk Mahasiswa

Sebelum mengerjakan soal-soal di bawah, kerjakan dulu latihan-latihan di atas.

1. Buatlah suatu fungsi pencarian yang, alih-alih mengembalikan `True`/`False`, mengembalikan *semua* index lokasi elemen yang dicari. Jadi, misal pada list daftar mahasiswa di halaman 40 kita mencari mahasiswa yang berasal dari Klaten, kita akan mendapatkan `[6, 8]`. Kalau yang dicari tidak ditemukan, fungsi ini akan mengembalikan list kosong.
2. Dari list daftar mahasiswa di atas, buatlah fungsi untuk menemukan uang saku yang terkecil di antara mereka.
3. Ubah program di atas agar mengembalikan objek mahasiswa yang mempunyai uang saku terkecil. Jika ada lebih dari satu mahasiswa yang uang sakunya terkecil, semua objek mahasiswa itu dikembalikan.
4. Buatlah suatu fungsi yang mengembalikan semua objek mahasiswa yang uang sakunya kurang dari 250000.
5. Buatlah suatu program untuk mencari suatu item di sebuah linked list.
6. Binary search. Ubahlah fungsi `binSe` di halaman 43 agar mengembalikan index lokasi elemen yang ditemukan. Kalau tidak ketemu, akan mengembalikan `False`.
7. Binary search. Ubahlah fungsi `binSe` itu agar mengembalikan semua index lokasi elemen yang ditemukan. Contoh: mencari angka 6 pada list `[2, 3, 5, 6, 6, 6, 8, 9, 9, 10,`

`11, 12, 13, 13, 14]` akan mengembalikan `[3, 4, 5]`. Karena sudah urut, "tinggal melihat kiri dan kanannya".

8. Pada permainan tebak angka yang sudah kamu buat di Modul 1 (soal nomer 12, halaman 16), kalau angka yang harus ditebak berada di antara 1 dan 100, seharusnya maksimal jumlah tebakan adalah 7. Kalau antara 1 dan 1000, maksimal jumlah tebakan adalah 10. Mengapa seperti itu? Bagaimanakah polanya?