Searching(Pencarian)

Pencarian (*searching*) merupakan proses yang fundamental dalam pengolahan data. Proses pencarian adalah menemukan nilai (data) tertentu di dalam sekumpulan data yang bertipe sama (baik bertipe dasar atau bertipe bentukan). Sebagai contoh, untuk mengubah (*update*) data tertentu, langkah pertama yang harus dilakukan adalah mencari keberadaan data tersebut di dalam kumpulannya. Jika data yang dicari ditemukan, maka data tersebut dapat diubah nilainya dengan data yang baru. Aktivitas awal yang sama juga dilakukan pada proses penambahan (*insert*) data baru. Proses penambahan data dimulai dengan mencari apakah data yang akan ditambahkan sudah terdapat di dalam kumpulan. Jika sudah ada dan mengasumsikan tidak boleh ada duplikasi data maka data tersebut tidak perlu ditambakan, tetapi jika belum ada, maka tambahkan.

Spesifikasi Masalah Pencarian

Pertama-tama kita menspesifikasikan masalah pencarian di dalam larik. Di dalam Bab 15 ini kita mendefinisikan persoalan pencarian secara umum sebagai berikut: *Diberikan larik L yang sudah terdefinisi elemen-elemennya, dan x adalah elemen yang bertipe sama dengan elemen larik L. Carilah x di dalam larik L.*

Hasil atau keluaran dari persoalan pencarian dapat bermacam-macam, bergantung pada spesifikasi rinci dari persoalan tersebut, misalnya:

a. Pencarian hanya untuk memeriksa keberadaan x. Keluaran yang diinginkan misalnya pesan (*message*) bahwa x ditemukan atau tidak ditemukan di dalam larik.

Contoh:

```
write('ditemukan!') atau
write('tidak ditemukan!')
```

b. Hasil pencarian adalah indeks elemen larik. Jika x ditemukan, maka indeks elemen larik tempat x berada diisikan ke dalam *idx*. Jika x tidak terdapat di dalam larik L, maka *idx* diisi dengan harga khusus, misalnya -1.
Contoh: Perhatikan larik di bawah ini:

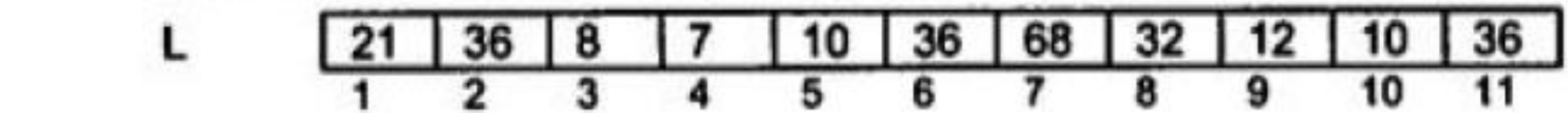
L	21	36	8	7	10	36	68	32	12	10	36
	1	2	3	4	5	6	7	8	9	10	11

Misalkan $x = 68$, maka $idx = 7$, dan bila $x = 100$, maka $idx = -1$.

c. Hasil pencarian adalah sebuah nilai *boolean* yang menyatakan status hasil pencarian. Jika *s* ditemukan, maka sebuah peubah bertipe *boolean*, misalnya "ketemu", diisi dengan nilai true, sebaliknya "ketemu" diisi dengan nilai false. Hasil pencarian ini selanjutnya disimpulkan pada bagian pemanggilan prosedur.

Contoh: Perhatikan larik *L* di atas:
Misalkan $x = 68$, maka ketemu = true, dan bila $x = 100$, maka ketemu = false.

Hal lain yang harus diperjelas dalam masalah pencarian adalah mengenai duplikasi data. Apabila X yang dicari terdapat lebih dari satu banyaknya di dalam larik L, maka hanya X yang pertama kali ditemukan yang diacu algoritma pencarian selesai. Sebagai contoh, perhatikan larik di bawah ini:



Larik L memiliki tiga buah nilai 36. Bila $x = 36$, maka algoritma pencarian selesai ketika X ditemukan pada elemen ke-2 dan menghasilkan $idx = 2$ (atau menghasilkan ketemu = true jika mengacu pada keluaran pencarian jenis c). Elemen 36 lainnya tidak dipertimbangkan lagi dalam pencarian.

Algoritma pencarian yang akan dibahas di dalam Bab 15 ini adalah:

1. Algoritma pencarian beruntun (*sequential search*).
2. Algoritma pencarian bagidua (*binary search*).

Untuk masing-masing algoritma, tipe larik yang digunakan didefinisikan di bagian deklarasi global seperti di bawah ini. Larik *L* bertipe *LarikInt*.

```
{ kamus data global }

DEKLARASI
  const Nmaks = 100     {jumlah maksimum elemen larik }
  type LarikInt = array [1..Nmaks] of integer
```

Algoritma Pencarian Beruntun

Algoritma pencarian yang yang paling sederhana, yaitu metode pencarian beruntun (*sequential search*). Nama lain algoritma pencarian beruntun adalah pencarian lurus (*linear search*).

Pada dasarnya, algoritma pencarian beruntun adalah proses membandingkan setiap elemen larik satu per satu secara beruntun, mulai dari elemen pertama, sampai elemen yang dicari ditemukan, atau seluruh elemen sudah diperiksa.

Contoh 15.1. Perhatikan larik L di bawah ini dengan $n = 6$ elemen:

13	16	14	21	76	15
1	2	3	4	5	6

```
DEKLARASI
   i : integer    { pencatat indeks larik }

ALGORITMA:
   i ← 1
   while (i < n ) and (L[i] ≠ x) do
      i ← i + 1
   endwhile
   { i = n or L[i] = x }

   if L[i] = x then          { x ditemukan }
     ketemu ← true
   else
     ketemu ← false          { x tidak ada di dalam larik L }
   endif
```

Algoritma 15.2 Prosedur pencarian beruntun (versi 1, hasil pencarian: *boolean*)

(ii) Fungsi pencarian beruntun:

```
function SeqSearch1(input L : LarikInt, input n : integer,
                    input x : integer) → boolean
{ Mengembalikan nilai true jika x ditemukan di dalam larik L[1..n],
atau nilai false jika x tidak ditemukan. }

DEKLARASI
   i : integer    { pencatat indeks larik }

ALGORITMA:
    i ← 1
    while (i < n ) and (L[i] ≠ x) do
       i ← i + 1
    endwhile
    { i = n or L[i] = x }

    if L[i] = x then     { x ditemukan }
      return true
    else
      return false       { x tidak ada di dalam larik L }
    endif
```

Algoritma 15.3 Fungsi pencarian beruntun (versi 1, hasil pencarian: *boolean*)

Contoh program utama yang memanggil prosedur SeqSearch1:

```
PROGRAM Pencarian
{ Program untuk mencari nilai tertentu di dalam larik }

DEKLARASI
   const Nmaks = 100  { jumlah maksimum elemen larik }
   type LarikInt : array[1..Nmaks] of integer

   L : LarikInt
   x : integer        { elemen yang dicari }
   found : boolean    { true jika x ditemukan, false jika tidak }
   n : integer        { ukuran larik }

   procedure BacaLarik(output L : LarikInt, input n : integer )
   { Mengisi elemen larik L[1..n] dengan nilai yang dibaca dari
piranti masukan }

   procedure SeqSearch1(input L : LarikInt, input n : integer,
                        input x :integer, output ketemu : boolean)
   { Mencari keberadaan nilai x di dalam larik L[1..n]. }


ALGORITMA:
   read(n)           { tentukan banyaknya elemen larik }
   BacaLarik(L, n){ baca elemen-elemen larik L)
   read(x)           { baca nilai yang dicari }
   SeqSearch1(L, n, x, found)    { cari }
   if found then  { found = true }
      write(x, ' ditemukan!')
   else
      write(x, ' tidak ditemukan!' )
   endif
```

(i) Prosedur pencarian beruntun:

```
procedure SeqSearch2(input L : larik, input n : integer, input x : integer,
                     output idx : integer)

{ Mencari keberadaan nilai x di dalam larik L[1..n]. }
{ K.Awal:  x dan elemen-elemen larik L[1..n] sudah terdefinisi.}
{ K.Akhir: idx berisi indeks larik L yang berisi nilai x. Jika x tidak
  ditemukan, maka idx diisi dengan nilai -15. }

DEKLARASI
   i : integer    { pencatat indeks larik }
```

```
ALGORITMA:
   i ← 1
   while (i < n ) and (L[i] ≠ x) do
      i ← i + 1
   endwhile
   { i = n or L[i] = x }

   if L[i] = x then        { x ditemukan }
     idx ← i
   else
     idx ← -1
   endif
```

2. Hasil pencarian: sebuah peubah *boolean* yang bernilai *true* bila *x* ditemukan atau bernilai *false* bila *x* tidak ditemukan.

(i) Prosedur pencarian beruntun:

```
procedure SeqSearch3(input L : LarikInt, input n : integer,
                     input x : integer, output ketemu : boolean)
{ Mencari keberadaan nilai x di dalam larik L[1..n]. }
{ K.Awal: nilai x, n, dan elemen larik L[1..n] sudah terdefinisi. }
{ K.Akhir: ketemu bernilai true jika x ditemukan. Jika x tidak
  ditemukan, ketemu bernilai false. }

DEKLARASI
   i : integer     { pencatat indeks larik }

ALGORITMA:
   i ← 1
   ketemu ← false
   while (i ≤ n ) and (not ketemu) do
     if L[i] = x then
        ketemu ← true
     else
        i ← i + 1
     endif
   endwhile
   { i > n or ketemu }
```

(ii) Fungsi pencarian beruntun:

```
function SeqSearch3(input L : LarikInt, input n : integer,
                    input x : integer) → boolean
{ Mengembalikan nilai true jika x ditemukan di dalam larik L[1..n].
Jika x tidak ditemukan, nilai yang dikembalikan adalah false. }

DEKLARASI
   i : integer     { pencatat indeks larik }

ALGORITMA:
   i ← 1
   ketemu ← false
   while (i ≤ n ) and (not ketemu) do
     if L[i] = x then
        ketemu ← true
     else
        i ← i + 1
     endif
   endwhile
   { i > n or ketemu }
   return ketemu
```

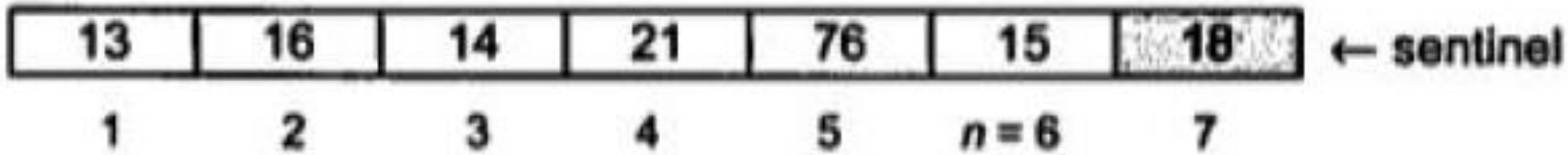
Algoritma 15.10 Fungsi pencarian beruntun (versi 2, hasil pencarian: *boolean*)

Metode Pencarian Beruntun dengan Sentinel

Jika pencarian bertujuan untuk menambahkan elemen baru setelah elemen terakhir larik, maka terdapat sebuah varian dari metode pencarian beruntun yang mangkus. Nilai x yang akan dicari sengaja ditambahkan terlebih dahulu pada elemen ke-n+15. Data yang ditambahkan setelah elemen terakhir larik ini disebut **sentinel**. Selanjutnya, pencarian beruntun dilakukan di dalam larik $L[1..n+1]$. Akibatnya, proses pencarian selalu menjamin bahwa x pasti berhasil ditemukan. Untuk menyimpulkan apakah x ditemukan pada elemen sentinel atau bukan, kita dapat mengetahuinya dengan melihat nilai *idx*. Jika $idx = n+1$ (yang berarti x ditemukan pada elemen sentinel), maka hal itu berarti bahwa x tidak terdapat di dalam larik L semula (sebelum penambahan sentinel). Keuntungannya, elemen sentinel otomatis sudah menjadi elemen yang ditambahkan ke dalam larik. Sebaliknya, jika $idx < n+1$ (yang berarti x ditemukan sebelum sentinel), maka hal itu berarti bahwa x sudah ada di dalam larik L semula.

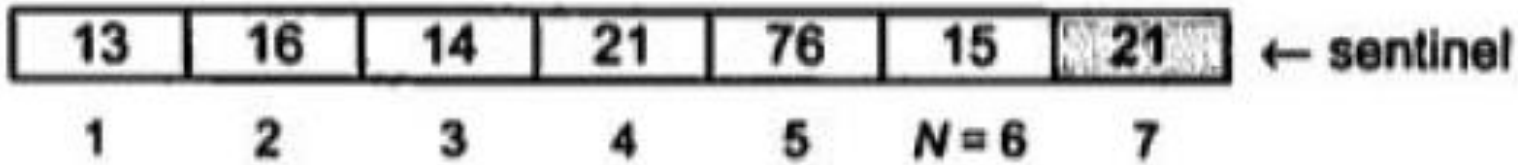
Contoh 15.3. Pencarian beruntun dengan menggunakan sentinel.

(a) $x = 18$



18 ditemukan pada elemen ke-n+1. Sentinel otomatis sudah ditambahkan ke dalam larik. Ukuran larik sekarang = 7.

(b) $x = 21$



Algoritma pencarian beruntun dengan sentinel kita tuliskan sebagai berikut:

```
procedure SeqSearchWithSentinel(input L : LarikInt, input n : integer,
                                input x : integer, output idx : integer)

{ Mencari x di dalam larik L[1..n] dengan menggunakan sentinel}
{ K.Awal: x dan elemen-elemen larik L[1..N] sudah terdefinisi
nilainya.}
{ K.Akhir: idx berisi indeks larik L yang berisi nilai x.
  Jika x tidak ditemukan, maka idx diisi dengan nilai -15. }

DEKLARASI
   i : integer    { pencatat indeks larik }

ALGORITMA:
   L[n + 1] ← x     { sentinel }
   i ← 1
   while (L[i] ≠ x) do
      i ← i + 1
   endwhile
   { L[i] = x }    { Pencarian selalu berhasil menemukan x }

   { Kita harus menyimpulkan apakah x ditemukan pada elemen
     sentinel atau bukan }

   if idx = n + 1 then  { x ditemukan pada elemen sentinel }
      idx ← -1    { berarti x belum ada pada larik L semula }
   else     { x ditemukan pada indeks < n + 1 }
      idx ← i
   endif
```

Algoritma 15.14 Pencarian beruntun dengan sentinel

Algoritma Pencarian Bagidua

Terdapat algoritma pencarian pada data terurut yang paling mangkus (*efficient*), yaitu algoritma **pencarian bagidua** atau **pencarian biner** (*binary search*). Algoritma ini digunakan untuk kebutuhan pencarian dengan waktu yang cepat. Sebenarnya, dalam kehidupan sehari-hari kita sering menerapkan pencarian bagidua. Untuk mencari arti kata tertentu di dalam kamus (misalnya kamus bahasa Inggris), kita tidak membuka kamus itu dari halaman awal sampai halaman akhir satu per satu, namun kita mencarinya dengan cara membelah atau membagi dua buku itu. Jika kata yang dicari tidak terletak di halaman pertengahan itu, kita mencari lagi di belahan bagian kiri atau belahan bagian kanan dengan cara membagi dua belahan yang dimaksud. Begitu seterusnya sampai kata yang dicari ditemukan. Hal ini hanya bisa dilakukan jika kata-kata di dalam kamus sudah terurut.

Prinsip pencarian dengan membagi data atas dua bagian mengilhami algoritma pencarian bagidua. Data yang disimpan di dalam larik harus sudah terurut. Untuk memudahkan pembahasan, selanjutnya kita misalkan elemen larik sudah terurut menurun. Dalam proses pencarian, kita memerlukan dua buah indeks larik, yaitu indeks terkecil dan indeks terbesar. Kita menyebut indeks terkecil sebagai indek ujung kiri larik dan indeks terbesar sebagai indeks ujung kanan larik. Istilah "kiri" dan "kanan" dinyatakan dengan membayangkan elemen larik terentang horizontal.

Misalkan indeks kiri adalah i dan indeks kanan adalah j. Pada mulanya, kita inisialisasi i dengan 1 dan j dengan n.

Langkah 1 : **Bagi dua elemen larik pada elemen tengah. Elemen tengah adalah elemen dengan indeks $k = (i + j)$ div 2.**
(Elemen tengah, $L[k]$, membagi larik menjadi dua bagian, yaitu bagian kiri $L[i..j]$ dan bagian kanan $L[k+1..j]$)

Langkah 2 : **Periksa apakah $L[k] = x$. Jika $L[k] = x$, pencarian selesai sebab x sudah ditemukan. Tetapi, jika $L[k] \neq x$, harus ditentukan**

apakah pencarian akan dilakukan di larik bagian kiri atau di bagian kanan. Jika $L[k] < x$, maka pencarian dilakukan lagi pada larik bagian kiri. Sebaliknya, jika $L[k] > x$, pencarian dilakukan lagi pada larik bagian kanan.

Langkah 3 : Ulangi Langkah 1 hingga x ditemukan atau $i > j$ (yaitu, ukuran larik sudah nol!)

(b) Algoritma Pencarian Bagidua

```
procedure CariNIM_2(input M : TabMhs, input n : integer,
                    input NIMmhs : integer, output IDX : integer)

{ Mencari NIMmhs di dalam larik M[1..n] yang sudah terurut menaik
berdasarkan NIM dengan metode pencarian bagidua. }
{ K.Awal : Larik M[1..n] sudah berisi data yang sudah terurut menaik,
dan NIMmhs adalah nilai yang akan dicari }
{ K.Akhir: idx berisi indeks larik tempat NIMmhs ditemukan, idx = -1
jika NIMmhs tidak  ditemukan }

DEKLARASI
  i, j : integer          { indeks kiri dan indeks kanan }
  k : integer             { indeks elemen tengah }
  ketemu : boolean        { flag untuk menentukan ketemu atau tidak }

ALGORITMA:
i ← 1       { ujung kiri larik }
j ← n       { ujung kanan larik }
ketemu ← false    { asumsikan x belum ditemukan }

while (not ketemu) and (i ≤ j) do

   k ← (i + j) div 2  { bagidua larik L pada posisi k }
   if (M[k].NIM = NIMmhs) then
     ketemu ← true
   else  { M[k].NIM ≠ NIMmhs }

      if  (M[k].NIM < NIMmhs) then
        { Lakukan pencarian pada larik bagian kanan, set indeks
          ujung kiri larik yang baru }
         i ← k + 1
      else
        { Lakukan pencarian pada larik bagian kiri, set indeks
          ujung kanan larik yang baru }
         j ← k - 1
      endif
   endif
endwhile
{ ketemu = true or i > j}
```