BAB VI
SEARCHING (PENCARIAN)

7. 1 Pencarian Beruntun (*Sequential Search*)

Prinsip kerja pencarian beruntun adalah membandingkan setiap elemen larik satu per satu secara beruntun, mulai dari elemen pertama sampai elemen yang dicari ditemukan atau semua elemen sudah diperiksa. Pencarian beruntun ada dua macam yaitu : pencarian beruntun pada larik terurut dan pencarian beruntun pada larik tidak terurut.

7.1.1 Pencarian Beruntun pada Larik yang Tidak Terurut

Pencarian dilakukan dengan memeriksa setiap elemen larik mulai dari elemen pertama sampai elemen yang dicari ditemukan atau sampai seluruh elemen sudah diperiksa.

Contoh : Perhatikan larik L di berikut ini :

44	55	12	42	94	18	06	67
1	2	3	4	5	6	7	8

Misalkan nilai yang dicari adalah X = 94

Maka, elemen yang diperiksa: 44, 55, 12, 42, 94 (ditemukan)

Indeks larik yang dikembalikan : idx = 4.

Misalkan nilai yang dicari adalah X = 44

Maka, elemen yang diperiksa: 44 (ditemukan)

Indeks larik yang dikembalikan : idx = 1.

Misalkan nilai yang dicari adalah X = 10

Maka, elemen yang diperiksa: 44, 55, 12, 42, 94, 18, 06, 67 (tidak ditemukan)

Indeks larik yang dikembalikan : idx = 0.

Algoritma pencarian beruntun sbb :

```
Procedure cari runtun1(L:larik, n:integer; x:integer; var idxX:integer);
{mencari X di dalam larik L[1..N] secara beruntun, dimulai dari elemen pertama sampai X ditemukan
atau seluruh elemen larik L sudah diperiksa. Keluaran dari proses ini adalah indeks idxX  dimana L[idxX]
:=X. idxX berharga 0 jika X tidak ditemukan}
Var i : integer;      {indeks untuk pencarian}
Begin
        i := 1;
        {periksa selama i < n dan L[i] ≠x}
        while (i < n) and (L[i] <> x) do
          begin
            i := i + 1;          {maju ke elemen berikutnya}
          end;
          {i = n atau L[i] = x}
          {simpulkan hasil pencarian}
            if (L[i] <> x ) then
               idxX := 0
              else
               idxX := i;
End;
```

Algoritma pencarian beruntun versi boolean :

```
Procedure cari runtun_versiboolean(L:larik, n:integer; x:integer; var idxX:integer);
{mencari X di dalam larik L[1..N] secara beruntun, dimulai dari elemen pertama sampai X ditemukan
      atau seluruh elemen larik L sudah diperiksa. Keluaran dari proses ini adalah indeks idxX  dimana
      L[idxX] :=X. idxX berharga 0 jika X tidak ditemukan versi boolean}
Var i : integer;      {indeks untuk pencarian}
    ketemu : integer;
      Begin
      i := 1;
      ketemu := false;
      {periksa selama i <= n dan ketemu masih bernilai false}
      while (i <= n) and (not ketemu) do
       begin
         i f L[i] = x then
              ketemu := true          {hentikan proses pencarian}
           else
              i := i + 1;             {maju ke elemen berikutnya}
        end;
        {i > n atau ketemu = true}
        {simpulkan hasil pencarian}
         if ketemu  then
              idxX := i
           else
              idxX := 0;
End;
```

7. 1. 2 Pencarian Beruntun dengan Sentinel

Sentinel merupakan elemen fiktif yang sengaja ditambahkan sesudah elemen terakhir larik. Jika elemen terakhir larik adalah L[N], maka sentinel dipasang pada elemen L[N + 1]. Sentinel ini harganya sama dengan da ta yang di cari. Akibatnya, proses pencarian selalu berhasil menemukan data yang dicari. Walaupun demikian harus diperiksa lagi letak data tersebut ditemukan, apakah :

(i) diantara elemen-elemen larik sesungguhnya yaitu dari L[1] sampai L[N], atau

(ii) pada elemen fiktif yaitu L[N + 1] yang berarti X sesungguhnya tidak terdapat di dalam larik L.

Jika X tidak ditemukan, maka sentinel tesebut sekaligus sudah ditambahkan. Perlu diingat batas pendefinisian indeks larik, sebab komputer tidak boleh mengakses elemen larik yang indeksnya melebihi rentang indeks yang sudah didefinisikan.

Perhatikan larik di bawah ini :

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

N = 6 (jumlah elemen larik semula)

(i) Misalkan elemen yang dicari adalah X = 21. Maka tambahkan 21 sebagai elemen sentinel di L [N + 1] :

13	16	14	21	76	21	21
1	2	3	4	5	6	7

Elemen yang diperiksa selama pencarian : 13, 16, 14, 21

Indeks larik yang dikembalikan : 4

Karena $4 \neq N+1$ berarti X = 21 terdapat di dalam larik L semula.

(ii) Misalkan elemen yang dicari adalah X = 13. Maka tambahkan 13 sebagai elemen sentinel di L [N + 1] :

13	16	14	21	76	21	13
1	2	3	4	5	6	7

Elemen yang diperiksa selama pencarian : 13

Indeks larik yang dikembalikan : 1

Karena 1 ≠ N+1 berarti X = 13 terdapat di dalam larik L semula.

(iii) Misalkan elemen yang dicari adalah X = 15. Maka tambahkan 15 sebagai elemen sentinel di L [N + 1] :

13	16	14	21	76	21	15
1	2	3	4	5	6	7

Elemen yang diperiksa selama pencarian : 13, 16, 14, 21, 76, 21, 15

Indeks larik yang dikembalikan : 7

Karena 7 = N+1 berarti X = 15 tidak terdapat di dalam larik L semula.

Algoritma Pencarian Beruntun dengan Sentinel

```
Const max : integer = 10;                    {jumlah maksimum elemen larik}
Type Larik = array [1..max+1] of integer; {tipe larik}

Procedure cari beruntunsentinel(L:larik, n:integer; x:integer;  var idxX:integer);
{mencari X di dalam larik L[1..N] secara beruntun dimulai dari elemen pertama sampai X ditemukan atau seluruh
elemen larik L sudah diperiksa. Keluaran dari proses ini adalah indeks idxX  dimana L[idxX] :=X. idxX berharga 0
jika X tidak ditemukan}
Var i : integer;        {indeks untuk pencarian}
    ketemu : boolean;
Begin
        i := 1;
        ketemu := false;
        {periksa selama I <= N dan ketemu masih bernilai false}
        while (I <= N) and (not ketemu) do
          begin
              if (L[I] = X) then
                ketemu : = true    {hentikan proses pencarian}
            else                                {L[I] ≠X}
                I := I + 1;                     {maju ke elemen berikutnya}
          end;

         {I > N or ketemu = true}
          if (ketemu) then                      {X ditemukan}
             idxX := I
            else                                {X tidak ditemukan di dalam larik}
              idxX := 0;
End;
```

7.1.3 Pencarian Beruntun pada Larik yang Terurut

Secara singkat proses pencarian pada tabel terurut adalah sebagai berikut : Dimulai dari elemen pertama pada tabel dilakukan pembandingan dengan elemen yang dicari. Jika elemen dalam tabel masih lebih kecil dari elemen yang dicari, pencarian dihentikan dan bisa dipastikan bahwa elemen yang dicari memang tidak ditemukan.

Jika elemen yang dicari tidak ditemukan dan kits ingin menyisipkan elemen tersebut pada posisinya yang tepat, maka seringkali diperlukan adanya penggeseran terhadap elemen-elemen yang nantinya akan terletak pada subskrib yang lebih besar dari elemen yang dicari.

```
Procedure cari runtun_ascaending(a:larik, var ada:boolean;var n:integer; var posisi:integer; data :integer);
{elemen vektor bertipe integer. Jika data yang dicari ditemukan, letaknya disimpan sebagai peubah posisi. Jika tidak
ditemukan, data baru tersebut akan disisipkan ke dalam vektor}
Var i, j : integer;      {indeks untuk pencarian}
Begin
        i := 1; j:=0; ada := false;
        repeat
          if a[i] = data then                {elemen yang dicari ditemukan}
                  begin
                          posisi := i;
                          ada := true;
                  end
          else if a[i] > data then           {elemen yang dicari tidak ditemukan}
                  begin
                          j := i;      {posisi elemen yang baru}
                          i := N-1;
                  end;
          inc(i);
      until (i = n) or ada;
      if not ada then                        {data tidak ketemu}
                  begin
                          if  j = 0 then          {elemen baru menempati subkrib terakhir}
                                  a [n+1] := data;
                          else                    {menyisip di tengah}
                              begin
                                  for i := n downto j do          {menggeser elemen}
                                      a[i+1] := a[i];
                                  a[j] := data;                   {menempatkan elemen baru}
                              end;
                          inc (n);                                {menambah ukuran vektor}
                  end;
End;
```

7. 2. Pencarian Biner /Bagi Dua (*Binary Search)*

Pencarian dua/biner (***binary search***) merupakan metode pencarian yang diterapkan pada sekumpulan data yang sudah terurut baik secara ***ascending*** maupun ***descending***. Salah satu keuntungan data yang terurut adalah memudahkan pencarian dalam hal ini adalah pencarian bagidua. **Syarat utama penerapan pencarian bagi dua adalah data yang sudah terurut (ascending/menaik atau descending/menurun)**. Selama proses pencarian diperlukan dua buah indeks larik yaitu indeks terkecil dan indeks terbesar. **Indeks terkecil sebagai indeks ujung kiri larik dan terbesar sebgai indeks ujung kanan larik.** Istilah **kiri dan kanan** dinyatakan dengan membayangkan elemen larik terentang horizontal.

Misalkan indeks kiri adalah **Atas (a)** dan indeks kanan adalah **Bawah (b)**. Pada mulanya, i**a=1** dan **ib=N**.

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

Langkah 2 : Periksa apakah L[k] = X (X adalah elemen yang dicari dalam larik). Jika L[k] = X, pencarian dihentikan karena X ditemukan. Jika L[k] <> X , harus ditentukan apakah pencarian akan dilakukan pada larik bagian kiri atau bagian kanan. Jika L[k] < X, maka pencarian dilakukan pada larik bagian kiri. Sebaliknya, jika L[k] > X, pencarian dilakukan pada larik bagian kanan.

Langkah 3 : Ulangi langkah 1 sampai X ditemukan atau ia > ib yaitu ukuran larik sudah nol.

Contoh pencarian biner :

Misalakan diberikan larik L, dengan 8 elemen yang sudah terurut **descending** seperi berikut :

94	67	55	44	42	18	12	06
ia=1	2	3	4	5	6	7	ib=8

(i) Misalkan elemen yang di cari adalah X = 44.

Langkah 1 :

ia = 1 dan ib = 8.

Indeks elemen tengah k = (1 + 8) div 2 = 4 (diarsir)

94	67	55	44	42	18	12	06
1	2	3	4	5	6	7	8

kiri kanan

Langkah 2 :

L[4] = 44 ? Ya. (X ditemukan, pencarian dihentikan).

(ii) Misalkan elemen yang di cari adalah X = 42.

Langkah 1 :

ia = 1 dan ib = 8.

Indeks elemen tengah k = (1 + 8) div 2 = 4 (diarsir)

94	67	55	44	42	18	12	06
ia=1	2	3	4	5	6	7	ib=8

Langkah 2 : kiri kanan

L[4] = 42 ? Tidak. Harus diputuskan apakah pencarian akan dilakukan di bagian kiri atau bagian kanan dengan pemeriksaan sbb :

L[4] > 42? Ya. Lakukan pencarian pada larik bagian kanan dengan

ia = k + 1 = 4 +1 = 5 dan ib = 8 (tetap)

42	18	12	06
ia=5	6	7	ib=8

Langkah 1' :

ia = 5 dan ib = 8.

Indeks elemen tengah k = (5 + 8) div 2 = 6 (diarsir)

42	18	12	06
5	6	7	8

Kiri' Kanan'

Langkah 2' :

L[6] = 42 ? Tidak. Harus diputuskan apakah pencarian akan dilakukan di bagian kiri atau bagian kanan dengan pemeriksaan sbb :

L[6] > 42? Tidak. Lakukan pencarian pada larik bagian kiri dengan

ia = 5 (tetap) dan ib = k - 1 = 6 –1 = 5

42
5

:

Langkah 1'' :

ia = 5 dan ib= 5

Indeks elemen tengah k = (5 + 5) div 2 =5 (diarsir)

42
5

:

Langkah 2''

L[5] = 42 ? Ya. (X ditemukan, pencarian dihentikan)

(iii) Misalkan elemen yang di cari adalah X = 97.

Langkah 1 :

ia = 1 dan ib = 8.

Indeks elemen tengah k = (1 + 8) div 2 = 4 (diarsir)

94	67	55	44	42	18	12	06
ia=1	2	3	4	5	6	7	ib=8

kiri kanan

Langkah 2 :

L[4] = 97 ? Tidak. Harus diputuskan apakah pencarian akan dilakukan di bagian kiri atau bagian kanan dengan pemeriksaan sbb :

L[4] > 97? Tidak. Lakukan pencarian pada larik bagian kiri dengan

ia = 1 (tetap) dan ib = k – 1 = 4 – 1= 3

94	67	55
ia=1	2	ib=3

Langkah 1' :

ia = 1 dan ib = 3.

Indeks elemen tengah k = (1 +3) div 2 = 2 (diarsir)

94	67	55
1	2	3

Kiri' Kanan'

Langkah 2' :

L[2] = 97 ? Tidak. Harus diputuskan apakah pencarian akan dilakukan di bagian kiri atau bagian kanan dengan pemeriksaan sbb :

L[2] > 97? Tidak. Lakukan pencarian pada larik bagian kiri dengan

ia = 1 (tetap) dan ib = k - 1 = 2 –1 = 1

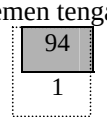
94
1

:

Langkah 1'' :

ia = 1 dan ib= 1

Indeks elemen tengah k = (1 + 1) div 2 =1 (diarsir)



Langkah 2'' ..:

L[1] = 97 ? Tidak. Harus diputuskan apakah pencarian akan dilakukan di bagian kiri atau bagian kanan dengan pemeriksaan sbb :

L[1] > 97? Tidak. Lakukan pencarian pada larik bagian kiri dengan

ia = 1 (tetap) dan ib = k - 1 = 1 – 1 = 0

karena ia > ib, maka tidak ada lagi bagian larik yang tersisa. Jadi, X tidak ditemukan di dalam larik , pencarian dihentikan.

Algoritma pencarian bagidua (biner)

```
Procedure cari binerdescending(L:larik, n:integer; x:integer; var idxX:integer);
{mencari X di dalam larik L[1..N] yang sudah terurut menaik dengan metode pencarian biner. Keluaran
dari proses ini adalah indeks idxX  dimana L[idxX] :=X. idxX berharga 0 jika X tidak ditemukan}
Var ia, ib, k : integer;
    ketemu : boolean;
Begin
        ia := 1;
        ib := N;
        ketemu := false;
        while (not ketemu) and (ia <= ib) do
          begin
            k := (ia + ib) div 2;          {bagidua larik L pada posisi k}
             if (L[k] = X) then
                ketemu : = true
             else                          {L[k] ≠X}
                 if (L[k] > X) then
           {akan dilakukan pencarian pada larik begian kanan, set indeks ujung kiri larik yang baru}
                      ia : = k + 1
                 else
                      ib : = k – 1;
           end;
           {ketemu = true or ia > ib}
           if (ketemu) then               {X ditemukan}
              idxX := k
            else                          {X tidak ditemukan di dalam larik}
              idxX := 0;
End;
```

Untuk pencarian biner pada larik yang sudah terurut menaik, hanya perlu mengganti

If (L[k] > X) then

dengan

If (L[k] < X) then

Untuk larik berukuran N elemen

- algoritma pencarian beruntun melakukan pembandingan elemen larik sebanyak N kali.
- algoritma pencarian biner melakukan pembandingan elemen larik sebanyak $^2\log(N)$ kali.

Karena $^2\log(N) < N$ untuk N yang besar, maka algoritma pencarian biner lebih cepat daripada algoritma beruntun.