

Algoritma Divide dan Conquer

Pemrogram bertanggung jawab atas implementasi solusi. Pembuatan program akan menjadi lebih sederhana jika masalah dapat dipecah menjadi sub masalah - sub masalah yang dapat dikelola.

Penyelesaian masalah dengan komputer berhadapan dengan 4 hal, yaitu :

1. Pemahaman keterhubungan elemen-elemen data yang relevan terhadap solusi secara menyeluruh.
2. Pengambilan keputusan mengenai operasi-operasi yang dilakukan terhadap elemen-elemen data.
3. Perancangan representasi elemen-elemen data di memori sehingga memenuhi kriteria berikut:
 - a. Memenuhi keterhubungan logik antara elemen-elemen data.
 - b. Operasi-operasi terhadap elemen-elemen data dapat dilakukan secara mudah dan efisien.
4. Pengambilan keputusan mengenai mengenai bahasa pemrograman terbaik untuk menerjemahkan solusi persoalan menjadi program.

STRATEGI DIVIDE AND CONQUER

Metode

Strategi Divide dan Conquer memecah masalah menjadi submasalah-submasalah independen yang lebih kecil sehingga solusi submasalah-submasalah dapat diperoleh secara mudah, solusi submasalah-submasalah digabung menjadi solusi seluruh masalah.

Skema umum algoritma divide dan conquer

```
Procedure DNC ( i,j : integer )
Var K : integer ;
  If SMALL (i,j) then SOLVE (i,j)
  Else begin
    K := DIVIDE (i,j)
    COMBINE (DNC(i,k),DNC(k+1,j))
  End if
```

Keterangan :

1. SMALL adalah fungsi yang mengirim BOOLEAN, menentukan apakah ukuran telah cukup kecil sehingga solusi dapat diperoleh. . Ukuran dinyatakan sebagai telah berukuran kecil bergantung masalah.
2. DIVIDE adalah fungsi membagi menjadi 2 bagian pada posisi K. Biasanya bagian berukuran sama.
3. COMBINE adalah fungsi menggabungkan solusi X dan Y submasalah. Solusi diperoleh dengan memanggil prosedur rekursif DNC.

Jika ukuran kedua submasalah sama, waktu komputasi DNC dideskripsikan hubungan rekuren berikut :

$$\begin{aligned} T(n) &= g(n), & n \text{ kecil} \\ &2 T(n/2) + f(n), & \text{selainnya} \end{aligned}$$

dimana :

- $T(n)$ adalah waktu untuk DNC dengan n masukan,
- $g(n)$ adalah waktu komputasi jawaban secara langsung untuk masukan kecil dan
- $f(n)$ adalah waktu COMBINE.

Untuk algoritma divide dan conquer yang menghasilkan submasalah-submasalah dengan tipe masalah yang sama dengan masalah awal, sangat alami untuk mendeskripsikan algoritma secara rekursi. Kemudian untuk meningkatkan efisiensi dilakukan penerjemahan menjadi bentuk iterasi.

Pemakaian teknik Divide dan Conquer banyak digunakan dalam menyelesaikan berbagai macam persoalan, antara lain :

1. Searching
2. Sorting

Algoritma Binary Search


Binary Search (Pencarian Biner) dapat dilakukan jika data sudah dalam keadaan urut. Dengan kata lain, apabila data belum dalam keadaan urut, pencarian biner tidak dapat dilakukan. Dalam kehidupan sehari-hari, sebenarnya kita juga sering menggunakan pencarian biner. Misalnya saat ingin mencari suatu kata dalam kamus.


Prinsip dari pencarian biner dapat dijelaskan sebagai berikut :


1. Mula-mula diambil posisi awal = 1 dan posisi akhir = N
2. Cari posisi data tengah dengan rumus $(\text{posisi awal} + \text{posisi akhir}) / 2$
3. Data yang dicari dibandingkan dengan data tengah.
4. Jika lebih kecil, proses dilakukan kembali tetapi posisi akhir dianggap sama dengan posisi tengah - 1.
5. Jika lebih besar, proses dilakukan kembali tetapi posisi awal dianggap sama dengan posisi tengah + 1.
6. Demikian seterusnya sampai data tengah sama dengan yang dicari.

Untuk lebih jelasnya, perhatikan contoh berikut. Misalkan kita ingin mencari 17 pada sekumpulan data berikut :

3	9	11	12	15	17	23	31	35
---	---	----	----	----	----	----	----	----


 awal



 tengah



 akhir


1. Mula-mula dicari data tengah, dengan rumus $(1 + 9) / 2 = 5$.
2. Berarti data tengah adalah data ke-5, yaitu 15.
3. Data yang dicari, yaitu 17, dibandingkan dengan data tengah ini.

4. Karena $17 > 15$, berarti proses dilanjutkan tetapi kali ini posisi awal dianggap sama dengan posisi tengah + 1 atau 6.

3	9	11	12	15	17	23	31	35
---	---	----	----	----	----	----	----	----



 awal


 tengah


 akhir

1. Data tengah yang baru didapat dengan rumus $(6 + 9) / 2 = 7$. Berarti data tengah yang baru adalah data ke-7, yaitu 23.
2. Data yang dicari, yaitu 17 dibandingkan dengan data tengah ini.
3. Karena $17 < 23$, berarti proses dilanjutkan tetapi kali ini posisi akhir dianggap sama dengan posisi tengah - 1 atau 6.


3	9	11	12	15	17	23	31	35
---	---	----	----	----	----	----	----	----



 awal = akhir

1. Data tengah yang baru didapat dengan rumus $(6 + 6) / 2 = 6$. Berarti data tengah yang baru adalah data ke-6, yaitu 17.
2. Data yang dicari dibandingkan dengan data tengah ini dan ternyata sama. Jadi data ditemukan pada indeks ke-6.
3. Bagaimana jika data yang dicari tidak ada, misalnya 16?
4. Pencarian biner ini akan berakhir jika data ditemukan atau posisi awal lebih besar dari posisi akhir.
5. Jika posisi awal sudah lebih besar daripada posisi akhir berarti data tidak ditemukan.

Untuk lebih jelasnya perhatikan proses pencarian 16 pada data di atas. Prosesnya hampir sama dengan pencarian 17. Tetapi setelah posisi awal = posisi akhir = 6, proses masih dilanjutkan lagi dengan posisi awal = 6 dan posisi akhir = 5

3	9	11	12	15	17	23	31	35
---	---	----	----	----	----	----	----	----


 akhir


 awal

Disini dapat dilihat bahwa posisi awal lebih besar daripada posisi akhir, yang artinya data tidak ditemukan.

Secara umum, algoritma pencarian biner dapat dituliskan sebagai berikut :

1. $l \leftarrow 1$.
2. $r \leftarrow N$.
3. $ketemu \leftarrow \text{false}$.
4. selama $(l \leq r)$ dan $(\text{not } ketemu)$ kerjakan baris 5 sampai dengan 8.
5. $m \leftarrow (l + r) / 2$
6. Jika $(\text{Data}[m] = x)$ maka $ketemu \leftarrow \text{true}$.
7. Jika $(x < \text{Data}[m])$ maka $r \leftarrow m - 1$.
8. Jika $(x > \text{Data}[m])$ maka $l \leftarrow m + 1$.
9. If $(ketemu)$ maka m adalah indeks dari data yang dicari, jika tidak data tidak ditemukan.

Berikut ini adalah contoh fungsi untuk mencari data menggunakan pencarian biner.

```

Function BinarySearch (x: word) : integer;
var
    l, r, m : word;
    ketemu : boolean;
begin

```

```

l := 1;
r := N;
ketemu := false;
while (1 <= r) and ( not ketemu ) do
begin
    m := (1 + r) div 2;
    if (Data [m] = x ) then
        Ketemu := true
    else if (x < Data [m] ) then
        r := m - 1
    else
        l := m + 1;
end;

if ( ketemu ) then
    BinarySearch := m
else
    BinarySearch := -1;
end;

```

Fungsi di atas akan mengembalikan indeks dari data yang dicari. Apabila data tidak ditemukan, maka yang yang dikembalikan adalah -1.

Jumlah perbandingan minimum pada pencarian biner adalah 1 kali, yaitu bila data yang dicari tepat berada di tengah-tengah. Jumlah perbandingan maksimum yang dilakukan dengan pencarian biner dapat dicari dengan rumus logaritma, yaitu :

$$C = \lceil \log_2(N) \rceil$$

Algoritma Quick Sort

Metode Quick atau yang sering disebut juga metode partisi diperkenalkan pertama kali oleh C. A. R. Hoare pada tahun 1962. Pada metode quick, jarak dari kedua elemen yang ditukarkan dibuat cukup besar dengan tujuan untuk mempertinggi efektivitasnya. Hal ini mengingat metode gelembung yang menggunakan jarak cukup dekat ternyata kurang efektif.

Proses pengurutan dengan metode quick dapat dijelaskan sebagai berikut : mula-mula dipilih data tertentu yang dinamakan pivot, misalnya x . Pivot ini harus diletakkan pada posisi ke- j sedemikian hingga data antara 1 sampai dengan $(j - 1)$ lebih kecil daripada x ; sedangkan data pada posisi ke- $(j+1)$ sampai dengan N lebih besar daripada x . Cara pengaturannya adalah menukarkan data di antara posisi 1 sampai dengan $(j - 1)$ yang lebih besar daripada x dengan data di antara posisi $(j + 1)$ sampai dengan N yang lebih kecil daripada x .

Algoritma penyisipan langsung sendiri dapat dituliskan sebagai berikut:

1. $x \leftarrow \text{Data}[(L + R) / 2]$.
2. $i \leftarrow L$
3. $j \leftarrow R$
4. Selama $(i \leq j)$ kerjakan baris 5 sampai dengan 12.
5. Selama $(\text{Data}[i] < x)$ kerjakan $i \leftarrow i + 1$
6. Selama $(\text{Data}[j] > x)$ kerjakan $j \leftarrow j - 1$
7. Jika $(i \leq j)$ maka kerjakan baris 8 sampai dengan 10; jika tidak kerjakan baris 11.
8. Tukar $\text{Data}[i]$ dengan $\text{Data}[j]$.
9. $i \leftarrow i + 1$
10. $j \leftarrow j - 1$
11. Jika $(L < j)$ kerjakan lagi baris 1 dengan $R = j$.
12. Jika $(i < R)$ kerjakan lagi baris 1 dengan $L = i$.

Jika suatu barisan yang terdiri dari n elemen yang ditempatkan dalam suatu array dan urutan yang diinginkan adalah urutan yang tidak turun (non decreasing) maka dapat digunakan metode Quick Sort yang dengan teknik Divide and Conquer.

Adapun algoritma Quick Sort tersebut terdiri dari dua prosedur yaitu prosedur PARTITION dan prosedur QUICKSORT. Berikut ini disajikan algoritma Quick Sort yang dimaksud, yaitu :

```

PROCEDURE QUICKSORT(p,q)
  IF p < q then j ← q + 1
    CALL PARTITION(p,j)
    CALL QUICKSORT(p,j-1)
    CALL QUICKSORT(j+1,q)
  END IF
END QUICKSORT

PROCEDURE PARTITION(m,p)
  INTEGER m,p,i ; GLOBAL A(m-1,p)
  V ← A(m) ; i ← m
  LOOP
    LOOP i ← i + 1 UNTIL A(i) >= V REPEAT
    LOOP p ← p - 1 UNTIL A(p) <= V REPEAT
    IF i < p THEN CALL INTERCHANGE (A(i),A(p))
      ELSE EXIT
    END IF
  REPEAT
  A(m) ← A(p)
  A(p) ← V
END PARTITION

```

Contoh :

Suatu Array A terdiri dari 9 elemen, yaitu :

A(1) = 65	A(4) = 80	A(7) = 60
A(2) = 70	A(5) = 85	A(8) = 50
A(3) = 75	A(6) = 60	A(9) = 45

Elemen-elemen tersebut akan disusun secara tidak turun berdasarkan algoritma Quick Sort.

Jalannya proses pada algoritma tersebut disajikan dalam bentuk tabel sebagai berikut :

1	2	3	4	5	6	7	8	9	10	i	p
65	70	75	80	85	60	55	50	45		2	9
65	45	75	80	85	60	55	50	70		3	8
65	45	50	80	85	60	55	75	70		4	7
65	45	50	55	85	60	80	75	70		5	6
65	45	50	55	60	85	80	75	70		6	5

☆

1	2	3	4	5	6	7	8	9	10	i	p
65	45	50	55	60	85	80	75	70		6	9
65	45	50	55	60	70	80	75	85		7	8
65	45	50	55	60	70	75	80	85		8	7

☆

Dan seterusnya

☆ EXIT

Analisisnya :

Hitung jumlah dari perbandingan-perbandingan elemennya dalam hal ini disimpan dalam variable $C(n)$ dan kita asumsikan bahwa :

- n elemen yang disortir berbeda,
- proses pembagian (partisi) elemen V dalam prosedur PARTITION dilakukan dengan proses seleksi secara acak.

Worst Case dari $C(n)$ dinotasikan dengan $C_w(n)$. $C(n)$ di dalam setiap pemanggilan prosedur PARTITION maksimum sebesar $(p - q + 1)$ kali.

Misalkan r adalah jumlah kumulatif dari elemen-elemen di dalam seluruh pemanggilan prosedur PARTITION pada setiap tingkat dari teknik rekursif tersebut. Pada tingkat pertama

terjadi pemanggilan prosedur PARTITION sebanyak satu (1) kali yakni CALL PARTITION (1, n + 1) dan nilai $r = n$. Pada tingkat kedua terjadi pemanggilan prosedur PARTITION paling banyak dua (2) kali dan nilai $r = n - 1$, dan seterusnya dengan cara yang sama pada tingkat berikutnya. Dengan demikian dari proses tersebut diperoleh $C_w(n)$ akan sama dengan jumlah seluruh tingkat (r) dan nilai r berkisar didalam interval $[2, n]$.

Jadi kompleksitas waktunya (Worst Case) = $C_w(n) = O(n^2)$ dan

Average Case = $CA(n) = O(n \log n)$.