PENGGUNAAN ALGORITMA DIVIDE AND CONQUER UNTUK OPTIMASI KONVERSI BILANGAN DESIMAL KE BINER

Danang Arief Setyawan
NIM : 13505090

Program Studi Teknik Informatika
Institut Teknologi Bandung
e-mail: das_centauri@yahoo.com

ABSTRAK

Tulisan ini berisi tentang penggunaan prinsip algoritma *Divide and Conquer* untuk melakukan optimasi pada proses konversi bilangan dari bilangan desimal (basis-10) ke bilangan biner (basis-2) metode konvensional. Metode konvensional yang dimaksud adalah metode pembagian sisa seperti yang umumnya diajarkan di sekolah-sekolah, yaitu dengan membagi bilangan desimal yang ingin diubah dengan bilangan basis secara berulang. Dengan menerapakan prinsip algoritma *Divide and Conquer*, maka proses konversi dapat dioptimasi dengan mengurangi jumlah operasi pembagian yang harus dilakukan.

Kata kunci: *Divide and Conquer*, Konversi, Desimal, Biner, Pembagian sisa.

1. PENDAHULUAN

Dalam bidang matematika, sistem bilangan merupakan salah satu bidang yang cukup penting. Sistem bilangan merupakan cara untuk menuliskan atau melambangkan nilai suatu bilangan dengan basis tertentu, dilambangkan dengan XXX_Y. Aturan yang dipakai adalah setiap angka/simbol yang digunakan untuk melambangkan nilai bilangan tersebut harus bernilai kurang dari basis yang digunakan, yaitu bilangan 0 s.d. (basis-1).

Sistem biilangan yang paling populer adalah sistem bilangan decimal (basis-10). Untuk bilangan desimal ini, angka/simbol yang digunakan tidak boleh melebihi 10-1, atau maksimal 9. Sistem ini merupakan sistem bilangan standard yang dipakai di dunia pendidikan dan juga di dalam kehidupan sehari-hari. Karena itu, kita tidak harus menuliskan angka basis di belakangnya, dengan demikian angka yang tidak diberi keterangan basis akan selalu kita anggap sebagai angka desimal. Selain sistem bilangan desimal tersebut, sistem bilangan lainnya yang sering kita jumpai adalah sistem bilangan biner (basis-2), oktal (basis-8) dan heksadesimal (basis-16). Penggunaan basis lainnya, seperti 4, 5 atau 7, juga dimungkinkan meskipun penggunaannya sangat jarang dijumpai.

Nilai suatu bilangan dengan basis tertentu dapat diperoleh dengan menggunakan persamaan berikut:

$$Nilai = X_1 \cdot Y^{n-1} + X_2 \cdot Y^{n-2} + ... + X_{n-1} \cdot Y^1 + X_n \cdot Y^0 \qquad \textbf{(1)}$$

Seperti yang kita ketahui bersama, untuk dunia elektronika dan komputer penggunaan basis yang paling mendasar adalah basis-2 dan perpangkatannya (basis-8, dan basis-16, basis-4 jarang dipakai). Hal ini dikarenakan prinsip elektronik yang digunakan adalah penggunaan kumpulan *gate* / saklar yang hanya mempunyai status terbuka atau tertutup yang biasa dilambangkan dengan angka 0 atau 1.

2. DASAR TEORI

2.1 Konversi Sistem Bilangan

Konversi sistem bilangan atau perubahan basis sistem bilangan dapat dilakukan dengan beberapa metode. Metode yang paling umum adalah dengan menggunakan metode pembagian sisa. Metode-metode lainnya pada umumnya jarang dipakai.

2.1.1 Metode Pembagian Sisa

Metode ini digunakan karena pengaplikasiannya sangat mudah dan langkah-langkahnya juga mudah dipahami.

Metode pembagian sisa didasarkan pada penggunaan angka hasil pembagian bulat dan angka sisa pembagian pada pembagian suatu bilangan dengan bilangan lainnya. Untuk memahami pengertian angka hasil pembagian bulat dan angka sisa pembagian, mari kita perhatikan contoh berikut :

$$20 \div 8 = \ldots$$

Perhatikan bahwa angka 20 dapat kita tulis sebagai berikut :

$$20 = 8 \cdot 2 + 4$$

Dengan kata lain, untuk mendapatkan angka 20 kita bisa mengalikan 8 dengan 2 kemudian menambahkannya dengan 4. Jika pernyataan itu dibalik, 20 dibagi 8 menghasilkan angka 2 dengan sisa 4. Singkatnya, pembagian tersebut menghasilkan angka hasi lpembagian bulat 2 dan angka sisa pembagian 2. Saya rasa cukup mudah untuk dipahami.

Disamping kemudahan pengaplikasiannya, metode juga ini memiliki kekurangan karena bilangan yang hendak diubah basisnya harus berupa bilangan desimal (basis-10). Mengapa demikian? Karena setiap orang sudah terbiasa dengan nilai bilangan desimal dibanding nilai bilangan non-desimal. Demikian juga untuk operasi bilangan, kebanyakan orang akan lebih cepat melakukan operasi bilangan desimal daripada bilangan non-desimal. Kita gunakan contoh operasi pembagian bilangan desimal dan oktal berikut :

$$72 \div 6 = 12 \qquad \text{(a)}$$

$$72_8 \div 6_8 = \ldots_8 \qquad \text{(b)}$$

Semua orang pasti dapat menemukan jawaban soal (a) dengan cepat dan benar, namun bagaimana dengan soal (b)? Belum tentu, karena cara pembagiannya sudah berbeda. Jawabannya pun bukan 12_8. Silahkan dibuktikan dengan kalkulator Anda! Tapi ingat bahwa kalkulator yang Anda pakai saat melakukan perhitungan harus dalam mode bilangan oktal. Berapa jawabannya?

Kalau Anda cermat, Anda akan tahu mengapa bisa demikian. Ya, karena angka 72_8 di atas nilainya tidak sama dengan 72_{10} melainkan 58_{10}. Sedangkan $6_8 = 6_{10}$. Jadi nilai seharusnya $(9,67)_{10} = (11,67)_8$.

Namun persoalan di atas tidak akan kita jumpai pada tulisan ini, karena bilangan yang ingin kita ubah sejak awal adalah bilangan bertipe desimal. Permasalahan yang kita hadapi sekarang tinggal bagaimana mengubah bilangan desimal menjadi bilangan non-desimal yang dalam hal ini kita gunakan metode pembagian sisa. Dalam proses pengubahan ini kita akan menggunakan iterasi atau pengulangan pada proses pembagiannya.

Langkah-langkahnya adalah sebagai berikut :

1. Bagi bilangan desimal (X) yang ingin kita ubah dengan basis (Y) yang baru.
2. Ambil sisa pembagian yang diperoleh, simpan hasil. Letakkan di bagian paling depan jika sudah ada bilangan sebelumnya.
3. Ganti bilangan X dengan sisa hasil pembagian bulat yang kita peroleh.
4. Ulangi pembagian selama X > 0.

2.1.2 Konversi Bilangan Basis-n ke Basis-n^y

Ada beberapa sifat khusus yang terdapat dalam konversi sistem bilangan ini. Hal ini terjadi jika basis bilangan yang ingin kita ubah adalah bilangan exponensial dari basis bilangan sebelumnya, begitu juga sebaliknya.

> Dalam koversi bilangan X_a ke bilangan Y_b, apabila $a=b^n$, maka untuk setiap angka pada X dapat diganti secara langsung dengan n-angka yang bernilai bersesuaian dengan angka tersebut.

Berikut contoh untuk memperjelas sifat tersebut :

$1_8 = 001_2$	$12_8 = 001\text{-}010_2$	$123_8 = 001\text{-}010\text{-}011_2$
$4_8 = 100_2$	$40_8 = 100\text{-}000_2$	$207_8 = 010\text{-}000\text{-}111_2$
$7_8 = 111_2$	$56_8 = 101\text{-}110_2$	$777_8 = 111\text{-}111\text{-}111_2$

Abaikan tanda (-) pada bilangan biner di sebelah kanan. Tanda tersebut hanya digunakan untuk memperjelas pengelompokan saja, tidak ada maksud lain. Begitu pula angka-angka 0 yang terletak di paling depan juga bisa anda abaikan.

Pertama-tama, tugas Anda adalah membuktikan sendiri bahwa semua konversi di atas adalah benar. Setelah itu perhatikan bahwa sebenarnya setiap angka pada bilangan oktal di sebelah kiri akan diganti dengan 3 angka biner yang nilainya bersesuaian.

Hal ini bisa terjadi karena :

- nilai maksimal bilangan biner 3 angka (111) akan sama dengan nilai maksimal 1 angka pada bilangan oktal (7), begitu juga sebaliknya. Nilai minimal keduanya akan sama-sama 0.
- Setiap perbedaan n indeks pada bilangan oktal berarti perbedaan bilangan pengalinya sebesar 8^n. Hal ini akan sama dengan perbedaan bilangan pengali sebesar 2^{3n}, dengan demikian pergeseran indeksnya adalah 3n. Dan karena jumlah angka biner yang digunakan untuk melambangkan setiap angka oktal sebanyak 3, maka tidak akan terjadi masalah.

2.2 Algoritma *Divide and Conquer*

Algoritma *Divide and Conquer* merupakan salah satu algoritma yang cukup populer di dunia ilmu komputer. *Divide and Conquer* merupakan algoritma yang berprinsip memecah-mecah permasalahan yang terlalu besar menjadi beberapa bagian kecil sehingga lebih mudah untuk diselesaikan.

Langkah-langkah umum algoritma *Divide and Conquer*:

1. *Divide*: membagi masalah menjadi beberapa upa-masalah yang memiliki kemiripan dengan masalah semula namun berukuran lebih kecil (idealnya berukuran hampir sama),

2. *Conquer*: memecahkan (menyelesaikan) masing-masing upa-masalah (secara rekursif)
3. *Combine:* mengabungkan solusi masing-masing upa-masalah sehingga membentuk solusi masalah semula.

```
Procedure DIVIDE_n_CONQUER(input n : integer)
{ Masukan: masukan yang berukuran n
  Keluaran: solusi dari masalah semula
}
Deklarasi
     r, k : integer
Algoritma
  if n ≤ n0 then {masalah sudah cukup kecil}
     SOLVE sub-masalah yang berukuran n ini
  else
     Bagi menjadi r sub-masalah, masing-masing
       berukuran n/k
     for masing-masing dari r upa-masalah do
        DIVIDE_n_CONQUER(n/k)
     endfor
     COMBINE solusi dari r sub-masalah menjadi
       solusi masalah semula
  endif
```

Skema umum algoritma *Divide and Conquer*

Sesuai dengan karakteristik pembagian dan pemecahan masalahnya tersebut, maka algoritma ini dapat berjalan baik pada persoalan yang bertipe *rekursif* (perulangan dengan memanggil dirinya sendiri). Dengan demikian, algoritma ini juga dapat diimplementasikan dengan cara *iteratif* (perulangan biasa), karena pada prinsipnya *iteratif* hampir sama dengan *rekursif*.

Salah satu penggunaan algoritma ini yang paling populer adalah dalam hal pengelolaan data bertipe *array* (elemen larik). Mengapa? Karena pengelolaan *array* pada umumnya selalu menggunakan prinsip *rekursif* atau *iteratif*. Penggunaan secara spesifik adalah untuk mencari nilai minimal dan maksimal serta untuk pengurutan elemen *array*. Dalam hal pengurutan ini ada 4 macam algoritma pengurutan yang berdasar pada algoritma *Divide and Conquer*, yaitu *merge sort*, *insert sort*, *quick sort* dan *selection sort*. *Merge sort* dan *quick sort* mempunyai kompleksitas algoritma $O(n\ ^2\log\ n)$. Hal ini lebih baik jika dibandingkan dengan pengurutan biasa dengan menggunakan algoritma *brute force*.

3. PENERAPAN ALGORITMA

3.1 Konversi tanpa Optimasi dengan menggunakan Metode Pembagian Sisa

Dengan dasar teori-teori di atas, sekarang kita akan membahas tentang penerapan algoritma *Divide and Conquer* dalam proses konversi sistem bilangan desimal ke biner ini.

Sesuai dengan pembahasan metode pembagian sisa di atas, maka kita melakukan pembagian bilangan desimal (X) yang hendak kita ubah dengan basis dari bilangan biner, yaitu 2. Ambil setiap angka sisa pembagian sebagai komponen jawaban, ganti angka desimal (X) dengan angka hasil pembagian bulat. Ulangi pembagian tersebut selama bilangan X yang kita dapat belum sama dengan 0. Kita lihat contoh konversi bilangan desimal 12 dan 43 berikut :

12		43	
6	0	21	1
3	0	10	1
1	1	5	0
0	1	2	1
		1	0
		0	1

Maka $12_{10} = 1100_2$ dan $43_{10} = 101011_2$

Perhatikan bahwa jumlah angka biner akan sama dengan jumlah pembagian yang kita lakukan. Angka jumlah pembagian ini bisa kita peroleh secara langsung dengan cara mencari angka 2^n yang bernilai lebih besar dan paling mendekati bilangan desimal yang akan kita ubah. Maka *n* akan menyatakan jumlah pembagian yang harus kita lakukan.

```
Procedure KONVERSI_1(input angka : long
                     input/output hasil: string)
{ Keluaran digunakan string karena pada umumnya
  kapasitas untuk bilangan sangat terbatas
}
Deklarasi
   string hasil;
Algoritma
   hasil = "";
   bagi = 0;
   banding = 0;
   while (angka > 0)
      hasil <- string(angka mod 2) + hasil;
      angka = angka div 2;
   end while
```

Skema procedur Konversi tanpa optimasi

Kompleksitas waktu algoritma :

$T(n) = O(n)$

dengan n menyatakan eksponen terkecil dari 2 yang mempunyai nilai 2^n lebih besar dari angka desimal

Permasalahan akan muncul jika bilangan desimal yang akan kita ubah mempunyai nilai yang sangat besar. Jumlah pembagian yang harus kita lakukan pun akan semakin banyak. Tentu saja hal ini akan membuat kerja komputer menjadi semakin berat karena harus melakukan pembagian yang cukup banyak.

Lantas bagaimana cara kita memperbaiki hal tersebut? Salah satunya adalah dengan menggunakan sifat khusus seperti yang kita bahas pada sub-bab 2.1.2. Di sini saya akan memanfaatkansifat tersebut untuk memangkas jumlah pembagian yang harus dilakukan.

3.2 Optimasi Konversi dengan penerapan Algoritma Divide and Conquer

Salah satu cara otimasi yang bisa kita lakukan adalah dengan membagi bilangan desimal yang hendak diubah dengan angka 8 (bukan 2). Di sinilah prinsip algoritma Divide and Conquer kita gunakan untuk melakukan optimasi. Kita pecah-pecah angka desimal yang kita gunakan dengan cara membaginya dengan angka 8 secara berulang. Angka-angka sisa pembagian yang kita peroleh kemudian kita ubah ke dalam bilangan biner sebelum kita gabungkan menjadi hasil jawaban.

Karena angka pembagi yang kita pakai adalah 8 (2^3), maka kita dapat mengurangi jumlah pembagian yang kita lakukan menjadi ± 1/3 dari jumlah semula. Hal ini tentu saja akan sangat berpengaruh pada kinerja dan waktu yang diperlukan oleh komputer mengingat proses pembagian merupakan salah satu proses yang cukup rumit.

Tentu saja optimasi ini harus kita bayar dengan menangani konversi bilangan oktal ke biner. Akan tetapi jika kita gunakan teknik perbandingan (tanpa harus melakukan konversi secara manual), maka proses ini akan menjadi sangat mudah dan cepat. Perbandingan yang dimaksud adalah dengan membandingkan angka-angka dari bilangan oktal dengan angka oktal 0 – 7, kemudian menggantinya dengan angka-angka biner yang bersesuaian. Penerapan algoritmanya adalah dengan menggunakan sintaks `case of`.

Begitu juga dengan permasalahan pemakaian memori (kompleksitas ruang) yang lebih besar yang muncul akibat penggunaan algoritma *rekursif*. Karena pada proses *rekursif*-nya kita tidak menggunakan banyak variabel yang memerlukan tempat yang begitu besar, maka hal ini bisa kita abaikan.

Dengan penggunaan optimasi ini, maka seharusnya proses konversi akan lebih cepat karena pemangkasan jumlah pembagian yang dilakukan. Mengapa saya katakan "seharusnya"? Karena pada kenyataannya setelah saya coba kedua algoritma di atas pada bahasa Java, baik yang tanpa optimasi maupun yang menggunakan optimasi akan sama-sama menghasilkan jawaban kurang dari 1 *milisecond*. Saya sendiri sempat terkejut melihat hasil tersebut, mengingat proses pembagian yang terjadi bisa mencapai 84 kali, 63 kali untuk algoritma pertama dan 21 kali untuk algoritma kedua (saya memakai kedua algoritma dalam satu program).

```
Procedure KONVERSI_2(input angka : long)
{ Keluaran digunakan string karena pada umumnya
  kapasitas untuk bilangan sangat terbatas
}
Deklarasi
   string hasil;
Algoritma
   hasil = "";
   KONVERSI_DnC(angka, hasil);
```

Skema procedur utama Konversi dengan optimasi

```
Procedure KONVERSI_DnC( input angka : long,
            input/output hasil : string )

Deklarasi
   temp1,temp2  : long;
Algoritma
   if (angka <= 8) then
      case (angka) of :
         0 : hasil <- hasil + "000";
         1 : hasil <- hasil + "001";
         2 : hasil <- hasil + "010";
         3 : hasil <- hasil + "011";
         4 : hasil <- hasil + "100";
         5 : hasil <- hasil + "101";
         6 : hasil <- hasil + "110";
         7 : hasil <- hasil + "111";
   else
      temp1 <- angka div 8
         KONVERSI_DnC(temp1, hasil);
      temp2 <- angka mod 8
         KONVERSI_DnC(temp2, hasil);
```

Skema procedur rekursif dengan menerapkanAlgoritma *Divide and Conquer*

Kompleksitas waktu algoritma :

$T(n) = O(n/3)$

dengan n menyatakan eksponen terkecil dari 2 yang mempunyai nilai 2^n lebih besar dari angka desimal

Sangat disayangkan saya tidak bisa mengetahui secara pasti dengan akurasi waktu dalam satuan *mikrosecond*, karena bahasa Java maupun bahasa-bahasa lain pada umumnya tidak menyediakan fasilitas untuk menghitung waktu hingga presisi sekecil itu. Begitu pula angka-angka desimal yang hendak dikonversi, terbentur pada angka *integer* maksimal 2^{63}-1. Akan tetapi saya yakin kalau angka masukan yang mampu ditangani bisa jauh lebih besar dari itu, atau presisi satuan waktu yang digunakan mencapai satuan *mikrosecond* maka perbedaan waktu eksekusinya akan terlihat.

Berikut saya berikan tabel data proses konversi yang saya lakukan dengan menggunakan bahasa Java. Untuk data jumlah pembagian, 1 angka menunjukkan 1 kali pembagian dengan `div` dan 1 kali pembagian dengan `mod` sekaligus. Jadi sebenarnya terjadi 2 kali pembagian. Sedangkan untuk data jumlah perbandingan, pada kasus `case of` dalam algoritma dengan Optimasi saya anggap perbandingan yang terjadi hanya 1 kali saja. Kalaupun dianggap seperti kasus `if-else` berantai, dimana semakin ke bawah maka jumlah perbandingan yang dilakukan semakin banyak, jumlahnya akan tidak jauh berbeda dengan jumlah perbandingan yang terjadi pada algoritma biasa. Hal ini sudah saya coba sendiri.

Tabel 1. Data Proses Konversi

Angka masukan	**Agoritma biasa**		**Algoritma dengan Optimasi**	
	Pembagian	Perbandingan	Pembagian	Perbandingan
15	4	5	2	5
265	9	10	3	7
13256	14	15	5	11
879783	20	21	7	15
6346542134	33	34	11	23
465434354343459	49	50	17	35
9223372036854775807	63	64	21	43

Dari tabel data proses konversi jelaslah bahwa dengan optimasi yang dilakukan, jumlah proses pembagian bisa dipangkas hingga menjad 1/3 jumlah semula. Untuk kasus-kasus bilangan desimal yang ingin diubah sangatlah besar, maka optimasi ini akan sangat terasa manfaatnya, terutama dalam segi kompleksitas waktu yang dibutuhkan. Optimasi ini akan semakin bagus jika pada kasus case of, perbandingan yang dilakukan oleh compiler dan mesin bersifat hanya satu kali sebagaimana yang saya prediksikan. Dengan demikian maka data jumlah perbandingan di atas menjadi valid, dan seperti kita lihat jumlah perbandingan pun dapat kita pangkas sedemikian rupa.

IV. KESIMPULAN

Algoritma konversi sistem bilangan dengan menggunakan algoritma dengan optimasi yang menerapkan algoritma *Divide and Conquer* lebih mangkus daripada algoritma konversi dengan metode pembagian sisa biasa jika dilihat dari segi kompleksitas waktunya. Hanya saja optimasi ini diimbangi dengan kenaikan pada kompleksitas ruangnya, meskipun pengaruhnya tidak sebesar optimasi yang kita lakukan.

REFERENSI

[1] "Algorithms", http://en.wikipedia.org, 2006

[2] Rinaldi Munir, "Strategi Algoritmik". Institut Teknologi Bandung, 2004.

[3] Rinaldi Munir, "Matematika Diskrit". Institut Teknologi Bandung, 2004.

[4] Java Development Team, "The Java™ Tutorial", Sun Microsystem, 2006.

[5] S. Dasgupta, C.H. Papadimitriou, dan U.V. Vazirani, "Divide and Conquer Algoritm", 2006.

[6] Wayne Goddard, "Introduction to Algorithms", Clemson University, 2004

[7] Edward Burch, "Divide-and-Conquer Multiplication", http://ozark.hendrix.edu/~Eburch/csbsju/cs/, 2005