PENDEKATAN MASALAH *TOWER OF HANOI* DENGAN ALGORITMA *DIVIDE AND CONQUER*

Gia Pusfita

Program Studi Informatika

Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung

Jl. Ganesha 10, Bandung

Email : if15082@students.if.itb.ac.id

ABSTRAK

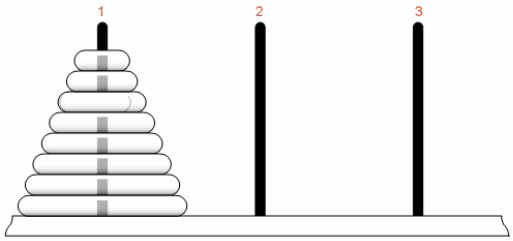
Ada berbagai teknik untuk menyelesaikan permasalahan *Tower of Hanoi*, namun cara yang cukup sangkil untuk menyelesaikannya adalah dengan menggunakan pendekatan algoritma *Divide and Conquer* yang sangat erat kaitannya dengan rekursifitas. Dengan membagi-bagi persoalan menjadi bagian-bagian yang lebih kecil dan mudah untuk dicari solusinya, permasalahan *Tower of Hanoi* menjadi lebih tereduksi. Dan dengan menggunakan metode rekursif penyelesaian masalah-masalah kecil yang memiliki pola yang sama dapat diringkas menjadi potongan kode yang lebih pendek. Dan kemudian penyelesaian dari sub-masalah tersebut dikombinasikan untuk menyelesaikan awal permasalahan

Kata kunci: *Divide and Conquer, Tower of Hanoi*

1. PENDAHULUAN

Dalam ranah ilmu informatika, program yang baik bukan hanya program yang dapat menyelesaikan masalah, tetapi juga menyelesaikan masalah dengan menggunakan algoritma yang efektif. Salah satu algoritma yang cukup sering digunakan utuk mereduksi masalah adalah algoritma *Divide and Conquer*. Dan berkenaan dengan hal tersebut pada kesempatan ini, saya berusaha menerapkan algoritma *Divide and Conquer* pada sebuah persoalan klasik yaitu *Tower* of Hanoi.

Permasalahan *Tower of Hanoi* muncul pada abad 19 di Eropa yang menceritakan tentang sebuah tugas di kuil Brahma. Beberapa pendeta diberikan kepingan-kepingan dan 3 buah tiang emas. Pada tiang pertama kita menumpukan 64 keping emas, dan setiap kepingan diatasnya selalu lebih kecil dari keping yang berada dibawah. Pendeta tersebut ditugaskan untuk memindahkan semua kepingan emas dari kepingan pertama ke kepingan ketiga, dengan syarat hanya satu keping yang dapat dipindahkan dalam satu waktu dan setiap keping tidak boleh ditempatkan diatas keping yang ukurannya lebih kecil. Pendeta tersebut diberi tahu bahwa ketika mereka akan selesai memindahkan 64 keping saat dunia kiamat.



Gambar 1. The *Tower* of Hanoi

2. METODE

Dasar pemikiran untuk menyelesaikan masalah tersebut adalah dengan memfokuskan perhatian kita bukan pada langkah pertama (memindahkan kepingan teratas pada suatu tempat), tetapi lebih pada langkah terberat yaitu memindahkan dari kepingan terbawah. Dan tidak ada cara lain untuk memindahkan kepingan terbawah sebelum 63 keping diatasnya dipindahkan terlebih dahulu, dan terlebih lagi, semua kepingan itu harus ada pada *tower* ke dua sehingga kita dapat memindahkan kepingan terbawah dari *tower* 1 ke *tower* 3. hal ini dikarenakan hanya satu keping saja yang boleh dipindahkan pada satu waktu dan kepingan terbesar(terbawah) tidak diperbolehkan berada diatas kepingan lain, dan pada saat kita memindahkan keping terbesar tersebut, tidak akan ada kepingan pada *tower* 1 dan 3.sehingga kita dapat merangkumkan langkah dari masalah ini dengan algoritma:

```
Move(63,1,2,3) // pindahkan 63 kepingan dari tower 1 ke
                  tower 2 (tower 3 sebagai tempat
                  penyimpanan sementara)

Cout << "pindahkan keeping ke 64 dari tower 1 ke tower
3" <<endl ;

Move(63,2,3,1) // pindahkan 63 keping dari tower 2 ke
tower             3 (tower 1 sebagai tempat penyimpanan
                  sementara)
```

2.1 Reduksi Umum

Sekarang kita sudah mempunyai langkah kecil menuju penyelesaian masalah, hanya saja ada satu masalah kecil lagi dimana kita harus mendeskripsikan bagaimana memindahkan 63 keping sebanyak 2 kali. Ini adalah sebuah langkah yang signifikan, dimana tidak ada alasan mengapa kita tidak memindahkan 63 keping tersisa tersebut dengan langkah yang sama.

2.2 *Divide and Conquer*

Algoritma ini menurut pengertiannya dapat didefinisikan sebagai berikut. *Divide* adalah membagi masalah menjadi upa-masalah yang memiliki kemiripan dengan masalah semula namun berukuran lebih kecil[1]. Dan *conquery* adalah memecahkan masing-masing upa-masalah sehingga membentuk solusi masalah semula[1].
Divide and Conquer merupakan salah satu dasar pemikiran rekursif. Kita sudah mendeskripsikan langkah utama dan menjelaskan bahwa akhir persoalan dapat diselesaikan denggan cara yang sama. Ini juga merupakan ide dasar dari divide and conquer dimana untuk menyelesaikan permasalahan, kita membagi pekerjaan kedalam bagian-bagian yang semakin kecil, setiap bagian memiliki cara penyelesaian masalah yang berpola sama dan lebih mudah dari masalah utama.
Untuk menuliskan algoritma secara lebih formal, kita perlu mengetahui pada setiap langkah *tower* mana yang akan digunakan sebagai tempat penyimpanan sementara, dan kita akan memanggil fungsi dengan spesifikasi :

Void move(int count,int start, int finish, int temp)

Prekondisi :	kita sudah mengetahui jumlah kepingan pada awal tower. Bagian teratas tower(jika banyak) dalam setiap temporary tower dan final tower lebih besar dari setiap kepingan saat awal.
Postkondisi:	kepingan teratas pada awal permainan sudah dipindahkan ke akhir; temp (digunakan sebagai temporary storage) sudah dikembalikan ke kondisi awal.

Sekarang tugas kita adalah untuk menyelesaikan persoalan dalam jumlah langkah yang berhingga dan oleh karena itu harus ada cara agar kerekursifan berhenti. Aturan yang paling mudah untuk menghentikan refkursif adalah ketika sudah tidak ada lagi kepingan untuk dipindahkan, dan sekarang kita dapat membuat program yang lebih lengkap untuk mendefiniskan aturan ini. Program utamanya adalah :

```
Const int disk = 64;

Void move(int count,int start,int finish,int temp)
/*Prekondisi  : tidak ada
  Postkondisi : simulasi menara Hanoi dapat berhenti
*/
Main()
{
   move(disks,1,3,2);
}
```

fungsi rekursif yang melakukan tugas ini adalah :

```
void move(int count, int start, int finish, int temp)
{
    if (count > 0) {
          move(count - 1, start, temp, finish);
          cout << "Move disk " << count << "
          from " << start << " to " << finish << "."
          << endl;
          move(count - 1, temp, finish, start);
        }
}
```

2.3 Penelusuran Program

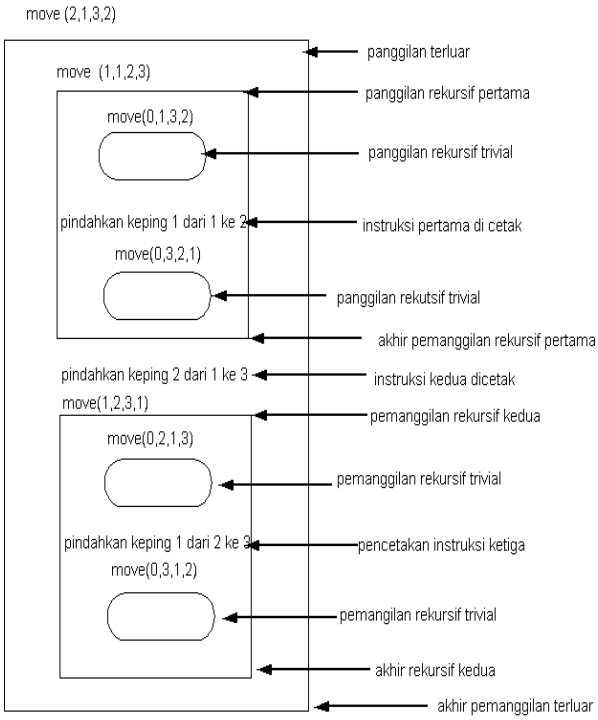
Salah satu cara yang berguna dalam mempelajari fungsi rekursif ketika diaplikasikan pada contoh yang kecil adalah dengan menelusuri jejak dari setiap eksekusi. Sebagaimana penelusuran yang ditampilkan pada gambar 2 pada masalah *Tower of Hanoi* dengan kasus saat jumlah dari kepingan adalah 2. setiap box pada diagram menunjukan apa yang terjadi ketika fungsi pertama kali dipanggil. Pemanggilan funsi yang paling luar adalah move(2,1,3,2) (pemanggilan dilakukan oleh main program) hasilnya adalah eksekusi dari 3 pernyataan berikut, sebagaimana pernyataan dari luar box dalam diagram.

```
Move(1,1,2,3);  // pindahkan 1 keping dari
          tower 1 ke
          tower 2 (menggunakan tower
          3)

Cout << “pindahkan keping 2 dari tower 1 ke tower
3” << endl;

Move(1,2,3,1)    // pindahkan 1 keping dari
          tower 2 ke tower 3
          (menggunakan tower 1).
```

Pernyataan pertama dan ketiga membuat sebuah panggilan yang rekursif. Pernyataan

Move(1,1,2,3)

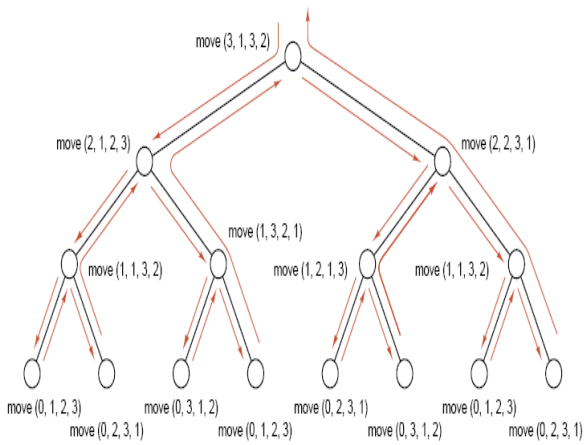
memanggil fungsi move lagi dari puncak, tetapi sekarang dengan menggunakan parameter baru. Oleh karena itu penryantaan ini menghasilkan pola dasar yang sama dengan eksekusi dari tiga pernyataan berikut, pernyataan terlihat sebagai pernyataan pada kotak dalam pertama.



Gambar 2. Penelusuran rekursifitas hanoi untuk kepingan ke 2

Move(0,1,3,2);
Cout << "pindahkan keping 1 dari *tower* 1 ke *tower* 2"<<endl;
Move(0,3,2,1);

Setelah box yang berkorespodensi dengan panggilan fungsi move(1,1,2,3) mengeluarkan output, kemudian box yang kedua berkorespondensi dengan memanggil fungsi move(1,2,3,1). Tetapi sebelum pernyataan ini dicapai, terdapat dua buah panggilan rekursif lagi datang dari kotak dalam pertama. Oleh karena itulah, kita harus dapat mengekspansi fungsi move(0,1,3,2). Tetapi fungsi tersebut tidak melakukan apa-apa karena parameter count adalah 0; sehingga fungsi move(0,1,3,2) tidak mengeksekusi fungsi atau pernyataan lain. Kita lihat semuanya itu sebagai korespondensi dari box kosong pertama dalam diagram.
Setelah panggilan kosong ini, muncul pernyataan output seperti yang terlihat dalam box terdalam pertama dan kemudian muncul panggilan fungsi lain yang tidak melakukan apa-apa Pemanggilan fungsi ini adalah akhir dari pemanggilan fungsi move(1,1,2,3), sehingga kembali lagi ketempat dimana ia dipanggil. Pernyataan berikutnya kemudian adalah pernyataan output dari box terluar, dan pada akhirnya pemanggilan fungsi move(1,2,3,1) selesai dikerjakan. Pemanggilan fungsi ini menghasilkan *statement* yang terlihat dari kotak dalam yang kedua dimana pada gilirannya akan diekspansi sebagai box kosong seperti yang ditunjukkan. Sekarang kita beralih pada kakas lain untuk merepresentasikan pemanggilan rekursif, sebuah kakas uang mengatur panggilan berulang lebih efektif dari yang dapat dilakukan oleh penelusuran program. Kakas ini adalah pohon rekursif.
Pohon rekursif untuk *Tower of Hanoi* dan laju eksekusi yang direpresemtasikan dengan warna merah digambarkan pada gambar 3. Dalam gambar tersebut hanya dipergunakan tiga buah keping untuk merepresentasikan *Tower of Hanoi* dengan alasan dengan menggunakan tiga buah keping sudah cukup untuk merepresentasikan masalah *Tower of Hanoi* dengan menggunakan pendekatan *Divide and Conquer*.



Gambar 3. Pohon rekursif

Pada gambar di atas dapat kita lihat bahwa *state* paling atas merupakan *state* awal dimana ketiga disk berada di tiang paling kiri, dan dua tiang lainnya kosong. Dan *state* yang berada paling kanan paling bawah menunjukan *state* akhir atau *state* yang ingin dicapai dimana ketiga keping sudah berada ditiang paling kanan, dan dua tiang lainnya kosong.
Dapat kita lihat banyak sekali kemungkinan yang ada untuk mencapai kondisi akhir, akan tetapi solusi yang

paling cepat dalam memecahkan masalah ini terdapat pada lintasan atau jalan menurun dari *state* paling atas menurun ke atas menuju *state* paling kanan bawah.
Dalam divide and conquer biasanya untuk menyelesaikan masalah digunakan pembagian masalah menjadi sub-masalah terus menerus hingga tercapai primitif sub-masalah dimana permasalahan sudah tidak dapat dibagi-bagi lagi dan dapat diselesaikan dengan cara yang simple.
Ide yang muncul dari penyelesaian masalah ini adalah bagaimana kita menemukan state tujuan dari posisi tiga keping tersebut sebagaimana ditunjukan dalam diagram pohon.
Sehingga dengan menggunakan metode divide and conquer kita dapat membagi ruang solusi yang ada menjadi sub-masalah primitif sehingga ruang solusi yang ada menjadi lebih kecil.

III. ANALISA

Sebagai catatan, program yang dibuat untuk *Tower of Hanoi* tidak hanya menghasilkan pemyelesaian yang lengkap, tetapi juga menghasilkan solusi terbaik yang mungkin dicapai dan faktanya solusi yang dapat ditemukan adalah :

```
Pindahkan keping 1 dari tower 1 ke tower 2.
Pindahkan keping 1 dari tower 2 ke tower 3.
Pindahkan keping 1 dari tower 3 ke tower 1.
```

Untuk menunjukan solusi yang tidak dapat direduksi perlu diperhatikan bahwa pada setiap tahap pekerjaan yang dilakukan dapat dirangkumkan sebagai perpindahan beberapa keping dari satu *tower* ke *tower* lain. Dan tidak ada cara lain untuk menyelesaikan pekerjaan ini terkecuali dengan memindahkan semua keping kecuali dengan diawali dengan memindahkan keping terbawah terlebih dahulu, kemudian mungkin membuat beberapa perpindahan yang berlebihan, kemudian memindahkan keping terbawah, lalu mungkin membuat perpindahan lagi dan pada akhirnya memindahkan keping teratas lagi.
Mari kita lihat seberapa banyak perpindahan/ rekursif yang dieksekusi sebelum beralih lagi ke kondisi awal dan pada akhirnya berhenti. Pertama kali fungsi move dipanggil, nilai count diisi dengan 64 sesuai dengan jumlah kepingan yang ada, dan setiap kali terjadi pemanggilan rekursif, nilai dari count berkurang sebanyak satu. Oleh karena itu jika kita mengekslude pemanggilan sampai dengan count == 0, dimana pada kondisi tersebut program tidak akan mengeksekusi apa-apa, maka kita mempunyai kedalaman rekursi sebanyak 64. dan jika digambarkan dengan diagram pohon rekursif maka akan memiliki 64 level. Tidak termasuk daun, setiap verteks dihasilkan dari dua pemanggilan rekursif, dan jumlah dari vertice dari setiap level menjadi dua kali lipat dari level yang diatasnya.
Dari pohon rekursif tersebut (meskipun sangat besar untuk digambar) kita dapat menghitung dengan mudah berapa banyak instruksi yang dibutuhkan untuk memindahkan 64 keping. Jumlah keseluruhan bukan daun(perpindahan yang dilakukan) adalah

$$1 + 2 + 4 + ... + 2^{63} = 2^0 + 2^1 + 2^2 + ... + 2^{63} = 2^{64} - 1$$

karena jumlah perpindahan yang dibutuhkan untuk memindahkan 64 keping adalah 2^{64} - 1. dan jika dikalkulasikan butuh waktu yang sangat lama untuk melakukan hal tersebut.
Harus dicatat bahwa penyelesaian masalah Hanoi yang utama adalah karena keterbatasan waktu, bukan karena kekurangan tempat. Tempat yang dibutuhkan hanyalah untuk tetap menjaga 64 panggilan rekursif, namun waktu yang diperlukan adalah dengan sebanyak 2^{64} kalkulasi. Dan belum ada komputer yang dapat menyelesaikan secara penuh masalah ini.

IV. KESIMPULAN

Dalam ilmu komputer, *Divide and Conquer* merupakan salah satu paradigma desain algoritma yang penting dan bekerja secara rekursif dengan memcah masalah menjadi dua atau lebih sub-masalah dengan type yang sama atau berhubungan hingga sub-masalah submasalah tersebut menjadi cukup mudah diselesaikan. Penyelesaian setiap sub-masalah kemudian dikombinasikan untuk memberi solusi dari awal permasalahan, algoritma *Divide and Conquer* sangat terikat dengan relasi kerekursifan antara fungsi.
Hal lain yang membantu memudahkan kita dalam penyeleaian masalah, terutama tentang menuliskan algoritma khusus dalam bentuk rekursif atau nonrekursif , starting point yang baik adalah membuat pohon rekursif untuk menggambarkan ruang solusi yang ada.
Dalam penerapannya, divide and conquer dapat digunakan untuk mereduksi masalah Tower of Hanoi yaitu memecah masalah penyusunan tower yang dalam makalah ini direpresentasikan dengan tiga keping untuk dibagi menjadi tiga sub-masalah terus menerus hingga tercapat sub-masalah primitif yang dapat diselesaikan dengan mudah. Penyelesaian masalah ini dapat juga diaplikasikan untuk tower dengan jumlah kepingan lebih dari tiga. Dalam hal ini persoalan dipilih tiga keping untuk meminimumkan ruang solusi yang ada.

REFERENSI

[1] Munir, rinaldi,. “Strategi Algoritmik”, Teknik Informatika ITB. Bandung. 2007
[2] Kruse, Robert L, “ Data Structure and Program Design in

C++", Prentice Hall, 2000.
[3] wikipedia, http://en.wikipedia.org/wiki/Divide_and_conquer_algorithm. 18 Mei 21.00
[4] wikipedia, http://en.wikipedia.org/wiki/*Tower_of_Hanoi* . 18 Mei 21.05