Penerapan Relasi Rekursif dan Matriks dalam Partisi Bilangan Bulat

Gilang Ardyamandala Al Assyifa (13515096)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13515096@std.stei.itb.ac.id

***Abstrak* – Makalah ini membahas mengenai penyelesaian persoalan partisi bilangan bulat melalui pendekatan rekursif dan penggunaan matriks. Penyelesaian dengan pendekatan rekursif saja menghasilkan program yang kurang mangkus untuk skala masukan yang besar, namun setelah dikombinasikan dengan penggunaan matriks, program menjadi berjalan jauh lebih cepat dalam menentukan partisi bilanan *n*.**

***Kata kunci* – matriks, partisi bilangan, rekursif, teori bilangan.**

I. PENDAHULUAN

Di era sekarang ini, bilangan sudah menjadi bagian yang tak terpisahkan dari kehidupan manusia, banyak sekali penerapan bilangan yang membawa manfaat besar bagi kehidupan manusia, komputer misalnya, komputer sangat erat kaitannaya dengan bilangan biner (basis 2) dan hexadesimal (basis 16) dalam pengoperasiannya. Bahkan untuk melakukan hal sederhana seperti mencacah barang pun, kita tetap tidak bisa lepas dari pemanfaatan bilangan karena memang sejatinya bilangan digunakan untuk pencacahan dan pengukuran.

Dari berbagai jenis bilangan, bilangan bulat bisa dikatakan sebagai bilangan yang paling sering kita gunakan sehari-hari, bilangan bulat merupakan bilangan yang bisa ditulis tanpa komponen pecahan, bilangan bulat terdiri atas bilangan cacah $(1,2,3,\dots)$, nol (0), dan cacah negatif $(-1,-2,-3,\dots)$.

Dalam bidang matematika, sangat banyak persoalan yang berkaitan dengan bilangan bulat, contohnya seperti bilangan prima, barisan fibonacci, aritmatika modulo, dan partisi bilangan bulat.

Algoritma dengan berbagai pendekatan pun dibuat untuk menyelesaikan pesoalan tersebut misalnya pendekatan dengan relasi rekurens dalam penyelesaian masalah partisi bilangan bulat.

II. DASAR TEORI

2.1 Teori Rekursif

Rekursif merupakan cara pendefinisian sebuah objek menggunakan terminologi dirinya sendiri. Objek yang didefinisikan bisa berupa gambar, video, prosedur, fungsi, dan lainnya.

Gambar 1. Contoh gambar yang rekursif
sumber : dokumen pribadi, arsip tpbcup

Dalam bidang informatika dan matematika, definisi rekursif sering digunakan dalam fungsi. Suatu fungsi dikatakan rekursif jika definisi fungsinya mengacu pada dirinya sendiri.

Fungsi rekursif disusun oleh dua bagian dasar:

1. Basis

   Basis merupakan bagian yang berisi nilai awal yang tidak mengacu pada dirinya sendiri. Bagian ini sekaligus menghentikan definisi rekursif dan memberikan sebuah nilai yang terdefinisi pada fungsi rekursif.
2. Rekurens

   Bagian yang mendefinisikan argument fungsi dalam terminologi dirinya sendiri. Setiap kali fungsi rekursif mengacu pada dirinya sendiri, argumen/parameter dari fungsi harus mendekati basis (nilai awal) agar rekurens berhenti dan mengeluarkan suatu nilai.

Contoh fungsi rekursif adalah fungsi faktorial berikut:

$$f(n) = \begin{cases} 1, & n = 0 \text{ (basis)} \\ n \times f(n-1), & n > 0 \text{ (rekurens)} \end{cases}$$

Jika kita menghitung $f(4)$ maka langkahnya seperti berikut:

$$(1)\ f(4) = 4 \times f(3)$$
$$(2)\quad f(3) = 3 \times f(2)$$
$$(3)\quad f(2) = 2 \times f(1)$$

(4) $f(1) = 1 \times f(0)$

(5) $f(0) = 1$

Pada baris (5) kita medapati bahwa nilai $f(0)$ terdefinisi secara langsung dengan nilai 1. Dengan melakukan runut-balik (*backtracking*) dari baris (5) ke baris (1), kita akan mendapatkan hasilnya:

$(5')$ $0! = 1$

$(4')$ $1! = 1 \times 0! = 1 \times 1 = 1$

$(3')$ $2! = 2 \times 1! = 2 \times 1 = 2$

$(2')$ $3! = 3 \times 2! = 3 \times 2 = 6$

$(1')$ $4! = 4 \times 3! = 4 \times 6 = 24$

Maka $f(4) = 24$.

2.2 Teori Matriks

Matriks didefinisikan sebagai susunan skalar elemen-elemen dalam bentuk baris dan kolom. Misalkan matriks A berukuran m baris dan n kolom $(m \times n)$, maka representasinya sebagai berikut:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & & a_{21} \\ \vdots & & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

Entri a_{ij} disebut elemen matriks pada baris ke-i dan kolom ke-j. Jika suatu matriks memiliki jumlah baris yang sama dengan jumlah kolomnya, maka matriks tersebut kita sebut sebagai matriks bujursangkar.

Matriks banyak digunakan untuk representasi yang bersifat dua dimensi seperti gambar, foto, maupun peta dua dimensi.

III. PARTISI BILANGAN BULAT

A. Deskripsi Permasalahan

Partisi bilangan bulat dimaksudkan jika terdapat bilangan positif n, maka cara menguraikan n sebagai jumlah dari beberapa bilangan bulat positif yang tidak menaik disebut sebagai partisi bilangan bulat n. Sebagai contoh, bilangan bulat 5 dapat diuraikan menjadi 7 cara, yaitu:

$5,$
$4 + 1,$
$3 + 2,$
$3 + 1 + 1,$
$2 + 2 + 1,$
$2 + 1 + 1 + 1,$ dan
$1 + 1 + 1 + 1 + 1.$

Jika $p(n)$ menyatakan banyaknya partisi bilangan dari n, maka $p(5) = 7$. (selanjutnya kita akan tetap menggunakan $p(n)$ untuk menyatakannya hal tersebut.)

B. Sejarah

Dalam sejarah, ternyata partisi bilangan bulat pertama kali ditanyakan oleh Leibniz kepada J. Bernouli melalui suratnya pada 1674. Pada terminologi modern, ia bertanya mengenai banyaknya partisi pada suatu bilangan bulat. Dia mengamati bahwa ada tiga partisi dari 3 $(3,\ 2+1,$ dan $1+1+1)$ sebagaimana ada lima partisi dari 4, tujuh partisi dari 5, dan seterusnya.

Setelah itu muncullah beberapa orang yang berkontribusi besar dalam perkembangan partisi bilangan, seperti: Euler, Sylvester, MacMahon, Rogers, Hardy, Ramanujan, dan Rademacher. Masing-masing dari mereka punya andil sendiri dalam perkembanyak partisi bilangan bulat.

IV. IMPLEMENTASI REKURSIF DAN MATRIKS DALAM PARTISI BILANGAN

4.1. Implementasi Rekursif

Sampai dengan saat ini sudah banyak algoritma yang dibuat untuk menyelesaikan persoalan partisi bilangan, dalam hal ini untuk mencari nilai $p(n)$, namun di makalah ini penulis akan menggunakan metode yang melibatkan relasi rekursif dalam penyelesaiannya, adapun ide dari algoritma ini adalah sebagai berikut:

Jika kita definisikan $p(n,m)$ sebagai banyaknya partisi bilangan bulat n dengan hanya menggunakan bilangan yang lebih kecil atau sama dengan m, maka:

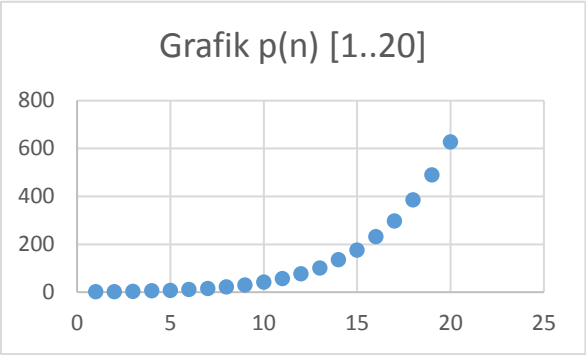
$$p(n,m) = \begin{cases} 0 & ,n < 0 \text{ atau } m < 1 \text{ (basis } error) \\ 1 & ,n < 2 \text{ atau } m = 1 \text{ (basis)} \\ p(n-m,m) + p(n,m-1), & \text{(rekurens)} \end{cases}$$

Fungsi rekursif itu benar dikarenakan, misalkan kita

Basis *error* digunakan untuk menangani kasus yang tidak terdefinisi karena tidak ada $p(n,m)$ untuk $n < 0$ atau $m < 1$. Basis *error* tersebut juga harus mengirimkan nilai 0 agar tidak memengaruhi keluaran fungsi rekursif tersebut. Berikut implementasinya dalam bahasa pemrograman C:

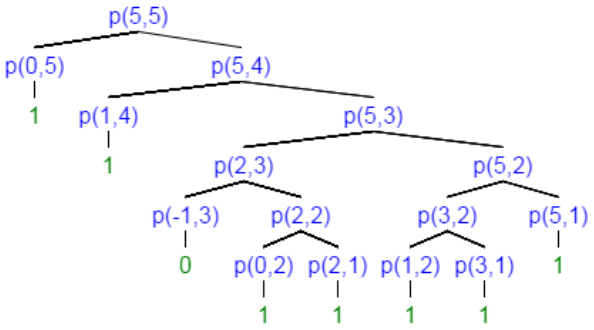
```
int partition (short n, short m) {
    if (n < 0 || m < 1) {
         return 0; /* basis error */
    } else if (n < 2 || m == 1) {
         return 1; /* basis */
    } else { /* rekurens */
         return   partition(n,   m-1)   +
partition(n - m, m);
     }
}
```

Berikut grafik yang dihasilkan dari uji coba $p(1) = 1$ sampai $p(20) = 627$,



Sumber : dokumen pribadi

Setelah dicoba dengan masukan n yang lebih besar ternyata program signifikan melambat, saat $n = 100$ program masih bisa menanganinya dalam waktu singkat, namun saat $n = 200$, programnya tak kunjung mengeluarkan hasil dari $p(200)$ dalam waktu yang cukup lama, untuk itu mari kita analisis program ini dengan menggunakan pohon biner sebagai representasi rekursif program tersebut di bawah ini:



Gambar 2. Pohon yang merepresentasikan p(5,5)
generator tree : http://mshang.ca/syntree/

Ada dua alasan mengapa algoritma tersebut kurang magkus, yang pertama, dari pohon biner tersebut terlihat bahwa terdapat pemanggilan $p(-1,3)$ yang mana tadi kita batasi sebagai basis *error*, ternyata untuk n yang semakin membesar, banyaknya $p(n,m)$ yang masuk ke basis $error$ juga semakin banyak sehingga menambah waktu runut-balik fungsi rekursif. Yang kedua, ternyata untuk n yang semakin membesar pula, banyak terdapat pemanggilan $p(n,m)$ yang sama berkali-kali, misalkan saja untuk $p(6)$, dalam rekurensnya terdapat pemanggilan $p(2,2)$ yang berkali-kali, hal ini tentunya juga berpengaruh ke performa algoritma dari segi waktu maupun memori.

4.2. Implementasi dengan Rekursif dan Matriks

Implementasi ini sedikit berbeda dengan implementasi (4.1) karena menggunakan matriks, matriks yang digunakan berupa matriks bujursangkar. Pada implementasi ini relasi rekursif yang digunakan tidak *pure*, melainkan kita menjadikan matriks tersebut sebagai objek yang direkursifkan. Dalam hal ini matriks secara rekursif mengisi dirinya perlahan-lahan dengan mengacu pada blok matriks yang sudah terisi sebelumnya. Pengisian tersebut berakhir sampai blok yang diminta terisi (jika yang diminta $p(n)$, maka sampai $p(n,n)$ terisi).

Berikut implementasinya dan program utamanya dalam bahasa pemrograman C:

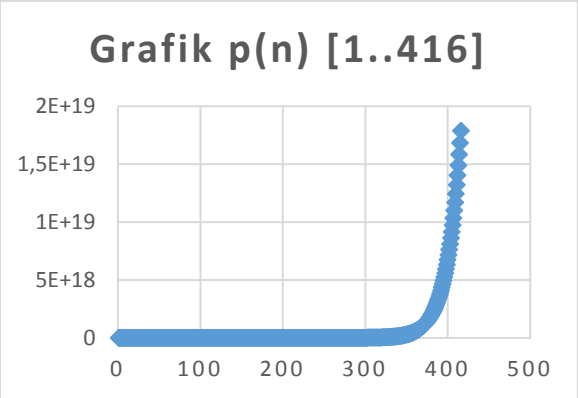
```
/* Program representasi rekursif dn
matriks */
#include <stdio.h>

#define p(k,n) arr[k][n]

/* Fungsi p(n) */
long long unsigned partition (int n) {
    long long unsigned arr[n+1][n+1];
    int i,j,x;
    for (i = 0; i <= n; i++) { /* proses
baris */
        for (j = 1; j <= n; j++) { /*
proses kolom */
            if (i < 2 || j < 2) {
                p(i,j) = 1; /* Blok
matriks yang menjadi basis */
            } else {
                p(i,j) = 0;
                x = i - 1;
                while (x >= 0 && (i-x) <=
j) {
                    p(i,j) += p(x,i-x);
/* Blok matriks mengacu ke blok matriks
lainnya */
                    x--;
                }
            }
        }
    }
    return (p(n,n));
}

/* Program utama */
int main () {
    printf("%llu", partition(416));
    return 0;
}
```

Dengan implementasi seperti ini, program tidak perlu menghitung $p(n,m)$ yang sama berkali-kali seperti pada implementasi (4.1), karena setelah program mendapatkan nilai $p(n,m)$ maka nilai tersebut akan diisikan di matriks, sehingga jika nantinya blok matriks yang lain membutuhkan nilai tersebut, program tidak perlu menghitungnya dari awal, cukup mengambil dari blok matriks yang ada.

Hasilnya, program dengan implementasi ini jauh lebih cepat dari program dengan implementasi (4.1), sebagai buktinya program tersebut berhasil dengan cepat

mendapatkan nilai dan grafik berikut:

$$p(416) = 17.873.792.969.689.876.004$$



Sumber : dokumen pribadi

Sebagai informasi bahwa $p(416)$ sudah merupakan $p(n)$ tertinggi yang bisa dicapai dengan *long long unsigned* pada bahasa C, jika ingin lebih besar lagi maka harus menggunakan tipe data yang lain ataupun menggunakan bahasa pemrograman yang lain yang mendukung *BigInteger*.

4.3. Analisis Lebih Lanjut

$p(416)$ merupakan notasi untuk menggambarkan banyaknya partisi yang bisa dibentuk dari bilangan bulat n. Dari dua grafik di atas, bisa kita lihat bahwa $p(n)$ bergerak secara eksponsial membesar, untuk $p(416)$ saja, jumlah digitnya sudah mencapai 26 digit yang mana *long long unsigned* pada bahasa C sudah tidak bisa lagi melanjutkannya. Namum dilihat dari segi programnya (4.2), eksekusi program tidak melambat secara eksponensial. Dari hasil percobaan, waktu eksekusi dari $p(416)$ kira-kira dua kalinya dari waktu eksekusi $p(200)$. Jadi program ini sudah cukup bagus dalam menyelesaikan persoalan partisi bilangan ini, hanya saja butuh tipe data yang lebih besar untuk menampung hasil yang lebih besar juga.

V. KESIMPULAN

Ternyata relasi rekursif dapat digunakan untuk menyelesaikan persoalan partisi bilangan n walaupun programnya berjalan kurang mangkus. Tetapi setelah dikombinasikan dengan penggunaan matriks, ternyata programnya jadi berjalan lebih cepat.

VI. APENDIKS

```
p(396) = 52536651244169751 63
p(397) = 55892332025954044 88
p(398) = 59457901147078745 97
p(399) = 63246214825042943 25
p(400) = 67270900517410419 26
p(401) = 71546402226539423 21
p(402) = 76088028433398792 69
p(403) = 80912002764844655 81
p(404) = 86035517593486550 60
p(405) = 91476790688591176 02
p(406) = 97255125137420217 29
p(407) = 10339097267123947241
p(408) = 10990600063775926994
p(409) = 11682316277192317780
p(410) = 12416677403151190382
p(411) = 13196258966925435702
p(412) = 14023788883518847344
p(413) = 14902156290309948968
p(414) = 15834420884488187770
p(415) = 16823822787139235544
p(416) = 17873792969689876004
```

Gambar 3.

20 p(n) terbesar yang bisa direpresentasikan oleh long long unsigned dalam bahasa C

VII. UCAPAN TERIMA KASIH

Syukur alhamdulillah penulis ucapkan kepada Tuhan yang Maha Esa karena telah memberikan kesempatan untuk menyelesaikan makalah ini. Selanjutnya penulis berterima kasih kepada orang tua yang selalu mendukung penulis. Penulis juga berterima kasih kepada Bapak Rinaldi Munir selaku dosen yang telah mengajarkan materi matematika diskrit kepada penulis, mata kuliah ini merupakan mata kuliah yang paling penulis sukai di semester 3. Terakhir, penulis ingin berterima kasih kepada pembaca makalah ini, semoga makalah ini dapat bermanfaat bagi pembaca, aamiin.

VIII. REFERENSI

[1] Munir, Rinaldi, *Matematika Diskrit edisi 3*, penerbit INFORMATIKA Bandung: 2010

[2] Herbert S. Wilf, "Lectures on Integer", University of Pennsylvania, 2000.

[3] https://bermatematika.net/2016/09/19/partisi-bilangan-asli/, diakses pada tanggal 08-09 Desember 2016.

[4] http://mathworld.wolfram.com/PartitionFunctionP.html, diakses pada tanggal 09 Desember 2016 pukul 09.30.

[5] https://www.math.psu.edu/vstein/alg/antheory/preprint/andrews/chapter.pdf, diakses pada tanggal 09 Desember 2016, pukul 12.21

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 9 Desember 2016

Gilang Ardyamandala Al Assyifa (13515096)