Pengoptimalan Penyelesaian Persoalan Partisi Bilangan Bulat dengan *Memoization* dan Pemrograman Dinamis

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

***Abstrak* - Makalah ini membahas mengenai pemanfaatan metode *memoization* dan pemrograman dinamis dalam mengoptimalkan penyelesaian, makalah ini membuktikan bahwa memoization dan pemrograman dinamis sangat efektif dalam mengoptimalkan penyelesaian persoalan.**

Kata kunci – partisi bilangan, teori bilangan, memoization, pemrograman dinamis.

I. PENDAHULUAN

Di abad-21 ini, bilangan sudah menjadi kebutuhan dari umat manusia, tidak bisa dipungkiri bahwa bilangan membawa banyak manfaat bagi kehidupan manusia. Bilangan digunakan manusia untuk mencacah, menghitung, menyatakan kuantitas, dan lainnya.

Dari beberapa jenis bilangan yang ada, bisa dikatakan bahwa bilangan bulat merupakan bilangan yang paling sering digunakan atau dimanfaatkan, untuk komputer misalnya, sistem bilangan biner (basis 2) diimplementasikan untuk *software* maupun *hardware*-nya.

Dalam matematika, terdapat suatu cabang ilmu yang mempelajari bilangan khususnya bilangan bulat, yaitu teori bilangan. Contoh bilangan bulat yang dipelajari di teori bilangan meliputi bilangan prima, barisan fibonacci, simbol jacobi, maupun partisi bilangan bulat yang akan dibahas pada makalah ini.

Sampai saat ini, sudah banyak algortima yang dibuat untuk menyelesaikan permasalahan partisi bilangan bulat. Dalam makalah ini, penulis akan menggunakan metode *memoization* dan pemrograman dinamis untuk menyelesaikan persoalan tersebut.

II. DASAR TEORI

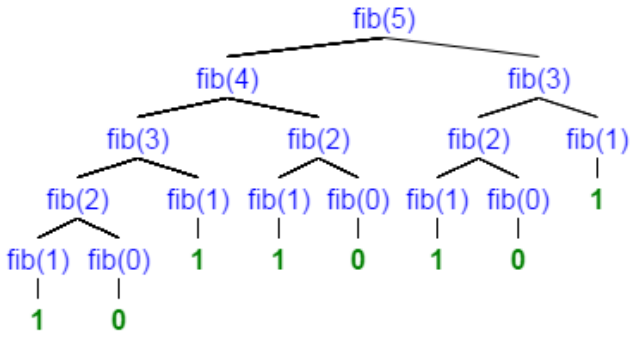
A. *Memoization*

Memoization adalah metode pengoptimalan algoritma yang mana program lebih memilih untuk menggunakan sub-solusi yang telah dihitung sebelumnya dibandingkan dengan menghitung ulang dari awal/dasar. Jenis algoritma yang biasa dioptimalkan dengan *memoization* ialah algoritma yang menggunakan relasi rekurens. Implementasi *memoization* umumnya menggunakan *hash-map*.

Agar semakin jelas, mari kita lihat penerapannya dalam persoalan bilangan fibonacci ini, bilangan fibonacci ke-n dengan n >= 0 didefinisikan sebagai berikut:

$$fib(n) = \begin{cases} 0, & n = 0\ (basis) \\ 1, & n = 1\ (basis) \\ fib(n-1) + fib(n-2), & n > 1\ (rekurens) \end{cases}$$

Jika kita mengilustrasikan pohon yang terbentuk dari penghitungan $fib(5)$, maka hasilnya seperti ini:

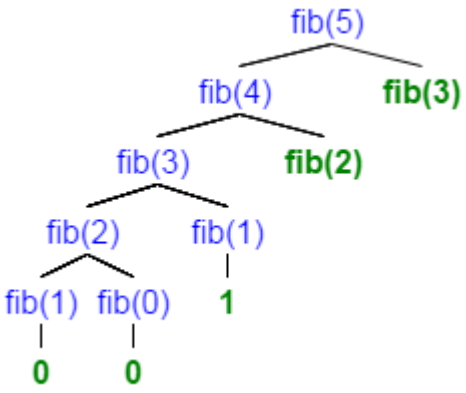


Gambar 1. Pohon yang merepresentasikan $fib(5)$ dengan rekurens biasa. *Generator tree : http://mshang.ca/syntree/*

Perhatikan bahwa $fib(3)$ dipanggil 2 kali, $fib(2)$ dipanggil 3 kali, $fib(1)$ dipanggil 5 kali, dan $fib(0)$ dipanggil 3 kali. Algoritma rekurens tersebut tentu tidaklah mangkus karena memiliki kompleksitas yang eksponensial.

Disinilah *memoization* mengambil perannya, selama perhitungan, nilai-nilai yang telah dihitung disimpan dengan struktur data tertentu, sehingga jika nanti membutuhkan nilainya program cukup menggunakan nilai yang telah disimpan tersebut, tidak perlu lagi menghitunggnya sampai basis/dasar. Metode tersebut tentu sangat memangkas waktu dan tempat yang digunakan.

Perhatikan pohon dari $fib(5)$ berikut setelah diterapkan *memoization*:



Gambar 2. Pohon yang merepresentasikan $fib(5)$ dengan *memoization*. *Generator tree : http://mshang.ca/syntree/*

B. *Pemrograman Dinamis*

Pemrograman dinamis adalah metode untuk memecahkan masalah dengan cara menguraikan solusi menjadi sekumpulan tahapan (*stage*) sehingga solusi persoalan bisa dipandang sebagai serangkaian keputusan yang berkaitan [7]. Pemrograman dinamis kerap kali digunakan untuk mengoptimalkan algoritma-algoritma yang terksesan *straightforward*. Implemetasi program dinamis sering kali menggunakan tabel (baik itu dua dimensi ataupun lebih) sehingga penggunaan memorinya lebih terkontrol.

Adapun karakteristik persoalan program dinamis adalah sebagai berikut:

1. Persoalan dapat dibagi menjadi beberapa tahap (*stage*) yang pada setiap tahapnya diambil suatu keputusan.
2. Masing-masing tahap terdiri dari sejumlah status yang berhubungan dengan tahap tersebut.
3. Solusi pada setiap tahap dibangun dari hasil solus tahap sebelumnya.

Ada dua pendekatan program dinamis, yaitu:

1. Program dinamis maju (*forward*), bergerak dari tahap 1, terus maju ke 2. 3, dan seterusnya sampai tahap n (solusi global).
2. Program dinamis mundur (*backward*), bergerak dari tahap n, terus mundur ke tahap n-1, n-2, dan seterusnya sampai tahap 1.

III. PARTISI BILANGAN BULAT

A. *Deskripsi Permasalahan*

Definisi dari partisi bilangan yaitu, jika terdapat suatu bilangan bulat positif n ($\mathbb{Z}^+$), partisi bilangan n menyatakan cara menguraikan n sebagai penjumlahan dari beberapa bilangan bulat positif yang tidak lebih besar daripada n dan deretnya tidak menaik. Sebagai contoh, bilangan bulat 5 dapat diuraikan menjadi 7 cara, yaitu:

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

Mulai dari sini, kita definisikan $p(n)$ sebagai banyaknya partisi bilangan dari n (contoh: $p(5)=7$).

B. *Sejarah*

Persoalan partisi bilangan bulat pertama kali dikemukakan oleh Leibniz kepada J. Bernouli memalui suratnya pada tahun 1674. Ia bertanya mengenai banyaknya partisi bilangan pada suatu bilangan bulat positif, dia mengamati bahwa nilai $(3) = 3$ $(3, 2+1, 1+1+1)$, sebagaimana $p(4)=5$, $p(5)=7$, dan seterusnya.

Setelah kemunculan persoalan tersebut, muncullah orang-orang yang ikut andil dalam perkembangan partisi bilangan bulatnya, mulai dari Leonhard Euler, James Joseph Sylvester, Percy Alexander MacMahon, Godfrey Harold Hardy, sampai pada matematikawan India, Srinivasa Ramanujan.

Pada pertengagan 2016, perbincangan terkait pertisi bilangan mulai hangat kembali setelah diluncurkannya film "The Man Who Knew Infinity" yang mengangkat kisah hidup seorang Srinivasa Ramanujan bersama penelitian partisi bilangan bulatnya. Mulai saat itu, penulis juga mulai tertarik dengan persoalan tersebut.

C. *Algoritma Rekursif Standar dalam Penyelesaian Persoalan Partisi Bilangan Bulat*

Terdapat algoritma rekursif standar untuk menyelesaikan mendapatkan nilai $p(n)$, jika $part(n,m)$ didefinisikan sebagai banyaknya partisi bilangan bulat n dengan hanya disusun oleh bilangan yang lebih kecil atau sama dengan m, maka:

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

Karena $part(n,n)$ merupakan banyaknya partisi bilangan bulat yang disusun oleh bilangan yang lebih kecil atau sama dengan n, maka $p(n) = part(n,n)$. Perlu diketahui bahwa algoritma ini memiliki komleksitas yang eksponensial sehingga tidak mangkus.

Penulis membuat program untuk algortma ini dalam bahasa Python dan Clojure yang dapat dilihat di https://github.com/gilang20/if2211-strategi-algoritma-makalah.

IV. IMPLEMENTASI *MEMOIZATION* DAN PEMROGRAMAN DINAMIS DALAM PARTISI BILANGAN BULAT

A. *Implementasi dengan Memoization*

Seperti yang telah dijelaskan pada Dasar Teori bagian A, *memoization* menyimpan nilai dari sub-solusi yang telah dihitung dan kemudian menggunakannya saat dibutuhkan sehingga tidak perlu menghitung ulang dari awal. Pada peroalan partisi bilangan bulat ini, penyimpanannya agak berbeda dengan contoh bilangan fibonacci, pada persoalan ini sub-solusi yang disimpan berbentuk tuple (n, m). Jika nilai dari sub-solusi sudah pernah dihitung, *return* nilai memo-nya, jika tidak lakukan rekursif seperti biasa sambil menyimpan nilainya. Berikut notasi implementasi *memoization*-nya.

```
memo = {} // Inisiasi kosong
part(n,m):
  // Nilai part(n,m) sudah ada atau belum?
  if (n,m) in memo: return memo[(n,m)]
  else if (n < 0 or m < 1): return 0
       else if (n < 2 or m = 1): return 1
       else: f = part(n-m,m) + part(n,m-1)
       memo[(n,m)] = f // Simpan nilainya
       return f
p(n): return part(n,n)
```

Dengan pemanfaatan memoization ini, pastinya banyak komputasi dan memori yang digunakan jauh lebih sedikit. Penulis membuat program untuk algoritma ini dalam bahasa Python dan Clojure yang bisa dilihat di https://github.com/gilang20/if2211-strategi-algoritma-makalah.

B. *Implementasi dengan Pemrograman Dinamis*

Pada implementasi ini, program memanfaatkan tabel dua dimensi (matriks). Algoritma ini diturunkan dari fungsi rekursif yang telah dibahas, namun ditransformasi menjadi persoalan program dinamis. Perhatikan bahwa $part(n, m) = 1$ untuk n=[0,1] atau m=[1]. Adapun transformasinya menjadi seperti ini:

$$part(n,m) = \begin{cases} 1, & n < 2 \; or \; m < 2 \\ \sum_{k=1}^{m} part(n-k,k), & otherwise \end{cases}$$

Pada implementasinya, program ini menggunakan matrix dengan ukuran (n+1) kali n. Program dinamis ini tergolong maju karena mulai dari step 1 menuju n. Agar lebih jelas, berikut ilustrasi matriks-nya:

n\m	1	2	3	4	5	6
0	1	1	1	1	**1**	1
1	1	1	1	**1**	1	1
2	1	2	**2**	2	2	2
3	1	**2**	3	3	3	3
4	**1**	3	4	5	5	5
5	1	3	5	6	**7**	7
6	1	4	7	9	10	11

Tabel 1. Ilustrasi matrix tabel untuk pemrograman dinamis.
Sumber : dibuat dengan excel, dokumen pribadi

Berikut implementasi program dinamisnya dalam bahasa Python:

```
# Penyelesaian persoalan Partisi Bilangan bulat dengan pemrograman dinamis
def integerPartition(n):
  tabel = [[0 for x in range(n)] for y in range(n+1)]
    for i in range(n+1):
      for j in range(n):
        if (i < 2 or j < 1):
          tabel[i][j] = 1
        else:
          tamp = 0
          brs = i-1
          kol = 0
          while (brs >= 0 and kol <= j):
            tamp += tabel[brs][kol]
            brs -= 1
            kol += 1
          tabel[i][j] = tamp
  return tabel[n][n-1]
```

C. *Mengoptimalkan Algoritma IV.B*

Jika kita lihat algoritma program dinamis tersebut, kita bisa melihat perhitungan yang *redundant* untuk mendapatkan nilai tabel[i][j], kenapa? Misalkan program baru selesai menghitung $tabel[6][4]$, lalu selanjutnya program tentu akan menghitung nilai $tabel[6][5]$. Pada perhitungan $tabel[6][5]$, program memanggil sebagian besar tabel yang sama dengan $tabel[6][4]$, Padahal perbedaan kedua nilai tabel tersebut hanya selisih $tabel[1][5]$ ($tabel[6][5] \geq tabel[6][4]$.

Dari fakta tersebut, kita dapat menyatakan $tabel[i][j] = tabel[i][j-1] + c$, dengan c bernilai 0 jika indeks matrix tak terdefinisi, dan bernilai >=1 jika terdefinisi (karena nilai minimum pada matrix adalah 1).

Sehingga terdapat sedikit perubahan pada kode program yang ditebalkan.

Dari

```
# Kode sebelum
          tamp = 0
          brs = i-1
          kol = 0
          while (brs >= 0 and kol <= j):
            tamp += tabel[brs][kol]
            brs -= 1
            kol += 1
# Kode sesudah
```

Menjadi

```
# Kode sebelum
        tamp = tabel[i][j-1]
        if (i-j >= 1): tamp += tabel[i-j-
1][j]
# Kode sesudah
```

V. HASIL EKSPERIMAN DAN ANALISIS

Pada eksperiman ini, penulis mengimplementasikan algoritma-algoritma yang telah dijelaskan sebelumnya dalam bahasa Python 3.5.2 dan Clojure 1.8.0. Adapun lingkungan yang penulis gunakan dalam eksperimen ini ialah *compiler online* https://repl.it/, penulis menggunakannya karena dari perbandingan, rata-rata waktu eksekusi di website tersebut lebih baik daripada eksekusi di laptop penulis. Adapun hasilnya sebagai berikut:

A. *Rekursif Standar*

	n	Clojure 1.8.0	Python 3.5.2
waktu eksekusi (ms)	5	0,3	0,143
	10	0,35	0,185
	20	1,15	0,628
	50	93,684	121,338
	70	1643,369	3096,113

Tabel 2. Waktu eksekusi penyelesaian partisi bilangan bulat dengan algoritma rekursif standar. Sumber : dokumen pribadi

Seperti yang diprediksikan, data di atas menunjukkan bahwa algoritma rekursif standar tidak mangkus dalam menyelesaikan persoalan partisi bilangan bulat dikarena memiliki kompleksitas yang eksponensial.

Hal tersebut dikarenakan banyak perhitungan *redundant* yang dilakukan oleh program sehingga simpul pada pohon rekursif yang terbentuk sangat banyak. Berikut datanya:

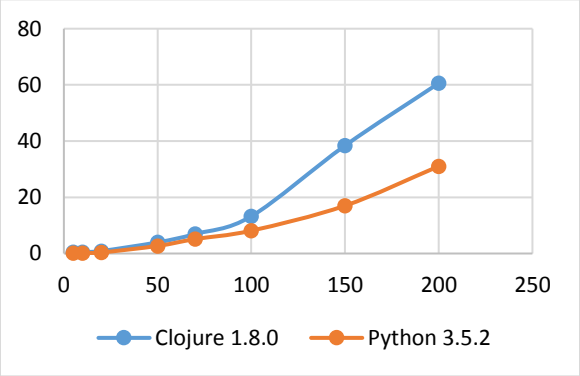
n	7	8	9	10	11	12	13
Banyak simpul	37	57	79	113	153	209	273

Tabel 3. Banyak simpul yang dibangkitkan untuk $p(n)$. Sumber : dokumen pribadi

B. *Dengan Memoization*

	n	Clojure 1.8.0	Python 3.5.2
waktu eksekusi (ms)	5	0,43	0,124
	10	0,478	0,175
	20	0,878	0,367
	50	3,968	2,688
	70	6,946	5,12
	100	13,236	8,124
	200	60,586	31,048
	400	Overflow	159,265

Tabel 4. Waktu eksekusi penyelesaian partisi bilangan bulat dengan algoritma rekursif standar. Sumber : dokumen pribadi



Grafik 1. Waktu eksekusi (ms) untuk menyelesaikan partisi bilangan n. dokumen pribadi

n	7	8	9	10	11	12	13
Banyak simpul	31	41	53	66	81	97	115

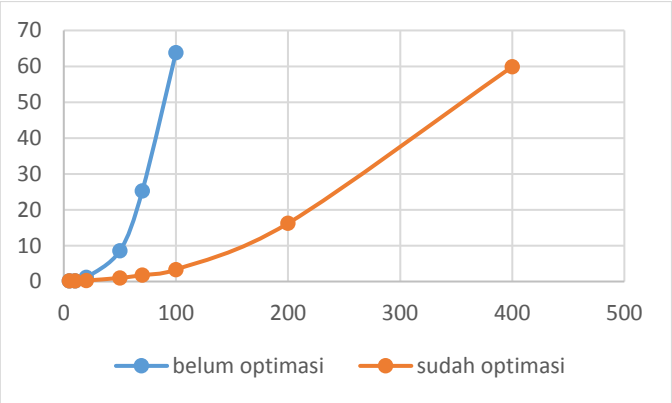
Tabel 5. Banyak simpul yang dibangkitkan untuk $p(n)$. Sumber :dokumem. pribadi

Dari data di atas, terbukti bahwa metode *memoization* membuat algoritma jauh lebih mangkus, bisa dibuktikan dengan waktu eksekusi yang jauh lebih cepat dibanding rekursif standar, selain itu juga banyak simpul yang dibangun pada algoritma dengan *memoization* jauh lebih sedikit dibanding simpul yang dibangun oleh algoritma rekursif standar.

C. Dengan Pemrograman Dinamis

		Python 3.5.2	
	n	belum optimasi	sudah optimasi
waktu eksekusi (ms)	5	0,132	0,133
	10	0,211	0,153
	20	1,122	0,284
	50	8,564	0,999
	70	25,23	1,781
	100	63,839	3,328
	200	>80	16,236
	400	>80	59,899

Tabel 6. Waktu eksekusi untuk menyelesaikan partisi bilangan n.
Sumber : dokumen pribadi



Grafik 2. Waktu eksekusi (ms) untuk menyelesaikan p(n).
Sumber : dokumen pribadi

D. Analisis Lebih Lanjut

Hasil terbaik didapatkan oleh algoritma pemrograman dinamis yang dioptimasi. Algoritma tersebut bisa lebih baik daripada *memoization* dikarenakan pada implementasinya, program ini hanya menggunakan struktur data tabel dua dimensi dan perulangan sederhana yang *straightforward*, sedangkan pada *memoization* program butuh menyimpan nilai dengan *hash-map* serta melakukan rekursif yang butuh lebih banyak sumber daya (*overhead* lebih banyak).

VI. KESIMPULAN

Hasil eksperiman di makalah ini menunjukkan bahwa *memoization* dan pemrograman dinamis berhasil mengoptimalkan penyelesaian persoalan partisi bilangan bulat n. Hasil pengoptimalan terbaik/termangkus didapatkan dengan menerapkan pemrograman dinamis yang dioptimalkan.

APENDIKS

```
p(2000) =  472081917561941388860143240679995951220034416 6
p(2001) =  485573291737900023757436560968748891269727314 3
p(2002) =  499446774218336614807483903544741638039378164 4
p(2003) =  513713090331689362277074546423508413938492842 6
p(2004) =  528383263759951707557208174656426042085890170 5
p(2005) =  543468624763963405906125899390404243060799007 4
p(2006) =  558980818633438305029157099275647140563304138 7
p(2007) =  574931814367814423077867666378967298416919511 6
p(2008) =  591333913594175240596537869159957244132462394 1
p(2009) =  608199759728658785940567803080921867028224678 5
p(2010) =  625542347387943217255115334717978795312568282 6
p(2011) =  643375032057574303741131672821567920464274966 0
p(2012) =  661711540024081605227555666131489028899933200 9
p(2013) =  680565978578016365739192060228659666340621791 1
p(2014) =  699952846495235300756706714541516427650506967 0
p(2015) =  719887044803950699479150359060112680160753413 7
p(2016) =  740383887845268716291284211917626231854231440 9
p(2017) =  761459114635144526966169456491278624644547889 1
Waktu eksekusi : 1662.9176139831543 ms
```

Gambar 3. Nilai partisi bilangan bulat dari 2000 sampai 2017.
Sumber : dokumen pribadi

Seluruh *source code* program dapat dilihat di:
https://github.com/gilang20/if2211-strategi-algoritma-makalah

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 Strategi Algoritma kepada penulis, mata kuliah ini merupakan mata kuliah yang paling penulis sukai di semester 4. Terakhir, penulis ingin berterima kasih kepada pembaca makalah ini, semoga makalah ini dapat bermanfaat bagi pembaca, aamiin.

REFERENSI

[1] Munir, Rinaldi, 2009, *Diktat Kuliah IF2211, Strategi Algoritma*, Bandung: Penerbit Informatika.

[2] Kenneth H. Rosen, 2011, "Elementary Number Theory 6th Edition" , PEARSON.

[3] https://bermatematika.net/2016/09/19/partisi-bilangan-asli/, diakses pada tabnggal 18-19 Mei 2017.

[4] http://courses.csail.mit.edu/6.006/fall09/lecture_notes/lecture18.pdf, diakses pada tanggal 18 Mei 2017.

[5] http://www.python-course.eu/python3_memoization.php, diakses pada tanggal 18 Mei 2017.

[6] http://programming.guide/dynamic-programming-vs-memoization-vs-tabulation.html, diakses pada tanggal 19 Mei 2017.

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, 19 Mei 2017

Gilang Ardyamandala Al Assyifa (13515096)