Optimasi perkalian matriks berantai dengan *dynamic programming*

Muh.Muslim Al Mujahid 13518054
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13518054@std.stei.itb.ac.id

***Abstract*—Perkalian matriks berantai adalah perkalian dari 3 atau lebih matriks. Operasi ini adalah salah satu operasi dasar dalam berbagai perhitungan. Namun, semakin banyak matriks yang dioperasikan biaya yang harus dikeluarkan akan semakin besar. Pada paper ini saya akan mencoba menjelaskan cara mengalikan matriks berantai dengan biaya minimum.**

Keywords—component; formatting; style; styling; insert (key words)

I. PENGENALAN

Dalam dunia matematika, matriks adalah larik dua dimensi dengan elemen berupa angka, simbol, atau ekspresi yang disusun berdasarkan baris dan kolom. Layaknya variabel pada umumnya matriks juga dapat dilakukan berbagai operasi seperti penjumlahan, pengurangan, perkalian, dan pembagian. Namun berbeda dengan angka, matriks angkar terdiri dari deretan angka, sehingga operasi yang dilakukan berbeda. Penjumlahan misalnya, dilakukan dengan menjumlahkan seluruh angka dengan posisi yang sama dari kedua matriks, begitu pula dengan pengurangan.

Perkalian matriks berantai (*matrix chain multiplication*) adalah perkalian dari deretan beberapa matriks. Matriks dikalikan satu persatu untuk menemukan hasil akhir. Akan tetapi pada banyak kasus perkalian matriks memiliki biaya yang sangat besar terhadap performa komputer karena banyaknya operasi yang dilakukan.

Perkalian matriks merupakan salah satu operasi dasar yang banyak digunakan di berbagai bidang, bahkan untuk saya sendiri sebagai seorang mahasiswa. Sehingga biaya yang dikeluarkan bisa dibilang sangat merugikan. Oleh karena itu saya tertarik untuk mengangkat topik ini tentang bagaimana cara melakukan perkalian matriks berantai dengan lebih efisien sehingga biaya yang dikeluarkan minimal.

II. TEORI

A. *Perkalian Matriks dan biayanya*

Perkalian matriks harus memenuhi beberapa syarat untuk dilakukan. Untuk perkalian dua buah matriks A dan B, jumlah kolom pada matriks A harus sama dengan jumlah baris pada matriks B. Perkalian dilakukan dengan mengalikan masing-masing elemen baris ke-i pada matriks A dengan elemen kolom ke-j pada matriks B, kemudian dijumlahkan. Hasilnya akan

$$\begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix} \begin{bmatrix} b_1 & b_2 & b_3 \\ b_4 & b_5 & b_6 \\ b_7 & b_8 & b_9 \end{bmatrix} = \begin{bmatrix} c_1 & c_2 & c_3 \\ c_4 & c_5 & c_6 \\ c_7 & c_8 & c_9 \end{bmatrix}$$

Gambar 2.1 : Ilustrasi perkalian matriks

menjadi elemen matriks C pada baris ke-i dan kolom ke-j. Matriks yang dihasilkan akan memiliki jumlah kolom sama dengan matriks A dan jumlah baris sama dengan matriks B.

Perkalian matriks berantai juga harus memenuhi syarat, yaitu tiap operasi perkalian yang dilakukan harus memenuhi syarat perkalian dua matriks. Misal kita memiliki operasi perkalian matriks

$$A1\ x\ A2\ x\ A3\ x \ldots x\ An$$

Maka perkalian dari hasil perkalian $A1\ x\ A2$ dan $A3$ harus memenuhi syarat perkalian dua buah matriks dan begitu seterusnya.

Perkalian matriks tidak memenuhi syarat komutatif, namun memenuhi syarat asosiatif.

$$A1\ x\ A2 \neq A2\ x\ A1$$

$$A1\ x\ (A2\ x\ A3) = (A1\ x\ A2)\ x\ A3$$

Biaya perkalian matriks didapatkan dengan menghitung banyaknya operasi perkalian bilangan yang dilakukan. Mengalikan sebuah matriks dengan ukuran $i \; x \; j$ dan matriks berukuran $j \; x \; k$ akan menghasilkan $(i \; x \; j \; x \; k)$ kali operasi perkalian bilangan. Perhitungan setiap satu elemen memerlukan j kali operasi perkalian, ada $i \; x \; k$ jumlah elemen, sehingga diperlukan $i \; x \; j \; x \; k$ kali operasi perkalian. Begitu pula dengan perkalian matriks beruntun, jika dikalikan matriks dengan ukuran $i \; x \; j, j \; x \; k, k \; x \; l, l \; x \; m, dan \; m \; x \; n$ akan menghasilkan operasi perkalian sejumlah $(i \; x \; j \; x \; k \; x \; l \; x \; m \; x \; n)$ kali.

B. Permasalahan

Diberikan deretan matriks A_1, A_2, .. A_n, cari urutan perkalian matriks sehingga menghasilkan banyaknya operasi perkalian yang paling minimum. Pada perkalian matriks berantai tidak selalu mendapatkan operasi perkalian yang minimum. Misal kita memiliki matriks A_1, A_2, dan A_3 dengan ukuran 3x3, 3x7, dan 7x4. Jika matriks-matriks tersebut dikalikan secara berurutan maka total operasi perkalian yang didapatkan adalah $(3 \; x \; 3 \; x \; 7) \; + (3 \; x \; 7 \; x \; 4) \; = 147$. Jika kita ubah urutan perkaliannya menjadi $A1 \; x \; (A2 \; x \; A3)$ maka total operasi perkalian yang dilakukan menjadi $(3 \; x \; 7 \; x \; 4) \; + \; (3 \; x \; 3 \; x \; 4) \; = \; 120$. Dengan mengubah urutan perkalian kita dapat melakukan operasi perkalian matriks dengan biaya minimum.

Perkalian matriks memenuhi syarat asosiatif, sehingga perkalian dapat dilakukan dari pasangan matriks berdampingan mana saja. Misal diberikan matriks A_1, A_2, A_3, dan A_4. Asusmsikan dimensi masing-masing matriks adalah sebagai berikut.

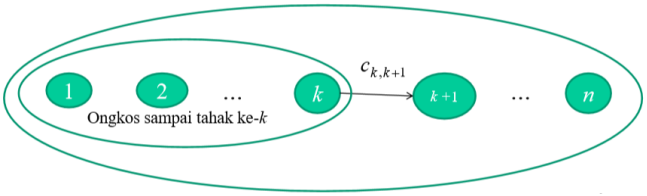
$$A1 = d0 \; x \; d1$$
$$A2 = d1 \; x \; d2$$
$$A3 = d2 \; x \; d3$$
$$A4 = d3 \; x \; d4$$

Berikut adalah lima kemungkinan urutan perkalian yang dapat dilakukan.

$$(A1A2)(A3A4) : d0d1d2 + d2d3d4 + d0d2d4$$
$$\big((A1A2)A3\big)A4 : d0d1d2 + d0d2d3 + d0d3d4$$
$$\big(A1(A2A3)\big)A4 : d1d2d3 + d0d1d3 + d0d3d4$$
$$A1((A2A3)A4) : d1d2d3 + d1d3d4 + d0d1d4$$
$$A1(A2(A3A4)) : d2d3d4 + d1d2d4 + d0d1d4$$

C. Dynamic Programming

Dynamic Programing (Pemrograman dinamis) salah satu metode pemecahan masalah yang kompleks dengan memecah permasalah menjadi beberapa permasalahan yang kecil, memecahkan masalah-masalah tersebut, dan menyimpannya ke dalam suatu bentuk struktur data. Solusi permasalahan pada suatu tahap diperoleh dari solusi persoalan tahap sebelumnya



Gambar 2.2 : Ilustrasi perhitungan biaya DP

Karakteristik penyelesaian DP adalah
1. Terdapat sejumlah berhingga pilihan yang mungkin
2. Solusi pada setiap tahap dibangun dari hasil solusi pada tahap sebelumnya
3. Menggunakan persyaratan optimasi untuk membatasi sejumlah pilihan yang harus dipertimbangkan pada suatu tahap

Karakteristik persoalan DP adalah
1. Persoalan dapat dibagi menjadi beberapa tahap, yang pada setiap tahap hanya diambil satu keputusan
2. Masing-masing tahap terdiri dari sejumlah status yang berhubungan dengan tahap tersebut.

pada DP, rangkaian keputusan yang optimal dibuat dengan menggunakan prinsip optimalitas, yaitu jika solusi total optimal, maka bagian solusi sampai tahap ke-k juga optimal. Artinya untuk mencari solusi pada tahap k+1, kita dapat menggunakan hasil solusi dari tahap k, tidak perlu mengulang dari awal. Biaya (Cost) pada tahap k+1 diperoleh dengan rumus.

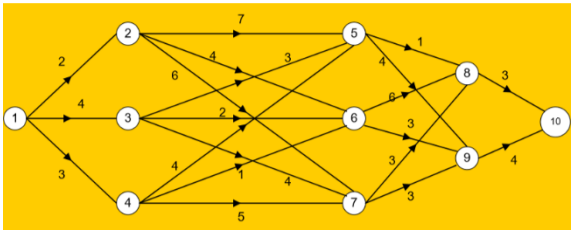
$$Cost(k+1) \; = \; Cost(k) \; + \; Cost(k, k+1)$$

Pada pemrograman dinamis terdapat dua pendekatan.
1. Program dinamis maju (forward/up-down). Program bergerak mulai dari tahap 1, sampai tahap ke n.
2. Program dinamis mundur (backward/bottop-up). Program bergerak mulai dari tahap n sampai tahap ke 1.

Langkah-langkah pengembangan algoritma PD.
1. Karakteristikkan struktur solusi optimal
2. Definiskan secara rekursif nilai solusi optimal
3. Hitung nilai solusi optimal secara maju atau mundur
4. Konstruksi solusi optimal

Perhatikan contoh permasalahan *Shortest Path* berikut.

Akan dicari rute terpendek dari simpul 1 ke simpul 10 pada graf berbobot berikut.



Gambar 2.3 : Contoh persoalan *Shortest path*

Persoalan ini dapat diselesaikan dengan DP karena persoalan dapat dipecah menjadi beberapa persoalan kecil atau tahap. Tiap tahap k adalah proses pemilihan simpul selanjutnya, yaitu untuk masing-masing tahap 1, 2, 3, dan 4, simpul yang dapat dipilih adalah {2, 3, 4}, {5, 6, 7}, {8, 9}, dan {10}.

Pada persoalan ini dapat didefinisikan

- Tahap(k) adalah proses memilih simpul berikutnya
- Status(x_k) adalah jalur yang telah dilalui sampai tahap ke k

Selanjutnya dapat didefinisikan basis dan relasi rekurans sebagai berikut.

Relasi rekurens berikut menyatakan lintasan terpendek dari status s ke x_4 pada tahap k:

$f_1(s) = C_{x1s}$ (Basis)
$f_k(s) = \min \{C_{xks} + f_{k-1}(x_k)\}$ (Rekurens)

Keterangan:
x_k : peubah keputusan pada tahap k, (k = 2, 3, 4, …)
C_{xks} : bobot (cost) sisi dari s ke x_k
$f_k(s)$: nilai minimum dari $f_k(x_k, s)$
$f_k(x_k, s)$: total bobot lintasan dari x_k ke s

tujuan program dinamis maju: mendapatkan $f_4(10)$ dengan mencari $f_1(s)$, $f_2(s)$, dan $f_3(s)$ terlebih dahulu.

III. IMPLEMENTASI

A. *Penerapan Dynamic programming pada optimasi perkalian matriks berantai*

Tahap(k) adalah proses mencari matriks yang akan dikalikan pada posisi ke-k.

Status(x(k)) menyatakan jumlah perkalian yang telah dilakukan sampai tahap k

Diberikan matriks A_1, A_2, A_3, A_4. Akan ada 4 tahap dengan peubah sebagai berikut.

x_1 = M[1, 1] = Banyaknya perkalian yang telah dilakukan sampai tahap ke-1

x_2 = M[1, 2] = Banyaknya perkalian yang telah dilakukan sampai tahap ke-2

x_3 = M[1, 3] = Banyaknya perkalian yang telah dilakukan sampai tahap ke-3

x_4 = M[1, 4] = Banyaknya perkalian yang telah dilakukan sampai tahap ke-4

Relasi rekurens

M[i, j] adalah banyaknya operasi perkalian yang diperlukan untuk perkalian matriks berantai A_1, A_2, … A_n, untuk $1 \leq i \leq j \leq n$

M[i, i] = 0, karena tidak ada operasi perkalian yang perlu dilakukan (basis)

Solusi optimal dari A_i x … x Aj dipecah pada suatu titik k, dengan $i \leq k < j$. sehingga M[i, j] = M[i, k] + M[k+1, j] + $d_{i-1}d_kd_j$

$$M[i,j] = \begin{cases} 0, & i = j \\ \min\{M[i,k] + M[k+1,j] \\ +d[i-1]d[k]d[j]\}, \text{Untuk i} \leq \text{k} < \text{j}, & i < j \end{cases}$$

(rekursif)

B. *Proses penyelesaian*

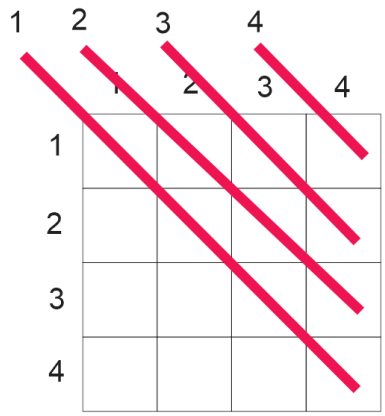
Misal diberikan matriks A1, A2, A3 dab A4 dengan ukuran sebagai berikut.

$$A1 = 3\ x\ 2$$
$$A2 = 2\ x\ 4$$
$$A3 = 4\ x\ 2$$
$$A4 = 2\ x\ 5$$

Akan dicari urutan perkalian matriks dengan biaya minimal (operasi perkalian paling sedikit).

Untuk menyesaikan persoalan ini akan digunakan 2 matriks C dan K yang berukuran 4 x 4. Matriks C digunakan untuk menyimpan biaya tiap perkalian matriks, dan matriks K digunakan untuk menyimpan jalur yang telah dilewati.

Pengisian matriks dilakukan secara diagonal sebagai berikut.



Gambar 3.1 : Ilustrasi urutan pengisian matriks C dan K

Tiap diagonal menandakan tahap 1, 2, 3, dan 4.

Dimensi matriks disimpan kedalam sebuah *array* S sebagai berikut.

3	2	4	2	5

Gambar 3.1 : Hasil perubahan dimensi matriks menjadi *array*

Kemudian dilanjutkan dengan eksekusi tahap 1 sampai 4.

Tahap 1

Menghitung C[1, 1], C[2, 2], C[3, 3], dan C[4, 4]

C[1, 1] = 0

C[2, 2] = 0

C[3, 3] = 0

C[4, 4] = 0

C

	1	2	3	4
1	0			
2		0		
3			0	
4				0

K

	1	2	3	4
1				
2				
3				
4				

Gambar 3.2 : Hasil matriks tahap 1

Tahap 2

Menghitung C[1, 2], C[2, 3], dan C[3 ,4]

$$C[1,2] = \min_{1 \le k < 2} \{C[1,1] + C[2,2] + 3x2x4 = 24\} = 24$$

$$C[2,3] = \min_{2 \le k < 3} \{C[2,2] + C[3,3] + 2x4x2 = 16\} = 16$$

$$C[3,4] = \min_{3 \le k < 4} \{C[3,3] + C[4,4] + 4x2x5 = 40\} = 40$$

C

	1	2	3	4
1	0	24		
2		0	16	
3			0	40
4				0

K

	1	2	3	4
1		1		
2			2	
3				3
4				

Gambar 3.2 : Hasil matriks tahap 1

Perhatikan bahwa K[1,2] = 1, karena pada perhitungan C[1, 2] yang memberikan nilai terendah adalah saat k = 1. Begitu pula dengan K[2, 3] dan K[3, 4].

Tahap 3

Menghitung C[1, 3] dan C[2, 4]

$$C[1,3] = \min_{1 \le k < 3} \begin{cases} C[1,1] + C[2,3] + 3x2x2 = 28 \\ C[1,2] + C[3,3] + 3x4x2 = 48 \end{cases} = 28$$

$$C[2,4] = \min_{2 \le k < 4} \begin{cases} C[2,2] + C[3,4] + 2x4x5 = 80 \\ C[2,3] + C[4,4] + 2x2x5 = 36 \end{cases} = 36$$

C

	1	2	3	4
1	0	24	28	
2		0	16	36
3			0	40
4				0

K

	1	2	3	4
1		1	1	
2			2	3
3				3
4				

Gambar 3.3 : Hasil matriks tahap 3

K[1, 3] dan K[2, 4] diisikan 1, 3 karena pada perhitungan C[1, 3] dan C[2, 4] yang memberikan nilai minimum adalah saat k = 1 dan k = 3

Tahap 4

Menghitung C[1,4]

$$C[1,4] = \min_{1 \le k < 4} \begin{cases} C[1,1] + C[2,4] + 3x2x5 = 86 \\ C[1,2] + C[3,4] + 3x4x5 = 124 \\ C[1,3] + C[4,4] + 3x2x5 = 58 \end{cases} = 58$$

C

	1	2	3	4
1	0	24	28	58
2		0	16	36
3			0	40
4				0

K

	1	2	3	4
1		1	1	3
2			2	3
3				3
4				

Gambar 3.4 : Hasil matriks tahap 1

K[1, 4] = 3, karena yang memberikan nilai minimum adalah saat k = 3.

Maka didapatkan nilai minimum operasi perkalian yang harus dilakukan adalah C[1,4] = 58.

Urutan perkalian matriks didapatkan dari tabel K pada baris pertama dengan penjelasan sebagai berikut.

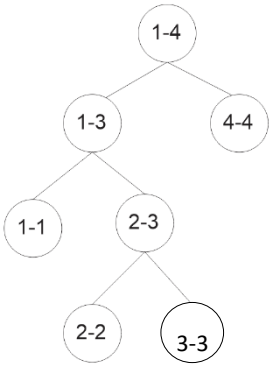
K[1,4] = 4, menandakan pada perkalian matriks A_1 sampai A_4 , A_4 akan dipisahkan oleh tanda kurung menjadi $(A_1A_2A_4)A_4$, dan seterusnya sebagai berikut.

$$K[1,4] = 4, (A1A2A3)(A4)$$

$$K[1,3] = 1, \big(A1(A2A3)\big)(A4)$$

$$K[1,2] = 1, \big((A1)(A2A3)\big)(A4)$$

Hasil ini dapat diperoleh dengan menggunakan graf sebaegai berikut.



Gambar 3.5 : Hasil generasi graf dari matriks K

Dengan tiap daun menandakan menandakan matriks secara individual.

IV. Analisis

A. *Efektifitas*

Metode ini bekerja dengan mencari urutan perkalian matriks yang menghasilkan banyak operasi perkalian bilangan yang paling minimum. Oleh karena itu akan semakin efektif jika digunakan pada perkalian matriks berantai dengan dimensi matriks yang beragam dan jumlah matriks yang banyak. Sebaliknya metode ini tidak akan berguna jika digunakan pada perkalian matriks berantai dengan dimensi yang sama karena dimanapun diletakkan tanda kurung akan tetapi menghasilkan total operasi perkalian bilangan yang sama.

B. *Kelebihan dan Kekurangan*

Algoritma akan diuji dengan 13 matriks dengan dimensi 23x7, 7x15, 15x32, 32x20, 20x54, 54x2, 2x33, 33x11, 11x7, 7x21, 21x13, 13x60, 60x10. Tiap elemen matriks bernilai 1 sehingga dipastikan tidak mempengaruhi perhitungan waktu eksekusi. Hasilnya adalah sebagai berikut.

```
E:\Kuliah\strategi-algoritma\paper>py mul_bf.py
0.04293012619018555
```

Gambar 4.1: waktu eksekusi dengan menggunakan bruteforce

```
E:\Kuliah\strategi-algoritma\paper>py mul_dp.py
0.0
```

Gambar 4.2: Waktu eksekusi dengan menggunakan DP

Ini membuktikan eksekusi dengan menggunakan DP jauh lebih cepat dibanding menggunakan metode *bruteforce*. Namun semakin besar jumlah matriks maka ruang yang diperlukan akan semakin besar karena untuk menjalankan metode ini diperlukan 2 matriks dengan ukuran n x n sehingga algoritma ini memiliki kompleksitas ruang $O(n^2)$.

C. *Hal yang masih dapat dikembangkan*

Seperti yang telah kita bahas algoritma ini memerlukan ruang yang cukup besar. Namun jika perhatikan kembali matriks yang digunakan sebenarnya hanya setengah bagian diagonalnya saja. Oleh karena itu menurut saya masih dapat dilakukan pengembangan terkait dengan kompleksitas ruang ini hingga didapatkan algoritma yang lebih efektif.

V. Kesimpulan

Dynamic programming dapat dimanfaatkan diberbagai bentuk permasalah asalkan memenuhi syarat-syarat yang telah disebutkan sebelumnya. Akan tetapi di berbagai kasus algoritma yang menggunakan DP memiliki kompleksitas ruang yang besar karena karakteristik DP yang menyimpan seluruh kemungkinan jawaban tiap tahap ke dalam sebuah struktur data misal tabel.

Salah satu permasalahan yang dipecahkan dengan menggunakan DP yang dibahas pada *paper* ini adalah Perkalian matriks beruntun (*Matrix chain multiplication*). Dengan menggunakan DP perhitungan matriks dibagi menjadi beberapa tahap dan untuk tiap tahap ditentukan susunan matriks yang menghasilkan biaya terkecil. Biaya perkalian matriks dihitung dari total operasi perkalian matriks yang dilakukan. Karena perkalian matriks bersifat asosiatif jadi pasangan matriks bersebelahan manapun yang dikalikan lebih dulu pada akhirnya akan memberikan hasil yang sama.

Algoritma ini dinilai sangat efektif karena berdasarkan hasil uji eksekusi dengan menggunakan DP jauh lebih cepat dibanding menggunakan *bruteforce*. Namun karena kompleksitas ruang yang tergolong besar dinilai masih kurang efisien karena faktanya yang digunakan hanya setengah bagian diagonal matriks. Oleh karena itu menurut saya masih diperlukan penelitian lebih lanjut terkait dengan topik tersebut.

LINK VIDEO

https://youtu.be/IXRlQC3M-dI

UCAPAN TERIMA KASIH

Puji dan syukur saya ucapkan kehadirat Allah SWT. Karena rahmat dan karunia-Nya lah saya masih sempat untuk menyelesaikan makalah ini dengan tepat waktu. Terima kasih saya ucapkan kepada seluruh dosen mata kuliah IF2211 Strategi Algoritma yang selama ini mengajar dan membimbing saya dalam penyelesaian makalah ini. Saya juga berterima kasih kepada kedua orang tua saya yang atas doa dan dukungan yang selalu diberikan beserta teman-teman yang telah membantu dalam penyelesaian makalah ini.

REFERENCES

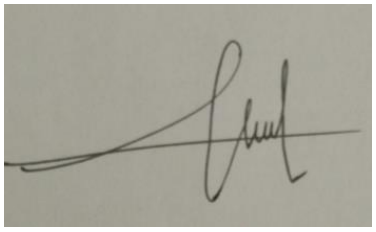
[1] Munir, Rinaldi. Diktat Kuliah IF2211 Strategi Algortima. Program Studi Teknik Informatika ITB. 2019.

[2] Dynamic Programming – Matrix-chain Multiplication. 2019. https://www.radford.edu/~nokie/classes/360/dp-matrix-parens.html (Diakses pada tanggal 2 Mei 2019 2019 pukul 06.00 WIB)

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, 3 Mei 2020



Muh. Muslim Al Mujahid 13518054