Perkalian Rantai Matriks dengan Menggunakan Pemrograman Dinamis

Haris Amrullah Lubis

Program Studi Teknik Informatika Institut Teknologi Bandung
Jalan Sangkuriang Dalam 36B Bandung
haris.lubis@gmail.com

ABSTRAK

Perkalian antara dua buah matriks telah dipecahkan dengan mudah dengan brute force dan *divide and conquer*. Namun, pada perkalian matriks berantai, terdapat persoalan yang cukup rumit dan jika dipecahkan dengan cara biasa akan memakan waktu secara eksponensial. Untuk memecahkan masalah tersebut digunakan pendekatan secara pemrograman dinamis.

Kata kunci: rantai matriks, pemrograman dinamis, *fully parenthesized*.

1. PENDAHULUAN

Pemrograman dinamis adalah teknik pemrograman yang dapat mengurangi *runtime* dari beberapa algoritma (tetapi tidak semua masalah memiliki karakteristik pemrograman dinamis) dari kompleksitas eksponensial ke kompleksitas polinomial. Banyak masalah-masalah di dalam dunia nyata yang hanya dapat dipecahkan dalam waktu yang memungkinkan dengan menggunakan pemrograman dinamis.

Untuk dapat menggunakan pemrograman dinamis, masalah harus memiliki karakteristik sebagai berikut:

1. *Optimal sub-structure*
   Solusi optimal dari sebuah masalah di dalamnya mengandung solusi-solusi optimal dari subproblem-subproblemnya.
2. *Overlapping sub-problems*
   Pada pemecahan tanpa menggunakan pemrograman dinamis, kita sering tanpa sengaja memecahkan submasalah yang sama berulang kali.

Pemrograman dinamis memiliki dua tipe. Pertama, pembangunan solusi dari kecil ke besar (bawah-atas). Kedua, penyimpanan hasil solusi per sub masalah di dalam sebuah table (atas-bawah ditambah pengingatan).

2. METODE

2.1 Analisis Biaya Perkalian Dua Buah Matriks

Berikut adalah algoritma perkalian dua buah matriks dengan menggunakan algoritma *brute-force*

```
PerkalianMatriks(A,B):
if kolom[A] != kolom[B] then
        error "dimensi tidak cocok"
else
        for i = 1 to baris[A] do
          for j = 1 to baris[B] do
                    C[i,j] = 0
                    for k = 1 to kolom[A] do
                              C[i,j] = C[i,j] + A[i,k] * B[k,j]
return C
```

Jika diketahui |A|=pxq dan |B|=qxr, maka kompleksitas waktunya = O(pqr).

Algoritma di atas bukanlah solusi optimum. Solusi optimumnya adalah dengan menggunakan algoritma *divide and conquer* yang dimodifikasi yang menghasilkan kompleksitas waktu O(n^2,83).

2.2 Masalah Perkalian Rantai Matriks

Masukan: Matriks A1, A2, …, An, dengan ukuran setiap Ai adalah Pi-1 x Pi

Keluaran: Hasil perkalian *fully parenthesized* A1, A2, …, An yang meminimumkan jumlah perkalian skalar

Hasil perkalian disebut *fully parenthesized* jika memenuhi salah satu dari

1. Matriks tunggal
2. Perkalian dari dua buah *fully parenthesized* matriks yang ditutup oleh tanda kurung

Contoh:

Terdapat tiga buah matriks dengan ukuran tiap buah matriksnya adalah A1(10x100), A2(100x5), A3(5x50).

Kita dapat melakukan *fully parenthesized* dengan dua cara:

1. (A1 (A2 A3)) = 100 x 5 x 50 + 10 x 100 x 50 = 75000
2. ((A1 A2) A3) = 10 x 100 x 5 + 10 x 5 x 50 = 7500 (10 kali lebih baik)

Dari kedua cara di atas, tampak bahwa biaya dari perkalian 3 matriks tersebut dapat berbeda secara signifikan. Biaya sangat bergantung pada pemilihan *fully parenthesized* dari matriks. Namun, pencarian secara *exhaustive* dengan mencari seluruh kemungkinan pengurungan memakan waktu secara eksponensial.

2.3 Pemecahan Masalah

Pertama, mencari sub struktur optimal dari masalah ini. Misalkan $A_{i..j}$ (i < j) menunjukkan hasil perkalian AiAi+1..Aj. Ai..j dapat diperoleh dengan memisahkannya menjadi Ai..k dan Ak+1..j dan kemudian mengalikannya dengan sub produk. Terdapat j-1 kemungkinan pemisahan.

Jika Ai..j di dalam tanda kurung optimal, maka Ai..k di dalam tanda kurung juga optimal dan Ak+1..j di dalam tanda kurung juga optimal.

Karena jika kedua pemisahan tersebut tidak optimal, maka terdapat pemisahan lain yang lebih baik, sehingga kita harus memilih pemisahan tersebut.

Kedua, formula rekursif. Untuk mencari A1..n, maka misalkan m[i,j] = jumlah minimum dari perkalian scalar yang dibutuhkan untuk menghitung Ai..j, maka Ai..j dapat ditemukan dengan cara memecahnya menjadi Ai..k Ak+1..j, sehingga kita memiliki

```
m[i,j]  = 0, if i=j
        = min i<=k<j { m[i,k]+m[k+1,j]+pi-1pkpj }, if i<j
S[i,j]  = nilai k saat nilai optimal pemisahan terjadi
```

Ketiga, menghitung biaya optimal.

```
Matric-Chain-Order(p)
n = length[p] – 1
for i = 1 to n do
        m[i,i] = 0
for l = 2 to n do
        for i = 1 to n-l+1 do
          j = i+l-1
          m[i,j] = infinity
          for k = i to j-1 do
                    q = m[i,k] + m[k+1,j] + pi-1*pk*pj
                    if q < m[i,j] then
                              m[i,j] = q
                              s[i,j] = k
return m and s
```

Keempat, membangun solusi optimal

```
Print-MCM(s,i,j)
if i=j then
        print Ai
else
        print "(" + Print-MCM(s,1,s[i,j]) + "*" + Print-
MCM(s,s[i,j]+1,j) + ")"
```

3. KESIMPULAN

- **Masalah perkalian rantai matriks dapat dipecahkan dengan menggunakan pendekatan pemrograman dinamis dengan kompleksitas waktu polinomial**

REFERENSI

[1] Rosen, Kenneth H. Discrete Mathematics and Its Application

[2] Knuth, Donald E, The Art of Computer Programming.