Analisis Kompleksitas Algoritma Pembentuk Klosur Refleksif, Setangkup, dan Menghantar dari Relasi

Muhammad Neo Cicero Koda - 13522108[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13522108@std.stei.itb.ac.id

Abstract*—Klosur relasi merupakan algoritma yang sering dipakai dalam topik relasi dengan pengaplikasian seperti pengiriman pesan antarkota. Analisis kompleksitas algoritma klosur relasi perlu dilakukan agar mengetahui efisiensi algoritma tersebut. Makalah ini bertujuan untuk menganalisis kompleksitas algoritma dari berbagai klosur relasi, seperti klosur menghantar, refleksif, dan setangkup. Hasil analisis menunjukkan bahwa masing-masing klosur tersebut memiliki kompleksitas waktu O(n), O(n^2), dan O(n^5).

***Keywords*—Klosur Menghantar, Klosur Refleksif, Klosur Setangkup, Kompleksitas Algoritma**

I. PENDAHULUAN

Analisis kompleksitas algoritma merupakan konsep yang penting dalam bidang informatika. Analisis kompleksitas algoritma diperlukan untuk menilai efisiensi suatu algoritma dalam menyelesaikan suatu masalah. Dengan menganalisis kompleksitas ruang atau waktu suatu algoritma dan mengidentifikasi kelemahannya, langkah-langkah untuk mengoptimalkan algoritma tersebut dapat ditentukan. Urgensi untuk meningkatkan efisiensi suatu algoritma semakin ditekankan dengan semakin membesarnya ukuran data yang diproses dan tuntutan waktu pemrosesan yang semakin berkurang. Oleh karena itu, analisis kompleksitas algoritma diperlukan karena dapat memberikan gambaran tentang efisiensi suatu algoritma serta membantu dalam mencari solusi untuk meningkatkan efisiensi algoritma tersebut.

Dalam bidang ilmu komputer dan matematika, relasi merupakan suatu konsep yang menggambarkan hubungan antarobjek. Klosur relasi merupakan pembentukan suatu relasi baru yang mengandung semua elemen relasi sebelumnya dan elemen-elemen baru sesedikit mungkin yang membuat relasi memiliki suatu sifat tertentu. Beberapa contoh jenis klosur adalah klosur refleksif, klosur setangkup, dan klosur menghantar. Klosur relasi merupakan suatu konsep yang penting karena dapat diaplikasikan ke berbagai kasus, seperti pengiriman pesan antarkota, basis data, kecerdasan buatan, dan teori graf.

Dalam makalah ini, akan ditentukan kompleksitas algoritma dari berbagai algoritma pembentuk klosur relasi, seperti klosur refleksif, setangkup, dan menghantar. Algoritma tersebut akan ditranslasikan terlebih dahulu menjadi bentuk kode dalam bahasa pemrograman Python. Melalui makalah ini, diharapkan pembaca mampu mendapatkan pehamaman yang lebih dalam tentang kompleksitas algoritma pembentuk klosur relasi. Pemahaman tersebut dapat digunakan untuk membuat perbaikan terhadap algoritma yang sudah ada agar lebih efisien.

II. DASAR TEORI

A. Relasi

Relasi merupakan hubungan atau keterkaitan antara elemen-elemen dari dua himpunan atau himpunan yang sama. Misalkan terdapat dua himpunan berupa A dan B, dua elemen berupa a dan b, dan R adalah sebuah relasi antara himpunan A dan B. R merupakan himpunan bagian dari A x B. Misalkan juga a $\in$ A dan b $\in$ B. Keterhubungan a dan b dapat digambarkan oleh pasangan terurut (a, b). Notasi a R b memiliki arti (a, b) $\in$ R. Dalam kata lain, relasi R menghubungkan relasi a dengan b.

Relasi dapat direpresentasikan dengan berbagai macam cara, seperti dengan diagram panah, tabel, matriks, graf berarah, dan diagram kartesian.

Misalkan terdapat himpunan A = {a_1, a_2, …, a_m) dan himpunan B = {b_1, b_2, …, b_m} dan kedua himpunan tersebut dihubungkan dengan relasi R. Relasi R dapat direpresentasikan sebagai matriks M. Misalkan m_{ij} merupakan elemen dari matriks M (i dan j masing-masing menyatakan baris dan kolom elemen pada matriks). Elemen m_{ij} dapat bernilai 1 atau 0. Nilai 1 memiliki arti bahwa a_i dan b_j dihubungkan oleh R, sedangkan nilai 0 memiliki arti bahwa a_i dan b_j tidak dihubungkan oleh R.

$$M = \begin{array}{c} \\ a_1 \\ a_2 \\ \vdots \\ a_m \end{array} \begin{array}{c} \begin{array}{cccc} b_1 & b_2 & \cdots & b_n \end{array} \\ \begin{bmatrix} m_{11} & m_{12} & \cdots & m_{1n} \\ m_{21} & m_{22} & \cdots & m_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ m_{m1} & m_{m2} & \cdots & m_{mn} \end{bmatrix} \end{array}$$

Gambar 1. Representasi relasi dengan matriks (Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/04-Relasi-dan-Fungsi-Bagian1-(2023).pdf)

Suatu relasi dapat memiliki beberapa sifat tertentu. Berikut adalah sifat-sifat yang dapat dimiliki oleh sebuah relasi:

1. Refleksif

   Suatu relasi R pada himpunan A disebut refleksif jika semua elemen pada A terhubung dengan dirinya sendiri oleh relasi. Dalam kata lain, untuk setiap a $\in$ A, (a, a) $\in$ R.

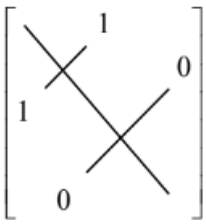
Relasi yang bersifat refleksif memiliki ciri khusus pada representasi matriks yang dimilikinya, yaitu setiap elemen pada diagonal utama bernilai 1.

Gambar 2. Representasi matriks suatu relasi yang bersifat refleksif (Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/04-Relasi-dan-Fungsi-Bagian1-(2023).pdf)

2. Setangkup

Suatu relasi dapat dikatakan setangkup jika untuk (a, b) ∈ A, jika (a, b) merupakan bagian dari relasi R, (b, a) juga merupakan bagian dari R. Jika terdapat (a, b) ∈ A sedemikian rupa sehingga (a, b) ∈ R, tetapi (b, a) ∉ R, relasi R tidak dapat dikatakan setangkup.

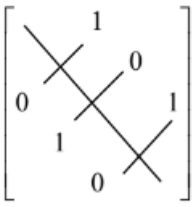
Ciri khas relasi setangkup pada representasi matriks yang dimilikinya adalah setiap elemen di atas diagonal utama merupakan pencerminan dari elemen di bawah diagonal utama. Dalam kata lain, untuk setiap elemen matriks M, $m_{ij} = m_{ji}$.



Gambar 3. Representasi matriks relasi setangkup (Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/04-Relasi-dan-Fungsi-Bagian1-(2023).pdf)

3. Tolak Setangkup

Suatu relasi pada himpunan A bersifat tolak setangkup jika untuk a, b ∈ A, (a, b) ∈ R dan (b, a) ∈ R hanya jika a = b. Ciri khas relasi tolak setangkup pada representasi matriks yang dimilikinya adalah untuk setiap elemen matriks yang bernilai satu, misal m_{ij}, selalu berlaku $m_{ji} = 0$ untuk $i \neq j$.



Gambar 4. Representasi matriks relasi tolak setangkup (Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/04-Relasi-dan-Fungsi-Bagian1-(2023).pdf)

4. Menghantar

Suatu relasi bersifat menghantar jika untuk (a, b, c) ∈ A, (a, b) ∈ R dan (b, c) ∈ R, maka (a, c) ∈ R. Relasi yang menghantar tidak memiliki ciri khas yang tampak pada matriks representasi yang dimilikinya.

Klosur relasi adalah suatu relasi baru yang dihasilkan dari suatu relasi R dan penambahan elemen relasi baru secara minimal sehingga relasi baru memiliki sifat tertentu, seperti refleksif, setangkup, ataupun menghantar. Berikut adalah langkah-langkah untuk menentukan beberapa klosur relasi:

1. Klosur refleksif

Misalkan terdapat himpunan A = {1, 2, 3, 4} dan relasi R = {(1, 2), (1, 3), (2, 2), (2, 3), (3, 3)}. Relasi R belum bersifat refleksif karena terdapat a ∈ A sedemikian rupa sehingga (a, a) ∉ R. Klosur refleksif dapat dicapai dengan menambahkan (1, 1) dan (4, 4) ke dalam R. Dalam kata lain, jika Δ = {(a, a) | a ∈ A}, klosur refleksif R adalah R ∪ Δ. Untuk sebuah matriks M yang menggambarkan relasi R, klosur refleksif dapat dicapai dengan megubah tiap elemen pada diagonal utama menjadi 1.

2. Klosur setangkup

Klosur setangkup sebuah relasi R adalah $R \cup R^{-1}$, dengan $R^{-1} = \{(b, a) \mid (a, b) \in R\}$. Untuk sebuah matriks M yang menggambarkan relasi R, klosur setangkup dicapai dengan mencari semua elemen matriks M yang memiliki nilai 1 ($m_{ij} = 1$) dan mengganti nilai m_{ji} menjadi 1.

3. Klosur menghantar

Klosur menghantar sebuah relasi R pada himpunan A adalah $R^* = R \cup R^2 \cup R^3 \cup \ldots \cup R^n$, dengan n merupakan jumlah elemen pada himpunan A. Untuk sebuah matriks M yang menggambarkan relasi R, klosur menghantar M didapatkan dari persamaan $M = M \vee M^2 \vee M^3 \vee \ldots \vee M^n$. Operasi ∨ melambangkan gabungan dari dua matriks relasi dan M^n melambangkan komposisi dari n buah matriks M.

B. Kompleksitas Algoritma

Kompleksitas algoritma adalah ukuran waktu atau ruang yang diperlukan untuk menjalankan suatu algoritma jika diberikan masukan berukuran n. Algoritma yang efisien adalah algoritma yang meminimumkan ukuran waktu atau ruang yang diperlukannya.

Mengukur kompleksitas waktu suatu algoritma tidak dapat dilakukan dengan mengukur waktu eksekusi komputer dalam menjalankan algoritma tersebut. Hal tersebut disebabkan oleh perbedaan arsitektur yang dimiliki oleh tiap komputer. Selain itu, bahasa pemrograman yang berbeda juga akan menghasilkan waktu eksekusi yang berbeda karena kode bahasa mesin yang dihasilkan *compiler* masing-masing Bahasa berbeda. Oleh karena itu, diperlukan suatu model abstrak untuk menggambarkan besaran waktu algoritma tersebut yang bebas dari keterbatasan perangkat keras dan bahasa pemrograman.

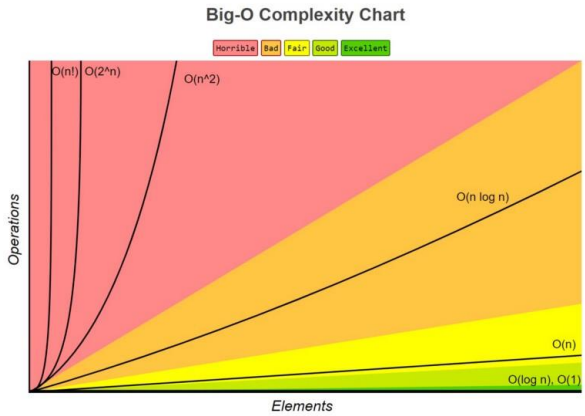
Kompleksitas waktu suatu algoritma dilambangkan dengan T(n). T(n) menyatakan banyaknya langkah yang dilakukan oleh algoritma untuk masukan sebesar n. Menghitung T(n) juga berguna untuk menghitung laju pertambahan waktu yang diperlukan oleh algoritma seiring dengan bertambahnya data masukan (n).

Langkah yang dihitung oleh T(n) merupakan operasi-operasi yang dilakukan oleh sebuah algoritma. Operasi tersebut dapat berupa operasi aritmatika, perbandingan, baca/tulis, pengisian nilai, dan lain-lain. Umumnya, T(n) tidak mempertimbangkan semua operasi yang dilakukan oleh suatu algoritma, tetapi hanya operasi yang umum dilakukan pada algoritma tersebut.

Kompleksitas waktu asimtotik adalah notasi kompleksitas waktu suatu algoritma ketika diberikan masukan n yang besar. Kompleksitas waktu asimtotik berguna ketika hal yang ingin ditinjau dari suatu algoritma hanya laju pertumbuhan waktu yang diperlukan oleh suatu algoritma ketika masukan n bertambah. Berikut adalah beberapa contoh notasi kompleksitas waktu asimtotik:

1. Notasi O-Besar (*Big-O Notation*)

   Misalkan terdapat suatu algoritma dengan kompleksitas waktu T(n), maka T(n) = O(f(n)) ketika terdapat konstanta C dan n_0 sedemikian rupa sehingga $T(n) \leq Cf(n)$ untuk $n \geq n_0$. Batas lebih atas T(n) ketika n berukuran besar adalah f(n). Notasi O-Besar suatu fungsi dapat ditentukan dengan melihat suku paling dominan dari T(n). Misalkan $T(n) = 3n^3 + 2n + 1$, maka $T(n) = O(n^3)$ karena suku paling dominan dari T(n) adalah n^3. Berikut adalah gambaran peningkatan kebutuhan waktu suatu algoritma seiring dengan bertambahnya masukan (n):



Gambar 5. Laju pertambahan kebutuhan waktu untuk beberapa fungsi dengan notasi O-Besar (Sumber: https://www.freecodecamp.org/news/all-you-need-to-know-about-big-o-notation-to-crack-your-next-coding-interview-9d575e7eec4/)

2. Notasi Ω-Besar (*Big-Omega Notation*)

   Jika terdapat suatu algoritma dengan kompleksitas waktu T(n), $T(n) = \Omega(g(n))$ ketika terdapat konstanta C dan n_0 sedemikian rupa sehingga $T(n) \geq Cg(n)$ untuk $n \geq n_0$.

3. Notasi Θ-Besar (*Big-Theta Notation*)

   Suatu algoritma dengan kompleksitas waktu T(n) berorde h(n) ($T(n) = \Theta(h(n))$) ketika $T(n) = O(h(n))$ dan $T(n) = \Omega(h(n))$.

III. Analisis Kompleksitas Algoritma Klosur Relasi

A. Implementasi Algoritma Klosur Relasi

Algoritma klosur relasi akan diimplementasikan dalam bahasa Python. Pada makalah ini, representasi relasi yang akan digunakan adalah representasi matriks. Alasan utama representasi tersebut digunakan adalah karena matriks mudah ditranslasikan dalam bentuk kode sebagai larik bersarang (*nested array*). Representasi matriks juga memudahkan dalam manipulasi hubungan pada suatu relasi karena penambahan/pengurangan elemen relasi dapat dilakukan dengan mengganti nilai 0 pada elemen matriks yang berkoresponden dengan 1 atau sebaliknya. Berikut adalah implementasi algoritma-algoritma klosur relasi:

1. Klosur Refleksif

   Algoritma klosur refleksif diimplementasikan dengan menjelajahi seluruh elemen diagonal utama pada matriks dan mengganti nilai pada elemen tersebut dengan nilai satu tanpa mengecek terlebih dahulu nilai awal elemen tersebut.

```python
def klosur_refleksif(relasi):
    for i in range(len(relasi)):
        relasi[i][i] = 1
    return relasi
```

Gambar 6. Algoritma pembentuk klosur refleksif dalam bahasa Python

2. Klosur Setangkup

   Algoritma klosur setangkup diimplementasikan dengan menjelajahi semua elemen pada matriks dan mencari elemen matriks yang bernilai 1 dan tidak terletak pada diagonal utama. Nilai elemen yang berseberangan dengan elemen tersebut akan diganti dengan 1.

```python
def klosur_setangkup(relasi):
    for i in range(len(relasi)):
        for j in range(len(relasi)):
            if relasi[i][j] == 1 and i != j:
                relasi[j][i] = 1
    return relasi
```

Gambar 7. Algoritma pembentuk klosur setangkup dalam bahasa Python

3. Klosur Menghantar

   Untuk algoritma klosur menghantar, diperlukan beberapa fungsi pembantu untuk menentukan komposisi relasi dan gabungan relasi.

   Algoritma klosur menghantar diimplementasikan dengan menginisialisasi matriks baru terlebih dahulu sebagai penyimpan hasil klosur. Kalang dalam digunakan untuk mendapatkan Mr^{j+1}, sedangkan kalang luar digunakan untuk menggabungkan Mr^{j+1} dengan gabungan komposisi pada iterasi sebelumnya.

```python
def komposisi_relasi(a, b):
    c = [[0 for i in range(len(a))] for j in range(len(a))]
    for i in range(len(c)):
        for j in range(len(c)):
            for k in range(len(c)):
                if a[i][k] * b[k][j] == 1:
                    c[i][j] = 1
                    break
    return c
```

Gambar 8. Fungsi komposisi relasi

```python
# Fungsi pembantu
def komposisi_relasi(a, b):
    c = [[0 for i in range(len(a))] for j in range(len(a))]
    for i in range(len(c)):
        for j in range(len(c)):
            for k in range(len(c)):
                if a[i][k] * b[k][j] == 1:
                    c[i][j] = 1
                    break
    return c

def union_relasi(a, b):
    for i in range(len(a)):
        for j in range(len(a)):
            if b[i][j] == 1:
                a[i][j] = 1
    return a
```

Gambar 9. Fungsi gabungan (*union*) dua relasi

```python
def klosur_transitif(relasi):
    n = len(relasi)
    result = [[0 for i in range(n)] for j in range(n)]
    for i in range(n):
        mr = relasi
        for j in range(i):
            mr = komposisi_relasi(mr, relasi)
        result = union_relasi(mr, result)
    return result
```

Gambar 10. Algoritma pembentuk klosur menghantar dalam bahasa Python

B. Pengujian Algoritma Klosur Relasi

Untuk memastikan bahwa algoritma berjalan dengan benar, dilakukan pengujian terhadap tiap algoritma terlebih dahulu. Berikut merupakan data uji dan hasil pengujian dari tiap algoritma:

```python
# Data uji
a = [[0, 1, 1],
     [0, 1, 0],
     [1, 0, 0]]

b = [[1, 0, 1, 0],
     [0, 0, 0, 1],
     [0, 1, 0, 1],
     [0, 0, 1, 1]]

c = [[0, 1, 1, 0],
     [0, 0, 1, 0],
     [1, 0, 0, 1],
     [0, 0, 0, 1]]
```

Gambar 11. Data uji yang dipakai

```
Hasil klosur refleksif matriks a:
[1, 1, 1]
[0, 1, 0]
[1, 0, 1]

Hasil klosur setangkup matriks b:
[1, 0, 1, 0]
[0, 0, 1, 1]
[1, 1, 0, 1]
[0, 1, 1, 1]

Hasil klosur menghantar matriks c:
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[0, 0, 0, 1]
```

Gambar 12. Hasil pengujian tiap algoritma

Berdasarkan hasil pengujian, dapat disimpulkan bahwa program sudah mengeluarkan *output* yang cocok dan berjalan dengan benar.

C. Analisis Kompleksitas Algoritma Klosur Relasi

Kompleksitas algoritma yang akan dianalisis adalah kompleksitas waktu (T(n)). Dalam kasus ini, n merupakan jumlah elemen dalam relasi atau jumlah baris/kolom pada matriks relasi.

1. Klosur refleksif

   Kompleksitas algoritma klosur refleksif didapatkan dari banyaknya operasi pengisian nilai yang dilakukan. Berdasarkan Gambar 5, algoritma tersebut selalu menjelajahi semua elemen pada diagonal utama matriks dan mengganti nilai tersebut dengan 1. Oleh karena itu, klosur refleksif memiliki kompleksitas waktu berupa $T_r(n) = n$ untuk kasus terbaik, terburuk, dan rata-rata. Suku paling dominan pada $T_r(n)$ adalah n sehingga $T_r(n)$ memiliki kompleksitas waktu asimtotik O(n).

   Suatu alternatif lain untuk algoritma klosur refleksif adalah melakukan pengecekan terlebih dahulu terhadap nilai pada elemen diagonal utama dan hanya mengganti nilai tersebut jika elemen bernilai 0. Namun, hal ini merupakan pendekatan yang kurang efisien karena perlu dilakukan operasi perbandingan terlebih dahulu untuk tiap elemen diagonal utama serta penggantian nilai elemen.

2. Klosur setangkup

   Operasi yang diperhitungkan dalam penentuan kompleksitas algoritma klosur setangkup adalah operasi perbandingan/pengecekan kondisi dan pengisian nilai. Berdasarkan Gambar 6, algoritma klosur setangkup selalu melakukan pengecekan kondisi pada tiap iterasi kalang bersarang, yaitu sebanyak n x n kali. Akan tetapi, pengisian nilai hanya dilakukan ketika kondisi bernilai *True*. Oleh karena itu, kompleksitas waktu untuk kasus terbaik algoritma ini adalah $T_{smin}(n) = n^2$, yaitu ketika tidak dilakukan pengisian nilai. Kompleksitas waktu kasus

terburuk algoritma ini adalah $T_{smax}(n) = 2n^2 – n$, yaitu ketika pengisian nilai selalu dilakukan pada tiap iterasi (kecuali ketika i = j).

Dengan meninjau kasus terburuk, suku paling dominan dari algoritma klosur setangkup adalah $2n^{2.}$ Oleh karena itu, algoritma klosur setangkup memiliki kompleksitas waktu asimtotik $O(n^2)$.

3. Klosur Menghantar

Untuk menentukan kompleksitas waktu algoritma klosur menghantar, kompleksitas waktu fungsi pembantu yang dipakai oleh algoritma tersebut juga harus dihitung. Kompleksitas waktu fungsi komposisi relasi hanya akan ditinjau dari jumlah operasi pengecekan kondisi yang dilakukan untuk menyederhanakan perhitungan. Dari Gambar 7, didapatkan bahwa fungsi komposisi relasi melibatkan tiga kalang bersarang. Perbandingan dilakukan pada tiap iterasi kalang tersebut sehingga dapat disimpulkan bahwa kompleksitas waktu fungsi komposisi relasi adalah $T_k(n) = n^3$.

Kompleksitas waktu fungsi gabungan relasi juga hanya akan ditinjau dari jumlah operasi pengecekan kondisi yang dilakukan untuk menyederhanakan perhitungan. Dari Gambar 7, didapatkan bahwa fungsi gabungan relasi (*union_relasi()*) melibatkan dua kalang bersarang dan perbandingan dilakukan pada tiap iterasi kalang tersebut. Oleh karena itu, kompleksitas waktu fungsi gabungan relasi adalah $T_u(n) = n^2$.

Berdasarkan Gambar 8, kalang luar algoritma berjalan sebanyak n kali dan kalang dalam berjalan sebanyak i kali. Oleh karena itu, fungsi komposisi relasi berjalan sebanyak 0 + 1 + 2 + … + n - 1 kali dalam algoritma klosur menghantar. Banyaknya pemanggilan fungsi komposisi relasi membentuk deret aritmatika bernilai $S_k = n(n - 1) / 2$. Fungsi gabungan relasi berada di dalam kalang luar dan di luar kalang dalam sehingga fungsi gabungan relasi dipanggil sebanyak $S_u = n$ kali.

Kompleksitas waktu algoritma klosur menghantar dapat ditentukan dengan menghitung perkalian antara kompleksitas waktu fungsi-fungsi pembantu dengan banyaknya pemanggilan fungsi tersebut dalam algoritma. Hal tersebut dapat dihitung dengan rumus $T_m(n) = S_kT_k(n) + S_uT_u(n)$. Dari hasil perhitungan, didapatkan $T_m(n) = 0.5n^5 - 0.5n^4 + n^3$. Suku yang paling dominan dari $T_m(n)$ adalah $0.5n^5$ sehingga algoritma klosur menghantar memiliki kompleksitas waktu asimtotik $O(n^5)$.

Dari hasil analisis ketiga algoritma klosur relasi, terdapat bahwa algoritma dengan kompleksitas waktu terbaik adalah algoritma klosur refleksif yang memiliki kompleksitas waktu asimtotik O(n). Algoritma terbaik selanjutnya adalah algoritma klosur setangkup dengan kompleksitas waktu asimtotik $O(n^2)$. Algoritma dengan kompleksitas waktu terburuk dimiliki oleh algoritma klosur menghantar dengan kompleksitas waktu asimtotik $O(n^5)$. Faktor utama buruknya algoritma klosur menghantar adalah perlunya dilakukan operasi perkalian matriks, yang sendirinya sudah memiliki kompleksitas $O(n^3)$, secara bersarang. Algoritma klosur transitif juga tidak seperti algoritma klosur lainnya yang hanya menjelajahi matriks sekali. Berikut adalah tabel perbandingan jumlah langkah yang diperlukan masing-masing algoritma untuk memproses sebuah matriks yang memiliki ukuran n x n:

n	$T_r(n)$	$T_{smax}(n)$	$T_m(n)$
1	1	1	1
2	2	6	16
5	5	45	1375
10	10	190	46000
50	50	4950	153250000
100	100	19900	4951000000
500	500	499500	1.56×10^{13}

Tabel 1. Perbandingan keperluan waktu untuk masing-masing algoritma klosur relasi

Berdasasrkan tabel tersebut, dapat dilihat bahwa perbedaan jumlah langkah yang diperlukan tiap fungsi semakin membesar seiring bertambahnya n. Perbedaan jumlah langkah pada $T_m(n)$ dengan algoritma lain sangat drastis. Diperlukan suatu cara agar kompleksitas pada algoritma klosur menghantar berkurang. Salah satu solusi yang dapat dilakukan adalah mencari cara untuk mengurangi kompleksitas pada fungsi komposisi relasi yang sendirinya memiliki kompleksitas waktu $O(n^3)$.

IV. KESIMPULAN

Pada makalah ini, telah dilakukan analisis terhadap kompleksitas algoritma tiga algoritma klosur relasi, yaitu klosur refleksif, setangkup, dan menghantar. Tujuan utama dilakukan hal tersebut adalah memahami kompleksitas waktu masing-masing algoritma dan implikasinya terhadap efisiensi. Hasil analisis menunjukkan bahwa tiap algoritma memiliki kompleksitas yang berbeda, yaitu O(n) untuk klosur refleksif, $O(n^2)$ untuk klosur setangkup, dan $O(n^5)$ untuk klosur menghantar. Kompleksitas waktu klosur menghantar yang sangat buruk menandakan bahwa perlu diadakan optimalisasi dalam algoritma klosur menghantar, terutama pada perhitungan komposisi relasi. Hal tersebut lebih ditekankan dengan berbagai macam aplikasi klosur menghantar, seperti menciptakan konektivitas antarkota.

V. UCAPAN TERIMA KASIH

Penulis mengucapkan puji syukur kepada Tuhan Yang Maha Esa atas berkat dan rahmat-Nya yang memberikan penulis kekuatan untuk menulis makalah berjudul “Analisis Kompleksitas Algoritma Pembentuk Klosur Refleksif, Setangkup, dan Menghantar dari Relasi” dengan baik. Penulis juga ingin mengucapkan terima kasih kepada Ibu Dr. Fariska Zakhralativa Ruskanda, S.T., M.T. selaku dosen pengampu mata kuliah Matematika Diskrit Semester Ganjil 2023/2024 kelas 02 yang telah memberikan ilmu sebagai bekal dalam penulisan makalah ini. Penulis juga ingin mengucapkan terima kasih kepada semua sumber referensi yang digunakan pada makalah ini. Penulis juga ingin mengucapkan permintaan maaf jika terdapat kesalahan dalam penulisan makalah ini.

REFERENSI

[1] Munir, Rinaldi. 2023. "Relasi dan Fungsi (Bagian 1)". https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/04-Relasi-dan-Fungsi-Bagian1-(2023).pdf, diakses pada 11 Desember 2023 19:20 WIB.

[2] Munir, Rinaldi. 2023. "Relasi dan Fungsi (Bagian 2)". https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/05-Relasi-dan-Fungsi-Bagian2-(2023).pdf, diakses pada 11 Desember 2023 19:30 WIB.

[3] Munir, Rinaldi. 2023. "Relasi dan Fungsi (Bagian 3)". https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/06-Relasi-dan-Fungsi-Bagian3-(2023).pdf, diakses pada 11 Desember 2023 21:10 WIB.

[4] Munir, Rinaldi. 2023. "Kompleksitas Algoritma (Bagian 1)". https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/24-Kompleksitas-Algoritma-Bagian1-2023.pdf, diakses pada 11 Desember 2023 21:15 WIB.

[5] Munir, Rinaldi. 2023. "Kompleksitas Algoritma (Bagian 2)". https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/25-Kompleksitas-Algoritma-Bagian2-2023.pdf, diakses pada 11 Desember 2023 21:15 WIB.

[6] Rosen K. H. (2019). *Discrete mathematics and its applications* (Eighth). McGraw-Hill.

TAUTAN KODE PROGRAM

https://github.com/neokoda/Makalah-Matdis/

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, 12 Desember 2023

Muhammad Neo Cicero Koda 13522108