LAPORAN TUGAS BESAR
ANALISIS EFISIENSI ALGORITMA PERPANGKATAN MATRIKS PERSEGI DENGAN PENDEKATAN ITERATIF DAN REKURSIF

Diajukan untuk memenuhi
salah satu tugas mata kuliah Analisis Kompleksitas Algoritma

Oleh :
Alif Ihsan (103012330079)
Arif Rahmatiana (103012300446)

PROGRAM STUDI S1 INFORMATIKA
FAKULTAS INFORMATIKA
UNIVERSITAS TELKOM

2024

A. Deskripsi Masalah

Pada banyak aplikasi komputasi, seperti pemodelan sistem dinamis, kriptografi, dan analisis graf, pemangkatan matriks persegi menjadi hal yang sering dilakukan. Dalam konteks ini, kita perlu menganalisis efisiensi dua pendekatan utama dalam mengimplementasikan algoritma perpangkatan matriks, yaitu pendekatan iteratif dan rekursif. Masalah yang dihadapi adalah bagaimana cara membandingkan waktu eksekusi (running time) dan kompleksitas kedua algoritma ini, serta dalam kondisi apa salah satu dari algoritma tersebut lebih efisien daripada yang lainnya.

B. Algoritma

1. Algoritma Iteratif

   Algoritma Iteratif adalah suatu prosedur yang menghasilkan pengulangan nilai variabel baru secara terus-menerus dari nilai lama.

   Program iteratif

```
// Fungsi untuk perkalian dua matriks A dan B menjadi matriks C
CodeMate
void multiplyMatrix(const matrix &A, const matrix &B, matrix &C)
{
    for (int i = 0; i < N; ++i)
    {
        for (int j = 0; j < N; ++j)
        {
            C[i][j] = 0;
            for (int k = 0; k < N; ++k)
            {
                C[i][j] = (C[i][j] + (1LL * A[i][k] * B[k][j]) % MOD) % MOD;
            }
        }
    }
}
```

```
// Fungsi untuk menghitung pangkat matriks A^pangkat
CodeMate
void matrixPower(const matrix &A, matrix &result, int pangkat)
{
    identityMatrix(result, N);

    matrix temp(N, vector<ll>(N));
    matrix base = A;

    while (pangkat > 0)
    {
        if (pangkat % 2 == 1)
        {
            multiplyMatrix(result, base, temp);
            result = temp;
        }
        multiplyMatrix(base, base, temp);
        base = temp;
        pangkat /= 2;
    }
}
```

Kompleksitas Waktu T(n)

a) Best Case

   Kemungkinan terbaik terjadi ketika pangkat matriks adalah nilai minimum, yaitu pangkat = 0, maka program hanya membuat matriks identitas. Kompleksitas waktu terbaiknya adalah:

   $T_{best}(n) = O(n^2)$.

b) Worst Case

   Kemungkinan terburuk terjadi ketika pangkat matriks adalah nilai maksimum dan bilangan besar.

   Fungsi multiplyMatrix

   Fungsi ini mengalikan dua matriks A dan B, dan menyimpan hasilnya di matriks C. Untuk mengalikan dua matriks N * N, kita memerlukan tiga loop bersarang:

   - Loop pertama untuk baris matriks A (iterasi sebanyak N),
   - Loop kedua untuk kolom matriks B (iterasi sebanyak N),
   - Loop ketiga untuk perhitungan elemen matriks hasil perkalian (iterasi sebanyak N).

   Kompleksitas: **$O(n^3)$**.

   Fungsi MatrixPower

   Fungsi ini menghitung pangkat dari matriks A dengan metode eksponen biner (dengan pembagian pangkat yang efisien). Dalam setiap iterasi, jika pangkat ganjil, fungsi akan mengalikan matriks hasil dengan basis, dan pada setiap langkah pangkat dibagi dua. Dengan demikian, jumlah iterasi berbanding logaritmik dengan nilai pangkat.

Kompleksitas waktu untuk setiap iterasi:

- Setiap iterasi memanggil fungsi multiplyMatrix, yang memiliki kompleksitas **$O(n^3)$**.
- Total iterasi adalah **$O(n^3 \log p)$.**

Jadi, total kompleksitas waktu untuk menghitung pangkat matriks adalah:

$O(n^3 \log p)$.

2. Algoritma Rekursif

   Algoritma Rekursif adalah suatu prosedur yang menghasilkan pengulangan dari fungsi diri sendiri.

   Program rekursif

```
// Matriks identitas untuk ukuran N x N
CodeMate
void identityMatrix(matrix& A, int n, int i = 0, int j = 0) {
    if (i == n){
        return;
    }
    if (j == n) {
        identityMatrix(A, n, i + 1, 0);
        return;
    }
    if (i == j) {
        A[i][j] = 1;
    } else {
        A[i][j] = 0;
    }
    identityMatrix(A, n, i, j + 1);
}
```

```cpp
// Fungsi untuk perkalian dua matriks A dan B menjadi matriks C
CodeMate
void multiplyMatrix(const matrix& A, const matrix& B, matrix& C, int i = 0, int j = 0, int k = 0) {
    if (i == N){
        return;
    }
    if (j == N) {
        multiplyMatrix(A, B, C, i + 1, 0, 0);
        return;
    }
    if (k == N) {
        C[i][j] = C[i][j] % MOD;
        multiplyMatrix(A, B, C, i, j + 1, 0);
        return;
    }
    C[i][j] = (C[i][j] + (1LL * A[i][k] * B[k][j]) % MOD) % MOD;
    multiplyMatrix(A, B, C, i, j, k + 1);
}
```

```cpp
// Fungsi untuk menghitung pangkat matriks A^pangkat
CodeMate
void matrixPower(const matrix& A, matrix& result, int pangkat, int i = 0, int j = 0) {
    if (pangkat == 0) {
        identityMatrix(result, N);
        return;
    }
    if (pangkat == 1) {
        result = A;
        return;
    }
    matrix temp(N, vector<ll>(N, 0));
    matrixPower(A, temp, pangkat / 2);
    multiplyMatrix(temp, temp, result);

    if (pangkat % 2 == 1) {
        matrix temp2(N, vector<ll>(N, 0));
        multiplyMatrix(result, A, temp2);
        result = temp2;
    }
}
```

Kompleksitas Waktu T(n)

Pada program tersebut, fungsi utama yang berperan adalah matrixPower dan multiplyMatrix

a) Best Case

Kemungkinan terbaik terjadi ketika pangkat matriks adalah nilai minimum, yaitu pangkat = 0, maka program hanya membuat matriks identitas. Kompleksitas waktu terbaiknya adalah:

$T_{best}(n) = O(n^2)$.

b) Worst Case

Kemungkinan terburuk terjadi ketika pangkat matriks adalah nilai maksimum dan bilangan besar. semakin dalam rekursi dalam fungsi matrixPower, sehingga membutuhkan lebih banyak waktu untuk menyelesaikan perpangkatan matriks.

Dari analisis di atas, persamaan waktu rekursif adalah:

Fungsi multiplyMatrix

Dalam fungsi ini, dilakukan perkalian dua matrix A dan B yang berukuran n x n, untuk menghasilkan matrix C.

*/p = pangkat

T(n) = O(n^3)

Fungsi matrixPower

T(pangkat) = { **O(n^2)** **, jika p = 1**

{ **T(p/2) + O(n^3)** **, jika p > 1**

Dengan menggunakan teorema master, kita dapat menentukan

a = 1 (hanya ada satu rekursi),

b = 2 (pangkat dibagi 2),

d = 3 (perkalian matriks O(n^3)).

Bandingkan a dan b^d dengan,

$$T(n) \in \begin{cases} O(n^d) & a < b^d \\ O(n^d \log n) & a = b^d \\ O(n^{\log_b a}) & a > b^d \end{cases}$$

a = 1, $b^d = 2^3 = 8$

$a < b^d$ maka,

T(n) = O(n^3). Namun, ada tambahan faktor rekursi karena masalah pangkat dibagi 2 (pangkat/2) dengan kedalaman rekursi **O**(**log p),** maka kompleksitasnya adalah **T(n) = O(n^3 log p)**

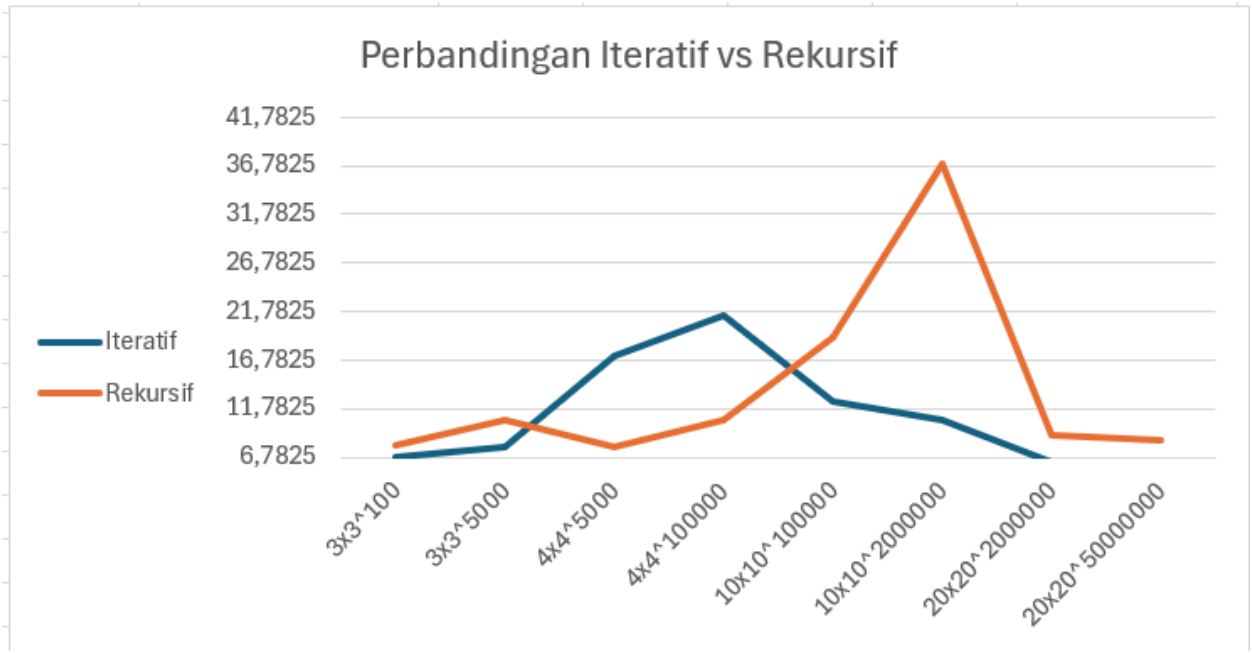
Oleh karena itu, kompleksitas keseluruhan dari algoritma tersebut adalah **T(n) = O(n^3 log p)**

C. Grafik Perbandingan *Running Time*

Grafik di bawah ini menunjukkan perbandingan running time antara kedua algoritma, yaitu algoritma iteratif dan algoritma rekursif.

Test Code :

Ukuran Matrix	Pangkat	Iteratif	Rekursif
3	100	6,7825	7,99901
3	5000	7,79867	10,5567
4	5000	17,2599	7,75466
4	100000	21,3167	10,5852
10	100000	12,4462	19,1407
10	2000000	10,5144	36,9619
20	2000000	6,31182	9,00594
20	50000000	5,92413	8,56828



D. Analisis Perbandingan Kedua Algoritma

Metode Iteratif:

Dalam praktiknya, metode iteratif sering kali lebih cepat karena menghindari overhead dari panggilan fungsi. Setiap iterasi dilakukan dalam satu blok kode, yang dapat dieksekusi lebih efisien oleh compiler.

Metode Rekursif:

Meskipun kompleksitasnya sama, metode rekursif mungkin lebih lambat dalam eksekusi karena overhead dari panggilan fungsi dan pengembalian nilai.

Berdasarkan grafik di atas, meskipun metode iteratif maupun rekursif memiliki kompleksitas yang sama, yaitu $O(N^3 * \log(\text{pangkat}))$. Namun, metode iteratif cenderung lebih baik dalam penggunaan memori dan kecepatan eksekusi dan lebih disarankan untuk digunakan. Penggunaan metode yang tepat tergantung pada konteks dan batasan yang ada, seperti ukuran matriks dan nilai pangkat yang ingin dihitung.

E. LINK GITHUB

alifihsan7/Tugas-Besar-Analisis-Kompleksitas-Algoritma atau https://github.com/alifihsan7/Tugas-Besar-Analisis-Kompleksitas-Algoritma

F. Poster

Universitas Telkom

ANALISIS EFISIENSI ALGORITMA PERPANGKATAN MATRIKS PERSEGI DENGAN PENDEKATAN ITERATIF DAN REKURSIF

DESKRIPSI PERMASALAHAN

Pemangkatan matriks persegi sering digunakan dalam aplikasi seperti pemodelan sistem dinamis, kriptografi, dan analisis graf. Dalam konteks ini, kita perlu menganalisis efisiensi dua pendekatan utama dalam mengimplementasikan algoritma perpangkatan matriks, yaitu pendekatan iteratif dan rekursif.

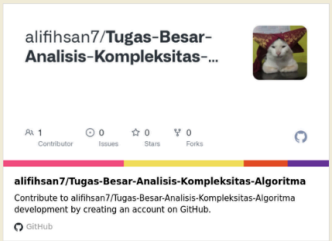
ALGORITMA ITERATIF

Algoritma Iteratif adalah suatu prosedur yang menghasilkan pengulangan nilai variabel baru secara terus-menerus dari nilai awal.

ALGORITMA REKURSIF

Algoritma Rekursif adalah suatu fungsi yang menghasilkan pengulangan dari fungsi diri sendiri.

GITHUB



https://github.com/alifihsan7/Tugas-Besar-Analisis-Kompleksitas-Algoritma.git

GRAFIK PERBANDINGAN

Oleh :
Alif Ihsan (103012330079)
Arif Rahmatiana (103012300446)

G. Referensi

Kaluge, G. R. (n.d.). *Kompleksitas dari algoritma-algoritma untuk menghitung bilangan Fibonacci*. Program Studi Teknik Informatika, Institut Teknologi Bandung. https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2009-2010/Makalah0910/MakalahStrukdis0910-028.pdf