Perancangan dan Analisis Algoritma

“ Binary Exponentiation ”

Disusun Oleh :

Mubarakh Hayatna Khairy

21343083

Dosen Pengampu :

Randi Proska Sandra, S.Pd.,M.Sc.

Prodi Informatika

Departemen Elektronika

Fakultas Teknik

Universitas Negeri Padang

1. PENJELASAN PROGRAM/ALGORITMA

Binary Exponential adalah salah satu algoritma yang digunakan untuk mempercepat operasi integer eksponensial. Algoritma ini bekerja dengan cara membagi pangkat menjadi pangkat genap secara berulang-ulang hingga mencapai pangkat yang diinginkan.

Misalnya, jika Anda ingin menghitung 2^10, algoritme Eksponensial Biner mengubahnya menjadi 2^8 * 2^2, yang kemudian dapat dihitung lebih cepat. Dengan cara ini, algoritma Eksponensial Biner mengurangi jumlah operasi yang diperlukan untuk menghitung eksponensial dari O(n) menjadi O(log n), di mana n adalah eksponen.

Berikut adalah langkah-langkah untuk mengimplementasikan Algoritma Eksponensial Biner:

a. Tentukan bilangan yang akan dipangkatkan (basis) dan pangkat yang diinginkan (eksponen).
b. Ubah eksponen menjadi representasi binernya.
c. Mulai dari bit terakhir (LSB) hingga bit pertama (MSB) dalam representasi biner, lakukan hal berikut:
    - Jika bit saat ini adalah 1, kalikan hasil sebelumnya dengan basis.
    - Naikkan basis ke pangkat 2 dan simpan hasilnya sebagai basis baru.
d. Setelah selesai, hasil akhirnya adalah perkalian dari semua iterasi.

Algoritma Eksponensial Biner sangat berguna dalam banyak aplikasi, terutama dalam Matematika dan Ilmu Komputer. Contoh penggunaan algoritma ini adalah dalam enkripsi RSA, menghitung nilai Fibonacci yang besar, menghitung matriks penentu, dan sebagainya.

Algoritma Eksponensial Biner juga dapat diterapkan untuk menghitung eksponensial modular, yaitu menghitung kekuatan modulo n. Untuk menghitung hasil modulo n, setiap kali hasilnya dikuadratkan (atau dikalikan dengan hasil kali dasar dengan hasil sebelumnya), hasilnya harus diambil modulo n.

Misalnya, jika Anda ingin menghitung 2^10 mod 5, algoritme Eksponensial Biner akan mengubahnya menjadi 2^8 mod 5 * 2^2 mod 5 = 1 * 4 = 4. Dalam hal ini, pangkat eksponensial dari 2^ 10 diperoleh dengan mengalikan 2 ^8 mod 5 dengan 2^2 mod 5 lalu keluarkan hasilnya modulo 5.

Selain itu, untuk mengoptimalkan kinerja algoritma Binary Exponentiation dapat dilakukan beberapa optimasi, seperti penggunaan unsigned integer untuk menghindari

integer overflow, penggunaan bit shifting untuk menggantikan operasi pembagian dan modulo, serta pemanfaatan properti integer seperti bilangan genap dan ganjil.

Efisiensi luar biasa aturan Horner berkurang ketika metode ini diterapkan untuk menghitung an, yang merupakan nilai xn pada x = a. Bahkan, itu merosot menjadi kekuatan kasar mengalikan a dengan sendirinya, dengan penambahan angka nol yang tidak berguna di antaranya. Sejak perhitungan an (sebenarnya, mod m) adalah operasi penting dalam beberapa cara penting metode pengujian dan enkripsi primality, kami sekarang mempertimbangkan dua algoritma untuk perhitungan berdasarkan ide perubahan-representasi. Mereka berdua mengeksploitasi representasi biner dari eksponen n, tetapi salah satunya memproses biner ini string kiri ke kanan, sedangkan yang kedua melakukannya dari kanan ke kiri. Membiarkan

$$n = b_1 ... b_i ... b_0$$

menjadi string bit yang mewakili bilangan bulat positif n dalam sistem bilangan biner.

Artinya nilai n dapat dihitung sebagai nilai polinomial

$$p(x) = b_1 x^1 + ... + b_i x^i + ... + b_0 \quad (6.4)$$

pada x = 2. Misalnya, jika n = 13, representasi binernya adalah 1101 dan

$$13 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$$

Mari menghitung nilai polinomial ini dengan menerapkan aturan Horner dan lihat apa yang tersirat dalam pengoperasian metode untuk menghitung daya

$$a^n = a^{p(2)} = a^{b_1 2^1 + ... + b_i 2^i + ... + b_0}$$

2. PSEUDOCODE

```
//Computes an by the left-to-right binary exponentiation
algorithm

//Input: A number a and a list b(n) of binary digits bI
,...,b0

// in the binary expansion of a positive integer n

//Output: The value of an

product ← a

for i ← I − 1 downto 0 do

product ← product * product

if bi = 1 product ← product * a

return produc
```

Penjelasan :

Pseudocode di atas adalah implementasi algoritma eksponensiasi biner dari kiri ke kanan. Algoritma ini menghitung nilai a^n, di mana n adalah bilangan bulat positif yang direpresentasikan dalam bentuk biner (dalam daftar b(n) yang terdiri dari n bit).

Langkah-langkah algoritma ini adalah sebagai berikut:

- Inisialisasi nilai produk dengan a, karena a^1 = a.
- Iterasi dari bit paling signifikan ke bit paling tidak signifikan dalam daftar b(n). Setiap kali melakukan iterasi, produk akan dikuadratkan (mengalikan dirinya sendiri), dan jika bit ke-i dalam daftar b(n) adalah 1, maka produk akan dikalikan dengan a.
- Setelah semua bit di dalam daftar b(n) telah diproses, nilai produk akan mengandung hasil a^n, dan akan dikembalikan sebagai keluaran dari algoritma.

Dalam algoritma ini, kita menghitung eksponen dari a dalam waktu O(log n) langkah, karena setiap kali kita mengalikan produk dengan dirinya sendiri, kita menggandakan eksponen. Dalam kasus terbaik, jika semua bit dalam daftar b(n) adalah 0, maka kita hanya perlu melakukan satu operasi perkalian, sehingga waktu eksekusi O(1). Sedangkan, dalam kasus terburuk, jika semua bit dalam daftar b(n) adalah 1, maka kita akan melakukan O(log n) kali operasi perkalian.

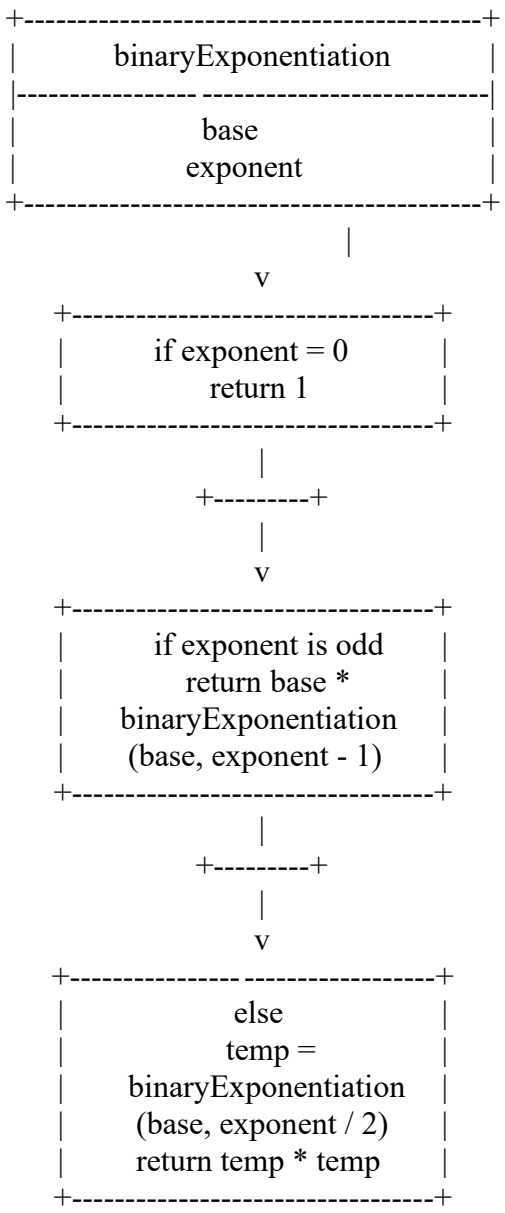
```
function binaryExponentiation(basis, exponen):
    if exponen = 0:
        return 1

    if exponen is odd:
        return basis * binaryExponentiation(basis,
exponen - 1)

    else:
        temp = binaryExponentiation(basis, exponen / 2)
        return temp * temp
```

Penjelasan:

Algoritma ini menggunakan fakta bahwa x^n dapat dihitung sebagai (x^2)^(n/2) jika n genap, dan x * x^(n-1) jika n ganjil. Ini mengarah pada pendekatan bagi dan taklukkan yang dapat menghitung eksponensial dalam waktu logaritma.

Fungsi mengambil basis dan eksponen, dan secara rekursif menghitung hasilnya menggunakan rumus di atas. Jika eksponen adalah 0, ia mengembalikan 1 (karena angka apa pun yang dipangkatkan 0 adalah 1). Jika eksponen ganjil, secara rekursif menghitung hasil untuk eksponen - 1 dan mengalikannya dengan basis. Jika eksponen genap, secara rekursif menghitung hasil untuk eksponen / 2 dan mengkuadratkannya.

Diagram blok untuk pseudocode binary exponentiation dapat diilustrasikan sebagai berikut:



3. SOURCE CODE

Berikut ini adalah contoh program Python yang menggunakan algoritma binary exponentiation:

```
# Fungsi binary exponentiation
def binary_exponentiation(base, exponent, modulus=None):
    """
    Fungsi untuk menghitung bilangan pangkat dengan algoritma binary exponentiation.

    Args:
        base (int): bilangan yang akan dipangkatkan
        exponent (int): pangkat dari bilangan
        modulus (int, optional): bilangan modulus untuk operasi modulo. Defaults to None.

    Returns:
        int: hasil pangkat bilangan
    """
    # Inisialisasi nilai result dengan 1
    result = 1

    # Jika modulus tidak diberikan, maka diambil modulo dengan 1
    base %= modulus if modulus else 1

    # Perulangan selama nilai exponent masih lebih besar dari 0
    while exponent > 0:
        # Jika exponent ganjil, maka result dikalikan dengan base
        if exponent % 2 == 1:
            result = (result * base) % modulus if modulus else result * base

        # Base dikuadratkan
        base = (base * base) % modulus if modulus else base * base

        # Exponent dibagi dua
        exponent //= 2

    # Hasil akhir diambil modulo dengan modulus jika modulus diberikan, atau tidak diambil modulo jika tidak ada modulus
    return result % modulus if modulus else result


# Contoh penggunaan program
if __name__ == '__main__':
    # Memasukkan bilangan yang akan dipangkatkan dan pangkatnya
    base = 237
    exponent = 919

    # Memasukkan bilangan modulus jika diperlukan
    modulus = 1057
```

```
        # Menghitung hasil pangkat dengan menggunakan binary
exponentiation
        result = binary_exponentiation(base, exponent, modulus)

        # Menampilkan hasil pangkat
        print(f"Hasil dari {base}^{exponent} mod {modulus} adalah
{result}")
```

Pada baris ke-3 hingga ke-13, terdapat definisi fungsi binary_exponentiation. Fungsi ini memiliki tiga parameter, yaitu base, exponent, dan modulus. base dan exponent adalah bilangan yang akan dipangkatkan dan pangkatnya, sedangkan modulus adalah bilangan modulus untuk operasi modulo. Jika modulus tidak diberikan, maka fungsi akan mengembalikan hasil pangkat base^exponent. Fungsi ini menggunakan algoritma binary exponentiation untuk menghitung hasil pangkat.

Pada baris ke-17, variabel result diinisialisasi dengan nilai 1. Pada baris ke-20, nilai base diambil modulo dengan modulus jika modulus diberikan, atau 1 jika tidak ada modulus.

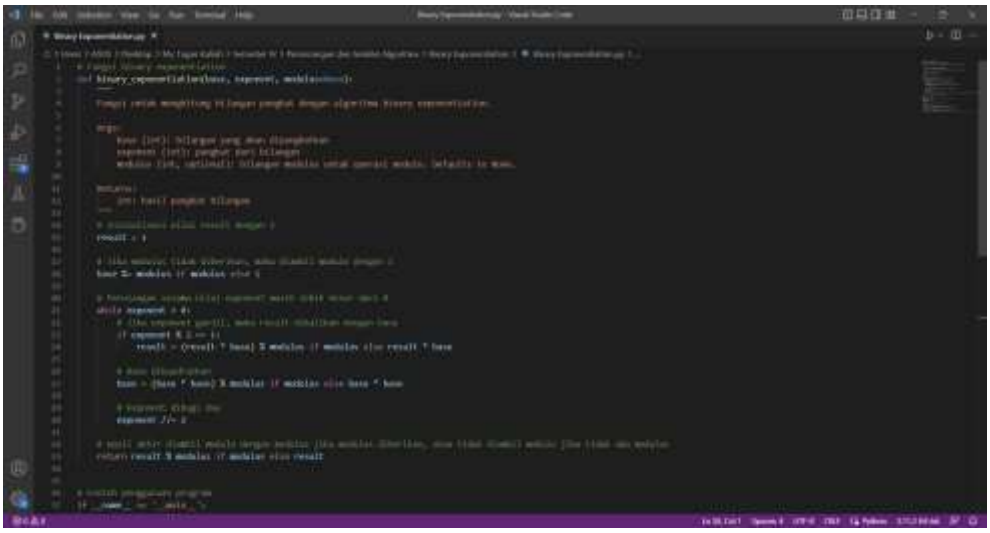
Pada baris ke-23 hingga ke-30, dilakukan perulangan selama nilai exponent masih lebih besar dari 0. Dalam setiap iterasi, dilakukan pengecekan apakah exponent adalah bilangan genap atau ganjil. Jika exponent ganjil, maka result akan dikalikan dengan base dan kemudian diambil modulo dengan modulus jika modulus diberikan, atau tidak diambil modulo jika tidak ada modulus. Kemudian, nilai base akan d ikuadratkan dan kemudian diambil modulo dengan modulus jika modulus diberikan, atau tidak diambil modulo jika tidak ada modulus. Nilai exponent kemudian dibagi dua.

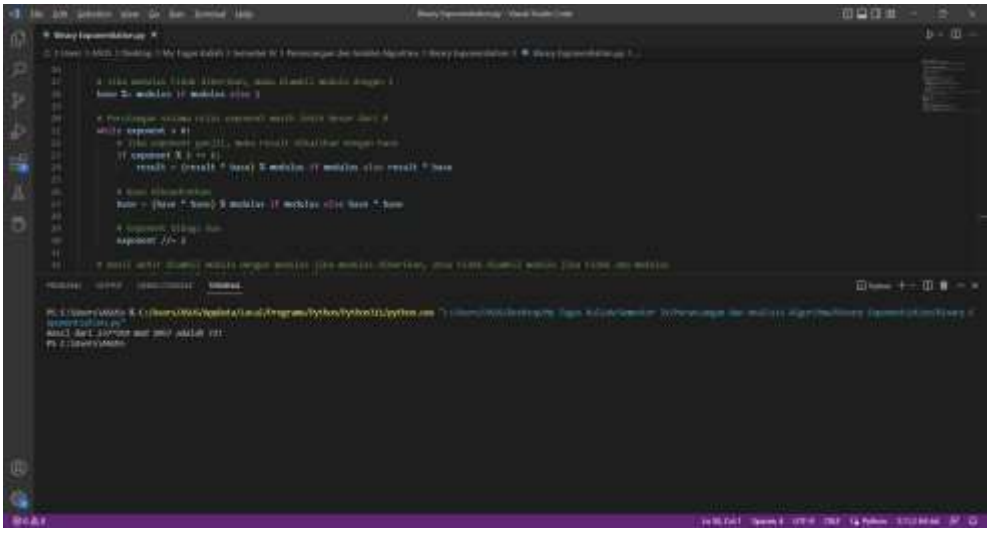
Setelah perulangan selesai dilakukan, pada baris ke-32 hingga ke-35, hasil akhir diambil modulo dengan modulus jika modulus diberikan, atau tidak diambil modulo jika tidak ada modulus. Nilai hasil akhir kemudian dikembalikan oleh fungsi.

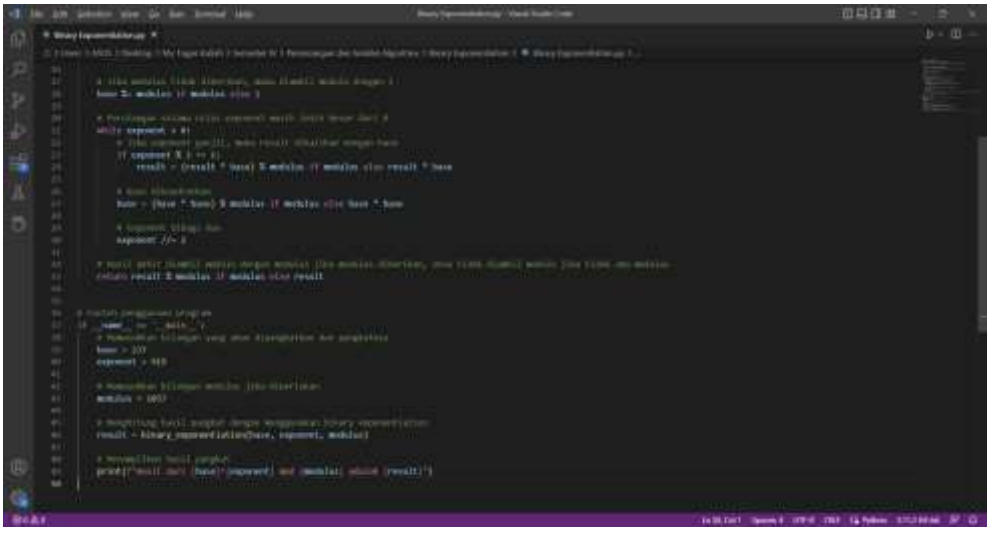
Pada baris ke-38 hingga ke-48, terdapat contoh penggunaan program. Pada contoh ini, base dan exponent diberikan nilai 237 dan 919. modulus juga diberikan nilai 1057. Kemudian, fungsi binary_exponentiation dipanggil dengan tiga parameter tersebut sebagai argumen. Hasil pangkat yang dikembalikan oleh fungsi kemudian disimpan ke dalam variabel result. Hasil pangkat kemudian ditampilkan ke layar menggunakan perintah print.

Dalam program ini, jika ingin menghitung pangkat tanpa menggunakan modulus, maka modulus dapat diabaikan pada saat memanggil fungsi binary_exponentiation. Namun, jika ingin menghitung hasil pangkat dengan menggunakan modulus, maka nilai modulus harus diberikan pada saat memanggil fungsi.

Hasil Program :







4. ANALISIS KEBUTUHAN WAKTU

1) Dari Pseudocode

a. Analisis menggunakan operasi/instruksi:

- Assignment: 2 (product ← a, product ← product * product)
- Multiplication: 2n (n adalah panjang list b)
- Conditional: n (untuk mengecek apakah bi = 1)
- Multiplication (inside conditional): n

Jumlah total operasi: 3n + 2

b. Analisis menggunakan jumlah operasi abstrak:

- Assignment: 2
- Multiplication: 2n
- Conditional: n
- Multiplication (inside conditional): n

Jumlah total operasi abstrak: 4n + 2

c. Analisis menggunakan pendekatan best-case, worst-case, dan average-case:

Best-case: ketika semua bilangan di list b bernilai 0, maka tidak perlu melakukan operasi perkalian lagi setelah operasi assignment awal, sehingga jumlah operasi adalah 2.

Worst-case: ketika semua bilangan di list b bernilai 1, maka setiap operasi perkalian akan dijalankan dan kondisi di dalam perulangan akan selalu bernilai benar, sehingga jumlah operasi adalah 4n + 2.

Average-case: tidak dapat dihitung secara tepat karena tergantung pada nilai-nilai yang terdapat pada list b.

Dari ketiga pendekatan di atas, dapat disimpulkan bahwa kompleksitas waktu algoritma ini adalah O(n) karena jumlah operasi terbesar terdapat pada operasi perulangan yang bergantung pada panjang list b.

2) Source code

Algoritma binary exponentiation adalah sebuah algoritma yang digunakan untuk menghitung hasil pangkat bilangan dengan cepat. Berikut adalah analisis kebutuhan waktu algoritma tersebut menggunakan tiga pendekatan yang berbeda:

a. Analisis berdasarkan operasi/instruksi yang dieksekusi:

Pada algoritma binary exponentiation, terdapat beberapa operasi yang dieksekusi pada setiap iterasi perulangan. Operasi yang dilakukan antara lain:

- Assignment: digunakan untuk menginisialisasi nilai result dan base
- Modulo (%): digunakan untuk mengambil modulo bilangan basis dan modulus (jika modulus diberikan)
- Perkalian (*): digunakan untuk menghitung hasil perkalian antara result dengan base, atau base dengan dirinya sendiri
- Pembagian bilangan bulat (//): digunakan untuk membagi nilai exponent dengan dua

Dengan begitu, jumlah operasi/instruksi yang dieksekusi oleh algoritma ini tergantung pada nilai pangkat yang diberikan. Secara umum, algoritma ini memiliki kompleksitas waktu O(log n) di mana n adalah nilai pangkat.

b. Analisis berdasarkan jumlah operasi abstrak:

Jumlah operasi abstrak pada algoritma binary exponentiation tergantung pada nilai pangkat. Jumlah operasi yang dilakukan pada setiap iterasi perulangan antara lain:

- Perkalian: 1 kali
- Modulo: 1 kali (jika modulus diberikan)
- Pembagian bilangan bulat: 1 kali
- Pengecekan bilangan ganjil: 1 kali (melalui operasi modulus)
- Assignment: 1 kali

Dengan demikian, jumlah operasi abstrak pada algoritma ini adalah O(log n).

c. Analisis menggunakan pendekatan best-case, worst-case, dan average-case:

Best-case: jika nilai pangkat exponent adalah 0, maka hasil pangkat akan selalu sama dengan 1. Oleh karena itu, algoritma ini hanya akan melakukan satu operasi assignment dan mengembalikan nilai 1 sebagai hasil. Dalam hal ini, waktu eksekusi algoritma adalah O(1).

Worst-case: jika nilai pangkat exponent sangat besar, maka algoritma ini harus melakukan banyak iterasi perulangan untuk menghitung hasil pangkat. Dalam hal ini, waktu eksekusi algoritma adalah O(log n).

Average-case: waktu eksekusi rata-rata algoritma ini adalah O(log n), mengingat jumlah operasi yang dilakukan pada setiap iterasi perulangan tidak

berbeda secara signifikan untuk setiap nilai pangkat. Namun, hal ini juga dapat dipengaruhi oleh distribusi nilai pangkat yang digunakan pada kasus tertentu.

5. REFERENSI

Introduction to the Design & Analysis of Algorithms 3rd Edition karya Anany Levitin

GeeksforGeeks. (2022). Binary Exponentiation (Power in Logarithmic time). Diakses pada 8 Maret 2023, dari https://www.geeksforgeeks.org/write-a-c-program-to-calculate-powxn/

CP-Algorithms. (2022). Binary Exponentiation. Diakses pada 8 Maret 2023, dari https://cp-algorithms.com/algebra/binary-exp.html

Khan Academy. (2022). Exponentiation algorithms. Diakses pada 8 Maret 2023, dari https://www.khanacademy.org/computing/computer-science/algorithms/recursive-algorithms/a/exponentiation-algorithms

Brilliant. (2022). Binary Exponentiation. Diakses pada 8 Maret 2023, dari https://brilliant.org/wiki/binary-expo/

Baase, S. & Van Gelder, A. (2008). Computer Algorithms: Introduction to Design and Analysis. Pearson Prentice Hall.

6. Lampiran LINK pengumpulan Github

https://github.com/MrKhairy10/Perancangan-Analisis-Algoritma---Transform-and-Conquer---Binary-Exponentiation/upload/main