**PERANCANGAN DAN ANALISIS ALGORITMA**

**BINARY EXPONENTIATION**

**DOSEN PENGAMPU : Randi Proska Sandra, S.Pd, M.Sc**

**Sesi : 202323430048**

****

**FADHIL RAHMAT**

**22343045**

**PRODI INFORMATIKA**

**FAKULTAS TEKNIK**

**DEPARTEMEN ELEKTRONIKA**

**UNIVERSITAS NEGERI PADANG**

**TAHUN AJARAN 2023/ 2024**

**BINARY EXPONENTIATION**

1. **Penjelasan Program/Algoritma**

Strategi algoritma transform-and-conquer adalah pendekatan algoritma yang mengubah masalah menjadi versi yang lebih sederhana atau berbeda, kemudian menyelesaikannya. Ini sering digunakan dalam algoritma yang memecah masalah menjadi sub-masalah yang lebih kecil dan lebih mudah untuk diselesaikan. Strategi ini memungkinkan peningkatan efisiensi dan kompleksitas waktu yang lebih rendah dibandingkan dengan pendekatan iteratif atau rekursif tradisional.

Algoritma Binary Exponentiation adalah contoh dari strategi transform-and-conquer. Algoritma ini digunakan untuk menghitung (A^B) dengan cara yang lebih efisien dibandingkan dengan pendekatan iteratif sederhana, yang memerlukan waktu O(B). Dengan Binary Exponentiation, kita dapat menghitung hasil dalam waktu O(log(B)), yang sangat berguna untuk kasus di mana B sangat besar, seperti dalam kompetisi pemrograman atau aplikasi yang memerlukan perhitungan eksponen dengan nilai yang sangat besar.

Ide di balik Binary Exponentiation adalah untuk memanfaatkan sifat eksponen biner. Jika B adalah bilangan genap, kita dapat menghitung (A^(B/2))^2, yang ekuivalen dengan (A^B). Jika B adalah bilangan ganjil, kita dapat menghitung A \* (A^(B-1)/2)^2. Dengan cara ini, kita dapat mengurangi jumlah perhitungan yang diperlukan dan mencapai kompleksitas waktu yang lebih baik.

Implementasi Binary Exponentiation dapat dilakukan secara rekursif atau iteratif. Implementasi rekursif lebih sederhana tetapi mungkin tidak efisien untuk nilai B yang sangat besar karena dapat menyebabkan overhead rekursi. Implementasi iteratif, di sisi lain, lebih efisien dalam hal waktu dan ruang karena tidak memerlukan panggilan fungsi rekursif.

Binary Exponentiation sangat berguna dalam kompetisi pemrograman dan aplikasi lainnya di mana perhitungan eksponen dengan eksponen yang sangat besar diperlukan, karena memiliki kompleksitas waktu O(log(B)), yang memungkinkan untuk menghitung hasil dengan cepat dan efisien.

1. **Pseudocode**

procedure binary\_exponentiation(base, exponent, mod):

result := 1

while exponent > 0:

if exponent % 2 == 1:

result := (result \* base) % mod

base := (base \* base) % mod

exponent := exponent // 2

return result

base := 5

exponent := 5

mod := 100000007

print("Hasilnya adalah", binary\_exponentiation(base, exponent, mod))

1. **Source Code :**

def eksponensiasi\_biner(basis, eksponen, mod):

hasil = 1

while eksponen > 0:

if eksponen % 2 == 1: # Jika eksponen ganjil

hasil = (hasil \* basis) % mod # Perbarui hasil dengan perkalian dengan basis dan modulo

basis = (basis \* basis) % mod # Perbarui basis dengan kuadratnya dan modulo

eksponen = eksponen // 2 # Kurangi eksponen menjadi setengahnya

return hasil

# Contoh penggunaan

basis = 5

eksponen = 5

mod = 100000007 # Angka baru untuk modulo

print("Hasilnya adalah", eksponensiasi\_biner(basis, eksponen, mod))

1. **Hasilnya**



1. **Analisis Kebutuhan Waktu**
2. Analisis menyeluruh dengan memperhatikan operasi/instruksi yang di eksekusi berdasarkan operator penugasan atau assignment (=, +=, \*=, dan lainnya) dan operator aritmatika (+, %, \* dan lainnya)

Dalam setiap iterasi while loop pada algoritma binary\_exponentiation, terdapat beberapa operasi aritmatika yang dieksekusi. Operasi ini meliputi perkalian (\*), modulus (%), dan pembagian (//). Secara umum, operasi-operasi ini memiliki kompleksitas waktu konstan, atau O(1), karena waktu yang dibutuhkan untuk mengeksekusi operasi tersebut tidak tergantung pada ukuran masukan. Namun, perlu diperhatikan bahwa jumlah operasi dalam setiap iterasi bergantung pada nilai eksponen yang digunakan. Semakin besar eksponen, semakin banyak iterasi dan operasi aritmatika yang dieksekusi.

1. Analisis berdasarkan jumlah operasi abstrak atau operasi khas.

Dalam konteks algoritma eksponensiasi biner, jumlah operasi abstrak yang dilakukan bergantung pada panjang biner dari eksponen yang digunakan. Jika n adalah panjang biner dari eksponen, maka jumlah operasi yang dieksekusi adalah sekitar O(n). Hal ini karena setiap bit dalam representasi biner eksponen akan memerlukan satu iterasi dalam loop while. Dengan demikian, semakin besar nilai eksponen, semakin banyak operasi yang perlu dieksekusi.

1. Analisis menggunakan pendekatan best-case (kasus terbaik), worst-case (kasus terburuk), dan average-case (kasus rata-rata)

* **Best-Case (Kasus Terbaik)**

Kasus terbaik terjadi ketika eksponen adalah 0. Dalam kasus ini, algoritma langsung mengembalikan nilai 1 tanpa melakukan iterasi apa pun. Oleh karena itu, kompleksitas waktu adalah O(1), karena waktu yang dibutuhkan tidak tergantung pada ukuran eksponen.

* **Worst-Case (Kasus Terburuk)**

Kasus terburuk terjadi ketika eksponen memiliki panjang biner yang besar. Dalam situasi ini, algoritma membutuhkan banyak iterasi dalam loop while, sebanyak panjang biner dari eksponen. Sebagai hasilnya, kompleksitas waktu menjadi O(n), di mana n adalah panjang biner eksponen.

* **Average-Case (Kasus Rata-rata)**

Untuk kasus rata-rata, diasumsikan bahwa setiap bit dalam eksponen memiliki kemungkinan yang sama untuk menjadi 0 atau 1. Oleh karena itu, kompleksitas waktu rata-rata juga sekitar O(n), mengikuti kesamaan dengan kasus terburuk. Dalam kasus ini, kompleksitas waktu bergantung pada panjang biner dari eksponen.

1. **Referensi**

https://epgp.inflibnet.ac.in/epgpdata/uploads/epgp\_content/S000007CS/P001064/M014962/ET/1459250173ET19.pdf

https://www.geeksforgeeks.org/binary-exponentiation-for-competitive-programming/

http://www.csc.villanova.edu/~mdamian/csc8301/notes/Lec8Notes.pdf

https://ebooks.inflibnet.ac.in/csp5/chapter/transform-and-conquer-design-paradigm/

https://cp-algorithms.com/algebra/binary-exp.html

http://www.csc.villanova.edu/~mdamian/Past/csc8301fa16/notes/Lec8.pdf

http://www.math.uaa.alaska.edu/~afkjm/cs351/handouts/transform-conquer.pdf

https://www.geeksforgeeks.org/transform-and-conquer-technique/

1. **Link Github**

https://github.com/Fadhil-22343045/Perancangan-dan-Analisis-Algoritma