Laporan Tugas Kelompok

Analisa Kompleksitas Waktu Algoritma *Boyer-Moore Horspool* pada Aplikasi Pencatatan Deadline Tugas Berbasis *Graphical User Interface*

Dibuat untuk memenuhi tugas akhir mata kuliah Desain dan Analisis Algoritma

Dosen Pengampu:
I Made Widiartha, S.Si., M.Kom

Disusun oleh:
Kelompok I

I Gede Widnyana	**(2208561016)**
Ni Made Viona Rara Santhi	**(2208561098)**

Kelas: C

PROGRAM STUDI INFORMATIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS UDAYANA

JIMBARAN

2023

A. Kompleksitas Waktu Algoritma *Boyer-Moore Horspool*

Berikut ini adalah perhitungan algoritma *Boyer-Moore Horspool* yang diterapkan dalam aplikasi pencatatan deadline tugas.

No	Source Code Algoritma *Boyer-Moore Horspool*
1	`def boyer_moore_horspool_search(pattern, text):`
2	`    m = len(pattern)  # O(1)`
3	`    n = len(text)     # O(1)`
4	`    pattern = pattern.lower()  # O(m)`
5	`    text = text.lower()        # O(n)`
6	`    last_occurrence = {pattern[i]: i for i in range(m)}  # O(m)`
7	
8	`    i = m - 1  # O(1)`
9	`    j = m - 1  # O(1)`
10	
11	`    while j < n:  # O(n) * (O(m) + O(1)) -> O(n*m)`
12	`        if pattern[i] == text[j]:  # O(1)`
13	`            if i == 0:  # O(1)`
14	`                return j  # O(1)`
15	`            else:`
16	`                i -= 1  # O(1)`
17	`                j -= 1  # O(1)`
18	`        else:`
19	`            last_occ = last_occurrence.get(text[j], -1)  # O(1)`
20	`            j = j + m - min(i, 1 + last_occ)  # O(1)`
21	`            i = m - 1  # O(1)`
22	
23	`    return None  # O(1)`
24	
25	`def search_phrase_in_text(phrase, text):`
26	`    return phrase.lower() in text.lower()  # O(m + n)`
27	
28	`def boyer_moore_horspool_search_name(name, data_array):`
29	`    name = name.lower()  # O(m)`
30	
31	`    for i, item in enumerate(data_array):  # O(k) * (O(m + n) + O(1))`
32	`        keterangan_lower = item["keterangan"].lower()  # O(n)`
33	`        if search_phrase_in_text(name, keterangan_lower):  # O(m + n)`
34	`            return i  # O(1)`
35	
36	`    return None  # O(1)`

Berikut ini adalah penjelasan detail analisis kompleksitas waktu algoritma *Boyer-Moore Horspool.*

1. **Fungsi boyer_moore_horspool_search**

Baris Kode	Kompleksitas	Penjelasan
m = len(pattern)	O(1)	Menghitung panjang string pola adalah operasi konstan.
n = len(text)	O(1)	Menghitung panjang string dari teks adalah operasi konstan.
pattern = pattern.lower()	O(m)	Mengubah ke huruf kecil dilakukan untuk setiap karakter dalam **pattern**.
text = text.lower()	O(n)	Mengubah ke huruf kecil dilakukan untuk setiap karakter dalam **text.**
last_occurrence = {pattern[i]: i for i in range(m)}	O(m)	Dictionary comprehension ini berjalan sebanyak **m** kali.
i = m - 1	O(1)	Operasi aritmatika (pengurangan) sederhana.
j = m - 1	O(1)	Operasi aritmatika (pengurangan) sederhana.
while j < n:	O(n*m)	Loop ini dapat berjalan hingga **n** kali. Di dalamnya, terdapat operasi yang berjalan hingga **m** kali.
if pattern[i] == text[j]:	O(1)	Perbandingan karakter adalah operasi konstan.
if i == 0:	O(1)	Perbandingan sederhana.
i -= 1	O(1)	Operasi aritmatika (pengurangan) sederhana.
j -= 1	O(1)	Operasi aritmatika (pengurangan) sederhana.
last_occ = last_occurrence.get(text[j], -1)	O(1)	Akses dictionary adalah operasi konstan.
j = j + m - min(i, 1 + last_occ)	O(1)	Beberapa operasi aritmatika sederhana.
i = m - 1	O(1)	Operasi aritmatika (pengurangan) sederhana.
return None	O(1)	Mengembalikan nilai adalah operasi konstan.

Kompleksitas Waktu: 12 x O(1) + O(n*m) + O(m) + O (m) + O (n) = O(m*n) + O(2m) + O(n) + O(m*n).

Sehingga notasi *Big-O* adalah **O(mn).**

Kompleksitas ini disebabkan oleh loop while, yang berjalan sebanyak n kali, dan dalam setiap iterasi, ada operasi yang tergantung pada m.

2. **Fungsi search_phrase_in_text**

Baris Kode	Kompleksitas	Penjelasan
return phrase.lower() in text.lower()	O(m + n)	Mengubah kedua string menjadi lowercase membutuhkan O(m) dan O(n), dan pengecekan **in** juga memerlukan hingga O(n).

Kompleksitas Waktu: O(m + n)

Sehingga notasi *Big-O* adalah **O(m+n)**.

Pengkonversian ke lowercase membutuhkan O(m) dan O(n), dan operasi in juga memerlukan O(n).

3. **Fungsi boyer_moore_horspool_search_name**

Baris Kode	Kompleksitas	Penjelasan
name = name.lower()	O(m)	Mengubah **name** menjadi lowercase, memerlukan O(m).
for i, item in enumerate(data_array):	O(k * (m + n))	Loop berjalan **k** kali (jumlah elemen dalam **data_array**). Setiap iterasi melibatkan operasi O(m + n).
keterangan_lower = item["keterangan"].lower()	O(n)	Mengubah **keterangan** menjadi lowercase, memerlukan O(n).
if search_phrase_in_text(name, keterangan_lower):	O(m + n)	Memanggil fungsi **search_phrase_in_text** yang memiliki kompleksitas O(m + n).
return i	O(1)	Mengembalikan nilai adalah operasi konstan.
return None	O(1)	Mengembalikan nilai adalah operasi konstan.

Kompleksitas Waktu: O(m) + O(k * (m + n)) + O(n) + O(m + n) + O(2)

Notasi *Big O* adalah **O (k * (m + n))**. Iterasi pada data_array dilakukan k kali (ukuran data_array). Dalam setiap iterasi, operasi pencarian membutuhkan O(m + n).

Kesimpulan :

Kompleksitas waktu terbesar dari algoritma ini adalah O(n*m) untuk fungsi boyer_moore_horspool_search, di mana n adalah panjang teks dan m adalah panjang pola. Untuk fungsi boyer_moore_horspool_search_name, kompleksitasnya adalah O(k * (m + n)), di mana k adalah jumlah elemen dalam data_array.

B. Kompleksitas Waktu Berdasarkan Referensi

Berdasarkan observasi penulis dalam mengkaji kompleksitas waktu dari sumber website https://www.geeksforgeeks.org/boyer-moore-algorithm-for-pattern-searching/. Hasil analisa

kompleksitas waktu dari algoritma boyer-moore horspool berdasarkan sumber itu adalah sebagai berikut.

Kompleksitas Waktu : O(n x m)

Ruang Tambahan : O(1)

Heuristik Karakter Buruk mungkin membutuhkan O(mn) waktu dalam kasus terburuk. Kasus terburuk terjadi ketika semua karakter teks dan pola sama. Misalnya, txt[] = "AAAAAAAAAAAAAAAAAAAA" dan pat[] = "AAAAA". *The Bad Character Heuristic* dapat mengambil O (n / m) dalam kasus terbaik. Kasus terbaik terjadi ketika semua karakter teks dan pola berbeda.

Time Complexity : O(n x m)

Auxiliary Space: O(1)

The Bad Character Heuristic may take $O(mn)$ time in worst case. The worst case occurs when all characters of the text and pattern are same. For example, txt[] = “AAAAAAAAAAAAAAAAA” and pat[] = “AAAAA”. The Bad Character Heuristic may take O(n/m) in the best case. The best case occurs when all the characters of the text and pattern are different.

Sumber : Boyer Moore Algorithm for Pattern Searching - GeeksforGeeks

DAFTAR PUSTAKA

GeeksforGeeks. 2012. *Boyer Moore Algorithm for Pattern Searching*. URL: https://www.geeksforgeeks.org/boyer-moore-algorithm-for-pattern-searching/. Diakses tanggal 27 Desember 2023.