

**Tugas Mandiri**  
**Perancangan dan Analisis Algoritma**



**DOSEN PENGAMPU:**

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

**OLEH:**

**Faza Azka Mahasya**

**23343038**

**PROGRAM STUDI PENDIDIKAN TEKNIK INFORMATIKA**

**DEPARTEMEN TEKNIK ELEKTRONIKA**

**FAKULTAS TEKNIK**

**UNIVERSITAS NEGERI PADANG**

**2024**

# Algoritma Boyer-Moore

## 1. Ringkasan Algoritma

Algoritma Boyer-Moore adalah salah satu algoritma pencarian string yang efisien. Algoritma ini dikembangkan oleh Robert S. Boyer dan J Strother Moore pada tahun 1977. Algoritma ini dikenal karena kemampuannya untuk melewati bagian-bagian teks yang tidak perlu diperiksa, sehingga mengurangi jumlah perbandingan yang dilakukan. Hal ini membuat algoritma Boyer-Moore sangat cepat, terutama ketika teks yang dicari memiliki panjang yang cukup besar.

Algoritma Boyer-Moore bekerja dengan membandingkan karakter dari pola (pattern) yang dicari dengan teks (text) dari kanan ke kiri. Jika terjadi ketidakcocokan, algoritma akan menggunakan dua aturan untuk menentukan seberapa jauh pola dapat digeser: Aturan **Bad Character** dan Aturan **Good Suffix**.

- **Aturan Bad Character** memanfaatkan informasi tentang karakter yang tidak cocok dalam pola untuk menentukan pergeseran yang optimal. Jika suatu karakter dalam pola tidak ditemukan di posisi yang bersesuaian dalam teks, maka pola dapat langsung digeser melewati karakter tersebut.
- **Aturan Good Suffix** menggunakan informasi tentang sufiks yang cocok dalam pola. Jika bagian akhir pola cocok dengan teks tetapi terjadi ketidaksesuaian pada karakter sebelumnya, pola dapat digeser ke posisi di mana bagian belakang pola masih relevan.

Karena kemampuannya untuk melewati banyak karakter yang tidak perlu diperiksa, algoritma Boyer-Moore sering kali lebih cepat daripada algoritma pencarian string lainnya seperti Knuth-Morris-Pratt atau algoritma brute force. Dalam banyak kasus, algoritma ini memiliki kompleksitas rata-rata  $O(n/m)$ , di mana  $n$  adalah panjang teks dan  $m$  adalah panjang pola, yang menjadikannya salah satu metode pencarian teks paling efisien yang tersedia.

Selain digunakan dalam pencarian teks biasa, algoritma ini juga diterapkan dalam berbagai bidang seperti **analisis DNA dalam bioinformatika**, **deteksi plagiarisme dalam dokumen teks**, serta **pencarian data dalam basis data besar**. Dengan

kemampuannya yang efisien, Boyer-Moore tetap menjadi salah satu algoritma standar dalam pemrosesan teks modern.

## 2. Pseudocode Algoritma Boyer-Moore

```
FUNCTION buat_tabel_karakter_buruk(pola):  
    ukuran = panjang(pola)  
  
    tabel = {} // Inisialisasi tabel kosong  
  
    FOR setiap karakter dalam pola:  
        tabel[karakter] = ukuran // Set nilai default untuk setiap karakter  
  
    FOR i dari 0 hingga ukuran - 2:  
        tabel[pola[i]] = ukuran - 1 - i // Update nilai untuk karakter yang ada dalam pola  
  
    RETURN tabel  
  
FUNCTION boyer_moore(teks, pola):  
    n = panjang(teks)  
    m = panjang(pola)  
  
    tabel_buruk = buat_tabel_karakter_buruk(pola) // Bangun tabel bad character heuristic  
  
    geser = 0 // Inisialisasi geser  
  
    WHILE geser <= n - m:  
        j = m - 1 // Mulai dari akhir pola  
  
        WHILE j >= 0 DAN pola[j] == teks[geser + j]:  
            j = j - 1 // Geser ke kiri jika karakter cocok  
  
        IF j < 0:
```

```

RETURN geser // Pola ditemukan, kembalikan indeks awal

ELSE:

    geser = geser + tabel_buruk.get(teks[geser + m - 1], m)
// Geser berdasarkan tabel bad character

RETURN -1 // Pola tidak ditemukan

// Contoh Penggunaan

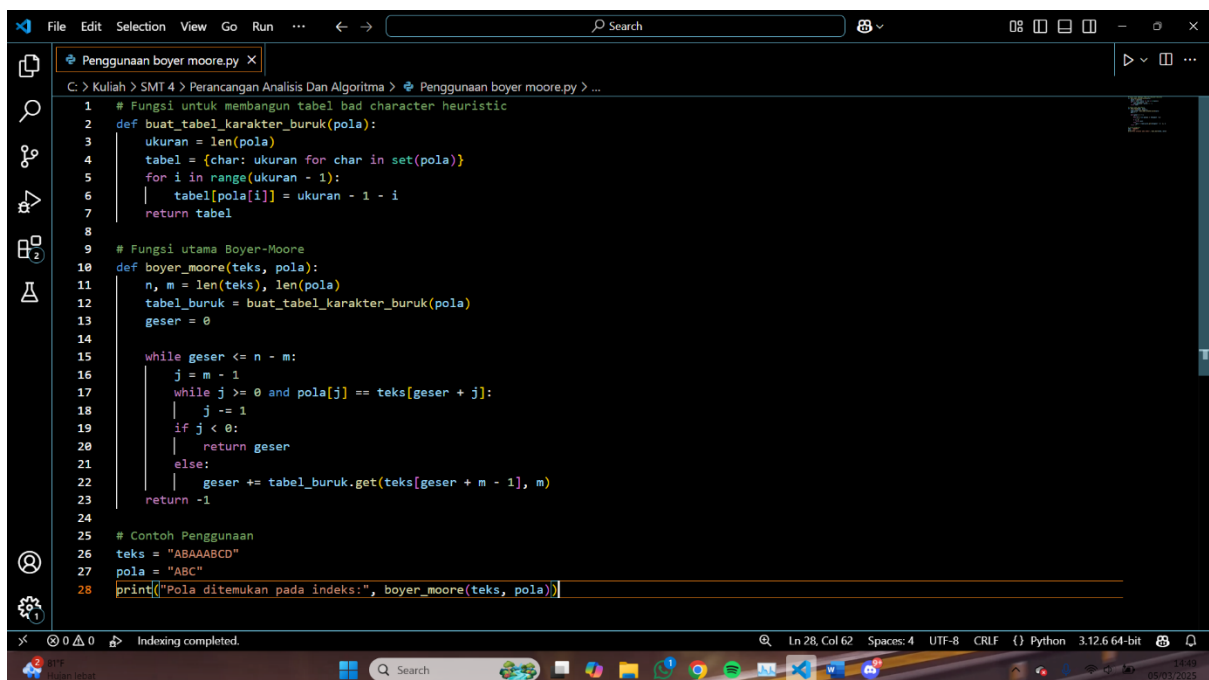
teks = "ABAAABCD"

pola = "ABC"

CETAK "Pola ditemukan pada indeks:", boyer_moore(teks, pola)

```

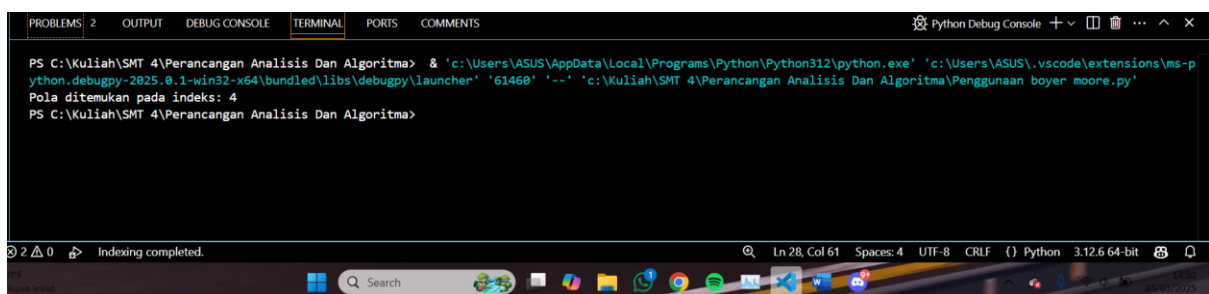
### 3. Implementasi Algoritma Boyer-Moore dalam program python



```

1 # Fungsi untuk membangun tabel bad character heuristic
2 def buat_tabel_karakter_buruk(pola):
3     ukuran = len(pola)
4     tabel = {char: ukuran for char in set(pola)}
5     for i in range(ukuran - 1):
6         tabel[pola[i]] = ukuran - 1 - i
7     return tabel
8
9 # Fungsi utama Boyer-Moore
10 def boyer_moore(teks, pola):
11     n, m = len(teks), len(pola)
12     tabel_buruk = buat_tabel_karakter_buruk(pola)
13     geser = 0
14
15     while geser <= n - m:
16         j = m - 1
17         while j >= 0 and pola[j] == teks[geser + j]:
18             j -= 1
19         if j < 0:
20             return geser
21         else:
22             geser += tabel_buruk.get(teks[geser + m - 1], m)
23     return -1
24
25 # Contoh Penggunaan
26 teks = "ABAAABCD"
27 pola = "ABC"
28 print("Pola ditemukan pada indeks:", boyer_moore(teks, pola))

```



```

PS C:\Kuliah\SMT 4\Perancangan Analisis Dan Algoritma> & 'c:\Users\ASUS\AppData\Local\Programs\Python\Python312\python.exe' 'c:\Users\ASUS\.vscode\extensions\ms-python.debugpy-2025.0.1-win32-x64\bundle\libs\debugpy\launcher' '61468' '--' 'c:\Kuliah\SMT 4\Perancangan Analisis Dan Algoritma\Penggunaan boyer moore.py'
Pola ditemukan pada indeks: 4
PS C:\Kuliah\SMT 4\Perancangan Analisis Dan Algoritma>

```

## 4. Analisis Kebutuhan Waktu

### 1) Analisis Menyeluruh

Total kebutuhan waktu algoritma Boyer-Moore dihitung dengan mempertimbangkan operasi yang dilakukan dalam setiap tahap eksekusi:

- Inisialisasi tabel karakter buruk: Membutuhkan  $O(m)$  waktu untuk membangun tabel karakter buruk.
- Inisialisasi tabel good suffix: Membutuhkan  $O(m)$  waktu untuk membangun tabel good suffix.
- Loop utama pencocokan: Dalam kasus terbaik, pola langsung ditemukan dan hanya membutuhkan  $O(1)$ . Dalam kasus rata-rata dan terburuk, algoritma dapat berjalan hingga  $O(n/m)$  kali pergeseran pola.

Total waktu eksekusi terbaik adalah  $O(n/m)$  dan terburuk adalah  $O(nm)$ .

### 2) Analisis Berdasarkan Operasi Abstrak

- Perbandingan karakter:  $O(m)$  dalam iterasi pencocokan per pergeseran.
- Pergantian posisi pencocokan: Bergantung pada heuristik, tetapi rata-rata  $O(n/m)$ .
- Total iterasi:  $O(n)$  dalam kasus terburuk jika semua karakter diuji.

Total kompleksitas tetap berkisar antara  $O(n/m)$  hingga  $O(nm)$  tergantung pada kasus spesifik.

### 3) Best-Case, Worst-Case, dan Average-Case

- Best-Case (Kasus Terbaik): Jika pola tidak ada dalam teks atau ditemukan dengan sedikit perbandingan, waktu eksekusi adalah  $O(n/m)$ .
- Average-Case (Kasus Rata-Rata): Jika pola ditemukan setelah beberapa pergeseran, tetapi tidak selalu langsung cocok, waktu eksekusi mendekati  $O(n/m)$ .
- Worst-Case (Kasus Terburuk): Jika pola memiliki struktur yang buruk dibandingkan dengan teks, waktu eksekusi bisa mencapai  $O(nm)$ .

## 5. Referensi

Anany Levitin, *Introduction to the Design & Analysis of Algorithms*, 3rd Edition.

## 6. Lampiran Link Github