**Tugas Mandiri**

**Algoritma boyer moore**

**Perancangan Dan Analisis Algotirma**



**Dosen Pengampu:**

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

**Oleh:**

**Ghenta Alif Alde**

**21343048**

**INFORMATIKA**

**JURUSAN TEKNIK ELEKTRONIKA FAKULTAS TEKNIK**

**UNIVERSITAS NEGERI PADANG**

**2023**

**Algoritma boyer moore**

1. **Penjelasan Algoritma Boyer Moore**

Algoritma Boyer-Moore yaitu salah satu algoritma pencarian string, dipublikasikan oleh Robert S. Boyer, dan J. Strother Moore pada tahun 1977.

Algoritme ini dianggap sebagai algoritma yang paling efisien pada aplikasi umum. Tidak seperti algoritma pencarian string yang ditemukan sebelumnya, algoritme Boyer-Moore mulai mencocokkan karakter dari sebelah kanan pattern. Ide di balik algoritma ini adalah bahwa dengan memulai pencocokan karakter dari kanan, dan bukan dari kiri, maka akan lebih banyak informasi yang didapat. Pattern merupakan variabel yang akan di cari pada suatu data. Dari data tersebut maka akan di analisis mulai dari dihitung jumlah, cari, dan di cocokan sesuai dengan pattern yang dimasukkan.

**Cara kerja algoritma boyer moore**

Dari sebuah pencarian dan pencocokan kata pada sebuah teks maka akan di mulai dengan langkah-langkah boyer moore sebagai berikut:

1. Algoritma Boyer-Moore mulai mencocokkan pattern pada awal teks
2. Dari kanan ke kiri, algoritma ini akan mencocokkan karakter per karakter pattern dengan karakter di teks yang bersesuaian, sampai salah satu kondisi berikut, yaitu:pertama karakter di pattern dan di teks yang dibandingkan tidak cocok (mismatch). Kedua semua karakter di pattern cocok. Kemudian algoritme akan memberitahukan penemuan di posisi ini.
3. Algoritme kemudian menggeser pattern dengan memaksimalkan nilai penggeseran good-suffix dan penggeseran bad-character, lalu mengulangi langkah 2 sampai pattern berada di ujung teks

**Implementasi algoritma boyer moore**

1. Pencarian dokumen
2. Aplikasi mesin pencari atau search engine
3. Aplikasi pencarian nama
4. Filtering kata pada sebuah form

**Kelebihan dan kelemahan algoritma boyer moore**

**Kelebihan**

Tidak seperti pencarian string lainnya Brute Force, Knuth-Morris-Pratt yang mempunyai cara kerja membandingkan satu – persatu karakter dari kiri ke kanan. Boyer-Moore membandingkan karakter dari kanan ke kiri dan memiliki loncatan karakter yang besarsehingga mempercepat pencarian string karena dengan hanya memeriksa sedikit karakter, dapat langsung diketahui bahwa string yang dicari tidak ditemukan dan dapat digeser ke posisi berikutnya.

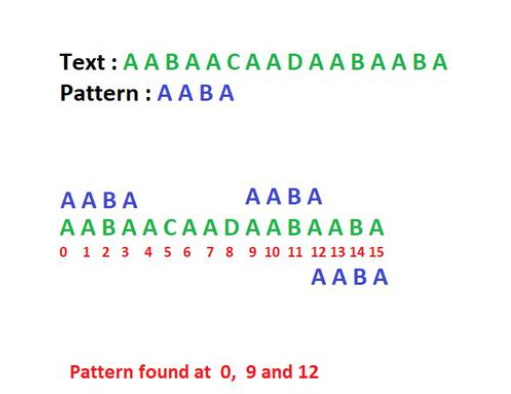
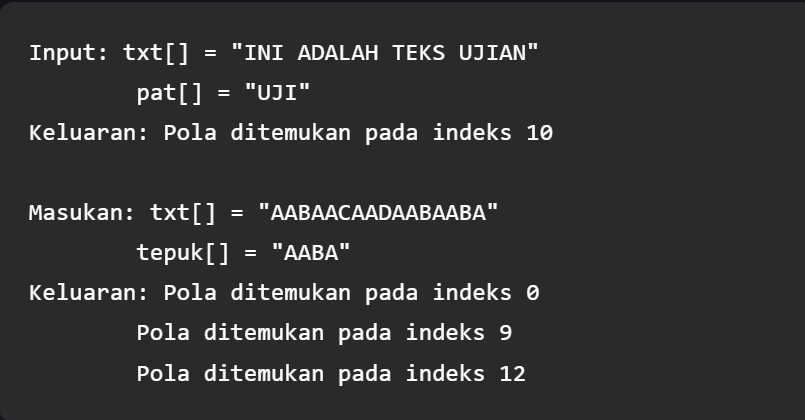
**Kelemahan**

Algoritma Boyer-Moore mencocokan Pattern dari kanan ke kiri oleh sebab itu  
kelemahan dari algoritma ini adalah ketika semua karakter memiliki kesamaan atau cocok dan hanya karakter terakhir atau karakter paling kiri yang berbeda maka pencarian ini akan memerlukan waktu yang sedikit lama (Utomo, 2008).

Pencarian pola merupakan masalah penting dalam ilmu komputer. Saat kita melakukan pencarian string di file notepad/word, browser, atau database, algoritma pencarian pola digunakan untuk menampilkan hasil pencarian. Pernyataan masalah tipikal adalah-

Diberikan teks txt[0..n-1] dan pola pat[0..m-1] di mana n adalah panjang teks dan m adalah panjang pola, tulis a fungsi pencarian(char pat[], char txt[]) yang mencetak semua kejadian pat[] di txt[]. Anda dapat berasumsi bahwa n > m.

Contoh:



Akan membahas algoritma pencarian pola Boyer Moore. Seperti algoritma KMP dan Finite Automata ,algoritma Boyer Moore juga mempraproses pola.

Boyer Moore adalah kombinasi dari dua pendekatan berikut.

1. Heuristik Karakter Buruk
2. Akhiran Heuristik yang Baik

Kedua heuristik di atas juga dapat digunakan secara mandiri untuk mencari pola dalam sebuah teks. Mari kita pahami dulu bagaimana dua pendekatan independen bekerja sama dalam algoritma Boyer Moore. Jika kita melihat algoritma Naif , itu menggeser pola di atas teks satu per satu. Algoritma KMP melakukan preprocessing terhadap pola sehingga pola dapat digeser lebih dari satu. Algoritma Boyer Moore melakukan preprocessing untuk alasan yang sama. Ini memproses pola dan membuat array yang berbeda untuk masing-masing dari dua heuristik. Pada setiap langkah, ia menggeser pola dengan maksimum slide yang disarankan oleh masing-masing dari dua heuristik. Jadi menggunakan offset terbesar yang disarankan oleh dua heuristik di setiap langkah .

Berbeda dengan algoritma pencarian pola sebelumnya, algoritma Boyer Moore memulai pencocokan dari karakter terakhir dari pola tersebut.

Pada postingan kali ini kita akan membahas heuristik bad character dan good suffix heuristic pada postingan selanjutnya.

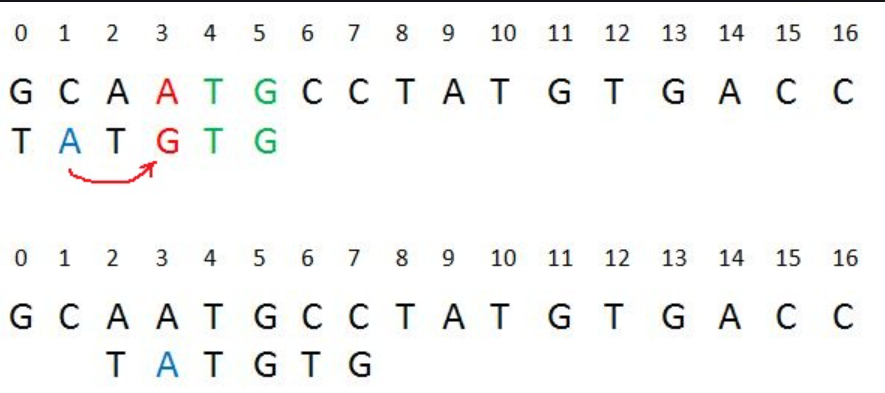
Heuristik Karakter Buruk

Gagasan heuristik karakter buruk itu sederhana. Karakter teks yang tidak cocok dengan karakter pola saat ini disebut Bad Character . Setelah ketidakcocokan, kami menggeser pola sampai

1. Ketidak cocokan menjadi pertandingan.
2. Pola P bergerak melewati karakter yang tidak cocok.

Kasus 1 – Ketidaksesuaian menjadi cocok

Kita akan mencari posisi kejadian terakhir dari karakter yang tidak cocok dalam pola, dan jika karakter yang tidak cocok ada di dalam pola, maka kita akan menggeser pola tersebut sehingga menjadi sejajar dengan karakter yang tidak cocok di teks T .

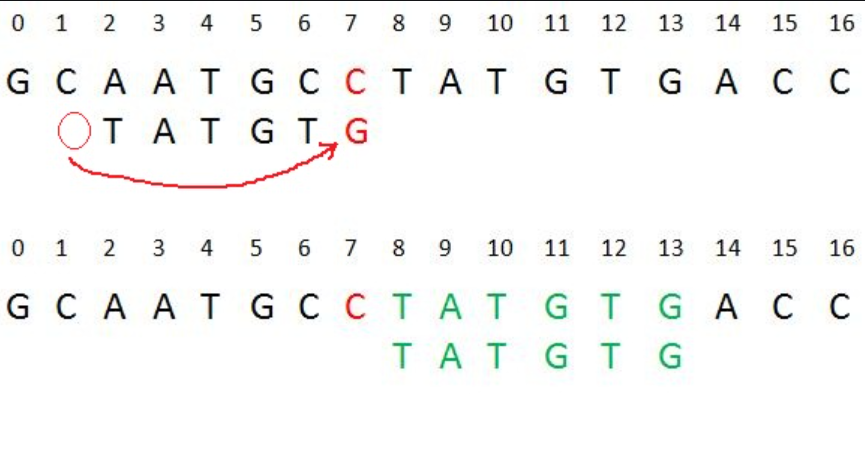


Penjelasan:

Pada contoh di atas, kita mendapatkan mismatch pada posisi 3. Disini karakter mismatch kita adalah “A”. Sekarang kita akan mencari kejadian terakhir dari "A" dalam pola. Kami mendapat "A" di posisi 1 dalam pola (ditampilkan dalam warna Biru) dan ini adalah kejadian terakhirnya. Sekarang kita akan menggeser pola 2 kali sehingga "A" dalam pola sejajar dengan "A" dalam teks.

Kasus 2 – Pola bergerak melewati karakter yang tidak cocok

Kita akan mencari posisi kemunculan terakhir karakter yang tidak cocok dalam pola dan jika karakter tidak ada kita akan menggeser pola melewati karakter yang tidak cocok.



Penjelasan:

Di sini kita memiliki ketidaksesuaian pada posisi 7. Ketidakcocokan karakter "C" tidak ada pada pola sebelum posisi 7 jadi kita akan menggeser pola melewati ke posisi 7 dan akhirnya pada contoh di atas kita mendapatkan pola yang cocok (ditampilkan di Hijau). Kami melakukan ini karena "C" tidak ada dalam pola sehingga pada setiap shift sebelum posisi 7 kami akan mendapatkan ketidakcocokan dan pencarian kami akan sia-sia.

Dalam implementasi berikut, kami memproses pola dan menyimpan kejadian terakhir dari setiap karakter yang mungkin dalam array berukuran sama dengan ukuran alfabet. Jika karakter tidak ada sama sekali, maka dapat mengakibatkan pergeseran sebesar m (panjang pola). Oleh karena itu, heuristik karakter buruk membutuhkan O(t/m) waktu dalam kasus terbaik.

1. **Program algoritma boyer moore**

# Program Python3 untuk Heuristik Karakter Buruk

# dari Algoritma Pencocokan String Boyer Moore

NO\_OF\_CHARS = 256

def badCharHeuristic(string, size):

'''

Fungsi pra-pemrosesan untuk

heuristik karakter buruk Boyer Moore

'''

# Inisialisasi semua kemunculan menjadi -1

badChar = [-1]\*NO\_OF\_CHARS

# Isi nilai aktual kemunculan terakhir

for i in range(size):

badChar[ord(string[i])] = i;

# kembalikan daftar yang diinisialisasi

return badChar

def search(txt, pat):

'''

Fungsi pencarian pola yang menggunakan

heuristik karakter buruk Algoritma Boyer Moore

'''

m = len(pat)

n = len(txt)

# Buat daftar karakter buruk dengan memanggil

# fungsi pra-pemrosesan badCharHeuristic()

# untuk pola yang diberikan

badChar = badCharHeuristic(pat, m)

# s adalah pergeseran pola terhadap teks

s = 0

while(s <= n-m):

j = m-1

# Terus kurangi indeks j dari pola saat

# karakter pola dan teks cocok

# pada pergeseran s ini

while j>=0 and pat[j] == txt[s+j]:

j -= 1

# Jika pola ada pada pergeseran saat ini,

# maka indeks j akan menjadi -1 setelah loop di atas

if j<0:

print("Pola muncul pada pergeseran = {}".format(s))

'''

Geser pola sehingga karakter berikutnya dalam teks

sejajar dengan kemunculan terakhirnya dalam pola.

Kondisi s+m < n diperlukan untuk kasus ketika

pola muncul di akhir teks

'''

s += (m-badChar[ord(txt[s+m])] if s+m<n else 1)

else:

'''

Geser pola sehingga karakter buruk dalam teks

sejajar dengan kemunculan terakhirnya dalam pola. Fungsi max

digunakan untuk memastikan bahwa kita mendapatkan pergeseran positif.

Kita mungkin mendapatkan pergeseran negatif jika kemunculan terakhir

karakter buruk dalam pola ada di sebelah kanan karakter saat ini.

'''

s += max(1, j-badChar[ord(txt[s+j])])

# Program utama untuk menguji fungsi di atas

def main():

txt = "ABAAABCD"

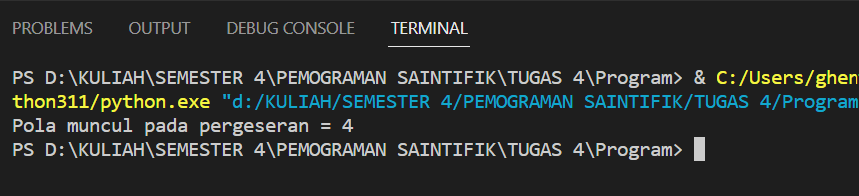
pat = "ABC"

search(txt, pat)

if \_\_name\_\_ == '\_\_main\_\_':

main()

**Hasil run;**



**penjelasan dari code program**

Fungsi badCharHeuristic digunakan untuk membangun tabel heuristik karakter buruk. Fungsi ini mengambil sebuah string dan ukurannya sebagai parameter. Fungsi ini kemudian menginisialisasi daftar badChar dengan semua nilai kemunculan menjadi -1, kemudian mengisi nilai aktual kemunculan terakhir dari setiap karakter di dalam string. Fungsi mengembalikan daftar badChar.

Fungsi search digunakan untuk mencari kemunculan pola dalam teks menggunakan heuristik karakter buruk. Fungsi ini mengambil teks dan pola sebagai parameter. Fungsi ini memanggil fungsi badCharHeuristic untuk membangun daftar karakter buruk untuk pola yang diberikan. Kemudian fungsi mengiterasi seluruh karakter dalam teks menggunakan pergeseran s dan membandingkan setiap karakter dari pola dengan karakter dalam teks. Jika pola cocok, maka indeks j dikurangi hingga pola dan teks tidak cocok. Jika indeks j menjadi -1, maka pola muncul pada pergeseran saat ini. Kemudian pergeseran s ditingkatkan sehingga karakter berikutnya dalam teks sejajar dengan kemunculan terakhir pola. Jika pola dan teks tidak cocok, maka fungsi menggeser pola sehingga karakter buruk dalam teks sejajar dengan kemunculan terakhirnya dalam pola.

Program utama hanya menguji fungsi search menggunakan teks "ABAAABCD" dan pola "ABC". Program ini mencetak pergeseran s di mana pola ditemukan dalam teks. Output yang diharapkan adalah "Pola muncul pada pergeseran = 4.

1. **Pseudocode algoritma boyer moore**

badCharHeuristic(string, size):

badChar <- list berisi -1 sebanyak 256 karakter ASCII

for i in range(size):

badChar[ord(string[i])] <- i

return badChar

search(txt, pat):

m <- len(pat)

n <- len(txt)

badChar <- badCharHeuristic(pat, m)

s <- 0

while s <= n-m:

j <- m-1

while j>=0 and pat[j] == txt[s+j]:

j -= 1

if j<0:

print("Pola muncul pada pergeseran = {}".format(s))

s += (m-badChar[ord(txt[s+m])] if s+m<n else 1)

else:

s += max(1, j-badChar[ord(txt[s+j])])

main():

txt <- "ABAAABCD"

pat <- "ABC"

search(txt, pat)

1. **Analisis kebutuhan waktu algoritma boyer moore**

Waktu yang dibutuhkan oleh algoritma Boyer-Moore untuk mencari sebuah pola pada sebuah teks tergantung pada panjang pola dan panjang teks. Algoritma Boyer-Moore memerlukan waktu linier O(n/m) dalam kasus terburuknya, di mana n adalah panjang teks dan m adalah panjang pola yang dicari. Namun, algoritma ini dapat mencapai kinerja yang jauh lebih cepat dalam beberapa kasus tergantung pada pola dan teks yang diberikan.

Heuristik karakter buruk yang digunakan oleh algoritma Boyer-Moore dapat mempercepat waktu pencarian karena dapat meminimalkan jumlah pergeseran. Pada kasus terburuk, di mana pola tidak cocok dengan teks, algoritma ini hanya memerlukan satu kali pemindahan pada setiap karakter di teks. Oleh karena itu, waktu yang dibutuhkan oleh algoritma Boyer-Moore untuk mencari sebuah pola pada sebuah teks berkisar dari O(n/m) hingga O(n).

1. **Referensi**

* <https://www.sistemphp.com/analisis-algoritma-boyer-moore/>
* <http://p2k.unkris.ac.id/id3/1-3065-2962/Algoritma-Boyer-Moore_27387_p2k-unkris.html>
* <https://www.sistemphp.com/analisis-algoritma-boyer-moore/>
* <https://www-geeksforgeeks-org.translate.goog/boyer-moore-algorithm-for-pattern-searching/?_x_tr_sl=en&_x_tr_tl=id&_x_tr_hl=id&_x_tr_pto=tc>

1. **Lampiran link github**

https://github.com/21343048/Tugas1-Ghenta-Alif-Alde\_Algoritma-boyer-moore