# Penerapan Algoritma Boyer Moore Pada Game Word Search Puzzles

Muhammad Fahkry Malta and 13519032
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13519032@std.stei.itb.ac.id:

***Abstrak* — Word Search Puzzles merupakan game yang populer dan dimainkan oleh sebagian besar masyarakat di seluruh dunia. Aturan game ini secara sederhana adalah menemukan kata dari huruf acak yang berbentuk kotak/tabel/matriks dengan asumsi kotak berbentuk persegi. Game ini dapat dipecahkan dengan algoritma string matching, salah satunya adalah algoritma Boyer-Moore.**

***Kata Kunci*—String Matching, Boyer Moore, Word Search Puzzles, Pemecahan, Pengujian, Kompleksitas, Efisiensi**

## I. Pendahuluan

Game sudah menjadi sangat populer pada zaman millenial ini, banyak sekali game-game yang beredar dan dimainkan secara masif oleh seluruh masyarakat dunia. Game yang sudah sangat terkenal ialah game berbentuk puzzle, game puzzle sudah sangat populer sejak dulu sebagai game untuk mengasah otak.

Game puzzle juga terdiri dari bermacam-macam jenis, salah satu jenisnya adalah Word Search Puzzles yang memanfaatkan huruf-huruf acak. Pemain diminta menemukan kata-kata yang memiliki arti atau kata-kata yang diminta didalam susunan huruf-huruf acak tersebut. Kata tersebut bisa didapatkan melalui arah horizontal, vertikal, maupun diagonal.

Algoritma Boyer-Moore merupakan salah satu Algoritma String Matching yang sudah sangat terkenal dan juga dikenal sangat efisien sehingga sangat dianjurkan menggunakan algoritma ini dalam melakukan string matching. Salah satu pengaplikasiannya adalah melalui pemecahan game Word Search Puzzles.

## II. Landasan Teori

### 2.1 Algoritma String Matching

Algoritma pencarian string atau sering disebut juga algoritma pencocokan string yaitu algoritma untuk melakukan pencarian semua kemunculan string pendek dan panjang, untuk string pendek yang disebut pattern dan string yang lebih panjang yang disebut teks.

1. string pendek (pattern) = pattern[0 …. m -1] yaitu string dengan panjang karakter (asumsi m <<< n) yang akan dicari di dalam teks.
2. string panjang (teks) = teks[0 …. n - 1] yaitu (long) string yang panjangnya n karakter

Algoritma pencarian string ini dapat juga diklasifikasikan menjadi 3 bagian menurut arah pencariannya , berikut ini adalah algoritma yang termasuk dalam algoritma ini.

- Dari arah yang paling alami yaitu dari kiri ke kanan, yang merupakan arah untuk membaca, algoritma yang termasuk kategori ini adalah:
    1. Algoritma Brute Force
    2. Algoritma dari Morris dan Pratt, yang kemudian dikembangkan oleh Knuth, Morris, dan Pratt
- Kategori kedua yaitu dari arah kanan ke kiri, arah yang biasanya menghasilkan hasil terbaik secara praktikal, contohnya adalah:
    1. Algoritma dari Boyer dan Moore, yang kemudian banyak dikembangkan, menjadi Algoritma turbo Boyer-Moore, Algoritma tuned Boyer-Moore, dan Algoritma Zhu-Takaoka;
- Dan kategori terakhir yaitu adalah dari arah yang ditentukan secara spesifik oleh algoritma tersebut, arah ini menghasilkan hasil terbaik secara teoritis, algoritma yang termasuk kategori ini adalah:
    1. Algoritma Colussi
    2. Algoritma Crochemore-Perrin

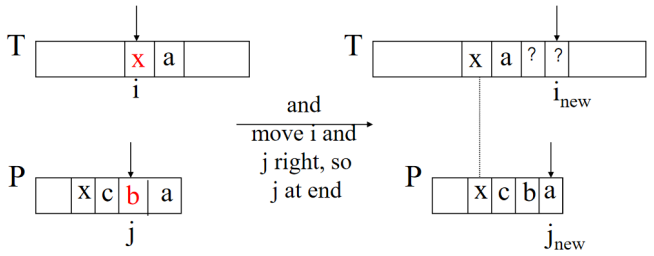
### 2.2 Algoritma Boyer Moore

- Definisi Boyer-Moore

Algoritma pencocokan string Boyer-Moore berbasis pada 2 (dua) teknik.

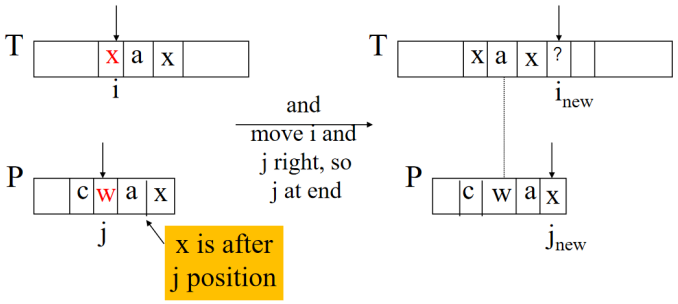
1. Teknik *looking-glass*
   Temukan P dalam T dengan bergerak mundur melalui P, mulai dari indeks akhir.

2. Teknik *character-jump*
   - ketika ketidakcocokan terjadi pada T [i] == x
   - karakter dalam pattern P [j] tidak sama dengan T [i]
   - Ada 3 kemungkinan kasus
     a. Jika P berisi x di suatu tempat, coba geser P ke kanan untuk menyelaraskan *last occurrence* x di P dengan T [i].



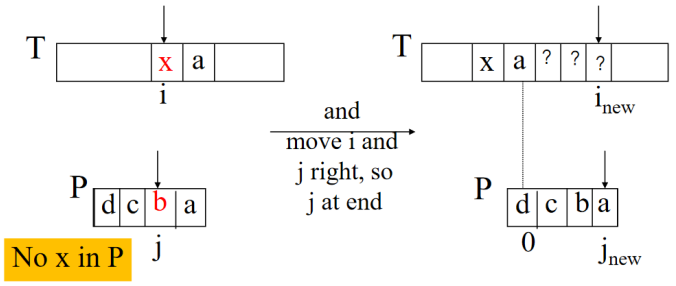

Gambar 1
(https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf)

b. Jika P berisi x di suatu tempat, tetapi pergeseran ke kanan ke *last occurrence* tidak dimungkinkan, maka geser Kanan sebanyak 1 karakter ke T [i + 1].



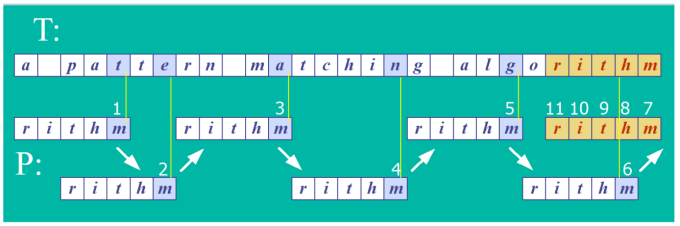

Gambar 2
(https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf)

c. Jika kasus 1 dan 2 tidak berlaku, geser P untuk menyejajarkan P [0] dengan T [i + 1].




Gambar 3
(https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf)

Contoh penggunaan Algoritma Boyer-Moore (1):




Gambar 4
(https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf)

- Fungsi *Last Occurance*

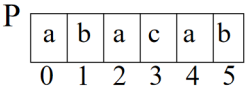
Algoritma Boyer-Moore memproses sebelumnya pattern P dan alfabet A untuk membangun fungsi *Last Occurance* L(). L() memetakan semua huruf di A menjadi integer.
L(x) didefinisikan sebagai: (x adalah huruf di A)

- Indeks terbesar i sedemikian rupa sehingga P [i] == x, atau
- -1 jika tidak ditemukan indeks i sedemikian rupa sehingga P [i] == x

Contoh:

## *L() Example*

- A = {a, b, c, d}
- P: "abacab"




| *x* | *a* | *b* | *c* | *d* |
|---|---|---|---|---|
| *L(x)* | 4 | 5 | 3 | -1 |

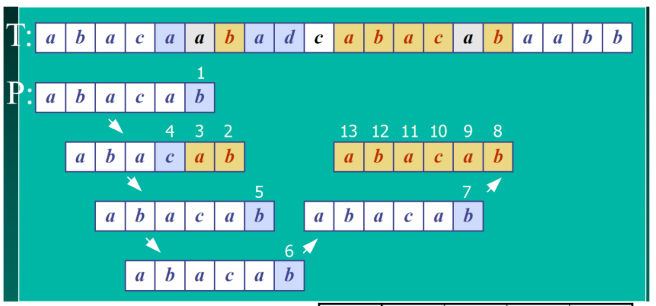
L() stores indexes into P[]

Gambar 5
(https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf)

Pada Algoritma Boyer-Moore, L() dihitung ketika pattern P sedang dibaca. Biasanya L() disimpan sebagai array.

Contoh penggunaan Algoritma Boyer-Moore (2):

## *Boyer-Moore Example (2)*




Jumlah perbandingan karakter: 13 kali

| *x* | *a* | *b* | *c* | *d* |
|---|---|---|---|---|
| *L(x)* | 4 | 5 | 3 | -1 |

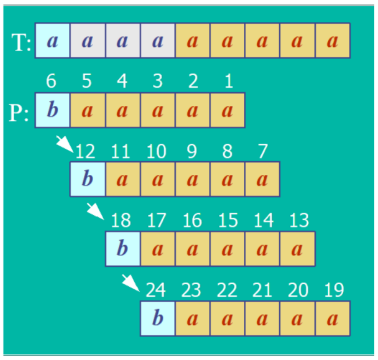
Gambar 6
(https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf)

Pada kasus terburuk, kompleksitas Algoritma Boyer-Moore adalah O(nm + A). Akan tetapi, Boyer-Moore cepat ketika alphabet (A) itu besar (banyak) , lambat ketika alphabet (A) Kecil (sedikit). Bagus pada pencarian teks bacaan (menggunakan huruf), tidak baik pada binary. Boyer-Moore lebih cepat secara signifikan dari Brute Force pada pencarian teks bacaan.

Contoh Worst Case (Kasus Terburuk):

- T: "aaaaa...a"
- P: "baaaaa"

Jumlah perbandingan karakter: 24 kali




Gambar 7
(https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf)

## 2.3 Word Search Puzzle

Word Search Puzzle adalah sebuah *game* atau permainan yang didesain menyerupai puzzle yang berisikan berbagai macam huruf. Tugas pemain adalah menemukan kumpulan huruf-huruf yang dapat membentuk suatu kata. Susunan huruf-huruf tersebut dapat berupa horizontal, vertikal, atau diagonal. Permainan ini dapat dimainkan sendiri maupun bersama-sama dengan teman, keluarga, dan orang lainnya. Jika dimainkan sendiri, pemain akan menyelesaikan permainan ketika semua kata ditemukan. Ketika bermain bersama orang lain, pemain yang menemukan kata terbanyak yang akan menjadi pemenangnya. Banyak kata, huruf, besar tabel, dan level kesulitan ditentukan oleh masing-masing pemain. Pada makalah ini diasumsikan jumlah baris dan kolom sama (matriks/tabel/kotak berbentuk persegi).


Gambar 8
(https://thewordsearch.com/)

# III. Implementasi dan Pengujian

## 3.1 Implementasi

Langkah-langkah yang harus dilakukan untuk memecahkan game Word Search Puzzles dengan menggunakan Algoritma Boyer-Moore adalah

1. Membagi pengecekan berdasarkan arahnya yaitu horizontal, vertikal, dan diagonal
2. Di setiap arah disimpan semua kemungkinan teks
3. Ukuran teks yang disimpan harus lebih dari sama dengan ukuran pattern
4. Simpan indeks-indeks penting pada setiap teks (misal: teks pada baris ke-3 dari tabel pada puzzle maka indeks baris ke-3 nya akan disimpan)
5. Menggunakan algoritma Boyer-Moore untuk mengecek pattern pada masing-masing teks yang sudah disimpan
6. Apabila ditemukan kata yang sesuai dengan pattern maka akan menyimpan indeks awal kata dan akhir kata
7. Menampilkan pattern beserta indeks awal dan akhirnya

Langkah 1:
Membagi pengecekan berdasarkan arahnya yaitu horizontal, vertikal, dan diagonal

Langkah 2:
Di setiap arah disimpan semua kemungkinan teks

```
procedure horizontal(puzzle, text_horizontal):
    i traversal [0, len(puzzle)]:
        {Menyimpan text pada setiap baris
         di puzzle ke text_horizontal}
        <Aksi>
```

```
procedure vertical(puzzle, text_vertical):
    i traversal [0, len(puzzle[0])]:
        j traversal [0, len(puzzle)]:
            {Menyimpan text pada setiap baris
             di puzzle ke text_horizontal}
            <Aksi>
```

```
procedure diagonal(puzzle, text_diagonal):
    {Mengambil dari indeks baris ke-0}
    i traversal [0, len(puzzle[0])]:
        b1 = 0
        k1 = i
        while <indeks b1 valid> and <indeks k1 valid>:
            {Mengambil pattern sampai akhir dari diagonal}
            <Aksi>
            b1 += 1
            k1 += 1
    {Mengambil dari indeks kolom ke-0}
    {Text diagonal dimulai dari indeks ke-len(puzzle[0])}
    j traversal [1, len(puzzle)]:
        b2 = j
        k2 = 0
        while <indeks b2 valid> and <indeks k2 valid>:
            {Mengambil pattern sampai akhir dari diagonal}
            <Aksi>
            b2 += 1
            k2 += 1
```

Langkah 3
Ukuran teks yang disimpan harus lebih dari sama dengan ukuran pattern. Apabila ukuran text didalam array lebih kecil dari ukuran pattern maka tidak akan dilakukan pengecekan terhadap text tersebut

Langkah 4
Simpan indeks-indeks penting pada setiap teks (misal: teks pada baris ke-3 dari tabel pada puzzle maka indeks baris ke-3 nya akan disimpan).
Pada langkah ketiga ini untuk masing-masing arah berbeda cara menyimpan indeks nya.

1. Untuk arah horizontal, indeks awal baris dari pattern akan tersimpan dari indeks array text_horizontal sedangkan indeks kolomnya akan tersimpan dari indeks text_horizontal[i].
2. Untuk arah vertikal, hampir sama seperti horizontal akan tetapi terbalik, yaitu indeks baris didapat dari text_vertical[i] sedangkan indeks kolom didapat dari indeks text_vertical.
3. Untuk arah diagonal, agak sedikit mempunyai catatan tentang cara pengambilan indeks awalnya, yaitu dengan memperhatikan indeks dari text_diagonal. Untuk indeks 0 sampai len(puzzle[0]) (panjang matriks puzzle baris pertama/setiap baris) maka indeks barisnya adalah 0 dan indeks kolomnya mengikuti indeks dari text_diagonal. Dan untuk indeks setelah itu maka indeks kolomnya sama dengan 0 dan indeks barisnya adalah indeks_text_diagonal dikurang len(puzzle[0]).

Kemudian untuk indeks akhir dari pattern:

1. Untuk arah horizontal, hanya tinggal menambahkan indeks kolom awal dengan panjang pattern kemudian dikurang satu (indeks_kolom_awal + panjang_pattern - 1), untuk indeks baris tetap sama untuk awal dan akhir.
2. Untuk arah vertical, sama seperti horizontal akan tetapi yang ditambahkan adalah indeks baris, sedangkan indeks kolom masih tetap sama.
3. Untuk arah diagonal, keduanya yaitu indeks baris dan kolom ditambahkan dengan cara yang sama seperti sebelumnya.

Langkah 5:
Menggunakan algoritma Boyer-Moore untuk mengecek pattern pada masing-masing teks yang sudah disimpan.
Berikut merupakan contoh Algoritma Boyer-Moore di Python:

```
#Jumlah karakter dalam ASCII
NUM_OF_CHAR = 256

# Fungsi untuk menyimpan array of integer dari
# tiap karakter

# Berisi indeks tiap karakter yang terdapat di
# Pattern sesuai letaknya di Pattern

# Indeks letaknya diletakan pada indeks array sesuai
# dengan nomor ASCII masing-masing karakter

def indexOfChar(pattern):
    arrayOfIndex = [-1]*NUM_OF_CHAR

    for i in range(len(pattern)):
        arrayOfIndex[ord(pattern[i])] = i

    return(arrayOfIndex)

def search_boyer_moore(txt, pat):
    arr_idx = indexOfChar(pat)
    n = len(txt)
    m = len(pat)
    freq = 0
    result = []

    shift = 0
    while(shift <= n - m):
        j = m - 1

        while(j >= 0 and pat[j] == txt[shift + j]):
            j -= 1

        # String pattern sesuai dengan string text
        if j < 0:
            print("String pattern sesuai dengan string text")
            print("pada indeks teks ke- ", shift)
            result.append(shift)
            freq += 1

            if (shift + m < n):
                shift += (m-arr_idx[ord(txt[shift+m])])

            else:
                shift += 1
```

```
        else :
            last_occur = j-arr_idx[ord(txt[shift+j])]
            # last occurrence harus berada di sebelah kiri
            # dari karakter yang sedang diperiksa (j)
            # Maka last_occur harus bernilai positif
            if (last_occur >= 1):
                shift += last_occur
            else:
                shift += 1

    if (freq == 0):
        return False, []
    return True, result
```

Algoritma ini menerima masukan text dan pattern dan mengembalikan nilai boolean (true atau false) dan indeks ke-berapa pattern tersebut dimulai di text.

Langkah 6:
Apabila ditemukan kata yang sesuai dengan pattern maka akan menyimpan indeks awal kata dan akhir kata

```
hasil = []
if (Search_Boyer_Moore(text_horizotal, <pat>) = True):
    {Menyimpan kata beserta indeks awal
     dan akhirnya didalam array hasil}
    {Indeks awal dan akhir dicari sesuai langkah masing-masing}
    <Aksi>

else if (Search_Boyer_Moore(text_vertical, <pat>) = True):
    {Menyimpan kata beserta indeks awal
     dan akhirnya didalam array hasil}
    {Indeks awal dan akhir dicari sesuai langkah masing-masing}
    <Aksi>

else if (Search_Boyer_Moore(text_diagonal, <pat>) = True):
    {Menyimpan kata beserta indeks awal
     dan akhirnya didalam array hasil}
    {Indeks awal dan akhir dicari sesuai langkah masing-masing}
    <Aksi>
```

Langkah 7:
Menampilkan pattern beserta indeks awal dan akhirnya

```
for i traversal[0,len(hasil)]:
        output (hasil)
{contoh output:
 harimau (0,0) (6,0)
 ikan (1,3) (1, 6)
 sate (2,4) (5,7)}
```

## 3.2 Pengujian

Pengujian program akan dilakukan dengan cara mencoba menjalankan simulasi algoritma Boyer-Moore dan Brute Force pada program sebanyak 10 kali pada ukuran papan 3 x 3, 10 x 10 serta 15 x 15.
Hasil Pengujian didapatkan dari sumber :
https://docplayer.info/29974898-Implementasi-algoritma-boyer-moore-pada-permainan-word-search-puzzle.html
Algoritma Brute Force dapat digunakan sebagai pembanding dalam hal performa efisiensi, kecepatan, dan kompleksitas.

Tabel 1.
Data uji coba algoritma Boyer-Moore dan Brute Force pada papan ukuran 3 x 3 (https://docplayer.info/29974898-Implementasi-algoritma-boyer-moore-pada-permainan-word-search-puzzle.html )

| No | Jumlah kata | Kata yang ditemukan | Waktu Algoritma Boyer-moore (s) | Waktu Algoritma Brute Force(s) |
|---|---|---|---|---|
| 1 | 2 | 2 | 0.173 | 0.251 |
| 2 | 2 | 2 | 0.334 | 0.592 |
| 3 | 2 | 2 | 0.475 | 0.991 |
| 4 | 2 | 2 | 0.467 | 1.291 |
| 5 | 2 | 2 | 0.501 | 1.112 |
| 6 | 2 | 2 | 0.478 | 1.192 |
| 7 | 2 | 2 | 0.273 | 0.587 |
| 8 | 2 | 2 | 0.175 | 0.452 |
| 9 | 2 | 2 | 0.512 | 1.331 |
| 10 | 2 | 2 | 0.332 | 0.681 |

Dari tabel diatas, didapatkan:
Rata-rata waktu memakai Algoritma Boyer-Moore = 0,372 detik
Rata-rata waktu memakai Algoritma Brute Force = 0,848 detik

Tabel 2.
Data uji coba algoritma Boyer-Moore dan Brute Force pada papan ukuran 10 x 10 (https://docplayer.info/29974898-Implementasi-algoritma-boyer-moore-pada-permainan-word-search-puzzle.html )

| No | Jumlah kata | Kata yang ditemukan | Waktu Algoritma Boyer-moore (s) | Waktu Algoritma Brute Force(s) |
|---|---|---|---|---|
| 1 | 11 | 11 | 11.248 | 67.983 |
| 2 | 11 | 11 | 13.199 | 79.865 |
| 3 | 11 | 11 | 11.768 | 63.95 |
| 4 | 11 | 11 | 9.855 | 54.957 |
| 5 | 11 | 11 | 11.574 | 67.765 |
| 6 | 11 | 11 | 6.118 | 32.005 |
| 7 | 11 | 11 | 7.41 | 40.932 |
| 8 | 11 | 11 | 15.341 | 98.494 |
| 9 | 11 | 11 | 10.168 | 58.573 |
| 10 | 11 | 11 | 12.264 | 71.654 |

Dari tabel diatas, didapatkan:
Rata-rata waktu memakai Algoritma Boyer-Moore = 10,8945 detik
Rata-rata waktu memakai Algoritma Brute Force = 63,6178 detik

Tabel 3
Data uji coba algoritma Boyer-Moore dan Brute Force pada papan ukuran 15 x 15 (https://docplayer.info/29974898-Implementasi-algoritma-boyer-moore-pada-permainan-word-search-puzzle.html )

| No | Jumlah kata | Kata yang ditemukan | Waktu Algoritma Boyer-moore (s) | Waktu Algoritma Brute Force(s) |
|---|---|---|---|---|
| 1 | 16 | 16 | 28.021 | 206.06 |
| 2 | 16 | 16 | 29.197 | 203.805 |
| 3 | 16 | 16 | 36.562 | 261.138 |
| 4 | 16 | 16 | 46.835 | 336.239 |
| 5 | 16 | 16 | 37.234 | 300.234 |
| 6 | 16 | 16 | 30.883 | 228.975 |
| 7 | 16 | 16 | 50.345 | 360.278 |
| 8 | 16 | 16 | 38.124 | 270.134 |
| 9 | 16 | 16 | 30.554 | 230.321 |
| 10 | 16 | 16 | 42.841 | 327.507 |

Dari tabel diatas, didapatkan:
Rata-rata waktu memakai Algoritma Boyer-Moore = 37,0596 detik
Rata-rata waktu memakai Algoritma Brute Force = 272,4691 detik

Dari ketiga tabel diatas didapatkan persentase keberhasilan adalah 100%.

## IV. Kesimpulan

Algoritma Boyer-Moore merupakan salah satu algoritma *String Matching* atau pencocokan string yang banyak sekali penerapannya di berbagai bidang keilmuan. Salah satu penerapan sederhananya adalah seperti topik yang penulis bawakan di makalah ini yaitu untuk memecahkan game Word Search Puzzles

Dengan menggunakan Algoritma Boyer-Moore, Kita dapat memecahkan game Word Search Puzzles dan bahkan dapat lebih efektif dan efisien dibandingkan menggunakan Algoritma Brute Force.

## V. Ucapan Terima Kasih

Puji syukur kepada Allah Subhanahu wa ta'ala yang telah memberikan rahmat dan karunia-Nya sehingga saya dapat menyelesaikan makalah ini.

Terima kasih juga kepada keluarga saya dan juga kepada para dosen pengampu mata kuliah Strategi Algoritma. Semoga ilmunya tetap berkah dan penulis dapat menerapkan ilmu yang didapat dari kuliah ini untuk kebaikan di masa depan.

## Referensi

[1] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf (diakses pada 11 Mei 2021 pukul 13.55)
[2] https://www.ilmuskripsi.com/2016/05/algoritma-pencarian-string-string.html (diakses pada 11/05/2021 pukul 13.55 WIB)
[3] https://thewordsearch.com/ (diakses pada 11/05/2021 pukul 14.51 WIB)
[4] https://docplayer.info/29974898-Implementasi-algoritma-boyer-moore-pada-permainan-word-search-puzzle.html (diakses pada 11/05/2021 pukul 16.00 WIB)
[5] https://www.geeksforgeeks.org/boyer-moore-algorithm-for-pattern-searching/ (diakses pada 11/05/2021 pukul 17.06 WIB)

## Video Link at Youtube

https://youtu.be/qKpnFU1-Jjk

## Pernyataan

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 11 Mei 2021

Muhammad Fahkry Malta 13519032