# *Penggunaan Algoritma Brute Force untuk Menyelesaikan Word Search Puzzle*

Michelle Theresia (13518050)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13518050@std.stei.itb.ac.id

***Abstract*—*Word Search* Puzzle adalah salah satu puzzle yang telah ada dalam bentuk perangkat lunak dan dapat dimainkan melalui website. Langkah manual penyelesaian *Word Search* Puzzle yaitu dengan melakukan pengecekan terhadap huruf-huruf di matriks soal dengan huruf-huruf dari kata=kata yang dicari satu per satu. Setelah menemukan yang sama, cek tetangga dari huruf matriks tersebut dan tarik garis lurus searah dengan huruf tetangga yang sesuai dengan huruf ke 2 (dua) kata yang dicari. Jika tidak sesuai lakukan pencarian dan mengulangi langkah pertama. Cara tersebut sangat mirip dengan algoritma *string matching*, yaitu algoritma *brute force*.**

***Keywords—word search puzzle; penyelesaian; pengecekan; huruf; matriks; brute force***

## I. PENDAHULUAN

Salah satu permainan yang mengasah otak adalah puzzle. Berdasarkan Cambridge Dictionary, definisi puzzle dalam makalah adalah sebuah masalah atau pertanyaan yang harus dijawab menggunakan kemampuan atau pengetahuan [1]. Puzzle biasanya merupakan kertas yang tercetak soal atau buku kumpulan puzzle. Saat ini, soal puzzle tersedia sebagai perangkat lunak atau dapat dikerjakan melalui website. Biasanya, puzzle dalam komputer memiliki fitur tambahan dibandingkan dengan puzzle tradisional, contohnya *hint*.

Salah satu puzzle yang telah ada dalam komputer adalah *word search* puzzle. Dalam implementasi penyelesaian word search puzzle, salah satu algoritma yang dapat digunakan adalah brute force.

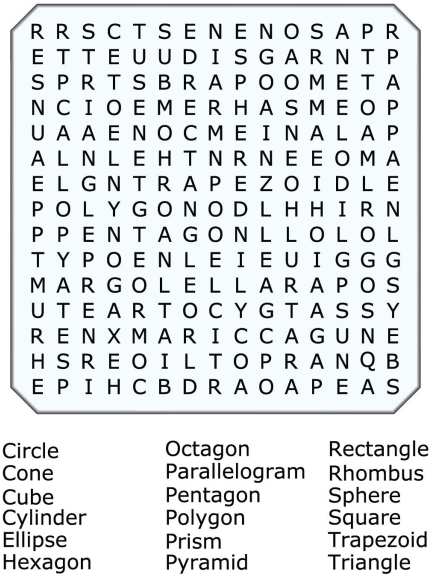
Definisi brute force berdasarkan Cambridge Dictionary yaitu menyelesaikan sesuatu dengan kekuatan yang besar [2]. Dalam algoritma brute force memiliki prinsip yang sama, penyelesaian masalah dengan tidak cerdas dan tidak mangkus (butuh jumlah komputasi yang besar dan waktu yang lama).

## II. DASAR TEORI

### A. *Word Search Puzzle*

Word Search Puzzle adalah sebuah permainan mencari kata-kata yang tertulis dari soal dari sebuah matriks yang berisi alfabet acak dengan jawaban tersembunyi. Strategi penyelesaian puzzle ini yaitu:

1. Perhatikan alfabet tiap baris serta tiap kolom untuk mencari huruf pertama kata yang dicari yang sesuai.
2. Cek 'tetangga' dari alfabet yang sesuai dengan huruf pertama kata yang dicari. Bandingkan alfabet 'tetangga' dengan huruf kedua dari kata yang dicari.
3. Jika ada alfabet tetangga yang sesuai, cek alfabet tetangga selanjutnya secara lurus. Kembali ke tahap 2 (dua).
4. Jika tidak ada alfabet tetangga yang sesuai, cari dengan pindah kolom setelahnya serta pindah baris hingga alfabet sesuai dengan huruf pertama kata yang dicari
5. Coret atau tandai huruf yang sesuai.
6. Ulangi langkah 1 sampai 5 hingga semua kata yang dicari ditemukan.




Gambar 2.1. Bentuk *word search* puzzle

Sumber: https://sciencenotes.org/2d-and-3d-shapes-word-search-puzzle/

Puzzle word search adalah jenis puzzle yang memiliki posisi ke-8 untuk peringkat jenis puzzle yang terkenal [3]. Puzzle word search dapat dimainkan oleh orang dewasa maupun anak-anak.

### B. *Algoritma Brute Force*

Algoritma brute force adalah pendekatan yang lempang (*straight forward*) untuk memecahkan suatu persoalan. Biasanya didasarkan pada pernyataan pada persoalan (*problem statement*) dan definisi konsep yang dilibatkan. Contoh berdasarkan pernyataan persoalan yaitu mencari elemen terbesar atau terkecil dan pencarian beruntun (*sequential search*). Contoh berdasarkan definisi konsep yang terlibat yaitu menghitung $a^n$ ($a > 0$, n bilangan bulat tak-negatif), menghitung n! (n bilangan bulat tak-negatif), mengalikan dua buah matriks, tes bilangan prima, algoritma pengurutan *brute force* (*selection sort*, *bubble sort*, dan mengevaluasi polinom.

Algoritma *brute force* memecahkan persoalan dengan sangat sederhana, langsung, dan jelas (*obvious way*). Algoritma *brute force* memiliki karakteristik:

1. Tidak "cerdas" dan tidak mangkus karena membutuhkan jumlah komputasi yang besar dan waktu yang lama dalam penyelesaiannya. Sesuai namanya "force" mengindikasikan "tenaga" dibandingkan "otak". Dapat dikatakan juga algoritma naif.
2. Lebih cocok untuk persoalan berukuran kecil karena sederhana dan implementasinya mudah. Sering digunakan sebagai basis pembanding dengan algoritma lain yang lebih mangkus.
3. Hampir semua persoalan dapat diselesaikan.

## III. Implementasi dan Uji

### A. *Implementasi*

Penulis membuat implementasi dengan matriks huruf berukuran 5x5 terdiri dari huruf kapital. Cara pencarian yaitu dari mencari huruf pada matriks yang sesuai dengan huruf pertama kata yang dicari. Kemudian cari 'tetangga' huruf pada matriks yang sesuai dengan huruf kedua kata. Jika ada yang sama, tarik garis lurus dari huruf awal mengikuti mengikuti huruf kedua dan ulangi dari pencocokan huruf pada matriks dengan huruf kata kesekian sesuai sudah langkah berapa. Berikut fungsi untuk pencarian huruf pertama pada matriks yang sesuai dengan huruf pertama kata yang dicari.

```
def findSameFirstWordInRow(row, word,
puzzle):
  col = 0
  while (puzzle[row][col] != word[0])
and col != (matrixSize-1):
    col = col + 1
  if(puzzle[row][col] == word[0]):
    return col
  else:
    return -1
```

Arah pencarian kata terdiri dari horizontal kiri ke kanan, vertikal atas ke bawah, diagonal kanan atas ke kiri bawah, dan diagonal kanan bawah ke kiri atas.

1. Horizontal kiri ke kanan

```
def checkHorizontalLToR(row, col,
word, puzzle):
  idx = []
  j = col
  if (col != (matrixSize-1)) and ((col
+ len(word) - 1) < matrixSize):
    w = 0
    i = row
    j = col
    match = True
    while(match and j < (col +
len(word))):
      if(word[w] == puzzle[i][j]):
        idx.append([i,j])
        w = w + 1
        j = j + 1
      else:
        match = False
    if(match):
      for k in range (len(idx)):
        temp =
puzzle[idx[k][0]][idx[k][1]]
        puzzle[idx[k][0]][idx[k][1]] =
"[" + temp + "]"
```

Syarat utama suatu pencarian memiliki kemungkinan ke arah horizontal kanan ke kiri yaitu dari titik matriks ditemukan huruf pertama yang sesuai dengan huruf pertama kata yang dicari yaitu jika titik ditambah dengan panjang kata yang dicari tidak melebihi panjang kolom, untuk kasus ini adalah 5 (lima) dan titik tidak terletak di kolom terakhir. Kemudian pengecekan secara searah horizontal dengan huruf selanjutnya

di kata pencarian. Jika ada salah satu huruf yang salah, hasil tidak akan di print.

2. Vertikal atas ke bawah

```
def checkVerticalUtoD(raw, col, word,
puzzle):
  idx = []
  i = row
  if (row != (matrixSize-1)) and ((row
+ len(word) - 1) < matrixSize):
    w = 0
    i = row
    j = col
    match = True
    while(match and i < (row +
len(word))):
      if(word[w] == puzzle[i][j]):
        idx.append([i,j])
        w = w + 1
        i = i + 1
      else:
        match = False
    if(match):
      for k in range (len(idx)):
        temp =
puzzle[idx[k][0]][idx[k][1]]
        puzzle[idx[k][0]][idx[k][1]] =
"[" + temp + "]"
```

Syarat utama suatu pencarian memiliki kemungkinan ke arah vertikal atas ke bawah yaitu dari titik matriks ditemukan huruf pertama yang sesuai dengan huruf pertama kata yang dicari yaitu jika titik ditambah dengan panjang kata yang dicari tidak melebihi panjang baris, untuk kasus ini adalah 5 (lima) dan titik tidak terletak di baris terakhir. Kemudian pengecekan secara searah vertikal ke bawah dengan huruf selanjutnya di kata pencarian. Jika ada salah satu huruf yang salah, hasil tidak akan di print.

3. Diagonal kiri bawah ke kiri kanan atas

```
def checkDiagonalLDtoRU(row, col,
word, puzzle):
  idx = []
  j = col
  if (row != 1) and (col !=
(matrixSize-1)) and ((col + len(word)
- 1) < matrixSize):
    w = 0
    i = row
    j = col
    match = True
    while(match and i > (row -
len(word))):
      if(word[w] == puzzle[i][j]):
        idx.append([i,j])
        w = w + 1
        i = i - 1
        j = j + 1
      else:
        match = False
    if(match):
      for k in range (len(idx)):
        temp =
puzzle[idx[k][0]][idx[k][1]]
        puzzle[idx[k][0]][idx[k][1]] =
"[" + temp + "]"
```

Syarat utama suatu pencarian memiliki kemungkinan ke arah diagonal kanan atas ke kiri bawah yaitu dari titik matriks ditemukan huruf pertama yang sesuai dengan huruf pertama kata yang dicari yaitu jika titik ditambah dengan panjang kata yang dicari tidak melebihi panjang kolom, untuk kasus ini adalah 5 (lima), titik tidak terletak pada baris pertama. Kemudian pengecekan secara searah diagonal ke kanan atas dengan huruf selanjutnya di kata pencarian. Jika ada salah satu huruf yang salah, hasil tidak akan di print.

4. Diagonal kiri atas ke kanan bawah

```
def checkDiagonalLUtoRD(row, col,
word, puzzle):
  idx = []
  j = col
  if (row != (matrixSize-1)) and (col
!= (matrixSize-1)) and ((col +
len(word) - 1) < matrixSize):
```

```
    w = 0
    i = row
    j = col
    match = True
    while(match and i < (row +
len(word))):
      if(word[w] == puzzle[i][j]):
        idx.append([i,j])
        w = w + 1
        i = i + 1
        j = j + 1
      else:
        match = False
    if(match):
      for k in range (len(idx)):
        temp =
puzzle[idx[k][0]][idx[k][1]]
        puzzle[idx[k][0]][idx[k][1]] =
"[" + temp + "]"
```

Syarat utama suatu pencarian memiliki kemungkinan ke arah diagonal kiri bawah ke kanan atas yaitu dari titik matriks ditemukan huruf pertama yang sesuai dengan huruf pertama kata yang dicari yaitu jika titik ditambah dengan panjang kata yang dicari tidak melebihi panjang kolom, untuk kasus ini adalah 5 (lima) dan titik tidak terletak di kolom terakhir dan baris terakhir. Kemudian pengecekan secara searah ke kanan bawah dengan huruf selanjutnya di kata pencarian. Jika ada salah satu huruf yang salah, hasil tidak akan di print.

*B. Uji dan Hasil Uji*

Berikut soal yang digunakan untuk pengujian

TABEL I. MATRIX PUZZLE

| | | | | |
|---|---|---|---|---|
| L | W | M | E | R |
| I | W | C | E | X |
| O | V | G | A | Q |
| N | I | N | I | T |
| T | R | A | E | B |

Himpunan kata yang dicari: {'CAT', 'BEAR', 'TIGER', 'LION'}

Hasil uji:

```
Soal:
======================
L W M E R
I W C E X
O V G A Q
N I N I T
T R A E B
======================
Jawaban yang diberi tanda [...]
======================
[L] W M E [R]
[I] W [C] [E] X
[O] V [G] [A] Q
[N] [I] N I [T]
[T] R A E B
======================
```

Semua kata yang dicari ditemukan.

## IV. KESIMPULAN

Berdasarkan analisis dan pengujian makalah, dapat disimpulkan algoritma *brute force* dapat digunakan untuk menyelesaikan *word search* puzzle dengan langkah-langkah strategi penyelesaian puzzle manual. Akan tetapi, dibutuhkan pengujian untuk algoritma lain yang lebih cepat dan efektif dari algoritma *brute force* untuk menyelesaikan *word search* puzzle.

## V. UCAPAN TERIMA KASIH

Penulis mengucapkan puji syukur kepada Tuhan Yang MahaEsa atas selesainya makalah ini. Penulis mengucapkan terima kasih kepada dosen IF2211 Strategi Algoritma atas bimbingan dan pengetahuan yang telah diberikan selama semester 4. Penulis juga mengucapkan terima kasih kepada keluarga atas menyediakan fasilitas yang penting atas selesainya makalah ini. Penulis turut mengucapkan terima kasih pada pihak-pihak yang belum disebutkan atas dukungan penyelesaian makalah ini.

## VIDEO LINK AT YOUTUBE

Video makalah ini dapat diakses melalui URL https://youtu.be/ozBLKe6nGd0 .

## REFERENCES

[1] Admin. "Meaning of Puzzle in English". Diakses dari https://dictionary.cambridge.org/dictionary/english/puzzle pada tanggal 3 Mei 2020.

[2] Admin. “Meaning of brute force in English”. Diakses dari https://dictionary.cambridge.org/dictionary/english/brute?q=brute+force pada tanggal 3 Mei 2020

[3] phdlist. (2011). “Most Popular Types of Puzzles”. Diakses dari http://shareranks.com/9085,Most-Popular-Types-of-Puzzles pada tanggal 3 Mei 2020

[4] Rinaldi Munir. (2014). “Algoritma *Brute Force*”. Diakses dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2017-2018/Algoritma-Brute-Force-(2016).pdf pada tanggal 3 Mei 2020.

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, 29 April 2020

Michelle Theresia 13518050