TUGAS BESAR STRATEGI ALGORITMA
ANALISIS PERBANDINGAN ALGORITMA BRUTE FORCE DAN BACKTRACKING DALAM WORD SEARCH PUZZLE

Disusun oleh :

Diva Annisa Febecca (1301204302)
Muhammad Mufid Utomo (1301204441)
Johanes Raphael Nandaputra (1301204243)

PROGRAM STUDI S1 INFORMATIKA
FAKULTAS INFORMATIKA
UNIVERSITAS TELKOM
BANDUNG
2022

DAFTAR ISI

ABSTRAK

Dalam menyelesaikan sebuah permainan *puzzle* dibutuhkan ketelitian dan kecerdasan. Salah satu permainan *puzzle* yang populer saat ini adalah *Word Search Puzzle*. Permainan *Word Search Puzzle* adalah permainan pencarian kata dalam kumpulan huruf yang tersusun secara acak dalam bentuk *array* dua dimensi (matriks). Permainan ini bisa diselesaikan dengan algoritma *brute force* dan *backtracking*. Laporan ini akan membahas tentang *brute force* dan *backtracking*, serta perbandingannya dalam menyelesaikan Word Search Puzzle berdasarkan kompleksitas dan efisiensi waktu dari program yang telah penulis buat.

Kata Kunci: *word search puzzle, algoritma, brute force, backtracking*

BAB I

PENDAHULUAN

Permainan atau biasa disebut dengan *game* merupakan sarana hiburan yang diminati dan dimainkan oleh banyak orang baik dari kalangan anak - anak, remaja maupun orang dewasa. Permainan juga dapat melatih dan mengasah kemampuan berpikir seseorang. Salah satu *game* yang dapat melatih dan mengasah kemampuan berpikir adalah *Word Search Game*. *Word Search Puzzle* adalah permainan pencarian kata dalam kumpulan huruf yang tersusun secara acak dalam bentuk *array* dua dimensi (matriks). Objektif dari permainan ini adalah menemukan semua kata tersembunyi di matriks permainan yang dapat ditemukan secara vertikal, horizontal, dan diagonal. Permainan ini menarik karena tidak hanya dapat mengasah kemampuan berpikir tetapi juga dapat meningkatkan ketelitian karena pemain harus teliti dalam mencari kata - kata diantara serangkaian huruf yang membentuk matriks.

Untuk mencari keseluruhan kata yang ada tidaklah mudah, pemain harus menemukan seluruh kata yang tersembunyi dalam matriks permainan. Salah satu cara untuk mencari kata dalam matriks huruf adalah dengan menggunakan Algoritma *Brute Force*. Algoritma ini dapat diterapkan untuk hampir semua permasalahan. Selain Algoritma *Brute Force*, Algoritma *Backtracking* dapat menjadi pilihan untuk mencari kata dalam matriks huruf. Pada laporan ini, akan dibahas mengenai *Word Search Puzzle* dan solusi algoritma untuk menyelesaikan permasalahan ini, serta menganalisis waktu yang dibutuhkan oleh masing - masing algoritma (Algoritma *Brute Force* dan Algoritma *Backtracking*) untuk menemukan solusinya.

BAB II

DASAR TEORI

2.1 *Word Search Puzzle*

Word Search Puzzle adalah permainan pencarian kata dalam kumpulan huruf yang tersusun secara acak dalam bentuk *array* dua dimensi (matriks). Permainan ini bertujuan untuk mencari semua kata tersembunyi dalam matriks yang dapat ditemukan secara vertikal, horizontal, dan diagonal. Contoh dari permasalahan ini adalah misalnya diberikan sebuah array dengan baris dan kolom sebagai berikut.

[‘T’, ‘G’, ‘ I’]

[‘R’, ‘A’, ‘O’]

[‘S’, ‘Q’, ‘C’]

Dari array tersebut akan ditemukan kata tersembunyi yaitu ‘CAT’. Dan dari beberapa huruf yang terdapat di dalamnya, ditemukan kata ‘CAT’ dengan huruf pertama ‘C’ terletak di array[2,2] dengan kata ‘CAT’ ditemukan di arah diagonal ke kiri atas dengan urutan :

$[2,2] \rightarrow [1,1] \rightarrow [0,0]$

C A T

2.2 Strategi Algoritma

2.2.1 Algoritma *Brute Force*

Algoritma *Brute force* adalah pendekatan langsung (*straightforward)* dalam penyelesaian suatu masalah berdasarkan pernyataan dan definisi konsepnya. Algoritma ini memecahkan masalah dengan cara yang sederhana dan jelas. Cara kerjanya adalah dengan membangkitkan semua kemungkinan solusi. Karakteristik algoritma *brute force* adalah tidak cerdas dan membutuhkan waktu yang lama sehingga lebih cocok untuk menyelesaikan masalah yang tergolong kecil karena implementasinya yang mudah diterapkan. Algoritma ini seringkali bersifat efektif tapi tidak efisien.

2.2.2 Algoritma *Backtracking*

Algoritma *Backtracking* (Runut Balik) merupakan perbaikan dari Algoritma *Brute Force*. Algoritma ini cukup mangkus untuk digunakan dalam beberapa penyelesaian masalah. Dasar teknik *Backtracking* adalah teknik pencarian *(searching)*. Teknik ini bertujuan untuk mendapatkan himpunan penyelesaian yang mungkin. Dari himpunan penyelesaian yang mungkin ini akan diperoleh hasil solusi yang optimal. Algoritma *Backtracking* adalah algoritma yang berbasis *Depth First Search* (DFS) sehingga cara kerjanya tidak perlu mencari semua kemungkinan solusi yang ada. Hanya pencarian yang mengarah ke solusi yang akan dipertimbangkan. Simpul - simpul yang tidak mengarah ke solusi akan dipangkas sehingga akan menghemat waktu pencarian solusi.

BAB III

IMPLEMENTASI

3.1 Penyelesaian *Word Search Puzzle* dengan Algoritma *Brute Force*

Pada kasus *Word Search Puzzle* dengan menggunakan *brute force,* program akan berjalan dengan langkah umum sebagai berikut:

1. Mencari huruf pertama dari kata tersembunyi yang ingin ditemukan dengan menggunakan linear search.
2. Jika huruf pertama ditemukan, melakukan pengecekan tiap tetangga untuk menemukan huruf kedua yang cocok.
3. Jika ditemukan tetangga yang cocok dengan huruf kedua akan dilakukan perulangan sesuai arah tersebut untuk mengecek apakah sesuai dengan kata tersembunyi yang dicari.
4. Jika tidak ditemukan tetangga yang cocok dengan huruf kedua, maka kembali ke langkah 1 untuk memulai pencarian huruf pertama kembali.
5. Jika ditemukan kata yang cocok, maka program selesai.

Berikut algoritma *Brute Force* untuk menyelesaikan *Word Search Puzzle* dalam bentuk code Python:

```
def BruteForce(puzzle, word):
  firstLetter = word[0]
  secondLetter = word[1]
  lenWord = len(word)

  for row in range(len(puzzle)):
    for col in range(len(puzzle[row])):
      if (puzzle[row][col] == firstLetter):

        # Cek Arah ATAS
        try:
          if (puzzle[row-1][col] == secondLetter) and (row-1 >= 0):
            match = True
            idx = 2
            while (idx < lenWord) and match:
              if (puzzle[row-idx][col] == word[idx]) and (row-idx >= 0):
                idx += 1
              else:
                match = False
            if match:
              print('Kata Tersembunyi ditemukan di Posisi: [{},{}]'.format(row+1,col+1))
              print("Arah : ", end="")
              print("VERTIKAL KE ATAS")
              return;
        except IndexError:
          pass
```

```
# Cek Arah KANAN ATAS
try:
  if (puzzle[row-1][col+1] == secondLetter) and (row-1 >= 0):
    match = True
    idx = 2
    while (idx < lenWord) and match:
      if (puzzle[row-idx][col+idx] == word[idx]) and (row-idx >= 0):
        idx += 1
      else:
        match = False
    if match:
      print('Kata Tersembunyi ditemukan di Posisi: [{},{}]'.format(row+1,col+1))
      print("Arah : ", end="")
      print("DIAGONAL KE KANAN ATAS")
      return;
except IndexError:
  pass

# Cek Arah KANAN
try:
  if (puzzle[row][col+1] == secondLetter):
    match = True
    idx = 2
    while (idx < lenWord) and match:
      if (puzzle[row][col+idx] == word[idx]):
        idx += 1
      else:
        match = False
    if match:
      print('Kata Tersembunyi ditemukan di Posisi: [{},{}]'.format(row+1,col+1))
      print("Arah : ", end="")
      print("HORIZONTAL KE KANAN")
      return;
except IndexError:
  print("kanan error")
  pass

# Cek Arah KANAN BAWAH
try:
  if (puzzle[row+1][col+1] == secondLetter):
    match = True
    idx = 2
    while (idx < lenWord) and match:
      if (puzzle[row+idx][col+idx] == word[idx]):
        idx += 1
      else:
        match = False
    if match:
      print('Kata Tersembunyi ditemukan di Posisi: [{},{}]'.format(row+1,col+1))
      print("Arah : ", end="")
      print("DIAGONAL KE KANAN BAWAH")
      return;
except IndexError:
  pass

# Cek Arah BAWAH
try:
  if (puzzle[row+1][col] == secondLetter):
```

```
            match = True
            idx = 2
            while (idx < lenWord) and match:
              if (puzzle[row+idx][col] == word[idx]):
                idx += 1
              else:
                match = False
            if match:
              print('Kata Tersembunyi ditemukan di Posisi: [{},{}]'.format(row+1,col+1))
              print("Arah : ", end="")
              print("VERTIKAL KE BAWAH")
              return;
        except IndexError:
          pass

        # Cek Arah KIRI BAWAH
        try:
          if (puzzle[row+1][col-1] == secondLetter) and (col-1 >= 0):
            match = True
            idx = 2
            while (idx < lenWord) and match:
              if (puzzle[row+idx][col-idx]) and (col-idx >= 0):
                idx += 1
              else:
                match = False
            if match:
              print('Kata Tersembunyi ditemukan di Posisi: [{},{}]'.format(row+1,col+1))
              print("Arah : ", end="")
              print("DIAGONAL KE KIRI BAWAH")
              return;
        except IndexError:
          pass

        # Cek Arah KIRI
        try:
          if (puzzle[row][col-1] == secondLetter) and (col-1 >= 0):
            match = True
            idx = 2
            while (idx < lenWord) and match:
              if (puzzle[row][col-idx] == word[idx]) and (col-idx >= 0):
                idx += 1
              else:
                match = False
            if match:
              print('Kata Tersembunyi ditemukan di Posisi: [{},{}]'.format(row+1,col+1))
              print("Arah : ", end="")
              print("HORIZONTAL KE KIRI")
              return;
        except IndexError:
          pass

        # Cek Arah KIRI ATAS
        try:
          if (puzzle[row-1][col-1] == secondLetter) and (row-1 >= 0 and col-1 >= 0):
            match = True
            idx = 2
            while (idx < lenWord) and match:
              if (puzzle[row-idx][col-idx] == word[idx]) and (row-idx >= 0 and col-idx >= 0):
                idx += 1
```

```
            else:
              match = False
          if match:
            print('Kata Tersembunyi ditemukan di Posisi: [{},{}]'.format(row+1,col+1))
            print("Arah : ", end="")
            print("DIAGONAL KE KIRI ATAS")
            return;
        except IndexError:
          pass

  print("Kata Tidak Ditemukan")
  return;
```

3.2 Penyelesaian *Word Search Puzzle* dengan Algoritma *Backtracking*

Pada kasus *Word Search Puzzle* dengan menggunakan *backtracking,* program akan berjalan seperti *brute force*, tapi yang membedakannya adalah *backtracking* akan sekaligus mengecek apakah tetangga yang dicek memiliki panjang yang sama atau lebih daripada panjang kata. Program berjalan sebagai berikut:

1. Mencari huruf pertama dari kata tersembunyi yang ingin ditemukan dengan menggunakan linear search.
2. Jika huruf pertama ditemukan, melakukan pengecekan tiap tetangga untuk menemukan huruf kedua yang cocok dan memiliki.
3. Jika ditemukan tetangga yang cocok dengan huruf kedua akan dilakukan perulangan sesuai arah tersebut untuk mengecek apakah sesuai dengan kata tersembunyi yang dicari.
4. Jika tidak ditemukan tetangga yang cocok dengan huruf kedua, maka kembali ke langkah 1 untuk memulai pencarian huruf pertama kembali.
5. Jika ditemukan kata yang cocok, maka program selesai.

Berikut algoritma *Backtracking* untuk menyelesaikan *Word Search Puzzle* dalam bentuk code Python:

```
def Backtracking(puzzle, word):
  firstLetter = word[0]
  secondLetter = word[1]
  lenWord = len(word)
  totRow = len(puzzle)
  totCol = len(puzzle[0])

  for row in range(len(puzzle)):
    for col in range(len(puzzle[row])):
      if (puzzle[row][col] == firstLetter):

        # Cek Arah ATAS
        try:
          if ((puzzle[row-1][col] == secondLetter) and (row-1 >= 0)) and (row-(lenWord-1) >= 0):
            match = True
            idx = 2
```

```
            while (idx < lenWord) and match:
              if (puzzle[row-idx][col] == word[idx]) and (row-idx >= 0):
                idx += 1
              else:
                match = False
            if match:
              print('Kata Tersembunyi ditemukan di Posisi: [{},{}]'.format(row+1,col+1))
              print("Arah : ", end="")
              print("VERTIKAL KE ATAS")
              return;
        except IndexError:
          pass

        # Cek Arah KANAN ATAS
        try:
          if ((puzzle[row-1][col+1] == secondLetter) and (row-1 >= 0)) and (row-(lenWord-1) >= 0 and
col+lenWord <= totCol):
            match = True
            idx = 2
            while (idx < lenWord) and match:
              if (puzzle[row-idx][col+idx] == word[idx]) and (row-idx >= 0):
                idx += 1
              else:
                match = False
            if match:
              print('Kata Tersembunyi ditemukan di Posisi: [{},{}]'.format(row+1,col+1))
              print("Arah : ", end="")
              print("DIAGONAL KE KANAN ATAS")
              return;
        except IndexError:
          pass

        # Cek Arah KANAN
        try:
          if (puzzle[row][col+1] == secondLetter) and (col+lenWord <= totCol):
            match = True
            idx = 2
            while (idx < lenWord) and match:
              if (puzzle[row][col+idx] == word[idx]):
                idx += 1
              else:
                match = False
            if match:
              print('Kata Tersembunyi ditemukan di Posisi: [{},{}]'.format(row+1,col+1))
              print("Arah : ", end="")
              print("HORIZONTAL KE KANAN")
              return;
        except IndexError:
          pass

        # Cek Arah KANAN BAWAH
        try:
          if (puzzle[row+1][col+1] == secondLetter) and (row+lenWord <= totRow and col+lenWord <=
totCol):
            match = True
            idx = 2
            while (idx < lenWord) and match:
              if (puzzle[row+idx][col+idx] == word[idx]):
                idx += 1
```

```
            else:
              match = False
          if match:
            print('Kata Tersembunyi ditemukan di Posisi: [{},{}]'.format(row+1,col+1))
            print("Arah : ", end="")
            print("DIAGONAL KE KANAN BAWAH")
            return;
        except IndexError:
          pass

        # Cek Arah BAWAH
        try:
          if (puzzle[row+1][col] == secondLetter) and (row+lenWord <= totRow):
            match = True
            idx = 2
            while (idx < lenWord) and match:
              if (puzzle[row+idx][col] == word[idx]):
                idx += 1
              else:
                match = False
            if match:
              print('Kata Tersembunyi ditemukan di Posisi: [{},{}]'.format(row+1,col+1))
              print("Arah : ", end="")
              print("VERTIKAL KE BAWAH")
              return;
        except IndexError:
          pass

        # Cek Arah KIRI BAWAH
        try:
          if ((puzzle[row+1][col-1] == secondLetter) and (col-1 >= 0)) and (row+lenWord <= totRow and
col-(lenWord-1) >= 0):
            match = True
            idx = 2
            while (idx < lenWord) and match:
              if (puzzle[row+idx][col-idx]) and (col-idx >= 0):
                idx += 1
              else:
                match = False
            if match:
              print('Kata Tersembunyi ditemukan di Posisi: [{},{}]'.format(row+1,col+1))
              print("Arah : ", end="")
              print("DIAGONAL KE KIRI BAWAH")
              return;
        except IndexError:
          pass

        # Cek Arah KIRI
        try:
          if ((puzzle[row][col-1] == secondLetter) and (col-1 >= 0)) and (col-(lenWord-1) >= 0):
            match = True
            idx = 2
            while (idx < lenWord) and match:
              if (puzzle[row][col-idx] == word[idx]) and (col-idx >= 0):
                idx += 1
              else:
                match = False
            if match:
              print('Kata Tersembunyi ditemukan di Posisi: [{},{}]'.format(row+1,col+1))
```

```
              print("Arah : ", end="")
              print("HORIZONTAL KE KIRI")
              return;
        except IndexError:
          pass

        # Cek Arah KIRI ATAS
        try:
          if ((puzzle[row-1][col-1] == secondLetter) and (row-1 >= 0 and col-1 >= 0)) and
(row-(lenWord-1) >= 0 and col-(lenWord-1) >= 0):
            match = True
            idx = 2
            while (idx < lenWord) and match:
              if (puzzle[row-idx][col-idx] == word[idx]) and (row-idx >= 0 and col-idx >= 0):
                idx += 1
              else:
                match = False
            if match:
              print('Kata Tersembunyi ditemukan di Posisi: [{},{}]'.format(row+1,col+1))
              print("Arah : ", end="")
              print("DIAGONAL KE KIRI ATAS")
              return;
        except IndexError:
          pass

  print("Kata Tidak Ditemukan")
  return;
```

BAB IV

ANALISIS

4.1 Analisis Kompleksitas Waktu Algoritma *Brute Force*

Kompleksitas waktu penyelesaian masalah *Word Search Puzzle* dengan menggunakan Algoritma *Brute Force* adalah sebagai berikut.

$$T(n) = \sum_{i=1}^{m} \sum_{i=1}^{n} 8 \sum_{i=2}^{l}$$

$$T(n) = m \cdot \sum_{i=1}^{n} 8 \sum_{i=2}^{l}$$

$$T(n) = m \cdot n \cdot 8 \cdot \sum_{i=2}^{l}$$

$$T(n) = m \cdot n \cdot 8 \cdot l$$

$$T(n) = O(m \cdot n \cdot l)$$

4.2 Analisis Kompleksitas Waktu Algoritma *Backtracking*

Kompleksitas waktu penyelesaian masalah *Word Search Puzzle* dengan menggunakan Algoritma *Backtracking* adalah sebagai berikut.

$$T(n) = \sum_{i=1}^{m} \sum_{i=1}^{n} 8 \sum_{i=2}^{l}$$

$$T(n) = m \cdot \sum_{i=1}^{n} 8 \sum_{i=2}^{l}$$

$$T(n) = m \cdot n \cdot 8 \cdot \sum_{i=2}^{l}$$

$$T(n) = m \cdot n \cdot 8 \cdot l$$

$$T(n) = O(m \cdot n \cdot l)$$

4.3 Perbandingan Efisiensi Algoritma *Backtracking* dan *Brute Force*

Algoritma *backtracking* dan *brute force* memang memiliki kompleksitas waktu yang sama. Perbedaan efisiensinya baru terlihat ketika kita menjalankannya selama beberapa kali, dengan puzzle yang tertera sebagai berikut. Durasi yang ditampilkan dalam detik.

```
Word Search Puzzle

E D S H O P E F U L

T E N S T T J C R N

A T E D P P M E A N

R E K N O O O D G P

E M R A N T I W A R

C O A P H R A S E D

L C D X T O J O E R

U L H E O H O K U M

H E W E U P C R N D

O W H N G O N E T P

S R E T H G I F I H

T M E C E H S U D O

E J D V N E P S I B

L L E T E R E E E I

S R D E D F P S R A
```

Kata yang Dicari	Durasi dengan *Brute Force*	Durasi dengan *Backtracking*
ANTIWAR	0.000167	0.000077
CHOCKED	0.000145	0.000097
DARKENS	0.000101	0.000092
EGGS	0.000142	0.000133
EXPANDS	0.000100	0.000064
FEVER	0.000116	0.000097
FIGHTERS	0.000096	0.000112

HEEDED	0.000273	0.000093
HOKUM	0.000110	0.000094
HOSTEL	0.000239	0.000102

Dari hasil di atas, terlihat bahwa algoritma *backtracking* hampir selalu lebih cepat daripada algoritma *brute force*. Setelah dijalankan sepuluh kali, algoritma *brute force* berhasil menemukan kata yang dicari dalam waktu rata-rata 0.000149 detik, sedangkan algoritma *backtracking* memiliki rata-rata 0.00096 detik.

BAB V

KESIMPULAN

Masalah *Word Search Puzzle* dapat diselesaikan dengan menggunakan beberapa algoritma seperti Algoritma *Brute Force* dan *Algoritma Backtracking*. Kedua algoritma tersebut memiliki kompleksitas waktu yang sama, yaitu O (m . n . l). Namun, dalam hal efisiensi Algoritma *Backtracking* lebih efisien dibandingkan Algoritma *Brute Force*. Maka dapat disimpulkan bahwa algoritma yang lebih baik dalam hal efisiensi untuk menyelesaikan masalah *Word Search Puzzle* adalah Algoritma *Backtracking.*

LAMPIRAN

Github

https://github.com/johanesrn/word-search-puzzle/

Colaboratory : https://colab.research.google.com/drive/1f3XuAPIZakCy5HZZPK0tcZmVKUjMKUOk?usp=sharing

DAFTAR PUSTAKA

NST, Muhammad Rival Anggi. 2010. *Analisis dan Implementasi Algoritma Runut Balik (Backtracking) pada Permainan Magic Square*.

Theresia, Michelle. 2019. *Penggunaan Algoritma Brute Force untuk Menyelesaikan Word Search Puzzle.* Diakses dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2019-2020/Makalah/stima2020k2-040.pdf

Zulen, Avlin Andhika. 2009. *Penerapan Algoritma Backtracking Pada Permainan Word Search Puzzle*. Diakses dari https://docplayer.info/35295559-Penerapan-algoritma-backtracking-pada-permainan-word-search-puzzle.html?_gl=1*17r3321*_ga*Y1NRM1R6WDg2aGdQVXFmSlFUR2wwSXBpZHZfRW9ha3BYOF9EUWdVNXFSdExOSUpSbjdka0Z1cUY5dEt1dklzZA#