Pengaplikasian Algoritma Brute Force dalam Penyelesaian Permainan *Teka - teki Wordscapes Search*

Zaidan Naufal Sudrajat / 13518021
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13518021@std.stei.itb.aci.id

***Abstract*—Wordscapes Search merupakan sebuah permainan berupa teka - teki kata. Pemain diberikan sebuah tabel berisi huruf huruf serta daftar kata. Pemain diharuskan menemukan kata kata pada daftar kata pada tabel huruf. Kata dapat dibentuk oleh daftar huruf dalam barisan mendatar (Horizontal) berurut ke depan , mendatar berurut ke belakang, menurun (Vertical) berurut ke depan, menaik berurut ke belakang, menyamping (Diagonal) ke kanan , dan diagonal ke kiri.**

Keywords—Brute Force ;* Teka - Teki*; Wordscapes Search; Word; Strategi Algoritma

I. PENDAHULUAN

Permainan teka - teki mencari pola merupakan salah satu game yang terkenal dan sering dimainkan oleh banyak orang. Permainan ini dapat dimainkan oleh semua umur dan tidak diperlukan kemampuan pemain yang tinggi. Permainan berjenis ini seringkali hanya membutuhkan ketelitian dan ketekunan sehingga semua orang dapat memainkannya. Pemain biasanya memainkan permainan ini untuk mengisi waktu kosongnya, atau juga dimainkan saat menunggu antrian, atau saat sedang berpergian. Contoh contoh permainan teka - teki seperti ini adalah teka - teki mencari pola warna (Candy Crush), mencari pola gambar (Onet), dan mencari pola kata (Wordscapes).

Wordscapes Search merupakan permainan berbasis android yang dikembangkan oleh Peoplefun. Pemain diberi sebuah daftar kata yang meiliki satu tema dan diminta untuk mencari kata kata tersebut pada daftar huruf yang disusun menjadi tabel huruf berbentuk matriks M x N. Jumlah kata - kata yang diberikan dan panjang kata bervariasi sesuai dengan tingkat kesulitan level teka teki. Pada level rendah hanya diberikan 6 buah kata dengan panjang 4 hingga 6 huruf, sedangkan pada level yang lebih tinggi dapat diberikan 10 kata dengan panjang 5 hingga 8 huruf.

Kata yang terdapat pada tabel huruf dapat berbentuk beurut dengan barisan horizontal , vertikal dan diagonal. Urutan huruf pada kata juga dapat dimulai dari huruf pertama berurut ke huruf terakhir atau pun sebaliknya. Contoh apabila diminta suatu kata "MINUTE", maka pola kata pada tabel huruf dapat berbentuk "M I N U T E" atau "E T U N I M"

Gambar 1. Tampilan Permainan Wordscapes Search

Pada saat memainkan permainan ini penulis membutuhkan waktu 1 hingga 3 menit dalam menyelesaikan satu teka - teki level rendah dan lebih dari 4 menit untuk menyelesaikan teka teki level tinggi. Pada saat makalah ini ditulis terdapat 15000 teka - teki yang ada pada aplikasi, sehingga apabila permainan ini ingin diselesaikan diperlukan waktu sekitar sebanyak 31 hari penuh tanpa henti. Dikarenakan dibutuhkan waktu yang sangat banyak untuk menyelesaikan game maka penulis memutuskan untuk membuat program yang dapat memudahkan penyelesaian permainan ini.

II. LANDASAN TEORI

A. Algoritma Brute Force

Algoritma Brute Force atau *exhaustive search* merupakan algortima dengan pendekatan yang lempang (*straightforward*) untuk memecahkan suatu persoalan. Algoritma ini didasarkan langsung pada pernyataan persoalan. Sehingga algoritma ini dapat dibilang sangat sederhana dan jelas caranya.

Algoritma Brute force memiliki karakteristik dapat menyelesaikan hampir semua problem, dan mencoba semua kemungkinan hingga ditemukan solusi dari masalah. Berikut salah satu contoh pengaplikasian algoritma *brute force* dalam memecahkan pengurutan menggunakan metode Buble Sort.

```
procedure BubbleSort (input/output s1, s2, ..., sn : integer, input n : integer)
{Mengurutkan s1, s2, ..., sn sehingga terurut menaik dengan metode pengurutan bubble sort.
Masukan: s1, s2, ..., sn
Luaran: s1, s2, ..., sn (terurut menaik) }
Deklarasi
  i    : integer   { pencacah untuk jumlah langkah }
  k    : integer   { pencacah,untuk pengapungan pada setiap langkah }
  temp : integer   { peubah bantu untuk pertukaran }

Algoritma:
  for i ← n – 1 downto 1 do
    for k ← 1 to i do
      if s[k+1] < s[k] then
        {pertukarkan s[k] dengan s[k+1]}
        temp ← s[k]
        s[k] ← s[k+1]
        s[k+1] ← temp
      endif
    endfor
  endfor
```

Gambar 2. Algoritma Brute Force pada Buble sort

Sumber : https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Brute-Force-(2021)-Bag1.pdf

Dikarenakan *Algoritma Brute Force* sangat *straightforward* , algoritma ini memiliki kinerja yang lambat dan memerlukan daya komputasi yang relatif besar. Kinerjanya dapat diperbaiki dengan menerapkan fungsi *heuristic* agar tidak diperlukan pencarian yang kemungkinan solusi secara penuh.

Heuristik mengacu pada teknik memecahkan persoalan berbasis pengalaman ,dari proses pembelajaran , dan penemuan solusi meskipun tidak dijamin optimal. Heuristic yang baik diharapkan dapat mengurangi daya komputasi dari algoritma *Brute Force.*

III. IMPLEMENTASI PROGRAM

A. Pengolahan Input

C R I S P Y J
R T A S T Y R
I Y S A E E K
T A S T M A R
S O E M S K O
Y G I P D A P
M S C H O P F

Gambar 3. Contoh daftar huruf input

Input akan dibaca oleh program dan disiapkan menjadi sebuah List of string , selain itu juga disiapkan sebuah tabel solusi yang berupa matriks seukuran daftar kata input yang akan digunakan untuk menampilkan jawaban.

Berikut merupakan implementasi dari pengolahan dan pembacaan input dan pembuatan tabel solusi:

```
def proceed_input(self):
"""
Membaca puzzle_file dan merubahnya menjadi list of string
"""
    with open(self.puzzleloc,"r") as puzzle_file:
        for lines in puzzle_file:
            self.puzzle.append(lines.strip("\n"))
        if (self.puzzle):
            self.shapes = (len(self.puzzle), len(self.puzzle[0]))
    self.createemptysolution()
```

```
def createemptysolution(self):
"""
Membuat tabel solusi sebesar tabel daftar kata
"""
    solution  = []
    lines = []
    for i in range(0,self.shapes[1]):
        lines.append("-")
    for j in range(0,self.shapes[0]):
        solution.append(lines.copy())
    self.solution = solution
```

B. Pencarian Kata pada daftar Huruf

Pencarian dilakukan dengan mengiterasi semua huruf pada daftar huruf. Setiap hurufnya akan dilakukan pencarian dengan iterasi dari posisi huruf 8 arah yaitu atas kiri, atas , kanan atas , kanan, kanan bawah, bawah, serta bawah kiri.
Pada metode pencarian ini digunakan fungsi heuristic yaitu :

- Solusi akan selalu terdapat pada daftar huruf
  Fungsi ini mengurangi iterasi untuk mengecek apabila solusi tidak ada pada input.
- Hanya terdapat satu solusi kata pada tiap daftar input
  Fungsi ini menjamin bahwa solusi pertama yang didapat merupakan solusi sebenarnya, dan tidak diperlukan iterasi untuk mencari solusi lain.
- Apabila pada barisan suatu arah panjangnya lebih pendek dari panjang kata maka tidak perlu diperiksa dikarenakan dipastikan bukan solusi.

Gambar 4. Pencarian arrow merah tidak perlu dilakukan

Fungsi heuristic ini memastikan iterasi tidak perlu dilakukan apabila posisi huruf ke arah barisan tidak memenuhi panjang huruf.

Ide dari iterasi pencarian adalah sebagai berikut untuk setiap huruf pada daftar tabel maka apabila huruf tersebut merupakan huruf awal dari kata yang dibutuhkan. Akan dilakukan iterasi 8 kali dari posisi huruf sesuai arah yang diberikan sebelumnya. Sebelum dilakukan iterasi diperiksa terlebih dahulu apakah kata akan dapat masuk ke arah iterasi tersebut. Contoh apabila arah iterasi ke samping kiri akan diperiksa apakah banyak huruf disamping huruf awal tersebut lebih banyak atau sama dengan panjang kata yang dibutuhkan (dikurangi huruf pertama). Apabila pengetesan berhasil maka kata dianggap fit dan iterasi akan dilanjutkan. Apabila pengetesan gagal maka akan dilanjutkan menggunakan arah lainnya.

Apabila pengetesan berhasil maka akan dilanjutkan dengan mengetes tiap huruf yang diiterasi sesuai arah merupakan huruf yang diinginkan apabila terdapat huruf yang berbeda maka iterasi diberhentikan dan akan dilanjutkan iterasi arah yang baru. Pengetesan akan berhasil apabila setiap huruf iterasi sepanjang kata merupakan sama dengan huruf pada kata.

Apabila setelah semua iterasi arah pada suatu posisi huruf awal belum berhasil memenuhi syarat. Maka akan digunakan posisi huruf awal kata lainnya. Pada akhirnya apabila didapat suatu posisi dan arah yang melewati semua tes. Maka Iterasi akan diberhentikan.

Berikut merupakan implementasi program untuk menentukan posisi dari kata dan apakah iterasi ke suatu arah perlu dilakukan atau tidak

```
def search(self,pos,word):
    for i in range(0,8):
        found = True
        if(self.iswordfit(len(word),pos,i)):
            dir_downward = self.dir_downward[i]
            dir_leftward = self.dir_leftward[i]
            for j in range(0,len(word)):
                row_pos = pos[0] + j*dir_downward
                col_pos = pos[1] + j*dir_leftward
                if(word[j] != self.puzzle[row_pos][col_pos]):
                    found = False
                    break
            if(found):
                self.makesolution(len(word),pos,i)
                return found
            return False
```

```
def iswordfit(self,lenght_word,pos,direction):
    end_pos_row = (lenght_word -1) *
                  self.dir_downward[direction] +
                  pos[0]
    if (end_pos_row < 0 or end_pos_row >= self.shapes[0]):
        return False
```

```
    end_pos_col = (lenght_word -1) *
                    self.dir_leftward[direction] +
                    pos[1]
    if (end_pos_col < 0 or end_pos_col >= self.shapes[1]):
      return False
  return True
```

Apabila ditemukan suatu solusi iterasi akan diberhentikan dan akan di temukan sebuah posisi huruf awal solusi dan arah iterasi jawaban. Kedua hal ini beserta dengan panjang kata sebagai jumlahnya iterasi akan digunakan sebagai cara untuk menampilkan solusi.

C. *Membuat Solusi dan Menampilkan solusi*

Solusi ditampilkan dalam bentuk tabel solusi yang sudah dipersiapkan sebelumnya. Berbentuk Matriks M x N yang akan berisi kata yang diminta dan simbol "-" menggambarkan huruf lain pada daftar huruf yang bukan merupakan solusi. Solusi dibuat agar *user* dapat dengan mudah mengambil solusi.

Gambar 5. Contoh tampilan hasil solusi

```
def makesolution(self,lenword,pos,direc):
    i = 0
    while(i<lenword):
      row_pos = pos[0] +  i*self.dir_downward[direc]
      col_pos = pos[1] +  i*self.dir_leftward[direc]
      self.solution[row_pos][col_pos]= self.puzzle[row_pos][col_pos]
      I += 1
```

```
def printsolution(self):
    for i in range(0,self.shapes[0]):
      for j in range(0,self.shapes[1]):
        print(self.solution[i][j]+" ", end = "")
          print() for j in range(0,self.shapes[0]):
        solution.append(lines.copy())
    self.solution = solution
```

IV. PENGUJIAN

A. *Program Uji*

Program yang dibuat oleh penulis secara umum menerapkan algoritma berikut :

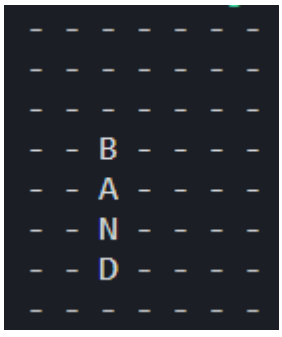
1. Membaca data masukkan berupa daftar huruf.
2. Meminta input pengguna berupa kata yang ingin diberikan posisinya.
3. Melakukan algoritma pencarian dan menyimpan hasil pencarian.
4. Menampilkan hasil pencarian

Berikut merupakan tampilan dari level permainan *Wordscape Search* yang akan digunakan sebagai uji :

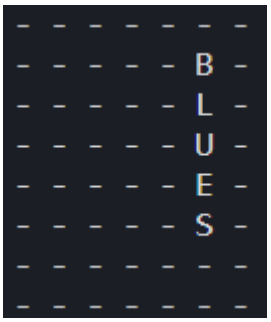
α +

B. *Hasil Uji*

1. Band

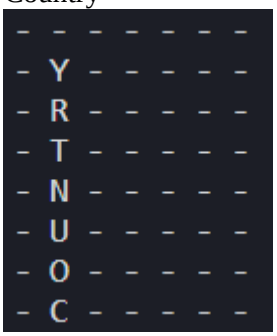


2. Blues

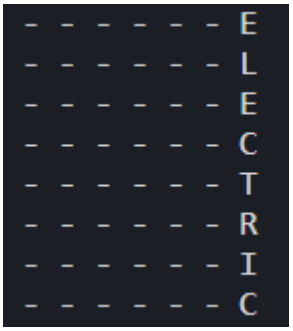


3. Chords

4. Country



5. Electric



C. *Tabel Waktu Pemrosesan*

Pengujian dilakukan pada laptop penulis dengan spesifikasi sebagai berikut:

- Operating System: Windows 10 Pro 64-bit
- Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz (8 CPUs), ~1.8GHz
- Memory: 8192MB RAM

TABLE I. TABEL WAKTU PEMROSESAN

Kata	Waktu Pemprosesan
Band	0.00012
Blues	0.00014
Chords	0.00015
Country	0.00023
Electric	0.00024
Folk	0.00021
Frets	0.00025
String	0.00020
Strum	0.00025

V. KESIMPULAN

Berdasarkan hasil pengamatan waktu pemproresan tabel 1, didapatkan kesimpulan bahwa menggunkan algoritma *Brute-Force* menggunakan fungsi heuristic dapat didapatkan daya komputasi yang tidak terlalu besar dengan waktu pemrosesan yang sedikit. Meskipun demikian, waktu pemrosesan tersebut dapat hanya terlihat cepat dikarenakan kecepatan prosesor penulis. Diperlukan sebuah pengaplikasian algoritma lainnya agar dapat dibandingkan *apple to apple* dan didapatkan waktu pemprosesan yang benar benar efektif.

UCAPAN TERIMAKASIH

Rasa syukur saya ucapkan kepada Allah SWT yang telah memberikan berkat, rahmat, dan karunia-Nya sehingga saya dapat menyelesaikan pembuatan makalah ini di tengah pandemi COVID-19 saat ini. Selanjutnya ucapan terima kasih. Ucapan terimakasih pula kepada para tim dosen Strategi Algoritma yang telah memberikan saya kesempatan untuk dapat melakukan eksplorasi menambah pengetahuan. Tak lupa pula kepada Dosen Prof. Ir. Dwi Hendratmo Widyantoro, M.Sc., Ph.D. yang telah mengaja di kelas K3 dengan penuh dedikasi. Saya harap makalah ini semoga dapat memberikan manfaat yang baik bagi kepada penulis dan juga masyarakat.

REFERENSI

[1] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Brute-Force-(2021)-Bag1.pdf diakses pada 10 Mei 2021

[2] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Brute-Force-(2021)-Bag2.pdf diakses pada 10 Mei 2021

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

Zaidan Naufal Sudrajat