Perbandingan Algoritma *Brute Force* dan *Backtracking* dalam Permainan *Word Search Puzzle*

Veren Iliana Kurniadi 13515078
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13515078@std.stei.itb.ac.id

***Abstract*—Permainan *word search puzzle* adalah permainan mencari kata dalam kumpulan huruf. Permainan ini bisa diselesaikan dengan berbagai strategi, seperti algoritma *brute force* dan *backtracking*. Makalah ini berisi pembahasan mengenai algoritma *brute force* dan *backtracking* serta penggunaannya dalam penyelesaian *word search puzzle*. Selain itu akan dibahas pula perbandingan algoritma *brute force* dan *backtracking* dalam menyelesaikan permainan *word search puzzle* berdasarkan analisis dari program yang dibuat penulis untuk mencari kata dalam matriks huruf.**

Kata kunci—brute force; word search; algoritma; backtracking.

I. PENDAHULUAN

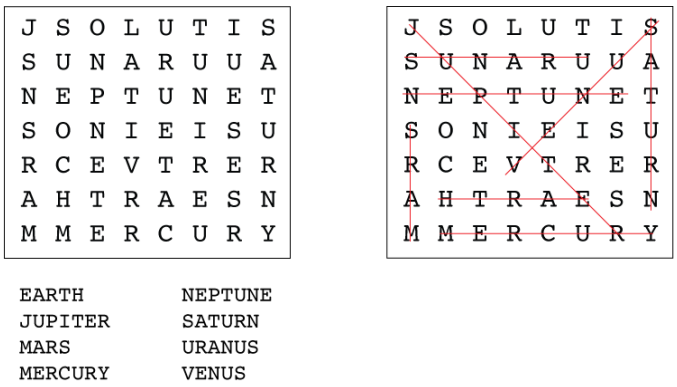
Permainan kata bukanlah hal yang sulit dijumpai saat ini. Permainan kata dapat ditemukan dalam berbagai bentuk yang menarik, seperti teka teki silang, *word search puzzle*, dan *scrabble*. Salah satu permainan kata yang cukup populer adalah *word search puzzle*. *Word search puzzle* dapat ditemukan di surat kabar atau majalah. Seiring dengan semakin banyaknya buku yang dibuat sebagai bacaan anak-anak, permainan ini juga dapat ditemukan di buku *puzzle* atau permainan untuk anak-anak.

Word search puzzle dimainkan dengan cara mencari kata tertentu dalam kumpulan huruf acak atau bisa disebut juga matriks huruf. Permainan kata seperti *word search puzzle* diminati tidak hanya oleh anak-anak, tapi juga segala usia. Permainan yang sederhana namun dapat mengasah otak membuat *word search puzzle* menjadi permainan yang menarik. Selain sebagai sarana hiburan, *word search puzzle* seringkali digunakan untuk tujuan edukasi, seperti membantu anak-anak dalam mengingat kata atau mengeja kata baru yang sedang dipelajari.

Salah satu cara untuk mencari kata dalam matriks huruf adalah dengan menggunakan algoritma *brute force*. Algoritma yang terbilang mudah ini seringkali digunakan orang untuk menyelesaikan suatu persoalan. Manusia biasanya secara alamiah akan menggunakan algoritma *brute force* untuk menyelesaikan persoalan baru yang ditemuinya. Selain *brute force*, algoritma *backtracking* juga dapat menjadi pilihan untuk mencari kata dalam matriks huruf. Dalam makalah ini, penulis akan membahas perbandingan algoritma *brute force* dan *backtracking* saat digunakan untuk mencari kata pada permainan *word search puzzle*.

II. WORD SEARCH PUZZLE

Word search puzzle pertama kali dibuat oleh Noeman E. Gibat. Puzzle ini dipublikasikan dalam the Selenby Digest pada 1 Maret 1968 di Norman, Oklahoma. Setelah itu *word search puzzle* mulai terkenal di daerahnya. Kemudian beberapa guru meminta salinannya untuk digunakan sebagai alat mengajar. Akhirnya permainan itu tersebar setelah seorang guru mengirimkan salinannya kepada teman-teman di kota lain.[3] Sekarang *word search puzzle* bisa ditemukan dengan mudah di koran, majalah, bahkan internet.



Gambar 1 – Word Search Puzzle (sumber : https://www.gmpuzzles.com/blog/word-search-rules-and-info/)

Word search puzzle adalah permainan kata dimana pemain harus menemukan beberapa kata tersembunyi dalam kumpulan huruf acak. Kumpulan huruf tersebut biasa diletakkan pada "papan" berbentuk segi empat atau dapat disebut juga matriks huruf. Kata-kata pada matriks huruf ini dapat ditemukan dengan delapan arah yang mungkin, yaitu :[2]

1. Vertikal ke atas
2. Vertikal ke bawah
3. Horizontal ke kanan
4. Horizontal ke kiri
5. Diagonal ke kanan atas
6. Diagonal ke kanan bawah
7. Diagonal ke kiri atas
8. Diagonal ke kiri bawah

Word search puzzle dapat diselesaikan dengan berbagai macam strategi algoritma, seperti *brute force, backtracking, breadth first search, depth first search,* dan *string matching*.

III. ALGORITMA BRUTE FORCE

A. Definisi

Brute force adalah sebuah pendekatan yang lempang (*straightforward*) untuk menyelesaikan suatu masalah berdasarkan pernyataan masalah dan definisi konsep yang dilibatkan. Algoritma *brute force* memecahkan masalah dengan sangat sederhana, langsung, dan dengan cara yang jelas (*obvious way*).[1]

B. Karakteristik

Algoritma *brute force* umumnya tidak "cerdas" dan tidak efektif, karena membutuhkan jumlah langkah yang besar dalam penyelesaiannya. Kadang-kadang algoritma *brute force* disebut juga algoritma naif (*naïve algorithm*).[1]

Algoritma *brute force* seringkali merupakan pilihan yang kurang disukai karena ketidakefektifannya, tetapi dengan mencari pola-pola yang mendasar, keteraturan, atau trik-trik khusus, biasanya akan membantu kita menemukan algoritma yang lebih cerdas dan lebih efektif. Algoritma *brute force* sering digunakan sebagai basis bila membandingkan beberapa alternatif algoritma yang efektif. [1]

Meskipun *brute force* bukan merupakan teknik pemecahan masalah yang efektif, namun teknik ini dapat diterapkan pada sebagian besar masalah. Bahkan ada masalah yang hanya dapat diselesaikan dengan teknik *brute force*. Algoritma *brute force* seringkali lebih mudah diimplementasikan daripada algoritma yang lebih canggih. [1]

C. Kekuatan dan Kelemahan

Kekuatan :

1. Dapat digunakan untuk memecahkan hampir sebagian besar masalah (*wide applicability*).
2. Sederhana dan mudah dimengerti.
3. Menghasilkan algoritma yang layak untuk beberapa masalah penting seperti pencarian, pengurutan, pencocokan string, perkalian matriks.
4. Menghasilkan algoritma baku (*standard*) untuk tugas komputasi sederhana seperti penjumlahan/perkalian n buah bilangan, menentukan elemen minimum atau maksimum di dalam list.[4]

Kelemahan :

1. Jarang menghasilkan algoritma yang efisien.
2. Beberapa algoritma *brute force* lambat sehingga tidak dapat diterima.
3. Tidak sekonstruktif atau sekreatif teknik pemecahan masalah lainnya.[4]

D. Sequential Search

Salah satu contoh penyelesaian masalah menggunakan algoritma *brute force* adalah sequential search. Sequential search adalah pencarian yang dilakukan secara terurut. Metode pencarian ini digunakan dalam penyelesaian *word search puzzle* yang akan dijelaskan pada bagian selanjutnya. [1]

Misalkan diberikan n buah bilangan bulat yang dinyatakan sebagai a_1, a_2, …, a_n. Carilah apakah x terdapat di dalam himpunan bilangan bulat tersebut. Jika x ditemukan, maka lokasi (indeks) elemen yang bernilai x disimpan di dalam peubah idx. Jika x tidak terdapat di dalam himpunan tersebut, maka idx diisi dengan nilai 0.[1]

Algoritma *brute force* : Setiap elemen di dalam himpunan dibandingkan dengan x mulai dari elemen pertama, a1, sampai ditemukan elemen bernilai sama (pencarian berhasil) atau sampai semua elemen sudah habis diperiksa (pencarian gagal). Jika a_i=x, maka i adalah lokasi tempat x berada (idx diisi dengan i). Jika x tidak ditemukan, maka idx diisi 0. [1]

IV. ALGORITMA BACKTRACKING

A. Dasar Teori

Backtracking adalah algoritma yang berbasis pada DFS (*Depth First Search*) untuk mencari solusi persoalan. Backtracking yang merupakan perbaikan algoritma *brute force*. Dengan metode ini, kita tidak perlu memeriksa semua kemungkinan solusi yang ada. Hanya pencarian yang mengarah ke solusi saja yang selalu dipertimbangkan. Akibatnya, waktu pencarian dapat dihemat. Backtracking lebih alami dinyatakan dalam algoritma rekursif. [1]

B. Properti Umum

Untuk menerapkan metode *backtracking*, property berikut didefinisikan:

1. Solusi persoalan
   Solusi dinyatakan sebagai vector dengan n-tuple:
   $X = (x_1,x_2,\ldots,x_n)$, $x_i \in S_i$, Mungkin saja $S_1=S_2=S_n$
2. Fungsi pembangkit nilai x_k
   Dinyatakan sebagai T(k)
3. Fungsi pembatas
   Dinyatakan sebagai $B(x_1,x_2,\ldots,x_i)$ [1]

C. Prinsip Pencarian Solusi

Langkah-langkah pencarian solusi pada pohon ruang status yang dibangun secara dinamis adalah sebagai berikut:

1. Solusi dicari dengan membentuk lintasan dari akar ke daun. Aturan pembentukan menggunakan metode DFS. Simpul-simpul yang sudah dilahirkan dinamakan simpul hidup. Simpul hidup yang sedang diperluas dinamakan simpul-E. Simpul dinomori dari atas ke bawah sesuai urutan kelahirannya.
2. Tiap kali simpul-E diperluas, lintasan yang dibangun bertambah panjang. Jika lintasan yang sedang dibentuk tidak mengarah ke solusi maka simpul-E tersebut "dibunuh" sehingga menjadi simpul mati. Fungsi yang digunakan untuk membunuh simpul-E adalah fungsi pembatas.

3. Jika pembentukan lintasan berakhir dengan simpul mati, maka proses pencarian diteruskan dengan membangkitkan simpul anak yang lainnya. Bila tidak ada lagi simpul anak yang dapat dibangkitkan, maka pencarian dilanjutkan dengan melakukan runut balik ke simpul hidup terdekat (simpul orangtua). Selanjutnya simpul ini menjadi simpul-E yang baru. Lintasan baru dibangun kembali sampai membentuk solusi.
4. Pencarian dihentikan bila solusi telah ditemukan atau tidak ada lagi simpul hidup untuk runut balik. [1]

D. Skema Umum

Dibawah ini diberikan skema umum algoritma *backtracking* versi iteratif karena versi ini yang akan digunakan dalam program uji. [1]

```
procedure Backtracking(input n:integer)
{ Mencari semua solusi persoalan dengan metode backtracking
  Masukan: n, yaitu panjang vector solusi
  Keluaran: solusi x = x[1],x[2],…,x[n]
}
Deklarasi
  k : integer
Algoritma
  k ← 1
  while k > 0 do
    if (x[k] belum dicoba sedemikian sehingga x[k] ← T(k))
        and (B(x[1],x[2],…,x[k])=true) then
      if (x[1],x[2],…,x[k]) adalah lintasan
          dari akar ke daun then
        CetakSolusi(x)
      endif
      k ← k+1
    else
      k ← k-1
    endif
  endwhile
  {k = 0}
```

V. PENGUJIAN DAN ANALISIS

Pengujian dilakukan dengan program yang dibuat oleh penulis. Tujuan pengujian ini adalah untuk membantu penyelesaian permainan *word search puzzle*. Program dibuat untuk mencari kata pada matriks huruf dengan menggunakan algoritma *brute force* dan *backtracking*. Untuk selanjutnya, kata yang dicari akan disebut *pattern*. Pengujian dilakukan hanya sampai solusi ditemukan.

A. Program uji

Program yang dibuat penulis untuk pengujian secara umum menggunakan algoritma sebagai berikut :

1. Mencari huruf pertama *pattern* pada matriks huruf dimulai dari indeks terakhir +1. (Pada pencarian awal, indeks diinisialisasi dengan 1,0)
2. Jika huruf pertama ditemukan, menelusuri delapan arah dari posisi huruf tersebut untuk menemukan kata yang cocok.
3. Jika huruf pertama tidak ditemukan, program selesai (pencarian gagal).
4. Jika tidak ditemukkan kata yang cocok, kembali ke langkah 1.
5. Jika ditemukan kata yang cocok, program selesai (pencarian berhasil).

Masukan program berupa :

1. File eksternal berisi jumlah baris dan kolom matriks huruf, isi matriks huruf.

Gambar 2 – File eksternal yang digunakan dalam pengujian

2. Kata yang dicari (*pattern*) dari masukan pengguna.

Keluaran program berupa :

1. Langkah penelusuran
2. Indeks huruf pertama ditemukan (baris,kolom)
3. Arah kata
4. Waktu eksekusi algoritma

B. Penyelesaian Word Search Puzzle

Kata yang digunakan pada pengujian ini adalah “VENUS”. Langkah pertama yang dilakukan untuk menyelesaikan *word search puzzle* adalah mencari huruf pertama dari *pattern* pada matriks huruf. Pencarian ini menggunakan *sequential search*. Langkah pertama ini dilakukan baik pada algoritma *brute force* ataupun algoritma *backtracking*. Pada pengujian ini, *sequential search* dilakukan sampai ditemukan “V” pada indeks 2,10.

A W X Q N B C U B S P D J N L P M I G R A N
E Y E H Y O Y K E V K B I F T Q K Y A V S Q
X K D X T E K B O E V T H I W B Z M B E T F
T Z Q H O G L D Z N N T R A P P I S T W R S
R K A H M P N O C U T R R D U L S F Y N O P
A Q L F X O L I H S G K X W U M O U B U N C
T N P S M A T A H K H H A T P I R J Z S O N
E J H B E A K I N P C A B Q L L N F O J M L
R S A L U B E N I E S A K G Q K R O M I Y E
R Y O D H P B C F W T B L A T Y L C C H E W
E F W A J K C I B N V C H B J W M A Q L H P
S T A R K W L S X G O W J A Y A Y A X T I U
T S O W R O N G M C S N U B O Y P T R H N R
R U A P X R U A A V E E F T N N M A Z J B P
I T O E W R R X P Z V X T Y J I E Q M F X O
A D I Y H S B Q I J E K N I V H K E A L V L
L O M D G S O L A R S Y S T E M U C K T B N

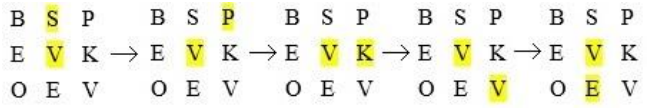
Gambar 3 – langkah 1 pencarian huruf pertama “V”

Langkah berikutnya setelah indeks pertama ditemukan adalah menelusuri delapan arah dari posisi huruf tersebut untuk menemukan kata yang cocok. Pencocokan yang dilakukan oada Misalkan panjang *pattern* n dan keluaran berupa integer yang menunjukkan arah (1: atas, 2: kanan atas, 3: kanan, 4: kanan bawah, 5: bawah, 6: kiri bawah, 7: kiri, 8: kiri atas), algoritma *brute force* adalah sebagai berikut. (k diinisialisasi 1)

1. Periksa pada jalur k apakah huruf ke-2 cocok dengan *pattern*.
2. Jika k=8 (semua jalur sudah habis diperiksa, pencarian gagal), arah diisi -1.
3. Jika pada jalur k, huruf ke-2 tidak cocok dengan *pattern*, kembali ke langkah 1 dengan k+1.
4. Jika pada jalur k, huruf ke-2 cocok, periksa pada jalur k huruf ke-3 sampai ke-n apakah cocok dengan *pattern*.
5. Jika pada jalur k, ada huruf ke-i dimana 2<i<=n tidak cocok dengan *pattern*, kembali ke langkah 1 dengan k+1.
6. Jika pada jalur k, huruf ke-3 sampai ke-n cocok dengan *pattern* (pencarian berhasil), maka arah diisi dengan k (integer yang sesuai).

Algoritma *backtracking* untuk menelusuri delapan jalur pada matriks huruf tidak berbeda jauh dengan algoritma *brute force*, namun penelusurannya dibatasi. Jika dengan *brute force* penelusuran dilakukan ke semua arah, dengan *backtracking* penelusuran hanya dilakukan jika jumlah huruf pada jalur yang sedang diproses lebih dari panjang *pattern*. Berikut ini *pseudo code* penelusuran delapan arah pada matriks huruf dengan algoritma *backtracking*.

Untuk mencari "VENUS", setelah mendapat indeks "V" dilakukan penelusuran 8 arah dengan algoritma di atas. Hal yang dilakukan pertama saat penelusuran arah adalah mencari huruf ke-2 yang sama dengan "E". Jika huruf ke-2 tidak cocok, penelusuran berlanjut ke arah berikutnya. Berikut ini penelusuran arah dengan *brute force*.

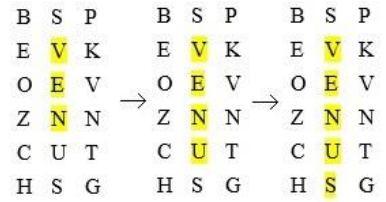


Gambar 4 – langkah 2 brute force (zoom in)

Pada pencarian huruf "E" dengan *backtracking*, arah 1 dan 2 tidak diproses karena jumlah huruf dari indeks huruf pertama ke ujung matriks melalui jalur 1 dan 2 < panjang *pattern*.

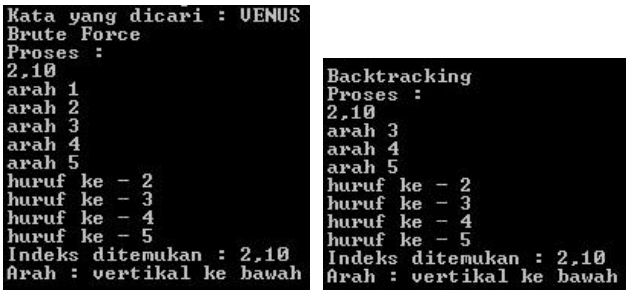
Gambar 5 – langkah 2 backtracking (zoom in)

Jika ditemukan huruf ke-2 yang cocok, lanjutkan pencocokan *pattern* ke huruf ke-3 dan seterusnya. Jika tidak, kembali ke langkah1 yaitu mencari huruf yang cocok dengan huruf pertama. Untuk pencocokan huruf ke-3 sampai ke-n pada pengujian ini, penelusuran sama untuk kedua algoritma.



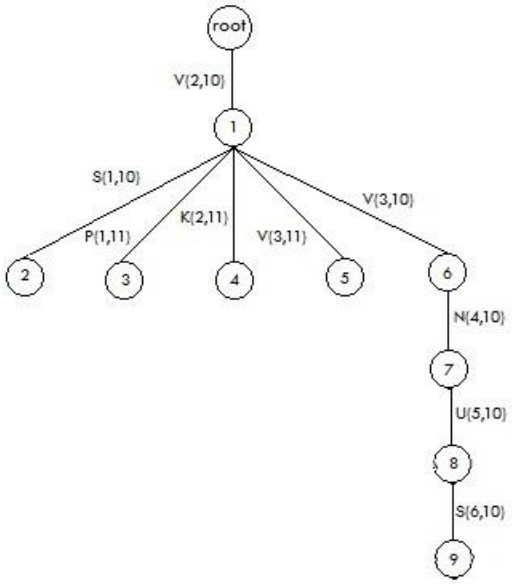
Gambar 6 – langkah 3 pencocokan huruf ke-3 sampai ke-n (zoom in)

Jika huruf ke-3 sampai ke-n cocok dengan *pattern*, pencarian berhasil, program selesai.



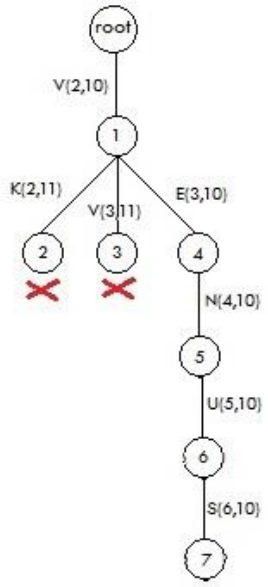
Gambar 7 – output program pencarian "VENUS"

Pada penyelesaian *word search puzzle* ini, simpul pada pohon ruang status dapat dianalogikan sebagai lokasi huruf (indeks matriks) yang diperiksa. Pohon ruang status untuk algoritma *brute force* dapat digambarkan sebagai berikut.



Gambar 8 – pohon ruang status brute force

Pohon ruang status untuk algoritma *backtracking* dapat digambarkan sebagai berikut.



Gambar 9 – pohon ruang status backtracking

C. Analisis

Berikut ini adalah proses penyelesaian *word search puzzle* dengan input kata "VENUS".

Algoritma	***Indeks huruf pertama***	***Indeks yang sedang diproses***	***Keterangan***
Brute Force	2,10	1,10	Huruf ke-2 tidak cocok
		1,11	Huruf ke-2 tidak cocok
		2,11	Huruf ke-2 tidak cocok
		3,11	Huruf ke-2 tidak cocok
		3,10	Huruf ke-2 cocok
		4,10	Huruf ke-3 cocok
		5,10	Huruf ke-4 cocok
		6,10	Huruf ke-5 cocok
		Jumlah simpul	9
Backtracking	2,10	2,11	Huruf ke-2 tidak cocok
		3,11	Huruf ke-2 tidak cocok
		3,10	Huruf ke-2 cocok
		4,10	Huruf ke-3 cocok
		5,10	Huruf ke-4 cocok
		6,10	Huruf ke-5 cocok
		Jumlah simpul	7

Tabel 1 – Tabel proses pencarian "VENUS"

Dengan metode *brute force*, penelusuran dilakukan dengen memeriksa semua kemungkinan yang ada. Sedangankan dengan metode *backtracking*, penelusuran dilakukan setelah melalui fungsi pembatas. Fungsi pembatas ini memeriksa apakah jumlah huruf pada jalur yang sedang diproses lebih dari panjang *pattern*. Jika jumlah huruf lebih dari panjang *pattern*, fungsi pembatas dapat dilewati(accept). Karena penelusuran dilakukan hanya jika telah melewati fungsi pembatas, langkah penelusuran yang dilakukan jadi lebih sedikit. Dari pohon ruang status dapat dilihat untuk algoritma *brute force* ada 9 simpul yang dibangkitkan, sedangkan untuk *backtracking* hanya 7 simpul yang dibangkitkan. Hal ini tentunya membuat waktu yang dibutuhkan untuk menjalankan algoritma *backtracking* lebih sedikit dibanding algoritma *brute force*.

Untuk mendukung hasil pengujian sebelumnya, dilakukan pengujian kembali dengan kasus yang sedikit bereda. Bila pada kasus sebelumnya kata yang dicari langsung ditemukan, pada kasus ini perlu lebih dari sekali menjalankan sequential search untuk sampai pada solusi. Kata yang dicari : "EXOPLANET".

Algoritma	***Indeks huruf pertama***	***Indeks yang sedang diproses***	***Keterangan***
Brute Force	2,1	1,1	Huruf ke-2 tidak cocok
		1,2	Huruf ke-2 tidak cocok
		2,2	Huruf ke-2 tidak cocok
		3,2	Huruf ke-2 tidak cocok
		3,1	Huruf ke-2 cocok
		4,1	Huruf ke-3 tidak cocok
	2,3	1,3	Huruf ke-2 tidak cocok
		1,4	Huruf ke-2 tidak cocok
		2,4	Huruf ke-2 tidak cocok
		3,4	Huruf ke-2 cocok
		4,5	Huruf ke-3 cocok
		5,6	Huruf ke-4 cocok
		6,7	Huruf ke-5 cocok
		7,8	Huruf ke-6 cocok
		8,9	Huruf ke-7 cocok
		9,10	Huruf ke-8 cocok
		10,11	Huruf ke-9 cocok
		Jumlah smpul	19
Backtracking	2,1	2,2	Huruf ke-2 tidak cocok
		3,2	Huruf ke-2 tidak cocok
		3,1	Huruf ke-2 cocok
		4,1	Huruf ke-3 tidak cocok
	2,3	2,4	Huruf ke-2 tidak cocok
		3,4	Huruf ke-2 cocok
		4,5	Huruf ke-3 cocok
		5,6	Huruf ke-4 cocok
		6,7	Huruf ke-5 cocok
		7,8	Huruf ke-6 cocok
		8,9	Huruf ke-7 cocok
		9,10	Huruf ke-8 cocok
		10,11	Huruf ke-9 cocok
		Jumlah smpul	15

Tabel 2 – Tabel proses pencarian "EXOPLANET"

Pada pencarian "E" pertama (indeks 2,1) tidak ditemukan kata yang cocok sehingga harus kembali dilakukan sequential search untuk mencari "E" yang lain (indeks 2,3). Selain itu, prosesnya sama seperti sebelumnya. Dari pengujian ini juga didapatkan hasil bahwa simpul yang dibangkitkan menggunakan algoritma *brute force* lebih banyak dibanding simpul yang dibangkitkan dengan algoritma *backtracking*. Hal ini menunjukkan algoritma *brute force* membutuhkan lebih banyak waktu untuk menyelesaikan permainan *word search puzzle* dibandingkan *backtracking*.

VI. KESIMPULAN

Algoritma *brute force* dan algoritma *backtracking* merupakan beberapa strategi sederhana yang dapat digunakan untuk menyelesaikan permainan *word search puzzle*. Penyelesaian dengan algoritma *brute force* ataupun *backtracking* tetap menggunakan metode sequential search terlebih dulu untuk mencari huruf pertama *pattern* yang cocok dengan huruf pada matriks huruf. Algoritma *brute force* lebih cocok digunakan untuk pencarian kata secara manual karena lebih mudah. Namun untuk pencarian kata dengan mesin, algoritma *backtracking* lebih cocok digunakan karena lebih efisien. Algoritma *brute force* membutuhkan waktu lebih banyak untuk mencari kata dalam matriks huruf dibandingkan algoritma *backtracking*.

VII. PENUTUP

Penulis mengucap syukur pada Tuhan YME untuk berkat dan rahmatnya sehingga penulis dapat menyelesaikan makalah ini. Penulis juga mengucapkan terima kasih kepada Bapak Dr. Ir. Rinaldi Munir, M.T sebagai dosen pengajar mata kuliah Strategi Algoritma yang telah memberi bimbingan selama satu semester ini sehingga penulis dapat membuat makalah ini. Semoga makalah ini dapat bermanfaat bagi para pembaca.

REFERENSI

[1] Munir, Rinaldi. 2009. *Diktat Kuliah IF2211 Strategi Algoritma*. Bandung: Penerbit Informatika ITB

[2] https://www.gm*puzzle*s.com/blog/word-search-rules-and-info/ diakses pada 18 Mei 2017, 11.43

[3] http://newsok.com/article/1893865 diakses pada 18 Mei 2017, 11.45

[4] http://faculty.simpson.edu/lydia.sinapova/www/cmsc250/LN250_Leviti n/L05-BruteForce.htm diakses pada 18 Mei 2017, 10.36

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, 18 Maret 2017

Veren Iliana Kurniadi
13515078