Penerapan Algoritma Runut-Balik untuk Menyelesaikan Permainan Pencarian Kata

Arfinda Ilmania /13515137
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10, 40132, Indonesia
13515137@std.stei.itb.ac.id

***Abstract*—Suatu permainan biasanya dibuat dengan tujuan untuk memberikan hiburan tersendiri bagi pemainnya. Selain itu, ada pula beberapa permainan yang didesain untuk mengasah otak hingga tak jarang membuat para pemainnya harus berpikir keras. Salah satunya adalah permainan pencarian kata atau dalam bahasa inggris dikenal dengan *Word Search Puzzle.* Pada makalah ini akan dijelaskan penggunaan algoritma runut-balik (*backtracking*) untuk pencarian solusi permainan ini.**

Keywords—runut-balik; algoritma; pencarian kata, puzzle

I. PENDAHULUAN

Puzzle adalah suatu jenis permainan yang bertujuan untuk mengetes kemampuan berpikir dan logika seseorang. Pada umumnya aturan bermain dalam permainan ini sangat sederhana. Permainan *puzzle* dibagi menjadi beberapa kategori seperti *mechanical puzzle* (menyusun gambar), *logic puzzle* (permainan *problem-solving*), *math puzzle (*permainan berhitung), *trivia puzzle* (biasanya berbentuk *quiz),* dan *word puzzle* (permainan kata).

Salah satu permainan *puzzle* yang menarik adalah "*Word Search Puzzle"* yang dibuat oleh LR Studios. Permainan ini dapat diunduh secara gratis di Playstore. "*Word Search Puzzle*" adalah permainan berjenis *word puzzle* yang tersusun dari sekumpulan huruf yang disusun acak dan biasanya membentuk persegi empat. Tujuan utama permainan ini adalah mencari dan menandai kata di dalam kumpulan huruf acak tersebut. Kata-kata yang dicari dapat diletakkan secara vertikal, horizontal, maupun diagonal. Dalam permainan ini kata yang dicari sudah disediakan oleh sistem, namun ada juga permainan kata yang tidak menyediakan kata sama sekali sehingga kita harus mencari tahu sendiri semua kemungkinan kata yang dapat terbentuk.

II. DASAR TEORI

A. Algoritma Runut-Balik

Algoritma Runut-Balik atau yang lebih dikenal dengan istilah *backtracking* adalah salah satu strategi pemecahan masalah dengan berbasis algoritma *Depth-First Search (*pencarian mendalam*)*. Algoritma ini merupakan bentuk yang lebih baik dibandingkan dengan algoritma *brute force.* Perbedaan yang paling mendasar adalah, algoritma ini tidak mencari semua kemungkinan solusi yang ada tetapi pencarian yang mengarah ke solusi saja yang dipertimpangkan. Sehingga waktu yang dibutuhkan relatif lebih cepat jika dibandingkan dengan *brute force.* Pada umumnya, algoritma runut-balik ini dinyatakan dalam algoritma rekursif.

Istilah runut-balik pertama kali diperkenalkan oleh D.H. Lehmer pada tahun 1950. Selanjutnya R.J. Walker, Golomb, dan Baumert menyajikan uraian umum tentang runut-balik dan penerapan pada berbagai persoalannya [HOR78].

Penerapan algoritma runut-balik banyak ditemukan dalam berbagai jenis program permainan serta masalah pada bidang kecerdasan buatan (*artificial intelligence*). Akan tetapi, kelemahan algoritma ini hanya bisa diaplikasikan terbatas pada tipe permasalahan yang memiliki solusi yang dapat dicari secara sistematis dan bertahap. Ada beberapa masalah yang tidak bisa diselesaikan dengan algoritma runut-balik, misalnya mencari suatu nilai yang diminta pada tabel yang tidak terurut.

Dalam penerapan metode runut-balik terdapat tiga properti umum yang didefinisikan sebagai berikut:

1. Solusi Persoalan, dinyatakan sebagai vektor dengan *n-tuple:*
   $X = (x_1, x_2, \ldots, X_n)$, $x_i \in$ himpunan berhingga S_i. Mungkin saja $S_1 = S_2 = \ldots = S_n$.
2. Fungsi Pembangkit nilai x_k , dinyatakan sebagai:
   T(k), yang membangkitkan nilai untuk x_k , yang merupakan komponen vektor solusi.
3. Fungsi Pembatas, dinyatakan sebagai $B(x_1, x_2, \ldots, x_k)$. Fungsi ini menentukan apakah $(x_1, x_2, \ldots, x_k)$ mengarah ke solusi. Jika ya, maka pembangkitan nilai untuk x_k-1 diteruskan. Jika tidak, maka $(x_1, x_2, \ldots, x_k)$ dibuang dan tidak dipertimbangkan lagi dalam solusi. Fungsi ini tidak selalu berbentuk fungsi matematis, bisa dinyatakan sebagai predikat yang bernilai *true* atau *false*, atau dalam bentuk lain yang ekivalen.

Skema umum pencarian solusi dengan algoritma runut-balik adalah sebagai berikut:

1. Solusi dicari dengan membentuk lintasan dari akar ke daun. Aturan perluasan simpul mengikuti metode *Depth-First Search* (pencarian mendalam). Simpul yang telah dibangkitkan dinamakan simpul hidup, sedangkan simpul hidup yang sedang diperluas dinamakan simpul ekspan. Pemberian nomor pada

simpul disesuaiakan dengan urutan pembangkitannya.

2. Saat simpul ekspan diperluas akan diperiksa apakah lintasan mengarah ke solusi. Jika tidak, maka simpul tersebut akan "dibunuh" dengan menggunakan fungsi pembatas sehingga menjadi simpul mati. Simpul yang sudah mati tidak akan pernah diperluas lagi.
3. Jika pembentukan lintasan berakhir pada simpul mati, maka pencarian diteruskan dengan membangkitkan simpul anak yang lain. Bila tidak adak lagi simpul anak yang dapat dibangkitkan, maka pencarian diteruskan dengan melakukan runut-balik ke simpul hidup terdekat. Simpul hidup tersebut kemudian menjadi simpul ekspan yang baru. Lintasan baru kemudian dibangun kembali hingga membentuk solusi.
4. Pencarian berhenti apabila solusi telah ditemukan atau tidak ada lagi simpul hidup untuk runut-balik (tidak ada solusi).

Secara umum langkah-langkah yang diambil dalam algoritma runut-balik dapat dilihat dalam *pseudo-code* berikut ini:

```
function solver(s: simpul) → boolean
    if (s adalah simpul daun) then
        if (n adalah simpul solusi) then
            → true
        else
            → false
    else
        untuk setiap simpul anak dari s
            if solver(a) benar then
               → true
    → false
```

Tabel T 1. Skema umum algoritma runut-balik versi rekursif

Fungsi solver adalah suatu fungsi rekursif yang mengembalikan nilai *boolean*. Ketika fungsi mengembalikan nilai *true*, berarti solusi berhasil ditemukan pada simpul s saat itu. Sebaliknya, jika fungsi mengembalikan *false* maka berarti solusi tidak ditemukan.

B. *Word Search Puzzle*

Word Search Puzzle adalah permaianan pencarian kata yang dibuat oleh LR Studios dan dapat diunduh secara gratis di Playstore.

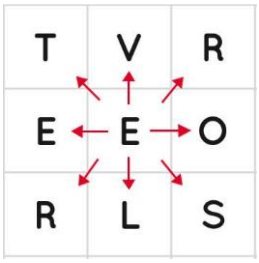
Dalam permainan ini, disajikan sekumpulan huruf yang disusun dengan urutan acak membentuk suatu persegi berukuran NxN. Permainan ini dibagi menjadi empat tingkat berdasarkan level kesulitannya yaitu *easy*, *medium*, *hard*, dan *master*. Setiap bertambah tingkat, besar persegi akan bertambah dan daftar kata akan semakin banyak. Pada makalah ini hanya akan digunakan level *easy* agar lebih mudah dalam penjabarannya.

Gambar 1. Tampilan permainan Word Search

Tujuan dalam permainan ini adalah mencari sejumlah kata yang telah disediakan, dalam kumpulan huruf acak tersebut. Kata-kata tersebut dapat disusun secara vertikal, horizontal, maupun diagonal. Apabila kata sudah ditemukan maka kita harus menandainya dengan melakukan *swap* dari huruf pertama hingga huruf terakhir sesuai dengan urutan kata.

Terdapat beberapa peraturan yang harus dipatuhi pemain, yaitu:

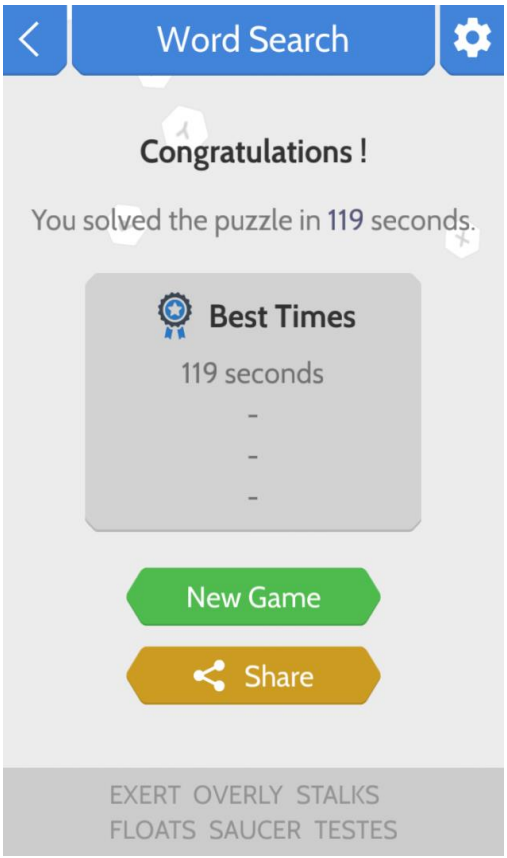
1. Pemain tidak diperbolehkan untuk mengulang huruf yang sebelumnya telah digunakan dalam menyelesaikan satu kata. Namun huruf yang telah digunakan dalam kata yang berbeda boleh digunakan kembali untuk membuat kata baru.
2. Pemain tidak boleh asal "lompat" ke sembarang huruf yang tidak bersebelahan dengan huruf tersebut. Huruf-huruf yang dipilih harus kontigu.
3. Pemain boleh berpindah ke arah mana saja asal tidak melanggar poin pertama dan kedua. Jadi gerakan yang memungkinkan adalah ke atas, bawah, kiri, kanan, diagonal kanan atas, diagonal kiri atas, diagonal kanan bawah, dan diagonal kiri bawah.



Gambar 2. Gerakan yang diperbolehkan

Pemain dikatakan menang jika ia berhasil menemukan

semua kata dalam daftar. Selama permainan berjalan terdapat *timer* yang mengukur lama pemain menyelesaikan permainan. Semakin sedikit waktu yang dibutuhkan maka akan semakin baik *score* yang didapat.



Gambar 3. Tampilan jika menang

III. ANALISIS PEMECAHAN MASALAH

Persoalan ini akan diselesaikan dengan algoritma runut-balik versi rekursif. Kumpulan huruf yang disusun secara acak direpresentasikan sebagai matriks karakter berukuran NxN. Selanjutnya kita akan memeriksa apakah kata terdapat dalam matriks tersebut dan mencetak lintasan solusinya.

Secara garis besar, langkah-langkah penyelesain masalah dengan algoritma runut-balik adalah:

1. Membuat sebuah matriks solusi bertipe bilangan bulat dengan ukuran yang sama dengan matriks karakter
2. Menginisiasi setiap sel dengan angka 0 serta sebuah counter yang dinisiasi dengan nilai 0.
3. Kemudian memeriksa setiap sel dalam matriks karakter, apabila cocok dengan karakter pada indeks pertama kata maka counter akan bertambah dan nilai sel pada matriks solusi dengan indeks yang sesuai dengan matriks karakter akan berubah nilainya menjadi nilai (counter+1) dan dilanutkan pada poin 4. Jika tidak cocok, maka pemeriksaan dilanjutkan pada indeks baris dan kolom berikutnya pada matriks karakter (kembali ke poin 3).
4. Lalu, akan diperiksa semua huruf yang ada di dekat huruf sekarang. Jika huruf tersebut cocok dengan indeks berikutnya pada kata maka pengecekan berlanjut. Jika tidak cocok, maka simpul tersebut "dibunuh" dan dilakukan *backtrack*.
5. Begitu seterusnya hingga kata ditemukan. Kondisi yang menandai kata telah ditemukan adalah ketika nilai indeks sama dengan panjang kata dikurang 1. Apabila semua sel dalam matriks karakter sudah dijelajahi namun hasilnya tetap *false*, maka solusi tidak ada. Namun dalam kasus ini solusi pasti ada.

Untuk lebih jelasnya, berikut ini *pseudo-code* algoritma runut-balik yang digunakan:

```
function searchMatrix(matrix: character[][],
word: string, solution: int[][]) → boolean
```

```
N : integer
N ← matrix.length
i traversal [0 ... N]
   j traversal [0 ... N]
      if search(matrix, word, solution, i, j,
      0, N) then
          → true
→ false
```

Tabel T 2. Fungsi searchMatrix()

```
function search(matrix: char[][], word:
string, solution: int[][], row: int, col:
int, idx: int, N: int) → boolean
```

```
path : integer
path ← 0

{mengecek apakah sel sudah diperiksa atau
karakter tidak cocok}
if (solution[row][col] ≠ 0 or word[idx] ≠ matrix[row][col]) then
   → false

if (idx = word.length - 1) then
   {kata telah ditemukan}
   solution[row][col] ← path+1
   → true

{mengganti nilai pada sel matriks solusi
dengan path yang sesuai}
solution[row][col] ← path+1

if (row+1 < N and search(matrix, word,
solution, row+1, col, idx+1, N)) then
   {mengecek ke bawah}
   → true
```

```
if (row-1 ≥ 0 and search(matrix, word, solution,
row-1, col, idx+1, N)) then
   {mengecek ke atas}
   → true

if (col+1 < N and search(matrix, word,
solution, row, col+1, idx+1, N)) then
   {mengecek ke kanan}
   → true

if (col-1 ≥ 0 and search(matrix, word, solution,
row, col-1, idx+1, N)) then
   {mengecek ke kiri}
   → true

if (row-1 ≥ 0 and col+1 < N and search(matrix,
word, solution, row-1, col+1, idx+1, N))
then
   {mengecek ke diagonal kanan atas}
   → true

if (row-1 ≥ 0 and col-1 ≥ 0 and search(matrix, word,
solution, row-1, col+1, idx+1, N)) then
   {mengecek ke diagonal kiri atas}
   → true

if (row+1 < N and col-1 ≥ 0 and search(matrix,
word, solution, row-1, col+1, idx+1, N))
then
   {mengecek ke diagonal kiri bawah}
   → true

if (row+1 < N and col+1 < N and
search(matrix, word, solution, row-1, col+1,
idx+1, N)) then
   {mengecek ke diagonal kanan bawah}
   → true

{jika tidak ada pilihan yang benar, maka
lakukan backtrack}
solution[row][col] ← 0
path-1
→ false
```

Tabel T 3. Fungsi search()

Pseudo-code diatas terdiri dari dua fungsi yaitu fungsi *searchMatrix* dan fungsi *search*. Fungsi *searchMatrix* adalah fungsi yang melakukan iterasi setiap sel dalam matriks karakter. Setiap proses memanggil fungsi *search*. Fungsi search adalah suatu fungsi rekursif. Apabila karakter pada sel sudah tidak cocok dengan indeks pertama kata maka simpul tidak akan dibangkitkan (atau langsung memenuhi basis saat memasuki fungsi *search*). Apabila cocok, maka simpul dibangkitkan hingga menemukan solusi.

Untuk memudahkan pemahaman, akan dijabarkan pemecahan solusi untuk salah satu kata yang ada pada permainan. Berikut ini adalah matriks karakter yang terdapat pada permainan:

T	V	R	E	X	E
E	E	O	T	S	R
R	L	S	T	E	S
L	F	O	C	T	S
Y	Z	U	A	S	K
V	W	S	T	A	L

Tabel T 4. Matriks karakter

Matriks tersebut berukuran 6x6, kemudian akan dibuat matriks solusi bertipe bilangan bulat dengan dimensi yang sama. Semua nilai dalam matriks bernilai 0 pada awalnya.

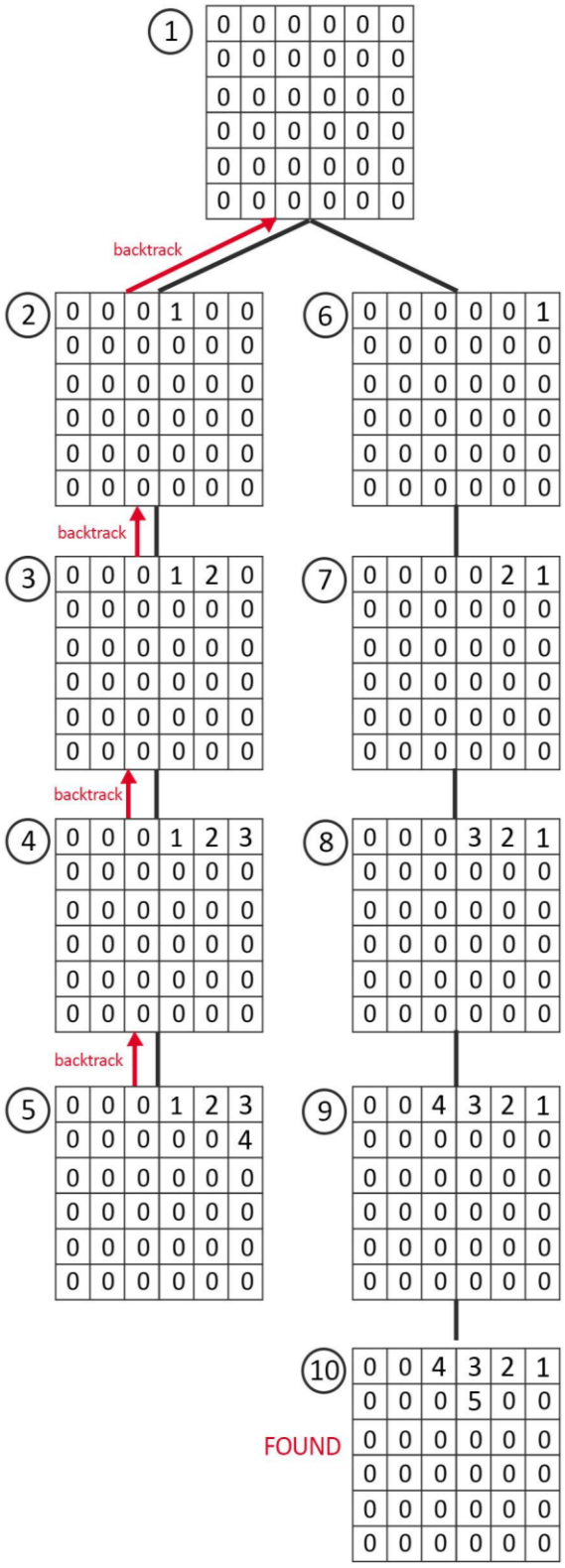
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Tabel T 5. Matriks solusi

Kata yang hendak dicari adalah “EXERT”. Pengecekan dimulai dari baris dan kolom ke 0 dalam matriks karakter dan huruf pertama dalam kata yaitu ‘E’. Indeks pada kata dimulai dari 0.

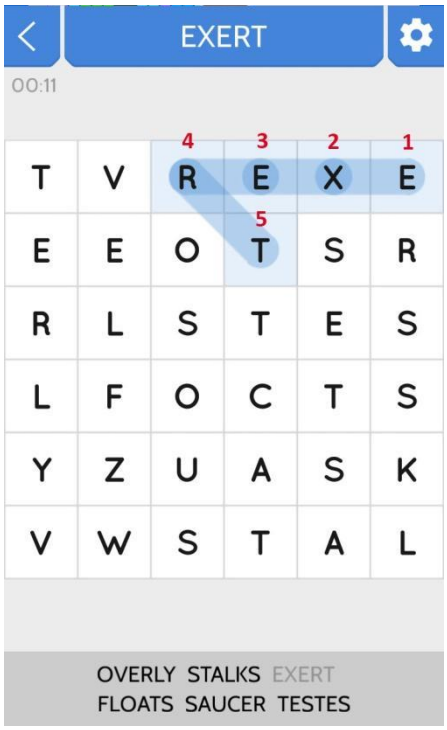
Gambar 4. Kata yang dicari

Berikut ini adalah pohon keputusan yang terbentuk:



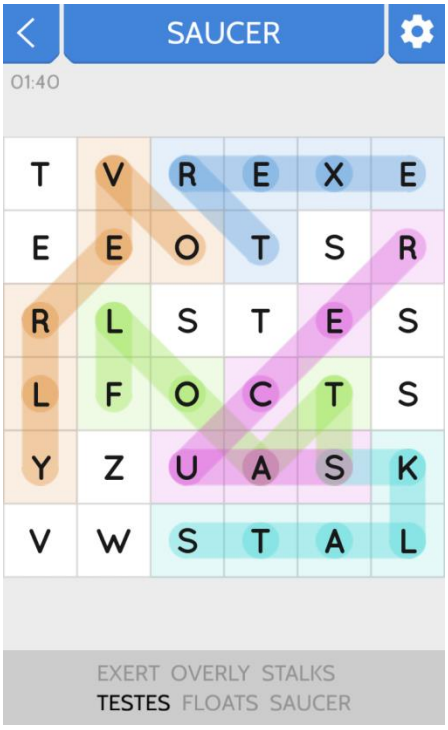
Gambar 5. Pohon keputusan

Solusi telah ditemukan dengan membangkitkan sepuluh simpul. Urutan karakter sesuai dengan matriks solusi yaitu:



Gambar 6. Pencarian kata EXERT

Cara yang sama berlaku untuk kata-kata yang lain. Sehingga akan didapat hasil sebagai berikut:



Gambar 7. Pencarian kata-kata lain

IV. KESIMPULAN DAN SARAN

Algoritma runut-balik (*backtracking*) dapat diterapkan di berbagai kasus, terutama kasus penemuan langkah atau jalan. Algoritma ini merupakan bentuk yang lebih baik daripada *brute force*, karena algoritma ini tidak memeriksa semua kemungkinan solusi yang ada. Metode ini hanya melakukan pencarian yang mengarah ke solusi saja. Sehingga waktunya pun akan lebih cepat dibanding *brute force*.

Penerapan algoritma runut-balik untuk pencarian kata pada permainan "*Word Search Puzzle"* dapat dibilang cukup efektif. Langkah yang diambil dapat lebih sedikit.

Akan tetapi akan membutuhkan waktu yang cukup lama apabila terdapat banyak kata yang harus dicari. Maka dari itu algoritma yang sudah dipaparkan diatas dapat diperbaiki lagi agar pencarian bisa sekaligus dilakukan untuk semua kata dalam waktu yang bersamaan.

UCAPAN TERIMA KASIH

Puji syukur saya panjatkan kepada Allah SWT, karena dengan karunia-Nya, saya dapat menyelesaikan makalah ini dengan baik. Terima kasih saya ucapkan kepada Ibu Nur Ulfa Maulidevi, Bapak Rinaldi Munir dan Ibu Mesayu Leylia Khodra selaku dosen pengajar mata kuliah IF2211 Strategi Algoritma, terutama pada bab algoritma Runut-Balik.

REFERENCES

[1] Kim, Scott. "What is Puzzle?". Diakses dari http://www.scottkim.com.previewc40.carrierzone.com/thinkinggames/whatisapuzzle/index.html pada 18 Mei 2017.

[2] Munir, Rinaldi. 2009. Diktat Kuliah IF2211, Strategi Algoritma, Program Studi Teknik Informatika, STEI, ITB.

[3] Sumitjain. 2015. Backtracking: Search a Word in Matrix. Diakses dari http://algorithms.tutorialhorizon.com/backtracking-search-a-word-in-a-matrix/ pada 18 Mei 2017.

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 Mei 2017

Arfinda Ilmania
13515137