Penerapan Algoritma Backtracking dan Regular Expression dalam Pencarian Solusi Permainan *Wordscapes*

Reyhan Emyr Arrosyid - 13519167
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13519167@std.stei.itb.ac.id

Abstrak*—Sejak dulu, manusia sudah mengenal hiburan. Hiburan yang dikenal manusia memiliki banyak bentuk, salah satunya adalah permainan. Salah satu permainan yang mudah untuk dimainkan di mana saja adalah permainan kata seperti teka-teki silang. Zaman sekarang, teka-teki silang sudah banyak diadapsi dalam bentuk aplikasi atau perangkat lunak. Salah satu aplikasi hasil adaptasi teka-teki silang adalah *Wordscapes*. Dalam permainan *Wordscapes, solusi permainan dapat ditemukan dengan algoritma backtracking dan regular expression. Algoritma backtracking digunakan untuk mencari solusi permainan dan regular expression digunakan untuk mencocokkan kata solusi yang dibangun dengan kata yang valid dalam kamus.

Kata kunci—Backtracking, Permainan, Regular expression, Wordscapes.

I. PENDAHULUAN

Sejak zaman dahulu, hiburan sudah menjadi bagian dari hidup manusia. Salah satu bentuk hiburan yang paling popular saat ini adalah *games* atau permainan. Salah satu permainan yang banyak digemari orang-orang permainan kata. Tidak dibutuhkan modal atau sumber daya yang banyak untuk bermain permainan kata, yang dibutuhkan hanyalah konsep permainannya dan wawasan tentang kosakata yang ada. Permainan kata juga tidak memakan waktu banyak sehingga permainan ini dapat dimainkan di mana saja. Selain mudah dimainkan, permainan kata juga dapat mengasah wawasan kosakata dan kemampuan bahasa seseorang. Salah satu jenis permainan kata adalah permainan teka-teki silang.

Pada masa sekarang, teknologi sudah tersebar dengan sangat luas sehingga banyak permainan yang dibuat ulang atau diadaptasi dengan memanfaatkan teknologi, permainan teka-teki silang merupakan salah satu dari permainan tersebut. Teka-teki silang biasanya dimainkan menggunakan pensil dan kertas, tetapi sekarang, karena teknologi, sudah banyak permainan teka-teki silang yang dimainkan pada aplikasi atau situs tertentu. Salah satu adaptasi dari teka-teki silang yang ada adalah permainan dengan judul *Wordscapes*.

Gambar 1. *Wordscapes*. Sumber: https://twitter.com/WordscapesGame

Wordscapes adalah sebuah aplikasi permainan kata yang dikembangkan oleh PeopleFun untuk Android dan IOS. Pada permainan *Wordscapes*, serupa dengan teka-teki silang, pemain akan diberikan kotak-kotak kosong yang harus diisi dengan huruf-huruf sehingga rangkaian huruf tersebut membentuk suatu kata yang valid. Namun, berbeda dari teka-teki silang umumnya, pemain tidak diberikan petunjuk untuk mengisi kotak yang kosong melainkan pemain diberi huruf-huruf yang merupakan bagian dari solusi permainan.

II. LANDASAN TEORI

A. *Algoritma Backtracking*

Algoritma *Backtracking* atau alogirtma runut-balik pertama kali diperkenalkan oleh D. H. Lenher pada tahun 1950. Setelah itu dalam perkembangannya, R.J Walker, Golomb, dan Baumert menyajikan deskripsi umum tentang algoritma backtracking ini.

Algoritma *Backtracking* dapat dianggap sebuah metode pemecahan masalah yang mangkus, terstruktur, dan sistematis. Algoritma *Bactracking* merupakan perbaikan dari *exhaustive search*. Pada *exhaustive search*, semua kemungkinan solusi dibangkitkan, dieksplorasi, dan dievaluasi satu per satu. Perbaikan yang dilakukan oleh algoritma *Backtracking* adalah dengan mengabaikan kandidat solusi yang tidak mengarah pada solusi dan hanya memperhitungkan dan mengevaluasi kandidat solusi yang mengarah ke solusi. Dengan kata lain, algoritma runut-balik memangkas simpul-simpul yang tidak mengarah ke solusi.

Terdapat tiga jenis persoalan dalam *backtracking*, yaitu:

1. *Decision Problem* yaitu pencarian suatu solusi yang mungkin.
2. *Optimization Problem* yaitu pencarian solusi terbaik.
3. *Enumeration Problem* yaitu pencarian seluruh solusi persoalan.

Dalam algoritma *Backtracking*, terdapat tiga properti umum sebagai berikut.

1. Solusi persoalan

   Solusi persoalan algoritma *Backtracking* biasanya dinyatakan dalam bentuk vektor dengan n-tuple: X = $\{x_1, x_2, \ldots, x_n\}$, $x_i \in S_i$. Umumnya $S_1 = S_2 = ... = S_n$.

2. Fungsi pembangkit

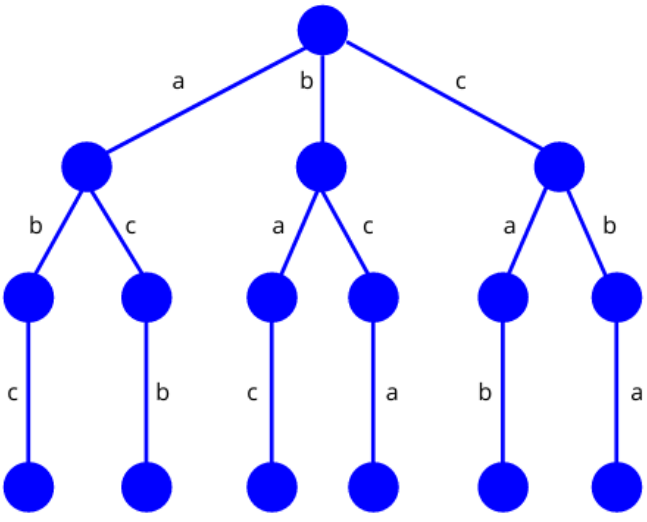
   Fungsi pembangkit digunakan untuk membangkitkan nilai x_k. Umumnya dinyatakan sebagai predikat T().

   T(x[1], x[2], …, x[k-1]) membangkitkan nilai untuk x_k, yang merupakan komponen vektor solusi.

3. Fungsi pembatas

   Fungsi pembatas algoritma *Backtracking* dinyatakan sebagai predikat B($x_1, x_2, \ldots, x_k$). B bernilai true jika ($x_1, x_2, \ldots, x_k$) mengarah ke solusi (tidak melanggar *constraints*) dan false jika tidak. Jika true, pembangkitan x_{k+1} dilanjutkan, namun jika false, ($x_1, x_2, \ldots, x_k$) dibuang.

Semua kemungkinan solusi dalam sebuah persoalan disebut dengnan ruang solusi yang dapat dimodelkan dengan struktur pohon. Setiap simpul pohon menyatakan status persoalan dan sisi atau cabang diberi label yaitu nilai-nilai x_i. Lintasan dari akar pohon ke daun menyatakan sebuah solusi yang mungkin dan seluruh lintasan dari akar ke daun membentuk ruang solusi.

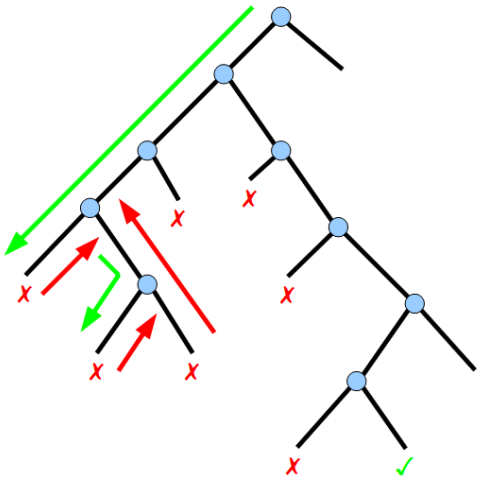


Gambar 2. Contoh pohon ruang solusi. Sumber: https://www.baeldung.com/cs/backtracking-algorithms

Adapun prinsip pencarian solusi dalam algoritma *Backtracking* adalah dengan pertama membangkitkan simpul-simpul status. Simpul status dibangkitkan sesuai dengan aturan *depth-first order* (DFS), yaitu pembangkitan dilakukan ke dalam terlebih dahulu.

Simpul-simpul yang sudah dibangkitkan dinamakan simpul hidup dan Simpul hidup yang sedang diperluas disebut dengan simpul-E. simpul-E akan dimatikan lalu menjadi simpul mati jika lintasan yang sedang dibentuk tidak mengarah pada solusi dengan menerapkan fugsi pembatas.

Jika proses pembentukan lintasan berhenti dengan simpul mati, maka proses pencarian *backtrack* ke simpul orangtua sebelumnya lalu diteruskan dengan membangkitkan simpul anak lainnya yang belum diekspan. Simpul anak baru ini akan menjadi simpul-E selanjutnya. Pencarian dihentikan bila *goal node* atau simpul tujuan telah tercapai.



Gambar 3. Ilustrasi algoritma *backtracking*. Sumber: https://www.w3.org/2011/Talks/01-14-steven-phenotype/

B. Pencocokan String

Pencocokan string, disebut juga *string matching* atau *pattern matching* adalah sebuah algoritma yang bertujuan untuk menemukan sebuah pola dalam teks atau string yang lebih panjang. Dalam pencocokan string didefinisikan beberapa hal berikut.

- T: Teks, yaitu sebuah string yang memiliki panjang n karakter.
- P: *Pattern*, yaitu string yang memiliki panjang m karakter (asumsi m < n) yang akan dicari dalam teks.

Dalam praktiknya, pencocokan string dapat diaplikasikan pada banyak bidang. Aplikasi yang paling umum digunakan adalah fitur pencarian dalam *text editor*. Aplikasi lain dari pencocokan string antara lain *web search engine*, analisis citra, *bioinformatics*, *spell checker*, dan pendeteksi plagiarisme pada karya bentuk tulisan.

C. Regular Expression

Regular expression atau biasa disebut regex adalah notasi standar yang mendeskripsikan suatu pola (*pattern*) berupa urutan karakter atau string. Regex sudah umum digunakan dalam persoalan pencocokan string dengan efisien. Regex sudah menjadi standar yang tersebar di semua tools dan bahasa pemrograman. Istilah regular expression berasal dari teori matematika dan ilmu komputer, yang merefleksikan sifat ekspresi matematis yang disebut *regularity*. Dalam ilmu komputer, regular expression diimplementasikan menggunakan Finite State Machine (FSM).

Dalam regex, ada banyak bentuk atau sintaks karakter untuk membentuk regex yaitu sebagai berikut.

- String literal

  String literal merupakan bentuk paling dasar dalam pencarian regular expression. String literal akan mencari pola pada teks yang sama persis seperti pola tersebut. Misalnya regex "stima" akan cocok dengan kata "stima".

- Metakarakter

  Metakarakter adalah karakter khusus dalam regular expression yang mempengaruhi proses pencocokan string. Meta karakter dalam regex tidak dicocokkan secara literal. Contoh dari metakarakter antara lain ".", "|", "*", "+", "^", dan "$".

- Karakter "|"

  Karakter "|" menyatakan *or* yaitu akan cocok dengan salah satu dari semua yang dihubungkan dengan "|". Sebagai contoh, regex "tubes|tucil" akan cocok dengan kata "tubes" atau "tucil".

- *Character Classes*
  - Kelas atau himpunan sederhana dapat dibangun dengan kurung siku ("[]"). Contohnya regex "[abc]" akan cocok dengan karakter "a", "b", atau "c".
  - Kurung siku dikombinasikan dengan karakter "^" akan mendefinisikan negasi dari karakter yang berada di dalam kurung siku. Contohnya regex "[^abc]" akan cocok dengan semua karakter selain "a", "b", dan "c".
  - Karakter "-" dalam suatu *class* akan mendefinisikan *range*. Contohnya regex "[a-zA-Z]" akan cocok dengan karakter "a" sampai "z" atau "A" sampai "Z".
  - Gabungan dua *range* juga dapat didefinisikan dengan regex. Contohnya regex "[a-d[m-p]]" akan cocok dengan "a" sampai "d" atau "m" sampai "p" (gabungan).
  - Irisan dua kelas dapat didefinisikan dengan regex dengan karakter "&&". Contohnya regex "[a-z&&[def]]" akan cocok dengan irisan a-z dan def yaitu "d", "e", atau "f".
  - Subtraksi du akelas dapat didefinisikan dengan regex dengan karakter "&&". Contohnya regex "[a-z&&[^bc]]" akan cocok dengan "a" sampai "z" kecuali "b" dan "c".
- *Predefined Character Class*
  - Regex "." atau wildcard akan cocok dengan semua karakter. Sebagai contoh, regex ".aku" akan cocok dengan "laku", "paku", "kaku", dan seterusnya.
  - Regex "\d" akan cocok dengan semua digit dari 0 sampai 9.
  - Regex "\D" akan cocok dengan karakter non-digit ([^0-9]).
  - Regex "\s" akan cocok dengan karakter *whitespace* atau [\t\n\x0B\f\r].
  - Regex "\S" akan cocok dengan karakter non-*whitespace* ([^\s]).
  - Regex "\w" akan cocok dengan *word character* atau [a-zA-Z_0-9].
  - Regex "\W" akan cocok dengan *non word character* atau [^\w].
- *Quantifier*

  Quantifier dalam regular expression digunakan untuk mendefinisikan jumlah perulangan pola.
  - Karakter "?" mendefinisikan jumlah kemunculan 0 atau 1 kali. Sebagai contoh, regex "a?" memiliki arti karakter "a" muncul stau kali atau tidak sama sekali.
  - Karakter "*" mendefinisikan jumlah kemunculan 0 atau banyak. Sebagai contoh, regex "a*" memiliki arti karakter "a" tidak muncul sama sekali atau banyak.
  - Karakter "+" mendefinisikan jumlah kemunculan satu atau banyak. Sebagai contoh, regex "a+" memiliki arti karakter "a" muncul satu kali atau banyak.
  - Karakter "{}" dapat digunakan untuk mendefinisikan jumlah kemunculan secara khusus. Terdapat beberapa kasus penggunaannya. Regex "a{n}" memiliki arti karakter "a" muncul tepat n kali, regex "a{n,}" memiliki arti karakter "a" muncul setidaknya n kali, dan regex "a{n, m}" memiliki arti karakter "a" muncul antara n sampai m kali.
- *Boundary Matchers*

  Boundary Matchers digunakan untuk mencocokkan pola yang berada pada posisi tertentu misalnya di awal atau akhir.
  - Karakter "^" dapat digunakan untuk mencocokkan pola pada awal baris. Contohnya regex "^a" dalam teks "a a" hanya akan cocok dengan karakter "a" yang pertama.

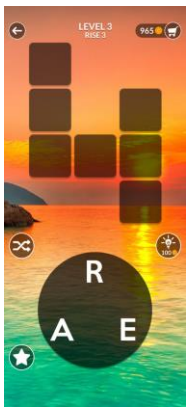
- Karakter “$” dapat digunakan untuk mencocokkan pola pada akhir baris. Contohnya regex “a$” dalam teks “a a” hanya akan cocok dengan karakter “a” yang kedua.
- Regex “\b” digunakan untuk mencocokkan pola pada batas kata. Contohnya regex “\ba\b” dalam teks “a aa” hanya akan cocok dengan karakter “a” yang pertama.
- Regex “\B” akan cocok dengan bukan batas kata. Contohnya regex “\Ba” dalam teks “a aa” hanya akan cocok dengan karakter “a” yang terakhir.
- Regex “\G” akan cocok dengan akhir match sebelumnya.
- Regex “\Z” akan cocok dengan akhir dari input tapi untuk *final terminator* jika ada.
- Regex “\z” akan cocok dengan akhir dari input.

III. Implementasi dan Pengujian

Untuk mengimplementasikan algoritma *Backtracking* dan *regular expression* dalam pemecahan persoalan dalam permainan *Wordscapes*, suatu *level* dalam permainan *Wordscapes* harus dimodelkan terlebih dahulu dalam sebuah program. Dalam program yang dibuat juga harus disimpan daftar kata-kata yang valid sebagai kamus permainan.

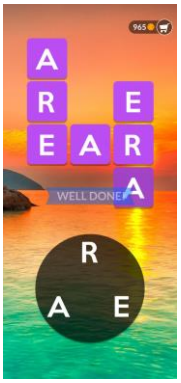
A. *Pemodelam Permainan Wordscapes*

Dalam permainan *Wordscapes*, pemain pertama-pertama akan dihadapkan dengan pilihan *level* yang ingin dimainkan. Setelah pemain memilih sebuah *level*, maka akan ditampilkan *level* tersebut yang terdiri atas kotak-kotak yang harus diisi untuk membentuk kata dan huruf yang dapat digunakan untuk membentuk setiap kata.



Gambar 4. Contoh *level* dalam *Wordscapes*. Sumber: Dokumen Penulis

Setelah layar *level* ditampilkan, pemain dapat memulai bermain dengan membentuk suatu kata menggunakan huruf yang tersedia. Pemain dapat membentuk kata dengan pertama menekan sutau huruf lalu menggeser ke huruf lain hingga membentuk suatu kata. Setelah pemain berhasil menemukan semua kata dalam *level*, pemain akan menyelesaikan *level* tersebut lalu akan berpindah ke *level* selanjutnya.



Gambar 5. *Level* yang sudah selesai. Sumber: Dokumen Penulis

Pada Gambar 5. terlihat solusi dari level pada Gambar 4. Dalam hal ini kata yang merupakan solusi adalah “are”, “ear” dan “era”.

Untuk memodelkan suatu *level* dalam sebuah program, perlu ditentukan komponen-komponen yang terdapat dalam *level* tersebut. Suatu *level* terdiri atas tiga komponen sebagai berikut.

1. *Grid* atau kotak-kotak yang akan diisi oleh huruf untuk membentuk suatu kata.
2. *Slot* yang menandakan kotak mana saja yang akan membentuk satu kata.
3. Huruf-huruf yang dapat digunakan untuk membentuk suatu kata.

Grid dapat direpresentasikan dengan matriks dengan ukuran yang disesuaikan dengan ukuran maksimal kotak dalam *level*, dalam hal *level* pada Gambar 4, *grid* dapat direpresentasikan dengan matriks berukuran 4x3 sebagai berikut.

Tabel 1. Representasi *grid*. Sumber: Dokumen Penulis

m\n	0	1	2
0	-	*	*
1	-	*	-
2	-	-	-
3	*	*	-

Pada representasinya, karakter “*” menandakan kotak yang tidak dapat diisi oleh huruf dan karakter “-“ menandakan kotak yang dapat diisi oleh huruf.

Slot dalam permainan dapat direpresentasikan dengan struktur data khusus yang mengandung beberapa informasi yaitu titik awal sebuah *slot*, jenis *slot* (vertikal atau horizontal), dan panjang *slot*. Dalam kasus ini terdapat tiga *slot* yaitu:

1. Awal = (0, 0), jenis = vertikal, panjang = 3
2. Awal = (2, 0), jenis = horizontal, panjang = 3
3. Awal = (1, 2), jenis = vertikal, panjang = 3

Keterangan: (a, b) mengacu pada elemen matriks pada baris a dan kolom b.

Huruf-huruf dapat dengan mudah direpresentasikan dengan daftar yang berisi huruf-huruf yang sama, dalam hal ini [R, E, A].

B. *Penerapan Backtracking dan Regular Expression*

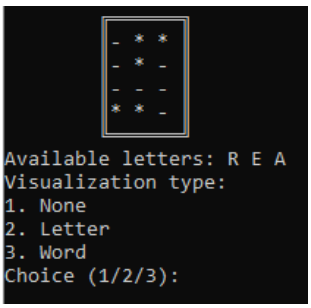
Sebelum menerapkan algoritma *Backtracking*, perlu ditentukan ruang solusi dan fungsi pembatasnya. Ruang solusi dari persoalan ini adalah semua *state* atau keadaan grid yang mungkin. Fungsi pembatas pada persoalan ini adalah fungsi untuk memeriksa apakah huruf yang ditempatkan pada suatu *slot* tidak melebihi semua huruf yang disediakan, apakah kata yang dapat dibentuk dari gabungan huruf tersebut merupakan kata yang valid dalam kamus, dan apakah suatu kata yang terbentuk belum terdapat pada *slot* lain. Algoritma *Backtracking* untuk persoalan ini adalah sebagai berikut.

1. Periksa kotak kotak pertama dalam *slot* pertama.
2. Untuk setiap huruf, periksa nilai fungsi pembatas.
3. Jika bernilai true, *assign* kotak tersebut dengan huruf tersebut lalu bangkitkan kotak berikutnya yaitu kotak kosong selanjutnya pada *slot* yang sama atau kotak kosong pertama pada *slot* selanjutnya, ulangi dari langkah 2 untuk kotak baru tersebut. Jika false coba huruf selanjutnya.
4. Jika semua huruf sudah dicoba dan tidak ada yang menuju solusi, lakukan *backtracking* ke kotak sebelumnya.
5. Program akan berhenti ketika semua kotak kosong sudah terisi dan tidak ada yang melanggar fungsi pembatas.

Regular expression digunakan dalam fungsi pembatas saat akan memeriksa kata yang dapat terbentuk merupakan kata valid atau bukan. pemeriksaan dilakukan dengan membandingkan regex yang terbentuk dengan mengganti kotak kosong dengan regex “\w” yang dibandingkan dengan kata dalam kamus. Pada regex yang terbentuk juga dimanfaatkan karakter “^” dan “$” agar ditemukan kata dengan panjang yang sama. Sebagai contoh, jika suatu slot berisi “AR-“, regex yang dihasilkan adalah “^AR\w$”. Pada regex juga tidak dipertimbangkan *case* dari sebuah huruf (*case insensitive*). Jika dalam kamus terdapat kata yang memenuhi regex tersebut, artinya pada slot tersebut dapat dibentuk suatu kata yang valid.

C. *Pengujian*

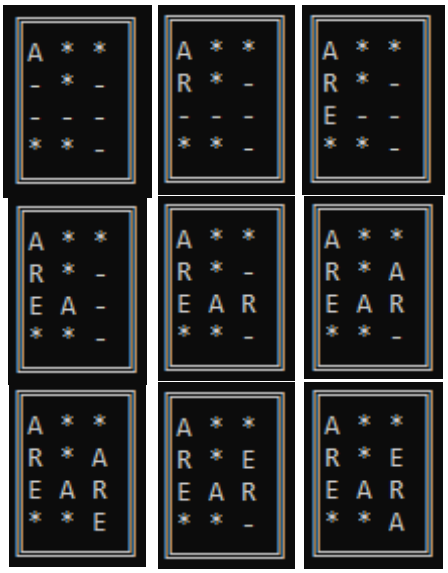
Berikut merupakan tampilan awal program.



Gambar 6. Tampilan awal program. Sumber: Dokumen Penulis

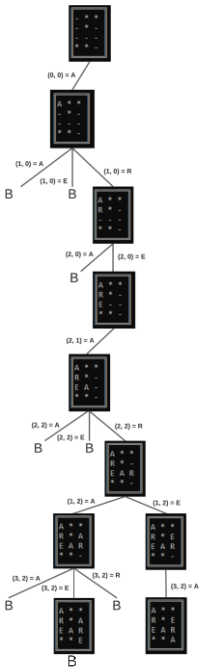
Pada tampilan Gambar 6, dapat dilihat representasi *grid* yang sudah disebutkan sebelumnya dan juga huruf-huruf yang dapat digunakan. Setelah itu pengguna akan diminta pilihan mode visualisasi. Mode visualisasi *None* hanya akan menampilkan solusi akhir jika ditemukan, mode *Letter* akan menampilkan visualisasi untuk setiap huruf yang ditambahkan atau diubah, dan mode visualisasi *Word* akan menampilkan visualisasi untuk setiap kata yang ditambahkan atau diubah.

Berdasarkan hasil pengujian, algoritma *Backtracking* berjalan dengan tahapan-tahapan menggunakan mode visualisasi *Letter* sebagai berikut.



Gambar 7. Langkah per langkah pengujian. Sumber: Dokumen Penulis

Berdasarkan hasil pengujian, dapat dilihat algoritma *Backtracking* berjalan dengan benar, *backtracking* dapat dilihat pada langkat ke-7 dan ke-8. Kata yang sebelumnya terbentuk yaitu “are” tidak valid karena sudah ada pada *slot* lain sehingga program harus mundur mencari kata lain. Jika hasil pengujian dibandingkan dengan solusi pada Gambar 5, dapat disimpulkan hasil program sudah sesuai dengan solusi aslinya. Pohon pencarian yang terbentuk adalah sebagai berikut.



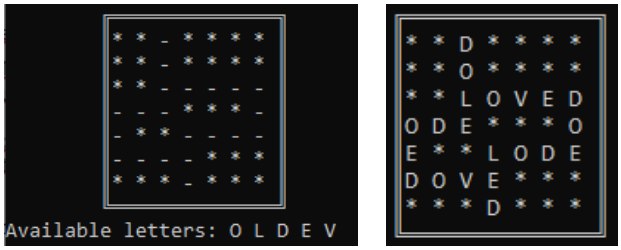
Gambar 8. Pohon pencarian solusi Sumber: Dokumen Penulis

Huruf B pada pohon pencarian menandakan simpul yang dimatikan oleh fungsi pembatas.

Selanjutnya akan dilakukan pengujian untuk *level* yang lain yaiut *level* berikut.

Gambar 9. *Level* lain *Wordscapes* Sumber: Dokumen Penulis

Hasil pengujian *level* dalam Gambar 9 adalah sebagai berikut.



Gambar 10. Hasil pengujain l*evel* pada Gambar 9. Sumber: Dokumen Penulis

Berdasarkan hasil pengujian, dapat dibandingkan hasil pengujian dibandingkan dengan solusi pada Gambar 9, dapat dilihat bahwa solusi yang ditemukan program berbeda dengan solusi sebenarnya. Hal ini dapat terjadi karena solusi permainan teka-teki silang seperti *Wordscapes* dapat memiliki lebih dari solusi yang valid dan solusi pada aplikasi *Wordscapes* merupakan salah satu solusi yang dipilih oleh pengembang permainan sedangkan program pengujian hanya mengembalikan satu solusi yang valid dan bukan semua solusi. Hal ini juga dapat terjadi karena kamus yang digunakan oleh aplikasi *Wordscpaes* berbeda dengan kamus yang digunakan pada program pengujian.

IV. Simpulan

Algoritma *Backtracking* dan regular expression dapat digunakna untuk memecahkan banyak persoalan, salah satunya permainan *Wordscapes*. Dalam permainan *Wordscapes*, algortima *Backtracking* digunakan untuk mencari sebuah solusi permainan dan regular expression digunakan untuk memeriksa sebuah kandidat solusi menuju ke solusi atau tidak.

Berdasarkan hasil pengujian yang sudah dilakukan sebelumnya, dapat disimpulkan bahwa algoritma *Backtracking* berhasil dalam menemukan solusi permainan *Wordscapes*. Namun, solusi yang ditemukan berbeda dengan solusi sebenarnya. Hal ini terjadi karena program pengujian hanya mengembalian solusi pertama yang ditemukan. Hal ini juga dapat terjadi karena perbedaan kamus yang digunakan. Meskipun solusi yang ditemukan berbeda, algoritma *Backtracking* sudah cocok digunakan untuk persoalan pencarian solusi pada permainan *Wordscapes*.

Pranala Video Youtube

Video penjelasan persoalan dalam makalah ini dapat dilihat pada laman berikut.

https://youtu.be/V6UZAEsNVgM

Ucapan Terima Kasih

Pertama-tama, penulis mengucapkan puji syukur kepada Allah SWT karena berkat kehendak-Nya penulis dapat menyelesaikan tugas makalah untuk mata kuliah Strategi Algoritma ini dengan baik. Selanjutnya, penulis juga ingin mengucapkan terima kasih kepada keluarga, teman, dan pihak lain yang terlibat dalam pembuatan makalah ini. Terakhir, penulis berterima kasih kepada Bapak Dr. Ir. Rinaldi Munir, MT. selaku dosen IF2211 Strategi Algoritma pada kelas K-04 yang sudah memberikan ilmu dan pengetahuan kepada penulis selama perkuliahan sehingga penulis dapat menyelesaikan makalah ini.

Daftar Pustaka

[1] WordscapesGame Twitter. https://twitter.com/WordscapesGame. Diakses pada 10 Mei 2021.

[2] Munir, Rinaldi. "*Algoritma Runut-balik (Backtracking)*". https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-backtracking-2021-Bagian1.pdf. Diakses pada 10 Mei 2021.

[3] Chaturvedi, Aayush, atulim, tripathipriyanshu1998. "*Backtracking Introduction*". https://www.geeksforgeeks.org/backtracking-introduction/. Diakses pada 10 Mei 2021.

[4] Datta, Subham. "*Backtracking Algorithms*". https://www.baeldung.com/cs/backtracking-algorithms. Diakses pada 10 Mei 2021.

[5] Pemberton, Steven. "*The Computer as Extended Phenotype (Computers, Genes and You)*". https://www.w3.org/2011/Talks/01-14-steven-phenotype/. Diakses 10 Mei 2021.

[6] Khodra, Masayu Leylia. "*Pencocokan string dengan Regular Expression (Regex)*". https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2018-2019/String-Matching-dengan-Regex-2019.pdf. Diakses pada 10 Mei 2021.

[7] Wibisono, Yudi, dan Masayu Leylia Khodra. "*Pengantar Regular Expression*". https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2019-2020/Modul-Praktikum-NLP-Regex.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.

Jakarta, 11 Mei 2021

Reyhan Emyr Arrosyid
13519167