Implementasi Algoritma Depth-First Search dalam Menyelesaikan Word Search Puzzle

Amira Izani - 13523143
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: izani1807@gmail.com , 13523143@std.stei.itb.ac.id

***Abstrak*— Word search puzzle merupakan permainan klasik yang menantang pemain untuk menemukan kata tersembunyi di antara kumpulan huruf acak. Permainan ini cocok untuk semua kalangan dan dapat memiliki tingkat kesulitan tinggi, terutama ketika kata yang dicari memiliki kemiripan huruf dengan karakter lain di sekitarnya atau diletakkan dalam arah yang berbalik. Dalam penelitian ini, penulis mengembangkan sebuah program untuk mencari solusi word search puzzle yang mampu mendeteksi kata dalam delapan arah pencarian, yaitu horizontal, vertikal, dan diagonal ke segala arah. Metode yang digunakan adalah algoritma Depth-First Search (DFS) yang dipadukan dengan teknik backtracking untuk menelusuri kemungkinan jalur huruf dari setiap posisi awal yang cocok. Hasil implementasi menunjukkan bahwa pendekatan ini efektif dalam menyelesaikan word search puzzle dengan berbagai ukuran dan tingkat kesulitan yang bervariasi. Program mampu menemukan kata dengan akurasi tinggi serta waktu eksekusi yang efisien dan stabil. Penelitian ini juga memberikan pemahaman praktis bagi penulis dalam menerapkan algoritma DFS dan backtracking pada kasus nyata. Selain itu, pendekatan ini dapat dikembangkan lebih lanjut untuk menangani varian permainan serupa dengan kompleksitas yang lebih tinggi dan tantangan algoritmik yang lebih luas.**

***Kata Kunci*—DFS, Backtracking, Word Search Puzzle**

I. PENDAHULUAN

Figure 1.1. Word Search Puzzle
Source: https://thewordsearch.com/

Word search puzzle adalah permainan pencarian kata-kata yang tersembunyi dalam sebuah kotak berisi huruf-huruf acak. Kotak ini umumnya berbentuk persegi atau persegi panjang, namun bentuk lain yang lebih kreatif seperti menyerupai objek (misalnya hewan atau tumbuhan) juga dapat digunakan. Kata-kata dapat disusun secara horizontal, vertikal, maupun diagonal, dan dapat dibaca dari arah depan maupun belakang. Satu huruf dalam kotak dapat menjadi bagian dari lebih dari satu kata, dan tidak semua huruf dalam kotak harus membentuk kata tertentu.

Tujuan permainan ini adalah menemukan dan menandai semua kata yang tersembunyi dalam kotak tersebut. Dalam versi yang lebih sederhana, daftar kata yang harus dicari biasanya disediakan di samping kotak. Pemain menyelesaikan puzzle dengan mencari dan melingkari huruf-huruf yang membentuk setiap kata, lalu mencoret kata tersebut dari daftar agar tidak dicari kembali. Umumnya, kata yang lebih panjang lebih mudah ditemukan karena lebih mencolok. Pada versi yang lebih menantang, daftar kata dapat disembunyikan atau ditiadakan sepenuhnya.

Word search puzzle sering dimanfaatkan dalam pembelajaran bahasa untuk memperluas kosakata dan melatih pengenalan pola huruf. Permainan ini juga mudah dijumpai di media cetak maupun aplikasi digital.

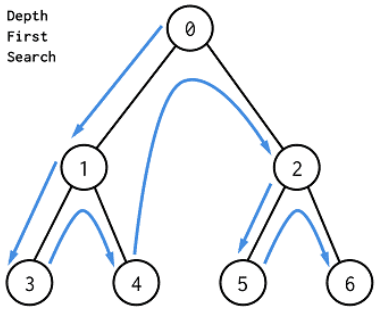

Figure 1.2. Depth First Search
Source: https://www-freelancinggig-com.translate.goog/blog/2019/02/06/what-is-the-difference-between-bfs-and-dfs-algorithms/?_x_tr_sl=en&_x_tr_tl=id&_x_tr_hl=id&_x_tr_pto=imgs

Depth-First Search (DFS) adalah algoritma penelusuran graf atau pohon yang bekerja dengan menjelajahi simpul (node) sedalam mungkin sebelum kembali dan menjelajahi cabang lainnya. Penelusuran dimulai dari simpul akar (root) atau simpul awal, lalu bergerak ke salah satu tetangganya hingga tidak ada simpul yang belum dikunjungi, lalu kembali (backtrack) ke simpul sebelumnya untuk menjelajah jalur lain.

DFS menggunakan struktur data stack dan mengikuti prinsip LIFO (Last In First Out). Algoritma ini membangun pohon atau lintasan pencarian secara mendalam terlebih dahulu.

Algoritma backtracking merupakan strategi pemecahan masalah yang digunakan untuk mengeksplorasi berbagai kemungkinan solusi dengan pendekatan mencoba jalur. Ketika suatu jalur yang diuji tidak menghasilkan solusi yang benar, algoritma akan "mundur" (backtrack) ke langkah sebelumnya dan mencoba jalur alternatif lainnya. Konsep ini mirip dengan menyusun puzzle, di mana kita mencoba berbagai potongan hingga menemukan kombinasi yang sesuai.

Penelitian ini bertujuan untuk mengembangkan sebuah Solusi untuk word search puzzle secara otomatis yang mampu mendeteksi kata dalam delapan arah pencarian dalam horizontal, vertikal, dan diagonal, baik maju maupun mundur. Motivasi dari penelitian ini adalah untuk mengatasi keterbatasan pencarian manual yang memerlukan waktu lama dan rawan kesalahan, terutama ketika ukuran papan puzzle besar atau susunan huruf menyerupai kata-kata lain. Dengan memanfaatkan algoritma Depth-First Search (DFS) dan backtracking, program diharapkan dapat menelusuri jalur huruf secara efisien dan sistematis, sehingga menghasilkan solusi yang cepat, akurat, dan terstruktur dalam menyelesaikan word search puzzle.

II. Landasan Teori

A. *Algoritma Depth First Search (DFS)*

Algoritma Depth First Search merupakan metode penelusuran graf atau pohon yang bekerja dengan menjelajahi simpul sedalam mungkin sebelum kembali (*backtrack*) ke simpul sebelumnya untuk mengeksplorasi cabang lain. Proses ini mengikuti prinsip LIFO (Last In First Out), umumnya diimplementasikan dengan menggunakan stack atau melalui pemanggilan fungsi rekursif.

Dalam konteks pencarian solusi, DFS membentuk pohon dinamis yang disebut sebagai pohon ruang status (*state space tree*), di mana simpul akar merepresentasikan keadaan awal dan simpul daun merupakan solusi (goal). DFS melakukan eksplorasi dari akar menuju cabang yang dalam terlebih dahulu sebelum menjelajahi cabang lain. Jika simpul yang dituju bukan solusi atau tidak dapat diekspansi, maka algoritma melakukan backtracking ke simpul sebelumnya dan mencoba cabang lain yang belum dikunjungi.

DFS cocok untuk masalah yang memerlukan eksplorasi menyeluruh secara mendalam dan ketika solusi mungkin berada jauh dari simpul awal. Kelebihannya terletak pada efisiensi ruang karena hanya menyimpan jalur yang sedang dieksplorasi. DFS sering digunakan dalam pemecahan masalah labirin, permainan papan, teka-teki seperti Sudoku dan crossword puzzle, serta sangat relevan untuk word search puzzle yang memerlukan pencarian dalam banyak arah dengan jalur huruf yang saling bersinggungan.

B. *Algoritma Runut-Balik (Backtracking)*

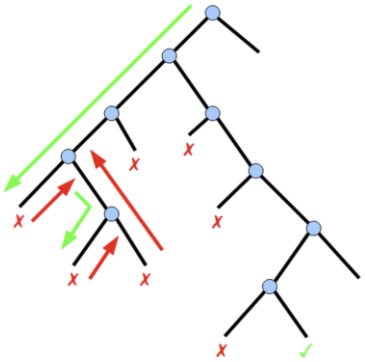

Figure 2.1 Algoritma Backtracking
Source: https://www.w3.org/2011/Talks/01-14-steven-phenotype/

Backtracking adalah metode pemecahan masalah yang sistematis dan efisien dengan menjelajahi ruang solusi melalui proses coba-coba. Berbeda dengan *exhaustive search* yang mengevaluasi semua kemungkinan solusi, backtracking hanya menelusuri cabang solusi yang memungkinkan. Jika suatu jalur tidak memenuhi syarat (constraint), maka algoritma akan berhenti menjelajah ke arah tersebut dan kembali (backtrack) ke simpul sebelumnya untuk mencoba jalur alternatif [6].

Solusi dalam algoritma ini biasanya dinyatakan dalam bentuk vektor n-tuple $X = (x_1, x_2, ..., x_n)$, dengan setiap komponen x_i berasal dari himpunan kemungkinan S_i. Proses pencarian dilakukan dengan membangkitkan simpul-simpul dalam bentuk pohon ruang status (state space tree), di mana lintasan dari akar ke daun merepresentasikan solusi potensial. Simpul yang tidak memenuhi fungsi pembatas (*bounding function*) akan dipangkas (*pruned*), sehingga mempercepat proses pencarian.

Algoritma backtracking mengikuti prinsip traversal Depth-First Search (DFS) dan bisa diimplementasikan dalam bentuk rekursif maupun iteratif. Salah satu contoh penerapannya adalah pada persoalan N-Queens, di mana backtracking digunakan untuk mencari posisi peletakan ratu yang tidak saling menyerang. Prinsip ini juga sangat sesuai untuk menyelesaikan word search puzzle, karena memungkinkan

pencarian jalur huruf yang valid dari berbagai arah dengan pemangkasan jalur yang tidak relevan.

III. Pembahasan

A. *Deskripsi Implementasi Program*

Program Word Search Puzzle solver ini dibangun menggunakan bahasa Python dengan pendekatan algoritma Depth-First Search (DFS) yang dipadukan dengan teknik backtracking. Program dirancang untuk membaca input dari file teks, melakukan pencarian kata dalam papan huruf dua dimensi, dan memberikan hasil dalam bentuk visualisasi terminal yang interaktif.

Input program berupa file .txt yang mengikuti struktur sebagai berikut:

1. Baris pertama berisi dua bilangan bulat N dan M yang menyatakan jumlah baris dan kolom papan.
2. Baris kedua berisi sebuah bilangan bulat X yang menyatakan jumlah kata yang ingin dicari.
3. N baris berikutnya masing-masing berisi M huruf (dapat dipisahkan dengan spasi) yang membentuk papan puzzle.
4. X baris terakhir berisi daftar kata yang menjadi target pencarian.

Contoh format input:

```
10 10
5
P U Z Z L E S B M Y
W F S P M F P S F D
J E C D R O W A X J
T L W H Y M J L M G
E M H M I C B J Z Y
B A C K T R A C K Y
H N V E L X I U D V
T X G E S F L Y P O
K U N I V E O Z M X
V O D H H C R A E S
DFS
BACKTRACK
PUZZLE
WORD
SEARCH
```

Setelah membaca dan memproses input, program akan mengiterasi setiap sel dalam papan dan memulai proses DFS apabila huruf pertama dari suatu kata cocok dengan huruf pada sel tersebut. Proses DFS dilakukan ke seluruh delapan arah pencarian yang mungkin, dan akan kembali (*backtrack*) jika jalur yang sedang ditelusuri tidak sesuai.

Output dari program tampilan papan puzzle dengan warna yang membedakan kata yang ditemukan menggunakan pustaka colorama, sehingga pengguna dapat langsung mengenali lokasi kata-kata tersebut dalam terminal.

Selain itu, program menyediakan fitur pencarian interaktif. Setelah hasil pencarian awal ditampilkan, pengguna dapat mengetik nama suatu kata, dan program akan memberikan informasi apakah kata tersebut ditemukan atau tidak, lengkap dengan koordinat awal dan arah penyusunannya. Fitur ini berguna jika pengguna ingin mencari satu kata tertentu tanpa harus membaca seluruh daftar hasil terlebih dahulu.

B. *Penjelasan Potongan kode untuk Solver Word Search Puzzle*

Berikut merupakan fungsi-fungsi yang digunakan dalam solver Word Search Puzzle menggunakan algoritma DFS dan Backtracking

puzzle.py

```python
DIRECTIONS = [
    (0, 1, "left to right"),
    (0, -1, "right to left"),
    (1, 0, "top to bottom"),
    (-1, 0, "bottom to top"),
    (1, 1, "diagonal top-left to bottom-right"),
    (-1, -1, "diagonal bottom-right to top-left"),
    (1, -1, "diagonal top-right to bottom-left"),
    (-1, 1, "diagonal bottom-left to top-right"),
]
```

Figure 2.1 Array berisi delapan kemungkinan arah kata

Array ini digunakan untuk menelusuri delapan kemungkinan jalur pembentukan kata dalam matriks huruf. Delapan arah tersebut mencakup seluruh kombinasi pergerakan horizontal (kanan dan kiri), vertikal (atas dan bawah), serta diagonal (empat arah). Pemilihan delapan arah ini didasarkan pada aturan word search puzzle klasik, di mana kata dapat diletakkan lurus dalam semua arah tanpa belokan.

puzzle.py

```python
def dfs(board, word, x, y, word_index, visited, path, direction=None):
    N, M = len(board), len(board[0])

    if word_index == len(word):
        return path.copy(), direction

    if (not in_bounds(x, y, N, M) or
        (x, y) in visited or
        board[x][y] != word[word_index]):
        return None

    visited.add((x, y))
    path.append((x, y))

    if direction is None:
        for dx, dy, direction_name in DIRECTIONS:
            nx, ny = x + dx, y + dy
            result = dfs(board, word, nx, ny, word_index + 1, visited, path,
            (dx, dy, direction_name))
            if result:
                return result
    else:
        dx, dy, direction_name = direction
        nx, ny = x + dx, y + dy
        result = dfs(board, word, nx, ny, word_index + 1, visited, path, direction)
        if result:
            return result

    visited.remove((x, y))
    path.pop()
    return None
```

Figure 2.2 Function dfs()

Fungsi dfs() bertanggung jawab untuk melakukan pencarian secara mendalam terhadap huruf-huruf dalam satu kata mulai dari titik awal yang cocok. Proses ini dilakukan secara rekursif, di mana setiap langkah akan memeriksa apakah huruf pada posisi saat ini sesuai dengan huruf dalam kata yang dicari. Jika cocok, pencarian akan berlanjut ke arah tertentu.

Jika pencarian mencapai seluruh huruf dalam kata (base case), maka fungsi akan mengembalikan jalur (path) yang telah ditemukan beserta arah penelusurannya. Jika ditemukan ketidaksesuaian atau simpul sudah dikunjungi sebelumnya, maka fungsi akan menghentikan pencarian dan kembali ke titik sebelumnya (proses backtracking).

Parameter visited adalah himpunan yang menyimpan koordinat huruf yang sudah dikunjungi selama proses pencarian untuk mencegah perulangan dan loop tak berujung. path merupakan daftar urutan koordinat huruf yang telah berhasil ditelusuri sejauh ini, dan akan dikembalikan sebagai solusi jika kata ditemukan. Parameter direction digunakan untuk menjaga agar pencarian tetap pada satu arah setelah arah awal ditentukan, sehingga algoritma tidak menyimpang dari jalur lurus dalam pencarian kata.

puzzle.py

```
def solve(board, words):
    N, M = len(board), len(board[0])
    results = []

    for word in words:
        found = False

        for i in range(N):
            for j in range(M):
                if board[i][j] == word[0]:
                    visited = set()
                    path = []

                    result = dfs(board, word, i, j, 0, visited, path, None)

                    if result:
                        result_path, direction_info = result
                        start = result_path[0]
                        end = result_path[-1]

                        results.append({
                            "word": word,
                            "start": (start[0]+1, start[1]+1),
                            "end": (end[0]+1, end[1]+1),
                            "direction": direction_info[2],
                            "path": result_path
                        })
                        found = True
                        break
```

puzzle.py

```
            if found:
                break

        if not found:
            results.append({
                "word": word,
                "start": None,
                "end": None,
                "direction": "not found",
                "path": []
            })

    return results
```

Figure 2.3 Function solve()

Fungsi solve() bertugas mengatur keseluruhan proses pencarian kata-kata yang terdapat pada daftar input. Fungsi ini pertama-tama membuat list kosong bernama results untuk menyimpan hasil pencarian dari masing-masing kata. Kemudian dilakukan iterasi terhadap setiap kata di dalam daftar words.

Untuk setiap kata, fungsi akan melakukan pencarian terhadap huruf pertamanya di seluruh sel dalam papan dua dimensi. Jika sebuah sel berisi huruf yang cocok dengan karakter pertama dari kata yang sedang dicari, maka proses pencarian dilanjutkan dengan memanggil fungsi dfs() dari koordinat tersebut.

Sebelum pemanggilan dfs(), dua struktur penting disiapkan: visited yang merupakan himpunan untuk menyimpan koordinat yang telah dikunjungi agar tidak terjadi pengulangan, serta path yang berisi jejak koordinat huruf-huruf yang sedang ditelusuri. Fungsi dfs() akan mengembalikan nilai jika kata ditemukan, berupa jalur koordinat (path) dan informasi arah penelusuran.

Jika kata ditemukan, fungsi akan menyimpan hasil pencarian tersebut ke dalam list results dalam bentuk dictionary berisi nama kata, posisi awal dan akhir (dalam format baris dan kolom mulai dari 1), arah penelusuran, dan jalur koordinat. Proses iterasi langsung dihentikan setelah satu kata ditemukan. Sebaliknya, jika hingga seluruh sel diperiksa kata tersebut tidak ditemukan, maka hasil dengan status "not found" juga akan disimpan dalam list results.

puzzle.py

```
def in_bounds(x, y, N, M):
    return 0 <= x < N and 0 <= y < M
```

Figure 2.4 Function in_bounds()

Fungsi in_bounds() merupkan fungsi helper untuk mencegah indeks keluar dari batas matriks saat DFS melakukan eksplorasi ke delapan arah. Validasi posisi ini memastikan setiap langkah DFS tetap berada di dalam papan,

sehingga mencegah error runtime dan menjaga keakuratan pencarian.

C. *Proses Input/Ouput Pengguna pada Command Line Interface (CLI)*

Program ini dijalankan melalui antarmuka baris perintah (Command Line Interface/CLI), yang memungkinkan interaksi langsung antara pengguna dan sistem. Struktur utama proses input/output terdapat dalam blok if __name__ == "__main__"

puzzle.py

```python
if __name__ == "__main__":
    filename = input("Masukkan path file input txt: ").strip()
    N, M, x, board, words = read_puzzle_from_txt(filename)

    start_time = time.time()
    results = solve(board, words)
    exec_time = time.time() - start_time

    found_count = sum(1 for res in results if res["start"])
    print(f"\nExecution time: {exec_time:.4f} detik")
    print(f"Banyak kata ditemukan: {found_count} dari {len(words)}")

    print("\nPuzzle dengan highlight hasil:")
    print_result(board, results)

    # Word Searching (Loop)
    while True:
        cari = input("\nCari kata (atau ketik /exit untuk keluar): ").strip()
        if cari.lower() == "/exit":
            break
        found = False
        for res in results:
            if res["word"].lower() == cari.lower() and res["start"]:
                baris, kolom = res['start']
                print(f"Kata '{cari}' ditemukan di baris ke-{baris} kolom ke-{kolom}
                arah {res['direction']}")
                found = True
                break
        if not found:
            print(f"Kata '{cari}' tidak ditemukan.")
```

Figure 2.5 Function Main()

Proses dimulai dengan meminta pengguna memasukkan path file teks berisi data puzzle. Setelah file dibaca dan data diproses, program menghitung waktu eksekusi pencarian kata menggunakan time.time() dan menampilkan jumlah kata yang berhasil ditemukan dari total target.

```
Masukkan path file input txt: 2.txt

Execution time: 0.0010 detik
Banyak kata ditemukan: 5 dari 5
```

Figure 2.6 Tampilan input dan output waktu eksekusi dan kata yang ditemukan

Hasil pencarian kemudian divisualisasikan di terminal menggunakan fungsi print_result(), di mana huruf-huruf yang termasuk dalam kata yang ditemukan diberi warna berbeda untuk memudahkan identifikasi.

```
Puzzle dengan highlight hasil:
    1   2   3   4   5   6   7   8   9   10
  +-------------------------------------+
1 | P | U | Z | Z | L | E | S | B | M | Y |
  +-------------------------------------+
2 | W | F | S | P | M | F | P | S | F | D |
  +-------------------------------------+
3 | J | E | C | D | R | O | W | A | X | J |
  +-------------------------------------+
4 | T | L | W | H | Y | M | J | L | M | G |
  +-------------------------------------+
5 | E | M | H | M | I | C | B | J | Z | Y |
  +-------------------------------------+
6 | B | A | C | K | T | R | A | C | K | Y |
  +-------------------------------------+
7 | H | N | V | E | L | X | I | U | D | V |
  +-------------------------------------+
8 | T | X | G | E | S | F | L | Y | P | O |
  +-------------------------------------+
9 | K | U | N | I | V | E | O | Z | M | X |
  +-------------------------------------+
10 | V | O | D | H | H | C | R | A | E | S |
  +-------------------------------------+
```

Figure 2.6 Tampilan output dari function print_result()

Setelah itu, program menyediakan fitur interaktif berupa loop input, di mana pengguna dapat mencari satu kata secara manual. Jika kata ditemukan, sistem menampilkan koordinat awal, arah, dan jalur huruf. Jika tidak, pengguna akan diberi tahu bahwa kata tersebut tidak ditemukan. Pengguna dapat keluar dari mode ini dengan mengetik perintah khusus /exit.

```
Cari kata (atau ketik /exit untuk keluar): search
Kata 'search' ditemukan di baris ke-10 kolom ke-10 arah right to left

Cari kata (atau ketik /exit untuk keluar): bfs
Kata 'bfs' tidak ditemukan.

Cari kata (atau ketik /exit untuk keluar): /exit
```

Figure 2.6 Fitur untuk mencari posisi suatu kata dari board

D. *Pengujian Kasus Implementasi Program*

1. Uji Kasus 1

Input:

```
14 16
10
ADTBZSTEBACENTWS
REPLEHWHEELCHAIR
TTJUGNIODZCGSZLE
LTSIRIFTRDINRPDR
UESNUZRSYRTIEEEU
FVEGRUAAWAATTEBT
OILVOSIPBGMHSPEC
TTAULWNDKOAGDKEE
GAMLOSSANNRIEASL
HDOVGSTTAFDAERTA
VLAAYIOACLORPASG
SFEIHCREKILTSKXV
STIUQHMPEEESIUGY
GFVISASGDSMGALAS
ABET
CENT
DRAGONFLIES
```

```
HARROWS
KNACKED
MELODRAMATIC
QUITS
SOUL
STRAIGHTING
URGE
```

Output:

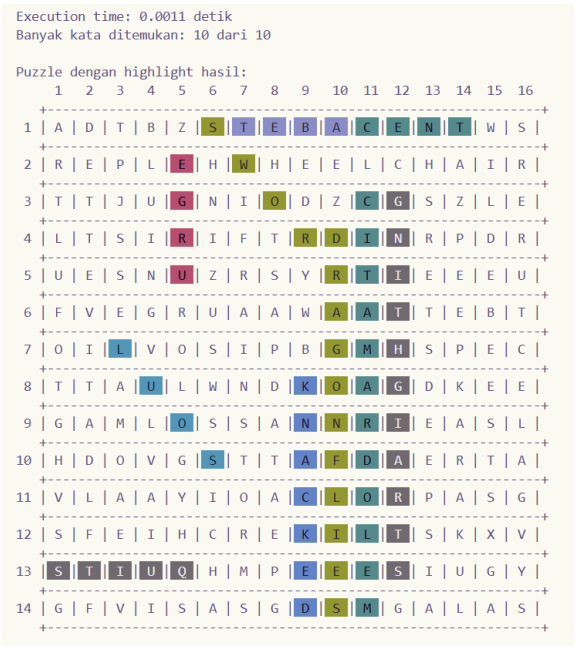

Figure 2.7 Output Uji Kasus 1

1. Uji Kasus 2

Input:

```
20 20
1
ERFRNEERFLRLUAAURFUR
FERRNALURUANENLLAULA
EURRNRNFFERFUARERUNA
RLRLAERNRENRRUNAFLFU
FAFFAAURFUNREALFRRRA
FNRRRRUARAALARNLNELU
FAARAFFULRNEENELUFFR
UFFFRARFELNFELAAREAR
ELFUREFULEUALNLFLFLE
ANNRENNEURRNRRNARNNU
NFFUAFAUUFNURNEFRUNN
EEUFNNENRANUALAUNARL
FFALLLUNFUNERLARENUF
NUEUFENULFRUAALENUAA
EEALEANFLFELNRRRNAAF
ENUURFRUUNULFLREUFEN
NRARENAEFRNFERRRAFEL
RAAEAEFRLARLNANFFENN
LAREUFLLAAEALENNLNLU
RNUELFNNUUAENEEUFANN
FUNERAL
```

Output:

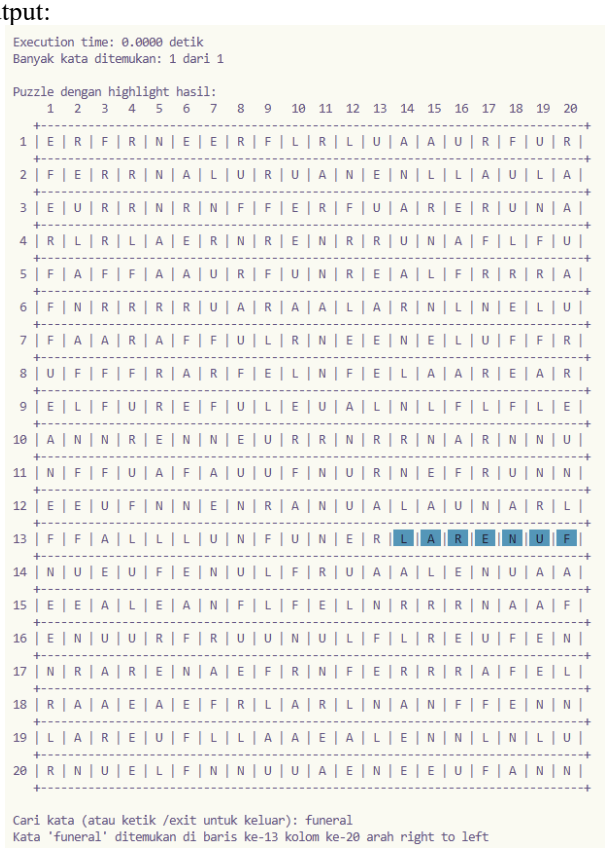

Figure 2.8 Output Uji Kasus 2

IV. KESIMPULAN

Penelitian ini menunjukkan bahwa algoritma Depth-First Search (DFS) dengan teknik backtracking dapat digunakan secara efektif dalam menyelesaikan permasalahan word search puzzle. Implementasi program dalam bahasa Python berhasil menemukan kata-kata di dalam papan huruf dengan akurasi tinggi dan waktu eksekusi yang efisien. Fitur interaktif serta visualisasi dengan pewarnaan huruf memudahkan pengguna dalam mencari hasil pencarian.

Penerapan DFS yang mendukung pencarian dalam delapan arah sesuai dengan aturan word search puzzle klasik, serta penggunaan struktur data seperti visited dan path, menjadikan pencarian lebih sistematis dan bebas dari pengulangan. Dengan desain program yang modula, sistem ini tidak hanya bermanfaat dalam konteks permainan edukatif, tetapi juga dapat dikembangkan lebih lanjut untuk kasus pencarian pola yang lebih kompleks. Penelitian ini sekaligus memperkuat pemahaman praktis tentang penerapan algoritma DFS dan Backtracking.

V. PENUTUP

REPOSITORY LINK AT GITHUB

https://github.com/amiraIzani/word-puzzle-solver.git

ACKNOWLEDGMENT

Puji dan syukur dipanjatkan ke hadirat Allah SWT atas rahmat dan karunia-Nya sehingga penulis dapat menyusun makalah ini. Penulis juga menyampaikan terima kasih kepada Pak Monterico Adrian, S.T., M.T. selaku dosen mata kuliah IF2211 Strategi Algoritma, yang telah memberikan bimbingan selama semester ini.

REFERENCES

[1] R. Munir, "Pencarian dalam Graf (BFS dan DFS) Bagian 1: Konsep Dasar dan Implementasi," 2025. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/13-BFS-DFS-(2025)-Bagian1.pdf. [Diakses 24 Juni 2025].

[2] R. Munir, "Pencarian dalam Graf (BFS dan DFS) Bagian 2: Aplikasi dan Kompleksitas," 2025. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/14-BFS-DFS-(2025)-Bagian2.pdf. [Diakses 24 Juni 2025].

[3] R. Munir, "Algoritma Backtracking Bagian 1: Prinsip dan Contoh Kasus," 2025. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/15-Algoritma-backtracking-(2025)-Bagian1.pdf. [Diakses 24 Juni 2025].

[4] R. Munir, "Algoritma Backtracking Bagian 2: Implementasi dan Studi Kasus," 2025. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/16-Algoritma-backtracking-(2025)-Bagian2.pdf. [Diakses 24 Juni 2025].

[5] GeeksforGeeks, "Difference between BFS and DFS," [Online]. Tersedia: https://www.geeksforgeeks.org/difference-between-bfs-and-dfs/. [Diakses: 24 Juni 2025].

[6] GeeksforGeeks, "Backtracking Algorithms," [Online]. Tersedia: https://www.geeksforgeeks.org/dsa/backtracking-algorithms/. [Diakses: 24 Juni 2025].

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, 24 Juni 2025

Amira Izani (13523143)