# LAPORAN TUGAS KECIL
# STRATEGI ALGORITMA
# IF 2211

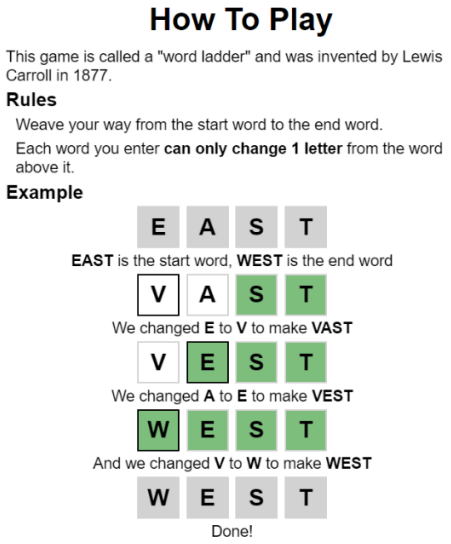

**Disusun oleh:**

**Dhafin Fawwaz Ikramullah**

**13522084**

**Daftar Isi**

## BAB I Deskripsi Masalah




**Gambar 1.** Ilustrasi dan Peraturan Permainan Word Ladder

(Sumber: : https://wordwormdormdork.com/)

**Word ladder** (juga dikenal sebagai Doublets, word-links, change-the-word puzzles, paragrams, laddergrams, atau word golf) adalah salah satu permainan kata yang terkenal bagi seluruh kalangan. Word ladder ditemukan oleh Lewis Carroll, seorang penulis dan matematikawan, pada tahun 1877. Pada permainan ini, pemain diberikan dua kata yang disebut sebagai start word dan end word. Untuk memenangkan permainan, pemain harus menemukan rantai kata yang dapat menghubungkan antara start word dan end word. Banyaknya huruf pada start word dan end word selalu sama. Tiap kata yang berdekatan dalam rantai kata tersebut hanya boleh berbeda satu huruf saja. Pada permainan ini, diharapkan solusi optimal, yaitu solusi yang meminimalkan banyaknya kata yang dimasukkan pada rantai kata. Berikut adalah ilustrasi serta aturan permainan.

**BAB II Landasan Teori**

Algoritma Uniform Cost Search merupakan sebuah algoritma pencarian untuk menelusuri graf dengan mempertimbangkan biaya dari simpul yang akan ditelusuri. Biaya dari simpul tersebut dihitung dengan mempertimbangkan simpul sekarang dengan simpul saat ini. Umumnya berupa kedalaman dari simpul saat ini dihitung dari simpul awal dan dapat dituliskan dengan notasi fungsi g(n). Biasanya fungsi g(n) ini akan dijadikan fungsi prioritas yang digunakan saat memasukkan simpul ke sebuah struktur data Priority Queue. Pada algoritma ini terdapat konsep backtracking yaitu saat sudah berhenti menulusuri sebuah jalur, maka akan kembali ke jalur lain sehingga semua jalur akan ditelusuri. Algoritma ini menjamin hasil dari pencarian memiliki solusi yang optimal.

Algoritma Greedy Best First Search merupakan sebuah algoritma pencarian untuk menelusuri graf dengan mempertimbangkan biaya dari simpul saat ini ke simpul tujuan secara heuristic. Biaya tersebut biasanya dapat dituliskan dalam notasi fungsi h(n). Sebenarnya terdapat beberapa varian dalam algoritma ini. Ada yang melakukan backtracking, tapi ada juga yang tidak. Yang akan dibahas di sini adalah yang tidak melakukan backtracking sehingga algoritma ini tidak menjamin untuk mendapatkan solusi yang optimal tapi akan melakukan pencarian dengan cepat.

Algoritma A Star dapat dibilang merupakan gabungan dari algoritma Uniform Cost Search dan algoritma Greedy Best First Search. Algoritma ini juga merupakan algoritma untuk menelusuri sebuah graf, tetapi dilakukan dengan adanya biaya yang mempertimbangkan simpul asal dengan simpul sekarang serta simpul sekarang dengan simpul tujuan. Biaya tersebut dituliskan dengan notasi fungsi f(n) yang didefinisikan dengan f(n) = g(n) + h(n). Tetapi terdapat sebuah syarat dari fungsi heuristic tersebut yaitu harus *admissible*. Maksudnya adalah perhitungan biaya yang digunakan selalu lebih kecil atau sama dengan biaya sebenarnya yang dilakukan.

## BAB III Pembahasan

Terdapat 3 jenis algoritma yang akan dibahas yaitu algoritma Uniform Cost Search, Greedy Best First Search, dan A Star. Masing-masing dari algoritma ini memiliki kelebihan dan kekurangannya masing-masing. Algoritma yang diimplementasikan akan menggunakan bahasa java. Sebelum melakukan pencarian, terdapat sebuah preprocessing terlebih dahulu untuk membersihkan serta menyimpan kamus dalam format binary agar pembacaan kamus lebih cepat. Proses ini tidak memengaruhi algoritma, hanya mempercepat inisialisasi program. Untuk setiap algoritma, kata akan dianggap sebagai simpul dari graf.

### A. Analisis dan Implementasi Uniform Cost Search

a. Algoritma

Algoritma ini diimplementasikan mirip dengan algoritma Breadth First Search namun menggunakan sebuah Priority Queue daripada queue biasa. Fungsi prioritas untuk menentukan posisi pemasukan priority queue ditentukan berdasarkan kedalaman dari sebuah simpul dari simpul asal. Semakin dalam kedalaman suatu simpul, maka akan dimasukkan ke elemen paling belakang di priority queue . Fungsi ini lah yang merupakan definisi dari fungsi g(n). Langkah dari algoritma adalah sebagai berikut.

1. Instansiasi sebuah Priority Queue yang pemasukan elemennya berdasarkan kedalaman simpul atau sama dengan fungsi g(n) pada algoritma ini.
2. Instansiasi sebuah set untuk menyimpan simpul-simpul yang sudah ditelusuri sehingga tidak perlu ditelusuri lagi.
3. Instansiasi juga sebuah larik untuk menyimpan hasil akhir pencarian.
4. Lakukan sebuah while loop sampai priority queue kosong
5. Ambil elemen terdepan (biaya terkecil) dari priority queue, cek jika simpul tersebut sudah pernah dilalui dengan mengecek di set. Jika sudah, maka skip dan ulangi ke langkah 3. Jika belum, simpan simpul tersebut ke set.
6. Jika simpul sekarang sudah merupakan tujuan, maka gunakan variabel prevNode pada struktur data GraphNode dan telusuri simpul sebelumnya (parent) hingga mencapai simpul asal. Simpan hasil penelusuran ini pada larik. Larik ini lah solusinya.
7. Jika simpul sekarang belum tujuan, maka keluarkan semua tetangga dari simpul sekarang, lalu masukkan semuanya ke priority queue berdasarkan fungsi g(n) sebelumnya

yaitu kedalaman dari simpul tersebut. Sekaligus simpan simpul sekarang sebagai simpul sebelum (prevNode) untuk semua tetangga tadi.

8. Saat keluar dari while loop, kita sudah akan mendapatkan solusi yang diinginkan.

b. Source Code

Class GraphNode (implements Comparable<GraphNode>)

```
...
public int getDepth(){
        if(cost != -1) return cost;

        if(prevNode == null){
            cost = 1;
        } else {
            cost = prevNode.getDepth() + 1;
        }
        return cost;
    }
...
    @Override
    public int compareTo(GraphNode o) {
        return getDepth() - o.getDepth(); // g(n), smaller is better
    }
...
```

Class UCSSolver (extends WordLadderSolver)

```
...
@Override
    void populateSolutionData(GraphAdjacencyMap graph, SolutionData
solutionData){
        PriorityQueue<GraphNode> queue = new PriorityQueue<GraphNode>();
        startSearch(graph, solutionData, queue);
    }
...
```

Class WorldLadderSolver

```
...
```

```
void startSearch(GraphAdjacencyMap graph, SolutionData solutionData,
PriorityQueue<GraphNode> queue){
        LinkedList<String> resultPath = solutionData.getSolutionPath();
        Set<String> visited = new HashSet<String>();

        queue.add(new GraphNode(source, null));

        while(!queue.isEmpty()){
            GraphNode currentNode = queue.poll();

            if (visited.contains(currentNode.word)) continue;
            visited.add(currentNode.word);

            if(currentNode.word.equals(destination)){
                // Trace back to the source
                GraphNode currentTracedNode = currentNode;
                while(currentTracedNode.prevNode != null){
                    resultPath.add(0, currentTracedNode.word);
                    currentTracedNode = currentTracedNode.prevNode;
                }
                resultPath.add(0, source);
                break;
            }

            ArrayList<GraphNode> neighborList = graph.get(currentNode.word);

            for(GraphNode neighbor : neighborList){
                neighbor.prevNode = currentNode;
                queue.add(neighbor);
            }
        }

        solutionData.setNodeVisited(visited.size());
    }
...
```

c. Analisis

Algoritma Uniform Cost Search akan menghasilkan solusi yang optimal. Pada kasus word ladder, algoritma ini memiliki kesamaan dengan Breadth First Search. Kesamaannya ada pada urutan simpul yang diekspan. Walaupun pada Breadth First Search simpul yang diekspan berurutan dan pada Uniform Cost Search simpul yang diekspan dilakukan berdasarkan biaya dari simpul tersebut atau fungsi g(n), urutan simpul yang diekspan akan sama karena fungsi g(n) yang definisinya merupakan kedalaman suatu simpul. Saat melakukan pencarian, kita akan menelusuri semua simpul di level tertentu hingga semuanya ditelusuri. Sudah pasti kedalaman dari level tertentu tersebut akan sama sehingga urutan ekspansi simpulnya juga akan sama.

**B. Analisis dan Implementasi Algoritma Greedy Best First Search**

a. Algoritma

Algoritma ini diimplementasikan dengan menelusuri tetangga dengan biaya terkecil tanpa melakukan backtracking. Akibat dari hal ini solusi yang diberikan bisa saja tidak optimal atau bahkan tidak menghasilkan solusi. Biaya pada algoritma ini dihitung berdasarkan jumlah perbedaan huruf dari simpul sekarang dengan simpul tujuan. Biaya ini lah yang merupakan definisi dari fungsi heuristic h(n). Langkah dari algoritma ini adalah sebagai berikut.

1. Instansiasi sebuah larik untuk menyimpan solusi dari pencarian
2. Instansiasi sebuah set untuk menyimpan simpul-simpul yang sudah ditelusuri agar tidak perlu ditelusuri lagi
3. Cek dulu jika simpul sekarang sudah merupakan tujuan, maka langsung tetapkan ini lah solusinya
4. Jika tidak, keluarkan semua tetangga dari simpul sekarang dan lakukan while loop sampai tetangga dari simpul sekarang kosong.
5. Di dalam while loop, simpan simpul sekarang sebagai elemen berikutnya dari larik solusi dan simpan juga simpul sekarang ke set.
6. Cari simpul dengan biaya paling rendah diantara semua tetangga dari simpul sekarang dengan fungsi heuristic h(n), dan pastikan semua tetangga tersebut belum ditelusuri dengan menggunakan set yang sudah diinstansiasi di awal. Jika tidak ditemukan, artinya solusi tidak ada.

7. Cek jika simpul yang ditemukan merupakan simpul tujuan. Jika iya, tetapkan larik sebagai solusi dan keluar dari while loop.
8. Ubah simpul sekarang menjadi simpul yang ditemukan tadi.
9. Ulangi langkah 5 hingga solusi ditemukan.
10. Saat keluar while loop, cek apakah simpul sekarang merupakan simpul solusi. Jika tidak maka solusinya tidak ada. Jika iya, maka tetapkan larik sebagai solusi.

b. Source Code

class GBFSSolver (extends Comparator<GraphNode>)

```
@Override
    void populateSolutionData(GraphAdjacencyMap graph, SolutionData
solutionData){
        LinkedList<String> resultPath = solutionData.getSolutionPath();
        Set<String> visited = new HashSet<String>();

        GraphNode currentNode = new GraphNode(source, null);
        if(currentNode.word.equals(destination)){ // if source is the
destination
            resultPath.add(currentNode.word);
            solutionData.setSolution(resultPath);
            return;
        }

        ArrayList<GraphNode> currentNeighborList = graph.get(source);
        while (!currentNeighborList.isEmpty()) {
            resultPath.add(currentNode.word);
            visited.add(currentNode.word);

            GraphNode minimumNode = findMinimumNodeNotInVisited(currentNode,
graph.get(currentNode.word), visited);
            if(minimumNode == null) break;

            if(minimumNode.word.equals(destination)){
                resultPath.add(minimumNode.word);
                currentNode = minimumNode;
```

```
                break;
            }

            currentNode = minimumNode;
            currentNeighborList = graph.get(currentNode.word);
        }

        if(!currentNode.word.equals(destination)){
            solutionData.setSolution(new LinkedList<String>());
        } else {
            solutionData.setSolution(resultPath);
        }

        solutionData.setNodeVisited(visited.size());
    }
```

c. Analisis

Pada persoalan world ladder, algoritma ini memang akan menghasilkan solusi dengan cepat karena tidak adanya backtracking. Tetapi hal ini juga mengakibatkan solusi yang dihasilkan tidak dijamin optimal. Bahkan bisa saja solusi tidak ditemukan. Ini terjadi karena bisa saja saat memilih simpul dengan biaya paling kecil, simpul tersebut mengarah ke simpul solusi dengan lebih jauh atau bahkan tidak mengarah ke solusi sama sekali. Sementara bisa saja simpul yang memiliki biaya lebih besar di awal mengarah ke solusi dengan lebih dekat.

## C. Analisis dan Implementasi Algoritma A Star (A*)

a. Algoritma

Algoritma ini diimplementasikan dengan priority queue untuk menyimpan simpul yang akan diekspan berdasarkan biaya dari simpul tersebut. Digunakan 2 buah fungsi untuk menentukan biaya sebuah simpul yaitu g(n) dan h(n). Kedua fungsi tersebut bisa digabung menjadi fungsi f(n) yaitu f(n) = g(n) + h(n). g(n) merupakan biaya dari kedalaman simpul dari simpul asal. h(n) merupakan fungsi heuristic yaitu jumlah perbedaan huruf dari simpul sekarang dengan simpul tujuan. Jumlah dari g(n) dan h(n) (yaitu f(n)) inilah yang akan digunakan untuk

menentukan posisi dari elemen yang dimasukkan ke priority queue. Langkah dari algoritma ini adalah sebagai berikut:

1. Instansiasi sebuah Priority Queue yang pemasukan elemennya berdasarkan kedalaman simpul dan perbedaan huruf simpul dengan simpul tujuan atau sama dengan fungsi f(n) pada algoritma ini.
2. Instansiasi sebuah set untuk menyimpan simpul-simpul yang sudah ditelusuri sehingga tidak perlu ditelusuri lagi.
3. Instansiasi juga sebuah larik untuk menyimpan hasil akhir pencarian.
4. Lakukan sebuah while loop sampai priority queue kosong
5. Ambil elemen terdepan (biaya terkecil) dari priority queue, cek jika simpul tersebut sudah pernah dilalui dengan mengecek di set. Jika sudah, maka skip dan ulangi ke langkah 3. Jika belum, simpan simpul tersebut ke set.
6. Jika simpul sekarang sudah merupakan tujuan, maka gunakan variabel prevNode pada struktur data GraphNode dan telusuri simpul sebelumnya (parent) hingga mencapai simpul asal. Simpan hasil penelusuran ini pada larik. Larik ini lah solusinya.
7. Jika simpul sekarang belum tujuan, maka keluarkan semua tetangga dari simpul sekarang, lalu masukkan semuanya ke priority queue berdasarkan fungsi g(n) sebelumnya yaitu kedalaman dari simpul tersebut. Sekaligus simpan simpul sekarang sebagai simpul sebelum (prevNode) untuk semua tetangga tadi.
8. Saat keluar dari while loop, kita sudah akan mendapatkan solusi yang diinginkan.

b. Source Code

Class AStarComparator (extends GBFSComparator)

```
...
// Basically the g(n) + h(n)
// g(n) is the cost from source to current
// h(n) is the heuristic function that calculated according to current to
destination
@Override
    public int compare(GraphNode a, GraphNode b){
        int gA = a.getDepth();
        int hA = wordDifference(a.word, destination);
```

```
        int fA = gA + hA;

        int gB = b.getDepth();
        int hB = wordDifference(b.word, destination);
        int fB = gB + hB;

        return fA - fB; // smaller difference is better
    }
...
```

Class AStarSolver (extends WordLadderSolver)

```
...
@Override
    void populateSolutionData(GraphAdjacencyMap graph, SolutionData
solutionData){
        PriorityQueue<GraphNode> queue = new
PriorityQueue<GraphNode>(comparator);
        startSearch(graph, solutionData, queue);
    }
...
```

Class WordLadderSolver

```
...
void startSearch(GraphAdjacencyMap graph, SolutionData solutionData,
PriorityQueue<GraphNode> queue){
        LinkedList<String> resultPath = solutionData.getSolutionPath();
        Set<String> visited = new HashSet<String>();

        queue.add(new GraphNode(source, null));

        while(!queue.isEmpty()){
            GraphNode currentNode = queue.poll();

            if (visited.contains(currentNode.word)) continue;
            visited.add(currentNode.word);

            if(currentNode.word.equals(destination)){
```

```
                // Trace back to the source
                GraphNode currentTracedNode = currentNode;
                while(currentTracedNode.prevNode != null){
                    resultPath.add(0, currentTracedNode.word);
                    currentTracedNode = currentTracedNode.prevNode;
                }
                resultPath.add(0, source);
                break;
            }

            ArrayList<GraphNode> neighborList = graph.get(currentNode.word);

            for(GraphNode neighbor : neighborList){
                neighbor.prevNode = currentNode;
                queue.add(neighbor);
            }
        }

        solutionData.setNodeVisited(visited.size());
    }
...
```

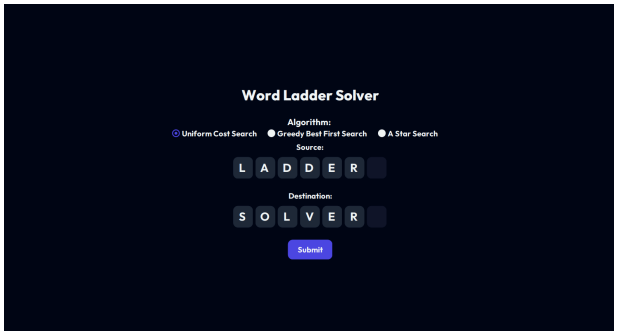
c. Analisis

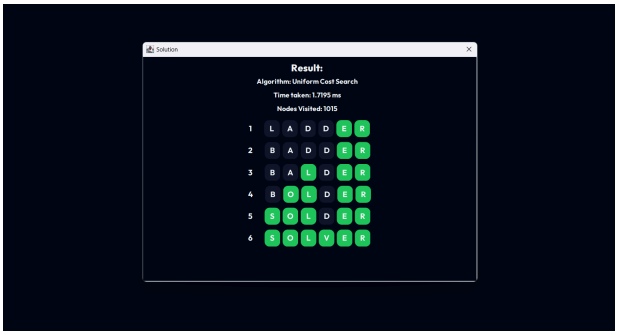
Algoritma ini dapat menjamin solusi optimal dan memiliki waktu eksekusi yang lebih cepat dibandingkan Uniform Cost Search. Solusi yang dihasilkan dijamin optimal karena algoritma yang digunakan *admissible*. Maksudnya adalah perhitungan biaya yang digunakan selalu lebih kecil atau sama dengan biaya sebenarnya yang dilakukan. Ini dapat dipastikan benar karena setiap simpul selanjutnya dari solusi pasti memiliki perbedaan 1 huruf. Algoritma ini juga akan sering lebih efisien dibanding Uniform Cost Search karena terdapat tambahan fungsi untuk menentukan simpul mana yang sebaiknya diekspan terlebih dahulu secara heuristic.

**D. Analisis dan Implementasi Graphical User Interface**

Graphical User Interface dibuat dengan cukup sederhana dan dikembangkan dengan kakas java swing. Berikut struktur foldernya.


Folder Asset berisi kamus dari algoritma yang akan diterapkan. Component berisi komponen-komponen untuk GUI, Solver berisi implementasi dari algoritma itu sendiri. Untuk Tampilan GUI sendiri yaitu sebagai berikut.



## BAB IV Hasil Pengujian

### A. Pengujian Waktu Eksekusi

Untuk mempermudah pengujian, disediakan command khusus test case yang dapat digunakan oleh yang ingin melakukan testing dengan test case dalam jumlah besar. Berikut ini dihasilkan dari satu buah input dalam file input.txt berisi:

```
funny bird
wheat toast
brute force
work late
waffle house
picking peaches
```

Digunakan pula kamus pada url berikut:
https://docs.oracle.com/javase/tutorial/collections/interfaces/examples/dictionary.txt

Lalu dijalankan dengan input command

```
java -cp bin TestCaseRunner ./test/input.txt ./src/Asset/dictionary.bin > ./test/output.txt
```

Berikut ini hasil test case secara terpisah

```
Funny → Bird

Algorithm: Uniform Cost Search
1.   funny
2.   bunny
3.   bunns
4.   bunds
5.   burds
6.   birds
Nodes visited: 510
Time taken: 0.6157 ms

Algorithm: Greedy Best First Search
1.   funny
2.   bunny
3.   bunns
4.   bunds
5.   binds
6.   birds
Nodes visited: 5
Time taken: 0.2954 ms

Algorithm: A* Search
1.   funny
2.   bunny
3.   bunns
```

```
Wheat → Toast

Algorithm: Uniform Cost Search
1.   wheat
2.   cheat
3.   cleat
4.   clept
5.   clapt
6.   coapt
7.   coast
8.   toast
Nodes visited: 929
Time taken: 0.7039 ms

Algorithm: Greedy Best First Search
No solution found.
Time taken: 0.185 ms

Algorithm: A* Search
1.   wheat
2.   cheat
3.   cleat
4.   bleat
5.   blest
6.   blast
7.   boast
```

```
4.   bunds
5.   burds
6.   birds
Nodes visited: 9
Time taken: 0.2567 ms
```

```
8.   toast
Nodes visited: 24
Time taken: 0.1663 ms
```

Brute → Force

```
Algorithm: Uniform Cost Search
1.   brute
2.   brume
3.   blume
4.   blame
5.   flame
6.   flams
7.   foams
8.   forms
9.   forme
10.  force
Nodes visited: 2832
Time taken: 4.8696 ms

Algorithm: Greedy Best First Search
No solution found.
Time taken: 0.7382 ms

Algorithm: A* Search
1.   brute
2.   brume
3.   blume
4.   blame
5.   blams
6.   flams
7.   foams
8.   forms
9.   forme
10.  force
Nodes visited: 88
Time taken: 0.9396 ms
```

Work → Late

```
Algorithm: Uniform Cost Search
1.   work
2.   wark
3.   wars
4.   lars
5.   lats
6.   late
Nodes visited: 1890
Time taken: 2.0635 ms

Algorithm: Greedy Best First Search
1.   work
2.   wark
3.   lark
4.   lack
5.   lace
6.   late
Nodes visited: 5
Time taken: 0.1696 ms

Algorithm: A* Search
1.   work
2.   wore
3.   fore
4.   fare
5.   fate
6.   late
Nodes visited: 36
Time taken: 0.1815 ms
```

Waffle → House

```
Algorithm: Uniform Cost Search
No solution found.
Time taken: 0.4092 ms

Algorithm: Greedy Best First Search
No solution found.
Time taken: 0.3926 ms
```

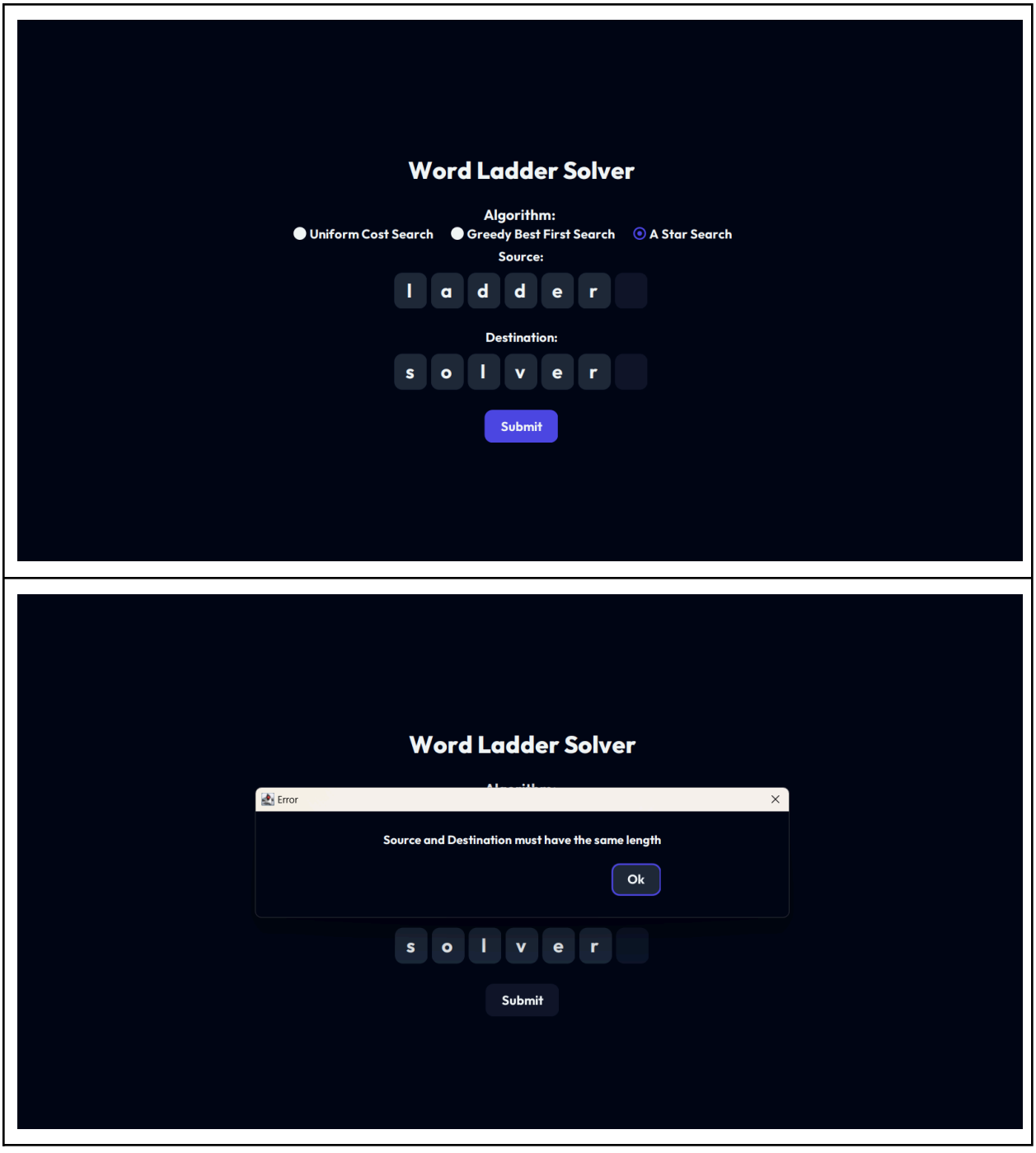
Picking → Peaches

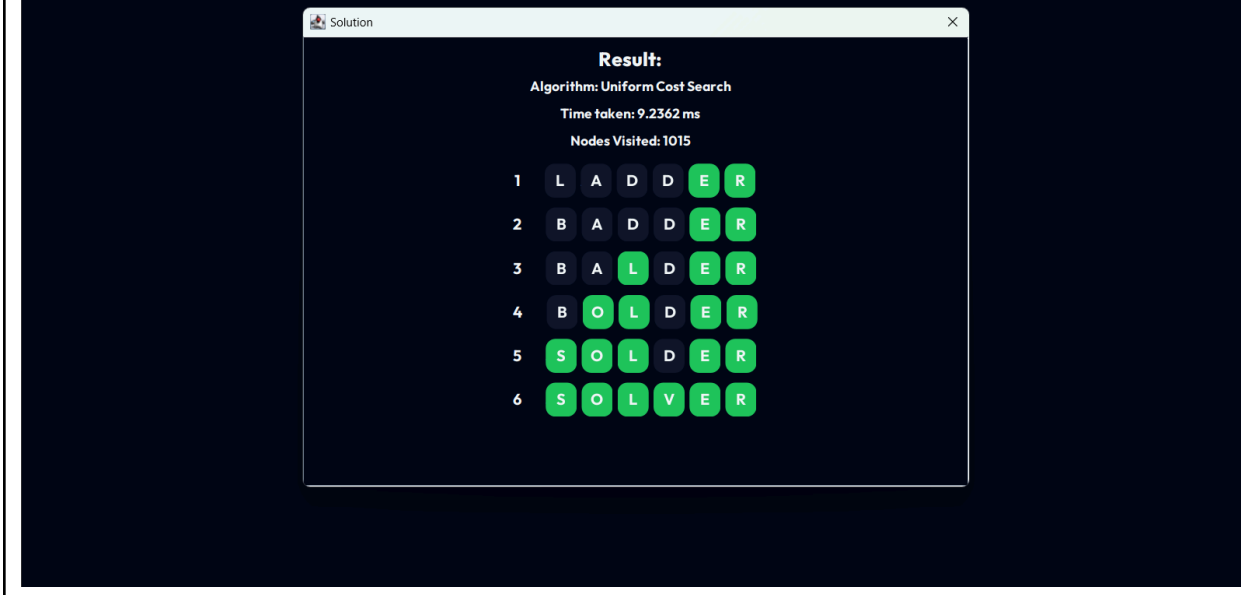
```
Algorithm: Uniform Cost Search
1.   picking
2.   pecking
3.   perking
4.   jerking
5.   jerkins
6.   jerkies
```

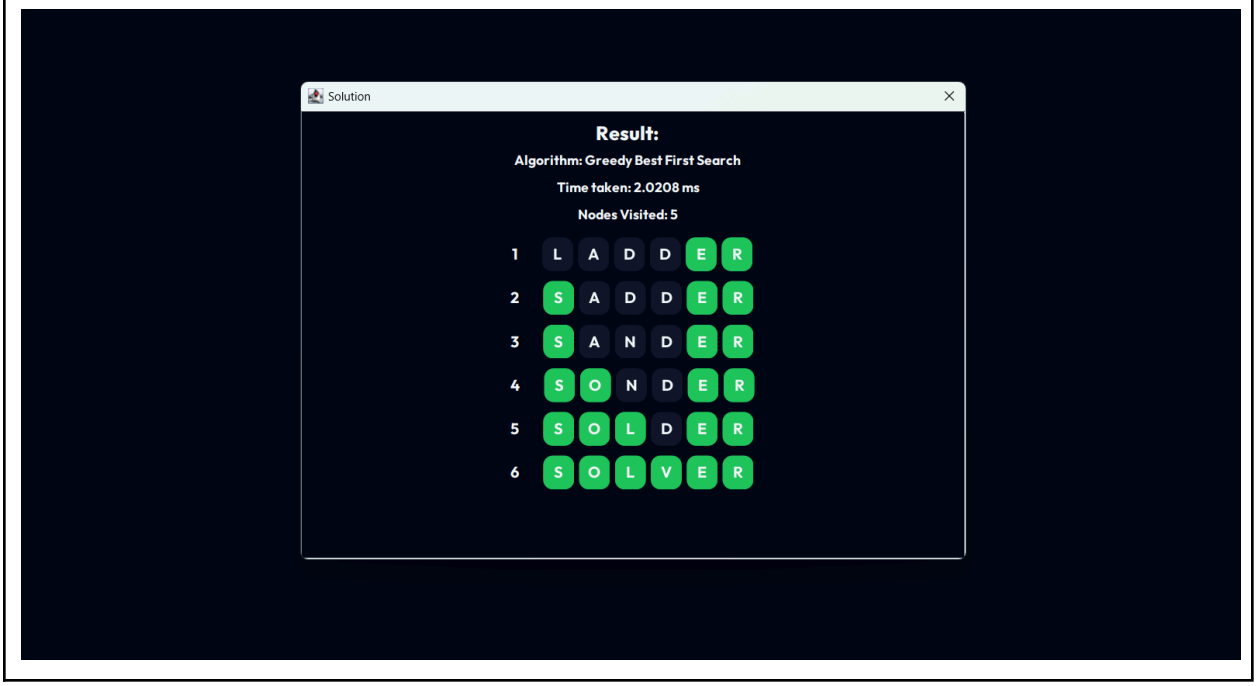
```
Algorithm: A* Search
No solution found.
Time taken: 0.3826 ms
```

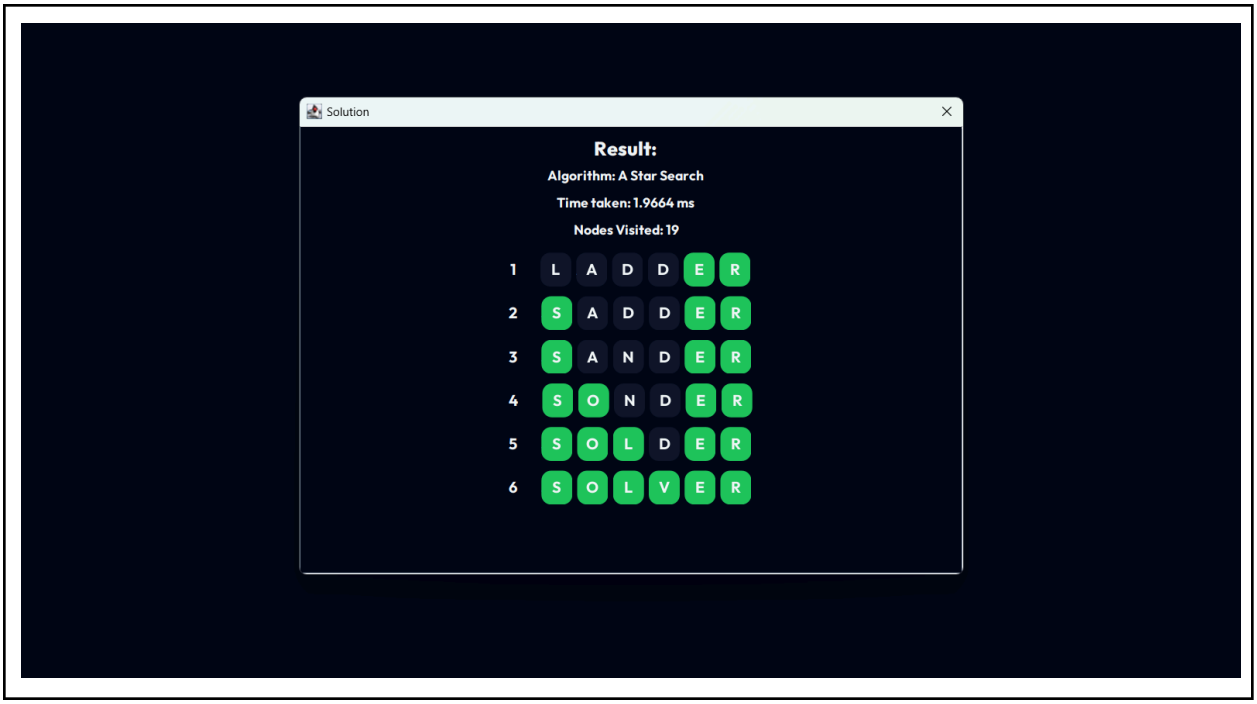
```
7.   jerkier
8.   perkier
9.   peakier
10.  beakier
11.  brakier
12.  brasier
13.  brasher
14.  brashes
15.  braches
16.  beaches
17.  peaches
Nodes visited: 4408
Time taken: 17.349 ms

Algorithm: Greedy Best First Search
No solution found.
Time taken: 3.8665 ms

Algorithm: A* Search
1.   picking
2.   pecking
3.   perking
4.   jerking
5.   jerkins
6.   jerkies
7.   jerkier
8.   perkier
9.   peakier
10.  beakier
11.  brakier
12.  brasier
13.  brasher
14.  brashes
15.  braches
16.  beaches
17.  peaches
Nodes visited: 2025
Time taken: 19.5632 ms
```

## B. Pengujian Graphical User Interface




Solution
Result:
Algorithm: Uniform Cost Search
Time taken: 9.2362 ms
Nodes Visited: 1015
1 L A D D E R
2 B A D D E R
3 B A L D E R
4 B O L D E R
5 S O L D E R
6 S O L V E R


Solution
Result:
Algorithm: Greedy Best First Search
Time taken: 2.0208 ms
Nodes Visited: 5
1 L A D D E R
2 S A D D E R
3 S A N D E R
4 S O N D E R
5 S O L D E R
6 S O L V E R


Solution
Result:
Algorithm: A Star Search
Time taken: 1.9664 ms
Nodes Visited: 19
1 L A D D E R
2 S A D D E R
3 S A N D E R
4 S O N D E R
5 S O L D E R
6 S O L V E R

**C. Analisis Hasil Pengujian**

Berdasarkan hasil pengujian dapat disimpulkan beberapa hal. Algoritma Uniform Cost Search dan A Star akan menjamin solusi optimal. Sedangkan Algoritma Greedy Best First Search belum tentu. Hal ini terjadi karena pada algoritma Greedy Best First Search, kita tidak melakukan backtracking sehingga jika simpul yang dilalui tidak optimal atau bahkan tidak mengarah ke solusi, maka kesalahan tersebut tidak akan diperbaiki. Berbeda dengan algoritma Uniform Cost Search dan A Star yang melakukan backtracking sehingga akan menjamin solusi optimal. Algoritma A Star sendiri, seperti yang dibahas sebelumnya, memiliki fungsi heuristic yang *admissible* sehingga dipastikan solusinya optimal.

Berdasarkan waktu eksekusi, urutan algoritma dari yang tercepat adalah Greedy Best First Search, A Star, Uniform Cost Search. Hal ini terjadi karena pada algoritma Greedy Best First Search, kita tidak melakukan backtracking. Jika simpul yang dilalui tidak optimal, kita tidak kembali ke simpul sebelumnya untuk memperbaiki kesalahan. Pada A Star, kita melakukan backtracking tapi tetap ada sebuah fungsi heuristic untuk memilih simpul mana dulu yang paling efisien untuk diekspan sehingga membuat algoritma lebih efisien. Sedangkan Uniform Cost Search juga melakukan backtracking tapi tanpa adanya sebuah fungsi heuristic untuk memilih simpul yang lebih baik untuk diekspan.

Berdasarkan jumlah simpul yang dilalui, atau juga dapat merepresentasikan memori yang dibutuhkan secara tidak langsung, dari yang terkecil adalah Greedy Best First Search, A Star, Uniform Cost Search. Hal ini terjadi karena simpul pada Greedy Best First disimpan pada sebuah larik dan hanya menyimpan simpul yang akan dijadikan kandidat untuk pemilihan simpul berikutnya saja dan jika tidak terpilih, maka simpul tersebut tidak dilalui. Uniform Cost Search dan A Star menggunakan Priority Queue dan akan menyimpan seluruh simpul pada kedalaman tertentu.

## BAB V Kesimpulan

Berbagai persoalan pemrograman dapat diselesaikan dengan berbagai algoritma. Salah satunya adalah algoritma Uniform Cost Search, Greedy Best First Search, dan A Star (A*) pada permainan world ladder. Masing-masing dari algoritma ini dapat digunakan untuk mencari solusi pada sebuah permainan bernama World Ladder. Dengan dibuktikan melalui pengujian, algoritma A Star akan menjamin solusi optimal serta tetap memiliki waktu eksekusi yang efisien. Algoritma Greedy Best First Search memang lebih cepat, tapi terdapat kemungkinan bahwa solusi yang diberikan tidak optimal atau bahkan tidak memiliki solusi. Sedangkan untuk algoritma Uniform Cost Search akan menjamin solusi yang optimal tapi masih kurang efisien.

Dari hasil pengujian kita dapat menyimpulkan bahwa algoritma paling baik jika ingin solusi yang optimal dan tetap memiliki waktu eksekusi yang efisien dan hemat memori, yang terbaik adalah algoritma A Star.

**BAB VI Lampiran**

Source code: https://github.com/DhafinFawwaz/Tucil3_13522084

| Poin | Ya | Tidak |
|---|---|---|
| 1. Program berhasil dijalankan. | ✓ | |
| 2. Program dapat menemukan rangkaian kata dari start word ke end word sesuai aturan permainan dengan algoritma UCS | ✓ | |
| 3. Solusi yang diberikan pada algoritma UCS optimal | ✓ | |
| 4. Program dapat menemukan rangkaian kata dari start word ke end word sesuai aturan permainan dengan algoritma Greedy Best First Search | ✓ | |
| 5. Program dapat menemukan rangkaian kata dari start word ke end word sesuai aturan permainan dengan algoritma A* | ✓ | |
| 6. Solusi yang diberikan pada algoritma A* optimal. | ✓ | |
| 7. **[Bonus]**: Program memiliki tampilan GUI. | ✓ | |