

LAPORAN TUGAS KECIL III
IF2211 STRATEGI ALGORITMA

Penyelesaian Permainan Word Ladder Menggunakan Algoritma UCS, Greedy Best First Search, dan A*



Disusun oleh
Francesco Michael Kusuma
13522038

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2023/2024

BAGIAN I

ANALISIS DAN IMPLEMENTASI

Algoritma UCS(Uniform Cost Search) adalah algoritma pencarian yang digunakan untuk mencari jalur terpendek dalam sebuah graf dengan bobot dari sumber pada setiap lintasannya. Algoritma ini mempertimbangkan semua jalur yang mungkin dari titik awal ke titik tujuan dan memilih jalur dengan biaya terendah pada setiap langkahnya. Pada permainan Word Ladder, biaya yang digunakan/g(n) adalah jarak node terdekat dari node sumber kata dan node sekarang. Sehingga, node yang ada saat ini memiliki biaya sebesar biaya node sebelumnya dijumlah 1. Untuk node sumber kata memiliki biaya 0. Implementasi algoritma UCS yang digunakan adalah sebagai berikut

1. Algoritma ini menggunakan kelas Node untuk menyimpan kata dengan tipe data string, biaya dengan tipe data integer, dan parent dengan tipe data Node. Selain itu, struktur data Priority Queue juga digunakan untuk menyimpan Node dan diurutkan mulai dari yang terkecil berdasarkan biaya yang dimiliki oleh Node. Lalu buat suatu list untuk bisa menampung kata-kata yang telah dikunjungi.
2. Buat kata awal menjadi Node dengan biaya 0 dan parent berisi null dan dimasukkan ke dalam Priority Queue.
3. Algoritma dijalankan selama Priority Queue masih belum kosong atau Node yang dikunjungi berisi kata tujuan yang ingin dicapai.
4. Ambil elemen terdepan pada Priority Queue dan masukkan kata yang ada pada Node ke list kata yang sudah dikunjungi.
5. Periksa apakah kata yang dikunjungi saat ini adalah kata tujuan atau bukan.
6. Jika kata yang dicapai adalah kata tujuan, track parent dari setiap Node untuk menciptakan jalur yang telah dilalui dan langsung kembali untuk memberikan hasil dari algoritma.
7. Jika kata yang dicapai bukanlah kata tujuan, cari kata-kata yang memungkinkan untuk bisa menjadi Node selanjutnya dan menjadi tetangga dari Node sekarang.
8. Buat Node baru dari setiap tetangga yang terhubung dengan Node saat ini.
9. Masukkan seluruh Node tersebut pada Priority Queue dengan kata adalah kata yang baru, biaya adalah biaya yang sekarang dijumlah 1, dan parent adalah Node saat ini
10. Ulangi tahap 4 hingga 10.
11. Jika Priority Queue sudah kosong, tetapi belum ada path yang terbentuk, artinya tidak ada path yang bisa mencapai kata tujuan.

Algoritma Greedy Best First Search adalah algoritma pencarian yang berbasis pada nilai heuristik. Algoritma ini mencari jalur dengan mempertimbangkan nilai heuristik dari setiap simpul. Pada permainan Word Ladder, nilai heuristik suatu kata/h(n) adalah jumlah huruf yang berbeda pada setiap indeks kata. Contoh, terdapat kata kamu dan tujuannya adalah kami, nilai heuristik dari kata kamu adalah 1 karena terdapat perbedaan huruf pada huruf terakhir. Contoh berikutnya adalah kata stock dan tujuannya adalah price, nilai heuristik dari kata stock adalah 4. Implementasi algoritma Greedy Best First Search yang digunakan adalah sebagai berikut

1. Algoritma ini tidak membutuhkan struktur data tambahan papun, tetapi hanya butuh untuk bisa menyimpan kata yang telah dikunjungi untuk mendeteksi siklus dan bahkan bisa menjadi jalur yang telah dilalui.
2. Algoritma berhalan selama path belum ditemukan dan kata yang dikunjungi belum dikunjungi sebelumnya.

3. Masukkan kata awal ke daftar kata yang telah dikunjungi dan menjadi kata yang dikunjungi saat ini.
4. Jika kata yang dikunjungi saat ini adalah kata tujuan, berikan hasil path dari algoritma.
5. Jika kata yang dikunjungi saat ini bukanlah kata tujuan, cari daftar kata yang bisa menjadi tetangga dari kata saat ini.
6. Jika tetangga kosong, artinya sudah tidak ada kata yang bisa mencapai kata tujuan, sehingga balikkan hasil bahwa tidak ada jalur.
7. Jika tetangga tidak kosong, cari kata mana yang memiliki nilai heuristik terkecil.
8. Ulang langkah 3 hingga 8.

Algoritma A Star adalah algoritma pencarian yang menggabungkan algoritma UCS dan algoritma Greedy Best First Search. Pada permainan Word Ladder, biaya yang digunakan algoritma ini adalah penjumlahan biaya dari kedua algoritma tersebut. Oleh karena itu, biaya tiap node adalah $f(n) = g(n) + h(n)$. Algoritma ini digunakan untuk mencari jalur terpendek atau solusi optimal antara dua titik dalam graf atau ruang pencarian. Implementasi algoritma A Star yang digunakan adalah sebagai berikut

1. Algoritma ini menggunakan kelas Node untuk menyimpan kata dengan tipe data string, biaya $g(n)$ dengan tipe data integer, biaya heuristik/ $h(n)$ dengan tipe data integer, biaya total dengan tipe data integer, dan parent dengan tipe data Node. Selain itu, struktur data Priority Queue juga digunakan untuk menyimpan Node dan diurutkan mulai dari yang terkecil berdasarkan biaya total yang dimiliki oleh Node. Lalu buat suatu list untuk bisa menampung kata-kata yang telah dikunjungi.
2. Buat kata awal menjadi Node dengan biaya $g(n)$ bernilai 0, biaya heuristik dengan menghitung nilai heuristik/ $h(n)$ dari kata awal menuju kata akhir, biaya total dengan menjumlahkan $g(n) + h(n)$, dan parent berisi null dan dimasukkan ke dalam Priority Queue.
3. Algoritma dijalankan selama Priority Queue masih belum kosong atau Node yang dikunjungi berisi kata tujuan yang ingin dicapai.
4. Ambil elemen terdepan pada Priority Queue dan masukkan kata yang ada pada Node ke list kata yang sudah dikunjungi.
5. Periksa apakah kata yang dikunjungi saat ini adalah kata tujuan atau bukan.
6. Jika kata yang dicapai adalah kata tujuan, track parent dari setiap Node untuk menciptakan jalur yang telah dilalui dan langsung kembali untuk memberikan hasil dari algoritma.
7. Jika kata yang dicapai bukanlah kata tujuan, cari kata-kata yang memungkinkan untuk bisa menjadi Node selanjutnya dan menjadi tetangga dari Node sekarang.
8. Buat Node baru dari setiap tetangga yang terhubung dengan Node saat ini.
9. Masukkan seluruh Node tersebut pada Priority Queue dengan kata adalah kata yang baru, biaya $g(n)$ adalah biaya $g(n)$ yang sekarang dijumlah 1, biaya heurstik $h(n)$ adalah nilai heuristik/ $h(n)$ dari kata yang baru dengan kata tujuan, biaya total adalah penjumlahan antara $g(n)+h(n)$, dan parent adalah Node saat ini
10. Ulangi tahap 4 hingga 10.
11. Jika Priority Queue sudah kosong, tetapi belum ada path yang terbentuk, artinya tidak ada path yang bisa mencapai kata tujuan.

Nilai heuristik yang digunakan pada algoritma GBFS dan algoritma A Star adalah menggunakan jarak Levenshtein antara kata saat ini dan kata tujuan dengan menghitung

berapa banyak karakter yang berbeda di antara keduanya. Jadi, fungsi heuristik ini memberikan nilai yang menggambarkan seberapa dekat atau jauhnya kata saat ini dari kata tujuan. Semakin sedikit perubahan yang diperlukan untuk mengubah kata saat ini menjadi kata tujuan, semakin kecil nilai heuristiknya, dan semakin dekat kata tersebut dengan tujuan. Tujuan tercapai saat nilai heuristik bernilai 0.

Misalkan nilai heuristik kata n adalah $h(n)$ dan nilai sesungguhnya langkah terpendek kata n ke tujuan adalah $h^*(n)$. Dengan menggunakan definisi fungsi heuristik yang telah disebutkan, dapat dijamin bahwa $h(n)$ akan selalu lebih kecil sama dengan $h^*(n)$. Sebagai contoh, kata belt ke debt. Nilai $h(belt)$ adalah 2, sedangkan nilai $h^*(belt)$ adalah 3 (dengan belt-beet-deet-debt). Ini menunjukkan bahwa fungsi heuristik ini admissible. Oleh karena itu, algoritma A Star yang digunakan pasti memberikan hasil yang optimal.

Pada kasus Word Ladder, algoritma UCS bisa dianggap sama dengan algoritma BFS. Ini dapat terjadi karena jarak antar kata yang bertetangga semua dianggap sama. Oleh karena itu, algoritma UCS yang menelusuri tetangga terdekatnya terlebih dahulu dapat diimplementasikan menggunakan algoritma BFS dan mempertimbangkan jumlah langkah tanpa memperhitungkan biaya. Namun, perlu diperhatikan bahwa di luar permainan Word Ladder, seperti dalam masalah di mana biaya setiap langkah berbeda, UCS dan BFS akan memberikan hasil yang berbeda karena cara mereka mempertimbangkan biaya. UCS akan mempertimbangkan biaya aktual, sementara BFS hanya mempertimbangkan kedalaman langkah.

Secara teoritis, algoritma A Star akan memberikan hasil yang lebih efisien dibanding dengan algoritma UCS. Ini dapat terjadi karena sebenarnya algoritma A Star melakukan hal yang sama dengan algoritma UCS, tetapi pemilihan node yang sama-sama dekat didasarkan pada nilai heuristik yang terkecil sehingga dapat meningkatkan efisiensi proses pencarian. Namun, meskipun A Star dapat memberikan hasil yang lebih efisien, perlu diingat bahwa efisiensi ini bergantung pada admissibility nilai heuristik pada algoritma. Jika nilai heuristik yang digunakan tidak admissible, bisa jadi algoritma A Star tidak memberikan hasil yang optimal dan efisien.

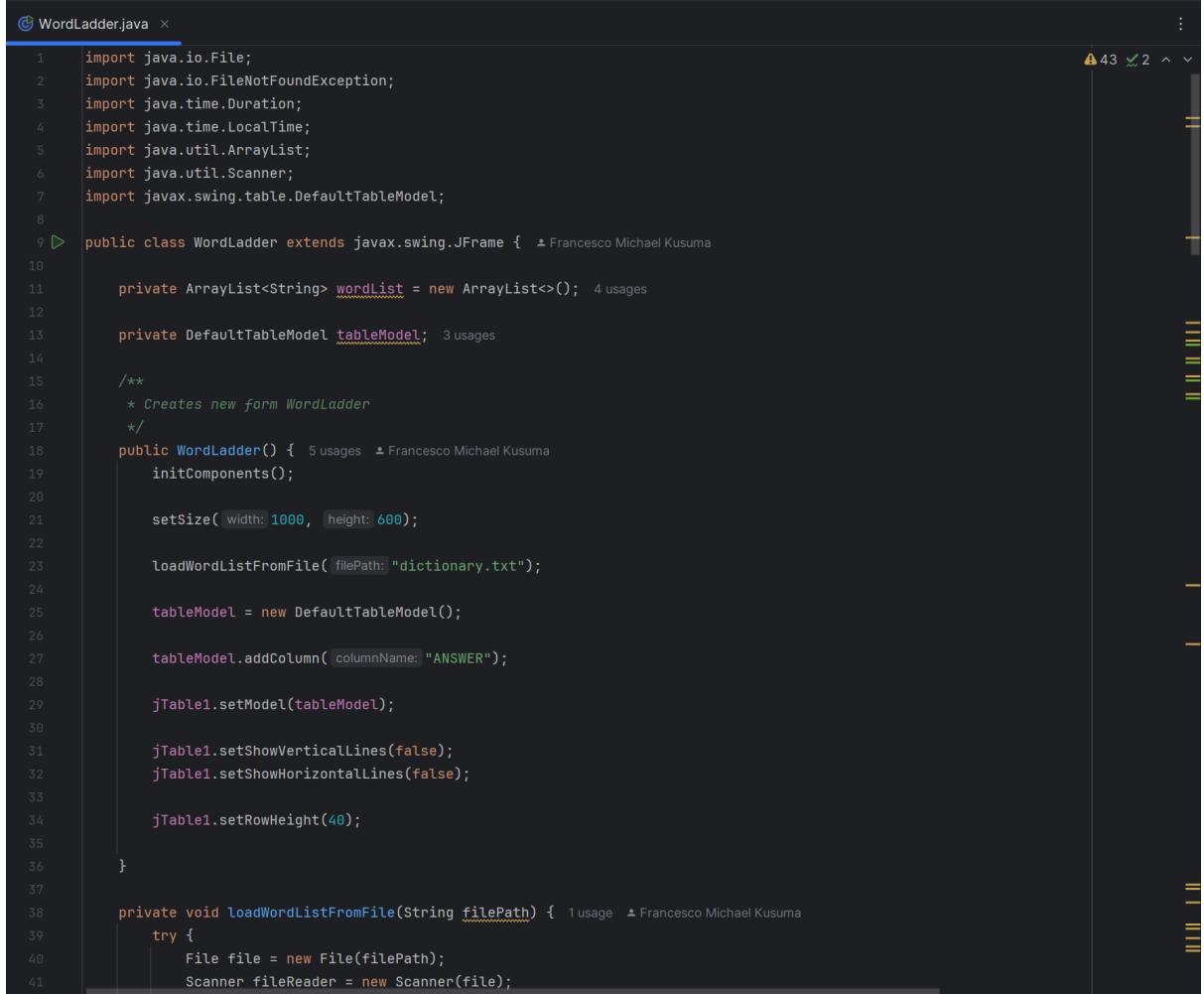
Secara teoritis, algoritma Greedy Best First Search tidak menjamin solusi yang optimal untuk persoalan Word Ladder. Algoritma GBFS mungkin menemukan solusi yang cepat dalam banyak kasus, tetapi tidak menjamin solusi yang optimal. Ini karena GBFS mungkin terjebak mengeksplorasi jalur yang mungkin tampak dekat dengan solusi (berdasarkan nilai heuristik), tetapi sebenarnya bukanlah jalur terpendek dalam hal jumlah langkah yang sebenarnya diperlukan untuk mencapai solusi. Ini mengakibatkan algoritma BGFS dapat terjebak pada optimum lokal atau bahkan mengalami siklus sehingga tidak menemukan solusi yang diharapkan.

BAGIAN II

SOURCE CODE

Pada proyek ini, terdapat 1 kelas yang dibuat, terdiri dari 1 main class(WordLadder), 1 abstract class(Graph), dan 3 inherited class(UCS, GBFS, dan Astar). Penjelasan kelas lebih lanjut :

1. WordLadder



The screenshot shows a Java code editor with the file "WordLadder.java" open. The code is a Java Swing application for a word ladder game. It imports various Java classes and defines a main class WordLadder that extends JFrame. The class contains methods for initializing components, loading a word list from a file, and setting up a jTable. The code uses annotations like JFrancesco Michael Kusuma and includes comments explaining the purpose of certain methods.

```
WordLadder.java x
1 import java.io.File;
2 import java.io.FileNotFoundException;
3 import java.time.Duration;
4 import java.time.LocalTime;
5 import java.util.ArrayList;
6 import java.util.Scanner;
7 import javax.swing.table.DefaultTableModel;
8
9 public class WordLadder extends javax.swing.JFrame { ▲ Francesco Michael Kusuma
10
11     private ArrayList<String> wordList = new ArrayList<>(); 4 usages
12
13     private DefaultTableModel tableModel; 3 usages
14
15     /**
16      * Creates new form WordLadder
17     */
18     public WordLadder() { 5 usages ▲ Francesco Michael Kusuma
19         initComponents();
20
21         setSize( width: 1000, height: 600);
22
23         loadWordListFromFile( filePath: "dictionary.txt");
24
25         tableModel = new DefaultTableModel();
26
27         tableModel.addColumn( columnName: "ANSWER");
28
29         jTable1.setModel(tableModel);
30
31         jTable1.setShowVerticalLines(false);
32         jTable1.setShowHorizontalLines(false);
33
34         jTable1.setRowHeight(40);
35
36     }
37
38     private void loadWordListFromFile(String filePath) { 1 usage ▲ Francesco Michael Kusuma
39         try {
40             File file = new File(filePath);
41             Scanner fileReader = new Scanner(file);
```

```
42     }
43     while(fileReader.hasNextLine()) {
44         String data = fileReader.nextLine();
45         wordList.add(data);
46     }
47     fileReader.close();
48 } catch (FileNotFoundException e) {
49     System.out.println("File not found");
50 }
51 }
52 /**
53 * This method is called from within the constructor to initialize the form.
54 * WARNING: Do NOT modify this code. The content of this method is always
55 * regenerated by the Form Editor.
56 */
57 [unchecked] 1 usage ▲ Francesco Michael Kusuma
58 // <editor-fold defaultstate="collapsed" desc="Generated Code">
59 > private void initComponents() { ... } // </editor-fold>
60
61 private void AlgorithmChoiceActionPerformed(java.awt.event.ActionEvent evt) {
62     // TODO add your handling code here:
63 }
64
65 private void EndInputActionPerformed(java.awt.event.ActionEvent evt) {
66     // TODO add your handling code here:
67 }
68
69 private void StartInputActionPerformed(java.awt.event.ActionEvent evt) {
70     // TODO add your handling code here:
71 }
72
73 private void SearchButtonActionPerformed(java.awt.event.ActionEvent evt) {
74     // TODO add your handling code here:
75
76     String source = StartInput.getText();
77     String destination = EndInput.getText();
78
79     if(source.isEmpty() || destination.isEmpty()) {
80
81         if(source.isEmpty() || destination.isEmpty()) {
82             ErrorInputMessage.setText("Please enter both start and end word.");
83             ErrorInputMessage.setVisible(true);
84             ErrorInputMessage.revalidate(); // Refresh the display
85             return;
86         }
87
88         if (!(wordList.contains(source) && wordList.contains(destination))) {
89             ErrorInputMessage.setText("Start or End is not in the word list.");
90             ErrorInputMessage.setVisible(true);
91             ErrorInputMessage.revalidate(); // Refresh the display
92             return;
93         }
94
95         if (source.length() != destination.length()) {
96             ErrorInputMessage.setText("The length must be same!!!");
97             ErrorInputMessage.setVisible(true);
98             ErrorInputMessage.revalidate(); // Refresh the display
99             return;
100        }
101
102        ArrayList<String> newWordList = new ArrayList<~>();
103
104        for(String data : wordList) {
105            if(data.length() == source.length()) {
106                newWordList.add(data);
107            }
108        }
109
110        ErrorInputMessage.setVisible(false);
111        executeAlgorithm(source, destination, newWordList);
112    }
113
114 /**
115 * @param args the command line arguments
116 */
117 public static void main(String args[]) { ▲ Francesco Michael Kusuma
118     WordLadder MyWordLadder = new WordLadder();
119     MyWordLadder.setVisible(true);
120 }
```

```

375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452

    private void executeAlgorithm(String source, String destination, ArrayList<String> wordList) { 1 usage ▲ Francesco Michael Kusuma
        ArrayList<String> ans = new ArrayList<String>();
        int nodeVisited = 0;
        LocalTime startTime = LocalTime.now();
        ProcessingLabel.setVisible(true);

        String algorithm = (String) AlgorithmChoice.getSelectedItem();

        switch (algorithm) {
            case "UCS":
                // UCS
                UCS ucs = new UCS(wordList);
                ans = ucs.shortestPath(source, destination);
                nodeVisited = ucs.getNodeVisited();
                break;
            case "Greedy Best First Search":
                // GBFS
                GBFS gbfs = new GBFS(wordList);
                ans = gbfs.shortestPath(source, destination);
                nodeVisited = gbfs.getNodeVisited();
                break;
            case "A*":
                // A Star
                AStar aStar = new AStar(wordList);
                ans = aStar.shortestPath(source, destination);
                nodeVisited = aStar.getNodeVisited();
                break;
            default:
                break;
        }

        LocalTime endTime = LocalTime.now();
        ProcessingLabel.setVisible(false);

        Duration duration = Duration.between(startTime, endTime);
        long milliseconds = duration.toMillis();

        DurationLabel.setText("The duration is " + milliseconds + " milliseconds");
        NodeLabel.setText("Visited " + nodeVisited + " Nodes");

        DurationLabel.setVisible(true);
        NodeLabel.setVisible(true);
        if (ans == null) {
            LengthLabel.setVisible(false);
            DefaultTableModel model = (DefaultTableModel) jTable1.getModel();
            model.setRowCount(0);
            model.addRow(new Object[]{"No path"});
            TableAnswer.setVisible(true);
        } else {
            LengthLabel.setText("The length is " + ans.size());
            TableAnswer.setVisible(true);
            LengthLabel.setVisible(true);
            DefaultTableModel model = (DefaultTableModel) jTable1.getModel();
            model.setRowCount(0); // Clear the table before adding new data
            for (String data : ans) {
                model.addRow(new Object[]{data});
            }
        }
    }

    // Variables declaration - do not modify
    private javax.swing.JComboBox<String> AlgorithmChoice; 7 usages
    private javax.swing.JLabel AlgorithmLabel; 7 usages
    private javax.swing.JPanel Answer; 6 usages
    private javax.swing.JLabel DurationLabel; 11 usages
    private javax.swing.JTextField EndInput; 6 usages
    private javax.swing.JLabel EndLabel; 6 usages
    private javax.swing.JLabel ErrorInputMessage; 18 usages
    private javax.swing.JLabel LengthLabel; 13 usages
    private javax.swing.JLabel NodeLabel; 12 usages
    private javax.swing.JLabel ProcessingLabel; 10 usages
    private javax.swing.JPanel Question; 6 usages
    private javax.swing.JButton SearchButton; 6 usages
    private javax.swing.JTextField StartInput; 6 usages
    private javax.swing.JLabel StartLabel; 6 usages
    private javax.swing.JScrollPane TableAnswer; 7 usages
    private javax.swing.JLabel Title; 8 usages
    private javax.swing.JPanel JPanel1; 6 usages

```

```
453     private javax.swing.JTable jTable1; 13 usages
454     // End of variables declaration
455 }
```

Pada kelas WordLadder, dilakukan beberapa hal untuk menginisiasi program, seperti membaca file yang berisi list kata yang tersedia, membuat tampilan untuk GUI(Graphical User Interface), dan sebagainya. Atribut yang ada pada kelas ini adalah wordList dan tableModel. wordList berguna untuk menyimpan kata-kata yang telah dibaca dari file. tableModel berguna untuk menyediakan tempat menampilkan solusi yang ada. Ada lima method pada kelas ini yang memiliki kegunaan masing-masing, yaitu

a. **loadWordListFromFile**

Method ini berfungsi untuk membaca file “dictionary.txt” untuk digunakan sebagai daftar kata yang akan digunakan pada permainan ini.

b. **initComponents**

Method ini berfungsi untuk membuat tampilan GUI yang telah diciptakan oleh Java Swing.

c. **SearchButtonActionPerformed**

Method ini berfungsi untuk melakukan aksi yang akan dilakukan saat tombol Search ditekan.

d. **main**

Method ini berfungsi untuk memulai program berjalan.

e. **executeAlgorithm**

Method ini berguna untuk menjalankan algoritma dan memberikan solusi yang telah ditemukan.

2. Graph

The screenshot shows a Java code editor with the file 'Graph.java' open. The code defines an abstract class 'Graph' with various methods for string comparison and manipulation. The code is color-coded for syntax highlighting.

```
1 import java.util.ArrayList;
2
3 @abstract class Graph { 3 usages 3 inheritors ▲ Francesco Michael Kusuma
4     protected int nodeVisited; 3 usages
5     protected ArrayList<String> word;
6
7     @private boolean connected(String s1, String s2) { 1 usage ▲ Francesco Michael Kusuma
8         if (s1.length() != s2.length()) {
9             return false;
10        }
11        int dif = 0;
12        for (int i = 0; i < s1.length(); i++) {
13            if (s1.charAt(i) != s2.charAt(i)) {
14                dif++;
15                if (dif > 1) {
16                    return false;
17                }
18            }
19        }
20        return dif == 1;
21    }
22
23    public Graph(ArrayList<String> wordList) { 3 usages ▲ Francesco Michael Kusuma
24        this.word = new ArrayList<>();
25        this.nodeVisited = 0;
26        this.word.addAll(wordList);
27    }
28
29    protected ArrayList<String> getNeighbors(String current) { 3 usages ▲ Francesco Michael Kusuma
30        ArrayList<String> neighbors = new ArrayList<>();
31        for (String kata : word) {
32            if (kata.length() == current.length() && !kata.equals(current) && connected(kata, current)) {
33                neighbors.add(kata);
34            }
35        }
36        return neighbors;
37    }
38
39    > public int getNodeVisited() { return this.nodeVisited; }
40
41    > public void setNodeVisited(int nodeVisited) { this.nodeVisited = nodeVisited; }
42
43
44
45    @protected int heuristic(String current, String end) { 6 usages ▲ Francesco Michael Kusuma
46        int count = 0;
47        for (int i = 0; i < current.length(); i++) {
48            if (current.charAt(i) != end.charAt(i)) {
49                count++;
50            }
51        }
52        return count;
53    }
54
55    > abstract ArrayList<String> shortestPath(String start, String end); 3 usages 3 implementations ▲ Francesco Michael Kusuma
56
57
58 }
```

Kelas Graph adalah kelas abstrak yang melakukan hal terkait dengan pemrosesan algoritma untuk digunakan pada kelas-kelas turunannya. Pada kelas ini, terdapat dua atribut yang bersifat protected, yaitu nodeVisited dan word. nodeVisited memiliki kegunaan untuk menyimpan hasil dari jumlah kata yang telah ditelusuri. word memiliki kegunaan untuk menjadi daftar kata yang telah disaring berdasarkan panjang kata persoalan untuk meningkatkan efisiensi program. Ada enam method yang dimiliki kelas ini dan memiliki kegunaan masing-masing, yaitu

a. connected

Method ini berfungsi untuk menentukan apakah suatu kata dapat menjadi kata selanjutnya.

b. getNeighbors

Method ini berfungsi untuk mencari daftar kata yang bisa menjadi kemungkinan untuk menjadi kata selanjutnya.

c. **getNodeVisited**

Method ini adalah getter yang berfungsi untuk mendapatkan nilai pada atribut nodeVisited.

d. **setNodeVisited**

Method ini adalah setter yang berfungsi untuk mengatur nilai pada atribut nodeVisited.

e. **heuristic**

Method ini berfungsi untuk mencari nilai heuristic/h(n) dari suatu kata.

f. **shortestPath**

Method ini adalah method abstrak yang akan berguna untuk mencari rute terpendek dari kata awal ke kata tujuan.

3. UCS

```
UCS.java x
1 import java.util.ArrayList;
2 import java.util.Comparator;
3 import java.util.PriorityQueue;
4
5 public class UCS extends Graph { 3 usages ▲ Francesco Michael Kusuma
6     public UCS(ArrayList<String> wordList) { super(wordList); }
7
8     @Override 3 usages ▲ Francesco Michael Kusuma
9     public ArrayList<String> shortestPath(String start, String end) {
10         PriorityQueue<Node> pq = new PriorityQueue<>(Comparator.comparingInt(node -> node.cost));
11         pq.add(new Node(start, cost: 0, parent: null));
12
13         ArrayList<String> visited = new ArrayList<>();
14
15         while (!pq.isEmpty()) {
16             Node currentNode = pq.poll();
17             String currentWord = currentNode.word;
18
19             if (currentWord.equals(end)) {
20                 visited.add(currentWord);
21                 setNodeVisited(visited.size());
22                 ArrayList<String> path = new ArrayList<>();
23                 while (currentNode != null) {
24                     path.addFirst(currentNode.word);
25                     currentNode = currentNode.parent;
26                 }
27                 return path;
28             }
29
30             if (!visited.contains(currentWord)) {
31                 visited.add(currentWord);
32                 ArrayList<String> neighbors = getNeighbors(currentWord);
33                 for (String neighbor : neighbors) {
34                     if (!visited.contains(neighbor)) {
35                         int cost = currentNode.cost + 1;
36                         pq.add(new Node(neighbor, cost, currentNode));
37                     }
38                 }
39             }
40         }
41     }
42     setNodeVisited(visited.size());
43
44     return null;
45 }
46
47 private static class Node { 6 usages ▲ Francesco Michael Kusuma
48     String word; 3 usages
49     int cost; 3 usages
50     Node parent; 2 usages
51
52     Node(String word, int cost, Node parent) { 2 usages ▲ Francesco Michael Kusuma
53         this.word = word;
54         this.cost = cost;
55         this.parent = parent;
56     }
57 }
58 }
```

Kelas UCS adalah kelas turunan yang berasal dari kelas Graph. Kelas ini memiliki kelas static Node untuk menciptakan node yang akan digunakan. Kelas node memiliki tiga atribut, yaitu

a. word

word akan menyimpan kata yang menjadi node

b. cost

cost akan menyimpan nilai $g(n)$ dari tiap node

c. parent

parent akan menyimpan node parent untuk melakukan pelacakan saat suatu path ditemukan

Kelas ini memiliki 1 method yang mengimplementasikan method shortestPath. Method ini menjalankan algoritma pencarian rute menggunakan algoritma UCS(Uniform Cost Search).

4. GBFS

```
 1 import java.util.ArrayList;
 2
 3 public class GBFS extends Graph { 2 usages ▾ Francesco Michael Kusuma *
 4 >     public GBFS(ArrayList<String> wordList) { super(wordList); }
 5
 6     @Override 3 usages ▾ Francesco Michael Kusuma *
 7     public ArrayList<String> shortestPath(String start, String end) {
 8         String currentWord = start;
 9         ArrayList<String> visited = new ArrayList<>();
10
11         while(!visited.contains(currentWord)) {
12             visited.add(currentWord);
13             if(currentWord.equals(end)) {
14                 // ada path
15                 setNodeVisited(visited.size());
16                 return visited;
17             }
18             ArrayList<String> neighbors = getNeighbors(currentWord);
19
20             if (neighbors.isEmpty()) {
21                 break;
22             }
23             currentWord = neighbors.getFirst();
24             int currentScore = heuristic(currentWord, end);
25             for(int i = 1; i < neighbors.size(); i++) {
26                 if(currentScore > heuristic(neighbors.get(i), end)) {
27                     currentScore = heuristic(neighbors.get(i), end);
28                     currentWord = neighbors.get(i);
29                 }
30             }
31         }
32         setNodeVisited(visited.size());
33         return null;
34     }
35 }
36 }
```

Kelas GBFS adalah kelas turunan yang berasal dari kelas Graph. Kelas ini memiliki 1 method yang mengimplementasikan method shortestPath. Method ini menjalankan algoritma pencarian rute menggunakan algoritma Greedy Best First Search.

5. Astar

```
 1 import java.util.ArrayList;
 2 import java.util.Comparator;
 3 import java.util.PriorityQueue;
 4
 5 public class AStar extends Graph { 2 usages ▲ Francesco Michael Kusuma
 6 >     public AStar(ArrayList<String> wordList) { super(wordList); }
 7
 8     @Override 3 usages ▲ Francesco Michael Kusuma
 9     public ArrayList<String> shortestPath(String start, String end) {
10         PriorityQueue<Node> pq = new PriorityQueue<>(Comparator.comparingInt(node -> node.totalEstimatedCost));
11         pq.add(new Node(start, cost: 0, heuristic(start, end), heuristic(start, end), parent: null));
12
13         ArrayList<String> visited = new ArrayList<>();
14
15         while (!pq.isEmpty()) {
16             Node currentNode = pq.poll();
17             String currentWord = currentNode.word;
18
19             if (currentWord.equals(end)) {
20                 visited.add(currentWord);
21                 setNodeVisited(visited.size());
22                 ArrayList<String> path = new ArrayList<>();
23                 while (currentNode != null) {
24                     path.addFirst(currentNode.word);
25                     currentNode = currentNode.parent;
26                 }
27                 return path;
28             }
29
30             if (!visited.contains(currentWord)) {
31                 visited.add(currentWord);
32                 ArrayList<String> neighbors = getNeighbors(currentWord);
33                 for (String neighbor : neighbors) {
34                     if (!visited.contains(neighbor)) {
35                         int cost = currentNode.cost + 1;
36                         int heuristicValue = heuristic(neighbor, end);
37                         int totalEstimatedCost = cost + heuristicValue;
38                         pq.add(new Node(neighbor, cost, heuristicValue, totalEstimatedCost, currentNode));
39                     }
40                 }
41             }
42         }
43     }
44
45     setNodeVisited(visited.size());
46     return null;
47 }
48
49 private static class Node { 6 usages ▲ Francesco Michael Kusuma
50     String word; 3 usages
51     int cost; 2 usages
52     int heuristic; 1 usage
53     int totalEstimatedCost; 2 usages
54     Node parent; 2 usages
55
56     Node(String word, int cost, int heuristic, int totalEstimatedCost, Node parent) { 2 usages ▲ Francesco Michael Kusuma
57         this.word = word;
58         this.cost = cost;
59         this.heuristic = heuristic;
60         this.totalEstimatedCost = totalEstimatedCost;
61         this.parent = parent;
62     }
63 }
64 }
```

Kelas AStar adalah kelas turunan yang berasal dari kelas Graph. Kelas ini memiliki kelas static Node untuk menciptakan node yang akan digunakan. Kelas node memiliki empat atribut, yaitu

a. word

word akan menyimpan kata yang menjadi node

b. cost

cost akan menyimpan nilai g(n) dari tiap node

c. **heuristic**

heuristic akan menyimpan nilai heuristic/h(n) dari tiap node

d. **totalEstimatedCost**

totalEstimatedCost akan menyimpan nilai $f(n) = g(n) + h(n)$ dari tiap node

e. **parent**

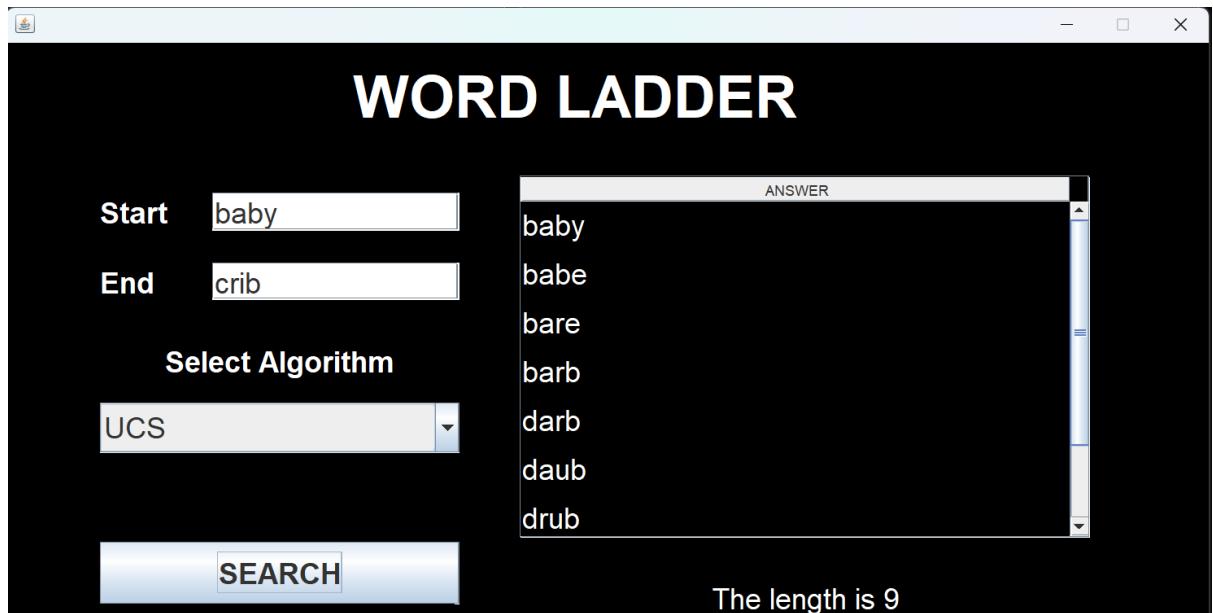
parent akan menyimpan node parent untuk melakukan pelacakan saat suatu path ditemukan

Kelas ini memiliki 1 method yang mengimplementasikan method shortestPath. Method ini menjalankan algoritma pencarian rute menggunakan algoritma A Star.

BAGIAN III
HASIL UJI COBA

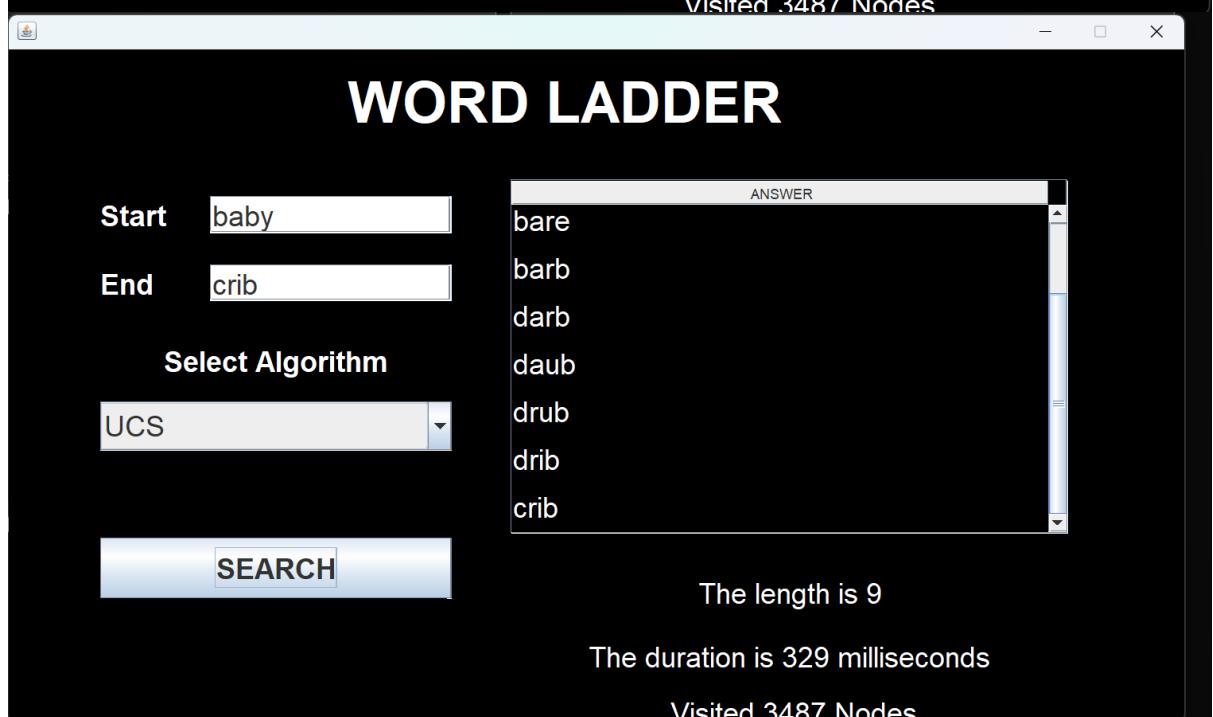
1. Baby – Crib

- a. UCS

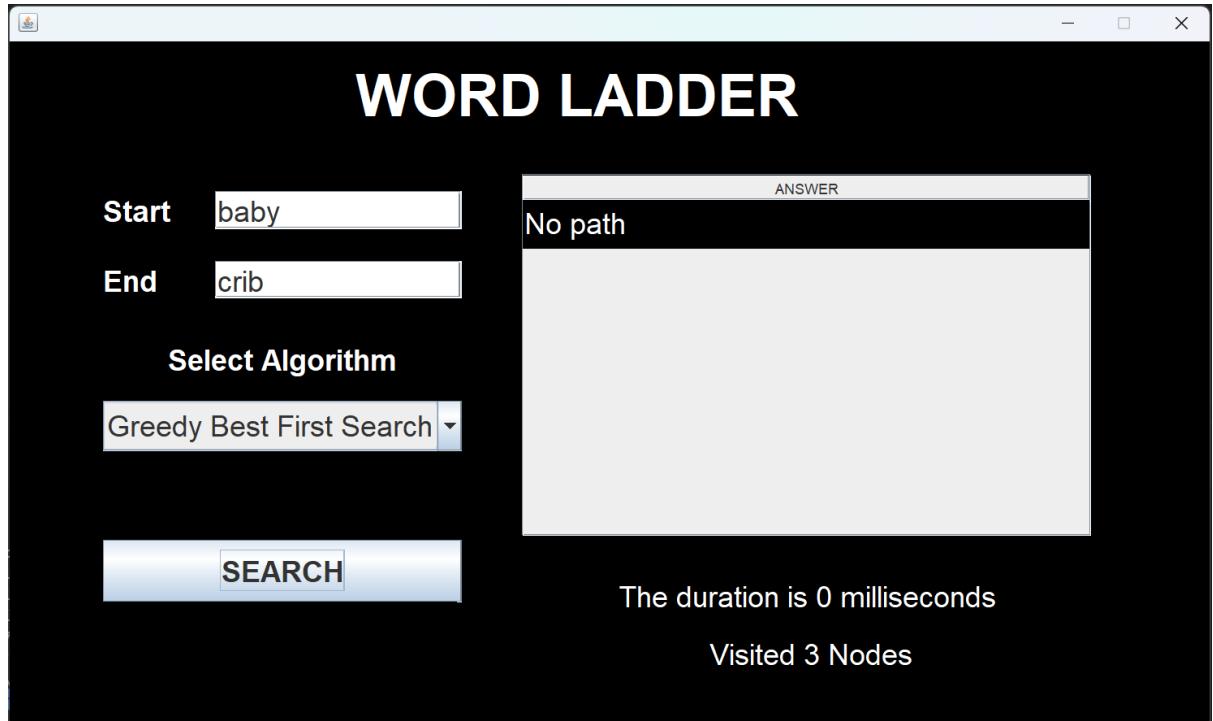


The duration is 329 milliseconds

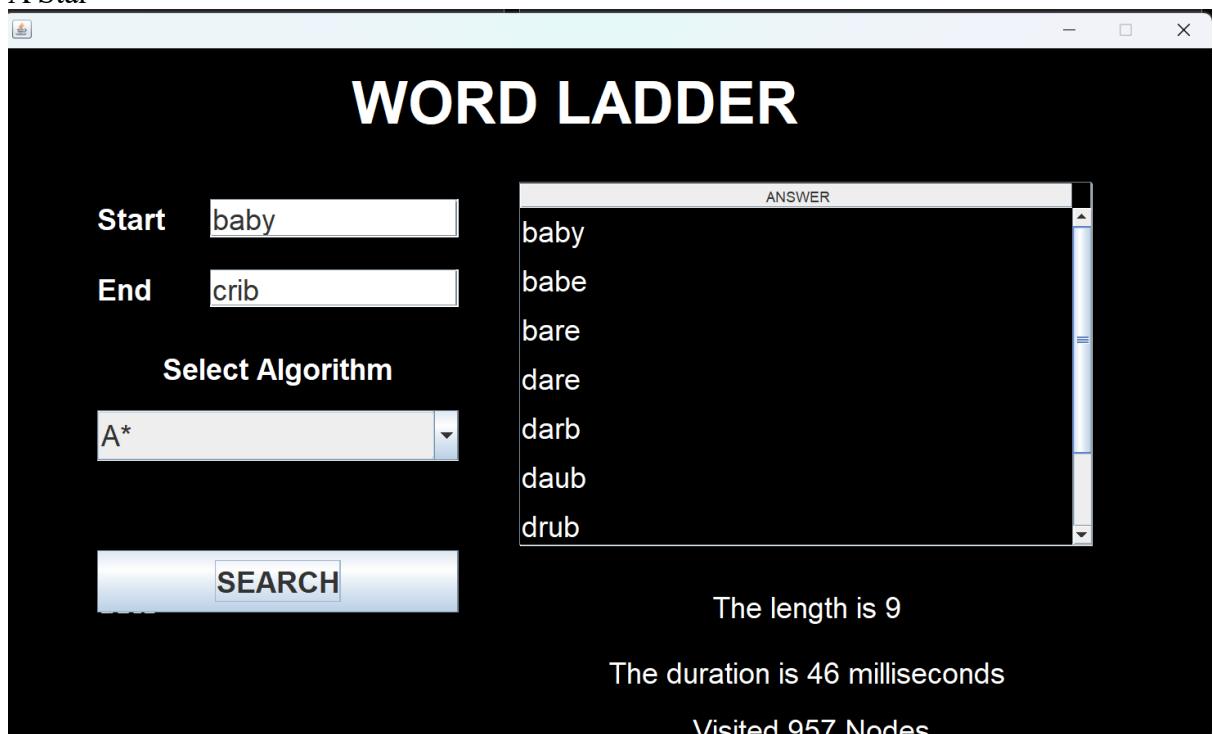
Visited 3487 Nodes

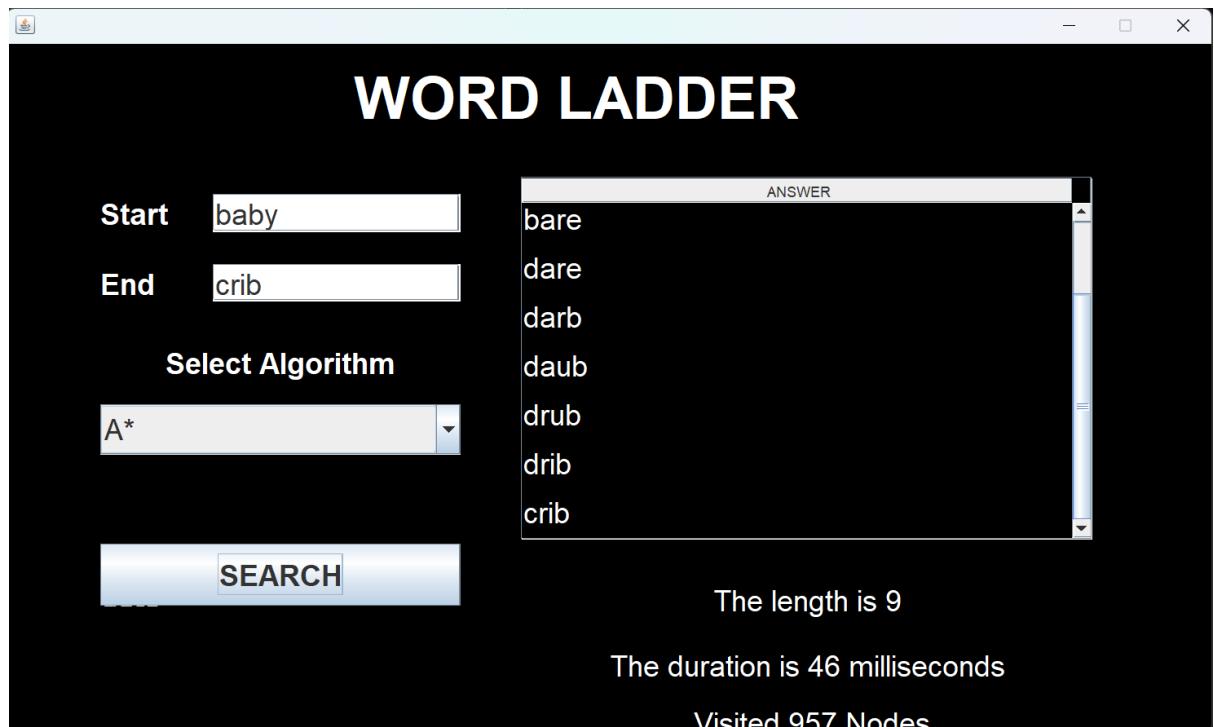


b. GBFS



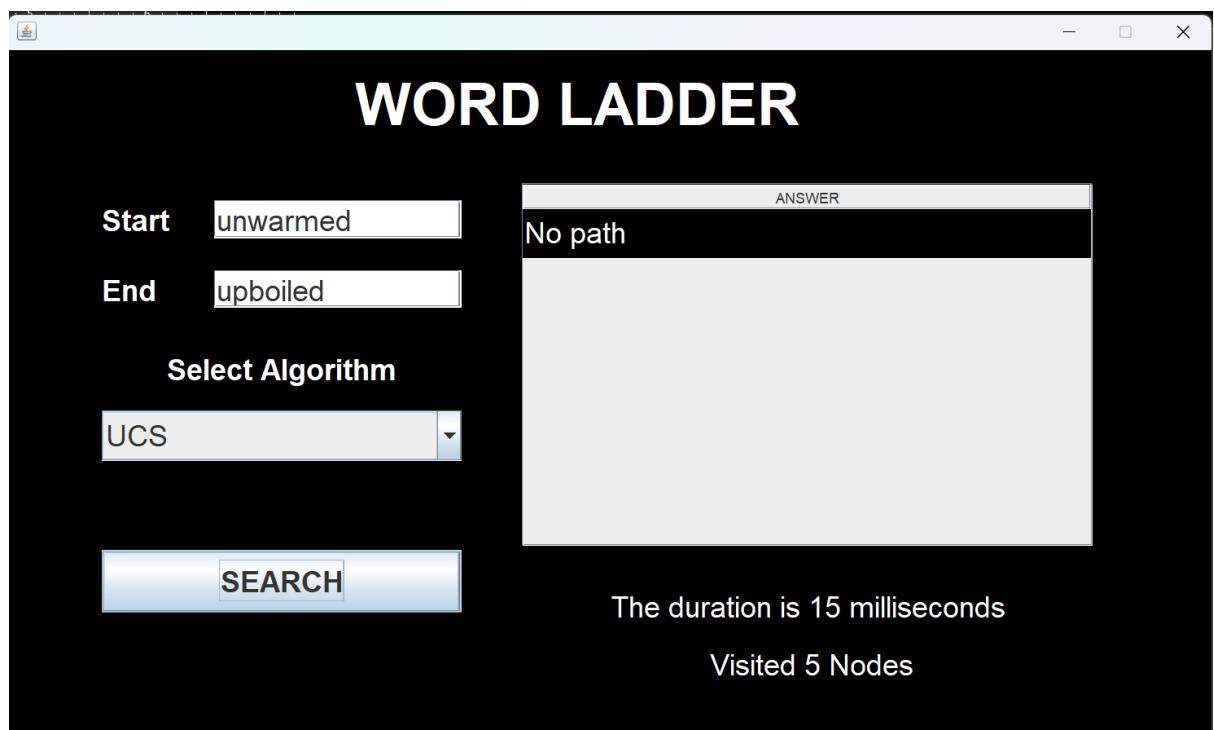
c. A Star



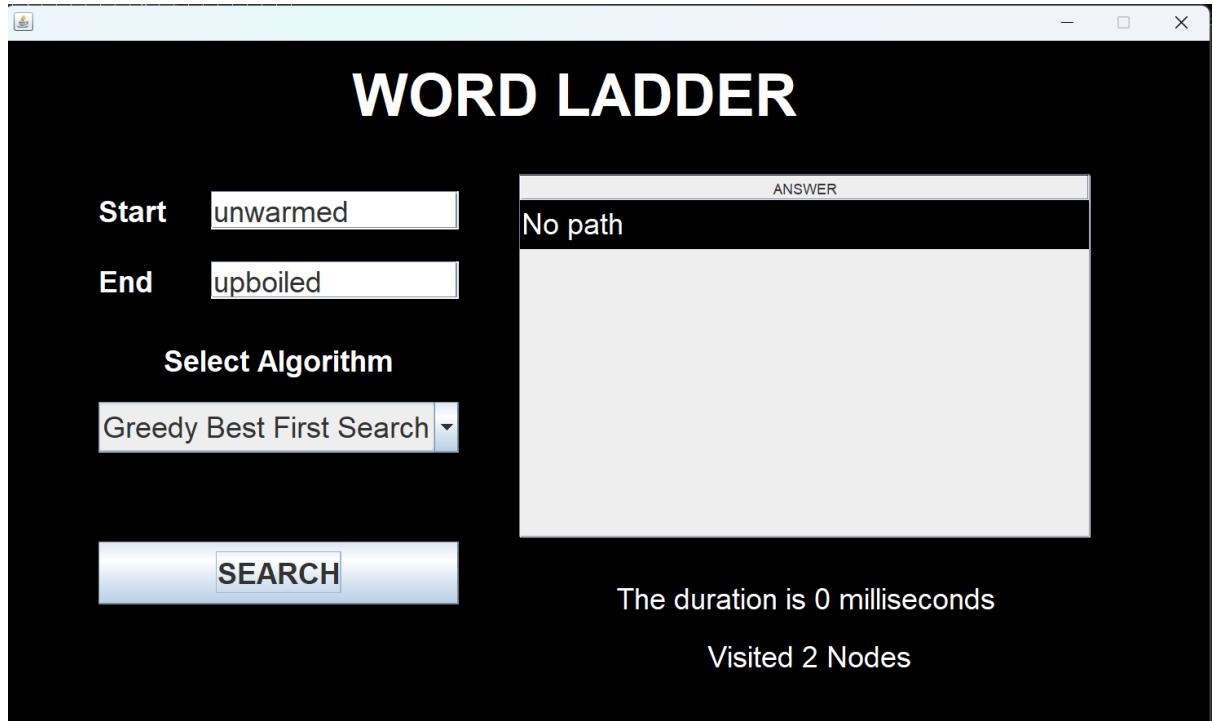


2. Unwarmed – Upboiled

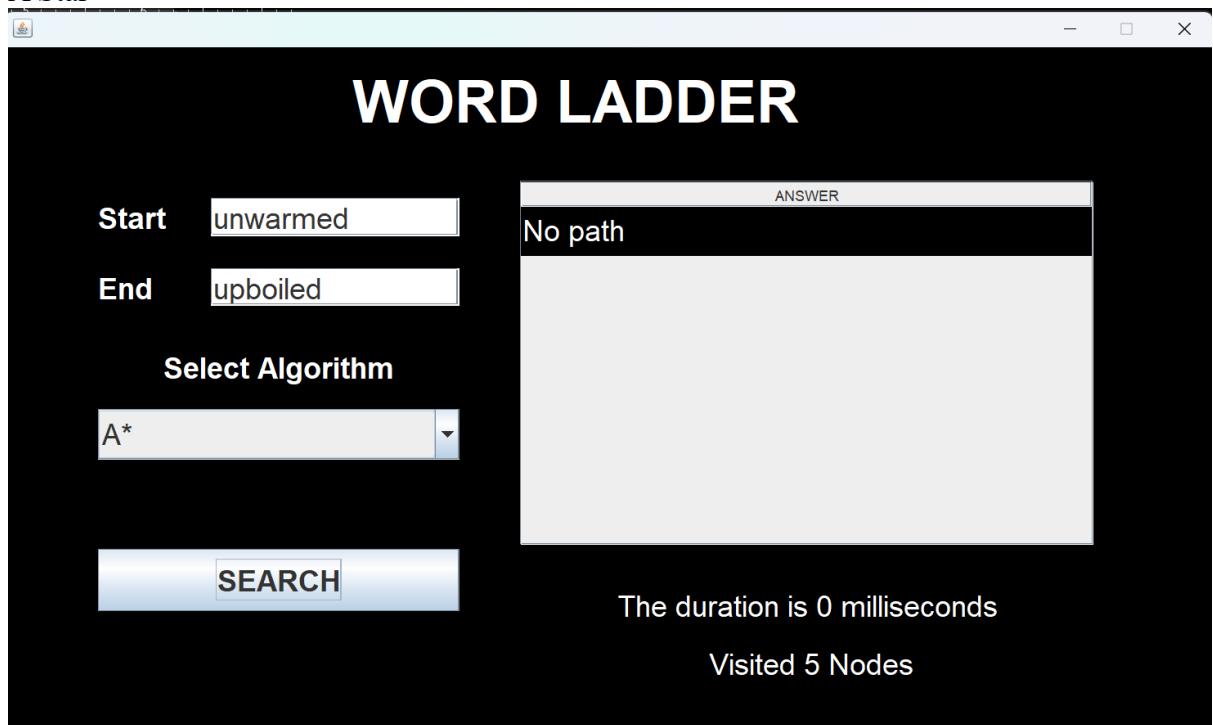
a. UCS



b. GBFS

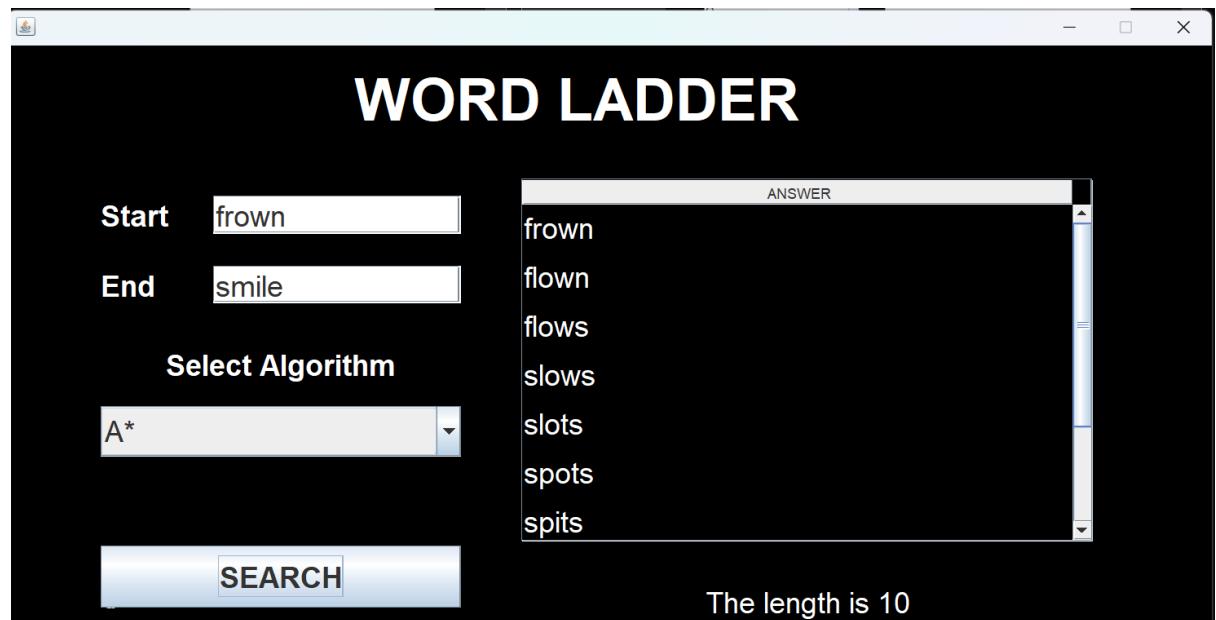


c. A Star



3. Frown – Smile

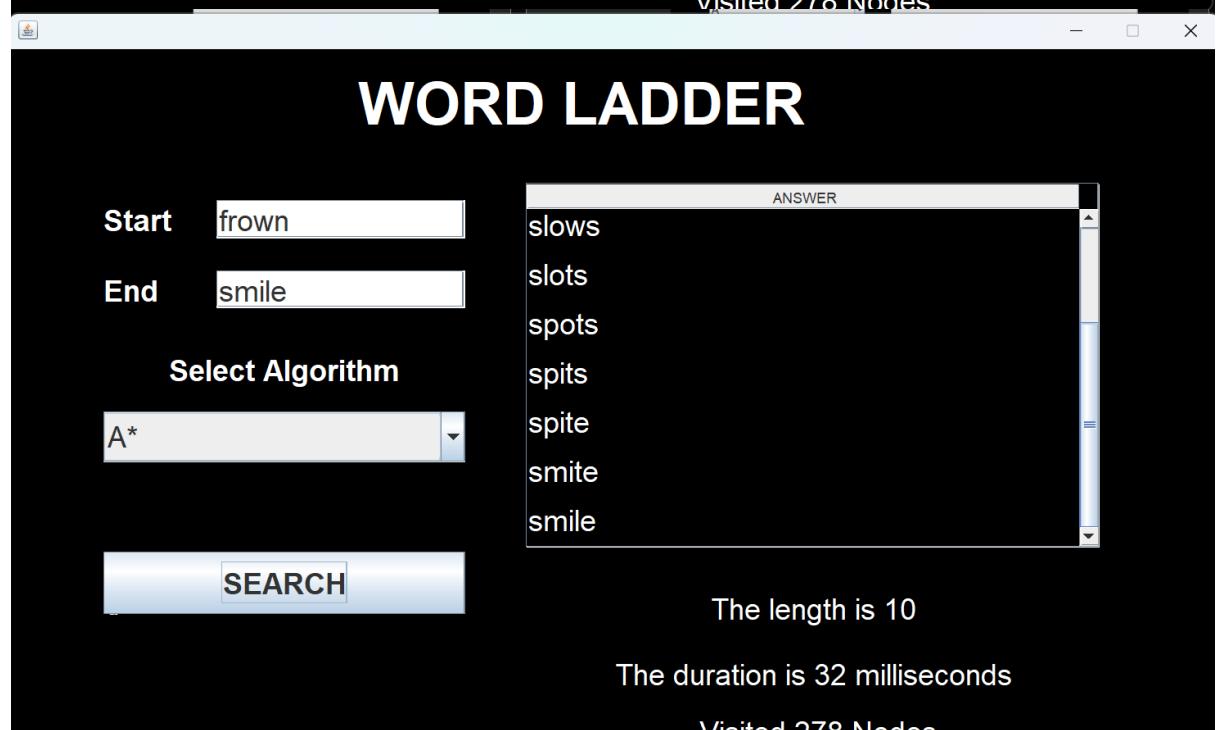
a. UCS



The length is 10

The duration is 32 milliseconds

Visited 278 Nodes

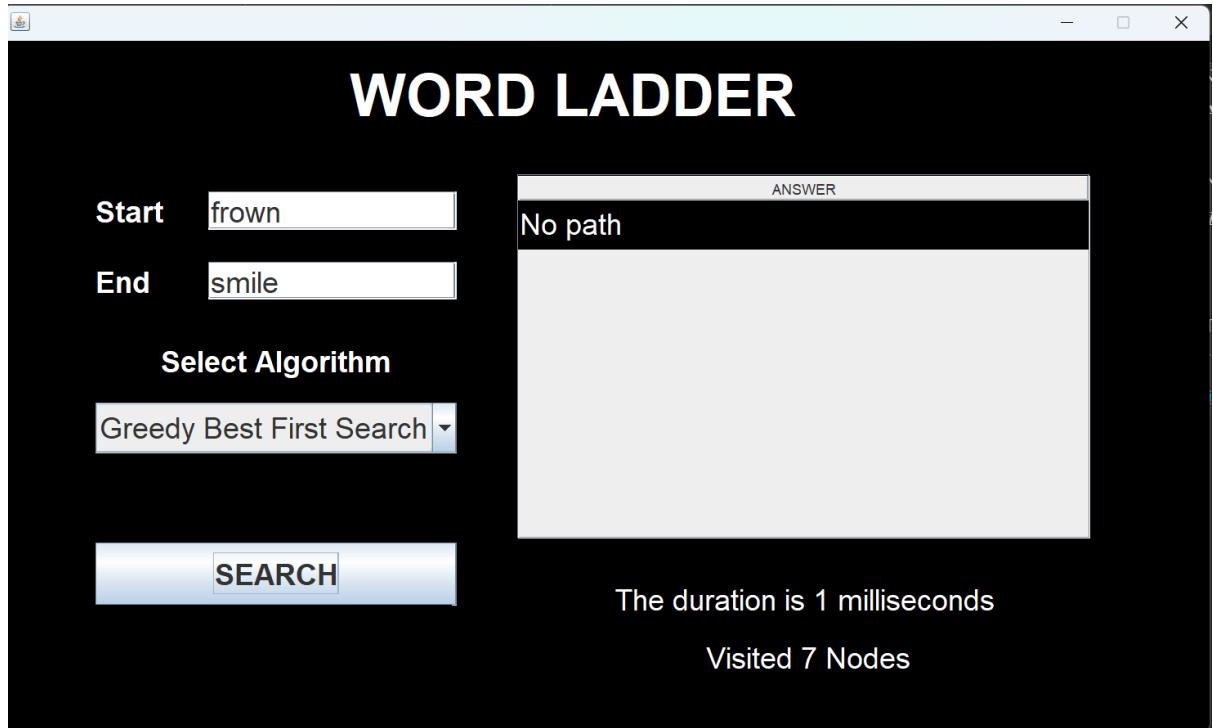


The length is 10

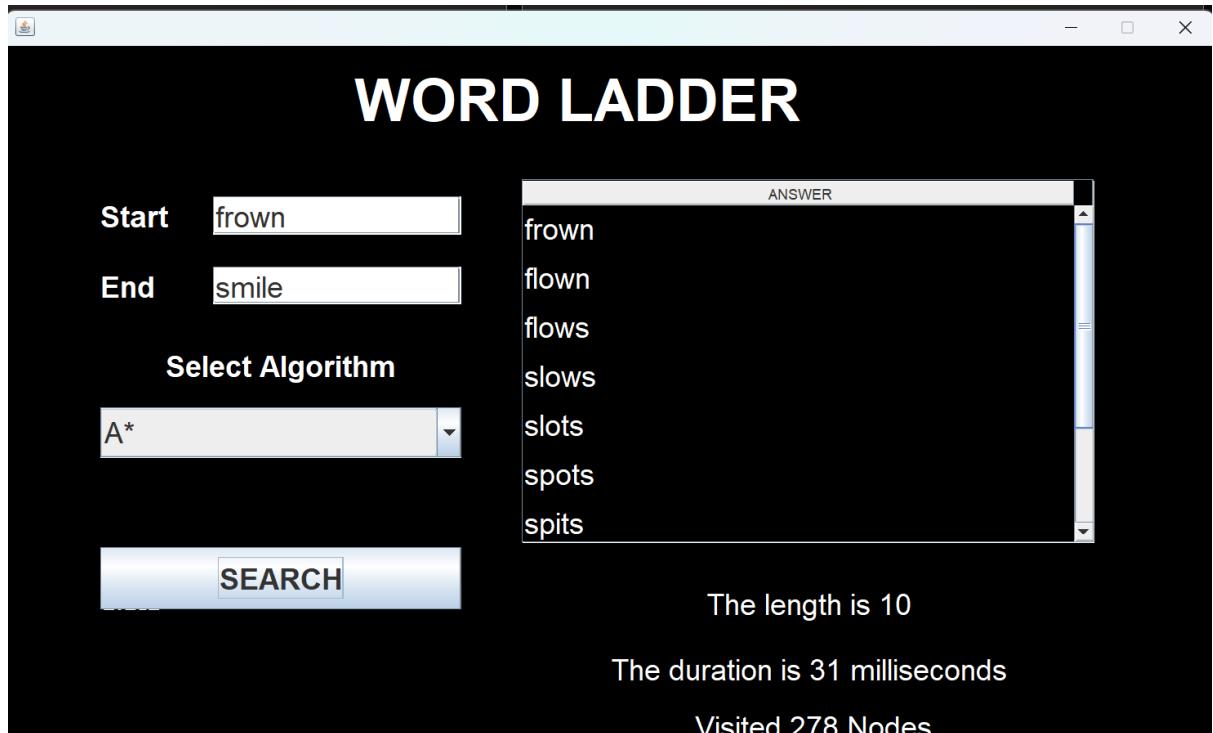
The duration is 32 milliseconds

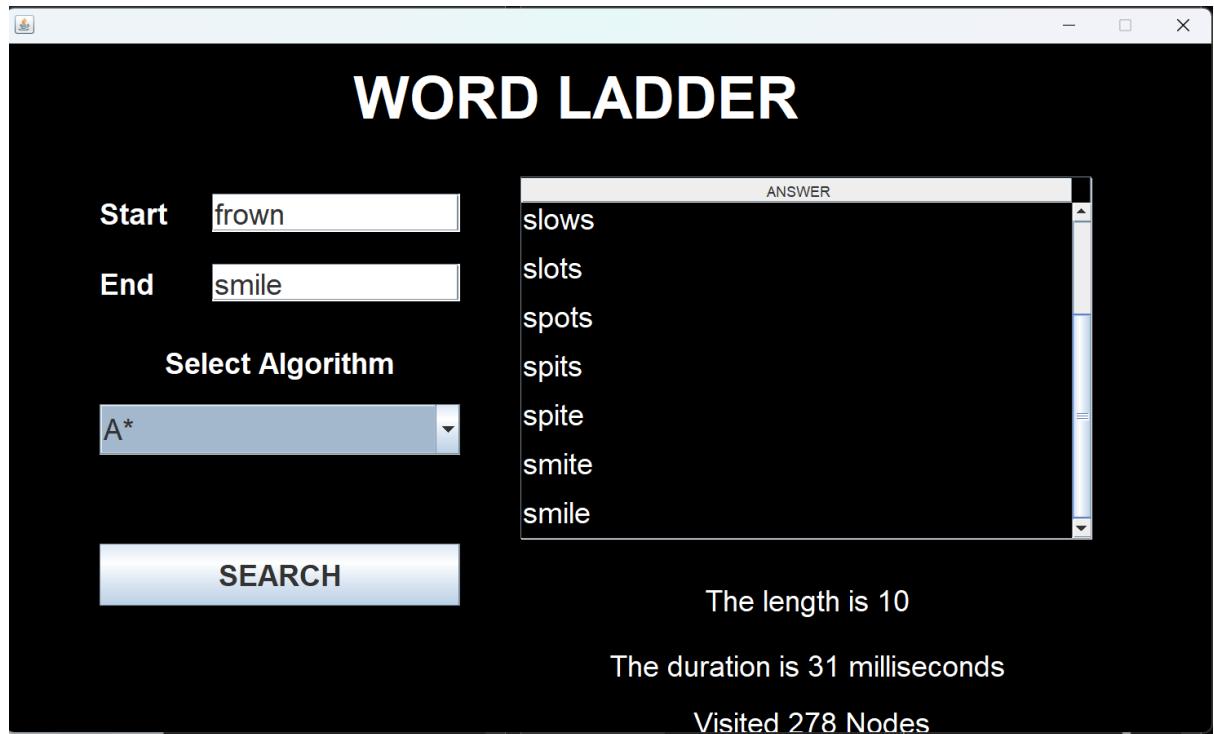
Visited 278 Nodes

b. GBFS

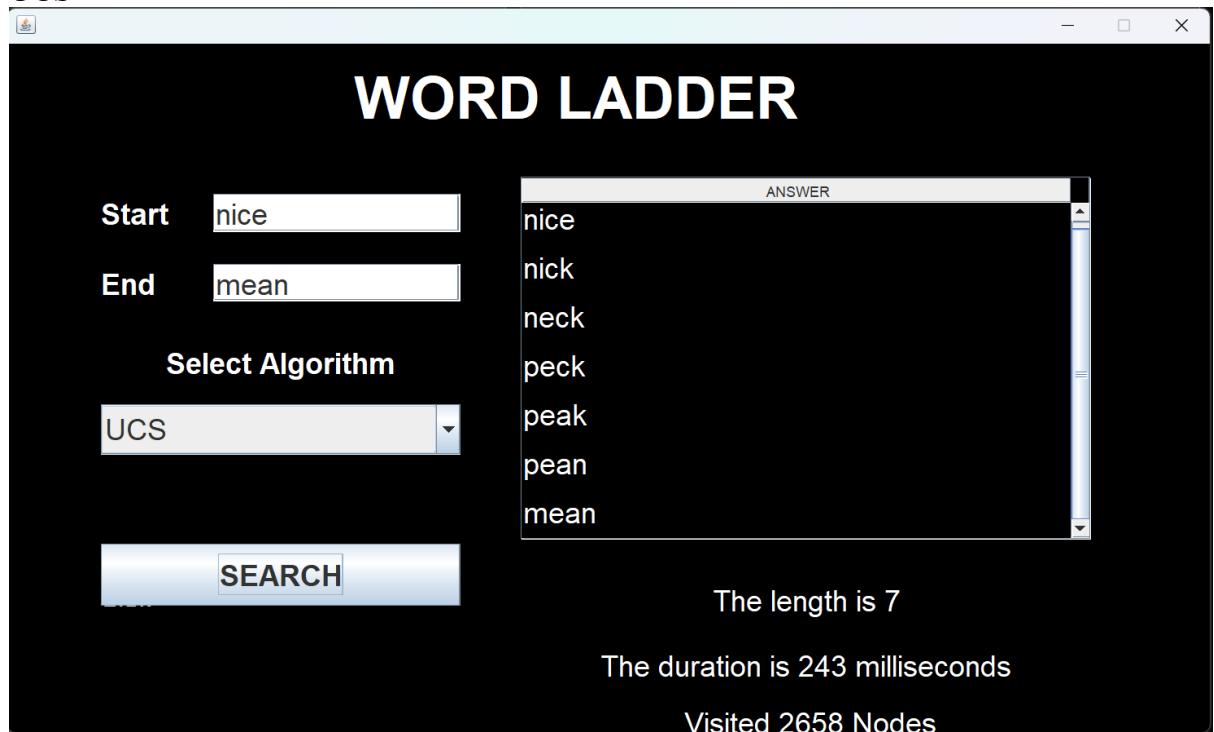


c. A Star

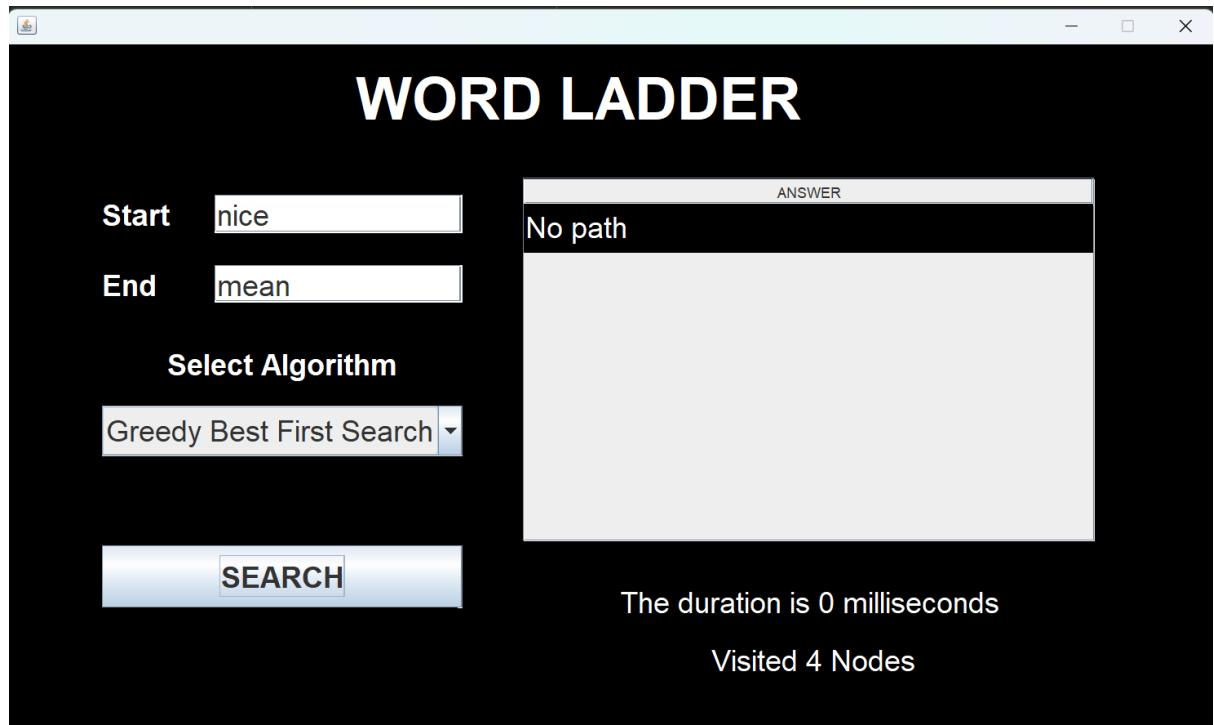




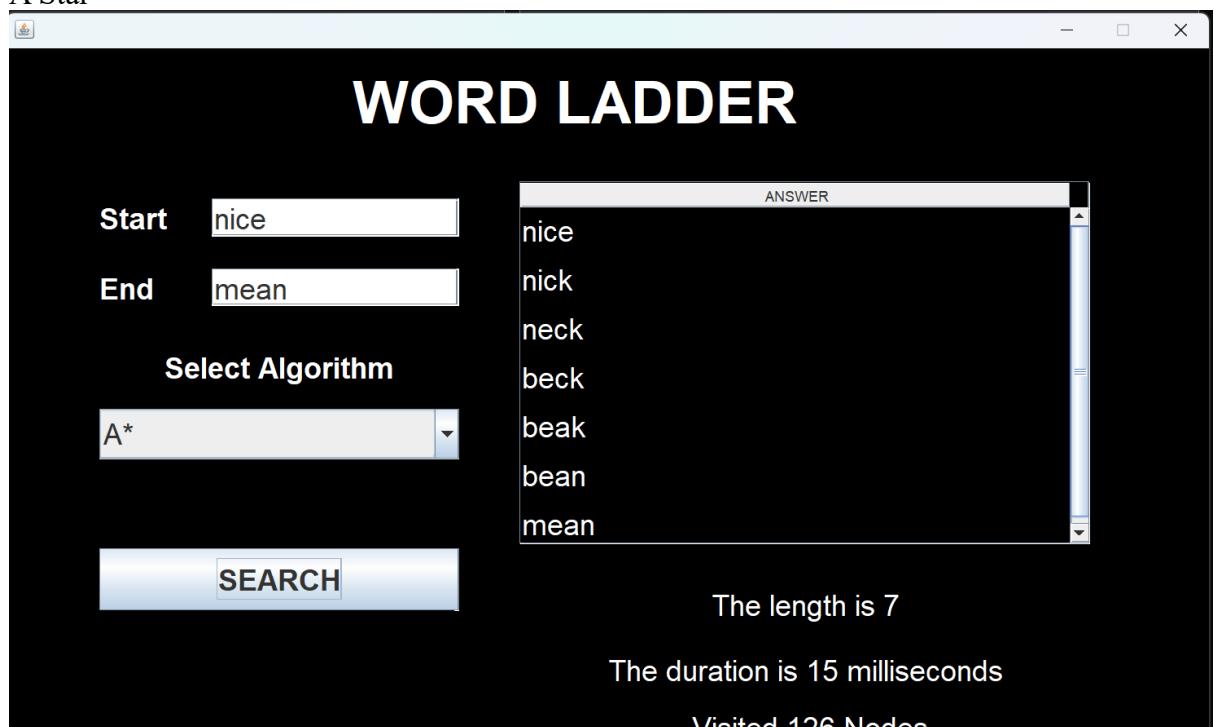
4. Nice – Mean
a. UCS



b. GBFS

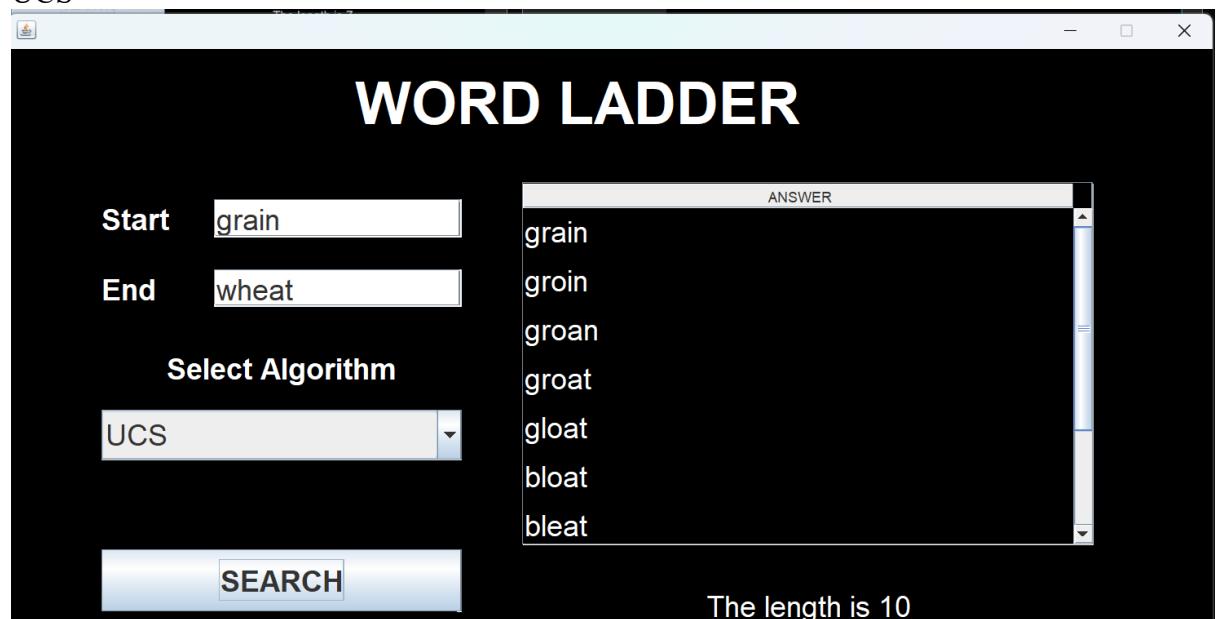


c. A Star



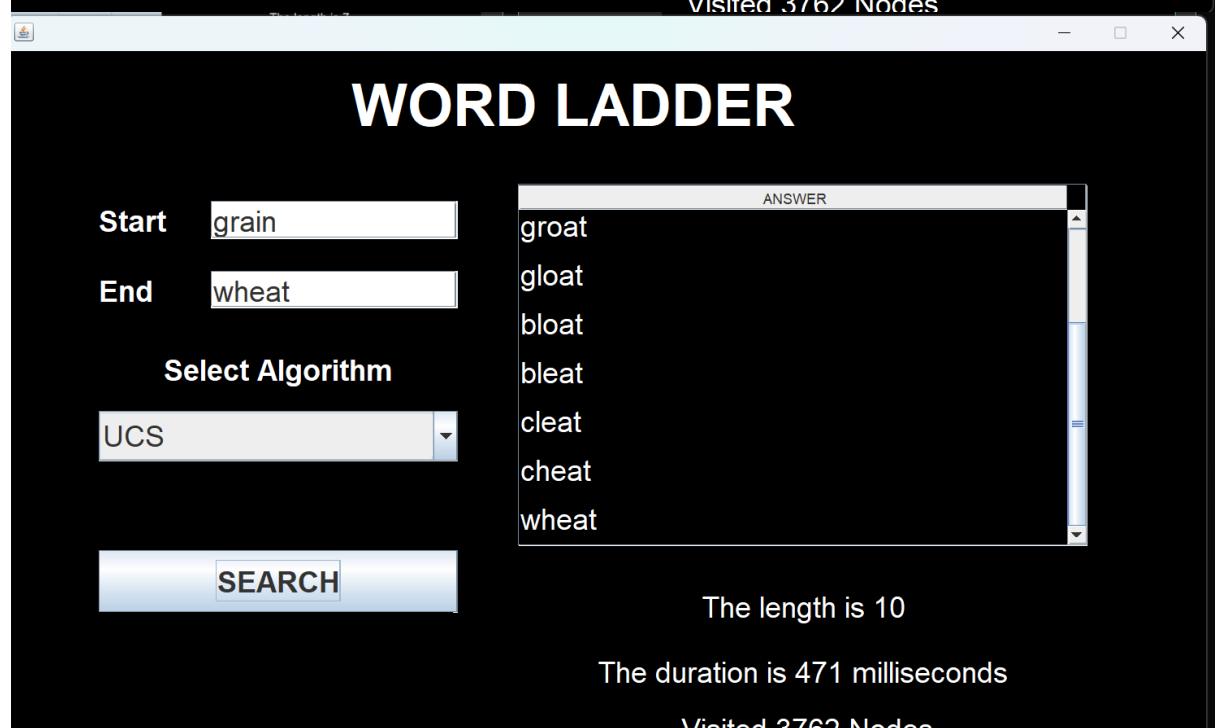
5. Grain – Wheat

a. UCS



The duration is 471 milliseconds

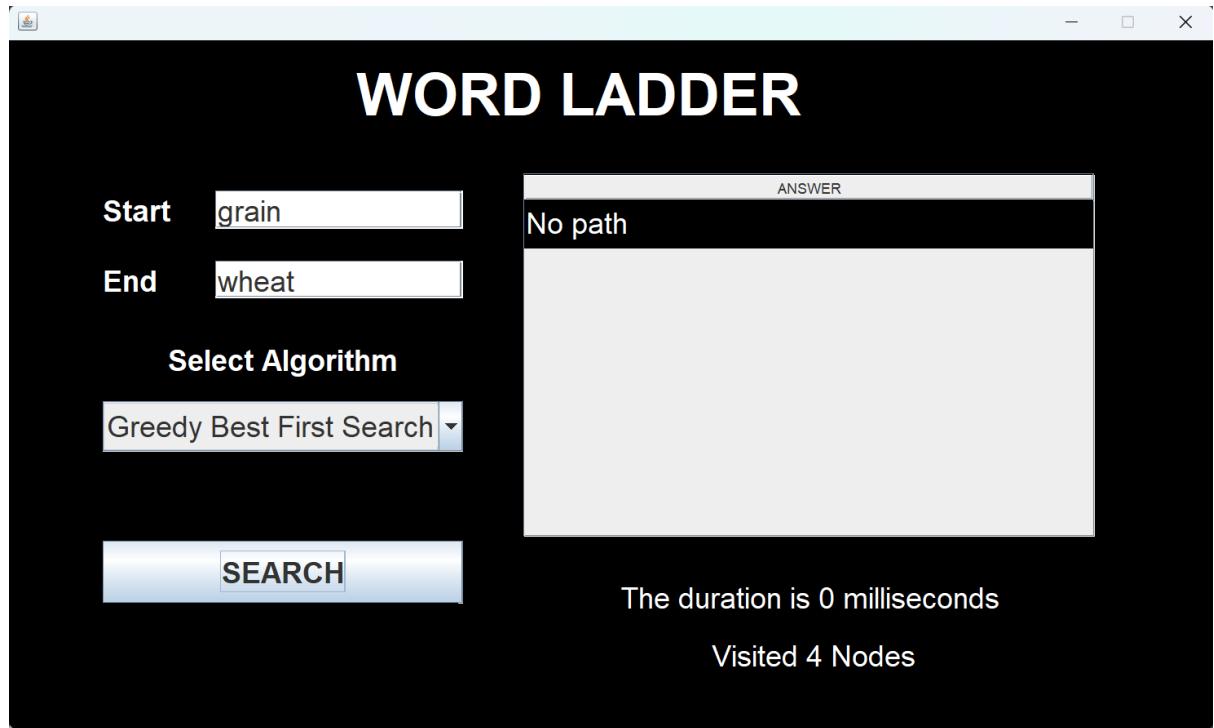
Visited 3762 Nodes



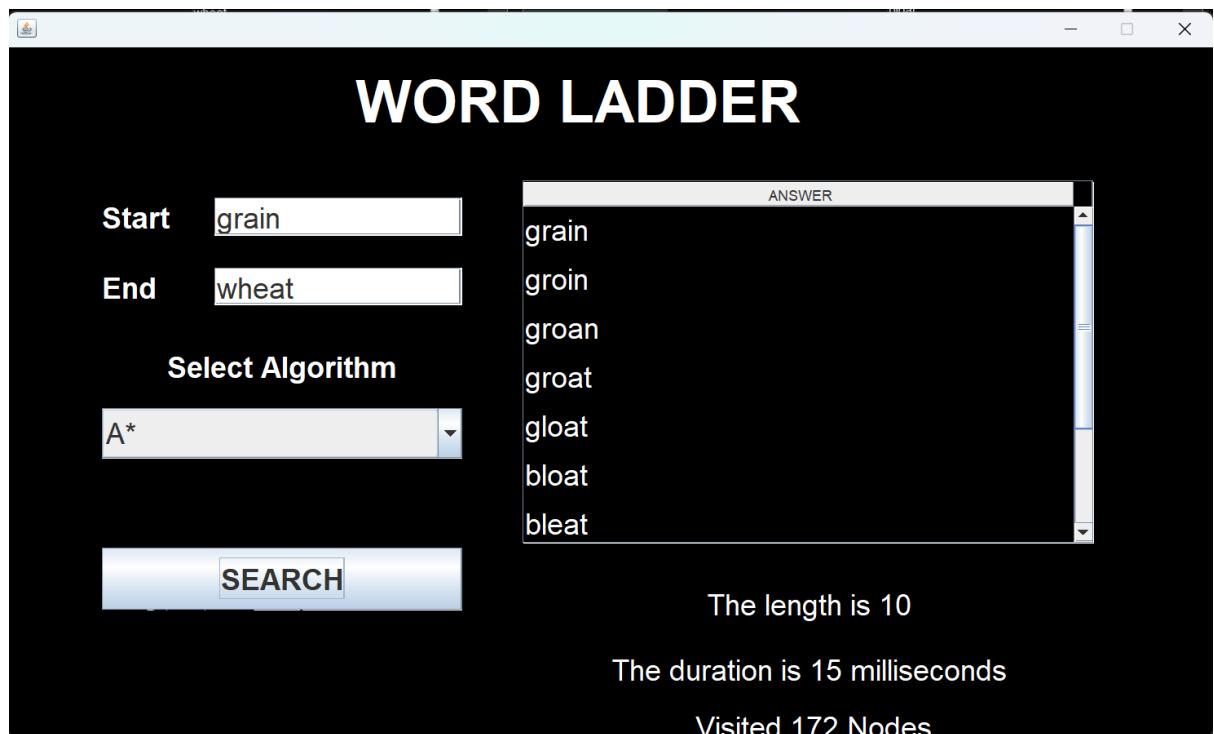
The duration is 471 milliseconds

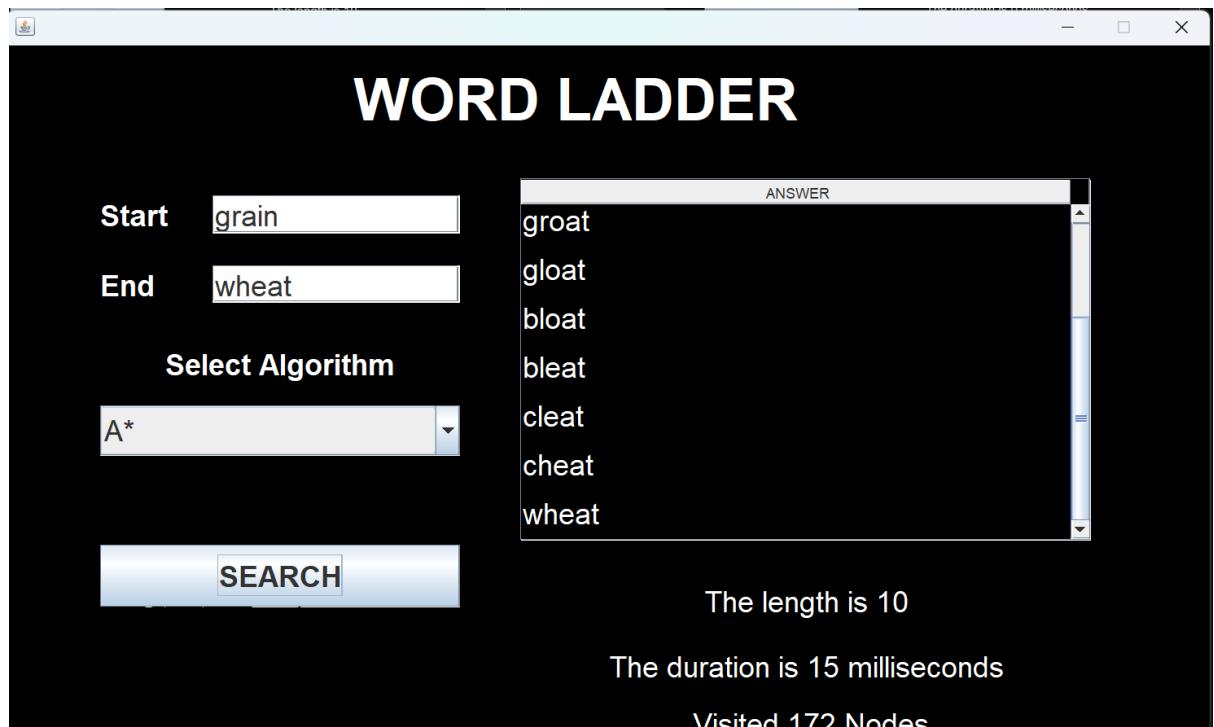
Visited 3762 Nodes

b. GBFS



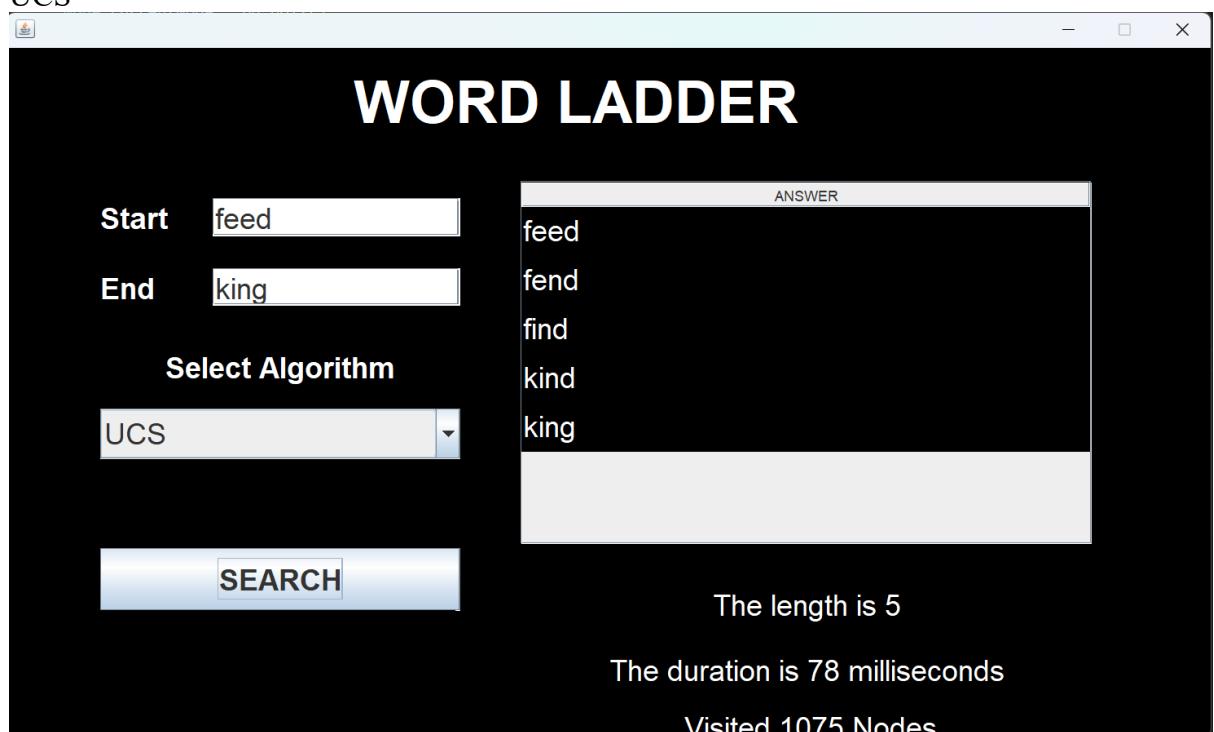
c. A Star



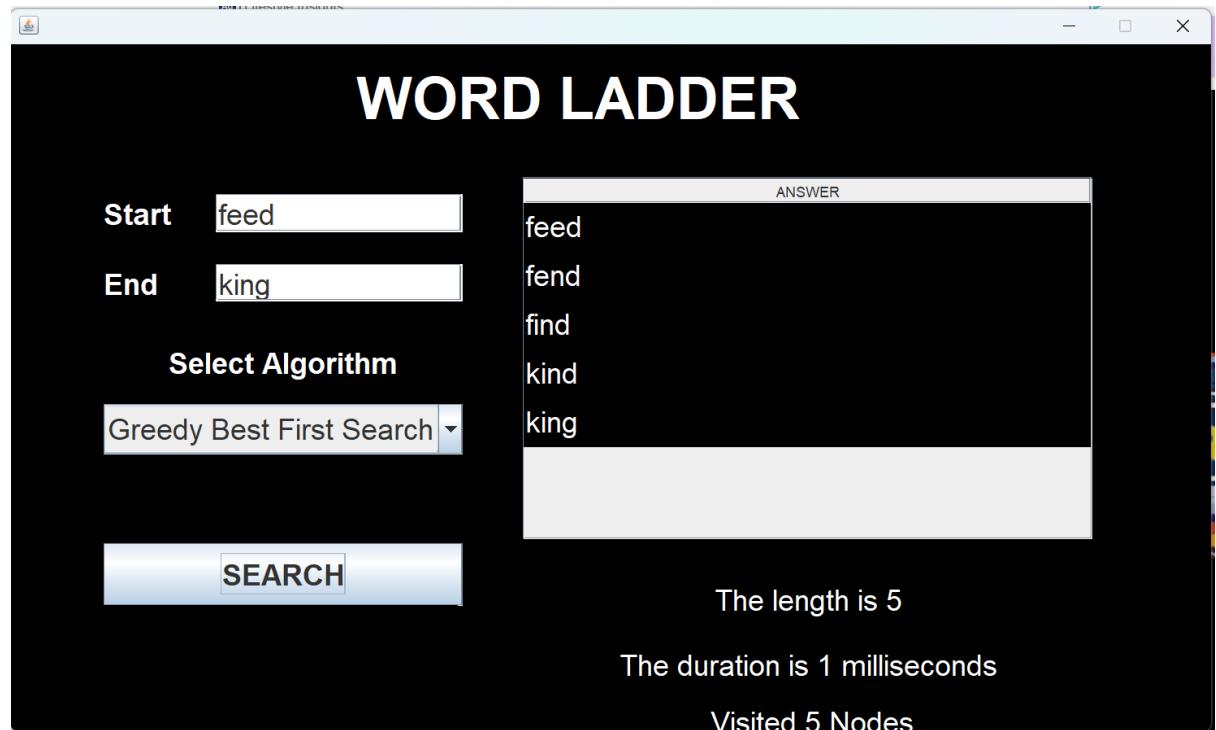


6. Feed – King

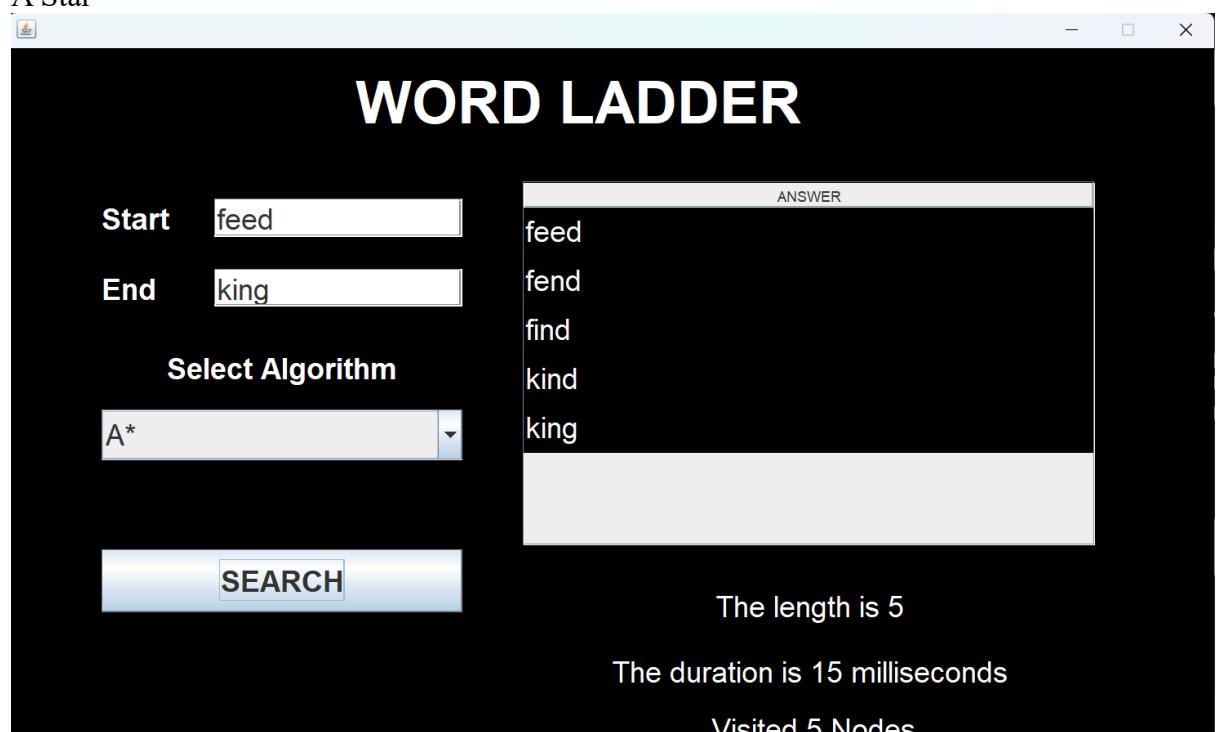
a. UCS



b. GBFS

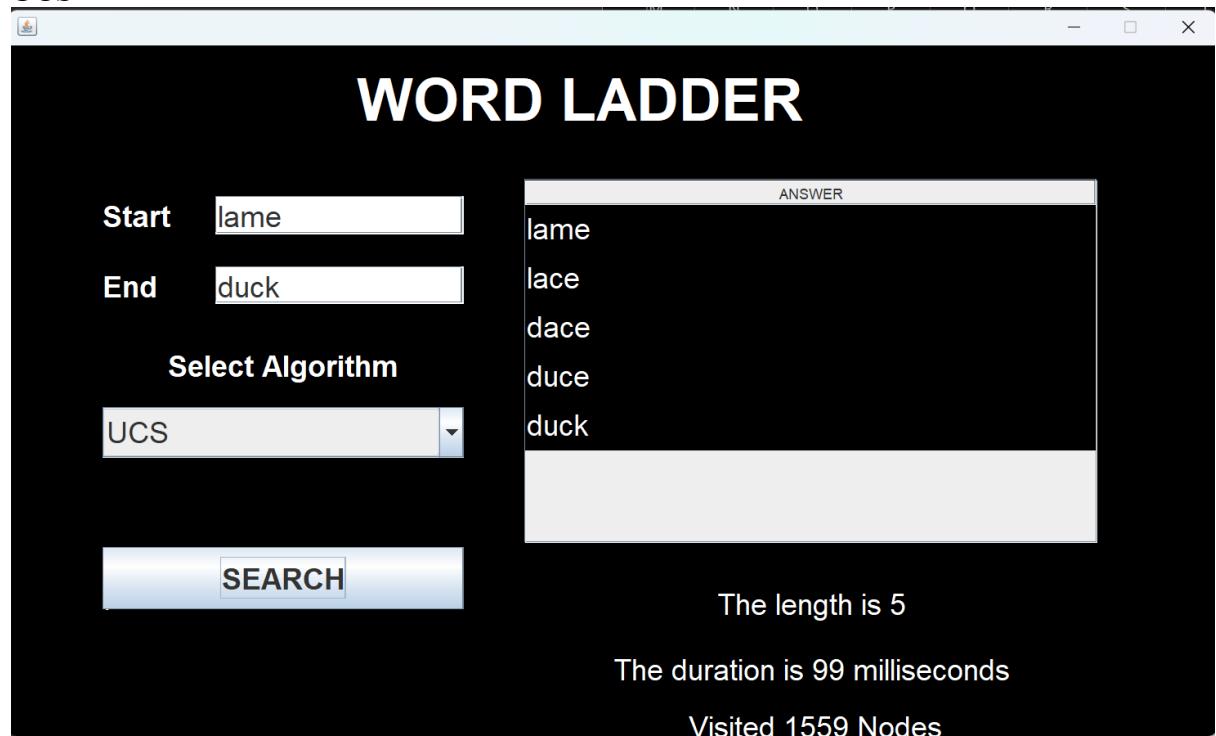


c. A Star

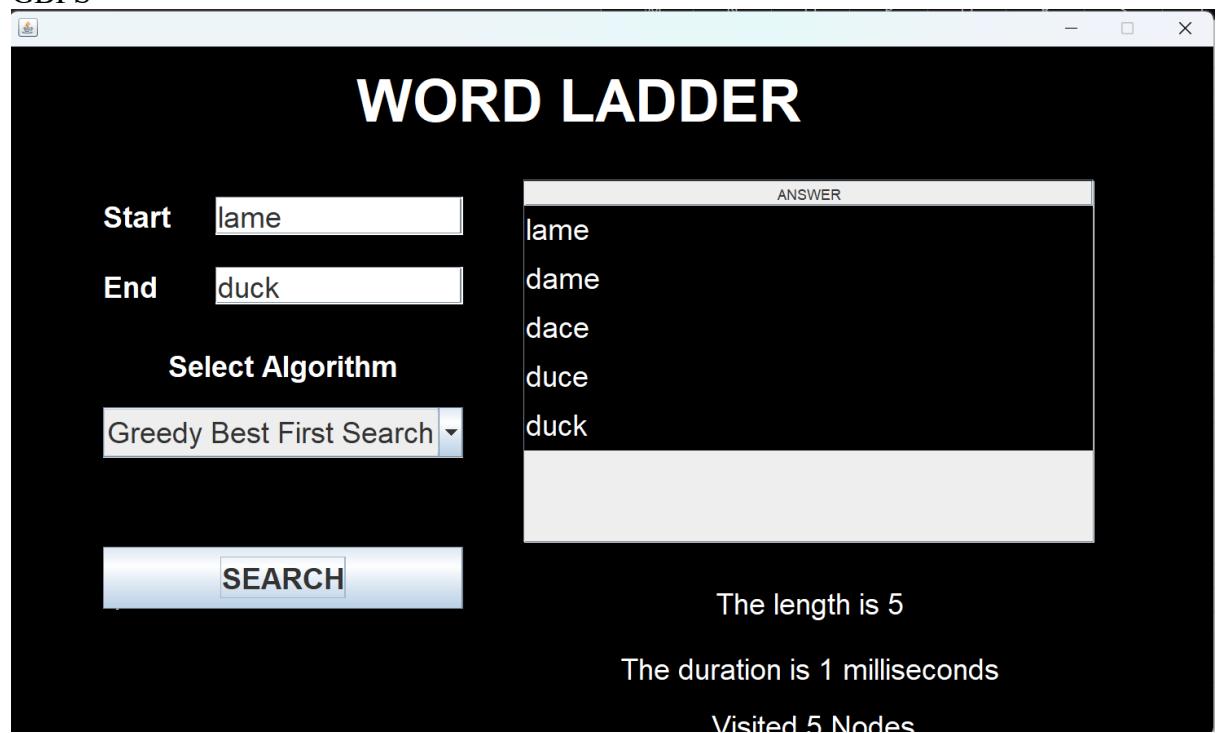


7. Lame – Duck

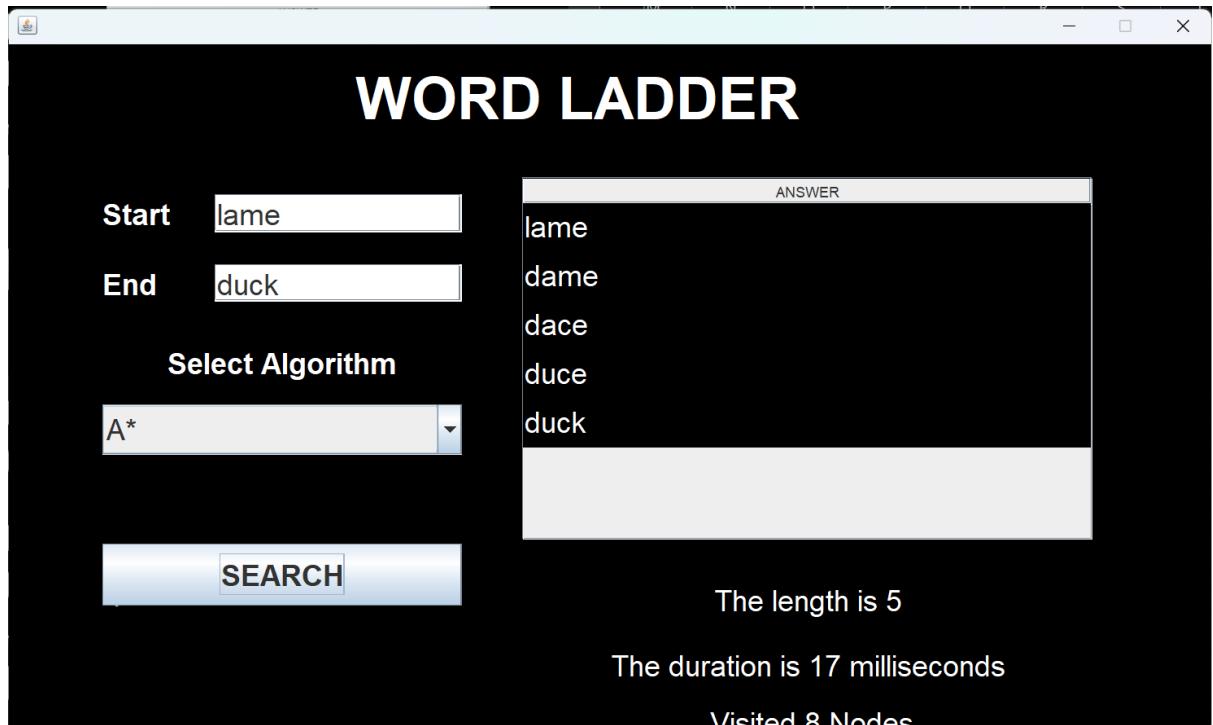
a. UCS



b. GBFS

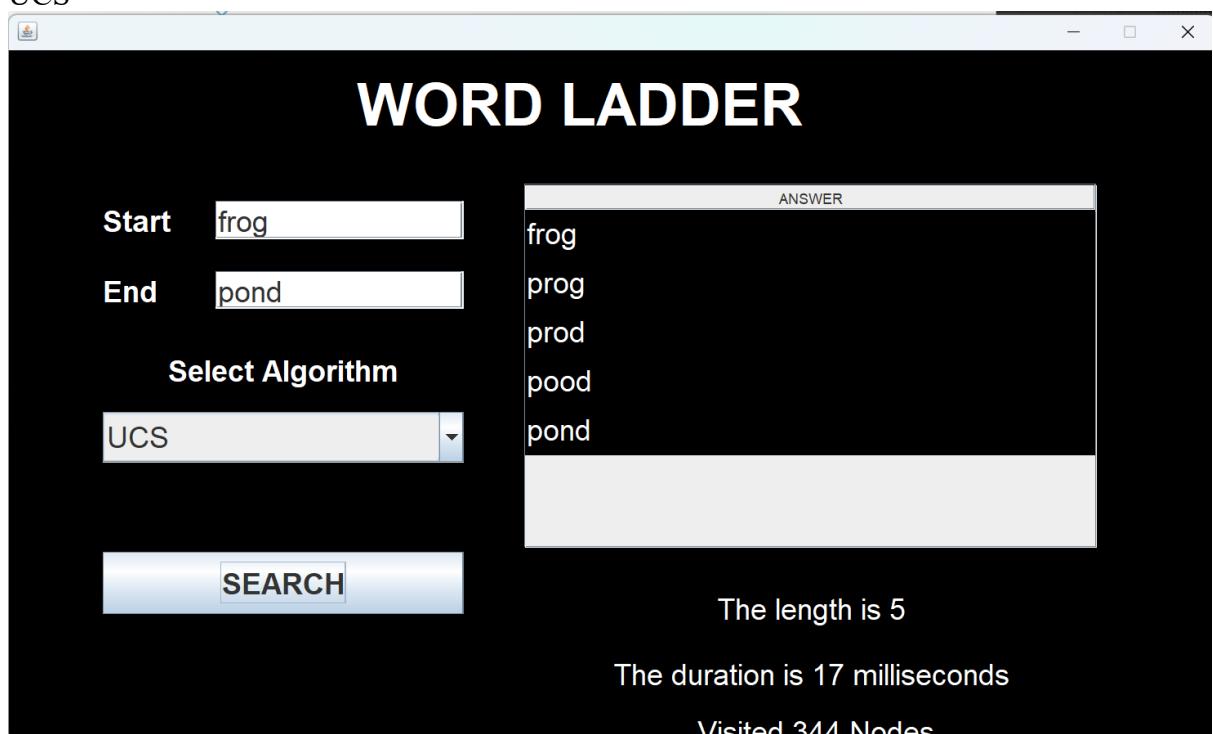


c. A Star

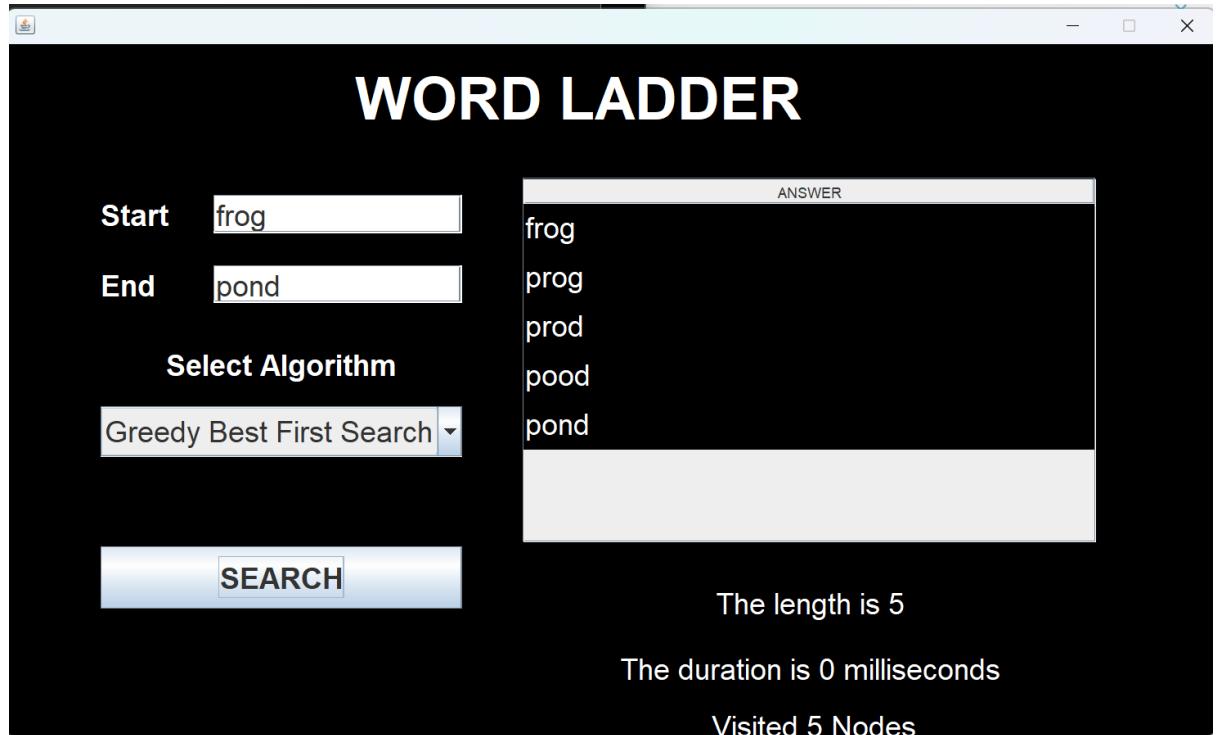


8. Frog – Pond

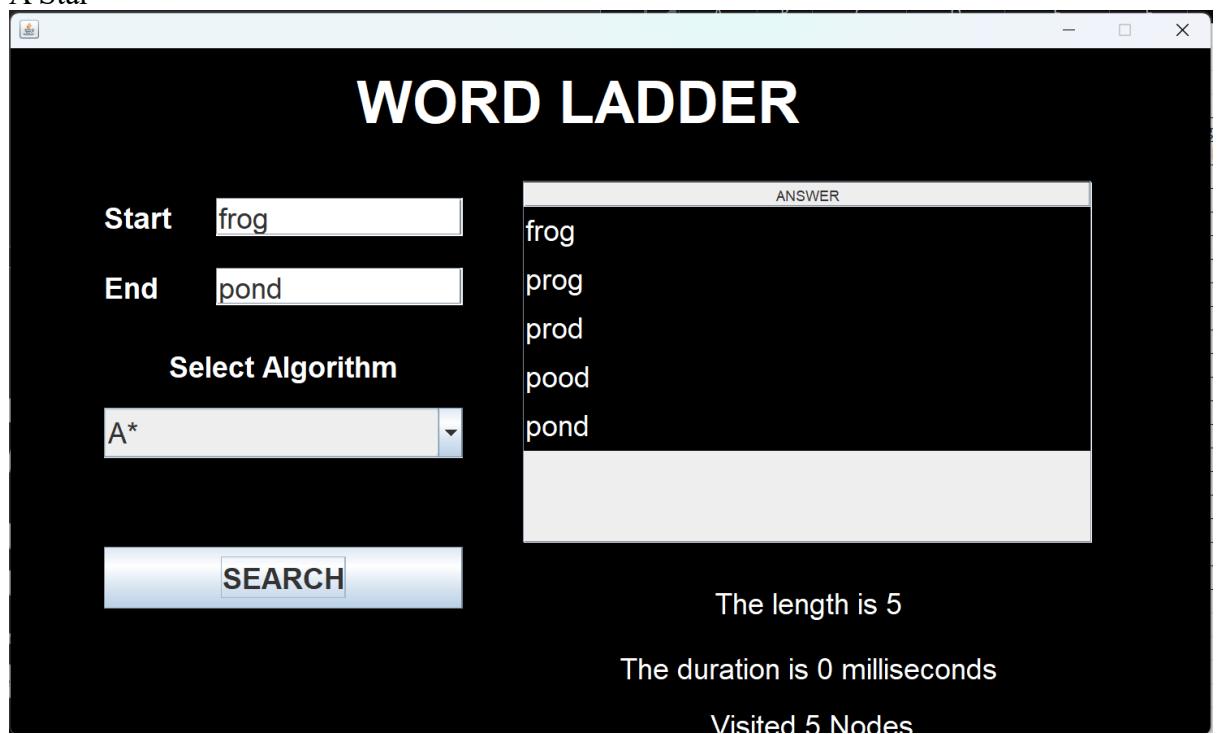
a. UCS



b. GBFS

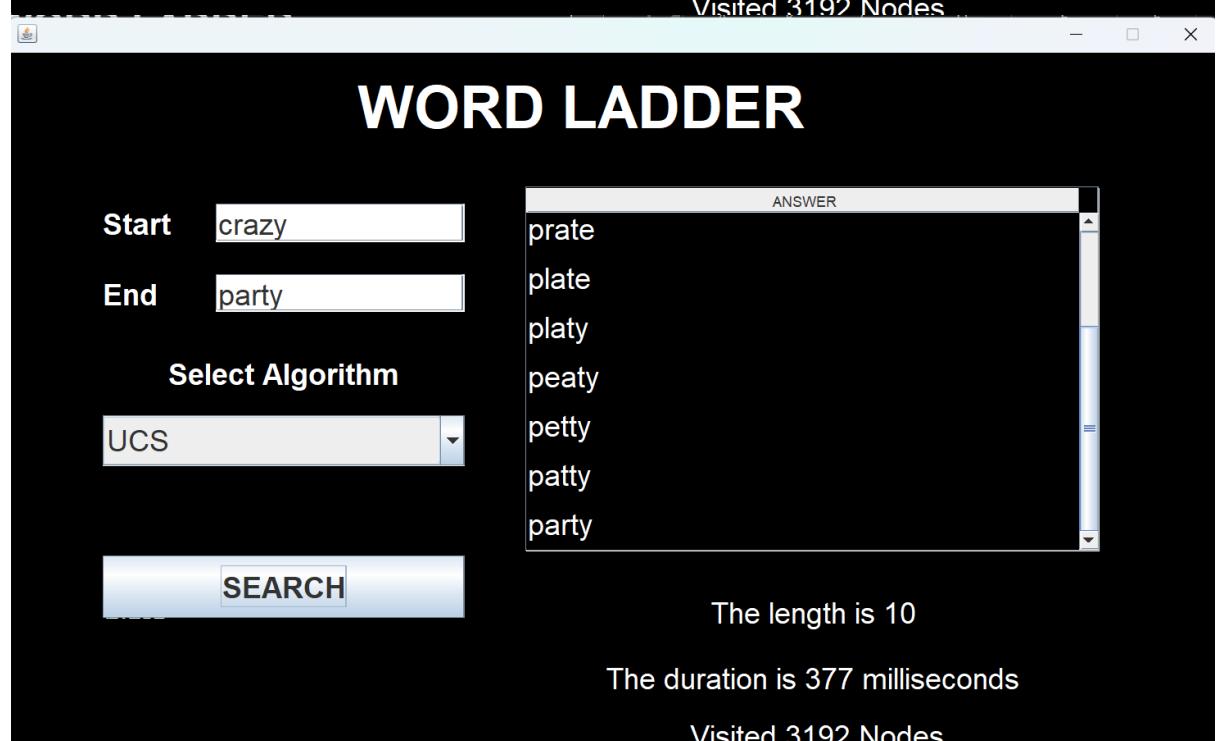
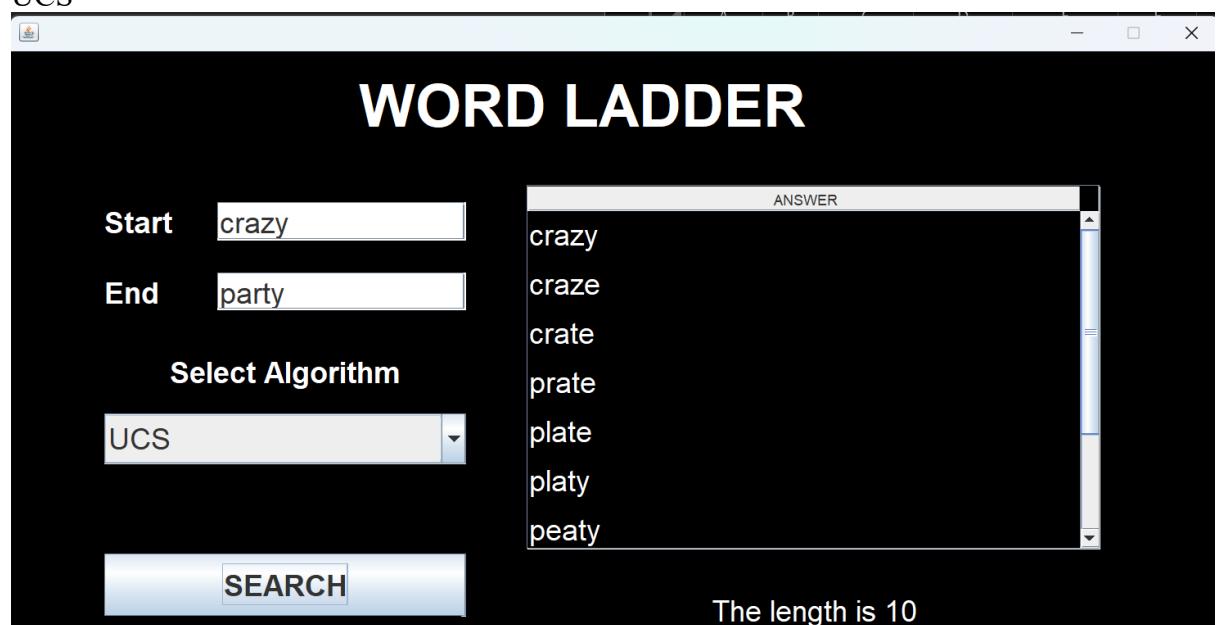


c. A Star



9. Crazy – Party

a. UCS



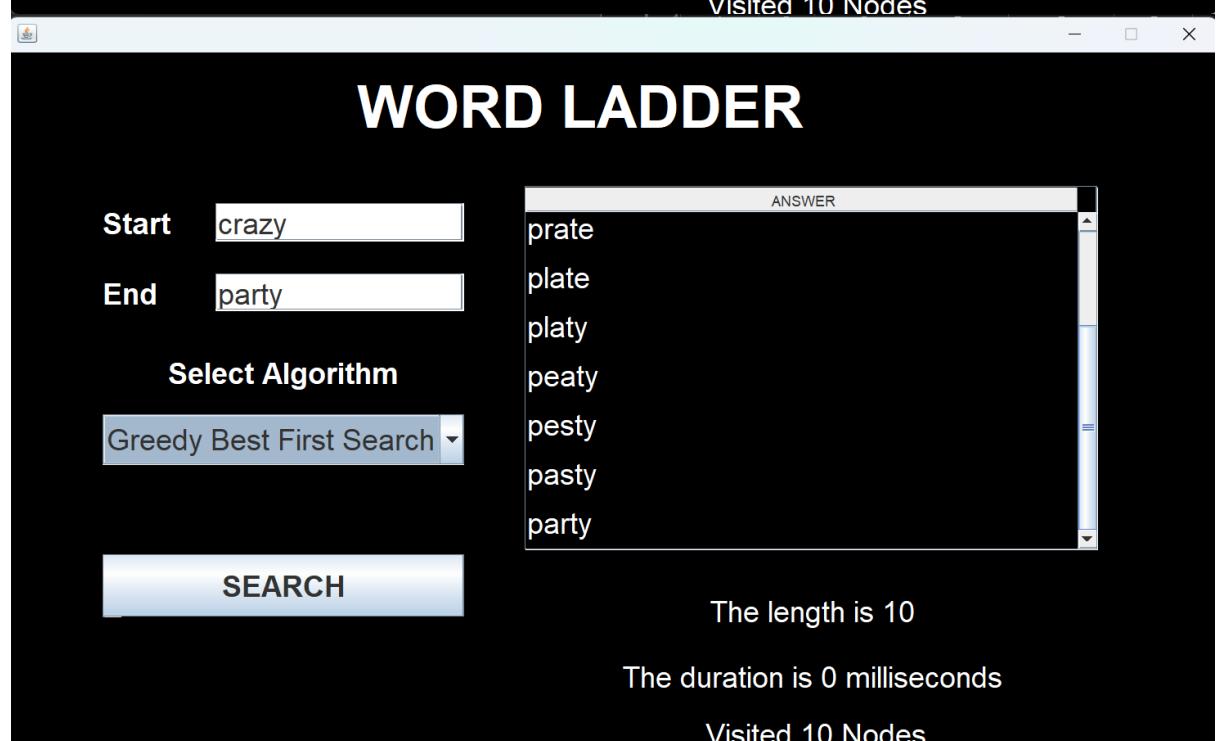
b. GBFS



The length is 10

The duration is 0 milliseconds

Visited 10 Nodes

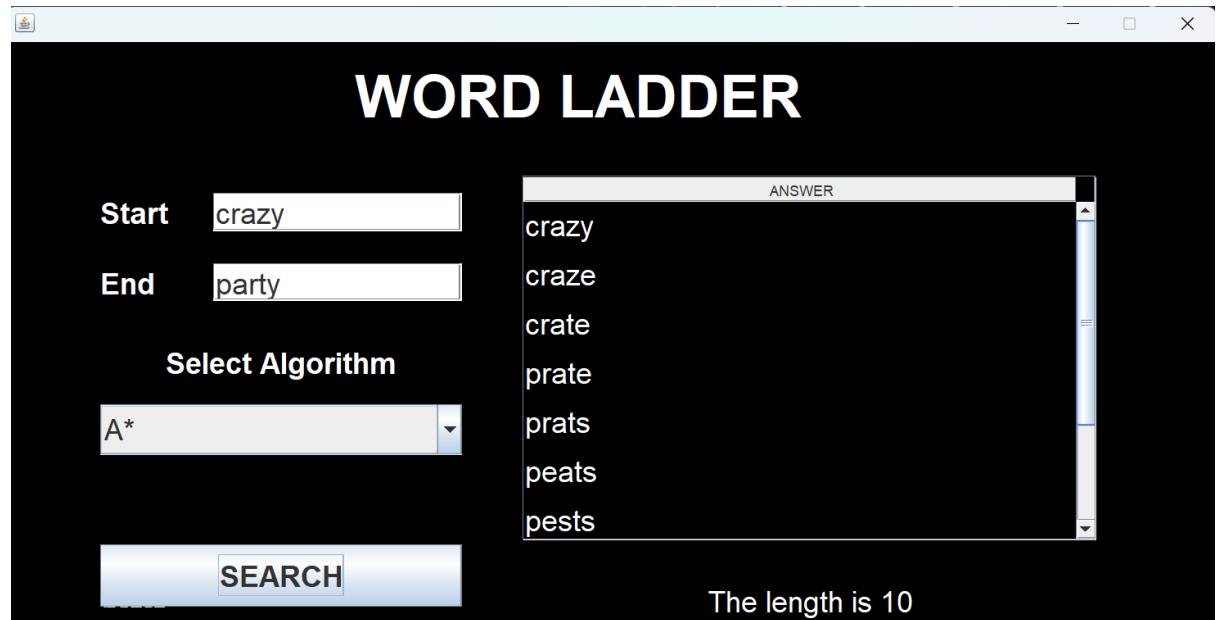


The length is 10

The duration is 0 milliseconds

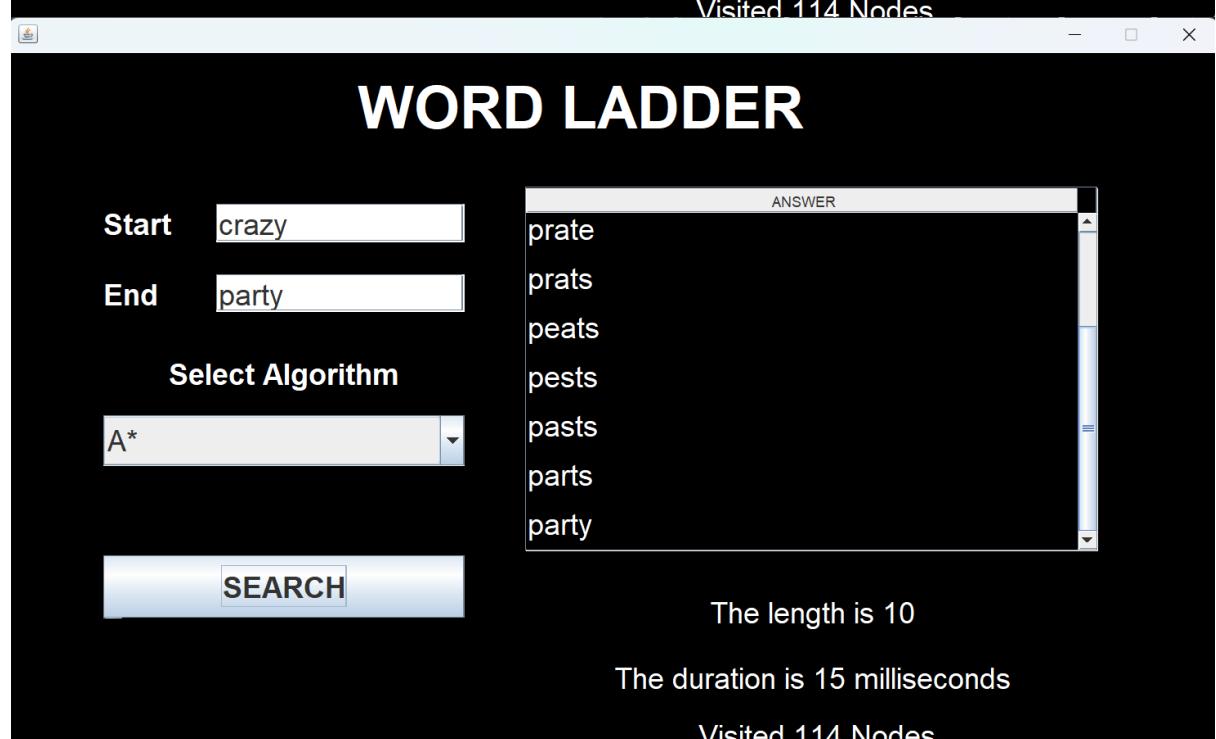
Visited 10 Nodes

c. A Star



The duration is 15 milliseconds

Visited 114 Nodes

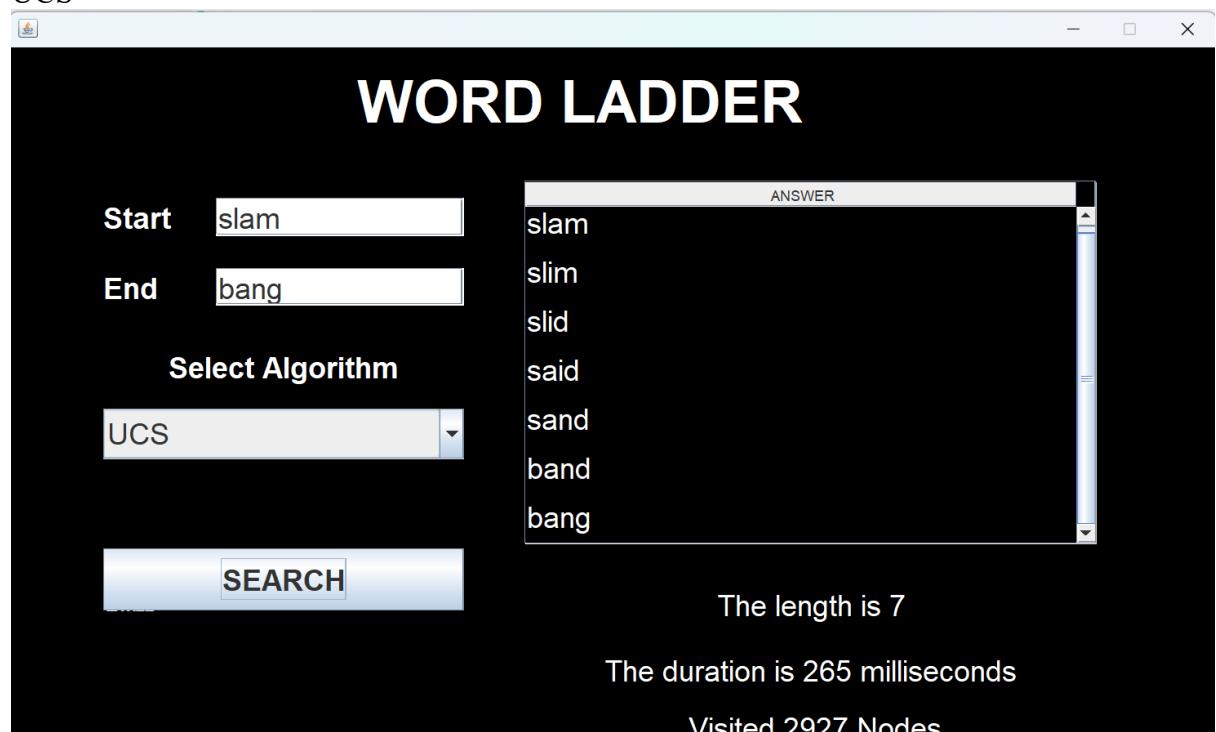


The duration is 15 milliseconds

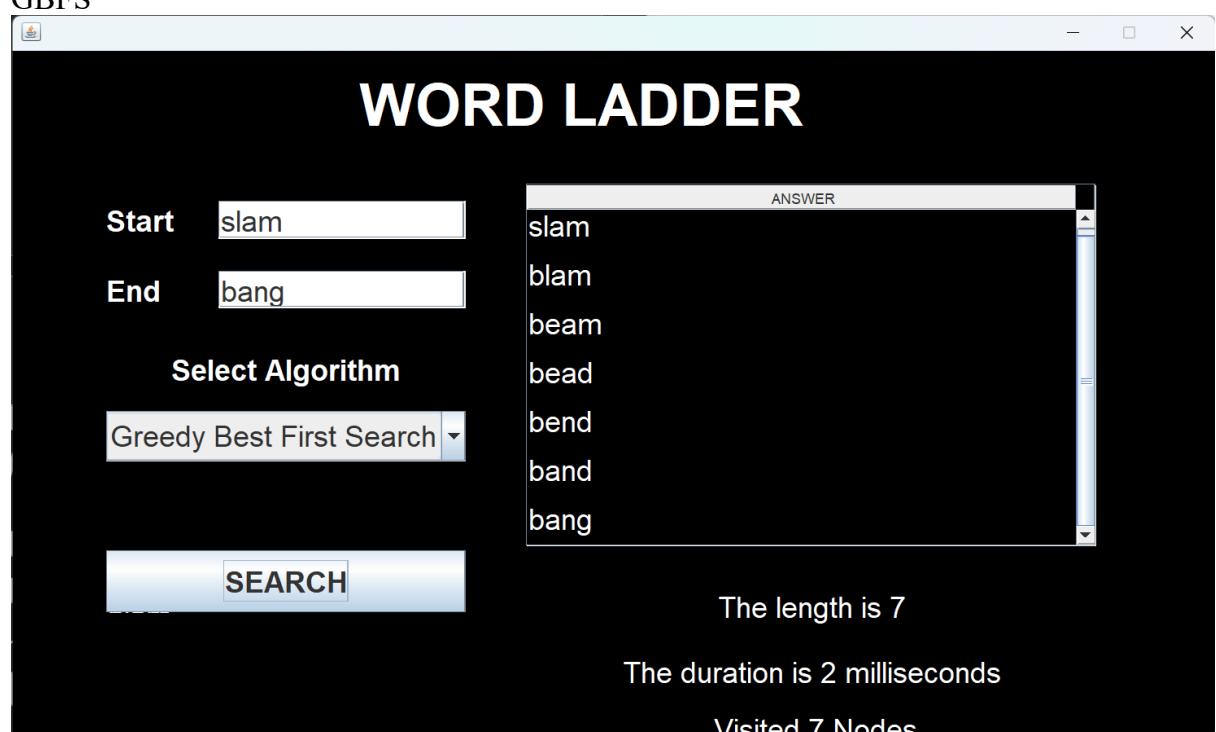
Visited 114 Nodes

10. Slam – Bang

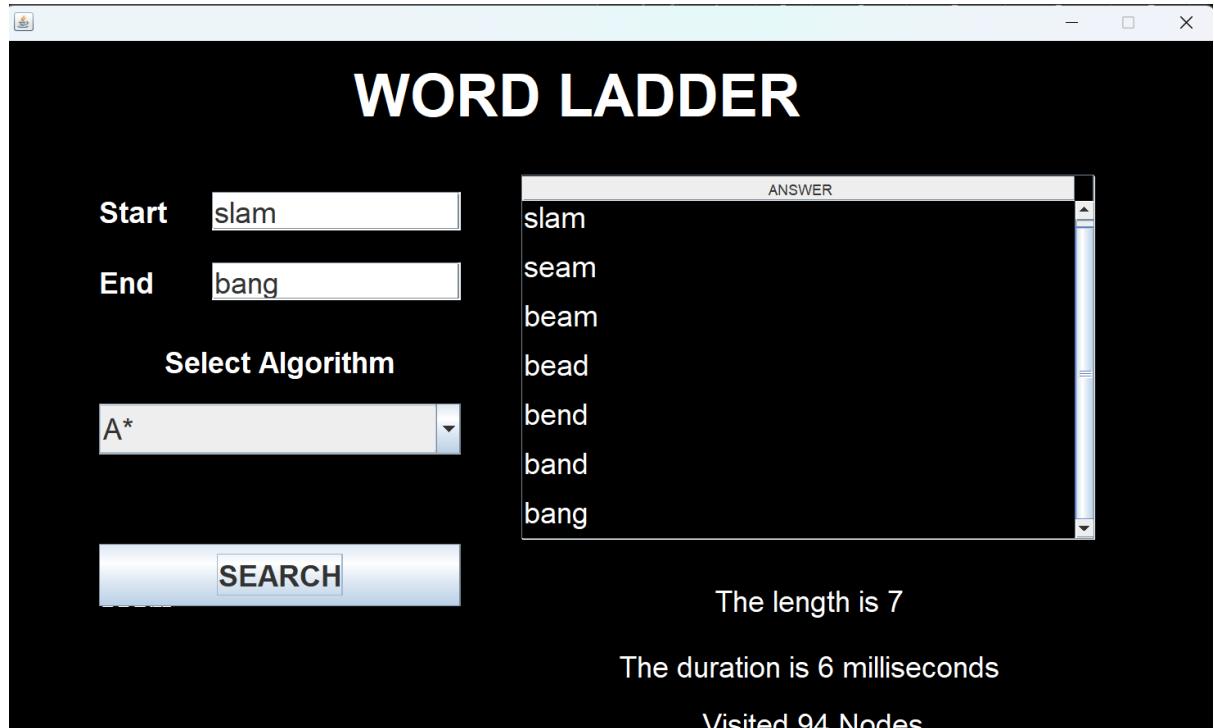
a. UCS



b. GBFS

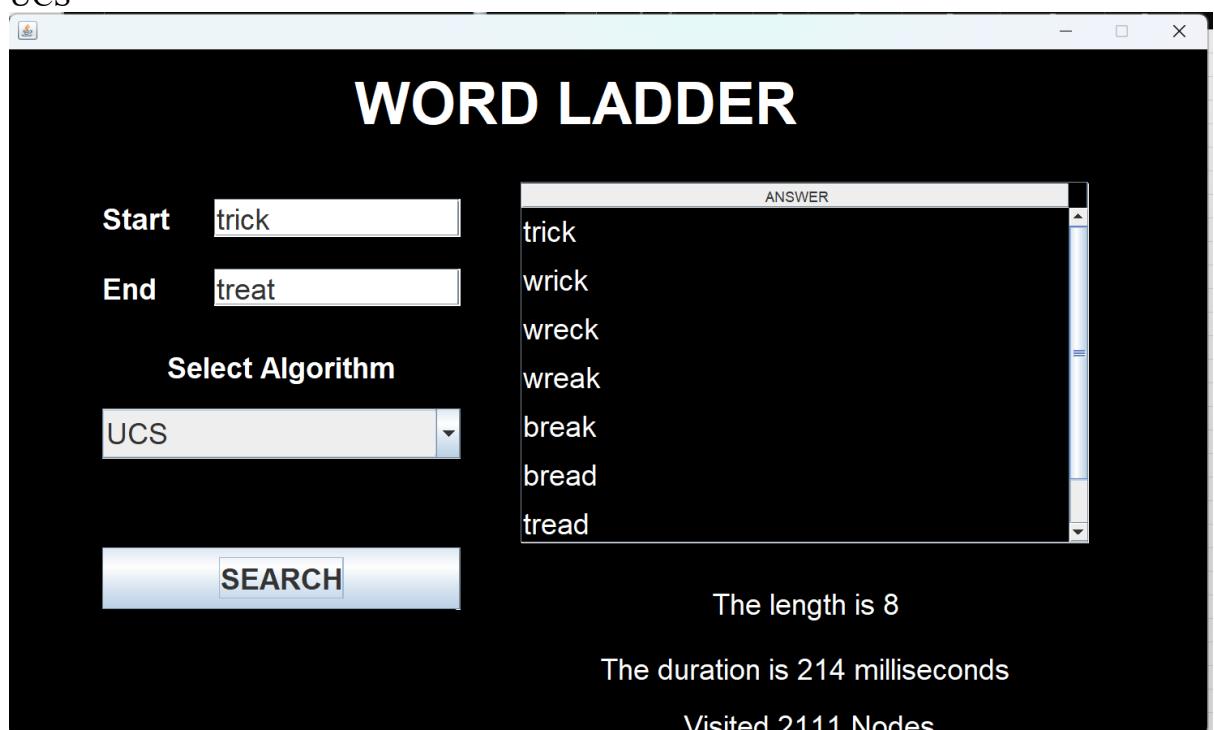


c. A Star



11. Trick – Treat

a. UCS



b. UCS

The WORD LADDER application interface. The search parameters are Start: trick and End: treat. The algorithm selected is UCS. The search results window is titled "ANSWER" and lists the words: wrick, wreck, wreak, break, bread, tread, treat. Below the results, the message "The length is 8" is displayed. At the bottom, performance metrics are shown: "The duration is 214 milliseconds" and "Visited 2111 Nodes".

WORD LADDER

Start trick

End treat

Select Algorithm

UCS

SEARCH

wrick
wreck
wreak
break
bread
tread
treat

ANSWER

The length is 8

The duration is 214 milliseconds

Visited 2111 Nodes

b. GBFS

The WORD LADDER application interface. The search parameters are Start: trick and End: treat. The algorithm selected is Greedy Best First Search. The search results window is titled "ANSWER" and displays the message "No path". Below the results, the message "The duration is 0 milliseconds" is displayed. At the bottom, the message "Visited 4 Nodes" is shown.

WORD LADDER

Start trick

End treat

Select Algorithm

Greedy Best First Search

SEARCH

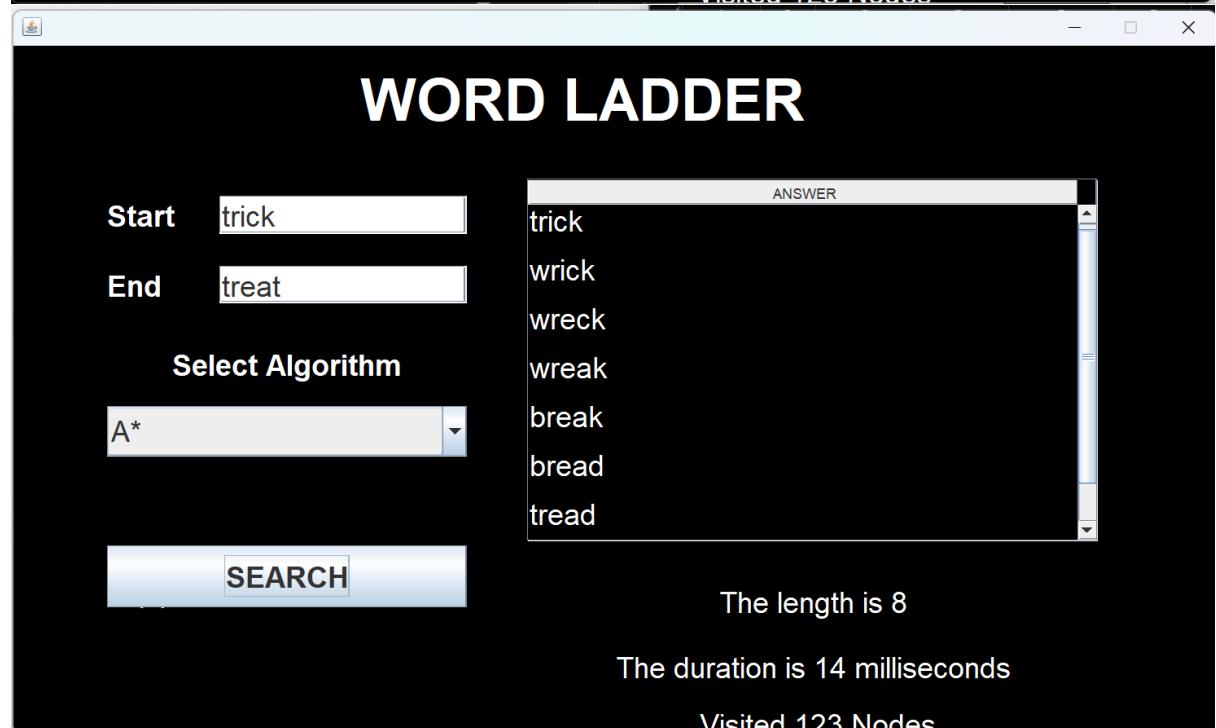
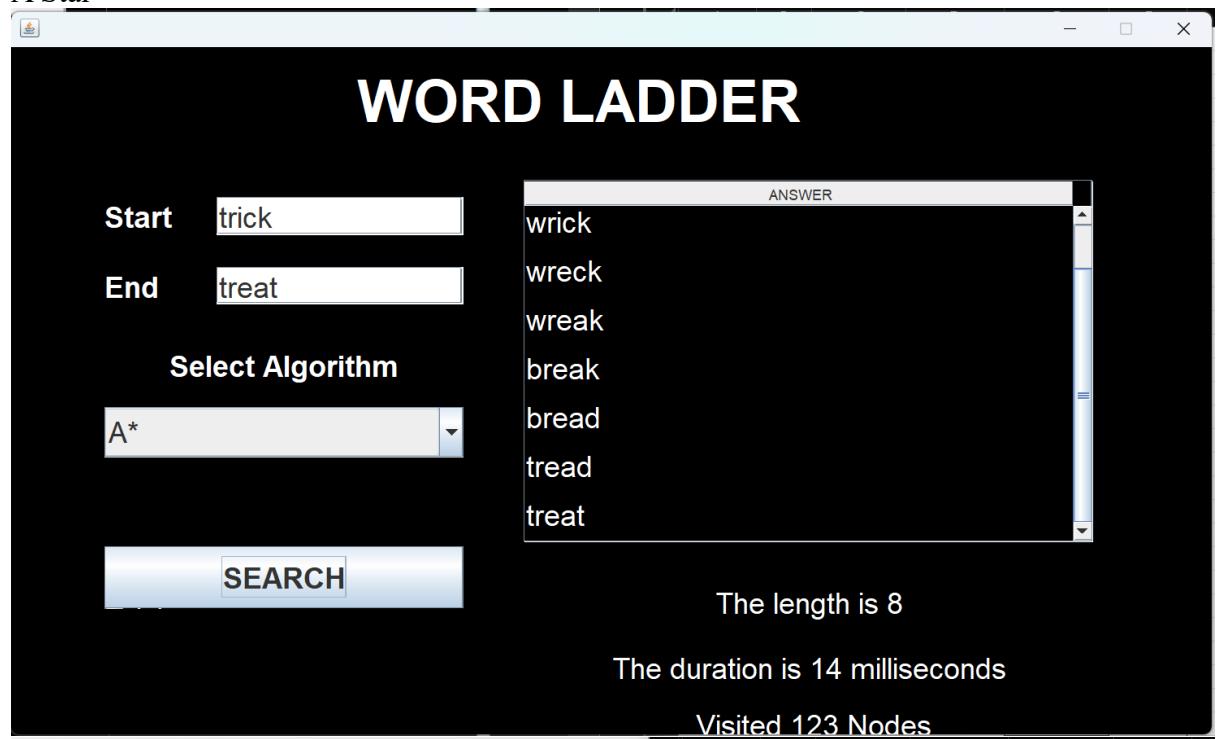
ANSWER

No path

The duration is 0 milliseconds

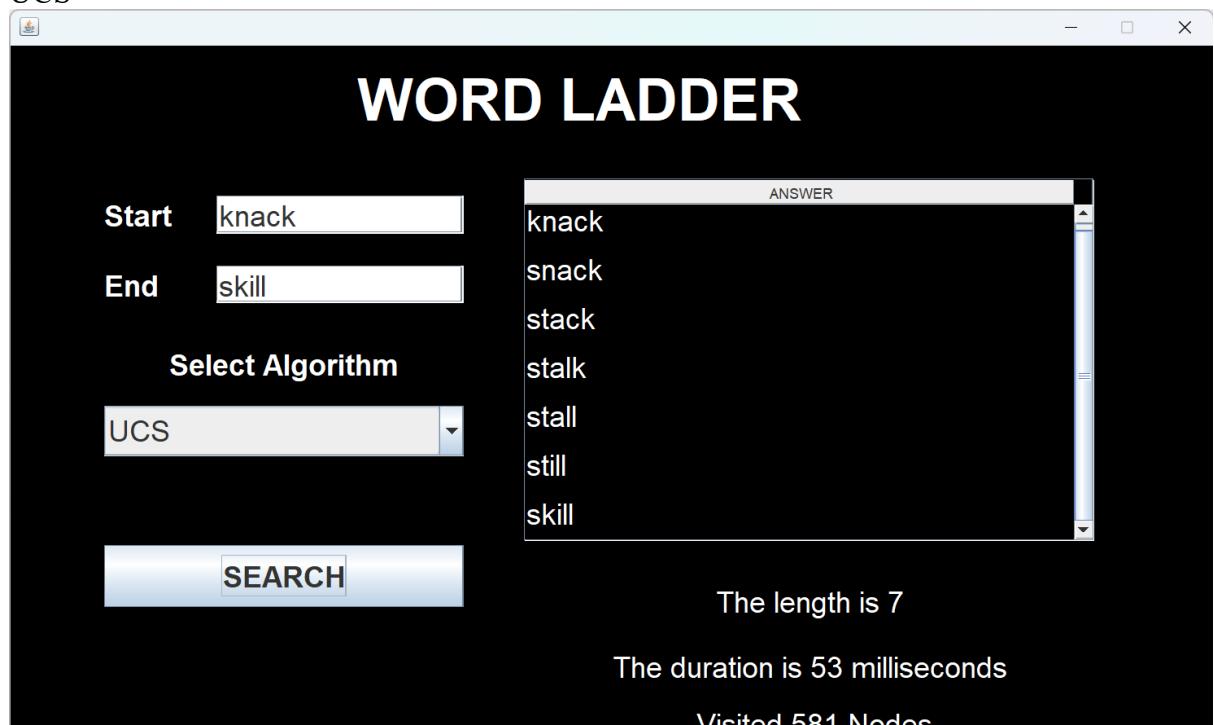
Visited 4 Nodes

c. A Star

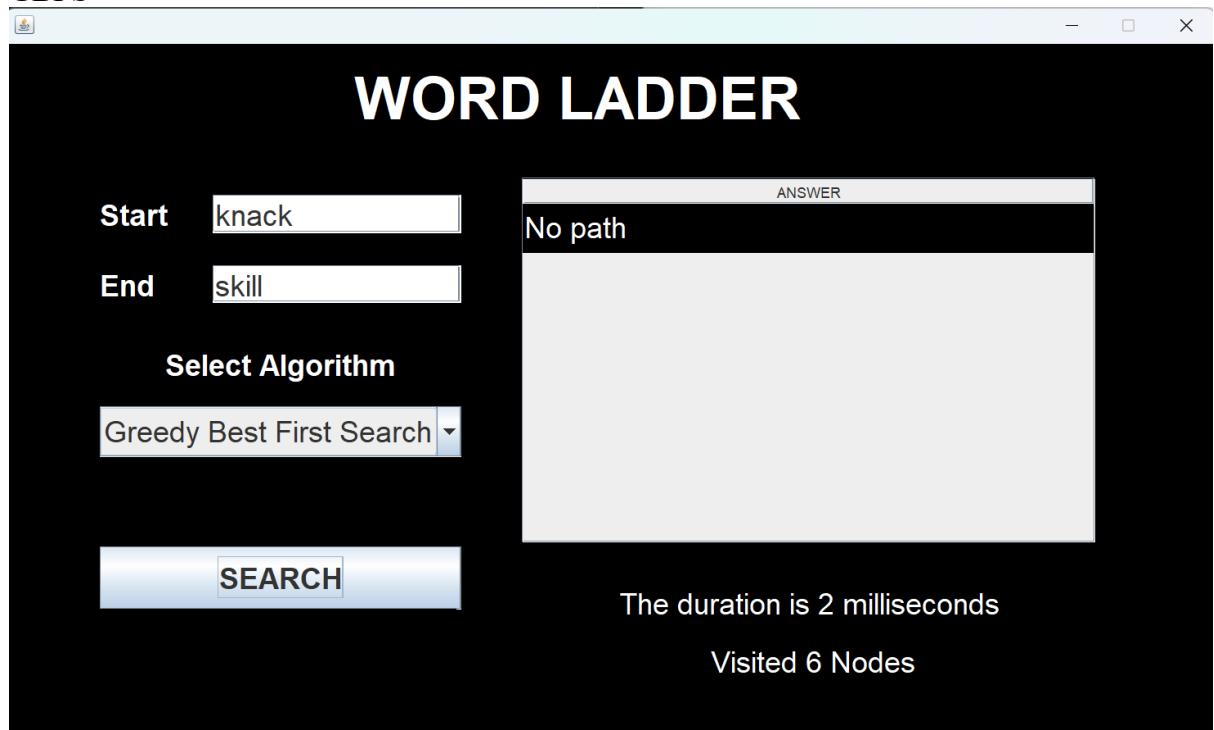


12. Knack – Skill

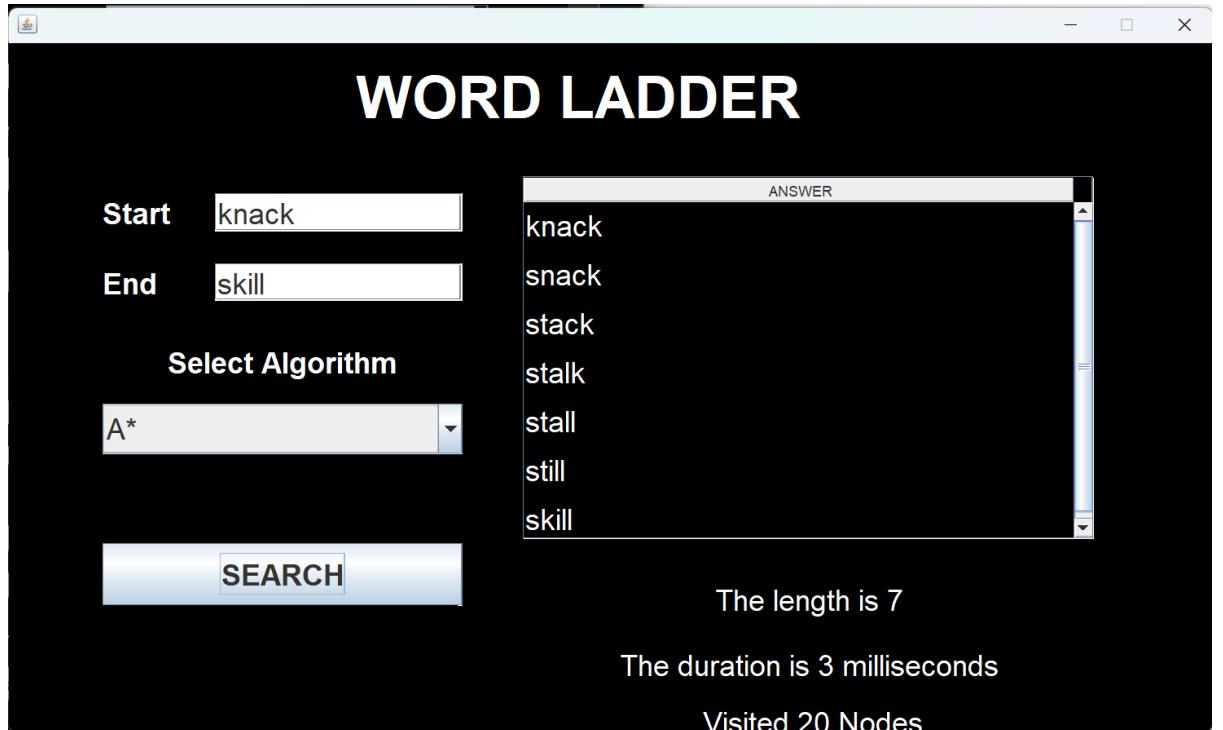
a. UCS



b. GBFS

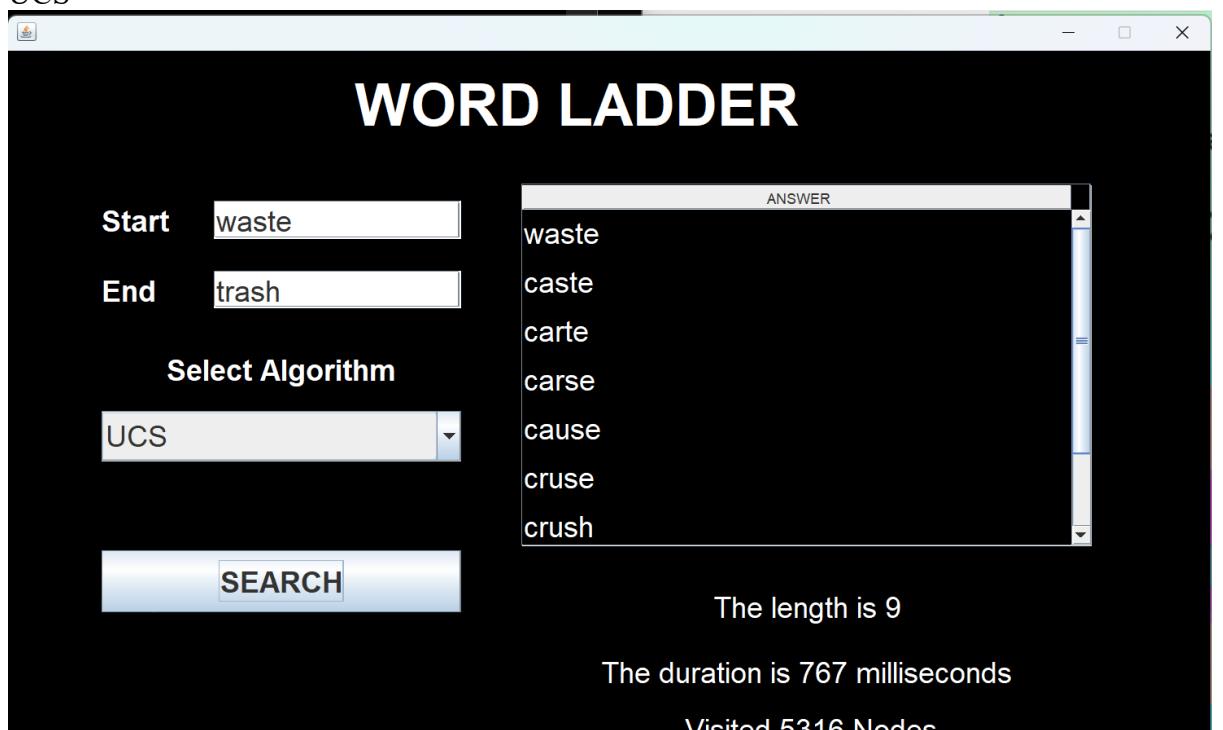


c. A Star



13. Waste – Trash

a. UCS



b. GBFS

The WORD LADDER application interface is shown in two states: one for UCS and one for GBFS.

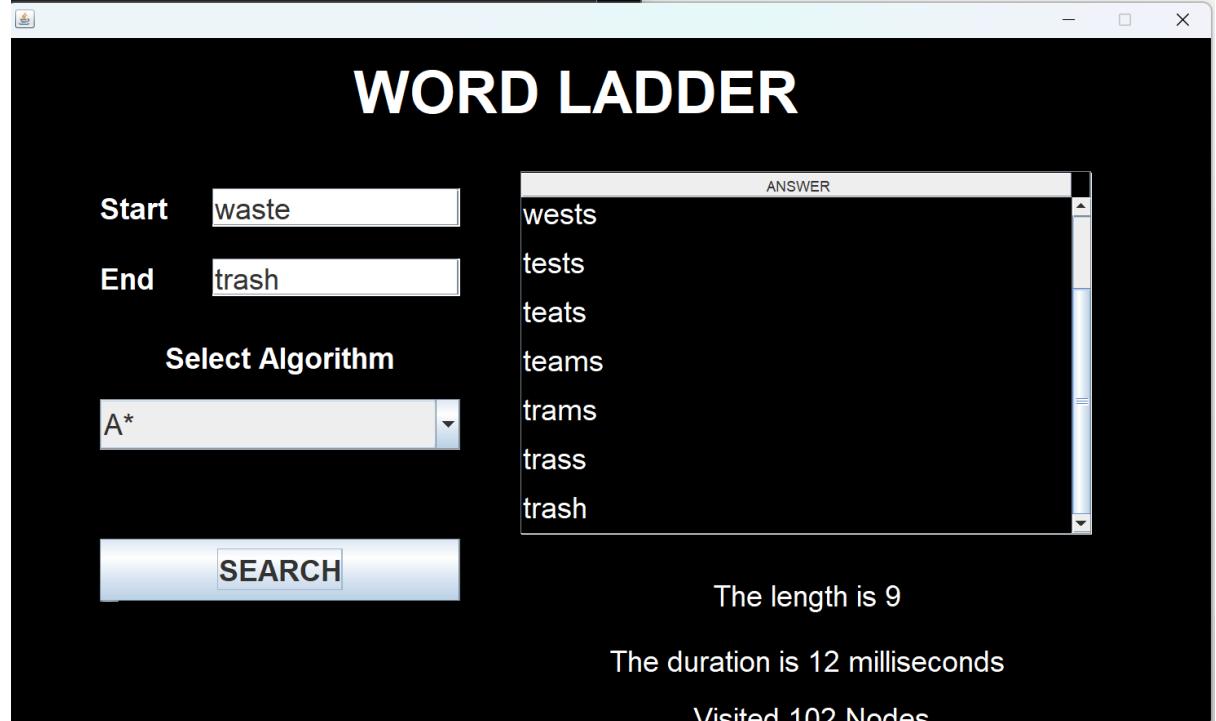
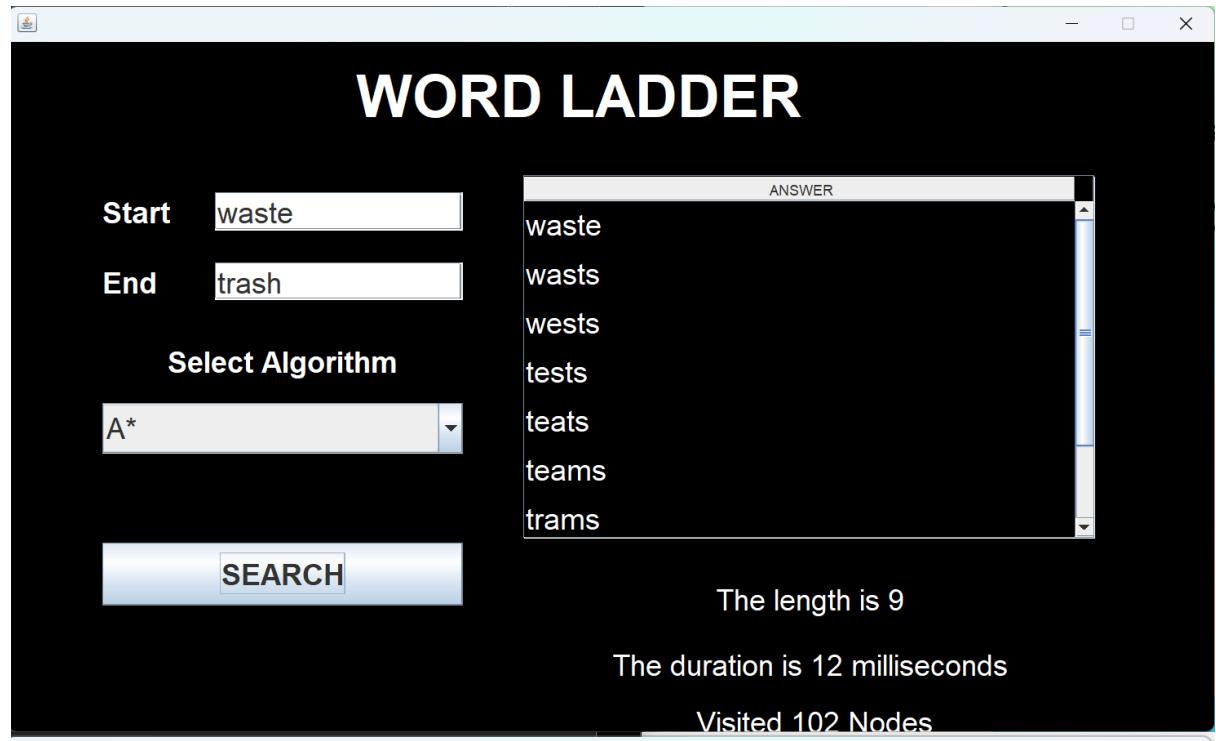
UCS Results:

- Start: waste
- End: trash
- Select Algorithm: UCS
- SEARCH button
- ANSWER window:
 - carte
 - carse
 - cause
 - cruse
 - crush
 - crash
 - trash
- The length is 9
- The duration is 767 milliseconds
- Visited 5316 Nodes

GBFS Results:

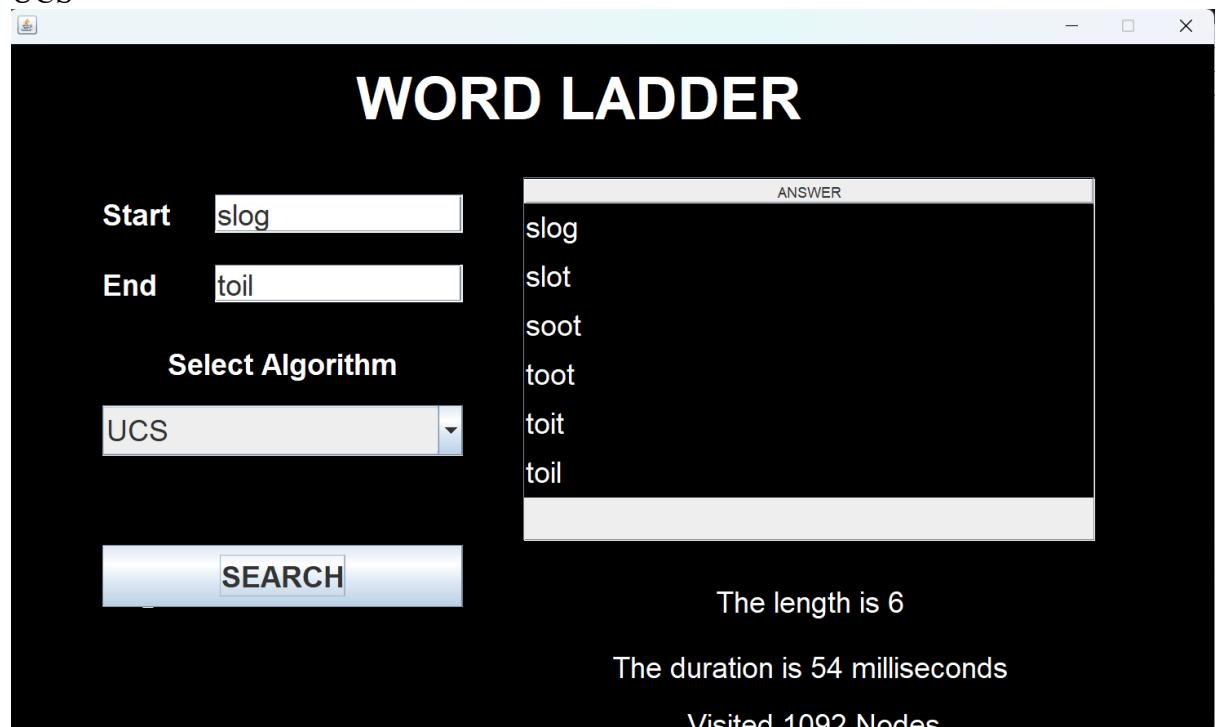
- Start: waste
- End: trash
- Select Algorithm: Greedy Best First Search
- SEARCH button
- ANSWER window:
 - No path
- The duration is 1 milliseconds
- Visited 4 Nodes

c. A Star

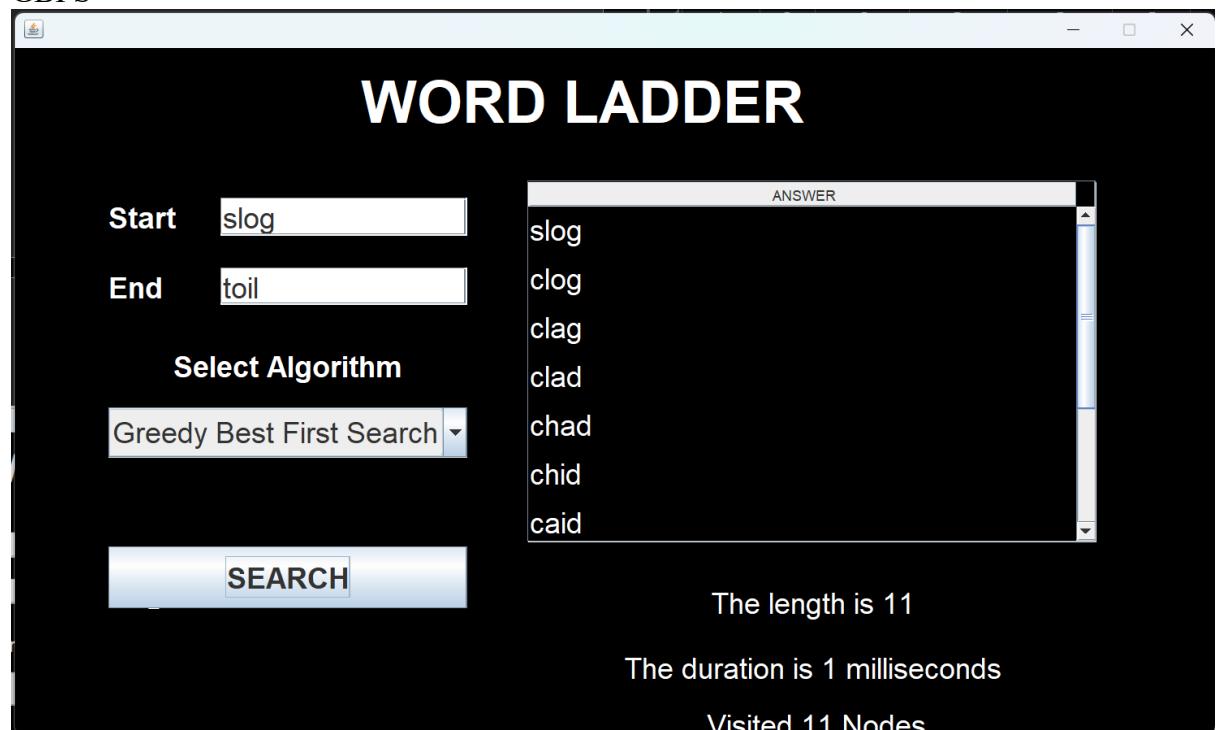


14. Slog – Toil

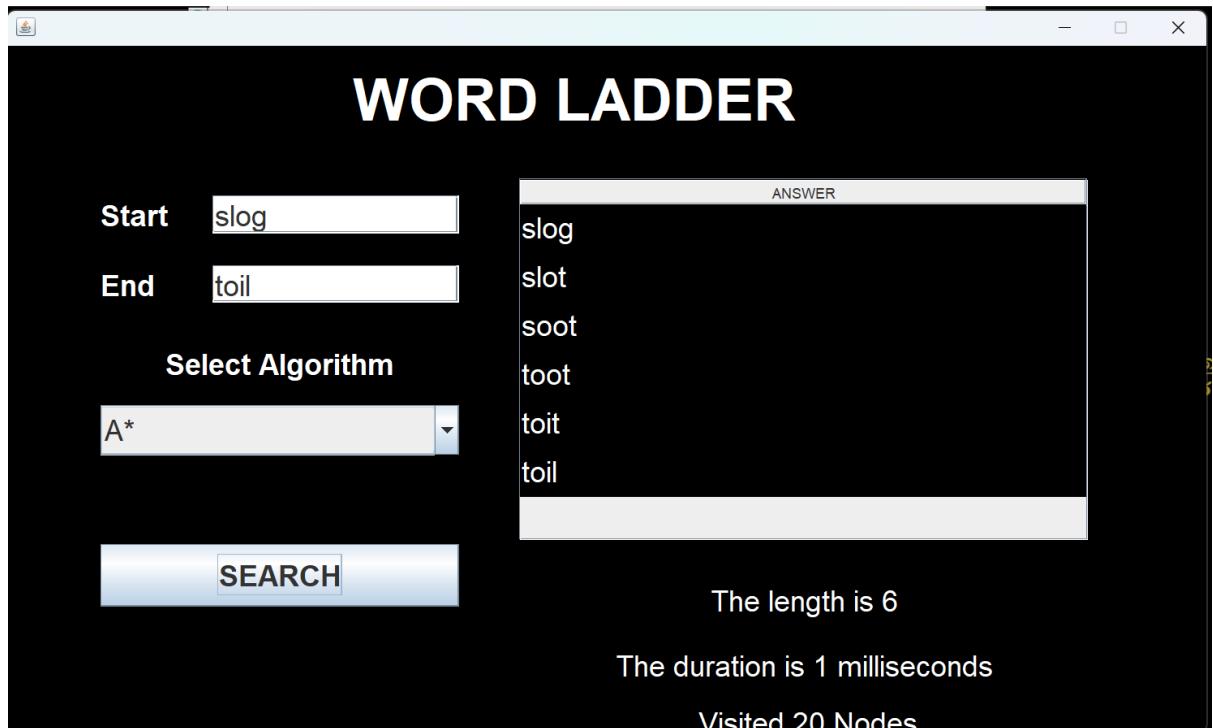
a. UCS



b. GBFS

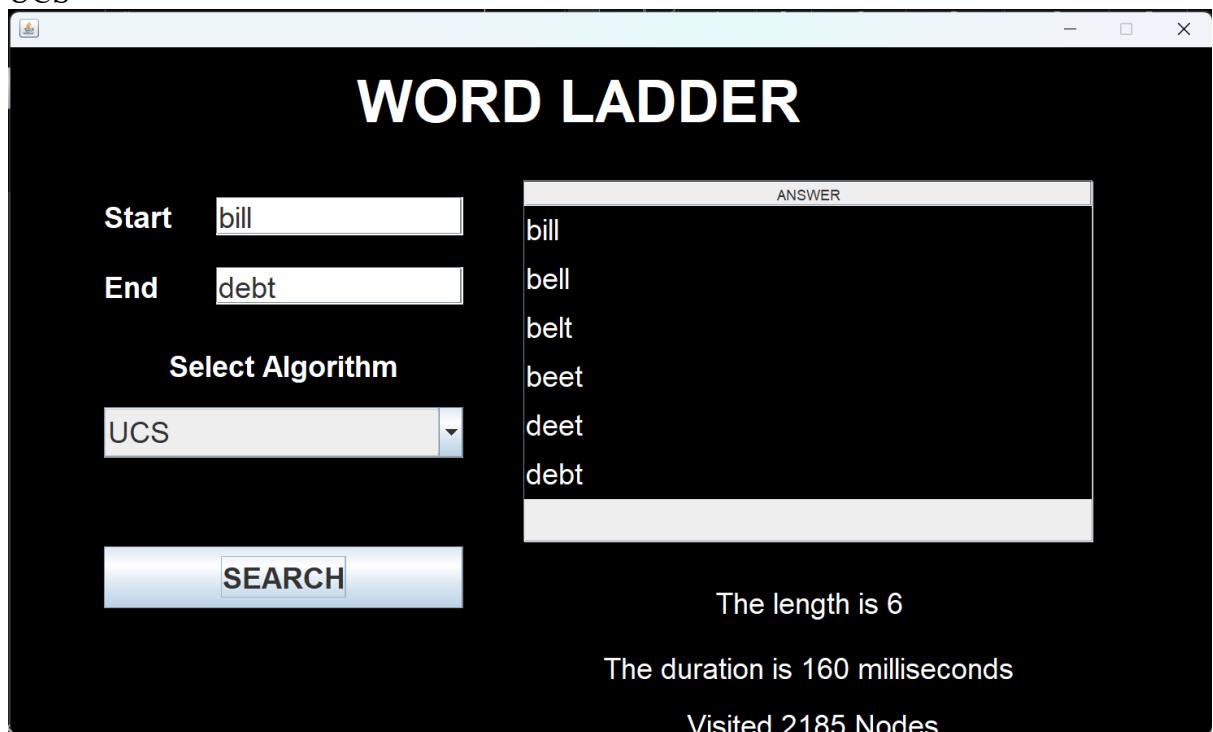


c. A Star

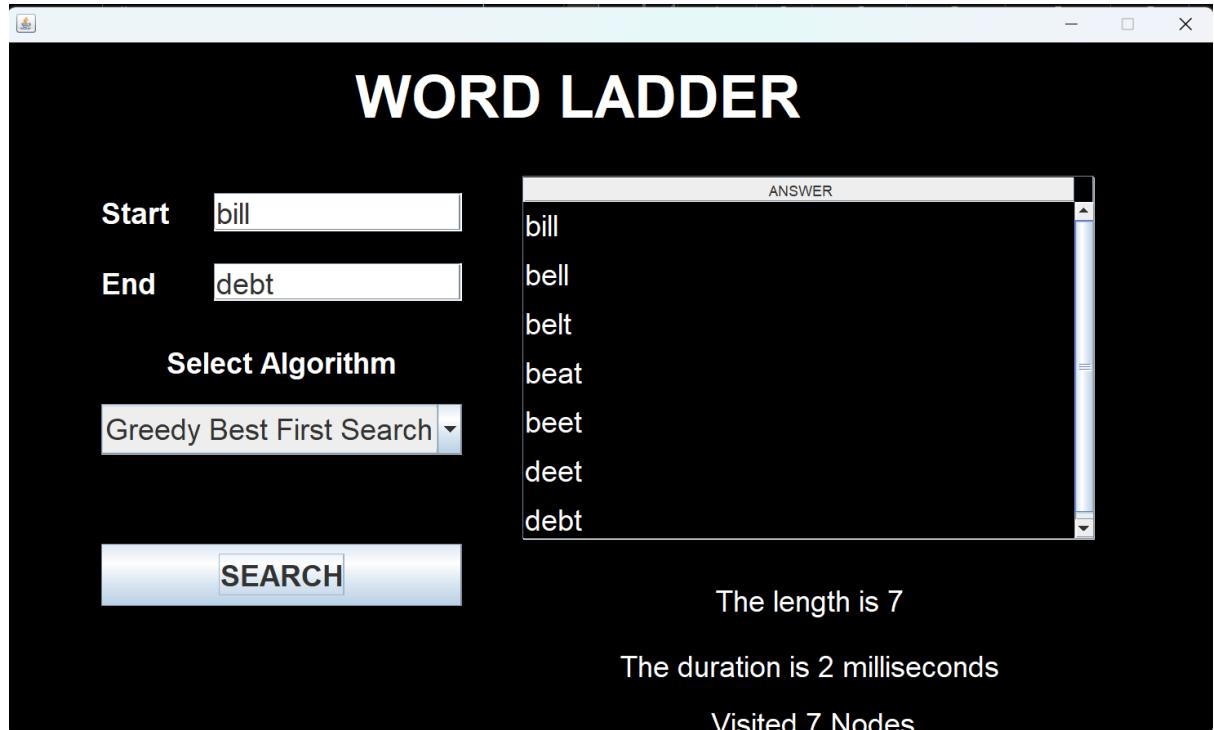


15. Bill – Debt

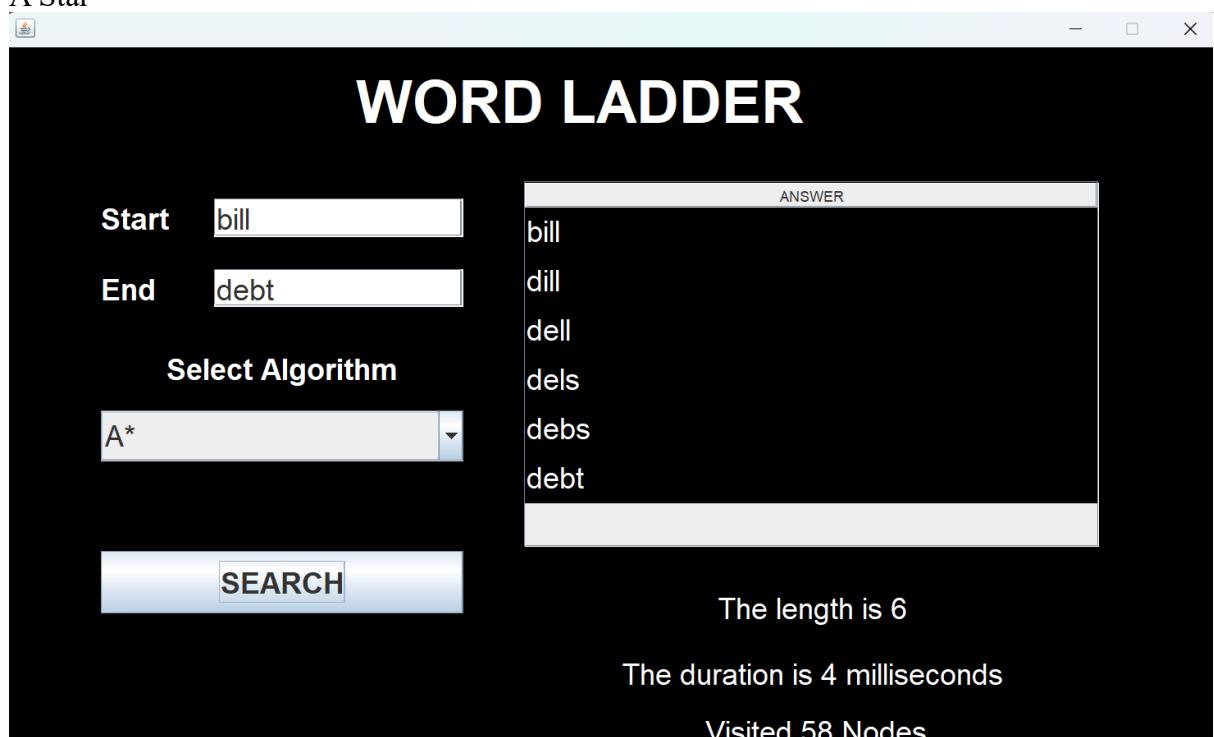
a. UCS



b. GBFS

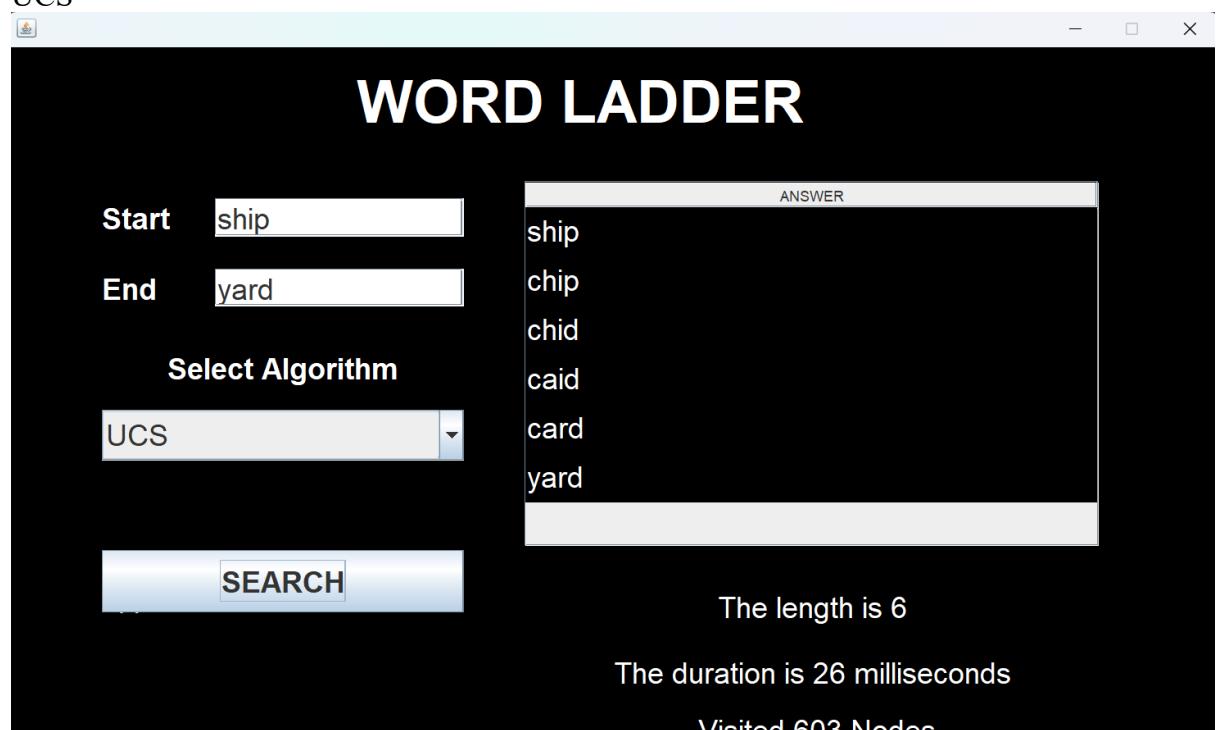


c. A Star

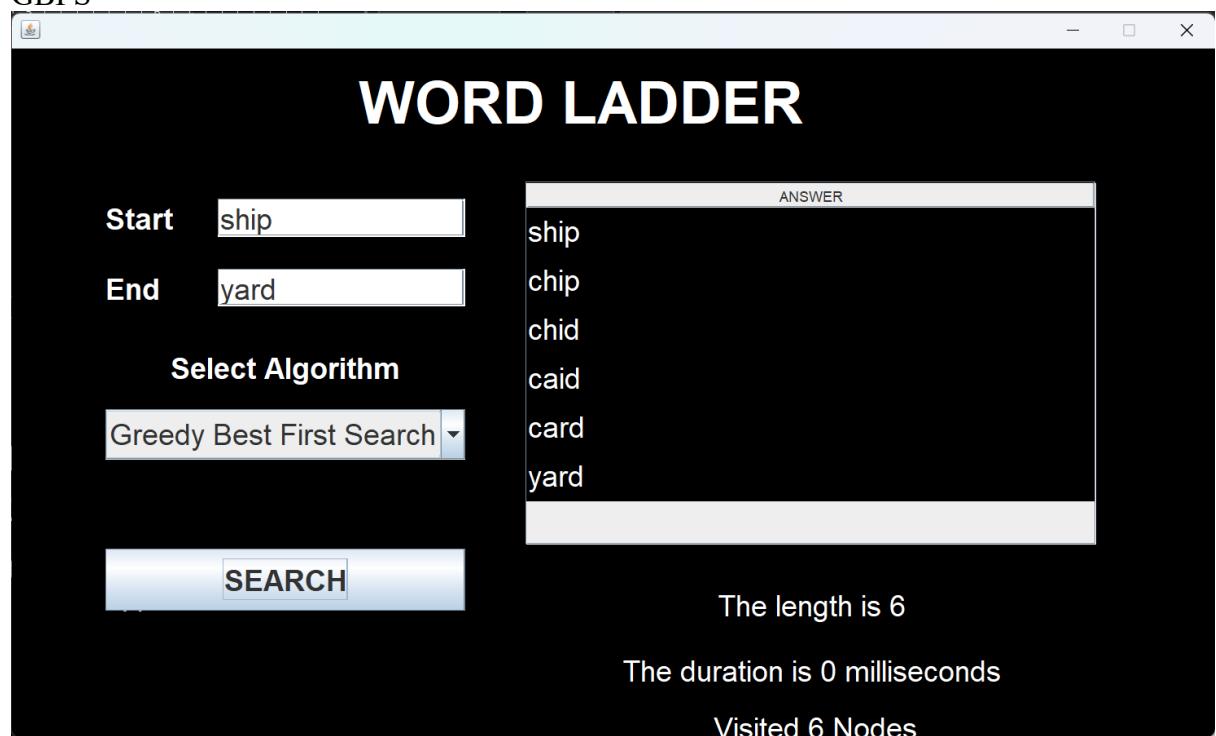


16. Ship – Yard

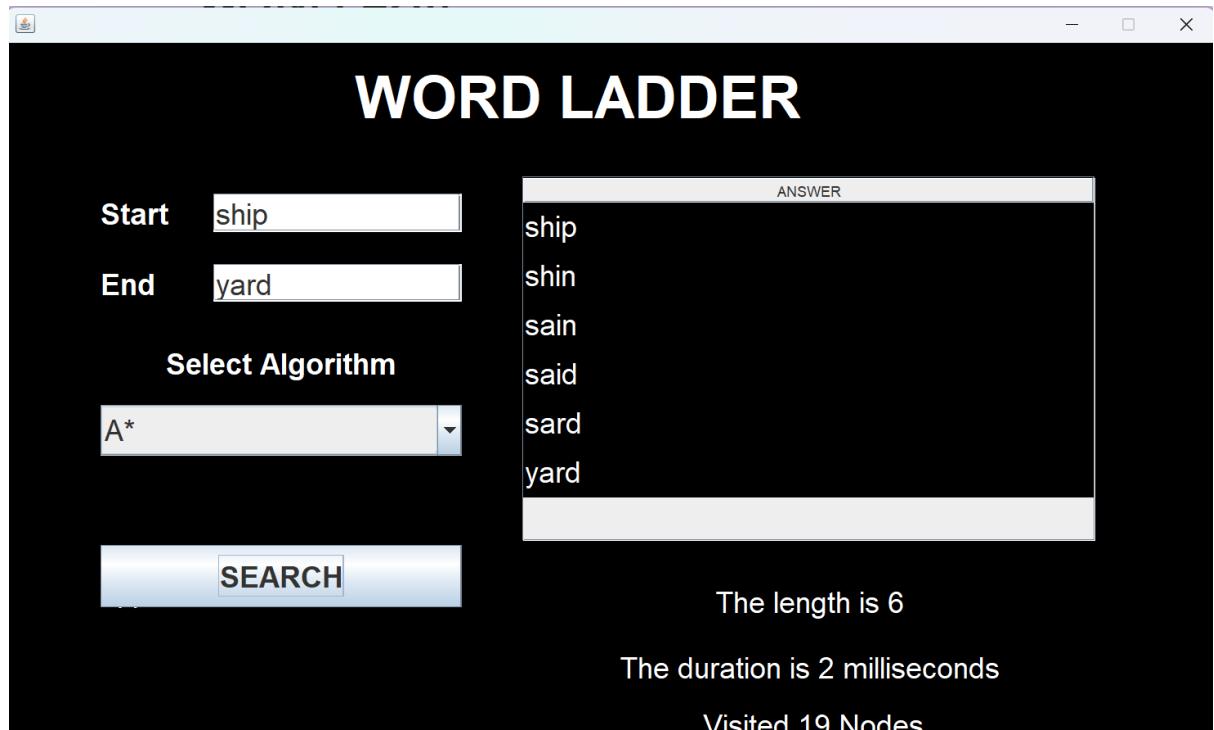
a. UCS



b. GBFS

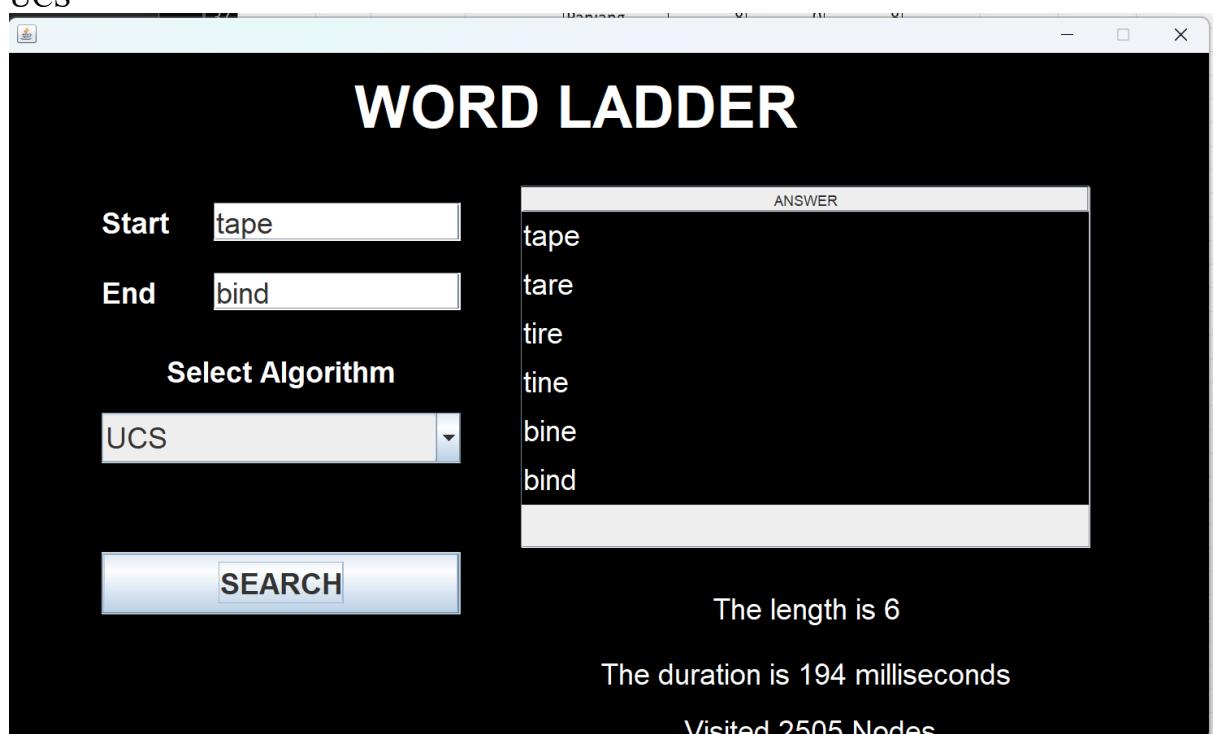


c. A Star

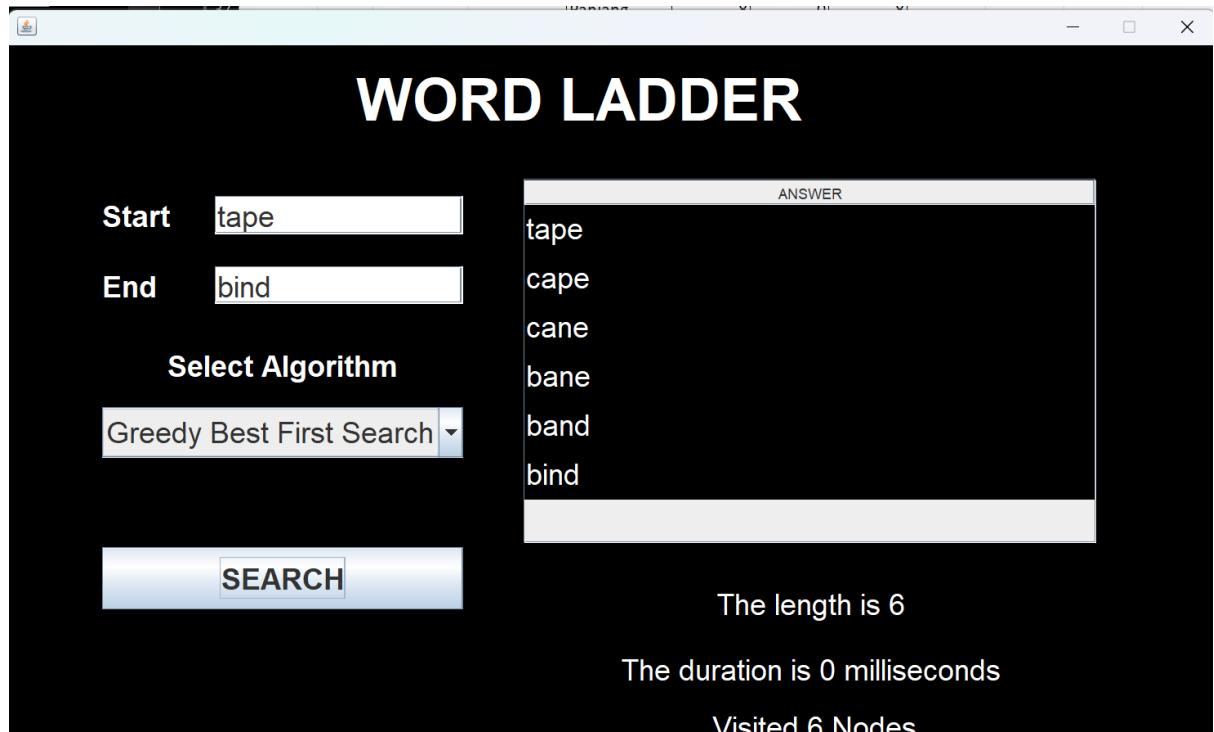


17. Tape – Bind

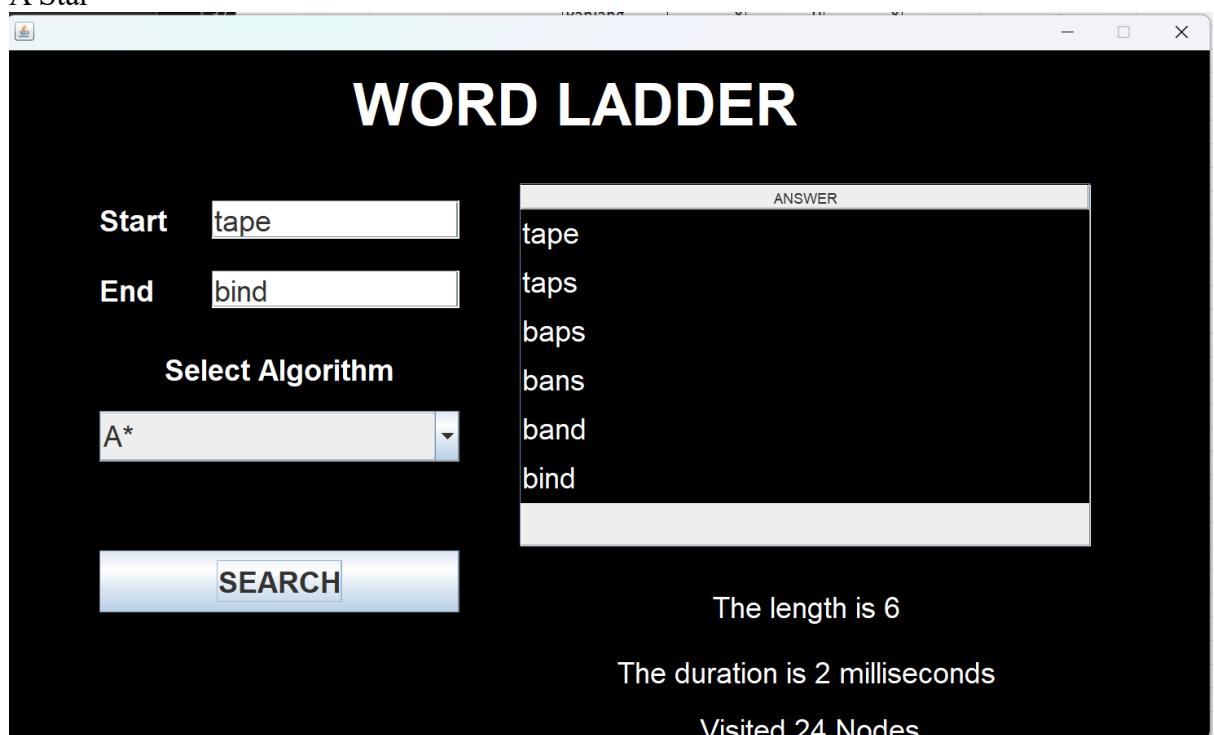
a. UCS



b. GBFS

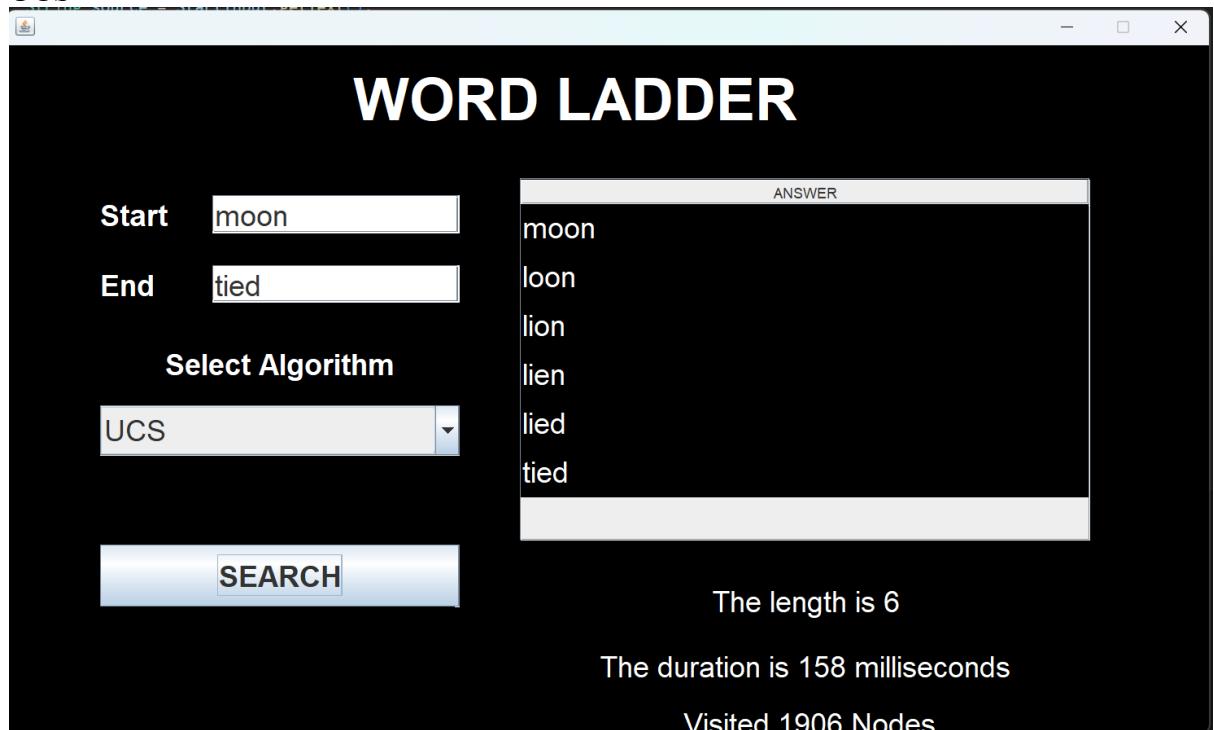


c. A Star

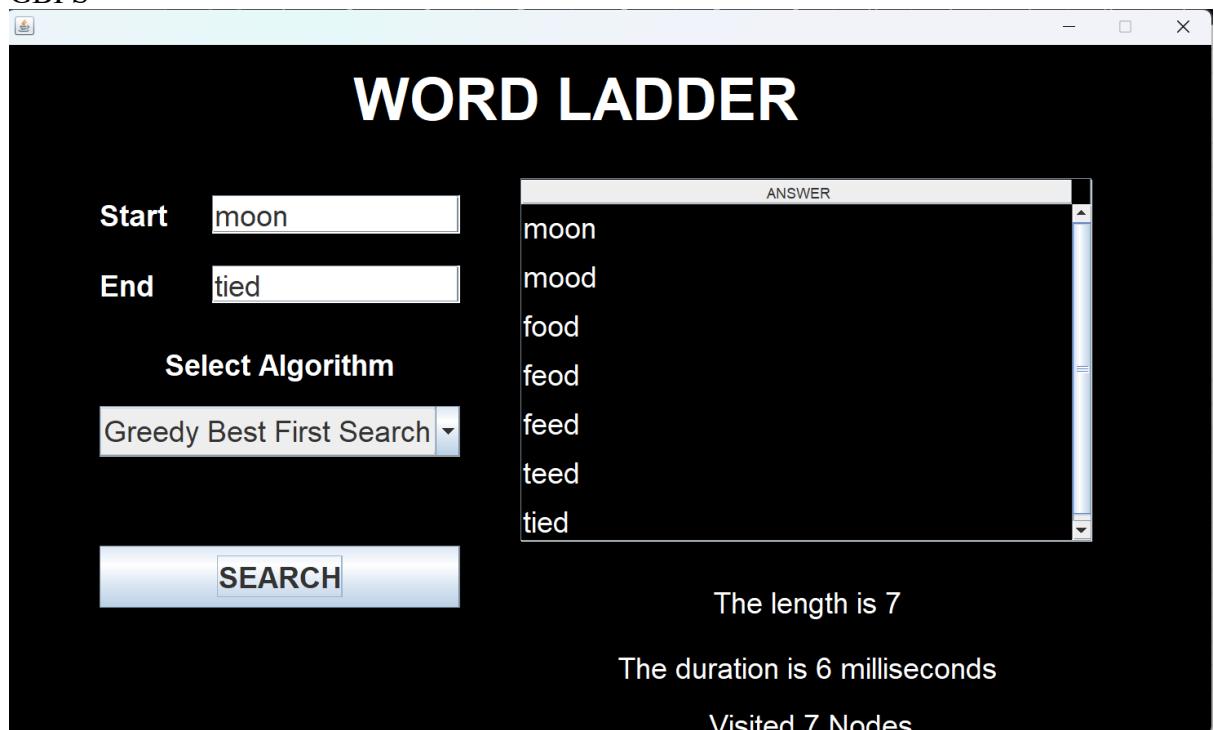


18. Moon – Tied

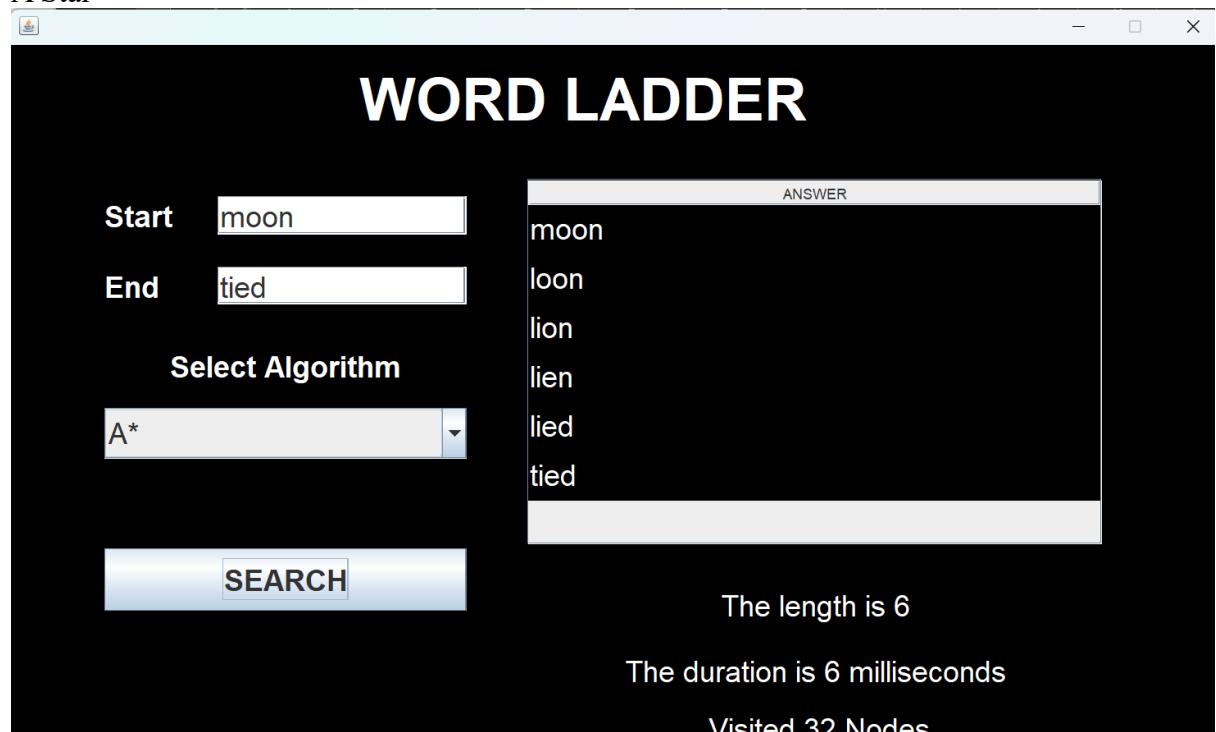
a. UCS



b. GBFS



c. A Star



BAGIAN IV

ANALISIS PERBANDINGAN

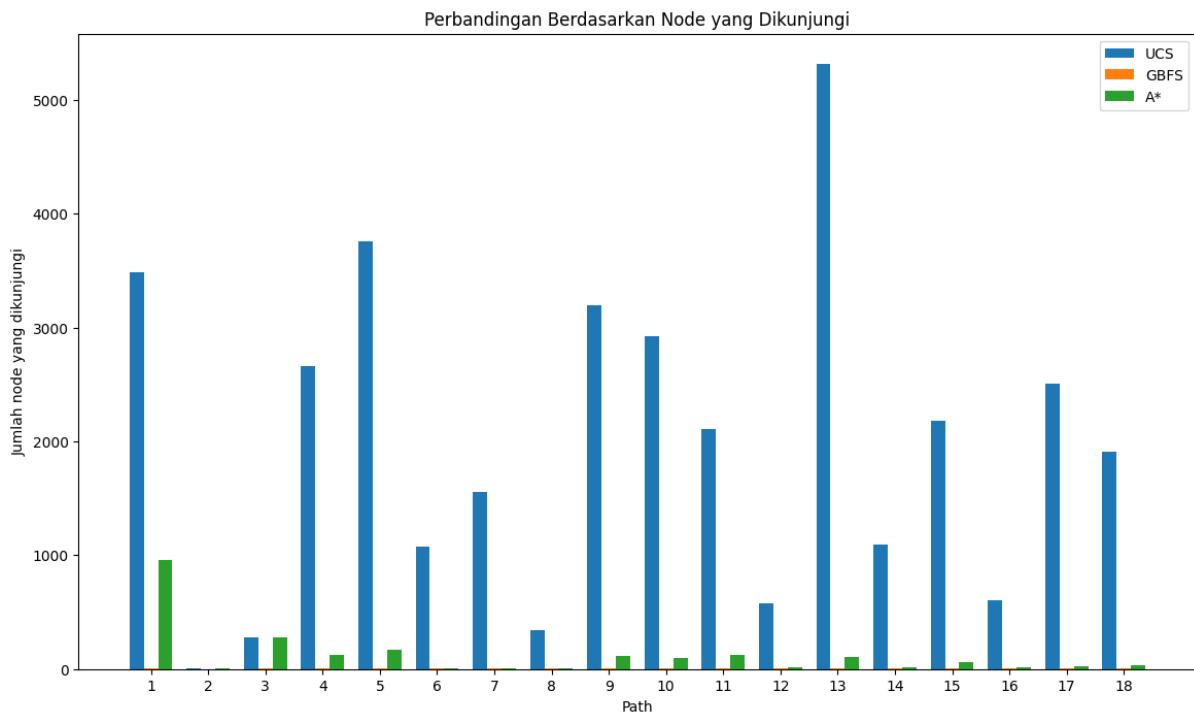
No	Path		Atribut	Algoritma		
	Start	End		UCS	GBFS	A*
1	baby	crib	Panjang	9	0	9
			Durasi(ms)	329	0	46
			Kunjungan	3487	3	957
2	unwarmed	upboiled	Panjang	0	0	0
			Durasi(ms)	15	0	0
			Kunjungan	5	2	5
3	frown	smile	Panjang	10	0	10
			Durasi(ms)	32	1	31
			Kunjungan	278	7	278
4	nice	mean	Panjang	7	0	7
			Durasi(ms)	243	0	15
			Kunjungan	2658	4	126
5	grain	wheat	Panjang	10	0	10
			Durasi(ms)	471	0	15
			Kunjungan	3762	4	172
6	feed	king	Panjang	5	5	5
			Durasi(ms)	78	1	15
			Kunjungan	1075	5	5
7	lame	duck	Panjang	5	5	5
			Durasi(ms)	99	1	17
			Kunjungan	1559	5	8
8	frog	pond	Panjang	5	5	5
			Durasi(ms)	17	0	0
			Kunjungan	344	5	5
9	crazy	party	Panjang	10	10	10
			Durasi(ms)	377	0	15
			Kunjungan	3192	10	114
10	slam	bang	Panjang	7	7	7
			Durasi(ms)	265	2	6
			Kunjungan	2927	7	94
11	trick	treat	Panjang	8	0	8
			Durasi(ms)	214	0	14
			Kunjungan	2111	4	123
12	knack	skill	Panjang	7	0	7
			Durasi(ms)	53	2	3
			Kunjungan	581	6	20
13	waste	trash	Panjang	9	0	9
			Durasi(ms)	767	1	12
			Kunjungan	5316	4	102
14	slog	toil	Panjang	6	11	6
			Durasi(ms)	54	1	1

			Kunjungan	1092	11	20
15	bill	debt	Panjang	6	7	6
			Durasi(ms)	160	2	4
			Kunjungan	2185	7	58
16	ship	yard	Panjang	6	6	6
			Durasi(ms)	26	0	2
			Kunjungan	603	6	19
17	tape	bind	Panjang	6	6	6
			Durasi(ms)	194	0	2
			Kunjungan	2505	6	24
18	moon	tied	Panjang	6	7	6
			Durasi(ms)	158	6	6
			Kunjungan	1906	7	32

Tabel 4.1 Tabel Data Uji Kasus

Tabel 4.1 adalah tabel yang memberikan data terkait panjang path, durasi eksekusi, dan jumlah kata yang dikunjungi oleh algoritma UCS, Greedy Best First Search, dan A Star dari uji kasus yang telah dilakukan pada Bab III. Untuk mempermudah analisis, dilakukan analisis berdasarkan tiga kategori, yang terdiri dari masing-masing atribut.

a. Berdasarkan Jumlah Kata yang Dikunjungi

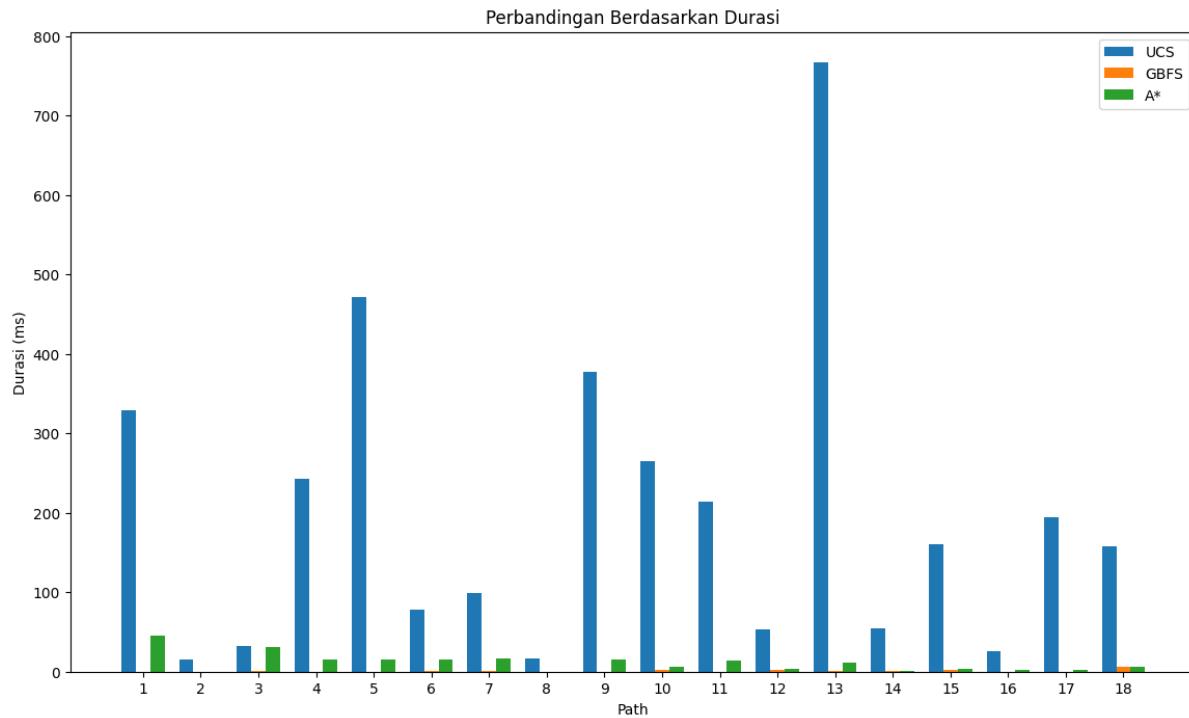


Gambar 4.1 Gambar grafik perbandingan berdasarkan node yang dikunjungi

Berdasarkan gambar 4.1, dapat dilihat bahwa algoritma UCS(Uniform Cost Search) cenderung mengunjungi kata yang lebih banyak, diikuti oleh algoritma A Star, dan algoritma GBFS(Greedy Best First Search). Hal ini disebabkan karena biaya/f(n) yang digunakan pada permainan Word Ladder ini adalah biaya sebelumnya dijumlah 1. Ini memiliki kesamaan dengan algoritma Breadth First Search yang akan menelusuri kata dengan biaya terkecil terlebih dahulu secara konsisten sehingga harus memeriksa banyak

kemungkinan untuk mencapai tujuan. Kemudian, dapat dilihat bahwa algoritma A Star mengunjungi lebih banyak kata dibanding algoritma Greedy Breadth First Search, tetapi lebih sedikit dibanding algoritma UCS. Algoritma GBFS mengunjungi kata paling sedikit karena algoritma ini langsung mencari nilai heuristic terkecil dan bisa berhenti dengan dua kasus, yaitu jika sudah bertemu dengan kata tujuan atau mengalami siklus pada saat proses algoritma berjalan.

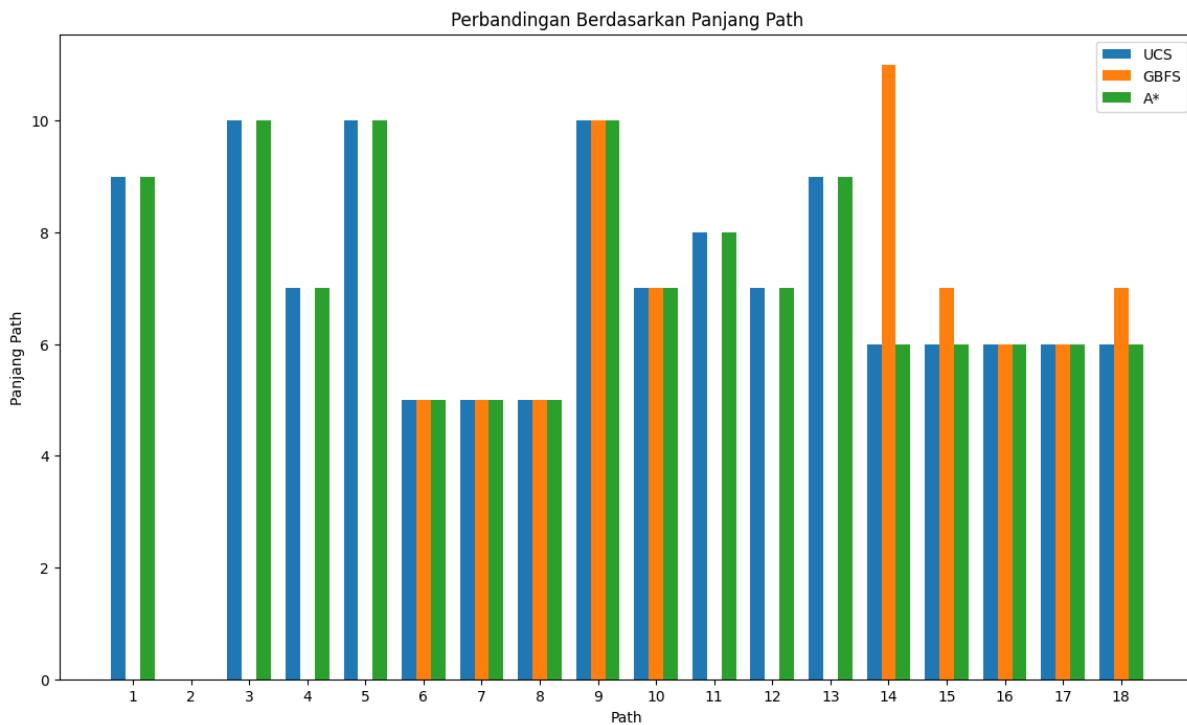
b. Berdasarkan Durasi Eksekusi



Gambar 4.2 Gambar grafik perbandingan berdasarkan durasi

Berdasarkan gambar 4.2, dapat dilihat bahwa algoritma UCS(Uniform Cost Search) memiliki durasi eksekusi yang lebih lama, diikuti dengan algoritma A Star, dan algoritma GBFS(Greedy Best First Search). Hal ini dapat terjadi karena algoritma UCS akan menelusuri kata terdekat terlebih dahulu sehingga kata yang harus dikunjungi cukup banyak dan berakibat pada durasi eksekusi yang lama. Algoritma A Star sebenarnya juga melakukan hal yang sama dengan algoritma UCS, tetapi terdapat nilai heuristic yang dapat membantu algoritma A Star untuk memilih kata mana yang harus diproses terlebih dahulu. Untuk algoritma GBFS, akan selalu dipilih satu kata yang memiliki nilai heuristic terkecil sehingga dapat dengan cepat mencapai solusi atau justru terjebak pada siklus dan tidak menemui solusi.

c. Berdasarkan Panjang Path



Gambar 4.3 Gambar grafik perbandingan berdasarkan panjang path

Berdasarkan gambar 4.3, dapat dilihat bahwa algoritma GBFS(Greedy Best First Search) memiliki jumlah path yang lebih panjang dibanding kedua algoritma lainnya atau bahkan bisa tidak menemui path untuk mencapai tujuan. Hal ini disebabkan karena algoritma GBFS hanya memilih kata selanjutnya yang memiliki nilai heuristik terkecil dan tidak bisa melakukan backtrack untuk kembali ke pilihan sebelumnya. Ini bisa mengakibatkan algoritma ini terjebak pada minimum lokal dan terjebak pada siklus sehingga tidak dapat menemui tujuan yang diinginkan. Sementara itu, algoritma UCS(Uniform Cost Search) dan algoritma A Star akan menemui path selalu terdapat kata selanjutnya yang dapat dibangkitkan. Selain itu, algoritma UCS dan A Star akan selalu memberikan hasil optimal. Algoritma UCS menelusuri kata terdekat terlebih dahulu sehingga akan selalu memberikan hasil yang optimal(jarak terpendek), apalagi jarak antara satu node dengan satu tetangga memiliki jarak yang sama dengan tetangga yang lain. Algoritma A Star juga melakukan hal yang dilakukan oleh algoritma UCS, tetapi ditambah dengan nilai heuristic sehingga dapat mempercepat proses pencarian. Oleh karena itu, algoritma UCS dan algoritma A Star akan selalu menemui hasil yang optimal.

Kesimpulan

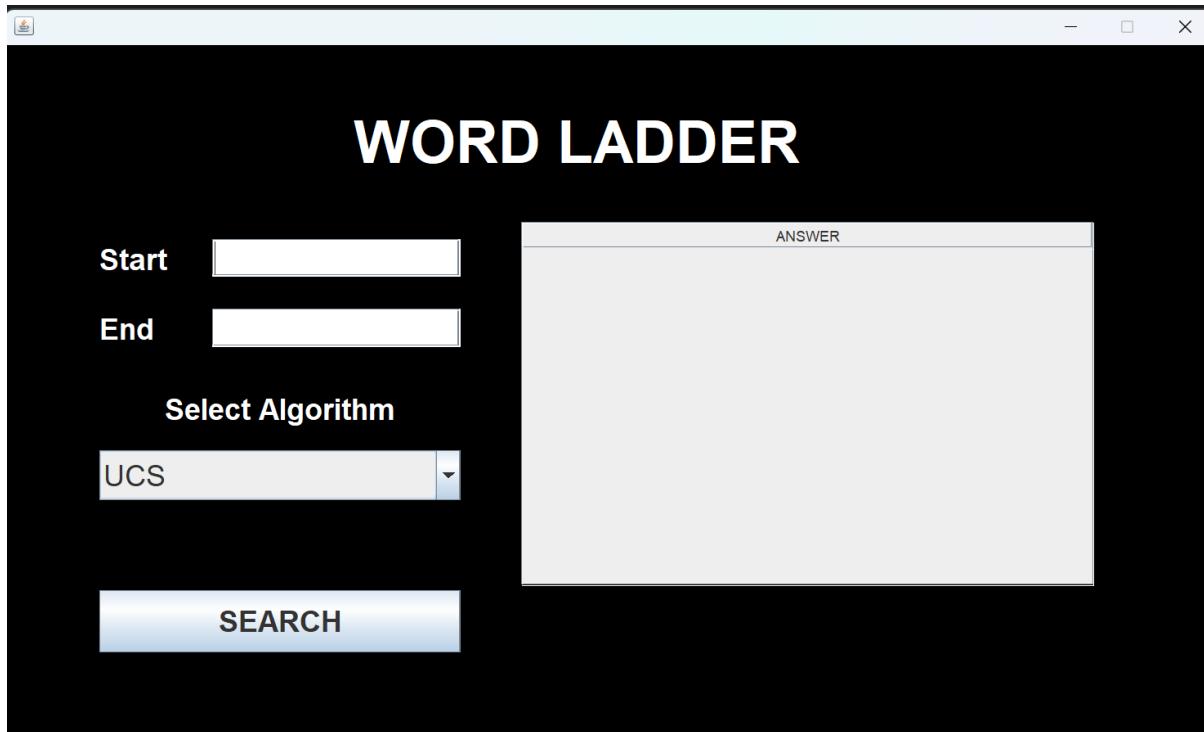
Algoritma UCS(Uniform Cost Search) memiliki durasi eksekusi yang cukup lama karena harus menelusuri kata yang cukup banyak. Namun, algoritma ini pasti akan memberikan hasil yang optimal.

Algoritma GBFS(Greedy Best First Search) memiliki durasi eksekusi yang singkat karena tidak perlu menelusuri banyak kata, hanya kata yang memiliki nilai heuristik terkecil. Namun, algoritma ini bisa saja tidak memberikan hasil yang optimal atau bahkan tidak menemukan solusi.

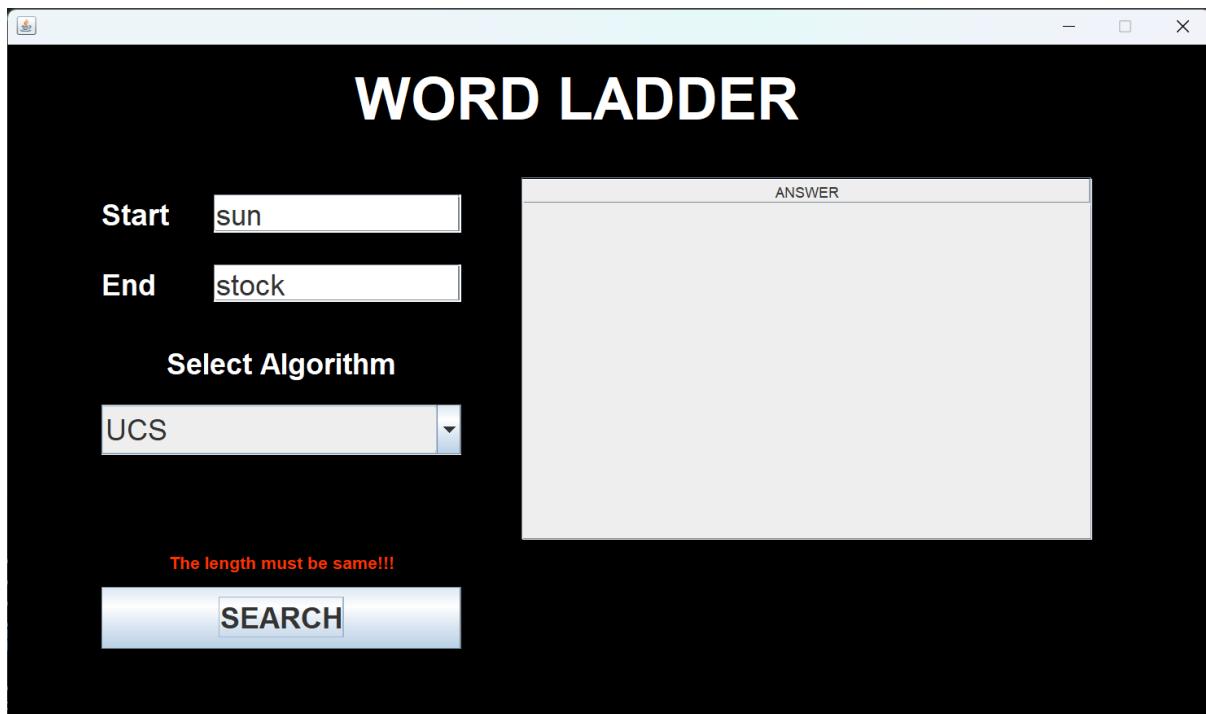
Algoritma A Star memiliki durasi eksekusi yang lebih singkat dibanding UCS karena penelusuran kata dibantu dengan nilai heuristik untuk memprioritaskan kata mana yang harus ditelusuri terlebih dahulu. Oleh karena itu, algoritma ini akan menghasilkan hasil yang optimal dengan durasi yang relatif singkat dibanding algoritma UCS.

BAGIAN V

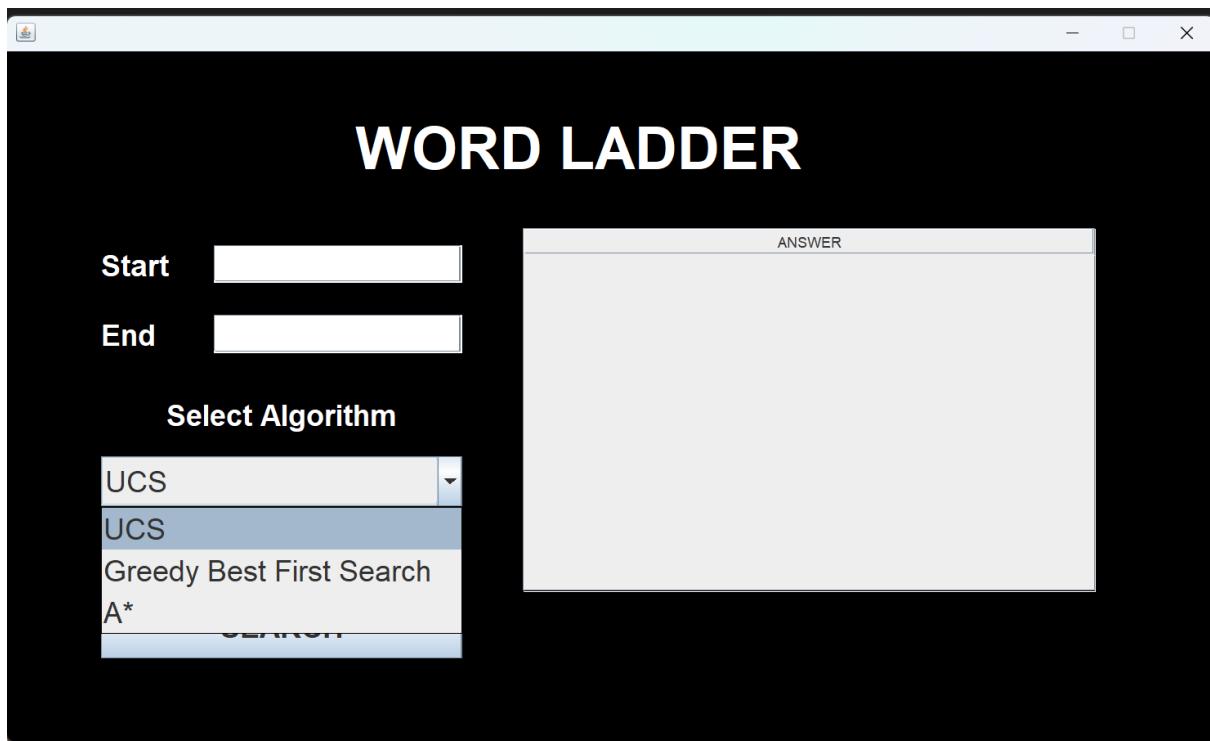
IMPLEMENTASI GUI(*GRAPHICAL USER INTERFACE*)



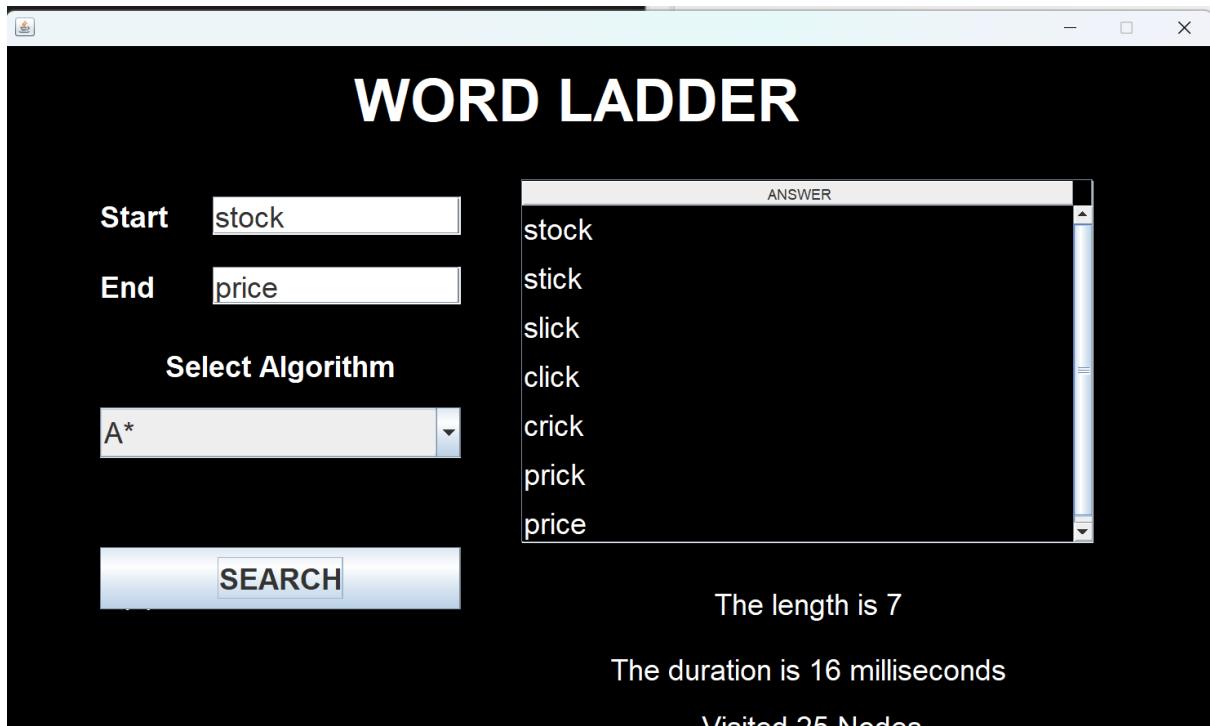
Pada proyek ini, GUI yang dibuat menggunakan kelas Java bernama Java Swing. Aplikasi Word Ladder adalah sebuah program yang memungkinkan pengguna untuk menjelajahi serangkaian kata yang berbeda satu per satu, dengan setiap kata dalam serangkaian hanya berbeda satu huruf dari kata sebelumnya. Aplikasi ini dilengkapi dengan antarmuka pengguna yang memudahkan pengguna untuk melakukan pencarian Word Ladder dengan cepat dan efisien. Saat pertama kali aplikasi dibuka, judul besar "WORD LADDER", tempat untuk melakukan masukan kata awal dan kata akhir, pemilihan algoritma, dan tombol Search terpampang dengan jelas pada layar. Pengguna kemudian diminta untuk memasukkan kata awal dan akhir ke dalam kolom yang sesuai.



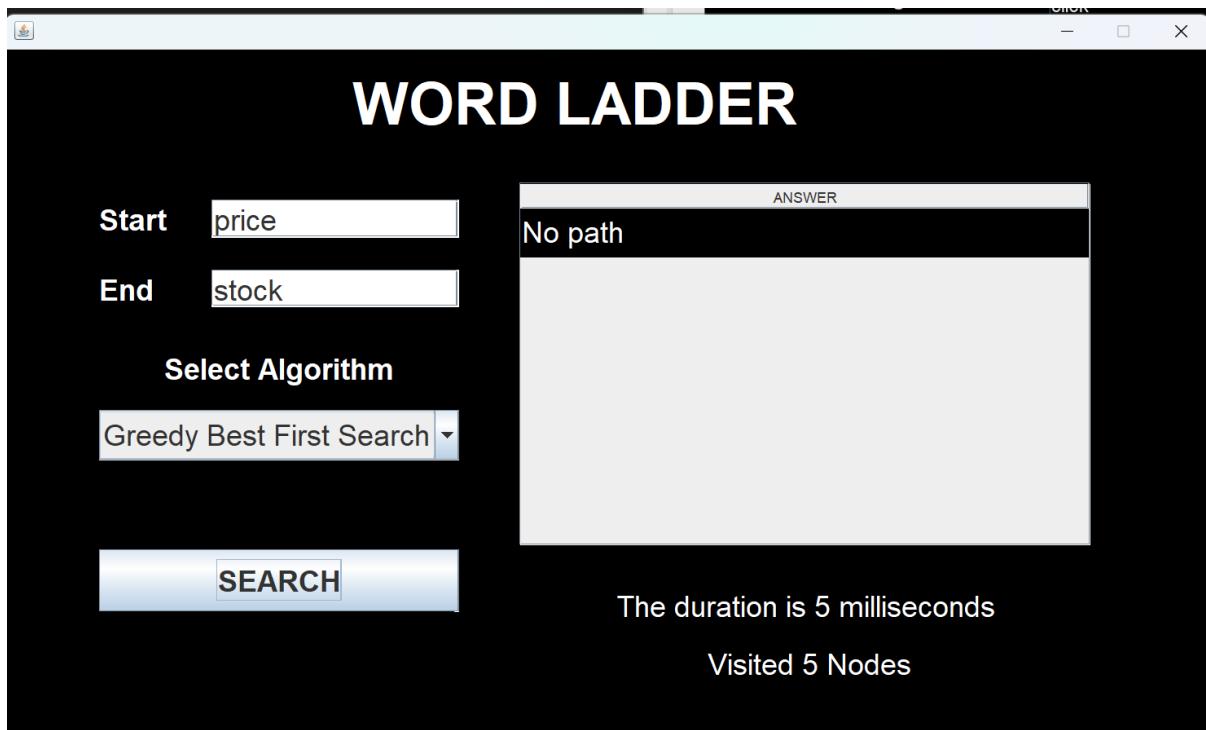
Pesan kesalahan yang jelas dan informatif akan muncul di antarmuka pengguna untuk memberi tahu pengguna tentang masalah yang terjadi. Misalnya, ketika pengguna tidak memasukkan kata awal atau akhir, atau keduanya, dan mencoba untuk memulai pencarian, pesan kesalahan akan muncul meminta mereka untuk memasukkan kedua kata tersebut sebelum melanjutkan. Selain itu, jika kata awal atau akhir yang dimasukkan tidak ada dalam daftar kata yang tersedia, pengguna akan diberi tahu bahwa salah satu atau kedua kata tersebut tidak valid dan harus dipertimbangkan kembali. Tidak hanya itu, jika panjang kata awal dan akhir tidak sama, itu tidak akan mungkin membentuk Word Ladder antara dua kata tersebut.



Ada juga pilihan untuk memilih algoritma pencarian yang diinginkan, seperti Uniform Cost Search (UCS), Greedy Best First Search, atau A*. Ini memberikan pengguna fleksibilitas dalam memilih pendekatan pencarian yang paling sesuai dengan kebutuhan.



Setelah memasukkan kata awal, kata akhir, dan memilih algoritma pencarian, pengguna cukup mengeklik tombol "SEARCH" untuk memulai proses pencarian. Setelah pencarian selesai, hasilnya ditampilkan dalam bentuk tabel. Jika jalur dari kata awal ke kata akhir ditemukan, kata-kata dalam jalur akan ditampilkan dalam tabel dengan urutan yang sesuai.



Namun, jika tidak ada jalur yang ditemukan, tabel akan menampilkan pesan yang sesuai. Selain itu, pengguna juga diberi informasi tambahan tentang pencarian, seperti durasi pencarian, jumlah node yang dikunjungi, dan panjang jalur. Hal ini membantu pengguna untuk memahami kinerja pencarian dan memberikan gambaran yang lebih jelas tentang hasilnya.

BAGIAN VI
REPOSITORY GITHUB

https://github.com/FrancescoMichael/Tucil1_13522038.git

BAGIAN VII
CHECKLIST

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	✓	