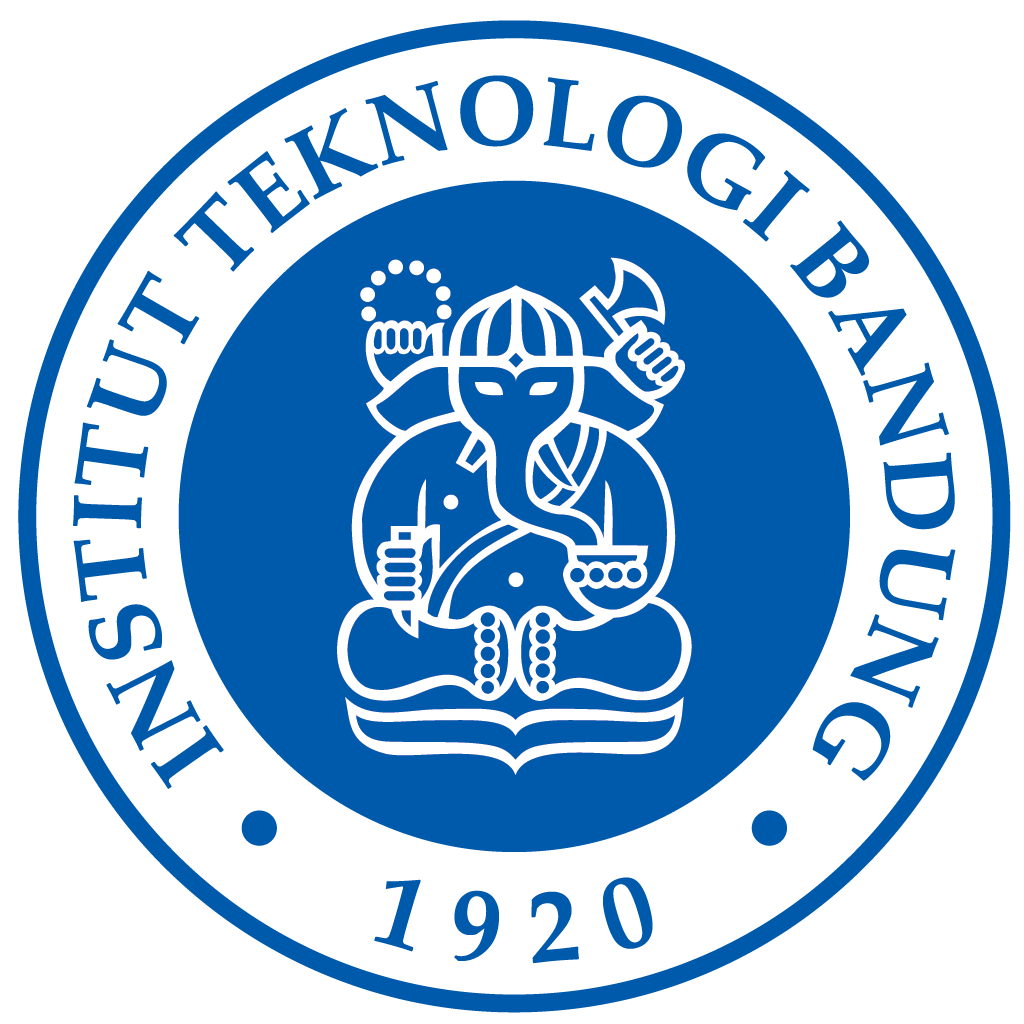
**LAPORAN TUGAS KECIL III**

**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**

Analisis dan implementasi dalam algoritma UCS, Greedy Best First Search, dan A\* (berisi deskripsi langkah-langkahnya, bukan notasi pseudocode). Analisis juga perlu memuat jawaban dari pertanyaan-pertanyaan berikut. ● Definisi dari f(n) dan g(n), sesuai dengan salindia kuliah. ● Apakah heuristik yang digunakan pada algoritma A\* admissible? Jelaskan sesuai definisi admissible dari salindia kuliah. ● Pada kasus word ladder, apakah algoritma UCS sama dengan BFS? (dalam artian urutan node yang dibangkitkan dan path yang dihasilkan sama) ● Secara teoritis, apakah algoritma A\* lebih efisien dibandingkan dengan algoritma UCS pada kasus word ladder? ● Secara teoritis, apakah algoritma Greedy Best First Search menjamin solusi optimal untuk persoalan word ladder

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 sebelumya.
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 heursitik 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, sedangkah 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.

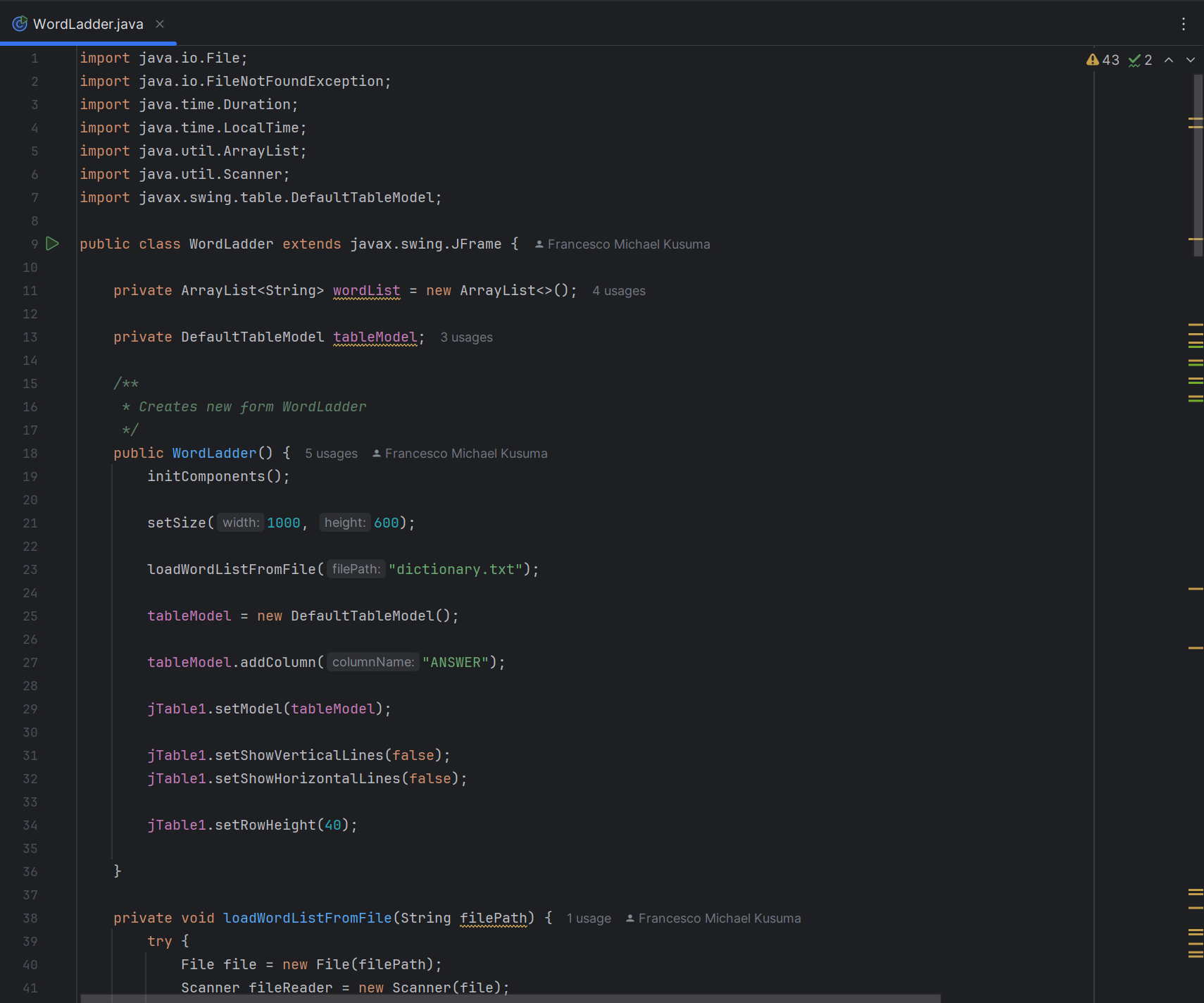
Secara teoritis, algoritma A Star akan memberikan hasil yang lebih efisien dibanding dengan algoritma UCS. Ini dapat terjadi karena sebenarnya algoritma A Star melakukan hal yang sama dengan algoritma UCS, tetapi pemilihan node yang sama-sama dekat didasarkan pada nilai heuristik yang terkecil sehingga dapat meningkatkan efisiensi proses pencarian. Namun, meskipun A Star dapat memberikan hasil yang lebih eifsien, 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 ooptimal dan efisien.

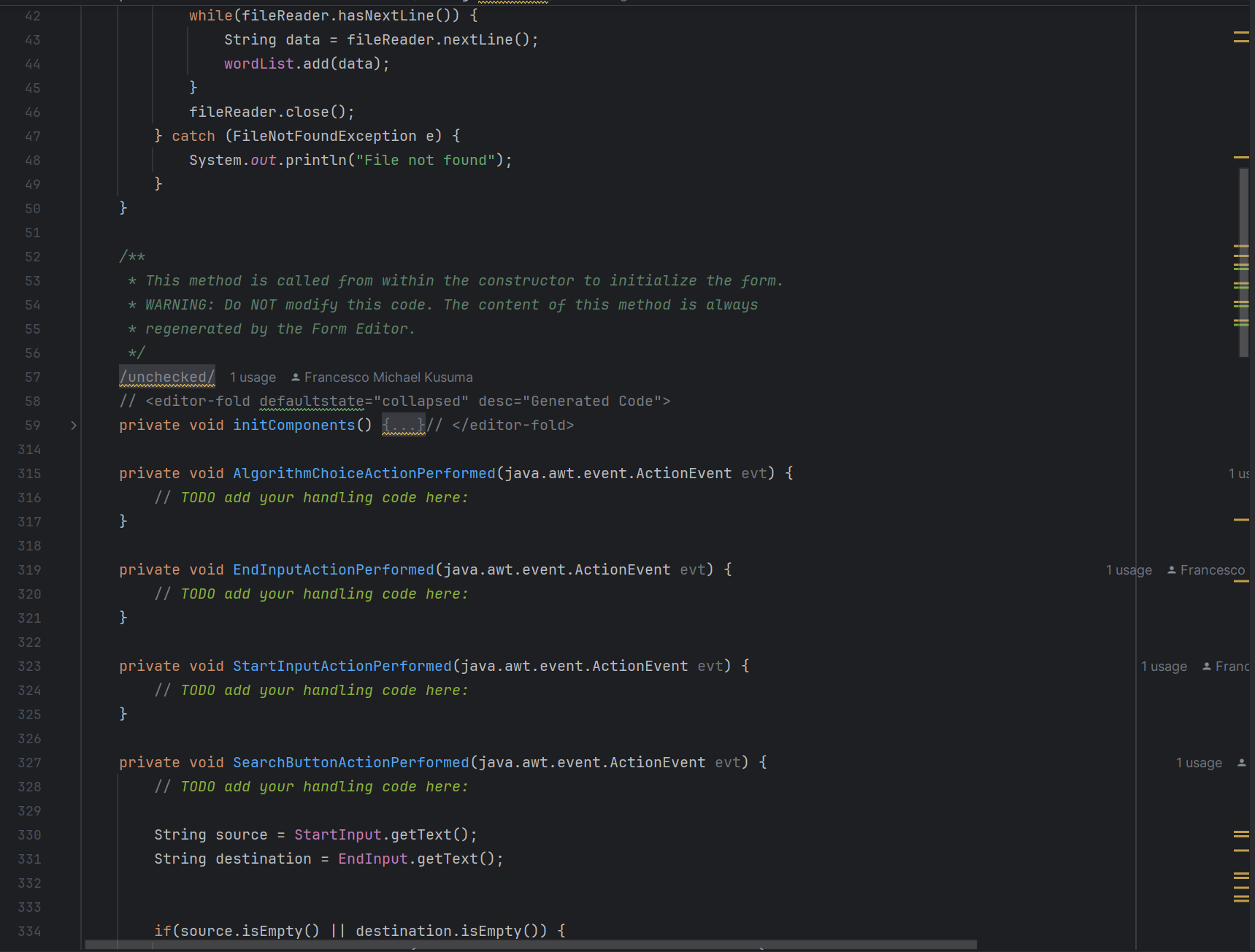
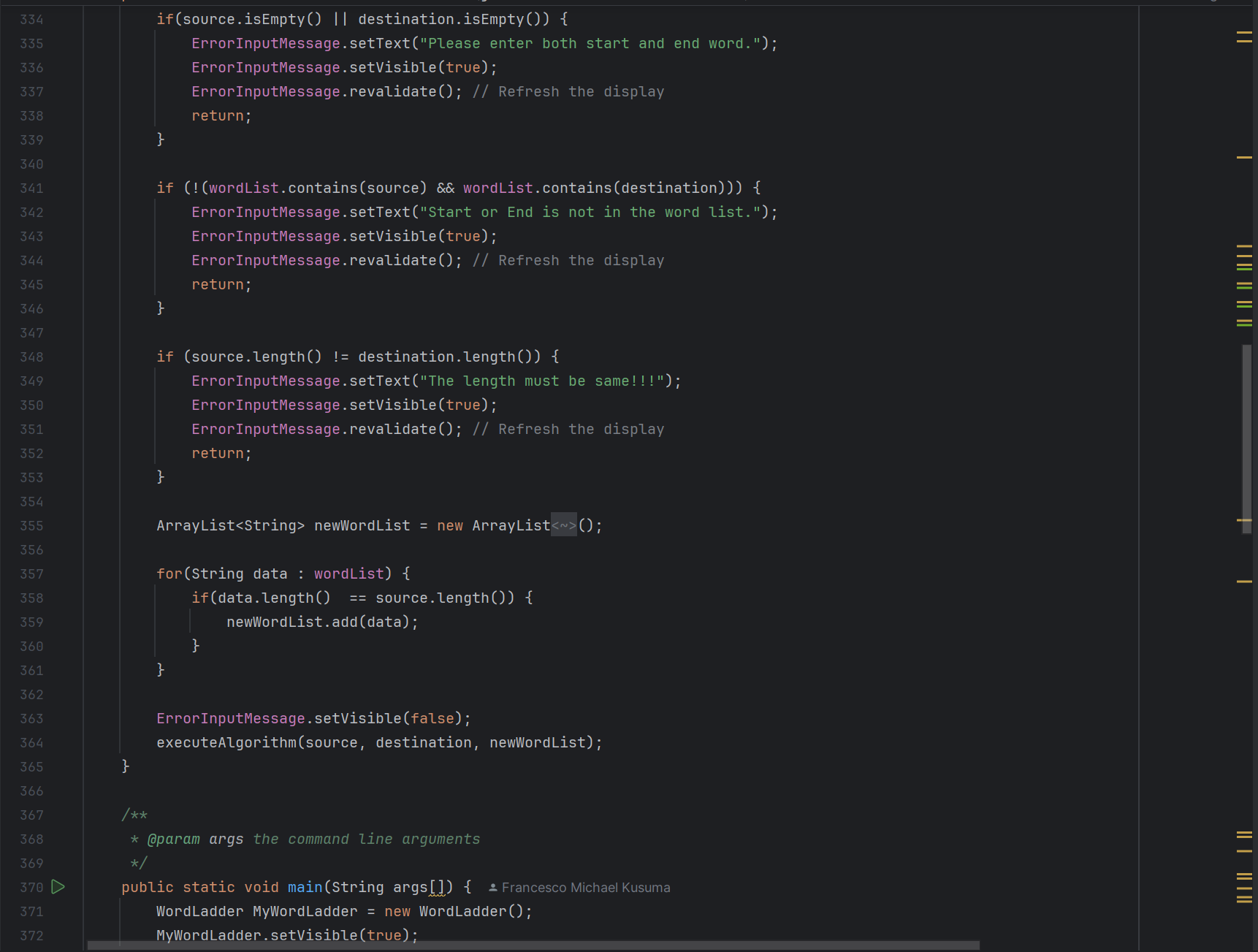
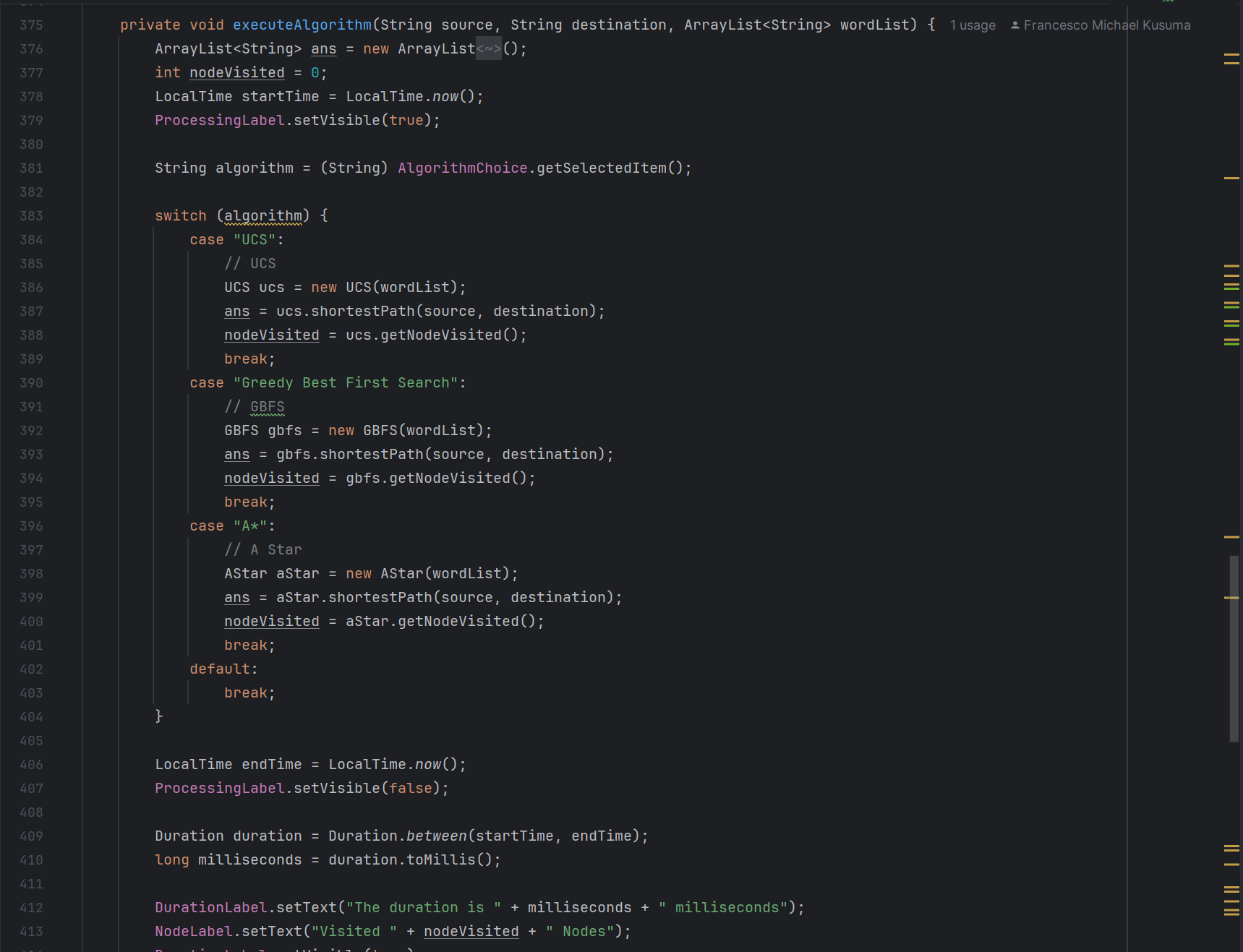
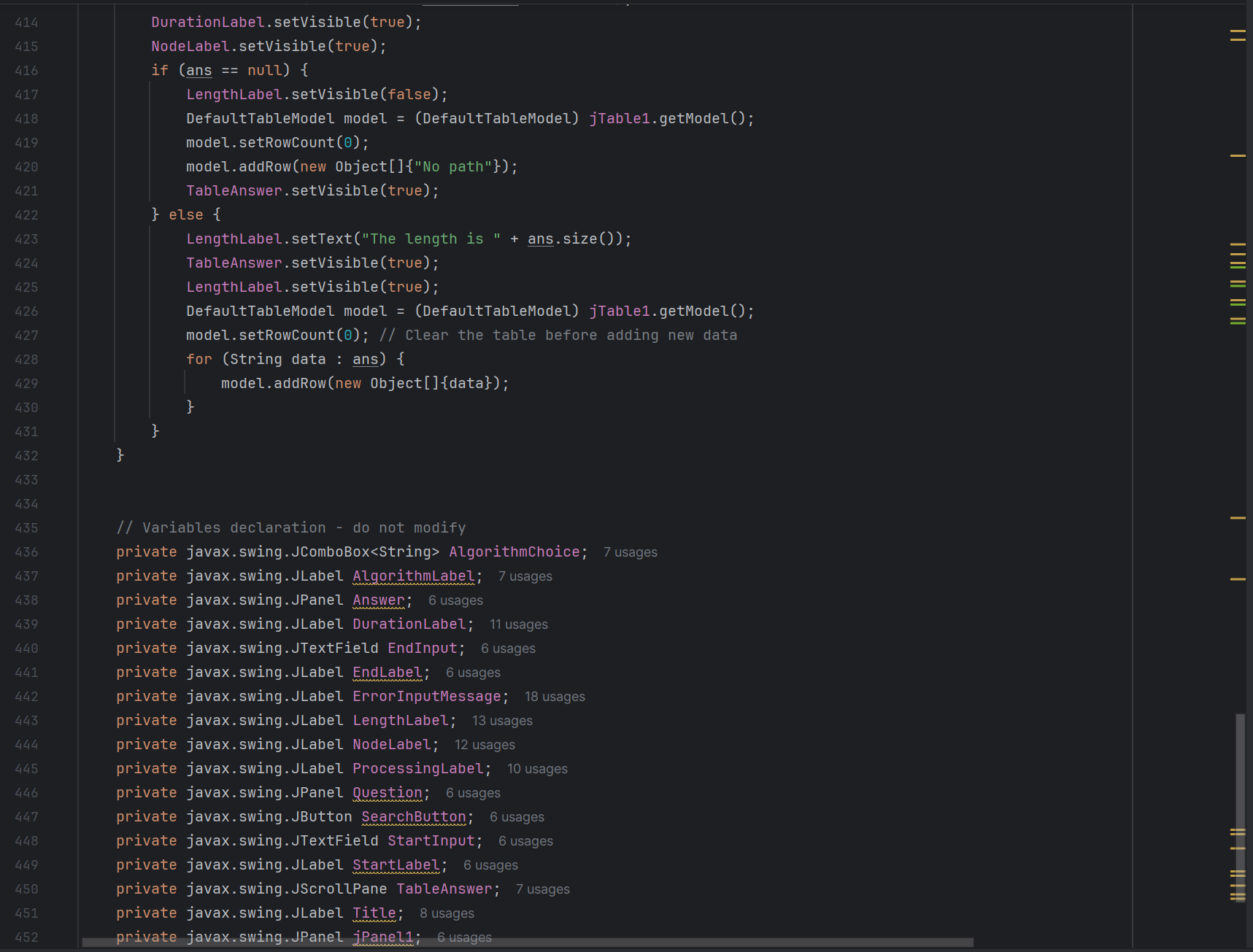
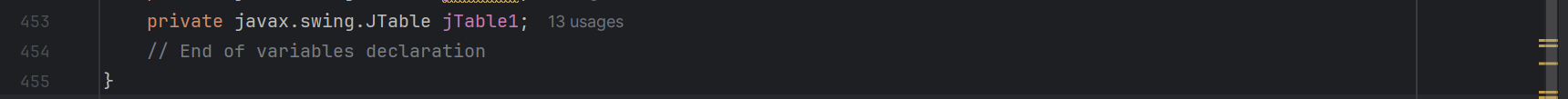
Secara teoritis, algoritma Greedy Best First Search tidak menjamin solusi yang optimal untuk persoalan Word Ladder. Algoritma GBFS mungkin menemukan solusi yang cepat dalam banyak kasus, tetapi tidak menjamin solusi yang optimal. Ini karena GBFS mungkin terjebak mengeksplorasi jalur yang mungkin tampak 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**

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

1. **loadWordListFromFile**

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

1. **initComponents**

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

1. **SearchButtonActionPerformed**

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

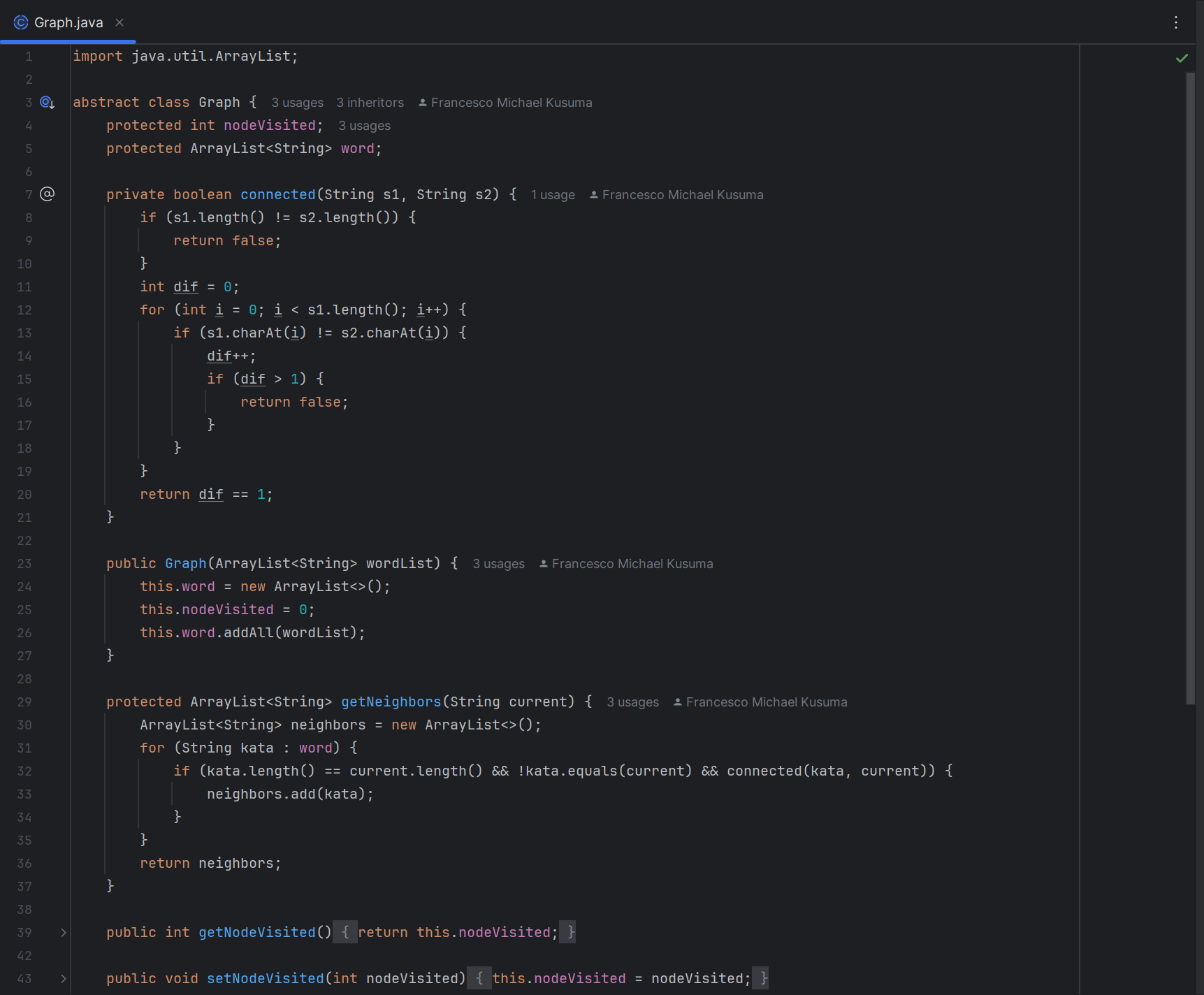
1. **main**

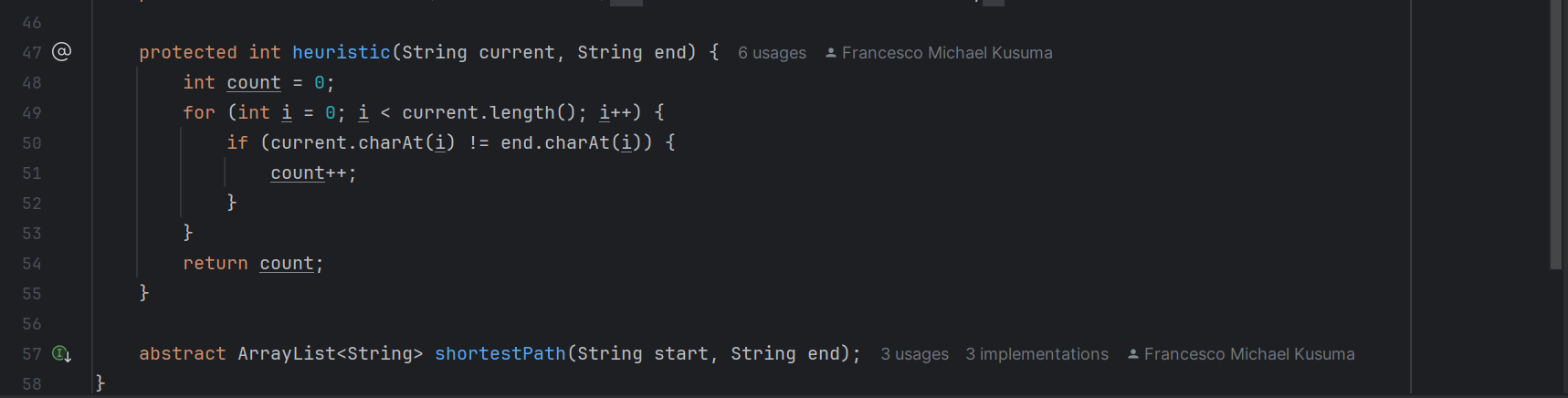
Method ini berfungsi untuk memulai program berjalan.

1. **executeAlgorithm**

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

1. **Graph**





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 ditelusui. 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

1. **connected**

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

1. **getNeighbors**

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

1. **getNodeVisited**

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

1. **setNodeVisited**

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

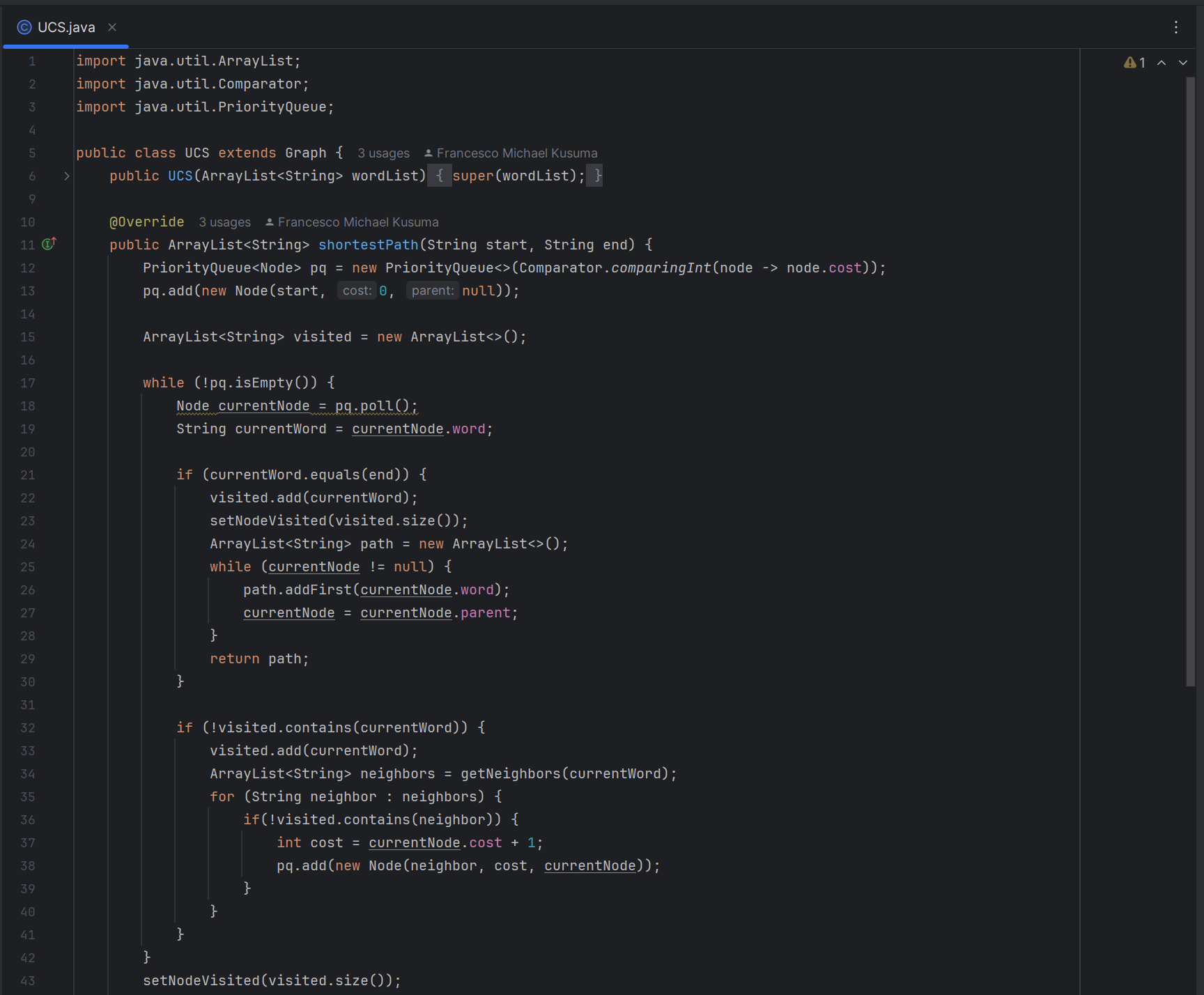
1. **heuristic**

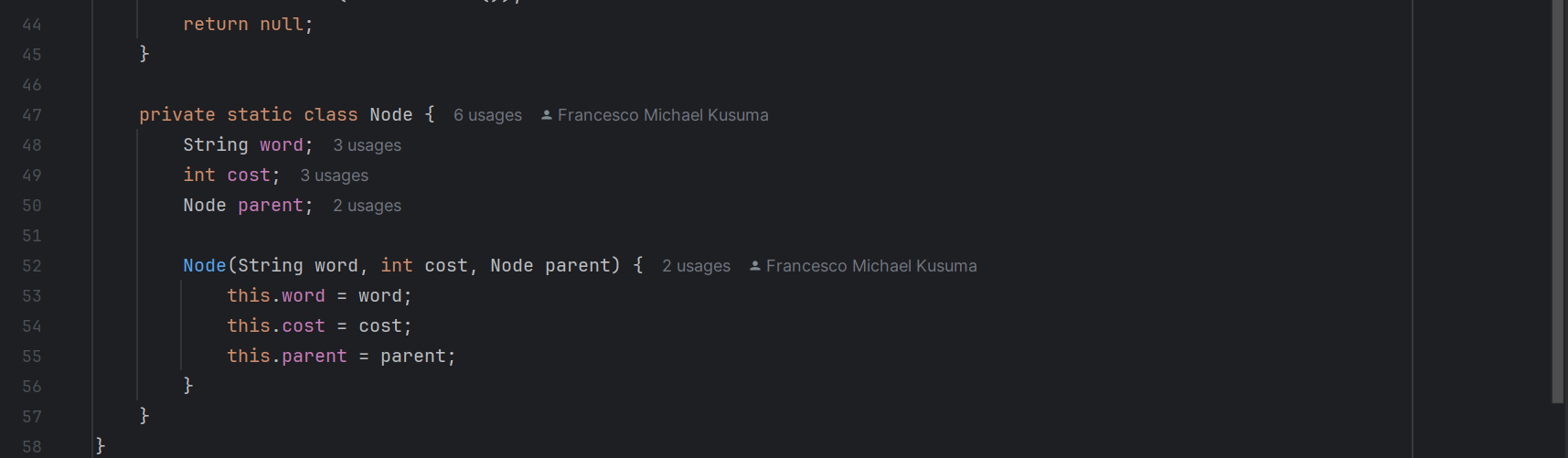
Method ini berfungsi untuk mencari nilai heuristic/h(n) dari suatu kata.

1. **shortestPath**

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

1. **UCS**





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

1. **word**

word akan menyimpan kata yang menjadi node

1. **cost**

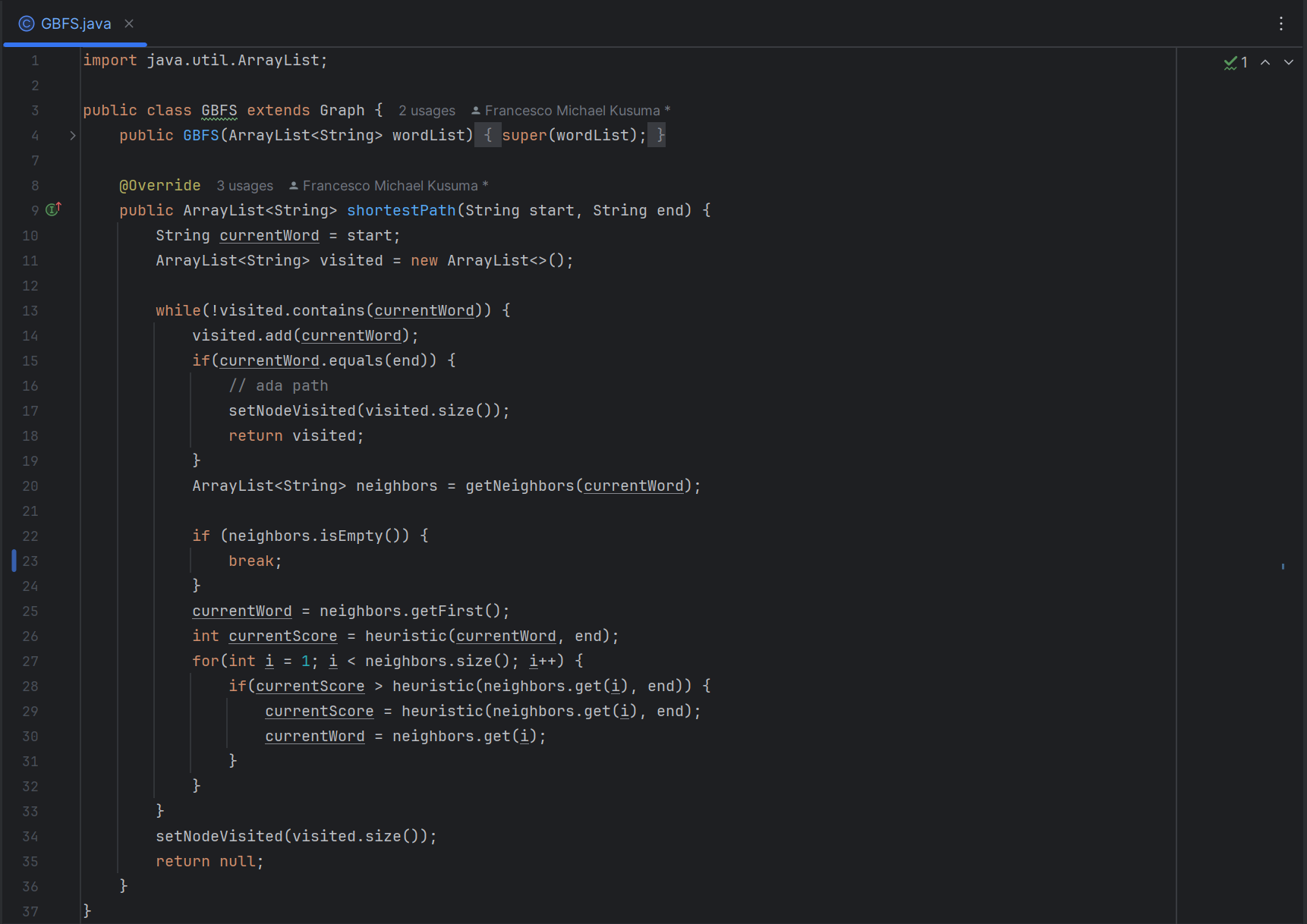
cost akan menyimpan nilai g(n) dari tiap node

1. **parent**

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

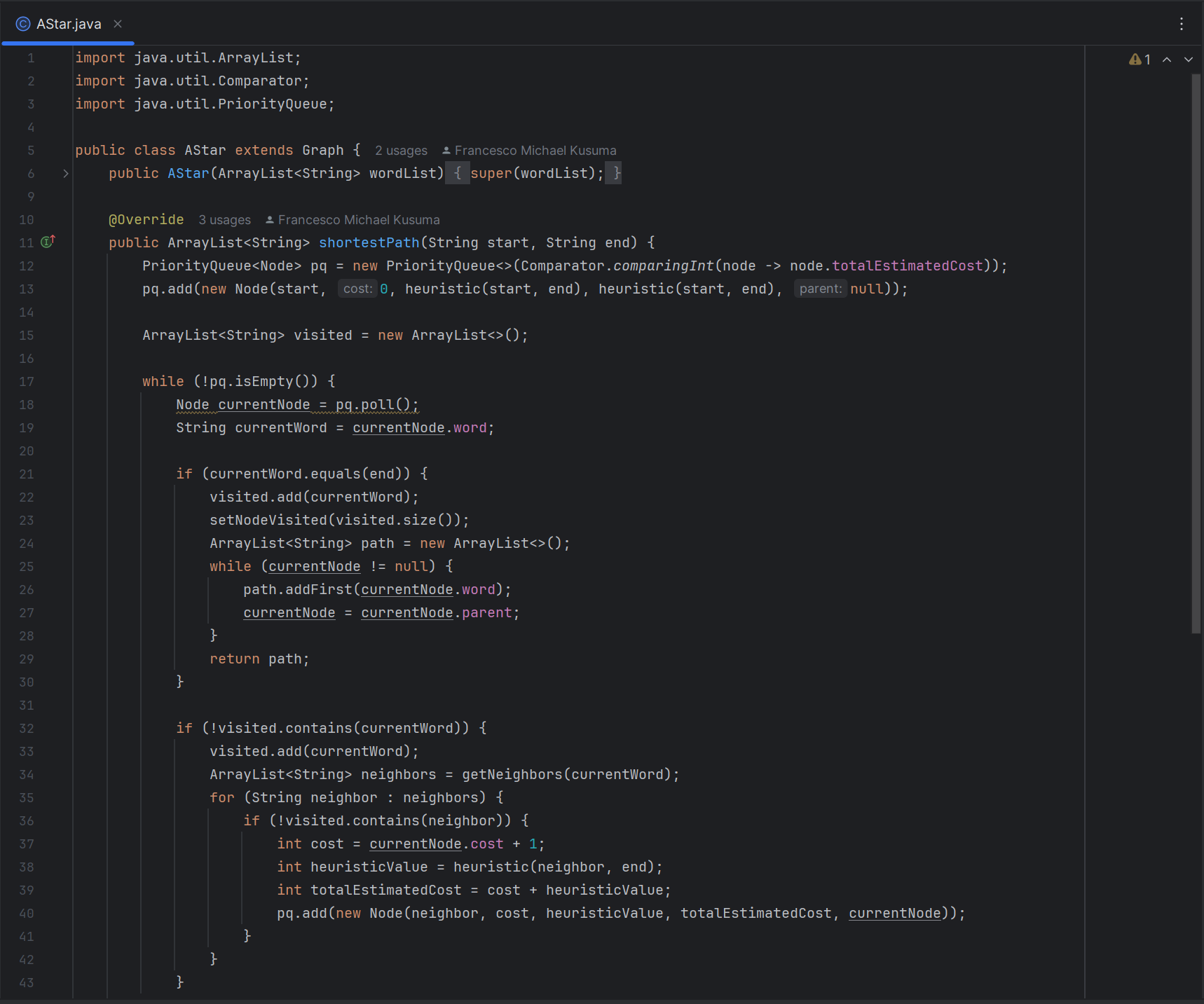
Kelas ini memiliki 1 method yang mengimplementasikan method shortestPath. Method ini menjalankan algoritma pencarian rute menggunakan algoritma UCS(Uniform Cost Search).

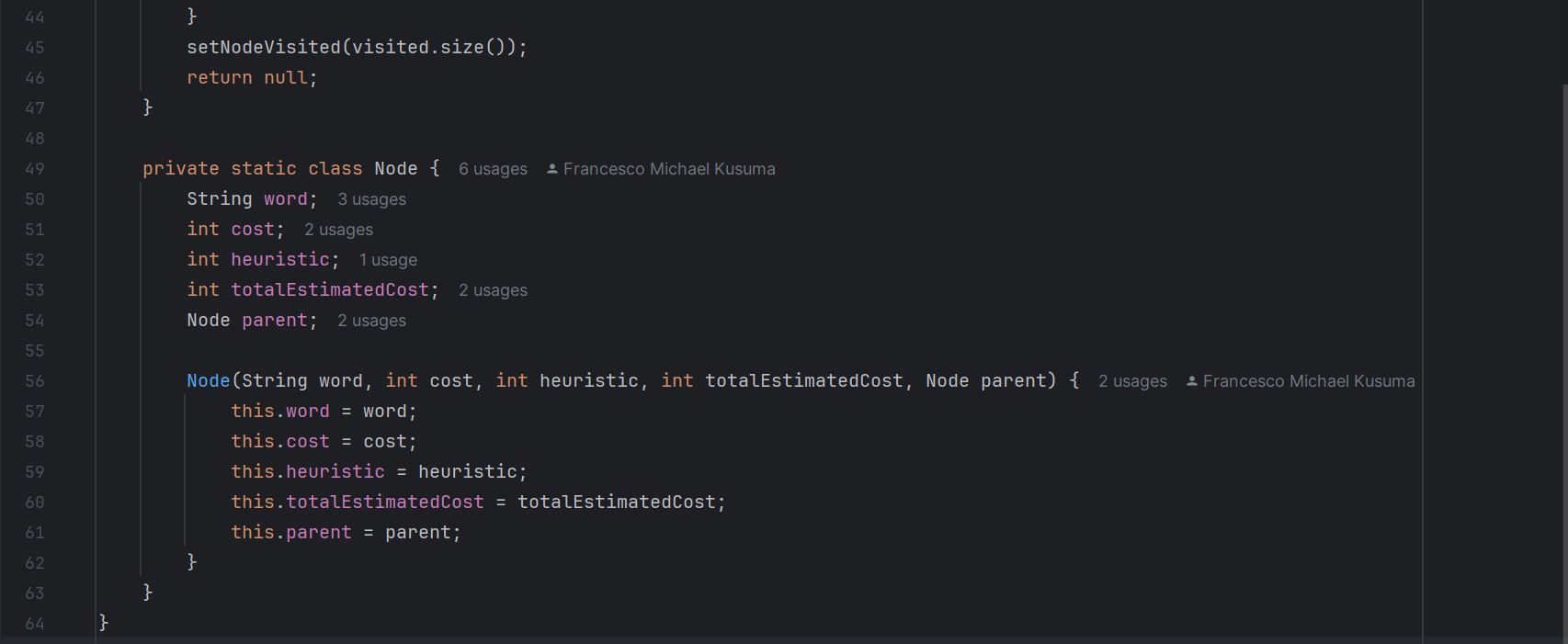
1. **GBFS**



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.

1. **Astar**





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

1. **word**

word akan menyimpan kata yang menjadi node

1. **cost**

cost akan menyimpan nilai g(n) dari tiap node

1. **heuristic**

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

1. **totalEstimatedCost**

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

1. **parent**

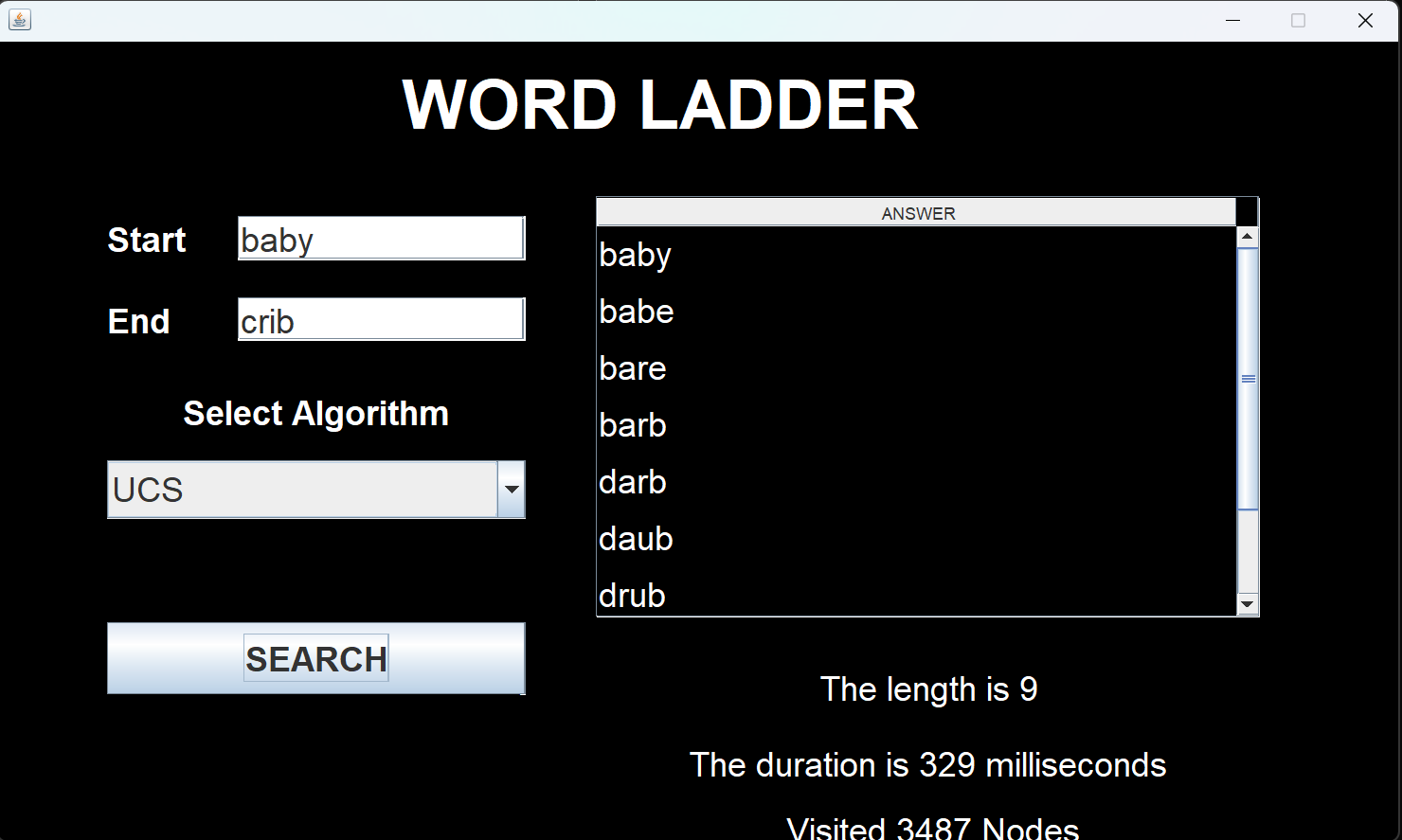
parent akan menyimpan node parent untuk melakukan pelacakan saat suatu path ditemukan

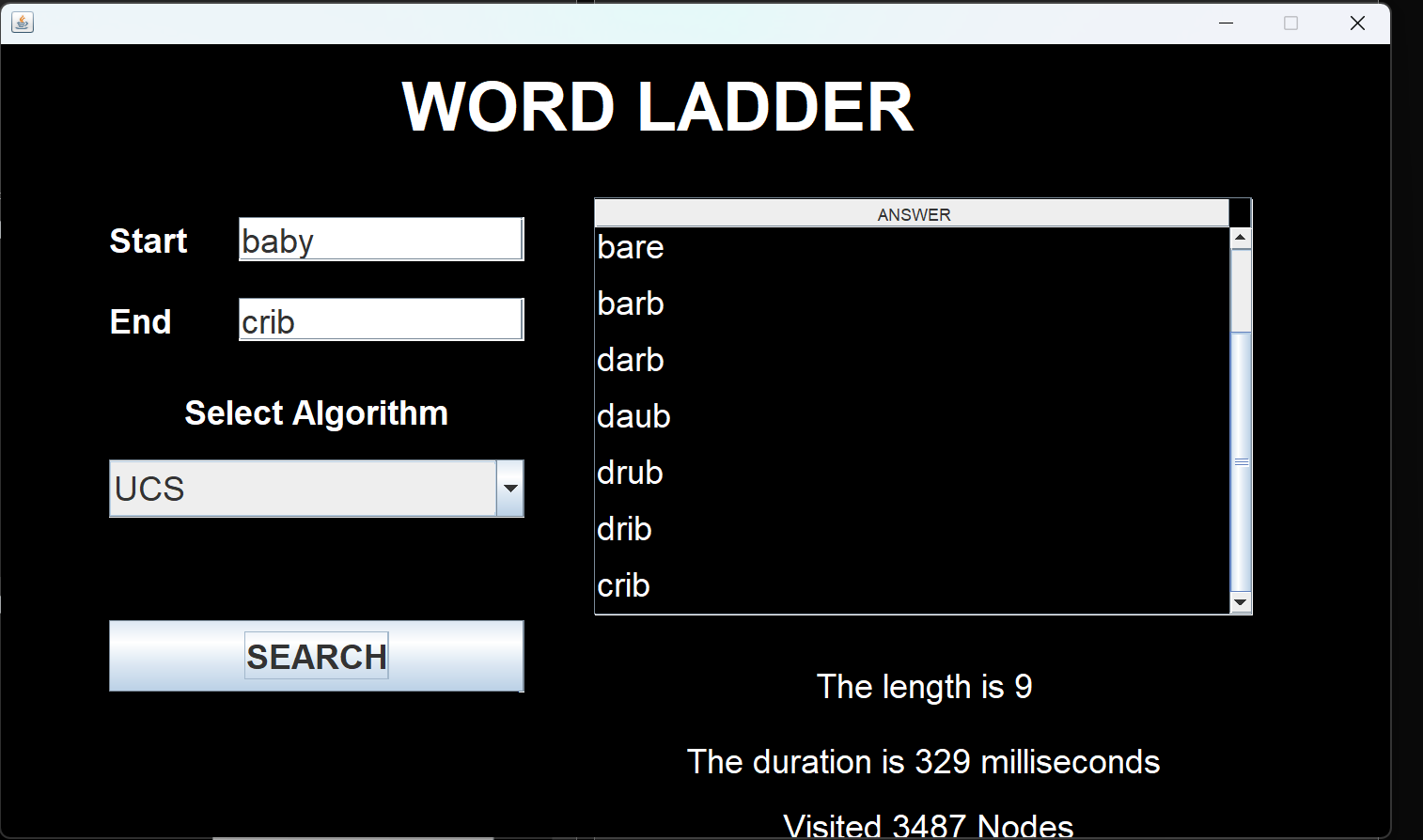
Kelas ini memiliki 1 method yang mengimplementasikan method shortestPath. Method ini menjalankan algoritma pencarian rute menggunakan algoritma A Star.

**BAGIAN III**

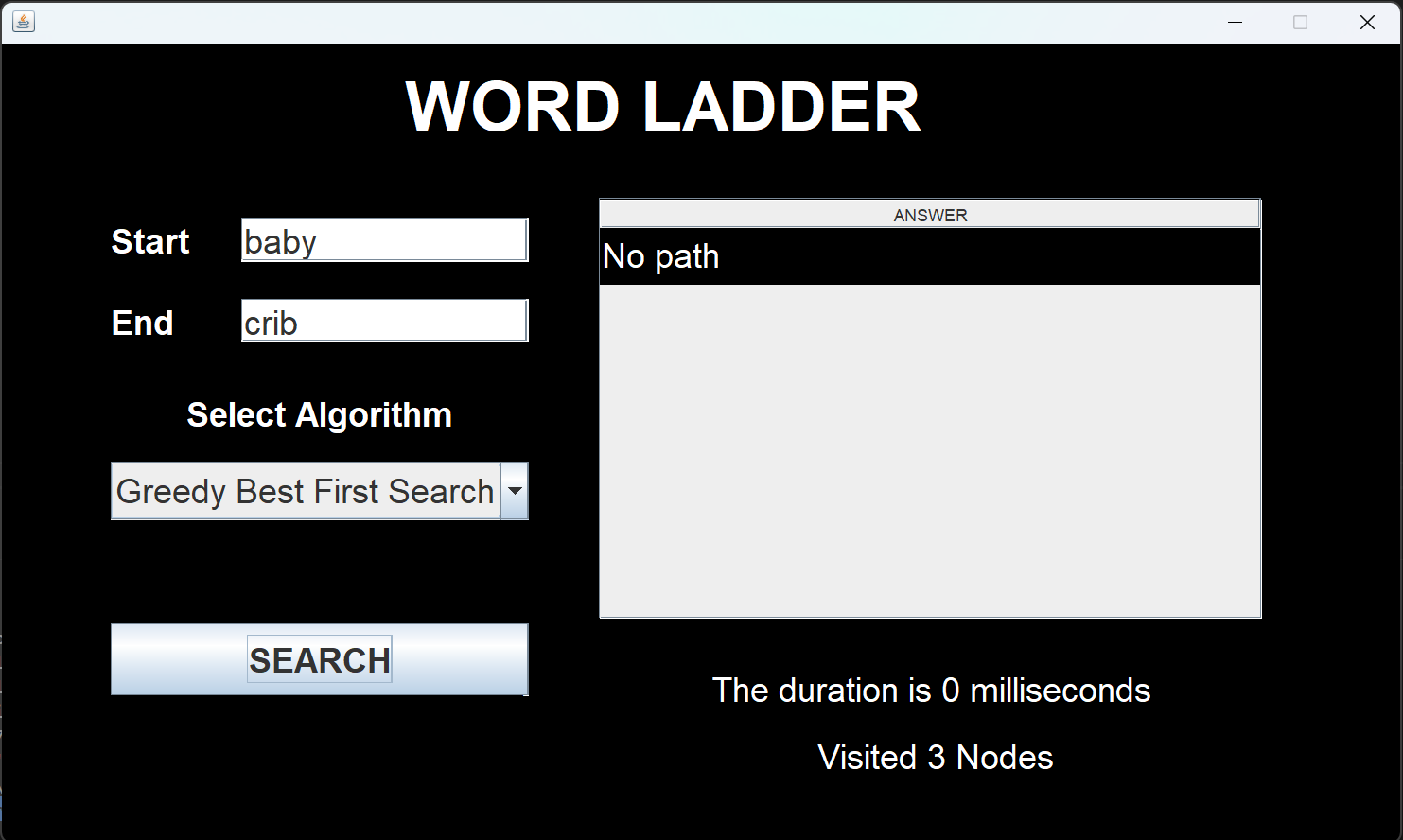
**HASIL UJI COBA**

1. Baby – Crib
2. UCS

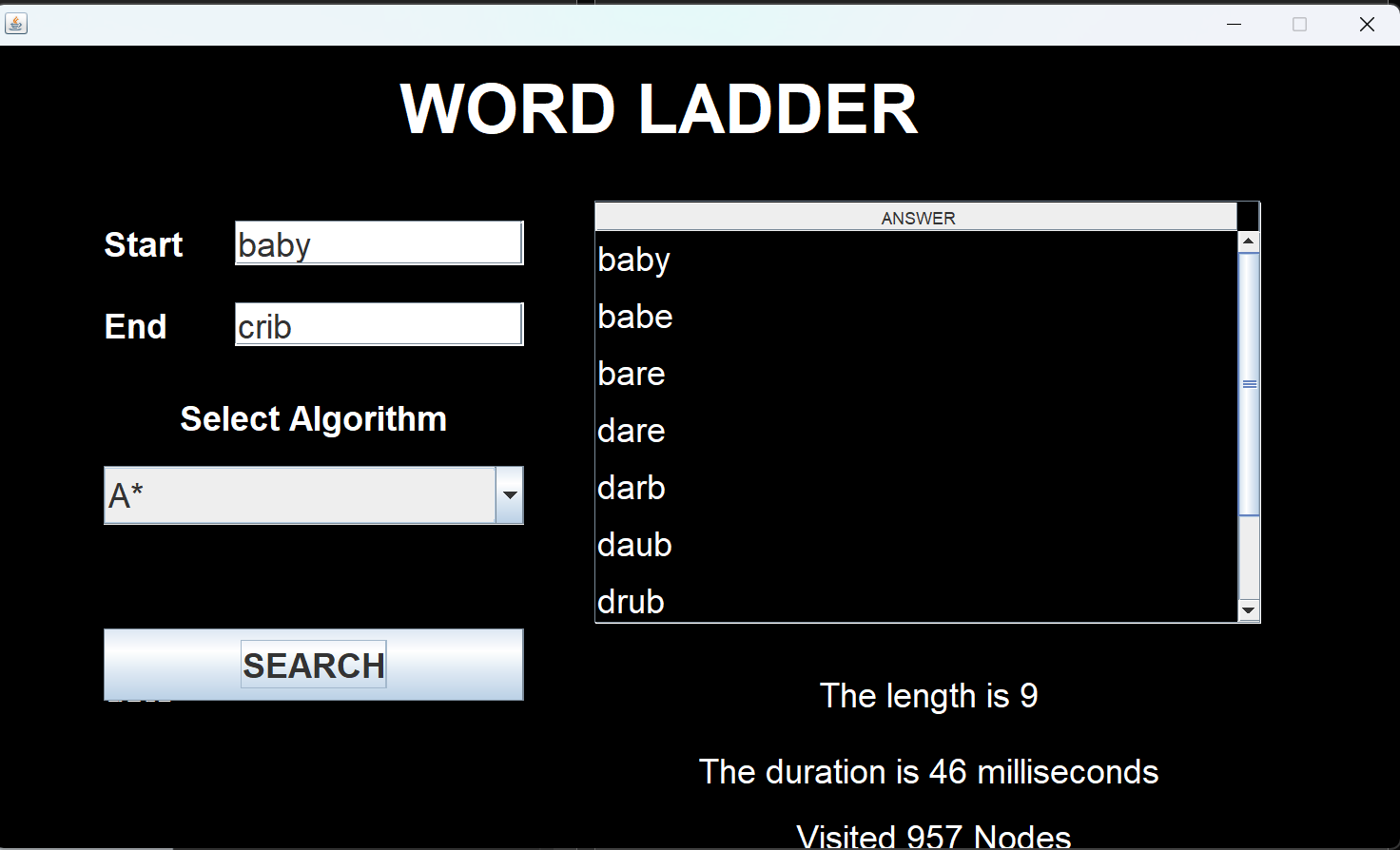


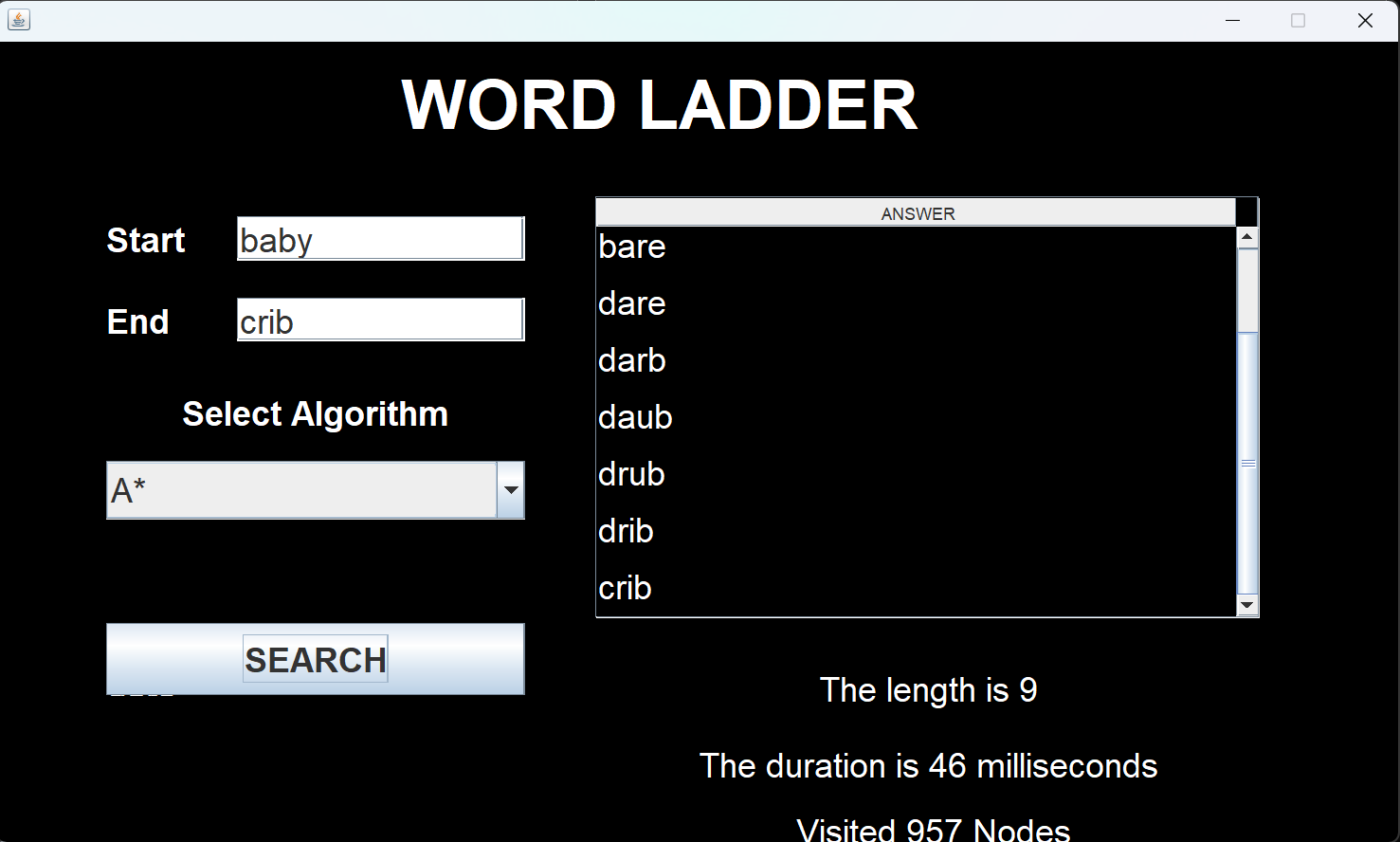


1. GBFS



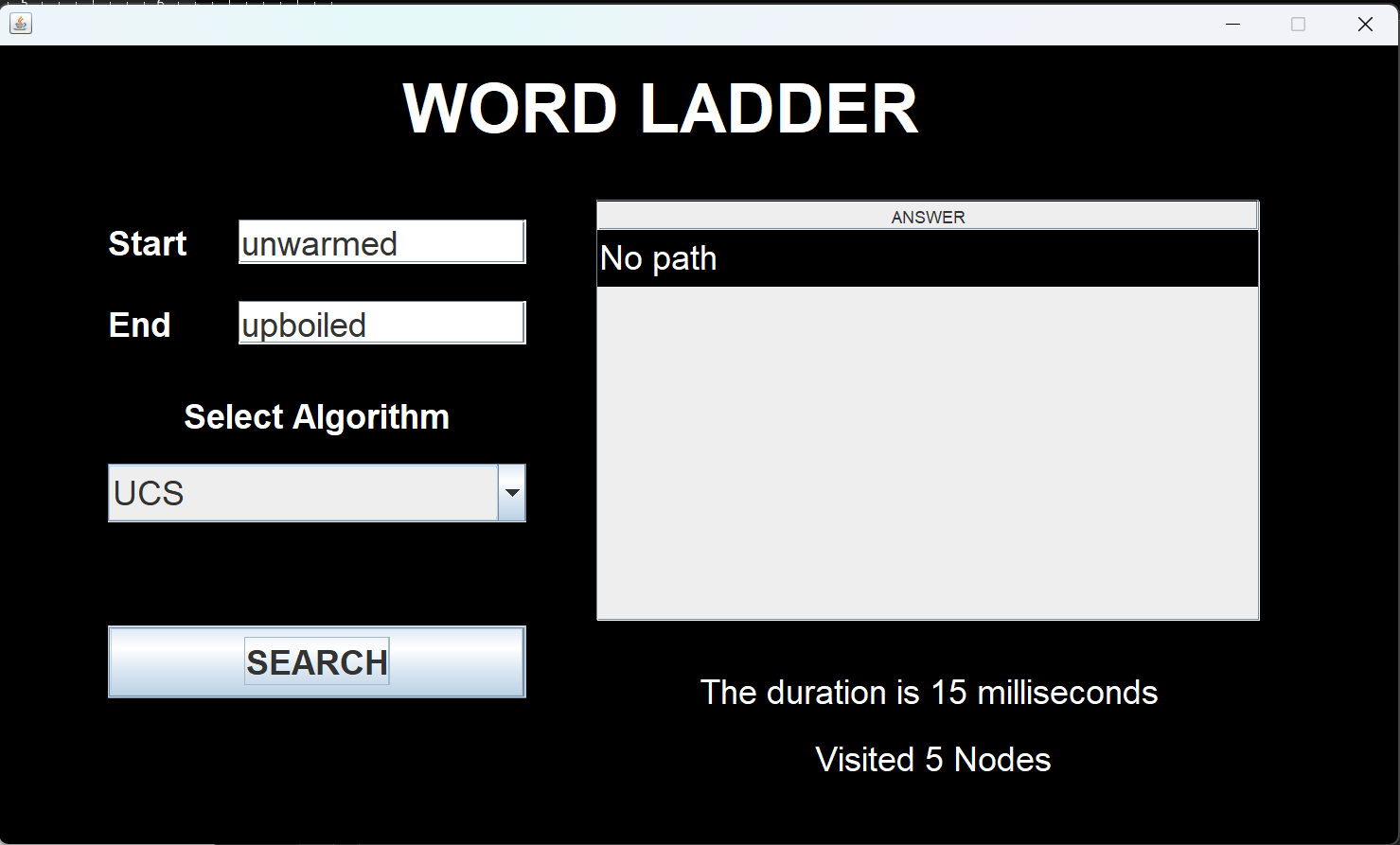
1. A Star



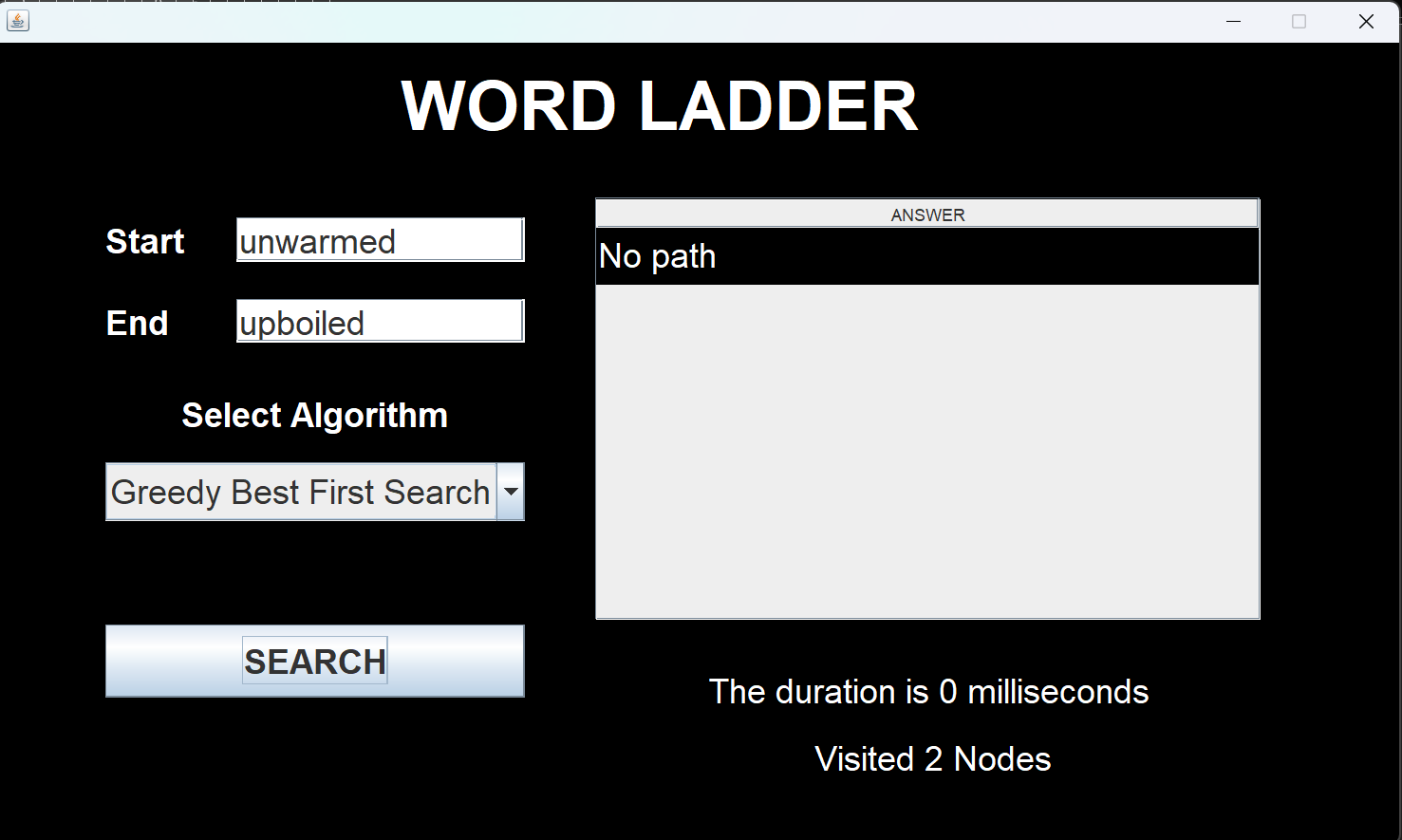


1. Unwarmed – Upboiled

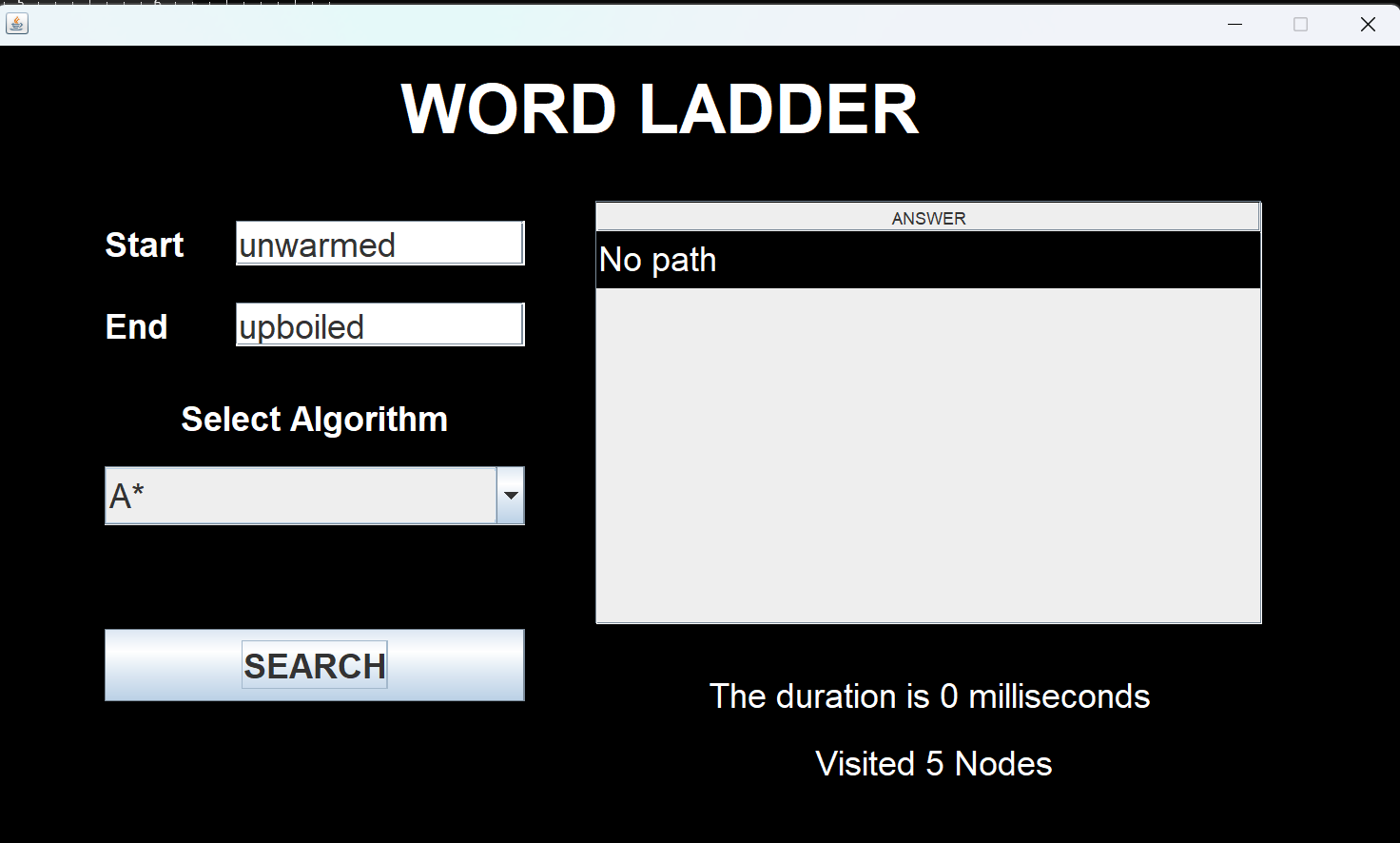
a. UCS



1. GBFS

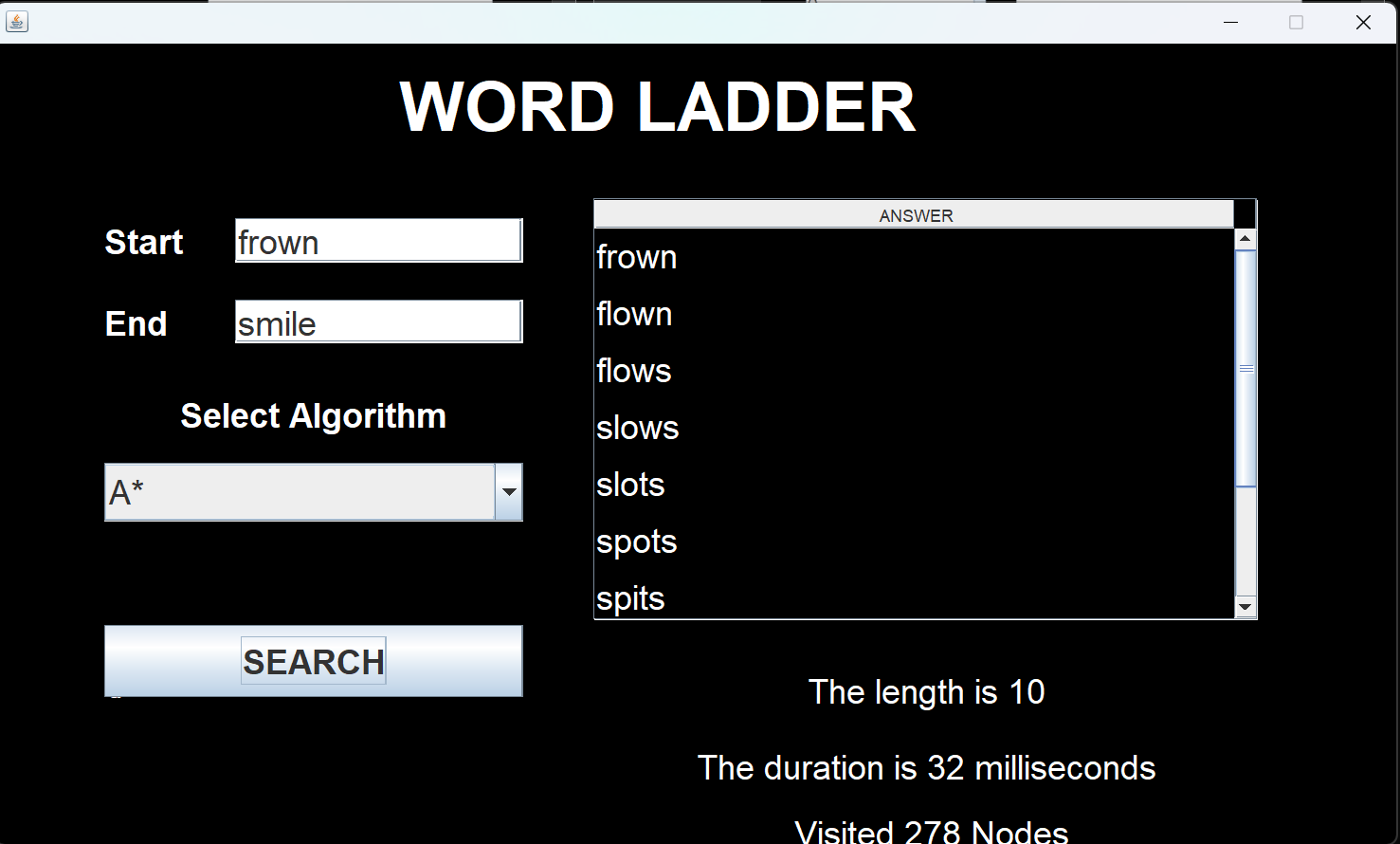


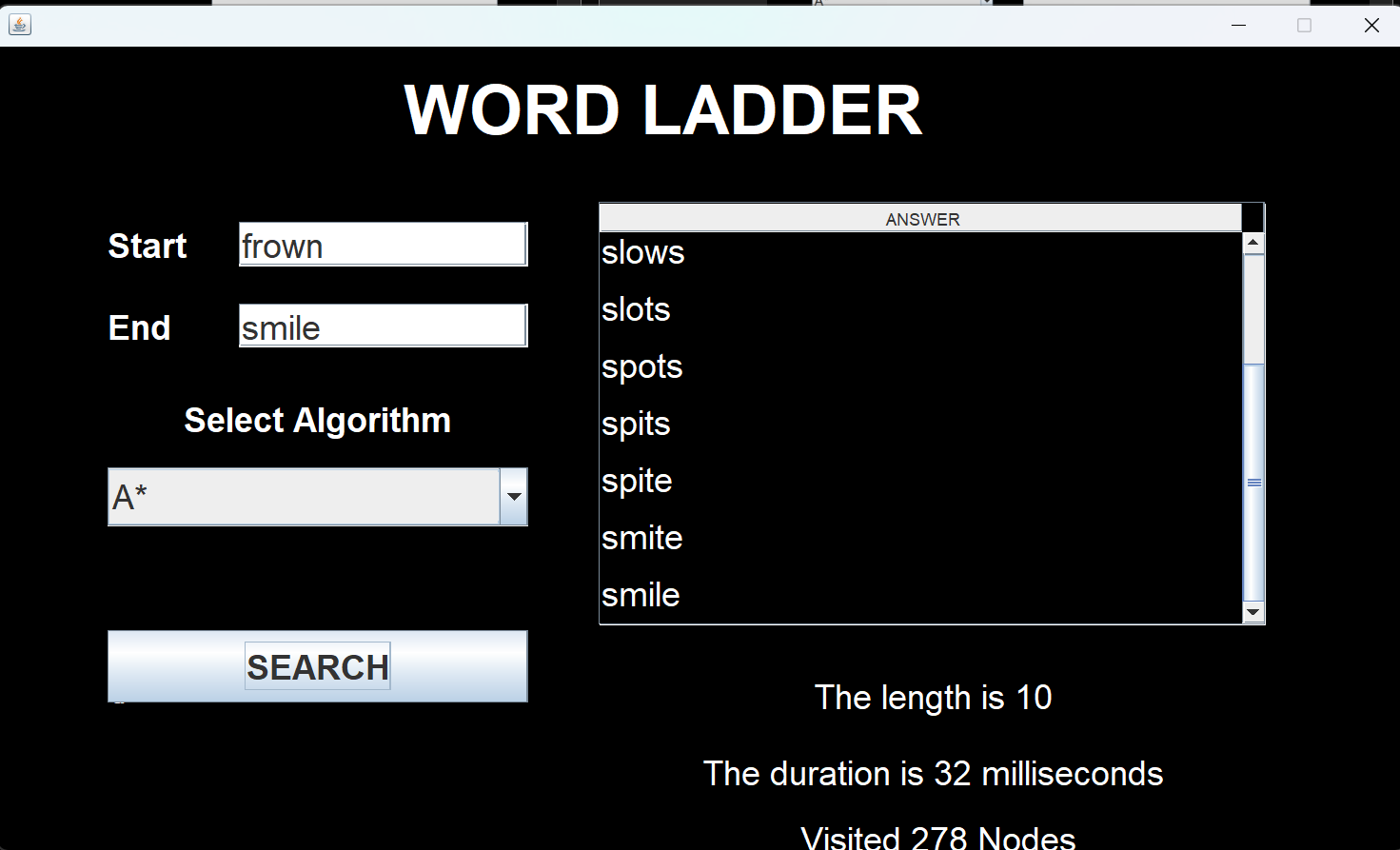
1. A Star



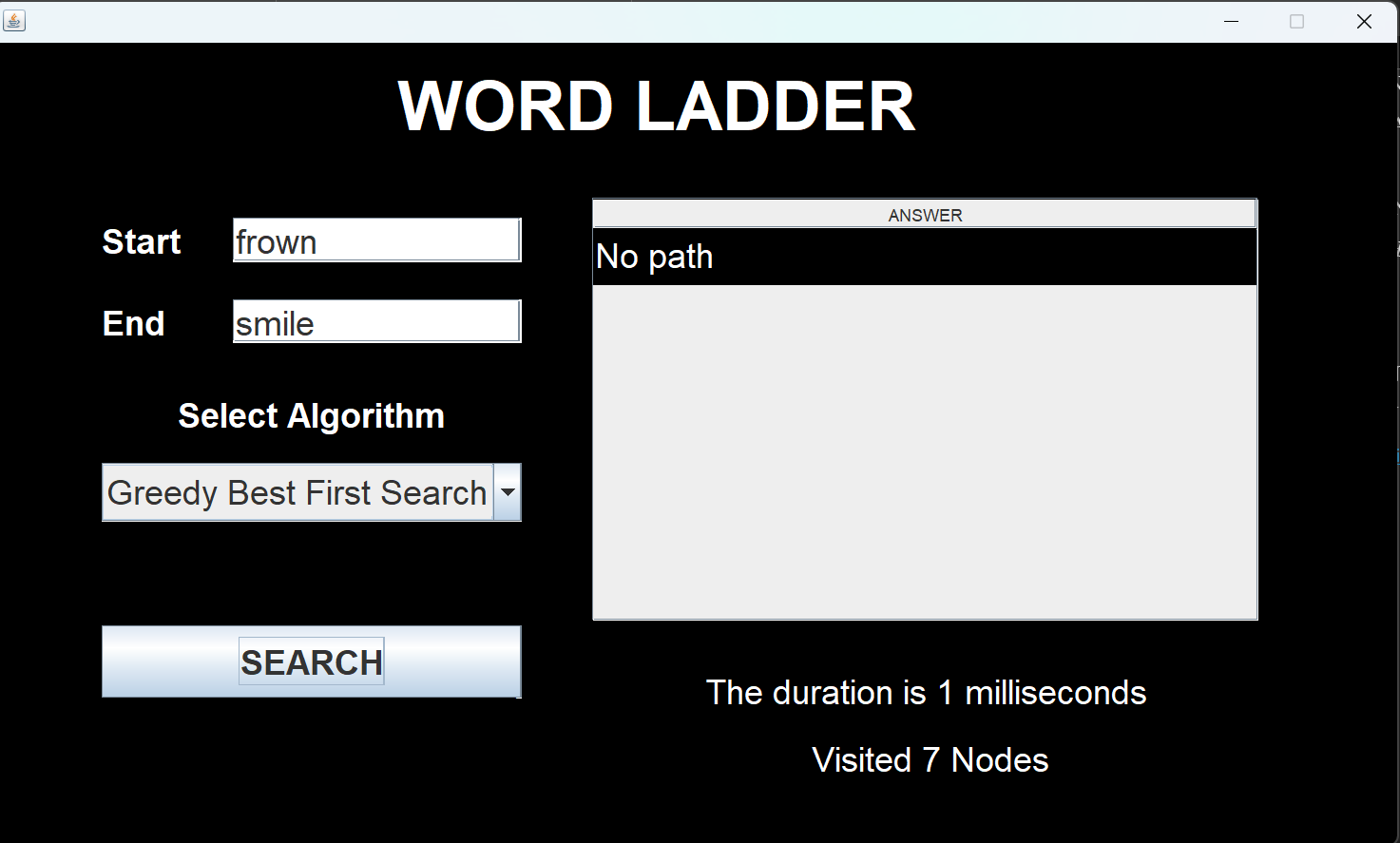
1. Frown – Smile

a. UCS

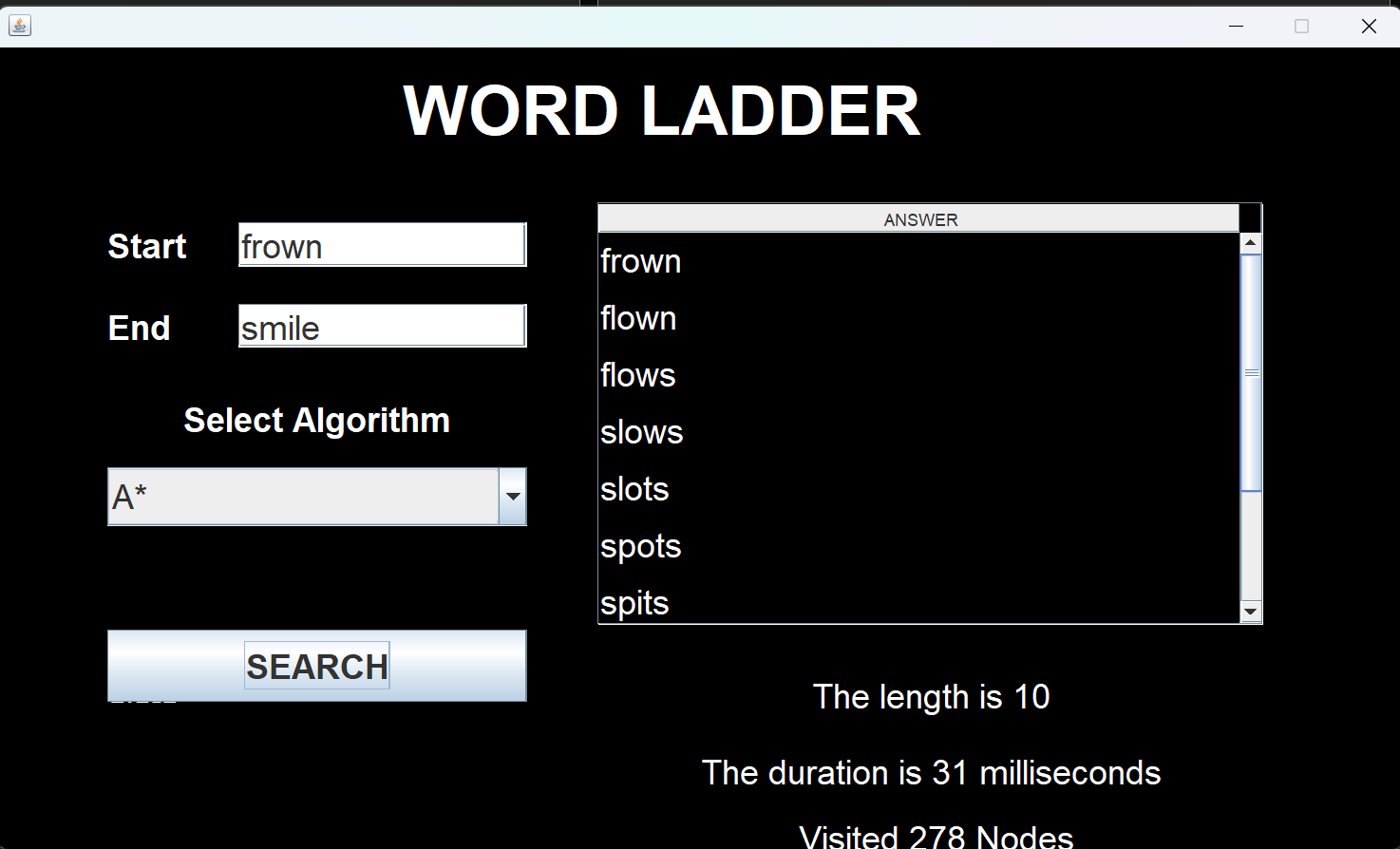


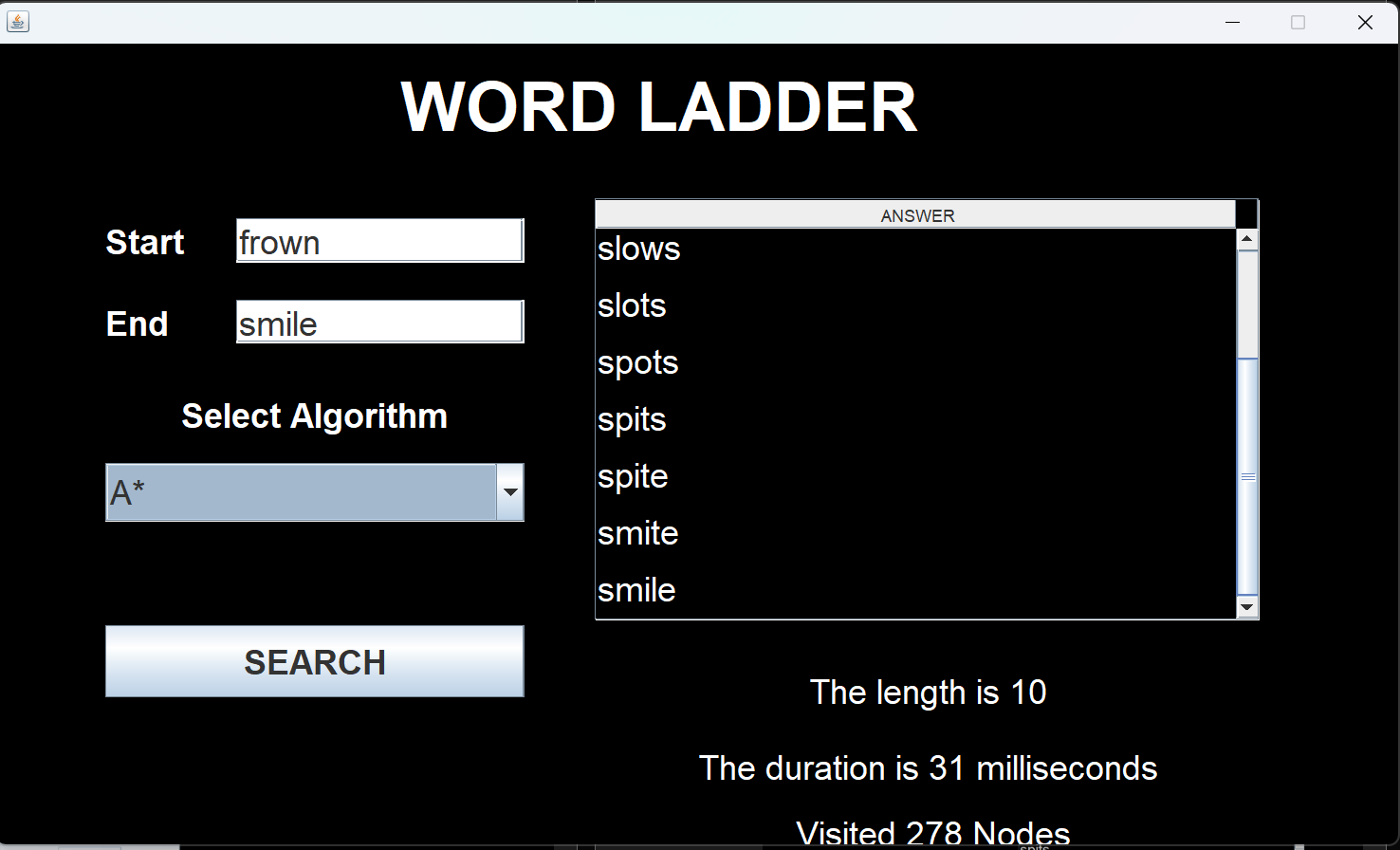


1. GBFS

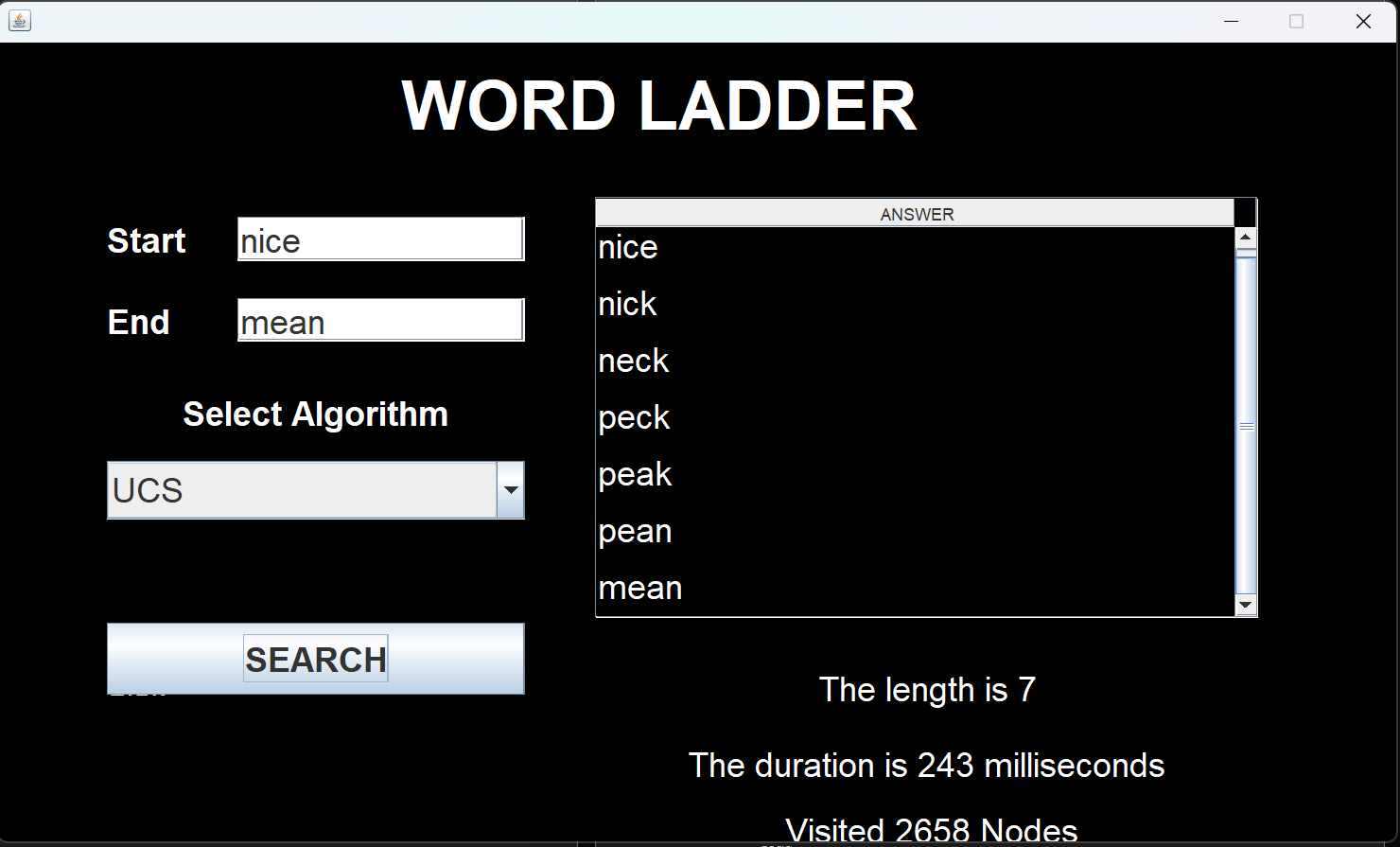


1. A Star

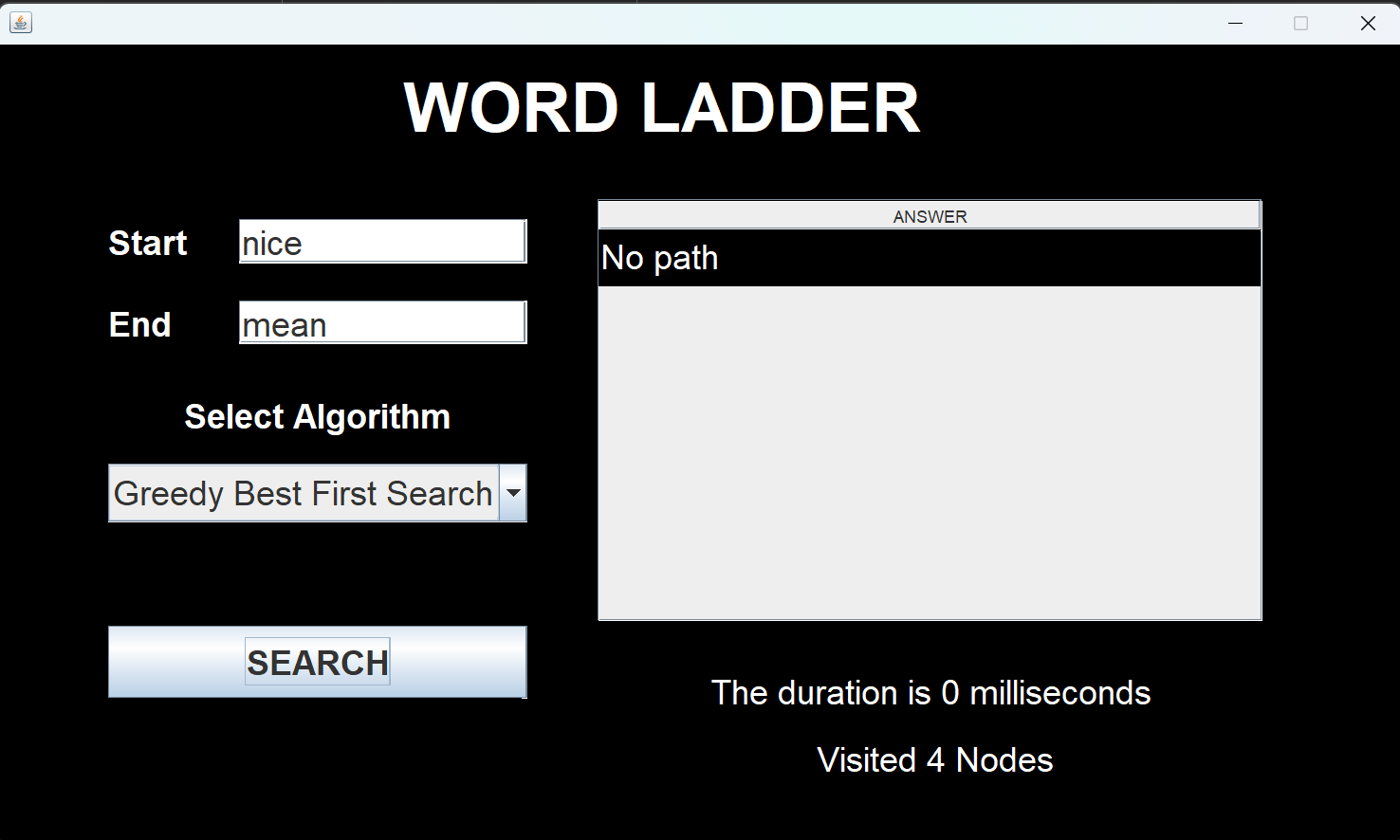




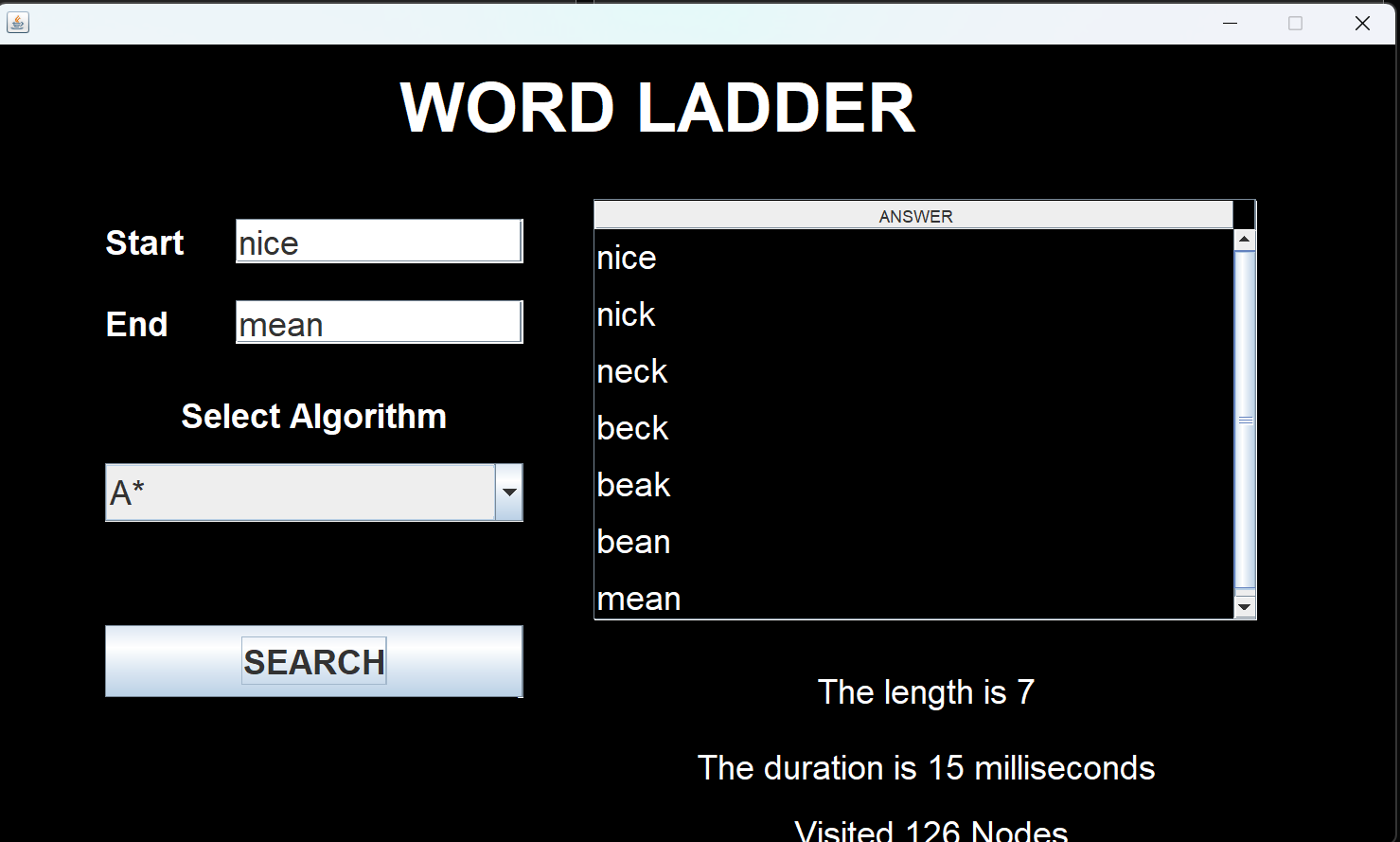
1. Nice – Mean
2. UCS



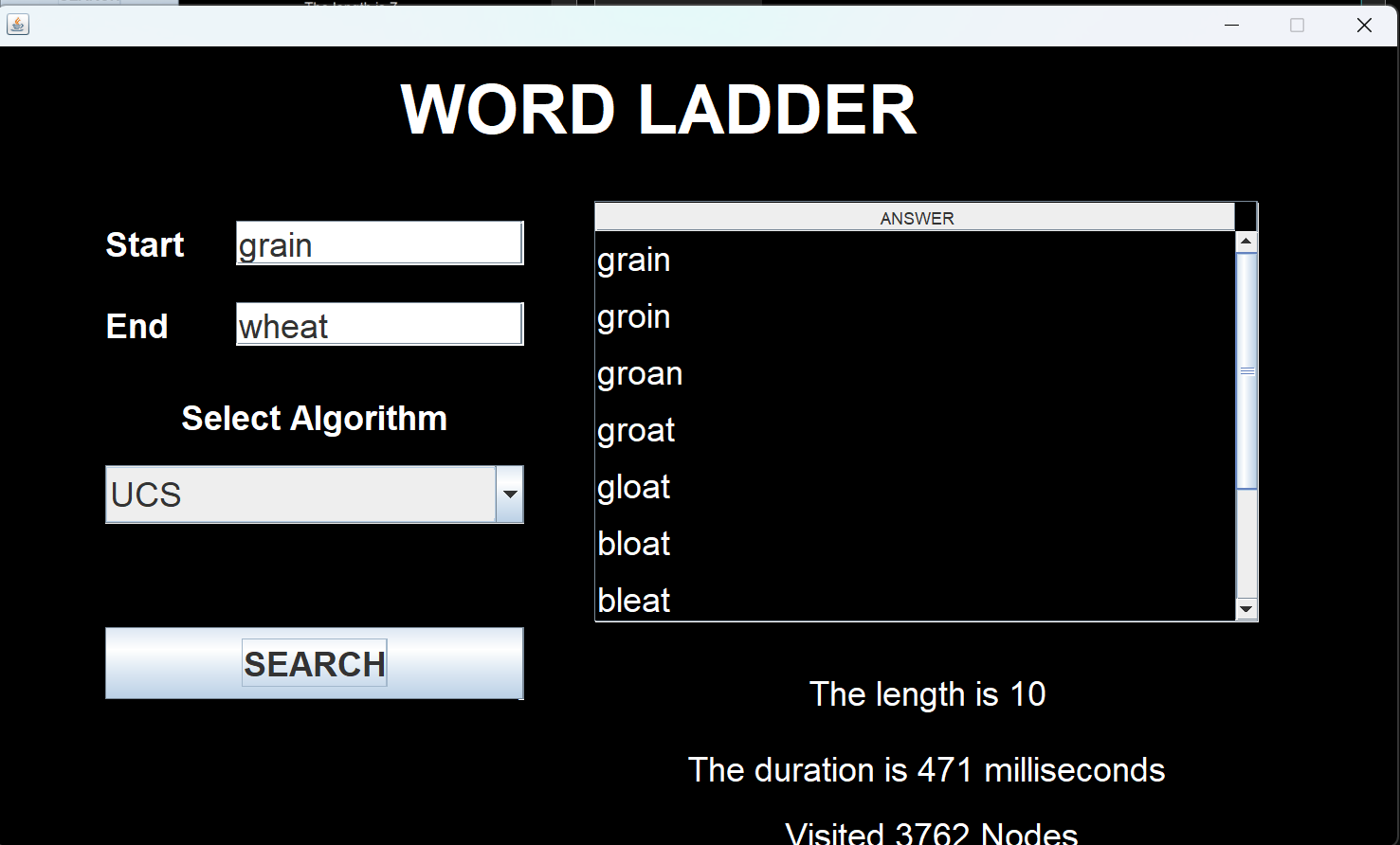
1. GBFS

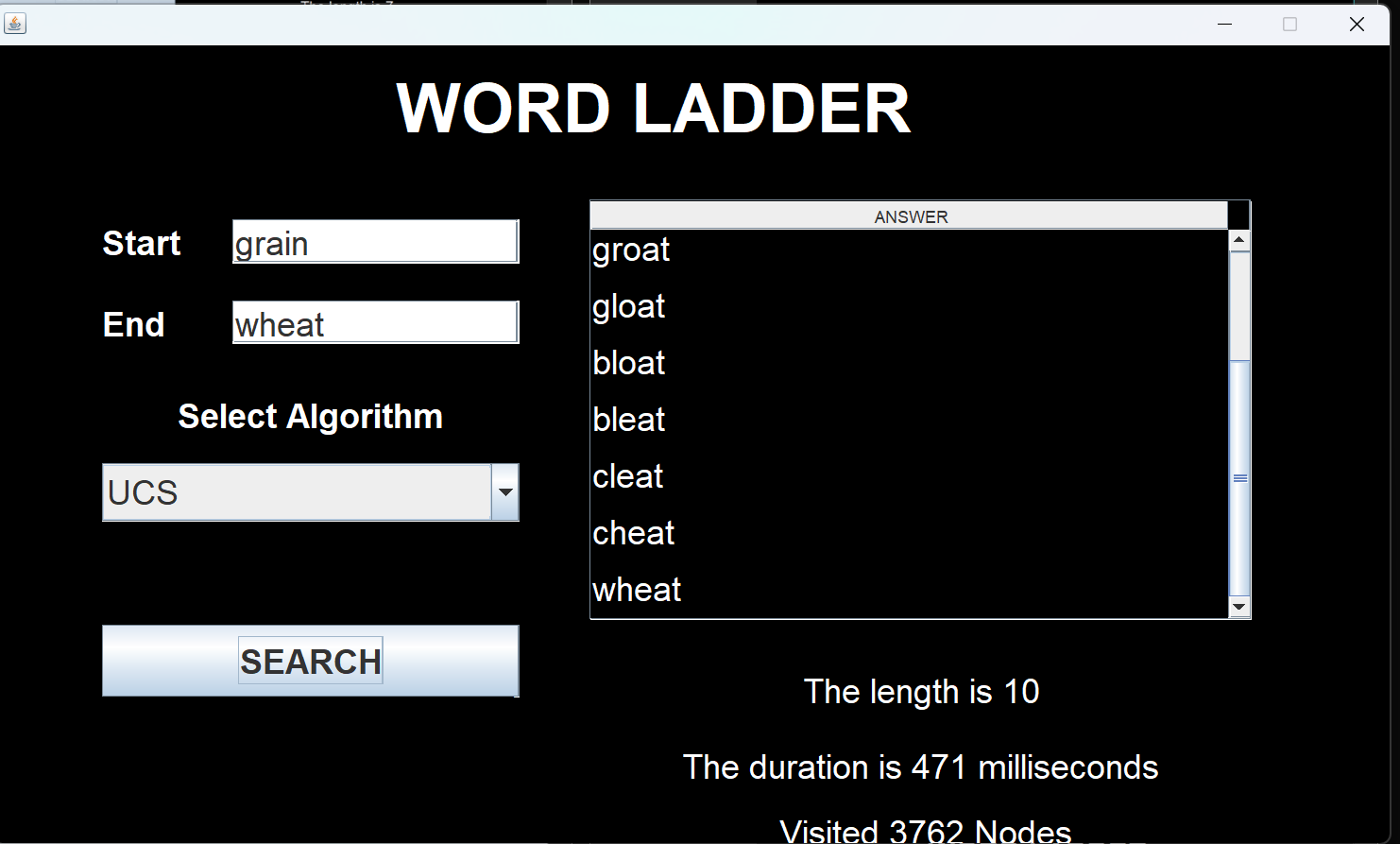


1. A Star

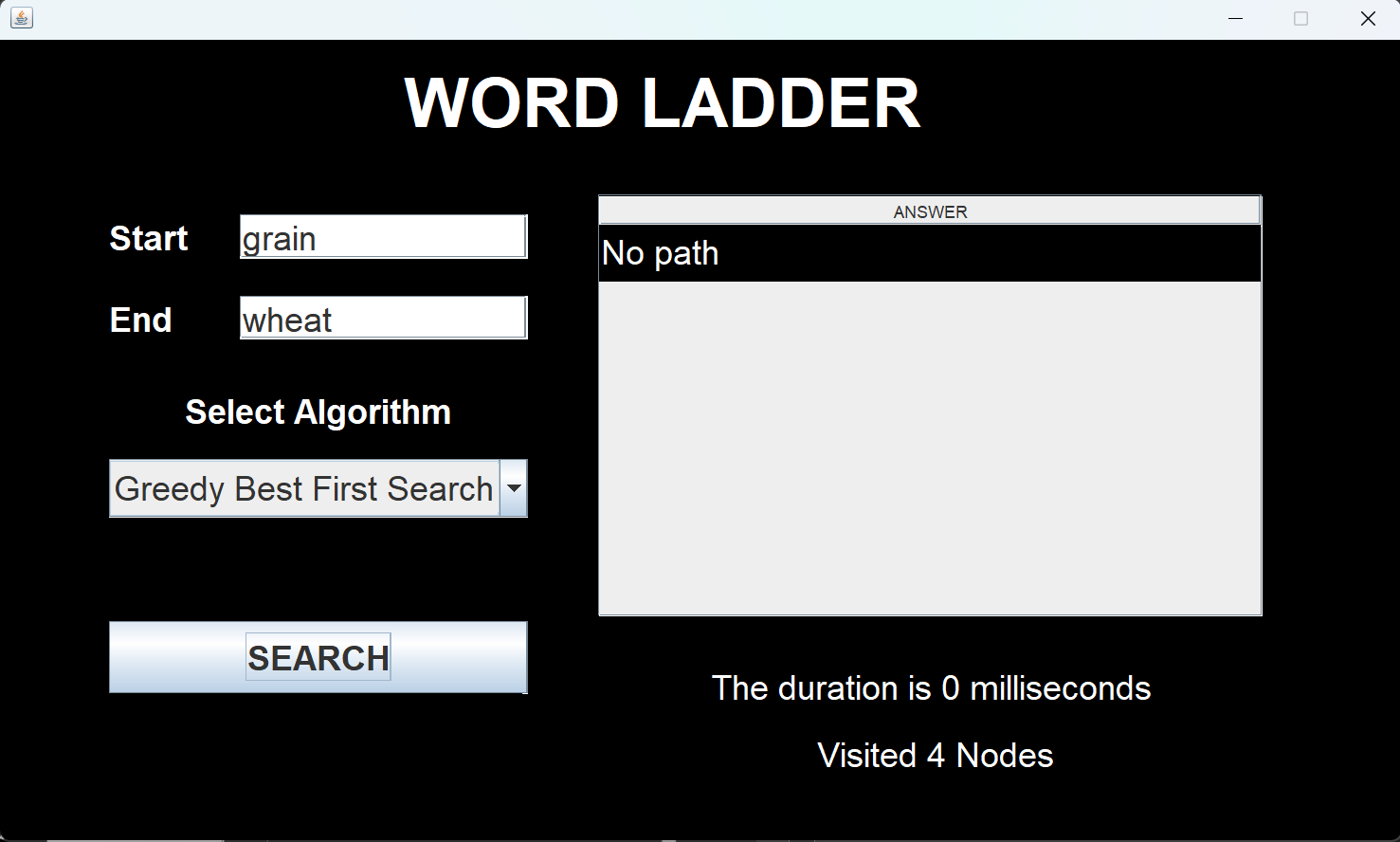


1. Grain – Wheat
2. UCS

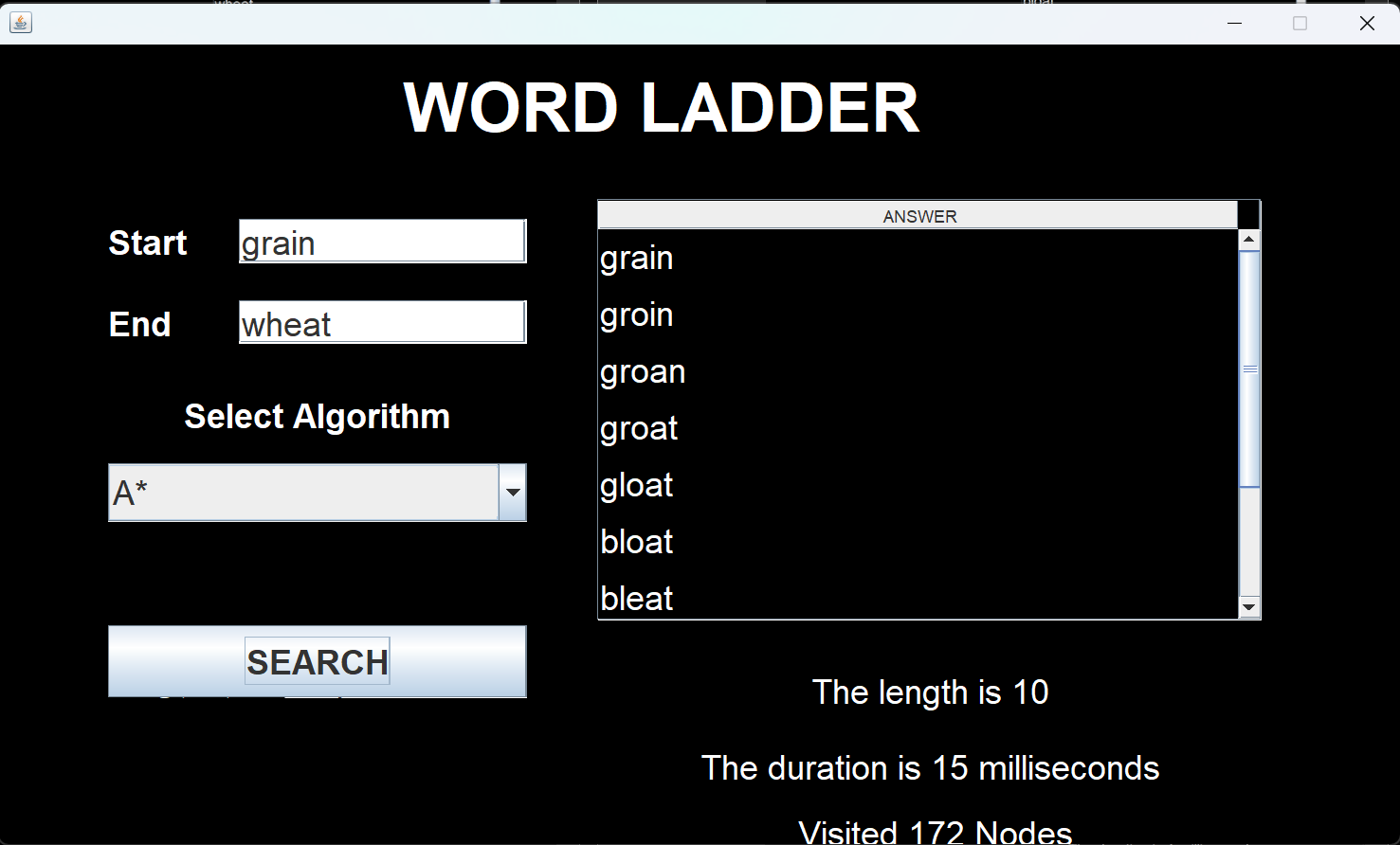


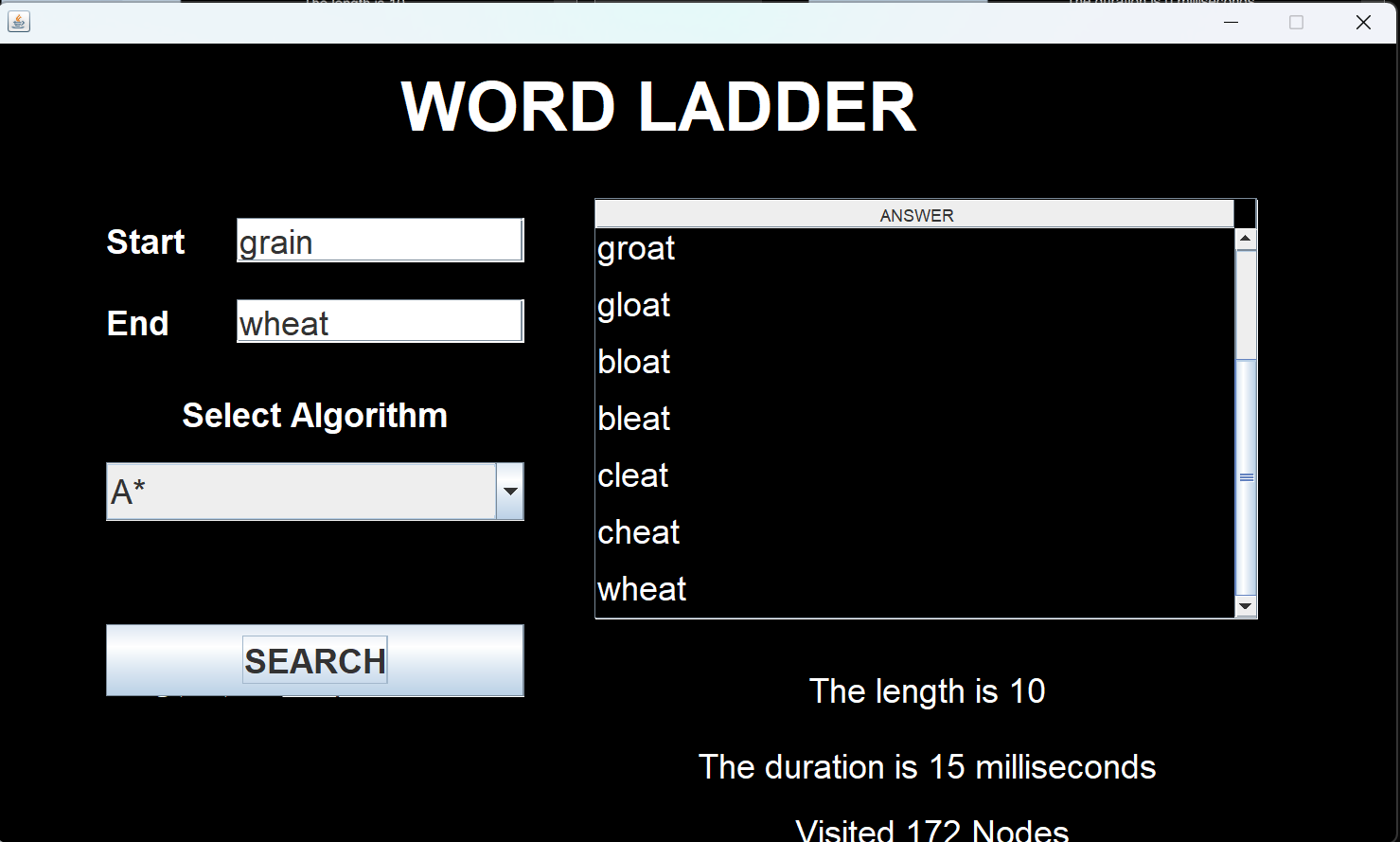


1. GBFS

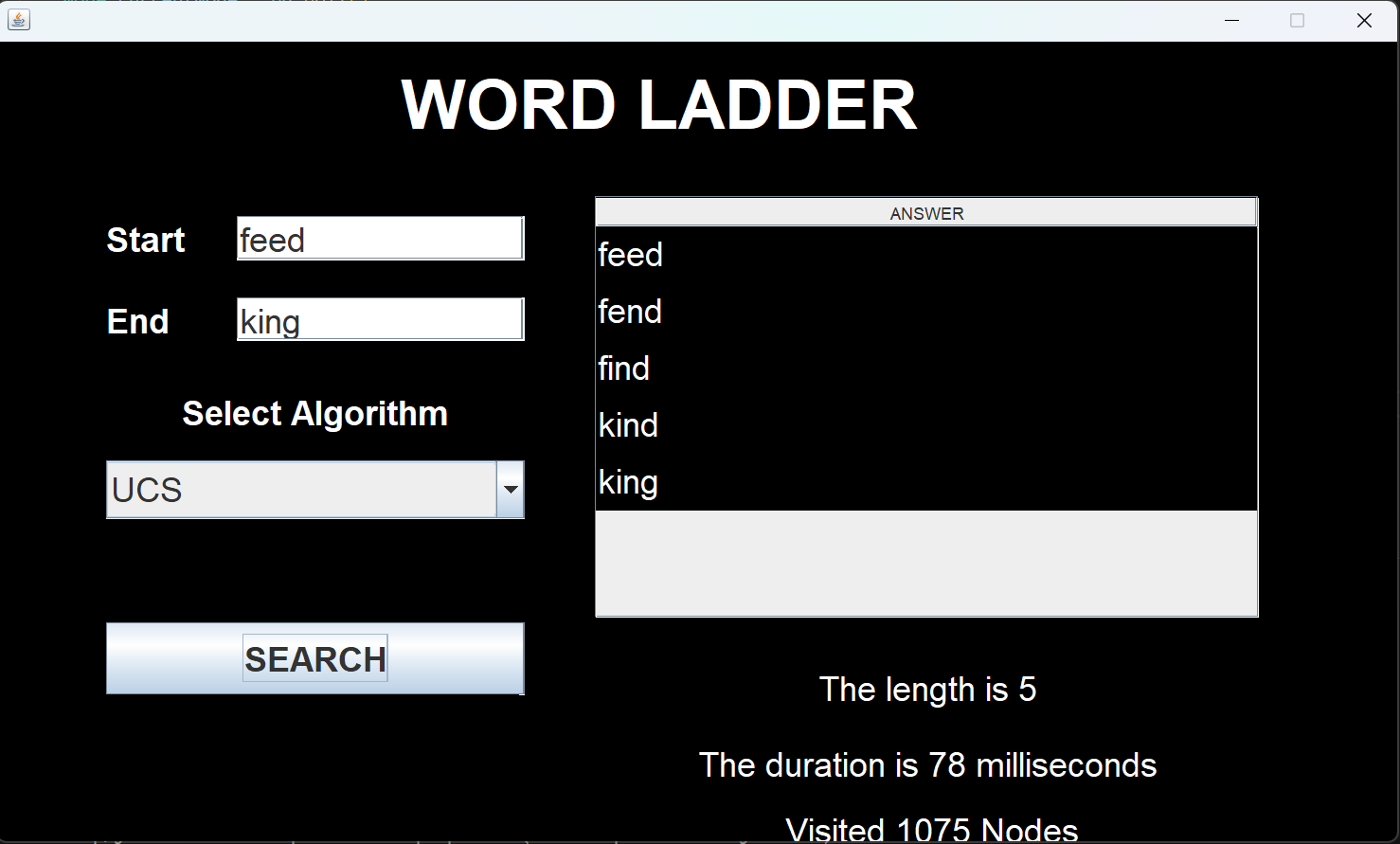


1. A Star

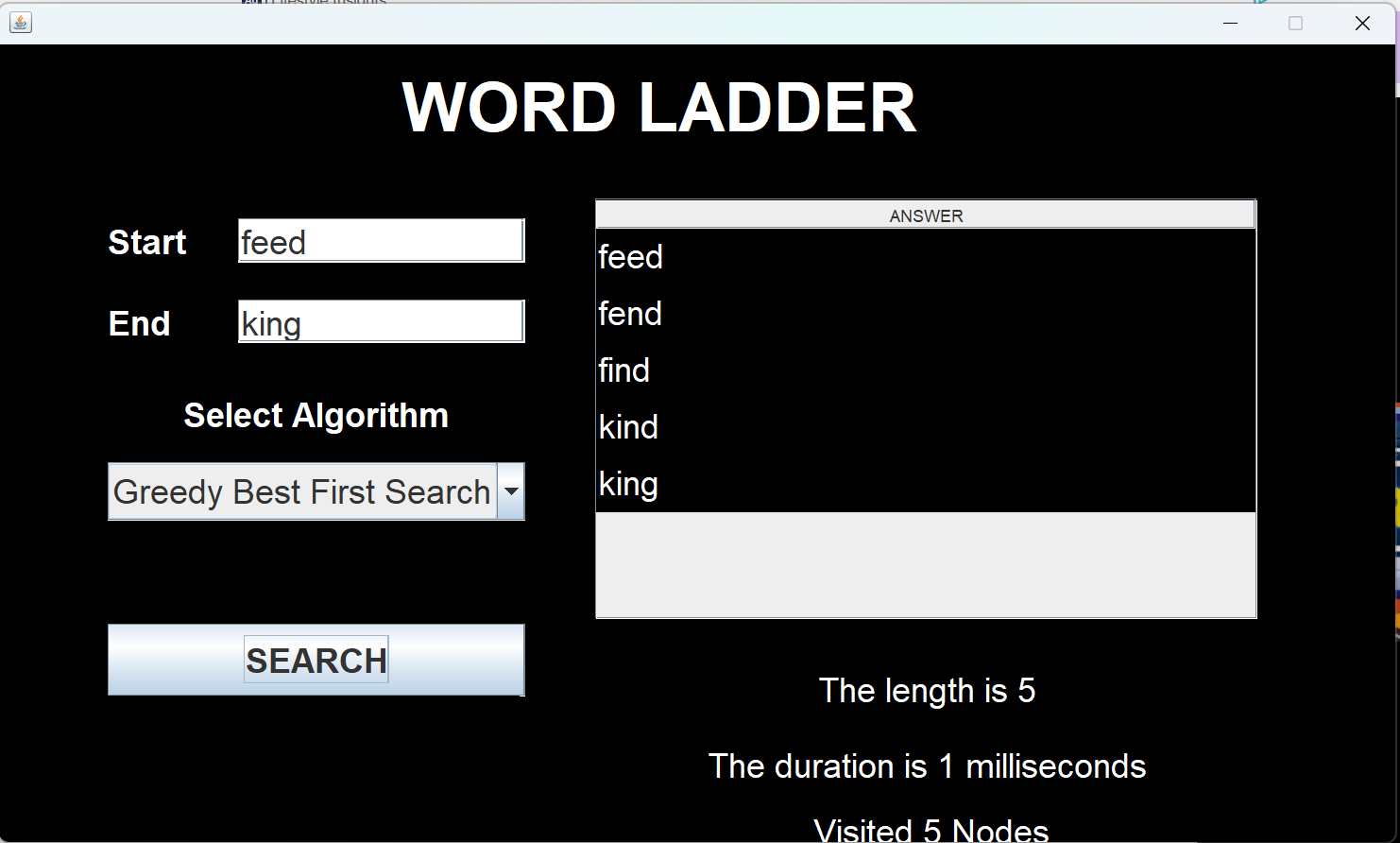




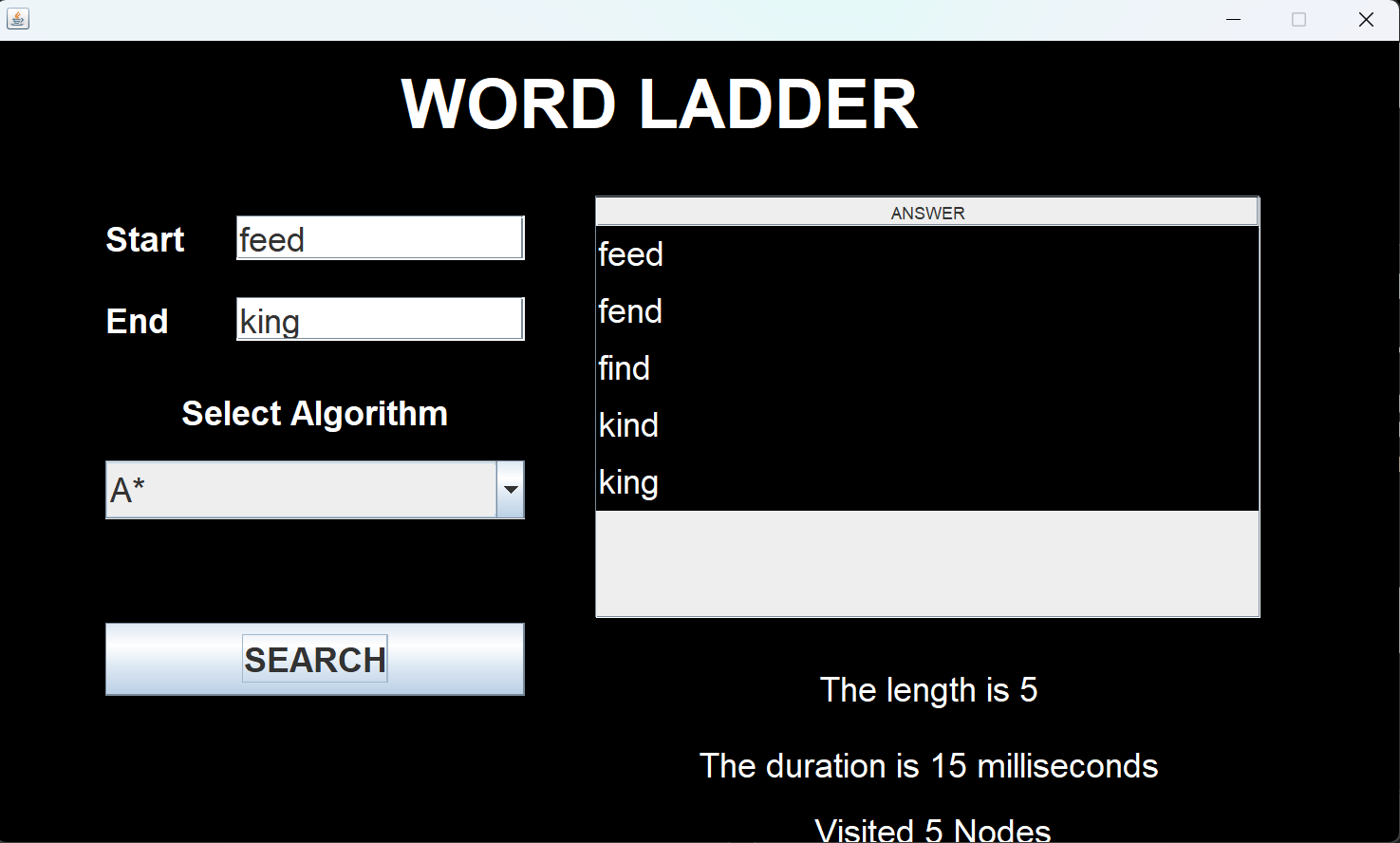
1. Feed – King
2. UCS



1. GBFS



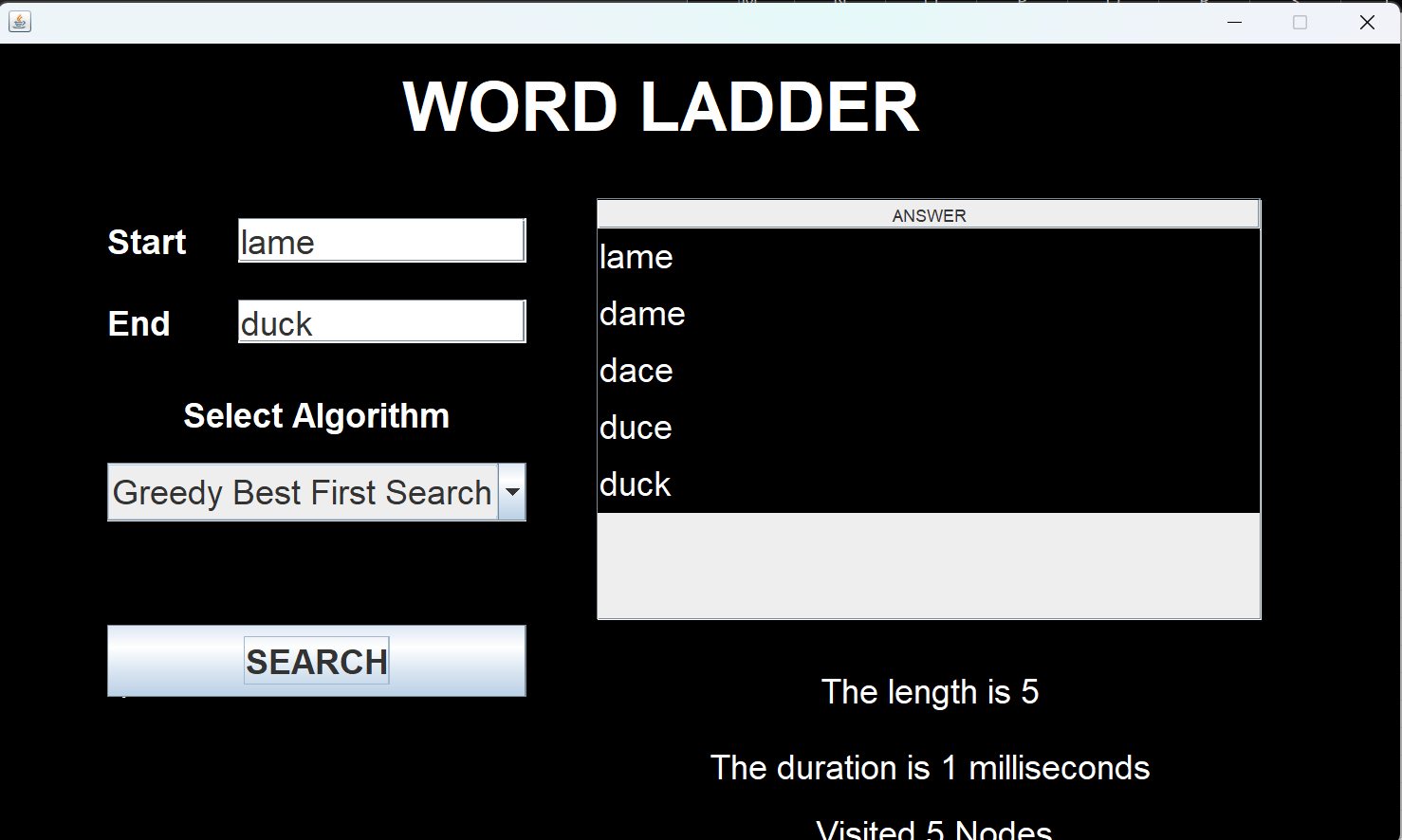
1. A Star



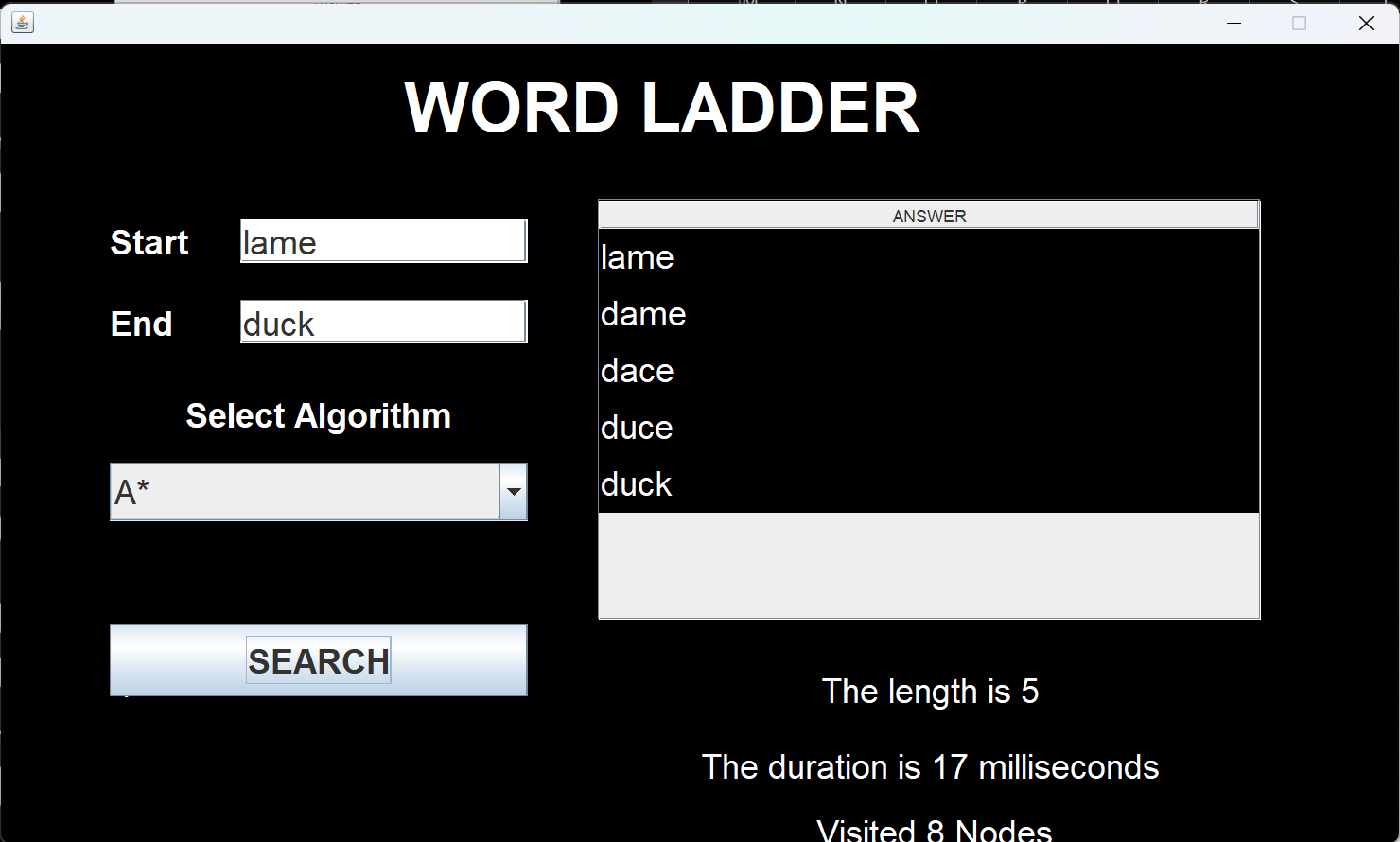
1. Lame – Duck
2. UCS



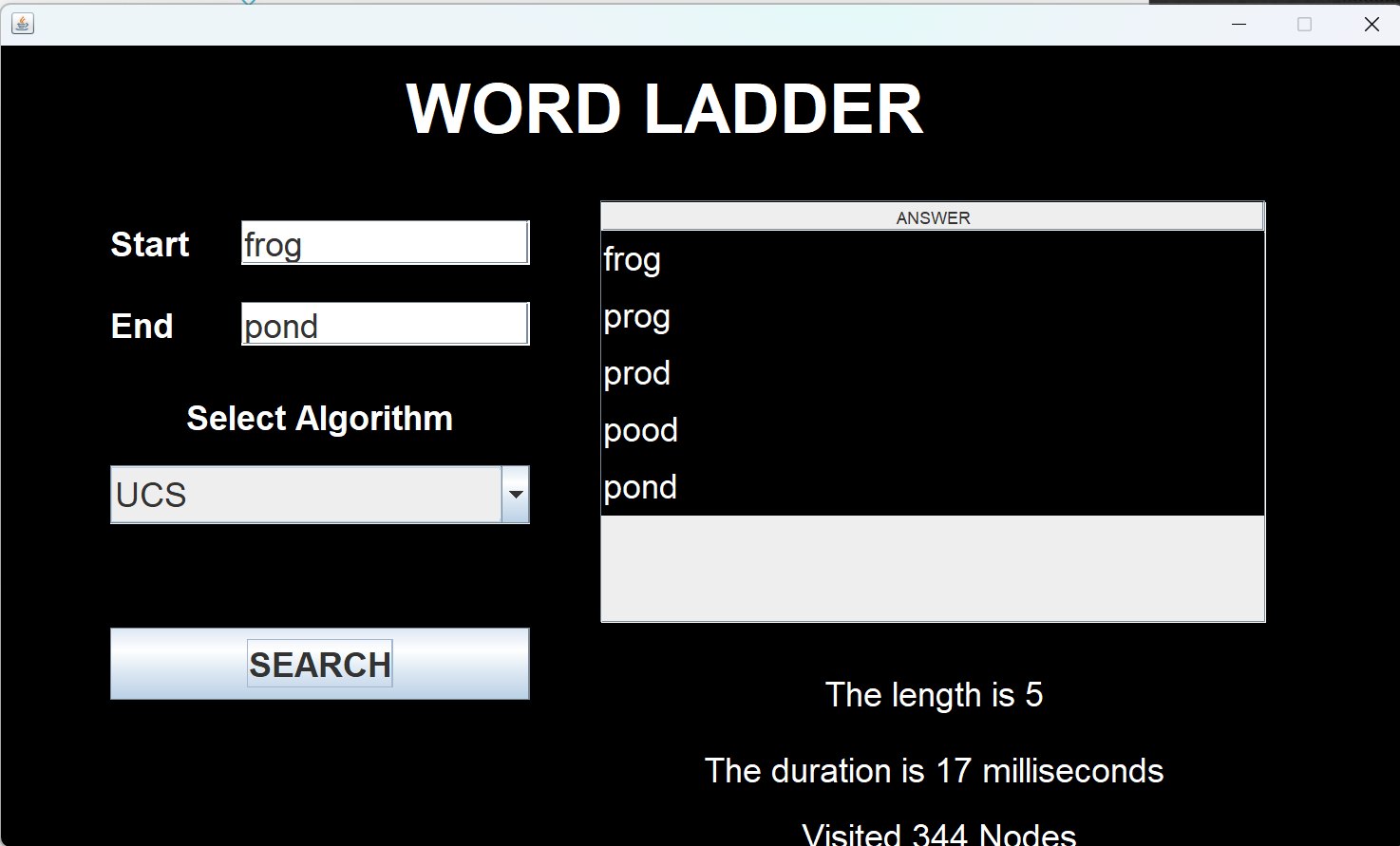
1. GBFS



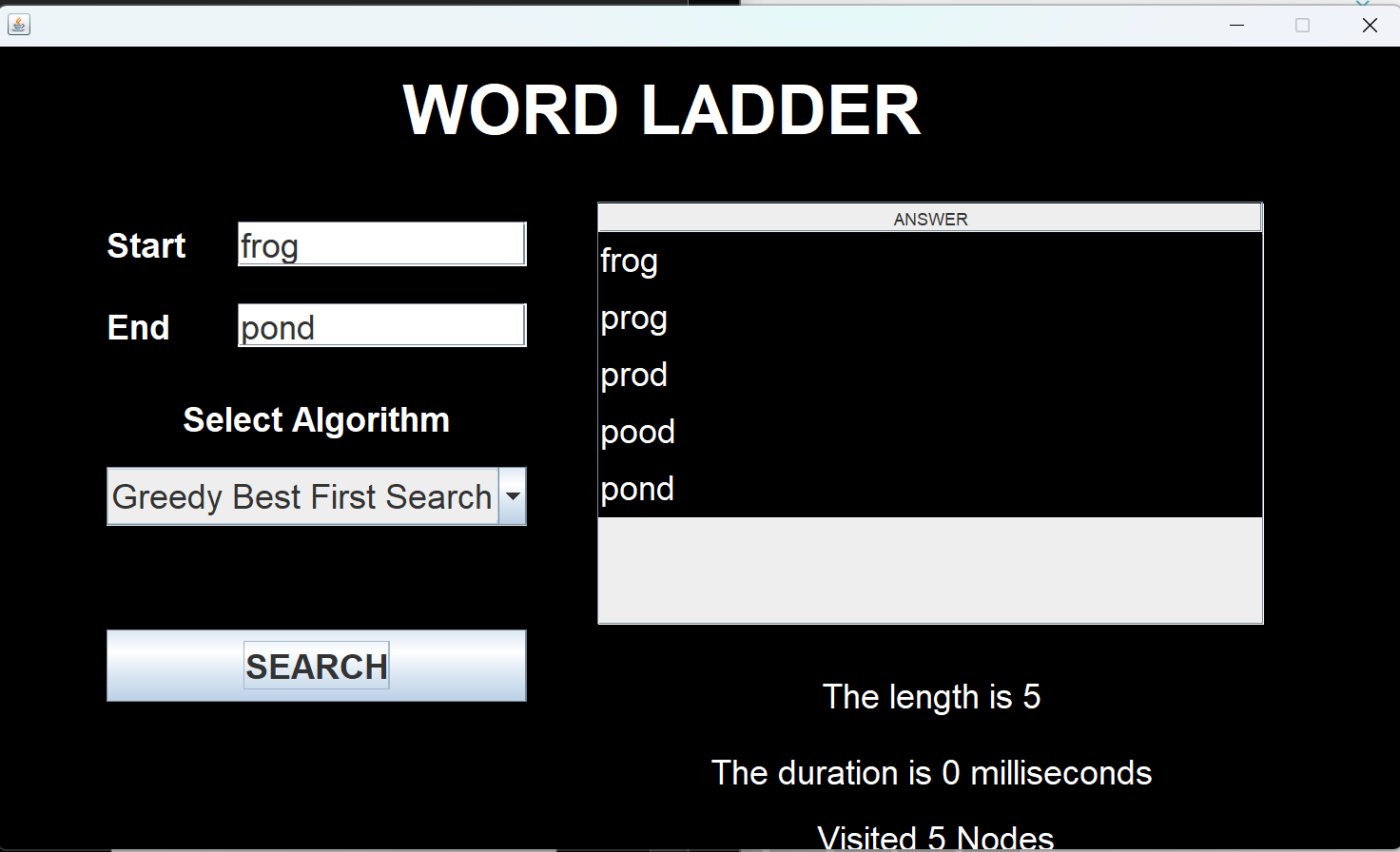
1. A Star



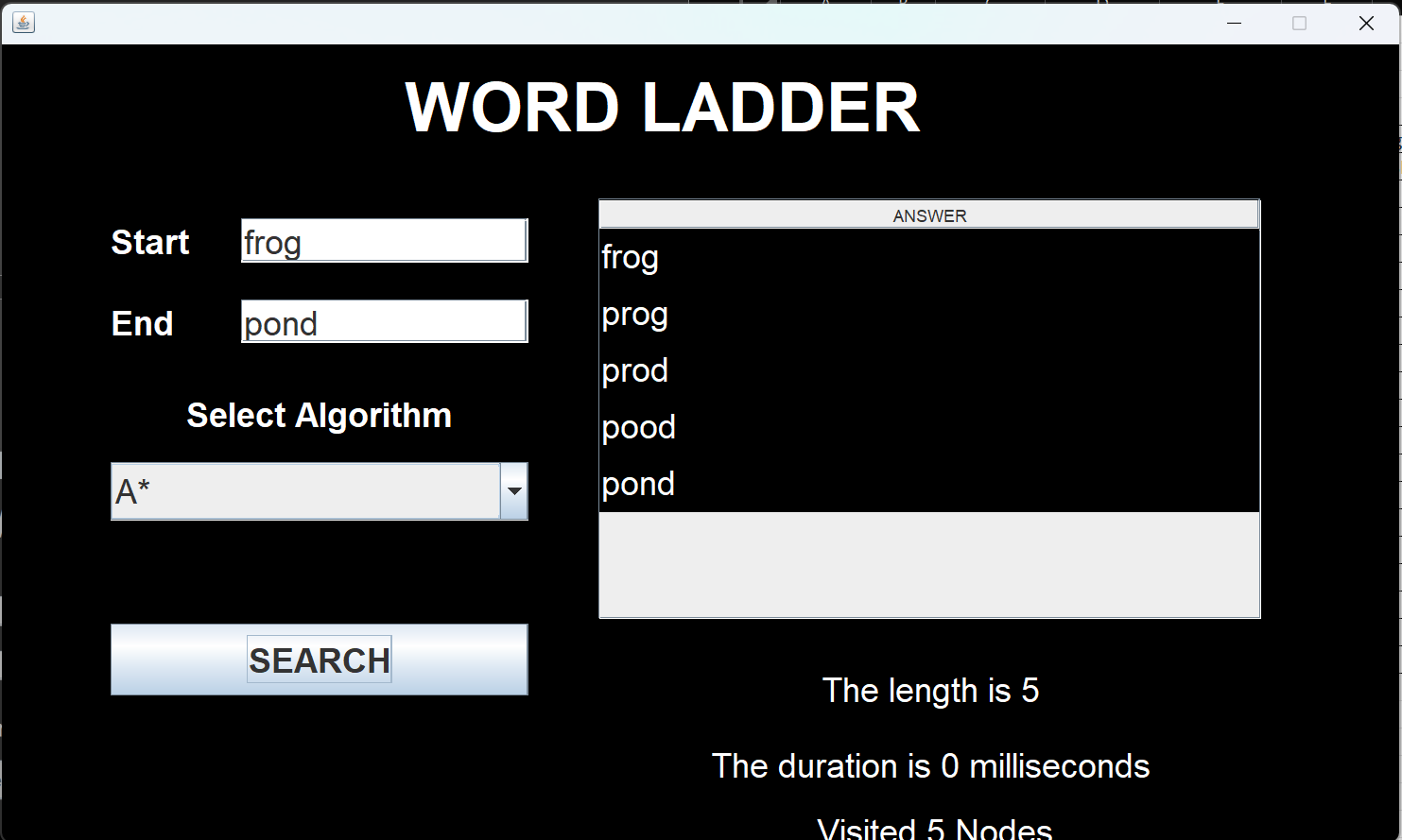
1. Frog – Pond
2. UCS



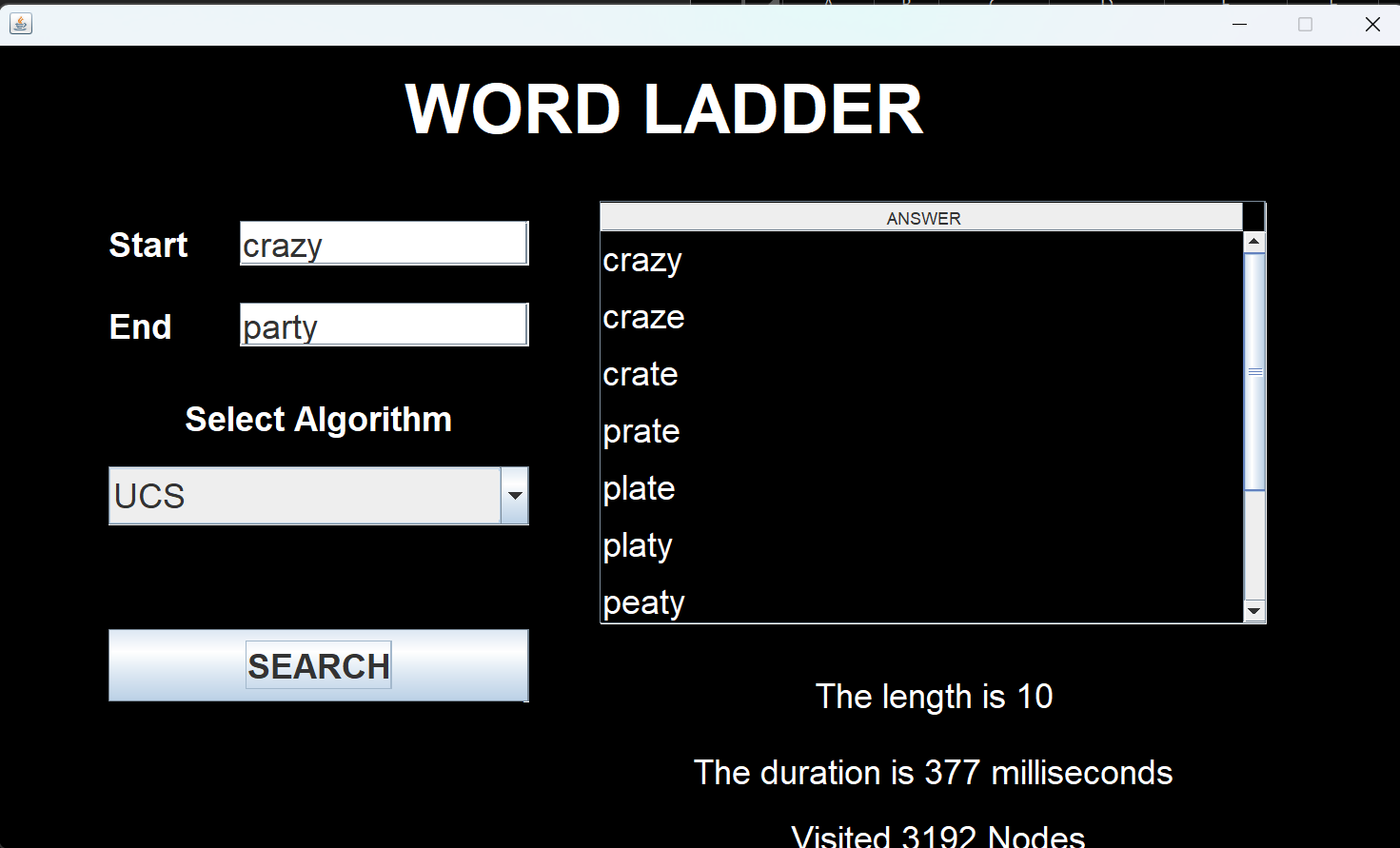
1. GBFS

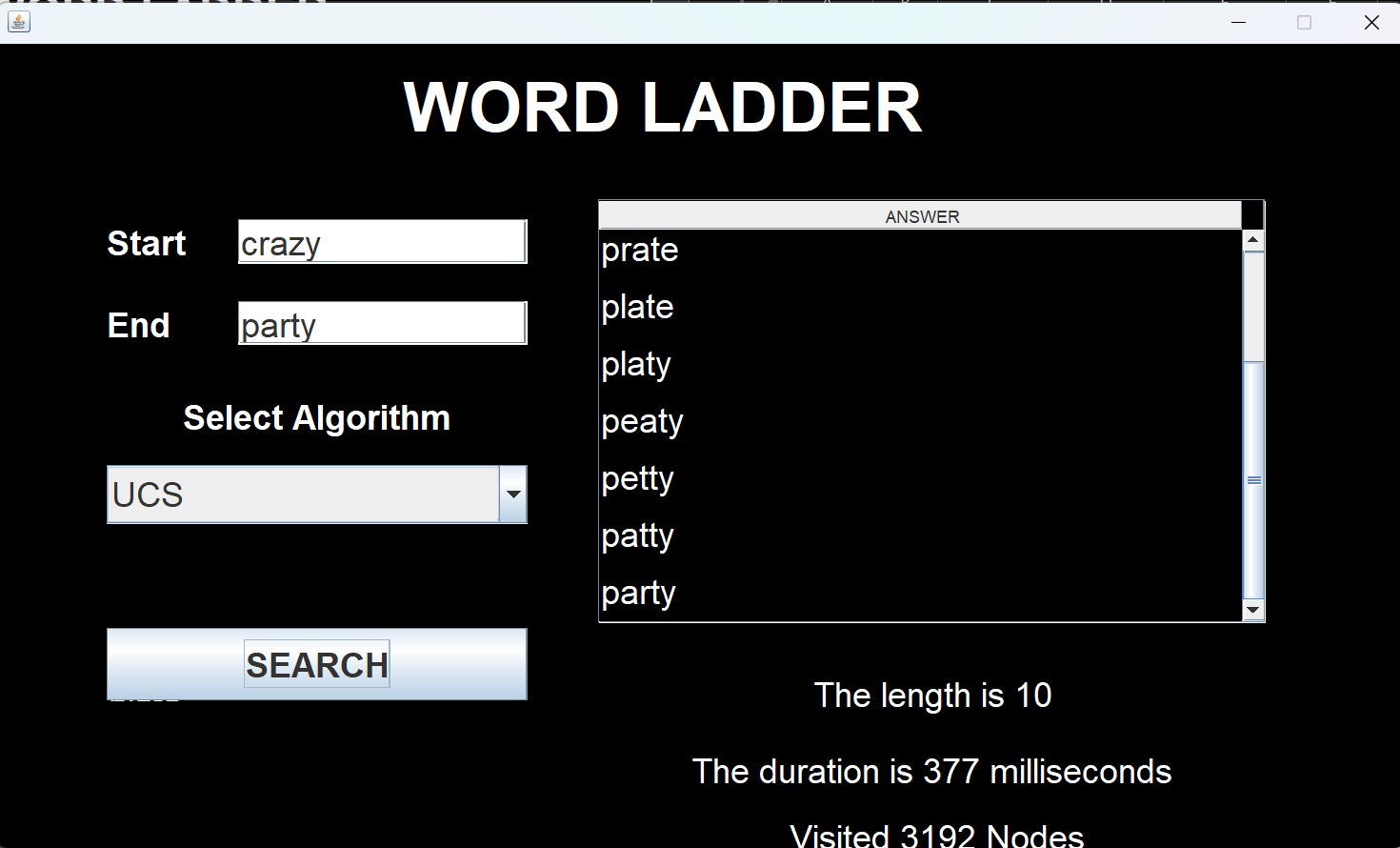


1. A Star

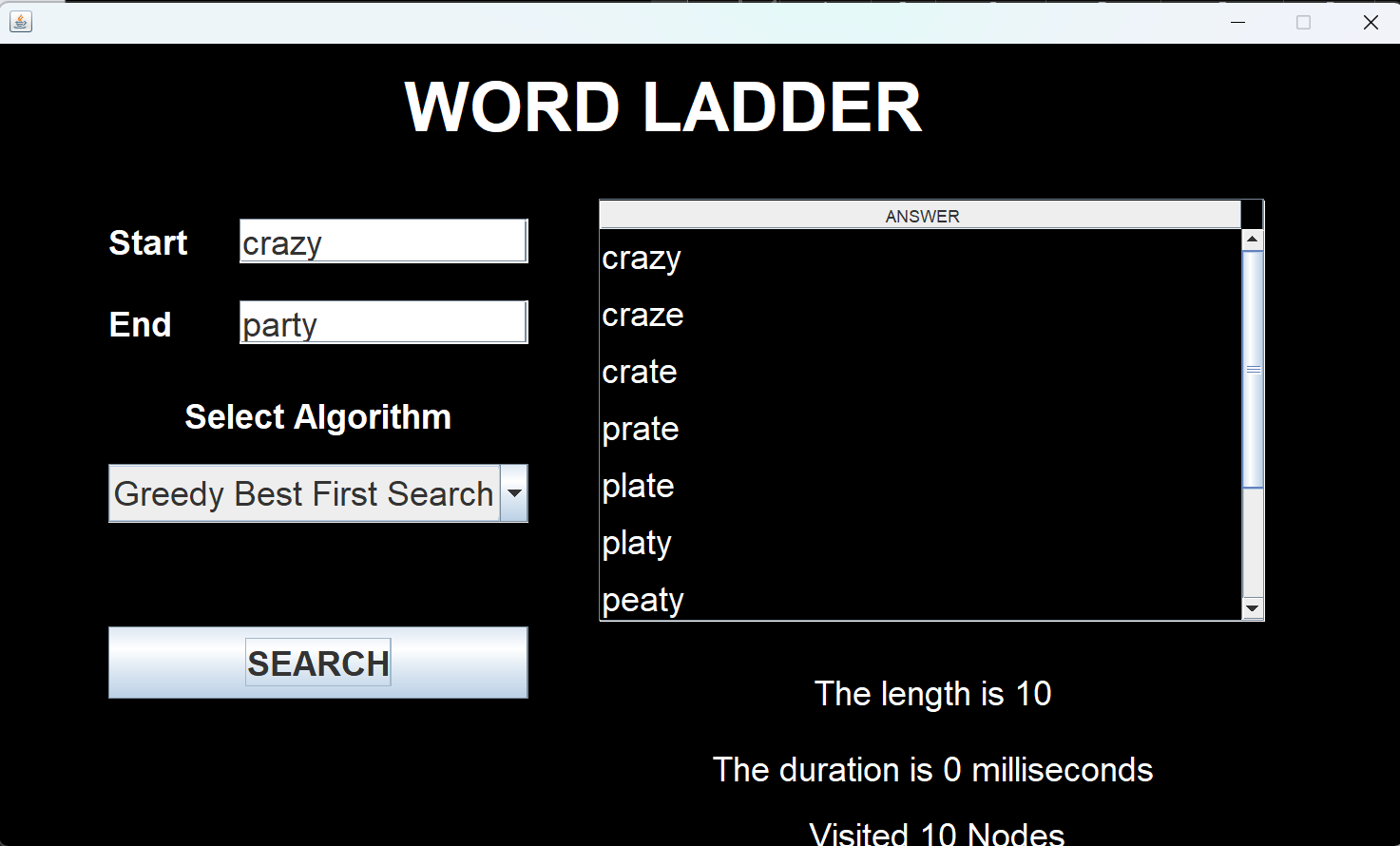


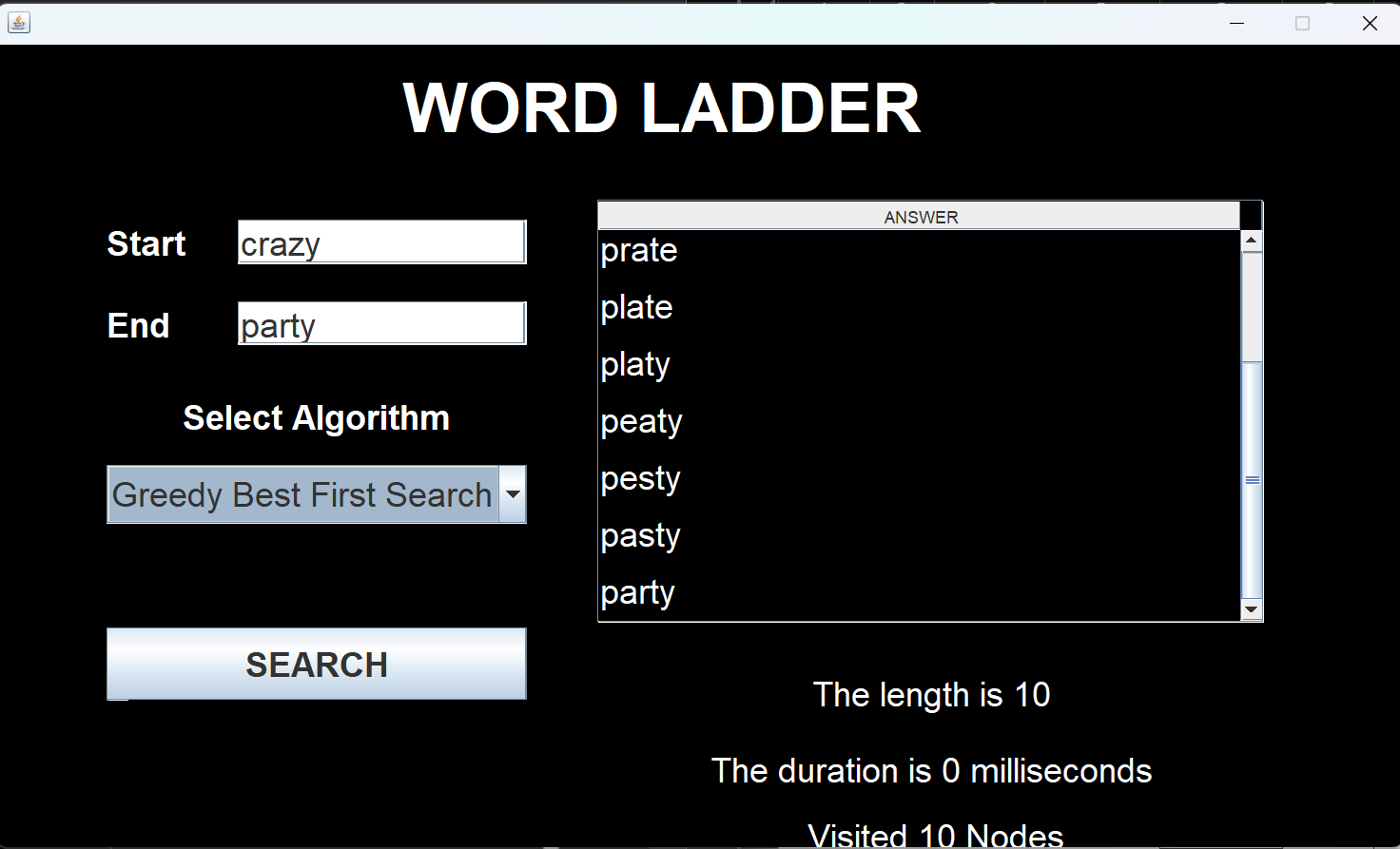
1. Crazy – Party
2. UCS



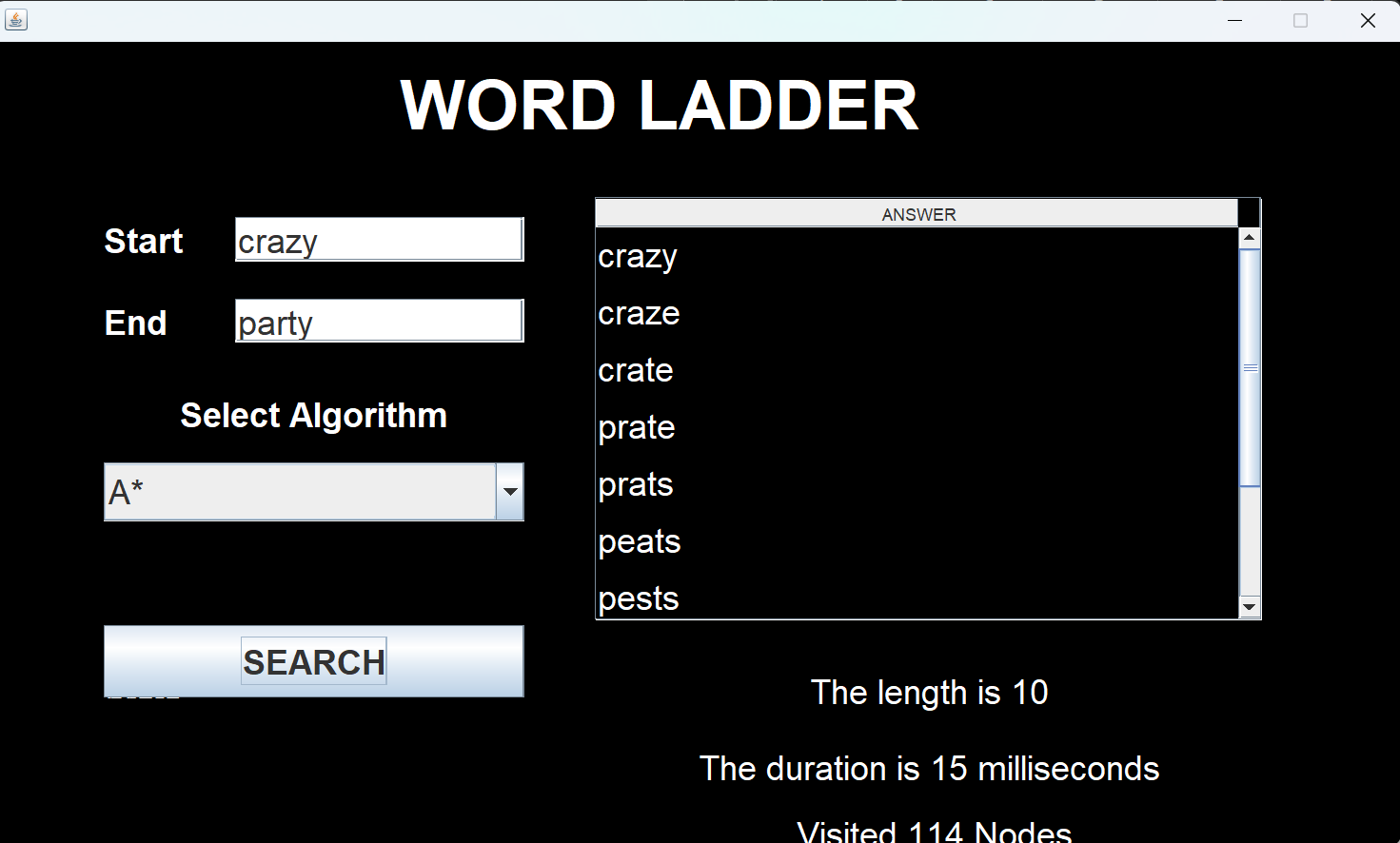


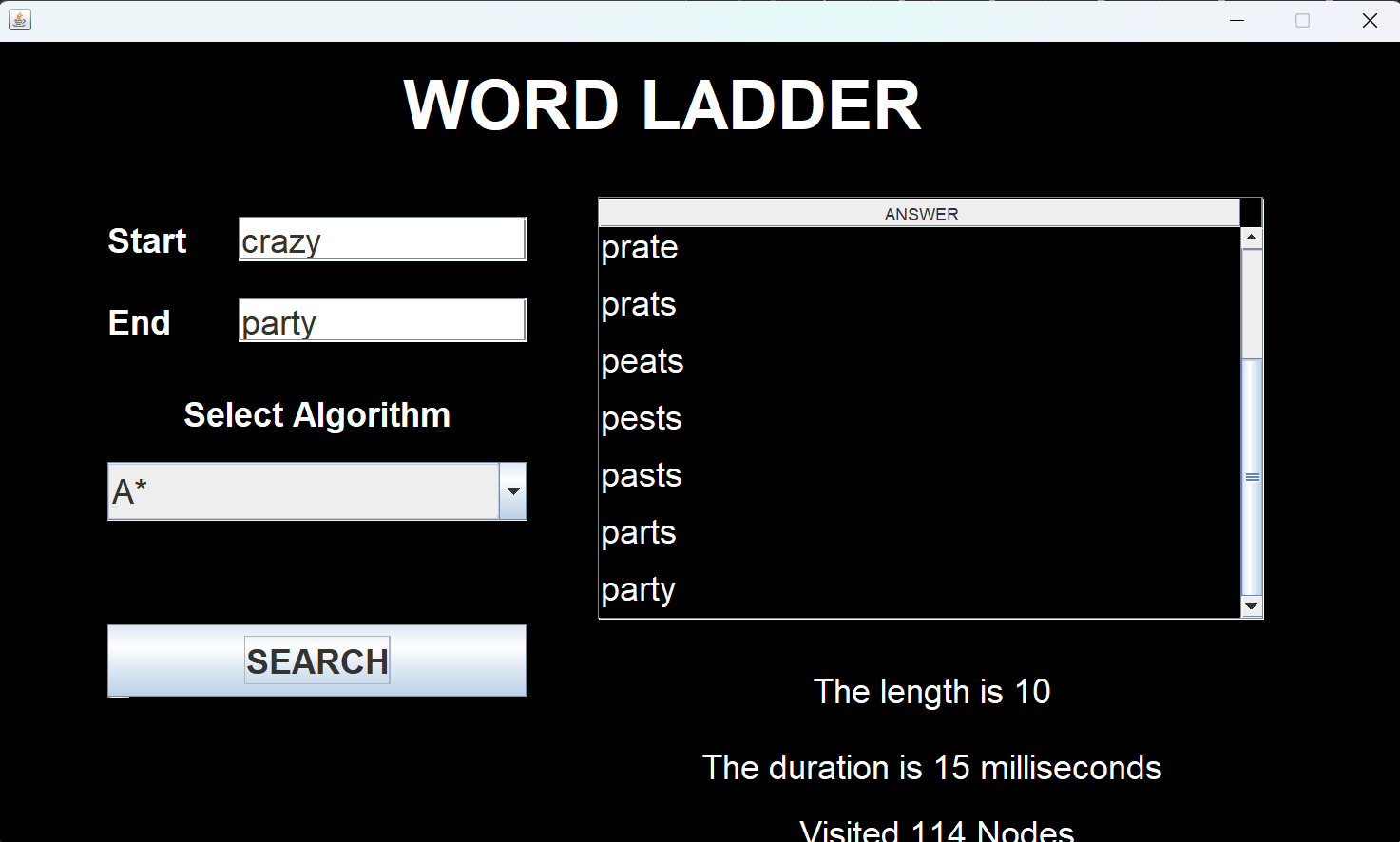
1. GBFS



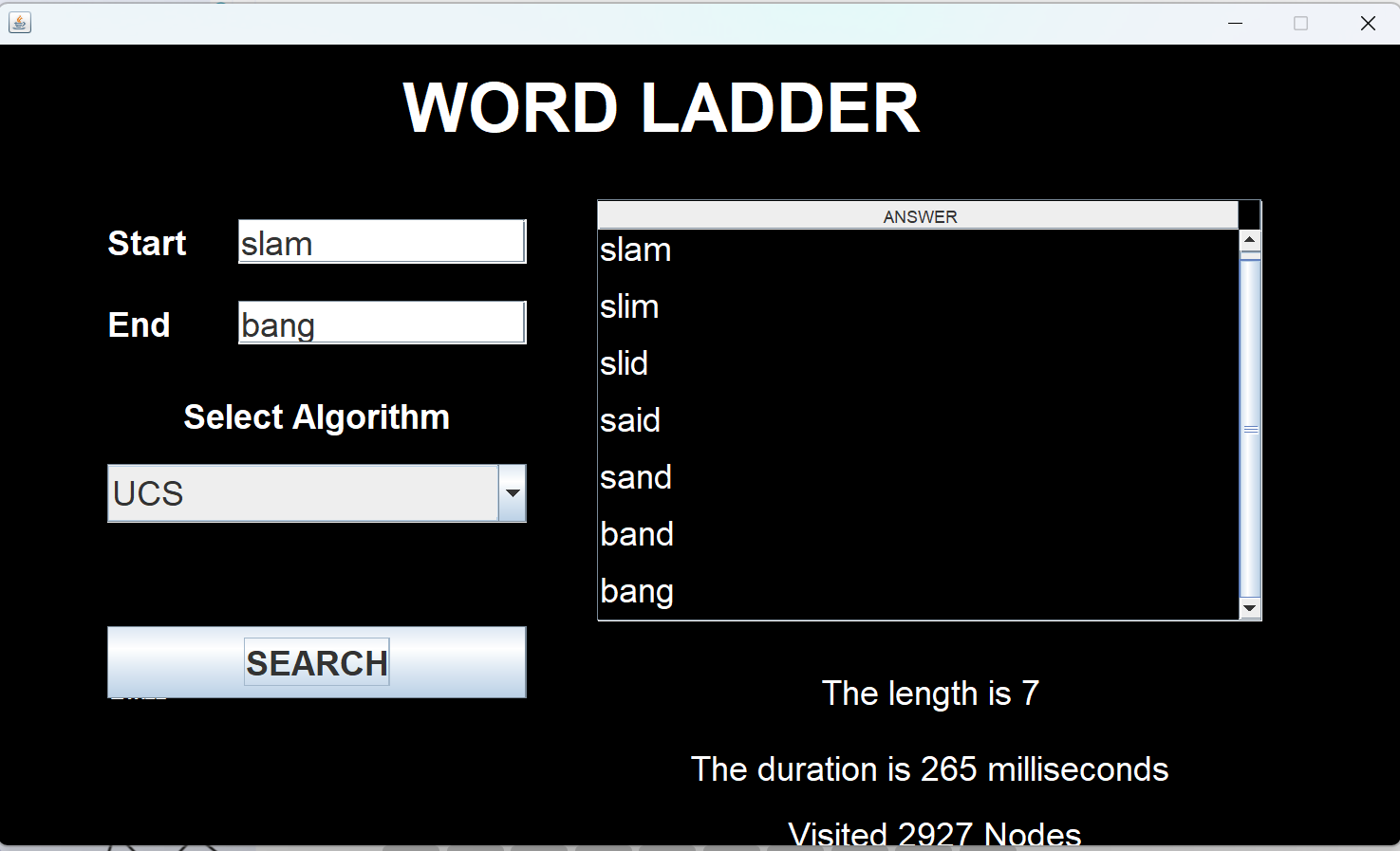


1. A Star

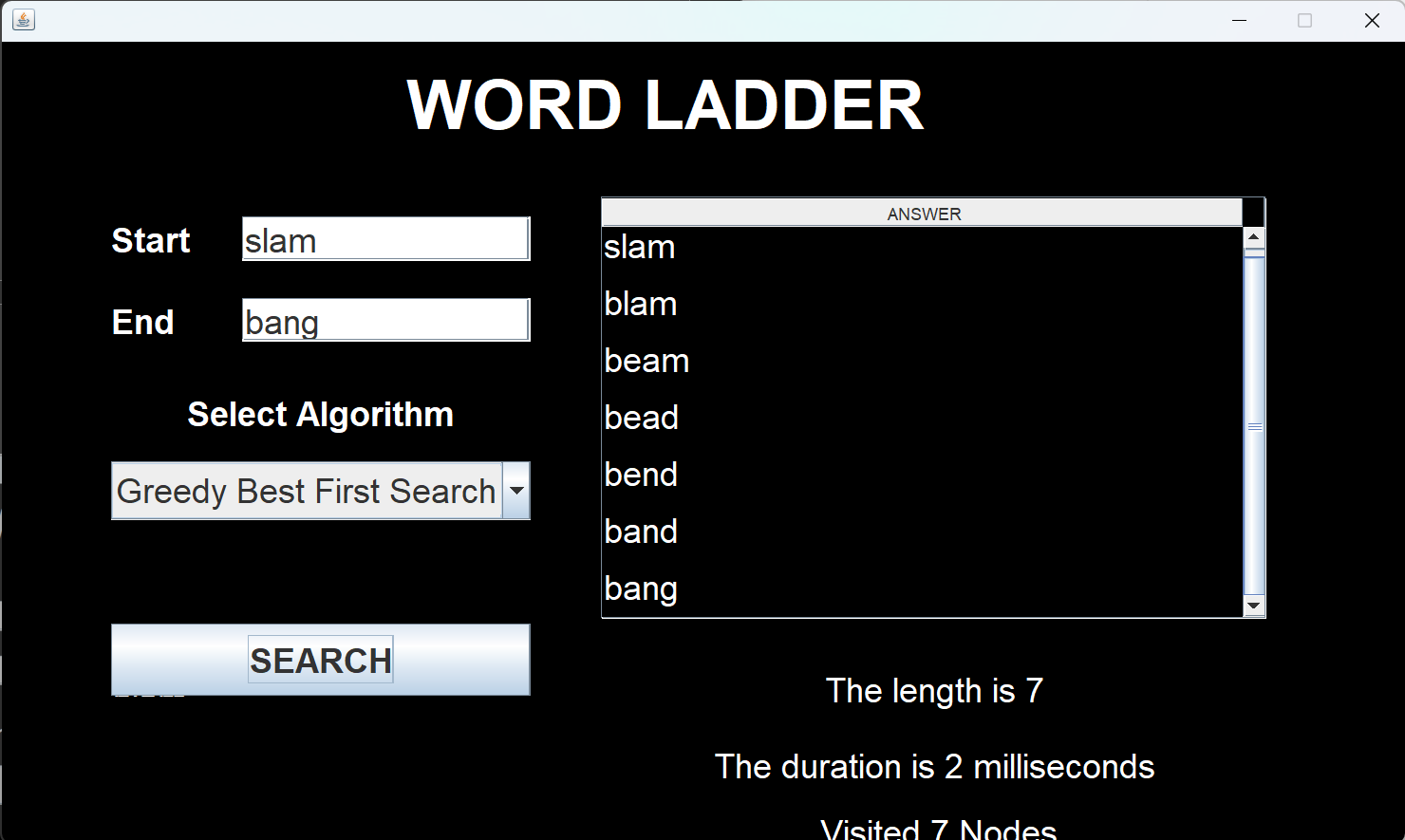




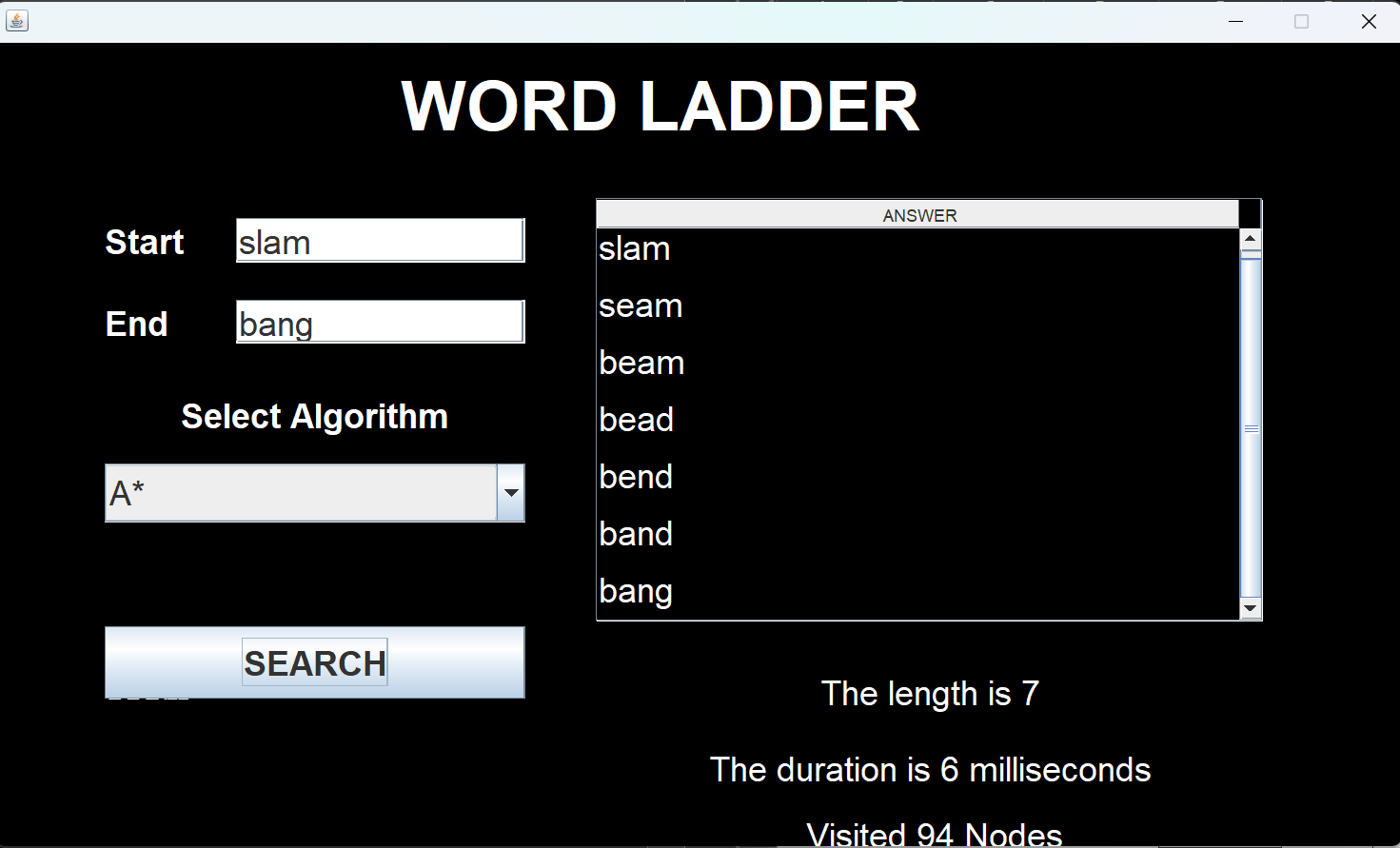
1. Slam – Bang
2. UCS



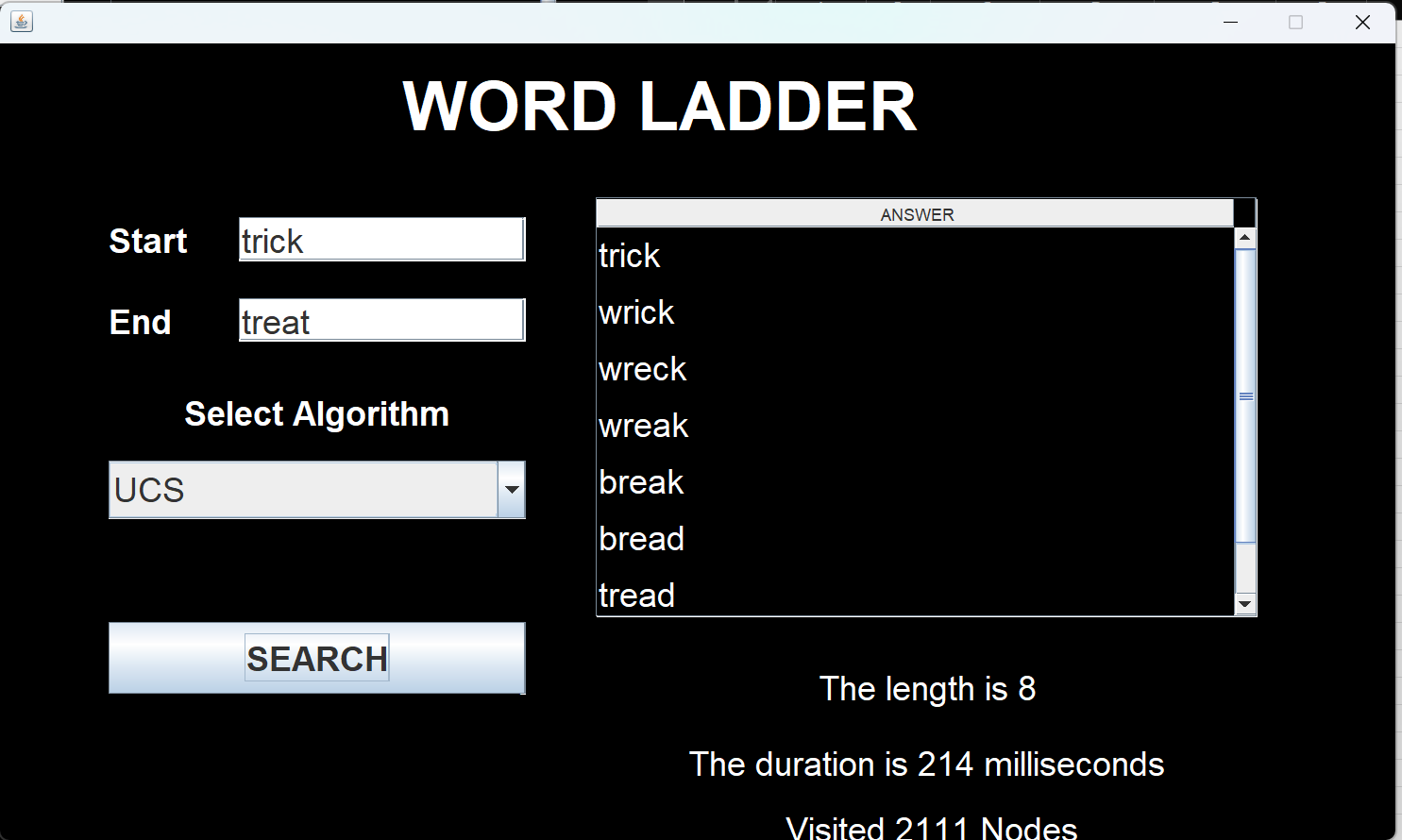
1. GBFS

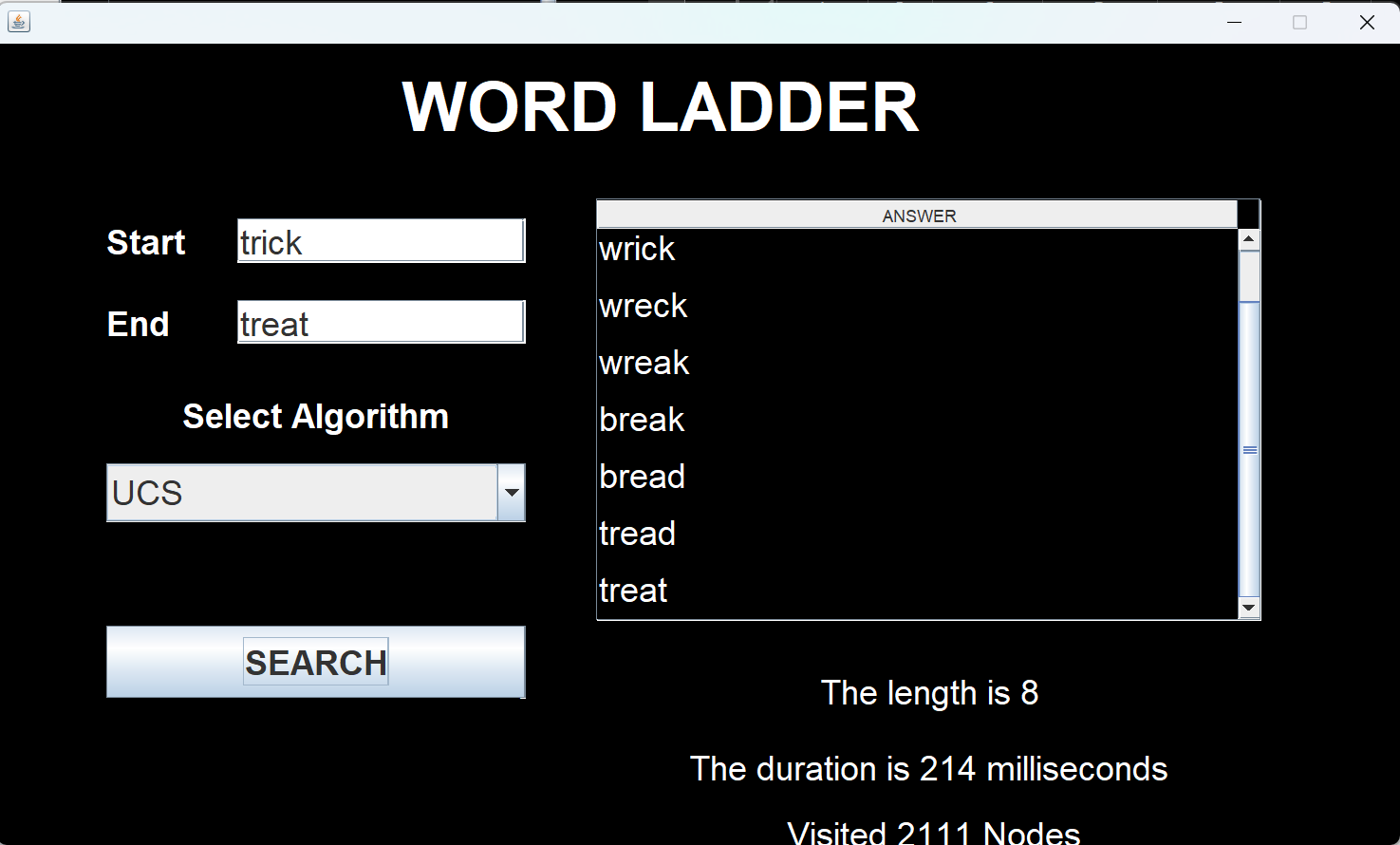


1. A Star

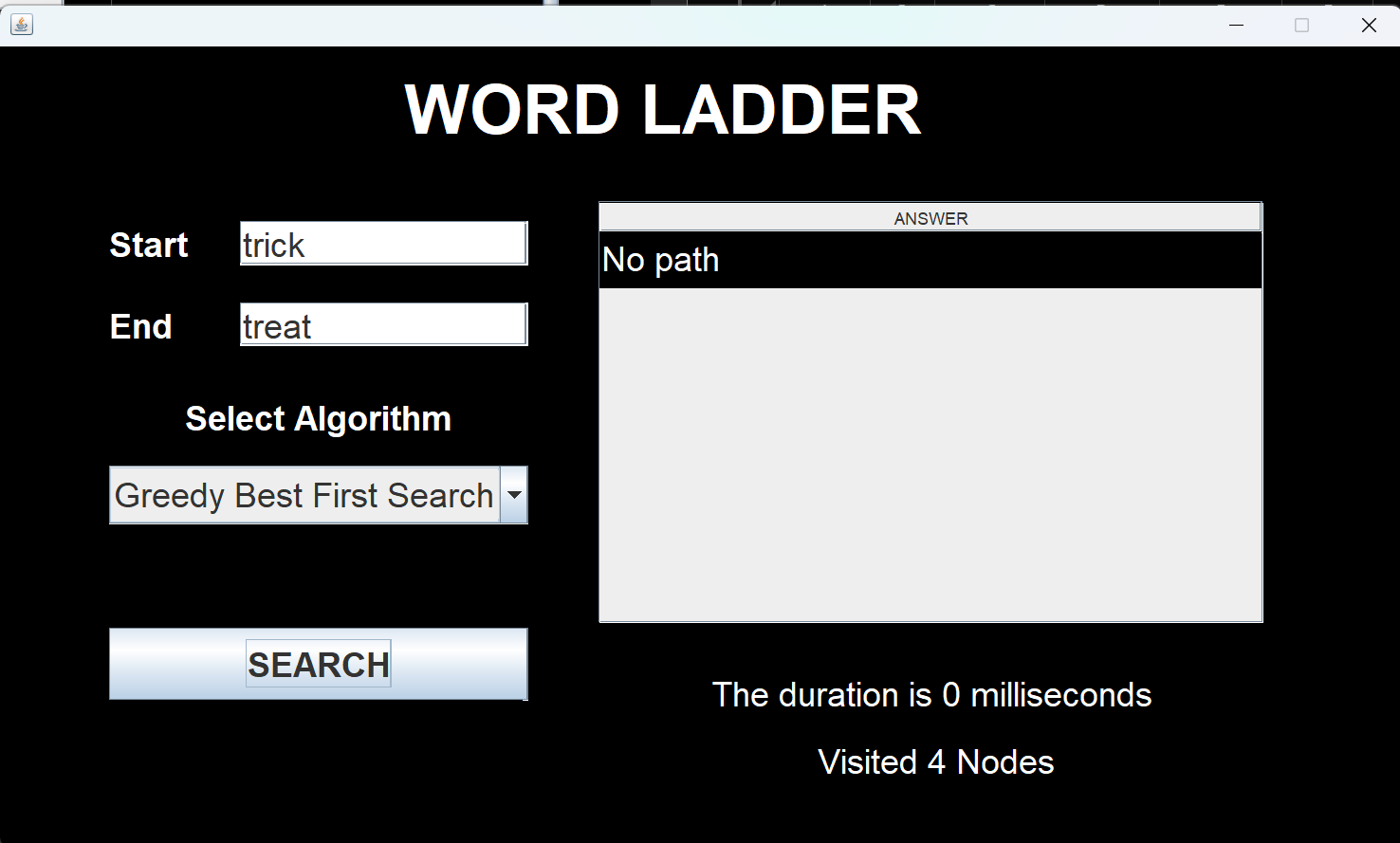


1. Trick – Treat
2. UCS

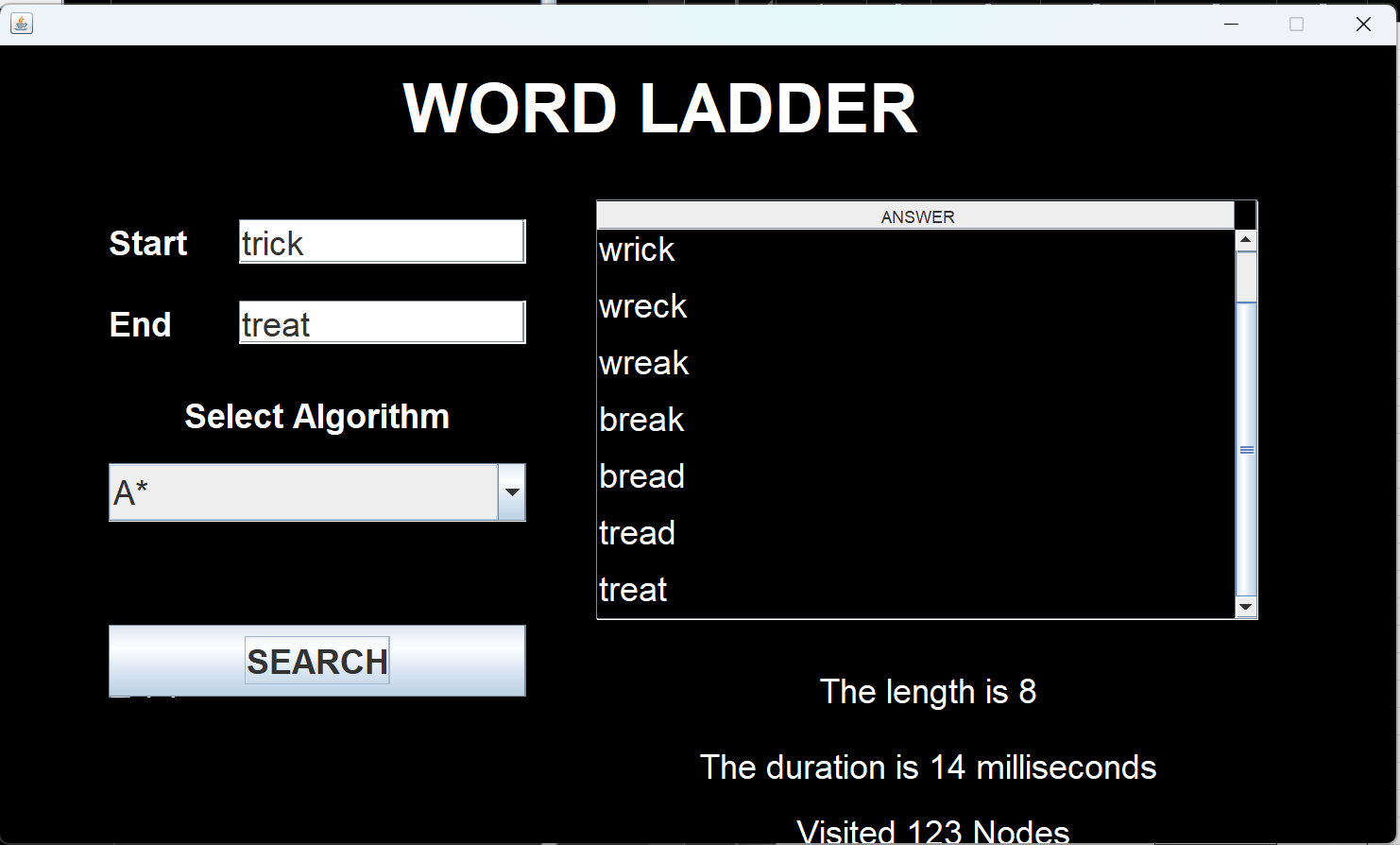


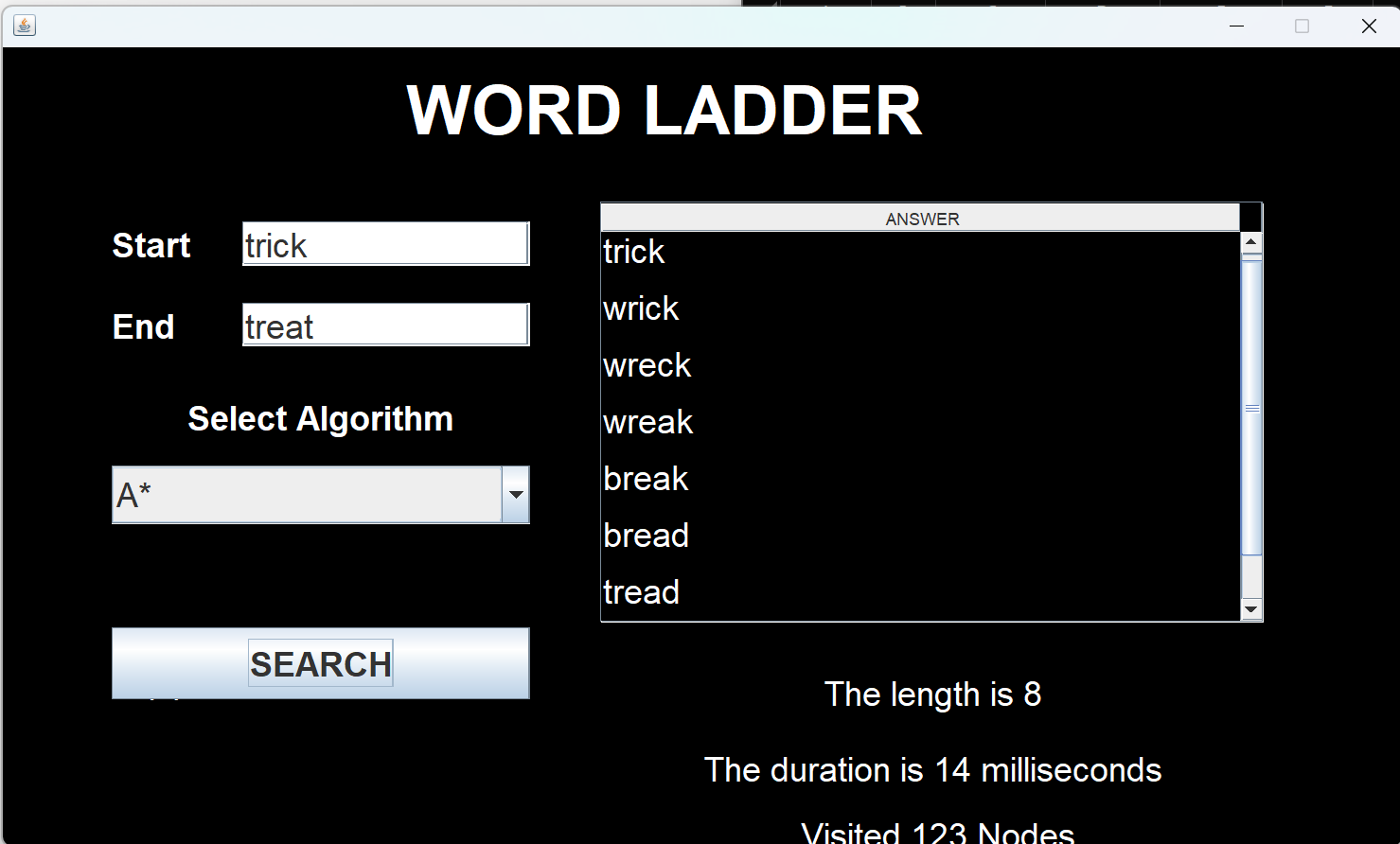


1. GBFS

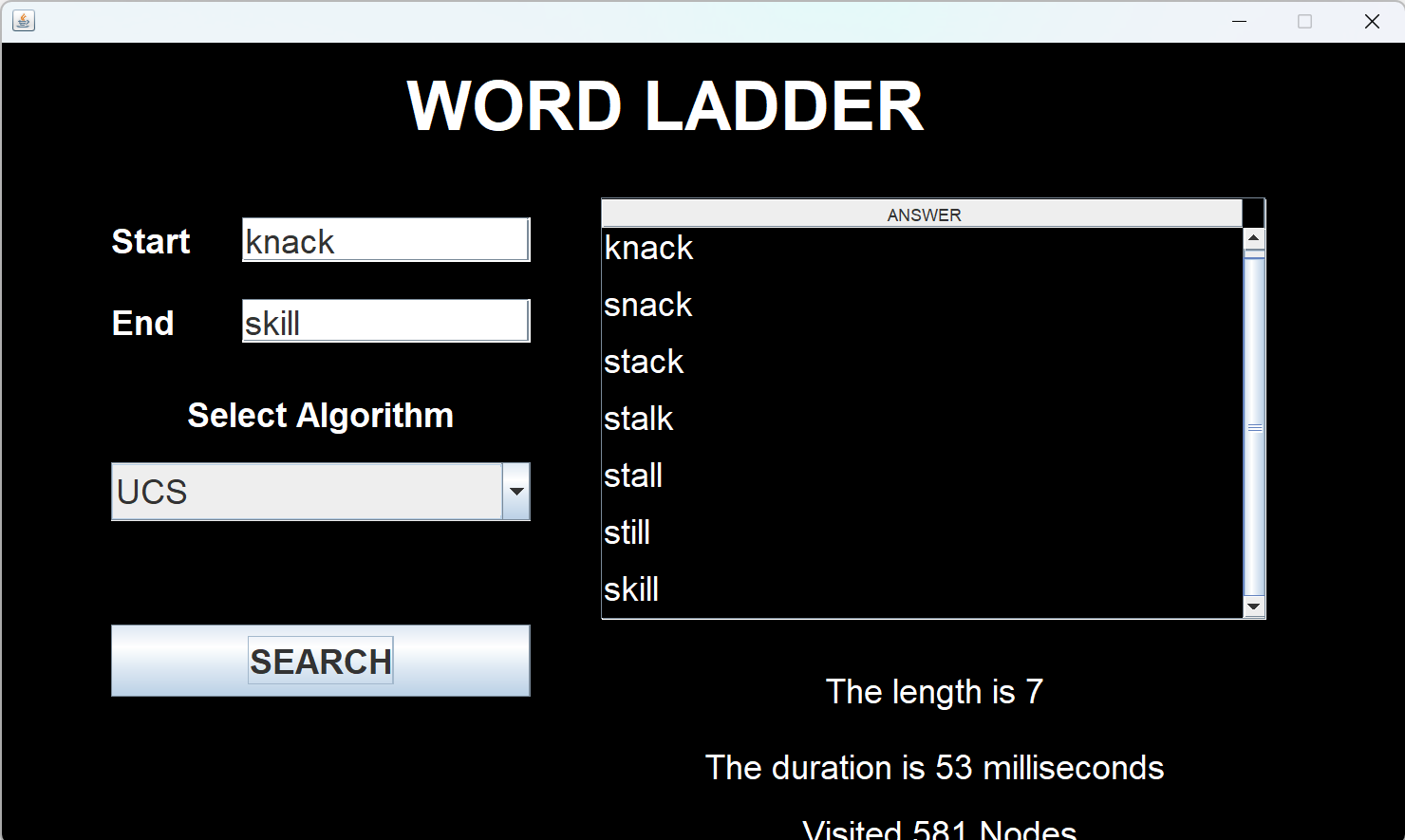


1. A Star

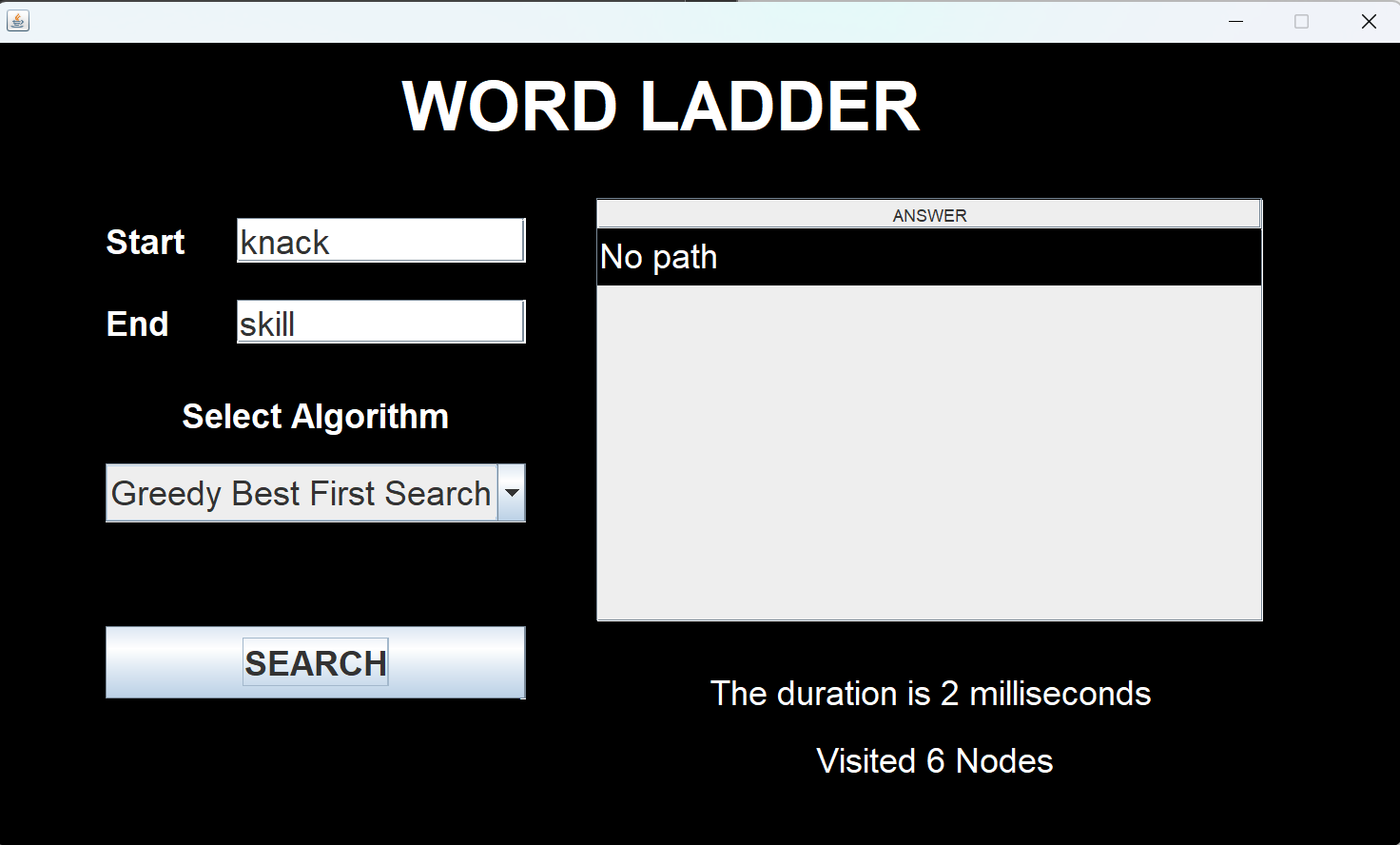




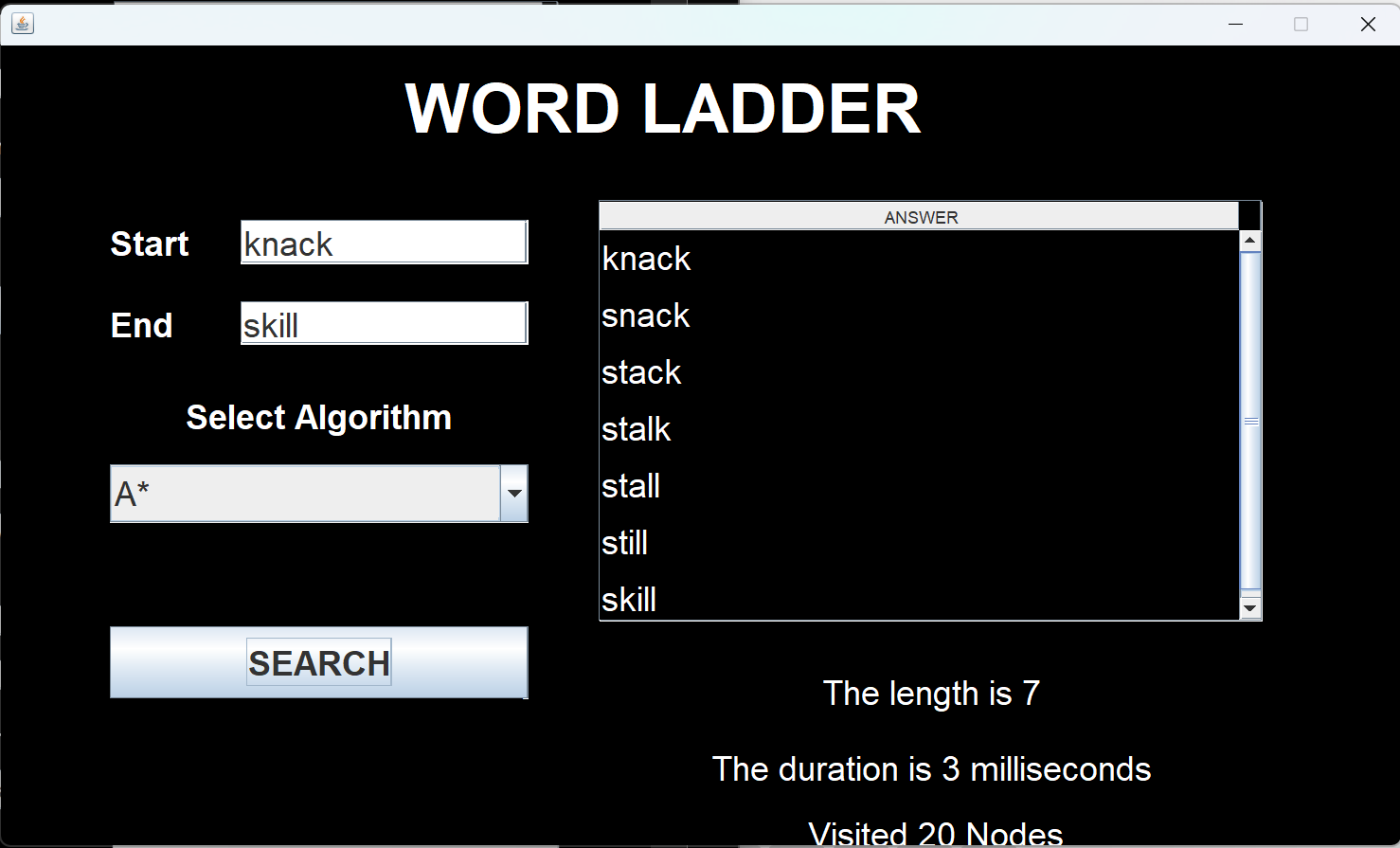
1. Knack – Skill
2. UCS



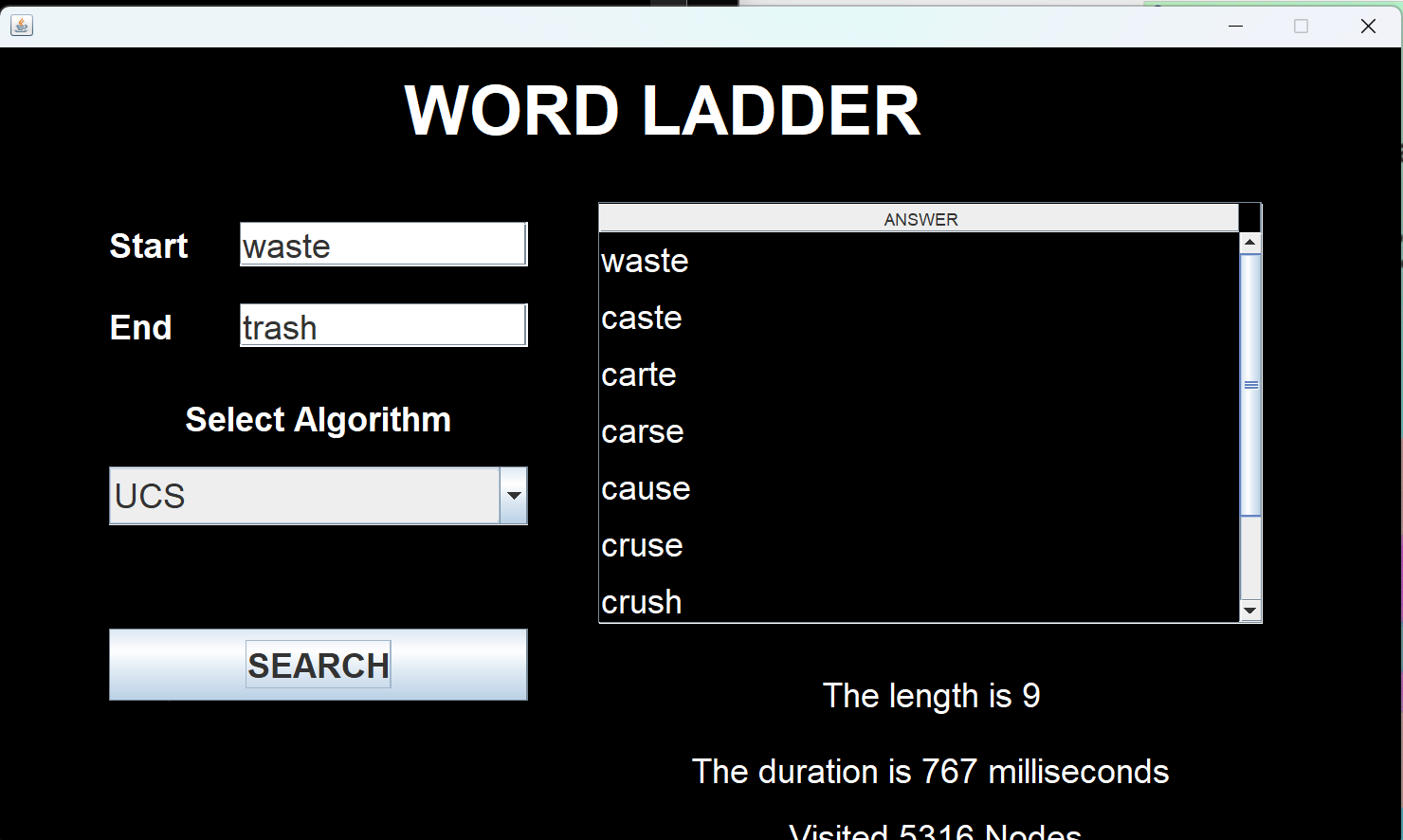
1. GBFS

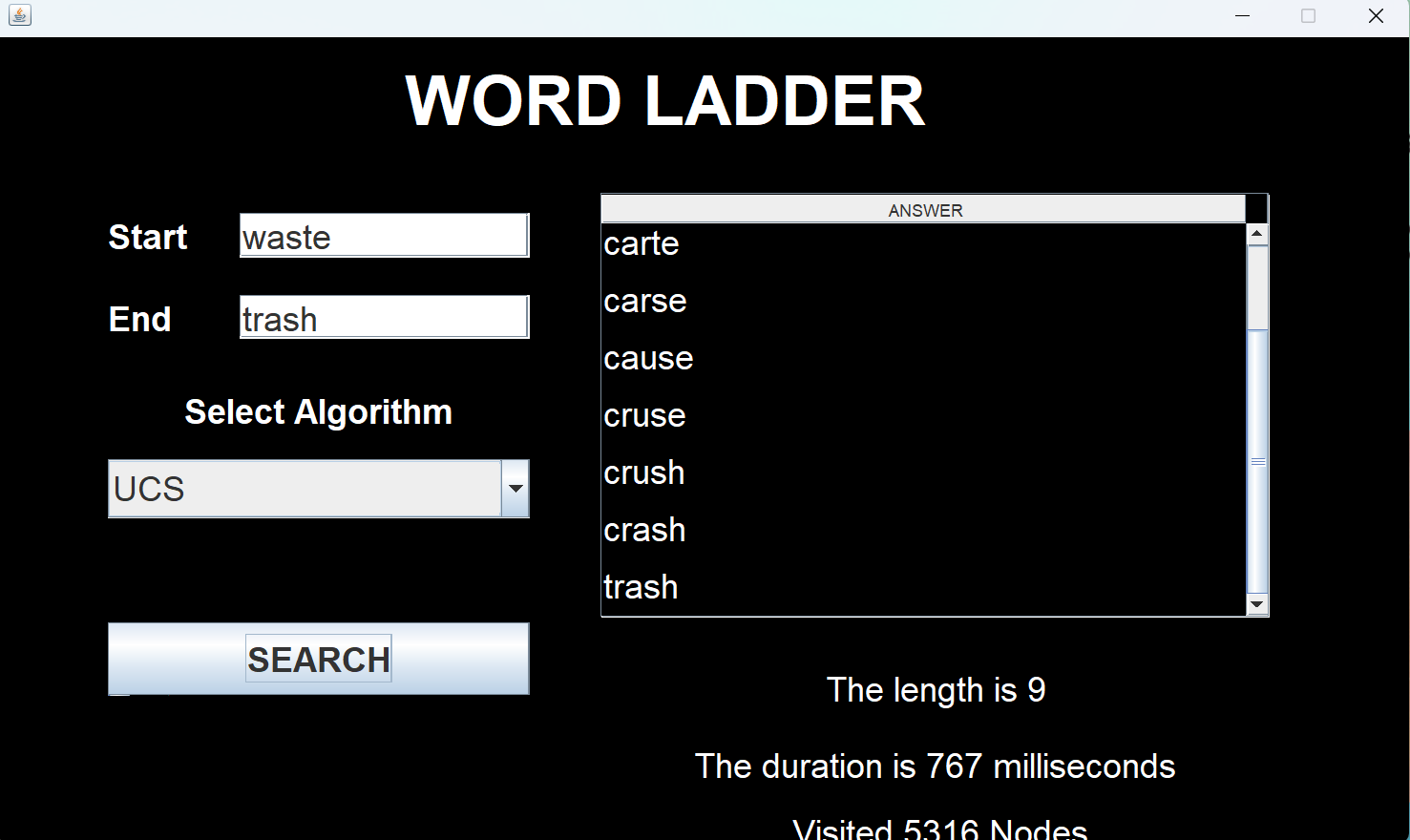


1. A Star

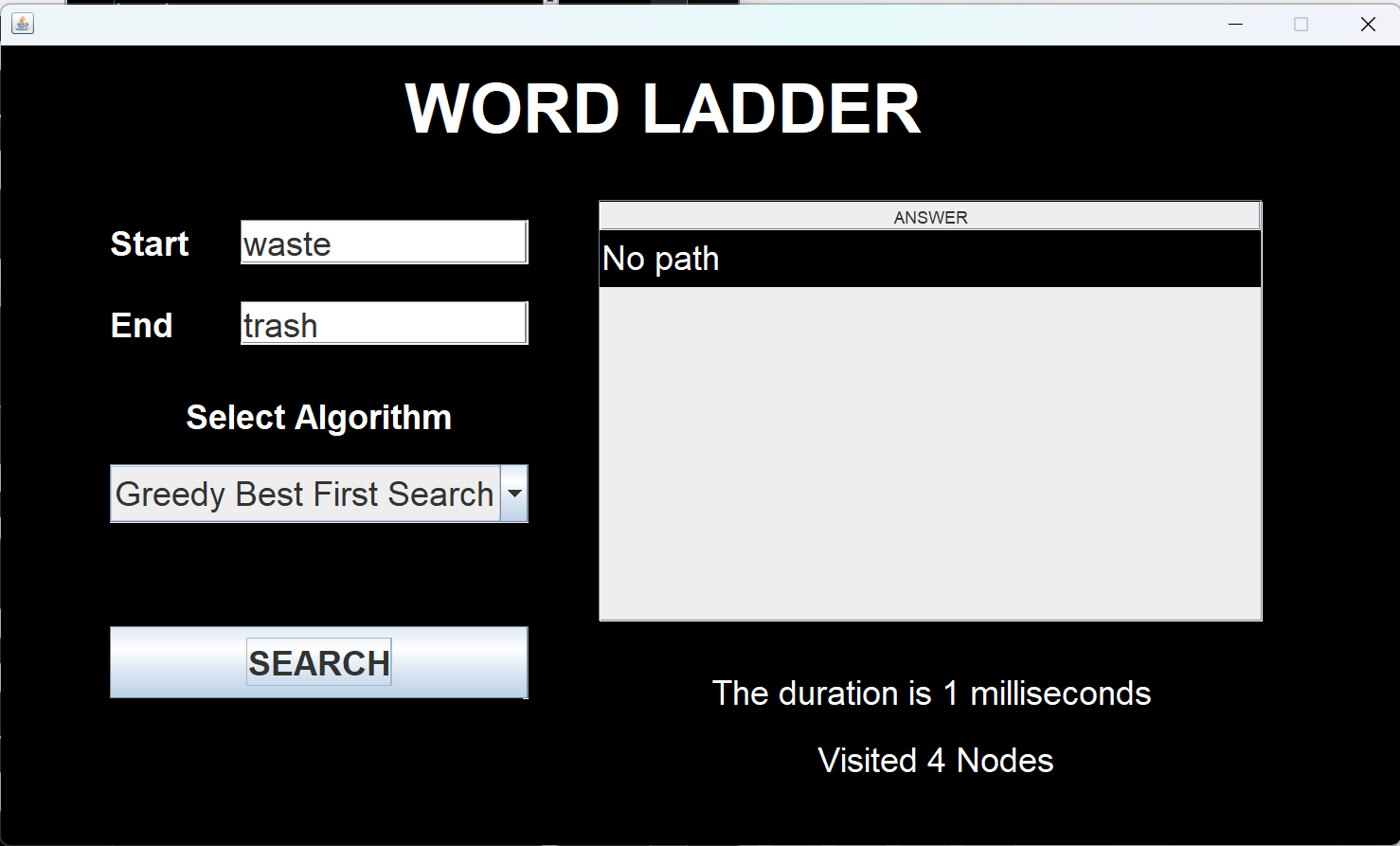


1. Waste – Trash
2. UCS

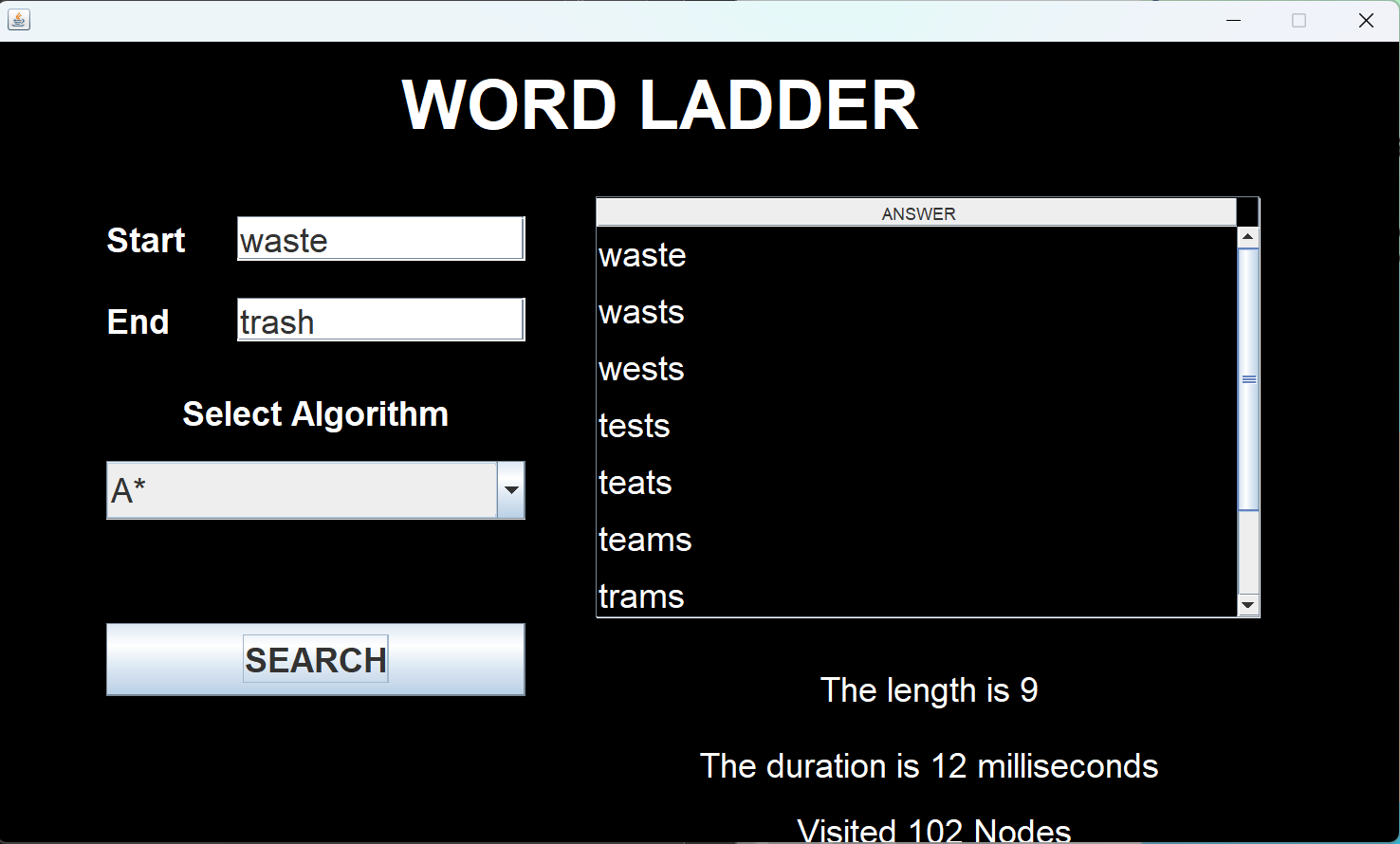


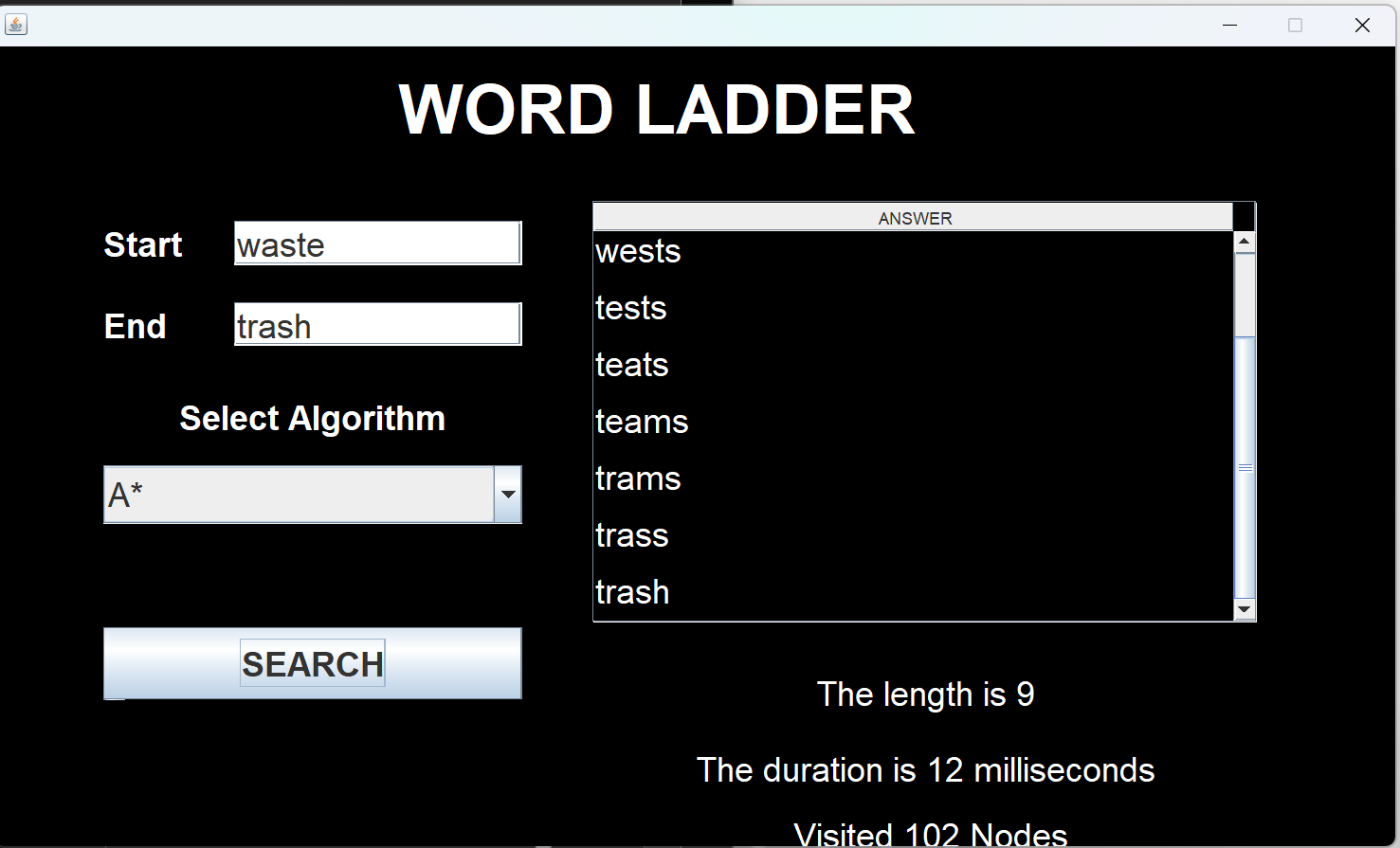


1. GBFS

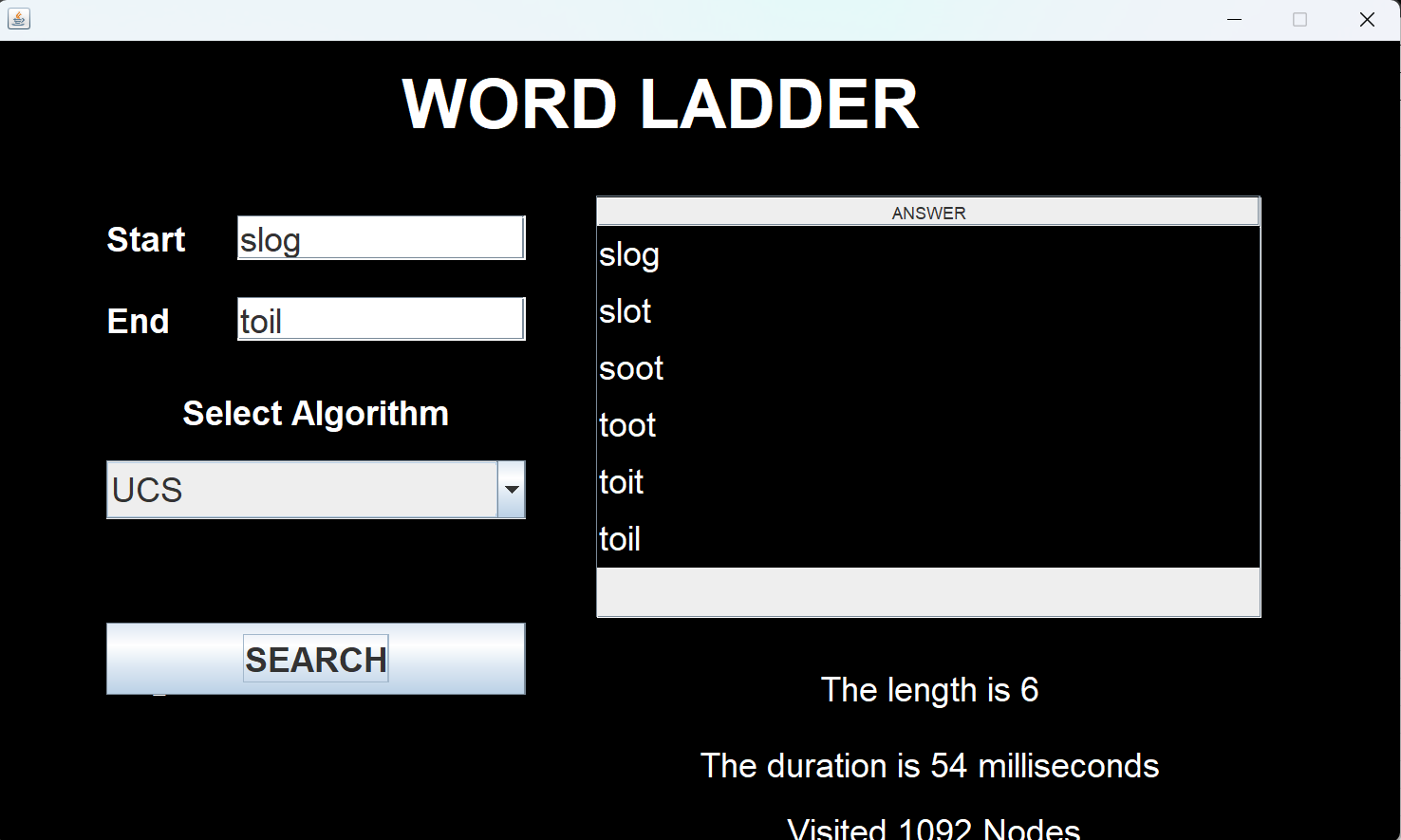


1. A Star

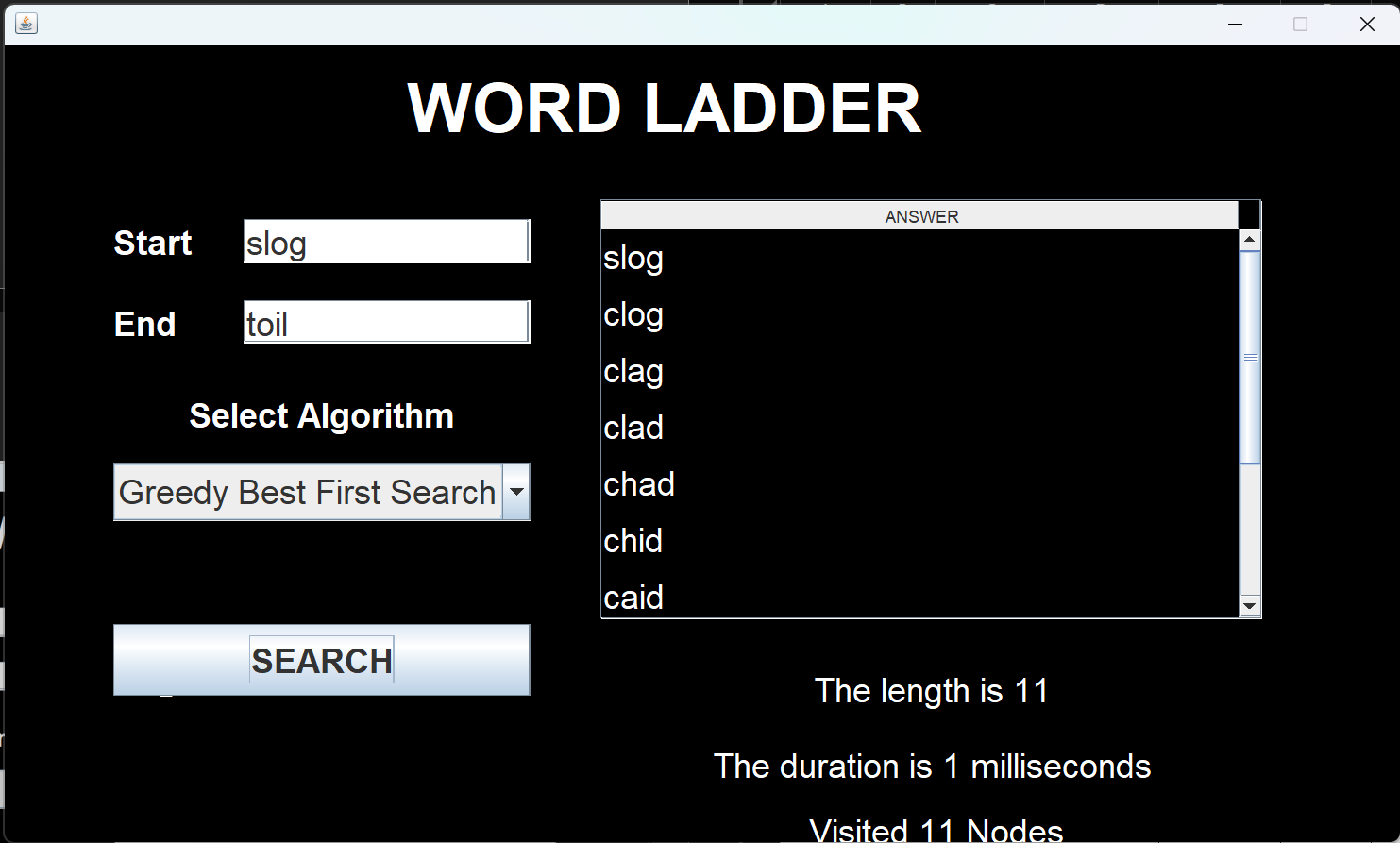




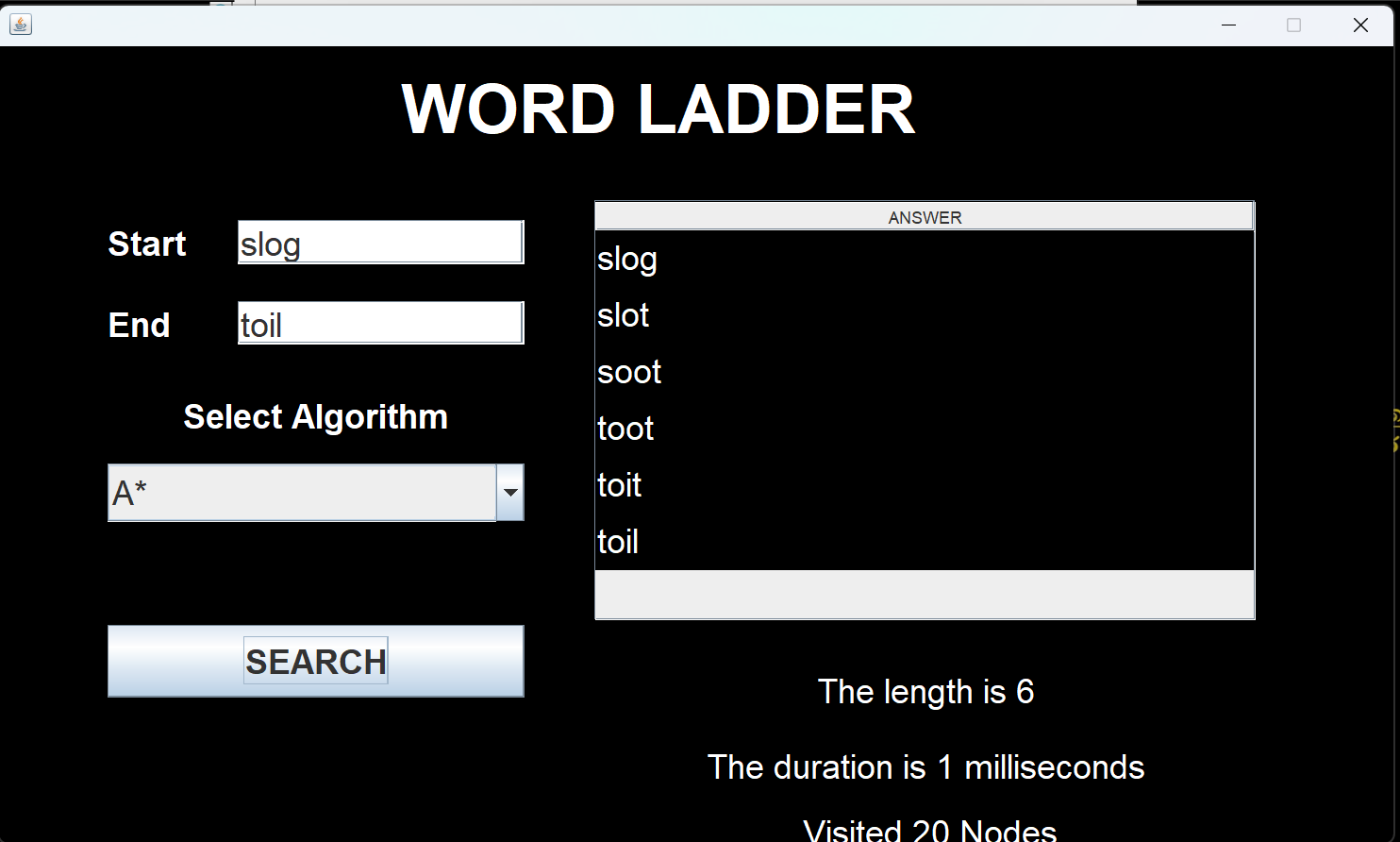
1. Slog – Toil
2. UCS



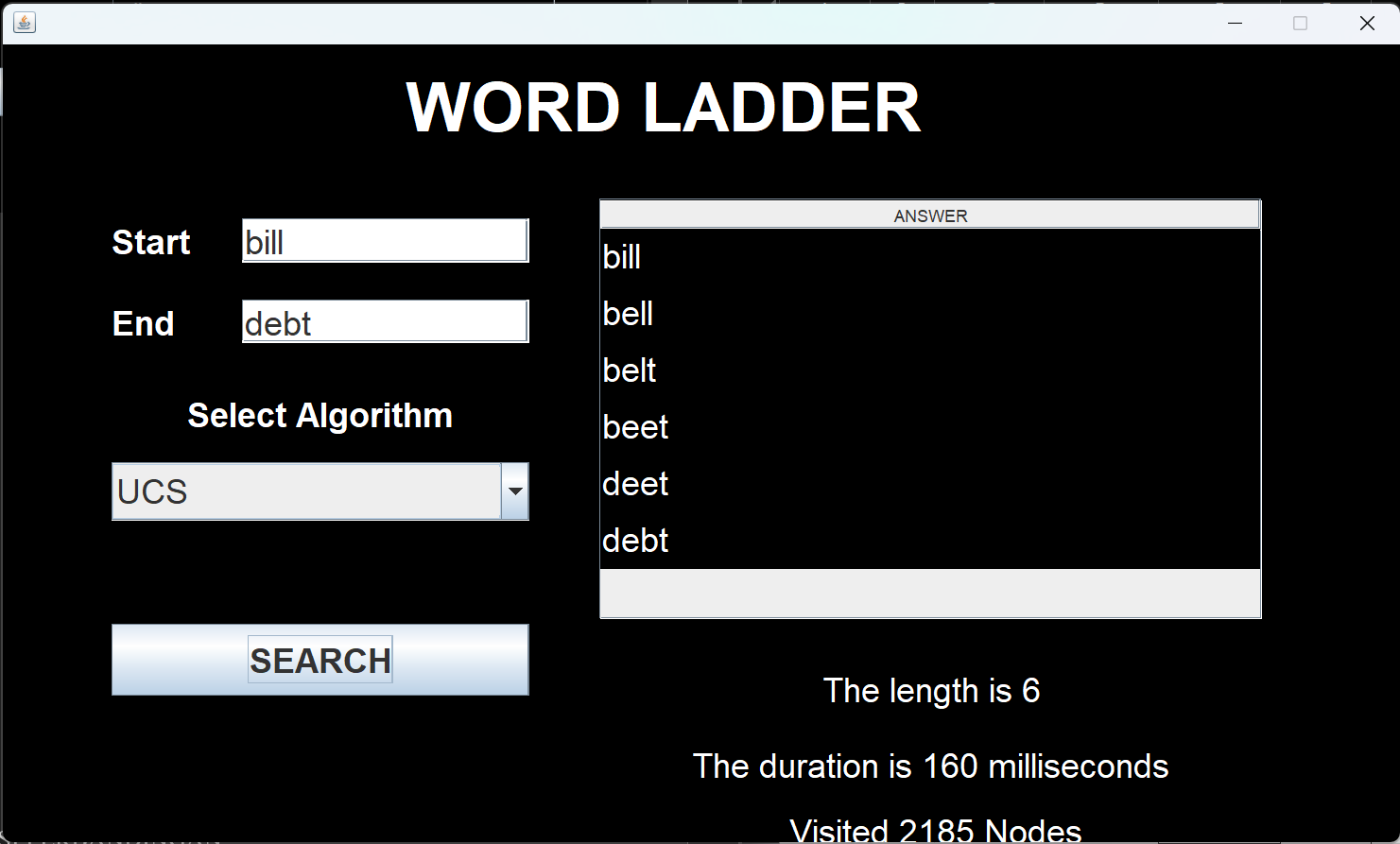
1. GBFS



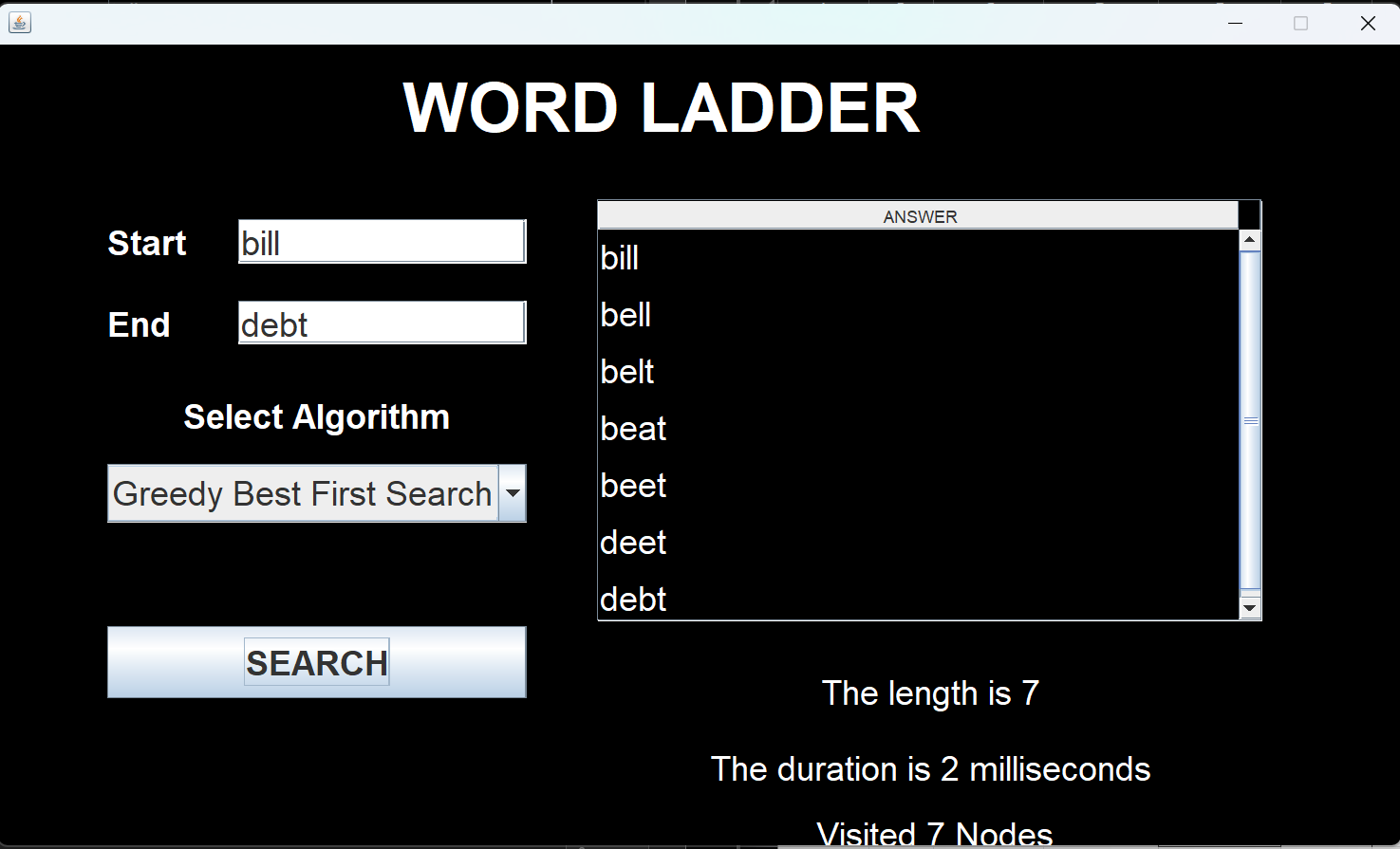
1. A Star



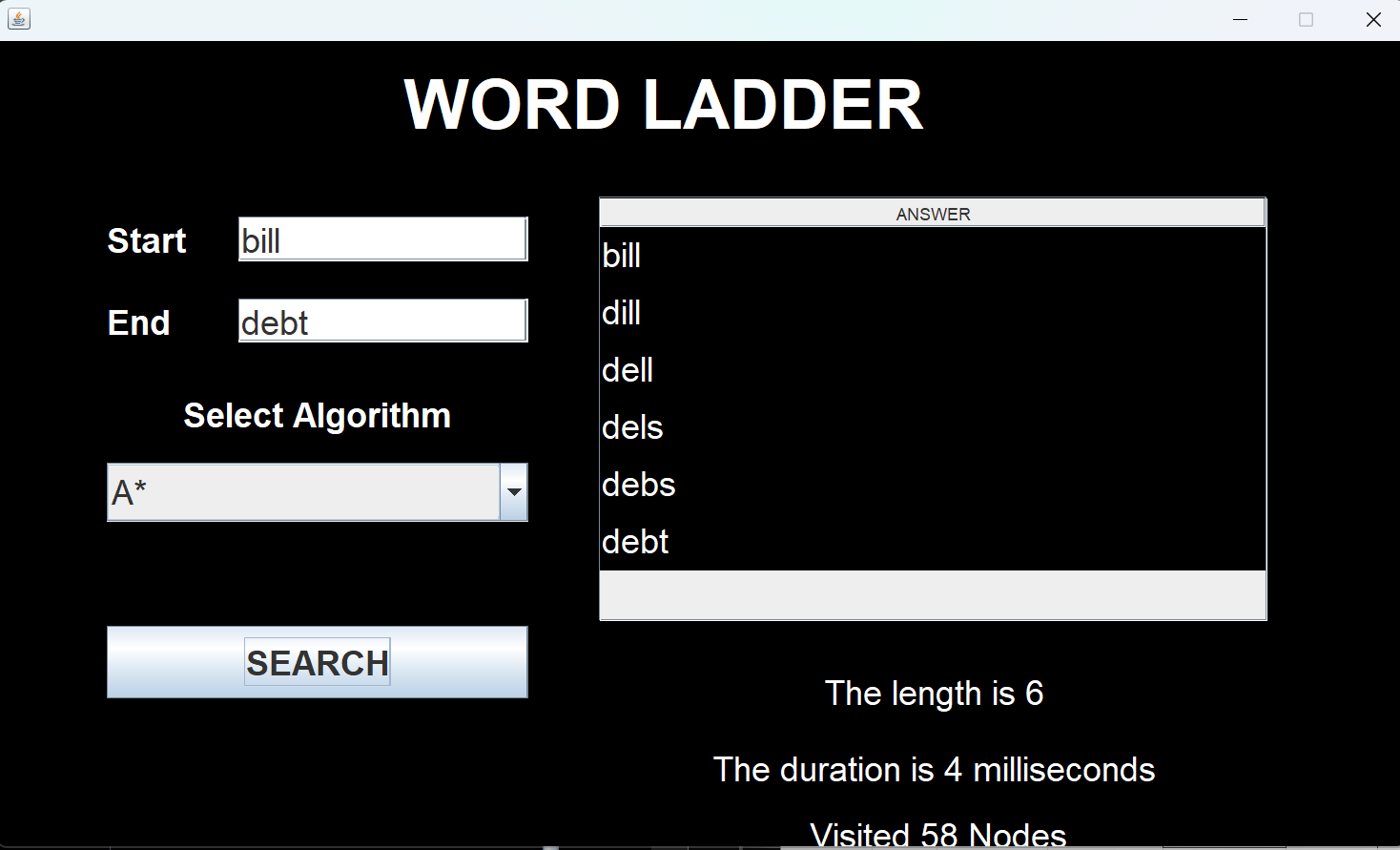
1. Bill – Debt
2. UCS



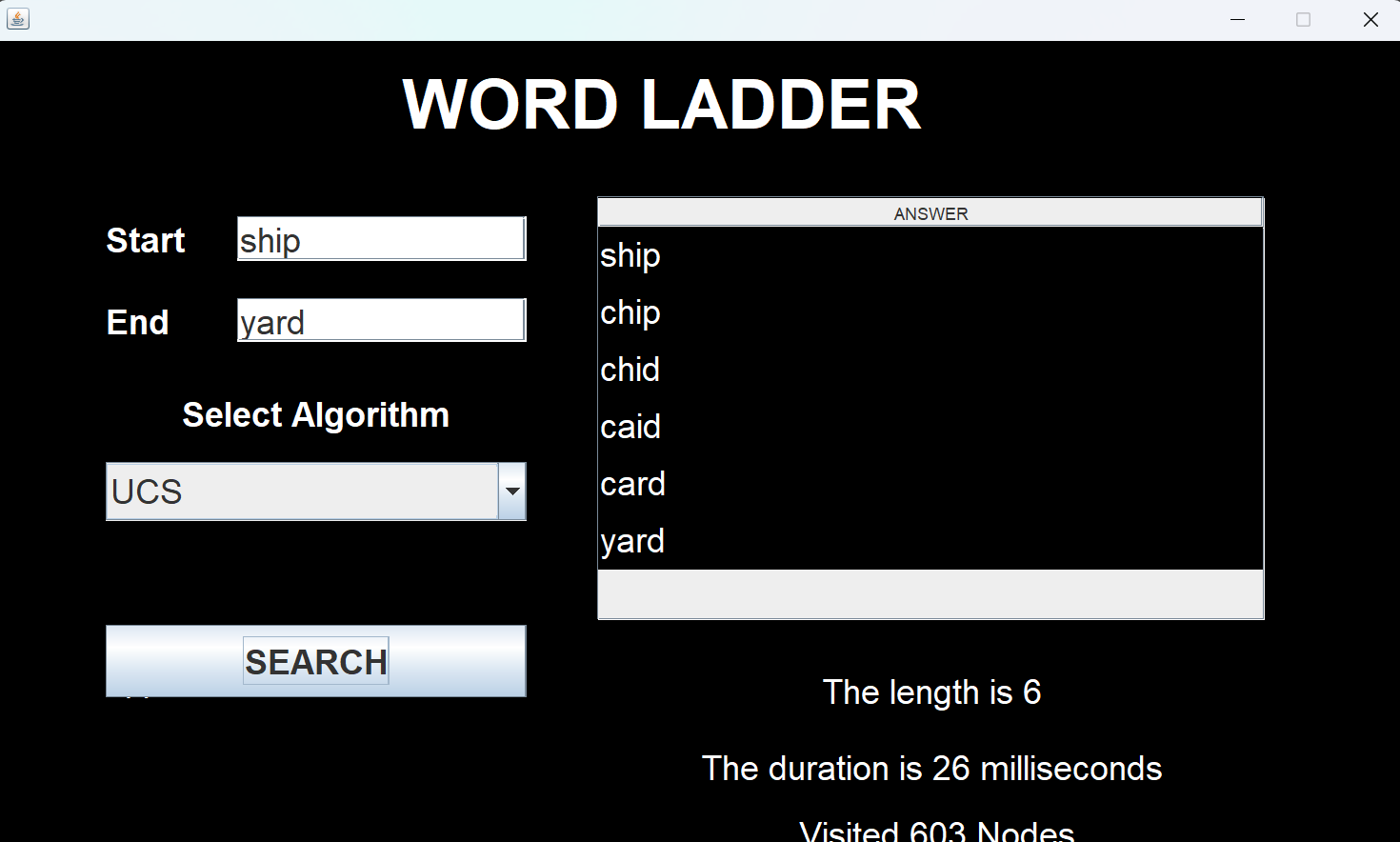
1. GBFS



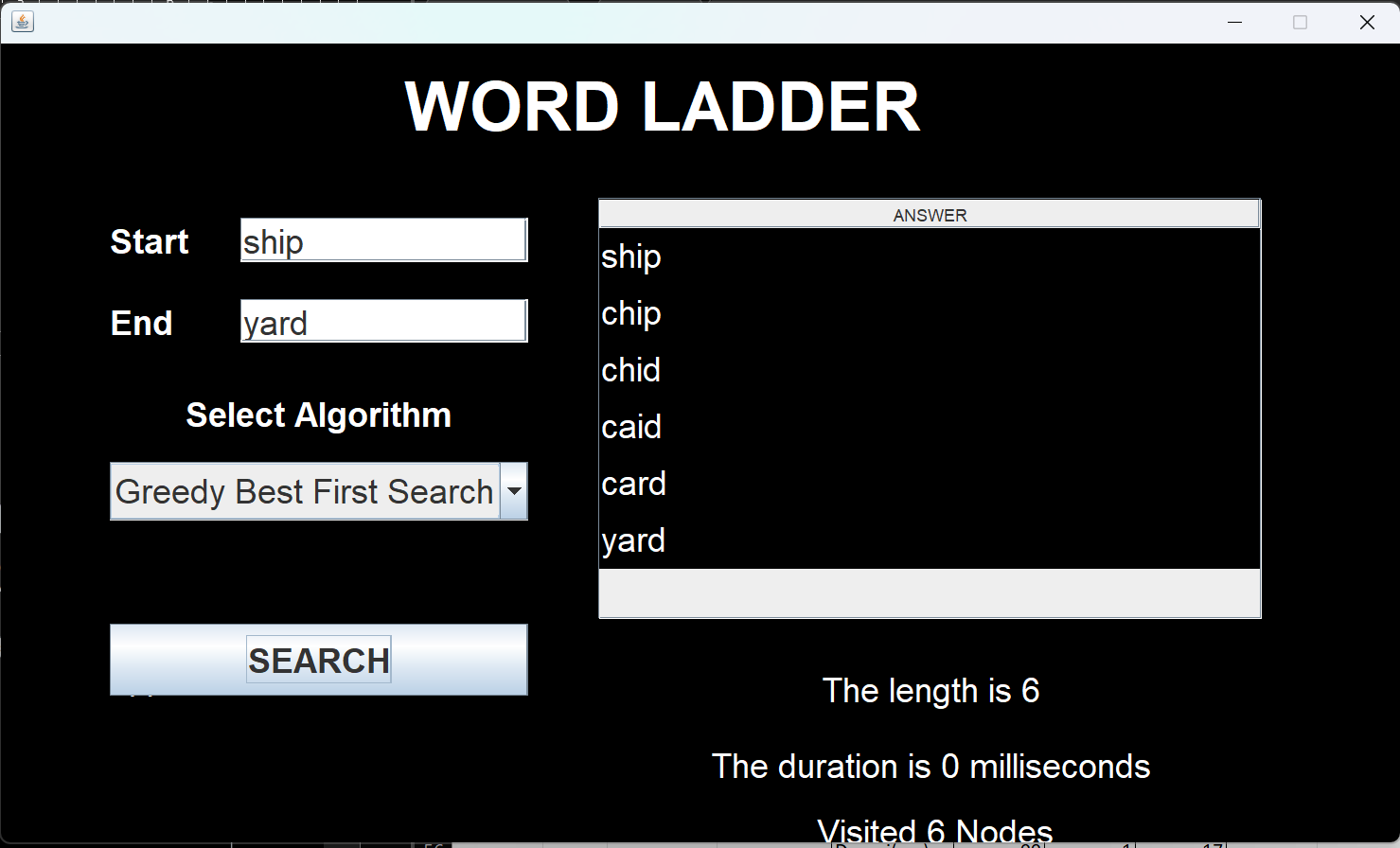
1. A Star



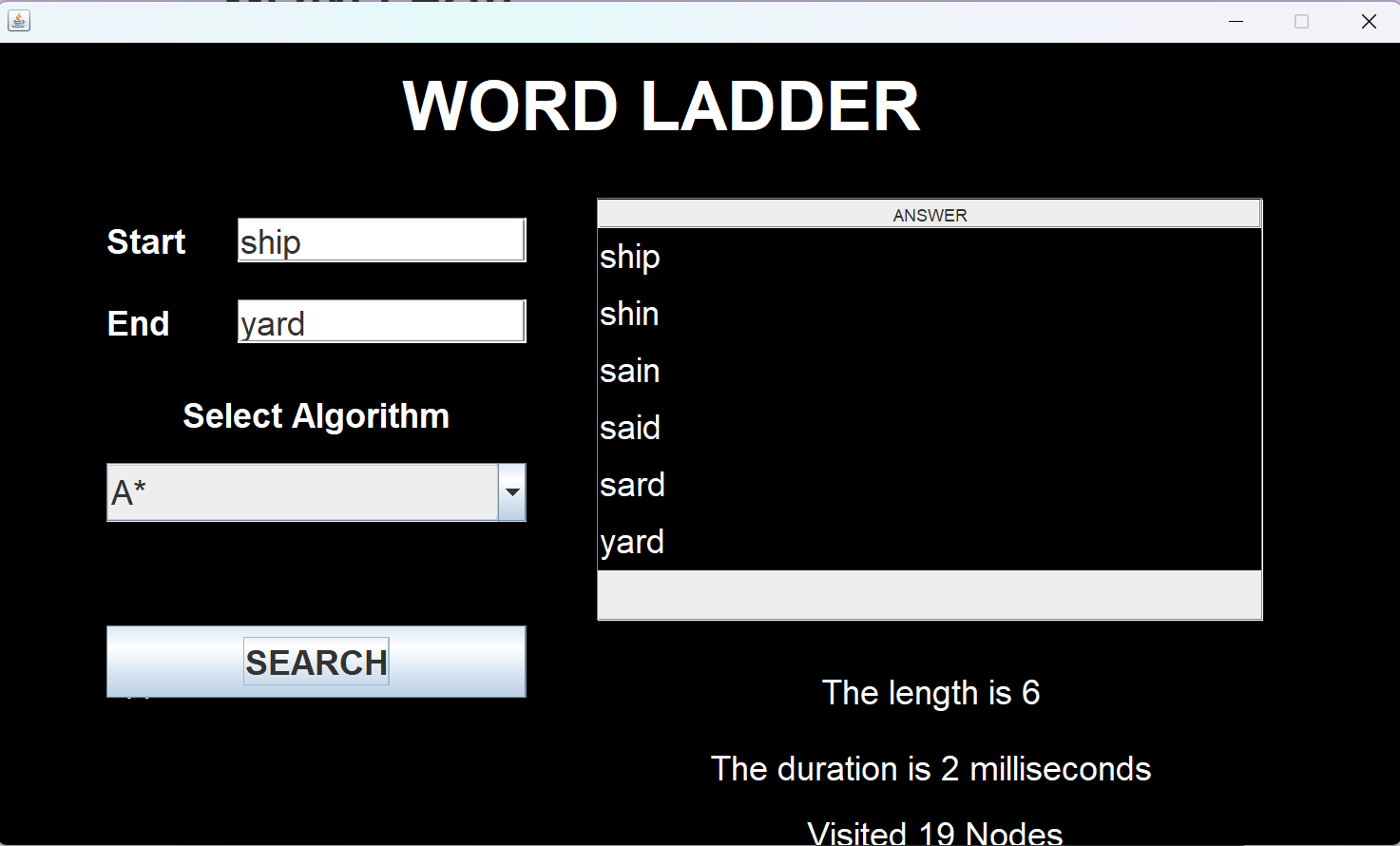
1. Ship – Yard
2. UCS



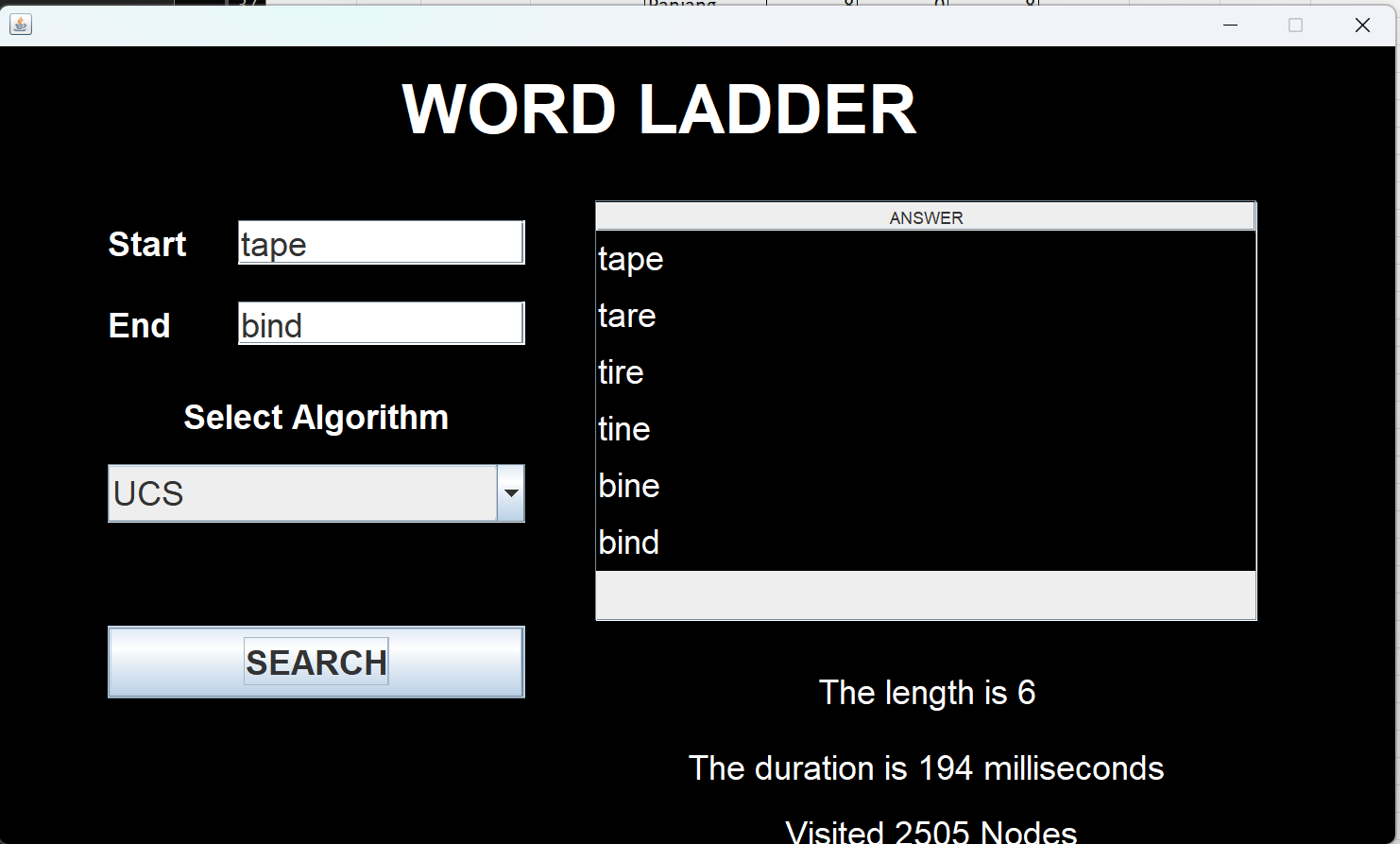
1. GBFS



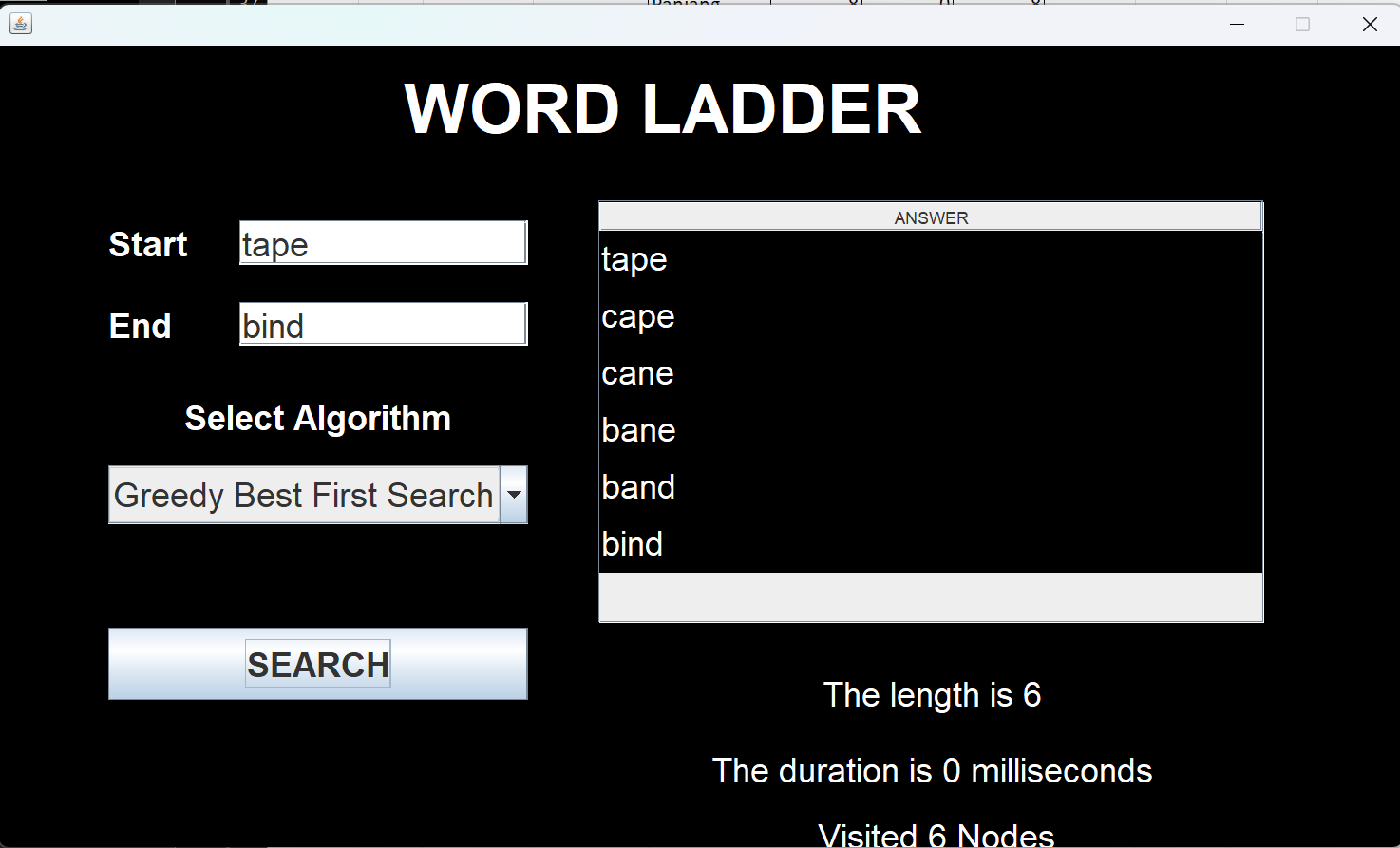
1. A Star



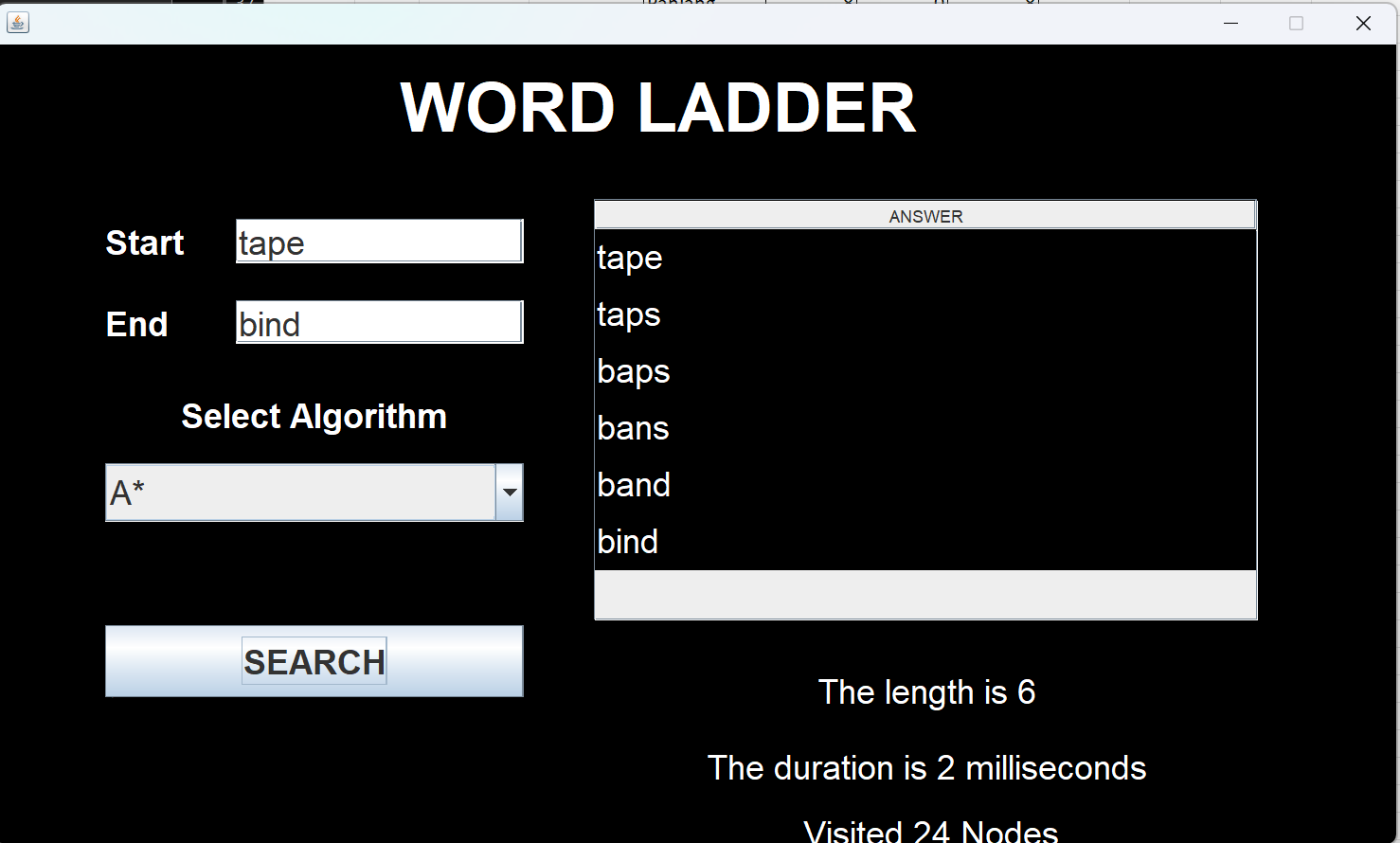
1. Tape – Bind
2. UCS



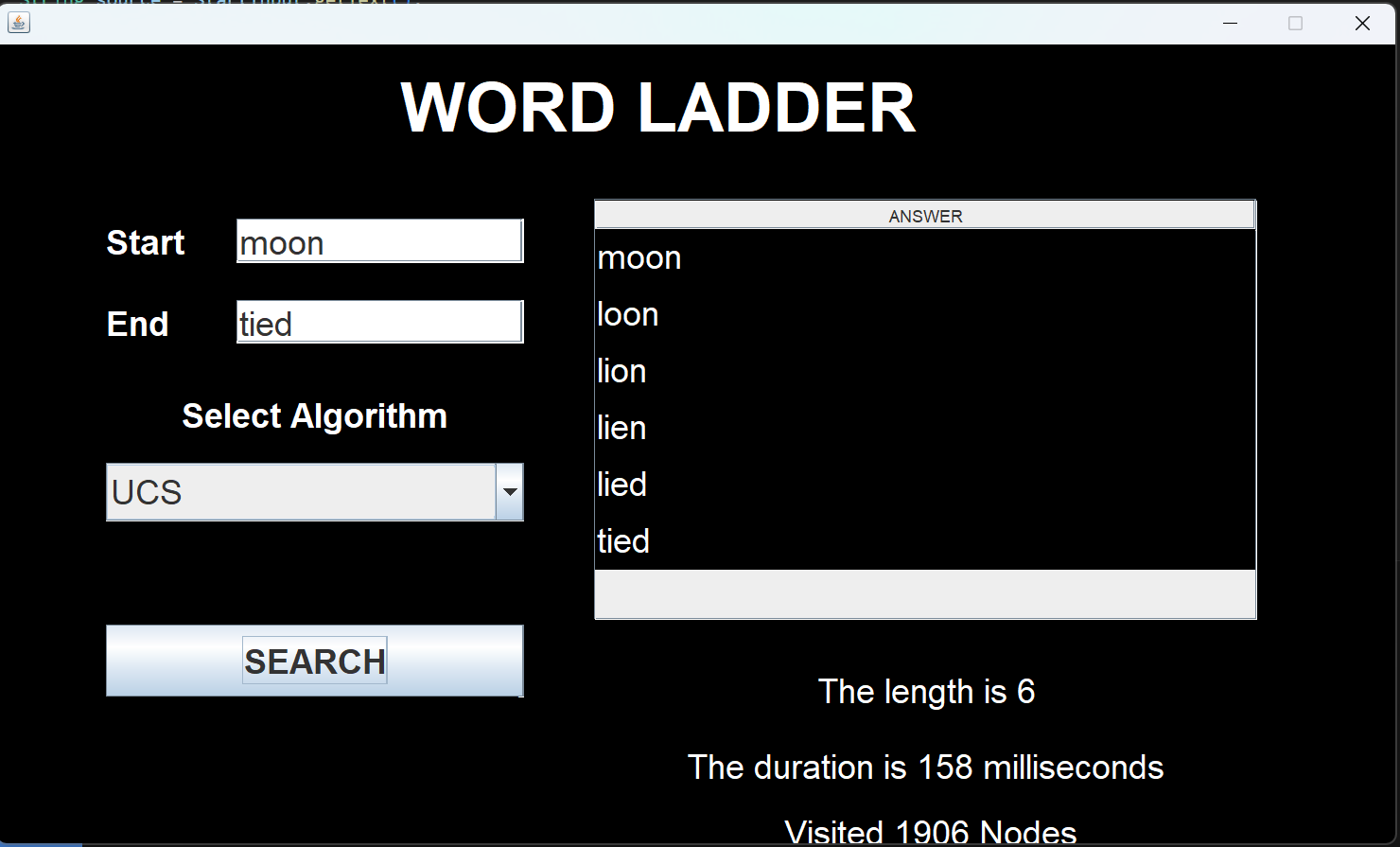
1. GBFS



1. A Star



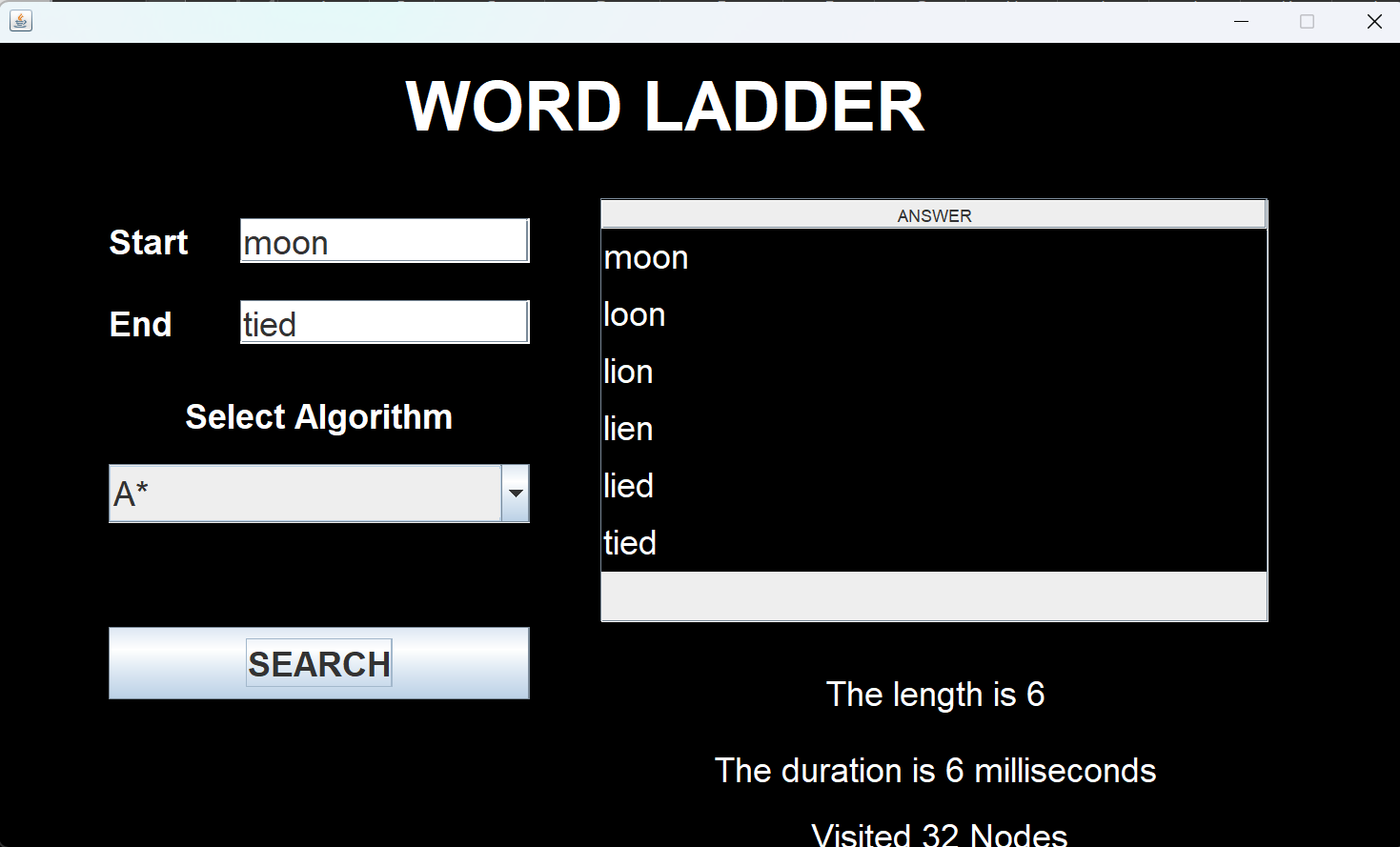
1. Moon – Tied
2. UCS



1. GBFS



1. A Star



**BAGIAN IV**

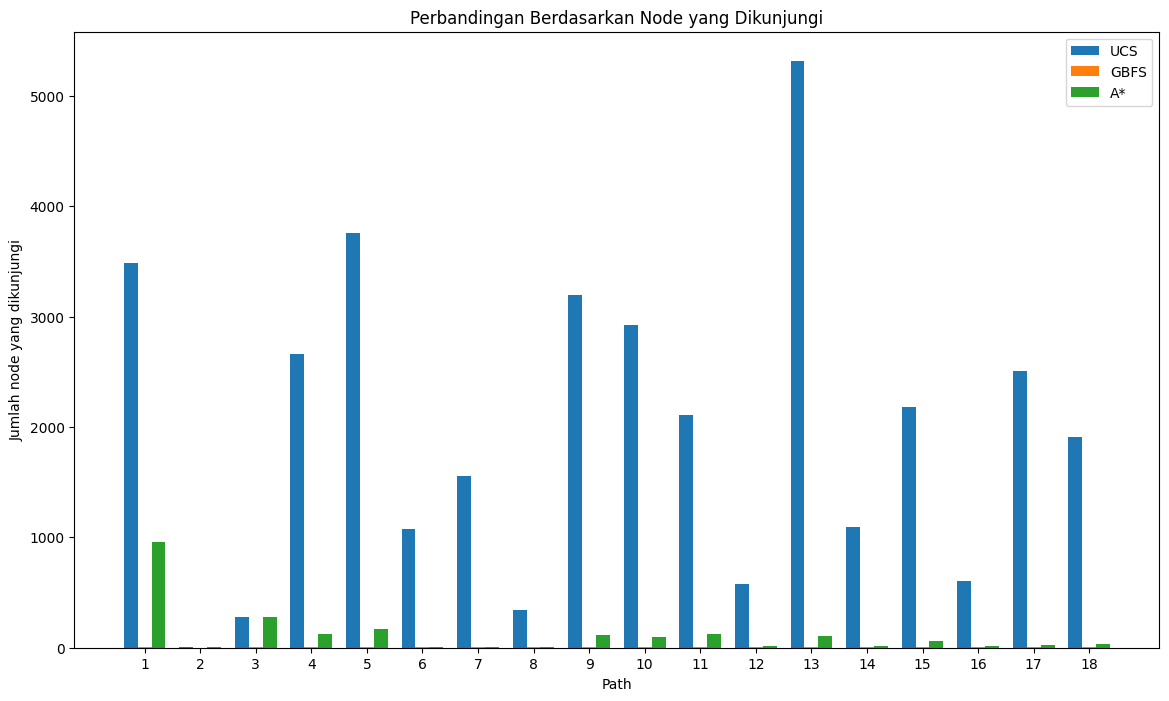
**ANALISIS PERBANDINGAN**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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

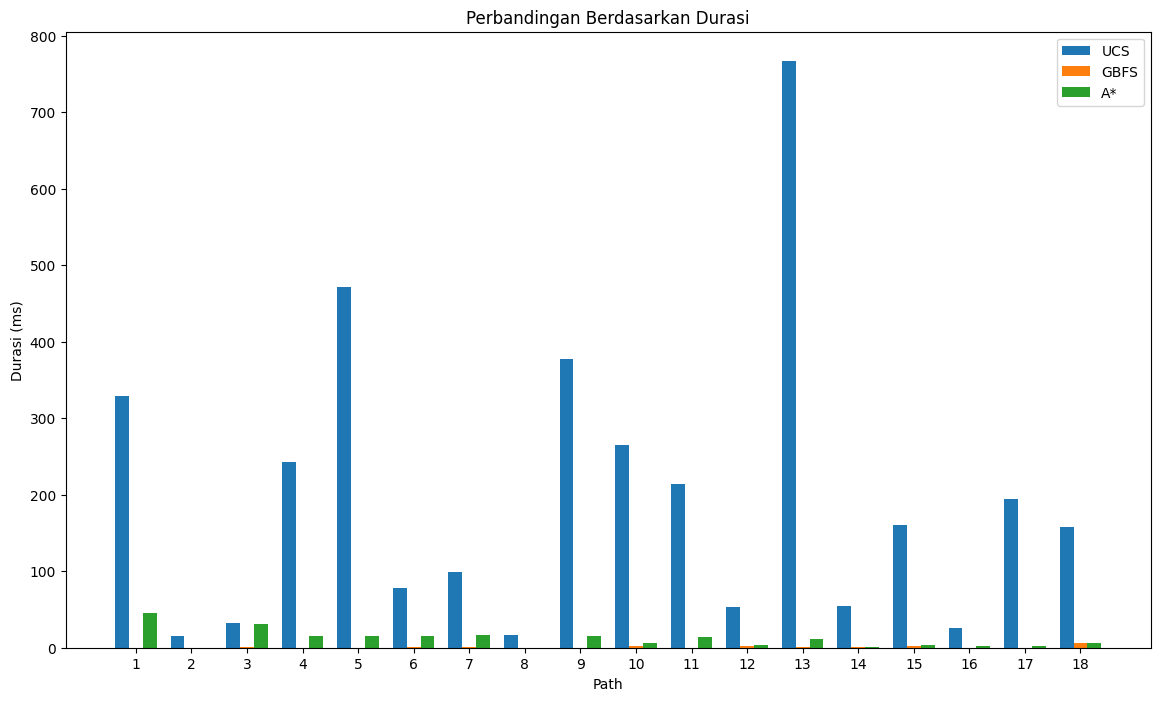
Tabel 4.1 adalah tabel yang memberikan data terkait panjang path, durasi eksekusi, dan jumlah kata yang dikunjungi oleh algoritma UCS, Greedy Best First Search, dan A Star dari uji kasus yang telah dilakukan pada Bab III. Untuk mempermudah analisis, dilakukan analisis berdasarkan tiga kategori, yang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

* 1. **Berdasarkan Jumlah Kata yang Dikunjungi**

 Gambar 4.1 Gambar grafik perbandingan berdasarkan node yang dikunjungi

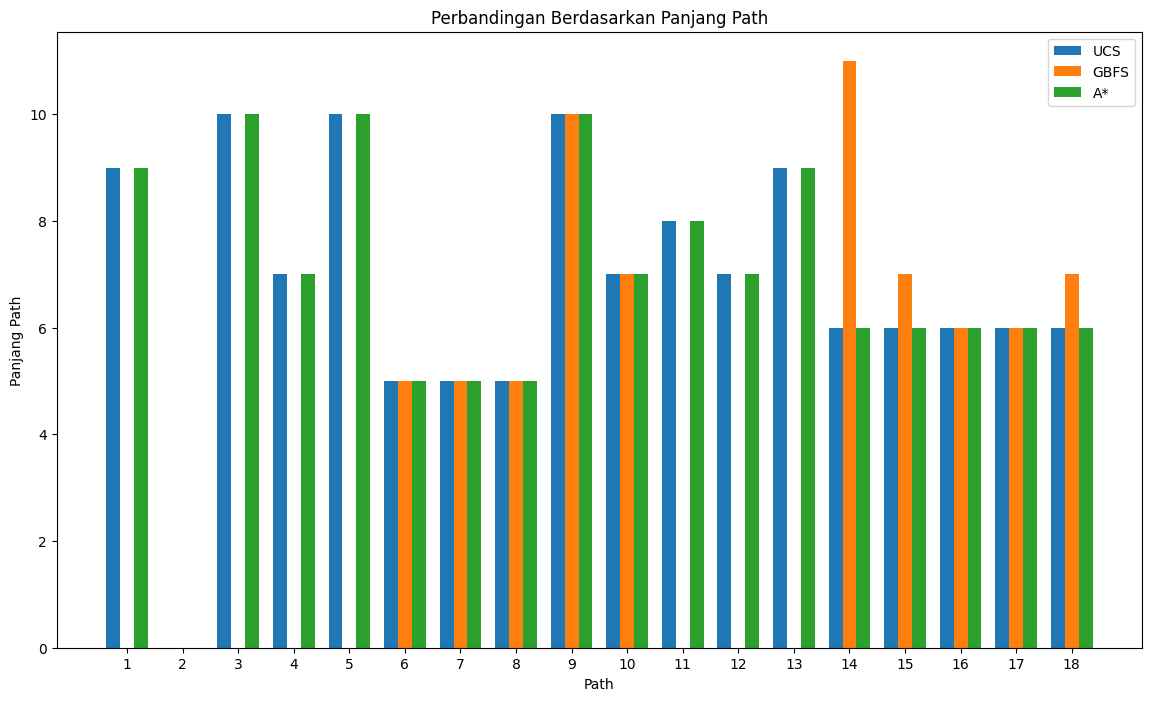
Berdasarkan gambar 4.1, dapat dilihat bawha algoritma UCS(Uniform Cost Search) cenderung mengunjungi kata yang lebih banyak, diikuti oleh algoritma A Star, dan algoritma GBFS(Greedy Best First Search). Hal ini disebebkan karena biaya/f(n) yang digunakan pada permainan Word Ladder ini adalah biaya sebelumnya 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.

* 1. **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 telebih 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 heuristik 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 heuristik terkecil sehingga dapat dengan cepat mencapai solusi atau justru terjebak pada siklus dan tidak menemui solusi.

* 1. **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 memillih kata selanjutnya yang memiliki nilai heursitic 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 selama 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 heuristik sehingga dapat mempercepat proses pencarian. Oleh karena itu, algoritma UCS dan algoritma A Star akan selalu menemui hasil yang optimal.

**Kesimpulan**

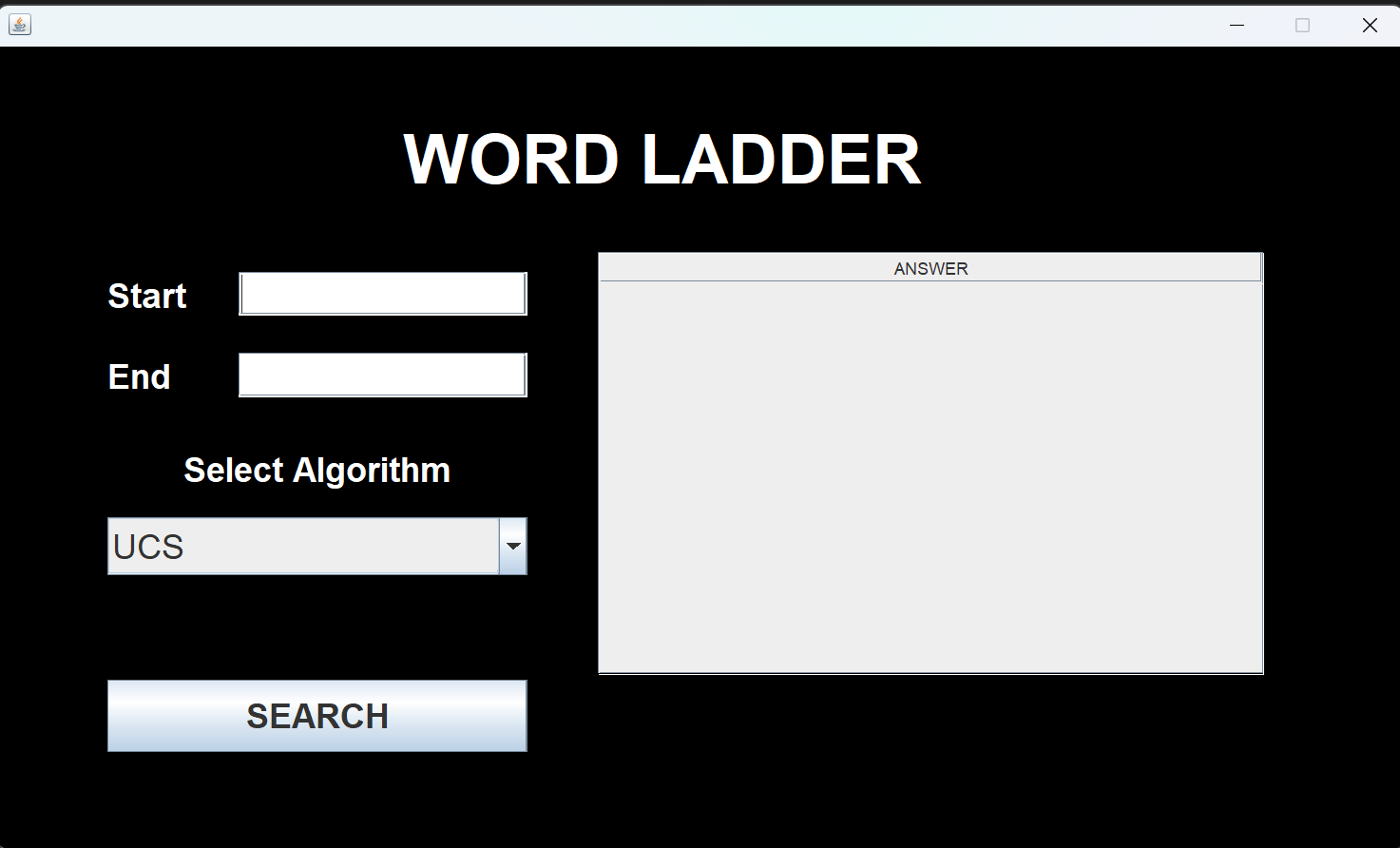
Algoritma UCS(Uniform Cost Search) memiliki durasi eksekusi yang cukup lama karena harus menelusuri kata yang cukup banyak. Namun, algoritma ini pasti akan memberikan hasil yang optimal.

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.

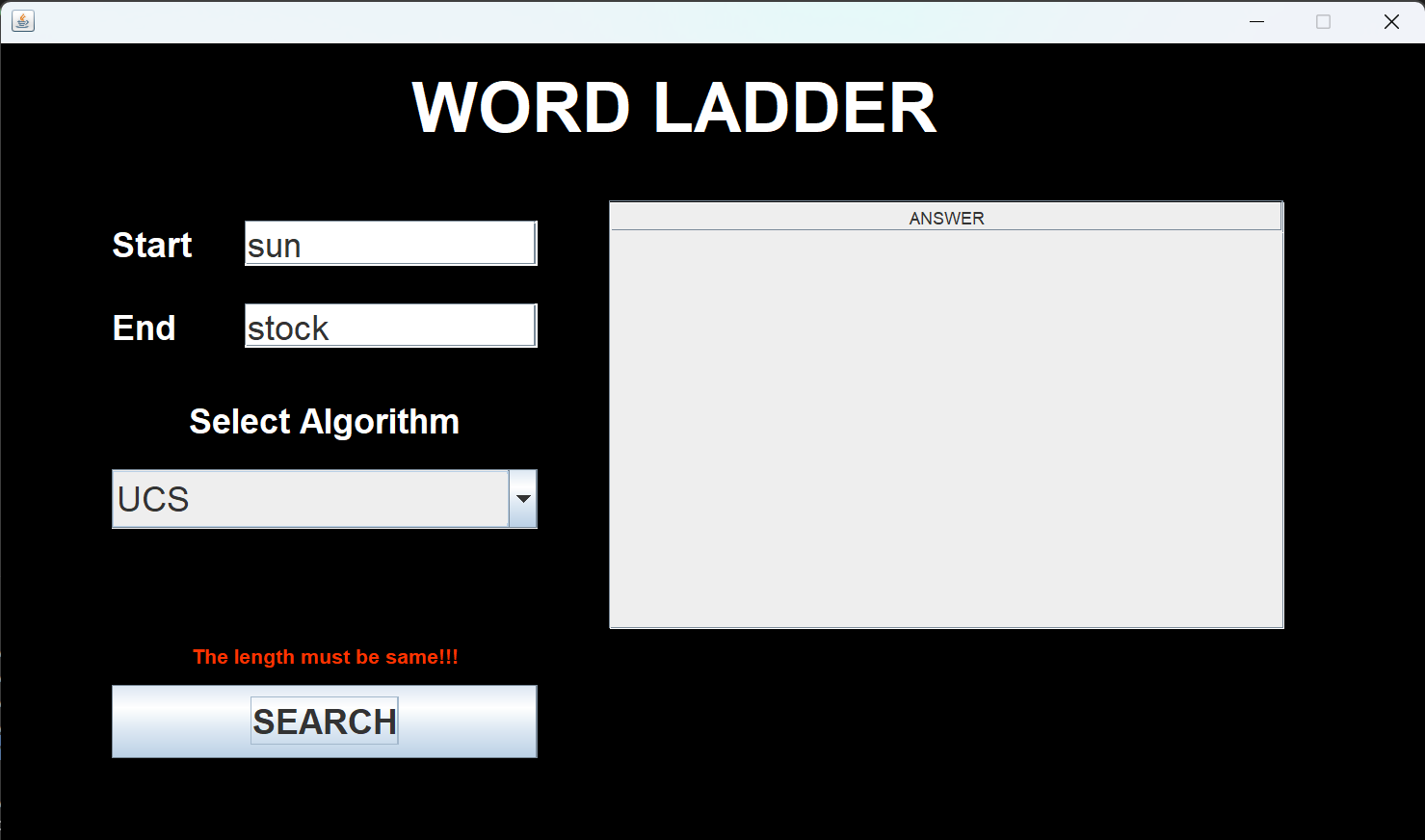
Algoritma A Star memiliki durasi eksekusi yang lebih singkat dibanding UCS karena penelusuran kata dibantu dengan nilai heuristik untuk memprioritaskan kata mana yang harus ditelusuri terlebih dahulu. Oleh karena itu, algoritma ini akan menghasilkan hasil yang optimal dengan durasi yang relatif singkat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

**BAGIAN V**

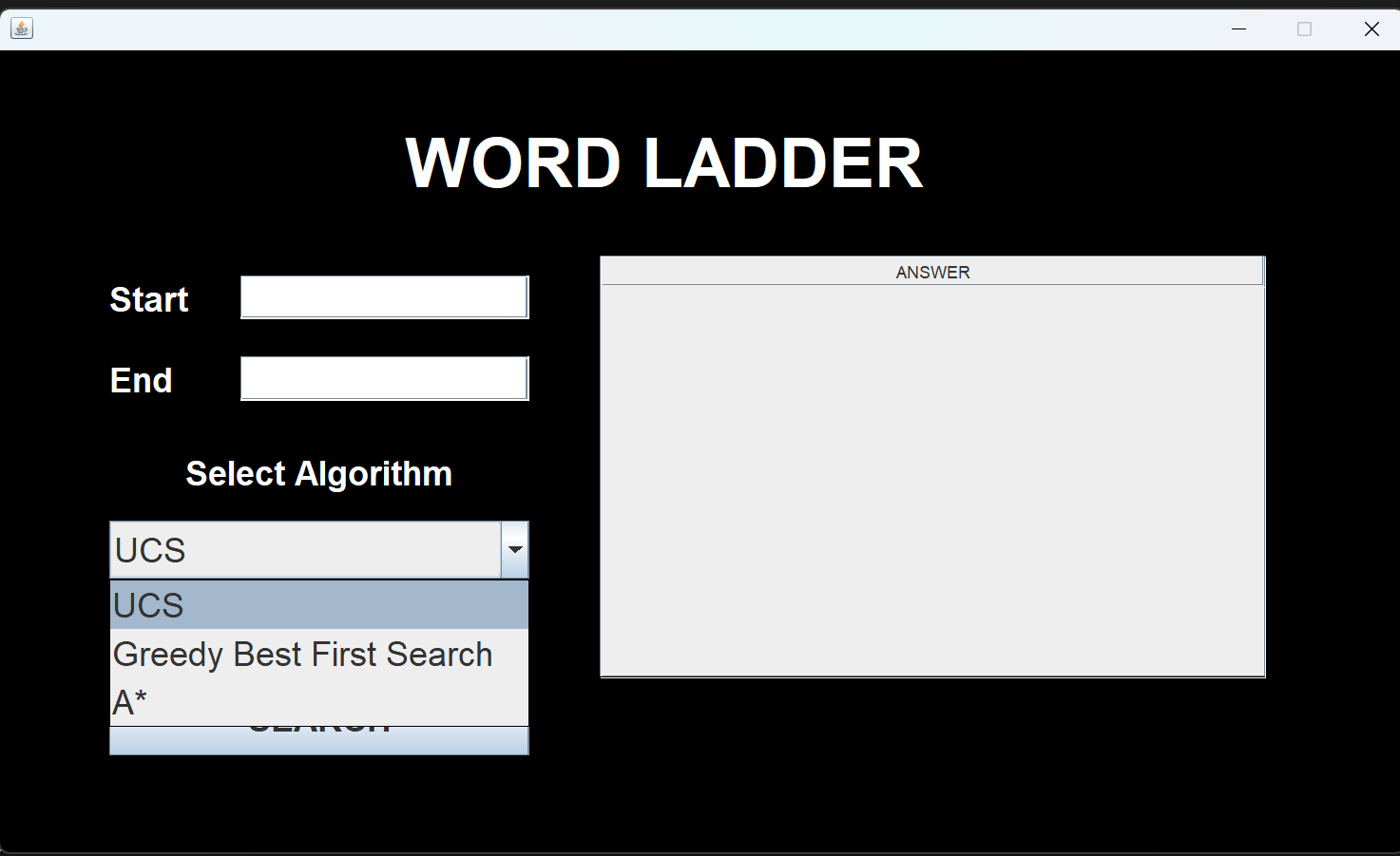
**IMPLEMENTASI GUI(*GRAPHICAL USER INTERFACE*)**



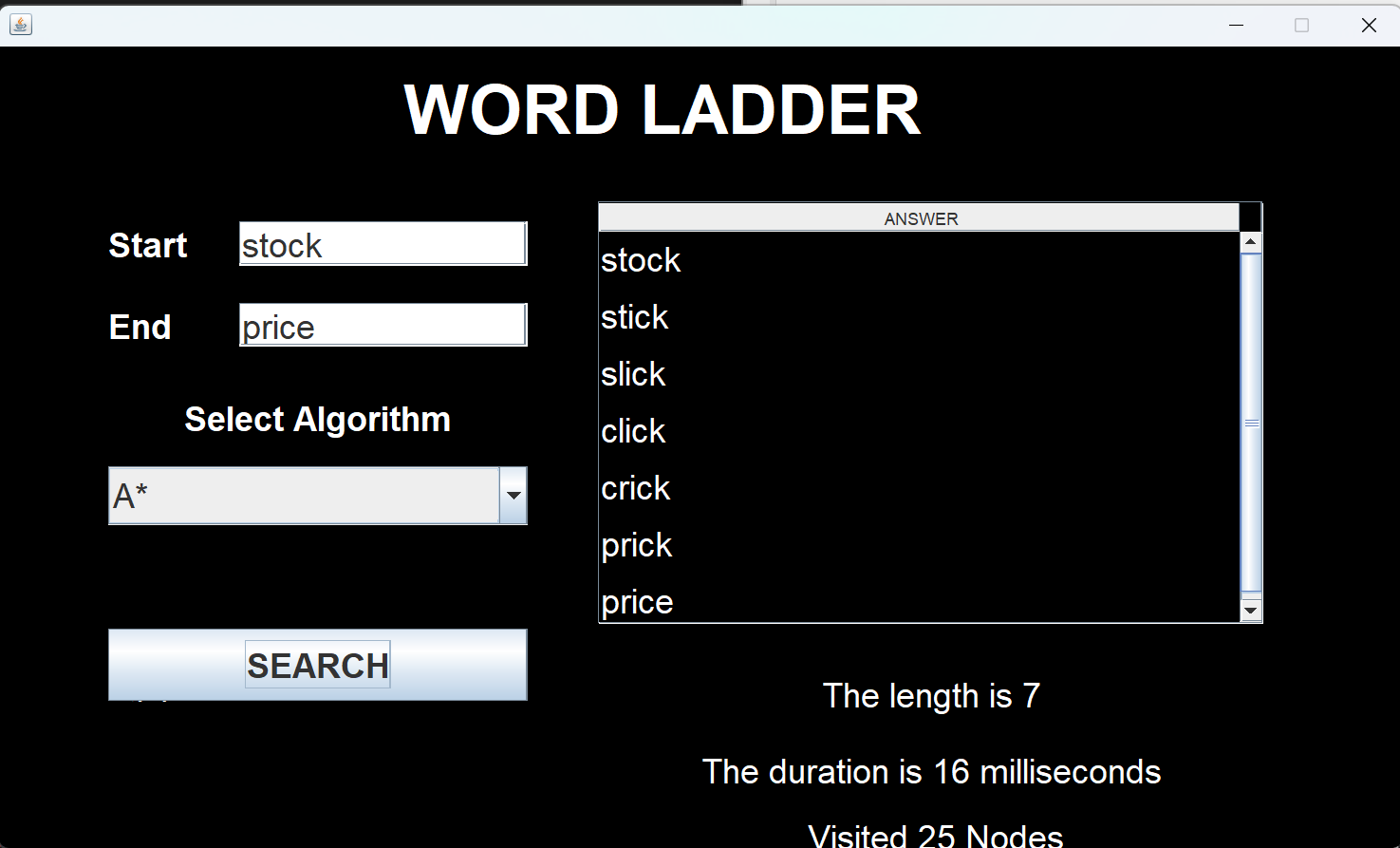
Pada proyek ini, GUI yang dibuuat menggunakan kakas Java bernama Java Swing. Aplikasi Word Ladder adalah sebuah program yang memungkinkan pengguna untuk menjelajahi serangkaian kata yang berbeda satu per satu, dengan setiap kata dalam serangkaian hanya berbeda satu huruf dari kata sebelumnya. Aplikasi 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.



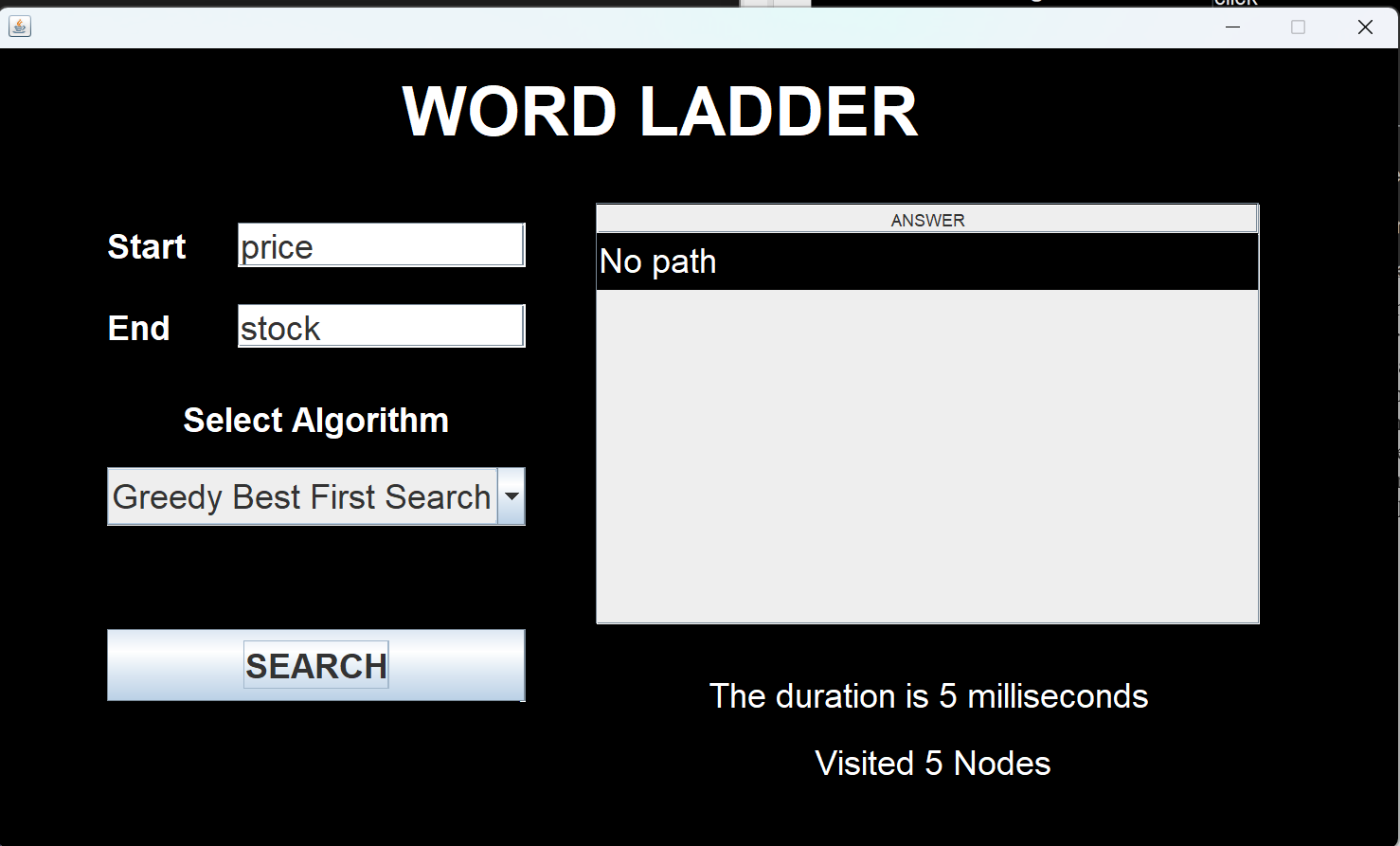
Pesan kesalahan yang jelas dan informatif akan muncul di antarmuka pengguna untuk memberi tahu pengguna tentang masalah yang terjadi. Misalnya, ketika pengguna tidak memasukkan kata awal atau akhir, atau keduanya, dan mencoba untuk memulai pencarian, pesan kesalahan akan muncul meminta mereka untuk memasukkan 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 V**

**CHECKLIST**

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