LAPORAN TUGAS KECIL 3

IF2211 Strategi Algoritma

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

Disusun oleh

Yusuf Ardian Sandi : 13522015

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

2024

Daftar Isi

Analisis dan implementasi dalam algoritma

Algoritma UCS

Mula-mula dibangkitakan node pertama yaitu startWord. Setelah itu, akan dicari semua anaknya yang memiliki perbedaan satu huruf dengan node parent dengan jumlah huruf yang sama. Kemudian, semua anak tersebut akan dimasukkan kedalam priority queue yang diurut berdasarkan banyak perbedaan huruf node tersebut jika dibandingkan dengan startWord. Lalu, semua anak dikunjungi dengan memperlakukan mereka seperti node parent sebelumnya. Namun, jika ditemukan node anak yang sudah pernah dikunjungi sebelumnya, node tersebut tidak dibangkitkan. Terakhir, jika telah ditemukan node anak yang sama dengan goalWord, akan dilakukan rekonstruksi path untuk mendapatkan array of string solusi nya. Jika tidak ditemukan goalWord hingga priority queue kosong, tidak ada solusi yang akan diberikan.

Algoritma Greedy Best-first Search (GreedyBFS)

Mula-mula dibangkitakan node pertama yaitu startWord. Setelah itu, akan dicari semua anaknya yang memiliki perbedaan satu huruf dengan node parent dengan jumlah huruf yang sama. Kemudian, semua anak tersebut akan dimasukkan kedalam priority queue yang diurut berdasarkan banyak perbedaan huruf node tersebut jika dibandingkan dengan goalWord. Lalu, semua anak dikunjungi dengan memperlakukan mereka seperti node parent sebelumnya. Namun, jika ditemukan node anak yang sudah pernah dikunjungi sebelumnya, node tersebut tidak dibangkitkan. Terakhir, jika telah ditemukan node anak yang sama dengan goalWord, akan dilakukan rekonstruksi path untuk mendapatkan array of string solusi nya. Jika tidak ditemukan goalWord hingga priority queue kosong, tidak ada solusi yang akan diberikan.

Algoritma UCS

Mula-mula dibangkitakan node pertama yaitu startWord. Setelah itu, akan dicari semua anaknya yang memiliki perbedaan satu huruf dengan node parent dengan jumlah huruf yang sama. Kemudian, semua anak tersebut akan dimasukkan kedalam priority queue yang diurut berdasarkan banyak perbedaan huruf node dengan startWord dan goalWord. Lalu, semua anak dikunjungi dengan memperlakukan mereka seperti node parent sebelumnya. Namun, jika ditemukan node anak yang sudah pernah dikunjungi sebelumnya, node tersebut tidak dibangkitkan. Terakhir, jika telah ditemukan node anak yang sama dengan goalWord, akan dilakukan rekonstruksi path untuk mendapatkan array of string solusi nya. Jika tidak ditemukan goalWord hingga priority queue kosong, tidak ada solusi yang akan diberikan.

Analisis sesuai pertanyaan spek

Pada program ini f(n) merupakan jumlah dari perbedaan huruf currentNode dengan startord dan perbedaan huruf currentNode dengan goalWord.

Pada program ini g(n) merupakan jumlah dari perbedaan huruf currentNode dengan startWord.

Heuristik yang digunakan pada algoritma A* akan selalu admissible karena hanya akan mencari perbedaan huruf currentNode dengan goalWord secara langsung. Hal ini hampir sama jika diibaratkan dengan mencari straight-line distance from n to goal.

Pada kasus word ladder, UCS dan BFS ini bisa dibilang sama karena mereka sama sama membangkitkan node yang memiliki perbedaan huruf terkecil antara currentNode dengan startWord. Hal ini disebabkan saat berpindah depth, itu artinya perbedaan huruf yang akan dihasilkan selanjutnya ialah bertambah satu. Oleh karena itu, UCS juga akan melakukan pencarian secara melebar.

Jika dibandingkan dengan UCS, A* akan lebih efisien karena dia hanya akan mencari total jarak (f(n)) yang paling rendah. Sedangkan UCS hanya akan mencari dengan urutan jarak currentNode dengan startWord yang paling rendah dimana tidak diketahui apakan next node akan mendekati goalWord atau tidak.

Algoritma GBFS secara teoritis harusnya sudah merupakan solusi optimal karena akan selalu mendahului pencarian dengan node yang paling mendekati dengan goalWord. Namun, karena gbfs tergolong “greedy”, algoritma ini tidak bisa dibilang akan selalu menjamin solusi yang optimal.

Source code program

Kelas Util

Kelas util bertanggung jawab atas fungsi yang menentukan jarak, yaitu f(n), g(n), maupun h(n). Selain itu, juga bertanggung jawab untuk memeriksa apakah dua kita hanya memiliki perbedaan satu huruf atau tidak.

```
public class Util {
    public static boolean isOneLetterDifferent(String a, String b)
{
        if (a.length() != b.length()) {
            return false;
        }

        int count = 0;
        for (int i = 0; i < a.length(); i++) {
            if (a.charAt(i) != b.charAt(i)) {
                count++;
                if (count > 1) {
                    return false;
                }
            }
        }

        return count == 1;
    }

    // Menghitung perbedaan huruf antara dua kata (untuk greedyBFS
dan UCS)
    public static int heuristic(String a, String b) {
        int count = 0;
        for (int i = 0; i < a.length(); i++) {
            if (a.charAt(i) != b.charAt(i)) {
                count++;
            }
        }
        return count;
```

```
    }

    // Menghitung perbedaan huruf antara dua kata (untuk A*)
    public static int fn(String startWord, String a, String 
goalWord) {
        int count = 0;
        for (int i = 0; i < a.length(); i++) {
            if (a.charAt(i) != goalWord.charAt(i)) {
                count++;                        // h(n)
            }
        }

        count += heuristic(startWord, a);    // g(n)
        return count;
    }
}
```

Kelas UCS

Kelas UCS bertanggung jawab untuk melakukan pencarian dengan algoritma UCS.

```
import java.util.*;

public class Ucs {
    public List<String> ucs(String start, String goal, 
List<String> dictionary) {
        HashMap<String, String> cameFrom = new HashMap<>();      // 
To build path
        Set<String> explored = new HashSet<>();          // To 
store the explored nodes
        String wordOfDepth = start;
        PriorityQueue<String> frontier = new PriorityQueue<>(new 
Comparator<String>() {
            @Override
            public int compare(String o1, String o2) {
```

```
                return Integer.compare(Util.heuristic(start, o1),
Util.heuristic(start, o2));
            }
        });

        // Calculate the used memory before
        long usedMemoryBefore = Runtime.getRuntime().totalMemory()
- Runtime.getRuntime().freeMemory();

        frontier.add(start);
        cameFrom.put(start, null);
        explored.add(frontier.element());

        int currentDepth = 0;

        while (!frontier.isEmpty()) {
            String current = frontier.poll();
            // System.out.println("Current: " + current);

            // Just to check if the start word == goal word
            if (current.equals(goal)) {
                // Build and return the path when the goal is
found
                List<String> path = new ArrayList<>();
                for (String node = goal; node != null; node =
cameFrom.get(node)) {
                    path.add(0, node);
                }
                System.out.println("\nNode visited: " +
explored.size() + " nodes");
                // Calculate the used memory after
                long usedMemoryAfter =
Runtime.getRuntime().totalMemory() -
Runtime.getRuntime().freeMemory();
                System.out.println("Memory usage: " +
(usedMemoryAfter - usedMemoryBefore) + " bytes\n");
                return path;
```

```
            }

            // print current depth
            // if (current.equals(wordOfDepth)) {
            //     System.out.println("Searching depth: " +
++currentDepth);
            // }

            for (String word : dictionary) {
                if (word.length() == start.length() &&
!explored.contains(word) && Util.isOneLetterDifferent(current,
word)) {
                    frontier.add(word);
                    cameFrom.put(word, current);
                    if (word.equals(goal)) {
                        // Build and return the path when the goal
is found
                        List<String> path = new ArrayList<>();
                        for (String node = goal; node != null;
node = cameFrom.get(node)) {
                            path.add(0, node);
                        }
                        System.out.println("\nNode visited: " +
explored.size() + " nodes");
                        // Calculate the used memory after
                        long usedMemoryAfter =
Runtime.getRuntime().totalMemory() -
Runtime.getRuntime().freeMemory();
                        System.out.println("Memory usage: " +
(usedMemoryAfter - usedMemoryBefore) + " bytes\n");
                        return path;
                    }
                    explored.add(word);
                    // if (current.equals(wordOfDepth)) {
                    //     wordOfDepth = word;
                    // }
                }
```

```
            }


        }

        System.out.println("\nNode visited: " + explored.size() +
" nodes");
        // Calculate the used memory after
        long usedMemoryAfter = Runtime.getRuntime().totalMemory()
- Runtime.getRuntime().freeMemory();
        System.out.println("Memory usage: " + (usedMemoryAfter -
usedMemoryBefore) + " bytes\n");
        return new ArrayList<>();
    }
}
```

Kelas GreedyBFS

Kelas ini bertanggung jawab untuk melakukan pencarian dengan algoritma GreedyBFS.

```
import java.util.*;


public class GreedyBFS {
    public List<String> greedyBFS(String start, String goal,
List<String> dictionary) {
        HashMap<String, String> cameFrom = new HashMap<>();      //
To build path
        Set<String> explored = new HashSet<>();          // To
store the explored nodes
        String wordOfDepth = start;
        PriorityQueue<String> frontier = new PriorityQueue<>(new
Comparator<String>() {
            @Override
            public int compare(String o1, String o2) {
                return Integer.compare(Util.heuristic(o1, goal),
```

```
Util.heuristic(o2, goal));
            }
        });
        Set<String> inFrontier = new HashSet<>();                //
Prevent backtracking

        // Calculate the used memory before
        long usedMemoryBefore = Runtime.getRuntime().totalMemory()
- Runtime.getRuntime().freeMemory();

        frontier.add(start);
        cameFrom.put(start, null);
        explored.add(frontier.element());

        int currentDepth = 0;

        while (!frontier.isEmpty()) {
            String current = frontier.poll();
            // System.out.println("Current: " + current);

            // Just to check if the start word == goal word
            if (current.equals(goal)) {
                // Build and return the path when the goal is
found
                List<String> path = new ArrayList<>();
                for (String node = goal; node != null; node =
cameFrom.get(node)) {
                    path.add(0, node);
                }
                System.out.println("\nNode visited: " +
explored.size() + " nodes");
                // Calculate the used memory after
                long usedMemoryAfter =
Runtime.getRuntime().totalMemory() -
Runtime.getRuntime().freeMemory();
                System.out.println("Memory usage: " +
```

```
(usedMemoryAfter - usedMemoryBefore) + " bytes\n");
                return path;
            }

            // print current depth
            // if (current.equals(wordOfDepth)) {
            //     System.out.println("Searching depth: " +
++currentDepth);
            // }

            for (String word : dictionary) {
                if (word.length() == start.length() &&
!explored.contains(word) && !inFrontier.contains(word) &&
Util.isOneLetterDifferent(current, word)) {
                    frontier.add(word);
                    cameFrom.put(word, current);
                    if (word.equals(goal)) {
                        // Build and return the path when the goal
is found
                        List<String> path = new ArrayList<>();
                        for (String node = goal; node != null;
node = cameFrom.get(node)) {
                            path.add(0, node);
                        }
                        System.out.println("\nNode visited: " +
explored.size() + " nodes");
                        // Calculate the used memory after
                        long usedMemoryAfter =
Runtime.getRuntime().totalMemory() -
Runtime.getRuntime().freeMemory();
                        System.out.println("Memory usage: " +
(usedMemoryAfter - usedMemoryBefore) + " bytes\n");
                        return path;
                    }
                    explored.add(word);
                    inFrontier.add(word);           // Prevent
backtracking
```

```
                    // if (current.equals(wordOfDepth)) {
                    //      wordOfDepth = word;
                    // }
                }
            }

        }
        System.out.println("\nNode visited: " + explored.size() +
" nodes");
        // Calculate the used memory after
        long usedMemoryAfter = Runtime.getRuntime().totalMemory()
- Runtime.getRuntime().freeMemory();
        System.out.println("Memory usage: " + (usedMemoryAfter -
usedMemoryBefore) + " bytes\n");
        return new ArrayList<>();
    }
}
```

Kelas Asearch

Kelas ini bertanggung jawab untuk melakukan pencarian dengan algoritma A* search.

```
import java.util.*;

public class Asearch {
    public List<String> aStarSearch(String start,String
goal,List<String> dictionary) {
        HashMap<String, String> cameFrom = new HashMap<>();      //
To build path
        Set<String> explored = new HashSet<>();          // To
store the explored nodes
```

```
        String wordOfDepth = start;
        PriorityQueue<String> frontier = new PriorityQueue<>(new
Comparator<String>() {
            @Override
            public int compare(String o1, String o2) {
                return Integer.compare(Util.fn(start, o1, goal),
Util.fn(start, o2, goal));
            }
        });

        // Calculate the used memory before
        long usedMemoryBefore = Runtime.getRuntime().totalMemory()
- Runtime.getRuntime().freeMemory();

        frontier.add(start);
        cameFrom.put(start, null);
        explored.add(frontier.element());

        int currentDepth = 0;

        while (!frontier.isEmpty()) {
            String current = frontier.poll();
            // System.out.println("Current: " + current);

            // Just to check if the start word == goal word
            if (current.equals(goal)) {
                // Build and return the path when the goal is
found
                List<String> path = new ArrayList<>();
                for (String node = goal; node != null; node =
cameFrom.get(node)) {
                    path.add(0, node);
                }
                System.out.println("\nNode visited: " +
explored.size() + " nodes");
                // Calculate the used memory after
```

```
                long usedMemoryAfter =
Runtime.getRuntime().totalMemory() -
Runtime.getRuntime().freeMemory();
                System.out.println("Memory usage: " +
(usedMemoryAfter - usedMemoryBefore) + " bytes\n");
                return path;
            }

            // print current depth
            // if (current.equals(wordOfDepth)) {
            //     System.out.println("Searching depth: " +
++currentDepth);
            // }

            for (String word : dictionary) {
                if (word.length() == start.length() &&
!explored.contains(word) && Util.isOneLetterDifferent(current,
word)) {
                    frontier.add(word);
                    cameFrom.put(word, current);
                    if (word.equals(goal)) {
                        // Build and return the path when the goal
is found
                        List<String> path = new ArrayList<>();
                        for (String node = goal; node != null;
node = cameFrom.get(node)) {
                            path.add(0, node);
                        }
                        System.out.println("\nNode visited: " +
explored.size() + " nodes");
                        // Calculate the used memory after
                        long usedMemoryAfter =
Runtime.getRuntime().totalMemory() -
Runtime.getRuntime().freeMemory();
                        System.out.println("Memory usage: " +
(usedMemoryAfter - usedMemoryBefore) + " bytes\n");
```

```java
                        return path;
                    }
                    explored.add(word);
                    // if (current.equals(wordOfDepth)) {
                    //      wordOfDepth = word;
                    // }
                }
            }

        }
        System.out.println("\nNode visited: " + explored.size() +
" nodes");
        // Calculate the used memory after
        long usedMemoryAfter = Runtime.getRuntime().totalMemory()
- Runtime.getRuntime().freeMemory();
        System.out.println("Memory usage: " + (usedMemoryAfter -
usedMemoryBefore) + " bytes\n");
        return new ArrayList<>();
    }
}
```

Kelas CSVReader

Kelas yang bertanggung jawab untuk membaca CSV untuk keperluan penyimpanan dictionary.

```java
// import org.apache.commons.csv.*;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.Reader;
```

```java
import java.util.ArrayList;
import java.util.List;
import java.util.Locale;

public class CSVReader {
    public CSVReader() {
    }

    public List<String> make_dictionary() {
        try {
            List<String> columnData = new ArrayList<>();

            BufferedReader br = new BufferedReader(new
FileReader("english-dictionary.csv"));

            String line = br.readLine();

            // Skip the first line
            line = br.readLine();
            int counter = 0;
            while (line != null) {
                String[] values = line.split(",");
                values[0] = values[0].toLowerCase(Locale.ROOT);
                // System.out.println(values[0]);

                // If the value doesnt unique or it contains
another character than letters, skip it
                if (columnData.contains(values[0]) ||
!values[0].matches("[a-zA-Z]+")) {
                    line = br.readLine();
                    continue;
                }
                columnData.add(values[0]);
                line = br.readLine();
                counter++;
                // print counter every 30000 lines
```

```
                if (counter % 30000 == 0) {
                    System.out.println(counter);
                }
            }

            // Print the length
            System.out.println("Length of dictionary: " +
columnData.size());

            br.close();

            return columnData;

        } catch(Exception e) {
            e.printStackTrace();
        }

        // return empty list
        return new ArrayList<>();
    }

    public static void main(String[] args) {
        CSVReader csvReader = new CSVReader();
        List<String> dictionary = csvReader.make_dictionary();

        // Generate new txt files that contain the words in the
dictionary with this format: {"String", "String", "String", ...}
        try {
            FileWriter myWriter = new
FileWriter("dictionary.txt");
            myWriter.write("{" + String.join(", ", dictionary) +
"}");
            myWriter.close();
            System.out.println("Successfully wrote to the file.");
        } catch (IOException e) {
            System.out.println("An error occurred.");
```

```
            e.printStackTrace();
        }
    }
}
```

Kelas Main

Kelas ini bertugas untuk membaca dictionary yang telah ada dan menjadikan nya dalam bentuk list. Lalu ini juga bertanggungjawab menerima input, menghitung waktu eksekusi, menghitung memory usage, dan mengeluarakn output. Kelas ini juga berfungsi sebagai program utama.

```
import java.io.FileWriter;
import java.io.IOException;
import java.util.*;
import java.nio.file.*;

public class Main {
    public static void main(String[] args) {

        // Create string list called dictionary
        try {
            // String content = new
String(Files.readAllBytes(Paths.get("../test/dictionaryn.txt")));
            // List<String> dictionary = new
ArrayList<>(Arrays.asList(content.split(", ")));

            List<String> dictionary =
Files.readAllLines(Paths.get("../test/dictAsisten.txt"));


            // Save the length of longest word in the dictionary
            int maxLength = 0;
            for (String word : dictionary) {
```

```
                if (word.length() > maxLength) {
                    maxLength = word.length();
                }
            }

            String startWord;
            String goalWord;
            Scanner scanner = new Scanner(System.in);

            // validate start and goal word
            while (true) {
                // Take input from terminal start word and goal
word
                System.out.print("Enter start word: ");
                startWord = scanner.nextLine();
                System.out.print("Enter goal word: ");
                goalWord = scanner.nextLine();
                System.out.println();
                if (dictionary.contains(startWord) &&
dictionary.contains(goalWord)) {
                    break;
                }
                else if (startWord.length() != goalWord.length())
{
                    System.out.println("Start word and goal word
must have the same length.");
                }
                else if (startWord.length() > maxLength ||
goalWord.length() > maxLength) {
                    System.out.println("Word must less than or
equal to " + maxLength + " characters.");
                } else if (!dictionary.contains(startWord)) {
                    System.out.println("Start word is not in the
dictionary.");
                } else {
                    System.out.println("Goal word is not in the
```

```
dictionary.");
                    }
                }

                int algorithm = 0;
                while (algorithm < 1 || algorithm > 3) {
                    // Choose the algorithm between UCS, Greedy
best-first search, and A* search
                    System.out.println("Choose the algorithm:");
                    System.out.println("1. UCS");
                    System.out.println("2. Greedy best-first search");
                    System.out.println("3. A* search");
                    System.out.print("Enter the number of the
algorithm: ");
                    algorithm = scanner.nextInt();
                    System.out.println();
                }

                scanner.close();

                long startTime = System.currentTimeMillis();


                List<String> path = null;
                // Maap print depth nya masih salah (hanya utk
kepentingan debugging)
                if (algorithm == 1) {
                    Ucs ucs = new Ucs();
                    path = ucs.ucs(startWord, goalWord, dictionary);
                } else if (algorithm == 2) {
                    GreedyBFS greedy = new GreedyBFS();
                    path = greedy.greedyBFS(startWord, goalWord,
dictionary);
                } else if (algorithm == 3) {
                    Asearch aStarSearch = new Asearch();
                    path = aStarSearch.aStarSearch(startWord,
```

```
goalWord, dictionary);
            }



            long endTime = System.currentTimeMillis();
            long executionTime = endTime - startTime;

            if (path.isEmpty()) {
                System.out.println("No path found.");
            } else {
                System.out.println("Path: " + path);
            }
            System.out.println("Execution time: " + executionTime
+ " ms");

        } catch (IOException e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }

    }
}
```

Tangkapan layar yang memperlihatkan input dan output

Note: Abaikan memory usage

<table>
<tr><th>Input</th><th colspan="2">Hasil</th></tr>
<tr><td rowspan="3">Start: my
End: to</td><td>UCS</td><td><pre>Enter start word: my
Enter goal word: to

Choose the algorithm:
1. UCS
2. Greedy best-first search
3. A* search
Enter the number of the algorithm: 1

Node visited: 73 nodes
Memory usage: 463448 bytes

Path: [my, mo, to]
Execution time: 40 ms</pre></td></tr>
<tr><td>GBFS</td><td><pre>Enter start word: my
Enter goal word: to

Choose the algorithm:
1. UCS
2. Greedy best-first search
3. A* search
Enter the number of the algorithm: 2

Node visited: 18 nodes
Memory usage: 0 bytes

Path: [my, mo, to]
Execution time: 24 ms</pre></td></tr>
<tr><td>A*</td><td><pre>Enter start word: my
Enter goal word: to

Choose the algorithm:
1. UCS
2. Greedy best-first search
3. A* search
Enter the number of the algorithm: 3

Node visited: 18 nodes
Memory usage: 464456 bytes

Path: [my, mo, to]
Execution time: 24 ms</pre></td></tr>
</table>

Start: cat End: dog	UCS	Enter start word: cat Enter goal word: dog Choose the algorithm: 1. UCS 2. Greedy best-first search 3. A* search Enter the number of the algorithm: 1 Node visited: 358 nodes Memory usage: 0 bytes Path: [cat, cam, dam, dag, dog] Execution time: 66 ms
	GBFS	Enter start word: cat Enter goal word: dog Choose the algorithm: 1. UCS 2. Greedy best-first search 3. A* search Enter the number of the algorithm: 2 Node visited: 55 nodes Memory usage: 463472 bytes Path: [cat, cot, cog, dog] Execution time: 21 ms
	A*	Enter start word: cat Enter goal word: dog Choose the algorithm: 1. UCS 2. Greedy best-first search 3. A* search Enter the number of the algorithm: 3 Node visited: 55 nodes Memory usage: 463448 bytes Path: [cat, cot, cog, dog] Execution time: 18 ms

Start: game End: shot	UCS	Enter start word: game Enter goal word: shot Choose the algorithm: 1. UCS 2. Greedy best-first search 3. A* search Enter the number of the algorithm: 1 Node visited: 1800 nodes Memory usage: 0 bytes Path: [game, gamp, gasp, gast, gait, grit, grot, grow, grew, gree, glee, glue, slue, sloe, shoe, shot] Execution time: 221 ms
	GBFS	Enter start word: game Enter goal word: shot Choose the algorithm: 1. UCS 2. Greedy best-first search 3. A* search Enter the number of the algorithm: 2 Node visited: 179 nodes Memory usage: 463464 bytes Path: [game, same, sabe, sabs, sals, salt, silt, sift, soft, soot, shot] Execution time: 47 ms
	A*	Enter start word: game Enter goal word: shot Choose the algorithm: 1. UCS 2. Greedy best-first search 3. A* search Enter the number of the algorithm: 3 Node visited: 670 nodes Memory usage: 0 bytes Path: [game, gamp, gasp, gast, gait, wait, whit, shit, shot] Execution time: 88 ms
Start: fluke End: quick	UCS	Enter start word: fluke Enter goal word: quick Choose the algorithm: 1. UCS 2. Greedy best-first search 3. A* search Enter the number of the algorithm: 1 Node visited: 6931 nodes Memory usage: 1386440 bytes Path: [fluke, flume, glume, glime, slime, stime, stile, smile, smite, suite, quite, quire, quirk, quick] Execution time: 2216 ms

	GBFS	Enter start word: fluke Enter goal word: quick Choose the algorithm: 1. UCS 2. Greedy best-first search 3. A* search Enter the number of the algorithm: 2 Node visited: 299 nodes Memory usage: 463472 bytes Path: [fluke, flute, flite, flits, flics, flick, slick, stick, stink, sting, suing, suint, quint, quirt, quirk, quick] Execution time: 110 ms
	A*	Enter start word: fluke Enter goal word: quick Choose the algorithm: 1. UCS 2. Greedy best-first search 3. A* search Enter the number of the algorithm: 3 Node visited: 355 nodes Memory usage: 463456 bytes Path: [fluke, flute, flite, flits, slits, suits, quits, quite, quire, quirk, quick] Execution time: 107 ms
Start: museum End: zephyr	UCS	Enter start word: museum Enter goal word: zephyr Choose the algorithm: 1. UCS 2. Greedy best-first search 3. A* search Enter the number of the algorithm: 1 Node visited: 1 nodes Memory usage: 463464 bytes No path found. Execution time: 21 ms

	GBFS	<pre>Enter start word: museum
		Enter goal word: zephyr
		
		Choose the algorithm:
		1. UCS
		2. Greedy best-first search
		3. A* search
		Enter the number of the algorithm: 2
		
		Node visited: 1 nodes
		Memory usage: 0 bytes
		
		No path found.
		Execution time: 23 ms</pre>

	A*	<pre>Enter start word: museum
		Enter goal word: zephyr
		
		Choose the algorithm:
		1. UCS
		2. Greedy best-first search
		3. A* search
		Enter the number of the algorithm: 3
		
		Node visited: 1 nodes
		Memory usage: 463448 bytes
		
		No path found.
		Execution time: 27 ms</pre>

Start: mangoes End: stirred	UCS	Enter start word: mangoes Enter goal word: stirred Choose the algorithm: 1. UCS 2. Greedy best-first search 3. A* search Enter the number of the algorithm: 1 Node visited: 3725 nodes Memory usage: 925640 bytes Path: [mangoes, mangles, dangles, dandles, dandies, pandies, pantie s, parties, parries, tarries, tarrier, terrier, tearier, learier, l eavier, heavier, headier, beadier, beakier, brakier, brasier, brash er, brashes, crashes, clashes, slashes, swashes, swishes, swisher, swither, slither, slather, slatier, platier, platter, platted, slat ted, swatted, swatter, smatter, smarter, starter, starver, starved, starred, stirred] Execution time: 1932 ms
	GBFS	Enter start word: mangoes Enter goal word: stirred Choose the algorithm: 1. UCS 2. Greedy best-first search 3. A* search Enter the number of the algorithm: 2 Node visited: 1012 nodes Memory usage: 0 bytes Path: [mangoes, mangles, mangled, jangled, jungled, bungled, burgle d, burbled, bubbled, bubbles, bubbies, hubbies, hobbies, lobbies, l oobies, loonies, loonier, moonier, moodier, woodier, wordier, worri er, worries, corries, carries, carried, curried, hurried, hurries, hurdies, burdies, burnies, bunnies, funnies, funnier, punnier, punk ier, punkies, punties, putties, puttied, juttied, jettied, jettier, pettier, peatier, platier, slatier, slather, slither, slitter, sli tted, spitted, spirted, skirted, skirred, stirred] Execution time: 491 ms

	A*	Enter start word: mangoes Enter goal word: stirred Choose the algorithm: 1. UCS 2. Greedy best-first search 3. A* search Enter the number of the algorithm: 3 Node visited: 2176 nodes Memory usage: 463448 bytes Path: [mangoes, mangles, mangled, dangled, dandled, candled, candied, bandied, bandies, baldies, ballies, bailies, dailies, doilies, doolies, coolies, cooties, footies, footles, footled, tootled, toothed, soothed, southed, soughed, coughed, couched, coached, coacted, coasted, boasted, blasted, blatted, slatted, slitted, spitted, spirted, skirted, skirred, stirred] Execution time: 1140 ms

Hasil analisis perbandingan solusi

Jika dilihat pada hasil testcase di atas, GBFS selalu mendapatkan node visited terkecil. Oleh karena itu, GBFS pastilah menggunakan memori yang lebih sedikit daripada UCS dan A*. Sedangkan memori yang digunakan pada A* lebih sedikit daripada UCS. Namun, dari segi optimum, A* menduduki peringkat paling atas. Hal ini disebabkan selain menggunakan h(n) pada GBFS, A* juga menggunakan g(n) sehiingga selain mengutamakan node yang lebih dekat dengan goalWord, node yang dekat dengan startWord juga diutamakan untuk dicari. Dari segi waktu eksekusi, GBFS dan A* cenderung lebih cepat daripada UCS. Namun, pada pencarian yang memerlukan tingkat kedalaman yang tinggi algoritma GBFS lebih cepat dibandingkan dengan. Hal ini juga berbanding lurus dengan node visited dari masing masing algoritma.

Pranala ke repository yang berisi kode program.

https://github.com/Yusufarsan/Tucil3_13522015