Laporan Tugas Kecil 3 IF2211 Strategi Algoritma

Disusun oleh:
Azmi Mahmud Bazeid (13522109)

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA INSTITUT TEKNOLOGI BANDUNG
2024

Analisis dan implementasi

Pada permainan ini, terdapat diberikan dua kata (katakanlah source dan destination). Kita harus mencari jalan untuk mendapatkan destination dari source dengan mengubah tepat 1 huruf untuk setiap langkah.

Persoalan ini dapat diselesaikan dengan teori graf. Andaikan semua kata-kata yang valid adalah node (simpul). Hubungkan 2 node jika dan hanya jika kedua kata tersebut memiliki perbedaan huruf tepat 1 huruf sehingga dalam 1 langkah dari node A, semua node yang bisa dicapai dari node A adalah node-node yang adjacent (berdekatan) dengan node A. Karena hanya dihubungkan dua node yang memiliki perbedaan tepat 1 huruf, maka dapat dianggap grafnya adalah unweighted (graf tak berbobot).

Maka persoalan, sama dengan mencari jarak terpendek dari node source ke node destination dengan menggunakan edges (sisi) yang sesedikit mungkin (mengingat grafnya adalah tak berbobot). Maka persoalan dapat diselesaikan menggunakan banyak algoritmanya diantaranya:

a. A* search algorithm

A* menentukan node mana yang akan divisit selanjutnya menggunakan fungsi f(n) yang menyatakan cost dari node n sehingga A* akan mengvisit node dengan nilai f terkecil.

Dalam A*, f(n) = g(n) + h(n) di mana g(n) adalah cost dari node source ke node n dan h(n) adalah fungsi heuristic yang memprediksi cost dari node n ke node destination.

Dalam kasus ini, g(n) adalah g(c) + 1 di mana pada saat itu, program sedang mengvisit node c dan sedang mengecek node n.

Dalam kasus ini, h(n) yang digunakan adalah banyaknya huruf yang berbeda antara node n dengan node destination. Suatu fungsi heuristic h(n), dinyatakan admissible jika dan hanya jika h(n) tidak pernah melebihi cost yang sesungguhnya (dalam bahasa matematis, h(n) <= h*(n) di mana h*(n) adalah cost yang sesungguhnya dari node n ke node destination). Perhatikan bahwa fungsi h(n) yang kita gunakan merupakan heuristic yang admissible karena pada setiap langkah, kita hanya boleh mengubah tepat 1 huruf, sehingga minimal langkah yang kita butuhkan adalah banyaknya huruf yang berbeda di node n dan di node destination sehingga memenuhi definisi admissible karena “never overestimates”.

Karena h(n) yang digunakan admissible, maka A* yang dilakukan optimal.

Langkah-langkah:

1. Buatlah priority queue untuk menyimpan node dan hasil f(node). Priority queue mengurutkan berdasarkan nilai f(node) terkecil.
2. Awalnya, semua node memiliki nilai f sama dengan tak hingga.
3. Awalnya (priority) queue diisi dengan source dan f(source) = 0 + h(source) = h(source).
4. Pop node n di queue nya.
5. Jika node n adalah destination, maka sudah ketemu.
6. Iterasi setiap neighbour dari node n. Hitung nilai f(neighbour) dan jika lebih kecil dari yang sebelumnya maka masukkan kedalam queue.
7. Jika queue sudah habis, maka tidak ada solusi. Jika masih belum habis, maka balik ke step 4.

b. Uniform cost search algorithm

Uniform cost search merupakan spesialisasi A* dengan h(n) = 0.

Mengulang paragraf awal bab Analisis dan Implementasi, kita telah mengetahui bahwa graf dapat dianggap unweighted atau dalam kata lain

weighted namun memiliki bobot yang sama semua. Oleh karena itu, kita dapat menggunakan queue yang biasa karena dengan menggunakan queue yang biasa, akan dijamin bahwa nilai f(n) dari node-node di queue akan terurut.

Karena kita menggunakan queue biasa, maka algoritma UCS dalam kasus ini ekuivalen dengan algoritma BFS.

Jika dibanding dengan A* search algorithm, maka algoritma UCS memiliki segi kompleksitas waktu lebih baik dibandingkan dengan A* search algorithm pada kasus ini karena kita hanya menggunakan queue biasa sehingga operasi push dan pop dapat dilakukan dalam O(1) dan bukan dalam O(log (size of priority queue)).

Langkah-langkah:

1. Buatlah queue untuk menyimpan node. Buatlah sebuah set yang menyatakan apakah suatu node sudah divisit.
2. Awalnya queue diisi dengan source dan source sudah divisit.
3. Pop node n di queue nya.
4. Jika node n adalah destination, maka sudah ketemu.
5. Iterasi setiap neighbour dari node n. Jika belum divisit, masukkan ke queuenya.
6. Jika queue sudah habis, maka tidak ada solusi. Jika masih belum habis, maka balik ke step 3.

c. Greedy best-first search variant 1

Greedy best-first search varian ini merupakan spesialisasi A* dengan g(n) = 0, dan h(n) seperti pada algoritma A* di atas (yaitu banyaknya huruf yang berbeda antara node n dengan node destination).

Secara teoretis, jawaban dari greedy best-first search varian ini belum tentu optimal karena bisa jadi algoritma ini stuck memilih node yang di mananya memiliki path yang lebih panjang dengan solusi optimalnya.

Namun, secara teoretis, greedy best-first search varian ini complete dalam arti bahwa jika ada solusi, maka algoritma ini akan selalu menemukan solusinya.

Langkah-langkah:
Lakukan hal yang mirip dengan A* namun h(n) = 0.

1. Buatlah priority queue untuk menyimpan node dan hasil h(node). Priority queue mengurutkan berdasarkan nilai h(node) terkecil.
2. Awalnya, semua node memiliki nilai h sama dengan tak hingga.
3. Awalnya (priority) queue diisi dengan source dan h(source) = 0.
4. Pop node n di queue nya.
5. Jika node n adalah destination, maka sudah ketemu.
6. Iterasi setiap neighbour dari node n. Hitung nilai h(neighbour) dan masukkan kedalam queue jika belum ada.
7. Jika queue sudah habis, maka tidak ada solusi. Jika masih belum habis, maka balik ke step 4.

d. Greedy best-first search variant 2

Greedy best-first search varian ini, memilih node dengan nilai terkecil dari sebuah fungsi heuristic h(n) (h(n) di sini masih sama dengan yang di algoritma A*) namun, algoritma ini tidak mengecek apakah suatu node sudah pernah divisit dan juga tidak melakukan backtracking sehingga belum tentu complete seperti varian pertama.

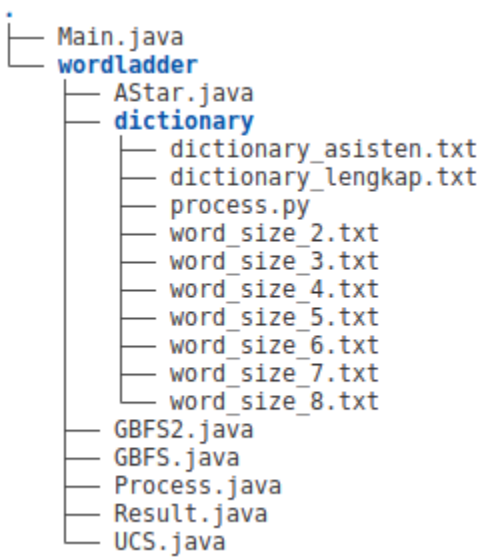
Secara teoretis, jawaban dari greedy best-first search varian ini belum tentu optimal karena bisa jadi algoritma ini stuck memilih node yang di mananya memiliki path yang lebih panjang dengan solusi optimalnya.

Langkah-langkah:

1. Awalnya node source sedang divisit.
2. Untuk setiap node n yang sedang divisit, iterasikan setiap neighbour dan visit neighbour dengan nilai h(neighbour) terkecil. Jika ada 2 neighbour yang memiliki nilai h yang sama, prioritaskan yang *lexicographically smallest*. Jika dalam melakukan visit terdapat repetisi, maka algoritma berhenti dan mengeluarkan tidak ada solusi.

Source code

Pohon directory src/



A. Main.java

```
import java.util.Scanner;

import wordladder.AStar;
import wordladder.GBFS;
import wordladder.GBFS2;
import wordladder.Process;
```

```
import wordladder.UCS;
import wordladder.Result;

public class Main {
   public static void main(String[] args) {
        Scanner userChoice = new Scanner(System.in);

        System.out.println("Choose the algorithm:");
        System.out.println("    1. A* search (optimal and complete)");
        System.out.println("    2. Uniform cost search (optimal and
complete)");
        System.out.println("    3. Greedy best-first search first variant
(not optimal but complete)");
        System.out.println("    4. Greedy best-first search second vairant
(not optimal and not complete)");
        System.out.println("Note:   ");
        System.out.println("Optimal: Always finds the best solution.");
        System.out.println("Complete: Always finds a solution if there are
any.\n");


        System.out.print("Algorithm choice: ");
        int algorithmChoice = userChoice.nextInt();
        if (algorithmChoice < 1 || algorithmChoice > 4) {
            System.out.println("Invalid choice.");
            System.exit(0);
        }

        System.out.print("Starting word: ");
        String source = userChoice.next();

        if (!Process.checkLength(source.length())) {
            System.out.println("No word with length " + source.length() + "
is in the dictionary.");
            System.exit(0);
        }

        Process.createGraph(source.length());
        if (!Process.checkWord(source)) {
            System.out.println(source + " is not in the dictionary.");
```

```
            System.exit(0);
        }

        System.out.print("Ending word: ");
        String destination = userChoice.next();
        userChoice.close();

        if (source.length() != destination.length()) {
            System.out.println("The lengths differ.");
            System.exit(0);
        }
        if (!Process.checkWord(destination)) {
            System.out.println(destination + " is not in the dictionary.");
            System.exit(0);
        }

        Result result = null;

        long startTime = System.nanoTime();
        if (algorithmChoice == 1) {
            result = AStar.solve(source, destination);
        } else if (algorithmChoice == 2) {
            result = UCS.solve(source, destination);
        } else if (algorithmChoice == 3) {
            result = GBFS.solve(source, destination);
        } else if (algorithmChoice == 4) {
            result = GBFS2.solve(source, destination);
        } else {
            System.out.println("Invalid algorithm choice");
            System.exit(0);
        }
        long endTime = System.nanoTime();

        long duration = (endTime - startTime) / 1000000;
        if (result.getPath() != null) {
            System.out.println(result.getPath());
            System.out.println("Number of nodes in the path: " +
result.getPath().size());
        } else {
            System.out.println("Path not found");
```

```
        }
        System.out.println("Total nodes visited: " +
result.getTotalNodesVisited());
        System.out.println("Time taken: " + duration + " ms");
    }
}
```

Class Main memiliki fungsi main di mana akan dilakukan user input, pengecekan user input, menjalankan algoritma yang dipilih, dan mengeluarkan hasil.

B. Process.java

```
package wordladder;

import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.TreeSet;

public class Process {
    public static Map<String, List<String>> adjacency = new HashMap<>();

    public static boolean checkLength(int length) {
        String pathLocation = String.format("%s%d%s",
"src/wordladder/dictionary/word_size_", length, ".txt");
        File file = new File(pathLocation);
        if (file.exists()) {
            return true;
        } else {
            return false;
```

```
        }
    }

    public static void createGraph(int length) {
        String pathLocation = String.format("%s%d%s",
"src/wordladder/dictionary/word_size_", length, ".txt");
        Path path = Paths.get(pathLocation);
        try {
            List<String> lines = Files.readAllLines(path);
            Set<String> treeSet = new TreeSet<>();
            treeSet.addAll(lines);
            for (String line : lines) {
                adjacency.put(line, new ArrayList<>());
            }

            for (int i = 0; i < lines.size(); i++) {
                String line = lines.get(i);
                StringBuilder sb = new StringBuilder(line);
                for (int j = 0; j < length; j++) {
                    char original = sb.charAt(j);
                    for (char k = 'a'; k <= 'z'; k++) {
                        if (k == original)
                            continue;
                        sb.setCharAt(j, k);
                        String to = sb.toString();
                        if (treeSet.contains(to)) {
                            adjacency.get(line).add(to);
                        }
                    }
                    sb.setCharAt(j, original);
                }
                adjacency.get(line).sort(Comparator.naturalOrder());
            }
        } catch (IOException e) {

        }
    }
    public static boolean checkWord(String word) {
        return adjacency.containsKey(word);
    }
```

```
}
```

Class Process memiliki 3 fungsi:

- Membaca suatu dictionary yang diberikan dan mengkonstruksi graf dengan representasi adjacency list.
- Mengecek apakah terdapat kata dengan panjang tertentu dalam dictionary.
- Mengecek apakah terdapat suatu kata dalam dictionary.

C. Result.java

```
package wordladder;

import java.util.List;

public class Result {
   private List<String> path;
   private int totalNodesVisited;

   public Result(List<String> path, int totalNodesVisited) {
       this.path = path;
       this.totalNodesVisited = totalNodesVisited;
   }

   public List<String> getPath() {
       return path;
   }

   public int getTotalNodesVisited() {
       return totalNodesVisited;
   }
}
```

Class Result digunakan untuk sebagai hasil return dari keempat algoritma. Class ini menyimpan dua hal yaitu pathnya jika ketemu (jika tidak, maka path == null) dan jumlah nodes yang dikunjungi. Class ini memiliki constructor dan memiliki getter untuk kedua field yang bersangkutan.

D. Node.java

```
package wordladder;

class Node {
    private String word;
    private int f;

    public Node(String word, int f) {
        this.word = word;
        this.f = f;
    }

    public String getWord() {
        return word;
    }

    public int getF() {
        return f;
    }
}
```

Class Node digunakan untuk menyimpan informasi sebuah node yang diperlukan yaitu kata yang direpresentasikan dan total cost. Terdapat constructor dan getters dalam class ini.

E. AStar.java

```
package wordladder;

import java.util.List;
import java.util.Map;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.HashMap;
import java.util.PriorityQueue;

public class AStar {
    public static Result solve(String source, String destination) {
        PriorityQueue<Node> priorityQueue = new
PriorityQueue<>(Comparator.comparingInt(Node::getF));
```

```
        Map<String, String> cameFrom = new HashMap<>();
        Map<String, Integer> gScore = new HashMap<>();
        Map<String, Integer> fScore = new HashMap<>();

        gScore.put(source, 0);
        fScore.put(source, h(source, destination));
        priorityQueue.add(new Node(source, 0 + h(source, destination)));

        while (!priorityQueue.isEmpty()) {
            Node current = priorityQueue.poll();
            String currentWord = current.getWord();

            if (currentWord.equals(destination)) {
                List<String> path = new ArrayList<>();
                while (!currentWord.equals(source)) {
                    path.add(currentWord);
                    currentWord = cameFrom.get(currentWord);
                }
                path.add(source);
                path = path.reversed();
                return new Result(path, fScore.size());
            }

            // Biar ga visit ulang sehingga dijamin tidak kuadratik.
            if (fScore.get(currentWord) != current.getF()) {
                continue;
            }

            int currentGscore = gScore.get(currentWord);
            for (String neighbour : Process.adjacency.get(currentWord)) {
                int neighbourGscore = currentGscore + 1;

                if (!gScore.containsKey(neighbour) || neighbourGscore <
gScore.get(neighbour)) {
                    cameFrom.put(neighbour, currentWord);
                    gScore.put(neighbour, neighbourGscore);
                    fScore.put(neighbour, neighbourGscore + h(neighbour,
destination));
                    priorityQueue.add(new Node(neighbour, neighbourGscore +
h(neighbour, destination)));
```

```
                }
            }
        }
        return new Result(null, fScore.size());
    }
    private static int h(String current, String destination) {
        int count = 0;
        for (int i = 0; i < current.length(); i++) {
            if (current.charAt(i) != destination.charAt(i)) {
                count++;
            }
        }
        return count;
    }
}
```

Class ini memiliki 2 fungsi yaitu solve() untuk menjalankan algoritmanya dan fungsi heuristic yang didefinisikan seperti penjelasan sebelumnya.

F. UCS.java

```
package wordladder;

import java.util.ArrayDeque;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Queue;
import java.util.Set;
import java.util.TreeSet;

public class UCS {
    public static Result solve(String source, String destination) {
        Queue<String> queue = new ArrayDeque<String>();
        Set<String> visited = new TreeSet<>();
        Map<String, String> cameFrom = new HashMap<>();

        queue.offer(source);
```

```
        visited.add(source);

        while (!queue.isEmpty()) {
            String current = queue.poll();

            for (String neighbour : Process.adjacency.get(current)) {
                if (visited.contains(neighbour)) {
                    continue;
                }

                visited.add(neighbour);
                cameFrom.put(neighbour, current);
                queue.offer(neighbour);

                if (neighbour.equals(destination)) {
                    List<String> path = new ArrayList<>();
                    while (!neighbour.equals(source)) {
                        path.add(neighbour);
                        neighbour = cameFrom.get(neighbour);
                    }
                    path.add(source);
                    path = path.reversed();
                    return new Result(path, visited.size());
                }
            }
        }
        return new Result(null, visited.size());
    }
}
```

Class ini memiliki 1 fungsi yaitu solve() untuk menjalankan algoritmanya.

G. GBFS.java

```
package wordladder;

import java.util.ArrayList;
```

```java
import java.util.Comparator;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.PriorityQueue;

public class GBFS {
   public static Result solve(String source, String destination) {
       PriorityQueue<Node> priorityQueue = new
PriorityQueue<>(Comparator.comparingInt(Node::getF)); // f(n) = h(n) di

// greedy best

// first search
       Map<String, String> cameFrom = new HashMap<>();
       Map<String, Integer> hScore = new HashMap<>();

       hScore.put(source, h(source, destination));
       priorityQueue.add(new Node(source, h(source, destination)));

       while (!priorityQueue.isEmpty()) {
           Node current = priorityQueue.poll();
           String currentWord = current.getWord();

           // Biar ga visit ulang sehingga dijamin tidak kuadratik.
           if (hScore.get(currentWord) != current.getF()) {
               continue;
           }

           for (String neighbour : Process.adjacency.get(currentWord)) {
               if (neighbour.equals(destination)) {
                   cameFrom.put(neighbour, currentWord);
                   List<String> path = new ArrayList<>();
                   while (!neighbour.equals(source)) {
                       path.add(neighbour);
                       neighbour = cameFrom.get(neighbour);
                   }
                   path.add(source);
                   path = path.reversed();
                   return new Result(path, hScore.size());
```

```
                }

                if (!hScore.containsKey(neighbour)) {
                    cameFrom.put(neighbour, currentWord);
                    hScore.put(neighbour, h(neighbour, destination));
                    priorityQueue.add(new Node(neighbour, h(neighbour,
destination)));
                }
            }
        }
        return new Result(null, hScore.size());
    }

    private static int h(String current, String destination) {
        int count = 0;
        for (int i = 0; i < current.length(); i++) {
            if (current.charAt(i) != destination.charAt(i)) {
                count++;
            }
        }
        return count;
    }
}
```

Class ini memiliki 2 fungsi yaitu solve() untuk menjalankan algoritmanya dan fungsi heuristic yang telah diulang lagi demi kejelasan program.

H. GBFS2.java

```
package wordladder;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.TreeSet;
```

```
public class GBFS2 {
   public static boolean dfs(String node, String destination, Set<String>
visited, Map<String, String> cameFrom) {
       if (node.equals(destination)) {
           return true;
       }
       String smallestNeighbour = null;
       int smallestH = 99999999;

       for (String neighbour : Process.adjacency.get(node)) {
           if (smallestNeighbour == null || h(neighbour, destination) <
smallestH) {
               smallestH = h(neighbour, destination);
               smallestNeighbour = neighbour;
           }
       }

       if (smallestNeighbour == null ||
visited.contains(smallestNeighbour)) {
           return false;
       }

       visited.add(smallestNeighbour);
       cameFrom.put(smallestNeighbour, node);
       return dfs(smallestNeighbour, destination, visited, cameFrom);
   }

   public static Result solve(String source, String destination) {
       Set<String> visited = new TreeSet<>();
       Map<String, String> cameFrom = new HashMap<>();

       visited.add(source);
       boolean found = dfs(source, destination, visited, cameFrom);

       if (!found) {
           return new Result(null, visited.size());
       }

       List<String> path = new ArrayList<>();
       String current = destination;
```

```
        while (!current.equals(source)) {
            path.add(current);
            current = cameFrom.get(current);
        }
        path.add(source);
        path = path.reversed();
        return new Result(path, visited.size());
    }

    private static int h(String current, String destination) {
        int count = 0;
        for (int i = 0; i < current.length(); i++) {
            if (current.charAt(i) != destination.charAt(i)) {
                count++;
            }
        }
        return count;
    }
}
```

Class ini memiliki 2 fungsi yaitu solve() untuk menjalankan algoritmanya dan fungsi heuristic yang telah diulang lagi demi kejelasan program.

I. process.py

```
import os
import glob

# Delete previous files

pattern = 'src/wordladder/dictionary/word_size_*.txt'
files_to_delete = glob.glob(pattern)

for file in files_to_delete:
    os.remove(file)
    print(f"Deleted file: {file}")
```

```
# Create new files

d = dict()

with open('src/wordladder/dictionary/dictionary_asisten.txt') as file:
    for line in file:
        line = line.strip()

        if len(line) in d:
            d[len(line)].append(line)
        else:
            d[len(line)] = [line]

for k, v in d.items():
    with open('src/wordladder/dictionary/word_size_' + str(k) + '.txt',
'x') as file:
        file.write('\n'.join(v))
```

Program ini digunakan untuk membaca dictionary (1 file berekstensi “.txt”) dan membuat banyak text file yang membagi dictionary berdasarkan panjang dari kata.

Dalam kasus ini, process.py sudah dijalankan dengan menggunakan dictionary milik asisten (dictionary_asisten.txt) dan telah dihasilkan 7 file yang memiliki format “word_size_[panjang kata].txt”.

Testing

a. Testcase 1

```
 cd /home/azzmi/projects/ITB/Tucil3_13522109 ; /usr/bin/env /home/azzmi/programs/jdk-21.0.1+12/bin/azzmi@azmicomp:~/projects/ITB/Tucil3_13522109$  cd /home/azzmi/projects/ITB/Tucil3_13522109 ; /usr/bin/env /home/azzmi/programs/jdk-21.0.1+12/bin/java -XX:+ShowCodeDetailsInExceptionMessages -cp /hii3_13522109_2f9f519e/bin Main kspaceStorage/25a4463f42c1b5f1ac5e1724a6
t_ws/Tuc
Choose the algorithm:
    1. A* search (optimal and complete)
    2. Uniform cost search (optimal and complete)
    3. Greedy best-first search first variant (not optimal but complete)
    4. Greedy best-first search second vairant (not optimal and not complete)
Note:
Optimal: Always finds the best solution.
Complete: Always finds a solution if there are any.

Algorithm choice: 1
Starting word: purple
Ending word: yellow
[purple, purply, purely, surely, sorely, sorels, morels, motels, botels, betels, bevels, levels, levees, levies, bevies, belies, belles, belled, balled, ballet, ballot, ballon, billon, billow, bellow, yellow]
Number of nodes in the path: 26
Total nodes visited: 5085
Time taken: 59 ms
Memory used: 59275 KB
azzmi@azmicomp:~/projects/ITB/Tucil3_13522109$ ^C

 cd /home/azzmi/projects/ITB/Tucil3_13522109 ; /usr/bin/env /home/azzmi/programs/jdk-21.0.1+12/bin/azzmi@azmicomp:~/projects/ITB/Tucil3_13522109$  cd /home/azzmi/projects/ITB/Tucil3_13522109 ; /usr/bin/env /home/azzmi/programs/jdk-21.0.1+12/bin/java -XX:+ShowCodeDetailsInExceptionMessages -cp /hii3_13522109_2f9f519e/bin Main kspaceStorage/25a4463f42c1b5f1ac5e1724a6
t_ws/Tuc
Choose the algorithm:
    1. A* search (optimal and complete)
    2. Uniform cost search (optimal and complete)
    3. Greedy best-first search first variant (not optimal but complete)
    4. Greedy best-first search second vairant (not optimal and not complete)
Note:
Optimal: Always finds the best solution.
Complete: Always finds a solution if there are any.

Algorithm choice: 2
Starting word: purple
Ending word: yellow
[purple, purply, purely, surely, sorely, sorels, morels, motels, botels, betels, bevels, levels, levees, levies, bevies, belies, belied, belled, balled, ballet, ballot, ballon, billon, billow, bellow, yellow]
Number of nodes in the path: 26
Total nodes visited: 7836
Time taken: 65 ms
Memory used: 59275 KB
azzmi@azmicomp:~/projects/ITB/Tucil3_13522109$ ^C

 cd /home/azzmi/projects/ITB/Tucil3_13522109 ; /usr/bin/env /home/azzmi/programs/jdk-21.0.1+12/bin/azzmi@azmicomp:~/projects/ITB/Tucil3_13522109$  cd /home/azzmi/projects/ITB/Tucil3_13522109 ; /usr/bin/env /home/azzmi/programs/jdk-21.0.1+12/bin/java -XX:+ShowCodeDetailsInExceptionMessages -cp /hii3_13522109_2f9f519e/bin Main kspaceStorage/25a4463f42c1b5f1ac5e1724a6
t_ws/Tuc
Choose the algorithm:
    1. A* search (optimal and complete)
    2. Uniform cost search (optimal and complete)
    3. Greedy best-first search first variant (not optimal but complete)
    4. Greedy best-first search second vairant (not optimal and not complete)
Note:
Optimal: Always finds the best solution.
Complete: Always finds a solution if there are any.

Algorithm choice: 3
Starting word: purple
Ending word: yellow
[purple, purply, purely, surely, sorely, sorels, morels, motels, botels, betels, bedels, bedell, befell, refell, refill, refile, revile, revise, remise, remiss, remits, demits, damies, danies, genies, gentes, rentes, renter, venter, vented, vetted, watted, welted, walled, felled, filled, filles, fillos, hillos, hallos, hallow, fallow, fellow, yellow]
Number of nodes in the path: 44
Total nodes visited: 450
Time taken: 12 ms
Memory used: 58322 KB
azzmi@azmicomp:~/projects/ITB/Tucil3_13522109$ ^C

 cd /home/azzmi/projects/ITB/Tucil3_13522109 ; /usr/bin/env /home/azzmi/programs/jdk-21.0.1+12/bin/azzmi@azmicomp:~/projects/ITB/Tucil3_13522109$  cd /home/azzmi/projects/ITB/Tucil3_13522109 ; /usr/bin/env /home/azzmi/programs/jdk-21.0.1+12/bin/java -XX:+ShowCodeDetailsInExceptionMessages -cp /hii3_13522109_2f9f519e/bin Main kspaceStorage/25a4463f42c1b5f1ac5e1724a6
t_ws/Tuc
Choose the algorithm:
    1. A* search (optimal and complete)
    2. Uniform cost search (optimal and complete)
    3. Greedy best-first search first variant (not optimal but complete)
    4. Greedy best-first search second vairant (not optimal and not complete)
Note:
Optimal: Always finds the best solution.
Complete: Always finds a solution if there are any.

Algorithm choice: 4
Starting word: purple
Ending word: yellow
Path not found
Total nodes visited: 2
Time taken: 1 ms
Memory used: 58363 KB
```

b. Testcase 2

```
t_ws/Tucil3_13522109_2f9f519e/bin Main
Choose the algorithm:
    1. A* search (optimal and complete)
    2. Uniform cost search (optimal and complete)
    3. Greedy best-first search first variant (not optimal but complete)
    4. Greedy best-first search second vairant (not optimal and not complete)
Note:
Optimal: Always finds the best solution.
Complete: Always finds a solution if there are any.

Algorithm choice: 1
Starting word: tone
Ending word: deaf
[tone, done, dene, dele, delf, deaf]
Number of nodes in the path: 6
Total nodes visited: 262
Time taken: 13 ms
Memory used: 20889 KB
azzmi@aznicomp:~/projects/ITB/Tucil3_13522109$ ^C

 cd /home/azzmi/projects/ITB/Tucil3_13522109 ; /usr/bin/env /home/azzmi/programs/jdk-21.0.1+12/
t_ws/Tucil3_13522109_2f9f519e/bin Main
Choose the algorithm:
    1. A* search (optimal and complete)
    2. Uniform cost search (optimal and complete)
    3. Greedy best-first search first variant (not optimal but complete)
    4. Greedy best-first search second vairant (not optimal and not complete)
Note:
Optimal: Always finds the best solution.
Complete: Always finds a solution if there are any.

Algorithm choice: 2
Starting word: tone
Ending word: deaf
[tone, done, dene, dele, delf, deaf]
Number of nodes in the path: 6
Total nodes visited: 2510
Time taken: 20 ms
Memory used: 20971 KB
azzmi@aznicomp:~/projects/ITB/Tucil3_13522109$ ^C

 cd /home/azzmi/projects/ITB/Tucil3_13522109 ; /usr/bin/env /home/azzmi/programs/jdk-21.0.1+12/
t_ws/Tucil3_13522109_2f9f519e/bin Main
Choose the algorithm:
    1. A* search (optimal and complete)
    2. Uniform cost search (optimal and complete)
    3. Greedy best-first search first variant (not optimal but complete)
    4. Greedy best-first search second vairant (not optimal and not complete)
Note:
Optimal: Always finds the best solution.
Complete: Always finds a solution if there are any.

Algorithm choice: 3
Starting word: tone
Ending word: deaf
[tone, done, dene, dele, delf, deaf]
Number of nodes in the path: 6
Total nodes visited: 62
Time taken: 11 ms
Memory used: 20889 KB
azzmi@aznicomp:~/projects/ITB/Tucil3_13522109$ ^C

 cd /home/azzmi/projects/ITB/Tucil3_13522109 ; /usr/bin/env /home/azzmi/programs/jdk-21.0.1+12/
t_ws/Tucil3_13522109_2f9f519e/bin Main
Choose the algorithm:
    1. A* search (optimal and complete)
    2. Uniform cost search (optimal and complete)
    3. Greedy best-first search first variant (not optimal but complete)
    4. Greedy best-first search second vairant (not optimal and not complete)
Note:
Optimal: Always finds the best solution.
Complete: Always finds a solution if there are any.

Algorithm choice: 4
Starting word: tone
Ending word: deaf
[tone, done, dene, deke, dele, delf, deaf]
Number of nodes in the path: 7
Total nodes visited: 7
Time taken: 2 ms
Memory used: 20889 KB
```

c. Testcase 3

```
Choose the algorithm:
    1. A* search (optimal and complete)
    2. Uniform cost search (optimal and complete)
    3. Greedy best-first search first variant (not optimal but complete)
    4. Greedy best-first search second vairant (not optimal and not complete)
Note:
Optimal: Always finds the best solution.
Complete: Always finds a solution if there are any.

Algorithm choice: 1
Starting word: black
Ending word: white
[black, clack, click, chick, chics, chits, whits, white]
Number of nodes in the path: 8
Total nodes visited: 306
Time taken: 11 ms
Memory used: 34603 KB
azzmi@azmicomp:~/projects/ITB/Tucil3_13522109$ ^C

 cd /home/azzmi/projects/ITB/Tucil3_13522109 ; /usr/bin/env /home/azzmi/programs/jdk-21.0.1+12/bin/ja
t_ws/Tucil3_13522109_2f9f519e/bin Main
Choose the algorithm:
    1. A* search (optimal and complete)
    2. Uniform cost search (optimal and complete)
    3. Greedy best-first search first variant (not optimal but complete)
    4. Greedy best-first search second vairant (not optimal and not complete)
Note:
Optimal: Always finds the best solution.
Complete: Always finds a solution if there are any.

Algorithm choice: 2
Starting word: black
Ending word: white
[black, blank, blink, clink, chink, chine, whine, white]
Number of nodes in the path: 8
Total nodes visited: 1753
Time taken: 11 ms
Memory used: 34603 KB
azzmi@azmicomp:~/projects/ITB/Tucil3_13522109$ ^C

 cd /home/azzmi/projects/ITB/Tucil3_13522109 ; /usr/bin/env /home/azzmi/programs/jdk-21.0.1+12/bin/ja
t_ws/Tucil3_13522109_2f9f519e/bin Main
Choose the algorithm:
    1. A* search (optimal and complete)
    2. Uniform cost search (optimal and complete)
    3. Greedy best-first search first variant (not optimal but complete)
    4. Greedy best-first search second vairant (not optimal and not complete)
Note:
Optimal: Always finds the best solution.
Complete: Always finds a solution if there are any.

Algorithm choice: 3
Starting word: black
Ending word: white
[black, slack, slick, slice, spice, spite, suite, suits, shits, whits, white]
Number of nodes in the path: 11
Total nodes visited: 85
Time taken: 10 ms
Memory used: 34603 KB
azzmi@azmicomp:~/projects/ITB/Tucil3_13522109$ ^C

 cd /home/azzmi/projects/ITB/Tucil3_13522109 ; /usr/bin/env /home/azzmi/programs/jdk-21.0.1+12/bin/ja
t_ws/Tucil3_13522109_2f9f519e/bin Main
Choose the algorithm:
    1. A* search (optimal and complete)
    2. Uniform cost search (optimal and complete)
    3. Greedy best-first search first variant (not optimal but complete)
    4. Greedy best-first search second vairant (not optimal and not complete)
Note:
Optimal: Always finds the best solution.
Complete: Always finds a solution if there are any.

Algorithm choice: 4
Starting word: black
Ending word: white
Path not found
Total nodes visited: 5
Time taken: 1 ms
Memory used: 34384 KB
```

d. Testcase 4

```
Choose the algorithm:
    1. A* search (optimal and complete)
    2. Uniform cost search (optimal and complete)
    3. Greedy best-first search first variant (not optimal but complete)
    4. Greedy best-first search second vairant (not optimal and not complete)
Note:
Optimal: Always finds the best solution.
Complete: Always finds a solution if there are any.

Algorithm choice: 1
Starting word: waited
Ending word: wallet
[waited, wailed, walled, wallet]
Number of nodes in the path: 4
Total nodes visited: 38
Time taken: 9 ms
Memory used: 59498 KB
azzmi@azmicomp:~/projects/ITB/Tucil3_13522109$ ^C

 cd /home/azzmi/projects/ITB/Tucil3_13522109 ; /usr/bin/env /home/azzmi/programs/jdk-21.0.1+12/bin/ja
t_ws/Tucil3_13522109_2f9f519e/bin Main
Choose the algorithm:
    1. A* search (optimal and complete)
    2. Uniform cost search (optimal and complete)
    3. Greedy best-first search first variant (not optimal but complete)
    4. Greedy best-first search second vairant (not optimal and not complete)
Note:
Optimal: Always finds the best solution.
Complete: Always finds a solution if there are any.

Algorithm choice: 2
Starting word: waited
Ending word: wallet
[waited, wailed, walled, wallet]
Number of nodes in the path: 4
Total nodes visited: 162
Time taken: 3 ms
Memory used: 58226 KB
azzmi@azmicomp:~/projects/ITB/Tucil3_13522109$ ^C

 cd /home/azzmi/projects/ITB/Tucil3_13522109 ; /usazzmi@azmicomp:~/projects/ITB/Tucil3_13522109$  cd
Choose the algorithm:
    1. A* search (optimal and complete)
    2. Uniform cost search (optimal and complete)
    3. Greedy best-first search first variant (not optimal but complete)
    4. Greedy best-first search second vairant (not optimal and not complete)
Note:
Optimal: Always finds the best solution.
Complete: Always finds a solution if there are any.

Algorithm choice: 3
Starting word: waited
Ending word: wallet
[waited, wailed, walled, wallet]
Number of nodes in the path: 4
Total nodes visited: 35
Time taken: 8 ms
Memory used: 60323 KB
azzmi@azmicomp:~/projects/ITB/Tucil3_13522109$ ^C

 cd /home/azzmi/projects/ITB/Tucil3_13522109 ; /usr/bin/env /home/azzmi/programs/jdk-21.0.1+12/bin/ja
t_ws/Tucil3_13522109_2f9f519e/bin Main
Choose the algorithm:
    1. A* search (optimal and complete)
    2. Uniform cost search (optimal and complete)
    3. Greedy best-first search first variant (not optimal but complete)
    4. Greedy best-first search second vairant (not optimal and not complete)
Note:
Optimal: Always finds the best solution.
Complete: Always finds a solution if there are any.

Algorithm choice: 4
Starting word: waited
Ending word: wallet
[waited, wailed, walled, wallet]
Number of nodes in the path: 4
Total nodes visited: 4
Time taken: 2 ms
Memory used: 58367 KB
```

e. Testcase 5

```
Choose the algorithm:
    1. A* search (optimal and complete)
    2. Uniform cost search (optimal and complete)
    3. Greedy best-first search first variant (not optimal but complete)
    4. Greedy best-first search second vairant (not optimal and not complete)
Note:
Optimal: Always finds the best solution.
Complete: Always finds a solution if there are any.

Algorithm choice: 1
Starting word: guide
Ending word: goods
[guide, guids, grids, grads, goads, goods]
Number of nodes in the path: 6
Total nodes visited: 68
Time taken: 15 ms
Memory used: 34471 KB
azzmi@azmicomp:~/projects/ITB/Tucil3_13522109$ ^C

 cd /home/azzmi/projects/ITB/Tucil3_13522109 ; /usr/bin/env /home/azzmi/programs/jdk-21.0.1+12/bin/j
t_ws/Tucil3_13522109_2f9f519e/bin Main
Choose the algorithm:
    1. A* search (optimal and complete)
    2. Uniform cost search (optimal and complete)
    3. Greedy best-first search first variant (not optimal but complete)
    4. Greedy best-first search second vairant (not optimal and not complete)
Note:
Optimal: Always finds the best solution.
Complete: Always finds a solution if there are any.

Algorithm choice: 2
Starting word: guide
Ending word: goods
[guide, glide, glade, glads, goads, goods]
Number of nodes in the path: 6
Total nodes visited: 269
Time taken: 3 ms
Memory used: 34122 KB
azzmi@azmicomp:~/projects/ITB/Tucil3_13522109$ ^C

 cd /home/azzmi/projects/ITB/Tucil3_13522109 ; /usr/bin/env /home/azzmi/programs/jdk-21.0.1+12/bin/j
t_ws/Tucil3_13522109_2f9f519e/bin Main
Choose the algorithm:
    1. A* search (optimal and complete)
    2. Uniform cost search (optimal and complete)
    3. Greedy best-first search first variant (not optimal but complete)
    4. Greedy best-first search second vairant (not optimal and not complete)
Note:
Optimal: Always finds the best solution.
Complete: Always finds a solution if there are any.

Algorithm choice: 3
Starting word: guide
Ending word: goods
[guide, guids, grids, grads, goads, goods]
Number of nodes in the path: 6
Total nodes visited: 26
Time taken: 13 ms
Memory used: 34462 KB
azzmi@azmicomp:~/projects/ITB/Tucil3_13522109$ ^C

 cd /home/azzmi/projects/ITB/Tucil3_13522109 ; /usr/bin/env /home/azzmi/programs/jdk-21.0.1+12/bin/j
t_ws/Tucil3_13522109_2f9f519e/bin Main
Choose the algorithm:
    1. A* search (optimal and complete)
    2. Uniform cost search (optimal and complete)
    3. Greedy best-first search first variant (not optimal but complete)
    4. Greedy best-first search second vairant (not optimal and not complete)
Note:
Optimal: Always finds the best solution.
Complete: Always finds a solution if there are any.

Algorithm choice: 4
Starting word: guide
Ending word: goods
[guide, guids, grids, grads, goads, goods]
Number of nodes in the path: 6
Total nodes visited: 6
Time taken: 2 ms
Memory used: 34384 KB
```

f. Testcase 6

```
Choose the algorithm:
    1. A* search (optimal and complete)
    2. Uniform cost search (optimal and complete)
    3. Greedy best-first search first variant (not optimal but complete)
    4. Greedy best-first search second vairant (not optimal and not complete)
Note:
Optimal: Always finds the best solution.
Complete: Always finds a solution if there are any.

Algorithm choice: 1
Starting word: angry
Ending word: happy
Path not found
Total nodes visited: 1
Time taken: 7 ms
Memory used: 34603 KB
azzmi@azmicomp:~/projects/ITB/Tucil3_13522109$ ^C

 cd /home/azzmi/projects/ITB/Tucil3_13522109 ; /usr/bin/env /home/azzmi/programs/jdk-21.0.1+12/bin
t_ws/Tucil3_13522109_2f9f519e/bin Main
Choose the algorithm:
    1. A* search (optimal and complete)
    2. Uniform cost search (optimal and complete)
    3. Greedy best-first search first variant (not optimal but complete)
    4. Greedy best-first search second vairant (not optimal and not complete)
Note:
Optimal: Always finds the best solution.
Complete: Always finds a solution if there are any.

Algorithm choice: 2
Starting word: angry
Ending word: happy
Path not found
Total nodes visited: 1
Time taken: 1 ms
Memory used: 34291 KB
azzmi@azmicomp:~/projects/ITB/Tucil3_13522109$ ^C

 cd /home/azzmi/projects/ITB/Tucil3_13522109 ; /usr/bin/env /home/azzmi/programs/jdk-21.0.1+12/bin
t_ws/Tucil3_13522109_2f9f519e/bin Main
Choose the algorithm:
    1. A* search (optimal and complete)
    2. Uniform cost search (optimal and complete)
    3. Greedy best-first search first variant (not optimal but complete)
    4. Greedy best-first search second vairant (not optimal and not complete)
Note:
Optimal: Always finds the best solution.
Complete: Always finds a solution if there are any.

Algorithm choice: 3
Starting word: angry
Ending word: happy
Path not found
Total nodes visited: 1
Time taken: 8 ms
Memory used: 34384 KB
azzmi@azmicomp:~/projects/ITB/Tucil3_13522109$ ^C

 cd /home/azzmi/projects/ITB/Tucil3_13522109 ; /usr/bin/env /home/azzmi/programs/jdk-21.0.1+12/bin
t_ws/Tucil3_13522109_2f9f519e/bin Main
Choose the algorithm:
    1. A* search (optimal and complete)
    2. Uniform cost search (optimal and complete)
    3. Greedy best-first search first variant (not optimal but complete)
    4. Greedy best-first search second vairant (not optimal and not complete)
Note:
Optimal: Always finds the best solution.
Complete: Always finds a solution if there are any.

Algorithm choice: 4
Starting word: angry
Ending word: happy
Path not found
Total nodes visited: 1
Time taken: 1 ms
Memory used: 34253 KB
```

Perbandingan Solusi

a. Optimalitas

Dari optimalitas, hanya A* dan UCS yang memberikan jawaban yang optimal jika eksis.

Kedua varian greedy tidak optimal sebab terdapat counterexample di mana pemilihan node yang terdekat tidak selalu memberikan solusi optima seperti pada testcase 1.

b. Completeness

Dari completeness, hanya A*, UCS, dan Greedy best-first search varian pertama yang complete.

c. Waktu eksekusi

Secara teori, UCS memiliki kompleksitas waktu lebih baik daripada A* pada permainan ini karena UCS menggunakan queue biasa sehingga operasi push dan pop dapat dilakukan dalam waktu konstan.

Namun, dalam uji testcase A* cenderung lebih cepat daripada UCS. Hal ini dikarenakan A* lebih cepat sampai solusi karena fungsi heuristik mempercepat algoritma menemukan solusi.

Pada algoritma greedy, kedua varian sangat cepat dikarenakan fungsi heuristik yang membantu dan algoritma lebih simpel (tidak banyak mengecek, dll).

d. Memory yang dibutuhkan

Memory yang dibutuhkan tergantung implementasi yang dibuat. Dalam kasus ini, keempat-empat algoritma menggunakan jumlah memory yang mirip karena mayoritas memori digunakan pada pembuatan graf.

Dari segi kompleksitas ruang, keempat algoritma memiliki kompleksitas ruang yang sama (priority queue, queue, dan map memakan space linier) sehingga memory yang dibutuhkan tergantung implementasi sehingga terdapat perbedaan konstanta faktor antar algoritma.

Lampiran

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

memiliki tampilan GUI		

Link github: https://github.com/RealAzzmi/Tucil3_13522109