Project Report

Data Structure and Algorithms

เรื่อง Word Ladder

เสนอ

รศ.ดร.รังสิพรรณ มฤคทัต

(Assoc.Prof. Rangsipan Marukatat, Ph.D)

จัดทำโดย

นายธนกฤต ชุติวงศ์ธนะพัฒน์	6513112
นายภูรินท์ พงษ์พานิช	6513135
นายจารุภัทร โชติสิทธานันท์	6513161
นางสาวชลิษา บัวทอง	6513163

Department of computer Engineering

Faculty of Engineering, Mahidol University

คำนำ

รายงานฉบับนี้เป็นส่วนหนึ่งของวิชา Data Structure and Algorithms (EGCO 221) โดยคณะผู้จัดทำได้จัดทำขึ้นเพื่ออธิบายการทำงานของโปรแกรมแก้ปัญหา Word Ladder ซึ่งเป็นโปรแกรมที่ใช้ในการคำนวณหาเส้นทางที่สั้นที่สุดที่ในการแปลงคำเริ่มต้น (source word) ไปเป็นคำเป้าหมาย (target word) โดยใช้ Data Structure และ Algorithms มาประยุกต์ใช้ในการแก้ปัญหานี้ โดยในรายงานประกอบไปด้วยคู่มือการใช้งานโปรแกรม การอธิบายการทำงานของ Code และ Algorithms รวมไปถึงข้อจำกัดต่าง ๆ ของโปรแกรม

คณะผู้จัดทำขอขอบพระคุณ รศ.ดร.รังสิพรรณ มฤคทัต ผู้ให้ความรู้ และแนวทางในการศึกษา

สุดท้ายนี้ทางคณะผู้จัดทำหวังว่ารายงานฉบับนี้ จะให้ความรู้และประโยชน์ไม่มากก็น้อยแก่ผู้อ่านทุกท่าน หากมีข้อผิดพลาดประการใด ผู้จัดทำจะขอน้อมรับไว้ และขออภัยมา ณ ที่นี้ด้วย

ขอขอบพระคุณ

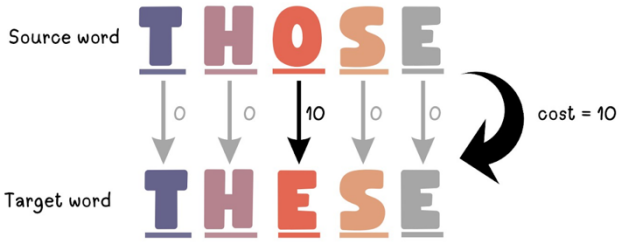
คณะผู้จัดทำ

สารบัญ

เกี่ยวกับโปรแกรม

โปรแกรมนี้เป็นการแก้ปัญหา Word Ladder โดยโปรแกรมจะทำการหาเส้นทางที่สั้นที่สุด (shortest path) ในการแปลงคำเริ่มต้น (source word) ไปเป็นคำเป้าหมาย (target word) โดยขั้นตอนการแปลงคำนั้นจะประกอบไปด้วย 2 รูปแบบ คือ

1. Ladder Step เป็นการแปลงตัวอักษรเพียงตัวเดียวต่อครั้ง ซึ่งจะนับค่าการเปลี่ยนแปลง (Transformation cost) จากระยะห่างของตัวอักษรที่เปลี่ยนไป ดังแสดงในตัวอย่าง

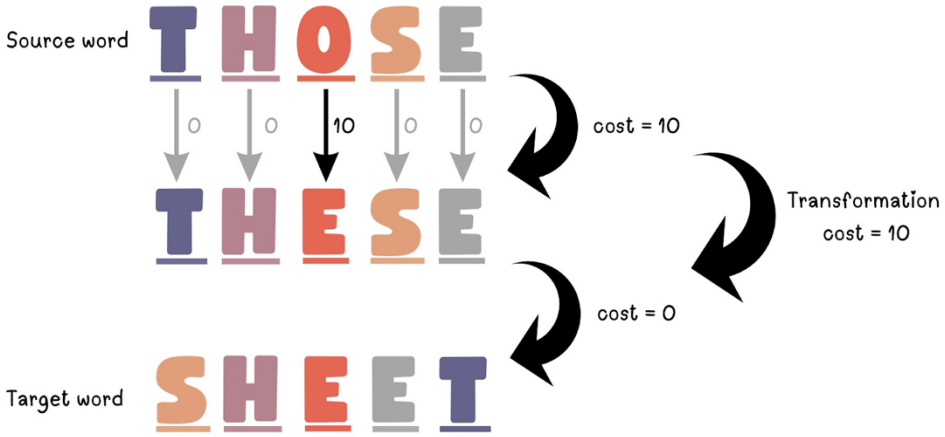


2. Elevetor Step ใช้เมื่อคำ 2 คำ สามารถสลับตำแหน่งตัวอักษรกันเองได้ (permutation) ซึ่งค่าการเปลี่ยนแปลง (Transformation cost) จะมีค่าเป็น 0 ดังแสดงในตัวอย่าง

โดยผู้ใช้งานโปรแกรมนี้จะสามารถเลือกคำเริ่มต้น (source word) และคำเป้าหมาย (target word) เองได้ เช่น

คำเริ่มต้น (source word) : those

คำเป้าหมาย (target word) : sheet



คู่มือการใช้งาน (User Manual)

1. ให้ผู้ใช้ทำการพิมพ์ชื่อไฟล์ที่ต้องการใช้งาน

```
Enter word file = words_5757.txt
```

หมายเหตุ : หากชื่อไฟล์ไม่ตรงกับไฟล์ที่มีอยู่จะไม่สามารถเปิดไฟล์ได้

```
Enter word file = wrds_5757.txt
File not found.
Enter word file =
```

2. โปรแกรมจะแสดงตัวเลือก 3 ตัวเลือก เพื่อให้ผู้ใช้เลือกใช้งานได้ตามต้องการโดยสามารถพิมพ์ได้ทั้งตัวอักษรพิมพ์เล็กและพิมพ์ใหญ่ ดังนี้

```
Enter menu >> (S = search, L = ladder, Q = quit)
```

2.1 พิมพ์ตัวอักษร S (S = search)

เพื่อ search หาคำศัพท์จากไฟล์ได้ เช่น พิมพ์ to โปรแกรมจะทำการ search หาคำศัพท์ที่ขึ้นต้นด้วยตัวอักษร to จากในไฟล์แล้วแสดงผลออกมาดังรูป

```
Search =
to
=== Available words ===
toads       toady       toast       today       toddy       toffs       toffy       togas       toile       toils
toked       token       toker       tokes       tolls       tombs       tomes       tommy       tonal       toned
toner       tones       tongs       tonic       tools       toons       tooth       toots       topaz       toped
toper       topes       topic       topoi       topos       toque       torah       torch       toric       torsi
torso       torte       torts       torus       total       toted       totem       toter       totes       totty
touch       tough       tours       touts       toves       towed       towel       tower       towns       toxic
toxin       toyed       toyer       toyon
```

2.2 พิมพ์ตัวอักษร L (L = ladder)

เพื่อทำการเลือกคำเริ่มต้น (source word) ที่ผู้ใช้ต้องการแปลงเป็นคำเป้าหมาย (target word) โดยการพิมพ์คำเริ่มต้น (source word) และคำเป้าหมาย (target word) ที่ต้องการ เช่น ต้องการแปลงคำว่า corns เป็น swing

```
Enter menu >> (S = search, L = ladder, Q = quit)
l
Enter 5 letter word 1 (Source word) =
corns
Enter 5 letter word 2 (Target word) =
swing
```

หมายเหตุ : หากเป็นคำที่ไม่มีอยู่ในไฟล์ ผู้ใช้จะต้องเลือกคำใหม่

```
Enter menu >> (S = search, L = ladder, Q = quit)
l
Enter 5 letter word 1 (Source word) =
mache
Word not found please enter new word.
Enter 5 letter word 1 (Source word) =
```

2.3 พิมพ์อักษร Q (Q = Quit) เพื่อจบการทำงานของโปรแกรม

```
Enter menu >> (S = search, L = ladder, Q = quit)
q


Process finished with exit code 0
```

3. โปรแกรมจะแสดงผลลัพท์ของการแปลงคำเริ่มต้น (source word) ไปเป็นคำเป้าหมาย (target word) โดยจะแสดงค่าการเปลี่ยนแปลง (Tranformation cost) ของแต่ละ step ในการแปลงคำ หากเป็นการเปลี่ยนตัวอักษรเพียงตัวเดียวจะนับค่าการเปลี่ยนแปลงคำ (Transformation cost) แบบ Ladder step และหากเป็นการสลับตัวแหน่งของตัวอักษร (permutation) จะนับค่าเปลี่ยนแปลงคำ (Transformation cost) แบบ Elevator step แล้วทำการนับค่าการเปลี่ยนแปลง

รวมจากคำเริ่มต้น (source word) ไปเป็นคำเป้าหมาย (target word) รวมทุก step แล้วแสดงผลดังรูป

```
Enter menu >> (S = search, L = ladder, Q = quit)
l
Enter 5 letter word 1 (Source word) =
corns
Enter 5 letter word 2 (Target word) =
swing

corns
corps (ladder   + 2)
crops (elevator + 0)
drops (ladder   + 1)
drips (ladder   + 6)
grips (ladder   + 3)
grins (ladder   + 2)
rings (elevator + 0)
tings (ladder   + 2)
sting (elevator + 0)
swing (ladder   + 3)

Transformation cost = 19
```

หมายเหตุ : หากคำเริ่มต้น (source word) ไม่สามารถแปลงเป็นคำเป้าหมาย (target word) ที่เราเลือกได้ โปรแกรมจะแสดงผลว่าไม่สามารถแปลงคำได้ ดังรูป

```
Enter menu >> (S = search, L = ladder, Q = quit)
l
Enter 5 letter word 1 (Source word) =
edger
Enter 5 letter word 2 (Target word) =
toyon

Cannot transform edger into toyon (Cannot find path)
```

4. โปรแกรมจะทำการวนลูปไปเรื่อยๆจนกว่าผู้ใช้จะพิมพ์ Q เพื่อจบการทำงานของโปรแกรม

```
Enter menu >> (S = search, L = ladder, Q = quit)
q


Process finished with exit code 0
```

Data Structure (โครงสร้างข้อมูล)

1. Package java.util.*

```
import java.util.*;
```

การใช้คำสั่ง import java.util.*; ในโปรแกรม Java จะเป็นการนำเข้าทุกคลาสและอินเทอร์เฟซที่อยู่ในแพ็กเกจ java.util เข้าสู่โปรแกรม เพื่อให้สามารถใช้งานคลาสและอินเทอร์เฟซเหล่านั้นได้โดยไม่ต้องระบุชื่อแพ็กเกจและชื่อคลาสหรืออินเทอร์เฟซทีละตัวแบบชัดเจนในโปรแกรม

ในโปรแกรมนี้ใช้คลาสและอินเทอร์เฟซที่อยู่ในแพ็กเกจ java.util เพื่อการดำเนินการต่าง ๆ เช่น การใช้ Scanner สำหรับการรับข้อมูลจากผู้ใช้ทางคีย์บอร์ด การจัดการกับข้อมูลในรูปแบบ Collection เช่น Set, List เป็นต้น

2. Package org.jgrapht

```
import org.jgrapht.GraphPath;
import org.jgrapht.alg.shortestpath.DijkstraShortestPath;
import org.jgrapht.graph.DefaultWeightedEdge;
import org.jgrapht.graph.SimpleWeightedGraph;
```

เป็นการเรียกใช้งานไลบรารี่ของภาษา Java ที่อยู่ใน Package org.jgrapht ซึ่งนำมาใช้ในการสร้างกราฟ และการดำเนินการต่างๆ บนกราฟ

- import org.jgrapht.GraphPath : ใช้ในการแทนเส้นทางในกราฟ เช่น เส้นทางที่สั้นที่สุด หรือเส้นทางอื่นๆ ในกราฟ
- import org.jgrapht.alg.shortestpath.DijkstraShortestPath :

เพื่อทำการค้นหาเส้นทางสั้นที่สุดในกราฟโดยใช้วิธี Dijkstra's Algorithm

- import org.jgrapht.graph.DefaultWeightedEdge :

เพื่อแทนเส้นเชื่อมในกราฟที่มีน้ำหนักบนเส้นเชื่อม

- import org.jgrapht.graph.SimpleWeightedGraph :

เพื่อสร้างกราฟที่มีน้ำหนักบนเส้นเชื่อม (weighted graph)

3. Class Main

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

    public static void CheckEdge(DataSet set){...}

    public static void storeData(Scanner fileScanner, DataSet set) {...}

    public static void Search(DataSet set) {...}

    public static void Ladder(DataSet set) {...}
}
```

main(String[] args) : เมธอดหลักของโปรแกรม ที่จะเริ่มต้นการทำงานของโปรแกรม ในเมธอดนี้จะมีการสร้างอ็อบเจ็ค DataSet(set) และอ่านข้อมูลจากไฟล์ที่ผู้ใช้ระบุ และเริ่มต้นการทำงานตามเมนูที่ผู้ใช้สามารถเลือกได้

```
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    DataSet set = new DataSet();
    String userChoice = "";
    boolean quit = false;
    String path = "src/main/java/Project2_135/words/";
    boolean fileFound = false;
    while (!fileFound) {
        try {
            System.out.print("Enter word file = ");
            String filename = scanner.nextLine();
            Scanner fileScanner = new Scanner(new File( pathname: path + filename));
            storeData(fileScanner, set);
            fileFound = true;
            fileScanner.close();
        } catch (Exception e) {
            System.out.println("File not found.");
        }
    }
```

```
    while (!quit) {
        System.out.println("\nEnter menu >> (S = search, L = ladder, Q = quit)");
        userChoice = scanner.nextLine();
        switch (userChoice.toLowerCase()) {
            case "q":
                quit = true;
                break;
            case "s":
                Search(set);
                break;
            case "l":
                Ladder(set);
                break;
            case "c":
                CheckEdge(set);
                break;
            default:
        }
    }
    System.out.println();
    scanner.close();
}
```

- public static void main(String[] args): เป็นเมธอดหลักที่ถูกเรียกเมื่อโปรแกรมถูกเรียกใช้ โดยเมธอดนี้จะทำหน้าที่ดังนี้
  - สร้าง Scanner object (scanner) เพื่อรับข้อมูลจากผู้ใช้ผ่านทางคีย์บอร์ด
  - สร้าง DataSet object (set) เพื่อใช้ในการเก็บข้อมูลคำศัพท์และการเชื่อมโยงระหว่างคำศัพท์
  - กำหนดตัวแปร userChoice ให้เป็นช่องว่างสำหรับเก็บค่าที่ผู้ใช้เลือกในเมนู
  - กำหนด quit เป็น false เพื่อใช้ในการตรวจสอบเมื่อผู้ใช้ต้องการออกจากโปรแกรม
  - กำหนด path ให้เป็นที่อยู่ของไฟล์คำศัพท์
  - กำหนด fileFound เป็น false เพื่อใช้ในการตรวจสอบว่าไฟล์ถูกพบหรือไม่
  - เริ่มต้นการวนลูปด้วย while (!fileFound) เพื่อให้ผู้ใช้ป้อนชื่อไฟล์คำศัพท์และโปรแกรมจะพยายามเปิดไฟล์
  - ในลูป while นี้จะมีการใช้ try-catch เพื่อจัดการกับข้อผิดพลาดที่เกิดขึ้นเมื่อไฟล์ไม่พบ
  - ในบล็อก try จะมีการรับชื่อไฟล์คำศัพท์จากผู้ใช้และเปิดไฟล์ด้วย Scanner object เพื่ออ่านข้อมูลและเก็บใน DataSet

- เมื่อไฟล์ถูกพบและข้อมูลถูกเก็บเรียบร้อยแล้ว ค่า fileFound จะถูกตั้งค่าเป็น “true” เพื่อออกจากลูป
- เมื่อไฟล์ถูกเรียกใช้เรียบร้อยแล้ว โปรแกรมจะแสดงเมนูให้ผู้ใช้เลือกใช้งาน
- การเลือกใช้งานจะถูกตรวจสอบในลูป while (!quit) โดยการเปรียบเทียบค่า userChoice ที่ผู้ใช้เลือก
- เมื่อผู้ใช้เลือกเมนู “Q” โปรแกรมจะออกจากลูป
- หากผู้ใช้เลือกเมนู “S” จะเรียกใช้เมธอด Search เพื่อค้นหาคำศัพท์ตามรูปแบบที่ผู้ใช้ระบุ
- หากผู้ใช้เลือกเมนู “L” จะเรียกใช้เมธอด Ladder เพื่อค้นหาเส้นทางระหว่างคำศัพท์สองคำ
- หากผู้ใช้เลือกเมนู “C” จะเรียกใช้เมธอด getEdge ที่จะทำการ return edge ทุก edge ที่เชื่อมอยู่กับโหนดนั้นๆ ออกมาเป็น set เพื่อตรวจสอบว่าคำนั้นๆเชื่อมอยู่กับโหนดใดบ้าง
- เมื่อผู้ใช้เลือกใช้งานเสร็จสิ้น โปรแกรมจะปิด Scanner object และจบการทำงาน

ตัวแปรและอาเรย์

- Scanner scanner: เก็บ Scanner object เพื่อรับข้อมูลจากผู้ใช้ผ่านทางคีย์บอร์ด
- DataSet set: เก็บ DataSet object เพื่อใช้ในการเก็บข้อมูลคำศัพท์และการเชื่อมโยงระหว่างคำศัพท์
- String userChoice : เก็บค่าที่ผู้ใช้เลือกในเมนู (S = search, L = ladder, Q = quit)
- boolean quit : เก็บค่าที่บอกว่าผู้ใช้ต้องการออกจากโปรแกรมหรือไม่
- String path : เก็บที่อยู่ของไฟล์คำศัพท์
- boolean fileFound : เก็บค่าที่บอกว่าไฟล์ถูกพบหรือไม่

checkEdge(DataSet set) : มีหน้าที่ในการแสดงข้อมูลของเส้นเชื่อม (edge) ที่เชื่อมต่อกับโหนด (node) ที่ผู้ใช้เลือกจาก input ที่รับเข้ามา

```
public static void CheckEdge(DataSet set){
    Scanner scanner = new Scanner(System.in);
    System.out.println("Select node to show data: ");
    String userInput = scanner.nextLine();
    System.out.println(set.getEdge(userInput));
}
```

- เมธอดนี้จะใช้ Scanner เพื่อรับข้อมูลที่ผู้ใช้ป้อนจากคีย์บอร์ดเพื่อเลือกโหนดที่ต้องการตรวจสอบข้อมูลของเส้นเชื่อม
- หลังจากนั้นโค้ดจะรับ input และใช้เมธอด getEdge ของอ็อบเจ็กต์ set เพื่อดึงเส้นเชื่อมที่เชื่อมต่อกับโหนดที่ผู้ใช้เลือก
- ข้อมูลที่ได้จากการเรียกใช้เมธอด getEdge จะถูกพิมพ์ออกทางหน้าจอเพื่อให้ผู้ใช้เห็น

storeData(Scanner fileScanner, DataSet set) : ใช้สำหรับอ่านข้อมูลจาก Scanner object ที่ใช้เพื่ออ่านไฟล์คำศัพท์ และจัดเก็บข้อมูลนั้นใน DataSet object เพื่อให้สามารถใช้งานได้ในภายหลัง

```
public static void storeData(Scanner fileScanner, DataSet set) {
    // Store data
    while (fileScanner.hasNextLine()) {
        String col = fileScanner.nextLine().trim();
        set.addData(col);
    }
    set.connectVertices();
}
```

public static void storeData(Scanner fileScanner, DataSet set):

- Scanner fileScanner : เป็น Scanner object ที่ใช้ในการอ่านข้อมูลจากไฟล์คำศัพท์ที่ผู้ใช้ป้อนเข้ามา
- DataSet set : เป็น DataSet object ที่ใช้ในการเก็บข้อมูลคำศัพท์

การทำงานของเมธอด

- เมื่อเรียกใช้ storeData จะเริ่มต้นด้วยการวนลูปของ fileScanner โดยใช้ while (fileScanner.hasNextLine()) เพื่อตรวจสอบว่ายังมีบรรทัดในไฟล์ที่ยังไม่ถูกอ่านหรือไม่
- ในลูป มันจะอ่านข้อมูลบรรทัดละบรรทัดจากไฟล์ด้วย fileScanner.nextLine() และเก็บไว้ในตัวแปร col
- ตัวแปร col จะถูกทำการลบช่องว่างด้านหน้าและด้านหลังของข้อความด้วย .trim() เพื่อล้างช่องว่างที่อาจจะเป็นของบรรทัด
- หลังจากนั้น ข้อมูลใน col จะถูกเพิ่มเข้าไปใน set โดยใช้เมธอด addData(col)
- เมื่อการเพิ่มข้อมูลลงใน set เสร็จสิ้นแล้ว เมธอด connectVertices จะถูกเรียกเพื่อเชื่อมโยงข้อมูลระหว่างคำศัพท์ในกราฟ

Search(DataSet set) : ใช้สำหรับค้นหาคำศัพท์ที่ตรงกับรูปแบบที่ผู้ใช้ระบุ (regular expression) และแสดงผลลัพธ์ที่พบออกทางหน้าจอ

```
public static void Search(DataSet set) {
    System.out.println("\nSearch = ");
    Scanner scanner = new Scanner(System.in);
    String regex = scanner.nextLine();
    Set<String> filteredSet = set.regularExpression(regex);
    if(filteredSet.isEmpty()){
        System.out.println("Word not found!!");
        return;
    }
    System.out.println("=== Available words ===");
    int counter = 0;
    List<String> list = new ArrayList<>(filteredSet);
    Collections.sort(list);
    for (String each : list) {
        System.out.print(each + "       ");
        counter++;
        if (counter == 10) {
            System.out.println();
            counter = 0;
        }
    }
    System.out.println();
}
```

public static void Search(DataSet set) :

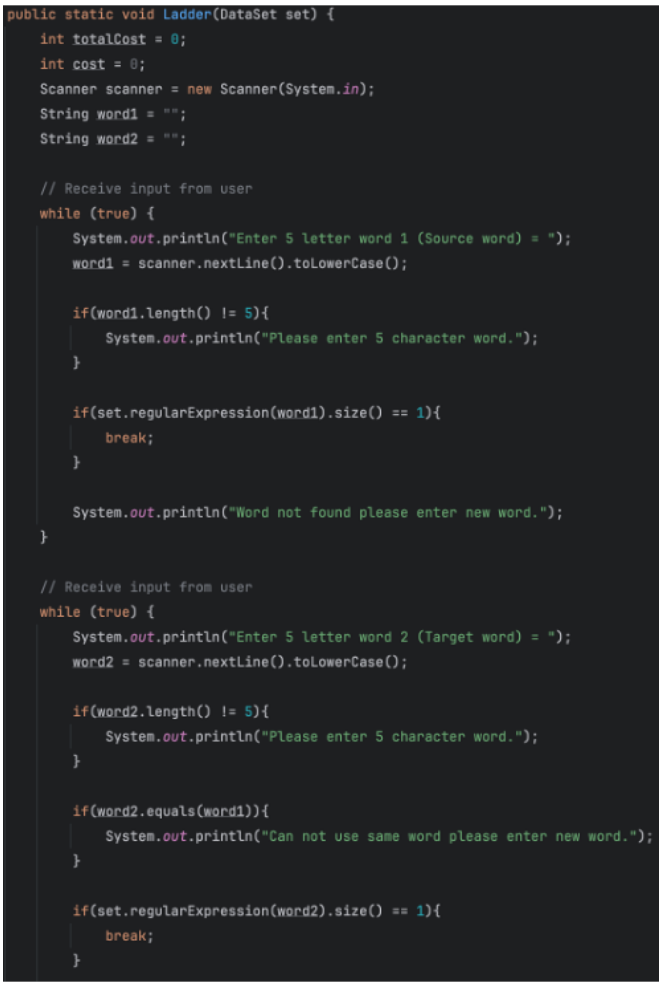
- DataSet set : เป็น DataSet object ที่ใช้ในการค้นหาคำศัพท์

การทำงานของเมธอด

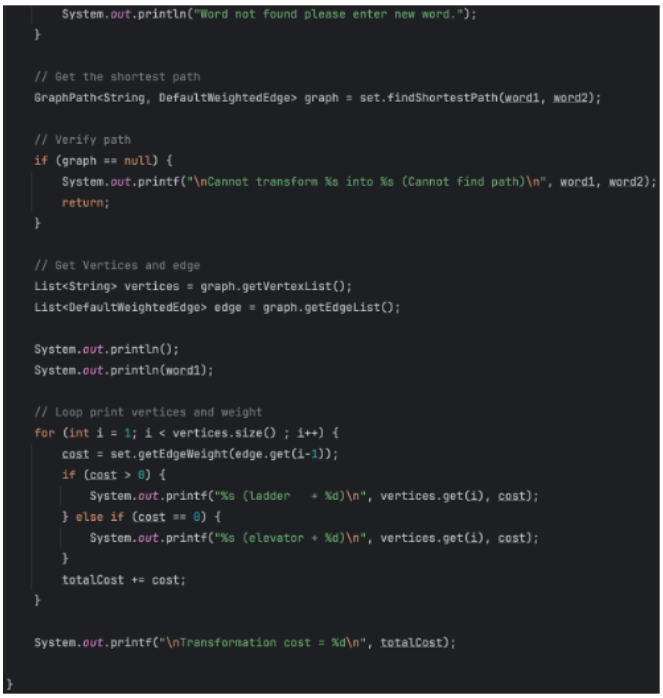
- เมื่อเรียกใช้ Search, โปรแกรมจะแสดงข้อความ Search = เพื่อให้ผู้ใช้ป้อนคำศัพท์ที่ต้องการค้นหา
- หลังจากผู้ใช้ป้อนคำศัพท์ที่ต้องการค้นหาแล้ว โปรแกรมจะใช้ Scanner object (`scanner`) ในการรับค่าข้อมูลที่ผู้ใช้ป้อนเข้ามา
- ค่าที่ผู้ใช้ป้อนจะถูกเก็บไว้ในตัวแปร regex

- เมื่อได้คำศัพท์ที่ต้องการค้นหาแล้ว โปรแกรมจะใช้เมธอด regularExpression ใน DataSet object (`set`) เพื่อค้นหาคำศัพท์ที่ตรงกับที่ผู้ใช้ระบุ
- ผลลัพธ์ที่ได้จะเก็บไว้ในตัวแปร filteredSet ซึ่งเป็น Set ของคำศัพท์ที่ตรงกับ regular expression
- โปรแกรมจะตรวจสอบว่า filteredSet มีขนาดเท่ากับ 0 หรือไม่ ถ้าเท่ากับ 0 แสดงว่าไม่พบคำศัพท์ที่ตรงกับที่ผู้ใช้ระบุ จะแสดงข้อความ “Word not found!!”
- ถ้า filteredSet ไม่เป็นค่าว่าง โปรแกรมจะแสดงรายการคำศัพท์ที่พบออกทางหน้าจอ โดยเรียงลำดับตามตัวอักษรและแสดงทีละ 10 คำศัพท์โดยใช้ลูป for-each และตัวแปร counter เพื่อควบคุมการแสดงผลลัพธ์แบบตาราง และเมื่อ counter ครบ 10 จะขึ้นบรรทัดใหม่ในการแสดงผลลัพธ์

Ladder(DataSet set) : ใช้สำหรับหาเส้นทางที่สั้นที่สุดระหว่างคำศัพท์สองคำและแสดงผลลัพธ์ทางหน้าจอ



```
public static void Ladder(DataSet set) {
    int totalCost = 0;
    int cost = 0;
    Scanner scanner = new Scanner(System.in);
    String word1 = "";
    String word2 = "";

    // Receive input from user
    while (true) {
        System.out.println("Enter 5 letter word 1 (Source word) = ");
        word1 = scanner.nextLine().toLowerCase();

        if(word1.length() != 5){
            System.out.println("Please enter 5 character word.");
        }

        if(set.regularExpression(word1).size() == 1){
            break;
        }

        System.out.println("Word not found please enter new word.");
    }

    // Receive input from user
    while (true) {
        System.out.println("Enter 5 letter word 2 (Target word) = ");
        word2 = scanner.nextLine().toLowerCase();

        if(word2.length() != 5){
            System.out.println("Please enter 5 character word.");
        }

        if(word2.equals(word1)){
            System.out.println("Can not use same word please enter new word.");
        }

        if(set.regularExpression(word2).size() == 1){
            break;
        }
```



```
        System.out.println("Word not found please enter new word.");
    }

    // Get the shortest path
    GraphPath<String, DefaultWeightedEdge> graph = set.findShortestPath(word1, word2);

    // Verify path
    if (graph == null) {
        System.out.printf("\nCannot transform %s into %s (Cannot find path)\n", word1, word2);
        return;
    }

    // Get Vertices and edge
    List<String> vertices = graph.getVertexList();
    List<DefaultWeightedEdge> edge = graph.getEdgeList();

    System.out.println();
    System.out.println(word1);

    // Loop print vertices and weight
    for (int i = 1; i < vertices.size() ; i++) {
        cost = set.getEdgeWeight(edge.get(i-1));
        if (cost > 0) {
            System.out.printf("%s (ladder   + %d)\n", vertices.get(i), cost);
        } else if (cost == 0) {
            System.out.printf("%s (elevator + %d)\n", vertices.get(i), cost);
        }
        totalCost += cost;
    }

    System.out.printf("\nTransformation cost = %d\n", totalCost);
}
```

public static void Ladder(DataSet set):

- DataSet set: เป็น DataSet object ที่ใช้ในการค้นหาเส้นทาง

การทำงานของเมธอด

- เมื่อเรียกใช้ Ladder โปรแกรมจะใช้ Scanner object (`scanner`) ในการรับค่าข้อมูลจากผู้ใช้ เพื่อป้อนคำศัพท์ที่เป็นต้นและปลายทาง
- คำศัพท์ที่ผู้ใช้ป้อนจะถูกเก็บไว้ในตัวแปร word1 และ word2 ตามลำดับ

- โปรแกรมจะตรวจสอบคำศัพท์ที่ผู้ใช้ป้อนว่ามีขนาด 5 ตัวอักษรหรือไม่ ถ้าไม่ใช่ โปรแกรมจะต้องรับข้อมูลใหม่จนกว่าคำศัพท์ที่ผู้ใช้ป้อนจะมีขนาดเป็น 5 ตัวอักษร
- หลังจากที่ได้รับคำศัพท์ที่ถูกต้องทั้งสองคำแล้ว โปรแกรมจะตรวจสอบว่าคำศัพท์ทั้งสองมีอยู่ใน DataSet หรือไม่ และตรวจสอบว่าคำศัพท์สองคำเป็นคำเดียวกันหรือไม่
- หากผู้ใช้ป้อนคำศัพท์ที่ไม่มีอยู่ในกราฟโปรแกรมจะแสดงให้ผู้ใช้ป้อนคำศัพท์คำใหม่
- หากคำศัพท์ทั้งสองคำมีอยู่ใน DataSet และไม่เป็นคำเดียวกัน โปรแกรมจะใช้เมธอด findShortestPath ใน DataSet object เพื่อหาเส้นทางที่สั้นที่สุดระหว่างคำศัพท์สองคำ
- ผลลัพธ์ที่ได้จะเก็บไว้ในตัวแปร graph ซึ่งเป็น GraphPath object ที่ประกอบด้วยเส้นทางที่สั้นที่สุดระหว่างคำศัพท์สองคำ
- หากไม่พบเส้นทางที่สั้นที่สุดระหว่างคำศัพท์สองคำ โปรแกรมจะแสดงข้อความว่า “Cannot transform [word1] into [word2] (Cannot find path)”
- หากพบเส้นทางที่สั้นที่สุดระหว่างคำศัพท์สองคำ โปรแกรมจะแสดงรายละเอียดของเส้นทางนี้ออกทางหน้าจอ โดยแสดงคำศัพท์ที่อยู่ในเส้นทางและค่าความห่างของแต่ละคู่คำที่ถูกบวกเข้ากับตัวแปร totalCost ซึ่งรวมค่าความห่างของคำศัพท์ทุกคู่ที่อยู่ในเส้นทางนี้
- หลังจากนั้น โปรแกรมจะแสดงผลลัพธ์รวมของค่าการเปลี่ยนแปลง ทั้งหมดออกทางหน้าจอ

4. Class DataSet

```
18
19  public class DataSet {
20      private final SimpleWeightedGraph<String, DefaultWeightedEdge> graph = new SimpleWeightedGraph<>(DefaultWeightedEdge.class);
21
22      public void addData(String data) {...}
28
29      public void connectVertices() {...}
53
54      public int getEdgeWeight(DefaultWeightedEdge edge) { return (int) graph.getEdgeWeight(edge); }
57
58      public GraphPath<String, DefaultWeightedEdge> findShortestPath(String startVertex, String targetVertex) {...}
62
63      public int findCost(String word1, String word2) {...}
72
73      private boolean arePermutation(String word1, String word2) {...}
88
89
90      public int isOneCharDiffOrPermutation(String word1, String word2) {...}
110
111     public Set<String> regularExpression(String regex) {...}
114
115     public Set<DefaultWeightedEdge> getEdge(String node) { return graph.edgesOf(node); }
118 }
119
```

เป็นคลาสที่ใช้ในการจัดการข้อมูลและกราฟที่เกี่ยวข้องกับคำศัพท์

addData(String data):

- มีหน้าที่รับข้อมูลคำศัพท์และเพิ่มโหนด (คำศัพท์) ลงในกราฟ
- ทำการตรวจสอบว่าข้อมูลไม่ใช่ค่าว่าง และเพิ่มโหนดใหม่ลงในกราฟ

```java
public class DataSet {
    private final SimpleWeightedGraph<String, DefaultWeightedEdge> graph = new SimpleWeightedGraph<>(
            DefaultWeightedEdge.class);

    public void addData(String data) {
        if (data == null || data.isEmpty()) {
            return;
        }
        graph.addVertex(data.toLowerCase());
    }

    public void connectVertices() {
        List<String> vertices = new ArrayList<>(graph.vertexSet());
        for (int i = 0; i < vertices.size(); i++) {
            for (int j = i + 1; j < vertices.size(); j++) {
                int cost = 0;
                String vertex1 = vertices.get(i);
                String vertex2 = vertices.get(j);
                int flag = isOneCharDiffOrPermutation(vertex1, vertex2);
                if (flag != -1) {
                    DefaultWeightedEdge edge = graph.addEdge(vertex1, vertex2);

                    if (flag == 1) {
                        cost = findCost(vertex1, vertex2);
                    }

                    if (edge != null) {
                        graph.setEdgeWeight(edge, cost);
                    }
                }
            }
        }
    }

    public int getEdgeWeight(DefaultWeightedEdge edge) {
        return (int) graph.getEdgeWeight(edge);
    }

    public GraphPath<String, DefaultWeightedEdge> findShortestPath(String startVertex, String targetVertex) {
        DijkstraShortestPath<String, DefaultWeightedEdge> path = new DijkstraShortestPath<>(graph);
        return path.getPath(startVertex, targetVertex);
    }

    public int findCost(String word1, String word2) {
        int i;
        for (i = 0; i < word1.length(); i++) {
            if (Character.toLowerCase(word1.charAt(i)) != Character.toLowerCase(word2.charAt(i))) {
                break;
            }
        }
        return Math.abs(Character.toLowerCase(word1.charAt(i)) - Character.toLowerCase(word2.charAt(i)));
    }
```

connectVertices():

- มีหน้าที่สร้างเส้นเชื่อมระหว่างโหนด (คำศัพท์) ที่มีความสัมพันธ์กัน โดยวนลูปผ่านโหนดทั้งหมดเพื่อหาคู่โหนดที่มีความสัมพันธ์
- ทำการเพิ่มเส้นเชื่อมระหว่างคู่โหนดดังกล่าว โดยใช้เงื่อนไขการตรวจสอบความสัมพันธ์ระหว่างคู่คำศัพท์

getEdgeWeight(DefaultWeightedEdge edge):

- มีหน้าที่รับเส้นเชื่อมและคืนค่าน้ำหนักบนเส้นเชื่อมนั้น

findShortestPath(String startVertex, String targetVertex):

- มีหน้าที่หาเส้นทางที่สั้นที่สุดระหว่างคำศัพท์เริ่มต้นและคำศัพท์ปลายทาง
- ใช้วิธีการวิเคราะห์ Dijkstra's Algorithm เพื่อค้นหาเส้นทางที่สั้นที่สุด

findCost(String word1, String word2):

- มีหน้าที่คำนวณค่าต่าง ๆ ระหว่างคำศัพท์สองคำ ความแตกต่างระหว่างตัวอักษรแต่ละตัว

```java
private boolean arePermutation(String word1, String word2) {
    char ch1[] = word1.toCharArray();
    char ch2[] = word2.toCharArray();

    // Sort both strings
    Arrays.sort(ch1);
    Arrays.sort(ch2);

    // Compare sorted strings
    for (int i = 0; i < 5; i++)
        if (ch1[i] != ch2[i])
            return false;

    return true;
}

public int isOneCharDiffOrPermutation(String word1, String word2) {

    // Check permutation
    if (arePermutation(word1, word2)) {
        return 2;
    }

    // Check one char diff
    int diffCount = 0;
    for (int i = 0; i < word1.length(); i++) {
        if (word1.charAt(i) != word2.charAt(i)) {
            diffCount++;
        }
    }

    if (diffCount == 1) {
        return 1;
    }
    return -1;
}

public Set<String> regularExpression(String regex) {
    return graph.vertexSet().stream().filter(s -> s.toLowerCase().startsWith(regex.toLowerCase()))
            .collect(Collectors.toSet());
}

public Set<DefaultWeightedEdge> getEdge(String node) {
    return graph.edgesOf(node);
}
}
```

arePermutation(String word1, String word2):

- มีหน้าที่ตรวจสอบว่าสองคำศัพท์เป็นการเรียงลำดับตัวอักษรเดียวกันหรือไม่

isOneCharDiffOrPermutation(String word1, String word2):

- มีหน้าที่ตรวจสอบว่าสองคำศัพท์มีความแตกต่างในตัวอักษรเพียงตัวเดียวหรือไม่ หรือเป็นการเรียงลำดับตัวอักษรเดียวกันหรือไม่

regularExpression(String regex):

- มีหน้าที่ค้นหาคำศัพท์ที่ตรงตามเงื่อนไขที่กำหนดโดยใช้ Regular Expression และคืนค่าเป็นเซ็ตของคำศัพท์ที่ตรงกับเงื่อนไขที่กำหนด

getEdge(String node):

- มีหน้าที่คืนเซ็ตของเส้นเชื่อม (edges) ที่เชื่อมต่อกับจุด (vertex) ที่ระบุด้วยชื่อ node ในกราฟ

addData(String data) : ใช้สำหรับการเพิ่มโหนด (Vertex) เข้าไปใน graph

```
public void addData(String data) {
    if (data == null || data.isEmpty()) {
        return;
    }
    graph.addVertex(data.toLowerCase());
}
```

public void addData(String data)

- รับค่าเป็น string data

การทำงานของเมธอด

- เมื่อเรียกใช้ addData, โปรแกรมจะทำการเช็คว่า data นั้นมีข้อมูลหรือไม่ หากมีข้อมูลจะทำการเพิ่ม vertex ที่ชื่อ data ลงไปในกราฟ

connectVertices() :

```
public void connectVertices() {
    List<String> vertices = new ArrayList<>(graph.vertexSet());
    for (int i = 0; i < vertices.size(); i++) {
        for (int j = i + 1; j < vertices.size(); j++) {
            int cost = 0;
            String vertex1 = vertices.get(i);
            String vertex2 = vertices.get(j);
            int flag = isOneCharDiffOrPermutation(vertex1, vertex2);

            // flag 2 = Permutation, 1 = One Char Diff, -1 = No any condition meet
            if (flag != -1) {
                DefaultWeightedEdge edge = graph.addEdge(vertex1, vertex2);

                if (flag == 1) {
                    cost = findCost(vertex1, vertex2);
                }

                if (edge != null) {
                    graph.setEdgeWeight(edge, cost);
                }
            }
        }
    }
}
```

ใช้เพื่อเชื่อมเส้นระหว่างโหนดในกราฟโดยการสร้างเส้นเชื่อมระหว่างโหนดที่มีความสัมพันธ์ตามเงื่อนไขที่กำหนดไว้ในเมธอด isOneCharDiffOrPermutation(String word1, String word2) และใช้เมธอด findCost(String word1, String word2) เพื่อหาค่าน้ำหนักของเส้นที่เชื่อมระหว่างโหนดนั้น โดยการวนลูปผ่าน list ของโหนดทั้งหมดที่มีอยู่ในกราฟ และสร้างเส้นเชื่อมระหว่างโหนดที่คั่นกันตามเงื่อนไขที่กำหนด ซึ่งตัวแปรที่เกี่ยวข้องกับเมธอดนี้มีดังนี้

1. vertices : เป็นรายการที่เก็บโหนดทั้งหมดในกราฟที่อยู่ในปัจจุบัน

2. cost : เก็บค่าน้ำหนักของเส้นเชื่อมระหว่างโหนดสองโหนด

3. vertex1 : เก็บชื่อของโหนดแรกที่จะเชื่อมเส้น

4. vertex2 : เก็บชื่อของโหนดที่สองที่จะเชื่อมเส้น

5. flag : เก็บสถานะที่บอกถึงความสัมพันธ์ระหว่างคู่โหนด vertex1 และ vertex2 โดยมีค่าเป็น 2 หากเป็น Permutation, 1 หากเป็น One Char Diff, และ -1 หากไม่เข้าเงื่อนไขใดๆ ทั้งสอง

ตัวแปร cost ถูกใช้เพื่อเก็บค่าน้ำหนักของเส้นระหว่างโหนด vertex1 และ vertex2 ซึ่งค่านี้จะถูกคำนวณด้วยเมธอด findCost(String word1, String word2) หากสถานะ flag เป็น 1 ซึ่งหมายถึงมีแค่ตัวอักษรเพียงตัวเดียวที่แตกต่างกันระหว่างคู่จุดนี้

สำหรับ DefaultWeightedEdge edge จะถูกใช้ในการสร้างเส้นเชื่อมระหว่างโหนด vertex1 และ vertex2 ในกราฟ หากสร้างเสร็จสิ้น ก็จะกำหนดค่าน้ำหนักของเส้นดังกล่าวด้วยเมธอด setEdgeWeight() โดยใช้ค่า cost ที่ได้มาก่อนหน้านี้

getEdgeWeight(DefaultWeightEdge edge) : ใช้สำหรับการหา weight ของ edge นั้นๆ

```
public int getEdgeWeight(DefaultWeightedEdge edge) {
    return (int) graph.getEdgeWeight(edge);
}
```

public int getEdgeWeight(DefaultWeightEdge edge)

- รับค่าเป็น DefaultWeightEdge

การทำงานของเมธอด

- เมื่อเรียกใช้ getEdgeWeight โปรแกรมจะทำการคืนค่าน้ำหนักของ edge นั้นๆ โดยหาข้อมูลจากกราฟ

findShortestPath(String startVertex, String targetVertex) : ใช้สำหรับการหาเส้นทางที่สั้นที่สุด

```
public GraphPath<String, DefaultWeightedEdge> findShortestPath(String startVertex, String targetVertex) {
    DijkstraShortestPath<String, DefaultWeightedEdge> path = new DijkstraShortestPath<>(graph);
    return path.getPath(startVertex, targetVertex);
}
```

public GraphPath<String, DefaultWeightedEdge> findShortestPath(String startVertex, String targetVertex)

- รับค่าเป็น ชื่อ ของ Vertex ที่ต้องการเริ่มค้นหา และชื่อของ Vertex ที่ต้องการค้นหา

การทำงานของเมธอด

- เมื่อเรียกใช้ getEdgeWeight, โปรแกรมจะทำการสร้าง object path ไว้สำหรับเก็บเส้นทาง
- โปรแกรมจะทำการคืนค่าเส้นทางที่สั้นที่สุดจาก startVertex ไปยัง targetVertex โดยใช้อัลกอริทึม Dijkstra

findCost(String word1, String word2) : ใช้สำหรับการหาค่าเพื่อไปใส่ใน weight ระหว่าง 2 คำนั้นๆ

```
public int findCost(String word1, String word2) {
    int i;
    for (i = 0; i < word1.length(); i++) {
        if (Character.toLowerCase(word1.charAt(i)) != Character.toLowerCase(word2.charAt(i))) {
            break;
        }
    }
    return Math.abs(Character.toLowerCase(word1.charAt(i)) - Character.toLowerCase(word2.charAt(i)));
}
```

public int findCost(String word1, String word2)

- รับค่าเป็น word ทั้ง 2 คำที่ต้องการเปรียบเทียบ

การทำงานของเมธอด

- เมื่อเรียกใช้ getEdgeWeight โปรแกรมจะทำการ loop เช็คและหาตำแหน่งของตัวอักษรที่แตกต่าง
- โปรแกรมจะทำการ คืนค่าแอบสรูทของการลบ ตำแหน่งตัวอักษร ทั้ง 2 ตัวแบบ ascii

arePermutation(String word1, String word2)

```java
private boolean arePermutation(String word1, String word2) {
    char ch1[] = word1.toCharArray();
    char ch2[] = word2.toCharArray();

    // Sort both strings
    Arrays.sort(ch1);
    Arrays.sort(ch2);

    // Compare sorted strings
    for (int i = 0; i < 5; i++)
        if (ch1[i] != ch2[i])
            return false;

    return true;
}
```

ใช้สำหรับตรวจสอบคำว่า word1 และ word2 เป็นอนุกรมเดียวกันหรือไม่ โดยมีหลักการทำงานดังนี้

1. แปลงทั้งสองคำเป็นอาร์เรย์ของตัวอักษร

2. เรียงลำดับตัวอักษรในทั้งสองอาร์เรย์

3. เปรียบเทียบตัวอักษรที่เรียงลำดับแล้วว่าเหมือนกันหรือไม่

4. คืนค่า true ถ้าตัวอักษรทั้งหมดในอาร์เรย์เหมือนกัน และคืนค่า false ถ้ามีตัวอักษรใดที่ไม่เหมือนกัน

isOneCharDiffOrPermutation(String word1, String word2)

```java
public int isOneCharDiffOrPermutation(String word1, String word2) {

    // Check permutation
    if (arePermutation(word1, word2)) {
        return 2;
    }

    // Check one char diff
    int diffCount = 0;
    for (int i = 0; i < word1.length(); i++) {
        if (word1.charAt(i) != word2.charAt(i)) {
            diffCount++;
        }
    }

    if (diffCount == 1) {
        return 1;
    }
    return -1;
}
```

ใช้สำหรับตรวจสอบว่าคำ word1 และ word2 เป็นการสลับตำแหน่งของคำกันหรือไม่ หรือว่าเป็นคำที่มีตัวอักษรต่างกันเพียงตัวเดียวหรือไม่ โดยมีหลักการทำงานดังนี้

1. เรียกใช้เมธอด arePermutation(word1, word2) เพื่อตรวจสอบว่า word1 และ word2 เป็นการสลับตำแหน่งของคำกันหรือไม่

2. หาก arePermutation(word1, word2) คืนค่าเป็น true คือ word1 และ word2 เป็นการสลับตำแหน่งของคำกัน จะคืนค่า 2

3. หาก arePermutation(word1, word2) คืนค่าเป็น false คือ word1 และ word2 ไม่ใช่การสลับตำแหน่งของคำ จะทำการตรวจสอบว่ามีตัวอักษรที่ต่างกันกี่ตัว

4. หากมีตัวอักษรที่ต่างกันเพียงตัวเดียว จะคืนค่า 1 เพื่อบ่งชี้ว่า word1 และ word2 เป็นคำที่มีการต่างกันเพียงตัวเดียว

5. หากไม่เป็นไปตามเงื่อนไขข้างต้นทั้งสอง จะคืนค่า -1 เพื่อบ่งชี้ว่า word1 และ word2 ไม่ได้มีลักษณะเหมือนกันทั้งสอง

regularExpression(String regex) : ใช้สำหรับการหาโหนดที่มีชื่อคล้ายกับ input

```
public Set<String> regularExpression(String regex) {
    return graph.vertexSet() Set<String>
                .stream() Stream<String>
                .filter(s -> s.toLowerCase()
                            .startsWith(regex.toLowerCase()))
                            .collect(Collectors.toSet());
}
```

public Set<String> regularExpression(String regex)

- รับค่าเป็น string regex

การทำงานของเมธอด

- เมื่อเรียกใช้ regularExpression, โปรแกรมจะคืนค่า เป็น set ของ node ที่มี regular expression ของ input
- regular expression คือ การกำหนดรูปแบบหรือกลุ่มคำ เพื่อเอาไว้ใช้ค้นหาข้อความต่างๆ ตามที่เราต้องการ สามารถค้นหาได้ทั้งอักขระธรรมดา หรือค้นหาความข้อที่กำหนดไว้ หรือจะเป็นอักขระพิเศษก็สามารถค้นหาได้

getEdge(String node) : ใช้สำหรับการหา edge ของโหนดนั้นๆ

```
public Set<DefaultWeightedEdge> getEdge(String node) {
    return graph.edgesOf(node);
}
```

public Set<DefaultWeightEdge> getEdge(String node)

- รับค่าเป็นชื่อ node

การทำงานของเมธอด

- เมื่อเรียกใช้ getEdge โปรแกรมจะคืนค่า เป็น set ของ edge ทั้งหมดของโหนดเป้าหมาย

Graph Data Structures

ประเภทของกราฟ

กราฟที่ใช้ในโปรแกรมนี้เป็นแบบ Undirected Weighted Graph คือกราฟที่ไม่มีทิศทาง (Undirected) และมีน้ำหนักบนเส้นเชื่อม (Weighted)

เหตุผลที่ใช้ Undirected Weighted Graph เป็นเครื่องมือที่เหมาะสมในกรณีนี้เพราะมีความเหมาะสมกับลักษณะของข้อมูลและปัญหาที่ต้องการแก้ไขดังนี้

1. กราฟไม่มีทิศทาง : ในกรณีที่เราต้องการจับคู่คำศัพท์ที่เชื่อมโยงกันโดยไม่มีการกำหนดทิศทางว่าคำศัพท์ใดเป็นคำศัพท์เริ่มต้นและคำศัพท์ใดเป็นคำศัพท์เป้าหมาย การใช้กราฟที่ไม่มีทิศทาง (Undirected) ทำให้สามารถแสดงความสัมพันธ์ระหว่างคำศัพท์ได้โดยไม่จำเป็นต้องกำหนดทิศทางของการหาเส้นทาง
2. กราฟที่มีน้ำหนักบนเส้นเชื่อม : การใช้น้ำหนักบนเส้นเชื่อม (Weight) ช่วยให้เราสามารถแสดงค่าความแตกต่างของตัวอักษรระหว่างคำศัพท์ได้ เช่น คำว่า drops กับคำว่า drips มีการใช้น้ำหนักบนเส้นเชื่อม (Weight) ระหว่างตัวอักษร o กับ i ซึ่งจะได้น้ำหนักบนเส้นเชื่อม (Weight) มีค่าเท่ากับ 6

การสร้างกราฟ

โปรแกรมนี้ใช้ไลบรารี JGraphT เพื่อการจัดการกราฟที่มีน้ำหนักบนเส้นเชื่อม (Weighted Graph) โดยมีคลาส “DataSet” เป็นคลาสที่ใช้ในการสร้างและจัดการกราฟเหล่านี้ โดยมีขั้นตอนการสร้างกราฟดังนี้

1. โปรแกรมสร้างกราฟด้วยคลาส “SimpleWeightedGraph” ซึ่งเป็นกราฟที่มีเส้นเชื่อมแบบไม่มีหัวลูกศร (undirected) และมีน้ำหนักบนเส้นเชื่อม เพื่อใช้เก็บคำศัพท์และค่าความแตกต่างระหว่างคำศัพท์

2. ในการสร้างกราฟ โปรแกรมสร้างกราฟใหม่ด้วยคำสั่ง “SimpleWeightedGraph<String, DefaultWeightedEdge> graph = new SimpleWeightedGraph<>(DefaultWeightedEdge.class);” โดยกำหนดให้โหนดเป็นชนิดข้อมูลเป็น “String” และเส้นเชื่อมเป็น “DefaultWeightedEdge” ซึ่งเป็นคลาสที่ใช้แสดงเส้นเชื่อมในกราฟที่มีน้ำหนัก

3. เมื่อมีคำศัพท์ใหม่ถูกเพิ่มเข้าไปในกราฟด้วยเมทอด “addData(String data)” จะถูกเรียกเพื่อเพิ่มโหนด (คำศัพท์) เข้าไปในกราฟ โดยทำการตรวจสอบก่อนว่าข้อมูลไม่ใช่ค่าว่างและทำการเพิ่มโหนดใหม่ลงในกราฟ

การกำหนดโหนดและเส้นเชื่อมของกราฟ

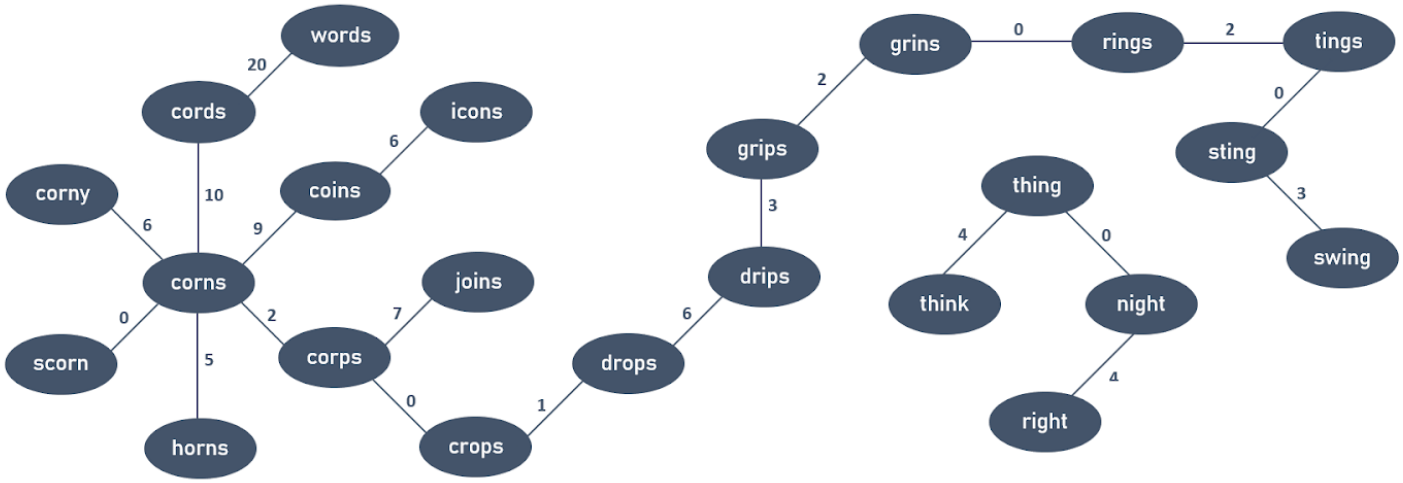
1. เริ่มจากการนำคำศัพท์จาก input file มาสร้างเป็นโหนด โดยใช้ฟังก์ชัน addData
2. สร้างเส้นเชื่อมระหว่างโหนด โดยเงื่อนไขที่ใช้ตรวจสอบก่อนการเพิ่มเส้นเชื่อมระหว่างโหนดหนึ่งกับอีกโหนดหนึ่งคือ
   - การแตกต่างกันเพียงตัวอักษรเดียว (Ladder Step) : ถ้าคำศัพท์สองคำมีการแตกต่างกันเพียงแค่ตัวอักษรเดียว หมายความว่ามีโอกาสเกิดการสร้างเส้นเชื่อมระหว่างคำศัพท์สองคำนั้น โดยในกรณีนี้ โปรแกรมจะตรวจสอบความระยะห่างระหว่างตัวอักษรที่แตกต่างกันและคำนวณค่า (cost) ที่เกิดขึ้นเมื่อทำการสลับตัวอักษรนั้น โดยการคำนวณค่านี้จะใช้ในการกำหนดน้ำหนักของเส้นเชื่อมระหว่างโหนด
   - การเป็น Permutation (Elevator Step) : ถ้าคำศัพท์สองคำเป็น Permutation กัน หมายความว่าสามารถสลับตำแหน่งของตัวอักษรในคำได้และจะได้คำศัพท์อีกคำ ในกรณีนี้

เส้นเชื่อมจะถูกสร้างระหว่างโหนดทั้งสอง เพราะมีความเป็นไปได้ที่จะมีการเรียงลำดับตัวอักษรแตกต่างกันระหว่างคำศัพท์สองคำนั้นอยู่

การเชื่อมโหนด

- เมื่อโหนด (คำศัพท์) ทั้งหมดถูกเพิ่มเข้าไปในกราฟแล้ว โปรแกรมจะทำการสร้างเส้นเชื่อมระหว่างโหนดโดยใช้เมทอด “connectVertices()”
- โปรแกรมจะวนลูปผ่านโหนดทั้งหมดในกราฟเพื่อหาคู่โหนดที่มีความสัมพันธ์ตรงตามเงื่อนไขข้างต้น โดยใช้เมทอด “isOneCharDiffOrPermutation(String word1, String word2)”
- หากพบว่าคู่คำศัพท์มีความสัมพันธ์ตามที่กำหนด โปรแกรมจะสร้างเส้นเชื่อมระหว่างคู่คำศัพท์นั้นด้วยเมทอด “addEdge(String vertex1, String vertex2)”
- จากนั้นจะทำการใส่ค่า weight ของเส้นเชื่อมโดยใช้เมธอด “setEdgeWeight(edge, cost);” โดยถ้าตรงกับเงื่อนไขที่คู่โหนด (คำศัพท์) มีตัวอักษรต่างกันเพียงตัวเดียว (Ladder Step) จะได้ค่า weight เท่ากับระยะห่างระหว่างตัวอักษรที่แตกต่างกัน และถ้าเป็นความสัมพันธ์แบบ permutation (Elevator Step) จะทำการ set ค่า weight เป็นของเส้นเชื่อมนั้นๆเป็น 0

รูปภาพแสดงตัวอย่างกราฟที่สร้างโดยโปรแกรม ซึ่งใช้ input file = “words_5757.txt”



อัลกอริทึมที่ใช้ในการแก้ปัญหา

การแก้ปัญหานี้ทางกลุ่มเลือกใช้ ShortestPathAlgorithm, functional programming, List, SimpleweightGraph

- **ShortestPathAlgorithm**

เลือกใช้ Dijksta's ShortestPath algorithm เพราะว่า weight ของทุก edge ใน graph มีแต่ positive edge

- **List**

vertices เป็น list ใน function connectVertices ที่เอาไว้เก็บค่า ทุกๆ node ของ graph เพื่อเตรียมเข้าสู่กระบวนการเชื่อม node

- **Functional programming**

ทำให้โค้ดเรียบง่ายแค่ในบรรทัดเดียว และไม่มีผลข้างเคียงต่อโค้ดส่วนอื่น

- **SimpleWeightGraph**

เพื่อทำเส้นเชื่อมในกราฟ (edge) โดยทุกเส้นเชื่อมจะมี weight โดยกราฟแสดงเป็น network ของคำ และ edge แทนการ transformation ของคำ โดย weight คือค่าของการ transformation จากคำที่ 1 ไปคำที่ 2

ในการแก้ปัญหานี้เราเลือกใช้โหนดในการเก็บข้อมูลของคำศัพท์ จากนั้นนำข้อมูลของแต่ละคำศัพท์ ไปใช้ในการสร้าง graph แบบ undirected weighted graph โดยแต่ละโหนดจะมี Weighted Edge เชื่อมไปยังโหนดอื่นๆ ที่ตรงกับเงื่อนไขใดเงื่อนไขหนึ่งใน 2 เงื่อนไขดังต่อไปนี้

1. Ladder Step : โหนดทั้ง 2 โหนด จะต้องมีตัวอักษรที่ต่างกันเพียงตัวเดียว กล่าวคือ ทั้ง 2 โหนดต้องมีตัวอักษรที่เหมือนกัน 4 ตัวอักษร (จาก 5 ตัวอักษร) โดยที่ 4 ตัวอักษรนั้นต้องมีตำแหน่งที่ตรงกัน
2. Elevator Step : โหนดทั้ง 2 โหนด จะต้อง Permutation กันหมายถึงการที่โหนดทั้ง 2 โหนดมีตัวอักษรเหมือนกันทั้ง 5 ตัวอักษร แต่การเรียงลำดับของตัวอักษรไม่เหมือนกัน

สำหรับค่า Weight ของ edge ระหว่างโหนดนั้นๆจะคิดตามเงื่อนไขที่กล่าวมาข้างต้น โดยจะให้ความสำคัญกับเงื่อนไขที่ 2 (Elevator step) ก่อนเงื่อนไขที่ 1 (Ladder step) เนื่องจากต้องการหาระยะทางที่สั้นที่สุด หากเป็นเงื่อนไขที่ 2 (Elevator Step) edge จะมี weight เท่ากับ 0 แต่หากเข้าเงื่อนไขที่ 1 (Ladder Step) edge จะมี weight เท่ากับระยะห่างระหว่างตัวอักษรที่แตกต่างกัน

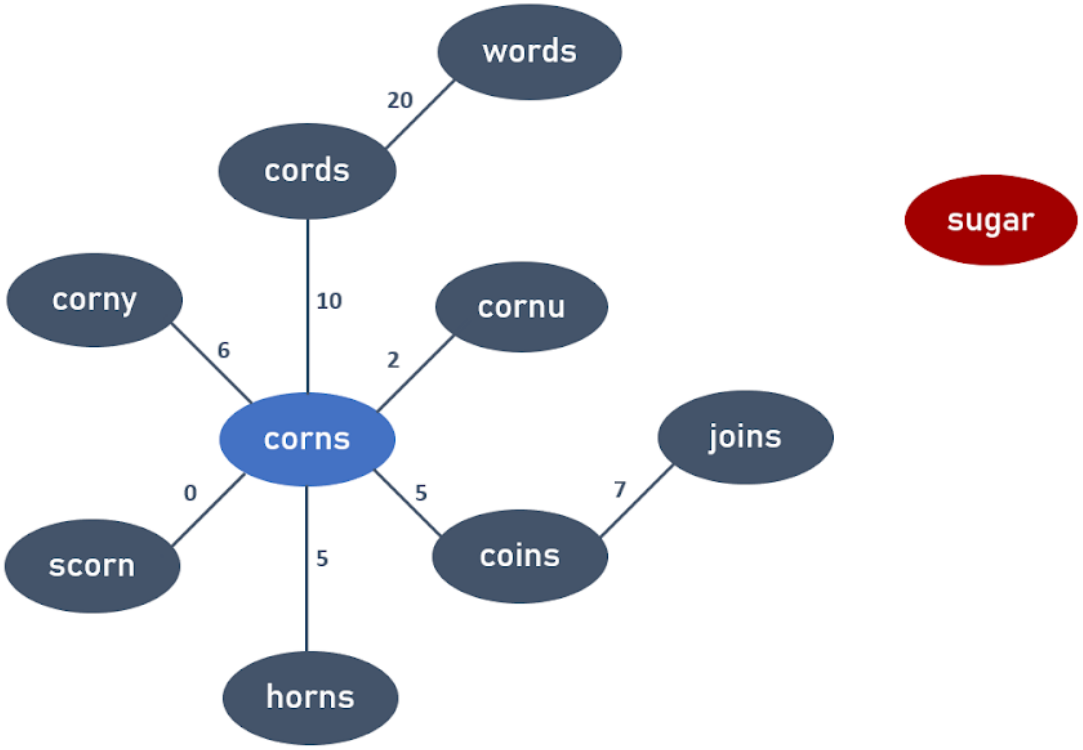
จากนั้นโปรแกรมจะทำการหาเส้นทางที่สั้นที่สุดในการเดินทางจากโหนดต้นทางไปยังโหนดปลายทาง เพื่อให้ได้ค่า weight รวม (Transformation cost) น้อยที่สุด จึงเลือกใช้ Dijkstra's Algorithm ในการแก้ปัญหา เนื่องจากกราฟนี้เป็นแบบ undirected weighted graph และ edge มีค่า weight เป็นค่าบวกทั้งหมด ดังนั้นการใช้ Dijkstra’s Algorithm จึงเหมาะสมอย่างยิ่ง

ขั้นตอนในการประมวลผล

1. นำคำศัพท์จาก input file ไปใส่เป็นโหนดใน undirected weighted graph
2. ทำการ loop และเปรียบเทียบทุกๆโหนดกับโหนดอื่นๆ หากเข้าเงื่อนไขให้ทำการสร้างเชื่อม edge และใส่ค่า weight ระหว่างโหนดนั้นๆ ตามที่กล่าวไว้ข้างต้น
3. เริ่มการประมวลผลหาเส้นทางที่สั้นที่สุดจากโหนดต้นทางไปยังโหนดปลายทางโดยใช้ Dijkstra’s Algorithm

4. ทำการแสดงผลเส้นทางที่สั้นที่สุดระหว่างโหนดต้นทางกับโหนดปลายทาง รวมถึงแสดงค่าการเปลี่ยนแปลง (Transformation cost) ในแต่ละ step และค่าการเปลี่ยนแปลง (Transformation cost) รวมทุก step ให้กับ User

จากการประมวลผลข้างต้นจะได้เส้นทางที่สั้นที่สุดของโหนดต้นทางกับโหนดปลายทาง แต่ในบางกรณีอาจไม่สามารถหาเส้นทางที่สั้นที่สุดสำหรับโหนดต้นทางไปยังโหนดปลายได้ เนื่องจาก 2 โหนดนี้ไม่มีเส้นทางที่สามารถเชื่อมถึงกันได้เลยทำให้ไม่สามารถหาเส้นทางที่สั้นที่สุดระหว่างโหนด 2 โหนดนั้นๆได้ เช่นจาก corns ไปหา sugar ดังแสดงในรูปด้านล่าง



จากการใช้ฟังก์ชัน getEdge ที่จะทำการ return edge ทุก edge ที่เชื่อมอยู่กับโหนดนั้นๆ ออกมาเป็น set เพื่อตรวจสอบว่า corns และ sugar เชื่อมอยู่กับโหนดใดบ้าง จะเห็นได้ว่า sugar เป็นโหนดลอยๆที่ไม่มีการเชื่อมกับโหนดไหนเลย ทำให้โปรแกรมไม่สามารถหาเส้นทางที่สั้นที่สุดจาก โหนดใดๆ หรือจาก corns ไป sugar ได้

```
public static void CheckEdge(DataSet set){
    Scanner scanner = new Scanner(System.in);
    System.out.println("Select node to show data: ");
    String userInput = scanner.nextLine();
    System.out.println(set.getEdge(userInput));
}
```

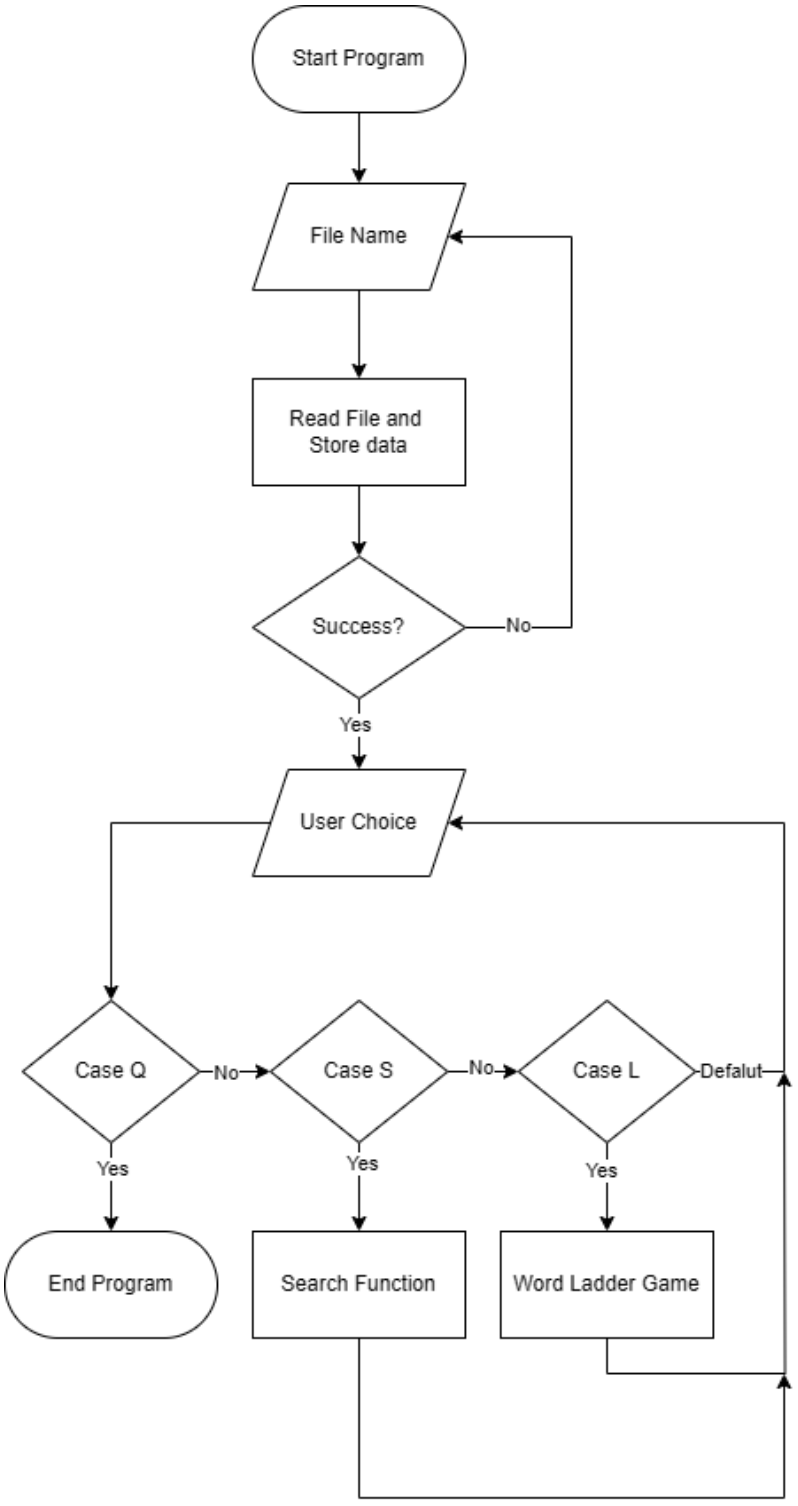
```
public Set<DefaultWeightedEdge> getEdge(String node) {
    return graph.edgesOf(node);
}
```

```
Select node to show data:
corns
[(coins : corns), (horns : corns), (cords : corns), (scorn : corns),
 (cores : corns), (corps : corns), (coons : corns), (corny : corns),
 (morns : corns), (corks : corns), (corns : cornu), (corns : corms)]
```

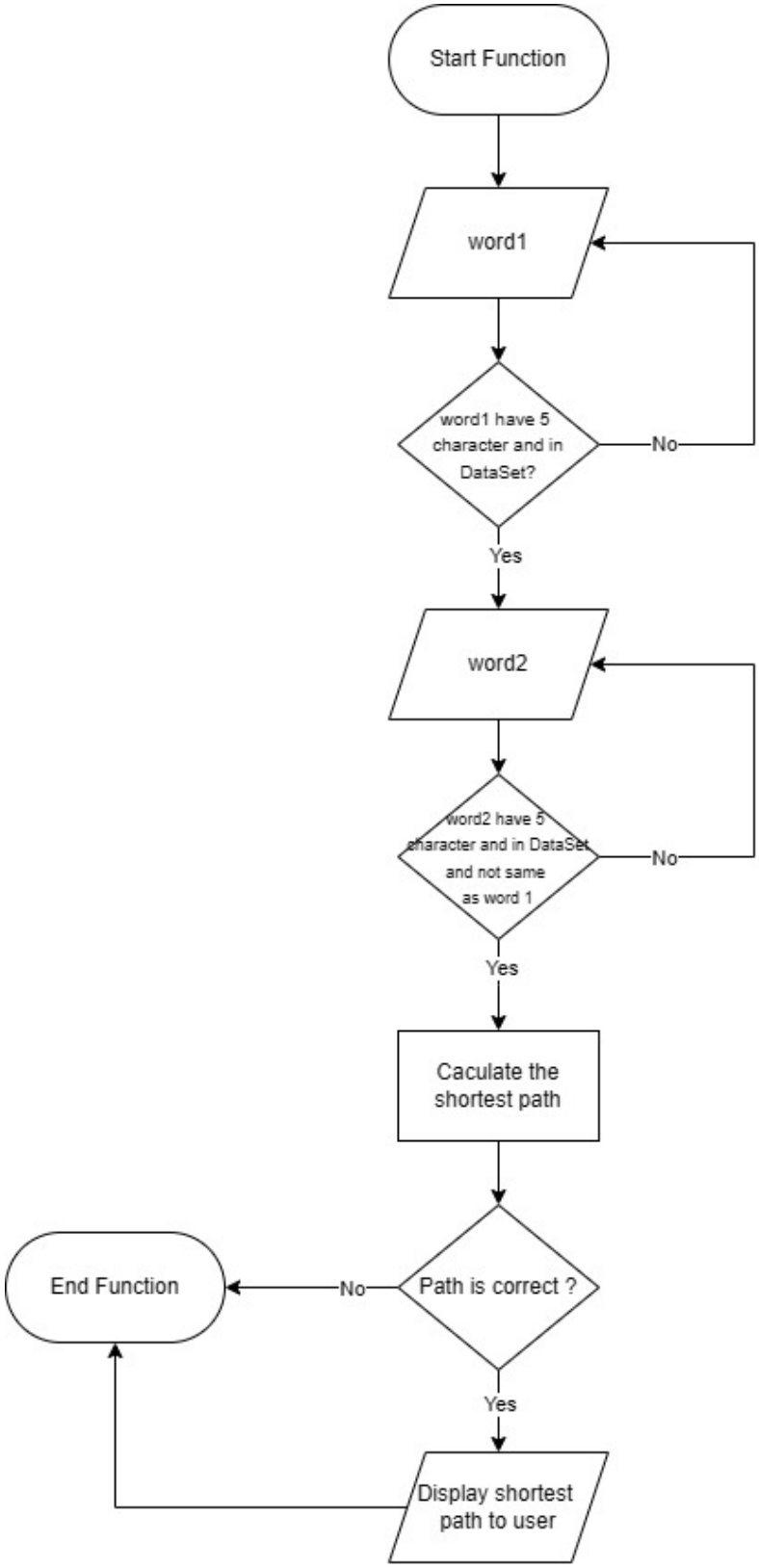
```
Select node to show data:
sugar
[]
```

Algorithm Flow Chart

แผนผังการทำงานของ Main Function



แผนผังการทำงานของ Ladder Function



ข้อจำกัดของโปรแกรม

โปรแกรมนี้รองรับได้แค่ภาษาอังกฤษเท่านั้น

โปรแกรมนี้สามารถใช้คำสั่ง Search ได้แค่ การ Search ตัวอักษรที่ขึ้นต้นของคำนั้นๆ เท่านั้น ตัวอย่างเช่น Search = “ed” จะแสดงผลแค่คำศัพท์ที่ขึ้นต้นด้วย “ed” เช่น “edges edits edger” เป็นต้น แต่จะไม่แสดงผลคำที่มี “ed” เป็นส่วนประกอบอื่นของคำ เช่น “bored needs” เป็นต้น

แหล่งอ้างอิง

1. Weighted graph algorithms. [ออนไลน์].(วันที่ค้าหาข้อมูล : 20 เมษายน 2567). เข้าถึงได้จาก : https://staffwww.fullcoll.edu/aclifton/cs133/lecture-23-weighted-graph-algorithms.html
2. Check if two strings are permutation of each other. [ออนไลน์].(วันที่ค้าหาข้อมูล : 20 เมษายน 2567). เข้าถึงได้จาก : https://www.geeksforgeeks.org/check-if-two-strings-are-permutation-of-each-other/
3. Regular Expression คืออะไร. [ออนไลน์].(วันที่ค้าหาข้อมูล : 20 เมษายน 2567). เข้าถึงได้จาก : https://medium.com/@_trw/regular-expressions-%E0%B8%84%E0%B8%B7%E0%B8%AD%E0%B8%AD%E0%B8%B0%E0%B9%84%E0%B8%A3-2fab4a91ea34