Project Report

เรื่อง

Word Ladder

เสนอ

รศ.ดร.รังสิพรรณ มฤคทัต

จัดทำโดย

ณัฐมน ว่องไววุฒิกุลเดช 6513165

นพรุจ ฤทธิ์เนติกุล 6513168

นิติวดี ลิมปยารยะ 6513169

แองเจลิน่า ชัยนิธิกรรณ 6513178

รายงานนี้เป็นส่วนหนึ่งของวิชา Data Structure and Algorithm

ภาคเรียนที่ 2 ปีการศึกษา 2566

คำนำ

รายงานฉบับนี้เป็นส่วนหนึ่งของวิชา Data Structure and Algorithm (EGCO 221) จัดทำขึ้นเพื่อใช้ประกอบอธิบายการทำงานโปรแกรมของ Word Ladder ประกอบไปด้วยส่วนของคู่มือการใช้งานโปรแกรมเบื้องต้น การสาธิตและการอธิบายในส่วนของ Algorithm รวมไปถึงข้อจำกัดต่างๆ ในการใช้งานโปรแกรม

ทางคณะผู้จัดทำขอขอบพระคุณ รศ.ดร.รังสิพรรณ มฤคทัต ผู้ให้ความรู้ และแนวทางการศึกษา สุดท้ายนี้ทางคณะผู้จัดทำหวังว่ารายงานฉบับนี้จะสามารถเป็นประโยชน์ไม่มากก็น้อยแก่ผู้อ่านทุกท่าน

ขอขอบพระคุณ

คณะผู้จัดทำ

สารบัญ

Manual

```
--- exec:3.1.0:exec (default-cli) @ EGCO221 ---
Choose filename or type any filename :
 1=words_5757.txt , 2=words_250.txt
->words_1111.txt
File words_1111.txt is not found. Please enter a new file name:
```

1. ให้ผู้ใช้งานป้อนชื่อไฟล์ที่จะใช้งานผ่าน keyboard

2. เมื่อป้อนชื่อไฟล์ที่ต้องการเรียบร้อยแล้วให้กดปุ่ม Enter

หมายเหตุ : 1. หากป้อนชื่อไฟล์ไม่ตรงกับไฟล์ที่มีจะไม่สามารถเปิดได้ ต้องทำการพิมพ์ชื่อไฟล์ใหม่อีกครั้ง

2. สามารถป้อนหมายเลข 1 เป็นคำสั่งลัดสำหรับไฟล์ words_5757.txt

และหมายเลข 2 สำหรับไฟล์ word_250.txt ได้ในรอบแรกของการป้อนชื่อไฟล์

```
Enter Menu >>> (s/S = search , l/L = ladder)
               (q/Q = quit , r/R = restart)
```

3. โปรแกรมจะแสดงรูปแบบตัวอักษรในการออกคำสั่งต่างๆ เพื่อให้ผู้ใช้ได้เลือกใช้งานตามความต้องการ โดยสามารถพิมพ์ได้ทั้งตัวอักษรพิมพ์เล็กและพิมพ์ใหญ่ ดังนี้

- พิมพ์ตัวอักษร S เมื่อต้องการค้นหาคำที่มีตัวอักษรที่ต้องการขึ้นต้น
- พิมพ์ตัวอักษร L เมื่อต้องการ transform คำนึงไปยังอีกคำ
- พิมพ์ตัวอักษร Q เพื่อจบการทำงาน
- พิมพ์ตัวอักษร R เพื่อเริ่มต้นโปรแกรมใหม่อีกครั้ง

4. เมื่อป้อนคำสั่งที่ต้องการเรียบร้อยแล้วให้กดปุ่ม Enter

```
Enter Menu >>> (s/S = search , l/L = ladder)
               (q/Q = quit , r/R = restart)
s
word to search =
car
========== Available words ==========
carat  cards  cared  carer  cares  caret  cargo  carne  carny  carob
carol  carom  caron  carps  carpy  carry  carte  carts  carve
Enter Menu >>> (s/S = search , l/L = ladder)
               (q/Q = quit , r/R = restart)
```

เมื่อผู้ใช้พิมพ์ตัวอักษร s โปรแกรมจะรับค่าอักษรที่ผู้ใช้จะกำหนดเพื่อหาคำศัพท์ โดยจะแสดงผลเป็นคำศัพท์ที่มีตัวอักษรขึ้นต้นจากตัวอักษรที่ผู้ใช้กำหนด และจะทำการแสดงคำสั่งทั้งหมดอีกครั้ง พร้อมกับรับคำสั่งใหม่ทันที

```
Enter Menu >>> (s/S = search , l/L = ladder)
               (q/Q = quit , r/R = restart)
s
word to search =
gg
========== Available words ==========

Enter Menu >>> (s/S = search , l/L = ladder)
               (q/Q = quit , r/R = restart)
```

หากไม่เจอคำที่ผู้ใช้ต้องการหาจะแสดงผลเป็น Available words

```
Enter Menu >>> (s/S = search , l/L = ladder)
               (q/Q = quit , r/R = restart)
l
----------word transformation----------
enter word 1 ->
first
 |
 to
 |
enter word 2 ->
spins
start transformation
first
rifts     elevator +0.0
riots     ladder   +9.0
trios     elevator +0.0
tries     ladder   +10.0
tiers     elevator +0.0
piers     ladder   +4.0
spier     elevator +0.0
spies     ladder   +1.0
spins     ladder   +9.0
total cost = +33.0
Enter Menu >>> (s/S = search , l/L = ladder)
               (q/Q = quit , r/R = restart)
```

เมื่อผู้ใช้พิมพ์ตัวอักษร l โปรแกรมจะรับคำศัพท์แรก และคำศัพท์ที่สอง เพื่อทำการแปลงคำศัพท์แรกไปสู่คำศัพท์ที่สอง และทำการแสดงผลกระบวนการที่ใช้ รวมทั้งผลรวมของการแปลงคำศัพท์ทั้งหมด และจะทำการแสดงคำสั่งทั้งหมดอีกครั้ง พร้อมกับรับคำสั่งใหม่ทันที

```
Enter Menu >>> (s/S = search , l/L = ladder)
               (q/Q = quit , r/R = restart)
l
----------word transformation----------
enter word 1 ->
name
 |
 to
 |
enter word 2 ->
spin
No word name
```

ในกรณีที่ผู้ใช้กรอกคำศัพท์ที่ไม่มีอยู่ในไฟล์จะแสดงผลว่า No word .....

หมายเหตุ : หากไม่มีทั้ง 2 คำจะขึ้นเพียงแค่คำศัพท์แรก

```
Enter Menu >>> (s/S = search , l/L = ladder)
               (q/Q = quit , r/R = restart)
l
----------word transformation----------
enter word 1 ->
mommy
 |
 to
 |
enter word 2 ->
first
NO SOLUTION
```

ในกรณีที่คำศัพท์ 2 คำนั้นไม่สามารถแปลงได้โปรแกรมจะแสดงผลว่า NO SOLUTION

```
Enter Menu >>> (s/S = search , l/L = ladder)
               (q/Q = quit , r/R = restart)
q
------------------------------------------------
BUILD SUCCESS
------------------------------------------------
```

เมื่อผู้ใช้ทำการพิมพ์ตัวอักษร q โปรแกรมจะจบการทำงานทันที

```
Enter Menu >>> (s/S = search , l/L = ladder)
               (q/Q = quit , r/R = restart)
r
Choose filename or type any filename :
 1=words_5757.txt , 2=words_250.txt
->
```

เมื่อผู้ใช้ทำการพิมพ์ตัวอักษร r โปรแกรมจะเริ่มต้นใหม่โดยให้ผู้ใช้งานป้อนชื่อไฟล์ที่ต้องการใหม่อีก

```
Please wait...
Enter Menu >>> (s/S = search , l/L = ladder)
               (q/Q = quit , r/R = restart)
h
Incorrect input!!
Enter Menu >>> (s/S = search , l/L = ladder)
               (q/Q = quit , r/R = restart)
```

เมื่อผู้ใช้ทำการพิมพ์ตัวอักษรนอกเหนือจากที่ระบุไว้ ตัวโปรแกรมจะแจ้งและให้พิมพ์คำสั่งใหม่อีกครั้ง

Graph data structure

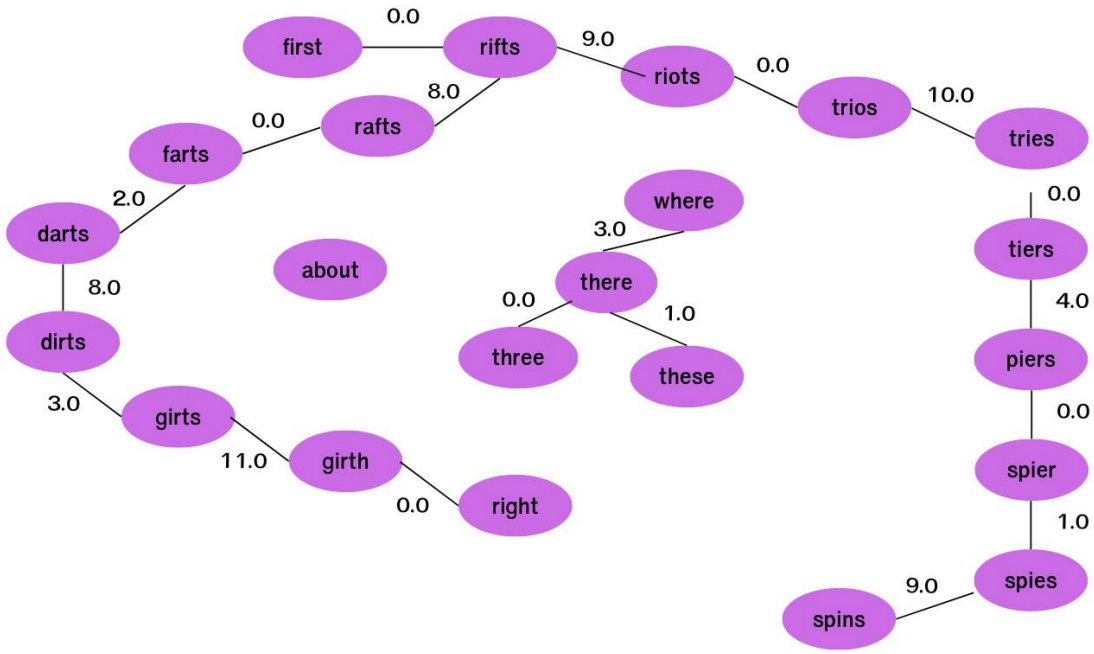
Simple Undirected Weighted Graph

- Object Graph มีชื่อว่า wordGraph โดยถูกสร้างขึ้นในคลาส word_graphs ด้วยการใช้คลาส SimpleWeightedGraph จากไลบรารี JGraphT ซึ่งเป็นกราฟแบบน้ำหนัก โดยประเภทของเส้นในกราฟนี้เป็นแบบไม่กำหนดทิศทาง (undirected) ซึ่งแต่ละเส้นมีน้ำหนักเพื่อบ่งบอกถึง Cost
- เหตุผลที่เลือกใช้ SimpleWeightedGraph คือ เราต้องการ shortest path เพื่อที่จะได้หาค่า cost ที่น้อยที่สุด ดังนั้นจึงเลือกใช้ Weighted Graph แต่เราไม่สนใจลำดับหรือทิศทางของคำศัพท์ ดังนั้นจึงไม่จำเป็นที่ Graph นี้ จะต้องระบุทิศทางอย่างชัดเจน จึงเลือกใช้ Undirected Graph
- Node ใช้เก็บคำศัพท์จากในไฟล์
- Edge ใช้เชื่อมคำศัพท์ที่สามารถ transform แบบ ladder step และ elevator step ได้ โดยมีน้ำหนักคือ cost
- Node สามารถเชื่อมกันได้ด้วย 2 กรณี (Edge) ได้แก่

1. Ladder step เมื่อคำศัพท์ 2 คำมีตัวอักษรที่ต่างกันเพียงหนึ่งตัวอักษรและเป็นตำแหน่งเดียวกัน สร้าง edge ที่มีน้ำหนักเท่ากับความห่างของตัวอักษรทั้งสองนั้น

2. Elevator step เมื่อมีตัวอักษรเหมือนกันทุกตัวอักษรต่างกันที่ลำดับการเรียง จะเกิด edge ที่มีน้ำหนักเท่ากับ 0.0

ตัวอย่างกราฟ (อย่างน้อย 20 nodes ในไฟล์ words_5757.txt.)



ยกตัวอย่างเช่น

node first และ node rifts สามารถเกิด elevator step ได้ เนื่องจากมีตัวอักษรเหมือนกันทุกตัว และกำหนดให้ egde มี weight มีค่าเท่ากับ 0.0

node rifts และ node rafts สามารถเกิด ladder step ได้ เนื่องจากมีตัวอักษรที่แตกต่างกันเพียงตัวเดียวและอยู่ในตำแหน่งเดียวกัน เชื่อมโดย egde ที่มีค่า weight มีค่าเท่ากับระยะห่างของตัวอักษรตามลำดับภาษาอังกฤษ ในที่นี้เป็นตัวอักษร a ในลำดับที่1 และตัวอักษร I ในลำดับที่ 9 ดังนั้นระยะห่างของตัวอักษรทั้งสองตัวจึงมีค่าเท่ากับ 8.0 weight จึงมีค่าเท่ากับ 8.0 นั่นเอง ในลักษณะคล้ายกัน node rifts และ node riots จึงมีค่า weight ของ edge เท่ากับ 9.0 จากระยะห่างของตัวอักษร f ในลำดับที่ 6 และตัวอักษร o ในลำดับที่ 15

Other data structures

- **Array**

มีการใช้ Array ในตำแหน่งต่าง ๆ ดังนี้

1. char[] x ในเมธอด main() ของคลาส wordladder

ใช้ในการเก็บตัวอักษร แปลงมาจาก word_search โดยเมธอด toCharArray() ใช้ Array เพื่อนำ index ไปเข้าถึงตัวอักษรตัวแรกสุดไปใช้งาน

2. char[] x ในเมธอด readfile() ของคลาส file

ใช้ในการเก็บตัวอักษร แปลงมาจาก word โดยเมธอด toCharArray() ใช้ Array เพื่อนำ index ไปเข้าถึงตัวอักษรตัวแรกสุดไปใช้งาน

เหตุผลที่ใช้ Array เพื่อแปลงคำ(string) เป็นตัวอักษรและใช้ Array indexing ในการเข้าถึงตัวอักษรเพื่อนำไปใช้ต่อไป

- **ArrayList**

มีการใช้ ArrayList ในตำแหน่งต่าง ๆ ดังนี้

1. ArrayList<String> list1 และ ArrayList<String> list2 ในเมธอด creatGraph() ของคลาส word_graphs

ใช้ list1 และ list2 เก็บ value จาก wordmap ซึ่งคือข้อมูลตัวอักษร (ที่แตกต่างกัน เช่น c1,c2 เป็น string) ที่อยู่ในคำศัพท์

เหตุผลที่ใช้ ArrayList เพื่อใช้เข้าถึงข้อมูลของ LinkedHashSet ในตำแหน่งต่างๆ และนำข้อมูลนั้นไปใช้เปรียบเทียบ

- **LinkHashSet**

มีการใช้ LinkedHashSet ในตำแหน่งต่าง ๆ ดังนี้

1. LinkedHashSet<String> char_key1, LinkedHashSet<String> char_key2 ในเมธอด creatGraph() ของคลาส word_graphs

เก็บ value จาก wordmap ซึ่งคือข้อมูลตัวอักษร (ที่แตกต่างกัน เช่น c1,c2 เป็น string) ที่อยู่ในคำศัพท์

2. LinkedHashSet<String> unionset, LinkedHashSet<String> present ในเมธอด game() ของคลาส word_graphs

โดย unionset ใช้เก็บคำศัพท์ที่ transform ในแต่ละ step ที่คำนวณจาก present และ present ใช้เก็บคำศัพท์ที่เป็น source กับ target node ของ edge ปัจจุบันใน loop

3. LinkedHashSet<String> tmp2 ในเมธอด readfile() ของคลาส file

ใช้ในการเก็บเซตของตัวอักษรที่แตกต่างกันในคำศัพท์ที่ถูกอ่านจากไฟล์ เพื่อนำไปเป็น value ของ wordmap2

เหตุผลที่ใช้ LinkedHashSet เนื่องจากช่วยให้สามารถจัดเก็บข้อมูลแบบเซตโดยไม่มีการซ้ำกัน สามารถใช้ method ของคลาสในการจัดการข้อมูลได้ และ LinkedHashSet มีการเรียงข้อมูลตามที่ถูกใส่ลงไป

- **TreeSet**

มีการใช้ TreeSet ในตำแหน่งต่าง ๆ ดังนี้

1. TreeSet<String> treetemp ในเมธอด readfile() ของคลาส file

โดย treetemp ใช้เก็บคำศัพท์ที่ถูกอ่านจากไฟล์ข้อมูลและจัดเก็บเป็นลำดับตาม natural order ของ String

2. TreeSet<String> allwo ในเมธอด main() ของคลาส wordladder

โดย allwo จะเก็บคำศัพท์ตามลำดับอักษร เก็บค่าที่ return จาก method returnTreemap() จากตัวแปร TM

3. TreeSet setword โดย treetemp ในเมธอด readfile() ของคลาส file

ใช้เก็บคำศัพท์ที่มีตัวอักษรตัวแรกที่เหมือนกัน จัดเก็บเป็นลำดับตาม natural order ของ String

เหตุผลที่ใช้ TreeSet เพราะข้อมูลมีการเรียงลำดับตามตัวอักษร ซึ่งช่วยให้ง่ายต่อการค้นหาและเรียงลำดับข้อมูล

- **TreeMap**

มีการใช้ TreeMap ในตำแหน่งต่าง ๆ ดังนี้

1. TreeMap<String, LinkedHashSet<String>> wordmap2 ในคลาส file

โดย Key เก็บคำศัพท์ และ Value เก็บ LinkedHashSet ของตัวอักษรที่แตกต่างกันในคำศัพท์ที่ถูกอ่านจากไฟล์

2. TreeMap<Character, TreeSet<String>> TM ในคลาส file

โดย Key เก็บตัวอักษรแรกของคำศัพท์ และ Value เก็บเซตของคำศัพท์ที่เริ่มต้นด้วยตัวอักษรนั้น ๆ

3. TreeMap<String , LinkedHashSet<String>> wordmap ในคลาส word_graphs

โดย Key เก็บคำศัพท์ และ Value เก็บ LinkedHashSet ของตัวอักษรที่แตกต่างกันในคำศัพท์

4. TreeMap<Character, TreeSet<String>> wordTS ในเมธอด main() ของคลาส wordladder

โดย Key เก็บตัวอักษรแรกของคำศัพท์ และ Value เก็บเซตของคำศัพท์ที่ถูกเรียงลำดับโดยไม่ซ้ำ

เหตุผลที่ใช้ TreeMap เนื่องจากทำให้สามารถจัดเก็บข้อมูลให้มีลำดับเรียงตาม key ได้ โดยจะเรียง key ตามลำดับของตัวอักษร ซึ่งช่วยให้การค้นหาและการจัดเรียงข้อมูลเป็นไปอย่างมีประสิทธิภาพ โดยเฉพาะในการค้นหาคำศัพท์ที่เริ่มต้นด้วยตัวอักษรที่กำหนดมา หรือในการจัดลำดับคำศัพท์ตามตัวอักษรแรกให้เป็นไปอย่างเหมาะสมโดยไม่ต้องเสียเวลาในการเรียงข้อมูลใหม่เอง

Graph Algorithms

- Dijkstra algorithm

โปรแกรมใช้ Object Graph ชื่อ `wordGraph` ซึ่งถูกประกาศและกำหนดค่าเริ่มต้นดังนี้:

```
protected Graph<String, DefaultWeightedEdge> wordGraph;

wordGraph = new SimpleWeightedGraph<>(DefaultWeightedEdge.class);
```

Object Graph นี้เป็นชนิด `weighted` (positive) และ `undirected` โดยมีน้ำหนักเนื่องจากแต่ละเส้นระหว่าง node มีน้ำหนักที่เกี่ยวข้อง และไม่มีทิศทางเนื่องจากความสัมพันธ์ระหว่างคำเป็นทางสองทาง ตัวอย่างเช่น ถ้า " sound " สามารถแปลงเป็น " three " ด้วย cost ที่กำหนด แล้ว " three " ก็สามารถแปลงเป็น " sound " ด้วย cost เดียวกันได้

อัลกอริทึม Dijkstra ถูกใช้ในเมธอด `game` เพื่อหาเส้นทางที่สั้นที่สุดระหว่างคำสองคำที่กำหนดในกราฟ อัลกอริทึมนี้ถูกเลือกเนื่องจากมันสามารถหาเส้นทางที่สั้นที่สุดในกราฟที่มีน้ำหนักเป็นบวก ซึ่งเหมาะสำหรับปัญหาการค้นหาลำดับการแปลงที่สั้นที่สุดระหว่างคำสองคำ และค้นหาได้รวดเร็วกว่าอัลกอริทึมอื่น ๆ

- Ladder และ Elevator

```
public void creatGraph(){
    Graphs.addAllVertices(destination: wordGraph, vertices: wordmap.keySet());
    Set<String> key = wordmap.keySet();
    double weight1 ;
    double weight2 ;
    double useweight=0;
    // 2 loop cuz for everyword check everyword
    for(String key1:key){
        for(String key2:key){
            if(key1.equalsIgnoreCase(anotherString: key2))continue;

            LinkedHashSet<String> char_key1 = new LinkedHashSet(c: wordmap.get(key: key1));

            char_key1.removeAll(c: wordmap.get(key: key2));

            int c_samechar=0;
                if(char_key1.size()==1){
                LinkedHashSet<String> char_key2 = new LinkedHashSet(c: wordmap.get(key: key2));
                List<String> list1 = new ArrayList<String>(c: wordmap.get(key: key1));
                List<String> list2 = new ArrayList<String>(c: char_key2);

                   for(int i=0;i<list1.size();i++){

                       String l1=list1.get(index: i);
                       String l2=list2.get(index: i);
                       if(l1.equals(anObject: l2)){
                    c_samechar++;
                       }
                   }
```

```
    if(c_samechar==4){
    char_key2.removeAll(c: list1);
    String x1 = char_key1.toString();
    String x2 = char_key2.toString();

     weight1 = checkDis(str: x1);
     weight2 = checkDis(str: x2);
     if(weight1>weight2) useweight=weight1-weight2;
     else  useweight=weight2-weight1;
     if(!wordGraph.containsEdge(v: key1,v1: key2)){
     wordGraph.addEdge(v: key1, v1: key2 );
     wordGraph.setEdgeWeight(sourceVertex: key1,targetVertex: key2,weight:useweight);
           }
       }
    }
     if(char_key1.size()==0){
        //System.out.printf("same 5\n");
         useweight = 0;
        if(!wordGraph.containsEdge(v: key1,v1: key2)){
    wordGraph.addEdge(v: key1, v1: key2 );
    wordGraph.setEdgeWeight(sourceVertex: key1,targetVertex: key2,weight:useweight);
        }
    }
```

ใน method creatGraph() ที่ใช้สำหรับสร้างกราฟเชื่อมระหว่างคำศัพท์ที่อ่านเข้ามาซึ่งเก็บคำศัพท์เหล่านั้นในรูปแบบ TreeMap ก่อนจะใช้ key มาสร้าง node ของแต่ละคำศัพท์ ในส่วนถัดไปได้วนจับคู่คำศัพท์เพื่อมาตรวจสอบว่าสามารถเกิด ladder step หรือ elevator step ได้หรือไม่ ได้ทำการลบตัวอักษรที่ซ้ำกัน (removeAll) ระหว่างคำศัพท์ทั้ง 2 เก็บไว้ในตัวแปร char_key1 เพื่อจะนำข้อมูลนี้ไปใช้ต่อในการพิจารณา

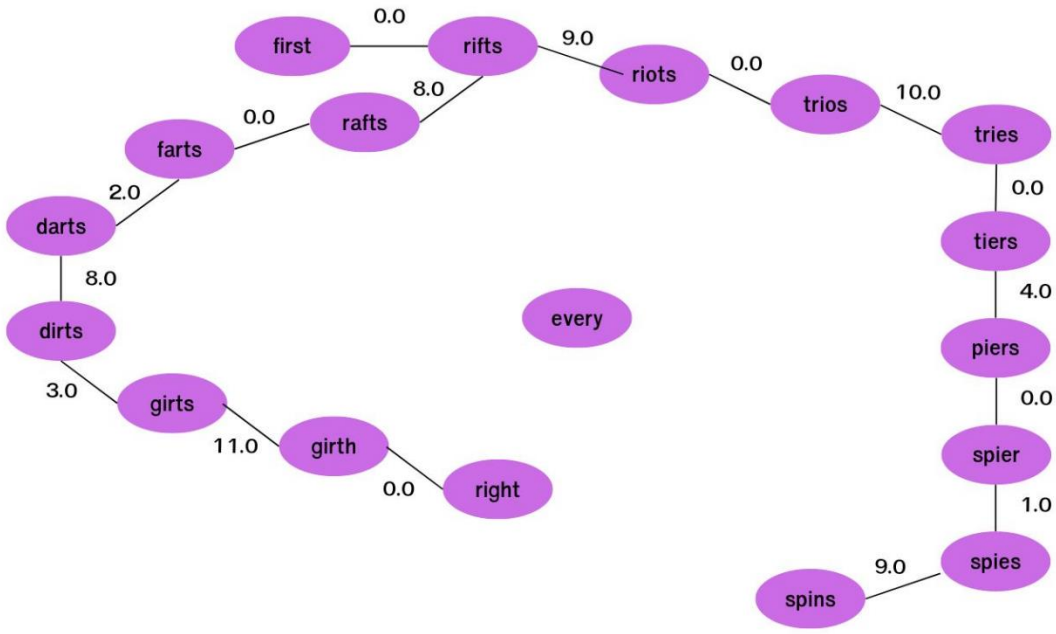
เมื่อข้อมูลทั้งสอง ลบกัน และ char_key1 เหลือเพียงตัวอักษรเดียวที่แตกต่างกันทั้ง char_key1, char_key2 และตัวอักษรนั้นยังต้องเป็นตำแหน่งเดียวกันด้วยเพื่อให้เข้าเงื่อนไขในการเกิด ladder step โดยใช้ข้อมูลในตัวแปร c_samechar ในการตรวจสอบที่ข้อมูลเกิดจากการวนลูปเพื่อดูว่าตัวอักษรที่แตกต่างกันนั้นอยู่ในตำแหน่งเดียวกันหรือไม่ และสร้าง edge ในการเชื่อม node ของสองคำศัพท์นี้ ซึ่งมีค่า weight เป็นความห่างของตัวอักษรทั้งสองที่แตกต่างกันในขอบเขตของจำนวนนับ ตามลำดับการเรียงในภาษาอังกฤษจากการเรียกใช้ method checkDis() เพื่อดูลำดับของตัวอักษร

ในส่วนที่ตัวอักษรของทั้งสองคำศัพท์เหมือนกันทุกตัวสะกดหรือสามารถเรียกได้ว่าเกิดการ intersect ทุกตัวอักษร จะเข้าเงื่อนไขในการเกิด elevator step และจะสร้าง edge ที่มี weight เป็น 0 เชื่อมระหว่างทั้งสอง node ของคำศัพท์

โดยสรุปแล้ว graph algorithm เลือกวิธีในการแปลงจากคำศัพท์นึงไปยังอีกคำผ่าน weight ของแต่ละ edge ที่มีค่าน้อยที่สุดโดยอาศัยกราฟที่ถูกสร้างขึ้น ซึ่งกราฟดังกล่าวเก็บข้อมูลทั้งหมดในการเกิด ladder step และ elevator step ไว้แล้ว กล่าวคือหาก node ใด ๆ มีทั้ง edge ที่เป็น ladder step และ elevator step จะเลือก elevator step ก่อน เพราะมี weight ที่น้อยกว่า

- Transformation failed

ตัวอย่างกราฟ (ในไฟล์ words_5757.txt.)



การแปลงระหว่างคำศัพท์ 2 คำ ไม่สำเร็จ เพราะไม่มี path ที่เชื่อมระหว่าง node ต้นทาง(คำศัพท์แรก) ไป node ปลายทาง(คำศัพท์ 2) ยกตัวอย่าง จากในรูปภาพ คำว่า every ไม่สามารถแปลงไปเป็นคำว่า first ได้ เนื่องจากไม่มี path ใดๆ เชื่อมถึงกัน หรือเรียกได้ว่า อยู่คนละ subgraph กัน

Limitation

1. ถ้าใน 1 คำมีตัวอักษรที่ซ้ำกันมากกว่า 3 ตัว อาจจะทำให้โปรแกรมทำงานผิดพลาด
2. ไฟล์ .txt ที่ใช้เก็บคำศัพท์ต้องเป็น 1 บรรทัดต่อ 1 คำ
3. การ search คำศัพท์ในเมนู search ด้วยคำที่มีอักษรมากกว่า 1 ตัว อาจมีคำที่ตัวอักษรตำแหน่งที่มากกว่า 2 ไม่ได้เรียงขึ้นต้นตามที่ใส่ไป เช่น

```
word to search =
vr
========== Available words ==========
vivre  vroom
```

บรรณานุกรม

โปรแกรมจำลอง Word Ladder นี้ถูกเขียนด้วยตัวผู้จัดทำเอง โดยมีการศึกษาเพิ่มเติมจาก

1.Using array: https://www.geeksforgeeks.org/how-to-get-elements-by-index-from-hashset-in-java/

2. Convert string to char Array: https://www.scaler.com/topics/how-to-convert-char-to-string-in-java/