EGCO 221 – Group Project 2 (Word Ladder)

1. Given a word file, find at least 1 solution to transform one word into another. Each transformation step can be done by taking either 1 ladder step or an elevator as follows.

   1.1 Ladder step. Only 1 character can be changed at a time. Transformation cost = distance between old and new characters.
   E.g. hears → heirs cost = 8 becase 'a' and 'i' differ by 8 characters

   1.2 Elevator step. Elevator can be taken only if 2 words are permutation of each other. Transformation cost = 0
   E.g. these → sheet cost = 0

2. **Programming**: the program must be written in Java.

   2.1 Read words from word file.
   - Small file (words_250.txt) is for you to test the functionality of your code
   - But the grading will be based on big file (words_5757.txt)
   - You can assume no input/spelling error, but your program must still be able to handle missing file.

   2.2 Search function. Since the input file contains thousands of words, the program must have a search function that lists all words containing or starting with certain string

   2.3 Word ladder game. Ask for source and target word from user. Generate word ladder/elevator to transform source word into target word. Report all transformation steps and total transformation cost. In some cases, the program may not be able to transform words – in that case, report that the transformation cannot be done.

   2.4 The program must be able to loop for another search or another ladder game. It should also be able to handle invalid user input in (2.2)-(2.3).

   2.5 Your source files (.java) must be in folder Project2_XXX where XXX = ID of the group representative, assuming that this folder is under Maven's "src/main/java" structure. The first lines of all source files must be comments containing English names & IDs of all members.

3. This word ladder puzzle is another well-known puzzle. There are many approaches to solving it, but you are required to use graphs and graph algorithms in this project.
   - You can search and use any algorithm or even pieces of code, provided that the sources are properly acknowledged.
   - These sources must be from your own research e.g. paper, textbook, Internet, etc.
   - Using classmates as references is considered cheating.

```
--- exec-maven-plugin:3.0.0:exec (default-cli) @ solutions ---
Enter word file =
words_5757.txt

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

Search =
cor
=== Available words ===
coral    cords    cordy    cored    corer    cores    corgi    corks    corky    corms
corns    cornu    corny    corps

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

Search =
sw
=== Available words ===
swabs    swags    swain    swami    swamp    swank    swans    swaps    sward    sware
swarf    swarm    swart    swash    swath    swats    sways    swear    sweat    swede
sweep    sweet    swell    swept    swift    swigs    swill    swims    swine    swing
swipe    swirl    swish    swiss    swive    swoon    swoop    sword    swore    sworn
swung

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

Enter menu >> (S = search, L = ladder, Q = quit)
l
Enter 5-letter word 1 =
corns
Enter 5 letter word 2 =
sugar

Cannot transform corns into sugar
```

Design a better user interface by yourself. Good user interface doesn't mean fancy output, but rather how your program is easy to understand & to use even without user manual. For example,

- Can user continue playing without having to restart the program?
- Are the instructions clear enough?
- Do you warn about valid user input and/or handle invalid user input?
- Is the output properly formatted and understandable?

4. **Report in THAI**: describe at least the following
   4.1 Short user manual
   4.2 Graph data structure
      - Be specific about your graph object: give the name of object and point out where it is in the program. Indicate its type (directed, undirected, weighted, unweighted) and reason to use this type of graph. Explain what nodes and edges in your graph represent.
      - Explain conditions that you check in order to add an edge from one node to another. Draw graph with at least 20 nodes (+ edges between them) to elaborate your graph construction. Nodes and edges in graph must reflect what your program really does & correspond to real data in words_5757.txt.

   4.3 Other data structures e.g. ArrayList, ArrayDeque, HashSet, HashMap, etc.
      - Be specific about each of your data structure objects: give the name of the object and point out where it is in the program. Explain reason to use this type of data structure.

   4.4 Graph algorithm(s) to find at least 1 word ladder solution for given source-target words. Source code listing without any explanation isn't counted as a proper report.
      - Which graph algorithms (e.g. BFS, DFS, Dijkstra, Kruskal, etc.) are used to solve the puzzle? Explain reason to use each of them.
      - Explain how ladder and elevator transformations are encoded or used, how your algorithm considers/chooses between ladder and elevator (or among all possible transformations).
      - Explain why some transformations fail e.g. from "corns" to "sugar". What is wrong with the graph? Draw/show some parts of the real graph (built from words_5757.txt) to elaborate your explanation.

   4.5 Any limitation of your program. Convenient limitation isn't counted as a proper limitation. For example, don't give a limitation that your program will crash if input file is missing or user enters invalid input just because you are too lazy to handle exceptions.

   4.6 References or declaration of originality
      - If you design the data structures/algorithms and write the whole program by yourself, explicitly declare that everything is done by yourself. But if it is found later that this is not true, it will be counted as cheating.
      - Otherwise, valid references/URLs must be given. The URLs must point to the code you take from (and I can access it without having to log in the website).
      - A report without valid references or declaration of originality will get point deduction.

*** The project can be done in a group of <= 4 students. But each group must do it by themselves. **Everyone involved in cheating will get ZERO point**

Grading

Programming (10 points)

2 points Requirements + any correct word ladder.

1 point Word ladder with minimum transformation cost.

2 points User interface + exception handling.

- This part must be your own effort. Even if 2 groups take algorithms or pieces of code from the same source, it would be impossible to have the same/similar coding for the user interface. Therefore, same (or too similar) user interface coding is considered cheating.

5 points Programming (2 points) + proper Java collection and JGraphT APIs (3 points)

- Although this is not a programming course, your program should still be well structured.
- Even if you use pieces of code from the Internet. Try converting it to proper OOP with proper Java collection and JGraphT (version 1.5.2) APIs. Don't implement your own Graph data structures or Graph algorithms if JGraphT classes or methods are available.

Report (10 points)

5 points Graph data structure, graph construction with demo, other data structures.

3 points Graph algorithms

2 points Manual, limitation, reference/declaration of originality, others (writing, format, etc.)

Submission

1. A report in only 1 PDF file. The front page must contain names & IDs of everyone in your group.
2. Source files (*.java). No need to include JGraphT library in your submission.
3. File readme.txt containing names & IDs of everyone in your group.
4. Put all files in folder Project2_XXX (see 2.5) and zip the folder.
   - The group representative submits the whole project to Google Classroom
   - The other group members submit only readme.txt to Google Classroom