### **Overview:**

|  |  |
| --- | --- |
| stone **A**tone a**L**one **C**lone clon**S** c**O**ons co**N**ns con**E**s cone**Y** **M**oney | Word ladders were invented by Lewis Carroll in 1878, the author of *Alice in Wonderland*. A ladder is a sequence of words that starts at the starting word, ends at the ending word, In a word ladder puzzle you have to change one word into another by altering a single letter at each step. Each word in the ladder must be a valid English word, and must have the same length. For example, to turn stone into money, one possible ladder is given on the left.  Many ladder puzzles have more than one possible solutions.  Another path from stone to money is:  stone store shore chore choke choky cooky cooey coney money |

### 

### 

### 

### **Objectives**

* Practice using the Stack ADT
* Practice using ArrayList ADT
* Practice using File Input/Output

### **Instructions**

**STAGE 1: BuildGraph.java**

1. Write a program called “BuildGraph.java” to create the word ladder graph.
   1. Download the “dictionary.txt” file from Google Drive
   2. Read the contents of the “dictionary.txt” and add all four letter words to an ADT ArrayList<String> words = new ArrayList<>();
   3. Create an output file called “graph.txt”
   4. for(int i=0;i<words.size();++i) {

print words.get(i)+ “ “ followed by list of words that differ by one letter with

words.get(i)

}

For example, “graph.txt” will contain

band land sand

bent

bike like mike

boat coat

card cord hard ward

cash cast dash hash mash nash rash

cast cash cost fast last

coat boat cost

cold cord fold hold

**STAGE 2:** “WordLadder.java” & “Vertex.java” (Download from Google Drive)

* 1. Implement a class called “WordLadder.java” with the following specification:

import java.io.\*;

import java.util.\*;

public class WordLadder {

private ArrayList<Node> list; // list representing the word graph

private LinkedList<Vertex> ladder = new LinkedList<>(); // representing the ladder

public WordLadder() {

list=new ArrayList<>();

}

/\*\*

\* Read the file containing the graph into list

\* @param fileName

\* @throws IOException

\*/

public void loadWordtMap(String fileName) throws IOException {

File infile = new File(fileName);

try(Scanner in = new Scanner(infile);)

{

while(in.hasNextLine()) {

// read a line

// split into a String[] called tokens based on “ “

Node word = // create a new Node based on tokens[0]

// add the reset of the tokens to word.addPath()

// add the word to “list”

}

}

}

/\*\*

\* Find the ladder between start and end city, if it exists. Otherwise return false

\* @param start

\* @param end

\* @return

\*/

public boolean findLadder(String start,String end) {

Stack<Vertex> aStack = new Stack<>();

Vertex startWord = new Vertex(start);

Vertex endWord = new Vertex(end);

Check to see if startWord and endWord are part of the graph. Otherwise, print

Error message and return false.

mark all cities as unvisited

private LinkedList<Vertex> visited = new LinkedList<>(); // visited list of vertices

Add startWord to aStack

Mark startWord as visited by adding it to visited

While ( aStack not empty and have not found a path to destCity) {

Get the path of the vertex at top of aStack

If ( path is empty )

Remove vertex from aStack

Else {

Remove any visited vertex from the path until you find an unvisited vertex (v)

Add v to aStack

Add v to visited

} // end of while

If ( aStack is not empty ) {

Build the ladder by adding vertices from “aStack” to “ladder”

Print the “ladder”

Return true;

} else return false;

} // findLadder

/\*\*

\* A node in the graph is represented by a Vertex and path to other words

\*/

class Node{

private Vertex word; // Graph vertex

private ArrayList<Vertex> path; // list of vertices in the path

public Node() {

path = new ArrayList<>();

}

public Node(Vertex word) {

this.word = word;

path = new ArrayList<>();

}

public void setword(Vertex word) {

this.word = word;

}

public Vertex getword() {

return word;

}

public ArrayList<Vertex> getPath(){

return path;

}

public void addPath(Vertex v) {

path.add(v);

}

public String toString() {

return word +"-->"+path;

}

public boolean equals(Object e) {

Vertex v = ((Node)e).getword();

return this.word.equals(v);

}

}

} // wordLadder

### **Example**

Enter start word:band

Enter end word:fake

aStack:[band]

visited:[band]

path:[land, sand]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

aStack:[band, land]

visited:[band, land]

path:[band, sand]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

aStack:[band]

visited:[band]

path:[sand]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

aStack:[band, sand]

visited:[band, sand]

path:[band, land, sane]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

aStack:[band, sand, sane]

visited:[band, sand, sane]

path:[sake, sand, sine]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

aStack:[band, sand, sane, sake]

visited:[band, sand, sane, sake]

path:[fake, lake, make, sane]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Found path of 5 cities

[band, sand, sane, sake, fake]

### **Grading:**

**Your assignment will be graded as following:**

1. **Your project must compile. Project will earn 0 points if there is any kind of syntax error. If you are using an IDE that creates a package, make sure to remove the import statement from all submitted files.**
2. **Here is the point breakdown:**

* **Correct implementation of the *BuildGraph* class - 30 points**
* **Correct implementation of the *LoadWordMap*  - 30 points**
* **Correct implementation of the *LoadWordMap*  - 40 points**

**SUBMIT:**

A zip folder containing:

1. BuildGraph.java
2. WordLadder.java
3. FindLadder.java (driver program)
4. Vertex.java
5. graph.txt