Names: Ronald Macmaster, Horng-Bin Justin Wei

UT EID: rpm953 and hjw396

**PA4: Fun with Word Ladders**

**1) Analysis**

**Problem Statement:**

Design a program to traverse a word ladder for any given pair of 5-lettered words.

Utilize the word classifications from a given dictionary of legal English 5-letter words.

Print a warning message if the word ladder does not exist.

The word ladder must not necessarily be the shortest word ladder.

**Input:**

Word pairs are input from a given file. The filename is specified through the command line.

General Word Pair input

dears fears

stone money

money smart

devil angel

atlas zebra

heart heart

babes child

mumbo ghost

ryan joe

hello buddy

hello world

heads tails

Valid input : 2 fiver-letter words in the English language.

**Input is not necessarily valid!** Report errors using exceptions.

**Output:**

Output to the console a word ladder from the starting word to the ending word.

The words in the ladder must be valid English words

If a ladder does not exist, Output: “There is no word ladder between … (word1) and (word2).”

If any of the input words are not valid English words, print

**Questions?**

Should we map every single 5-letter words in a single graph?

How do we perform a search for the words?

**2) Design**

**Architecture Models:**

1. **System Use-Case Diagram**
2. **UML class diagram**
3. **ADT class description for each class.**
4. **4) functional block diagram**

**OUTPUT**

**Return Account Statement**

* Print out a compiled account statement for each customer

**PROCESS**

**Customer Bank Accounts**

* 4 bank accounts per customer
* Checking, Saving, Auto, and Student Loan

**Perform Transactions**

* Perform Transfers
* Log the action
* Handle and log errors

**Compile Account Statement**

* Put together an account statement based off final account totals
* Label the totals accordingly

**INPUT**

**Bank Account Transactions**

* Customer ID#
* Transaction Type
* [Amount]
* Account Type
* [Account Type 2]

1. **Functional Block Diagram**

getAccountString()

getCustomerName()

getCustomerAddress()

getAccountBalance()

Main

Transaction

ServiceCustomerAccount()

Customer

Withdraw()

Deposit()

depositFunds()

withdrawFunds()

transferFunds()

addAccountInterest()

getAccountBalance()

getTransactionAmount()

getTransactionType()

getAccountType1()

getAccountType2()

BankAccount

SavingsAccount

CheckingAccount

addInterest()

fine()

overDraw()

**Algorithms**

**Driver Algorithm:** (Assign4Driver)

1. Read / Clean word list from the dictionary
2. Build the Word Graph from dictionary data
3. Loop:
   1. Read in word pair
   2. Output delimiter \*\*\*\*\*\*\*\*\*\*\*\*\*
   3. If word data invalid: throw exception and repeat
   4. Compute Ladder from start to end
   5. If no ladder exists: Print no ladder exists! And repeat
   6. Else: print word ladder and repeat
4. Output delimiter \*\*\*\*\*\*\*\*\*\*\*\*\*
5. End driver.

**Probable Graph Search Algorithms:**

Breadth-first search

BFS(Graph, roof){

For each node in G

N distance = infinity

N parent = null

Empty Queue Q

Root distance = 0

Q enque(roof)

While(not empty){

For each node adjacent to current

If(n.distance = invfinity)

n.distance = current.distance+1

n.parent = current

Q.enqueue(N)

Depth-first search

DFS(Graph, root)

Empty Stack S

S.push(root)

While(S, not empty)

V = s.pop()

If v not discovered

Label v discovered

For all adjacent edges (v, w)

s.push(w)

A paragraph describing the rationale behind your design. This would include: a) How does your OOD reflect the interaction and behavior of the real-world objects that it models b) What alternatives did you consider? What were the advantages/disadvantages of each alternative both from a programming perspective and a user perspective? c) What are some expansions or possible flexibilities that your design offers for future enhancements? d) How does your design adhere to principles of good design: OOD, cohesion, coupling, info hiding,