**DOMAIN COMPARATOR**

**Overview**

This program takes input in the form of set of domain names and operates different functionalities like finding the manipulation difference and sequence of intermediate transformations and calculating the number of nearby domains.

**Files and external data**

Users need to provide a set of domains names in the load function of object of domain class.

**Data structures and their relations to each other**

Program uses maintains Vertices and Edges of the domain names provided in the set. HashMap find its key usage in this program. Graphs are used in the program along with Hashmap to handle manipulations and various other required functionalities. Additionally, TreeSet is also used to manage set detached domain name vertices for later usage and better efficiency while handling the various operations involved.

To iterate over the data structures, it uses for loop, while loop and iterators.

**Assumptions**

* Load function will be called every time with its required argument after creating the Domains class instance.
* EditPath returns given domain name when both domain names are same.
* Edit distance value zero is returned when same domain names are provided.
* UpperCase domain and Lowercase domain with same text string are treated to be different.

**Key algorithms and design elements**

Levenshtein algorithm is used to calculate the manipulation difference between two domain name strings. A recursively built approach is used in Levenshtein which calculates difference between two nodes. First node is used for reference and all other domains are stored in form of levels (which accounts to the manipulation difference) from the base node. The map is then processed to add connections between levels and in between same level. These nodes are added with information about neighbours and adjacent nodes thereby making them Vertices and Edges of graph.

These Vertices are processed by Dijkstra's algorithm to traverse all the nodes and set some information. Then this nodes are traversed upon as per the requirements to fetch out crucial information.

**Limitations**

* Efficiency and speed of program decreases with increase in number of domain names provided.
* Vertices with no neighbouring vertices store unnecessary space.
* Algorithm is internally required to compute paths between vertices before handling operations.

**Test Cases**

1. For each Edit Path and Edit Distance:

* Both domain names are equal.
* First domain name is not present in set.
* Second domain name is not present in set.
* Domain names with zero difference is provided as input.
* Reverse sequence of domain name is provided.
* Source Domain not convertible to target domain.
* Distant far domains are provided as input.
* Detached domain vertex is provided as either of domain string
* Same Node string is provided as input for both arguments

1. For Num Nearby:

* Provide domain name not present in domain set.
* Provide domain name present in domain set but not attached to any other set.
* Provide distance larger than possible vertex availability.
* Provide base vertex as input.
* Provide last Vertex as input.

**Citations and References**

**1.** Jain, D. (2019). *Calculating Levenstein Distance | Baeldung*. [online] Baeldung. Available at: https://www.baeldung.com/java-levenshtein-distance [Accessed 4 Feb. 2019].

**2.** YouTube. (2019). *002 Dijkstra implementation*. [online] Available at: https://www.youtube.com/watch?v=mw-2fU6jXqM [Accessed 4 Feb. 2019].