Word Ladder

.

Word ladder (also known as **Doublets**, **word-links**, **change-the-word puzzles**, **paragrams**, **laddergrams**, or **Word golf**) is a word game invented by Lewis Carroll. A word ladder puzzle begins with two words, and to solve the puzzle one must find a chain of other words to link the two, in which two adjacent words (that is, words in successive steps) differ by one letter.

This program takes 2 words from the user and prints the shortest word ladder, starting from the source word and ending at the destination word. To find the shortest Word Ladder that exists between two words we have used Dijkstra's Shortest Path Algorithm. We have used a txt file containing almost all possible dictionary words, from Dictionary by Merriam-Webster. Hence this program also involves File Handling.

In this program the following data structures have been used:

1. Graphs
2. AVL Trees
3. Stacks
4. Queues

Functions in the main.c file

1. **Graph *Read_and_AddWords(char *fname)**: A function to insert the words from the given .txt file and inserts each word as a vertex.
2. **void AddEdges(Graph *G)** :for each word, generates all possible words that differ by one letter, and adds edges to and fro these words in the graph

Functions in the graph.c file

1. **Graph *CreateGraph(int V)** : Creating a directed graph with an initial capacity of V vertices.At this point graph is limited to at most V vertices.Implementation of graph using adjacency list.
2. **int AddVertex(Graph *G, char *name)** : The following func adds a vertex of a given word(name) to G, returning a unique integer id. identifying this vertex.

Returns -1 if addition of vertex failed, that is if the graph is full in addition to being unable to create more vertices

(i)Makes use of **Insert** function i.e. present in avltree.c file : Inserts the given value into the AVL tree, rebalancing the tree as necessary. Returns a pointer to the root of the new tree; if the value to insert is already in the tree, nothing happens and a pointer to the root of the original tree is returned.

3. **int Word2Vert(Graph *G, char *Name)** : Looks up a vertex by the given word(name), returning its vertex no. if found this value will be >= 0. It returns -1 if not found.

(i)Makes use of **Contains** function i.e. present in avltree.c file ~ Searches for the given value, if found, returns pointer to node in tree, otherwise NULL is returned.

4. **char *Vert2Word(Graph *G, Vertex v)**: Looks up a vertex by number, returning a pointer to its name; returns NULL if v is invalid.

5. **int AddEdge(Graph *G, Vertex src, Vertex dest, int weight)**: Adds a directed edge (src, dest, weight) to G. If successful, true i.e a non- zero value is returned. if the edge could not be added (i.e. due to invalid vertex ids), then false (0) is returned.edges are stored "in order by destination"
6. **Vertex *Neighbors(Graph *G, Vertex v)** : neighbors are vertices adjacent to the given vertex v. The neighbors are returned in a dynamically-allocated array, in ascending order; despite there being multi-edges i.e edges with same dest. all edges are stored in order and hence edges with the same dest appear next to each other returns NULL if v is not a valid vertex id.

7. **int PopMin(Queue *unvisitedQ, int distance[])** : searches for the smallest vertex distance in the queue, saving other elements in a stack so we can put back later.

8. **Vertex *Dijkstra(Graph *G, Vertex src, Vertex dest)**: Performs Dijkstra's shortest path algorithm to find the shortest path from src to dest. Returns a dynamically-allocated array of vertices denoting this path; the array will start with src, contain 0 or more vertices that lead to dest, followed by dest, and ending with -1. If there is no path from src to dest, the array will contain only -1. returns NULL if src or dest are not valid vertex ids.

It’s not necessary that the word ladder formed between the first and the second will be the same as that when both are swapped, eg. in the following situation. The two words entered are black and blush

```
**HI! Welcome to the Word Ladder Program **

>>>>Building Graph from 'dictionary.txt'...

Please Enter a Word from the Dictionary (**press ENTER to quit**):  black
Please Enter another Word from the Dictionary (**press ENTER to quit**): blush
** Shortest word ladder **
   black
   slack
   stack
   stuck
   stunk
   stung
   swung
   swing
   swine
   twine
   twice
   trice
   trick
   brick
   brisk
   brusk
   brush
   blush
   Length: 17
```

```
**HI! Welcome to the Word Ladder Program **

>>>>Building Graph from 'dictionary.txt'...

Please Enter a Word from the Dictionary (**press ENTER to quit**):  blush
Please Enter another Word from the Dictionary (**press ENTER to quit**): black
** Shortest word ladder **
   blush
   brush
   brusk
   brisk
   brick
   breck
   break
   bleak
   bleat
   blent
   blend
   bland
   blank
   black
   Length: 13
```

Also it would be best if the two words entered are of the same length. A word ladder does not exist between two words of different lengths.