CS 2302 - Lab 7

Sofia Gutierrez

Olac Fuentes

**Introduction**

The purpose of this lab was to determine whether a graph has a Hamiltonian Cycle which is a cycle that visits every vertex found within an undirected graph, exactly once. This was done three times using different algorithms. Then the edit distance between two chosen words would be printed. The following design algorithms were implemented in this lab: Randomization algorithm, Backtracking, Dynamic Programming (for the edit distance function).

**Design and Implementation**

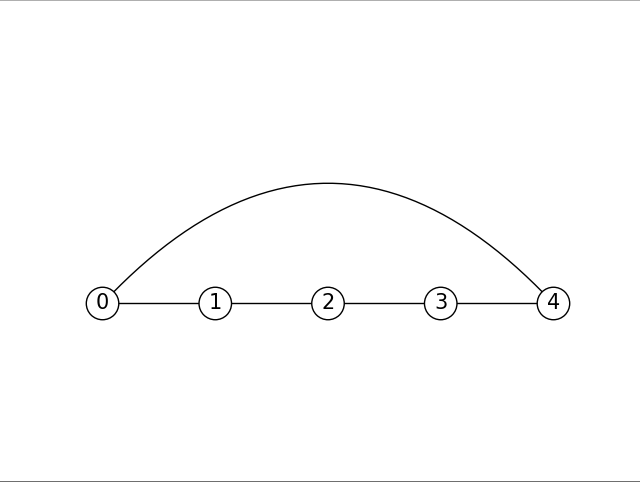
The first design technique implement within the source code was Randomization. This algorithm required the use of 4 methods in total. The first function was named ‘in\_degree’, which was a function to help find the number of edges going into the current vertex (in-degree). In order for this to be a valid Hamiltonian Cycle the current vertex must an in-degree of exactly 2. Anything more or less is not considered a valid Hamiltonian Cycle. Then came the function ‘connected\_components’ which returns a disjoint set forest of all the chosen vertices and edges. This is needed in our next function named randomized\_hamiltonian which would look for a Hamiltonian path. Again a Hamiltonian Cycle is when every vertex is visited exactly once. For the graph to be considered a Hamiltonian Cycle then the function ‘connected\_components’ would have to return a number 1, meaning the cycle has 1 connected component and therefore a Hamiltonian Cycle.

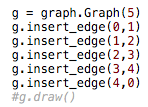
Then came the function ‘backtracking\_hamiltonian’ which took in a list of edges and from that list an edge list graph was created. This edge list was then tested to see if it’s length was the same as the vertices that were hardcoded in the main function. If the same length, then the graph would then be transformed into an adjacency list. Again we would call our ‘connected\_component’ function sending the current graph and make sure ‘connected\_component’ returned a 1, meaning that all vertices are connected within the graph. Once again the in-degree would also be tested to see if all the vertices within had a in-degree of 2.

Then came the function ‘edit\_distance’ which used Dynamic Programming. Dynamic Programming is recursively defining the value of an optimal solution. It also computes solutions to all possible subproblems (starting with the simplest) and stores them in a table. Then it computes the solution to the original problem by combining solutions to subproblems. This was learned in class. This was an easier function to implement considering it was given to us and only slight modifications needed to be done to it. Those slight modifications included modifying the function to allow replacements only in the case where the characters being interchanged are both vowels or consonants.

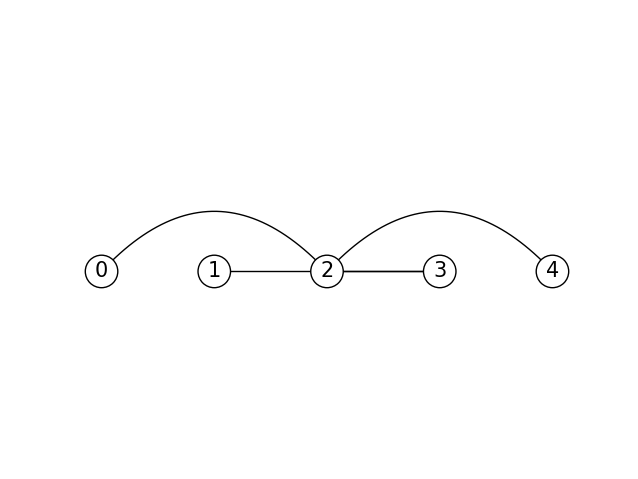
The main function showed the graphs that were hard coded. The first graph would intentionally produce a Hamiltonian Cycle when tested by the algorithms. The second graph would not be considered a valid Hamiltonian Cycle. The user was then prompted to insert their chosen two words in order to test the ‘edit\_distance’ function

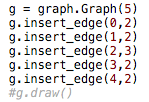
**Experimental Results**

****

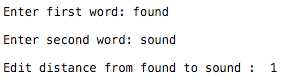
****

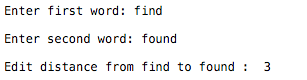
The images above include the graph created with specific edges hardcoded. This graph was processed through the ‘connected\_components’ function. This graph clearly shows that the connected component is equal to 1 and therefore, it is a valid Hamiltonian Cycle.

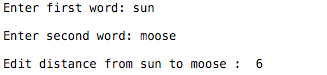
****

****

The images above include the graph created with specific edges hardcoded. This graph was processed through the ‘connected\_components’ function. This graph clearly shows that the connected component is not equal to 1 and therefore, it is not a valid Hamiltonian Cycle.

****

****

****

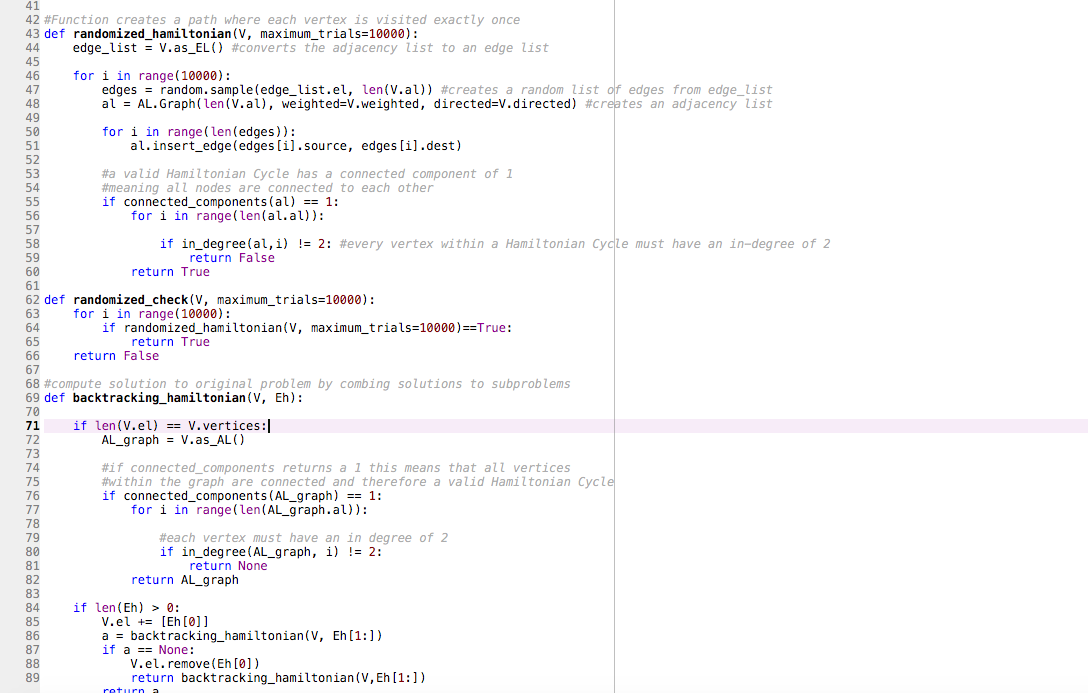
The images above show the outcomes for the ‘edit\_distance’ function using Dynamic Programming. The first two words were very similar and produced an edit distance of 1. The next pair of words had somewhat some similarity and this produced an edit distance of 3. The final set of words produced an edit distance of 6.

**Conclusion**

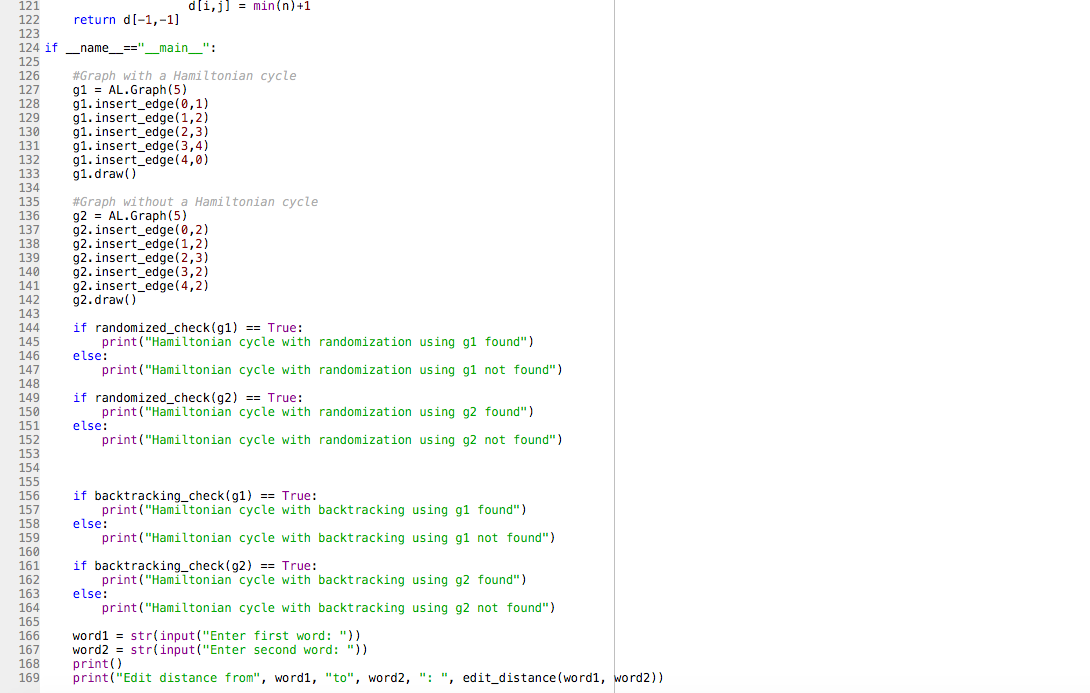
This lab challenged me to code algorithm designs. Randomization and Backtracking for me were the most difficult to implement. These two algorithms required a lot of trial and error but after several attempts the correct output was produced. The function to calculate the edit distance between two words inputted by the user was easier to implement considering we were supplied with the base of it and the function only had to be modified slightly.

**Appendix**









**Academic Honesty Certification**

I certify that this project is entirely my own work. I wrote, debugged, and tested the code being presented, performed the experiments, and wrote the report. I also certify that I did not share my code or report or provided inappropriate assistance to any student in the class.

Sofia Gutierrez December 9, 2019