CS 2302 Data Structures
Fall 2019

Lab Report #7

Due: December 6th, 2019

Professor: Olac Fuentes

TA: Anindita Nath

Introduction

For this lab we were tasked with figuring out if an undirected graph has a Hamiltonian Cycle using a randomization algorithm and a backtracking Algorithm, as well as using dynamic programing to modify the edit distance of two words.

Proposed Solution Design and Implementation

Randomization Algorithm:

For this method I would use and edge list representation of a graph (E) and the number of vertices it contained(V) as parameters, then I would first check to see if the graph had enough edges to complete a Hamiltonian cycle which is at least V edges. Next, I would run a for loop to try 2^(len(|E|)) times to find if a graph contains a cycle. I would then create a new edge list representation of containing V random edges from the edge list. After the new graph has been formed, I would then convert the newly formed graph into an adjacency list representation and use the method cycle to test of the graph contained one connected component as well as return the path of the graph starting at zero. Next, I would use another method called check to test if the in-degree of every vertex in the graph is two as well as if the out-degree of every vertex is two. If the newly formed path passed all the test, then the path of the found Hamiltonian cycle would be returned and if no path was found the program would return nothing.

Backtracking:

For this method I would use adjacency list representation of the graph created(G), the position(pos) of the method which starts with the value zero and an empty list(used) as the initial parameters. From there I would start off by checking if the position given is in the used list and if not would add it to the list of used vertices. Then I would check the length of the used list and if it equals to the number of vertices in the list it would then check to see if the final element of the used list has zero as its destination for one of its out-degrees. If the length of the used list does not equal to the number of vertices the program runs another for-loop to try every edged connected to the position recursively until a Hamiltonian cycle if found and then the used list, it returned which is the path. If no such cycle exists within the graph nothing is returned.

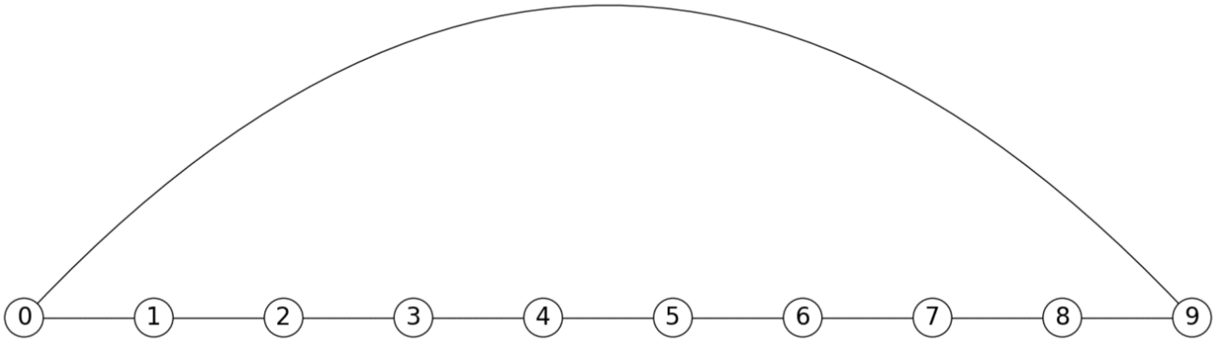
Dynamic Programming:

For this method I edited the edit distance method given to use in class to accommodate the newly required cases to modify the string if both letters where vowels or consonants. I added a list of all vowels and when checking for the edit distance of a word I would use that list to see if both words where vowels or consonants and is so would change the letter of the first word to match the letter in the first word. If the second word was modified, I would also print the new word and then I would return the modified edit distance of the string.

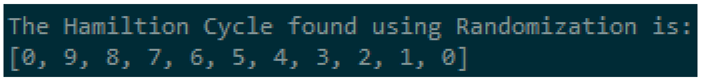
Experimental Results:

Hamiltonian Cycle:

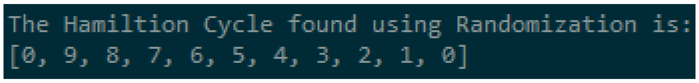
Test 1:



Through randomization the program found two possible solutions:

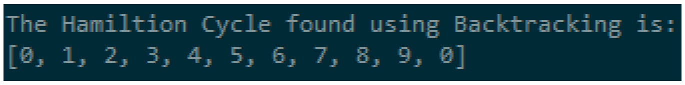


```
The Hamiltion Cycle found using Randomization is:
[0, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```



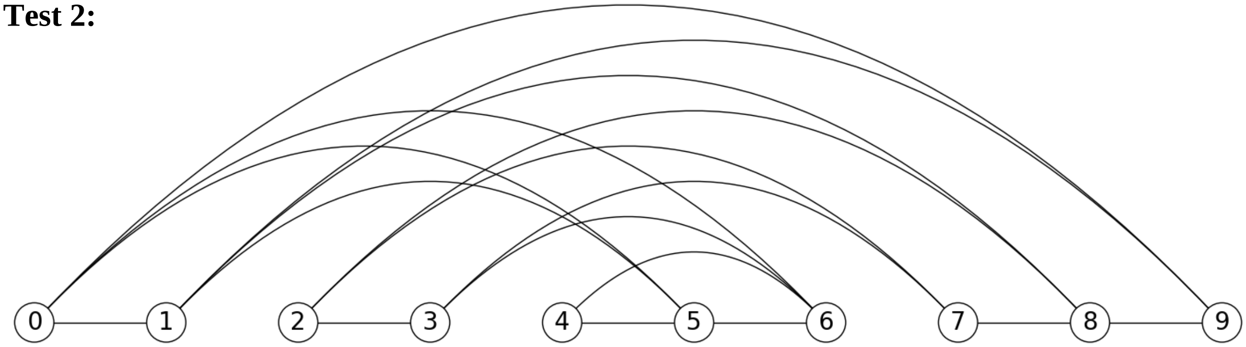
```
The Hamiltion Cycle found using Randomization is:
[0, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

but with backtracking only the first solution is found and returned:

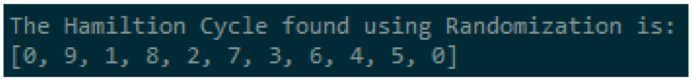


```
The Hamiltion Cycle found using Backtracking is:
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
```

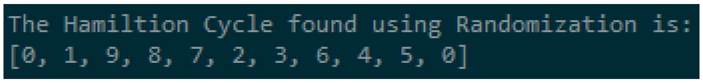
Test 2:



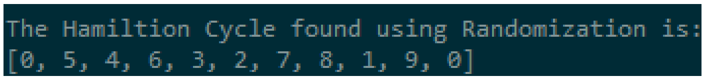
Through randomization I was able to find multiple Hamiltonian paths such as:



```
The Hamiltion Cycle found using Randomization is:
[0, 9, 1, 8, 2, 7, 3, 6, 4, 5, 0]
```

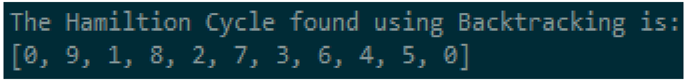


```
The Hamiltion Cycle found using Randomization is:
[0, 1, 9, 8, 7, 2, 3, 6, 4, 5, 0]
```



```
The Hamiltion Cycle found using Randomization is:
[0, 5, 4, 6, 3, 2, 7, 8, 1, 9, 0]
```

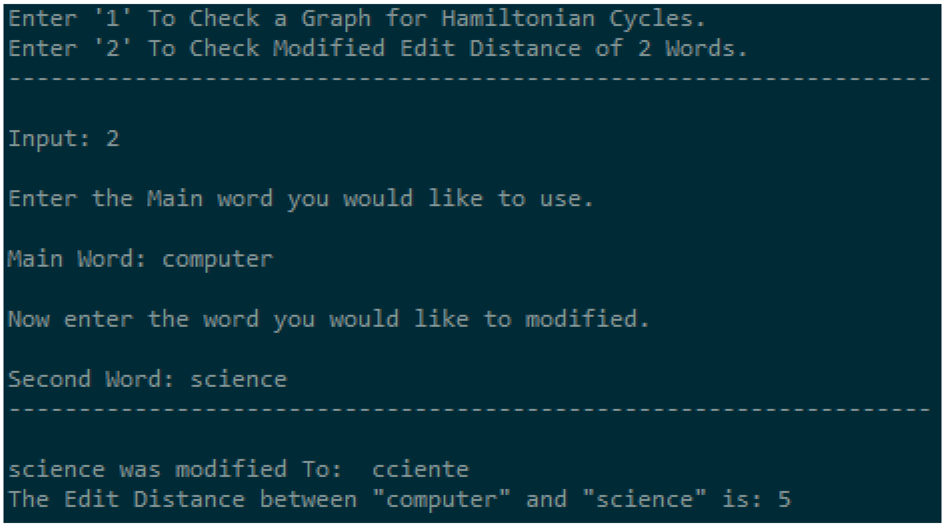
but again, with backtracking only the first solution is found and returned:



```
The Hamiltion Cycle found using Backtracking is:
[0, 9, 1, 8, 2, 7, 3, 6, 4, 5, 0]
```

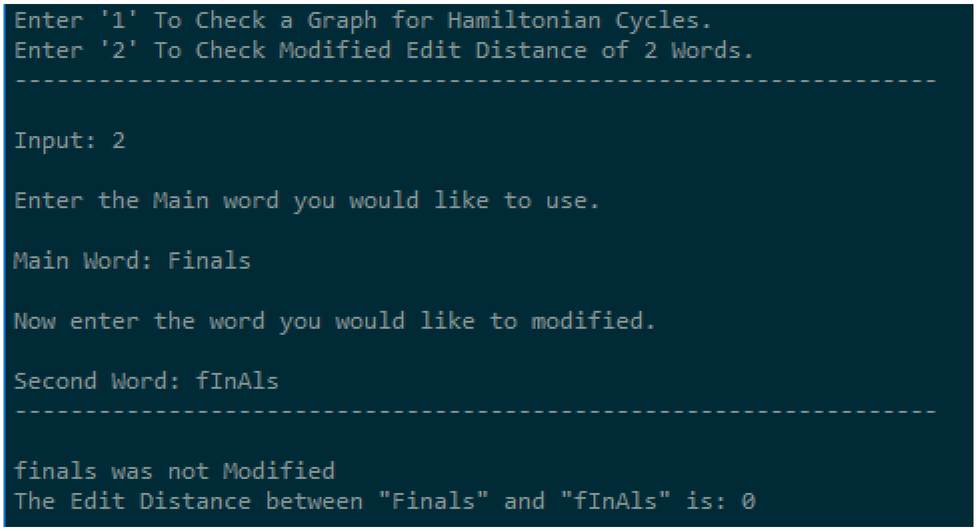
Modified Edit-Distance:

Test 1:



```
Enter '1' To Check a Graph for Hamiltonian Cycles.
Enter '2' To Check Modified Edit Distance of 2 Words.
------------------------------------------------------------------

Input: 2

Enter the Main word you would like to use.

Main Word: computer

Now enter the word you would like to modified.

Second Word: science
------------------------------------------------------------------

science was modified To:  cciente
The Edit Distance between "computer" and "science" is: 5
```

Test 2:



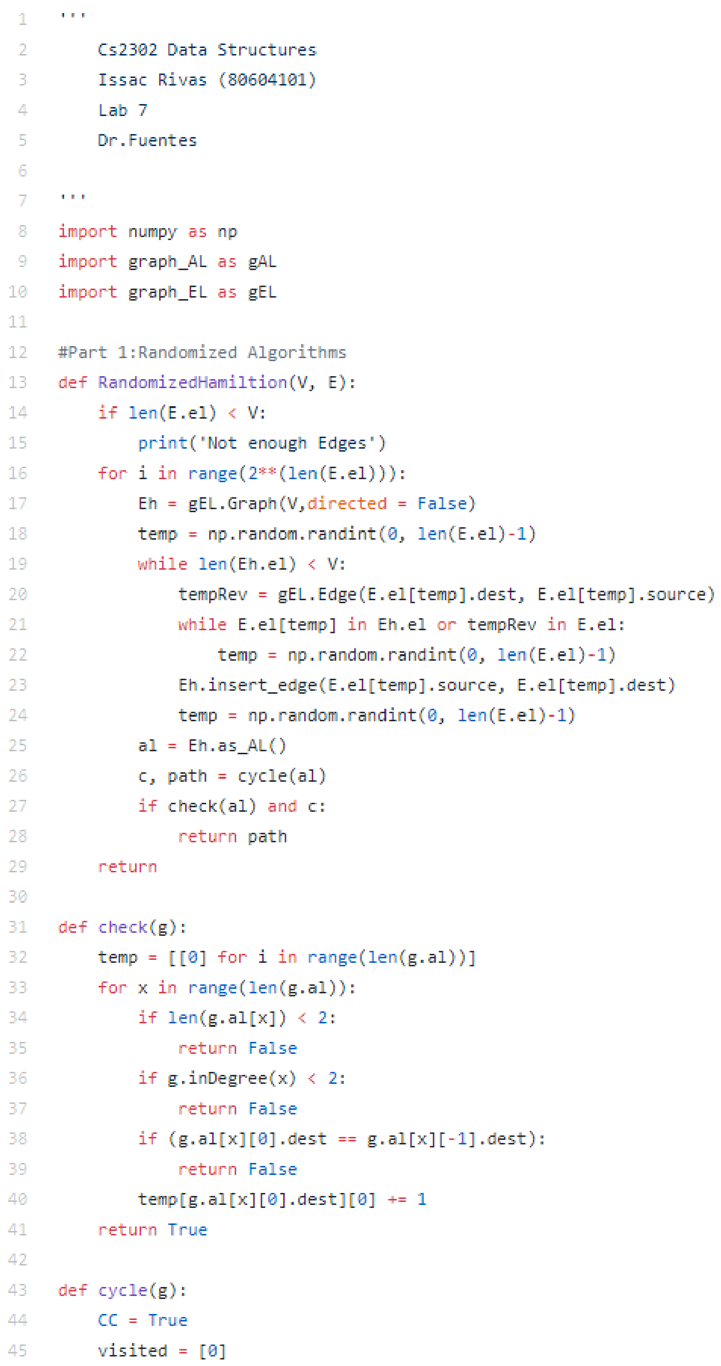
```
Enter '1' To Check a Graph for Hamiltonian Cycles.
Enter '2' To Check Modified Edit Distance of 2 Words.
------------------------------------------------------------------

Input: 2

Enter the Main word you would like to use.

Main Word: Finals

Now enter the word you would like to modified.

Second Word: fInAls
------------------------------------------------------------------

finals was not Modified
The Edit Distance between "Finals" and "fInAls" is: 0
```

Conclusion:

In conclusion by using different graphing methods I was able to determine Hamiltonian cycles in graphs buy cratering another graph and seeing if the newly formed graph met all the requirements of a Hamiltonian cycle. Also, by modifying edit distance of words I was able to lower the edit distance by changing two vowels or two consonants, as well as I was better able to understand how using graphing representations can help solve a variety of problems.

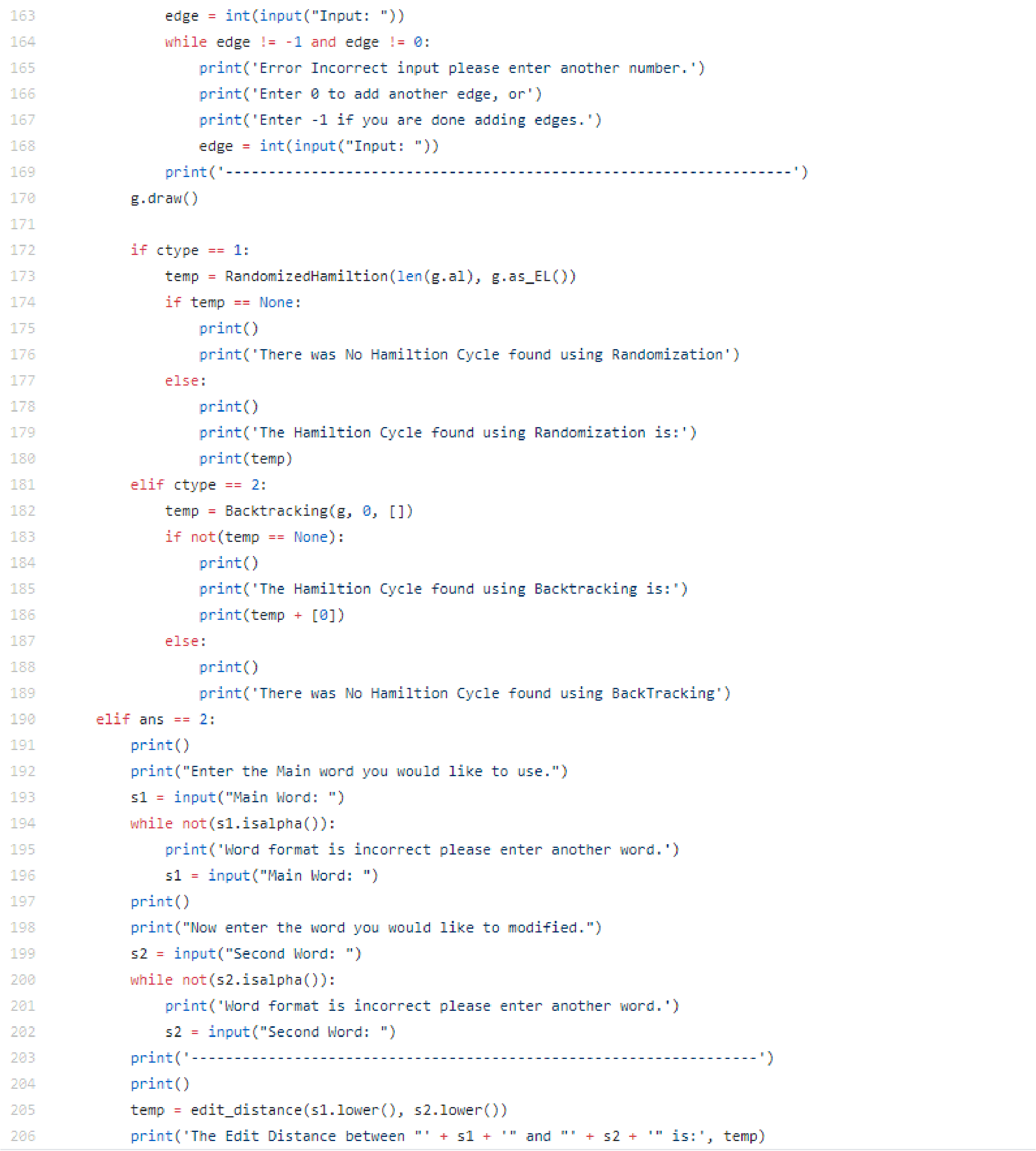
Appendix:

Main.py:



```python
'''
    Cs2302 Data Structures
    Issac Rivas (80604101)
    Lab 7
    Dr.Fuentes

'''
import numpy as np
import graph_AL as gAL
import graph_EL as gEL

#Part 1:Randomized Algorithms
def RandomizedHamiltion(V, E):
    if len(E.el) < V:
        print('Not enough Edges')
    for i in range(2**(len(E.el))):
        Eh = gEL.Graph(V,directed = False)
        temp = np.random.randint(0, len(E.el)-1)
        while len(Eh.el) < V:
            tempRev = gEL.Edge(E.el[temp].dest, E.el[temp].source)
            while E.el[temp] in Eh.el or tempRev in E.el:
                temp = np.random.randint(0, len(E.el)-1)
            Eh.insert_edge(E.el[temp].source, E.el[temp].dest)
            temp = np.random.randint(0, len(E.el)-1)
        al = Eh.as_AL()
        c, path = cycle(al)
        if check(al) and c:
            return path
    return

def check(g):
    temp = [[0] for i in range(len(g.al))]
    for x in range(len(g.al)):
        if len(g.al[x]) < 2:
            return False
        if g.inDegree(x) < 2:
            return False
        if (g.al[x][0].dest == g.al[x][-1].dest):
            return False
        temp[g.al[x][0].dest][0] += 1
    return True

def cycle(g):
    CC = True
    visited = [0]
    x = 0
    while len(visited) != len(g.al):
        if not(g.al[x][0].dest in visited):
            visited += [(g.al[x][0].dest)]
            x = g.al[x][0].dest
        else:
            visited += [(g.al[x][-1].dest)]
            x = g.al[x][-1].dest
    visited += [0]
    for x in range(len(g.al)):
        if not(x in visited):
            CC = False
    if CC:
        return True, visited
    return False, visited

#Part 2: Backtracking
def Backtracking(G, pos, used):
    if pos not in set(used):
        used.append(pos)
        if len(used)==len(G.al):
            for x in range(len(G.al[used[-1]])):
                if G.al[used[-1]][x].dest == 0:
                    return used
            return [-1]
        for nextV in range(len(G.al[pos])):
            new = [i for i in used]
            trial = Backtracking(G, G.al[pos][nextV].dest, new)
            if trial is not None:
                return trial

#Part3 3: Dynamic Programming
def edit_distance(s1,s2):
    vowels = ['a', 'e', 'i', 'o', 'u']
    normal = s2
    d = np.zeros((len(s1)+1,len(s2)+1),dtype=int)
    d[0,:] = np.arange(len(s2)+1)
    d[:,0] = np.arange(len(s1)+1)
    for i in range(1,len(s1)+1):
        for j in range(1,len(s2)+1):
            if s1[i-1] == s2[j-1]:
                d[i,j] =d[i-1,j-1]
            else:
                if (s1[i-1] in vowels) and (s2[j-1] in vowels) and (i == j):
                    s2 = s2[:j-1] + s1[i-1] + s2[j:]
                    d[i,j] =d[i-1,j-1]
                elif not(s1[i-1] in vowels) and not(s2[j-1] in vowels) and (i == j):
                    s2 = s2[:j-1] + s1[i-1] + s2[j:]
                    d[i,j] =d[i-1,j-1]
                else:
                    n = [d[i,j-1],d[i-1,j-1],d[i-1,j]]
                    d[i,j] = min(n)+1
    if normal != s2:
        print(normal, 'was modified To: ', s2)
    else:
```

```python
    print('------------------------------------------------------------------')
    ans = int(input("Input: "))
    while(ans != 1 and ans != 2):
        print()
        print("Incorrect input entered please Try again.")
        print("Enter 1 To Check a Graph for Hamiltonian Cycles.")
        print("Enter 2 To Check Modified Edit Distance of 2 Words.")
        ans = int(input("Input: "))
    if ans == 1:
        print("Choose table implementation:")
        print("Enter '1'  to solve with a Randomized Algorithm.")
        print("Enter '2'  to solve using Backtracking.")
        print('------------------------------------------------------------------')
        ctype = int(input("Input: "))
        while(ctype != 1 and ctype != 2):
            print()
            print("Incorrect input entered please Try again.")
            print("Enter '1'  to solve with a Randomized Algorithm.")
            print("Enter '2'  to solve using Backtracking.")
            ctype = int(input("Input: "))

        #Build Graph
        print('Enter How many Verticies You want the graph to have.')
        vert = int(input("Input: "))
        while(vert <= 2):
            print('Error, Enter a Number Greater then Two.')
            print('Enter How many Verticies You want the graph to have.')
            vert = int(input("Input: "))
        print('------------------------------------------------------------------')
        g = gAL.Graph(vert,directed = False)
        print()
        print('Now Enter all the edges for your graph.')
        edge = 0
        pos = 0
        pos2 = 0
        while edge != -1:
            print('Enter the first vertice.')
            print('Then enter the vertice it will connect to.')
            pos = int(input("Main vertice: "))
            while pos > vert or pos < 0:
                print('Error Incorrect input please enter another number.')
                print('Enter the first vertice.')
                pos = int(input("Main vertice: "))
            pos2 = int(input("Connecting vertice: "))
            while pos2 > vert or pos2 < 0:
                print('Error Incorrect input please enter another number.')
                print('Then enter the vertice it will connect to.')
                pos2 = int(input("Connecting vertice:: "))
            g.insert_edge(pos, pos2)

            print()
            print('Enter 0 to add another edge, or')
            print('Enter -1 if you are done adding edges.')
            edge = int(input("Input: "))
```



```
            edge = int(input("Input: "))
            while edge != -1 and edge != 0:
                print('Error Incorrect input please enter another number.')
                print('Enter 0 to add another edge, or')
                print('Enter -1 if you are done adding edges.')
                edge = int(input("Input: "))
            print('-------------------------------------------------------------------')
        g.draw()

        if ctype == 1:
            temp = RandomizedHamiltion(len(g.al), g.as_EL())
            if temp == None:
                print()
                print('There was No Hamiltion Cycle found using Randomization')
            else:
                print()
                print('The Hamiltion Cycle found using Randomization is:')
                print(temp)
        elif ctype == 2:
            temp = Backtracking(g, 0, [])
            if not(temp == None):
                print()
                print('The Hamiltion Cycle found using Backtracking is:')
                print(temp + [0])
            else:
                print()
                print('There was No Hamiltion Cycle found using BackTracking')
    elif ans == 2:
        print()
        print("Enter the Main word you would like to use.")
        s1 = input("Main Word: ")
        while not(s1.isalpha()):
            print('Word format is incorrect please enter another word.')
            s1 = input("Main Word: ")
        print()
        print("Now enter the word you would like to modified.")
        s2 = input("Second Word: ")
        while not(s2.isalpha()):
            print('Word format is incorrect please enter another word.')
            s2 = input("Second Word: ")
        print('-------------------------------------------------------------------')
        print()
        temp = edit_distance(s1.lower(), s2.lower())
        print('The Edit Distance between "' + s1 + '" and "' + s2 + '" is:', temp)
```

I Issac Rivas, certify that this project is entirely my own work. I wrote, debugged, and tested the code being presented, preformed 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.