**CS 2302 Data Structures**

**Spring 2019**

**Lab Report #7**

**Due: December 4, 2019**

Professor: Olac Fuentes

TA: Anindita Nath

**Introduction**

For this lab we were asked to implement a Hamiltonian cycle using different algorithm designs. This lab focused more on the implementation of randomized algorithms, backtracking algorithms, and dynamic programming algorithms. The randomized and backtracking algorithm designs were used to find a Hamiltonian cycle while the dynamic programming algorithm design was used to modify the edit distance function that was used in class. A Hamiltonian cycle is described as a cycle in an undirected graph that visits each vertex exactly once. Determining whether a graph contains a Hamiltonian cycle is a NP-complete problem which indicates that determining whether a subset of the set of edges forms a Hamiltonian cycle is easy but finding such set might require testing all possible subsets. The algorithm designs required by this lab should avoid having to test all possible subsets.

**Proposed Solution Design and Implementation**

**Part #1**

The first part of the lab required us to implement a search for a Hamiltonian cycle using a randomized design algorithm. The function would take a V and a E. V is the vertices in the graph while was the edges in the graph. The function would then create a variable called maxTrials which was set to 250 but could essentially be set to any number. This declaration of a trials number allows us to classify this function as a randomized design algorithm. The function would then implement a base case if the list of vertices was equal to 2. If this was the case, then it would append to Eh which at this point is empty. If the length of vertices was greater than 2 then we would enter a for loop that would iterate the amount of maxTrials that were set. From this we would call on a shuffle of the elements in the list of the edges present in the graph. Eh would be initialized to the size of V. From this point we would check if the vertex we are at has 1 connected component and the in- degree of every vertex in V is 2 we can return a Eh which would indicate there is a Hamiltonian cycle present. The if statement would be applied on every V.

**Part #2**

The second part of the lab required us to implement a search for a Hamiltonian cycle using a backtracking design algorithm. The function takes a V and a E like the previous function. The list of the edges is then made into a graph that is adjacency list. Once this is done the graph is passed to the backtracking function. This function iterates over all possible paths in the graph when searching for a Hamiltonian cycle. Through each iteration the number of visited vertices is increased until it reaches all the vertices in the graph. If an edge exists from the starting vertex to the final one, then the graph contains a Hamiltonian cycle. The function then returns a list containing the path.

**Part #3**

The third part of the lab required us to modify the edit distance function given in class to allow replacements only in the case where the characters being interchanged are both vowels, or both consonants. This function takes in string 1, string 2 and a list v containing the vowels. The main change that is present is that if replacement is not possible under the condition that we cannot replace vowels with consonants and consonants with vowels then this is equal to two delete and insert operations making the total cost equal to 2. From this we can find the edit distance between the two strings while taking into account the conditions set by the lab.

**Experimental Results**

**Part #1 and Part #2  
E = [(11, 25), (25, 7), (7, 11), (4, 25), (11, 4), (7, 4)] V = [11, 25, 7, 4]  
Backtracking Hamiltonian  
[(11, 25), (25, 7), (7, 4), (4, 11)]  
Randomized Hamiltonian  
[(11, 4), (7, 11), (25, 7), (4, 25)]**

**E = [(11,25), (7,11), (4,2), (11,4), (25,2), (5,4)] V = [11,25,7,4,2]  
Backtracking Hamiltonian  
None**

**Randomized Hamiltonian None**

**Part #3**

﻿First string:aab

Second string:abc

Distance: 2

﻿First string:jack

Second string:rose

Distance: 5

**Conclusion**

In conclusion this lab furthered our understanding of graphs while also implementing three different types of design algorithms. These three different types of algorithm design were randomized algorithms, backtracking algorithms and dynamic programming. Randomized and backtracking algorithms were used to search for a Hamiltonian cycle in a graph while dynamic programming was used to modify the edit distance that was provided in class. Overall this lab improved our understanding of graphs while also increasing our ability to manipulate and implement different type of designs when it comes to algorithms.

**Appendix**

﻿def backtracking(g, v, temp):

if len(temp) == len(g):

for i in g[v]:

if i == 0:

return [v]

return None

for i in g[v]:

if i not in temp:

newTemp = list(temp)

newTemp.append(i)

p = backtracking(g, i, newTemp)

if p is not None:

p.append(v)

return p

return None

def Backtracking\_Hamiltonian(V, E):

g = [[] for i in range(len(V))]

for j in E:

index1 = -1

index2 = -1

for i in range(len(V)):

if V[i] == j[0]:

index1 = i

if V[i] == j[1]:

index2 = i

if index1 != -1 and index2 != -1:

g[index2].append(index1)

g[index1].append(index2)

p = backtracking(g, 0, [0])

if p is None:

return None

p.reverse()

Eh = []

for i in range(len(p) - 1):

Eh.append((V[p[i]], V[p[i+1]]))

Eh.append((V[p[-1]], V[0]))

return Eh

﻿import random

def Randomized\_Hamiltonian(V, E):

maxTrials = 250

if len(V) == 2:

Eh = []

for i in E:

if (i[0] == V[0] and i[1] == V[1]) or (i[0] == V[1] and i[1] == V[0]):

Eh.append(i)

return Eh

return None

for i in range(maxTrials):

random.shuffle(E)

Eh = E[0:len(V)]

connected = False

for j in Eh:

if j[0] != j[1]:

connected = True

break

if connected:

vert = []

for e in Eh:

vert.extend(e)

hamiltonian = True

for v in V:

if vert.count(v) != 2:

hamiltonian = False

break

if hamiltonian:

return Eh

return None

﻿def distance(s1, s2,v):

p = range(len(s2) + 1)

for i, x1 in enumerate(s1):

d = [i + 1]

for j, x2 in enumerate(s2):

insert = p[j + 1] + 1

delete = d[j] + 1

if x1 == x2:

subs = p[j]

elif (x1 in vowels and x2 in v) or (x1 not in v and x2 not in v):

subs = p[j] + 1

else:

subs = p[j] + 2

d.append(min(insert, delete, subs))

p = d

return p[-1]

if \_\_name\_\_ == "\_\_main\_\_":

vowels = ('a', 'e', 'i', 'u', 'o', 'A', 'E', 'I', 'U', 'O')

s1 = input("First string:")

s2 = input("Second string:")

print("Distance:", distance(s1, s2,vowels))

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