Lab Assignment 7

ISMAEL VILLALOBOS

PROFESSOR FUENTES

TA: DITA NATH

12-7-19

Introduction

This lab assignment requires knowledge of the three algorithm design techniques that are required. This includes randomization, backtracking, and dynamic programming. These techniques are especially helpful when attempting to solve NP-Complete problems such as the one from this assignment to detect Hamiltonian cycles and edit-distance problems. Given the pseudo code :

```
Randomized Hamiltonian(V,E)
    for i in range(maximum trials)
    let Eh be a random subset of E of size V
    if graph (V,Eh) has 1 connected component and the in-degree of every vertex in V is 2
        return Eh # Eh forms a Hamiltonian cycle
return None # No Hamiltonian cycle was found
```

For the second part of the assignment we were asked to determine if there is a Hamiltonian cycle in a graph using backtracking. Finally, we had an edit distance problem with given constraints using dynamic programming. We had to:

Modify the edit distance function provided in class to allow replacements onlyin the case where the characters being interchanged are both vowels, or both consonants. For example,'a' can be replaced by 'e' but not by 's', and 't' can be replaced by 'w' but not by 'u'

Proposed Solutions

Randomization: For this method we have a function takes in 2 parameters, V for an input to create graph and the number of tries you want to run. Using V turned into an edge list for easy traversal. We pick edges at random to create a random graph. Next we create an adjacency list and insert every random edge that we have created. By doing this we can utilize connected components to determine if a Hamiltonian Cycle exists. Lastly, we must check the in degree of every vertex and it must equal 2.

Backtracking: The main function takes parameter V (graph) and converts it to an edge list for ease of traversal, similar to our randomization implementation. We create an empty edge list to be used in our utility function. Then we call our utility function with previously mentioned graphs passed as arguments. I first started with a base case where we stop if the number of edges in our first edge list graph (V) is the same as the number of vertices. This satisfies the condition where our subset is the size of vertices. If this is true, we go into

our conditions to check for a Hamiltonian Cycle. Here, we convert our edge list graph to an adjacency list to be used to check for connected components and to check for in-degrees. Our second base case is if the list of edges is empty, return None. Then we go into our recursive calls, where we take the first edge and add it in our list of edges and assign to our graph. Our first recursive call is taking the graph and the first edge of our list. If that does not return None, return the value. The second recursive call is with the first edge removed from our argument.

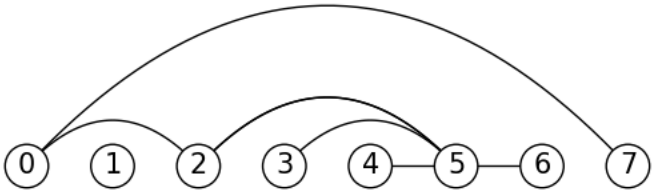
Dynamic Programming: Our last task is to implement dynamic programming to solve the edit distance problem.
Additionally, constraints are added to where replacements are only allowed for the characters being changed are both vowels or both consonants. To do this the edit distance provided in the class website had to be modified as per lab instructions. I first created a list of vowels (a,e,i,o,u). Second, was to modify the code to only make replacements if current characters are both vowels or both consonants. If the check is valid, then it will find the minimum of the three values (insert, replace, remove). Otherwise, replacement will not be considered in our minimum value.

Experimental Results

```
In [2]: runfile('C:/Users/Ismael/Desktop/CS2302Fall19/
lab7V2.py', wdir='C:/Users/Ismael/Desktop/CS2302Fall19')
Reloaded modules: dsf, graph_AL, graph_EL
1. Hamiltonian cycle
2. Random edges for Hamiltonian cycle
3. Test modified and unmodified edit distance function
with input strings

Select choice: 1
Randomization: Hamiltonian Cycle
[[(1,1)(5,1)] [(0,1)(2,1)] [(1,1)(3,1)] [(2,1)(4,1)]
[(3,1)(5,1)] [(0,1)(4,1)] ]
Backtracking: Hamiltonian Cycle
Randomization: Not a Hamiltonian Cycle
Backtracking: Not a Hamiltonian Cycle
```

Using custom graph




```
Select choice: 2

Enter graph size: 8

Enter number of random edges: 9

Randomization: Not a Hamiltonian Cycle
Backtracking: Not a Hamiltonian Cycle
```

```
In [4]: runfile('C:/Users/Ismael/Desktop/CS2302Fall19/
lab7V2.py', wdir='C:/Users/Ismael/Desktop/CS2302Fall19')
Reloaded modules: dsf, graph_AL, graph_EL
1. Hamiltonian cycle
2. Random edges for Hamiltonian cycle
3. Test modified and unmodified edit distance function
with input strings

Select choice: 3

Enter word #1: miners

Enter word #2: money
unmodified edit distance: 3
modified edit distance: 3
```

Conclusion

This lab has taught us to use different algorithm techniques to be used for problem solving implementations especially with NP-complete such as Hamiltonian cycles and edit distance problems that are not solvable in realistic time. Running the program with

different inputs has shown me that the randomization approach is less precise than backtracking but faster. On the other hand, backtracking takes longer for larger graphs but you get a definite output. The approach on this was slightly easier than other labs as we were given pseudocode and a base code to modify.

Academic Honesty

"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."

Name: Ismael Villalobos

Appendix:

```
        '''
Ismael Villalobos
12-3-19
Lab Assignment 7
Professor Fuentes
TA-Dita Nath
Purpose:
'''

import dsf
import graph_AL as AL
import graph_EL as EL
import random
import numpy as np
import time

# Function to get number of in-degrees of a given vertex
# Inputs: G as graph, v as the vertex
# Output: Number of in-degrees of vertex v
def in_degree(G, v):
        indeg = 0
        for i in range(len(G.al)):
                for j in G.al[i]:
                        if j.dest == v:
```

```
                                        indeg += 1
            return indeg

# From class website
def connected_components(g):
    vertices = len(g.al)
    components = vertices
    s = dsf.DSF(vertices)
    for v in range(vertices):
        for edge in g.al[v]:
            components -= s.union(v,edge.dest)
    return components#, s

# Function to detect Hamiltonian cycle using randomization
# Inputs: V as graph, and tests as number of random tests desired
# Output: Boolean. True if Hamiltonian cycle is detected, False if otherwise
def ham_random(V, tests):

    # turn v into edge list to be picked from randomly
    edge_list=V.as_EL()

    for t in range(tests):
        # add random edges seleceted from edge list into list
        edge=random.sample(edge_list.el, len(V.al))

        # use list of random edges and insert into adjacency list
        al=AL.Graph(len(V.al),weighted=V.weighted, directed=V.directed)
        for i in range(len(edge)):
            al.insert_edge(edge[i].source,edge[i].dest)

        # check if there is only 1 connected componenet
        if connected_components(al) == 1:

            # check in degree of every vertex is 2
            for i in range(len(al.al)):
                if in_degree(al,i) != 2:
```

```
                return False
        return True


# Randomized Hamiltonian cycle tester
# Inputs: V as an adjacency list graph, test as range of tests desired
# Output: Determines if V is a Hamiltonian cycle graph
def ham_random_test(V,tests):


    for i in range(100):
        if ham_random(V, tests)==True:
            return "Hamiltonian Cycle"
    return "Not a Hamiltonian Cycle"


# Backtracking helper function
# Inputs: edge_list as list of edges and graph as input graph
# Output: returns None is Hamiltonian cycle is not detected
# and True if there is a cycle
def ham_backtrack_(V,Eh):
                # Base Case
                if len(V.el) == V.vertices:
                                graphAL = V.as_AL()
    # check if there is only 1 connected componenet
                                if connected_components(graphAL) == 1:
                                                # check in degree of every vertex is 2
                                                for i in range(len(graphAL.al)):
                                                                if in_degree(graphAL, i) != 2:
                                                                                return None
                                                return graphAL
    # check if list of edges is empty
                if len(Eh) == 0:
                                return
                else:
        # Recursive calls
                                V.el = V.el + [Eh[0]] # take first edge
                                a = ham_backtrack_(V,Eh[1:])
                                if a is not None:
```

```
                    return a
                V.el.remove(Eh[0]) # do not take first edge
                return ham_backtrack_(V,Eh[1:])

# Backtracking main function.
# Input: V as input graph
def ham_backtrack(V):
    # Convert V as an edge list and assign it to Eh
    Eh = V.as_EL()

    # Create an edge list graph with the same parameters as V
    el = EL.Graph(len(V.al), weighted=V.weighted, directed=V.directed)

    return ham_backtrack_(el,Eh.el)

# Simplified Backtracking Hamiltonian cycle test
# Input: V as graph
# Output: Determines if graph is creates a Hamiltonian cycle or not
def ham_backtrack_test(V):
    ham=ham_backtrack(V)
    if isinstance(ham, AL.Graph):
        ham.display()
        return "Hamiltonian Cycle"
    else:
        return "Not a Hamiltonian Cycle"

# From class website
# Inputs: s1, s2 as strings
# Output: Minimum number of operations to convert s1 to s2
def edit_distance(s1,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:
            n = [d[i,j-1],d[i-1,j-1],d[i-1,j]]
            d[i,j] = min(n)+1
    return d[-1,-1]

# From class website, with modifications to only allow replacements with both
# vowels or both consonants
# Inputs: s1, s2 as strings
# Output: Minimum number of operations to convert s1 to s2
def edit_distance_modified(s1,s2):
    v = ['a','e','i','o','u']

    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:
                # allow replacements only in the case where the characters
                # being interchanged are both vowels, or both consonants
                if (s1[i-1] in v and s2[j-1] in v) or (s1[i-1] not in v and s2[j-1] not in v):
                    n = [d[i,j-1],d[i-1,j-1],d[i-1,j]]
                    d[i,j] = min(n)+1
                else:
                    n = [d[i,j-1],d[i-1,j]]
                    d[i,j] = min(n)+1
    return d[-1,-1]



# Function to create custom graph with variable size and number of random edges.
# Inputs: size as number of vertices in the graph and num_edges and desired number of edges
# Output: graph with desired number of vertices and random edges
def custom_graph(size, num_edges):
```

```
    g=AL.Graph(size)

    for i in range(num_edges):
        g.insert_edge(random.randint(0,size-1),random.randint(0,size-1))

    return g

if __name__=="__main__":

    print('1. Hamiltonian cycle')
    print('2. Random edges for Hamiltonian cycle')
    print('3. Test modified and unmodified edit distance function with input strings')

    choice=int(input('Select choice: '))

    if choice==1:
        g1 = AL.Graph(6) # Graph Hamiltonian cycle
        g1.insert_edge(0,1)
        g1.insert_edge(1,2)
        g1.insert_edge(2,3)
        g1.insert_edge(3,4)
        g1.insert_edge(4,5)
        g1.insert_edge(5,0)
        g1.draw()

        g2 = AL.Graph(6) # Graph without Hamiltonian cycle
        g2.insert_edge(0,1)
        g2.insert_edge(1,2)
        g2.insert_edge(2,3)
        g2.insert_edge(3,4)
        g2.insert_edge(4,5)
        g2.draw()

        print('Randomization:',ham_random_test(g1,1000)) #Ham cycle
        print('Backtracking:',ham_backtrack_test(g1)) #Ham cycle
```

```
    print('Randomization:',ham_random_test(g2,1000)) #Not Ham cycle
    print('Backtracking:',ham_backtrack_test(g2)) #Not Ham cycle

if choice==2:
    graph_size=int(input('Enter graph size: '))
    edge_count=int(input('Enter number of random edges: '))

    g3=custom_graph(graph_size, edge_count)
    g3.draw()
    print('\nRandomization:',ham_random_test(g3,1000))
    print('Backtracking:',ham_backtrack_test(g3))

if choice==3:
    word1=str(input('Enter word #1: '))
    word2=str(input('Enter word #2: '))

    print('unmodified edit distance:',edit_distance(word1,word2))
    print('modified edit distance:',edit_distance_modified(word1,word2))
```