Laboratory 7 – Algorithm Design Techniques

CS 2302 – Data structures Fall 2019

Carlos cardenas

INSTRUCTOR: OLAC FUENTES

Contents

[Introduction 2](#_Toc15996822)

[Implementation 2](#_Toc15996823)

[Experimental results 3](#_Toc15996824)

[Conclusion 9](#_Toc15996825)

[Honesty Certification 14](#_Toc15996827)

# Introduction

Dynamic programming is an optimization technique, which divides the problem into smaller sub-problems and after solving each sub-problem, dynamic programming combines all the solutions to get ultimate solution. Unlike divide and conquer method, dynamic programming reuses the solution to the sub-problems many times.

Recursive algorithm for Fibonacci Series is an example of dynamic programming.

Backtracking is an optimization technique to solve combinational problems. It is applied to both programmatic and real-life problems. Eight queen problem, Sudoku puzzle and going through a maze are popular examples where backtracking algorithm is used.

In backtracking, we start with a possible solution, which satisfies all the required conditions. Then we move to the next level and if that level does not produce a satisfactory solution, we return one level back and start with a new option.

An algorithm that uses random numbers to decide what to do next anywhere in its logic is called Randomized Algorithm. For example, in Randomized Quick Sort, we use random number to pick the next pivot (or we randomly shuffle the array). Typically, this randomness is used to reduce time complexity or space complexity in other standard algorithms.

# Implementation

To implement the Randomized Hamiltonian cycle algorithm, I used the help of previous functions and classes done before. The in degree and connected components functions given to us in class were used, along with the AL and EL classes done in lab 6, and the DSF class also given to us in class.

To create a randomized Hamiltonian Cyle, the first step was to create an edge list and use the random function to choose random edges from that list. Next, use the AL class to turn the edges into an adjacency list. Now we simply traverse the list and if the vertex has a connected component, check if the in degree of that vertex is 2, if any on the vertices in the list does not have an in degree of 2, then a Hamiltonian Cycle does not exist.

Next, we use a tester function with a given range to use the randomization as the max number of tests.

To create the backtracking algorithm of the Hamiltonian cycle, a helper function was created. Since backtracking has recursive properties then two base cases were created: one to check if the number of vertices was equal to the length of the list, and another to check if the list was empty. Same as before, we check if the vertex has a connected component and an in degree of 2 and return the adjacency list representation.

Now the recursive call is made by first taking the first element of the list and another if we decide to not take the first element. Finally, we test the function in a similar way as with the previous algorithm.

To modify the edit distance algorithm, first we create a set with the vowels. We will use this to compare the two letters. Then, we simply modify the original function to check if the current letters are both in the vowels set, and if they both are not.

.

# Experimental Results

Graphs to be tested:

Hamiltonian Cycle:

A picture containing object

Description automatically generated

No Hamiltonian Cycle:

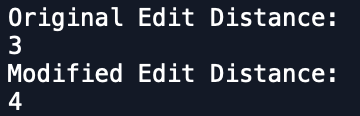
A picture containing object

Description automatically generated

A screenshot of a cell phone

Description automatically generated

Edit Distance words were “hello” and “hail”:



# Conclusion

Algorithm design techniques are a very important aspect of computer science. They are essential to make algorithms faster and more efficient, while saving time and space complexity. I personally liked creating the randomized Hamiltonian Cycle as it implemented knowledge from previous labs to do. I also like dynamic programming since, in my opinion, it’s the best algorithm design technique because it fixes the downsides of recursive algorithms.

# Appendix

﻿#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

"""

Created on Fri Dec 6 10:52:40 2019

@author: CarlosCardenas

"""

import numpy as np

import dsf

import graph\_AL as AL

import graph\_EL as EL

import random

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

def in\_degree(G, v):

c = 0

for i in range(len(G.al)):

for j in G.al[i]:

if j.dest == v:

c = c + 1

return c

def R\_Hamiltonian(V, E):

edge\_list = V.as\_EL()

for i in range(E):

Eh = random.sample(edge\_list.el, len(V.al))

# Creates an adjecenct list with the generated edges

al = AL.Graph(len(V.al), weighted = V.weighted, directed = V.directed)

for j in range(len(Eh)):

al.insert\_edge(Eh[j].source, Eh[j].dest)

if connected\_components(al) == 1:

for i in range(len(al.al)):

if in\_degree(al, i) != 2:

return False

return True

def test\_RH(V,E):

for i in range(100):

if R\_Hamiltonian(V,E) == True:

return "graph is a Hamiltonian Cycle"

return "graph is not a Hamiltonian Cycle"

# Recursive helper function

def Helper\_BT(V, E):

if len(V.el) == V.vertices:

graphAL = V.as\_AL()

if connected\_components(graphAL) == 1:

for i in range(len(graphAL.al)):

if in\_degree(graphAL, i) != 2:

return None

return graphAL

if len(E) == 0:

return

else:

V.el = V.el + [E[0]]

edges = Helper\_BT(V,E[1:])

if edges is not None:

return edges

V.el.remove(E[0])

return Helper\_BT(V, E[1:])

def BT(V):

E = V.as\_EL()

el = EL.Graph(len(V.al), weighted=V.weighted, directed=V.directed)

return Helper\_BT(el,E.el)

def test\_BT(V):

h = BT(V)

if isinstance(h, AL.Graph): #checks if object if of class

h.display()

return "is a Hamiltonian Cycle"

else:

return "is not a Hamiltonian Cycle"

#Original edit distance

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]

#Modified edit distance

def Vowel\_edit\_distance(s1,s2):

vowels = ['a','e','i','o','u'] #add a set with the vowels

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]]

if min(n) == d[i-1,j-1]:

if (not s1[i-1] in vowels and s2[j-1] in vowels) or (s1[i-1] in vowels and not s2[j-1] in vowels):

n = [d[i,j-1],d[i-1,j]]

d[i,j] = min(n)+1

return d[-1,-1]

# Hamiltonian Cycle Graph

g1 = AL.Graph(5)

g1.insert\_edge(0, 1)

g1.insert\_edge(1, 3)

g1.insert\_edge(1, 2)

g1.insert\_edge(3, 4)

g1.insert\_edge(2, 3)

g1.insert\_edge(4, 0)

g1.draw()

# No Hamiltonian Cycle Graph

g2 = AL.Graph(5)

g2.insert\_edge(0, 1)

g2.insert\_edge(1, 3)

g2.insert\_edge(1, 2)

g2.insert\_edge(2, 3)

g2.insert\_edge(4, 0)

g2.draw()

print("Graph 1", test\_RH(g1, 100))

print("Graph 2", test\_RH(g2, 100))

print()

# Hamiltonian Cycle Graph

g1 = AL.Graph(5)

g1.insert\_edge(0, 1)

g1.insert\_edge(1, 3)

g1.insert\_edge(1, 2)

g1.insert\_edge(3, 4)

g1.insert\_edge(2, 3)

g1.insert\_edge(4, 0)

g1.draw()

# No Hamiltonian Cycle Graph

g2 = AL.Graph(5)

g2.insert\_edge(0, 1)

g2.insert\_edge(1, 3)

g2.insert\_edge(1, 2)

g2.insert\_edge(2, 3)

g2.insert\_edge(4, 0)

g2.draw()

print("Graph 1", test\_BT(g1))

print("Graph 2", test\_BT(g2))

print()

print("Original Edit Distance:")

print(edit\_distance("hello","hail"))

print("Modified Edit Distance:")

print(Vowel\_edit\_distance("hello","hail"))

# 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 provide inappropriate assistance to any student in the class.

12/ 06/ 2019

Carlos Cardenas Date