CS 2302 Data Structures
Fall 2019

Lab Report #6

Due: November 19th, 2019

Professor: Olac Fuentes

TA: Anindita Nath

Introduction

For this lab we were tasked with three methods (insert_edge, delete_edge, and display) for an adjacency list, adjacency matrix and edge list then also create three more methods in each file that would allow us to create different graph representations of each (as_AL, as_AM, as_EL) and used them to covert each graph representation to an adjacency list and draw it within the spyder IDE. After all these methods where tasked with using these different graph representations to find the path of a word problem using breadth first search and depth first search.

Proposed Solution Design and Implementation

Part 1:

First, I created each method for an adjacency list because I felt it was the easiest. Then used similar techniques to create the same method for an adjacency matrix and edge list. After I went back each graph file and added the three methods (as_AL, as_AM, as_EL) an when completed would allow any of the three graph representations to be converted to the other two representations. To test that the conversions worked I used them to draw each graph using the .draw() method.

Part 2:

Next, in all three graph files I created two more methods (DFS “Depth first search” & BFS ”Breadth first search”) to find the solution path for the word problem, “You have a fox, a chicken and a sack of grain. You must cross a river with only one of them at a time. If you leave the fox with the chicken, he will eat it; if you leave the chicken with the grain, he will eat it. How can you get all three across safely?” My initial solution was (0,5,4,7,2,11,10,15) as this is what I came up with working out the problem on paper. To find this I first inputted all valid edges into an adjacency list.

Source	Dest
0	5
2	7
2	11
4	5
4	7
4	13
8	11
8	13
10	15
11	10

They were valid because the binary representations of the numbers did not break any rule of the word problem when they were moving to their destinations. After words to save time I used the as_AM and as EL methods to convert the adjacency list to an adjacency matrix and an edge list.

Experimental Results:

Adjacency list:

```
Adjacency List representation
[[(5,1)] [] [(7,1)(11,1)] [] [(5,1)(7,1)(13,1)] [(0,1)(4,1)] [] [(2,1)(4,1)] [(11,1)(13,1)] [] [(15,1)(11,1)] [(2,1)
(8,1)(10,1)] [] [(4,1)(8,1)] [] [(10,1)] ]
```

The Path List after Breadth First Search:

```
Breadth First Search Path is: 0 , 5 , 4 , 7 , 2 , 11 , 10 , 15
[[5], [], [11], [], [7, 13], [4], [], [2], [], [], [15], [10], [], [8], [], []]
```

The Path List after Depth First Search:

```
Depth First Search Path is: 0 , 5 , 4 , 13 , 8 , 11 , 10 , 15
[[5], [], [], [], [7, 13], [4], [], [], [11], [], [15], [2, 10], [], [8], [], []]
```

Adjacency Matrix:

```
Adjacency Matrix representation
[[-1 -1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1]
 [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1]
 [-1 -1 -1 -1 -1 -1 -1  1 -1 -1 -1  1 -1 -1 -1 -1]
 [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1]
 [-1 -1 -1 -1 -1  1 -1  1 -1 -1 -1 -1 -1  1 -1 -1]
 [ 1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1]
 [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1]
 [-1 -1  1 -1  1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1]
 [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  1 -1  1 -1 -1]
 [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1]
 [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  1 -1 -1 -1  1]
 [-1 -1  1 -1 -1 -1 -1 -1  1 -1  1 -1 -1 -1 -1 -1]
 [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1]
 [-1 -1 -1 -1  1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1]
 [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1]
 [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1  1 -1 -1 -1 -1 -1]]
```

The Path List after Breadth First Search:

```
Breadth First Search Path is: 0 , 5 , 4 , 7 , 2 , 11 , 10 , 15
[[5], [], [11], [], [7, 13], [4], [], [2], [], [], [15], [10], [], [8], [], []]
```

The Path List after Depth First Search:

```
Depth First Search Path is: 0 , 5 , 4 , 13 , 8 , 11 , 10 , 15
[[5], [], [], [], [7, 13], [4], [], [], [11], [], [15], [2, 10], [], [8], [], []]
```

Edge List:

```
Edge List representation
0 5 1
2 7 1
2 11 1
4 5 1
4 7 1
4 13 1
5 0 1
5 4 1
7 2 1
7 4 1
8 11 1
8 13 1
10 15 1
10 11 1
11 2 1
11 8 1
11 10 1
13 4 1
13 8 1
15 10 1
```

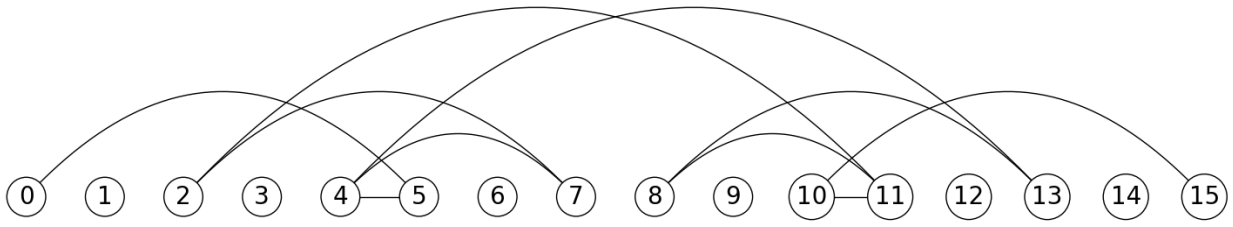
The Path List after Breadth First Search:

```
Breadth First Search Path is: 0 , 5 , 4 , 7 , 2 , 11 , 10 , 15
[[5], [], [11], [], [7, 13], [4], [], [2], [], [], [15], [10], [], [8], [], []]
```

The Path List after Depth First Search:

```
Depth First Search Path is: 0 , 5 , 4 , 13 , 8 , 11 , 10 , 15
[[5], [], [], [], [7, 13], [4], [], [], [11], [], [15], [2, 10], [], [8], [], []]
```

Drawing:



Conclusion

In conclusion, I was able to better understand how the two different search methods (breadth first and depth first) where able to solve the problem differently based on weather, they used a stack or a que when sorting through the graph representations. As in the case if the depth first search using a stack to find a path with the last destination when there were more than 1 path. Likewise, the breadth first search used the first path when there where multiple paths. Both still finding a solution to the problem but in different ways with different answers.

Appendix

Main.py:

```
'''
    Cs2302 Data Structures
    Issac Rivas (80604101)
    Lab 6
    Dr.Fuentes

'''
import graph_AL as gAL
import graph_AM as gAM
import graph_EL as gEL

def printPathBFS(list):
    x = 0
    print('Path is: ', end = '')
    while x != 15:
        print(x, ', ', end = '')
        x = list[x][0]
    print(x, end = '')
    print()

def printPathDFS(list):
    x = 0
    print('Path is: ', end = '')
    while x != 15:
        print(x, ', ', end = '')
        x = list[x][-1]
    print(x, end = '')
    print()


if __name__ == "__main__":

    g = gAL.Graph(16,directed = False)
    '''
        Proposed Solution:
        Take Chicken Across, Go Back Alone, Take the Fox Across With him, Returm With the chicken,
        Take The Grain Back Across,Leave the grain With the fox and cross back alone, Cross with the chicken.
        (0,5,4,7,2,11,10,15)

    '''

    #Legal Moves

    g.insert_edge(0,5)
    g.insert_edge(2,7)
    g.insert_edge(2,11)
```

```
    g.insert_edge(4,5)
    g.insert_edge(4,7)
    g.insert_edge(4,13)
    g.insert_edge(8,11)
    g.insert_edge(8,13)
    g.insert_edge(10,15)
    g.insert_edge(11,10)


    print('Adjacency List representation')
    g.display()
    g.draw()
    print()
    print('Breadth First Search ', end = '')
    printPathBFS(g.BFS())
    print('Depth First Search ', end = '')
    printPathDFS(g.DFS())
    print('------------------------------------------------------------')
    print()
    print('Adjacency Matrix representation')
    g2 = g.as_AM()
    g2.display()
    g2.draw()
    print()
    print('Breadth First Search ', end = '')
    printPathBFS(g2.BFS())
    print('Depth First Search ', end = '')
    printPathDFS(g2.DFS())
    print('------------------------------------------------------------')
    print()
    print('Edge List representation')
    g3 = g.as_EL()
    g3.display()
    g3.draw()
    print()
    print('Breadth First Search ', end = '')
    printPathBFS(g3.BFS())
    print('Depth First Search ', end = '')
    printPathDFS(g3.DFS())

```

graph_AL.py:

```
import numpy as np
import matplotlib.pyplot as plt
import math
from scipy.interpolate import interp1d
import graph_AM as gAM
import graph_EL as gEL

class Edge:
    def __init__(self, dest, weight=1):
        self.dest = dest
        self.weight = weight

class Graph:
    # Constructor
    def __init__(self, vertices, weighted=False, directed = False):
        self.al = [[] for i in range(vertices)]
        self.weighted = weighted
        self.directed = directed
        self.representation = 'AL'

    def insert_edge(self,source,dest,weight=1):
        if source >= len(self.al) or dest>=len(self.al) or source <0 or dest<0:
            print('Error, vertex number out of range')
        elif weight!=1 and not self.weighted:
            print('Error, inserting weighted edge to unweighted graph')
        else:
            self.al[source].append(Edge(dest,weight))
            if not self.directed:
                self.al[dest].append(Edge(source,weight))

    def delete_edge_(self,source,dest):
        i = 0
        for edge in self.al[source]:
            if edge.dest == dest:
                self.al[source].pop(i)
                return True
            i+=1
        return False

    def delete_edge(self,source,dest):
        if source >= len(self.al) or dest>=len(self.al) or source <0 or dest<0:
            print('Error, vertex number out of range')
        else:
            deleted = self.delete_edge_(source,dest)
            if not self.directed:
                deleted = self.delete_edge_(dest,source)
```

```
                deleted = self.delete_edge_(dest,source)
        if not deleted:
            print('Error, edge to delete not found')

    def display(self):
        print('[',end='')
        for i in range(len(self.al)):
            print('[',end='')
            for edge in self.al[i]:
                print('('+str(edge.dest)+','+str(edge.weight)+')',end='')
            print(']',end=' ')
        print(']')

    def draw(self):
        scale = 30
        fig, ax = plt.subplots()
        for i in range(len(self.al)):
            for edge in self.al[i]:
                d,w = edge.dest, edge.weight
                if self.directed or d>i:
                    x = np.linspace(i*scale,d*scale)
                    x0 = np.linspace(i*scale,d*scale,num=5)
                    diff = np.abs(d-i)
                    if diff == 1:
                        y0 = [0,0,0,0,0]
                    else:
                        y0 = [0,-6*diff,-8*diff,-6*diff,0]
                    f = interp1d(x0, y0, kind='cubic')
                    y = f(x)
                    s = np.sign(i-d)
                    ax.plot(x,s*y,linewidth=1,color='k')
                    if self.directed:
                        xd = [x0[2]+2*s,x0[2],x0[2]+2*s]
                        yd = [y0[2]-1,y0[2],y0[2]+1]
                        yd = [y*s for y in yd]
                        ax.plot(xd,yd,linewidth=1,color='k')
                    if self.weighted:
                        xd = [x0[2]+2*s,x0[2],x0[2]+2*s]
                        yd = [y0[2]-1,y0[2],y0[2]+1]
                        yd = [y*s for y in yd]
                        ax.text(xd[2]-s*2,yd[2]+3*s, str(w), size=12,ha="center", va="center")
            ax.plot([i*scale,i*scale],[0,0],linewidth=1,color='k')
            ax.text(i*scale,0, str(i), size=20,ha="center", va="center",
             bbox=dict(facecolor='w',boxstyle="circle"))
        ax.axis('off')
        ax.set_aspect(1.0)
```

```
92
93      def as_EL(self):
94          temp = gEL.Graph(16,directed = False)
95          for x in range(len(self.al)):
96              for y in range(len(self.al[x])):
97                  temp.insert_edge(x, self.al[x][y].dest)
98          return temp
99
100     def as_AM(self):
101         temp = gAM.Graph(len(self.al),directed = False)
102         for x in range(len(self.al)):
103             for y in range(len(self.al[x])):
104                 temp.insert_edge(x, self.al[x][y].dest)
105         return temp
106
107     def as_AL(self):
108         return self
109
110
111     def BFS(self):
112         frontierQueue = []
113         discoveredSet = []
114         frontierQueue.append(0)
115         discoveredSet.append(0)
116         path = [[] for i in range(len(self.al))]
117
118         while(len(frontierQueue) >  0):
119            currentV = frontierQueue.pop(0)
120            for x in range(len(self.al[currentV])):
121                if(self.al[currentV][x].dest not in discoveredSet):
122                    frontierQueue.append(self.al[currentV][x].dest)
123                    discoveredSet.append(self.al[currentV][x].dest)
124                    path[currentV].append(self.al[currentV][x].dest)
125
126         return path
127
128     def DFS(self):
129         Stack = []
130         discoveredSet = []
131         Stack.append(0)
132         discoveredSet.append(0)
133         path = [[] for i in range(len(self.al))]
134
135         while(len(Stack) >  0):
136            currentV = Stack.pop()
137            for x in range(len(self.al[currentV])):
138                if(self.al[currentV][x].dest not in discoveredSet):
139                    Stack.append(self.al[currentV][x].dest)
140                    discoveredSet.append(self.al[currentV][x].dest)
141                    path[currentV].append(self.al[currentV][x].dest)
142
143         return path
```

Graph_AM.py:

```python
import numpy as np
import matplotlib.pyplot as plt
import math
from scipy.interpolate import interp1d
import graph_AL as gAL
import graph_EL as gEL

class Graph:
    # Constructor
    def __init__(self, vertices, weighted=False, directed = False):
        self.vertices = vertices
        self.am = np.zeros((vertices,vertices),dtype=int)-1
        self.weighted = weighted
        self.directed = directed
        self.representation = 'AM'

    def insert_edge(self,source,dest,weight=1):
        if self.directed != True:
            self.am[source][dest], self.am[dest][source] = weight, weight
        self.am[source][dest] = weight

    def delete_edge(self,source,dest):
        if self.directed != True:
            self.am[source][dest], self.am[dest][source] = -1, -1
        self.am[source][dest] = -1

    def display(self):
        print(self.am)

    def draw(self):
        temp = self.as_AL()
        temp.draw()

    def as_EL(self):
        temp = gEL.Graph(16,directed = False)
        for x in range(len(self.am)):
            for y in range(len(self.am[x])):
                if self.am[x][y] != -1:
                    temp.insert_edge(x, y)
        return temp

    def as_AM(self):
        return self

    def as_AL(self):
        temp = gAL.Graph(16,directed = False)
```

```
        for x in range(len(self.am)):
            for y in range(len(self.am[x])):
                if self.am[x][y] != -1:
                    temp.insert_edge(x, y)
        return temp

    def BFS(self):
        frontierQueue = []
        discoveredSet = []
        frontierQueue.append(0)
        discoveredSet.append(0)
        path = [[] for i in range(self.vertices)]

        while(len(frontierQueue) >  0):
           currentV = frontierQueue.pop(0)
           for x in range(len(self.am[currentV])):
               if(x not in discoveredSet and self.am[currentV][x] != -1):
                   frontierQueue.append(x)
                   discoveredSet.append(x)
                   path[currentV].append(x)

        return path

    def DFS(self):
        Stack = []
        discoveredSet = []
        Stack.append(0)
        discoveredSet.append(0)
        path = [[] for i in range(self.vertices)]

        while(len(Stack) >  0):
           currentV = Stack.pop()
           for x in range(len(self.am[currentV])):
               if(x not in discoveredSet and self.am[currentV][x] != -1):
                   Stack.append(x)
                   discoveredSet.append(x)
                   path[currentV].append(x)

        return path

```

Graph_EL.py:

```python
import numpy as np
import matplotlib.pyplot as plt
import math
from scipy.interpolate import interp1d
import graph_AL as gAL
import graph_AM as gAM

class Edge:
    def __init__(self, source, dest, weight=1):
        self.source = source
        self.dest = dest
        self.weight = weight

class Graph:
    # Constructor
    def __init__(self, vert, weighted=False, directed = False):
        self.el = []
        self.vert = vert
        self.weighted = weighted
        self.directed = directed
        self.representation = 'EL'

    def insert_edge(self,source,dest,weight=1):
        if self.directed != True:
            for x in range(len(self.el)):
                if self.el[x].source == source and self.el[x].dest == dest and self.el[x].weight == weight:
                    return
        self.el.append(Edge(source, dest, weight))


    def delete_edge(self,source,dest):
        for x in range(len(self.el)-1):
            if self.el[x].source == source and self.el[x].dest == dest:
                del self.el[x]


    def display(self):
        for x in range(len(self.el)):
            print(self.el[x].source, self.el[x].dest, self.el[x].weight)

    def draw(self):
         temp = self.as_AL()
         temp.draw()

    def as_EL(self):
        return self
```

```
47
48      def as_AM(self):
49          temp = gAM.Graph(self.vert ,directed = False)
50          for x in range(len(self.el)):
51              temp.insert_edge(self.el[x].source, self.el[x].dest)
52          return temp
53
54      def as_AL(self):
55          temp = gAL.Graph(self.vert ,directed = False)
56          for x in range(len(self.el)):
57              temp.insert_edge(self.el[x].source, self.el[x].dest)
58          return temp
59
60      def BFS(self):
61          frontierQueue = []
62          discoveredSet = []
63          frontierQueue.append(0)
64          discoveredSet.append(0)
65          path = [[] for i in range(self.vert)]
66
67          while(len(frontierQueue) >  0):
68             currentV = frontierQueue.pop(0)
69             for x in range(len(self.el)):
70                 if(self.el[x].dest not in discoveredSet and self.el[x].source ==  currentV):
71                     frontierQueue.append(self.el[x].dest)
72                     discoveredSet.append(self.el[x].dest)
73                     path[currentV].append(self.el[x].dest)
74
75          return path
76
77      def DFS(self):
78          Stack = []
79          discoveredSet = []
80          Stack.append(0)
81          discoveredSet.append(0)
82          path = [[] for i in range(self.vert)]
83
84          while(len(Stack) >  0):
85             currentV = Stack.pop()
86             for x in range(len(self.el)):
87                 if(self.el[x].dest not in discoveredSet and self.el[x].source ==  currentV):
88                     Stack.append(self.el[x].dest)
89                     discoveredSet.append(self.el[x].dest)
90                     path[currentV].append(self.el[x].dest)
91
92          return path
```

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.