LAB 6: Graph Search: Uninformed search and Informed search

6.1 Objectives:

1. To learn and Implement Depth-First-Search algorithm

6.2 Depth-first search:

Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. One starts at the root (selecting some arbitrary node as the root in the case of a graph) and explores as far as possible along each branch before backtracking.

6.2.1 Pseudocode:

Input: A graph G and a vertex v of G
Output: All vertices reachable from v labelled as discovered

A recursive implementation of DFS:

```
1  procedure DFS(G,v):
2      label v as discovered
3      for all edges from v to w in G.adjacentEdges(v) do
4          if vertex w is not labeled as discovered then
5              recursively call DFS(G,w)
```

A non-recursive implementation of DFS:

```
1  procedure DFS-iterative(G,v):
2      let S be a stack
3      S.push(v)
4      while S is not empty
5          v = S.pop()
6          if v is not labeled as discovered:
7              label v as discovered
8              for all edges from v to w in G.adjacentEdges(v) do
9                  S.push(w)
```

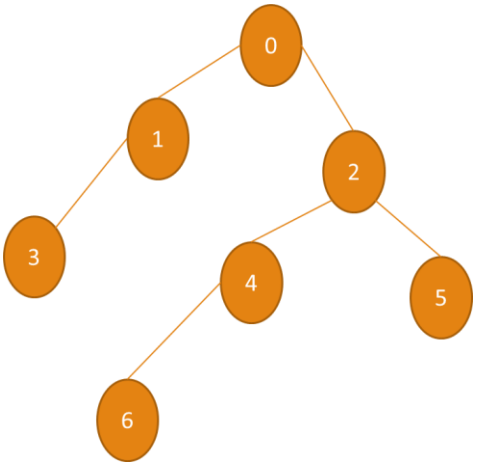
6.2.2 Iterative Deepening Depth-first search:

Iterative Deepening Depth-first search (ID-DFS) is a state space/graph search strategy in which a depth-limited version of depth-first search is run repeatedly with increasing depth limits until the goal is found. IDDFS is equivalent to breadth-first search, but uses much less memory; on each iteration, it visits the nodes in the search tree in the same order as depth-first search, but the cumulative order in which nodes are first visited is effectively breadth-first.

6.2.3 Pseudocode:

```
Set all nodes to "not visited";

s = new Stack();    ******* Change to use a stack

s.push(initial node);    ***** Push() stores a value in a stack

while ( s ≠ empty ) do
{
  x = s.pop();         ****** Pop() remove a value from the stack

  if ( x has not been visited )
  {
    visited[x] = true;        // Visit node x !

    for ( every edge (x, y)  /* we are using all edges ! */ )
      if ( y has not been visited )
            s.push(y);       ***** Use push() !
  }
}
```

Example:



Code:

```
vertexList = ['0', '1', '2', '3', '4', '5', '6']
edgeList = [(0,1), (0,2), (1,0) , (1,3) , (2,0) , (2,4) , (2,5) , (3,1),
(4,2) , (4,6), (5,2), (6,4)]
graphs = (vertexList, edgeList)
```

```
def dfs(graph, start):
    vertexList, edgeList = graph
    visitedVertex = []
    stack = [start]
    adjacencyList = [[] for vertex in vertexList]

    for edge in edgeList:
        adjacencyList[edge[0]].append(edge[1])

    while stack:
        current = stack.pop()
        for neighbor in adjacencyList[current]:
            if not neighbor in visitedVertex:
                stack.append(neighbor)
        visitedVertex.append(current)
    return visitedVertex

print(dfs(graphs, 0))
```

6.3 Breadth-first search:

Breadth-first search (BFS) is an algorithm that is used to graph data or searching tree or traversing structures. The full form of BFS is the Breadth-first search.

The algorithm efficiently visits and marks all the key nodes in a graph in an accurate breadthwise fashion. This algorithm selects a single node (initial or source point) in a graph and then visits all the nodes adjacent to the selected node. Remember, BFS accesses these nodes one by one.

Once the algorithm visits and marks the starting node, then it moves towards the nearest unvisited nodes and analyses them. Once visited, all nodes are marked. These iterations continue until all the nodes of the graph have been successfully visited and marked.

6.3.1 Graph traversals

A graph traversal is a commonly used methodology for locating the vertex position in the graph. It is an advanced search algorithm that can analyze the graph with speed and precision along with marking the sequence of the visited vertices. This process enables you to quickly visit each node in a graph without being locked in an infinite loop.

6.3.2 How BFS works:

1. Graph traversal requires the algorithm to visit, check, and/or update every single un-visited node in a tree-like structure. Graph traversals are categorized by the order in which they visit the nodes on the graph.

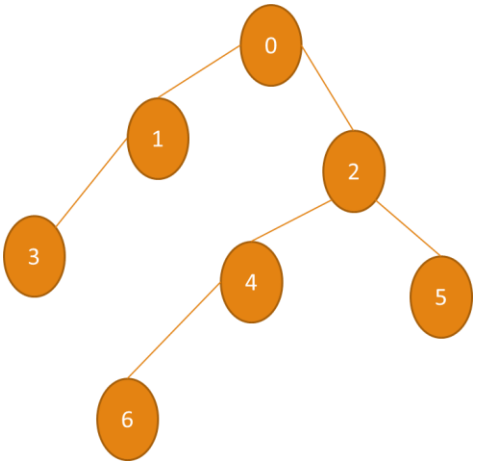
2. BFS algorithm starts the operation from the first or starting node in a graph and traverses it thoroughly. Once it successfully traverses the initial node, then the next non-traversed vertex in the graph is visited and marked.

3. Hence, you can say that all the nodes adjacent to the current vertex are visited and traversed in the first iteration. A simple queue methodology is utilized to implement the working of a BFS algorithm

6.3.3 Pseudocode:

```
Set all nodes to "not visited";

  q = new Queue();

  q.enqueue(initial node);

  while ( q ≠ empty ) do
  {
    x = q.dequeue();

    if ( x has not been visited )
    {
      visited[x] = true;        // Visit node x !

      for ( every edge (x, y)  /* we are using all edges ! */ )
        if ( y has not been visited )
              q.enqueue(y);      // Use the edge (x,y) !!!
    }
  }
```

Example:



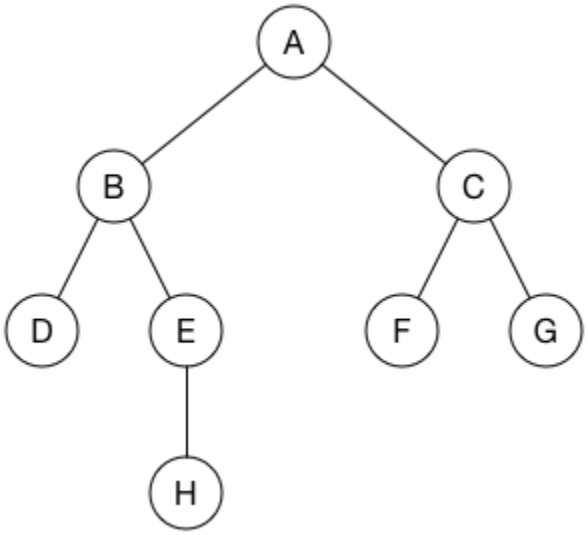
Code:

```
vertexList = ['0', '1', '2', '3', '4', '5', '6']
edgeList = [(0,1), (0,2), (1,0) , (1,3) , (2,0) , (2,4) , (2,5) , (3,1),
(4,2) , (4,6), (5,2), (6,4)]
graphs = (vertexList, edgeList)
def bfs(graph, start):
    vertexList, edgeList = graph
    visitedList = []
```

```
    queue = [start]
    adjacencyList = [[] for vertex in vertexList]

    # fill adjacencyList from graph
    for edge in edgeList:
        adjacencyList[edge[0]].append(edge[1])

    # bfs
    while queue:
        current = queue.pop()
        for neighbor in adjacencyList[current]:
            if not neighbor in visitedList:
                queue.insert(0,neighbor)
        visitedList.append(current)
    return visitedList

print(bfs(graphs, 0))
```

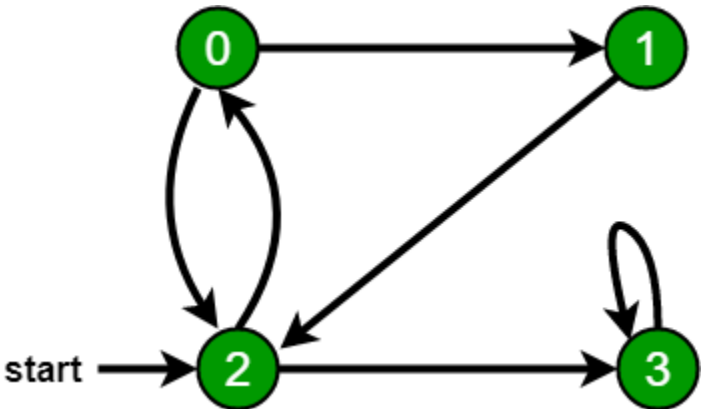
6.4 TASKS

TASK: Consider the below graph;



Use BFS to find the shortest path between A and F. (Hint: the distance between any consecutive vertices is 1, i.e. distance between A and D is 2 ((A to B=1) + (B to D=1) = 2)

TASK: Consider the below graph;



Using DFS, check if there is any path exists between any two nodes? Also the return the path.

e.g. If user two vertices i.e. 2 and 1; the program should return : Yes the paths exist, which are [2,1],[2,0,1].

LAB 7: Introduction to NumPy, Pandas, Scikit-learn and Matplotlib Python Packages

7.1 Objectives:

2. To learn about Python most widely used libraries in machine learning

7.2 Different Python Packages

7.2.1 NUMPY

NumPy is the cornerstone toolbox for scientific computing with Python. NumPy provides, among other things, support for multidimensional arrays with basic operations on them and useful linear algebra functions. Many toolboxes use the NumPy array representations as an efficient basic data structure.

Examples

#importing numpy package

```
mport numpy as np
b=np.array([[[1,2,3,5],[2,3,4,4]],[[1,2,3,5],[2,3,4,4]]])

#printing the data type
print(b.dtype)
out: int32

#printing the dimension of the NumPy array
print(b.ndim)
Out: 3

#printing the shape the NumPy array
print(b.shape)
Out: (2, 2, 4)

# printing the size the NumPy array i.e. total number of elements
print(b.size)
out: 16

#to generate an array of numerical numbers from 10 to 100 with 2 steps
c=np.arange(10,100,2)
print(c)

#to generate an array of zeros
b=np.zeros(10)
print(b)

#to generate a array of ones
b=np.ones(10)
print(b)
```

```
#to shuffle data
a=np.random.permutation(c)
print(a)

#to generate random uniform noise
d=np.random.rand(1000)

#to generate random gaussian noise
d=np.random.randn(1000)

#to select random integer
f=np.random.randint(10,40)

#to reshape an NumPy array
d=np.arange(20).reshape(4,5)
print(d)

#slicing
print(d[1:3,2:4])
```

7.2.2 SCIKIT-Learn

Scikit-learn is a machine learning library built from NumPy, SciPy, and Matplotlib. Scikit-learn offers simple and efficient tools for common tasks in data analysis such as classification, regression, clustering, dimensionality reduction, model selection, and preprocessing.

Example

```
#Loading an example dataset

from sklearn import datasets

iris = datasets.load_iris()

digits = datasets.load_digits()
```

7.2.3 PANDAS

Pandas5 provides high-performance data structures and data analysis tools. The key feature of Pandas is a fast and efficient DataFrame object for data manipulation with integrated indexing. The DataFrame structure can be seen as a spreadsheet which offers very flexible ways of working with it. You can easily transform any dataset in the way you want, by reshaping it and adding or removing columns or rows. It also provides high-performance functions for aggregating, merging, and joining datasets. Pandas also has tools for importing and exporting data from different formats: comma-separated value (CSV), text files, Microsoft Excel, SQL databases, and the fast HDF5 format. In many situations, the data you have in such formats will not be complete or totally structured. For such cases, Pandas offers handling of missing data and intelligent data alignment. Furthermore, Pandas provides a convenient Matplotlib interface.

Examples

```
#importing pandas library
import pandas as pd
```

```
#creating Pandas series
a=pd.Series([1,2,3,4],index=['a','b','c','d'])
print(a)

marks={"A":10,"B":20,"C":30}
print(marks)
grades={"A":2,"B":3,"C":5}

#converting dictionaries to the Pandas series
pd1=pd.Series(marks)
print(pd1)
pd2=pd.Series(grades)
#print(marks)
#print(pd1)
#Pandas DataFrame
pd3=pd.DataFrame({"marks":pd1,"grades":pd2})
print(pd3)

#adding dictionary to the Pandas DataFrame
pd3["percentage"]=pd3["marks"]/100
print(pd3)

#deleting from Pandas Dataframe
del pd3["percentage"]

#Thresholding
print(pd3[pd3['marks']>10])
print(pd3)
```

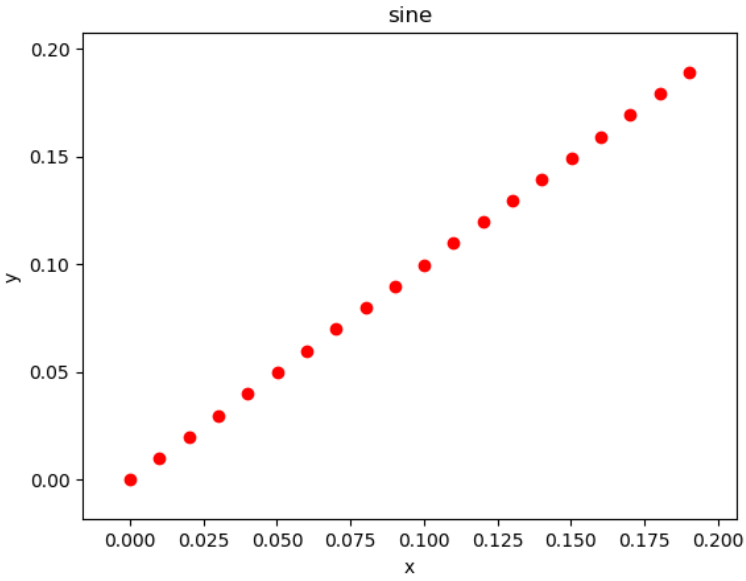
7.2.4 Matplotlib

Matplotlib produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shell, web application servers, and various graphical user interface toolkits. matplotlib.pyplot is a collection of functions that make matplotlib work like MATLAB. Majority of plotting commands in pyplot have MATLAB analogs with similar arguments.

Example

#importing Matplotlib.Pyplot

```
import matplotlib.pyplot as plt
x=np.linspace(0,10,1000)

#conitonus plotting

plt.plot(x,np.sin(x), color="red")

# Discrete Plotting
plt.scatter(x[0:20],np.sin(x[0:20]), color="red")
plt.xlabel("x")
plt.ylabel("y")
plt.title("sine")
plt.show()
```



7.3 TASKS

`TASK 1:`

Write a NumPy program to create a random 10x4 array and extract the first five rows of the array and store them into a variable.

`TASK 2:`

Write a Pandas program to select the rows where the number of attempts in the examination is greater than 2.
Sample Python dictionary data and list labels:
exam_data = {'name': ['Anastasia', 'Dima', 'Katherine', 'James', 'Emily', 'Michael', 'Matthew', 'Laura', 'Kevin', 'Jonas'],
'score': [12.5, 9, 16.5, np.nan, 9, 20, 14.5, np.nan, 8, 19],
'attempts': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
'qualify': ['yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'no', 'yes']}
labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
Expected Output:
Number of attempts in the examination is greater than 2:
name score attempts qualify
b Dima 9.0 3 no
d James NaN 3 no
f Michael 20.0 3 yes

`TASK 3:`

From the sample data given in TASK 2; write a program to calculate the average of the scores. The program should be able to ignore NaN values.

Expected Output:

The average score is : 13.56