Department of Computer Technology and Information Systems

CTIS264 – Computer Algorithms

Spring 2019 - 2020

Lab Guide 5 – Week 6

Instructor : Füsun YÜRÜTEN **Assistant** : Leyla SEZER	
OBJECTIVE: • **Empirical Analysis of Polynomial Evaluation** • **Writing DFS with dictionaries** • **Writing BFS with dictionaries**	

Q1. Write a Python program that makes the empirical analysis of the polynomial evaluation with the below Brute-Force Algorithm;

- Writes a function that gets **'a'** as a list and '**x'**, then calculates and returns the polynomial values of p(x).

Problem: Find the value of polynomial

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x^1 + a_0$$

at a point $x = x_0$

ALGORITHM POLINOM(A[0..n-1], X)

p ← *a[0]*
power ← *1*
for *i* ← *1* ***to*** *n* ***do***
    power ← *power* ∗ *x*
    p ← *p* + *a[i]* ∗ *power*
return *p*

- Program generates a random array. Size of the array is 1000 and the numbers between 0-100000.
- X value will be read by the main program.

Output:

```
Enter the value of X: 2

6546592718091443012521151843030979365516097314738788090621968610717843782688542839657966
7320017859566233980833847601454706135547097230656326397933767916101308117417790282004900
4362094967560417081240477729960406111244641301841378166718474052564821100655007023628964
7585305734363840534358989276459227193250879

0.0286100 seconds
```

Q2. Write a Python program that makes the empirical analysis of the polynomial evaluation with the below Brute-Force Algorithm; Set the value of X to 2.

```
ALGORITHM POLINOM( A[0..n-1], X)
p ← 0.0
for i ← n downto 0 do
   power ← 1
  for j ← 1 to i do
       power ← power * x
           p ← p + a[i] * power
return p
```

Output:

```
8.230999583414169e+306
0.1002443 seconds
```

Q3. Write a Python program which will print the DFS TRAVERSAL LIST for a given **connected graph**;

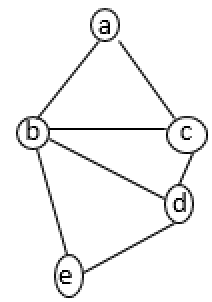
- Reads the number of vertexes for this graph
- Reads the VERTEX LIST in the given number as an one-dim array
- Reads two dimensional array for matrix representation of a graph – ADJACENCY MATRIX
- Defines a GRAPH as a dictionary which contains all the vertexes as a key, and it's neighbors as a list. You have to use the adjacency matrix and the vertex list that you read
- Implements the below recursive Depth First Search Algorithm that displays the visited graph vertices.

```
ALGORITHM DFS(Graph, vertex, path[0..n])
// Implements a depth-first-search traversal of a given graph
//INPUT: Graph{vertex: list of neighbor vertexes}
//          vertex, beginning vertex to search
//          path will be used to construct traversal list in recursion
//OUTPUT: List of vertices in DFS traversal - path

//visits recursively all the unvisited vertices of a connected graph
add vertex to path
for each neighbor in graph[vertex] do

     if neighbor not in path          //for the non-visited neighbors
       then  path← DFS(Graph, neighbor, path)
return path
```

⇨ You may use the following examples for testing your program

Example 1:

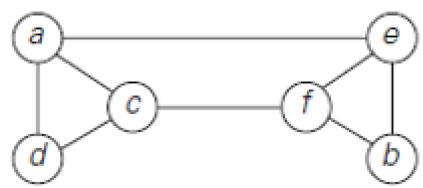
Output:




```
Enter the number of vertexes: 5
Enter the vertex list for graph:
abcde
Enter the adjacency matrix:
0 1 1 0 0
1 0 1 1 1
1 1 0 1 0
0 1 1 0 1
0 1 0 1 0
The graph is
{'a': ['b', 'c'], 'b': ['a', 'c', 'd', 'e'], 'c': ['a', 'b', 'd'], 'd': ['b', 'c', 'e'],
'e': ['b', 'd']}
DFS traversal:
['a', 'b', 'c', 'd', 'e']
```

Example 2:

Output:




```
Enter the number of vertexes: 6
Enter the vertex list for graph:
abcde
Enter the adjacency matrix:
0 0 1 1 1 0
0 0 0 0 1 1
1 0 0 1 0 1
1 0 1 0 0 0
1 1 0 0 0 1
0 1 1 0 1 0
The graph is
{'a': ['c', 'd', 'e'], 'b': ['e', 'f'], 'c': ['a', 'd', 'f'], 'd': ['a', 'c'], 'e':
['a', 'b', 'f'], f: ['b', 'c', 'e'] }
DFS traversal:
['a', 'c', 'd', 'f', 'b' , 'e']
```

Q4. Write a Python program which will print the BFS TRAVERSAL LIST for a given **connected graph**;

- Reads the number of vertexes for this graph
- Reads the VERTEX LIST in the given number as an one-dim array
- Reads two dimensional array for matrix representation of a graph – ADJACENCY MATRIX
- Defines a GRAPH as a dictionary which contains all the vertexes as a key, and it's neighbors as a list. You have to use the adjacency matrix and the vertex list that you read
- Implements the below algorithm for Breadth First Search Algorithm by using queues that displays the visited graph vertices.

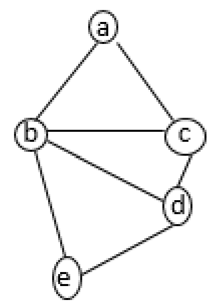
ALGORITHM BFS(Graph, vertex)

```
// Implements a Breadth-first-search traversal of a given graph
//INPUT: Graph{vertex: list of neighbor vertexes}
//              vertex, beginning vertex to search
//OUTPUT: List of visited vertices in BFS traversal

//visits all the unvisited vertices of a connected graph using a queue
Put the beginning vertex to queue
Put the beginning vertex to visited list

while queue not ends do
      currvertex ← pop first element from queue
      for each neighbor in graph[currvertex] do
              if neighbor not in visited                //for the non-visited neighbors
                 then add neighbor to visited
                      add neighbor to queue
return visited
```

⇨ You may use the following examples for testing your program

Example 1:

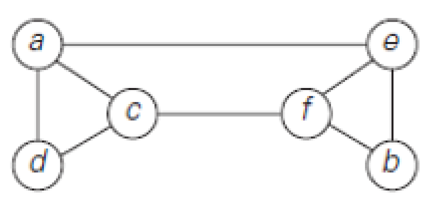
Output:




```
 Enter the number of vertexes: 5
 Enter the vertex list for graph:
 abcde
 Enter the adjacency matrix:
 0 1 1 0 0
 1 0 1 1 1
 1 1 0 1 0
 0 1 1 0 1
 0 1 0 1 0
 The graph is
 {'a': ['b', 'c'], 'b': ['a', 'c', 'd', 'e'], 'c': ['a', 'b', 'd'], 'd': ['b', 'c', 'e'],
 'e': ['b', 'd']}
BFS TRAVERSAL :  ['a', 'b', 'c', 'd', 'e']
```

Example 2:

Output:




```
 Enter the number of vertexes: 6
 Enter the vertex list for graph:
 abcdef
 Enter the adjacency matrix:
 0 0 1 1 1 0
 0 0 0 0 1 1
 1 0 0 1 0 1
 1 0 1 0 0 0
 1 1 0 0 0 1
 0 1 1 0 1 0
 The graph is
 {'a': ['c', 'd', ‘e’], 'b': ['e', 'f'], 'c': ['a', 'd', 'f'], 'd': ['a', 'c'], 'e':
 [‘a’, 'b', 'f'], f: [‘b’, ‘c’, ‘e’] }
 BFS traversal:
    ['a', 'c', 'd', 'e', 'f', 'b']
```