Victor-Stefan Iacob

Practice 1 – Graph Algorithms

Python

graph.py – class Graph

Attributes:

- num – the number of vertices
- pairs – the number of edges/pairs
- out – a dictionary that has the vertices as keys, and each of them has a list of out bound neighbours
- in - a dictionary that has the vertices as keys, and each of them has a list of in bound neighbours
- cost - a dictionary that has both of the vertices as keys(tuples of 2 vertices), and each of them has the cost as value

Methods:

- __init__(self)
  This is a default constructor function for a graph. (all the attributes are 0 or empty).
  Complexity: Theta(1)

- readGraph(self, file_name)
  Precondition: file_name must be a valid a string that contains the name of the file
  This is a function that reads a graph from a file. The file should contain the number of vertices and the number of the pairs on the first line and in the following lines it should contain the endpoints with the cost of each one of them.
  It is used for reading files that include the number of vertices.
  Complexity: Theta(n+m)

- readGraph2(self, file_name)
  Precondition: file_name must be a valid a string that contains the name of the file

This is a function that reads a graph from a file. The file should contain the number of the pairs on the first line and in the following lines it should contain the endpoints with the cost of each one of them.
It is used for reading files that do not include the number of vertices.
Complexity: Theta(n+m)

- check_file(self, file_name)
  Precondition: file_name must be a valid a string that contains the name of the file
  This function checks the file to see if it is the method 1 or the method 2 of reading.
  Complexity: Theta(n+m)

- saveGraph(self, file_name)
  Precondition: file_name must a string that contains the name of the file in which we want to save
  This is a function that saves a graph in a file. The file will contain the number of vertices and the number of the pairs on the first line and in the following lines it will contain the endpoints with the cost of each one of them.
  Complexity: Theta(n+m)

- parseOut(self, x):
  Precondition: x must be a valid integer which represents a vertex from the graph
  This function returns a list of out bound neighbours of a given vertex.
  Complexity: O(n)

- parseIn(self, x):
  Precondition: x must be a valid integer which represents a vertex from the graph
  This function returns a list of in bound neighbours of a given vertex.
  Complexity: O(n)

- parseEdges(self):
  This function returns the list of the edges in the graph. (with both endpoints)
  Complexity: Theta(m)

- vertices_number(self):
  This function returns the number of vertices in the graph.
  Complexity: Theta(1)

- pairs_number(self):

This function returns the number of edges/pairs in the graph.
Complexity: Theta(1)

- is_vertex(self, x):
  x: an integer which represents a vertex
  This function returns True if the vertex exists and false if it does not exist.
  Complexity: O(n)

- is_pair(self, p1, p2):
  p1, p2: integers that represent the endpoints of a pair
  This function returns True if the edge exists and false if it does not exist.
  Complexity: O(m)

- outDegree_num(self, x):
  Precondition: x must be a valid integer which represents a vertex from the graph
  This function returns the number of out bound neighbours of a vertex.
  Complexity: Theta(1)

- inDegree_num(self, x):
  Precondition: x must be a valid integer which represents a vertex from the graph
  This function returns the number of in bound neighbours of a vertex.
  Complexity: Theta(1)

- get_cost(self, p1, p2):
  Precondition: p1, p2 must be integers that represent the endpoints of a pair in the graph
  This function returns the cost of an edge that starts from p1 and end in p2.
  Complexity: Theta(1)

- set_cost(self, p1, p2, cost):
  Precondition: p1, p2 must be integers that represent the endpoints of a pair in the graph
  This function changes the cost of an edge that starts from p1 and end in p2.
  Complexity: Theta(1)

- add_edge(self, p1, p2, cost):
  Precondition: p1, p2 must be integers that represent the endpoints of a pair in the graph
  cost: integer that represents the cost of an edge
  This function adds an edge to the set of edges of the graph.
  The edge has to be unique, because it is validated before, in the ui part.

Complexity: Theta(1)

- remove_edge(self, p1, p2)
  Precondition: p1, p2 must be integers that represent the endpoints of a pair in the graph
  This function removes an edge from the set of edges of the graph.
  The edge has to be in the set, because it is validated before, in the ui part.
  Complexity: Theta(1)

- add_vertex(self, x)
  Precondition: x must be an integer which represents a vertex that is not in the graph
  This function adds a vertex to the graph.
  The vertex has to be unique, because it is validated before, in the ui part.
  Complexity: Theta(1)

- remove_vertex(self, x)
  Precondition: x must be an integer which represents a vertex from the graph
  This function removes a vertex from the graph.
  The vertex has to belong to the graph.
  Complexity: O(max(outDegree,inDegree)*m+n)

- copy_graph(self)
  This function returns a copy of the graph.
  Complexity: Theta(n+m)

- clear_graph(self)
  This function reinitialises a default graph(all attributes become 0 or empy).
  Complexity: Theta(1)

generateGraph.py – class generateGraph

Methods:

- generate_graph(vertices, paris)
  vertices: an integer given by the user that represents the number of vertices
  pairs: an integer given by the user that represents the number of pairs

This function generates and returns a random graph if the parameters are correctly given. It generates all the possible edges in the graph, then it substratcs randomly the number of pairs given by the user.
Complexity: Theta(n*n)

*for complexity: n represents the number of vertices

m represents the number of pairs/edges