Gog Andreea-Alexandra

Group 913

*Documentation – Assigment 1*

**Class Graph:**

* Has no parameters
* Has 4 constructors:
* Din: a dictionary that has every vertex as a key. The value associated with the key is a list with all the predecessors of that vertex (empty list if vertex has no predecessors). Is an empty dictionary at first.
* Dout: a dictionary that has every vertex as a key. The value associated with the key is a list with all the successors of that vertex (empty list if vertex has no successors). Is an empty dictionary at first.
* DCost: a dictionary that has every edge as a key. The value associated with the key is the cost of the edge. Is an empty dictionary at first.
* NrVertices: number of vertices (0 in the begining)

**Class UI:**

* Has no parameters
* Has 2 constructors:
* Graph: a graph object
* CopyGraph: a copy of the graph (is the graph read from the file if the user does not set it diffrent)

**Option 1: Read graph from file**

* It has 2 option: reading from a file that has on the first line the number of vertices and edges of the graph followed by multiple lines with information about de edges (source, target, cost) or the second version where the file contains only lines about the edges and isolated points are alone on the line
* It receives the file name as parameter given by the user, returns nothing (sets info about the graph)
* Complexity: O(m+n)

**Option 2: Print graph**

* It prints on the output the information from the graph: each edge with source, target and cost and isolated vertices are specified
* Complexity: O(m+n)

**Option 3: Search for vertex**

* It checks if the vertex given by the user is found in the list of keys from DIn (dictionary in)
* It returns true if found, false otherwise
* Complexity: O(n)

**Option 4: Search for edge**

* Precondition: both vertices must exist (source and target)
* Return: true if target is found as a value for the key source in DOut, false otherwise
* Complexity: O(n)

**Option 5: Add vertex**

* Precondition: vertex does not already exist
* Return: an isolated vertex is added (empty list in DIn, DOut)
* Complexity: O(n)

**Add edge**

* Precondition: edge does not already exist, but vertices exist
* Return: the source is added as predecessor for target and targe is added as a successor for the source
* Complexity: O(n)

**Option 6: Remove vertex**

* Precondition: vertex exists
* Return: vertex is eliminated as a predecessor from every vertex
* Complexity: O(n)

**Remove edge**

* Precondition: vertices exist, edge exists
* Return: targes is deleted as a successor of the source and the source is deleted as a predecessor of the target and the key (source, target) is deleted from DCost
* Complexity: O(n)

**Option 7: Modify cost**

* Precondition: edge exists
* Modifies the value from DCost

**Option 8: Get number of vertices**

* Return: numbers of keys from DIn
* Theta(1)

**Option 9: Get in degree/ out degree**

* Precondition: vertex exists
* Return: length of list from Din/Dout (number of predecessors and successors)
* Complexity: O(n)

**Option 10: Parse the set of vertices**

* Return: list with all keys from Din (even if isolated, a vertex is added with empty list as value in Din)
* Theta(1)

**Option 11: Parse the inbound vertices**

* Precondition: vertex exists
* Return: list with values associated to the key vertex in Din
* Theta(1)

**Option 12: Parse the outbound vertices**

* Precondition: vertex exists
* Return: list with values associated to the key vertex in DOut
* Theta(1)

**Option 13: Save graph**

* It saves the graph info into a file given as a parameter from the user (each line is an edge: source, target and cost and isolated vertices are a line with only the vertex)
* Complexity (n\*i)

**Option 14: Copy graph**

* It creates another object graph that have the same info as the intial one(user has the option to create the copy or replace the graph with a copy: if none was made is replaced with the graph read from the file)
* Complexity: Theta(1)

**Option 15: Generate graph**

* It has as parameters number of vertices and edges
* Precondition: number of vertices allows the function to create such a graph
* Return: a random graph (random, valid combinations of vertices with a random cost)
* Complexity O(n+n)