**Documentation Graph – Practical Work No. 1**

**Specification:**

The class OrderedGraphCost will provide information for a directed graph.

Python:

\_\_init\_\_(self, nrV):

Constructs a graph with a nrV number of vertices and without edges

addEdge(self, v1, v2, cost):

Adds an edge between v1 and v2 with given cost

setOfVertices(self):

returns an iterable list with the set of vertices

isVertex(self, val):

returns true if val is a vertex, false otherwise

isEdge(self, v1, v2):

return true if there is an edge between vertices v1 and v2

inDegree(self, val):

returns the indegree of vertex val

outDegree(self, val);

returns the outdegree of vertex val

inboundEdge(self, x):

returns an iterable – the inbound edges of x

outboundEdge(self, x):

returns an iterable – the outbound edges of x

changeCost(self, x, y, val):

changes the cost of edge (x, y) with val

getCost(self, x, y):

returns the cost of edge (x, y)

nrEdges(self):

return the number of edges

removeEdge(self, v1, v2):

removes the edge between v1 and b2

nrVertives(self):

returns the number of vertices

addVertex(self):

adds a vertex to the graph

removeVertex(self, val):

removes a vertex from a graph

\_\_str\_\_(self):

returns a string representing the graph

C++:

explicit OrderedGraphCost(int nrVertices);

Constructs a graph with nrVertices number of vertices and without edges

int nrVertices();

returns the number of vertices

std::pair<std::map<int, std::vector<int>>::iterator , std::map<int, std::vector<int>>::iterator> setOfVertices();

returns the set of vertices as a pair of 2 iterators

bool isVertex(int v);

returns true if v is a vertex, false otherwise

bool isEdge(int v1, int v2);

returns true if there is an edge between v1 and v2

void addEdge(int v1, int v2, int cost);

adds an edge with given cost between v1 and v2

int inDegree(int x);

returns the indegree of x

int outDegree(int x);

returns the outdegree of x

std::pair <std::vector<int>::iterator, std::vector<int>::iterator> inboundEdge(int x);

returns the set of inbound edges as a pair of 2 iterators

std::pair <std::vector<int>::iterator, std::vector<int>::iterator> outboundEdge(int x);

returns the set of outbound edges as a pair of 2 iterators

void changeCost(int x, int y, int val);

changes the cost of the edge (x, y)

int getCost(int x, int y);

returns the cost of the edge (x, y)

void removeEdge(int x, int y);

removes the edge (x, y)

void addVertex(int val);

adds a vertex to the graph

void removeVertex(int val);

removes a vertex from the graph

std::string toString();

transforms the graph into a string

**Implementation:**

**Python:**

The implementation uses 3 dictionaries:

def \_\_init\_\_(*self*, *nrV*):

self.\_dictIn = {}

self.\_dictOut = {}

self.\_costDict = {}

*for* i in range(nrV):

self.\_dictIn[i] = []

self.\_dictOut[i] = []

First 2 dictionaries have values from an int to a list of integers – the inbound and outbound edges

The last one is one that has as key a pair (x, y) which is the edge and has as a value an integer – the cost

**C++:**

The implementation uses 3 maps:

1. between a std::pair<int, int> and an int (edge – cost)
2. Between int and std::vector<int> (inbound edges)
3. Between int and std::vector<int> (outbound edges)

The Graph has the private elements

private:  
 std::map<int, std::vector<int>> outMap, inMap;  
 std::map<std::pair<int, int>, int> costMap;