DirectedGraph Class Specification

Implementation:

```
class DirectedGraph {

 public:

       int n, m;

   // Automatically initialized to an empty map
    vector<int> vertices;
    unordered_map<int, vector<int>> in_vertices;
    unordered_map<int, vector<int>> out_vertices;
    unordered_map<long long, int> edges;
}
```

Constructor & Destructor:

- `DirectedGraph()`: Constructs an empty directed graph.
- `DirectedGraph(int n)`: Constructs a directed graph with n vertices.
- `~DirectedGraph()`: Destroys the directed graph and releases resources.

Operators:

- `DirectedGraph& operator=(const DirectedGraph& other_graph) noexcept`: Assigns the contents of `other_graph` to this graph. Does not throw exceptions.

Member Functions:

- `int GetNrOfVertices() const`: Returns the number of vertices in the graph.
- `int GetNrOfEdges() const`: Returns the number of edges in the graph.
- `void AddEdge(int v1, int v2, int cost)`: Adds a directed edge from `v1` to `v2` with the given cost.
    - Throws an exception if `v1` or `v2` is not a valid vertex.
    - Throws an exception if the edge already exists.
- `void AddVertex(int v)`: Adds a new vertex `v` to the graph.
    - Throws an exception if `v` already exists.
- `void RemoveVertex(int v)`: Removes vertex `v` and all associated edges.
    - Throws an exception if `v` does not exist.
- `void RemoveEdge(int v1, int v2)`: Removes the directed edge from `v1` to `v2`.

    - Throws an exception if the edge does not exist.
- `int GetInDegree(int v)`: Returns the in-degree of vertex v.
- `int GetOutDegree(int v)`: Returns the out-degree of vertex v.
- `int FindEdge(int v1, int v2)`: Checks if an edge from v1 to v2 exists, returning its cost or an indicator.
- `void ChangeCost(int v1, int v2, int new_cost)`: Changes the cost of an existing edge.
    - Throws an exception if the edge does not exist.
- `int GetCost(int v1, int v2)`: Returns the cost of the edge from v1 to v2.
    - Throws an exception if the edge does not exist.
- `void ClearGraph()`: Clears all vertices and edges from the graph.
- `pair<int, int> GetEndpoints(long long edge_id)`: Returns the endpoints of the given edge ID.
- `pair<unordered_map<long long, int>::iterator, unordered_map<long long, int>::iterator> IterEdges()`: Returns iterators for all edges.
- `pair<vector<long long>::iterator, vector<long long>::iterator> IterOutboundEdges(int v)`: Returns iterators for all outbound edges of vertex v.
- `pair<vector<long long>::iterator, vector<long long>::iterator> IterInboundEdges(int v)`: Returns iterators for all inbound edges of vertex v.
- `pair<vector<int>::iterator, vector<int>::iterator> IterVertices()`: Returns iterators for all vertices in the graph.

Validation Functions:

- `bool ValidVertex(int v)`: Checks if v is a valid vertex within the graph.
- `bool ValidEdge(int v1, int v2)`: Checks if an edge from v1 to v2 is valid.
- `bool ExistingVertex(int v)`: Checks if v exists in the graph.
- `bool ExistingEdge(int v1, int v2)`: Checks if an edge from v1 to v2 exists in the graph.

I/O Functions:

- `void WriteGraph(DirectedGraph& dgraph, string filename)`: Writes the graph `dgraph` to a file named `filename`.

- `void ReadGraph(DirectedGraph& dgraph, string filename)`: Reads the graph from the file `filename` and populates the `dgraph` object.
- `DirectedGraph GenerateRandomGraph(int n, int m)`: Generates a random directed graph with `n` vertices and `m` edges.

Edge Class Specification

Implementation:

```
class Edge {
   public:
      long long edge_id;
      Edge(int a, int b); // unique identifier for an edge order sensitive
// Cantor's pairing function: f(x, y) = (x + y) * (x + y + 1) / 2 + y
};
```