Documentación de la tarea integradora 3

Martin Cifuentes, Gabriel Delgado y Felipe Barreto

Contenido

[Requerimientos 2](#_Toc87637448)

[Requerimientos no funcionales 2](#_Toc87637449)

[Diagrama de clases 2](#_Toc87637450)

[TAD 2](#_Toc87637451)

[TAD Graph 2](#_Toc87637452)

[TAD SimpleWeightedGraph 2](#_Toc87637453)

[TAD SimpleGraph 3](#_Toc87637454)

[TAD Vertex 6](#_Toc87637455)

[TAD Edge 7](#_Toc87637456)

[JUNIT 7](#_Toc87637457)

[Setup scenarios del modelo 7](#_Toc87637458)

[Diseño de casos de prueba del modelo 7](#_Toc87637459)

[Seup scenarios de las estructuras de datos 7](#_Toc87637460)

[Diseño de casos de prueba del modelo 7](#_Toc87637461)

Documentación

# Requerimientos

# Requerimientos no funcionales

# Diagrama de clases

# TAD

## TAD Graph

|  |
| --- |
| **TAD** Graph |
| Graph = {nodes=ArrayList<Vertex>(); edges=ArrayList<Edge>(); } |
| { inv: nodes.getSize() >= 0 && edges.getSize() >= 0} |
| Operaciones primitivas:  Graph  getNodes null -> ArrayList<Vertex>  setNodes ArrayList<Vertex> -> null  getEdges null -> ArrayList<Edge>  setEdges ArrayList<Edge> -> null  addVertex Vertex -> null  addEdge Edge -> null  BFS Vertex -> null  DFS null -> null  Djikstra (Vertex, Vertex) -> null  FloydWarshall null -> null |

|  |
| --- |
| Graph()  **“Constructora”**  “Constructor de la clase, crea un objeto con un arraylist de vértices y de aristas vacíos”  { pre: TRUE }    { post:  Graph = {nodes.getSize() == 0 && edges.getSize() == 0}} |

|  |
| --- |
| getNodes()  **“Analizadora”**  “Retorna la lista de vértices”  { pre: Graph = {nodes != null, edges != null}}    { post:  nodes} |
| setNodes(ArrayList<Vertex> nodes)  **“Modificadora”**  “Asigna la lista de nodos enviada a través del parámetro al atributo ‘nodes’ ”  { pre: Graph={nodes != null, edges != null}}    { post:  Graph = {this.node = nodes, edges != null}} |

|  |
| --- |
| getEdges()  **“Analizadora”**  “Devuelve la lista de aristas”  { pre: Graph = {nodes != null, edges != null}}    { post:  edges} |

|  |
| --- |
| setEdges(ArrayList<Edge> edges)  **“Modificadora”**  “Asigna la lista de aristas enviada a través del parámetro al atributo ‘edges’ ”  { pre: Graph ={nodes != null, edges != null}}    { post:  Graph = {nodes != null, this.edges = edges}} |

|  |
| --- |
| addVertex(Vertex v)  **“Modificadora”**  “Añade un nuevo vértice a la lista de vértices”  { pre: Graph ={nodes != null, edges != null} && v != null}    { post:  Graph = {nodes.getSize() = nodes.getSize() + 1, edges != null} && nodes.get(nodes.getSize() - 1) == v } |

|  |
| --- |
| addEdge(Edge e)  **“Modificadora”**  “Añade una nueva arista a la lista de aristas ”  { pre: Graph ={nodes != null, edges != null} && e != null}    { post:  Graph = {edges.getSize() = edges.getSize() + 1, nodes != null} && edges.get(edges.getSize() - 1) == e} |

|  |
| --- |
| BFS()  **“Analizadora”**  “Retorna el análisis del Breadth First Search del grafo”  { pre: Graph ={nodes != null, edges != null}}    { post: } |

|  |
| --- |
| DFS()  **“Analizadora”**  “Retorna el análisis del Depth First Search del grafo”  { pre: Graph ={nodes != null, edges != null}}    { post: } |

|  |
| --- |
| Djikstra(Vertex init, Vertex end)  **“Analizadora”**  “Retorna el camino más corto entre el vértice inicial y el final”  { pre: { pre: Graph ={nodes != null, edges != null} && }}    { post: |

|  |
| --- |
| FloydWarshall()  **“Analizadora”**  “Retorna el análisis de los caminos más cortos entre cada par de vértices”  {pre: { pre: Graph ={nodes != null, edges != null}}}    { post: |

## TAD SimpleWeightedGraph

|  |
| --- |
| **TAD** SimpleWeightedGraph |
| SimpleWeightedGraph = {nodes=ArrayList<Vertex>(); edges=ArrayList<Edge>(); } |
| { inv: nodes.getSize() >= 0 && edges.getSize() >= 0} |
| Operaciones primitivas:  SimpleWeightedGraph  getNodes null -> ArrayList<Vertex>  setNodes ArrayList<Vertex> -> null  getEdges null -> ArrayList<Edge>  setEdges ArrayList<Edge> -> null  addVertex Vertex -> null  addEdge Edge -> null  BFS Vertex -> null  DFS null -> null  Djikstra (Vertex, Vertex) -> null  FloydWarshall null -> null |

|  |
| --- |
| SimpleWeightedGraph ()  **“Constructora”**  “Constructor de la clase, crea un objeto con un arraylist de vértices y de aristas vacíos”  { pre: TRUE }    { post:  SimpleWeightedGraph = {nodes.getSize() == 0 && edges.getSize() == 0}} |

|  |
| --- |
| getNodes()  **“Analizadora”**  “Retorna la lista de vértices”  { pre: SimpleWeightedGraph = {nodes != null, edges != null}}    { post:  nodes} |
| setNodes(ArrayList<Vertex> nodes)  **“Modificadora”**  “Asigna la lista de nodos enviada a través del parámetro al atributo ‘nodes’ ”  { pre: SimpleWeightedGraph ={nodes != null, edges != null}}    { post:  SimpleWeightedGraph = {this.node = nodes, edges != null}} |

|  |
| --- |
| getEdges()  **“Analizadora”**  “Devuelve la lista de aristas”  { pre: SimpleWeightedGraph = {nodes != null, edges != null}}    { post:  edges} |

|  |
| --- |
| setEdges(ArrayList<Edge> edges)  **“Modificadora”**  “Asigna la lista de aristas enviada a través del parámetro al atributo ‘edges’ ”  { pre: SimpleWeightedGraph ={nodes != null, edges != null}}    { post:  SimpleWeightedGraph = {nodes != null, this.edges = edges}} |

|  |
| --- |
| addVertex(Vertex v)  **“Modificadora”**  “Añade un nuevo vértice a la lista de vértices”  { pre: SimpleWeightedGraph ={nodes != null, edges != null} && v != null}    { post:  SimpleWeightedGraph = {nodes.getSize() = nodes.getSize() + 1, edges != null} && nodes.get(nodes.getSize() - 1) == v } |

|  |
| --- |
| addEdge(Edge e)  **“Modificadora”**  “Añade una nueva arista a la lista de aristas ”  { pre: SimpleWeightedGraph ={nodes != null, edges != null} && e != null}    { post:  SimpleWeightedGraph = {edges.getSize() = edges.getSize() + 1, nodes != null} && edges.get(edges.getSize() - 1) == e} |

|  |
| --- |
| BFS()  **“Analizadora”**  “Retorna el análisis del Breadth First Search del grafo”  { pre: SimpleWeightedGraph ={nodes != null, edges != null}}    { post: } |

|  |
| --- |
| DFS()  **“Analizadora”**  “Retorna el análisis del Depth First Search del grafo”  { pre: SimpleWeightedGraph ={nodes != null, edges != null}}    { post: } |

|  |
| --- |
| Djikstra(Vertex init, Vertex end)  **“Analizadora”**  “Retorna el camino más corto entre el vértice inicial y el final”  { pre: { pre: SimpleWeightedGraph ={nodes != null, edges != null} && }}    { post: |

|  |
| --- |
| FloydWarshall()  **“Analizadora”**  “Retorna el análisis de los caminos más cortos entre cada par de vértices”  {pre: { pre: SimpleWeightedGraph ={nodes != null, edges != null}}}    { post: |

## TAD SimpleGraph

|  |
| --- |
| **TAD** SimpleGraph |
| SimpleGraph = {nodes=ArrayList<Vertex>(); edges=ArrayList<Edge>(); } |
| { inv: nodes.getSize() >= 0 && edges.getSize() >= 0} |
| Operaciones primitivas:  SimpleGraph  getNodes null -> ArrayList<Vertex>  setNodes ArrayList<Vertex> -> null  getEdges null -> ArrayList<Edge>  setEdges ArrayList<Edge> -> null  addVertex Vertex -> null  addEdge Edge -> null  BFS Vertex -> null  DFS null -> null  Djikstra (Vertex, Vertex) -> null  FloydWarshall null -> null |

|  |
| --- |
| SimpleGraph()  **“Constructora”**  “Constructor de la clase, crea un objeto con un arraylist de vértices y de aristas vacíos”  { pre: TRUE }    { post:  SimpleGraph = {nodes.getSize() == 0 && edges.getSize() == 0}} |

|  |
| --- |
| getNodes()  **“Analizadora”**  “Retorna la lista de vértices”  { pre: SimpleGraph = {nodes != null, edges != null}}    { post:  nodes} |
| setNodes(ArrayList<Vertex> nodes)  **“Modificadora”**  “Asigna la lista de nodos enviada a través del parámetro al atributo ‘nodes’ ”  { pre: SimpleGraph={nodes != null, edges != null}}    { post:  SimpleGraph = {this.node = nodes, edges != null}} |

|  |
| --- |
| getEdges()  **“Analizadora”**  “Devuelve la lista de aristas”  { pre: SimpleGraph = {nodes != null, edges != null}}    { post:  edges} |

|  |
| --- |
| setEdges(ArrayList<Edge> edges)  **“Modificadora”**  “Asigna la lista de aristas enviada a través del parámetro al atributo ‘edges’ ”  { pre: SimpleGraph={nodes != null, edges != null}}    { post:  SimpleGraph = {nodes != null, this.edges = edges}} |

|  |
| --- |
| addVertex(Vertex v)  **“Modificadora”**  “Añade un nuevo vértice a la lista de vértices”  { pre: SimpleGraph={nodes != null, edges != null} && v != null}    { post:  SimpleGraph = {nodes.getSize() = nodes.getSize() + 1, edges != null} && nodes.get(nodes.getSize() - 1) == v } |

|  |
| --- |
| addEdge(Edge e)  **“Modificadora”**  “Añade una nueva arista a la lista de aristas ”  { pre: SimpleGraph={nodes != null, edges != null} && e != null}    { post:  SimpleGraph = {edges.getSize() = edges.getSize() + 1, nodes != null} && edges.get(edges.getSize() - 1) == e} |

|  |
| --- |
| BFS()  **“Analizadora”**  “Retorna el análisis del Breadth First Search del grafo”  { pre: SimpleGraph={nodes != null, edges != null}}    { post: } |

|  |
| --- |
| DFS()  **“Analizadora”**  “Retorna el análisis del Depth First Search del grafo”  { pre: SimpleGraph={nodes != null, edges != null}}    { post: } |

|  |
| --- |
| Djikstra(Vertex init, Vertex end)  **“Analizadora”**  “Retorna el camino más corto entre el vértice inicial y el final”  { pre: { pre: SimpleGraph={nodes != null, edges != null} && }}    { post: |

|  |
| --- |
| FloydWarshall()  **“Analizadora”**  “Retorna el análisis de los caminos más cortos entre cada par de vértices”  {pre: { pre: SimpleGraph={nodes != null, edges != null}}}    { post: |

## TAD Vertex

|  |
| --- |
| **TAD** Vertex |
|  |
| {inv: } |
| Operaciones primitivas: |

## TAD Edge

|  |
| --- |
| **TAD** Edge |
|  |
| {inv: } |
| Operaciones primitivas: |

# JUNIT

## Setup scenarios del modelo

## Diseño de casos de prueba del modelo

## Seup scenarios de las estructuras de datos

## Diseño de casos de prueba del modelo