Método de la ingeniería

Fase 1: Identificación del problema

- **Contexto del problema:** La universidad Icesi, es una universidad privada sin ánimo de lucro ubicada en el suroccidente de Colombia en la ciudad de Cali, departamento del Valle del Cauca. Esta universidad cuenta con un área de aproximada mente 164 mil metros cuadrados que consta de instalaciones deportivas, edificios, zonas verdes, restaurantes y parqueaderos.
  Actualmente, las personas que frecuentan en la universidad son estudiantes, profesores y colaboradores de la misma, algunos de estos estudiantes son microempresarios que venden comida a toda la comunidad universitaria. Para todos los mencionados anteriormente es necesario desplazarse por todo el campus universitario de manera eficiente ya que el aprendizaje activo (el cual es el modelo de enseñanza de la universidad Icesi) les quita mucho tiempo y necesitan movilizarse de un lugar del campus a otro de manera que no les quite tanto tiempo y puedan cumplir sus labores cada uno.

- **Problema:** La universidad universitaria quiere implementar un programa el cual permita a sus usuarios ver la ruta más cercana que hay de un lugar a otro en el campus, lo cual disminuiría el tiempo para llegar a su destino.

- **Requerimientos Funcionales:**

  Requerimiento funcional 1: Encontrar el camino más corto desde un edificio a otro.
  Entradas: Edificio de llegada y edificio de salida.
  Salidas: Camino más corto entre los dos edificios.

  Requerimiento funcional 2: Conocer cuál es el camino más corto, el cual conecta todos los edificios de la universidad.
  Entradas: Edificio inicial.
  Salidas: Camino más corto que conecta todos los edificios de la universidad.

  Requerimiento funcional 3: Visualizar el camino más corto entre dos edificios.
  Entradas: Ninguna.
  Salidas: Visualización del camino más corto entre dos edificios.
  Requerimiento funcional 4: Visualizar el camino más corto que conecta a todos los edificios.
  Entradas: Ninguna.

Salidas: Visualización del camino más corto entre dos edificios.

- **Requerimientos No Funcionales:**

  Requerimiento no funcional 1: La complejidad del algoritmo Dijkstra debe de ser igual al número de vértices multiplicado por número de aristas del grafo, osea O(n^2).

Objetivo: Probar que el método instert(T objeto) funcione correctamente para diferentes casos de prueba				
Clase: OurGraph		**Método: Insert**		
Caso #	**Descripción:**	**Escenario**	**Valores de entrada**	**Resultado**
1 (Caso estandar)	Se crea un grafo vacio, en el cual sus vértices y aristas contienen enteros	scenarioOne()	Se entregan dos vértices, uno con valor 1 y el otro con valor 2. El peso de la arista que va a ser 2, y lo que contiene la arista que será 3	El algoritmo insert() debe de devolver el valor true, indicando que se insertó todo correctamente
2 (Caso interesante)	Se crea un grafo vacio, en el cual sus vértices y aristas contienen Strings	scenarioTwo()	Se entregan dos vértices, uno con valor “Juan” y el otro con valor “Sebastian”. El peso de la arista que va a ser 2, y lo que contiene la arista que será “Puerta”	El algoritmo insert() debe de devolver el valor true, indicando que se insertó todo correctamente
3 (Caso limite)	Se crea un grafo con vértices que representan edificios, y sus aristas son caminos	scenarioThree()	Se entregan dos vértices uno que representan un objeto tipo edificio y el cada uno. El peso de la	El algoritmo insert() debe de devolver el valor true, indicando que se insertó todo correctamente

			arista que va a ser 2, y lo que contiene la arista que será un objeto de tipo camino	

Objetivo: Probar que el método Search(T objeto) funcione correctamente para diferentes casos de prueba				
Clase: OurGraph		**Método: Search**		
Caso #	**Descripción:**	**Escenario**	**Valores de entrada**	**Resultado**
1 (Caso estandar)	Se crea un grafo vacio, en el cual sus vértices y aristas contienen enteros	scenarioOne()	Se entrega un vértice a buscar con valor 1	El algoritmo Search() debe de devolver el valor 1, indicando que se encontró el vértice correctamente
2 (Caso interesante)	Se crea un grafo vacio, en el cual sus vértices y aristas contienen Strings	scenarioTwo()	Se entrega un vértice a buscar de tipo cadena con valor “Juan”	El algoritmo Search() debe de devolver el valor “Juan”, indicando que se encontró el vertice correctamente
3 (Caso limite)	Se crea un grafo con vértices que representan edificios, y sus aristas son caminos	scenarioThree()	Se entregan un vértice a buscar de tipo edificio	El algoritmo Search() debe de devolver el valor edificio indicando que se encontró el vértice correctamente

Objetivo: Probar que el método Delete(T objeto) funcione correctamente para diferentes casos de prueba				
Clase: OurGraph		**Método: Delete**		
Caso #	**Descripción:**	**Escenario**	**Valores de entrada**	**Resultado**
1 (Caso estandar)	Se crea un grafo vacio, en el cual sus vértices y aristas contienen enteros	scenarioOne()	Se entrega un vértice a borrar con valor 1	El algoritmo Delete() debe de devolver el valor true, indicando que se borró el vértice correctamente
2 (Caso interesante)	Se crea un grafo vacio, en el cual sus vértices y aristas contienen Strings	scenarioTwo()	Se entrega un vértice a borrar de tipo cadena con valor “Juan”	El algoritmo Delete() debe de devolver el valor true, indicando que se borró el vertice correctamente
3 (Caso limite)	Se crea un grafo con vértices que representan edificios, y sus aristas son caminos	scenarioThree()	Se entregan un vértice a borrar de tipo edificio con valor	El algoritmo Delete() debe de devolver el edificio, indicando que se borró el vértice correctamente

Objetivo: Probar el correcto funcionamiento del método de búsqueda de amplitud BFS				
Clase: OurGraph		**Método: BFS(Vértice)**		
Caso #	**Descripción:**	**Escenario**	**Valores de entrada**	**Resultado**
1	Se crea un grafo, el cual sus vértices contendrán edificios y sus aristas contendrán caminos	scenarioFour()	El método recibirá un vértice, el cual será la raíz del árbol	Se espera que el resultado sea un árbol cuya raíz sea el vértice pasado como parámetro y cuyo recorrido sea en amplitud

Objetivo: Probar el correcto funcionamiento del método de búsqueda de amplitud DFS				
Clase: OurGraph		**Método: DFS(Vértice)**		
Caso #	**Descripción:**	**Escenario**	**Valores de entrada**	**Resultado**
1	Se crea un grafo, el cual sus vértices contendrán edificios y sus aristas contendrán caminos	scenarioFour()	El método recibirá un vértice, el cual será la raíz del árbol	Se espera que el resultado sea un árbol cuya raíz sea el vértice pasado como parámetro y cuyo recorrido sea en profundidad

Objetivo: Probar el correcto funcionamiento del método Dijkstra				
Clase: OurGraph		**Método: Dijkstra (Vértice)**		
Caso #	**Descripción:**	**Escenario**	**Valores de entrada**	**Resultado**
1	Se crea un grafo, el cual sus vértices contendrán edificios y sus aristas contendrán caminos	scenarioFour()	El método recibirá un vértice, el cual contendrá un edificio	Se espera que el método devuelva un arreglo de enteros con las distancias más cortas del vértice pasado como parámetro hacia todos los demás

Objetivo: Probar el correcto funcionamiento del método Floyd-Warshall				
Clase: OurGraph		**Método: Floyd-Warshall (Matriz de adyacencia)**		
Caso #	**Descripción:**	**Escenario**	**Valores de entrada**	**Resultado**
1	Se crea un grafo, el cual sus vértices contendrán edificios y sus aristas contendrán caminos	scenarioFour()	El método recibirá una matriz de adyacencia, la cual representa todos los caminos que hay de la universidad para llegar a cualquier edificio	Se espera que el método devuelva una matriz de adyacencia con los caminos más cortos que hay de un vértice a los demás.

Objetivo: Probar el correcto funcionamiento del método que encuentra el árbol de mínima expansión Prim				
Clase: OurGraph		**Método: Prim (Vértice)**		
Caso #	**Descripción:**	**Escenario**	**Valores de entrada**	**Resultado**
1	Se crea un grafo, el cual sus vértices contendrán edificios y sus aristas contendrán caminos	scenarioFour()	El método recibirá un vértice como parámetro, el cual será la raíz del árbol	Se espera que el resultado sea un árbol cuya raíz sea el vértice pasado como parámetro y que contenga todos los vértices

Objetivo: Probar el correcto funcionamiento del método que encuentra el árbol de mínima expansión Kruskal				
Clase: OurGraph		**Método: Kruskal (Vértice)**		
Caso #	**Descripción:**	**Escenario**	**Valores de entrada**	**Resultado**
1	Se crea un grafo, el cual sus vértices contendrán edificios y sus aristas contendrán caminos	scenarioFour()	El método recibirá un vértice como parámetro, el cual será la raíz del árbol	Se espera que el resultado sea un árbol cuya raíz sea el vértice pasado como parámetro y que contenga todos los vértices

Diagrama de clases

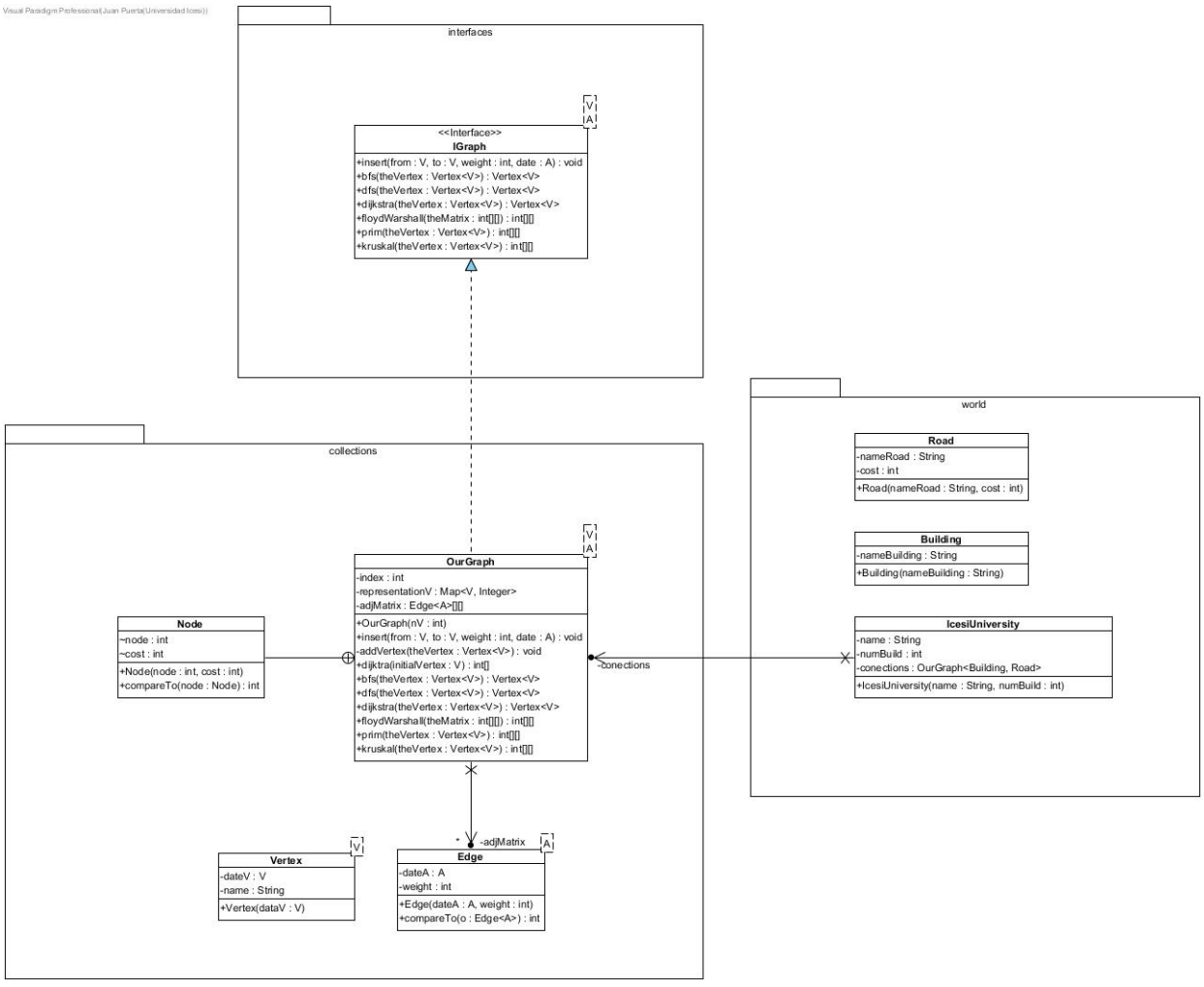
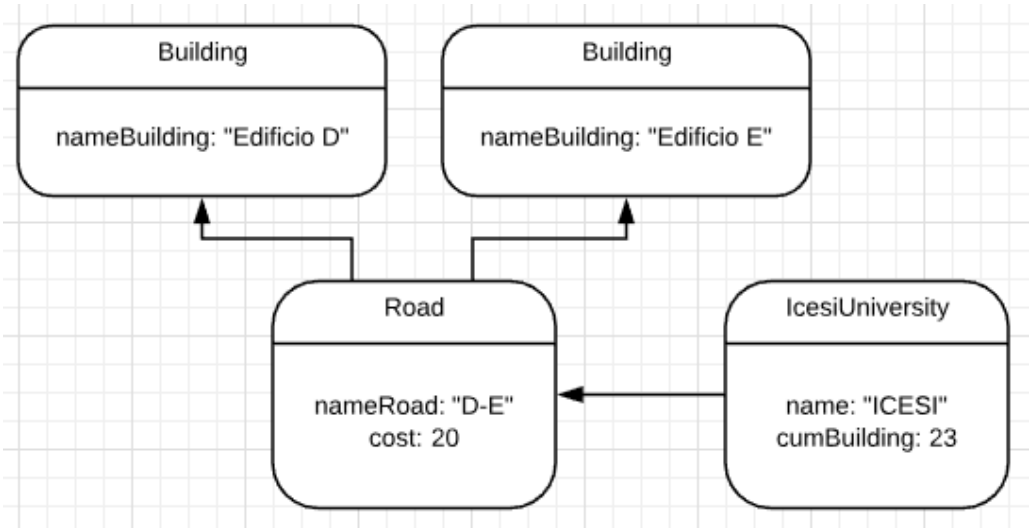


Diagrama de objetos



Fase 7: Implementación

La implementación de la solución se encuentra en el siguiente repositorio de github**:**

https://github.com/Juan-Puerta/ProyectoGrafo