# 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 en el cual sus vértices contienen los enteros {1,2,3,4,5} respectivamente, y sus aristas con pesos enteros {3,3,2,6,4} | stageOne() | Se entregan dos vértices, uno con valor 4 y el otro con valor 6. El peso de la arista que va a ser 2, y el dato que contiene la arista que  será 60 | El algoritmo insert() debe de devolver el valor 6, indicando que se insertó todo correctamente |
| **2**  (Caso interesante) | Se crea un grafo en el cual sus vértices contienen los strings {A,B,C,D,E} respectivamente, y sus aristas con pesos enteros {3,3,2,6,4} | stageTwo() | Se entregan dos vértices, uno con valor “D” y el otro con valor “F”. El peso de la arista que va a ser 2, y el dato que contiene la arista que  será “DF” | El algoritmo insert() debe de devolver el valor “F”, indicando que se insertó todo correctamente |
| **3**  (Caso limite) | Se crea un grafo universidad con vértices que representan los edificios {A,B,C,D,E}, y sus aristas son caminos representados como {AB,BC,BD,DC,CE} | stageThree() | Se entregan dos vértices uno que representan un objeto tipo edificio con valor “F” y el otro representa el vértice con el cual será conectado con valor “D”. Y un camino con valor “DF” con peso 2 | El algoritmo insert() debe de devolver el valor “F”, indicando que se insertó todo  correctamente |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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 en el cual sus vértices contienen los enteros {1,2,3,4,5} respectivamente, y sus aristas con pesos enteros {3,3,2,6,4} | stageOne() | Se entregan 3 vértices de tipo entero a buscar con valores 1,3,5 respectivamente | El algoritmo Search() debe de devolver los valores 1, 3,5 respectivamente indicando que se encontraron los vértices  correctamente |
| **2**  (Caso interesante) | Se crea un grafo en el cual sus vértices contienen los strings {A,B,C,D,E} respectivamente, y sus aristas con pesos enteros {3,3,2,6,4} | stageTwo() | Se entregan 3 vértices de tipo String a buscar con valores “B”,”C”, ”D” respectivamente | El algoritmo Search() debe de devolver los valores “B”,”C”,”D” respectivamente indicando que se encontraron los  vértices correctamente |
| **3**  (Caso limite) | Se crea un grafo universidad con vértices que representan los edificios {A,B,C,D,E}, y sus aristas son caminos representados como {AB,BC,BD,DC,CE} | stageThree() | Se entregan dos vertices vértice a buscar de tipo edificio con valores “A” y “E” respectivamente | El algoritmo Search() debe de devolver los valores “A” Y “E” indicando que se encontraron los  vértices correctamente |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Objetivo:** Probar el correcto funcionamiento del método Dijkstra | | | | |
| **Clase: OurGraph** | | **Método: Dijkstra (Vértice)** | | |
| **Caso #** | **Descripción:** | **Escenario** | **Valores de**  **entrada** | **Resultado** |
| 3 | Se crea un grafo universidad con vértices que representan los edificios {A,B,C,D,E}, y sus aristas son caminos representados como {AB,BC,BD,DC,CE} | stageThree() | El método recibe dos vértices “A” y “C” que corresponden a edificios | Se espera  que el método  devuelva una  pila de  Strings con  las distancias más cortas  del vértice  pasado como  parámetro  hacia todos  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** |
| 3 | Se crea un grafo universidad con vértices que representan los edificios {A,B,C,D,E}, y sus aristas son caminos representados como {AB,BC,BD,DC,CE} | stageThree() | El método recibe una pila con los vértices a buscar el camino mas corto | Se espera  que el  resultado sea  una pila con los pesos de las aristas que corresponden al camino más corto entre dos vértices dados |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Objetivo:** Probar el correcto funcionamiento del método de búsqueda de  amplitud BFS | | | | |
| **Clase: OurGraph** | | **Método: BFS(Vertex)** | | |
| **Caso #** | **Descripción:** | **Escenario** | **Valores de entrada** | **Resultado** |
| **1**  (Caso estandar) | Se crea un grafo en el cual sus vértices contienen los enteros {1,2,3,4,5} respectivamente, y sus aristas con pesos enteros {3,3,2,6,4} | stageOne() | Se el vértice con valor entero 1 a buscar el camino por amplitud | Devuelve un arreglo de enteros con los valores {1,2,3,4,5} correspondientes al orden de la búsqueda por profundidad |
| **2**  (Caso interesante) | Se crea un grafo en el cual sus vértices contienen los strings {A,B,C,D,E} respectivamente, y sus aristas con pesos enteros {3,3,2,6,4} | stageTwo() | Se el vértice de tipo String con valor “A” a buscar el camino por amplitud | Devuelve un arreglo de Strings con los valores {“A”,”B”,”D”,”C”,”E”} correspondientes al orden de la búsqueda por profundidad |
| **3**  (Caso limite) | Se crea un grafo universidad con vértices que representan los edificios {A,B,C,D,E}, y sus aristas son caminos representados como {AB,BC,BD,DC,CE} | stageThree() | Se el vértice de tipo Edificio con valor “A” a buscar el camino por amplitud | Devuelve un arreglo de Strings con los valores {“A”,”B”,”C”,”D”,”E”} correspondientes al orden de la búsqueda por profundidad |

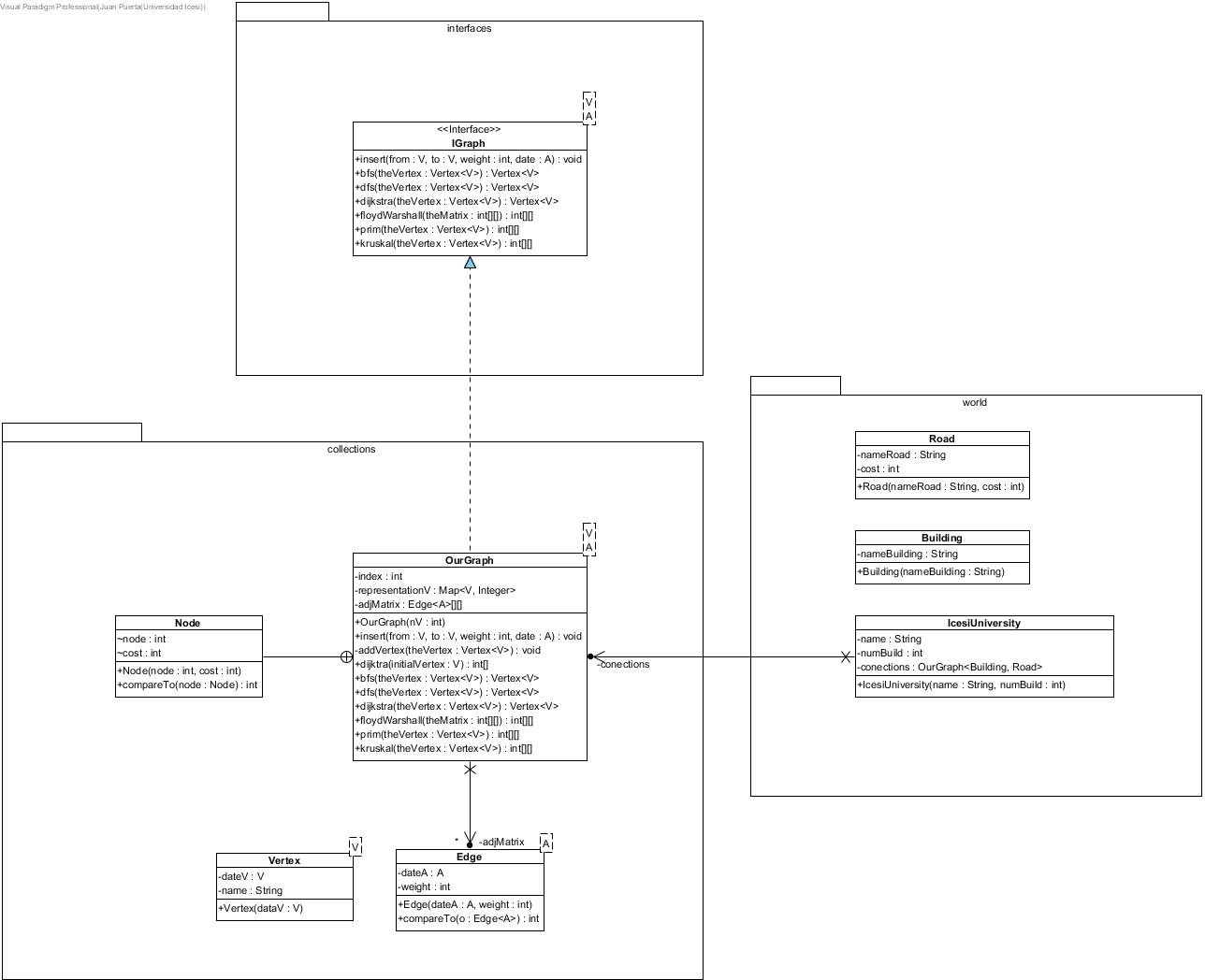
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Objetivo:** Probar el correcto funcionamiento del método de búsqueda por profundidad BFS | | | | |
| **Clase: OurGraph** | | **Método: DFS(Vertex)** | | |
| **Caso #** | **Descripción:** | **Escenario** | **Valores de entrada** | **Resultado** |
| **1**  (Caso estandar) | Se crea un grafo en el cual sus vértices contienen los enteros {1,2,3,4,5} respectivamente, y sus aristas con pesos enteros {3,3,2,6,4} | stageOne() | Se el vértice con valor entero 1 a buscar el camino por amplitud | Devuelve un arreglo de enteros con los valores {1,2,4,3,5} correspondientes al orden de la búsqueda por profundidad |
| **2**  (Caso interesante) | Se crea un grafo en el cual sus vértices contienen los strings {A,B,C,D,E} respectivamente, y sus aristas con pesos enteros {3,3,2,6,4} | stageTwo() | Se el vértice de tipo String con valor “A” a buscar el camino por amplitud | Devuelve un arreglo de Strings con los valores {“A”,”B”,”D”,”C”,”E”} correspondientes al orden de la búsqueda por profundidad |
| **3**  (Caso limite) | Se crea un grafo universidad con vértices que representan los edificios {A,B,C,D,E}, y sus aristas son caminos representados como {AB,BC,BD,DC,CE} | stageThree() | Se el vértice de tipo Edificio con valor “A” a buscar el camino por amplitud | Devuelve un arreglo de Strings con los valores {“A”,”B”,”D”,”C”,”E”} correspondientes al orden de la búsqueda por profundidad |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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 | scenarioThree() | 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. |

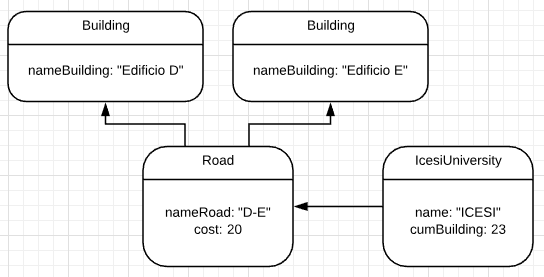
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Objetivo:** Probar el correcto funcionamiento del método que encuentra el árbol de mínima expansión Kruskal | | | | |
| **Clase: OurGraph** | | **Método: Kruskal (Vértice)** | | |
| **Caso #** | **Descripción:** | **Escenario** | **Valores de**  **entrada** | **Resultado** |
| 1 | Se crea un  grafo, el cual  sus vértices  contendrán  edificios y sus  aristas  contendrán  caminos | scenarioThree() | 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 Objeto**

**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