**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 cual 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 No Funcionales:**

**Requerimiento NO funcional 1:** Visualizar el camino más corto entre dos edificios.

**Entradas:** Ninguna.

**Salidas:** Visualización del camino más corto entre dos edificios.

**Requerimiento NO funcional 2:** Visualizar el camino más corto que conecta a todos los edificios.

**Entradas:** Ninguna.

**Salidas:** Visualización del camino más corto entre dos edificios.

***Fase 2: Recopilación de información***

***Definiciones:***

***Fase 3: Búsqueda de soluciones creativas***

***Fase 4: Transición de la formulación de ideas a los diseños preliminares***

* **Descarte de ideas no factibles**

Se descartaron las siguientes opciones de la búsqueda de soluciones creativas debido a:

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

***Fase 5: Evaluación y selección de la mejor solución.***

Actualmente tenemos tres alternativas de solución, las cuales resuelven nuestro problema de buscar raíces, como sólo necesitamos dos alternativas, definiremos una serie de criterios que nos ayudaran a escoger las dos mejores en base al número de puntos.

**Criterios:**

* **Criterio A:**
* **Criterio B:**
* **Criterio C:**
* **Criterio D:**
* **Criterio E:**

**Evaluación:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **#**  **Alternativa** | **Criterio A** | **Criterio B** | **Criterio C** | **Criterio D** | **Criterio E** | **Total Puntos** |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

Con base en los resultados obtenidos vamos a descartar la alternativa 7 e implementar las alternativas y para la solución de nuestro problema.

***Paso 6: Preparación de informe y especificaciones.***

Con lo que con concluimos con este respectivo a este estudio y su respectivo análisis, buscando los mejores algoritmos para ser implementados en el programa especificado y así satisfacer los requerimientos antes planeados, se han seleccionado el Método de, esto por varios criterios anteriormente especificados, es por lo anterior que ahora en adelante podemos comenzar a implementar todo lo aquí expresado, siguiente las pautas necesarias y coherentes para un proceso óptimo.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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 arista que va a ser 2, y lo que contiene la arista que será un objeto de tipo camino | El algoritmo insert() debe de devolver el valor true, 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 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 con valor ¿? | El algoritmo Search() debe de devolver el valor ¿?, 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 valor ¿?, 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 |

* **Diseños preliminares**

Para los diseños preliminares hemos decidido hacer el pseudocódigo de los 3 algoritmos más importantes para así podernos dar una idea de cómo implementarlos en código. Además, aprovecharemos el pseudocódigo para sacar la complejidad espacial y temporal de cada algoritmo.

**Método 1:**

|  |  |  |  |
| --- | --- | --- | --- |
| **#** |  |  |  |
| 1 |  |  |  |
| 2 |  |  |  |
| 3 |  |  |  |
| 4 |  |  |  |
| 5 |  |  |  |
| 6 |  |  |  |
| 7 |  |  |  |
| 8 |  |  |  |
| 9 |  |  |  |
| 10 |  |  |  |
| 11 |  |  |  |
| 12 |  |  |  |
| 13 |  |  |  |
| 14 |  |  |  |
| 15 |  |  |  |
| 16 |  |  |  |
| 17 |  |  |  |
| 18 |  |  |  |
| 19 |  |  |  |
| 20 |  |  |  |
| 21 |  |  |  |
| 22 |  |  |  |
| 23 |  |  |  |
| 24 |  |  |  |
| 25 |  |  |  |
| 26 |  |  |  |
| 27 |  |  |  |
| 28 |  |  |  |
| 29 |  |  |  |
| 30 |  |  |  |
| 31 |  |  |  |
| 32 |  |  |  |
| 33 |  |  |  |
| 34 |  |  |  |
| 35 |  |  |  |
| 36 |  |  |  |
| 37 |  |  |  |
| 38 |  |  |  |

**Método 2:**

|  |  |  |  |
| --- | --- | --- | --- |
| **#** |  |  |  |
| 1 |  |  |  |
| 2 |  |  |  |
| 3 |  |  |  |
| 4 |  |  |  |
| 5 |  |  |  |
| 6 |  |  |  |

**Método 3:**

|  |  |  |  |
| --- | --- | --- | --- |
| **#** |  |  |  |
| 1 |  |  |  |
| 2 |  |  |  |
| 3 |  |  |  |
| 4 |  |  |  |
| 5 |  |  |  |
| 6 |  |  |  |
| 7 |  |  |  |
| 8 |  |  |  |

**Análisis de complejidad espacial**

** Método 1:**

**T(n) = 1**

**T(n) = 𝑪**

Por medio del T(n) podemos concluir que la notación asintótica para la complejidad espacial del método 1 es: 𝑂(1).

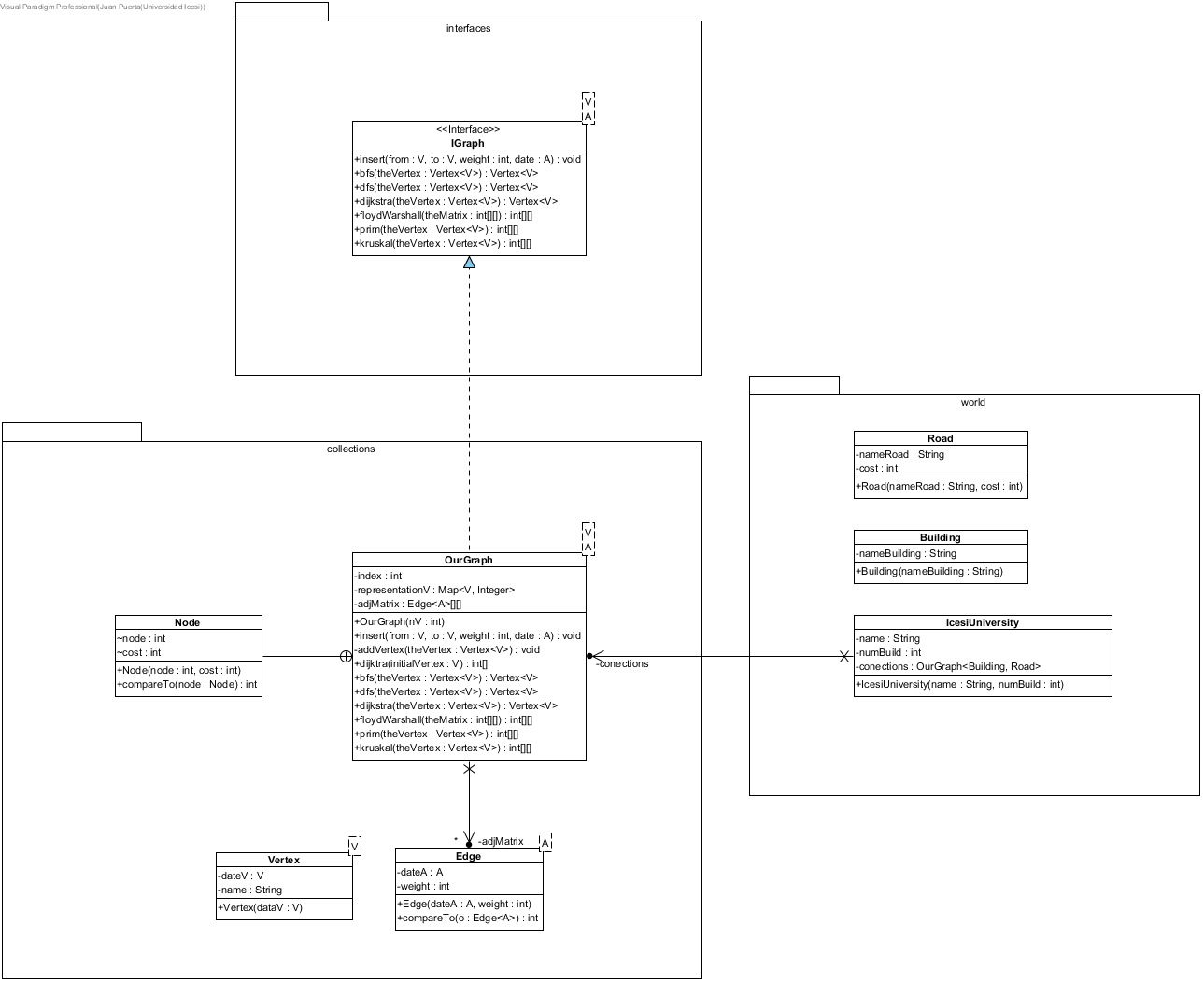
** Análisis de complejidad temporal**

** Método 1:**

**T(n) = 𝟐𝒏 + 𝟔   
T(n) = 𝑨𝒏 + 𝑩**

Por medio del T(n) podemos concluir que la notación asintótica para la complejidad temporal del método 1 es: 𝑂(𝑛).

* **Diagrama de clases**

****

**Fase 7: Implementación**

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

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