Laura Rubio A00346353

Douglas López A00347533

David Obando A00348505

**Problemática:**

Con el paso del tiempo la universidad Icesi se está expandiendo con el propósito de recibir más estudiantes, lo que lleva a la comunidad estudiantil, de personal y aspirantes a recurrir a herramientas que le permitan facilitar la movilidad dentro de la institución, en ocasiones se necesita saber ¿dónde está ubicado un edificio X?, es importante tener en cuenta que es posible contar con usuarios que tengan problemas de visión, por lo tanto se necesitan de ayudas visuales para los espacios y edificios de la universidad. Además, para el usuario también es importante conocer cuál es la ruta más eficiente hacia su siguiente clase desde un edificio X a uno Y, así mismo saber la distancia entre estos y el tiempo que le toma recorrer esa distancia. Analizando las necesidades del personal de limpieza se encontró que estos requieren encontrar un camino eficiente que les permita recorrer todos los edificios de la universidad una única vez, para de este modo mantener las zonas en buen estado, otra funcionalidad adicional es al momento de ingresar a la institución en vehículo o moto, el usuario requiere que dada la entrada en la que se encuentra cual es el área de parqueo más próxima a su edifico de destino y que ruta puede tomar para llegar a esta. También se presenta la siguiente dificultad a la hora de almuerzo, pues el principal interés del usuario es encontrar el sitio de comidas más próximo a su ubicación actual. Finalmente, otro aspecto importante es conocer qué sitios de interés u otros edificios se encuentren en cerca de la posición donde se encuentra el usuario.

**Requerimientos:**

**Nombre:** Buscar edificio.

* Permite buscar y encontrar el edificio, así mismo edificios o espacios cercanos al edificio buscado.

**Entrada:** Edificio a buscar por el usuario.

**Salida:** Edificio encontrado con su respectiva información o mensaje de no encontrado.

**Nombre:** Distancia del destino.

* Permite hallar la distancia desde el edificio en que se encuentre el usuario hasta el edificio hacia dónde se dirige

**Entrada:** Edificio de actual y edificio de destino.

**Salida:** Número que establece la distancia entre dos edificios.

**Nombre:** Tiempo de llegada.

* Permite conocer el tiempo estimado de recorrido de la distancia establecida entre el sitio actual y el de destino

**Entrada:** Distancia entre el edifico actual y el edificio de destino

**Salida:** Tiempo aproximado en cruzar la distancia entre dos edificios.

**Nombre:** Camino más corto entre dos edificios.

* Permite conocer la distancia más corta entre dos edificios dados

**Entrada:** Edificio actual y edificio de destino

**Salida:** Camino que representa la menor distancia.

**Nombre:** Camino eficiente de la universidad

* Permite hallar una ruta eficiente en términos de tiempo y distancia que me ayude a recorrer la universidad visitando cada edificio una única vez.

**Entrada:** Ubicación actual.

**Salida:** Ruta eficiente para recorrer la universidad.

**Nombre:** Edificios adyacentes a ubicación actual

* Permite saber qué edificios o espacios de la universidad se encuentran cerca de mi ubicación actual.

**Entrada:** Ubicación actual.

**Salida:** Edificios o espacios cercanos a la posición del usuario.

**Nombre:** Ruta de parqueadero.

* Muestra al usuario la ruta más corta para llegar desde la entrada en la que se encuentra hasta un área de parqueo cercana al edificio al que desea acceder el usuario.

**Entrada:** Número de entrada de la universidad y edificio de destino

**Salida:** Ruta hacia la zona de parqueo.

**2. Recopilación de información.**

La universidad Icesi cuenta con diversos edificios; en la actualidad, algunos se distribuyen alfabéticamente de A - N y otros con un nombre distinto, contando con un total alrededor de 18 edificios. También la universidad cuenta con varios centros lúdicos, zonas libres y parqueaderos para que las personas puedan usarlos dependiendo de su interés.

Los edificios contienen una cierta cantidad de pisos, baños, auditorios, entre otros atributos, pero estos detalles no es de nuestra incumbencia, ya que solamente nos interesa la ubicación o posición del edificio.

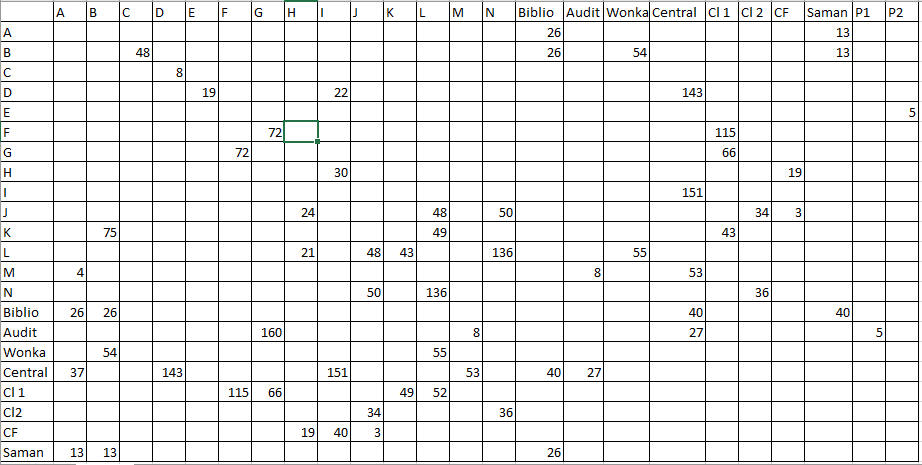
Por un lado, es necesario conocer el listado de edificios que componen la universidad, el cual son los siguientes:

* Edificio A
* Edificio B
* Edificio C
* Edificio D
* Edificio E
* Edificio F
* Edificio G
* Edificio H
* Edificio I (Bienestar)
* Edificio J
* Edificio K
* Edificio L
* Edificio M
* Edificio N

Por otro lado, como se dijo anteriormente, la universidad también está compuesta por centros lúdicos, parqueaderos, y zona libres denominados:

* Coliseo 1
* Coliseo 2
* Samán
* Parqueadero 1
* Parqueadero 2

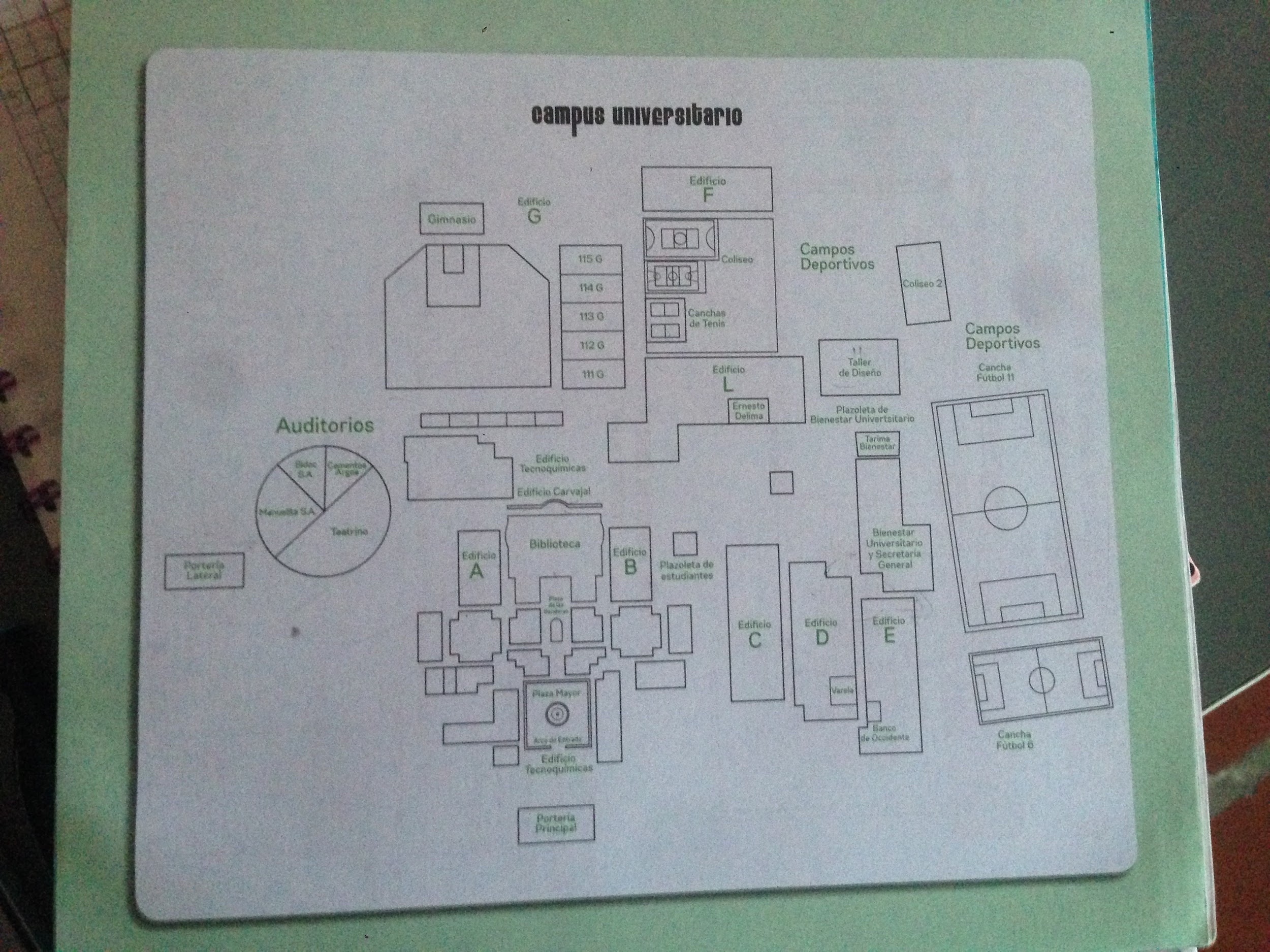
Por último, es de suma importancia saber las diferentes rutas y distancias que hay entre la conexión de los sitios o lugares mencionados anteriormente. Por consiguiente, es necesario ubicar las rutas que nos conectan los edificios, una breve descripción que se puede encontrar es la siguiente:



*tabla 1*

En la tabla 1, podemos ver qué edificios se conectan entre sí y un valor numérico, dado en metros, que es la distancia que hay entre ellos. Por ejemplo, el edificio A me conecta con el edificio Biblioteca con una distancia aproximada de 26 metros. Así sucesivamente se puede analizar la tabla para comprender las conexiones entre todos los edificios de la universidad Icesi. Cabe aclarar que cuando existe una relación entre edificios, a esto se le conoce como una ruta para la persona o una manera de poder llegar a un edificio, en otras palabras si una persona está ubicada en un edificio X y tiene una conexión con otro edificio Y, quiere decir que existe un camino para poder llegar al edificio Y.

Otra manera de poder comprender la infraestructura de la universidad Icesi es mediante el mapa que nos ofrece la universidad, sin embargo este no abarca con la totalidad de edificios pero si nos asemeja una realidad bastante descriptiva de lo que sería la composición de las rutas y edificios.



*gráfico 1*

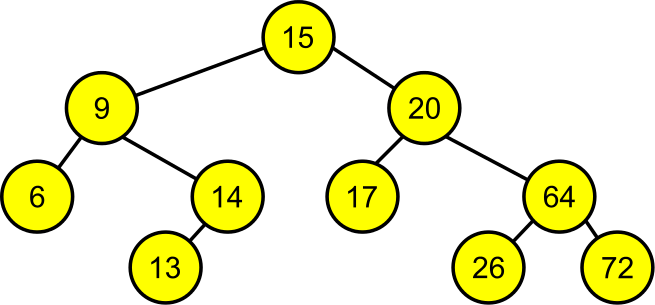
Si relacionamos el gráfico 1 con la tabla 1, podemos entender fácilmente las rutas que se plantean para los vínculos que contiene cada edificio con respecto a otro. También se puede analizar los diferentes coliseos mencionados anteriormente, lo único que no se logra analizar en el gráfico son los parqueaderos y unos cuantos edificio que no son nombrados explícitamente.

Ahora bien, durante un arduo proceso teórico acerca de las diferentes estructuras de datos e implementaciones de estas, podemos llegar a categorizarlas para así comprender una solución bastante benemérita a nuestro problema, así que, en primer lugar consideraremos unas cuantas estructuras que se han estudiado profundamente y podrían ser de ayuda para la resolución de nuestro problema, esto no quiere decir que van a ser implementadas o van ser las estructuras fundamentales para la solución, sino que se tendrán en cuenta para después analizarlas en un sentido específico frente a diferentes conveniencias presentadas para la eficiencia de nuestro aplicativo. A continuación se representará las características de algunas estructuras de datos para así poder comprender la función de estas; además de que nos contextualizará frente a las expectativas del problema.

Empezamos con unas estructuras que suenan llamativas como lo son los árboles binarios, básicamente la función es almacenar datos de forma ordenada para suplir una búsqueda, pero cada árbol tiene sus propias características que lo diferencian de uno a otro:

1. **Árbol Binario de Búsqueda (ABB):**

Árbol en el cual cada uno de sus nodos puede tener no más de 2 hijos. En el cual se cumple con la jerarquización de que todos los nodos a la izquierda de la raíz son menores que ella y todos los nodos a la derecha de la raíz son mayores que ella. Por lo tanto, el subárbol izquierdo de cualquier nodo (si no está vacío) contiene valores menores que el que contiene dicho nodo, y el subárbol derecho (si no está vacío) contiene valores mayores. Así, es una buena opción para garantizar una búsqueda eficiente de los jugadores guardados.



2. **Árbol Rojo y negro**

Es un Árbol Binario de búsqueda que debe cumplir con las siguientes propiedades:

1 - Cada nodo es rojo o negro.

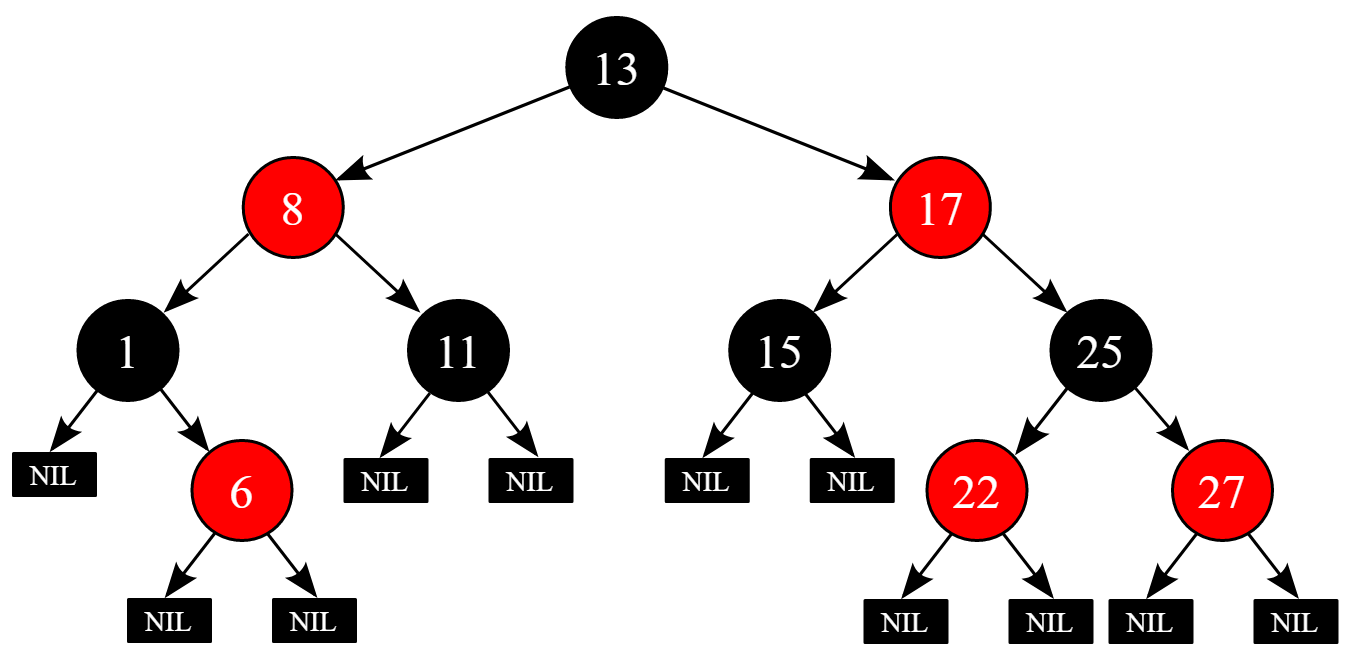
2 - La raíz es negra.

3 - Toda hoja (NIL) es negra.

4 - Si un nodo es rojo, sus dos hijos son negros. (no puede haber dos rojos consecutivos en un camino)

5 - Todocamino desde un nodo a cualquier hoja descendente contiene el mismo número de nodos negros.

Así pues, de acuerdo a la propiedad 5, todo árbol rojinegro con n nodos internos tiene una altura menor o igual que 2 log(n+1). Por lo tanto, las operaciones de búsqueda pueden implementarse en tiempo O(log n) para árboles rojinegros con n nodos, así como sus funciones para insertar y eliminar.



Además para ahorrar espacio en memoria, en este árbol se crea un nodo auxiliar T.nil de color negro, al cual van apuntar todos los nodos que apuntan a un nodo NIL. Por lo tanto, se convierte en una de nuestras alternativas para facilitar la organización de jugadores y así mismo su búsqueda.

**3. Arbol AVL:**

Es un tipo especial de árbol binario.Es un árbol de búsqueda binaria autobalanceable, su objetivo es intentar mantener su altura, o el número de niveles de nodos bajo la raíz, tan pequeños como sea posible en todo momento, automáticamente, esto con la intención de tener una complejidad O(log n) en las operaciones de búsqueda, inserción y eliminación.

Su propiedad consiste en que para todos los nodos, la altura de la rama izquierda no difiere en más de una unidad de la altura de la rama derecha o viceversa, para esto cada nodo contiene el dato que controla el factor de equilibrio.

El factor de equilibrio es la diferencia entre las alturas del árbol derecho y el izquierdo, por definición, para un árbol AVL, este valor debe ser -1,0 ó 1.

Si el factor de equilibrio de un nodo es:

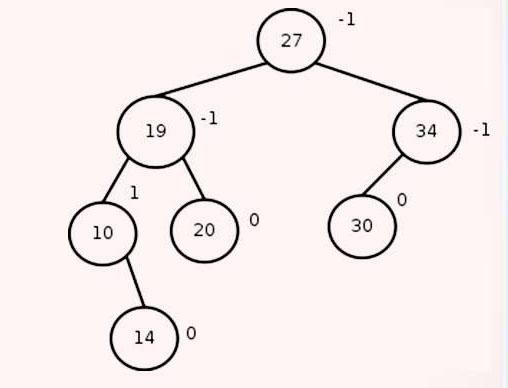
0 : el nodo está equilibrado y sus subárboles tienen exactamente la misma altura.

1 : el nodo está equilibrado y su subárbol derecho es un nivel más alto.

-1: el nodo está equilibrado y su subárbol izquierdo es un nivel más alto.

2 o mayor: es necesario equilibrar.

Para lograr el reequilibrado en un árbol AVL es necesario utilizar las operaciones de rotaciones, existen dos casos: rotación simple o rotación doble.



Por otro lado, nos encontramos con unas estructuras bastante peculiares, ya que nos ofrece una forma exclusiva de almacenar datos o un modelo a seguir para poder llevar un orden a la hora de extraer o insertar un dato, el cual son las siguientes:

**4. Cola.**

La cola es una estructura de dato, que permite el almacenamiento y recuperación de datos; caracterizada por usar el método FIFO el cual consiste en sacar el elemento que primero ha ingresado a la estructura, en otras palabras, si tenemos una serie de datos que se han ingresado de la siguiente forma:

**Q** = { e1, e2, e3, e4…, en} donde *e* es igual al tipo de dato que estamos almacenando. Por ende, solo nos permitirá obtener el elemento *e1* de la cola, ya que este ha sido el primer elemento insertado o agregado.

**5. Pila:**

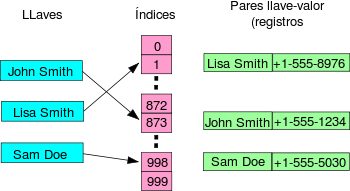
La pila es una estructura de dato muy similar a la Cola, ya que nos permite almacenar y recuperar datos, pero en este caso la Pila nos ofrece el sistema LIFO, el cual consiste en poder recuperar o extraer el dato que se ha ingresado de último, en otras palabras, si tenemos una serie de datos que se han ingresado de la siguiente manera:

**S** = { e1, e2, e3, …., en} donde *e* es igual al tipo de dato que estamos almacenando.

Como se mencionó anteriormente, solo nos permite tomar o extraer el último elemento que se ha ingresado a la pila, entonces solo nos dejará obtener el elemento *en* de la pila **S.**

Una estructura muy eficiente para la búsqueda de datos es conocida como la tabla hash y una breve descripción

**6. Tabla Hash:** Es una estructura de datos especializada para almacenar grandes cantidades de datos, consiste en un nodo que contiene un índice y una llave, sus operaciones son bastante eficientes en especial su búsqueda, ya que con el índice único para cada elemento se puede acceder fácilmente a él.



**7. Grafo**

Por último cabe añadir una estructura de datos llamada grafo, el cual consiste en una serie de vértices o nodos que están conectados entre sí mediante aristas para mostrar una relación entre elementos de un conjunto.

Existen también diferentes tipos de grafos los cuales son:

**Grafo Simple.**

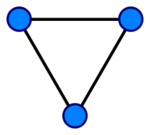
**Multigrafo.**

**Grafo dirigido.**

Por ahora solamente nos enfocaremos en lo que es un grafo simple y un multigrafo dirigido.

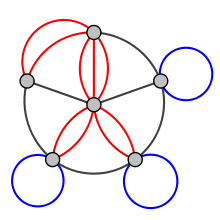
**Grafo simple:**

Dícese del grafo que no tiene lazos ni aristas múltiples entre sus vértices. En otras palabras un grafo simple es un grafo en los que existe a lo más una arista que une dos vértices distintos.

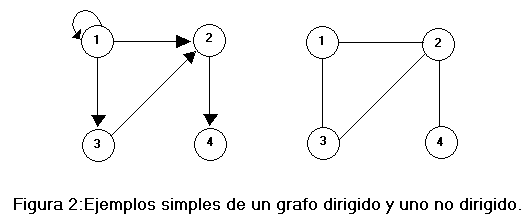


**Multigrafo:**

Es un grafo que está facultado para tener aristas múltiples, esto quiere decir, varias aristas que relacionan el mismo vértice o nodo. De esta forma dos nodos pueden estar conectados por más de una arista.



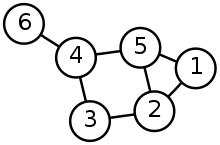
**Grafo dirigido.**Es aquel grafo donde las aristas tienen una dirección y sentido, es decir que la arista es una flecha que apunta hacia un nodo. Donde el vértice A conocido como origen, es la cola y el vértice B es el destino.



Para el grafo dirigido también también puede existir un multigrafo, el cual es conocido como multidirigido o multigrafo dirigido.

**Adyacencia.**

Un vértice adyacente de un vértice v en un grafo es un vértice que está conectado a v mediante una arista. Gráficamente es:



En la anterior imagen podemos decir que, el vértice 5 es adyacente a los vértices 1, 2 y 4, pero no es adyacente a los vértice 3 y 6.

**Grado de una vértice.**

Es el número de incidentes al vértice, es decir el número de aristas que tiene el vértice. Y se denota con el símbolo δ(v).

**Tipos de grafos simples.**

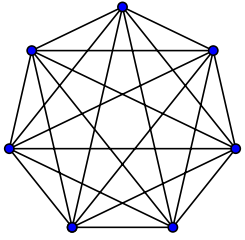
Aquí nos encontramos con los siguientes grafos:

**Grafos completos.**

**Ciclos.**

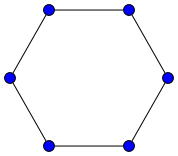
**Ruedas.**

**Grafos completos:** es un grafo simple donde cada par de vértices está conectado por una arista.



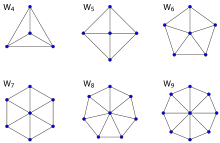
**Grafo ciclo.**

En Teoría de grafos, un Grafo ciclo o simplemente ciclo es un grafo que se asemeja a un polígono de n lados. Consiste en un camino cerrado en el que no se repite ningún vértice a excepción del primero que aparece dos veces como principio y fin del camino. Un Grafo ciclo de n vértices se denota Cn. El número de vértices en un grafo Cn es igual al número de aristas, y cada vértice tiene grado par, por lo tanto cada vértice tiene dos aristas incidentes.



**Grafo Rueda**

un grafo rueda (Wn), o simplemente rueda, es un grafo con n vértices que se forma conectando un único vértice a todos los vértices de un ciclo-(n-1).



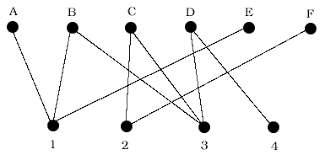
**Grafo Bipartito.**

En teoría de grafos, un grafo bipartito es un grafo G=(N,E) cuyos vértices se pueden separar en dos conjuntos disjuntos U y V, es decir, tal que se cumple:

**U** U **V = N**

**U** n **V = 0**

En otras palabras, si tenemos un grafo G en el cual podemos obtener dos conjuntos de vértices ya sea V1 Y V2 donde los vértices de cada conjunto no se relacionan o tiene aristas entre sí, pero si se relacionan con el otro conjunto, entonces es un grafo bipartito.



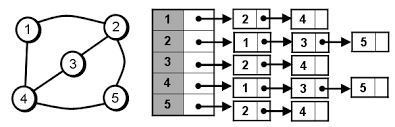
La anterior imagen sería un grafo bipartito, ya que si tenemos nuestro primer conjunto V1 con los vértices A,B,C,D,E,F y el conjunto V2 con los vértices 1,2,3,4 podemos ver que en el conjunto V1 ningún vértice se relaciona, igualmente para el V2, pero si se relacionan los conjuntos, es decir, el vértice A del conjunto V1 se relaciona o tiene una arista con el vértice 1 del conjunto V2, por tanto es un grafo bipartito.

Por un lado, los grafos se pueden representar de diferentes maneras, entre ellas están:

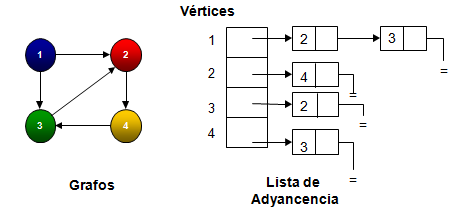
* **Lista de adyacencia.**
* **Matriz de adyacencia.**
* **Matriz de incidencia.**

**Lista de adyacencia:**

Es una forma de representar un grafo sin aristas múltiples. Especifican los vértices que son adyacentes a cada uno de los vértices del grafo.

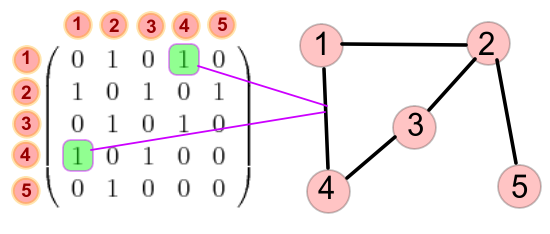


En el caso de un grafo dirigido, los vértices adyacentes son los cuales las aristas tienen la dirección hacía otro vértice e incluso hacia él mismo.



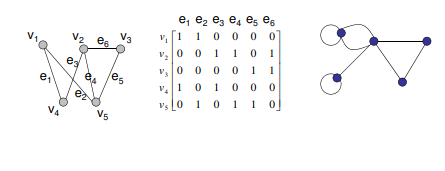
**Matriz de adyacencia**

Es una matriz de tamaño n x n, donde las columnas y filas están dadas por todos los vértices del grafo y representa los vértices adyacentes de todos los vértices del grafo, si existe vértices adyacentes en un vértice se coloca 1, en caso contrario se rellena de 0.

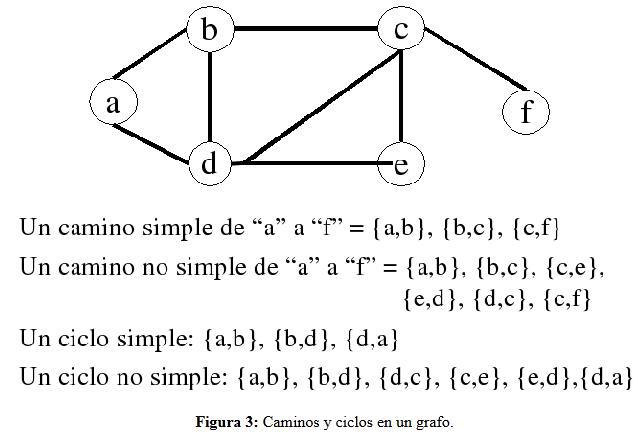


**Matriz de incidencia.**

Prácticamente la matriz de incidencia consta de una matriz n x m, donde las n-filas componen los vértices del grafo, y m-columnas componen el peso o la etiqueta de la arista. En el cual se observa cada arista a qué vértice está incidiendo o está conectando y se la marca 1 si incide a algún vértice, en caso contrario se coloca 0. Esto quiere decir que se forma una matriz binaria.



Un **camino** del grafo una secuencia de aristas que comienzan en un vértice del grafo y  
recorre ciertas aristas del grafo siempre conectando pares de vértices  
adyacentes.



El camino es un circuito si comienza y termina en el mismo vértice y no tiene aristas repetidas, esto es, si u = v, y tiene longitud mayor a cero.

Un grafo no dirigido es **conexo** cuando todos sus vértices tiene una relación o una forma de comunicarse entre sí, ya sea simplemente o relacionándose a través de uno o más vértices.

Se le llama **vértice de corte** a aquel que al ser eliminado junto a todas las aristas incidentes en él produce un subgrafo con más componentes conexas.

Una **arista de corte** es aquella que cuya eliminación produce un grafo con más componentes conexas que el grafo original se llama arista de corte o puente.

Llegamos a un punto en el cuál nos preguntamos qué funciones o beneficios puede generar un grafo además de relacionar un conjunto de elementos, pues bien, nos ofrece unos diferentes métodos de búsqueda, en estos nos podemos encontrar con los siguientes:

* **Búsqueda ciega o sin información**
* **Búsqueda heurística**
* **BFS**
* **DFS**

**Búsqueda ciega o sin información:**

Consiste en una búsqueda en el cual no usa información sobre el problema, normalmente se realiza un recorrido exhaustivo.

Consiste en coger un vértice y el proceso de búsqueda se concibe con la construcción de un recorrido del grafo.

**Búsqueda heurística:**

Utiliza información acerca del problema, como los costos, pesos, etc. Esto quiere decir que posee información muy valiosa para orientar a la búsqueda.

De acuerdo a lo anterior es necesario buscar una estrategia de búsqueda, ya sea por amplitud o profundidad.

**BFS:**

El BFS o búsqueda por amplitud es de uno de los más simples de búsqueda de grafos.

Básicamente, se empieza con un vértice origen y se exploran todos los vecinos de este nodo. A continuación para cada uno de los vecinos se exploran sus respectivos  
vecinos adyacentes, y así hasta que se recorra todo el grafo.Cabe aclarar que sirve para grafos dirigidos y no dirigidos.

Consta en principio con los nodos o vértices de color blanco para saber que no se ha visitado, una vez visitado lo cambia a color gris, y empieza a visitar sus vértices adyacentes, una vez visita todos sus vértices adyacentes cambia de color gris a negro.

**DFS:**

El DFS es un algoritmo de búsqueda por profundidad que permite recorrer todos los nodos de un grafo o árbol de manera ordenada, pero no uniforme.

Expande un vértice hasta que no se pueda más y regresa. Este proceso continúa hasta que se descubran todos los vértices alcanzables desde el vértice original.

De igual forma, el DFS sigue el mismo proceso del BFS, colorea los vértices para indicar su estado.

Un **grafo ponderado** Son aquellos en los que se asigna un número a cada una de las aristas.

Uno de los problema de los grafos ponderados es el de determinar un camino de longitud mínima entre dos vértices de una red. Más concretamente, se define la longitud de un camino en un grafo ponderado como la suma de los pesos de las aristas de ese camino.

Existen algunos algoritmos que nos ayudan a determinar el camino mínimo entre dos vértices.

* **Dijkstra**
* **Floyd - Warshall**

**Dijkstra:**

Consiste en ir explorando todos los caminos más cortos que parten del vértice origen y que llevan a todos los demás vértices; cuando se obtiene el camino más corto desde el vértice origen hasta el resto de los vértices que componen el grafo, el algoritmo se detiene..

**Floyd - Warshall.**

Es un algoritmo de análisis sobre grafos que permite encontrar el camino mínimo en grafos dirigidos ponderados. El algoritmo encuentra el camino entre todos los pares de vértices en una única ejecución, constituyendo un ejemplo de programación dinámica. Cabe aclarar que este algoritmo trabaja con la matriz D inicializada con las distancias directas entre todo par de nodos, es decir la matriz de pesos.

Compara todos los posibles caminos a través del grafo entre cada par de vértices. El algoritmo es capaz de hacer esto con sólo V3 comparaciones, lo hace mejorando una estimación del camino más corto entre dos vértices, hasta que se sabe que la estimación es óptima.

Un grafo también nos ofrece la oportunidad de encontrar árboles recubridores mínimos en un grafo conexo no dirigido y cuyas aristas están etiquetadas.

En realidad, ¿Qué es un árbol recubridor mínimo?

Un árbol recubridor mínimo de un grafo, es un subgrafo que tiene que ser un árbol y contender todos los vértices del grafo inicial. Por lo que existen algunos algoritmos que nos ayudan a encontrarlos:

* **Algoritmo de Prim**
* **Algoritmo Kruskal**

**Algoritmo de Prim**

El algoritmo incrementa continuamente el tamaño de un árbol, comenzando por un vértice inicial al que se le van agregando sucesivamente vértices cuya distancia a los anteriores es mínima. Esto significa que en cada paso, las aristas a considerar son aquellas que inciden en vértices que ya pertenecen al árbol.  
  
El árbol recubridor mínimo está completamente construido cuando no quedan más vértices por agregar.  
  
El algoritmo podría ser informalmente descrito siguiendo los siguientes pasos:  
  
1. Inicializar un árbol con un único vértice, elegido arbitrariamente del grafo.  
2. Aumentar el árbol por un lado. Llamamos lado a la unión entre dos vértices: de las posibles uniones que pueden conectar el árbol a los vértices que no están aún en el árbol, encontrar el lado de menor distancia y unirlo al árbol.  
3. Repetir el paso 2 (hasta que todos los vértices pertenecen al árbol)

**Algoritmo de Kruskal**

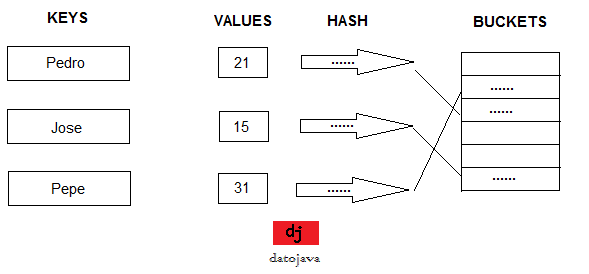
El algoritmo de Kruskal es un algoritmo de la teoría de grafos para encontrar un árbol recubridor mínimo en un grafo conexo y ponderado. Es decir, busca un subconjunto de aristas que, formando un árbol, incluyen todos los vértices y donde el valor de la suma de todas las aristas del árbol es el mínimo.

El algoritmo de Kruskal es un ejemplo de algoritmo voraz que funciona de la siguiente manera:  
  
1. se crea un bosque B (un conjunto de árboles), donde cada vértice del grafo es un árbol separado  
2. se crea un conjunto C que contenga a todas las aristas del grafo  
3. mientras C es no vacío:  
 - eliminar una arista de peso mínimo de C  
 - si esa arista conecta dos árboles diferentes se añade al bosque, combinando los dos árboles en un solo árbol  
 - en caso contrario, se desecha la arista.

Al acabar el algoritmo, el bosque tiene un solo componente, el cual forma un árbol de expansión mínimo del grafo.  
  
En un árbol de expansión mínimo se cumple:  
  
la cantidad de aristas del árbol es la cantidad de nodos menos uno (1).

**8. HashMap**

HashMap es otra estructura de datos que almacena datos asociando una llave a un valor, sirve para muchas cosas y tiene ciertas características que la definen, por ejemplo, no permite key duplicados, cada key tiene que estar asociado a un valor como máximo, si agregas un key que ya existe sobrescribe el valor del key anterior, sólo permite Object types lo que quiere decir que no puedes poner un valor primitivo.



**3. Búsqueda de soluciones creativas.**

En resumen, hasta ahora llevamos recolectando información acerca de las diferentes alternativas que pueden ser óptimas para la resolución de nuestro problema, primordialmente se indagó sobre las siguientes 8 estructuras de datos

* Árbol ABB
* Árbol Rojo y Negro
* Árbol AVL
* Cola
* Pila
* Tabla Hash
* Grafos
* HashMap

No obstante, es trascendental llevar a cabo ideas centradas solamente a las condiciones de nuestro problema.

En primer lugar, el problema está basado en una cantidad de edificios y rutas, el cual se puede tornar bastante tedioso a la hora de cumplir con los diferentes requerimientos, ya que se necesita almacenarlos como una principal estrategia, una vez dicho esto, hay que tener en cuenta que cada edificio tiene una etiqueta propia o una manera de poderlo de identificar, es decir no existe un edificio que se llame igual a otro. En consecuencia, en base a lo que se consultó, se puede afirmar que la estructura que más se asemeja a suplir la necesidad de almacenamiento es el HashMap, debido a que cumple con una key única que puede ser el nombre del edificio junto a su respectivo valor de almacenamiento (value).

En segundo lugar, se debe almacenar las diferentes rutas que conectan los diferentes edificios, como se pudo ver en la fase anterior cada edificio contiene o la componen por lo menos una ruta para llegar a él; es difícil pensar en una estructura de almacenamiento que me relacione esto, pero en realidad, si analizamos la estructura de grafo, sería una alternativa interesante, ya que como bien sabemos un grafo lo componen vértices y aristas, en el cual un par de vértices están conectados por lo menos con una arista, lo que conlleva a pensar en nuestro problema, porque de un edificio puede salir una ruta hacia otro, a modo de ejemplo podemos ver en la *tabla 1* y en el *gráfico 1* que del edificio A podemos encontrarnos una ruta que nos lleve al edificio B de la universidad Icesi. Por tanto, los vértices serían nuestros edificios y las rutas nuestras aristas.

Por último y en relación con lo anterior, el grafo nos ofrece gran parte de la solución a nuestro problema, sabemos que a través de ellos podemos encontrar el camino más corto entre un vértice (recordemos que nuestros vértices son nuestros edificios) a otro. También sabemos que un grafo puede ser ponderado, es decir que las aristas tienen un peso. Por ende, para poder saber cuál es nuestro camino que tiene menor costo, es decir por dónde podemos llegar más rápido a nuestro destino, debemos saber las distancias de nuestras rutas, esto nos conlleva afirmar que nuestras aristas tienen que tener un peso, que es la distancia que hay entre un vértice a otro.

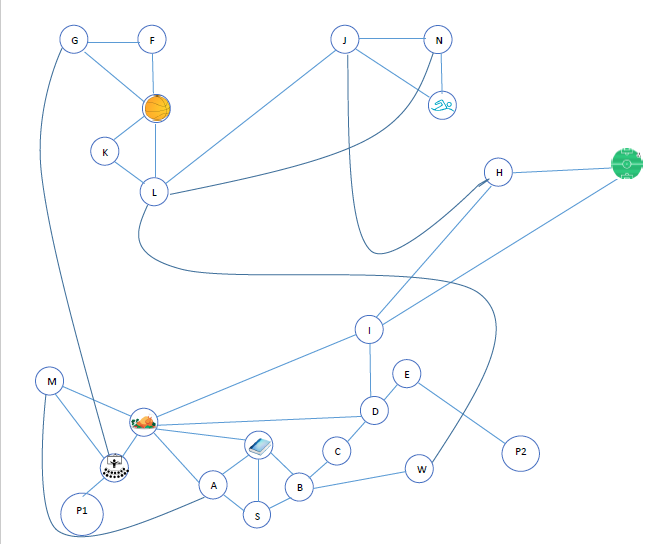
Sin duda alguna, los diferentes métodos y algoritmos que podemos realizar con un grafo nos van a llevar a usar otras estructuras de datos como las colas y pilas, ya que muchas veces necesitamos ir almacenando datos e irlos sacando en un orden específico como lo es FIFO o LIFO, respectivamente; también nos sirve para ir liberando espacio en memoria, así nuestro problema va ser un poco más eficiente adaptándolo a la realidad.

Con las ideas anteriormente dadas, se consiguió llegar a una preferencia entre las 8 estructuras para la solución fundamental de nuestro problema, las cuales son:

* HashMap
* Grafos
* Colas
* Pilas

Sin embargo, hay que tener en cuenta otra idea y es la parte externa del aplicativo, la parte en la que el usuario va a interactuar con las diferentes funciones; simplemente una parte descriptiva es que él pueda asignar la posición o el edificio en el que se encuentra y designar el destino, así el programa le va a mostrar el camino más corto para llegar a su destino, o mejor aún, muchas veces él solamente puede marcar la ubicación actual y el mismo programa se encargará de mostrarle las rutas más cortas hacia todos los edificios con su respectiva distancia.

A continuación se mostrará una representación de lo que sería el grafo de los edificios de la universidad Icesi con sus diferentes rutas:



Algunos nombre de los edificios fueron colocados con imágenes, ya que por ser un poco extensos no cabían dentro del vértice. Por lo que:

* El vértice W es el edificio Wonka
* El vértice con el plato de comida, es el edificio Central o centro de comida.
* El vértice adyacente a central, es el edificio auditorios.
* El vértice que contiene el libro, es el edificio Biblioteca.
* El vértice con la cancha de fútbol, es el sitio donde se encuentra las canchas de fútbol 11.
* El vértice del nadador, es el coliseo 2.
* El vértice con el balón de baloncesto, es el coliseo 1.
* Los vértices P1 y P2 corresponden al parqueadero de la biblioteca.

**4. Transición de las ideas a los Diseños Preliminares.**

En esta fase se va a indagar el porqué se descartaron algunas alternativas nombradas en la fase anterior; recapitulando, estas alternativas fueron:

* Árbol ABB, Rojo y Negro, AVL
* Tabla Hash

Cabe aclarar que las ideas descartadas fueron más que las ideas escogidas, por lo que se llevará a cabo un proceso de justificación en cada caso, empezando por las ideas descartadas, donde se explicará a partir de nuestras percepciones, ideas y requisitos del problema el porqué fueron excluidas, y para el caso de las ideas no descartadas se inducirá un diseño preliminar, esto para conocer una mayor información sobre cada una de las ideas.

**Árboles ABB, AVL, Rojo y Negro.**

Una de las principales funciones de los tres diferentes árboles es el almacenamiento ordenado, esto quiere decir que cada dato que se va ingresando al árbol se va ubicando de manera en que los datos menores a la raíz se almacenan hacía la izquierda, y los mayores hacia la derecha.

Estos 3 árboles los diferencian su balance; se dice que un árbol está balanceado cuando para todos los nodos, la altura de la rama izquierda no difiera en más de una unidad a la altura de la rama derecha o viceversa. Esto hace que la búsqueda dentro de un árbol sea mucho más eficiente.

El árbol menos balanceado es el ABB, ya que no usa estrategias para balancear su árbol.

Los árboles que usan estrategias para estar balanceados son el rojo y negro, y el AVL, estos nos ofrecen una búsqueda bastante eficiente.

Ahora bien, sabiendo un poco sobre el funcionamiento de los árboles binarios, y si lo relacionamos con el problema, vemos que no es necesario para resolver algún requerimiento, ya que no necesitamos fundamentalmente organizar datos para una búsqueda eficiente; además de que un árbol binario solamente nos ofrece información sobre sus nodos, y ya vimos que no necesitamos sólo esa información, sino también de sus conexiones.

**Tabla Hash:**

Tabla Hash es una estructura de datos muy útil para las búsquedas eficientes, su tiempo promedio es O(1), sin importar la cantidad de elementos que contenga la tabla, su diferencia con la estructura HashMap es muy poco, pero decidimos descartar esta estructura debido a que la tabla Hash no nos ofrece un orden de iteración predecible, por si alguna vez necesitamos conocer la totalidad de los edificios o rutas que están almacenadas dentro de la tabla Hash.

En contextualización, todo lo dicho anteriormente pertenece a las ideas descartadas, donde fueron aclaradas las razones por las cuales no se tuvieron en cuenta para la solución de nuestro problema. Ahora, seguimos con un diseño preliminar, que consiste en conocer u obtener más información de las ideas que no descartamos, o sea de las alternativas que tomamos como referencia para la solución de nuestro problema. Para recordar, estas alternativas fueron las siguientes estructuras de datos:

* HashMap
* Grafos
* Cola
* Pila

**Diseño preliminar.**

Aún nos queda lo más relevante, y es llevar a cabo las razones de nuestras alternativas que servirán para la resolución de nuestro problema (ideas no descartadas), sin embargo, estas razones se plantearon a través de ciertas ideas concretas, por esto, se formuló un diseño preliminar básico y sencillo para comprender la relación que existe entre las alternativas y el problema que se nos plantea.

Antes de analizar el diseño preliminar, cabe recalcar que se inducirá una estructura de diseño, que a continuación se plantea:

* Ideas
* FODA
* Análisis
* Conclusión

En primer lugar, las ideas se trataron un poco dispersas, por la razón de que es un proceso que no es organizado, pero es importante para una buena disidencia de alternativas, aunque esto no quiere decir que es un mal comienzo, ya que es una base para el seguimiento de nuestras escogencias. Algunas de nuestras ideas fueron:

* Es indudable que los edificios tienen su propio nombre, y es por esto que es fácil encontrar o reconocer alguno apenas son mencionados.
* Otra idea relevante, fue la relación entre las conexiones de los edificios, cómo en realidad está infraestructurada la universidad, qué es lo que más nos conviene para alcanzar el objetivo de poder entender y comprender el sistema de rutas de la universidad.

En segundo lugar, para hacer un énfasis en cada idea, se clasificó las fortalezas, oportunidades, debilidades y amenazas de nuestras alternativas para así verificar la potencialidad de nuestra resolución.

Para nuestra primera idea se tomó como base la estructura de datos **HashMap.**

|  |  |  |
| --- | --- | --- |
|  | Factores internos | Factores externos |
| Puntos positivos | Fortalezas   * Guarda datos con una única clave. | Oportunidades   * Busca en tiempo eficiente O(1). |
| Puntos negativos | Debilidades   * Guarda claves y valores nulos. | Amenazas   * Claves repetidas, valores variables. |

*tabla 1*

Para la **tabla 1** tenemos que los puntos positivos se asemeja más a lo planteado en el problema, porque nos ofrece un sistema de almacenamiento con una referencia única, en otras palabras almacena un edificio o ruta única dentro de la tabla o sea que no vamos a encontrar otro edificio o ruta igual dentro de la estructura, sin embargo, también tenemos puntos negativos, sabemos que el HashMap nos permite guardar elementos nulos dentro de él, pero gracias a las funciones de java puede ser controlado, ya que puede ser una debilidad para nuestro aplicativo. Por otra parte, también nos podemos encontrar con que se guarde una clave que ya exista dentro de la estructura, por lo que ella se encarga de sobrescribir el valor que se encontraba antes por el actual, pero no es malo del todo ya que nos está asegurando de no repetir dos claves en la misma tabla. Aún así puede ser controlado por el programa.

Para segunda idea, se tomaron como base las estructuras **cola** y **pila**.

|  |  |  |
| --- | --- | --- |
|  | Factores internos | Factores externos |
| Puntos positivos | Fortalezas   * Almacena cualquier tipo de dato. * Es dinámico. | Oportunidades   * Métodos de inserción, supresión y extracción. |
| Puntos negativos | Debilidades   * No busca elementos. | Amenazas   * Datos nulos. |

*tabla 1.1*

Para la **tabla 1.1** nos encontramos que nuestros puntos positivos, nos ofrece un almacenamiento dinámico, es decir no tenemos que colocar un límite de espacio; además de que nos ofrece un sistema de extracción único, en el caso de la cola sale siempre el primero que ha ingresado, y en la pila sale el último que ingresa. Por otro lado, los puntos negativos se relacionan en la manera de buscar elementos, ya que esta estructura no nos ofrece una búsqueda, pero en nuestra problemática solo con un sistema de búsqueda como lo ofrece el HashMap es suficiente, así que no nos afecta en este sentido. Cabe aclarar que las colas y pilas se usarán en el sentido de la implementación de algunos algoritmos para el funcionamiento correcto del grafo.

Por último, como otra alternativa para resolución de nuestra idea uno y dos aparece la última estructura de datos escogida: **Grafos.**

|  |  |  |
| --- | --- | --- |
|  | Factores internos | Factores externos |
| Puntos positivos | Fortalezas   * Relación de elementos. | Oportunidades   * Diferentes métodos de recorridos y caminos cortos. |
| Puntos negativos | Debilidades | Amenazas   * Vértices sin conexión. |

*tabla 1.2*

Para la **tabla 1.2** nos encontramos con la estructura grafo, en los puntos positivos, como se ha venido hablando, nos ofrece entender la relación entre elementos o conjunto de elementos, en nuestro caso son puntos positivos, puesto que encontramos la manera de representar gráficamente y codificar el grafo planteado en la fase anterior, por consiguiente la cantidad de métodos que nos ofrece es una oportunidad gratificante para la solución de nuestro problema, ya que como se mencionó anteriormente, justo es lo que se necesita para encontrar, por ejemplo, el camino más corto de un edificio a otro. Por otra parte, una amenaza que puede presentar un programa de grafos es que los vértices no esten relacionados, ya que si fuera así, no representaría la estructura primordial de un grafo y por consiguiente sería ilógico aplicar los diferentes métodos que nos ofrecería. Pero en realidad, esto no es gran problema, ya que sabemos que nuestros vértices si están relacionados entre sí.

Para resumir, todas las alternativas seleccionadas para la resolución de nuestro problema tienen puntos negativos, sin embargo, estos puntos no influyen mucho en nuestro aplicativo, porque, por una parte pueden repararse gracias a las fuerzas y oportunidades de cada alternativa, y por otra parte, la codificación nos ofrece una ciertas posibilidades para evitar muchos errores, por ejemplo en el caso de que el usuario quiera ingresar o almacenar un dato que no soporte o sea el indicado para el grafo, pila o cola.

**Fase 5. Evaluación y selección de la mejor opción.**

En la evaluación vamos a tener en cuenta varios criterios:

**Criterio A:** Es dinámico

[2] Dinámico

[1]Fijo

Este criterio se evalúa si la alternativa presenta un comportamiento activo o cambiante para el aplicativo.

Se califica 1 si no varía dentro del programa.

Se califica 2 si varía dentro del programa.

**Criterio B:** Suple con la solución planteada en el problema.

[3] Sí

[2] Suple pero no del todo.

[1] No

Este segundo criterio, evalúa si la alternativa satisface totalmente el requerimiento que se nos presentó en el problema.

Se califica 3, si lo satisface totalmente.

Se califica 2, si lo satisface de vez en cuando.

Se califica 1, si no lo satisface.

**Criterio C**: Mejora el rendimiento (en tiempo de ejecución) para el aplicativo

[3] Sí

[2] Muy poco

[1] No varía

Para este tercer criterio, se evalúa si la alternativa aumenta el rendimiento del aplicativo.

Se califica 3, si en realidad sirve para mejorar la ejecución del aplicativo.

Se califica 2, si mejora en tiempo de ejecución pero muy insignificante.

Se califica 1, no mejora en lo absoluto.

**Criterio D:**  Organiza bien los datos dentro del problema.

[3] Sí

[2] Aproximadamente ordenado

[1] No

Para este último criterio, se evalúa si la estructura de dato nos asegura un ordenamiento en cuanto a los datos que estamos usando para la comprensión y desarrollo del aplicativo.

Se califica 3, si los ordena de manera adecuada como lo requiere el problema.

Se califica 2, se aproxima a un orden medianamente asequible para el proble.

Se califica 1, no ordena los datos.

***Evaluación:***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Métodos / Criterios | **Criterio A** | **Criterio B** | **Criterio C** | **Criterio D** | **Total** |
| Grafos | 2 | 3 | 3 | 3 | 11 |
| HashMap | 2 | 3 | 3 | 2 | 10 |
| Cola y Pila | 2 | 2 | 2 | 1 | 7 |

*tabla 2*

En la **tabla 2** se puede ver claramente una calificación objetiva para cada una de las estructuras seleccionadas, por lo que empezaremos analizar cada una de estas dependiendo de su puntaje.

En primer lugar, los grafos ha sido la estructura que ha sacado la máxima puntuación, si analizamos cumple con cada criterio planteado, en pocas palabras es la estructura fundamental, ya que es la que opera y modela toda la información. En cuanto al criterio A es una estructura dinámica, puesto que podemos agregar un vértice y una arista las veces que queramos, no tiene límites. Para el criterio B y C es rectificante la puntuación porque si soluciona gran parte del problema, además de por ser una estructura basada en información relacionada las operaciones mejorarán el rendimiento del programa. Por último, el criterio D es un rotundo sí porque nos organiza de manera entendible la relación que existe entre los edificios.

En segundo lugar, HashMap en comparación a los grafos se diferenciaron frente a los criterios en uno solo, y es que el HashMap no organiza rotundamente los datos, pero si tiene una aproximación en cuanto al problema, ya que los guarda de una manera que nos asegura que no se repita ningún dato dentro de la estructura.

Por último, las colas y las pilas, son estructuras que obtuvieron el menor puntaje, debido a que a pesar de ser dinámicas y de que no organiza los datos, presenta una participación mínima dentro de la solución del problema, son estructuras alternativas que ofrecen sus fortalezas para llevar a cabo una solución mínima del programa.

**6. Preparación de Informes y Especificaciones**

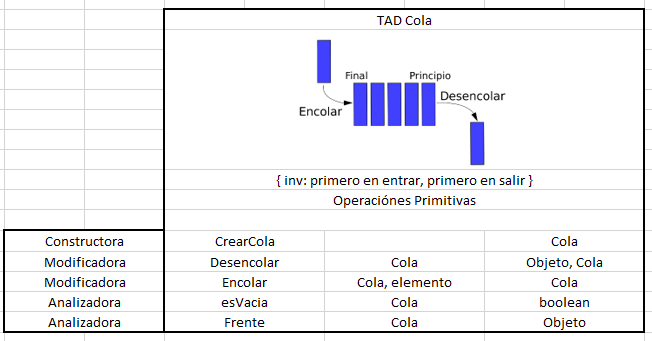
En esta fase se llevará a cabo la parte del diseño de nuestro método de ingeniería para la solución del problema.

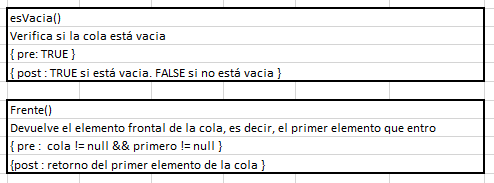
En primer lugar, se presentará los Tipos de Datos Abstractos de nuestras estructuras de datos que se implementarán en la codificación, para así llevar una comprensión detallada de las operaciones que nos ofrece cada una de estas.

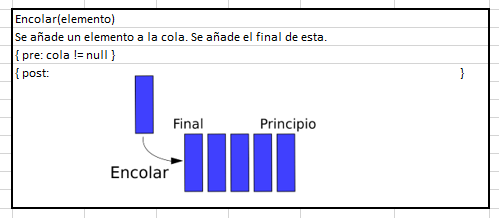
En segundo lugar, también se incluirá el diagrama de clases respectivo para la descripción de nuestra codificación de manera que cualquier programador o persona relacionada con el tema pueda entender las clases, los atributos, las operaciones(métodos) y las relaciones entre objetos.

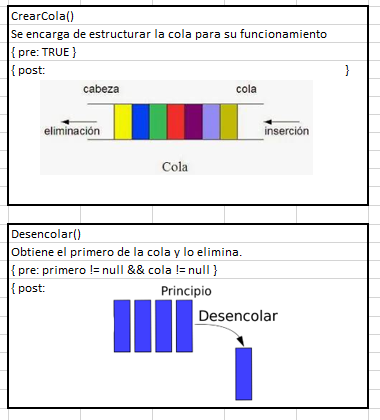
Por último, para comprobar y verificar el comportamiento de los métodos o las salidas que esperamos en cada operación, se usará el diseño de casos de pruebas unitarias, para detallar las entradas, y las salidas que se deben dar para un buen funcionamiento del aplicativo.

**TADs:**

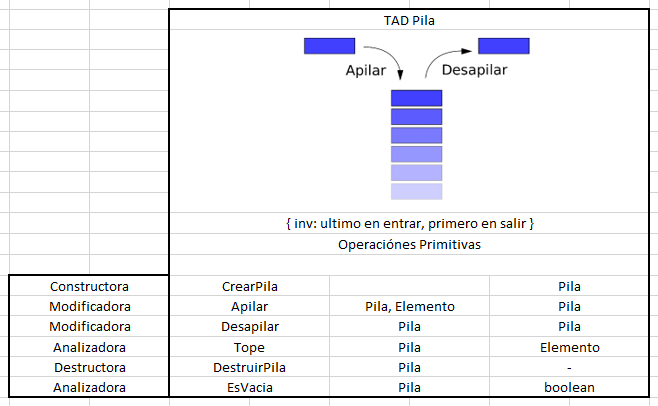
****

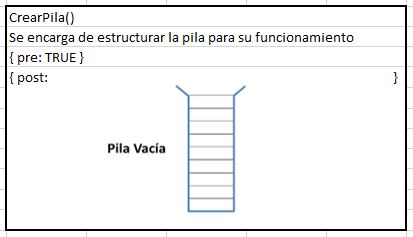
****

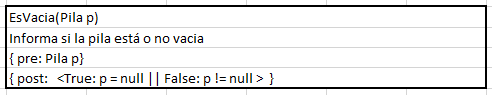


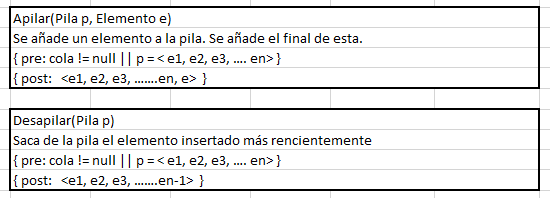
****

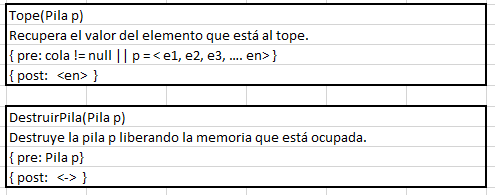
**Pila**

****

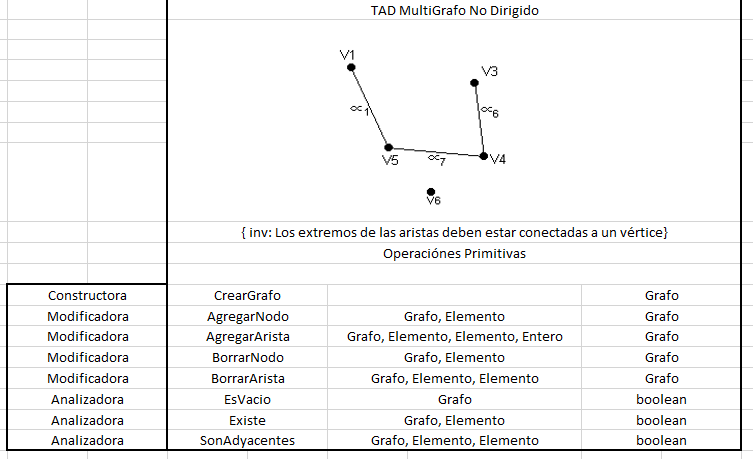
****

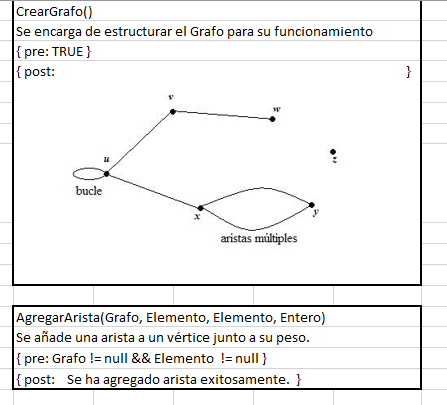
****

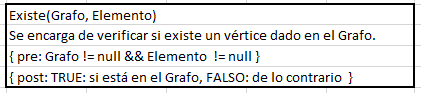
****

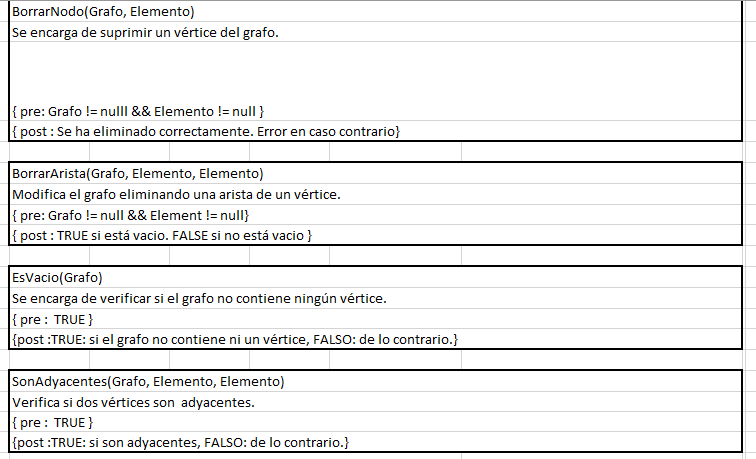
****

**Grafos:**

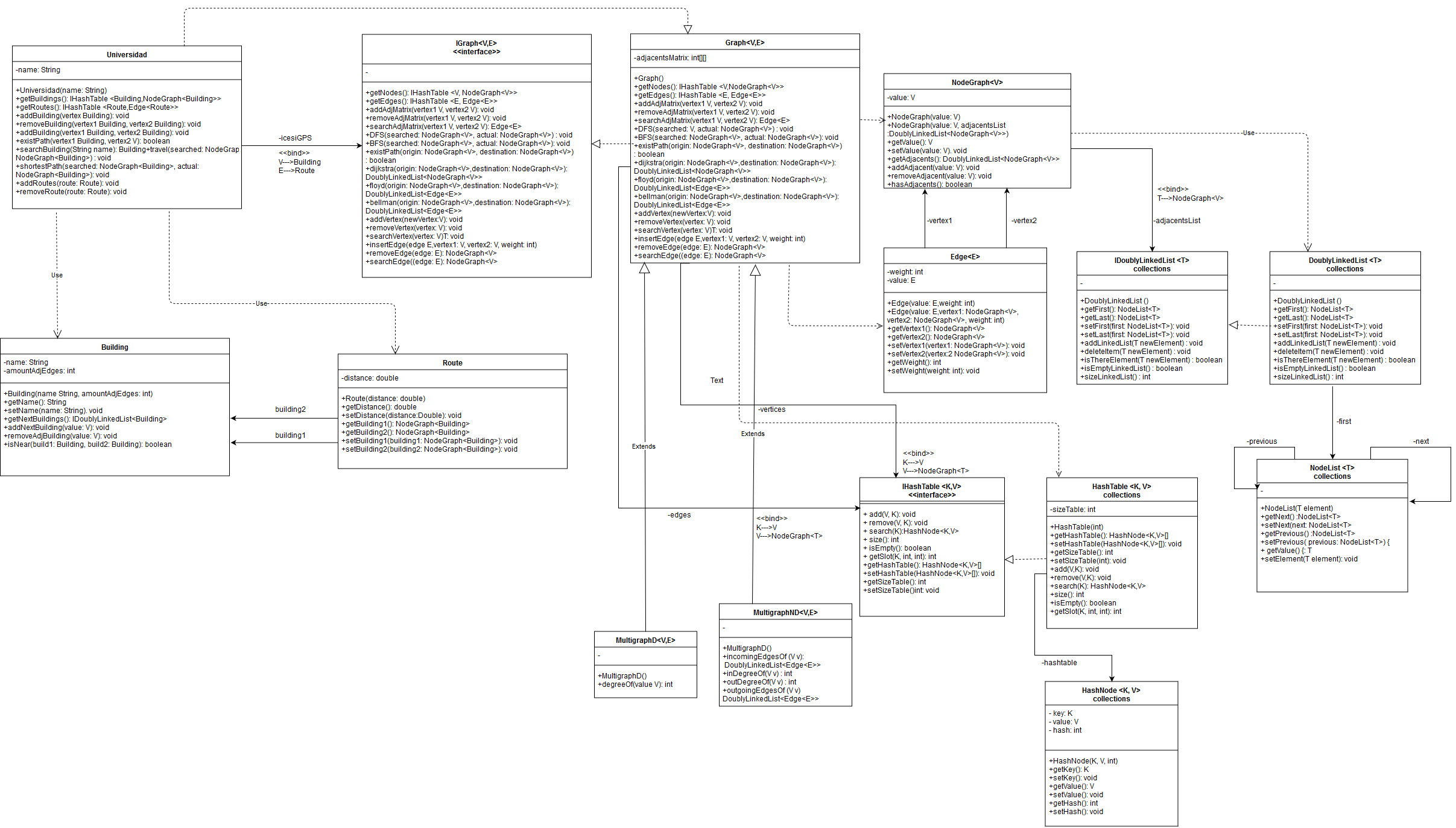
****

****





**Diagrama de clases.**

****

NOTA: para ver mejor el diagrama de clases, se encuentra adjunto al github en formato .jpg.

Diseño de pruebas:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Clase | Método | Escenario | Valores | Resultado |
| GraphTest | addVertexTest() | Se instanció un objeto de tipo Grafo. | Se agregan los siguientes edificios al grafo:  vertex=” vertice1”  key=”0” | Los objetos han sido agregados al grafo de forma adecuada, cada vértice del grafo contiene la información correcta |
| GraphTest | removeVertexTest() | Se instanció un objeto de tipo Grafo, se crearon 5 vértices y 8 aristas y se agregaron al grafo. | Key=”0” | El objeto fue eliminado exitosamente ya no hace parte del grafo |
| GraphTest | searchTest() | Se instanció un objeto de tipo Grafo . | edifico1=” A” | El objeto ha sido encontrado en el grafo. |
| GraphTest | insertEdgeTest | Se instanció un objeto de tipo Grafo . | Vertex1=” vertice0”  Key1=”0”  vertex2=” vertice1”  key2=”1”  edge=6.0  keyEdge=”0” | Los objetos han sido agregados al grafo de forma adecuada, la arista del grafo contiene los vértices correctos |
| GraphTest | removeEdgeTest | Se instanció un objeto de tipo Grafo . | Vertex1=” vertice0”  Key1=”0”  vertex2=” vertice1”  key2=”1”  edge=6.0  keyEdge=”0” | El objeto fue eliminado exitosamente ya no hace parte del grafo |
| GraphTest | areAdjacentsTest | Se instanció un objeto de tipo Grafo, se crearon 5 vértices y 8 aristas y se agregaron al grafo. | Key=”0”  Key2=”1”  Edge=6.0  keyEdge=”0” | El resultado es true, dado que es un nodo no dirigido para ambos sentidos el resultado es correcto |
| GraphTest | dikstraTest | Se instanció un objeto de tipo Grafo, se crearon 5 vértices y 8 aristas y se agregaron al grafo. | Way1=8+10  Way2=14+13  Way3=14+9+12  Way4=8+20+12 | Se comparan todos los posibles caminos de un vertive inicio a un vértice final para verificar que Dijkstra retono el de menor valor |
| GraphTest | floydWarshallTest | Se instanció un objeto de tipo Grafo, se crearon 5 vértices y 8 aristas y se agregaron al grafo. |  | Se buscan dos caminos corto de un vértice a otro con dijkstra y se revisa la matriz de Floyd para saber si encontró los caminos minimos hacia todos los vertices del grafo |
| GraphTest | PrimTest | Se instanció un objeto de tipo Grafo, se crearon 5 vértices y 8 aristas y se agregaron al grafo. | menorEdge=7  arrayD=dijkstra desde el vertice de la arista 7 | El camino hallado por dijkstra desde la arista de menor peso debe ser igual al camino hallado por prim |

**Bibliografía**

[**https://es.wikipedia.org/wiki/Grafo**](https://es.wikipedia.org/wiki/Grafo)

Grafo. wikipedia

[**https://es.wikipedia.org/wiki/Multigrafo**](https://es.wikipedia.org/wiki/Multigrafo)

Multigrafo. wikipedia

[**https://upload.wikimedia.org/wikipedia/commons/5/5f/Introducci%C3%B3n\_a\_la\_Teor**](https://upload.wikimedia.org/wikipedia/commons/5/5f/Introducci%C3%B3n_a_la_Teor%C3%ADa_de_Grafos.pdf)

[**%C3%ADa\_de\_Grafos.pdf**](https://upload.wikimedia.org/wikipedia/commons/5/5f/Introducci%C3%B3n_a_la_Teor%C3%ADa_de_Grafos.pdf)

Teoría de grafos. wikimedia

[**https://es.wikipedia.org/wiki/Vecindad\_(teor%C3%ADa\_de\_grafos)**](https://es.wikipedia.org/wiki/Vecindad_(teor%C3%ADa_de_grafos))

Vecindad de grafos. Wikipedia

[**https://es.wikipedia.org/wiki/Grafo\_ciclo**](https://es.wikipedia.org/wiki/Grafo_ciclo)

Ciclo en un grafo. wikipedia

[**https://compdiscretas.wordpress.com/2012/11/20/tema-2-recorridos-en-un-grafo-concepto-de-caminocircuito-conexidad-y-planaridad/**](https://compdiscretas.wordpress.com/2012/11/20/tema-2-recorridos-en-un-grafo-concepto-de-caminocircuito-conexidad-y-planaridad/)

Camino y circuito. compdiscretas

[**https://es.wikipedia.org/wiki/Algoritmo\_de\_Dijkstra**](https://es.wikipedia.org/wiki/Algoritmo_de_Dijkstra)

Algoritmo Dijkstra. Wikipedia

[**https://www.ecured.cu/Floyd-Warshall**](https://www.ecured.cu/Floyd-Warshall)

Algoritmo Floyd-Warshall. Ecured

[**https://es.wikipedia.org/wiki/%C3%81rbol\_recubridor\_m%C3%ADnimo**](https://es.wikipedia.org/wiki/%C3%81rbol_recubridor_m%C3%ADnimo)

Árbol recubridor mínimo. Wikipedia

[**http://datojava.blogspot.com/2016/07/queEsHashMapYComoFunciona.html**](http://datojava.blogspot.com/2016/07/queEsHashMapYComoFunciona.html)

HashMap. Datojava

[**https://es.stackoverflow.com/questions/1234/cu%C3%A1l-es-la-diferencia-entre-hashmap-y-hashtable-en-java**](https://es.stackoverflow.com/questions/1234/cu%C3%A1l-es-la-diferencia-entre-hashmap-y-hashtable-en-java)

Diferencia entre HashTable y HashMap. stackoverflow