Implementación del algoritmo Dijkstra en el campus UNI

Facultad de Ciencias
Profesores:
Ronald Mas Huaman
Angel Ramirez Gutierrez
Integrantes:
Mijail Frank Poccohuanca Copacondori, *20171045C*
Cristhian Wiki Sánchez Sauñe, *20180517A*
Jorge Luis Cáceres Naupari, *201955011*
Davis Alderete Valencia, *20181139K*
Aldo Luna Bueno, *20170325B*

***Resumen*—-En este trabajo se presentará una aplicación real de la teoría de grafos. La aplicación que escogimos fue el de la optimización de rutas dentro de la Universidad Nacional de Ingeniería, la cual da solución a la siguiente pregunta: ¿cuál es la ruta más corta entre dos puntos de nuestra universidad, haciendo algunas paradas previas?. Primero, revisaremos el marco teórico en el cual se describirá brevemente un poco de la teoría de grafos y el algoritmo que usaremos para nuestra aplicación(Dijkstra). Luego, se dará detalle de la aplicación en sí, para la cual implementaremos un programa creado en Java en base al algoritmo mencionado anteriormente. Posteriormente, se mencionarán brevemente algunas aplicaciones prácticas y finalmente se darán nuestras apreciaciones y conclusiones sobre la aplicación escogida.**

I. INTRODUCCIÓN

UN mapa es una representación visual de un área completa o una parte de un área, típicamente representada en una superficie plana.

El trabajo de un mapa es ilustrar características específicas y detalladas de un área en particular, que se utiliza con mayor frecuencia para ilustrar la geografía. Hay muchos tipos de mapas; estático, bidimensional, tridimensional, etc. Los mapas intentan representar varias cosas, como límites políticos, características físicas, carreteras, topografía, población, climas, recursos naturales y actividades económicas. Un mapa es útil para cualquier persona, ya que los mapas contienen mucha información. Depende de un individuo cómo lo usa. Los mapas se usan generalmente para: análisis, confirmación, comunicación, decoración, inversión, exploración, estimulación de hipótesis, navegación, etc. Algunos mapas se crean con objetivos muy específicos en mente. El objetivo de este trabajo de investigación es crear un mapa para determinar la ruta más corta entre dos puntos de la UNI.

II. OBJETIVOS

- Presentar una aplicación real de grafos, la cual se estudió previamente en el curso.
- Dar solución a la problemática del tiempo de desplazamiento, que afecta a todos los estudiantes de a pie.
- Diseñar un modelo para la ruta más corta e implementarlo en Java.

III. FUNDAMENTOS TEÓRICOS

III-A. Algoritmo de Dijkstra:

El algoritmo de Dijkstra (o el algoritmo de Caminos Mínimos de Dijkstra, algoritmo SPF) es un algoritmo para encontrar las rutas más cortas entre los nodos en un grafo, que puede representar, por ejemplo, redes de carreteras. Fue concebido por el informático Edsger W. Dijkstra en 1956 y publicado tres años después. La idea subyacente en este algoritmo 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. Se trata de una especialización de la búsqueda de costo uniforme y, como tal, no funciona en grafos con aristas de coste negativo (al elegir siempre el nodo con distancia menor, pueden quedar excluidos de la búsqueda nodos que en próximas iteraciones bajarían el costo general del camino al pasar por una arista con costo negativo).

III-B. Algoritmo:

Llamemos al nodo con el que estamos iniciando ”nodo inicial”. Diremos que la distancia de un nodo arbitrario Y será la distancia desde el nodo inicial hacia el nodo Y. El algoritmo de Dijkstra asignará algunas distancias iniciales para todos los nodos de nuestro grafo y tratará de mejorarlas paso a paso.

- Paso 1: Marcamos a todos los nodos como no visitados. Creamos un conjunto con todos los nodos no visitados al cual llamaremos como conjunto no visitado.

- Paso 2: Asignamos a cada nodo un valor de distancia tentativo: asignamos 0 para nuestro nodo inicial e infinito para los demás nodos. Establecemos el nodo inicial como nodo actual.
- Paso 3: Para el nodo actual, consideramos todos sus vecinos no visitados y calculamos sus distancias provisionales a través del nodo actual. Comparamos la distancia recién calculada con el valor asignado actual y asignamos el más pequeño. Por ejemplo, si el nodo actual A está marcado con una distancia de 6, y la arista que lo conecta con un vecino B tiene una longitud 2, entonces la distancia de B a A será 6 + 2 = 8. Si B se marcó previamente con una distancia mayor que 8, entonces lo cambiamos con 8. De lo contrario, se mantendrá el valor actual.
- Paso 4: Cuando hayamos terminado de considerar todos los vecinos no visitados del nodo actual, marcamos este nodo como visitado y lo removemos del conjunto no visitado. Un nodo visitado no será comprobado otra vez.
- Paso 5: Si el nodo de destino se ha marcado como visitado (cuando se planifica una ruta entre dos nodos específicos), o si el conjunto no visitado es vacío, o si la distancia tentativa más pequeña entre los nodos en el conjunto no visitado es infinita (cuando se planifica un recorrido completo; ocurre cuando no hay conexión entre el nodo inicial y los nodos restantes no visitados), entonces nos detenemos. El algoritmo ha terminado.
- Paso 6: De lo contrario, seleccionamos el nodo no visitado que está marcado con la distancia tentativa más pequeña, y lo asignamos como el nuevo ”nodo actualz volvemos al paso 3.

Al planificar una ruta, en realidad no es necesario esperar hasta que el nodo de destino sea visitado como se indicó anteriormente: el algoritmo puede detenerse una vez que el nodo de destino tenga la menor distancia tentativa entre todos los nodos no visitados (y por lo tanto podría seleccionarse como el siguiente actual).

III-C. Pseudocódigo

En el siguiente algoritmo, el código u = vértice en Q con mínimo dist[u], busca el vértice u en el conjunto de vértices Q que tiene el menor valor dist[u]. long(u, v) devuelve la longitud de la arista de unión (es decir, la distancia entre) los dos nodos vecinos u y v. La variable alt en la línea 16 es la longitud de la ruta desde el nodo inicial al nodo vecino v si iba a pasar por u. Si esta ruta es más corta que la ruta más corta actual registrada para v, esa ruta actual se reemplaza con esta ruta alternativa. El vector prev[] se rellena con un puntero al nodo inmediato anterior en el grafo original para obtener la ruta más corta al vértice inicial.

```
1   Función Dijkstra(Grafo, vértice inicial)
2    crea el conjunto de vértices Q
3
4    para cada vértice v del grafo:
5        dist[v] = INFINITO
6        prev[v] = INDEFINIDO
7        añade v a Q
8   dist[vertice inicial] = 0
9
10   mientras Q es no vacío:
11       u=vértice en Q con mínimo dist[u]
12
13       eliminar u de Q
14
15       para cada vecino v de u que está en Q:
16           alt = dist[u]+long(u,v)
17           si alt < dist[v]
18               dist[v] = alt
19               prev[v] = u
20   retornar dist[], prev[]
```

Si solo estamos interesados en el camino más corto entre los vértices de origen y destino, podemos terminar la búsqueda después de la línea 13 si u = objetivo. Ahora podemos leer la ruta más corta desde el origen hasta el destino mediante iteración inversa:

```
1   S = secuencia vacía
2   u = vértice de destino
3   si prev[u] es definido o u = vértice de origen
4       mientras u es definido:
5           //Construye la ruta más corta
6           inserta u en el inicio de S
7           //Push en la pila S
8           u = prev[u]
9           //Recorre hasta el vértice inicial
```

Ahora la secuencia S es la lista de vértices que constituyen una de las rutas más cortas desde el origen hasta el destino, o la secuencia vacía si no existe una ruta.

Un problema más general sería encontrar todas las rutas más cortas entre el origen y el destino (puede haber varios diferentes de la misma longitud). Luego, en lugar de almacenar solo un nodo en cada entrada de prev[], almacenaríamos todos los nodos que satisfagan la condición de minimalidad. Por ejemplo, si tanto el nodo A como el nodo B se conectan al nodo objetivo y ambos se encuentran en diferentes caminos más cortos a través del nodo objetivo (porque el costo del camino en ambos casos es el mismo), entonces agregaríamos tanto A como B a prev[nodo objetivo]. Cuando se completa el algoritmo, la estructura de datos prev[] en realidad describirá un grafo que es un subconjunto del grafo original con algunos bordes eliminados. Su propiedad clave será que, si el algoritmo se ejecutó con algún nodo inicial, entonces cada ruta desde ese nodo a cualquier otro nodo en el nuevo grafo será la ruta más corta entre esos nodos en el grafo original, y todas las rutas de esa longitud desde el grafo original estará presente en el nuevo grafo. Luego, para encontrar realmente todas estas rutas más cortas entre dos nodos dados, usaríamos un algoritmo de búsqueda de rutas en el nuevo grafo, como la DFS (Deep First Search, algoritmo de búsqueda en profundidad).

III-D. Prueba de la validez del algoritmo

La prueba del algoritmo de Dijkstra se construye por inducción en el número de nodos visitados.

Llamemos s al nodo inicial.

Hipótesis inductiva: Para cada nodo visitado v, dist[v] se considera la distancia más corta desde s hasta v; y para cada nodo no visitado u, se supone que dist[u] es la distancia más corta cuando viaja a través de nodos visitados únicamente, desde s hasta u. Esta suposición solo se considera si existe una ruta; de lo contrario, la distancia se establece en infinito. (Nota: no asumimos que dist[u] es la distancia más corta real para los nodos no visitados).

Caso base: es cuando solo hay un nodo visitado, que es el nodo inicial s, en cuyo caso la hipótesis es trivial.

Paso inductivo: suponga la hipótesis de los nodos visitados n-1 (cumple para n-1). En ese caso, elegimos una arista vu donde u tiene la menor dist [u] de los nodos no visitados (v es un nodo ya visitado vecino a u que no fue visitado) y la arista vu es tal que dist [u] = dist [v] + longitud [v, u]. dist [u] se considera la distancia más corta desde s hacia u porque si hubiera una ruta más corta, y si w fuera el primer nodo no visitado en esa ruta, entonces según la hipótesis original dist [w] $>$ dist [u] que crea una contradicción, ya que el camino ya no es menor a dist[u]. Del mismo modo, si hubiera una ruta más corta hacia u sin usar nodos no visitados, y si el penúltimo nodo de esa ruta fuera w, entonces habríamos tenido dist[u] = dist [w] + longitud [w, u], que también es una contradicción porque no es menor a dist[u] definido anteriormente.

Después de procesar u (al marcarlo como visitado), seguirá siendo cierto que para cada nodo no visitado w, dist [w] será la distancia más corta desde la fuente hasta w utilizando solo los nodos visitados, porque si hubiera un camino más corto que no pasa por u tendríamos lo que encontramos anteriormente, y si hubiera una ruta más corta usando u la habríamos actualizado al procesar u. Lo cual significa que cumple para n nodos visitados a partir de n-1 nodos visitados por lo tanto la prueba estaría completa.

III-E. Ejemplo del Algoritmo de Dijkstra

Se tiene el siguiente grafo para aplicar el algoritmo de Dijkstra:

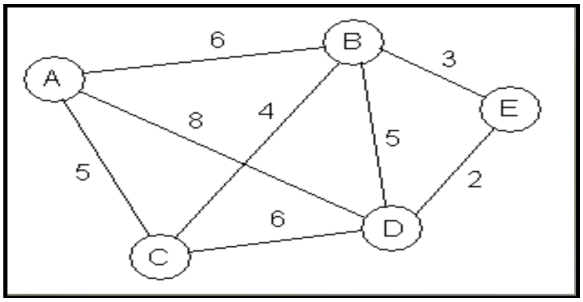


Figura 1: Grafo a aplicar

Iteración 1: Se escoge el nodo inicial, en este caso el nodo A, a continuación, se marca en el nodo la distancia desde el nodo anterior, pero como no lo hay se deja como nulo.

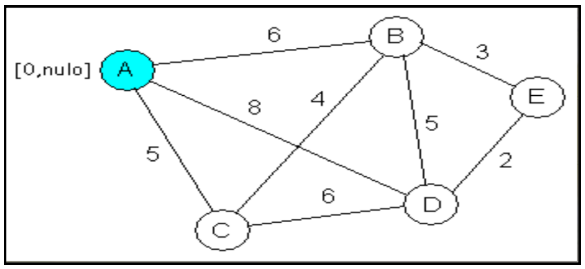


Figura 2: Iteración 1

Iteración 2: De los nodos adyacentes de A, se marca el peso acumulado junto con el nodo antecesor, es decir A.

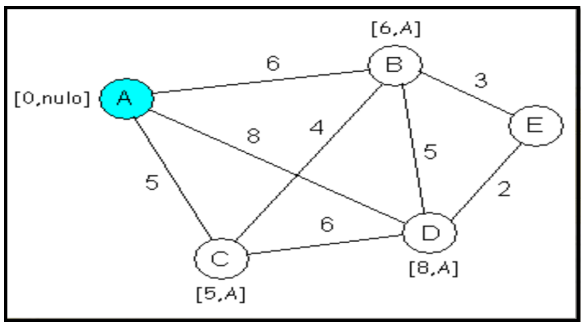


Figura 3: Iteración 2

Iteración 3: De los nodos ya visitados se escoge nodo de menos peso acumulado, en este caso, el nodo C.

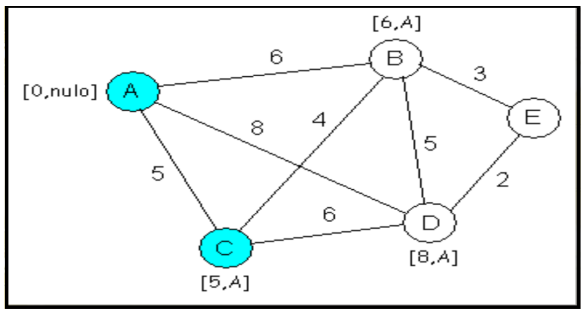


Figura 4: Iteración 3

Iteración 4: Se toman los nodos adyacentes a C que no estén marcados de azul y se calculan sus pesos acumulados, sumando el que ya se tiene con el peso de la siguiente ruta, entonces:

- Para B: 5 + 4 = 9. Como el nodo B ya poseía un peso acumulado de 6, no se modifica porque 6 es menor que 9.
- Para D: 5 + 6 = 11. El nodo C ya poseía un peso de 8, por lo tanto, tampoco se modifica.

El grafo no sufre cambios.

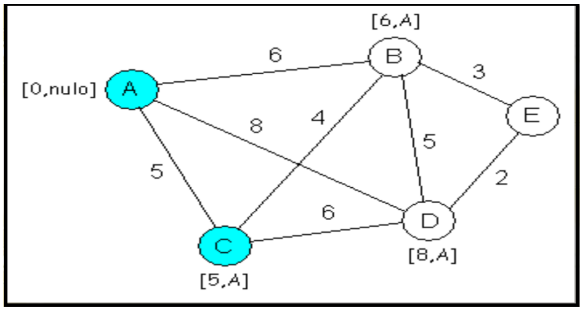


Figura 5: Iteración 4

Iteración 5: De los nodos visitados y no marcados, se busca el de menor peso acumulado, en este caso el nodo B.

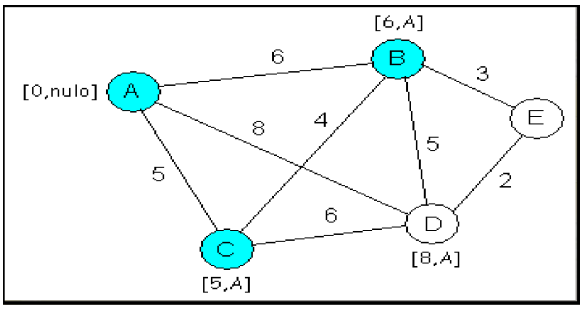


Figura 6: Iteración 5

Iteración 6: De los nodos adyacentes de B, se marca el peso acumulado junto al nodo antecesor (nodo B), entonces:

- Para E: 6 + 3 = 9, se pone 9 en E ya 9 es menor que infinito.
- Para D: 6 + 5 = 11, para D ya poseía un 8, por lo tanto, no sufre ningún cambio.

El grafo queda entonces así:

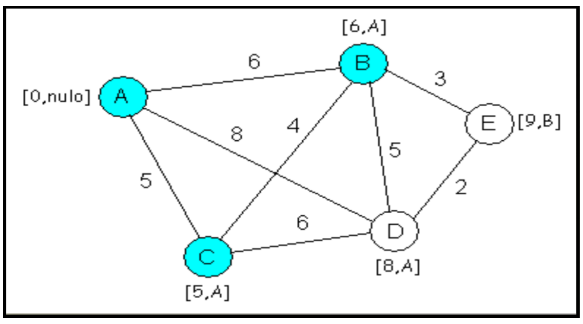


Figura 7: Iteración 6

Iteración 7: De los nodos ya visitados y no marcados de azul, se toma el de menor peso, en este caso el D.

Iteración 8: De los nodos adyacentes de D que no están marcados se calculan los pesos acumulados.

- Para E: 8 + 2 = 10. Como E poseía un peso de 9, no se modifica.

El grafo no tiene cambios.

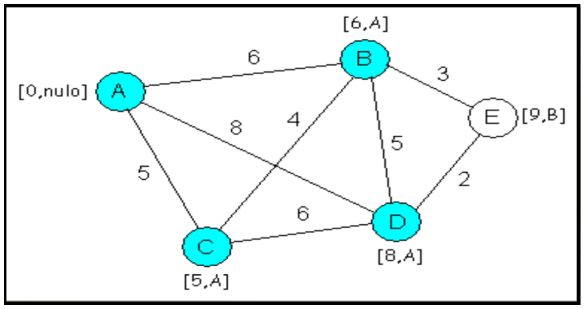


Figura 8: Iteración 7

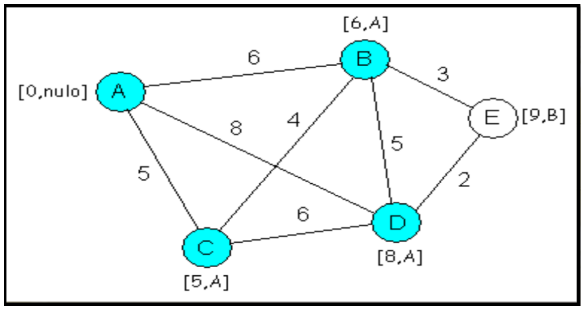


Figura 9: Iteración 8

Iteración 9: Como queda solo un nodo por seleccionar, se da por terminado el algoritmo con el que ya se puede determinar la ruta más corta del nodo de inicio (nodo A) a cualquier otro nodo.

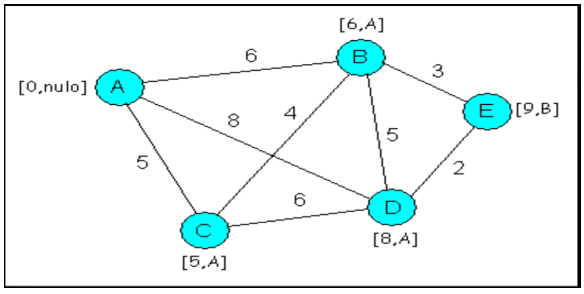


Figura 10: Iteración 9

Ruta más corta del nodo A al nodo E: Teniendo en cuenta la solución, se marca el camino de menor distancia tomando como guía el nodo antecesor, hasta llegar al destino requerido. En este caso el camino seria A, B, E.

IV. APLICACIONES GENERALES

La teoría de grafos tiene muchas aplicaciones en la vida real. A continuación, mencionaremos algunas de ellas:

- En la modelación de trayectos como el de los que siguen los buses por las calles de una ciudad. Aquí se pueden obtener caminos óptimos aplicando diferentes algoritmos como por ejemplo, el algoritmo de Floyd.
- En la electrónica, para la síntesis de circuitos secuenciales, contadores o sistemas de apertura.

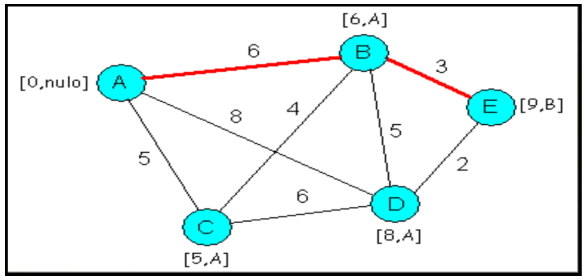


Figura 11: Camino más corto de A hacia E

- En el control de producción, para proyectar redes de ordenadores, para diseñar módulos electrónicos y proyectar sistemas físicos con parámetros localizados (mecánicos, acústicos y eléctricos).
- Mediante las redes sociales, determinado preveedor de un servicio tendrá a su disposición, publicidad más específica para cada usuario. Un ejemplo es Facebook, que nos sugiere productos en función de las páginas que visitamos. Estas relaciones se construyen apartir de grafos, sobre ingentes cantidades de datos de casi todas personas alrededor del globo.

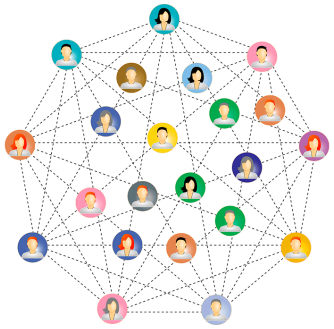


Figura 12: Búsqueda de relaciones en las redes sociales

- Existen muchas otras aplicaciones en diversas áreas de la ciencia que se muestran a continuación.

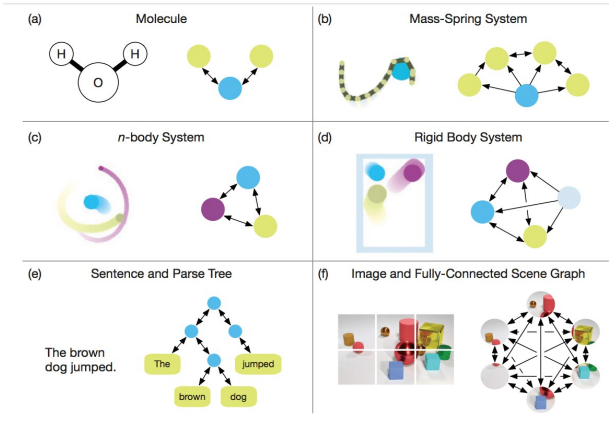


Figura 13: (a) Representación de una molécula (b) Un sistema masa-resorte (c) Un sistema de n-cuerpos (d) Un sistema de cuerpo rígido (e) Búsqueda semántica entre palabras. (f) Segmentación de una imagen.

V. APLICACIÓN ESCOGIDA

La aplicación escogida fue la optimización de rutas dentro de la Universidad de Nacional de Ingeniería.

V-A. Planteamiento del problema y justificación

Dentro de la UNI, existen muchas rutas para de ir de un lugar a otro. La vida del estudiante universitario demanda que este se movilice a través de toda la universidad, y no solo entre la puerta de la universidad y su salón de clases. Esto se da por las diversas actividades que el estudiante realiza como deportes, charlas, talleres, conferencias, cursos extras, etcétera. Por tal motivo, el estudiante universitario está constantemente respondiendo a la siguiente pregunta: ¿cuál es la ruta más corta entre dos puntos de nuestra universidad?. Si bien los estudiantes realizan la ruta más corta intuitivamente, no tienen la certeza de que esa sea la ruta más corta. Resolviendo este problema, el estudiante podría ganar unos minutos al día y a la larga, en un año, esos minutos se convertirían en horas.

V-B. Solución del problema

Para dar solución a este problema, se empleará el algoritmo de Dijkstra, explicado en los fundamentos teóricos, y Google Maps. El programa está estructurado de la siguiente forma:

V-B1. Obtención de las coordenadas dentro de la UNI: De forma manual, seleccionamos las coordenadas por las que queremos desplazarnos alrededor de la universidad, esto debido que no hay una base de datos como tal, requerido para los fines de este trabajo. Las coordenadas las obtenemos en la versión GoogleMaps de escritorio. Añadimos estas coordenadas en un archivo de texto, siguiendo el siguiente orden por cada línea:

[Latitud inicial] [Longitud inicial] [Latitud final] [Longitud final] "[nombre de la via]"[tipo de via]

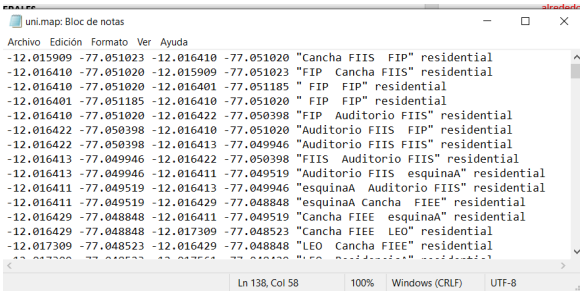


Figura 14: Coordenadas dentro de un .txt

Y para nuestros fines hemos usado 2 tipos de vías, una vía peatonal especialmente diseñada para los estudiantes, y otro vehicular. La primera, por ser peatonal, le asignamos un peso positivo mayor al segundo, debido a que el algoritmo no buscará usar estas vías, debido a que son lentas, y lo que buscamos en sí, es la forma de llegar más rápidamente, del punto A al punto B, lo que implica implica tomar las vías más rápidas. Consideramos una velocidad estándar de 15 km/h para los vehículos, y de 6.5km/h para los estudiantes de pie.

V-B2. Generador gráfico de coordenadas: Estas coordenadas posteriormente se van a cargar en una interfaz que implementamos, y debido a que la mayoría de nosotros (estudiantes de C.C.) tiene conocimientos en Programación Orientada a Objetos, lo programamos en Java (IDE Eclipse). Y como estos datos se obtuvieron de GoogleMaps, que mejor idea que la montarla sobre la API que proporciona Google para fines educativos(API JAVASCRIPT GOOGLEMAPS). Las coordenadas se muestran como puntos azules en el mapa.

Figura 15: Coordenadas en el mapa (en azul)

V-B3. Creación del grafo: En nuestro grafo los nodos serán los principales lugares dentro de la universidad, mientras las aristas serán las distintas vias que las unen, ahora el peso de una arista (u,v) será el tiempo que le tomaría a un vehículo o peatón en desplazarse desde u hasta v en el mapa.Todo esto teniendo en cuenta la velocidad de cada vía.

En la práctica solo usaremosa una velocidad límite para todas la rutas(6.5 km/h), pues la idea es que solo la implementen estudiantes sin coche.

V-B4. Coloración de grafos: Llegado a este punto implementamos un método que coloree la unión entre dos puntos (un grafo), este será de color negro como se muestra a continuación:

Figura 16: Grafo que une el pabellón R y la biblioteca FC

V-B5. Cálculo de la ruta más corta: Llegado aquí, implementaremos el algoritmo de Dijkstra, mencionado en los puntos anteriores, para 2 puntos cualesquiera del mapa. Teniendo como prioridad las vías vehiculares, sobre las peatonales. Tras muchas iteraciones podemos obtener esto:

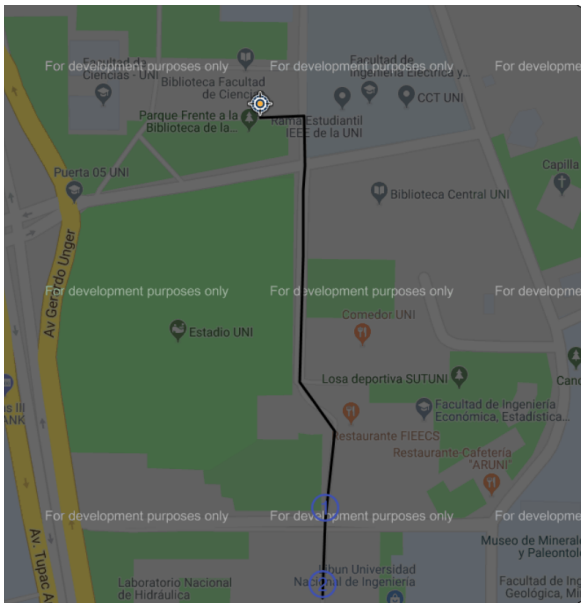


Figura 17: Ruta más corta obtenida para llegar de la biblioteca FC hasta el pabellón J.

V-B6. Cálculo del tiempo y la distancia: Como añadido del programa, calcularemos la distancia recorrida para unir ambos puntos, así como el tiempo que implementaremos, estas son unas aproximaciones, pues habrá factores que la afecten (construcciones, por ejemplo).

Para hallar esto usaremos las coordenadas halladas anteriormente usando la siguiente formula:

$$D = 2 * R * \operatorname{arc\,sen}\sqrt{\operatorname{sen}^2\frac{\Delta lat}{2} + \cos lat1 * \cos lat2 * \operatorname{sen}^2\frac{\Delta lon}{2}} \quad (1)$$

$lat \rightarrow latitud$
$lon \rightarrow longitud$
$\Delta lat = lat2 - lat1$
$\Delta lon = lon2 - lon1$
$R = 6372{,}795477598 Km (Radiodelatierra)$

```
private double getDist(double lat1, double lon1, double lat2, double lon2)
{
    int R = 6373; // radio de la tierra en kilómetros
    double lat1rad = Math.toRadians(lat1);
    double lat2rad = Math.toRadians(lat2);
    double deltaLat = Math.toRadians(lat2-lat1);
    double deltaLon = Math.toRadians(lon2-lon1);

    double a = Math.sin(deltaLat/2) * Math.sin(deltaLat/2) +
            Math.cos(lat1rad) * Math.cos(lat2rad) *
            Math.sin(deltaLon/2) * Math.sin(deltaLon/2);
    double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));

    double d = R * c;
    return d;
}
```

Figura 18: Código en Java para hallar la distancia

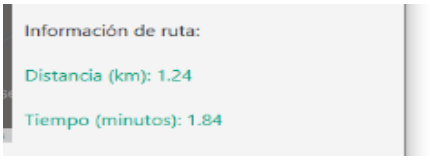


Figura 19: Visualización en la ventana

V-B7. Visualización del trabajo del algoritmo: ¿Pero cómo se obtuvo la ruta mostrada arriba?

Como nos interesa saber el funcionamiento de búsqueda en si crearemos un método que muestre en tiempo real como estuvo funcionando el algoritmo hasta antes de mostrar la ruta final más corta. Dicho proceso se mostrará con vértices de color rojo, uno apareciendo seguidamente de otro, hasta que converja la ruta.

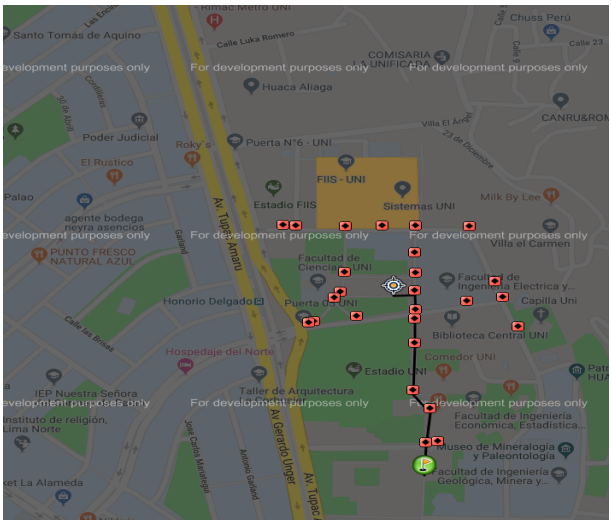


Figura 20: 30 posibles rutas que pudo tomar el algoritmo, antes de encontrar la que minimice el coste (tiempo).

V-B8. Resultados: Finalmente decidimos montar todo esto en una interfaz que sea amigable con el usuario, y monte todas las características mencionadas. A continuación, haremos un breve recorrido por la interfaz.

Figura 21: Panel de carga, con información del título del proyecto, y los integrantes

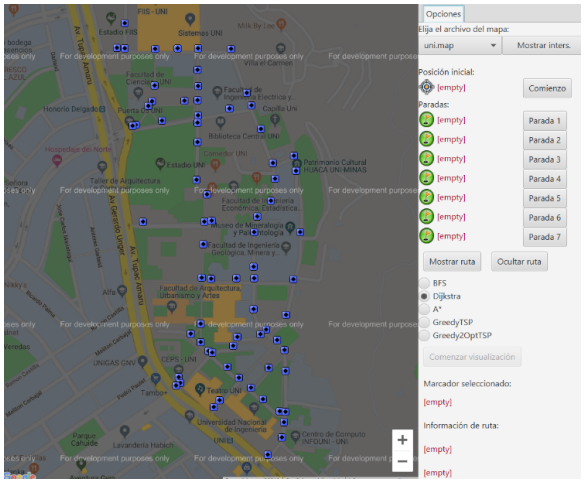


Figura 22: Panel general de control y visualización

Figura 23: Pestaña Opciones

En **Opciones**, elegimos la ruta inicial, y podemos elegir hasta 7 paradas. Para hacer esto, primero seleccionamos la coordenada en azul, y posteriormente hacemos clic en la **Parada n** que seleccionemos al azar. Y si queremos buscar la ruta, simplemente hacemos clic en **Mostrar ruta**. Si queremos seleccionar otras paradas, simplemente hacemos clic en **Ocultar ruta** y elegimos a nuestro albedrío. Finalmente, si queremos visualizar el funcionamiento detrás del algoritmo, seleccionamos **Visualización**, y se nos mostrará los vértices posibles en color rojo.

Esta aplicación se desarrollo teniendo en cuenta la geografía de nuestra universidad, así como la velocidad en las diferentes vías. Si se quiere pulir aún más este trabajo, se pueden añadir más vértices manualmente, así como añadir un rango de velocidades más variado para diferentes circunstancias. Si el lector es de otra casa de estudios, puede implementarlo manualmente en su universidad, ya que no se cuenta con el **.txt** de datos en internet. También como nota adicional, el archivo no se puede exportar como un **JAR ejecutable**, debido a problemas con la API de Google, al parecer existe una incompatibilidad de servicios o permisos. Si usted es un estudiante de la UNI, puede mejorar este trabajo a su voluntad, toda la documentación se proporciona en las referencias, así como el enlace al **GITHUB**.

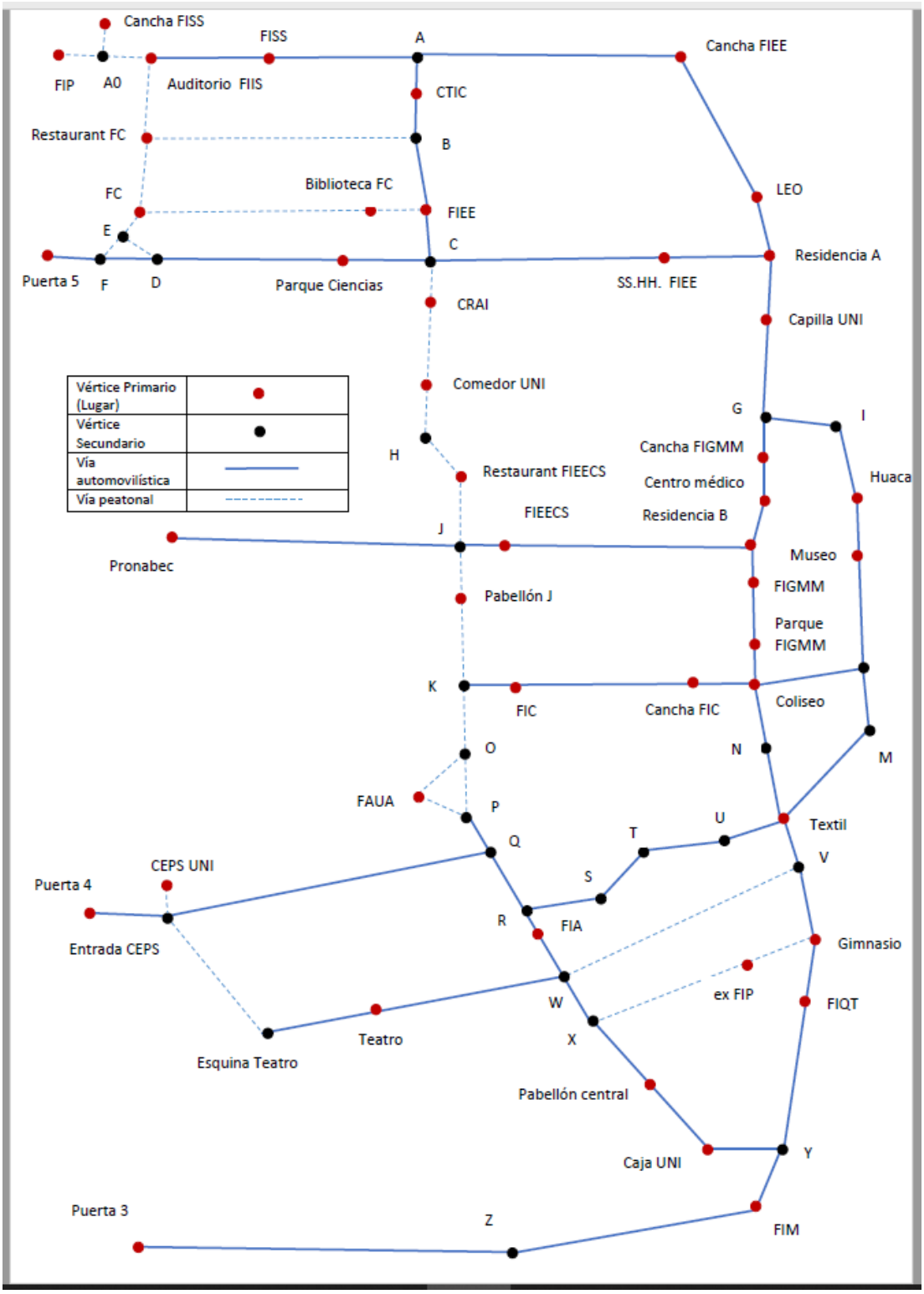
VI. CONCLUSIONES

- Se presentó una aplicación real de la teoría de grafos, la cual puede ser utilizada por cualquier estudiante de la universidad para ahorrar tiempo al trasladarse por la universidad.
- Al ser una aplicación en fase de prueba, presenta errores de precisión en el tiempo. Hace falta un estudio minucioso de las velocidades en las diferentes vías de la UNI.
- Queda a posteriori, la implementación en la plataforma Android, de manera que cualquier estudiante con su Smartphone pueda hacer uso de ella.
- Dijkstra nos brinda la posibilidad de ir de A hasta D, pasando por B y C de la mejor manera posible. Aunque tiene limitantes, tales como factores externos que hacen que cada grafo tenga pesos que varían con el tiempo (debido al tráfico a cierta hora del día, por ejemplo), para lo cual el algoritmo no está preparado en sí, y que también está lejos de este estudio.
- El trabajo de investigación fue de enriquecedor, pues fomentó el desarrollo de nuestras capacidades como programadores, y también nos hizo investigar sobre muchos otros algoritmos optimizadores de árboles, ajenos a los que vimos en clase, tales como Kruskal.

VII. REFERENCIAS

- Matousek, J., Nesetril, J. (2008). Ïnvitación a la matemática discreta". Barcelona: Reverté.
- Mehlhorn, Kurt; Sanders, Peter (2008). Çhapter 10. Shortest Paths". Algorithms and Data Structures: The Basic Toolbox. Springer.
- Dijkstra, E. W. (1959). .ª note on two problems in connexion with graphs". Numerische Mathematik.
- Gass, Saul; Fu, Michael (2013). "Dijkstra's Algorithm". Encyclopedia of Operations Research and Management Science.
- Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001). "Section 24.3: Dijkstra's algorithm". Introduction to Algorithms (Second ed.). MIT Press and McGraw–Hill.
- Sánchez Sauñe, C. (2019). HiroForYou/UNIDIJKSTRA. [en línea] GitHub. Disponible en: https://github.com/HiroForYou/UNIDIJKSTRA [Accessado 28 Nov. 2019].

ANEXO



Grafo del campus universitario