Universidad Politécnica
de Madrid

Escuela Técnica Superior de
Ingenieros Informáticos

Grado en Matemáticas e Informática

Trabajo Fin de Grado

Aplicación de Algoritmos de Búsqueda en la Optimización de Caminos Mínimos en Grafos de Decisión

Autor: Sergio González de la Lama

Tutor: Vicente Martínez Orga

Madrid, Mayo 2024

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Matemáticas e Informática
Título: Aplicación de Algoritmos de Búsqueda en la Optimización de Caminos Mínimos en Grafos de Decisión
Mayo, 2024

Autor: Sergio González de la Lama

Tutor:
Vicente Martínez Orga
Departamento de Inteligencia Artificial
ETSI Informáticos
Universidad Politécnica de Madrid

Agradecimientos

En primer lugar, quiero agradecer a mi tutor Vicente por guiarme y mostrar tanta atención como ha mostrado, tanto cuando estaba en Suecia como en el desarrollo del trabajo en Madrid. También al tribunal por evaluarme de forma objetiva.

En segundo lugar, gracias a mis amigos tanto de la universidad como del colegio. Habéis sido fundamentales para que pueda rendir al máximo en estos últimos años y poder conseguir todo lo que me he propuesto. Coca Junior será el mejor equipo dentro y fuera del campo.

También quiero agradecer a mis tíos y abuelos por estar siempre apoyándome durante mi educación universitaria, tanto en los buenos como en los malos momentos. Gracias por vivir mis alegrías y mis penas como si fuesen vuestras. Y sobre todo a ti, Nana, que confiaste en mí siempre que lo necesité.

No me olvido tampoco de Marta, quién ha sido una de las personas más importantes de mi vida y a la que le debo gran parte de mi estabilidad emocional. Nunca hubiese imaginado que crecería tanto personalmente contigo a mi lado.

Por último, quiero agradecer a mis padres y mi hermana por haberme educado como soy. Mamá, papá, os debo todo lo que soy y todo lo que seré, gracias por darme la vida que tengo y todo lo que consiga será por vosotros. Lidia, espero tenerte a mi lado toda mi vida, eres un pilar fundamental para mí y sin ti todo esto no sería posible.

Resumen

Este trabajo estudia la mejor manera de encontrar el camino óptimo entre dos nodos específicos en un grafo. Para ello, se va a utilizar el algoritmo A* (A Estrella), un método de búsqueda "mejor primero" que selecciona nodos basándose en una función coste y otra heurística para determinar el camino mínimo de un origen a un destino.

Este algoritmo se ha desarrollado en una versión adaptada del mapa del metro de Estocolmo, permitiendo a los usuarios elegir su punto de partida y destino para calcular la ruta óptima entre ambas estaciones. La implementación se ha realizado en Python, un lenguaje adecuado para el manejo de estructuras de datos complejas como los grafos, destacando por su versatilidad y la disponibilidad de bibliotecas especializadas.

Para facilitar la interacción del usuario con el sistema, se ha desarrollado una interfaz web usando HTML, que es accesible y comprensible para usuarios de diversos niveles técnicos. Esta interfaz, que también integra CSS y Python, permite a los usuarios seleccionar estaciones mediante desplegables, proporcionando una experiencia de usuario fluida y eficiente.

La aplicación de este algoritmo ayudará a optimizar las rutas de un origen a un destino en cuanto a tiempo y distancia, mejorando la experiencia del usuario y fomentando el uso del transporte público. Esto será una opción más sostenible con un menor impacto en el medio ambiente.

Abstract

This work studies the best way to find the optimal path between two specific nodes in a graph. For this purpose, the A* (A Star) algorithm will be used, a "best-first" search method that selects nodes based on a cost function and a heuristic function to determine the shortest path from a source to a destination.

This algorithm has been developed using an adapted version of the Stockholm metro map, allowing users to choose their starting point and destination to calculate the optimal route between the two stations. The implementation has been carried out in Python, a language suitable for handling complex data structures like graphs, known for its versatility and the availability of specialized libraries.

To facilitate user interaction with the system, a web interface has been developed using HTML, which is accessible and understandable to users of various technical levels. This interface, which also integrates CSS and Python, allows users to select stations through dropdown menus, providing a smooth and efficient user experience.

The application of this algorithm will help optimize routes from a source to a destination in terms of time and distance, improving the user experience and encouraging the use of public transportation. This will be a more sustainable option with a lower environmental impact.

Tabla de contenidos

1 Introducción

Durante años, las tecnologías enfocadas a encontrar un camino óptimo entre dos nodos distintos de un grafo ponderado han sido muy estudiadas para encontrar las mejores soluciones. Desde la básica búsqueda en profundidad o en anchura hasta el popular algoritmo de Dijkstra, todos tienen ciertas propiedades que las hacen realmente eficaces para el problema. Sin embargo, hay un algoritmo que funciona mejor y más rápido que los demás, el algoritmo A estrella (A*).

El algoritmo A* es un algoritmo de búsqueda del tipo "mejor primero". Este algoritmo se diferencia de otros por su uso de una heurística, que será crucial para poder encontrar los mejores nodos de un grafo evaluando todas nodos posibles. La elección de esta heurística será el factor que priorice la elección de los componentes del camino óptimo. El funcionamiento de este algoritmo y su importancia en la optimización de las rutas será explicado más adelante en el documento.

Para poder visualizar el funcionamiento del A*, se ha elegido adaptar el mapa de Estocolmo de transporte público, usando tanto líneas de metro como líneas de tren. Gracias a este mapa, un usuario podrá elegir una parada origen y otra parada destino y la aplicación enseñará cual es la ruta mínima entre ambos nodos, asemejándose a cualquier aplicación de navegación actual.

Ilustración 1. Stadion - Parada de metro de Estocolmo

La implementación de esta aplicación se ha llevado a cabo en Python, un lenguaje de programación particularmente adecuado para el manejo de estructuras de datos complejas como los grafos, gracias a su versatilidad y la amplia disponibilidad de bibliotecas especializadas que facilitan el trabajo con este tipo de estructuras.

La elección de Python no solo se debe a su eficacia en el manejo de grafos, sino también a su simplicidad sintáctica y la eficiencia en el tiempo de desarrollo, lo que permite centrarse más en la lógica del algoritmo y menos en detalles de bajo nivel. La implementación en Python, combinada con el algoritmo A*, proporciona una solución robusta y eficiente para el problema de navegación y búsqueda de rutas óptimas en redes de transporte complejas, como es el caso del sistema de metro de una gran ciudad.

Para poder desarrollar una aplicación que pueda ser intuitiva para el usuario y fácil de manejar para que ocurran el menor número de errores, se ha elegido construir una página web con el lenguaje HTML.

La elección de HTML como la base para la implementación de esta página web se debe a varias razones clave. Primero, HTML es la opción más popular para la creación de contenido web y permite una estructura clara y bien definida de los elementos en la página, lo que facilita que los usuarios naveguen intuitivamente por la interfaz. Además, HTML es altamente compatible con una amplia gama de dispositivos y navegadores, asegurando que la aplicación sea accesible para una audiencia amplia sin requerir software especializado. Por último, el hecho de que sea tan popular significa que hay más documentación disponible para incluir elementos más originales y funcionalidades atractivas.

En la interfaz, los usuarios pueden fácilmente seleccionar su estación de origen y destino a través de menús desplegables o escribiéndolo. Esta interactividad se logra combinando HTML con CSS para el diseño. Esta combinación no solo mejora la estética de la página, sino que también contribuye a una experiencia de usuario más agradable y eficiente.

Al proporcionar una plataforma que es tanto visualmente atractiva como fácil de usar, se fomenta una mayor interacción y satisfacción del usuario, lo que es fundamental en aplicaciones de planificación de rutas y navegación. Esta interfaz web sirve como un puente eficaz entre la complejidad técnica del algoritmo de búsqueda de caminos y la simplicidad requerida por los usuarios finales para una experiencia de usuario óptima.

1.1 Alcance y objetivos

Uno de los propósitos principales de este estudio se centra en el área de la planificación urbana y la administración del transporte público. Al ofrecer un sistema efectivo para encontrar las mejores rutas en el metro de Estocolmo, este proyecto no solo mejora la experiencia del usuario, sino que también proporciona información útil para los urbanistas. Los beneficios de una App de navegación son infinitos: desde modificar el mapa de transporte público en zonas con alta afluencia de personas hasta ubicar con facilidad tu destino en lugares desconocidos o descubrir nuevas ciudades sin haberlas visitado nunca antes.

Al hacer más fácil viajar por el metro y planificar rutas, este proyecto ayuda a mejorar la movilidad en la ciudad al reducir los tiempos de desplazamiento y hacer más accesibles los servicios de transporte. Esto es especialmente crucial en una ciudad con una alta densidad de población donde la eficacia del transporte público es vital para el día a día de los residentes. La interfaz gráfica creada garantiza que personas de cualquier edad y nivel de habilidad tecnológica puedan aprovechar la herramienta, fomentando una mayor inclusión en la comunidad urbana.

Técnicamente hablando, este proyecto también influye significativamente en el campo de la tecnología de la información al mostrar cómo se pueden utilizar algoritmos avanzados de búsqueda y heurísticas en situaciones reales. La implementación de la interfaz de usuario en HTML y su integración con sistemas backend complejos es un gran ejemplo de cómo se pueden diseñar soluciones tecnológicas que sean potentes y accesibles para el usuario final.

Por último, la estructura modular y flexible de este proyecto permite su sencilla adaptación o expansión a diferentes ciudades y sistemas de transporte. Esto ofrece un potencial considerable para futuras investigaciones y desarrollos de aplicaciones de navegación.

Por ende, este proyecto no solo logra el propósito inmediato de mejorar la navegación en el metro de Estocolmo, sino que también sienta las bases para investigaciones y aplicaciones futuras en áreas vinculadas a la tecnología y la planificación urbana. También ofrece una base sólida para investigar otras posibles mejoras y avances en el campo de la inteligencia artificial y la interacción persona-computadora.

1.2 Estado del arte

Actualmente, ya existe el concepto de recorrer un grafo (ponderado o no) para encontrar el camino mínimo siguiendo distintos algoritmos de búsqueda. Además, también podemos estar familiarizados con las aplicaciones de navegación que nos proporcionan un camino entre un origen y un destino.

Por todo ello, se va a analizar el contexto existente para poder desarrollar el trabajo.

1.2.1 Algoritmos de búsqueda

Como indica Jahaziel Ponce [13], "un algoritmo de búsqueda es un conjunto de instrucciones que están diseñadas para localizar un elemento con ciertas propiedades dentro de una estructura de datos". Por ejemplo, podemos buscar el mejor movimiento en una partida de ajedrez o el elemento más pequeño dentro de una colección de elementos.

Para poder entender mejor el concepto, se necesita clasificar estos algoritmos en tres grandes grupos: búsqueda secuencial (lineal), búsqueda binaria y búsqueda de caminos mínimos.

1.2.1.1 Búsqueda secuencial

La búsqueda lineal o secuencial consiste en [14] recorrer un vector/array desde el primer elemento hasta el último y comprobar si alguno de estos elementos contiene el valor buscado.

En este tipo de algoritmos, no hace falta que el vector este ordenado de una forma particular, por lo que encontrar el valor puede ser en la primera iteración o en la última independientemente de su valor. Este será su mayor punto desfavorable ya que la complejidad de la búsqueda puede llegar a ser muy alta si el vector es de gran tamaño.

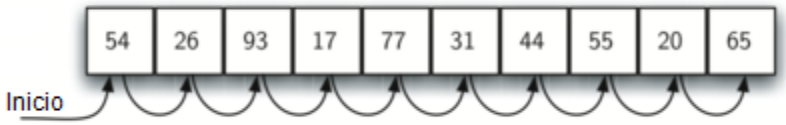


Ilustración 2. Búsqueda secuencial.

1.2.1.2 Búsqueda binaria

Este algoritmo [14] funciona efectivamente en vectores que, a diferencia del secuencial, están ordenados. Este algoritmo funciona eliminando la mitad de los elementos en cada iteración: comparará el valor buscado con el primer tramo del vector ordenado, y si está dentro se elimina la otra mitad del array. Si no se encuentra en esta mitad, realizará lo propio con la otra mitad. Este proceso lo iterará tantas veces como necesite hasta encontrar el valor. Sin embargo, necesitará menos iteraciones que el algoritmo secuencial, aunque necesitará que el vector este ordenador de alguna forma.

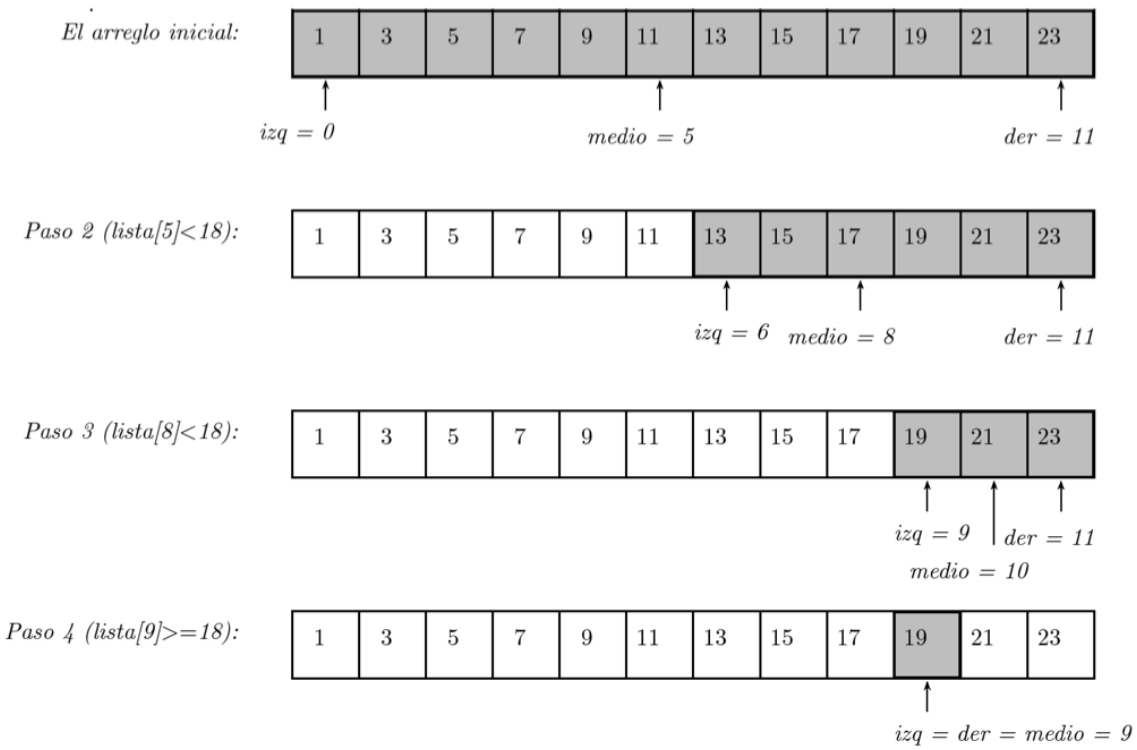


Ilustración 3. Búsqueda binaria.

Como se puede observar en la Ilustración 3, el valor a encontrar es el 19. En el paso 2 eliminará todos los elementos menores o iguales que 11 (marcado como elemento central del vector) y trabajará con los demás valores, y así sucesivamente hasta encontrarlo en el paso cuatro.

En caso de haber trabajado con el algoritmo secuencial, en vez de cuatro iteraciones se hubiese requerido diez iteraciones, por lo que se puede concluir que, en vectores ordenador, la búsqueda binaria es más eficiente.

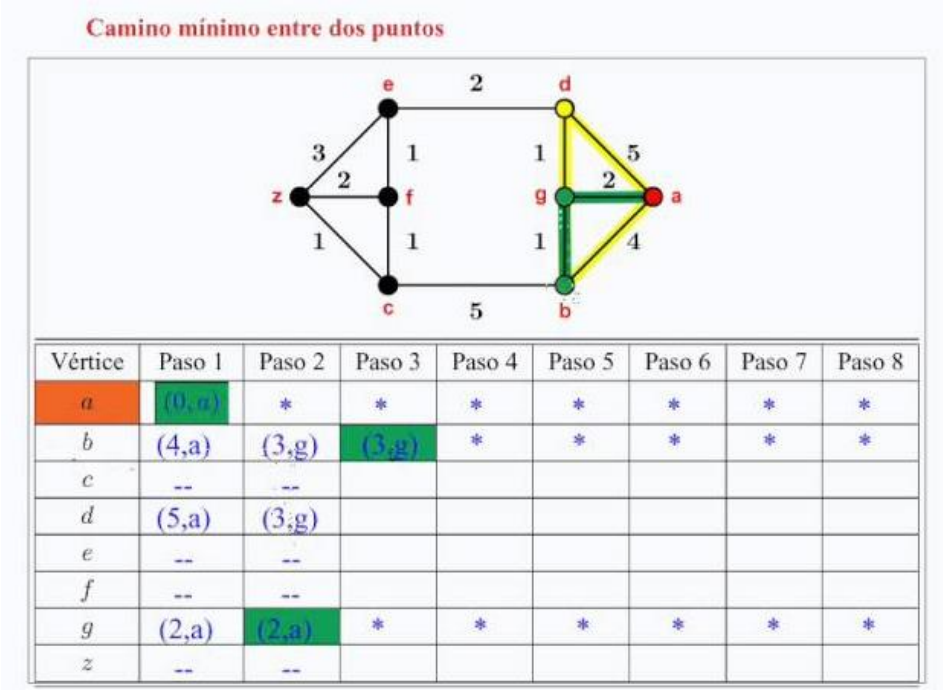
1.2.1.3 Algoritmos de búsqueda de caminos mínimos

Otro tipo de algoritmos de búsqueda son los dedicados a los grafos más complejos y por ello no se limitan a vectores lineales.

En el contexto de un problema de búsqueda de caminos óptimos en grafos ponderados con numerosos nodos, donde los pesos de las aristas representan la longitud de las vías que unen cada parada de metro, es esencial seleccionar un algoritmo eficiente y escalable. Los anteriores algoritmos de búsqueda lineales no serían realmente útiles para ello.

Existen algoritmos distintos para el problema planteado:

- Algoritmo de Dijkstra: [1] la idea de este algoritmo es la determinación del camino más corto dado un vértice origen al resto de los vértices en un grafo con pesos en cada arista. Utiliza una técnica de búsqueda en anchura para encontrar el camino más corto desde un vértice origen hacia todos los demás vértices en un grafo con aristas de peso no negativo. Su complejidad es generalmente $O(n^2)$ [4], donde n es el número de nodos, pero puede mejorar con el uso de una cola de prioridad binaria a $O((n + m) \log n)$, donde m es el número de aristas. Este algoritmo es muy empleado en sistemas de navegación y aplicaciones de mapeo GPS debido a su robustez y precisión en la determinación de rutas más cortas.



Vértice	Paso 1	Paso 2	Paso 3	Paso 4	Paso 5	Paso 6	Paso 7	Paso 8
a	(0,a)	*	*	*	*	*	*	*
b	(4,a)	(3,g)	(3,g)	*	*	*	*	*
c	--	--						
d	(5,a)	(3,g)						
e	--	--						
f	--	--						
g	(2,a)	(2,a)	*	*	*	*	*	*
z	--	--						

Ilustración 4. Algoritmo de Dijkstra

Su funcionamiento se basa en [15] ir explorando cada nodo desde A evaluando las distancias entre los nodos. Como se muestra en la Ilustración 4, desde A podríamos ir a G con un coste 2 (la distancia) o a B con un coste 4 o a D con un coste 5. El siguiente nodo visitado sería G por tener un coste menor a otros nodos y evaluaríamos a que otro nodo seguir siguiendo la misma lógica, que sería B. A modo de aclaración, el siguiente nodo a visitar sería D desde G, ya que el coste es menor respecto a ir a C desde B.

- Algoritmo de Bellman-Ford: [2] es parecido al algoritmo de Dijkstra. Hallará el camino con la ruta óptima permitiendo pesos negativos. Sin embargo, no ofrece ninguna ventaja sobre nuestro problema, ya que no usamos pesos negativos al trabajar con distancias.
- Algoritmo de Floyd-Warshall: [3] funciona comparando todos los caminos entre pares de vértices y acabará mejorando paulatinamente la estimación hasta llegar al más óptimo. Aunque funciona bien en grafos pequeños, su complejidad es muy alta ($O(n^3)$) y ofrece una escasa escalabilidad a proyectos más grandes.

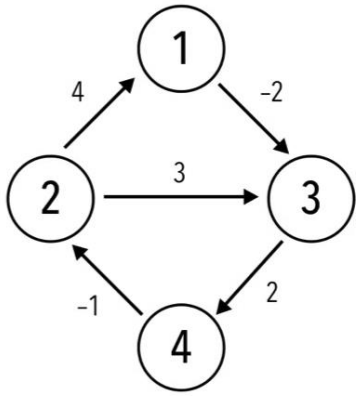


Ilustración 5. Grafo para mostrar el algoritmo de Floyd-Warshall

Nodo destino

Nodo inicial

	1	2	3	4
1	0		-2	
2	4	0	3	
3			0	2
4		-1		0

Ilustración 6. Tabla que recoge las conexiones entre nodos

Para mostrar su funcionamiento [16], las Ilustraciones 5 y 6 serán de gran utilidad, siendo la imagen 5 un grafo de ejemplo y la imagen 6 el primer paso del algoritmo, donde se marcan que nodos puedo visitar a partir de cual y sus respectivos pesos, siendo los espacios en blanco indicadores de que no se puede acceder. El siguiente sería completar la tabla con la siguiente fórmula:

$dist[i][j] > dist[i][k] + dist[k][j]$

Donde *dist* es el valor dentro de cada posición en la matriz, i la fila, j la columna y k un valor entre 1 y 4. En caso de ser una casilla vacía, *dist* valdrá infinito.

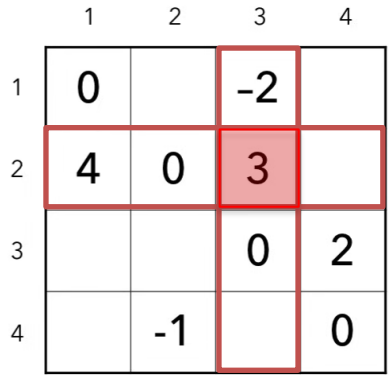


Ilustración 7. Modificación de la tabla

Gracias a la ilustración se puede observar un paso del algoritmo, donde:

$dist[2][3] > dist[2][1] + dist[1][3] \rightarrow 3 > 4 + (-2)$

Como no cumple la ecuación, la casilla $dist[2][3]$ se cambiará por $dist[2][1] + dist[1][3] = 2$

Estas iteraciones se repetirán por toda la tabla hasta completar todos los huecos en blanco y actualizar los valores correspondientes.

- Algoritmo A*: Es una mejora del algoritmo de Dijkstra que incorpora heurísticas basadas en la estimación del costo mínimo desde un nodo hasta el nodo objetivo. [5] Combina aspectos de la búsqueda en anchura y profundidad para optimizar el proceso de búsqueda. Su complejidad dependerá de la calidad de la heurística. En el mejor caso, puede ser significativamente más rápido que Dijkstra si la heurística es buena. Este algoritmo es especialmente útil en la búsqueda sobre espacios grandes, como en videojuegos o en la planificación de rutas urbanas, donde se requiere eficiencia debido al alto número de nodos y conexiones.

Dado el anterior análisis, podemos concluir que los mejores algoritmos para poder estudiar el problema son el de Dijkstra y el A*. Sin embargo, hay muchas diferencias entre ambos algoritmos que pueden ayudar a tomar una decisión.

En cuanto a la eficiencia, [6] A* generalmente requiere menos iteraciones que Dijkstra para encontrar el camino óptimo si la heurística es adecuada, en concreto podemos encontrar ejemplos donde sea 4 veces más rápido. Esto se debe a que A* no necesita explorar todos los caminos de manera uniforme, sino que se dirige prioritariamente hacia el destino, lo que reduce el número de nodos evaluados.

A lo que se refiere con escalabilidad, en redes de metro con muchos nodos y conexiones complejas, A* puede manejar mejor el incremento en el tamaño del grafo gracias a su capacidad de focalizar la búsqueda.

La aplicabilidad del algoritmo de Dijkstra es excelente para asegurar el camino mínimo en términos de distancia o coste en cualquier situación, pero A* es preferible cuando se tiene una buena estimación de la distancia restante hasta el objetivo, permitiendo que el algoritmo sea más selectivo en sus caminos.

Dado que el objetivo es manejar un grafo grande y optimizar la búsqueda, el algoritmo A* es mejor por su capacidad para adaptarse a mapas extensos y su eficiencia mejorada en términos de tiempo de cómputo y recursos utilizados, ya que se cuenta con una heurística apropiada que estima de forma efectiva las distancias hasta el objetivo.

1.2.2 Aplicaciones de navegación

Las aplicaciones de navegación son herramientas digitales diseñadas para ayudar a los usuarios a determinar la mejor ruta de un lugar a otro. Se utilizan principalmente para facilitar el desplazamiento, mejorar la eficiencia en el tráfico y apoyar la planificación de rutas en diversos contextos, como el transporte personal y público.

Normalmente, las aplicaciones de navegación tienen distintos propósitos como la planificación de rutas que ayudan a encontrar el camino más corto de un lugar a otro considerando distintas variables como el tráfico, la distancia y las preferencias personales, la navegación e información del transporte en tiempo real para saber en qué lugar te encuentras en todo momento o el tiempo que tarda un medio de transporte en llegar a una parada concreta. En resumen, ayudan a optimizar los diferentes viajes que tengamos que realizar.

Existen muchas aplicaciones que ayudan a lograr estos objetivos, entre otros:

- ***Metro de Madrid***

Antes de la construcción de la línea 1 de metro, [7] en el último tercio del siglo XIX, Madrid ya contaba con transporte público. En concreto un tranvía de tracción animal que fue sustituido paulatinamente por tranvías de vapor. La primera línea se fue creada para comunicar Sol y Cuatro Caminos en 1919 y posteriormente se fueron creando nuevas líneas para poder comunicar toda la ciudad hasta la actualidad.

Ilustración 8. Metro de Sol en la actualidad

Tras la expansión del metro, una forma óptima de coordinarlo y que los madrileños y turistas pudiesen utilizarlo con control de su ruta y destino, se decidió crear la aplicación "Metro de Madrid", que cuenta con información acerca [8] del tiempo de espera en tu estación más cercana, el tiempo de espera de los siguientes trenes en cualquier estación de la red, las estaciones más cercanas a tu ubicación, el estado de las instalaciones, calcular el mejor trayecto, etc.

Ilustración 9. Interfaz de la Página Principal del Metro de Madrid

- ***Metro de Nueva York***

Es el sistema de transporte ferroviario más grande de Estados Unidos, y de los más grandes del mundo con 469 estaciones. [9] La primera línea subterránea se construyó en 1904 y las líneas se agrupaban en dos sistemas separados por propiedad privada: *"Brookling Rapid Transit Company"* e *"Interborough Rapid Transit Company"*.

Para poder saber que ruta realizar para ir de una parada a otra, [10] la aplicación más importante es "MyTransit NYC Subway, Bus, Rail", que promete las mismas funcionalidades que la aplicación del Metro de Madrid, y además [11] una base de datos realmente precisa para poder cubrir las necesidades de cada usuario de forma global en vez de tener que depender de otras aplicaciones. Para ello, utilizan diferentes algoritmos de Inteligencia Artificial y "*Machine Learning*", y además ofrecen su código para fines educativos.

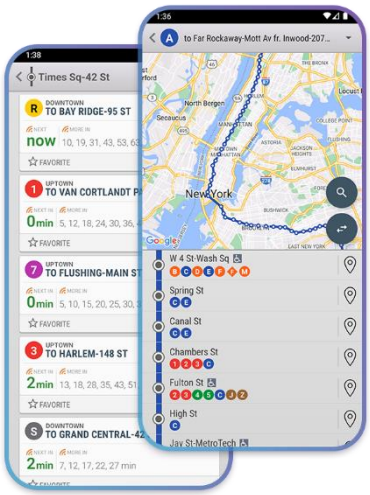


Ilustración 10. Interfaz de la aplicación MyTransit

- ***Metro de Estocolmo***

El metro de Estocolmo no es solo un transporte más de la ciudad, sino una atracción turística [12]. Desde que empezó a construirse en 1944 a finales de la Segunda Guerra Mundial, algunas entre las más de 100 estaciones que cuenta esta red están decoradas como verdaderas obras de arte, que lo hace muy atractivo para visitar para los diferentes turistas y un concepto del que los ciudadanos están orgullosos. Su aplicación cuenta, además de poder consultar la ruta y los tiempos de los diferentes medios de transporte disponible, de pago de billetes tanto simple como bonos de diferentes horas/días y es muy cómodo para poder renovar los viajes y que no se produzca ningún contratiempo.

Ilustración 11. Consulta de rutas en la aplicación

Ilustración 12. Pago de billetes y abonos en la aplicación

1.2.3 Librerías

En Python, que es el entorno donde se desarrolla este proyecto, existen numerosas librerías para poder implementar el algoritmo de búsqueda y poder trabajar con grafos ponderados.

- ***Networkx***

Ilustración 13. Icono de Networkx

Esta librería es de las más usadas para trabajar con grafos y redes. Permite crear, manipular y analizar grafos de manera eficiente [17].

Su principal ventaja es su capacidad con grafos de gran complejidad y tamaño, perfecto para el problema propuesto.

Además, es una librería tan común que los algoritmos mencionados anteriormente ya están implementados directamente como funciones.

Al ser la librería más popular, es la que se acaba usando para el trabajo. Para un uso debido de la herramienta, se necesitó el uso de su documentación [23] y [24].

- ***Igraph***

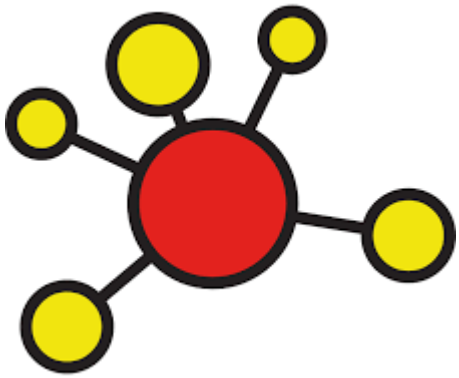


Ilustración 14. Icono de igraph

Es una librería con un alto rendimiento, pero requiere familiarización para poder usarla de la mejor manera al ser funciones algo menos intuitivas. Sin embargo, esto le permite al programador hacer grafos más personalizables.

- ***Graph-tool***

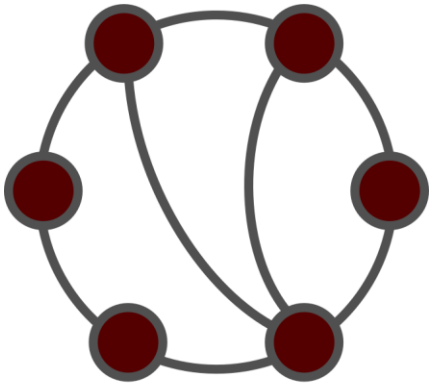


Ilustración 15. Icono de graph-tool

Es una herramienta con una curva de aprendizaje aún más profunda. Aunque la documentación es muy completa, suele resultar mucho más compleja de perfeccionar. Su complejidad se razona porque es una librería basada en C++ y se le han implementado unos enlaces para poder utilizarse en Python.

2 Desarrollo

2.1 Algoritmo A*

El algoritmo que se usará para este trabajo será el algoritmo A*. Por lo razonado en apartados anteriores es el más adecuado para poder resolver el problema del camino mínimo en un grafo al tener tantos nodos y aristas ponderadas. Sin embargo, hay ciertos aspectos específicos en este trabajo que es necesario definir para entender la implementación.

Los elementos esenciales de este algoritmo son:

- Nodos: son elementos posibles a los que podemos acceder. Tendremos un grafo con n nodos.
- Aristas ponderadas: son las posibles rutas que unen los distintos nodos. Todos los nodos que precisen de una conexión serán de forma bidireccional representando las líneas del metro de Estocolmo. Esto quiere decir que no será un grafo completo, o lo que es lo mismo, no todos los nodos estarán conectados entre ellos.
- Nodo origen y nodo destino. Lo definiremos como S y X respectivamente.
- Costo acumulado: $g(n)$ representa el costo total del camino desde el nodo inicial hasta el nodo n. Particularmente reflejará la distancia real en metros, que recorre el tren entre una parada y otra.
- Heurística: $h(n)$ es una estimación del costo más bajo posible desde un nodo n hasta el siguiente nodo en nuestra ruta. Esta función nunca debe sobreestimar el costo real, es decir, este valor no debe ser mayor que la $g(n)$ para cada par de nodos. En nuestro problema, será la distancia en línea recta en metros entre una parada y otra.
- Función de evaluación: se define como $f(n) = g(n) + h(n)$, donde $f(n)$ es la estimación del costo total del camino más corto pasando por el nodo n.
- Lista de cerrados: nodos visitados por el algoritmo para no analizarlos de manera redundante.
- Lista de abiertos: nodos alcanzados por cada nodo contenido en la lista de cerrados. En cada iteración se añadirán nuevos caminos en esta lista dependiendo del alcance del nodo que se acaba de añadir en la lista de cerrados.

Se analizará el comportamiento de este algoritmo en las diferentes iteraciones siguiendo las siguientes imágenes:

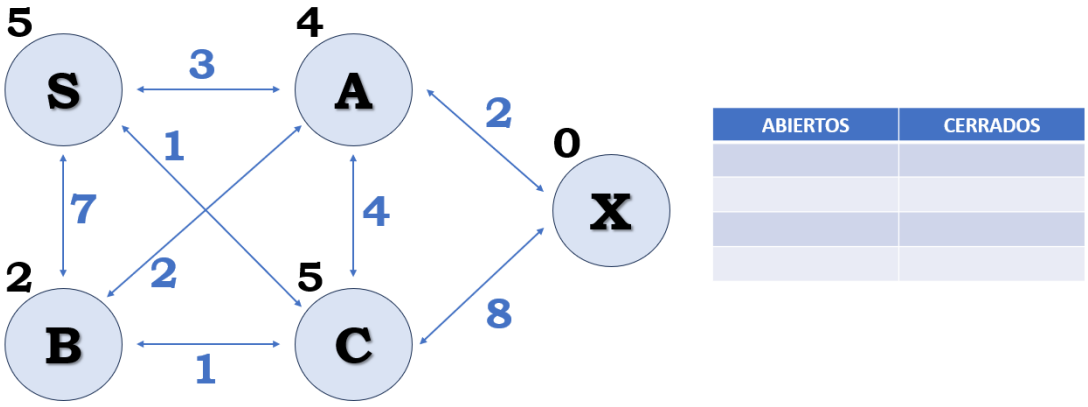


Ilustración 16. Grafo a analizar y lista de abiertos y cerrados

Como se puede observar en la Ilustración 16, se analizará un grafo de 5 nodos completo. El costo real está marcado en azul al lado de cada arista, y la heurística de cada nodo está al lado de cada nodo en negrita. Para nuestro problema, el costo real será el camino que realizará el metro bajo tierra, incluyendo curvas, y la heurística será el camino en línea recta.

El primer nodo a analizar es el nodo *S*, por lo que se introducirá en la lista de cerrados con su función de evaluación [18]. En este caso:

$f(n) = 0 + 5 = 5.$

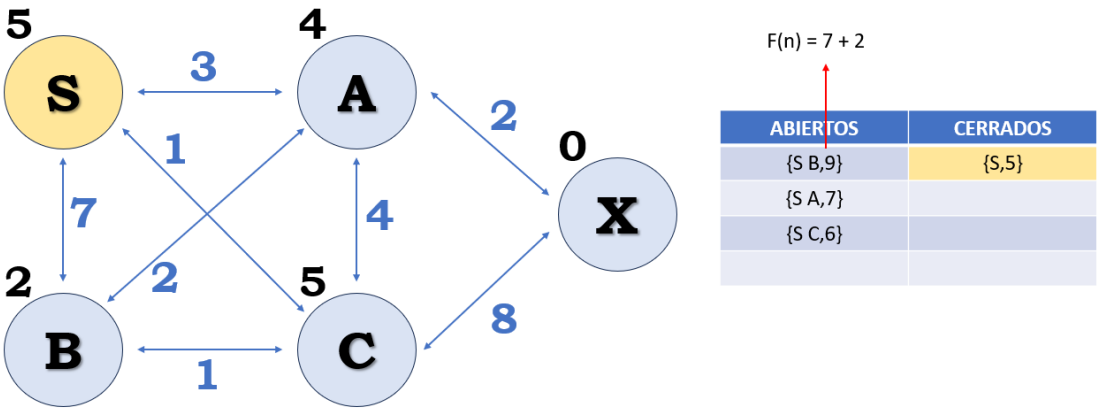
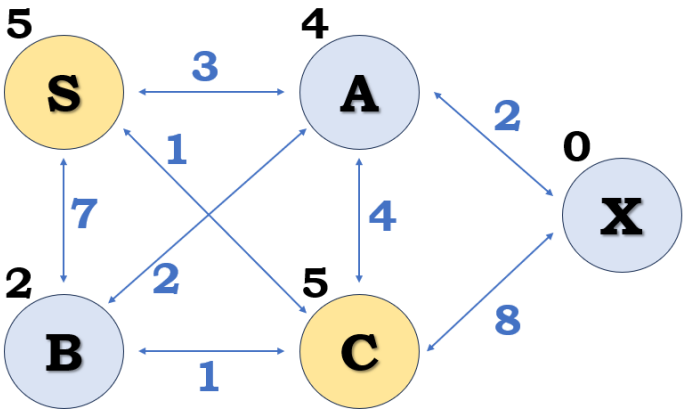


Ilustración 17. Primera iteración.

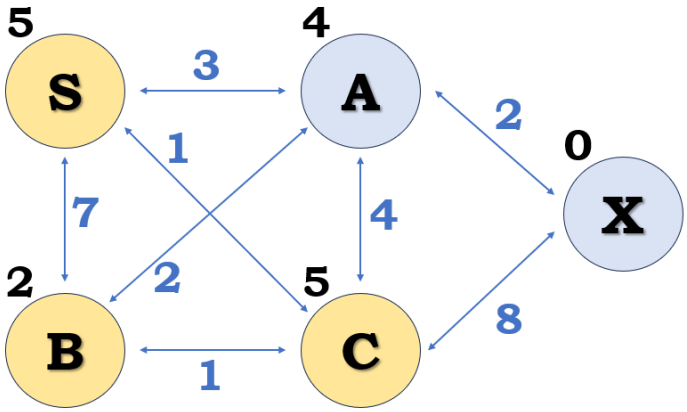
En la primera iteración se actualiza la lista de abiertos con los nodos que están conectados mediante una arista con el nodo que acabamos de añadir y su respectiva función *f(n)* como está señalado en la figura. Para la siguiente iteración, se escogerá aquel camino que tenga una *f(n)* menor que las demás.



ABIERTOS	CERRADOS
{S B,9}	{S,5}
{S A,7}	{S C,6}
{S C B, 4}	
{S C X, 9}	

Ilustración 18. Segunda iteración.

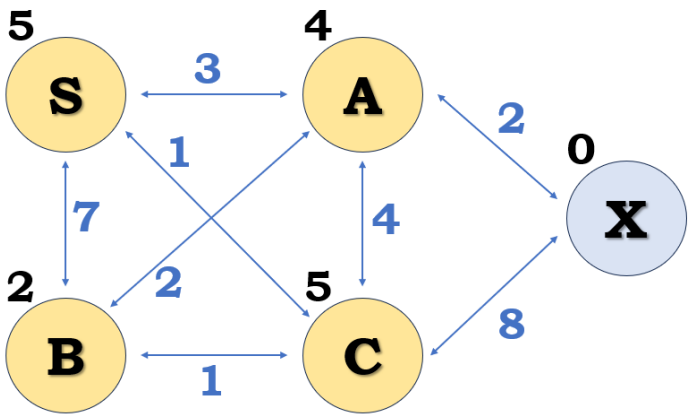
El siguiente elemento introducido en la lista de cerrados es {S C,6} que representa el camino S -> C con una función *f(n)* igual a 6. Consecuentemente se actualiza lista de abiertos con los nuevos nodos a los que se puede llegar desde el nodo C. En la ilustración 18, S -> C -> A no se ha añadido a la lista de abiertos porque A ya estaba añadido a esta lista y la g en S -> A que es 3 es menor que la g en S -> C -> A que es 5. En cambio, S -> C -> B si se puede añadir [19]. Tampoco se añadirán aquellos nodos que estén en la lista de cerrados, como S.



ABIERTOS	CERRADOS
{S B,9}	{S,5}
{S A,7}	{S C,6}
{S C X, 9}	{S C B, 4}

Ilustración 19. Tercera iteración.

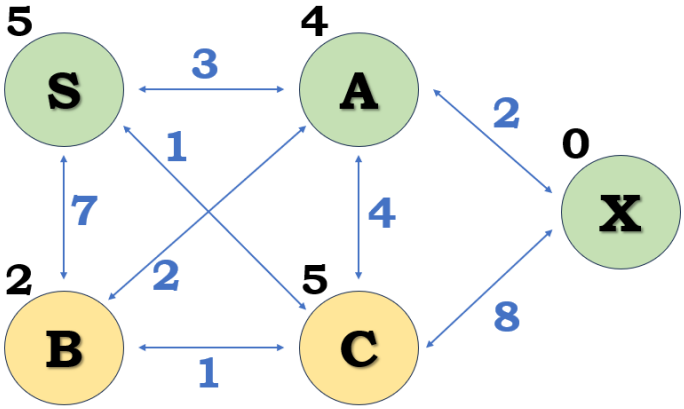
Al igual que en la segunda iteración, se elegirá aquel camino que tenga una *f(n)* menor, en este caso S -> C -> B. No se añade S -> C -> B -> A porque el nodo A ya ha sido visitado con un coste menor.



ABIERTOS	CERRADOS
{S B,9}	{S,5}
{S C X, 9}	{S C,6}
{S A X, 5}	{S C B, 4}
	{S A,7}

Ilustración 20. Cuarta iteración.

Se realiza una nueva iteración desde A. En este caso, no añadimos el camino S -> A -> C a nuestra lista de abiertos porque C ya ha está en la lista de cerrados, al igual que pasa con el nodo B.



ABIERTOS	CERRADOS
{S B,9}	{S,5}
{S C X, 9}	{S C,6}
	{S C B, 4}
	{S A,7}
	{S A X, 5}

Ilustración 21. Quinta y última iteración. El camino óptimo está marcado en verde.

En la Ilustración 21 y última iteración podemos ver que se añade a la lista de cerrados el nodo final X, por lo que el algoritmo ha terminado y el camino mínimo será S -> A -> X.

Para poder implementar este algoritmo en nuestro proyecto y poder crear un camino óptimo entre paradas de metro, necesitamos hacer una modificación importante en la heurística para que se adapte mejor al problema.

En el esquema que queremos simular del metro, también incluye ciertas líneas de tren que se explicarán más adelante en el trabajo. Esto quiere decir que el usuario en un camino determinado podría necesitar hacer un transbordo entre metro y tren. Como es bien sabido, moverse de un andén de metro a un andén de tren es mucho más lento que moverse de un andén de metro a otro de metro, y esto se tiene que penalizar de alguna forma.

Por ello, la heurística tendrá en cuenta tanto la distancia en línea recta para que el camino elegido sea lo más corto posible, y teniendo en cuenta que en el transbordo entre distintos tipos de transporte se pierde algo más de tiempo. Todo ello ayudará a tener un camino mejorado.

2.2 Diseño de la aplicación

2.2.1 Diseño preliminar

Antes de empezar a implementar el código, se necesitó definir unos requisitos y un diseño de bajo y alto nivel para poder clarificar los objetivos y contenidos de la aplicación.

2.2.1.1 Especificación de requisitos

Entre los requisitos principales se encuentran:

1. El usuario deberá entrar en la aplicación y entender que es una web, para encontrar un camino entre una parada origen y otra destino en el mapa de metro de Estocolmo.
2. Tendrá que proporcionar como input las paradas que desea, sin cometer errores. Si los comete, se deberán especificar de forma correcta para poder ser corregidos por el mismo usuario.
3. El output/salida de la web será proporcionar una lista de paradas que tiene que pasar el usuario para llegar a su destino. Este camino será definido con la ayuda del algoritmo A*.
4. Mostrar al usuario tanto la distancia recorrida como el tiempo que tardará en recorrer el camino.

2.2.1.2 Diseño de alto nivel

Ya definidos unos requisitos, se necesita establecer como el usuario va a interactuar con la aplicación.

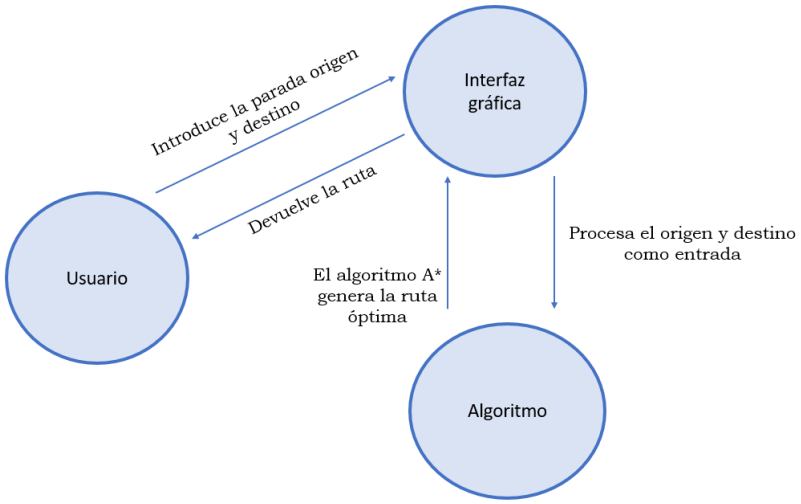


Ilustración 22. Diseño de alto nivel

Como se puede ver en la imagen, el usuario sería el primero en interactuar con el programa. Introduciría un origen y destino en la interfaz y al darle a "Enviar" el algoritmo podrá procesar el origen y destino. Posteriormente en el código Python, el algoritmo A* generará una ruta óptima de coste mínimo que se imprimirá en la interfaz para que, finalmente, el usuario pueda trabajar con ella.

Es un esquema simple sin necesidad de usar bases de datos ya que todas las paradas están ya introducidas en el código y no hay necesidad de crear un registro del usuario para una aplicación de este estilo.

Una vez diseñado, se podrá implementar tanto el algoritmo como el diseño de la aplicación.

2.2.2 Mapa de Estocolmo

Para el proyecto se ha usado un mapa del metro de Estocolmo, pero simplificado para reducir una complejidad innecesaria para nuestra aplicación. Esta edición se ha realizado con la aplicación Photoshop.

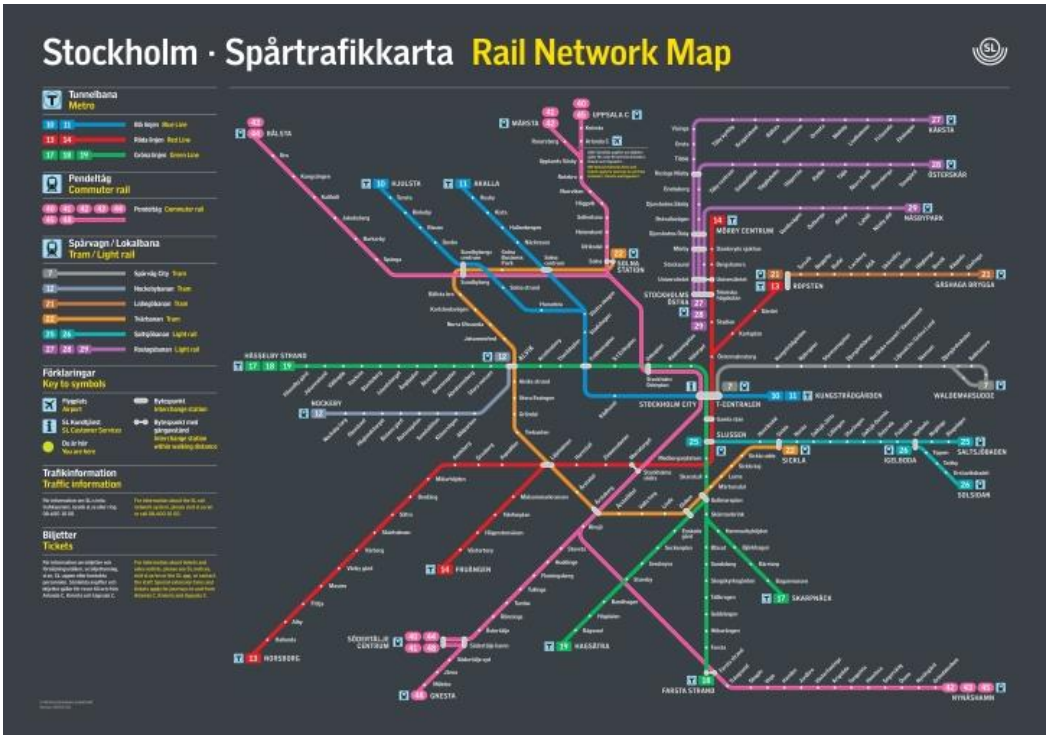


Ilustración 23. Mapa de Estocolmo completo.

Ilustración 24. Mapa de Estocolmo modificado

Gracias a esta imagen con la que trabajará el usuario, podemos identificar 5 líneas distintas dependiendo del color con sus respectivas líneas. También hay que aclarar que la línea naranja y rosa son las de tren ya que tienen un icono distinto a las líneas azul, roja y verde, que son de metro.

2.2.3 Toma de datos

Para poder configurar el grafo con el que trabajaremos, tuvimos que medir con la herramienta de Google Maps "Medir distancia" tanto la distancia en línea recta (para la heurística) como la distancia que hace el metro en el subsuelo (para el coste).

Ilustración 25. Herramienta de Google Maps para medir distancias

Haciendo esto en todas las paradas que necesitamos contemplar, se pudo almacenar todos los datos en unas tablas Excel para poder después extrapolarlo al código Python.

Vastra Skogen	1120	1118
Solna Centrum	1410	1330
Huvudsta (desde Vastra skogen)	1130	1120
Solna Strand	876,02	859,76
Sundbybergs Centrum	806,14	790,85
Abrahamsberg		
Stora mossen	715,91	715,91
Alvik	980,96	906,67
Kristineberg	1220	1190
Thorildsplan	704,64	696,74
Fridhemsplan	942,02	926,57
S:t Eriksplan	844,16	842,72
Odenplan	937,32	816,62
Radmansgatan	657,47	592,1
Hotorget	612,52	611,89
T-Centralen	374,72	367,65
Gamla Stan	1220	1020
Slussen	498,97	496,7
Medborgarplatsen	566,99	565,31
Skanstull	739,74	739,63
Gullmarsplan	1021	1020
Globen	615,4	583,88
Skarmarbrink (desde Gullmarsplan)	686,07	655,42
Ropsten		
Gardet	1210	1060
Karlaplan	1100	1050
Ostermalmstorg	1230	1190

Ilustración 26. Ejemplo de la recopilación de datos en Excel

Posteriormente, se incluyó un tiempo aproximado de duración de la ruta. Para eso, se investigó el tiempo medio que tarda un tren en recorrer las paradas de metro en las diferentes líneas. Por ejemplo, se llegó a la conclusión que [20] de media en la línea azul el metro circulaba a una velocidad de 40.93 kilómetros por hora, la línea roja a 35.92 km/h y en la verde 29.6 km/h. En la Ilustración 26 se pueden observar estos datos.

Line	Stretch	Travel time[10]	Length	Stations (in "innerstan")
10	Kungsträdgården – Hjulsta	23 min	15.1 km (9.4 mi)	14, (5)
11	Kungsträdgården – Akalla	22 min	15.6 km (9.7 mi)	12, (5)
13	Norsborg – Ropsten	44 min	26.6 km (16.5 mi)	25, (10)
14	Fruängen – Mörby centrum	33 min	19.5 km (12.1 mi)	19, (9)
17	Skarpnäck – Åkeshov	43 min	19.6 km (12.2 mi)	24, (12)
18	Farsta strand – Alvik	37 min	18.4 km (11.4 mi)	23, (12)
19	Hagsätra – Hässelby strand	55 min	28.6 km (17.8 mi)	35, (12)
Entire metro network			*108 km (67 mi)*	*100, (25)*

Ilustración 27. Tiempo, distancia y paradas de cada línea [20].

También se investigó cual es la velocidad media de los trenes correspondientes a la línea rosa y naranja, el cual era de [21] 60 km/h.

Con todos estos datos conseguimos hacer un cálculo básico y poder obtener una aproximación de cuanto tarda en cada ruta dependiendo de las líneas por las que vaya. Aunque no se trabaje con tiempos actualizados a tiempo real de cuanto tarda cada tren en pasar por la estación, esto ayudará a tener una suposición del tiempo más acertada.

Para tomar los datos y calcular los tiempos de transbordos, se han analizado los horarios de los metros de Estocolmo.

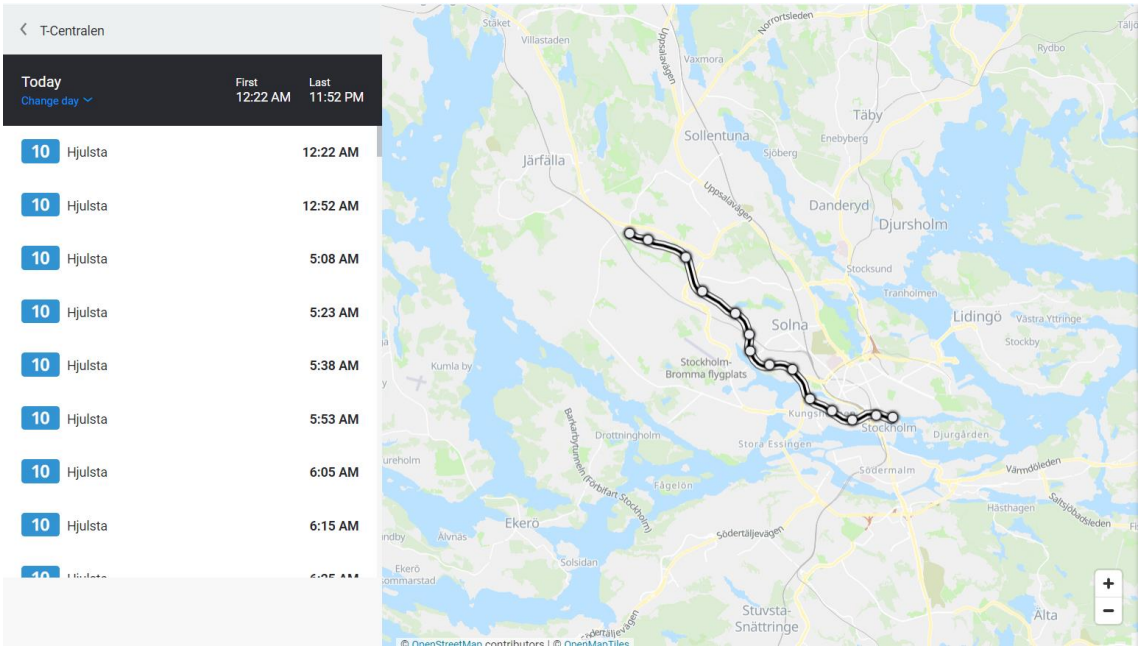


Ilustración 28. [29] Horarios de la línea 10

Como se puede observar en la Ilustración 28, se pueden obtener los horarios de los metros de cada línea. Todos estos horarios se pusieron en un formato determinado para que la aplicación pudiese leer un archivo JSON y tener los tiempos de transbordos actualizados correctamente, como se puede observar en el anexo 6.7.

2.2.4 Clases y funciones

Para poder mostrar la estructura, tanto general como específica, de la aplicación web, recurriremos tanto a un diagrama de clases como a la explicación de cada método usado.

2.2.4.1 Diagrama de clases de la aplicación

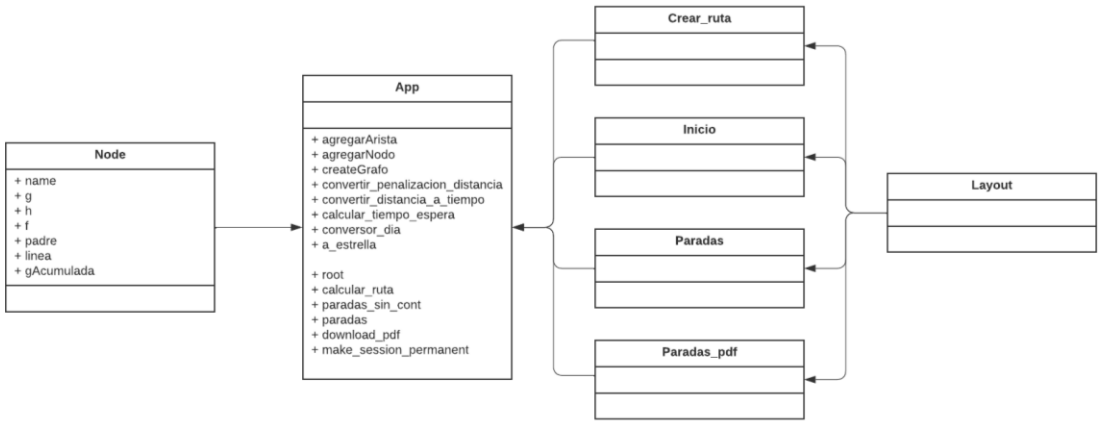


Ilustración 29. Diagrama de clases de la aplicación

Como se puede observar en el diagrama de clases, la aplicación cuenta con 6 clases distintas. Cinco de ellas pertenecientes a la interfaz gráfica (*Crear_ruta*, *Inicio*, *Paradas*, *Paradas_pdf* y *Layout*), *Node* seá una clase importante para poder generar el grafo y trabajar con él y por último la clase *App* que será desde donde ejecutaremos el código e integra todas las partes. Además, en esta misma clase esta creado el algoritmo con sus respectivas clases auxiliares y está implementado la clase *Node* para facilitar el código.

2.2.4.2 Librerías y funciones

Las librerías usadas en todo el código han sido:

- *Networkx*: como se explicó en el apartado 1.2.3 donde se explican las librerías para manejar grafos ponderados, esta librería la usaremos para la creación, manipulación y estudio de la estructura, dinámica y funciones de grafos complejos. Particularmente, haremos uso de la función `nx.DiGraph()` para crear un grafo dirigido, `add_edge()` para añadir aristas dirigidas y `neighbors(nodo)` para sacar una lista de vecinos de un nodo.
  Hay que tener en cuenta que *networkx* funciona añadiendo los nodos y sus respectivos vecinos en una estructura matricial. Por ejemplo, supongamos *G* mi grafo, *nodo* un nodo cualquiera y *vecino* un vecino del nodo, *G[nodo][vecino]* será la arista que conecta ambos nodos y podremos analizar sus atributos. Esto nos será muy útil ya que guardamos las distancias correspondientes a la *g* y *h* en las aristas.
- *Heapq:* proporciona una implementación de la cola de prioridad basada en una cola binaria. Se han utilizado funciones como `heapq.heappush()`

o *heapq.heappop()* para añadir o eliminar elementos de la cola de prioridad. También se ha utilizado *heapq.heapify()* para mantener la estructura de prioridad sobre algún parámetro en caso de eliminar alguno. Estas funciones se utilizarán en el algoritmo A*.

- *Flask:* framework web ligero para Python. Se utiliza para poder manejar la aplicación web y sus interacciones. Algunas componentes usadas son *flask*, *render_template* para poder visualizar una pestaña HTML desde un código Python, *request* para obtener parámetros introducidos por el usuario, *session* para mantener la sesión de la web y que no tenga que correr el código continuamente y *url_for* y *make_response* para poder generar un PDF correctamente.
- *Datetime:* para el manejo de fechas y tiempos. Se usa *datetime* para conseguir la hora o fecha actual y *timedelta* para dar un formato de hora a unos números y trabajar como si fuese una hora.
- *Time:* manejo de tiempo en general. Solo se usa *time.strftime()* para convertir de un formato hora a una forma determinada para poder imprimirlo en el HTML.
- *Json:* manejo de datos en formato JSON. Se ha utilizado para poder visualizar los datos en un PDF. Principalmente los componentes usados han sido *json.dumps()* y *json.loads()* para convertir objetos Python a JSON y viceversa.
- *Weasyprint:* herramienta para renderizar HTML y CSS en archivos PDF. Se ha utilizado la componente HTML dentro de esta librería para convertir una cadena HTML en un objeto que se pueda visualizar en un PDF.

En cuanto a las clases y funciones, se van a analizar los implementados dentro de la clase app.py (Python). Esta clase es la que almacena todos los métodos para que el algoritmo A* y la página web funcionen correctamente.

- *class Node:* clase que definirá que atributos tendrá cada nodo visitado. Esto será *name, g, h, f, padre* (nodo inmediatamente anterior en el camino creado), *línea* y *gAcumulada.* Cuenta con en constructor y el método *__lt__()* que define el comportamiento de comparación de nodos basado en el coste total *f*.
- *def agregar_arista(G, u, v, g, h, linea):* método que se encarga de agregar una arista al grafo. Llamará a la clase *Edge* y creará la arista de forma bidireccional, es decir, primero la dirección *u->v* y después *v->u*. Se inicializará con la *g* (coste) y *h* (heurística) y *líneas* correspondientes a los datos recogidos.
- *def agregar_nodo(name, f=0, g=0, h=0, padre=None, linea="", gAcumulada=0)*: de forma análoga a *agregarArista(),* llamará a la clase *Node* para crear el nodo correspondiente. Se inicializará con el nombre correspondiente y todos los demás valores serán 0 o vacíos.
- *def createGrafo()*: incluye la definición de todas las aristas y nodos. Llamará repetidamente a los métodos *agregarArista()* y *agregarNodo()* tantas veces como aristas y nodos queramos añadir respectivamente.
- *def convertir_distancia_a_tiempo(distancia, linea)*: convierte la distancia entre dos nodos a minutos teniendo en cuenta la velocidad a la que va el tren en cada una de las líneas.
- *def calcular_penalización_distancia(línea_actual, línea_nueva)*: añade una penalización determinada si hace un transbordo entre las líneas roja, azul y verde (metro) a una naranja o rosa (tren).

- *def calcular_tiempo_espera(hora_actual, dia_actual, línea_nueva)*: como se define en el encabezado de la función, la hora y día en la que se quiere realizar la ruta y la línea a la que se hace el transbordo influirá en el tiempo de transbordo medio que el usuario tardará en realizarlo. Todos estos datos están recogidos en el apartado 2.2.3.
  Además, también tiene en cuenta cuando el tren está fuera de servicio. Calculará el tiempo que falta para que salga el primer tren, y se contemplará a la hora de calcular los diferentes tiempos de transbordos o rutas.
- *def conversor_dia(dia):* función auxiliar para traducir los nombres de los días de inglés a español. Esto es porque las librerías que manejan la fecha devuelven el valor en inglés y se quería mostrar al usuario en español.
- *def a_estrella(G, start, end):* método principal de la aplicación e implementará paso a paso lo explicado en el apartado 2.1. Además, combina las librerías *networkx* y *heapq* para poder realizar correctamente el algoritmo. Adicionalmente, hacemos uso de la clase *Node* para poder crear una nueva instancia para cada nuevo nodo añadido en la lista de abiertos. Esto hará que podamos manejar mejor las comparaciones entre nodos y poder recorrer mejor la ruta final en busca de los nodos pertenecientes a la misma.
  Al estar todo en algoritmo en un bucle con un número de iteraciones hasta N (número de nodos) que significa una complejidad de O(N), extraer cada nodo con el menor *f* de la lista de abiertos tiene una complejidad de O(log N) y la expansión a los vecinos contando tanto la eliminación como la reorganización de la cola de prioridad con una complejidad de O(A) y O(log A) respectivamente siendo A el número de aristas, nos queda una complejidad total del algoritmo de O(A + Nlog(N)) en el peor de los casos.

Los métodos explicados a continuación son los necesarios para definir la interfaz gráfica en HTML de la aplicación. También están dentro de app.py.

- *def root():* página de inicio de la aplicación. En el se encontrará información de las paradas más importantes de Estocolmo. Esta información se incluye en este apartado porque [22] hay 90 paradas de metro que podrían ser consideradas obras de arte. Por ello, seis paradas que están implementadas en el proyecto están explicadas brevemente y con una foto descriptiva.
- *def crear_ruta():* contiene el mapa del metro y la parte indispensable en cuanto a la interacción con el programa. En esta pestaña se podrá elegir el origen y el destino apoyándose en el mapa proporcionado y en un desplegable con todas las paradas de metro ordenadas alfabéticamente. También se le da al usuario la opción de poder escribir la parada y se le irá recomendando la parada deseada autocompletándolo con la ayuda del desplegable para reducir el número de errores.
  Además, podrá elegir o no la hora a la que quiere salir y la hora de llegada proporcionada en la pestaña siguiente se adaptará a la hora que se introduzca. En caso de no introducir ninguna hora, el programa asumirá que la hora de salida es la actual.
  De igual forma se puede introducir el día de la semana en el que se quiere iniciar la ruta. En caso de iniciarlo en el viernes o sábado, los tiempos de transbordo y frecuencia de trenes y metros será distinto que otros días. También tiene un manejo de errores en caso de no introducir la parada correcta en cualquier campo.

- *def paradas_sin_cont():* en caso de acceder a la pestaña *Paradas* sin haber seleccionado un origen o un destino, se abrirá una pestaña donde se indicará que no hay una ruta seleccionada.
- *def paradas():* método que muestra la ruta, la distancia total de la ruta y el tiempo aproximado. Para añadir más información, se aporta también cuánto tardará en realizar los transbordos y una descripción de los transbordos a realizar. En caso de que a la hora seleccionada no esté el servicio de metro en funcionamiento, se avisará al usuario y también se notificará de la hora a la que saldrá el primer metro.
  En esta pestaña también estará descrita la hora de llegada aproximada, el día y hora de salida y el tiempo total de la ruta incluyendo los transbordos.
  Las diferentes paradas se mostrarán del color de su respectiva línea para mejorar la interactividad con el usuario.
  Tendrá la opción de volver a la página *crear_ruta* pero se le avisará al usuario de que se borrará la ruta creada, o generar un pdf con el camino a recorrer y toda la información adicional para guardarlo para futuras ocasiones.
- *def download_pdf():* método que genera el pdf con la información de las paradas a seguir para completar el camino.
- *def make_session_permanent():* método para asegurar que la sesión va a permanecer abierta durante 7 días en caso de inactividad.

La clase Inicio (HTML)se refiere a la pestaña que conecta con el método *root().*

La clase Crear_ruta (HTML)es una pestaña que conecta con el método *crear_ruta().*

Paradas (HTML) conecta con los métodos *paradas_sin_cont()* y *paradas()* y Paradas_pdf (HTML) se vincula con el método *download_pdf().*

Por último, la clase Layout (CSS) es el método que establece el diseño de la página y todos los componentes. Será la parte responsable de que la aplicación sea intuitivo y llamativo.

En el anexo, apartado 6, se podrá encontrar la especificación del código, con todos los métodos y clases implementados. Cada anexo es un archivo distinto.

2.3 Interfaz gráfica

La interfaz gráfica es una parte importante para poder mostrar al usuario el funcionamiento del proyecto. Como se ha mencionado previamente, cuenta con cuatro pestañas distintas para poder proporcionar información de las paradas del metro de Estocolmo y, lo más importante y el objetivo del trabajo, poder generar una ruta gracias al algoritmo A*.

Para poder visualizar correctamente el funcionamiento de la página web, se desarrollará un esquema de diseño, un mapa de navegación y por último la descripción de las pestañas.

2.3.1 Esquema de diseño

Se cuentan con 3 esquemas de diseño:

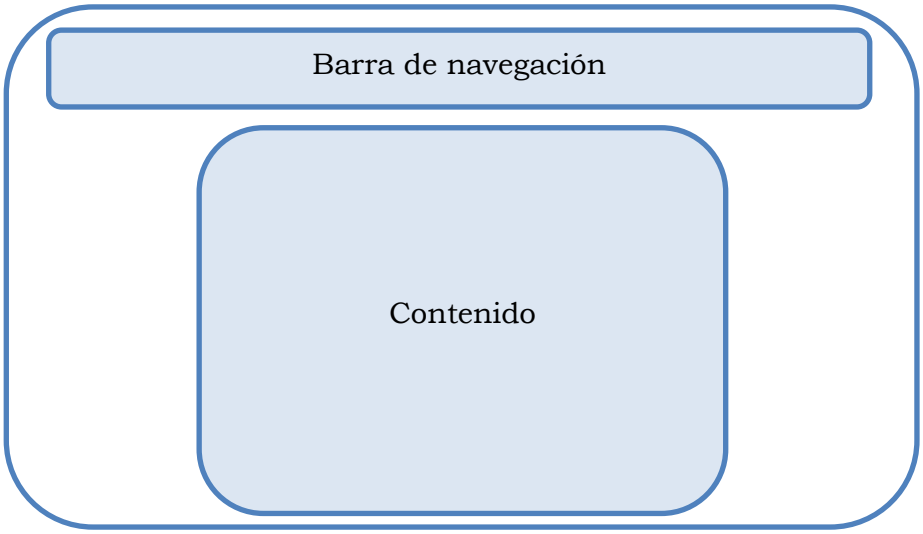


Ilustración 30. Esquema de diseño de la pestaña Inicio

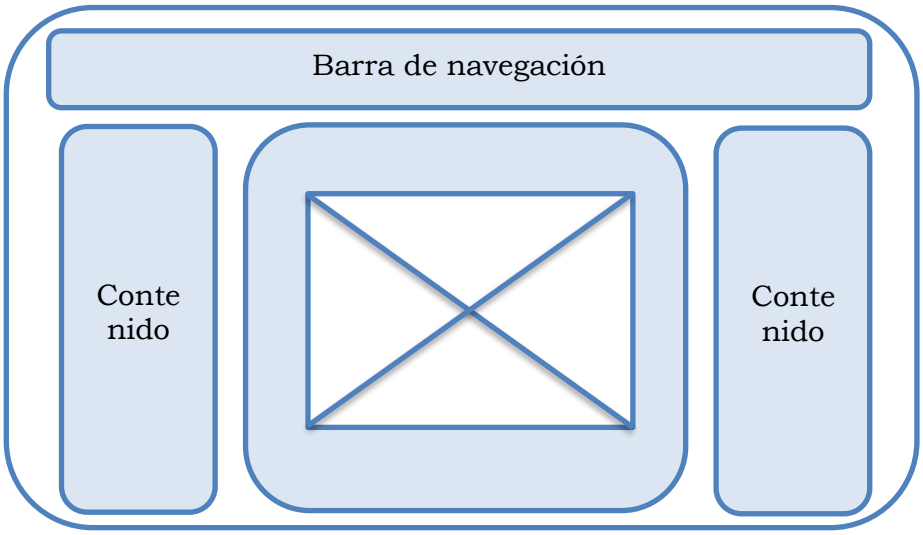


Ilustración 31. Esquema de diseño de la pestaña crear_ruta

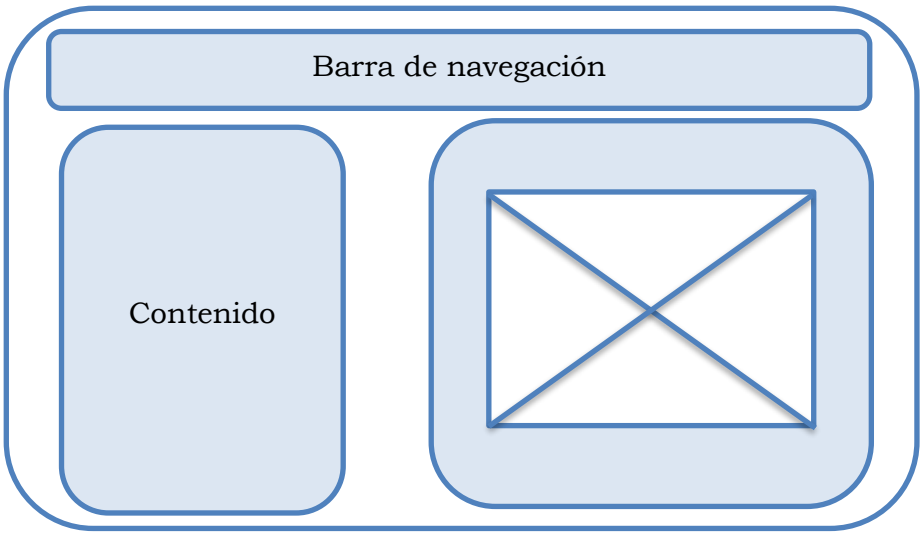


Ilustración 32. Esquema de diseño de la pestaña paradas

Las Ilustraciones 30, 31 y 32 ayudan a entender la composición de los elementos generales de la aplicación, donde en todas las pestañas se mantiene una barra de navegación para mejorar la fluidez de interacción del usuario y el diseño es intuitivo y poco cargado para mantener la atención de la persona que lo usa.

2.3.2 Mapa de navegación

Para poder imaginarse como funciona la aplicación web, es muy útil diseñar un mapa de navegación previo a la implementación para que el diseño de los distintos componentes permita situarse en cualquier pestaña en cualquier momento gracias a la barra de navegación y además saber a qué pestañas deberíamos poder acceder y en que fase se pueden visualizar.

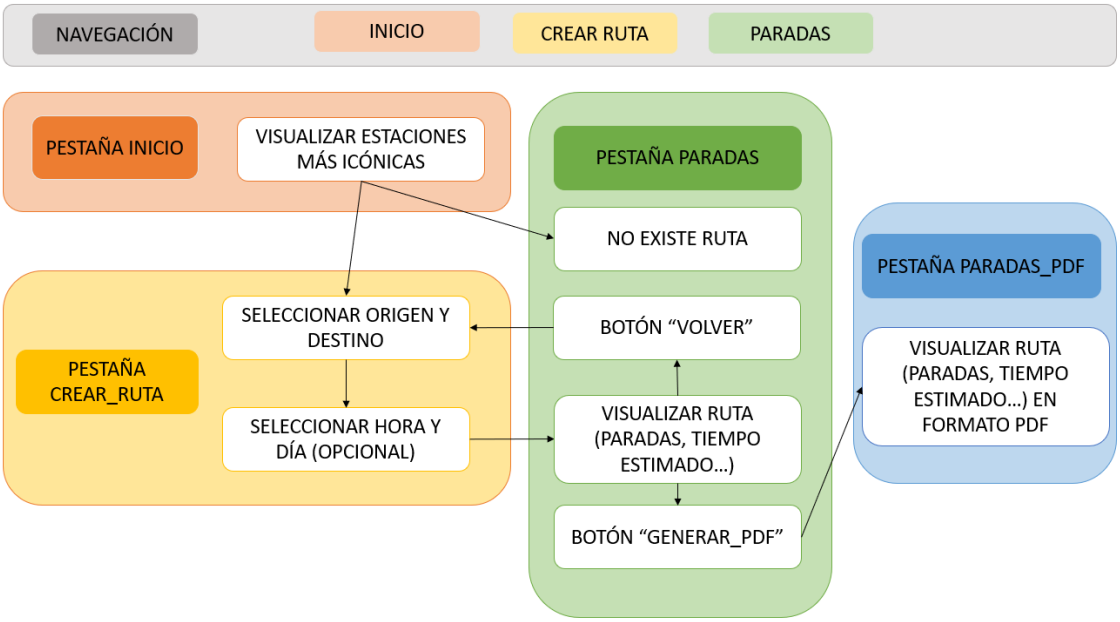


Ilustración 33. Mapa de navegación orientativo

Como se puede observar, la barra de navegación conecta todas las pestañas menos la de *paradas_pdf*, que solo se podrá visualizar desde el botón *generar_pdf* en la pestaña paradas. Desde la pestaña de inicio se podrá visualizar cuales son las paradas más icónicas o representativas del sistema de metro de Estocolmo con una breve descripción de las mismas.

En la pestaña *crear_ruta*, se deberá seleccionar el origen y destino y opcionalmente la hora y el día. Solo si se ha seleccionado el origen y destino se podrá visualizar una ruta en la pestaña *paradas*. En cualquier otro caso, se emitirá un mensaje que indica que hay que elegir dichas paradas.

En la pestaña *paradas*, se visualizará la ruta con sus respectivas paradas, el tiempo de la ruta, la distancia recorrida, los transbordos a realizar y el tiempo que se va a emplear en dichos transbordos y por último la hora y el día de llegada. Toda esta información ayuda al usuario a saber los detalles de la ruta especificada. En caso de querer guardarlo, se podrá generar un PDF y se le dirigirá a la pestaña *paradas_pdf* donde se podrá descargar toda la información en este formato.

2.3.3 Pestañas

Tras ver la explicación del diseño, las distintas pestañas o funcionalidades que se pueden encontrar en este proyecto son los siguientes:

- Pestaña *inicio*

Ilustración 34 y 35. Pestaña inicio

En estas imágenes se muestra como es la pantalla de inicio nada más compilar el código.

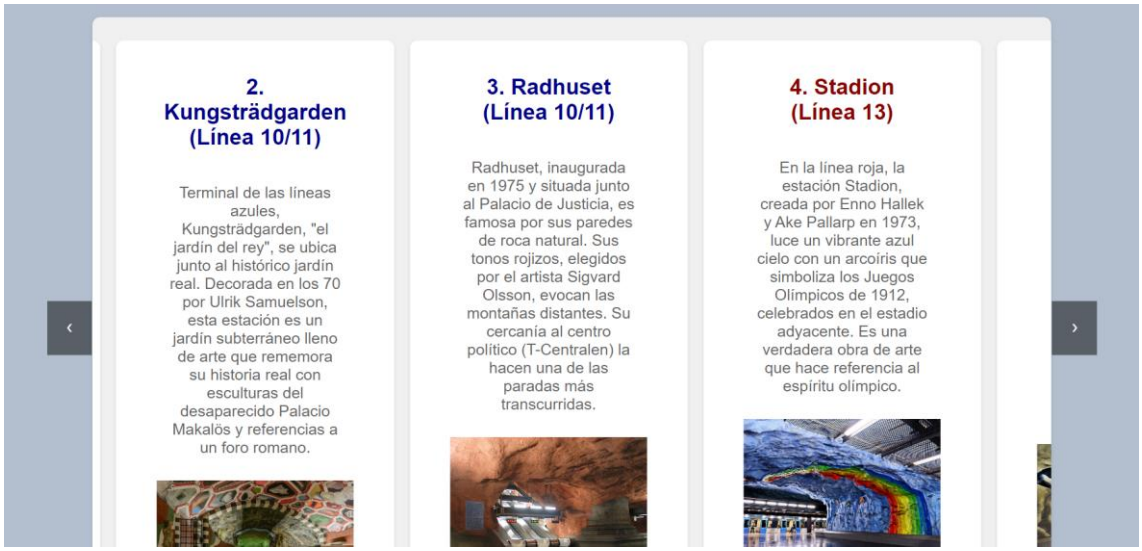


Ilustración 36. Pestaña inicio

Cuando se selecciona a las flechas, se podrá visualizar la información de las paradas más icónicas.

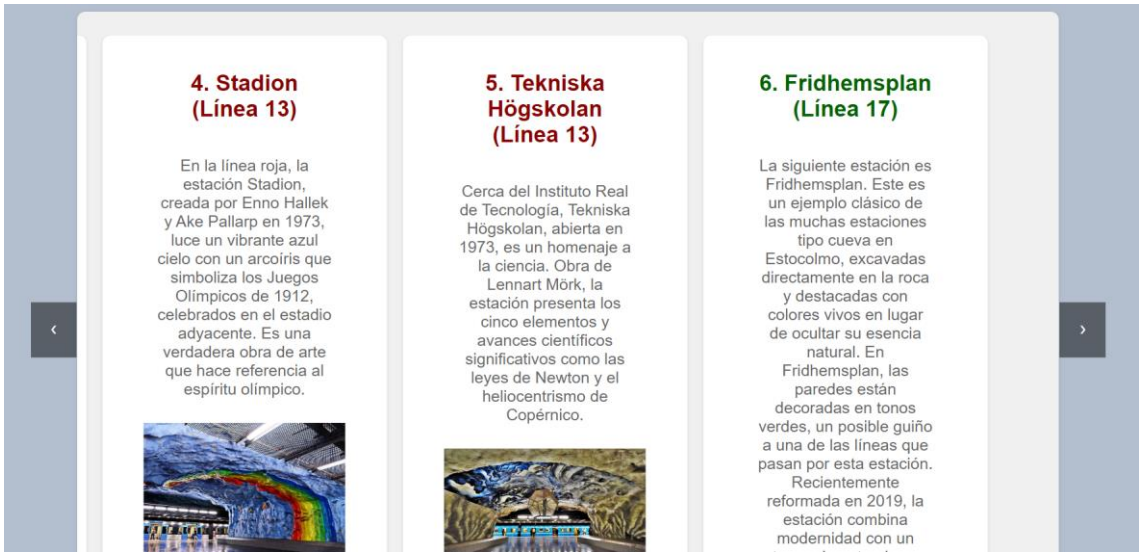


Ilustración 37. Pestaña inicio

En total hay 6 paradas explicadas con su correspondiente foto. Esto ayudará al usuario a entrar en contexto de porque el metro de Estocolmo destaca sobre otros sistemas de metro de todo el mundo. El diseño escogido como cartas laterales hace que sea un formato más atractivo para el usuario.

- Pestaña *crear_ruta*

Ilustración 38 y 39. Pestaña crear_ruta

Las ilustraciones 31 y 32 muestran una de las pantallas más importantes, la responsable de crear la ruta. Como se puede observar, se tendrá que proporcionar un origen y destinos válidos y la hora de salida y el día de la semana es opcional.

También cuenta con un texto donde explica que el transbordo de metro a tren y viceversa será penalizado al tardar más tiempo en realizarlo que un transbordo entre líneas de metro.

Ilustración 40 y 41. Menú desplegable y autocompletar de las paradas.

Para evitar errores, hay un menú desplegable con todas las paradas ordenadas alfabéticamente donde se puede seleccionar la parada que se desee o escribirla y el menú desplegable se acotará en base a lo que se esté introduciendo en la entrada de texto.

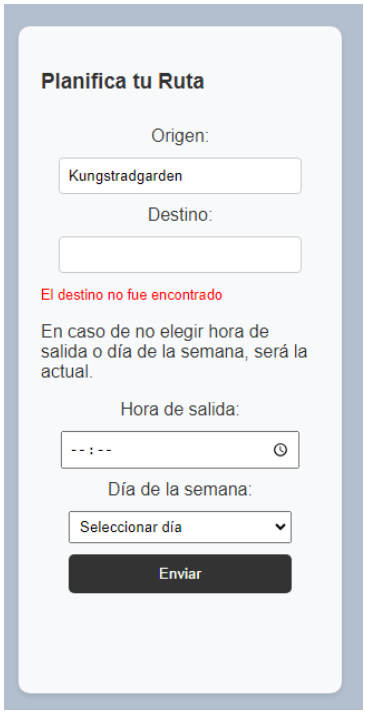
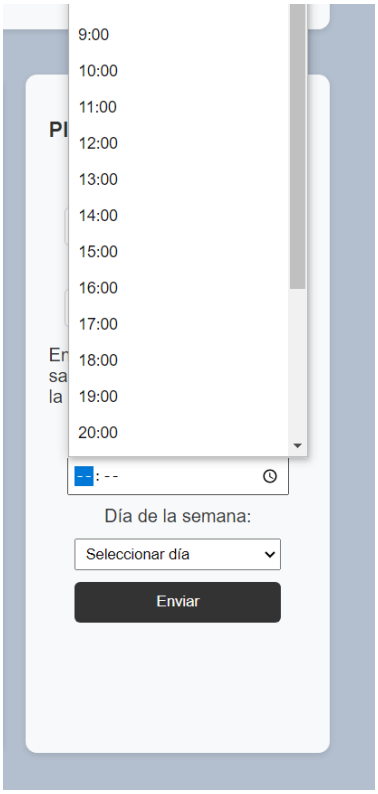


Ilustración 42. Caso de error.

En caso de introducir incorrectamente algún dato, se mostrará un mensaje de error indicando el problema.

En caso de querer seleccionar una hora de salida, se podrá hacer uso de un desplegable que proporciona un horario de 5:00 hasta las 1:00. En caso de querer proporcionar otra hora, también se podrá escribir numéricamente.



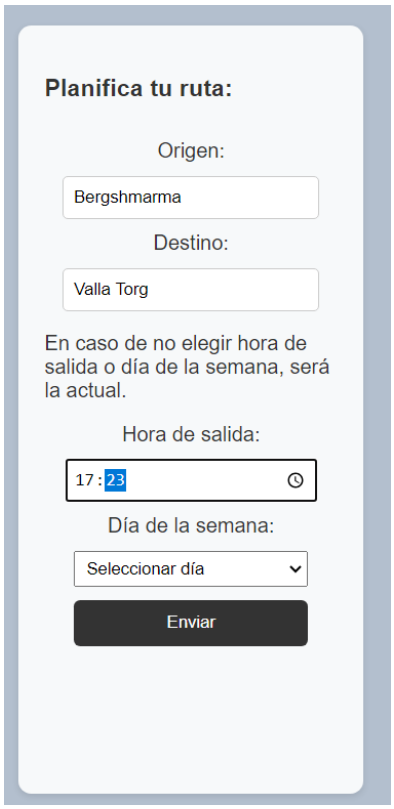


Ilustración 43 y 44. Completar la hora de salida

En caso de querer seleccionar un día de la semana, para evitar errores solo se puede seleccionar a través del desplegable.

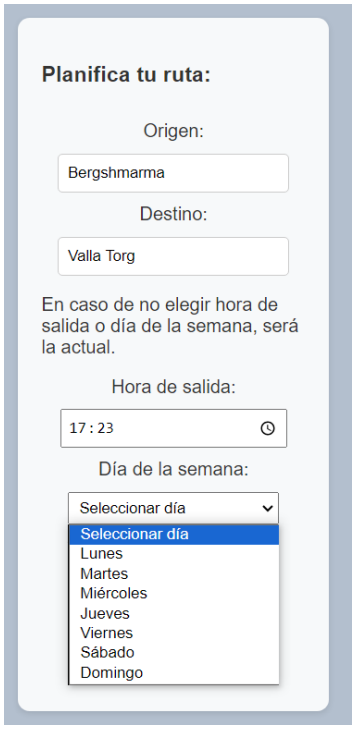


Ilustración 45. Seleccionar día

- Pestaña *paradas*

Ilustración 46, 47 y 48. Pestaña paradas

Al darle al botón de “Enviar” con todos los datos proporcionados de forma correcta, se dirigirá a la pestaña paradas, donde se mostrarán las paradas de la ruta. Como se puede observar, las paradas se encuentran del color de la línea en la que se encuentran para ayudar al usuario a seguir la ruta.

Además, se proporciona el tiempo estimado en realizar la ruta y la distancia total, el tiempo de transbordo total con una descripción de lo que tardará en cada cambio de línea. Finalmente, también se muestra el tiempo total de viaje incluyendo los transbordos y la hora de salida y llegada, con el día de salida.

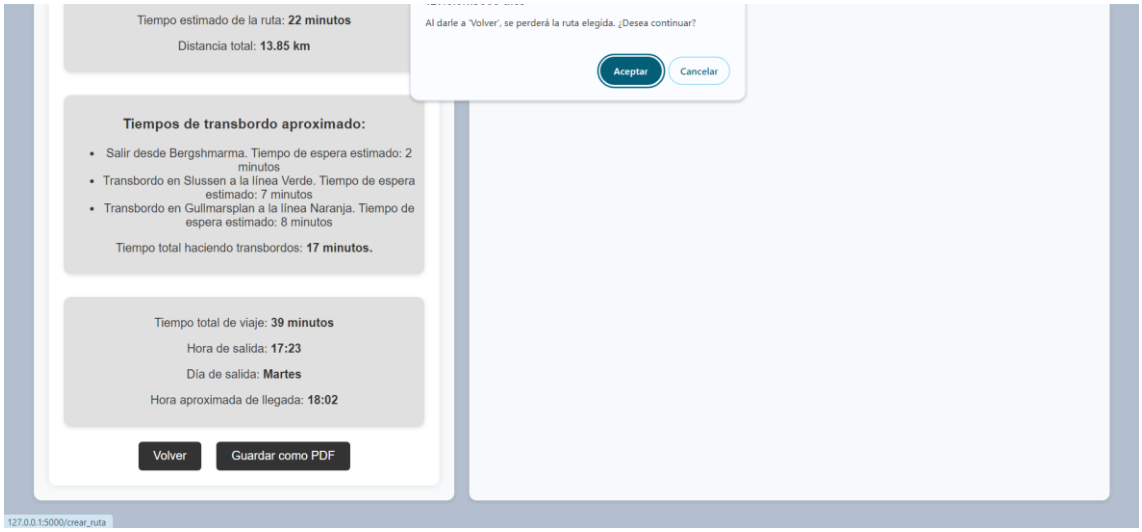


Ilustración 49. Presionar el botón volver

En caso de darle al botón volver, también se generará un aviso de si desea continuar, indicando que la ruta mostrada se eliminará. En caso de darle a aceptar en el Pop-up, se redirigirá a la pestaña *crear_ruta* para poder indicar nuevas paradas otra vez, con todos los campos vacíos. En caso de querer modificar algún aspecto de la ruta, se tendrá que seleccionar “Mapa” en la barra de navegación y la información no se perderá.

- Pestaña *paradas_pdf*

Ilustración 50, 51 y 52. Pestaña paradas_pdf

En caso de querer guardar la ruta, se dará al botón “Guardar como PDF” y se generará un PDF con la información de las paradas, la distancia, el tiempo, la hora de llegada aproximada y el mapa para tenerlo de referencia. Con este formato, da la opción al usuario de imprimir, descargar o incluso hacer anotaciones.

Para el desarrollo de la interfaz, se ha necesita hacer uso de la siguiente documentación: [25] y [26] para la familiarización con el entorno, [27] para poder implementar numerosos estilos y componentes y [28] para el menú desplegable de las paradas.

3 Resultados y conclusiones

El objetivo principal del trabajo es optimizar las rutas entre dos nodos específicos en un grafo, utilizando el algoritmo A*. La implementación se llevó a cabo en Python, utilizando un modelo adaptado del mapa del metro de Estocolmo. Los resultados obtenidos indican que el algoritmo A* es altamente eficaz para encontrar rutas óptimas en grafos complejos como los del metro de Estocolmo.

La aplicación desarrollada permite a los usuarios seleccionar sus puntos de partida y destino, generando de manera eficiente la ruta óptima en términos de distancia y tiempo. La interfaz gráfica basada en HTML y CSS, integrada con el backend en Python, ofrece una experiencia de usuario fluida y accesible.

Se realizaron pruebas exhaustivas en diferentes escenarios, incluyendo rutas simples y complejas con múltiples transbordos. Los resultados mostraron que el algoritmo A* no solo encuentra la ruta más corta, sino que también maneja eficazmente los transbordos entre líneas de metro y tren, penalizando adecuadamente el tiempo adicional requerido para cambiar de un medio de transporte a otro.

En escenarios complejos, el algoritmo A* demostró ser más eficiente que otros métodos, explorando menos nodos y encontrando rutas óptimas con mayor rapidez. Además, la heurística utilizada permitió focalizar la búsqueda, lo que resultó en una significativa reducción del tiempo de cálculo.

Se ha considerado como alternativa al algoritmo A* el algoritmo de Dijkstra. Se observó que el algoritmo A* es significativamente más rápido, especialmente en grafos grandes y complejos. La heurística utilizada en A* permitió reducir el número de nodos evaluados, mejorando así la eficiencia del proceso de búsqueda.

El algoritmo de Dijkstra, aunque robusto y preciso, evalúa todos los posibles caminos de manera uniforme, lo que puede ser ineficiente en grafos de gran tamaño. En contraste, el algoritmo A* se beneficia de la heurística que guía la búsqueda hacia el objetivo, evitando la exploración innecesaria de nodos y caminos que no contribuyen a la solución óptima.

También se ha analizado la complejidad de ambos algoritmos usando colas de prioridad binarias, siendo el de Dijkstra O((A+V)logV) y el algoritmo A* implementado O (A + VlogV), siendo A el número de aristas y V el número de vértices.

El algoritmo destaca por su capacidad de combinar la búsqueda heurística con el cálculo del coste real, lo que lo hace más eficiente que otros algoritmos como Bellman-Ford y Floyd-Warshall. Además, la heurística empleada en A* permite una estimación precisa de la distancia restante hasta el objetivo, reduciendo significativamente el número de nodos explorados y, por ende, el tiempo de cálculo.

La implementación en Python es la mejor en comparación con otros lenguajes, ya que sus librerías manejan óptimamente el uso de grafos complejos. Como se ha analizado en este proyecto, la librería networkx es la mejor para el problema propuesto.

La implementación del código junto con una interfaz gráfica amigable desarrollada en HTML, permite que la aplicación sea útil para usuarios finales

de diversos niveles técnicos. Para reforzar esta idea, se realizaron distintos test de usabilidad a usuarios no necesariamente familiarizados con desarrollo de páginas web y los resultados fueron muy importantes para perfilar distintos componentes o algún aspecto de cualquier pestaña para que el trabajo fuese lo más atractivo posible.

Esta interfaz también podría haber sido implementada en Python usando diferentes librerías como *tkinter*, pero la visualización y aspecto de la página en comparación a HTML es anticuada y poco llamativa para el usuario.

En conclusión, el uso del algoritmo A* para la optimización de rutas en el metro de Estocolmo demuestra ser una solución robusta y eficiente, y el desarrollo de una interfaz gráfica en un entorno donde se pueden entender fácilmente las distintas funciones del programa hace que el mapa del metro de Estocolmo sea la mejor solución para conocer la ruta óptima entre dos paradas de metro o tren.

4 Análisis de Impacto

El proyecto tiene un impacto positivo en múltiples áreas, demostrando que la tecnología puede ser una herramienta poderosa para mejorar la eficiencia y la sostenibilidad en la movilidad urbana. La combinación de un algoritmo eficiente como A* y una interfaz de usuario bien diseñada asegura que esta solución sea no solo técnica y operativamente sólida, sino también accesible y beneficiosa para una amplia gama de usuarios.

Desde la creación de la idea como el desarrollo del algoritmo y posteriormente de la interfaz gráfica se ha tenido en cuenta los diferentes impactos que puede tener en la sociedad la aplicación web que se ha desarrollado.

En esta sección se analizarán diferentes contextos donde el impacto ha sido destacable.

4.1 Impacto Personal

El desarrollo de este proyecto ha significado un antes y un después en mi educación universitaria. He tenido que aprender a lidiar con diferentes problemas que nunca he experimentado como aprender lenguajes como Python o HTML profundamente para desarrollar el trabajo, o generar las suficientes ideas para que sea una aplicación atractiva y completa.

Por ello, todo lo que se ha aprendido realizando este trabajo, desde distintos lenguajes de programación hasta la diferencia entre algoritmos de búsqueda para grafos ponderados dirigidos y saber cuál es mejor en cada caso, me ayudará en un futuro a solucionar cualquier problema relacionado con estos temas. Sin embargo, ha sido todo un reto familiarizarse con los distintos entornos de trabajo usados para realizarlo.

4.2 Impacto Empresarial

En toda empresa se busca una reducción de costes para obtener más beneficios. Este trabajo ayuda a lograrlo dando una solución óptima a distintos usuarios, pudiendo crear una organización capaz de monitorizar las ventajas de este algoritmo en el trabajo propuesto. Estas organizaciones pueden estar ligadas al gobierno en caso de que propongan reducciones de costes optimizando las rutas entre distintas paradas con el uso del A* o privadas, pudiendo ser una aplicación de navegación sobre el transporte público.

Ambas opciones serán propuestas interesantes para formar una idea de negocio.

4.3 Impacto Social

Desde una perspectiva social, la mejora en la planificación de rutas puede contribuir a una movilidad urbana más eficiente. Los usuarios pueden reducir sus tiempos de viaje, lo que mejora la calidad de vida en ciudades densamente pobladas. Además, la interfaz intuitiva asegura que personas de todas las edades y habilidades tecnológicas puedan beneficiarse de la herramienta, promoviendo una mayor inclusión.

Este impacto está alineado con el Objetivo de Desarrollo Sostenible (ODS) 11: "Ciudades y comunidades sostenibles", que promueve el acceso a sistemas de transporte públicos, seguros, asequibles, accesibles y sostenibles para todos.

4.4 Impacto Económico

La optimización de rutas puede llevar a una disminución en los costos operativos del transporte público, al permitir una mejor planificación y distribución de los recursos. Esto puede resultar en tarifas más bajas para los usuarios y una mayor eficiencia en el uso de fondos públicos destinados al transporte. Como se ha mencionado en el Impacto Social, la mejora en la eficiencia del transporte público puede incentivar su uso, reduciendo la dependencia de vehículos privados y promoviendo una economía más sostenible.

Este impacto se relaciona con el ODS 9: "Industria, innovación e infraestructura", que busca construir infraestructuras resilientes, promover la industrialización inclusiva y sostenible, y fomentar la innovación.

4.5 Impacto Medioambiental

Una mejor planificación de rutas contribuye a que los usuarios se decanten por el uso transporte público frente al privado. Se reducen las emisiones de gases contaminantes, al disminuir el uso de vehículos en circulación.

Esto se alinea con los Objetivos de Desarrollo Sostenible (ODS) de la Agenda 2030, específicamente con el ODS 13: "Acción por el clima", que promueve la adopción de medidas urgentes para combatir el cambio climático y sus efectos. Al promover el uso del transporte público, se puede reducir la huella de carbono de las ciudades, contribuyendo así a la mitigación del cambio climático y una reducción de la contaminación ambiental y acústica presente en todo tipo de ciudades, principalmente en las capitales de los países.

4.6 Impacto Cultural

El proyecto también tiene un impacto cultural al facilitar el acceso a diferentes puntos de interés en la ciudad, promoviendo así el turismo y la apreciación del patrimonio cultural de Estocolmo. Una mejor accesibilidad puede fomentar un mayor flujo de turistas y residentes a las diversas atracciones culturales, enriqueciendo la experiencia urbana.

Con todo esto se busca cumplir el ODS 8: "Trabajo decente y crecimiento económico", que busca promover el turismo sostenible, creando empleos y promoviendo la cultura y los productos locales.

4.7 Proyección a futuro

Este trabajo es una recopilación de información de diferentes algoritmos de búsqueda para encontrar la mejor la ruta entre paradas de metro de Estocolmo. Se puede expandir el número de paradas, añadir medios de transporte público y tener en cuenta distintos factores que pueden alterar la ruta como los días festivos o eventos importantes, los cuales cambiarían el tiempo de transbordos o la apertura o cierre de distintas estaciones. Además, la publicación de este trabajo y su acogida por cualquier organización será importante para mejorar diferentes aspectos mencionados previamente.

Aplicar algoritmos como el A* tienen enorme potencial para resolver problemas relacionados con grafos dirigidos ponderados de forma eficiente y este proyecto es uno de muchos casos de aplicación.

5 Bibliografía

[1] Estefania Cassingena Navone (2022). *Algoritmo de la ruta más corta de Dijkstra - Introducción gráfica y detallada* [Online]. Disponible: https://www.freecodecamp.org/espanol/news/algoritmo-de-la-ruta-mas-corta-de-dijkstra-introduccion-grafica/

[2] Jhosimar George Arias Figueroa (2013). *Camino más corto, algoritmo de Bellman-Ford* [Online]. Disponible: https://jariasf.wordpress.com/2013/01/01/camino-mas-corto-algoritmo-de-bellman-ford/

[3] Ingrid Mendoza & Luisa Ruedas (2021). *Algoritmo de Floyd-Warshall* [Online]. Disponible: https://medium.com/algoritmo-floyd-warshall/algoritmo-de-floyd-warshall-e1fd1a900d8

[4] Sealtiel Gro.Loza (2011). *Algoritmo de Dijkstra y Algoritmo de Bellman-Ford* [Online]. Disponible: https://prezi.com/7egkjhlo-mgl/algoritmo-de-dijkstra-y-algoritmo-de-bellman-ford/

[5] Muchos colaboradores (2011). *Algoritmo de búsqueda A** [Online]. Disponible: https://es.wikipedia.org/wiki/Algoritmo_de_b%C3%BAsqueda_A*

[6] Santiago Fiorino (2024). *Comparando Algoritmos: A* vs Dijkstra, en el mapa de la ciudad.* [Online]. Disponible: https://www.youtube.com/watch?v=oMgfGkFSgI0&ab_channel=SantiagoFiorino

[7] Diego Salvador. *Breve Historia del metro de Madrid* [Online]. Disponible: https://www.rutasconhistoria.es/articulos/breve-historia-del-metro-de-madrid

[8] Equipo del Metro de Madrid (2024). *Aplicaciones oficiales de Metro de Madrid* [Online]. Disponible: https://www.metromadrid.es/es/viaja-en-metro/apps

[9] Equipo del metro de Nueva York (2024). *Metro de Nueva York* [Online]. Disponible: https://es.wikipedia.org/wiki/Metro_de_Nueva_York

[10] Angie (2016). *Apps útiles para viajar a Nueva York* [Online]. Disponible: https://prezi.com/7egkjhlo-mgl/algoritmo-de-dijkstra-y-algoritmo-de-bellman-ford/

[11] MyTransit (2024). *MyTransit app* [Online]. Disponible: https://www.mytrans.it/

[12] Civitatis. Metro de Estocolmo [Online]. Disponible: https://www.estocolmo.es/metro

[13] Jahaziel Ponce (2021). *Algoritmos de búsqueda* [Online]. Disponible: https://jahazielponce.com/algoritmos-de-busqueda/

[14] Universidad de Granada. *Capítulo 5: Arrays y cadenas* [Online]. Disponible: https://ccia.ugr.es/~jfv/ed1/c/cdrom/cap5/f_cap56.htm

[15] Juan Antonio Gómez (2013). *Algoritmo de Dijkstra (3) – Teoría de Grafos* [Online]. Disponible: https://www.youtube.com/watch?app=desktop&v=S_5W5bImTH4&ab_channel=JuanAntonioGomez

[16] Michael Sambol (2017). *Algoritmo de Floyd-Warshall en 4 minutos* [Online]. Disponible: https://www.youtube.com/watch?v=4OQeCuLYj-4&ab_channel=MichaelSambol

[17] Fernando Carazo, Joaquín Amat (2023). *Introducción a grafos y redes con Python* [Online]. Disponible: https://cienciadedatos.net/documentos/pygml01-introduccion-grafos-redes-python

[18] Pepe Cantoral (2023). *A* search | Búsqueda de A star* [Online]. Disponible: https://www.youtube.com/watch?v=yxN6yR_7yJM&ab_channel=PepeCantoral%2CPh.D

[19] Nicholas Swift (2017). *Easy A* (star) pathfinding* [Online]. Disponible: https://medium.com/@nicholas.w.swift/easy-a-star-pathfinding-7e6689c7f7b2

[20] Múltiples colaboradores (2023-2024). *Stockholm metro* [Online]. Disponible: https://en.wikipedia.org/wiki/Stockholm_Metro

[21] Múltiples colaboradores (2020-2024). *Stockholm commuter rail* [Online]. Disponible: https://en.wikipedia.org/wiki/Stockholm_Metro

[22] Sara Copé (2023). *Descubre cuáles son las estaciones más bonitas del metro de Estocolmo* [Online]. Disponible: https://sonandoviajesblog.com/2021/10/29/estaciones-mas-bonitas-metro-estocolmo/

[23] Múltiples colaboradores. *NetworkX. Network Analysis in Python* [Online]. Disponible: https://networkx.org/documentation/stable/reference/algorithms/index.html

[24] Stiven Muñóz Murillo (2022). *Librería de grafos de Python – Introducción a NetworkX* [Online]. Disponible: https://www.youtube.com/watch?v=As5NEWj1h4s&t=1178s&ab_channel=StivenMu%C3%B1ozMurillo

[25] Múltiples colaboradores (2010-2024). *Flask. User´s guide* [Online]. Disponible: https://flask.palletsprojects.com/en/3.0.x/

[26] Uskokrum2010 (2021). *Tutorial Flask: Framework de Python para Aplicaciones Web* [Online]. Disponible: https://www.youtube.com/watch?v=1DmVCPB6H8&ab_channel=UskoKruM2010

[27] Múltiples colaboradores. *Build fast, responsive sites with Bootstrap* [Online]. Disponible: https://getbootstrap.com/

[28] María Braeuner (2022). *Etiqueta select de HTML: Cómo hacer un menú desplegable o lista combinada* [Online]. Disponible: https://www.freecodecamp.org/espanol/news/etiqueta-select-de-html-como-hacer-un-menu-desplegable-o-lista-combinada/

[29] Múltiples colaboradores (2024). *SL Tunnelbana - Schedules, Routes and Stops* [Online]. Disponible: https://moovitapp.com/index/en/public_transit-lines-Stockholm-1083-1673395

6 Anexos

6.1 Anexo A. Archivo app.py

Código fuente del archivo app.py que incluye la implementación del algoritmo A* y el manejo de distintos métodos para crear la interfaz gráfica.

```
  1. import networkx as nx
  2. import heapq
  3. from flask import Flask, render_template, request, session, make_response,
url_for
  4. from datetime import datetime, timedelta
  5. import time
  6. import json
  7. from weasyprint import HTML, CSS
  8. import os
  9.
 10. # Clase que albergara todos los nodos de nuestro grafo. Siempre que se trabaje
con nodos, se trabajará con esta clase.
 11. class Node:
 12.     def __init__(self, name, f=0, g=0, h=0, padre=None, linea="", gAcumulada=0):
 13.         self.name = name
 14.         self.f = f
 15.         self.g = g
 16.         self.h = h
 17.         self.padre = padre
 18.         self.linea = linea
 19.         self.gAcumulada = gAcumulada
 20.
 21.     def __lt__(self, other):
 22.         return self.f < other.f
 23.
 24. # Crear aristas en el grafo bidireccionales
 25. def agregar_arista(G, u, v, g, h, linea):
 26.     G.add_edge(u, v, g=g, h=h, l=linea)
 27.     G.add_edge(v, u, g=g, h=h, l=linea)
 28.
 29. def agregar_nodo(name, f=0, g=0, h=0, padre=None, linea="", gAcumulada=0):
 30.     return Node(name, f, g, h, padre, linea, gAcumulada)
 31.
 32. # Crea el grafo de cero cada vez que se inicia la aplicacion. Se encuentran
todos los nodos y aristas con sus respectivos valores
 33. def crear_grafo():
 34.     G = nx.DiGraph()
 35.     nodes = [
 36.         agregar_nodo("Kungstradgarden"), agregar_nodo("T-Centralen"),
 37.         agregar_nodo("Radhuset"), agregar_nodo("Fridhemsplan"),
 38.         agregar_nodo("Stadshagen"), agregar_nodo("Vastra Skogen"),
agregar_nodo("Solna Centrum"),
 39.         agregar_nodo("Huvudsta"), agregar_nodo("Solna Strand"),
 40.         agregar_nodo("Sundbybergs Centrum"), agregar_nodo("Abrahamsberg"),
 41.         agregar_nodo("Stora Mossen"), agregar_nodo("Alvik"),
 42.         agregar_nodo("Kristineberg"), agregar_nodo("Thorildsplan"),
 43.         agregar_nodo("S:t Eriksplan"), agregar_nodo("Odenplan"),
 44.         agregar_nodo("Radmansgatan"), agregar_nodo("Hotorget"),
 45.         agregar_nodo("Gamla Stan"), agregar_nodo("Slussen"),
 46.         agregar_nodo("Medborgarplatsen"), agregar_nodo("Skanstull"),
 47.         agregar_nodo("Gullmarsplan"), agregar_nodo("Globen"),
 48.         agregar_nodo("Skarmirbrink"), agregar_nodo("Ropsten"),
 49.         agregar_nodo("Gardet"), agregar_nodo("Karlaplan"),
 50.         agregar_nodo("Ostermalmstorg"), agregar_nodo("Bergshmarma"),
```

```
51.         agregar_nodo("Universitet"), agregar_nodo("Tekniska Hogskolan"),
52.         agregar_nodo("Stadion"), agregar_nodo("Mariatorget"),
53.         agregar_nodo("Zinkensdamm"), agregar_nodo("Liljeholmen"),
54.         agregar_nodo("Aspudden"), agregar_nodo("Axelsberg"),
55.         agregar_nodo("Midsommarkransen"), agregar_nodo("Telefonplan"),
56.         agregar_nodo("Solna Stadion"), agregar_nodo("Solna Business Park"),
57.         agregar_nodo("Karlsbodavagen"), agregar_nodo("Trekanten"),
58.         agregar_nodo("Arstadal"), agregar_nodo("Arstaberg"),
59.         agregar_nodo("Valla Torg"), agregar_nodo("Luma"),
60.         agregar_nodo("Sickla Udde"), agregar_nodo("Sickla"),
61.         agregar_nodo("Stockholm Sodra")
62.     ]
63.
64.     # Las aristas tendran como atributos en origen, destino, la g, la h y la
linea a la que pertenece
65.     edges = [
66.         agregar_arista(G, "Kungstradgarden", "T-Centralen", 570.65, 574,
"Azul"),
67.         agregar_arista(G, "T-Centralen", "Radhuset", 1001, 990, "Azul"),
68.         agregar_arista(G, "Radhuset", "Fridhemsplan", 983.52, 952.28, "Azul"),
69.         agregar_arista(G, "Fridhemsplan", "Stadshagen", 1118, 1115, "Azul"),
70.         agregar_arista(G, "Stadshagen", "Vastra Skogen", 1120, 1118, "Azul"),
71.         agregar_arista(G, "Vastra Skogen", "Solna Centrum", 1410, 1330, "Azul"),
72.         agregar_arista(G, "Vastra Skogen", "Huvudsta", 1130, 1120, "Azul"),
73.         agregar_arista(G, "Huvudsta", "Solna Strand", 876.02, 859.76, "Azul"),
74.         agregar_arista(G, "Solna Strand", "Sundbybergs Centrum", 806.14, 790.85,
"Azul"),
75.         agregar_arista(G, "Abrahamsberg", "Stora Mossen", 715.91, 715.91,
"Verde"),
76.         agregar_arista(G, "Stora Mossen", "Alvik", 980.96, 906.67, "Verde"),
77.         agregar_arista(G, "Alvik", "Kristineberg", 1220, 1190, "Verde"),
78.         agregar_arista(G, "Kristineberg", "Thorildsplan", 704.64, 696.74,
"Verde"),
79.         agregar_arista(G, "Thorildsplan", "Fridhemsplan", 942.02, 926.57,
"Verde"),
80.         agregar_arista(G, "Fridhemsplan", "S:t Eriksplan", 844.16, 842.72,
"Verde"),
81.         agregar_arista(G, "S:t Eriksplan", "Odenplan", 937.32, 816.62, "Verde"),
82.         agregar_arista(G, "Odenplan", "Radmansgatan", 657.47, 592.1, "Verde"),
83.         agregar_arista(G, "Radmansgatan", "Hotorget", 612.52, 611.89, "Verde"),
84.         agregar_arista(G, "Hotorget", "T-Centralen", 374.72, 367.65, "Verde"),
85.         agregar_arista(G, "T-Centralen", "Gamla Stan", 1220, 1020, "Verde"),
86.         agregar_arista(G, "Gamla Stan", "Slussen", 498.97, 496.7, "Verde"),
87.         agregar_arista(G, "Slussen", "Medborgarplatsen", 566.99, 565.31,
"Verde"),
88.         agregar_arista(G, "Medborgarplatsen", "Skanstull", 739.74, 739.63,
"Verde"),
89.         agregar_arista(G, "Skanstull", "Gullmarsplan", 1021, 1020, "Verde"),
90.         agregar_arista(G, "Gullmarsplan", "Globen", 615.4, 583.88, "Verde"),
91.         agregar_arista(G, "Gullmarsplan", "Skarmirbrink", 686.07, 655.42,
"Verde"),
92.         agregar_arista(G, "Ropsten", "Gardet", 1210, 1060, "Roja"),
93.         agregar_arista(G, "Gardet", "Karlaplan", 1100, 1050, "Roja"),
94.         agregar_arista(G, "Karlaplan", "Ostermalmstorg", 1230, 1190, "Roja"),
95.         agregar_arista(G, "Bergshmarma", "Universitet", 2150, 2050, "Roja"),
96.         agregar_arista(G, "Universitet", "Tekniska Hogskolan", 2500, 2400,
"Roja"),
97.         agregar_arista(G, "Tekniska Hogskolan", "Stadion", 660, 638, "Roja"),
98.         agregar_arista(G, "Stadion", "Ostermalmstorg", 1240, 987.86, "Roja"),
99.         agregar_arista(G, "Ostermalmstorg", "T-Centralen", 954.87, 872.99,
"Roja"),
100.        agregar_arista(G, "T-Centralen", "Gamla Stan", 1220, 1020, "Roja"),
101.        agregar_arista(G, "Gamla Stan", "Slussen", 498.97, 496.7, "Roja"),
102.        agregar_arista(G, "Slussen", "Mariatorget", 1070, 856.7, "Roja"),
103.        agregar_arista(G, "Mariatorget", "Zinkensdamm", 404.28, 404.28, "Roja"),
104.        agregar_arista(G, "Zinkensdamm", "Liljeholmen", 1890, 1810, "Roja"),
```

```
105.         agregar_arista(G, "Liljeholmen", "Aspudden", 1280, 1260, "Roja"),
106.         agregar_arista(G, "Aspudden", "Axelsberg", 1575, 1570, "Roja"),
107.         agregar_arista(G, "Liljeholmen", "Midsommarkransen", 1120, 1080,
"Roja"),
108.         agregar_arista(G, "Midsommarkransen", "Telefonplan", 962.02, 937.02,
"Roja"),
109.         agregar_arista(G, "Solna Stadion", "Solna Centrum", 771.8, 715.89,
"Naranja"),
110.         agregar_arista(G, "Solna Centrum", "Solna Business Park", 1180, 1000,
"Naranja"),
111.         agregar_arista(G, "Solna Business Park", "Sundbyberg Centrum", 645.49,
492.55, "Naranja"),
112.         agregar_arista(G, "Sundbyberg Centrum", "Karlsbodavagen", 1070, 736.13,
"Naranja"),
113.         agregar_arista(G, "Karlsbodavagen", "Alvik", 2910, 2701, "Naranja"),
114.         agregar_arista(G, "Alvik", "Trekanten", 3590, 3070, "Naranja"),
115.         agregar_arista(G, "Trekanten", "Liljeholmen", 630.5, 553.12, "Naranja"),
116.         agregar_arista(G, "Liljeholmen", "Arstadal", 644.28, 527.64, "Naranja"),
117.         agregar_arista(G, "Arstadal", "Arstaberg", 797.81, 766.18, "Naranja"),
118.         agregar_arista(G, "Arstaberg", "Valla Torg", 1200, 1160, "Naranja"),
119.         agregar_arista(G, "Valla Torg", "Globen", 1680, 1610, "Naranja"),
120.         agregar_arista(G, "Globen", "Gullmarsplan", 615.4, 583.88, "Naranja"),
121.         agregar_arista(G, "Gullmarsplan", "Luma", 1300, 1050, "Naranja"),
122.         agregar_arista(G, "Luma", "Sickla Udde", 1020, 754.08, "Naranja"),
123.         agregar_arista(G, "Sickla Udde", "Sickla", 855.42, 738.29, "Naranja"),
124.         agregar_arista(G, "Sundyberg Centrum", "Odenplan", 5040, 4920, "Rosa"),
125.         agregar_arista(G, "Solna Stadion", "Odenplan", 3600, 3320, "Rosa"),
126.         agregar_arista(G, "Odenplan", "T-Centralen", 1700, 1450, "Rosa"),
127.         agregar_arista(G, "T-Centralen", "Stockholm Sodra", 2150, 1860, "Rosa"),
128.         agregar_arista(G, "Stockholm Sodra", "Arstaberg", 2700, 2540, "Rosa")
129.     ]
130.
131.     return G, nodes
132.
133. # Funcion auxiliar para paras de distancias a tiempos.
134. # Para ser lo mas concreto posible, se calcula el tiempo empleado en recorrer
una distancia dependiendo de la linea en la que esta
135. def convertir_distancia_a_tiempo(distancia, linea):
136.     velocidades = {
137.         "Azul": 40.93,
138.         "Roja": 35.92,
139.         "Verde": 29.60,
140.         "Naranja": 60,
141.         "Rosa": 60
142.     }
143.     velocidad = velocidades[linea]  # Obtener la velocidad para la línea dada
144.     distancia=distancia / 1000
145.     tiempo_horas = distancia / velocidad  # Tiempo en horas
146.     tiempo_minutos = tiempo_horas * 60  # Convertir a minutos
147.     return tiempo_minutos
148.
149. # Metodo para añadir una penalizacion si se cambia de una linea de metro a otra
de tren y viceversa
150. def calcular_penalizacion_distancia(linea_actual, linea_nueva):
151.     # Definir las líneas de tren y metro
152.     lineas_tren = {"Naranja", "Rosa"}
153.     lineas_metro = {"Verde", "Roja", "Azul"}
154.
155.     # Verificar si hay un transbordo entre tren y metro
156.     if (linea_actual in lineas_tren and linea_nueva in lineas_metro) or
(linea_actual in lineas_metro and linea_nueva in lineas_tren):
157.         return 1000
158.     return 0
159.
160. # Funcion para calcular el tiempo de transbordos. Tambien se tiene en cuenta si
se esta en horarios de fuera de servicio
```

```
161. def calcular_tiempo_espera(hora_actual, dia_actual, linea,horarios,cont):
162.     dia_actual=conversor_dia(dia_actual)
163.     horarios_linea = horarios['lineas'][linea]['dias'][dia_actual]
164.     hora_salida_dt = datetime.strptime(hora_actual.strftime('%H:%M'), '%H:%M')
165.
166.     # Franjas horarias en las que no hay servicio
167.     sin_servicio = {
168.         "Azul": [("00:51", "05:06")],
169.         "Roja": [("00:45", "05:00")],
170.         "Verde": [("00:31", "05:16")],
171.         "Naranja": [("01:51", "05:30")],
172.         "Rosa": [("01:51", "05:30")]
173.     }
174.
175.     # Si es fin de semana, actualizar las franjas horarias de las líneas Naranja
y Rosa
176.     if dia_actual in ["Saturday", "Sunday"]:
177.         sin_servicio["Naranja"] = [("02:21", "06:00")]
178.         sin_servicio["Rosa"] = [("02:21", "06:00")]
179.
180.     # Verificar si la hora_salida está dentro de alguna franja sin servicio
181.     for franja in sin_servicio[linea]:
182.         inicio_franja = datetime.strptime(franja[0], '%H:%M').time()
183.         fin_franja = datetime.strptime(franja[1], '%H:%M').time()
184.         if inicio_franja <= hora_salida_dt.time() <= fin_franja:
185.             # Calcular el tiempo hasta el próximo servicio
186.             fin_franja_dt = datetime.combine(hora_salida_dt.date(), fin_franja)
187.             tiempo_espera = (fin_franja_dt - hora_salida_dt).total_seconds() /
60
188.             if cont == 0:
189.                 return int(tiempo_espera), fin_franja_dt
190.             else:
191.                 return int(tiempo_espera),hora_salida_dt
192.
193.     for horario in horarios_linea:
194.         horario_dt = datetime.strptime(horario, '%H:%M')
195.         if horario_dt.time() > hora_salida_dt.time():
196.             tiempo_espera = (horario_dt - hora_salida_dt).total_seconds() / 60
197.             if cont == 0:
198.                 return int(tiempo_espera),hora_salida_dt
199.             return int(tiempo_espera),hora_salida_dt
200.
201.     return None
202.
203. # Funcion auxiliar para traducir los nombres de la semana
204. def conversor_dia(dia):
205.     if dia == "Monday":
206.         dia = "Lunes"
207.     if dia == "Tuesday":
208.         dia ="Martes"
209.     if dia == "Wednesday":
210.         dia ="Miercoles"
211.     if dia == "Thursday":
212.         dia ="Jueves"
213.     if dia == "Friday":
214.         dia ="Viernes"
215.     if dia == "Saturday":
216.         dia ="Sabado"
217.     if dia == "Sunday":
218.         dia ="Domingo"
219.     return dia
220.
221. # Funcion principal del algoritmo A estrella
222. def a_estrella(G, start, end):
223.     open_list = []
224.     closed_list = set()
```

```
    # node_map se utilizara para comparar mejor que nodos tienen mejor g y actualizar eficientemente la lista de abiertos
    node_map = {}

    start_node = Node(name=start, gAcumulada=0)
    end_node = Node(name=end)

    # Agregar el nodo inicial a la open_list con prioridad basada en su valor 'f'
    heapq.heappush(open_list, (start_node.f, start_node))

    node_map[start_node.name] = start_node

    while open_list:
        # Extraer el nodo con el valor 'f' más bajo de la open_list
        _, current_node = heapq.heappop(open_list)
        node_map.pop(current_node.name, None)

        if current_node.name in closed_list:
            continue

        closed_list.add(current_node.name)

        if current_node.name == end_node.name:
            path = []
            while current_node:
                path.append(current_node)
                current_node = current_node.padre
            return path[::-1]

        for neighbor in G.neighbors(current_node.name):
            if neighbor in closed_list:
                continue

            # g es la distancia entre current_node y neighbor
            g = G[current_node.name][neighbor]['g']
            linea = G[current_node.name][neighbor]['l']
            # h es la estimación heurística desde neighbor hasta el objetivo, ajustada por la penalización
            # G.nodes[neighbor] devuelve todos los atributos de neighbor
            if neighbor == end:
                heuristica = 0
            else:
                heuristica = G[current_node.name][neighbor]['h']

            # Calcular la penalización por transbordo
            penalizacion = calcular_penalizacion_distancia(current_node.linea, linea)
            h = heuristica + penalizacion
            # gAcumulada es la distancia total acumulada desde el nodo inicial hasta el vecino actual
            gAcumulada = current_node.gAcumulada + g

            f = gAcumulada + h  # Coste total estimado

            neighbor_node = Node(
                name=neighbor,
                f=f,
                g=g,
                h=h,
                padre=current_node,
                linea=linea,
                gAcumulada=gAcumulada
            )

```

```
285.             # Comprobar que si hay un nodo que ya ha sido añadido a los abiertos
con una g mayor al que estamos añadiendo, eliminar el anterior
286.             if neighbor in node_map and node_map[neighbor].g > g:
287.                 open_list.remove((node_map[neighbor].f, node_map[neighbor]))
288.                 heapq.heapify(open_list)
289.
290.             if neighbor not in node_map or node_map[neighbor].g > g:
291.                 heapq.heappush(open_list, (neighbor_node.f, neighbor_node))
292.                 node_map[neighbor] = neighbor_node
293.
294.     return None
295.
296.
297. app = Flask(__name__)
298.
299. app.secret_key = 'metro'
300.
301. @app.route('/')
302. def root():
303.     return render_template('inicio.html')
304.
305. @app.route('/paradas', methods=['POST'])
306. def paradas():
307.     # Obtencion de los distintos parametros de la pestaña crear_ruta
308.     origen = request.form['origen']
309.     destino = request.form['destino']
310.     hora_salida = request.form.get('hora')
311.     dia = request.form.get('dia')
312.
313.     G, nodes = crear_grafo()
314.
315.     indice_origen = next((index for (index, d) in enumerate(nodes) if d.name ==
origen), None)
316.     indice_destino = next((index for (index, d) in enumerate(nodes) if d.name ==
destino), None)
317.
318.     error_message = {}
319.     if indice_origen is None:
320.         error_message['origen'] = 'El origen no fue encontrado'
321.     if indice_destino is None:
322.         error_message['destino'] = 'El destino no fue encontrado'
323.
324.     # Si no se ha proporcionado un origen o un destino
325.     if error_message:
326.         nombres_parada = [n.name for n in nodes]
327.         data = {
328.             'nombres_paradas': nombres_parada
329.         }
330.         return render_template('crear_ruta.html', error=error_message,
data=data)
331.
332.
333.     tiempos_transbordo = []
334.     tiempo_total_transbordos = 0
335.
336.     # Introducir hora y dia es opcional. En caso de no introducir nada sera la
actual
337.     if hora_salida:
338.         hora_salida_dt = datetime.strptime(hora_salida, '%H:%M')
339.     else:
340.         hora_salida_dt = datetime.now()
341.
342.     if not dia:
343.         dia_actual = hora_salida_dt.strftime('%A')
344.     else:
345.         dia_actual = dia
```

```
346.
347.     ruta = a_estrella(G, origen, destino)
348.
349.     if not ruta:
350.         error_message['ruta'] = 'No se encontró una ruta entre el origen y el
destino'
351.         nombres_parada = [n.name for n in nodes]
352.         data = {
353.             'nombres_paradas': nombres_parada
354.         }
355.         return render_template('crear_ruta.html', error=error_message,
data=data)
356.
357.     ruta[0].linea = ruta[1].linea
358.
359.
360.     # Obtener la ruta del directorio del script
361.     base_dir = os.path.dirname(os.path.abspath(__file__))
362.
363.     # Construir la ruta completa al archivo JSON
364.     json_path = os.path.join(base_dir, 'horarios.json')
365.
366.     # Cargar el archivo JSON con los horarios
367.     with open(json_path, 'r') as file:
368.         horarios = json.load(file)
369.
370.     # Calculo de los transbordos
371.     cont = 0
372.     for i, nodo in enumerate(ruta):
373.         # La primera iteracion no calcula transbordos sino que proporciona
cuanto queda para el primer tren que tiene que coger
374.         if i == 0:
375.             tiempo_espera, primer_fin_franja_dt =
calcular_tiempo_espera(hora_salida_dt, dia_actual, nodo.linea, horarios, cont)
376.             if tiempo_espera > 59:
377.                 horas = tiempo_espera/60
378.                 minutos = (horas - int(horas)) * 60
379.                 if minutos == 0:
380.                     tiempos_transbordo.append(f"Salir desde {ruta[i].name}.
Tiempo de espera estimado: {int(horas)} horas")
381.                 else:
382.                     tiempos_transbordo.append(f"Salir desde {ruta[i].name}.
Tiempo de espera estimado: {int(horas)} horas y {int(minutos)} minutos")
383.             else:
384.                 tiempos_transbordo.append(f"Salir desde {ruta[i].name}. Tiempo
de espera estimado: {tiempo_espera} minutos")
385.             tiempo_total_transbordos += tiempo_espera
386.         # Calculo transbordos
387.         if i > 0 and nodo.linea != ruta[i - 1].linea:
388.             # El primer tramo se hará con la hora desde que salio el primer tren
389.             if cont == 0:
390.                 tiempo_espera, fin_franja_dt =
calcular_tiempo_espera(primer_fin_franja_dt, dia_actual, nodo.linea, horarios, cont)
391.             # El segundo transbordo se calcula con la hora anterior y asi
sucesivamente
392.             else:
393.                 tiempo_espera,fin_franja_dt=
calcular_tiempo_espera(fin_franja_dt, dia_actual, nodo.linea, horarios, cont)
394.             if tiempo_espera > 59:
395.                 horas = tiempo_espera/60
396.                 minutos = (horas - int(horas)) * 60
397.                 if minutos == 0:
398.                     tiempos_transbordo.append(f"Transbordo en {ruta[i-1].name} a
la línea {nodo.linea}. Tiempo de espera estimado: {int(horas)} horas ")
399.                 else:
```

```
400.                     tiempos_transbordo.append(f"Transbordo en {ruta[i-1].name} a
la línea {nodo.linea}. Tiempo de espera estimado: {int(horas)} horas y {int(minutos)}
minutos")
401.             else:
402.                 tiempos_transbordo.append(f"Transbordo en {ruta[i-1].name} a la
línea {nodo.linea}. Tiempo de espera estimado: {tiempo_espera} minutos")
403.             tiempo_total_transbordos += tiempo_espera
404.             cont += 1
405.
406.
407.     tiempo=0
408.     for nodo in ruta:
409.         tiempo = tiempo + convertir_distancia_a_tiempo(nodo.g,nodo.linea)
410.     tiempo_total = tiempo + tiempo_total_transbordos
411.     distanciaTotal = ruta[-1].gAcumulada / 1000
412.
413.     tiempoaux = tiempo
414.     segundos = tiempoaux - int(tiempoaux)
415.     segundos = segundos * 60
416.     segundos = int(segundos)
417.     horas = tiempo / 60
418.     minutos_horas = horas - int(horas)
419.     minutos_horas = minutos_horas * 60
420.     distanciaTotal = round(distanciaTotal, 2)
421.
422.     tiempoaux2 = tiempo_total
423.     segundos_total = tiempoaux2 - int(tiempoaux2)
424.     segundos_total = segundos_total * 60
425.     segundos_total = int(segundos_total)
426.     horas_total = tiempo_total / 60
427.     minutos_horas_total = horas_total - int(horas_total)
428.     minutos_horas_total = minutos_horas_total * 60
429.
430.     # En caso de que tiempo total de transbordos sea mayor que 60 darle un valor
de horas y minutos
431.     tiempo_total_transbordos_horas=tiempo_total_transbordos/60
432.     tiempo_total_transbordos_minutos=(tiempo_total_transbordos_horas -
int(tiempo_total_transbordos_horas))*60
433.
434.
435.     tiempo_estimado = timedelta(minutes=int(tiempo_total), seconds=segundos)
436.     hora_llegada_dt = (hora_salida_dt + tiempo_estimado).time()
437.     hora_llegada = hora_llegada_dt.strftime('%H:%M')
438.
439.     dia_actual=conversor_dia(dia_actual)
440.     data = {
441.         'numero_elem': len(ruta),
442.         'ruta': [{'name': nodo.name, 'linea': nodo.linea} for nodo in ruta],
443.         'minutos': int(tiempo),
444.         'segundos': segundos,
445.         'horas': int(horas),
446.         'minutos_horas': int(minutos_horas),
447.         'distancia': distanciaTotal,
448.         'hora_llegada': hora_llegada,
449.         'tiempos_transbordo': tiempos_transbordo,
450.         'tiempo_total_transbordos' : tiempo_total_transbordos,
451.         'hora_salida':hora_salida_dt.strftime('%H:%M'),
452.         'dia_actual' : dia_actual,
453.
454.         'segundos_total' : segundos_total,
455.         'horas_total' : int(horas_total),
456.         'minutos_horas_total' : int(minutos_horas_total),
457.         'minutos_total' : int(tiempo_total),
458.         'tiempo_total_transbordos_horas' : int(tiempo_total_transbordos_horas),
459.         'tiempo_total_transbordos_minutos' :
int(tiempo_total_transbordos_minutos),
```

```
460.
461.         'fin_franja' : primer_fin_franja_dt.strftime('%H:%M')
462.     }
463.
464.     session['data'] = json.dumps(data)
465.     return render_template('paradas.html', data=data)
466.
467. @app.route('/crear_ruta')
468. def crear_ruta():
469.     G, nodes = crear_grafo()
470.     nombres_parada = sorted([n.name for n in nodes])  # Ordenar alfabéticamente
471.     hora_actual = time.strftime('%H:%M')
472.     data = {
473.         'nombres_paradas': nombres_parada,
474.         'hora_actual': hora_actual
475.     }
476.     return render_template('crear_ruta.html', data=data)
477.
478. @app.route('/paradas')  # Cuando no se ha seleccionado una ruta
479. def paradas_sin_cont():
480.     lista_nombres = []
481.     data = {
482.         'lista': lista_nombres,
483.         'numero_elem': len(lista_nombres)
484.     }
485.     return render_template('paradas.html', data=data)
486.
487. @app.route('/paradas_pdf')
488. def download_pdf():
489.     data = json.loads(session['data'])
490.
491.     # Ruta absoluta de la imagen
492.     image_path = url_for('static', filename='images/mapa-estocolmo.png')
493.
494.     # Renderiza la plantilla HTML, pasando la ruta de la imagen
495.     rendered_html = render_template('paradas_pdf.html', data=data,
image_path=image_path)
496.
497.     html = HTML(string=rendered_html, base_url=request.url_root)
498.
499.     # Genera el PDF
500.     pdf = html.write_pdf(stylesheets=[CSS(string='''
501.         @page { size: A4; margin: 0; }
502.         body { margin: 0; padding: 0; }
503.     ''')])
504.
505.     # Crea una respuesta y envía el archivo PDF generado
506.     response = make_response(pdf)
507.     response.headers['Content-Type'] = 'application/pdf'
508.     response.headers['Content-Disposition'] = 'inline;
filename=paradas_y_mapa.pdf'
509.
510.     return response
511.
512. @app.before_request
513. def make_session_permanent():
514.     session.permanent = True
515.     app.permanent_session_lifetime = timedelta(days=7)
516.
517. if __name__ == '__main__':
518.     G, nodes = crear_grafo()
519.     app.run(debug=True, port=5000)
```

6.2 Anexo B. Archivo inicio.html

Código fuente perteneciente a la pestaña inicio

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Paradas del Metro de Estocolmo</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/layout.css') }}">

</head>
<body>
    <div class="navbar">
        <img src="{{ url_for('static', filename='images/icono.png') }}" alt="Icono">
        <a href="/paradas">Mi ruta</a>
        <a href="/crear_ruta">Mapa</a>
        <a href="/">Inicio</a>
    </div>

    <h1>Descubre las paradas más icónicas del Metro de Estocolmo</h1>

    <div class="blog-container">
        <button class="nav-button left" onclick="slide(-1)">❮</button>
        <div class="card-wrapper" id="card-wrapper">
            <!-- T-Centralen -->
            <div class="post">
                <h2 id="t-centralen">1. T-Centralen (Línea 10/11/13/17/44)</h2>
                <p>La emblemática T-Centralen, primera estación adornada en 1957 y
cruce de las principales líneas de metro.
                    Más de 20 artistas han contribuido a su decoración, incluyendo a
Per Olof Ultvert, cuyas obras capturan la
                    esencia de la flora y la laboriosidad humana.</p>
                <img src="{{ url_for('static', filename='images/T-Centralen.png') }}"
alt="T-Centralen Station">
            </div>

            <!-- Kungsträdgarden -->
            <div class="post">
                <h2 id="kungstradgarden">2. Kungsträdgarden (Línea 10/11)</h2>
                <p>Terminal de las líneas azules, Kungsträdgarden, "el jardín del
rey", se ubica junto al histórico jardín real. Decorada en los 70 por Ulrik Samuelson,
esta estación es un jardín subterráneo lleno de arte que rememora su historia real
                    con esculturas del desaparecido Palacio Makalös y referencias a
un foro romano.</p>
                <img src="{{ url_for('static', filename='images/kungstradgarden.jpg')
}}" alt="Kungsträdgarden Station">
            </div>

            <!-- Radhuset -->
            <div class="post">
                <h2 id="radhuset">3. Radhuset (Línea 10/11)</h2>
                <p>Radhuset, inaugurada en 1975 y situada junto al Palacio de
Justicia, es famosa por sus paredes de roca natural.
                    Sus tonos rojizos, elegidos por el artista Sigvard Olsson, evocan
las montañas distantes. Su cercanía al centro político
                (T-Centralen) la hacen una de las paradas más transcurridas.</p>
                <img src="{{ url_for('static', filename='images/radhuset.jpg') }}"
alt="Radhuset Station">
            </div>

            <!-- Stadion -->
            <div class="post">
                <h2 id="stadion">4. Stadion (Línea 13)</h2>
                <p>En la línea roja, la estación Stadion, creada por Enno Hallek y
Ake Pallarp en 1973,
```

```
                    luce un vibrante azul cielo con un arcoíris que simboliza los
Juegos Olímpicos de 1912,
                    celebrados en el estadio adyacente. Es una verdadera obra de arte
que hace referencia al espíritu olímpico.</p>
                <img src="{{ url_for('static', filename='images/stadion.jpg') }}"
alt="Stadion Station">
            </div>

            <!-- Tekniska Högskolan -->
            <div class="post">
                <h2 id="tekniska-hogskolan">5. Tekniska Högskolan (Línea 13)</h2>
                <p>Cerca del Instituto Real de Tecnología, Tekniska Högskolan,
abierta en 1973,
                    es un homenaje a la ciencia. Obra de Lennart Mörk,
                    la estación presenta los cinco elementos y avances científicos
significativos como las leyes de Newton y el
                    heliocentrismo de Copérnico.</p>
                <img src="{{ url_for('static', filename='images/teknsika
hogskolan.jpg') }}" alt="Tekniska Högskolan Station">
            </div>

            <!-- Fridhemsplan -->
            <div class="post">
                <h2 id="fridhemsplan">6. Fridhemsplan (Línea 17)</h2>
                <p>La siguiente estación es Fridhemsplan.
                    Este es un ejemplo clásico de las muchas estaciones tipo cueva en
Estocolmo, excavadas directamente en la roca y
                    destacadas con colores vivos en lugar de ocultar su esencia
natural. En Fridhemsplan, las paredes están decoradas en
                    tonos verdes, un posible guiño a una de las líneas que pasan por
esta estación. Recientemente reformada en 2019,
                    la estación combina modernidad con un toque de naturaleza.</p>
                <img src="{{ url_for('static', filename='images/fridhemsplan.jpg')
}}" alt="Fridhemsplan Station">
            </div>
        </div>
        <button class="nav-button right" onclick="slide(1)">></button>
    </div>

    <script>
        let currentIndex = 0;

        function slide(direction) {
            const cardWrapper = document.getElementById('card-wrapper');
            const posts = document.querySelectorAll('.post');
            const totalPosts = posts.length;
            const visibleCards = 3;
            const cardWidth = posts[0].offsetWidth + 20; // Card width + margin

            currentIndex += direction;

            if (currentIndex < 0) {
                currentIndex = 0;
            } else if (currentIndex > totalPosts - visibleCards) {
                currentIndex = totalPosts - visibleCards;
            }

            const newTransformValue = -(currentIndex * cardWidth);
            cardWrapper.style.transform = `translateX(${newTransformValue}px)`;
        }
    </script>
</body>
</html>

```

6.3 Anexo C. Archivo crear_ruta.html

Código fuente perteneciente a la pestaña crear_ruta

```
<!DOCTYPE html>
<html>
<head>
    <title>Mapa</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/layout.css') }}">

</head>

<body>
    <div class="navbar">
        <img src="{{ url_for('static', filename='images/icono.png') }}" alt="Icono"
style="float: left; width: 40px; height: 40px;">
        <a href="/paradas">Mi ruta</a>
        <a href="/crear_ruta">Mapa</a>
        <a href="/">Inicio</a>
    </div>
        <h1> Bienvenido/a al Mapa de Estocolmo</h1>
    <div class="inicio-container">
        <p class="intro-texto">
            Esta aplicación te permite explorar y planificar tus rutas entre las
diferentes paradas del metro de Estocolmo de manera eficiente y fácil.
            Simplemente selecciona tu parada de origen y tu destino deseado para ver
las mejores opciones de ruta. Ideal para residentes y visitantes que desean moverse por la
ciudad con confianza.
        </p>
    </div>
    <div class="main-container">
        <div class="sidebar">
            <h2>Información a tener en cuenta:</h2>
            <p>Las líneas 22 y 44 (naranja y rosa respectivamente) son líneas de tren.
El recorrido puede parecer más rápido si se usan estas vías,
                pero el transbordo de metro a tren y viceversa puede afectar al tiempo
del trayecto real.
            </p>
        </div>

        <div class="contenido-inicio">
            <div class="imagen-lado">
                <img src="{{ url_for('static', filename='images/mapa-estocolmo.png')
}}" alt="Mapa del metro de Estocolmo">
            </div>
        </div>

        <div class="formulario-destino">
            <h3>Planifica tu ruta:</h3>
            <form action="/paradas" method="post">
                <label for="orig">Origen:</label>
                <input type="text" id="orig" name="origen" list="nombres_paradas"
value="{{ request.form.origen if request.form.origen else '' }}">
                {% if 'origen' in error %}
                    <span class="error-message">{{ error['origen'] }}</span>
                {% endif %}

                <label for="dest">Destino:</label>
                <input type="text" id="dest" name="destino" list="nombres_paradas"
value="{{ request.form.destino if request.form.destino else '' }}">
                {% if 'destino' in error %}
                    <span class="error-message">{{ error['destino'] }}</span>
                {% endif %}
                <p text-align="center">En caso de no elegir hora de salida o día de la
semana, será la actual.
                </p>
                <label for="hora">Hora de salida:</label>
```

```
54.                 <input type="time" id="hora" name="hora" list="horas_lista" value="{{
request.form.hora if request.form.hora else '' }}">
55.                 <datalist id="horas_lista">
56.                     {% for h in range(5, 25) %}
57.                         <option value="{{ '%02d:00'|format(h%24) }}">
58.                     {% endfor %}
59.                     <option value="01:00">
60.                 </datalist>
61.
62.                 <label for="dia">Día de la semana:</label>
63.                 <select id="dia" name="dia" value="{{ request.form.dia if
request.form.dia else '' }}">
64.                     <option value="">Seleccionar día</option>
65.                     <option value="Monday">Lunes</option>
66.                     <option value="Tuesday">Martes</option>
67.                     <option value="Wednesday">Miércoles</option>
68.                     <option value="Thursday">Jueves</option>
69.                     <option value="Friday">Viernes</option>
70.                     <option value="Saturday">Sábado</option>
71.                     <option value="Sunday">Domingo</option>
72.                 </select>
73.
74.                 <datalist id="nombres_paradas">
75.                     {% for c in data.nombres_paradas %}
76.                         <option value="{{ c }}">
77.                     {% endfor %}
78.                 </datalist>
79.
80.                 <input type="submit" value="Enviar" />
81.             </form>
82.         </div>
83.     </div>
84. </body>
85. </html>
86.
```

6.4 Anexo D. Archivo paradas.html

Código fuente perteneciente a la pestaña paradas

```
<!DOCTYPE html>
<html>
<head>
    <title>Mi ruta</title>
    <link rel="stylesheet" href="{{ url_for('static',filename='css/layout.css') }}">
    <script type="text/javascript">
        function confirmarVolver() {
            return confirm("Al darle a 'Volver', se perderá la ruta elegida. ¿Desea continuar?");
        }
    </script>
</head>
<body>

    <div class="navbar">
        <img src="{{ url_for('static', filename='images/icono.png') }}" alt="Icono" style="float: left; width: 40px; height: 40px;">
        <a href="/paradas">Mi ruta</a>
        <a href="/crear_ruta">Mapa</a>
        <a href="/">Inicio</a>
    </div>

    <div class="paradas-container">
        <div class="sidebar_paradas">
            {% if data.numero_elem > 0 %}
            <div class="container_paradas">
                <h1>Paradas en tu ruta:</h1>
                <ul class="lista-paradas">
                    {% for c in data.ruta %}
                        <li class="{{ c.linea }}">{{ c.name }}</li>
                    {% endfor %}
                </ul>


                <div class="info-bloque">
                    {%if data.minutos > 59 %}
                        <p>Tiempo estimado de la ruta: <span class="info-detalle">{{ data.horas }} hora y {{ data.minutos_horas }} minutos</span></p>
                        <p>Distancia total: <span class="info-detalle">{{ data.distancia }} km</span></p>

                    {% else %}
                        <p>Tiempo estimado de la ruta: <span class="info-detalle">{{ data.minutos }} minutos</span></p>
                        <p>Distancia total: <span class="info-detalle">{{ data.distancia }} km</span></p>
                    {% endif %}
                </div>
                {%if data.tiempo_total_transbordos > 1 %}
                    <div class="info-bloque">
                        <h3>Tiempos de transbordo aproximado:</h3>
                        <ul>
                            {% for t in data.tiempos_transbordo %}
                                <li>{{ t }}</li>
                            {% endfor %}
                        </ul>
                        {%if data.tiempo_total_transbordos > 59 %}
                            <p>Tiempo total haciendo transbordos: <span class="info-detalle">{{ data.tiempo_total_transbordos_horas }} horas y {{ data.tiempo_total_transbordos_minutos }} minutos. </span></p>
```

```
                            <span class="error-message2">Los trenes no se encuentran en servicio en este momento. Hora de salida del próximo tren: {{ data.fin_franja}}</span>
                        {% else %}
                            <p>Tiempo total haciendo transbordos: <span class="info-detalle">{{ data.tiempo_total_transbordos }} minutos. </span></p>
                        {% endif %}
                    </div>
                    <div class="info-bloque">
                        {%if data.minutos_total > 59 %}
                            <p>Tiempo total de viaje: <span class="info-detalle">{{ data.horas_total }} hora y {{ data.minutos_horas_total }} minutos </span></p>
                            <p>Hora de salida: <span class="info-detalle">{{ data.hora_salida }}</span></p>
                            <p>Día de salida: <span class="info-detalle">{{ data.dia_actual }}</span></p>
                            <p>Hora aproximada de llegada: <span class="info-detalle">{{ data.hora_llegada }}</span></p>
                        {% else %}
                            <p>Tiempo total de viaje: <span class="info-detalle">{{ data.minutos_total }} minutos</span></p>
                            <p>Hora de salida: <span class="info-detalle">{{ data.hora_salida }}</span></p>
                            <p>Día de salida: <span class="info-detalle">{{ data.dia_actual }}</span></p>
                            <p>Hora aproximada de llegada: <span class="info-detalle">{{ data.hora_llegada }}</span></p>
                        {% endif %}

                    </div>
                {% else %}
                    <div class="info-bloque">
                        <p>Hora de salida: <span class="info-detalle">{{ data.hora_salida }}</span></p>
                        <p>Día de salida: <span class="info-detalle">{{ data.dia_actual }}</span></p>
                        <p>Hora aproximada de llegada: <span class="info-detalle">{{ data.hora_llegada }}</span></p>
                    </div>
                {% endif %}

                <div class="boton-container">
                    <a href="{{ url_for('crear_ruta') }}" class="boton" onclick="return confirmarVolver();">Volver</a>
                    <a href="{{ url_for('download_pdf') }}" class="boton" target="_blank">Guardar como PDF</a>
                </div>

            </div>
            {% else %}
            <div class="container_paradas">
                <h2>Todavía no has seleccionado un origen y destino.</h2>
            </div>
            {% endif %}
        </div>
        <div class="contenido-inicio">

            <div class="imagen-lado">
                <h1> Mapa del metro: </h1>
                <img src="{{ url_for('static', filename='images/mapa-estocolmo.png') }}" alt="Mapa de Estocolmo">
            </div>
        </div>
    </div>


</body>
</html>
```

6.5 Anexo E. Archivo paradas_pdf.html

Código fuente perteneciente a la pestaña paradas_pdf

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Ruta_pdf</title>
    <link rel="stylesheet" href="{{ url_for('static',filename='css/layout.css') }}">

</head>
<body>

    <div class="container_paradas">
        <ul class="lista-paradas">
            <h1>Paradas en tu Ruta</h1>
            {% for nodo in data.ruta %}
                <li class="{{ nodo.linea }}">{{ nodo.name }}</li>
            {% endfor %}
        </ul>

        <div class="info-bloque">
            {%if data.minutos > 59 %}
                <p>Tiempo estimado de la ruta: <span class="info-detalle">{{
data.horas }} hora y {{ data.minutos_horas }} minutos</span></p>
                <p>Distancia total: <span class="info-detalle">{{ data.distancia }}
km</span></p>

            {% else %}
                <p>Tiempo estimado de la ruta: <span class="info-detalle">{{
data.minutos }} minutos</span></p>
                <p>Distancia total: <span class="info-detalle">{{ data.distancia }}
km</span></p>
            {% endif %}
        </div>
        {%if data.tiempo_total_transbordos > 1 %}
            <div class="info-bloque">
                <h3>Tiempos de transbordo aproximado:</h3>
                <ul>
                    {% for t in data.tiempos_transbordo %}
                        <li>{{ t }}</li>
                    {% endfor %}
                </ul>
                {%if data.tiempo_total_transbordos > 59 %}
                    <p>Tiempo total haciendo transbordos: <span class="info-
detalle">{{ data.tiempo_total_transbordos_horas }} horas y {{
data.tiempo_total_transbordos_minutos }} minutos. </span></p>
                    <span class="error-message2">Los trenes no se encuentran en
servicio en este momento. Hora de salida del próximo tren: {{ data.fin_franja}}</span>
                {% else %}
                    <p>Tiempo total haciendo transbordos: <span class="info-
detalle">{{ data.tiempo_total_transbordos }} minutos. </span></p>
                {% endif %}
            </div>
            <div class="info-bloque">
                {%if data.minutos_total > 59 %}
                    <p>Tiempo total de viaje: <span class="info-detalle">{{
data.horas_total }} hora y {{ data.minutos_horas_total }} minutos </span></p>
                    <p>Hora de salida: <span class="info-detalle">{{ data.hora_salida
}}</span></p>
                    <p>Día de salida: <span class="info-detalle">{{ data.dia_actual
}}</span></p>
                    <p>Hora aproximada de llegada: <span class="info-detalle">{{
data.hora_llegada }}</span></p>
                {% else %}
                    <p>Tiempo total de viaje: <span class="info-detalle">{{
data.minutos_total }} minutos</span></p>
```

```
52.                     <p>Hora de salida: <span class="info-detalle">{{ data.hora_salida
}}</span></p>
53.                     <p>Día de salida: <span class="info-detalle">{{ data.dia_actual
}}</span></p>
54.                     <p>Hora aproximada de llegada: <span class="info-detalle">{{
data.hora_llegada }}</span></p>
55.                 {% endif %}
56.
57.             </div>
58.         {% else %}
59.             <div class="info-bloque">
60.                 <p>Hora de salida: <span class="info-detalle">{{ data.hora_salida
}}</span></p>
61.                 <p>Día de salida: <span class="info-detalle">{{ data.dia_actual
}}</span></p>
62.                 <p>Hora aproximada de llegada: <span class="info-detalle">{{
data.hora_llegada }}</span></p>
63.             </div>
64.         {% endif %}
65.
66.         <div class="contenido-inicio">
67.
68.             <div class="imagen-lado">
69.                 <h1> Mapa del metro: </h1>
70.                 <img src="{{ url_for('static', filename='images/mapa-estocolmo.png')
}}" alt="Mapa de Estocolmo">
71.             </div>
72.         </div>
73.     </div>
74. </body>
75. </html>
76.
```

6.6 Anexo F. Archivo layout.css

Código fuente correspondiente al estilo de la página web en CSS.

```css
body {
    font-family: Arial, sans-serif;
    background-color:rgba(40, 75, 118, 0.353);

    color: #333;
    margin: 0;
    padding: 20px;
}

.blog-container {
    width: 80%;
    margin: 20px auto;
    background-color:#f0f0f0;
    padding: 20px;
    box-shadow: 0 2px 5px rgba(0,0,0,0.1);
    border-radius: 10px;
}

.post {
    background-color: white;
    margin: 10px;
    padding: 20px;
    box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
    border-radius: 10px;
    min-width: calc(33.333% - 40px);
    box-sizing: border-box;
    display: flex;
    flex-direction: column;
    align-items: center;
    text-align: center;
}

.post h2 {
    font-size: 28px;
    margin-left: 15%;
    margin-right: 15%;
}

.post img {
    width: 80%;
    height: auto;
    margin-top: 10px;
}

.post p {
    font-size: 20px;
    color: #666;
    text-align: center;
    margin-left: 15%;
    margin-right: 15%;
}


#kungstradgarden, #t-centralen, #radhuset {
    color: #00008B;
}

#stadion, #tekniska-hogskolan {
    color: #8B0000;
}
#fridhemsplan {
    color: #006400;
}

```

```
65. .container {
66.     max-width: 600px;
67.     margin: auto;
68.     background: white;
69.     padding: 20px;
70.     box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
71.     align-items: center;
72. }
73.
74.
75. .lista-paradas {
76.     list-style: none;
77.     padding: 0;
78. }
79.
80. .lista-paradas li {
81.     padding: 10px;
82.     border-bottom: 1px solid #ddd;
83. }
84.
85. .lista-paradas li:last-child {
86.     border: none;
87. }
88.
89. .inicio-container {
90.     text-align: center;
91.     padding: 20px;
92.     background-color:rgba(255, 255, 255, 0.9);
93.     box-shadow: 0 2px 5px rgba(0,0,0,0.1);
94.     margin-bottom: 20px;
95.     margin-left: 20px;
96.     margin-right: 20px;
97.     border-radius: 10px;
98. }
99.
100.
101. .main-container {
102.     display: flex;
103.     justify-content: space-between;
104.     align-items: stretch;
105.     padding: 20px;
106.     gap: 20px;
107.
108. }
109.
110. .sidebar, .formulario-destino {
111.     position: static;
112.     width: 250px;
113.     background-color: rgba(255, 255, 255, 0.9);
114.     padding: 20px;
115.     height: 560px;
116.     box-shadow: 0 2px 5px rgba(0,0,0,0.1);
117.     display: flex;
118.     flex-direction: column;
119.     border-radius: 10px;
120.
121. }
122.
123. .contenido-inicio {
124.     flex-grow: 1;
125.     align-items: flex-start;
126.     align-items: center;
127.     justify-content: center;
128.     gap: 80px;
129.     padding: 20px;
130.     background-color: rgba(255, 255, 255, 0.9);
131.     box-shadow: 0 2px 5px rgba(0,0,0,0.1);
132.     border-radius: 10px;
133. }
134.
```

```
.imagen-lado img {
    width: 100%;
    max-width: 900px;

}


.boton {
    display: inline-block;
    background-color: #333;
    color: white;
    padding: 10px 20px;
    text-decoration: none;
    border-radius: 5px;
    font-size: 16px;
}

.boton:hover {
    background-color: #333;
}


.navbar {
    overflow: hidden;
    background-color: #333;
    font-family: Arial, sans-serif;
    border-radius: 10px;
}

.navbar img {
    float: left;
    width: 40px;
    height: 40px;
    margin-left: 2px;
    margin-top: 3px;
}

.navbar a {
    float: right;
    font-size: 16px;
    color: white;
    text-align: center;
    padding: 14px 20px;
    text-decoration: none;
}


.navbar a:hover {
    background-color: #ddd;
    color: black;
}


.campo {
    display: flex;
    flex-direction: column;
}

.campo label {
    font-weight: bold;
}

.campo input[type="text"] {
    padding: 5px;
    font-size: 16px;
}
input, select {
    display: block;
    width: 80%;
    margin: 10px auto;
```

```
205.     padding: 5px;
206. }
207.
208. label {
209.     display: block;
210.     text-align: center;
211.     margin: 10px auto;
212. }
213.
214. h1 {
215.     color: #333;
216.     font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
217.     font-size: 60px;
218.     text-align: center;
219. }
220.
221. .intro-texto {
222.     font-style: italic;
223.     margin-bottom: 20px;
224.     margin-left: 8%;
225.     margin-right: 8%;
226.     font-size: 25px;
227. }
228.
229. input[type="text"], input[list="nombres_paradas"] {
230.     width: 80%;
231.     padding: 8px;
232.     border: 1px solid #ccc;
233.     border-radius: 4px;
234.
235. }
236.
237. input[type="submit"] {
238.     background-color: #333;
239.     color: white;
240.     padding: 10px 20px;
241.     border: none;
242.     border-radius: 5px;
243.     cursor: pointer;
244. }
245.
246. input[type="submit"]:hover {
247.     background-color: #333;
248. }
249.
250. .error-message {
251.     color: red;
252.     font-size: 0.8em;
253.     margin-top: 5px;
254. }
255.
256. .error-message2 {
257.     color: red;
258.     font-size: 18px;
259.     margin-top: 5px;
260. }
261. .paradas-container {
262.     display: flex;
263.     justify-content: space-between;
264.     align-items: stretch;
265.     padding: 20px;
266.     gap: 20px;
267. }
268.
269. .sidebar_paradas {
270.     width: 40%;
271.     background-color: rgba(255, 255, 255, 0.9);
272.     padding: 20px;
273.     box-shadow: 0 2px 5px rgba(0,0,0,0.1);
274.     display: flex;
```

```
    flex-direction: column;
    border-radius: 10px;
}

.container_paradas {
    background: white;
    padding: 20px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    border-radius: 10px;
}

.Roja { background-color: #fc1c1c7e; }
.Verde { background-color: #25db255b; }
.Azul { background-color: #52b1f98d; }
.Rosa { background-color: #ff6e907c; }
.Naranja { background-color: #ffa60098; }
.Gris { background-color : #5d848396;}


.info-bloque {
    text-align: center;
    margin-top: 30px;
    padding: 10px;
    background-color: #e0e0e0;
    border-radius: 10px;
    box-shadow: 0 2px 5px rgba(0,0,0,0.1);
}

.info-detalle {
    font-weight: bold;
    color: #333;
}

.boton-container {
    display: flex;
    justify-content: center;
    gap: 20px;
    margin-top: 20px;
}

.card-wrapper {
    display: flex;
    transition: transform 0.5s ease;
}
.blog-container {
    width: 80%;
    margin: 20px auto;
    background-color: #f0f0f0;
    padding: 20px;
    box-shadow: 0 2px 5px rgba(0,0,0,0.1);
    border-radius: 10px;
    overflow: hidden;
}

.nav-button {
    position: absolute;
    top: 100%;
    transform: translateY(-50%);
    background-color: rgba(0, 0, 0, 0.5);
    color: white;
    border: none;
    padding: 25px;
    cursor: pointer;
    z-index: 10;
}

.nav-button.left {
    left: 90px;
}

```

```
345. .nav-button.right {
346.     right: 90px;
347. }
348.
```

6.7 Anexo G. Archivo horarios.json

Código fuente correspondiente al JSON de los horarios de los metros y trenes

```
{
    "lineas": {
      "Azul": {
        "dias": {
          "Lunes": [
            "00:21", "00:51", "05:06", "05:21", "05:36", "05:52", "06:04", "06:14",
            "06:24", "06:34", "06:43", "06:51", "06:59", "07:07", "07:15", "07:23",
            "07:31", "07:38", "07:46", "07:53", "08:00", "08:07", "08:14", "08:21",
            "08:28", "08:35", "08:42", "08:49", "08:56", "09:04", "09:14", "09:24",
            "09:34", "09:44", "09:54", "10:04", "10:14", "10:24", "10:34", "10:44",
            "10:54", "11:04", "11:14", "11:24", "11:34", "11:44", "11:54", "12:04",
            "12:14", "12:24", "12:34", "12:44", "12:54", "13:04", "13:14", "13:24",
            "13:34", "13:44", "13:54", "14:04", "14:14", "14:22", "14:30", "14:38",
            "14:46", "14:54", "15:02", "15:10", "15:18", "15:26", "15:34", "15:41",
            "15:48", "15:55", "16:02", "16:09", "16:16", "16:23", "16:30", "16:37",
            "16:44", "16:51", "16:58", "17:05", "17:12", "17:19", "17:26", "17:33",
            "17:40", "17:47", "17:54", "18:01", "18:08", "18:15", "18:22", "18:29",
            "18:36", "18:44", "18:54", "19:04", "19:14", "19:24", "19:34", "19:44",
            "19:54", "20:04", "20:14", "20:24", "20:34", "20:44", "20:54", "21:06",
            "21:21", "21:36", "21:51", "22:06", "22:21", "22:36", "22:51", "23:06",
            "23:21", "23:36", "23:51"
          ],
          "Martes": [
            "00:21", "00:51", "05:06", "05:21", "05:36", "05:52", "06:04", "06:14",
            "06:24", "06:34", "06:43", "06:51", "06:59", "07:07", "07:15", "07:23",
            "07:31", "07:38", "07:46", "07:53", "08:00", "08:07", "08:14", "08:21",
            "08:28", "08:35", "08:42", "08:49", "08:56", "09:04", "09:14", "09:24",
            "09:34", "09:44", "09:54", "10:04", "10:14", "10:24", "10:34", "10:44",
            "10:54", "11:04", "11:14", "11:24", "11:34", "11:44", "11:54", "12:04",
            "12:14", "12:24", "12:34", "12:44", "12:54", "13:04", "13:14", "13:24",
            "13:34", "13:44", "13:54", "14:04", "14:14", "14:22", "14:30", "14:38",
            "14:46", "14:54", "15:02", "15:10", "15:18", "15:26", "15:34", "15:41",
            "15:48", "15:55", "16:02", "16:09", "16:16", "16:23", "16:30", "16:37",
            "16:44", "16:51", "16:58", "17:05", "17:12", "17:19", "17:26", "17:33",
            "17:40", "17:47", "17:54", "18:01", "18:08", "18:15", "18:22", "18:29",
            "18:36", "18:44", "18:54", "19:04", "19:14", "19:24", "19:34", "19:44",
            "19:54", "20:04", "20:14", "20:24", "20:34", "20:44", "20:54", "21:06",
            "21:21", "21:36", "21:51", "22:06", "22:21", "22:36", "22:51", "23:06",
            "23:21", "23:36", "23:51"
          ],
          "Miercoles": [
            "00:21", "00:51", "05:06", "05:21", "05:36", "05:52", "06:04", "06:14",
            "06:24", "06:34", "06:43", "06:51", "06:59", "07:07", "07:15", "07:23",
            "07:31", "07:38", "07:46", "07:53", "08:00", "08:07", "08:14", "08:21",
            "08:28", "08:35", "08:42", "08:49", "08:56", "09:04", "09:14", "09:24",
            "09:34", "09:44", "09:54", "10:04", "10:14", "10:24", "10:34", "10:44",
            "10:54", "11:04", "11:14", "11:24", "11:34", "11:44", "11:54", "12:04",
            "12:14", "12:24", "12:34", "12:44", "12:54", "13:04", "13:14", "13:24",
            "13:34", "13:44", "13:54", "14:04", "14:14", "14:22", "14:30", "14:38",
            "14:46", "14:54", "15:02", "15:10", "15:18", "15:26", "15:34", "15:41",
            "15:48", "15:55", "16:02", "16:09", "16:16", "16:23", "16:30", "16:37",
            "16:44", "16:51", "16:58", "17:05", "17:12", "17:19", "17:26", "17:33",
            "17:40", "17:47", "17:54", "18:01", "18:08", "18:15", "18:22", "18:29",
            "18:36", "18:44", "18:54", "19:04", "19:14", "19:24", "19:34", "19:44",
            "19:54", "20:04", "20:14", "20:24", "20:34", "20:44", "20:54", "21:06",
            "21:21", "21:36", "21:51", "22:06", "22:21", "22:36", "22:51", "23:06",
            "23:21", "23:36", "23:51"
          ],
          "Jueves": [
            "00:21", "00:51", "05:06", "05:21", "05:36", "05:52", "06:04", "06:14",
            "06:24", "06:34", "06:43", "06:51", "06:59", "07:07", "07:15", "07:23",
            "07:31", "07:38", "07:46", "07:53", "08:00", "08:07", "08:14", "08:21",
            "08:28", "08:35", "08:42", "08:49", "08:56", "09:04", "09:14", "09:24",
            "09:34", "09:44", "09:54", "10:04", "10:14", "10:24", "10:34", "10:44",
```

```
            "10:54", "11:04", "11:14", "11:24", "11:34", "11:44", "11:54", "12:04",
            "12:14", "12:24", "12:34", "12:44", "12:54", "13:04", "13:14", "13:24",
            "13:34", "13:44", "13:54", "14:04", "14:14", "14:22", "14:30", "14:38",
            "14:46", "14:54", "15:02", "15:10", "15:18", "15:26", "15:34", "15:41",
            "15:48", "15:55", "16:02", "16:09", "16:16", "16:23", "16:30", "16:37",
            "16:44", "16:51", "16:58", "17:05", "17:12", "17:19", "17:26", "17:33",
            "17:40", "17:47", "17:54", "18:01", "18:08", "18:15", "18:22", "18:29",
            "18:36", "18:44", "18:54", "19:04", "19:14", "19:24", "19:34", "19:44",
            "19:54", "20:04", "20:14", "20:24", "20:34", "20:44", "20:54", "21:06",
            "21:21", "21:36", "21:51", "22:06", "22:21", "22:36", "22:51", "23:06",
            "23:21", "23:36", "23:51"
          ],
          "Viernes": [
            "00:21", "00:51", "05:06", "05:21", "05:36", "05:52", "06:04", "06:14",
            "06:24", "06:34", "06:43", "06:51", "06:59", "07:07", "07:15", "07:23",
            "07:31", "07:38", "07:46", "07:53", "08:00", "08:07", "08:14", "08:21",
            "08:28", "08:35", "08:42", "08:49", "08:56", "09:04", "09:14", "09:24",
            "09:34", "09:44", "09:54", "10:04", "10:14", "10:24", "10:34", "10:44",
            "10:54", "11:04", "11:14", "11:24", "11:34", "11:44", "11:54", "12:04",
            "12:14", "12:24", "12:34", "12:44", "12:54", "13:04", "13:14", "13:24",
            "13:34", "13:44", "13:54", "14:04", "14:14", "14:22", "14:30", "14:38",
            "14:46", "14:54", "15:02", "15:10", "15:18", "15:26", "15:34", "15:41",
            "15:48", "15:55", "16:02", "16:09", "16:16", "16:23", "16:30", "16:37",
            "16:44", "16:51", "16:58", "17:05", "17:12", "17:19", "17:26", "17:33",
            "17:40", "17:47", "17:54", "18:01", "18:08", "18:15", "18:22", "18:29",
            "18:36", "18:44", "18:54", "19:04", "19:14", "19:24", "19:34", "19:44",
            "19:54", "20:04", "20:14", "20:24", "20:34", "20:44", "20:54", "21:06",
            "21:21", "21:36", "21:51", "22:06", "22:21", "22:36", "22:51", "23:06",
            "23:21", "23:36", "23:51"
          ],
          "Sabado": [
            "00:06", "00:21", "00:36", "00:51", "01:06", "01:21", "01:36", "01:51",
            "02:13", "02:43", "03:13", "03:43", "04:13", "04:43", "05:13", "05:44",
            "06:06", "06:21", "06:36", "06:51", "07:06", "07:21", "07:36", "07:51",
            "08:06", "08:21", "08:34", "08:44", "08:54", "09:04", "09:14", "09:24",
            "09:34", "09:44", "09:54", "10:04", "10:14", "10:24", "10:34", "10:44",
            "10:54", "11:04", "11:14", "11:24", "11:34", "11:44", "11:54", "12:04",
            "12:14", "12:24", "12:34", "12:44", "12:54", "13:04", "13:14", "13:24",
            "13:34", "13:44", "13:54", "14:04", "14:14", "14:24", "14:34", "14:44",
            "14:54", "15:04", "15:14", "15:24", "15:34", "15:44", "15:54", "16:04",
            "16:14", "16:24", "16:34", "16:44", "16:54", "17:04", "17:14", "17:24",
            "17:34", "17:44", "17:54", "18:04", "18:14", "18:24", "18:34", "18:44",
            "18:54", "19:06", "19:21", "19:36", "19:51", "20:06", "20:21", "20:36",
            "20:51", "21:06", "21:21", "21:36", "21:51", "22:06", "22:21", "22:36",
            "22:51", "23:06", "23:21", "23:36", "23:51"
          ],
          "Domingo": [
            "00:06", "00:21", "00:36", "00:51", "01:06", "01:21", "01:36", "01:51",
            "02:13", "02:43", "03:13", "03:43", "04:13", "04:43", "05:13", "05:44",
            "06:06", "06:21", "06:36", "06:51", "07:06", "07:21", "07:36", "07:51",
            "08:06", "08:21", "08:34", "08:44", "08:54", "09:04", "09:14", "09:24",
            "09:34", "09:44", "09:54", "10:04", "10:14", "10:24", "10:34", "10:44",
            "10:54", "11:04", "11:14", "11:24", "11:34", "11:44", "11:54", "12:04",
            "12:14", "12:24", "12:34", "12:44", "12:54", "13:04", "13:14", "13:24",
            "13:34", "13:44", "13:54", "14:04", "14:14", "14:24", "14:34", "14:44",
            "14:54", "15:04", "15:14", "15:24", "15:34", "15:44", "15:54", "16:04",
            "16:14", "16:24", "16:34", "16:44", "16:54", "17:04", "17:14", "17:24",
            "17:34", "17:44", "17:54", "18:04", "18:14", "18:24", "18:34", "18:44",
            "18:54", "19:06", "19:21", "19:36", "19:51", "20:06", "20:21", "20:36",
            "20:51", "21:06", "21:21", "21:36", "21:51", "22:06", "22:21", "22:36",
            "22:51", "23:06", "23:21", "23:36", "23:51"
          ]
        }
      },
      "Roja": {
        "dias": {
          "Lunes": [
            "00:15", "00:45", "05:00", "05:15", "05:30", "05:45", "05:58", "06:08",
            "06:17", "06:25", "06:30", "06:35", "06:40", "06:45", "06:50", "06:55",
            "07:00", "07:05", "07:10", "07:15", "07:20", "07:25", "07:30", "07:35",
```

```
            "07:40", "07:45", "07:50", "07:55", "08:00", "08:05", "08:10", "08:15",
            "08:20", "08:25", "08:30", "08:35", "08:40", "08:45", "08:50", "08:55",
            "09:00", "09:05", "09:10", "09:15", "09:20", "09:25", "09:30", "09:35",
            "09:45", "09:55", "10:05", "10:15", "10:24", "10:34", "10:44", "10:54",
            "11:04", "11:14", "11:24", "11:34", "11:44", "11:54", "12:04", "12:14",
            "12:24", "12:34", "12:44", "12:54", "13:04", "13:14", "13:24", "13:34",
            "13:44", "13:54", "14:04", "14:14", "14:24", "14:34", "14:44", "14:50",
            "14:55", "15:00", "15:05", "15:10", "15:15", "15:20", "15:25", "15:30",
            "15:35", "15:40", "15:45", "15:50", "15:55", "16:00", "16:05", "16:10",
            "16:15", "16:20", "16:25", "16:30", "16:35", "16:40", "16:45", "16:50",
            "16:55", "17:00", "17:05", "17:10", "17:15", "17:20", "17:25", "17:30",
            "17:35", "17:40", "17:45", "17:50", "17:55", "18:00", "18:05", "18:10",
            "18:15", "18:20", "18:25", "18:30", "18:35", "18:40", "18:45", "18:50",
            "18:55", "19:00", "19:05", "19:14", "19:24", "19:34", "19:44", "19:54",
            "20:05", "20:15", "20:25", "20:35", "20:44", "20:54", "21:04", "21:14",
            "21:24", "21:34", "21:44", "21:59", "22:14", "22:29", "22:44", "22:59",
            "23:14", "23:29", "23:44", "23:45", "23:59"
          ],
          "Martes": [
            "00:15", "00:45", "05:00", "05:15", "05:30", "05:45", "05:58", "06:08",
            "06:17", "06:25", "06:30", "06:35", "06:40", "06:45", "06:50", "06:55",
            "07:00", "07:05", "07:10", "07:15", "07:20", "07:25", "07:30", "07:35",
            "07:40", "07:45", "07:50", "07:55", "08:00", "08:05", "08:10", "08:15",
            "08:20", "08:25", "08:30", "08:35", "08:40", "08:45", "08:50", "08:55",
            "09:00", "09:05", "09:10", "09:15", "09:20", "09:25", "09:30", "09:35",
            "09:45", "09:55", "10:05", "10:15", "10:24", "10:34", "10:44", "10:54",
            "11:04", "11:14", "11:24", "11:34", "11:44", "11:54", "12:04", "12:14",
            "12:24", "12:34", "12:44", "12:54", "13:04", "13:14", "13:24", "13:34",
            "13:44", "13:54", "14:04", "14:14", "14:24", "14:34", "14:44", "14:50",
            "14:55", "15:00", "15:05", "15:10", "15:15", "15:20", "15:25", "15:30",
            "15:35", "15:40", "15:45", "15:50", "15:55", "16:00", "16:05", "16:10",
            "16:15", "16:20", "16:25", "16:30", "16:35", "16:40", "16:45", "16:50",
            "16:55", "17:00", "17:05", "17:10", "17:15", "17:20", "17:25", "17:30",
            "17:35", "17:40", "17:45", "17:50", "17:55", "18:00", "18:05", "18:10",
            "18:15", "18:20", "18:25", "18:30", "18:35", "18:40", "18:45", "18:50",
            "18:55", "19:00", "19:05", "19:14", "19:24", "19:34", "19:44", "19:54",
            "20:05", "20:15", "20:25", "20:35", "20:44", "20:54", "21:04", "21:14",
            "21:24", "21:34", "21:44", "21:59", "22:14", "22:29", "22:44", "22:59",
            "23:14", "23:29", "23:44", "23:45", "23:59"
          ],
          "Miercoles": [
            "00:15", "00:45", "05:00", "05:15", "05:30", "05:45", "05:58", "06:08",
            "06:17", "06:25", "06:30", "06:35", "06:40", "06:45", "06:50", "06:55",
            "07:00", "07:05", "07:10", "07:15", "07:20", "07:25", "07:30", "07:35",
            "07:40", "07:45", "07:50", "07:55", "08:00", "08:05", "08:10", "08:15",
            "08:20", "08:25", "08:30", "08:35", "08:40", "08:45", "08:50", "08:55",
            "09:00", "09:05", "09:10", "09:15", "09:20", "09:25", "09:30", "09:35",
            "09:45", "09:55", "10:05", "10:15", "10:24", "10:34", "10:44", "10:54",
            "11:04", "11:14", "11:24", "11:34", "11:44", "11:54", "12:04", "12:14",
            "12:24", "12:34", "12:44", "12:54", "13:04", "13:14", "13:24", "13:34",
            "13:44", "13:54", "14:04", "14:14", "14:24", "14:34", "14:44", "14:50",
            "14:55", "15:00", "15:05", "15:10", "15:15", "15:20", "15:25", "15:30",
            "15:35", "15:40", "15:45", "15:50", "15:55", "16:00", "16:05", "16:10",
            "16:15", "16:20", "16:25", "16:30", "16:35", "16:40", "16:45", "16:50",
            "16:55", "17:00", "17:05", "17:10", "17:15", "17:20", "17:25", "17:30",
            "17:35", "17:40", "17:45", "17:50", "17:55", "18:00", "18:05", "18:10",
            "18:15", "18:20", "18:25", "18:30", "18:35", "18:40", "18:45", "18:50",
            "18:55", "19:00", "19:05", "19:14", "19:24", "19:34", "19:44", "19:54",
            "20:05", "20:15", "20:25", "20:35", "20:44", "20:54", "21:04", "21:14",
            "21:24", "21:34", "21:44", "21:59", "22:14", "22:29", "22:44", "22:59",
            "23:14", "23:29", "23:44", "23:45", "23:59"
          ],
          "Jueves": [
            "00:15", "00:45", "05:00", "05:15", "05:30", "05:45", "05:58", "06:08",
            "06:17", "06:25", "06:30", "06:35", "06:40", "06:45", "06:50", "06:55",
            "07:00", "07:05", "07:10", "07:15", "07:20", "07:25", "07:30", "07:35",
            "07:40", "07:45", "07:50", "07:55", "08:00", "08:05", "08:10", "08:15",
            "08:20", "08:25", "08:30", "08:35", "08:40", "08:45", "08:50", "08:55",
            "09:00", "09:05", "09:10", "09:15", "09:20", "09:25", "09:30", "09:35",
            "09:45", "09:55", "10:05", "10:15", "10:24", "10:34", "10:44", "10:54",
```

```
205.             "11:04", "11:14", "11:24", "11:34", "11:44", "11:54", "12:04", "12:14",
206.             "12:24", "12:34", "12:44", "12:54", "13:04", "13:14", "13:24", "13:34",
207.             "13:44", "13:54", "14:04", "14:14", "14:24", "14:34", "14:44", "14:50",
208.             "14:55", "15:00", "15:05", "15:10", "15:15", "15:20", "15:25", "15:30",
209.             "15:35", "15:40", "15:45", "15:50", "15:55", "16:00", "16:05", "16:10",
210.             "16:15", "16:20", "16:25", "16:30", "16:35", "16:40", "16:45", "16:50",
211.             "16:55", "17:00", "17:05", "17:10", "17:15", "17:20", "17:25", "17:30",
212.             "17:35", "17:40", "17:45", "17:50", "17:55", "18:00", "18:05", "18:10",
213.             "18:15", "18:20", "18:25", "18:30", "18:35", "18:40", "18:45", "18:50",
214.             "18:55", "19:00", "19:05", "19:14", "19:24", "19:34", "19:44", "19:54",
215.             "20:05", "20:15", "20:25", "20:35", "20:44", "20:54", "21:04", "21:14",
216.             "21:24", "21:34", "21:44", "21:59", "22:14", "22:29", "22:44", "22:59",
217.             "23:14", "23:29", "23:44", "23:45", "23:59"
218.           ],
219.           "Viernes": [
220.             "00:15", "00:45", "05:00", "05:15", "05:30", "05:45", "05:58", "06:08",
221.             "06:17", "06:25", "06:30", "06:35", "06:40", "06:45", "06:50", "06:55",
222.             "07:00", "07:05", "07:10", "07:15", "07:20", "07:25", "07:30", "07:35",
223.             "07:40", "07:45", "07:50", "07:55", "08:00", "08:05", "08:10", "08:15",
224.             "08:20", "08:25", "08:30", "08:35", "08:40", "08:45", "08:50", "08:55",
225.             "09:00", "09:05", "09:10", "09:15", "09:20", "09:25", "09:30", "09:35",
226.             "09:45", "09:55", "10:05", "10:15", "10:24", "10:34", "10:44", "10:54",
227.             "11:04", "11:14", "11:24", "11:34", "11:44", "11:54", "12:04", "12:14",
228.             "12:24", "12:34", "12:44", "12:54", "13:04", "13:14", "13:24", "13:34",
229.             "13:44", "13:54", "14:04", "14:14", "14:24", "14:34", "14:44", "14:50",
230.             "14:55", "15:00", "15:05", "15:10", "15:15", "15:20", "15:25", "15:30",
231.             "15:35", "15:40", "15:45", "15:50", "15:55", "16:00", "16:05", "16:10",
232.             "16:15", "16:20", "16:25", "16:30", "16:35", "16:40", "16:45", "16:50",
233.             "16:55", "17:00", "17:05", "17:10", "17:15", "17:20", "17:25", "17:30",
234.             "17:35", "17:40", "17:45", "17:50", "17:55", "18:00", "18:05", "18:10",
235.             "18:15", "18:20", "18:25", "18:30", "18:35", "18:40", "18:45", "18:50",
236.             "18:55", "19:00", "19:05", "19:14", "19:24", "19:34", "19:44", "19:54",
237.             "20:05", "20:15", "20:25", "20:35", "20:44", "20:54", "21:04", "21:14",
238.             "21:24", "21:34", "21:44", "21:59", "22:14", "22:29", "22:44", "22:59",
239.             "23:14", "23:29", "23:44", "23:45", "23:59"
240.           ],
241.           "Sabado": [
242.             "00:16", "00:31", "00:46", "01:01", "01:31", "02:01", "02:31", "03:01",
243.             "03:31", "04:02", "04:32", "05:02", "05:31", "06:01", "06:16", "06:31",
244.             "06:46", "07:01", "07:16", "07:31", "07:46", "08:01", "08:16", "08:31",
245.             "08:46", "08:58", "09:08", "09:18", "09:28", "09:38", "09:48", "09:58",
246.             "10:08", "10:18", "10:28", "10:38", "10:48", "10:58", "11:08", "11:18",
247.             "11:28", "11:38", "11:48", "11:58", "12:08", "12:18", "12:28", "12:38",
248.             "12:48", "12:58", "13:08", "13:18", "13:28", "13:38", "13:48", "13:58",
249.             "14:08", "14:18", "14:28", "14:38", "14:48", "14:58", "15:08", "15:18",
250.             "15:28", "15:38", "15:48", "15:58", "16:08", "16:18", "16:28", "16:38",
251.             "16:48", "16:58", "17:08", "17:18", "17:28", "17:38", "17:48", "17:58",
252.             "18:09", "18:19", "18:29", "18:39", "18:49", "18:58", "19:08", "19:18",
253.             "19:28", "19:38", "19:48", "19:58", "20:08", "20:18", "20:28", "20:38",
254.             "20:48", "20:58", "21:08", "21:18", "21:31", "21:46", "22:01", "22:16",
255.             "22:31", "22:46", "23:01", "23:16", "23:31", "23:46"
256.           ],
257.           "Domingo": [
258.             "00:16", "00:31", "00:46", "01:01", "01:31", "02:01", "02:31", "03:01",
259.             "03:31", "04:02", "04:32", "05:02", "05:31", "06:01", "06:16", "06:31",
260.             "06:46", "07:01", "07:16", "07:31", "07:46", "08:01", "08:16", "08:31",
261.             "08:46", "08:58", "09:08", "09:18", "09:28", "09:38", "09:48", "09:58",
262.             "10:08", "10:18", "10:28", "10:38", "10:48", "10:58", "11:08", "11:18",
263.             "11:28", "11:38", "11:48", "11:58", "12:08", "12:18", "12:28", "12:38",
264.             "12:48", "12:58", "13:08", "13:18", "13:28", "13:38", "13:48", "13:58",
265.             "14:08", "14:18", "14:28", "14:38", "14:48", "14:58", "15:08", "15:18",
266.             "15:28", "15:38", "15:48", "15:58", "16:08", "16:18", "16:28", "16:38",
267.             "16:48", "16:58", "17:08", "17:18", "17:28", "17:38", "17:48", "17:58",
268.             "18:09", "18:19", "18:29", "18:39", "18:49", "18:58", "19:08", "19:18",
269.             "19:28", "19:38", "19:48", "19:58", "20:08", "20:18", "20:28", "20:38",
270.             "20:48", "20:58", "21:08", "21:18", "21:31", "21:46", "22:01", "22:16",
271.             "22:31", "22:46", "23:01", "23:16", "23:31", "23:46"
272.           ]
273.         }
274.       },
```

```
275.       "Verde": {
276.         "dias": {
277.           "Lunes": [
278.             "00:31", "05:16", "05:31", "05:46", "06:01", "06:16", "06:26", "06:40",
279.             "06:51", "07:01", "07:09", "07:15", "07:23", "07:33", "07:43", "07:50",
280.             "07:58", "08:08", "08:18", "08:22", "08:28", "08:38", "08:49", "08:58",
281.             "09:07", "09:17", "09:27", "09:37", "09:47", "09:57", "10:07", "10:17",
282.             "10:27", "10:37", "10:47", "10:57", "11:07", "11:17", "11:27", "11:37",
283.             "11:47", "11:57", "12:07", "12:17", "12:27", "12:37", "12:47", "12:57",
284.             "13:07", "13:17", "13:27", "13:37", "13:47", "13:57", "14:07", "14:17",
285.             "14:27", "14:37", "14:47", "14:57", "15:07", "15:17", "15:27", "15:37",
286.             "15:47", "15:55", "16:03", "16:10", "16:15", "16:22", "16:30", "16:37",
287.             "16:45", "16:52", "17:00", "17:07", "17:15", "17:22", "17:30", "17:37",
288.             "17:45", "17:51", "17:57", "18:04", "18:12", "18:20", "18:27", "18:35",
289.             "18:42", "18:50", "18:57", "19:07", "19:17", "19:27", "19:37", "19:47",
290.             "19:57", "20:07", "20:17", "20:27", "20:38", "20:48", "20:58", "21:08",
291.             "21:17", "21:30", "21:39", "21:45", "22:00", "22:15", "22:30", "22:45",
292.             "23:00", "23:15", "23:30", "23:45"
293.           ],
294.           "Martes": [
295.             "00:31", "05:16", "05:31", "05:46", "06:01", "06:16", "06:26", "06:40",
296.             "06:51", "07:01", "07:09", "07:15", "07:23", "07:33", "07:43", "07:50",
297.             "07:58", "08:08", "08:18", "08:22", "08:28", "08:38", "08:49", "08:58",
298.             "09:07", "09:17", "09:27", "09:37", "09:47", "09:57", "10:07", "10:17",
299.             "10:27", "10:37", "10:47", "10:57", "11:07", "11:17", "11:27", "11:37",
300.             "11:47", "11:57", "12:07", "12:17", "12:27", "12:37", "12:47", "12:57",
301.             "13:07", "13:17", "13:27", "13:37", "13:47", "13:57", "14:07", "14:17",
302.             "14:27", "14:37", "14:47", "14:57", "15:07", "15:17", "15:27", "15:37",
303.             "15:47", "15:55", "16:03", "16:10", "16:15", "16:22", "16:30", "16:37",
304.             "16:45", "16:52", "17:00", "17:07", "17:15", "17:22", "17:30", "17:37",
305.             "17:45", "17:51", "17:57", "18:04", "18:12", "18:20", "18:27", "18:35",
306.             "18:42", "18:50", "18:57", "19:07", "19:17", "19:27", "19:37", "19:47",
307.             "19:57", "20:07", "20:17", "20:27", "20:38", "20:48", "20:58", "21:08",
308.             "21:17", "21:30", "21:39", "21:45", "22:00", "22:15", "22:30", "22:45",
309.             "23:00", "23:15", "23:30", "23:45"
310.           ],
311.           "Miercoles": [
312.             "00:31", "05:16", "05:31", "05:46", "06:01", "06:16", "06:26", "06:40",
313.             "06:51", "07:01", "07:09", "07:15", "07:23", "07:33", "07:43", "07:50",
314.             "07:58", "08:08", "08:18", "08:22", "08:28", "08:38", "08:49", "08:58",
315.             "09:07", "09:17", "09:27", "09:37", "09:47", "09:57", "10:07", "10:17",
316.             "10:27", "10:37", "10:47", "10:57", "11:07", "11:17", "11:27", "11:37",
317.             "11:47", "11:57", "12:07", "12:17", "12:27", "12:37", "12:47", "12:57",
318.             "13:07", "13:17", "13:27", "13:37", "13:47", "13:57", "14:07", "14:17",
319.             "14:27", "14:37", "14:47", "14:57", "15:07", "15:17", "15:27", "15:37",
320.             "15:47", "15:55", "16:03", "16:10", "16:15", "16:22", "16:30", "16:37",
321.             "16:45", "16:52", "17:00", "17:07", "17:15", "17:22", "17:30", "17:37",
322.             "17:45", "17:51", "17:57", "18:04", "18:12", "18:20", "18:27", "18:35",
323.             "18:42", "18:50", "18:57", "19:07", "19:17", "19:27", "19:37", "19:47",
324.             "19:57", "20:07", "20:17", "20:27", "20:38", "20:48", "20:58", "21:08",
325.             "21:17", "21:30", "21:39", "21:45", "22:00", "22:15", "22:30", "22:45",
326.             "23:00", "23:15", "23:30", "23:45"
327.           ],
328.           "Jueves": [
329.             "00:31", "05:16", "05:31", "05:46", "06:01", "06:16", "06:26", "06:40",
330.             "06:51", "07:01", "07:09", "07:15", "07:23", "07:33", "07:43", "07:50",
331.             "07:58", "08:08", "08:18", "08:22", "08:28", "08:38", "08:49", "08:58",
332.             "09:07", "09:17", "09:27", "09:37", "09:47", "09:57", "10:07", "10:17",
333.             "10:27", "10:37", "10:47", "10:57", "11:07", "11:17", "11:27", "11:37",
334.             "11:47", "11:57", "12:07", "12:17", "12:27", "12:37", "12:47", "12:57",
335.             "13:07", "13:17", "13:27", "13:37", "13:47", "13:57", "14:07", "14:17",
336.             "14:27", "14:37", "14:47", "14:57", "15:07", "15:17", "15:27", "15:37",
337.             "15:47", "15:55", "16:03", "16:10", "16:15", "16:22", "16:30", "16:37",
338.             "16:45", "16:52", "17:00", "17:07", "17:15", "17:22", "17:30", "17:37",
339.             "17:45", "17:51", "17:57", "18:04", "18:12", "18:20", "18:27", "18:35",
340.             "18:42", "18:50", "18:57", "19:07", "19:17", "19:27", "19:37", "19:47",
341.             "19:57", "20:07", "20:17", "20:27", "20:38", "20:48", "20:58", "21:08",
342.             "21:17", "21:30", "21:39", "21:45", "22:00", "22:15", "22:30", "22:45",
343.             "23:00", "23:15", "23:30", "23:45"
344.           ],
```

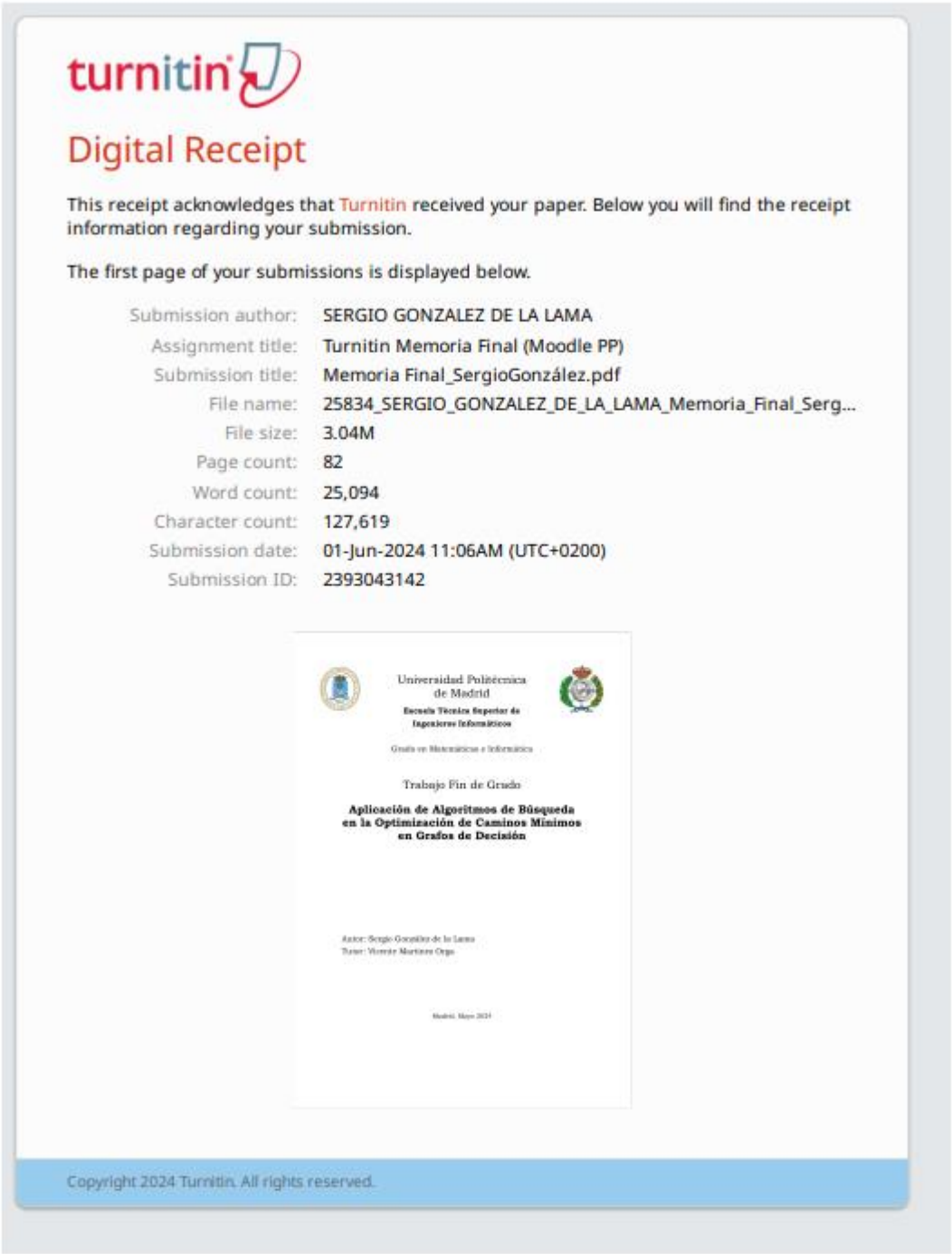
```
345.           "Viernes": [
346.             "00:31", "05:16", "05:31", "05:46", "06:01", "06:16", "06:26", "06:40",
347.             "06:51", "07:01", "07:09", "07:15", "07:23", "07:33", "07:43", "07:50",
348.             "07:58", "08:08", "08:18", "08:22", "08:28", "08:38", "08:49", "08:58",
349.             "09:07", "09:17", "09:27", "09:37", "09:47", "09:57", "10:07", "10:17",
350.             "10:27", "10:37", "10:47", "10:57", "11:07", "11:17", "11:27", "11:37",
351.             "11:47", "11:57", "12:07", "12:17", "12:27", "12:37", "12:47", "12:57",
352.             "13:07", "13:17", "13:27", "13:37", "13:47", "13:57", "14:07", "14:17",
353.             "14:27", "14:37", "14:47", "14:57", "15:07", "15:17", "15:27", "15:37",
354.             "15:47", "15:55", "16:03", "16:10", "16:15", "16:22", "16:30", "16:37",
355.             "16:45", "16:52", "17:00", "17:07", "17:15", "17:22", "17:30", "17:37",
356.             "17:45", "17:51", "17:57", "18:04", "18:12", "18:20", "18:27", "18:35",
357.             "18:42", "18:50", "18:57", "19:07", "19:17", "19:27", "19:37", "19:47",
358.             "19:57", "20:07", "20:17", "20:27", "20:38", "20:48", "20:58", "21:08",
359.             "21:17", "21:30", "21:39", "21:45", "22:00", "22:15", "22:30", "22:45",
360.             "23:00", "23:15", "23:30", "23:45"
361.           ],
362.           "Sabado": [
363.             "00:31", "00:46", "01:01", "01:31", "02:01", "02:31", "03:01", "03:36",
364.             "04:06", "04:36", "05:06", "05:36", "06:06", "06:29", "06:44", "06:59",
365.             "07:15", "07:30", "07:45", "08:00", "08:15", "08:29", "08:43", "08:56",
366.             "09:10", "09:16", "09:26", "09:36", "09:46", "09:56", "10:06", "10:16",
367.             "10:26", "10:36", "10:46", "10:56", "11:06", "11:16", "11:26", "11:36",
368.             "11:46", "11:56", "12:06", "12:16", "12:26", "12:36", "12:46", "12:56",
369.             "13:06", "13:16", "13:26", "13:36", "13:46", "13:56", "14:06", "14:16",
370.             "14:26", "14:36", "14:46", "14:56", "15:06", "15:16", "15:26", "15:36",
371.             "15:46", "15:56", "16:06", "16:16", "16:26", "16:36", "16:46", "16:56",
372.             "17:06", "17:16", "17:26", "17:36", "17:46", "17:56", "18:06", "18:16",
373.             "18:26", "18:36", "18:46", "18:56", "19:06", "19:16", "19:26", "19:36",
374.             "19:46", "19:56", "20:06", "20:16", "20:26", "20:36", "20:46", "20:56",
375.             "21:06", "21:16", "21:26", "21:36", "21:46", "21:56", "22:06", "22:16",
376.             "22:26", "22:36", "22:46", "22:56", "23:06", "23:16", "23:30", "23:45",
377.             "00:00", "00:15", "00:30", "00:45", "01:00", "01:15", "01:30", "01:45"
378.           ],
379.           "Domingo": [
380.             "00:31", "00:46", "01:01", "01:31", "02:01", "02:31", "03:01", "03:36",
381.             "04:06", "04:36", "05:06", "05:36", "06:06", "06:29", "06:44", "06:59",
382.             "07:15", "07:30", "07:45", "08:00", "08:15", "08:29", "08:43", "08:56",
383.             "09:10", "09:16", "09:26", "09:36", "09:46", "09:56", "10:06", "10:16",
384.             "10:26", "10:36", "10:46", "10:56", "11:06", "11:16", "11:26", "11:36",
385.             "11:46", "11:56", "12:06", "12:16", "12:26", "12:36", "12:46", "12:56",
386.             "13:06", "13:16", "13:26", "13:36", "13:46", "13:56", "14:06", "14:16",
387.             "14:26", "14:36", "14:46", "14:56", "15:06", "15:16", "15:26", "15:36",
388.             "15:46", "15:56", "16:06", "16:16", "16:26", "16:36", "16:46", "16:56",
389.             "17:06", "17:16", "17:26", "17:36", "17:46", "17:56", "18:06", "18:16",
390.             "18:26", "18:36", "18:46", "18:56", "19:06", "19:16", "19:26", "19:36",
391.             "19:46", "19:56", "20:06", "20:16", "20:26", "20:36", "20:46", "20:56",
392.             "21:06", "21:16", "21:26", "21:36", "21:46", "21:56", "22:06", "22:16",
393.             "22:26", "22:36", "22:46", "22:56", "23:06", "23:16", "23:30", "23:45",
394.             "00:00", "00:15", "00:30", "00:45", "01:00", "01:15", "01:30", "01:45"
395.           ]
396.         }
397.       },
398.       "Naranja": {
399.         "dias": {
400.           "Lunes": [
401.             "00:51", "01:25", "01:51", "05:30", "06:10", "06:30", "06:50", "07:10",
402.             "07:30", "07:50", "08:10", "08:31", "08:54", "09:12", "09:36", "10:06",
403.             "10:36", "11:06", "11:36", "12:06", "12:36", "13:06", "13:36", "14:06",
404.             "14:36", "15:06", "15:36", "16:06", "16:40", "17:10", "17:31", "17:50",
405.             "18:11", "18:30", "18:50", "19:10", "19:24", "19:52", "20:21", "20:51",
406.             "21:21", "21:51", "22:21", "22:51", "23:21", "23:51"
407.           ],
408.           "Martes": [
409.             "00:51", "01:25", "01:51", "05:30", "06:10", "06:30", "06:50", "07:10",
410.             "07:30", "07:50", "08:10", "08:31", "08:54", "09:12", "09:36", "10:06",
411.             "10:36", "11:06", "11:36", "12:06", "12:36", "13:06", "13:36", "14:06",
412.             "14:36", "15:06", "15:36", "16:06", "16:40", "17:10", "17:31", "17:50",
413.             "18:11", "18:30", "18:50", "19:10", "19:24", "19:52", "20:21", "20:51",
414.             "21:21", "21:51", "22:21", "22:51", "23:21", "23:51"
```

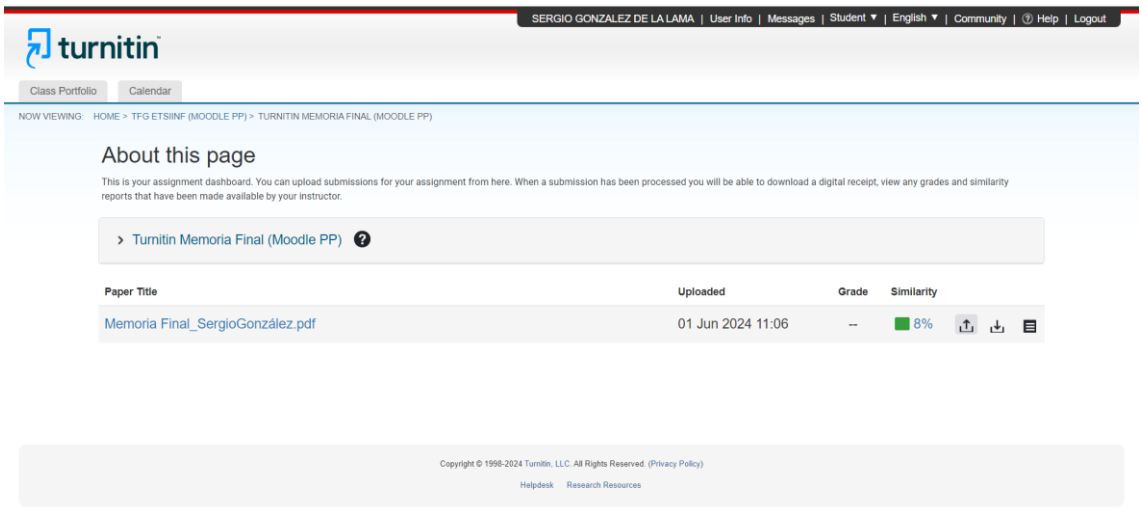
```
          ],
          "Miercoles": [
            "00:51", "01:25", "01:51", "05:30", "06:10", "06:30", "06:50", "07:10",
            "07:30", "07:50", "08:10", "08:31", "08:54", "09:12", "09:36", "10:06",
            "10:36", "11:06", "11:36", "12:06", "12:36", "13:06", "13:36", "14:06",
            "14:36", "15:06", "15:36", "16:06", "16:40", "17:10", "17:31", "17:50",
            "18:11", "18:30", "18:50", "19:10", "19:24", "19:52", "20:21", "20:51",
            "21:21", "21:51", "22:21", "22:51", "23:21", "23:51"
          ],
          "Jueves": [
            "00:51", "01:25", "01:51", "05:30", "06:10", "06:30", "06:50", "07:10",
            "07:30", "07:50", "08:10", "08:31", "08:54", "09:12", "09:36", "10:06",
            "10:36", "11:06", "11:36", "12:06", "12:36", "13:06", "13:36", "14:06",
            "14:36", "15:06", "15:36", "16:06", "16:40", "17:10", "17:31", "17:50",
            "18:11", "18:30", "18:50", "19:10", "19:24", "19:52", "20:21", "20:51",
            "21:21", "21:51", "22:21", "22:51", "23:21", "23:51"
          ],
          "Viernes": [
            "00:51", "01:25", "01:51", "05:30", "06:10", "06:30", "06:50", "07:10",
            "07:30", "07:50", "08:10", "08:31", "08:54", "09:12", "09:36", "10:06",
            "10:36", "11:06", "11:36", "12:06", "12:36", "13:06", "13:36", "14:06",
            "14:36", "15:06", "15:36", "16:06", "16:40", "17:10", "17:31", "17:50",
            "18:11", "18:30", "18:50", "19:10", "19:24", "19:52", "20:21", "20:51",
            "21:21", "21:51", "22:21", "22:51", "23:21", "23:51"
          ],
          "Sabado": [
            "00:51", "01:21", "01:51", "02:21", "06:00", "06:30", "07:00", "07:30",
            "08:00", "08:30", "09:00", "09:30", "10:00", "10:36", "11:06", "11:36",
            "12:06", "12:36", "13:06", "13:36", "14:06", "14:36", "15:06", "15:36",
            "16:06", "16:36", "17:06", "17:36", "18:06", "18:36", "19:06", "19:36",
            "19:52", "20:22", "20:51", "21:21", "21:51", "22:21", "22:51", "23:21",
            "23:51"
          ],
          "Domingo": [
            "00:51", "01:21", "01:51", "02:21", "06:00", "06:30", "07:00", "07:30",
            "08:00", "08:30", "09:00", "09:30", "10:00", "10:36", "11:06", "11:36",
            "12:06", "12:36", "13:06", "13:36", "14:06", "14:36", "15:06", "15:36",
            "16:06", "16:36", "17:06", "17:36", "18:06", "18:36", "19:06", "19:36",
            "19:52", "20:22", "20:51", "21:21", "21:51", "22:21", "22:51", "23:21",
            "23:51"
          ]
        }
      },
      "Rosa": {
        "dias": {
          "Lunes": [
            "00:51", "01:25", "01:51", "05:30", "06:10", "06:30", "06:50", "07:10",
            "07:30", "07:50", "08:10", "08:31", "08:54", "09:12", "09:36", "10:06",
            "10:36", "11:06", "11:36", "12:06", "12:36", "13:06", "13:36", "14:06",
            "14:36", "15:06", "15:36", "16:06", "16:40", "17:10", "17:31", "17:50",
            "18:11", "18:30", "18:50", "19:10", "19:24", "19:52", "20:21", "20:51",
            "21:21", "21:51", "22:21", "22:51", "23:21", "23:51"
          ],
          "Martes": [
            "00:51", "01:25", "01:51", "05:30", "06:10", "06:30", "06:50", "07:10",
            "07:30", "07:50", "08:10", "08:31", "08:54", "09:12", "09:36", "10:06",
            "10:36", "11:06", "11:36", "12:06", "12:36", "13:06", "13:36", "14:06",
            "14:36", "15:06", "15:36", "16:06", "16:40", "17:10", "17:31", "17:50",
            "18:11", "18:30", "18:50", "19:10", "19:24", "19:52", "20:21", "20:51",
            "21:21", "21:51", "22:21", "22:51", "23:21", "23:51"
          ],
          "Miercoles": [
            "00:51", "01:25", "01:51", "05:30", "06:10", "06:30", "06:50", "07:10",
            "07:30", "07:50", "08:10", "08:31", "08:54", "09:12", "09:36", "10:06",
            "10:36", "11:06", "11:36", "12:06", "12:36", "13:06", "13:36", "14:06",
            "14:36", "15:06", "15:36", "16:06", "16:40", "17:10", "17:31", "17:50",
            "18:11", "18:30", "18:50", "19:10", "19:24", "19:52", "20:21", "20:51",
            "21:21", "21:51", "22:21", "22:51", "23:21", "23:51"
          ],
          "Jueves": [
```

```
485.             "00:51", "01:25", "01:51", "05:30", "06:10", "06:30", "06:50", "07:10",
486.             "07:30", "07:50", "08:10", "08:31", "08:54", "09:12", "09:36", "10:06",
487.             "10:36", "11:06", "11:36", "12:06", "12:36", "13:06", "13:36", "14:06",
488.             "14:36", "15:06", "15:36", "16:06", "16:40", "17:10", "17:31", "17:50",
489.             "18:11", "18:30", "18:50", "19:10", "19:24", "19:52", "20:21", "20:51",
490.             "21:21", "21:51", "22:21", "22:51", "23:21", "23:51"
491.           ],
492.           "Viernes": [
493.             "00:51", "01:25", "01:51", "05:30", "06:10", "06:30", "06:50", "07:10",
494.             "07:30", "07:50", "08:10", "08:31", "08:54", "09:12", "09:36", "10:06",
495.             "10:36", "11:06", "11:36", "12:06", "12:36", "13:06", "13:36", "14:06",
496.             "14:36", "15:06", "15:36", "16:06", "16:40", "17:10", "17:31", "17:50",
497.             "18:11", "18:30", "18:50", "19:10", "19:24", "19:52", "20:21", "20:51",
498.             "21:21", "21:51", "22:21", "22:51", "23:21", "23:51"
499.           ],
500.           "Sabado": [
501.             "00:51", "01:21", "01:51", "02:21", "06:00", "06:30", "07:00", "07:30",
502.             "08:00", "08:30", "09:00", "09:30", "10:00", "10:36", "11:06", "11:36",
503.             "12:06", "12:36", "13:06", "13:36", "14:06", "14:36", "15:06", "15:36",
504.             "16:06", "16:36", "17:06", "17:36", "18:06", "18:36", "19:06", "19:36",
505.             "19:52", "20:22", "20:51", "21:21", "21:51", "22:21", "22:51", "23:21",
506.             "23:51"
507.           ],
508.           "Domingo": [
509.             "00:51", "01:21", "01:51", "02:21", "06:00", "06:30", "07:00", "07:30",
510.             "08:00", "08:30", "09:00", "09:30", "10:00", "10:36", "11:06", "11:36",
511.             "12:06", "12:36", "13:06", "13:36", "14:06", "14:36", "15:06", "15:36",
512.             "16:06", "16:36", "17:06", "17:36", "18:06", "18:36", "19:06", "19:36",
513.             "19:52", "20:22", "20:51", "21:21", "21:51", "22:21", "22:51", "23:21",
514.             "23:51"
515.           ]
516.         }
517.       }
518.     }
519.   }
520.
521.
```

6.8 Anexo H. Recibo de originalidad

La siguiente imagen muestra el recibo digital de que se ha entregado la memoria en la herramienta de control de plagio “Turnitin”. Según esta herramienta, da una similitud de un 8% con otras páginas y/o proyectos. Es algo asequible teniendo en cuenta el número de hojas y la cantidad de referencias usadas.




SERGIO GONZALEZ DE LA LAMA | User Info | Messages | Student | English | Community | Help | Logout
turnitin
Class Portfolio
Calendar
NOW VIEWING: HOME > TFG ETSIINF (MOODLE PP) > TURNITIN MEMORIA FINAL (MOODLE PP)
About this page
This is your assignment dashboard. You can upload submissions for your assignment from here. When a submission has been processed you will be able to download a digital receipt, view any grades and similarity reports that have been made available by your instructor.
Turnitin Memoria Final (Moodle PP)
Paper Title
Uploaded
Grade
Similarity
Memoria Final_SergioGonzález.pdf
01 Jun 2024 11:06
--
8%
Copyright © 1998-2024 Turnitin, LLC. All Rights Reserved. (Privacy Policy)
Helpdesk
Research Resources

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
	Fecha/Hora	Sat Jun 01 19:13:51 CEST 2024
	Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
	Numero de Serie	561
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)