



# Tarea 6

**Entrega Final y Examen**

Álvaro Fernández Villa  
Antonio García de la Barrera Amo  
José Hervás Moreno

## Contenido

Tarea 6: Entrega Final y Examen .....	2
Introducción al problema y dominio .....	2
Resolución del Problema y estrategias implementadas .....	2
Estructuras y clases definidas.....	5
Elementos especialmente importantes de nuestra implementación .....	9
Casos De Estudio .....	10
Opinión Personal de la Práctica.....	17
Manual de Usuario .....	18

# Tarea 6: Entrega Final y Examen

## Introducción al problema y dominio

El problema propuesto consiste en la **búsqueda de una ruta** entre un origen y una serie de puntos a recorrer. Para llegar a una solución se ha implementado un proyecto, el cual, tras introducirle el origen y los destinos determinara la ruta entre ellos para visitar todos los puntos a recorrer partiendo de un punto de origen.

El problema consiste en una **búsqueda en un espacio de estados**, por lo que se han desarrollado **diferentes estrategias** de búsqueda que nos servirán para obtener la solución deseada, evaluando de esta forma, cual es la más óptima para cada caso en concreto.

El **dominio del problema** es la información geográfica. La necesaria para llevar a cabo la implementación del proyecto se ha obtenido de [openStreetMap](#), un proyecto colaborativo para la creación y edición de mapas, del cual se obtiene toda la información sobre las calles necesaria para la resolución del problema.

Por tanto, el punto de origen y los puntos a recorrer son nodos de la información obtenida de openStreetMap y representados por un identificador único determinado por dicha plataforma. OpenStreetMap utiliza una estructura de datos topológica.

Para su posterior uso, la solución generada será impresa en un **fichero .gpx**, el cual contendrá toda la información de la ruta generada, así como, otros datos de interés como el tiempo en el que se ha llegado a la solución, la complejidad espacial, el coste que tiene la solución o la estrategia utilizada.

El lenguaje de programación seleccionado para realizar esta práctica ha sido **Java**, utilizando la librería **OSMOSIS** para poder recopilar la información necesaria de openStreetMap.

## Resolución del Problema y estrategias implementadas

Se ha desarrollado un algoritmo de búsqueda para lograr una **solución** al problema propuesto. Este algoritmo se basa en el uso de un **árbol de búsqueda** gestionado por una **cola de prioridad** (Frontera) que formara los **nodos** (hojas) de dicho árbol de búsqueda. Para ordenar la cola de prioridad se hace uso del **valor** generado en cada estrategia, de esta forma, se controla el orden en el que los nodos son insertados en ella.

Hasta que no se encuentre una solución o la cola de prioridad se encuentre vacía se irán generando **sucesores** que serán añadidos a la frontera y de este modo generarán un nuevo nivel en el árbol de búsqueda.

Se encontrará una **solución** cuando los objetivos de un estado en una hoja estén vacíos, lo que significará que todos los destinos han sido recorridos y se puede dar por concluida la búsqueda.

```
public static boolean busquedaAcotada(Problema problema, String estrategia, int profundidadMax, boolean poda) {
    NodoArbol nodoInicial = new NodoArbol(null, problema.getEstadoInicial(), 0.0, 0.0, 0.0, 0.0);
    frontera.insertarFrontera(nodoInicial);

    //System.out.println(nodoInicial);

    boolean solucion = false;

    while(solucion == false && !frontera.esVacia()){

        nodoActual = frontera.eliminarFrontera();
        if(problema.testObjetivo(nodoActual.getEstado())){
            solucion = true;
        }
        else if(nodoActual.getProfundidad() < profundidadMax) {
            ArrayList<NodoVecino> vecinos = problema.sucesores(nodos.get(nodoActual.getEstado().getLocalizacion()));
            //System.out.println(vecinos);
            ArrayList<NodoArbol> sucesores = crearSucesores(problema, vecinos, nodoActual, estrategia, profundidadMax, poda);
        }

    }

    return solucion;
}
```

El problema puede ser **optimizado** mediante una poda de nodos hoja, mediante la cual se rebajara la complejidad espacial al no añadir aquellos nodos que no interesen para la búsqueda de la solución. Esto implica que un nodo hijo será añadido a la frontera siempre y cuando su valor sea mejor que un valor obtenido anteriormente, si no será desechado y no formara parte de la frontera.

```
if (poda) {
    if (!sePoda(nodo)) {
        nodosPoda.put(nodo.getEstado().toString(), nodo.getValor());
        frontera.insertarFrontera(nodo);
    }
} else {
    frontera.insertarFrontera(nodo);
}
```

El método booleano que nos dirá si debemos podar o no es el siguiente:

```
public static boolean sePoda (NodoArbol nodo) {
    boolean sePoda = false;

    if(nodosPoda.containsKey(nodo.getEstado().toString())) {
        if(nodosPoda.get(nodo.getEstado().toString()) < nodo.getValor()) {
            nodosPoda.remove(nodo.getEstado().toString());
            nodosPoda.put(nodo.getEstado().toString(), nodo.getValor());
            sePoda = true;
        }
    }

    return sePoda;
}
```

Se ha establecido una profundidad máxima para todas las estrategias, de esta forma se impide que el algoritmo realice **búsquedas infinitas**, de esta forma pueda encontrar la solución al problema.

Las **estrategias**, como se ha comentado anteriormente, se usan para ordenar los nodos hoja dentro de la cola de prioridad. Las cuales son:

```
if (estrategia.equalsIgnoreCase("anchura")) { // Anchura
    valor = profundidad;
} else if (estrategia.equalsIgnoreCase("profundidad simple")) { // Prof Simple
    valor = -profundidad;
} else if (estrategia.equalsIgnoreCase("profundidad acotada")) { // Prof acotada
    valor = -profundidad;
} else if (estrategia.equalsIgnoreCase("profundidad iterativa")) { // Prof iterativa
    valor = -profundidad;
} else if (estrategia.equalsIgnoreCase("Costo Uniforme")) { // Costo Uniforme
    valor = costo;
} else if (estrategia.equalsIgnoreCase("voraz")) { // Voraz
    valor = problema.heuristica(nodos, estado);
} else if (estrategia.equalsIgnoreCase("a*")) { // A*
    valor = problema.heuristica(nodos, estado) + costo;
}
```

**Anchura:** El valor es calculado haciendo uso de un incremento de la profundidad.

**Profundidad\_Simple:** El valor es calculado restando la profundidad en la que se encuentra el nodo hoja.

**Profundidad\_Acotada:** Al igual que la profundidad simple, el valor se calcula restando la profundidad a la que se encuentra el nodo hoja, con la particularidad que no se generaran sucesores con una profundidad mayor a la estipulada.

**Profundidad\_Iterativa:** El cálculo del valor se hace de la misma forma que el resto de las profundidades, pero este caso en particular ira incrementando un número determinado la profundidad hasta llegar a la profundidad máxima. Para facilitar su cálculo se ha recurrido a la implementación de un código auxiliar.

```
public static boolean busqueda(Problema problema, String estrategia, int profundidadMax,
    int profundidadIterativa, boolean poda){

    int profundidadActual = profundidadIterativa;
    boolean solucion = false;
    while(solucion == false && profundidadActual <= profundidadMax){
        solucion = busquedaAcotada(problema, estrategia, profundidadActual, poda);
        profundidadActual = profundidadActual + profundidadIterativa;
    }
    return solucion;
}
```

**Costo\_Uniforme:** El valor se calcular mediante el costo del nodo hoja.

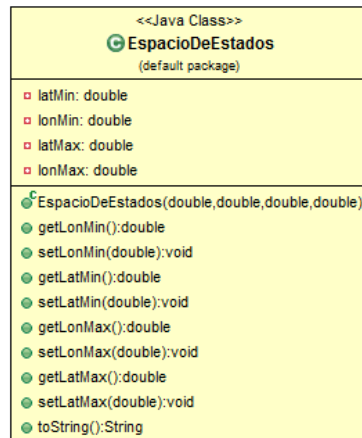
**Voraz:** Se hace uso de la heurística, la cual determina la distancia entre el nodo hoja y los nodos destino.

**A\*:** Se hace uso de la heurística, determinando el valor mediante la distancia entre el nodo hoja y los nodos destino más el costo del camino.

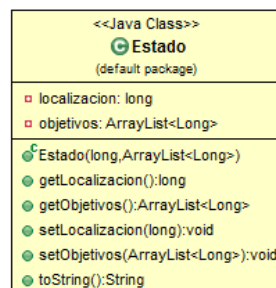
## Estructuras y clases definidas

Las **clases** utilizadas para la correcta implementación del problema propuesto son:

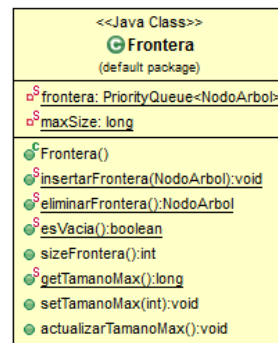
**EspacioDeEstados:** Esta clase representa un espacio de estados, caracterizado por sus atributos: latMin, lonMin, latMax y lonMax.



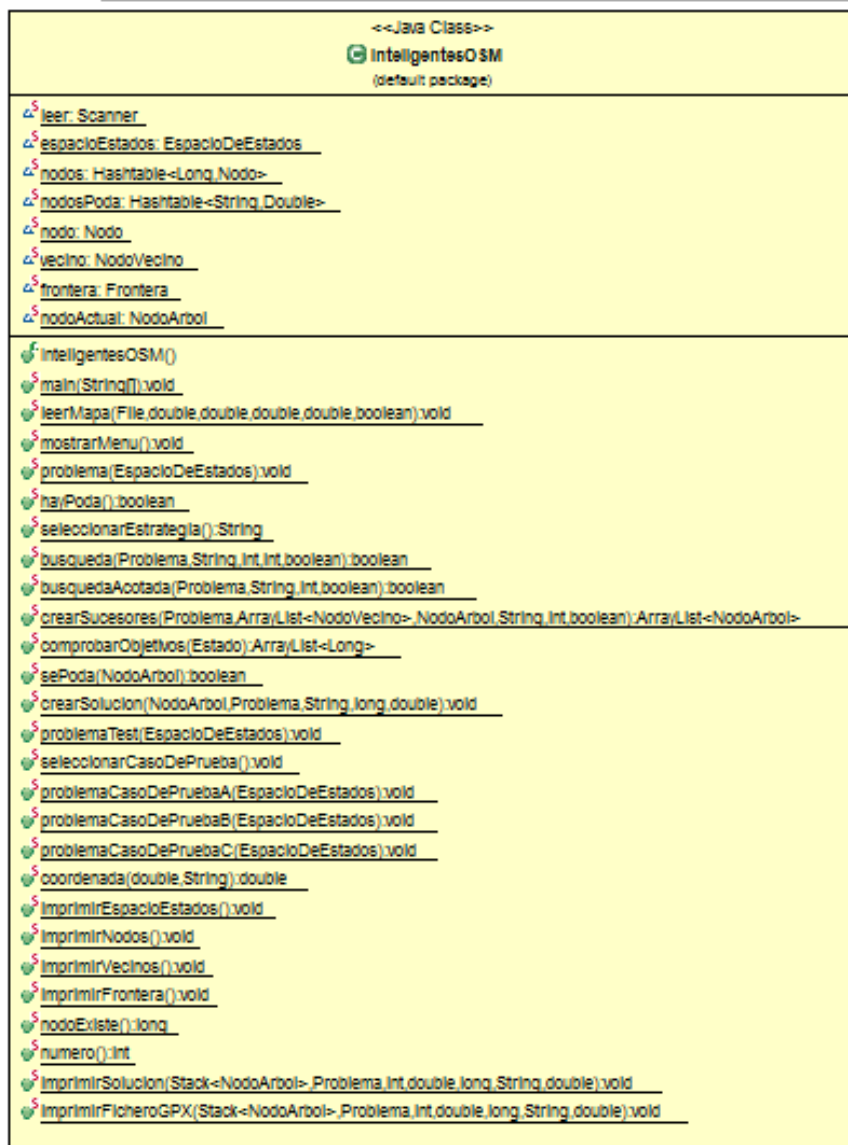
**Estado:** Esta clase representa un estado concreto, entendido como la conjunción de sus atributos, los cuales son: localización y objetivos. Mientras que para imprimir la información del estado utilizamos el *toString()*.



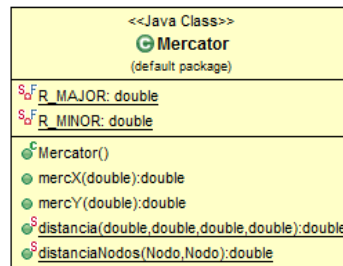
**Frontera:** Esta clase implementa la estructura de datos utilizada para la obtención de la solución. Podemos añadir o eliminar nodos, comprobar si la frontera está vacía, ver cuál es el tamaño actual, o el número máximo de nodos que puede contener.



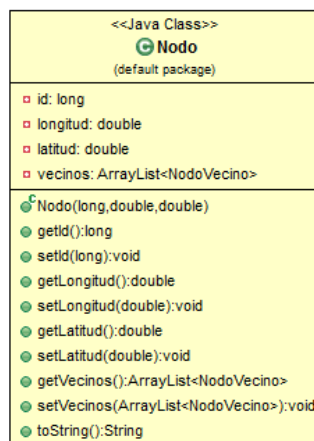
**InteligentesOSM:** Es la clase principal, donde invocaremos al resto de clases en función de las opciones que elija el usuario.



**Mercator:** Esta clase implementa el cálculo de la distancia entre nodos contiguos.

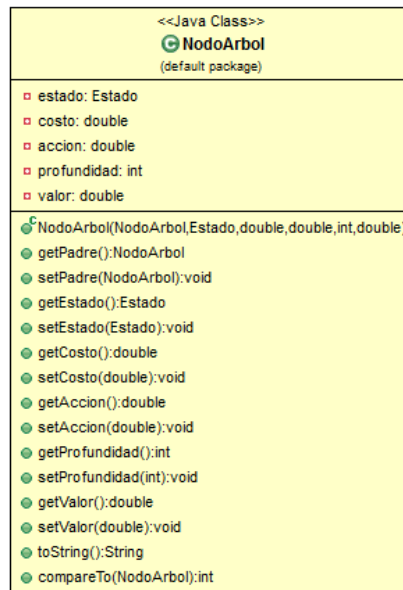


**Nodo:** Esta clase representa un nodo geográfico, representado por los atributos: id, longitud y latitud, y su relación con sus vecinos a través de un ArrayList de `NodoVecino`. Además de los getter y setter necesarios contiene un `toString()`, que posteriormente será llamado para imprimir toda la información de cada nodo en la estructura de datos.

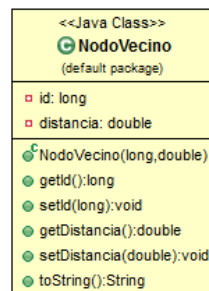


**NodoArbol:** Esta clase representa un nodo del árbol de búsqueda, representado por los atributos: padre, estado, costo, acción, profundidad y valor. Cabe mencionar también el método `compareTo()`, que permite establecer un orden de prioridad entre los `NodoArbol` en la frontera.

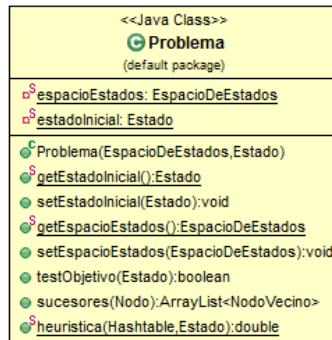




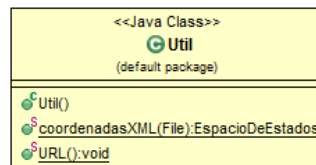
**NodoVecino:** Esta clase representa un nodo geográfico vecino de otro nodo geográfico, caracterizado por sus atributos: id y distancia.



**Problema:** Esta clase representa un problema concreto, entendido como la conjunción de sus atributos, los cuales son: espacioEstado y estadoInicial. Los métodos más importantes son el de los *sucesores()*, con el que mostramos los sucesores del nodo, la función *testObjetivo()*, que devuelve *true* cuando en la búsqueda encuentra solución, y la *heuristica()*, para calcular la distancia máxima entre el estado del nodo y todos sus objetivos. De esta forma cuando el usuario elija la estrategia voraz su valoración será igual a la de la heurística, mientras que si elige A\* su valoración es la heurística más el costo.



**Util:** Conjunto de métodos de utilidad para la resolución del problema.



A continuación se analizarán las estructuras más importantes desarrolladas durante la implementación del proyecto:

- **Hashtable<Long, Nodo> nodo:** Esta estructura almacena los **nodos geográficos** de OSM mediante una hashtable con el identificador del nodo como key y un objeto **Nodo** como valor.
- **ArrayList <NodoVecino> vecinos:** Este ArrayList, contenido en la clase **Nodo**, contiene la información de acceso a los **vecinos de un nodo**, contenida en la clase **NodoVecino**.
- **PriorityQueue<NodoArbol> frontera:** Para llevar a cabo la estructura de la frontera se ha utilizado una **cola con prioridad** (PriorityQueue), porque esta estructura permite añadir los nodos del árbol de búsqueda de menor a mayor valor de los nodos.
- **Hashtable<String, Double> nodosPoda:** Con el fin de **optimizar** el problema nos vamos a crear otra hashtable con la que vamos a disminuir la complejidad espacial. Esto se consigue no añadiendo a la frontera los nodos que no van a formar parte de la solución, ya que los vamos a podar en los nodos hoja. Sólo se añadirán a la frontera aquellos nodos cuyo valor sea mejor que el valor anterior, sino los descartamos.

## Elementos especialmente importantes de nuestra implementación

Una de las estructuras más importante que se ha implementado ha sido la Hashtable. El uso en diferentes contextos a lo largo del proyecto ha logrado optimizar en su conjunto el programa desarrollado. Gracias a su versatilidad se ha utilizado en diversos

contextos, como la creación del grafo de adyacencia de los nodos geográficos que se utilizan como base para el resto del problema o la posibilidad de almacenar un valor para determinar si el nodo hoja debe ser introducido a la frontera o no, utilizado en la optimización.

Otro de los pilares fundamentales es la cola de prioridad que utilizamos para almacenar los nodos hoja y de los cuales se genera una solución válida. Esto es posible gracias al valor generado por las diferentes estrategias, el cual organiza la entrada de los nodos a esta estructura eficiente para este caso.

Para el desarrollo del proyecto se han tenido en cuenta otros elementos, que se han definido para ser lo más óptimos posibles, de esta forma reducir la complejidad temporal. Un buen ejemplo es el uso de elementos recursivos como podemos observar en el método de búsqueda utilizando cuando se quiere aplicar una estrategia iterativa.

Una de las decisiones que se han tomado ha sido la no implementación de una interfaz gráfica por el costo frente a la eficiencia que podría desempeñar en nuestro proyecto.

La selección de donde tomar los datos con los cuales trabajar, leer el mapa en local o descargarlo desde internet de forma totalmente automática, consigue una implementación más fácil para el usuario, así como, los diferentes menús para utilizar el programa en cuestión.

## Casos De Estudio

El **espacio de estados** utilizado en nuestros casos de uso viene definido por las siguientes coordenadas:

[38.98396, -3.93201, 38.98875, -3.92111]

Los estados iniciales para cada caso serán:

**Caso A:** [368287512, [806368904, 812954860]]

**Caso B:** [812954647, [368287510, 803292581, 812954573, 803292576]]

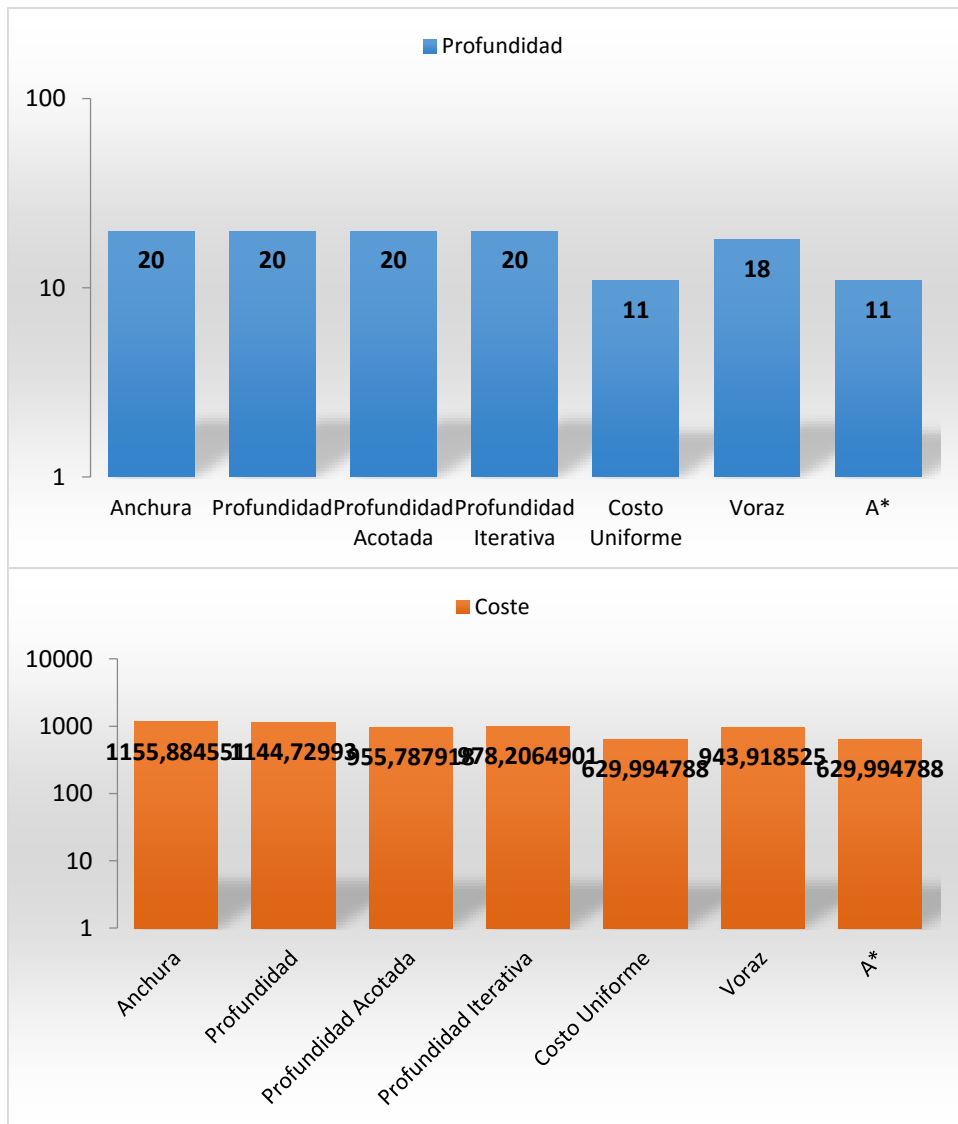
**Caso C:** [525959937, [812954573, 504656546, 803292576, 803292445, 765309500, 803292856]]

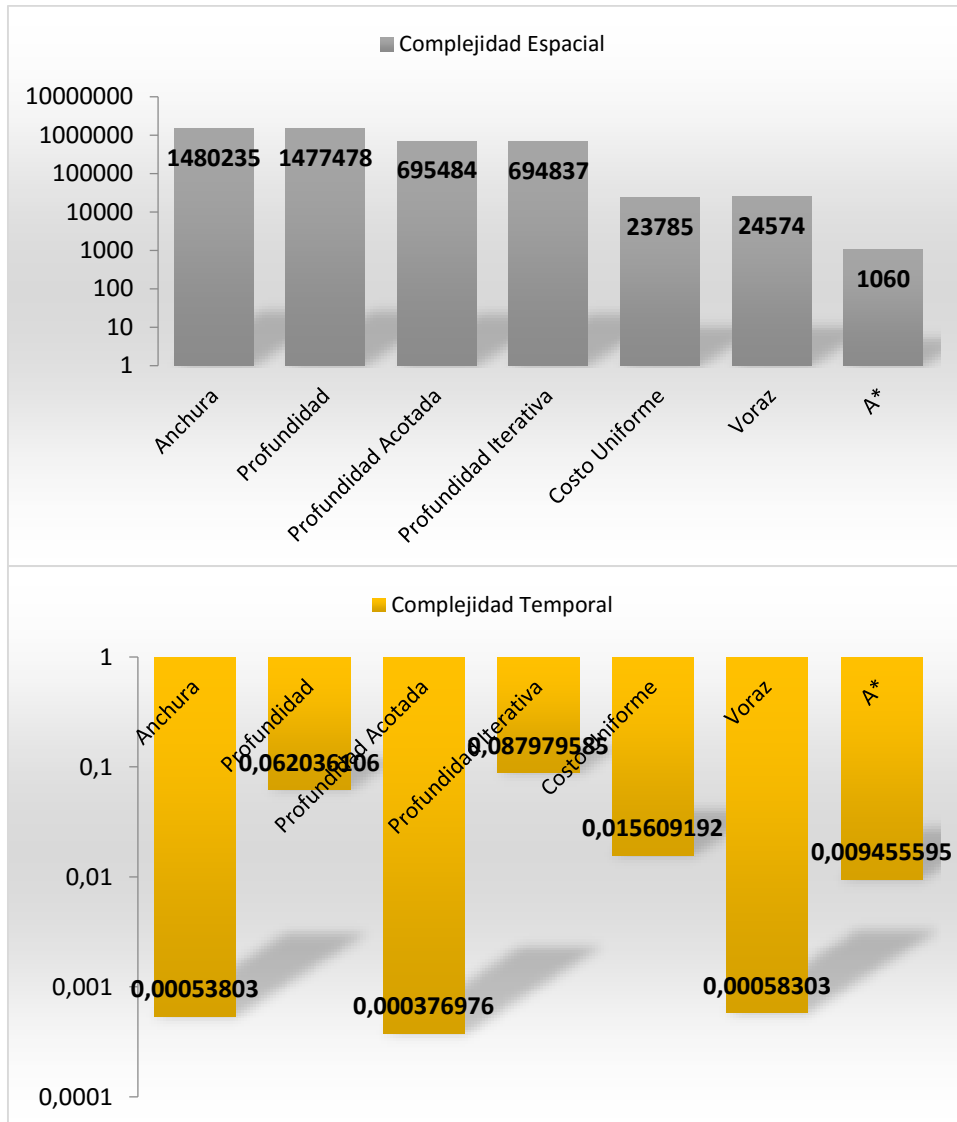
En cada uno de los siguientes casos **estudiaremos** como afecta la elección de una estrategia u otra según la complejidad del problema, así como la complejidad espacial y temporal que obtenemos al utilizarlas en cada problema propuesto.

## Caso A

En este primer caso de estudio vamos a analizar cómo se comporta nuestro sistema con una **complejidad pequeña**, es decir, para una profundidad máxima de 20 y un incremento de profundidad de 5, dando los siguientes resultados para cada una de las estrategias utilizadas:

CASO A				
	Profundidad	Coste	Complejidad Espacial	Complejidad Temporal
Anchura	20	1155,884551	1480235	0,00053803
Profundidad	20	1144,72993	1477478	0,062036106
Profundidad Acotada	20	955,787918	695484	0,000376976
Profundidad Iterativa	20	978,2064901	694837	0,087979585
Costo Uniforme	11	629,994788	23785	0,015609192
Voraz	18	943,918525	24574	0,00058303
A*	11	629,994788	1060	0,009455595

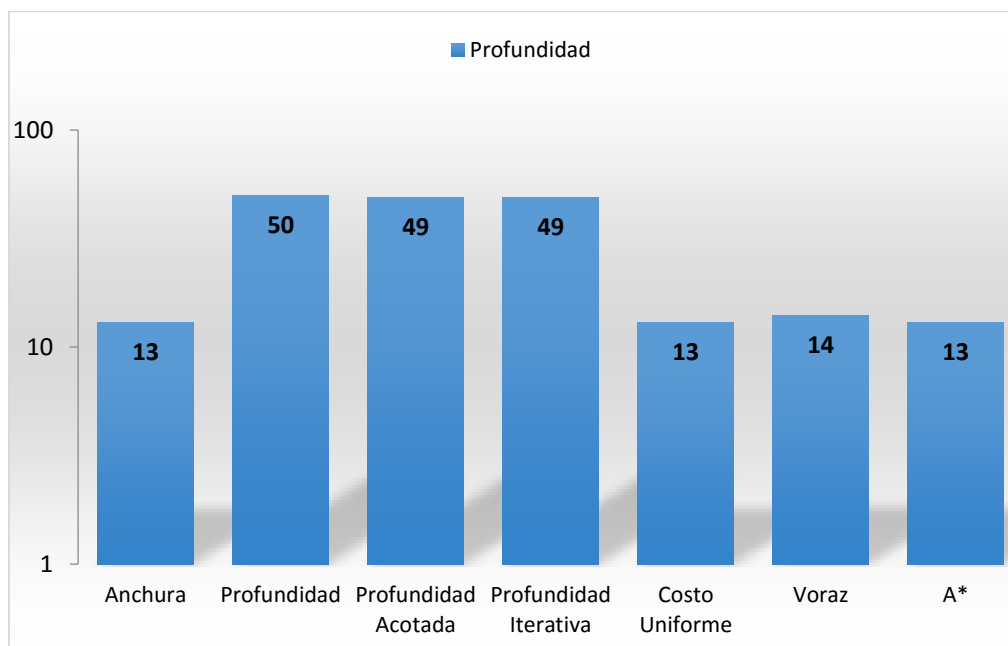


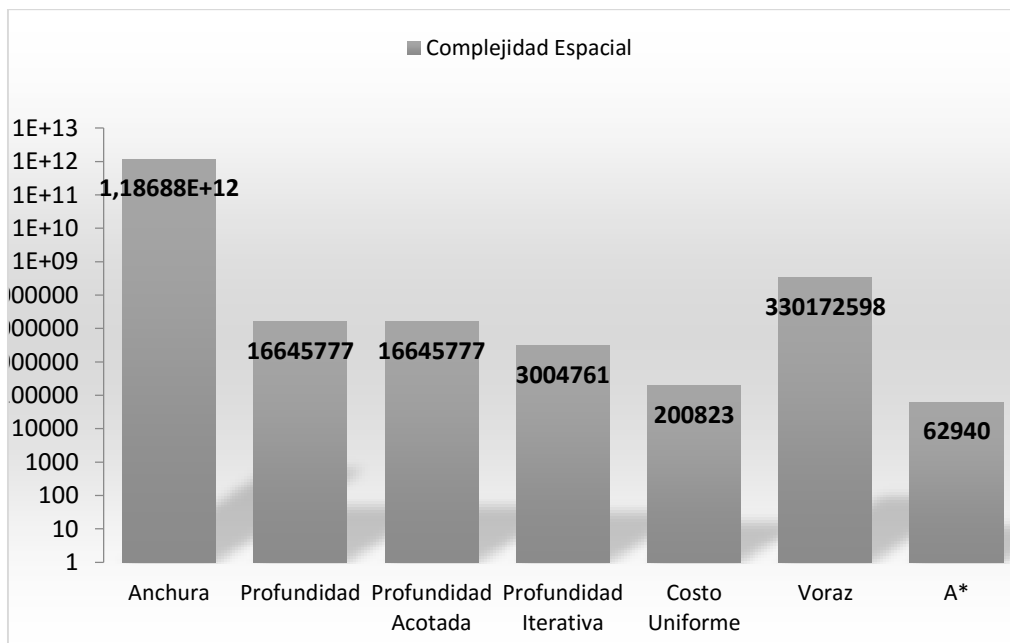
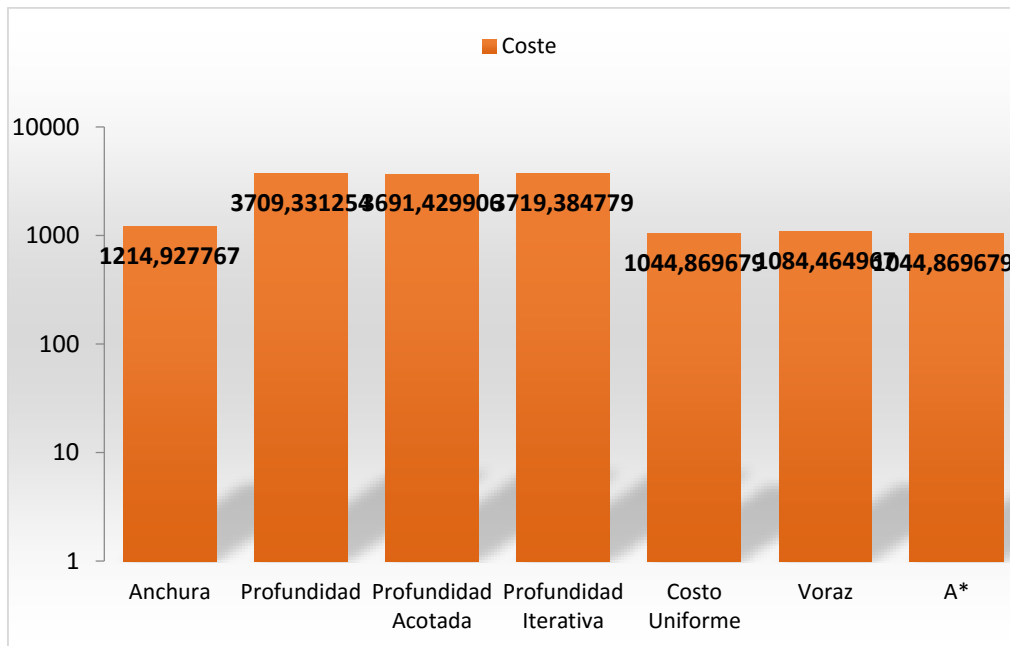


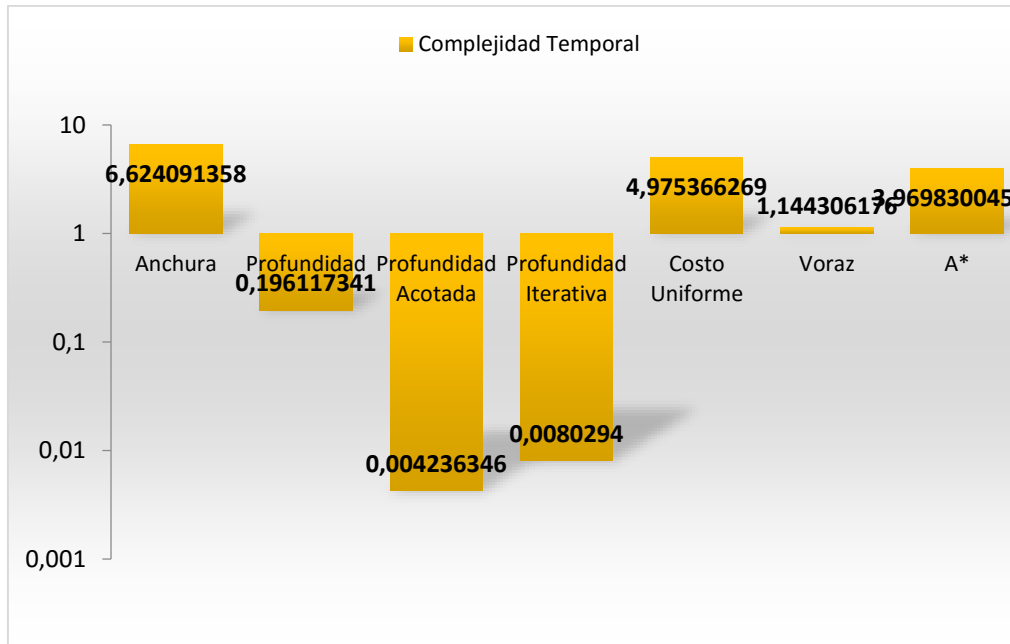
## Caso B

En este segundo caso de estudio vamos a analizar cómo se comporta nuestro sistema con una **complejidad media**, es decir, para una profundidad máxima de 50 y un incremento de profundidad de 10, dando los siguientes resultados para cada una de las estrategias utilizadas:

CASO B				
	Profundidad	Coste	Complejidad Espacial	Complejidad Temporal
Anchura	13	1214,927767	1,18688E+12	6,624091358
Profundidad	50	3709,331254	16645777	0,196117341
Profundidad Acotada	49	3691,429906	16645777	0,004236346
Profundidad Iterativa	49	3719,384779	3004761	0,0080294
Costo Uniforme	13	1044,869679	200823	4,975366269
Voraz	14	1084,464967	330172598	1,144306176
A*	13	1044,869679	62940	3,969830045







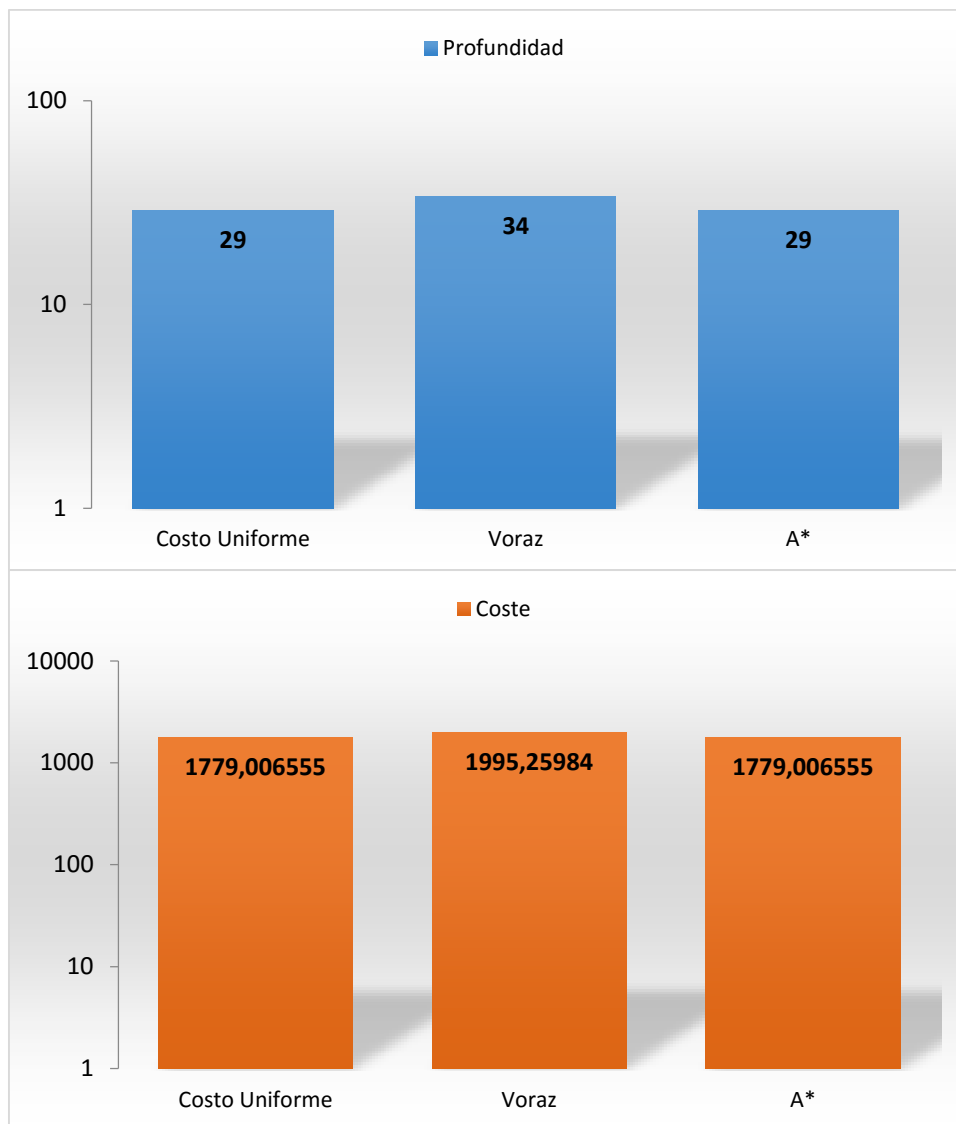
## Caso C

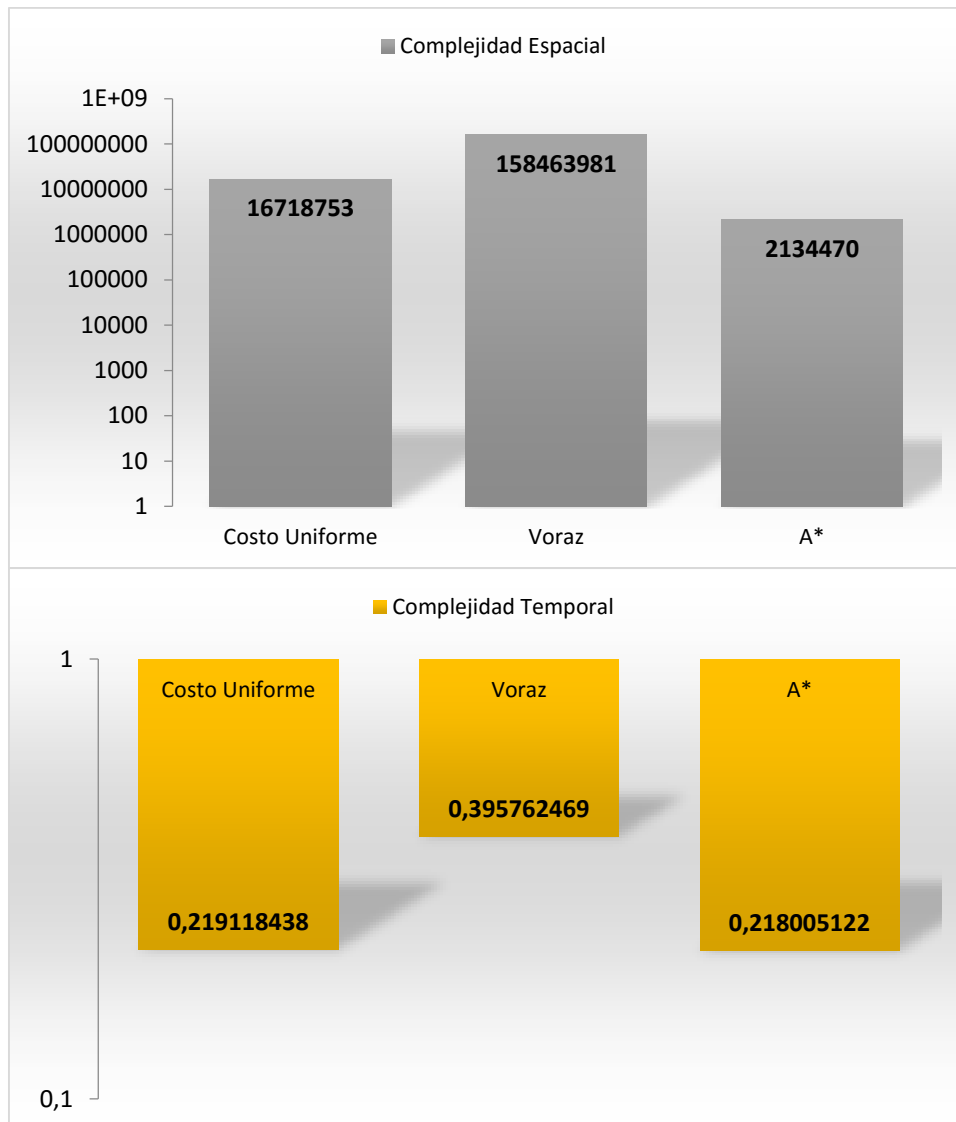
En este tercer caso de estudio vamos a analizar cómo se comporta nuestro sistema con una **complejidad alta**, es decir, para una profundidad máxima de 100 y un incremento de profundidad de 20, dando los siguientes resultados para cada una de las estrategias utilizadas:

CASO C				
	Profundidad	Coste	Complejidad Espacial	Complejidad Temporal
Costo Uniforme	29	1779,006555	16718753	0,219118438
Voraz	34	1995,25984	158463981	0,395762469
A*	29	1779,006555	2134470	0,218005122

Como se puede apreciar, en este caso en particular no se ha desarrollado la comparativa con todas las estrategias puesto que, al ser su complejidad la más alta de los casos de prueba las estrategias no óptimas como la búsqueda en anchura y la búsqueda en las diferentes profundidades no eran interesantes a analizar.







## Opinión Personal de la Práctica

La realización de esta práctica nos ha permitido implementar los conocimientos vistos en teoría, aplicándolos a casos reales. Nos ha dado la posibilidad de estudiar e indagar en los problemas de búsquedas, una parte fundamental de la asignatura.

Hemos podido comprobar de primera mano las complejidades de cada una de las estrategias y como la elección de las mismas según el problema a resolver es importante ya que su eficiencia varía.

El formar un equipo de varios participantes nos ha permitido discutir cómo afrontar la implementación del problema propuesto, dando lugar a diferentes puntos de vista sobre un mismo problema, por lo que nos hemos visto en la obligación de determinar

criterios comunes para completar la tarea, algo enriquecedor ya que no siempre nuestras ideas son las mejores y tenemos que estar abiertos a otras posibles.

El tema escogido, la realización de rutas entre varios puntos, nos ha parecido interesante. Es algo cotidiano y el poder implementar con los conocimientos adquiridos una pequeña muestra es algo alentador. Además el hecho de mostrar la solución en un fichero GPX para poder comprobarlo en un visualizador nos hace ver la funcionalidad de esta práctica y observar en el mapa el camino generado nos hace ver que no solo hemos implementado algo que maneja números y letras, sino que tiene un fin práctico y mundano.

La única dificultad que hemos encontrado fue el hecho de no saber muy bien cómo empezar a desenvolvernó con ella por la inexperiencia con la tecnología utilizada, ya que al ser algo nuevo para nosotros planteaba un reto que de cierta forma nos abrumaba, pero nos hemos dado cuenta hito a hito que podíamos llegar a completarla.

## Manual de Usuario

Cuando el usuario ejecute la aplicación le aparecerá el siguiente menú, donde podrá elegir entre leer el mapa desde un archivo local, o introducir las coordenadas del mapa sobre el que se va a generar la ruta o rutas:

```
Que desea hacer
[1] Leer Mapa En Local
[2] Introducir Coordenadas
```

En el caso de que introduzcamos un número distinto de 1 o 2 saldrá el control de errores hasta que introduzcamos una opción válida.

```
Que desea hacer
[1] Leer Mapa En Local
[2] Introducir Coordenadas
3
No ha elegido una opcion
Que desea hacer
[1] Leer Mapa En Local
[2] Introducir Coordenadas
```

Si introducimos un 1, utilizaremos el mapa de Ciudad Real que tiene las siguientes coordenadas: [38.9839600, -3.9320100, 38.9887500, -3.9211100]. Se nos desplegará el siguiente menú:

```

Selecciona una opcion
[1] Mostrar todos los nodos
[2] Mostrar los vecinos de un nodo
[3] Resolver Problema
[4] Casos de Prueba
[5] Test
[6] Salir

```

De nuevo, si introducimos un valor distinto de los que aparecen en el menú nos saltará el control de errores.

```

Selecciona una opcion
[1] Mostrar todos los nodos
[2] Mostrar los vecinos de un nodo
[3] Resolver Problema
[4] Casos de Prueba
[5] Test
[6] Salir
7
No ha elegido una opcion
Selecciona una opcion
[1] Mostrar todos los nodos
[2] Mostrar los vecinos de un nodo
[3] Resolver Problema
[4] Casos de Prueba
[5] Test
[6] Salir

```

Introduciendo un 1 nos saldrán todos los nodos. En el siguiente fragmento podemos ver un ejemplo de los nodos, junto con su información y nodos vecinos:

```

Nodo [id=812954547, longitud=-3.9264887, latitud=38.9850622, vecinos=[NodoVecino [id=812954421, distancia=73.05072991010515], NodoVecino [id=812954564, distan
Nodo [id=528058542, longitud=-3.9242277, latitud=38.9842028, vecinos=[NodoVecino [id=803292248, distancia=60.66683648794455], NodoVecino [id=3634500805, dista
Nodo [id=2689810002, longitud=-3.9215895, latitud=38.9846119, vecinos=[]]
Nodo [id=812954527, longitud=-3.9281829, latitud=38.9849835, vecinos=[NodoVecino [id=804689200, distancia=36.855867151323665], NodoVecino [id=812954451, dista
Nodo [id=2689810000, longitud=-3.9219498, latitud=38.9845425, vecinos=[]]
Nodo [id=528058525, longitud=-3.9244984, latitud=38.9849697, vecinos=[NodoVecino [id=803292856, distancia=74.18377632263616], NodoVecino [id=765309509, distan
Nodo [id=2689809998, longitud=-3.9214942, latitud=38.9842607, vecinos=[]]
Nodo [id=2689809996, longitud=-3.9236424, latitud=38.9842392, vecinos=[]]
Nodo [id=2689809995, longitud=-3.9251104, latitud=38.984056, vecinos=[NodoVecino [id=504657842, distancia=8.001703210896098], NodoVecino [id=3634500805, dista
Nodo [id=2689809993, longitud=-3.9250493, latitud=38.9839953, vecinos=[]]
Nodo [id=2689809990, longitud=-3.9241827, latitud=38.9837625, vecinos=[]]
Nodo [id=2689809988, longitud=-3.9251264, latitud=38.9836185, vecinos=[]]
Nodo [id=2689809984, longitud=-3.9252493, latitud=38.9836063, vecinos=[NodoVecino [id=3634500806, distancia=5.493621387415493], NodoVecino [id=3634500565, dis
Nodo [id=812954508, longitud=-3.9284602, latitud=38.9850314, vecinos=[NodoVecino [id=812954504, distancia=5.289801945846147], NodoVecino [id=812954451, distan
Nodo [id=812954504, longitud=-3.9284998, latitud=38.9850519, vecinos=[NodoVecino [id=812954501, distancia=13.502748882429724], NodoVecino [id=812954508, dista
Nodo [id=812954501, longitud=-3.9285262, latitud=38.9851443, vecinos=[NodoVecino [id=812954498, distancia=3.9483401843706667], NodoVecino [id=812954504, dista
Nodo [id=812954498, longitud=-3.9285614, latitud=38.9851409, vecinos=[NodoVecino [id=812954493, distancia=8.301300960456357], NodoVecino [id=812954501, distan
Nodo [id=812954493, longitud=-3.9285558, latitud=38.9851100, vecinos=[NodoVecino [id=812954488, distancia=2.075576547605697], NodoVecino [id=812954498, distan

```

Introduciendo un 2 podemos ver los nodos vecinos de un nodo. Para ello el sistema nos preguntará sobre que nodo queremos conocer los vecinos:

```

Introduzca el Nodo del que desea conocer los vecinos
812954421
Nodo [id=812954421, longitud=-3.9271444, latitud=38.9850417, vecinos=[NodoVecino [id=525959937, distancia=6.151571272963814], NodoVecino [id=804689167, distan

```

Introduciendo un 3 podemos generar la ruta con la solución al problema introducido, para esto hay que ir rellenando la información que nos pida la aplicación. Vamos a ver un ejemplo:

```
Introduce el nodo Inicio
368287512
Introduce el numero de objetivos
2
Introduce el nodo Objetivo
806368904
Introduce el nodo Objetivo
812954860
Introduce la profundidad maxima
50
Introduce la profundidad iterativa
50
```

En primer lugar introducimos el nodo inicio y los nodos objetivos, a continuación nos pregunta por la profundidad máxima e iterativa, en el caso de la iterativa sólo nos hace falta si después elegimos la estrategia de profundidad iterativa, en el resto de los casos podemos ignorarla. Luego nos pregunta si queremos realizar la poda, si elegimos que sí encontrará la solución más rápidamente que si elegimos que no, ya que el sistema es más óptimo podando que sino no lo hacemos:

```
Desea realizar la poda?
[1] Si
[2] No
1
```

Posteriormente debemos elegir la estrategia de búsqueda para intentar encontrar la solución. En nuestro ejemplo vamos a elegir la opción 7, correspondiente a la estrategia A\*:

```
[1] Anchura
[2] Profundidad Simple
[3] Profundidad Acotada|
[4] Profundidad Iterativa
[5] Costo Uniforme
[6] Voraz
[7] A*
7
Tiempo Inicial: 2083419735470717
Tiempo Final: 2083419742433136
Tiempo Total: 6962419
Espacio de Estados: [38.98396,-3.93201,38.98875,-3.92111]

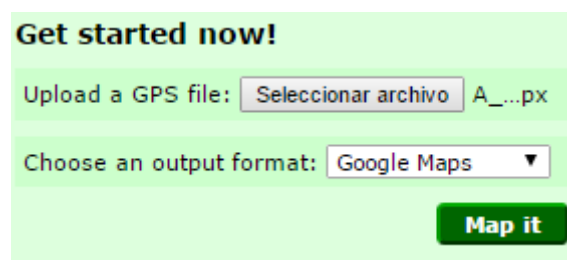
Nodo Origen: 368287512
Nodo Destino 1: 806368904
Nodo Destino 2: 812954860

Estrategia: A*
Profundidad Maxima: 11
Coste de la Solucion: 629.9947879766246
Complejidad Espacial: 1060
Complejidad Temporal: 0.006962419 Segundos
```

Como vemos el sistema ha encontrado solución, podemos ver el coste, la profundidad donde la ha encontrado y las complejidades espacial y temporal. Ahora nos pedirá que introduzcamos un nombre para generar el fichero GPX y podemos comprobar así la solución:

```
Introduzca el nombre del fichero
A_asterisco
```

Una vez que se ha generado el fichero se puede visualizar mediante un navegador web y un visualizador GPX online, como por ejemplo, <http://www.gpsvisualizer.com/>. Seleccionamos el fichero A\_asterisco.gpx y le damos a *Map it*:



Automáticamente nos saldrá la solución a nuestro problema:



Las opciones 4 y 5 del menú son opciones para los desarrolladores de la aplicación, donde han ido haciendo las pruebas pertinentes y elaborando los tres casos de prueba pedidos con las distintas complejidades. Si elegimos la opción 4, nos saldrá un submenú donde podremos ver la solución a los casos de prueba:

- [1] Caso A
- [2] Caso B
- [3] Caso C

Una vez elegida la complejidad que queremos ver: A – pequeña, B – mediana y C – grande, el sistema nos irá pidiendo los mismos datos que hemos visto anteriormente (opción 3 del menú).

La otra opción de generar la solución es introducir manualmente las coordenadas:

```
Que desea hacer
[1] Leer Mapa En Local
[2] Introducir Coordenadas
2
Introduzca la latitud minima:
38.98763
Introduzca la longitud minima:
-3.93043
Introduzca la latitud maxima:
38.98150
Introduzca la longitud maxima:
-3.91874
Comenzamos
Selecciona una opcion
[1] Mostrar todos los nodos
[2] Mostrar los vecinos de un nodo
[3] Resolver Problema
[4] Casos de Prueba
[5] Test
[6] Salir
```

A partir de aquí es lo mismo que se ha explicado anteriormente.

Por último, si introducimos un 6 saldremos de la aplicación.