Ricardo Manuel Ruiz Díaz
Representación del Conocimiento

Índice

Introducción

A continuación, vamos a describir el algoritmo de búsqueda A Estrella escrito en prolog.

El Algoritmo de búsqueda A estrella, es una algoritmo para buscar el camino entre un origen y un fin, teniendo la peculiaridad que ese camino es el que menos coste tiene de todos los posibles.

A continuación, expongamos el siguiente ejemplo, nos encontramos en este mapa y deseamos ir desde *Mehadia* a *Bucharest*:

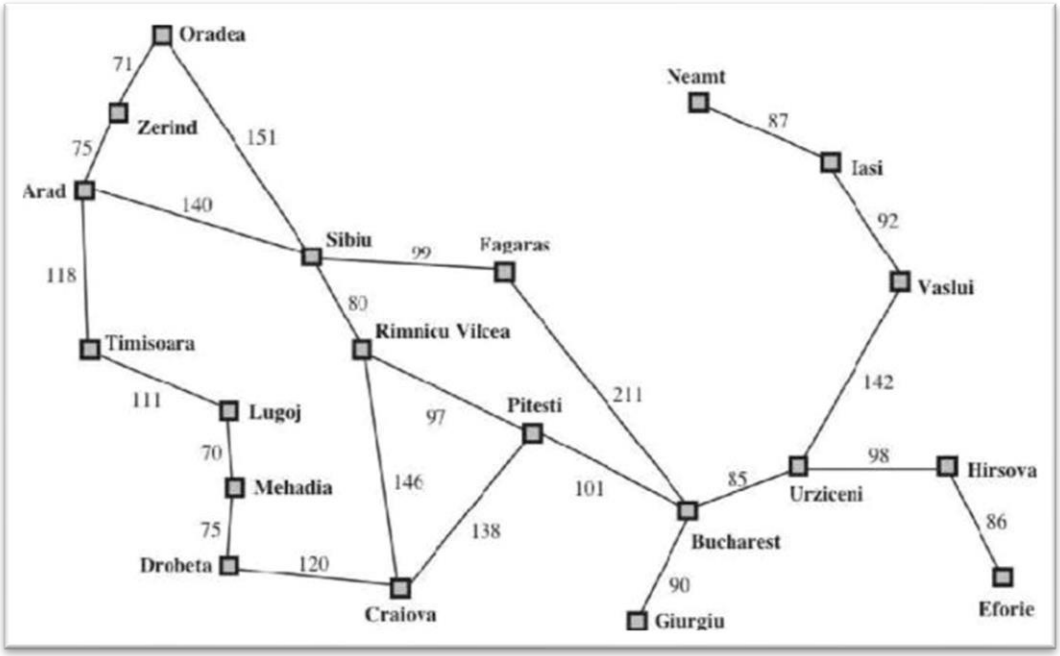

Ilustración 1. Grafo para ejemplo

Pues el algoritmo nos da el siguiente resultado :

```
Camino a recorrer: Mehadia, Dobreta, Craiova, Pitesti, Bucharest,
La distancia total recorrida fue de: 434 kilometros.
```

Ilustración 2. Resultado Ejemplo.

Observamos que el resultado que nos da es el que menor coste tiene.

Descripción del Código

Primero, debemos de señalar, que el grafo lo tenemos insertado en el código de la siguiente forma:

- *indice_ciudad(NumeroNodo, Nombre).*
- *transiciones(NumeroNodoInicial, NumeroNodoFinal, Distancia).*
- *Heurística en línea recta(NumeroNodoInicial, NumeroNodoFinal, Distancia).*

Encontrar Camino (+Origen, +Destino, -[Camino|Coste])

La primera función que nos encontramos sería la principal *encontrarCamino(NombreOrigen, NombreDestino, Resultado)*, donde Resultado es una lista de nodos por los que pasa con el coste concatenado en la posición última de la lista.

Esta función primero busca en el grafo los índices que hace referencia al nodo según su nombre, llama a *buscaHeurísitica* que es la que se encarga de crear los caminos y resolver el algoritmo, y por último invertimos la lista ya que partimos desde el destino hasta el origen para poder imprimirlo por pantalla bien.

Busca Heurística (+Caminos, -Solucion , +Destino)

Esta función, es la más importante ya que es la que se encarga de escoger el próximo camino, eliminar (pasar a visitados) los nodos que ya no nos sirve. También extiende los caminos, es decir, añade a la lista de caminos los caminos nuevos que se abren al "posicionarse" en el nuevo nodo.

Además, se encarga de ir concatenando los caminos seleccionados.

Escoger Próximo (+Caminos, -Proximo , +Destino)

Aquí, se obtienen todos los caminos recorridos hasta el momento y realiza las comparaciones necesarias.

Sólo se detiene cuando se encuentra el menor camino, que lo sabemos gracias a la heurística.

Extender Siguiente Camino (+Prox, -NuevosCaminos)

Este método, según el nodo *Prox* que tengamos, realiza las comprobaciones necesarias (existe movimiento, no está en la lista de camino, …) para poder añadir los siguientes caminos a la lista.

Actualizar Costes Caminos(+ListaCaminos, -NuevosCaminos)

En esta función, según la lista de caminos que llevamos, va actualizando los pesos (distancia) de los nuevos caminos.

Verifica Movimientos (+Origen, +Destino, +Distancia)

Este método, revisa si el movimiento a realizar de ciudad origen a ciudad destino con la distancia expresada existe.

Invierte Camino (+Lista1, -Lista1Invertida)

La única función de este método es invertir una lista.

Concatena Caminos (+Lista, -[Nodo], -NuevaLista)

En este caso, la función que realiza es la de concatenar a una lista un nodo visitado nuevo.

Borra Caminos (+Prox, +Caminos, -CaminosRestantes)

Esta función, borra los caminos innecesarios que tengamos en nuestra lista de abiertos.

ImprimerCamino(+[ListaCiudades | Coste])

En este caso, esta función va imprimiendo todas las ciudades de izquierda a derecha de la lista, y cuando sólo queda un elemento en la lista, sabemos que es el coste y cambia el formate del *write* para sacar por pantalla el coste.

Nuevas Funciones

Para el correcto funcionamiento del algoritmo, hemos tenido que utilizar las siguientes funciones no dadas en las clases de teoría:

- **!**: llamado negación como fracaso, es la negación de la clausula anterior a la exclamación, pero tiene la particularidad de que con este modo se libera la restricción sintáctica de que las regals tengan que ser cláusulas de Horn.
- **not()**: es la negación de lo que tenga entre parentesis.
- **member()**: es cierto si el primer elemento de la función es mienbro del segundo elemento del parámetro, utilizado para saber si un nodo o camino estaba en la lista.

Bibliografía

Hay que destacar que el código de la práctica lo hemos sacado de la siguiente URL, siendo el propietario de él Jorge Carlos Valverde Rebaza:

- http://jc-info.blogspot.com/2012/04/star-algorithm-prolog-code.html

Definición del algoritmo de búsqueda A Estrella:

- https://es.wikipedia.org/wiki/Algoritmo_de_b%C3%BAsqueda_A*

Explicación negación como fracaso / not:

- http://dit.upm.es/~gfer/ssii/rcsi/rcsisu85.html

Explicación member:

- https://www.swi-prolog.org/pldoc/man?predicate=member/2