SEGUNDO TRABAJO DE LABORATORIO

Inteligencia Artificial 2017

24 de mayo del 2017

Raúl Calvo Laorden & Miguel Ángel García Romeral

Índice

Introducción de la práctica

El objetivo de esta práctica es elaborar un programa Racket que tenga como entrada una lista de ciudades, una matriz de adyacencia ponderada entre estas ciudades con la distancia en kilómetros, el método de búsqueda a elegir entre Anchura, Profundidad, Profundidad Iterativa y Coste Uniforme, y par de ciudades, un origen y otro destino.

En la salida de este programa se muestra la ruta entre la ciudad de partida y la de llegada, con su coste en kilómetros tota. En caso de no existir ninguna ruta entre ambas ciudades, se avisa por pantalla al usuario.

Utilización del programa

Primero debemos tener preparados los datos de entrada del programa, por lo que necesitamos una lista de ciudades, una matriz de adyacencia, el método decidido y las ciudades destino y origen. Un ejemplo es la que se muestra a continuación:

(list "A" "B" "C" "D" "E")

(list (list 0 1 7 inf 11)(list inf 0 6 3 inf)(list 5 6 0 2 2)(list inf inf 3 0 inf)(list 2 inf inf inf 0))

En este caso creamos una lista con 5 ciudades y su matriz de adyacencia. A continuación, debemos seleccionar un algoritmo de búsqueda sabiendo que debe ser 1 (Anchura), 2 (Profundidad), 3 (Profundidad Iterativa), 4 (Coste Uniforme). En este ejemplo seleccionaremos el modo 2, es decir, la búsqueda en profundidad.

Por último, debemos saber la ciudad origen y la ciudad destino, que en este ejemplo serán "A" y "E" respectivamente.

Una vez tenemos todos los datos confirmados, esto es que las ciudades están en la lista y el tamaño de la lista coincide con el de las dimensiones de la matriz, ejecutamos la función.

(run (list "A" "B" "C" "D" "E") (list (list 0 1 7 inf 11)(list inf 0 6 3 inf)(list 5 6 0 2 2)(list inf inf 3 0 inf)(list 2 inf inf inf 0)) 2 "A" "E")

Funcionamiento del software

A continuación, describimos las funciones principales. Es importante mencionar que, debido a la definición del problema y la arquitectura de Racket, algunos algoritmos han sido modificados con el fin de mantener de resolver todos los fallos que Racket pudiese ocasionar. Un ejemplo es la búsqueda en profundidad, en caso de que encontrase un ciclo, Racket continuaría la búsqueda mientras tenga memoria disponible. Para evitar fallos así, las ramas del grafo se podan cuando la ciudad que se comprueba ya se encuentra en la ruta o cuando la ruta supera el número de ciudades, todo ello sin alterar la filosofía de los algoritmos.

Anchura

(anchura)

Inicializa el método en anchura con dos parámetros: la posición de la ciudad destino y la lista de abiertos representada mediante una lista de listas de 4 elementos (actual fin lista_visitados coste_camino).

(anch fin rest)

Recibe por parámetros el número de nodo fin (1 a N) y la lista de abiertos, representado mediante una tupla o lista de 4 elementos: (actual fin lista_visitados coste_camino). Se coge la cabeza de la lista de abiertos y se comprueba si la longitud de la ruta es menor que la lista de ciudades, porque si fuese mayor no se habría llegado el destino. Si el nodo fin y el nodo actual coinciden y el coste es menor que infinito se comprueba esa solución (llamando a verificar_ruta) y se sigue mirando la lista de abiertos. Si no coincide se aplica el método de anchura pero se le añaden a la lista de los sucesores del nodo actual. Y así hasta que la lista de abierto está vacía que acaba la búsqueda.

(add_hijos queda ruta act total)

Devuelve una lista con los sucesores de un nodo a la lista actual para el método en anchura, ya que los pone al final de la lista de nodos a visitar. Se le pasan como parámetro un nodo auxiliar para recorrer por la lista (de 1 a N), la ruta actual, el nodo actual (de 1 a N), y el coste total de la ruta actual.

Profundidad

(profundidad)

Aplica el método en profundidad llamando a la función (prof ant inicio fin ruta total tope) pasándole como parámetros el anterior, el inicio, el fin, la ruta, el total y el número de ciudades como tope. Este tope sirve para reutilizar el código de la búsqueda en profundidad iterativa. El tope de la búsqueda en profundidad es el número de ciudades.

(prof ant inicio fin ruta total tope)

Verifica si un nodo es solución en profundidad. Para esto llama a verificar_ruta y si no es solución se llama a (prof_aux actual lista_dest ruta total fin rec_list tope).

(prof_aux actual lista_dest ruta total fin rec_list tope)

Crea la lista de sucesores y trata los nodos en orden para verificar que se cumple a búsqueda en profundidad. Para esto recibe como parámetro el nodo inicial, los costes de llegar desde el nodo inicial a los demás nodos, la ruta actual el coste total, el nodo fin, un indicador de que nodo visitar y el tope para dejar de buscar, es decir, el número de ciudades posibles.

Profundidad iterativa

(profundidad_iterativa tope)

Aplica el método en profundidad iterativa. Busca una posible solución con un tope. Si no ha encontrado ninguna solución, se busca de nuevo con un tope más grande. En el momento en que haya una solución, no buscará ninguna más, por lo que tienen prioridad las rutas con menos saltos que las de menor coste. Para hacer esto llama a la función (prof ant inicio fin ruta total tope) indicando el tope, empezando por 2. Y sumándole 1 cada vez que termina y no ha encontrado ninguna solución hasta superar el número de ciudades que hay.

Coste Uniforme

(coste_uniforme)

Aplica el método en profundidad. Para esto llama a (cu ant inicio fin ruta total).

(cu ant inicio fin ruta total)

Verifica si un nodo es solución en coste uniforme. Para esto llama a (verificar_ruta) y si no es solución se llama a (cu_aux actual lista_dest ruta total fin) para que expanda sus nodos.

(cu_aux actual lista_dest ruta total fin)

Busca entre los hijos de un nodo el que tenga menor coste para ser expandido. Una vez expandido, buscará de nuevo entre sus hermanos recursivamente (sin tener en cuenta a los ya seleccionados) y expande esos hermanos.

Otros

(get_coord de a)

Obtiene el valor de la posición de la fila de y columna a. El rango de elementos va de 1 a N.

(repeticiones elemento lista)

Indica cuantas veces esta un elemento en la lista. Se utiliza para comprobar que un comino no pasa dos veces por la misma ciudad.

(switch_algoritmo)

Realiza el algoritmo que se le ha pasado. Se utiliza en el método principal del programa.

(verificar_ruta ruta total)

Comprueba que la solución encontrada es mejor que la anterior y si es así la pone como nueva.

(minimo a b)

Devuelve el valor más pequeño entre a y b.

(menor lista)

Devuelve el menor valor de una lista.