ANÁLISIS Y DISEÑO DE ALGORITMOS

INFORME DE LA PRÁCTICA 3

Diciembre de 2015

Senmao Ji Ye
Jairo Velasco Martín

Introducción.

El problema del caballo puede generalizarse como la búsqueda de un camino hamiltoniano en un grafo, esto es encontrar un camino que recorra todos los nodos pasando una única vez por cada uno. Aunque el problema de encontrar un camino hamiltoniano es NP completo, este es un caso especial que se puede resolver en tiempo lineal.

Hemos planteado una solución por fuerza bruta empleando búsqueda primero en profundidad, e implementado una solución que aprovecha la heurística de la regla de Warnsdorff.

Instrucciones para la ejecución del programa.

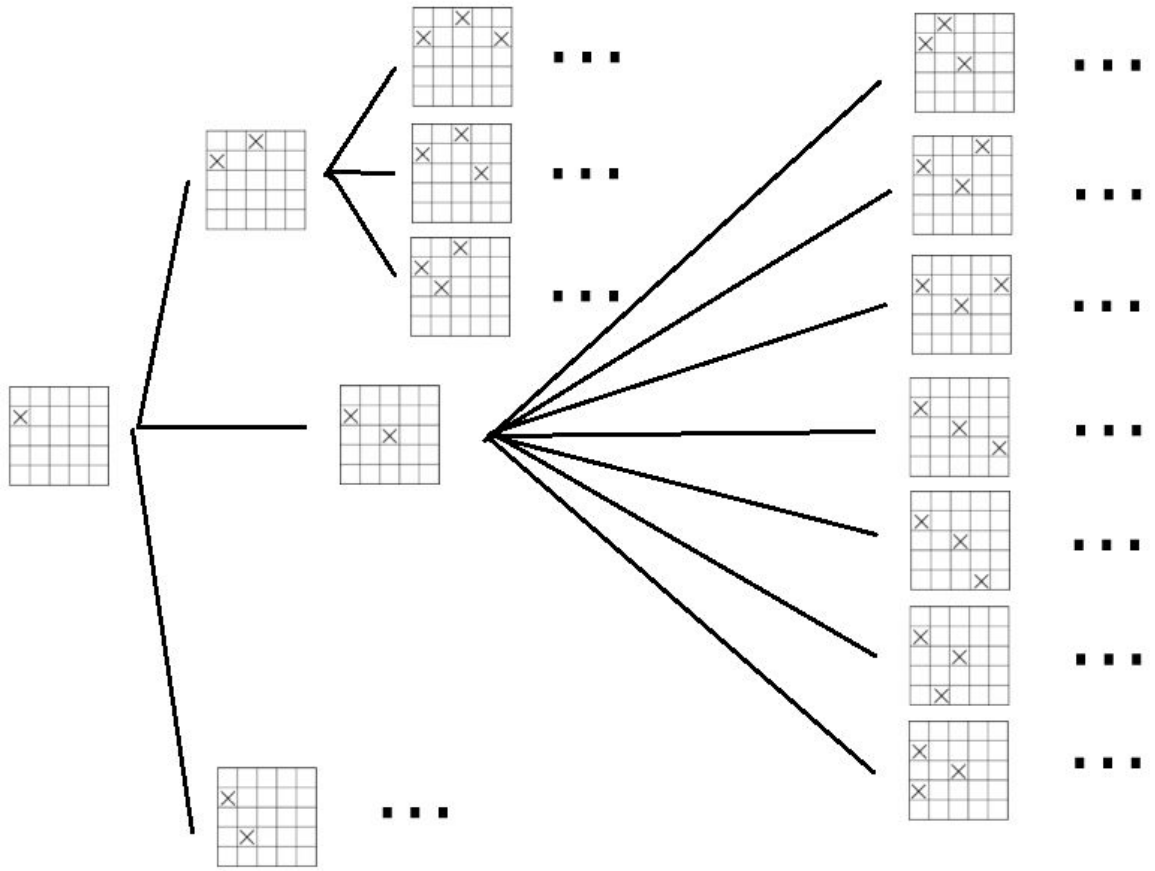
Para la realización de la práctica hemos empleado Java 7 y el IDE Netbeans.

Adjuntamos el proyecto completo de Netbeans, no obstante el programa está compuesto por un único fuente llamado ProblemaCaballo.java, que puede ser compilado y ejecutado en cualquier otro entorno. Este pide por teclado las coordenadas X e Y de la casilla de partida, y el un valor N que determinará el tamaño del tablero (será NxN). No hemos tenido en cuenta tableros rectangulares, por eso sólo permitimos crear tableros cuadrados. Si se introduce una casilla fuera de rango, vuelven a solicitarse los datos.

La salida del programa es un fichero llamado salida_p3_senjiye_jaivela.txt que se genera en el directorio de ejecución.

Solución por fuerza bruta.

Una posible solución para el problema consiste en aplicar fuerza bruta, la idea es generar un grafo dirigido de tipo árbol que contenga todos los posibles caminos, dicho grafo tiene como nodos los recorridos posibles a través tablero y por aristas las posibles transiciones.

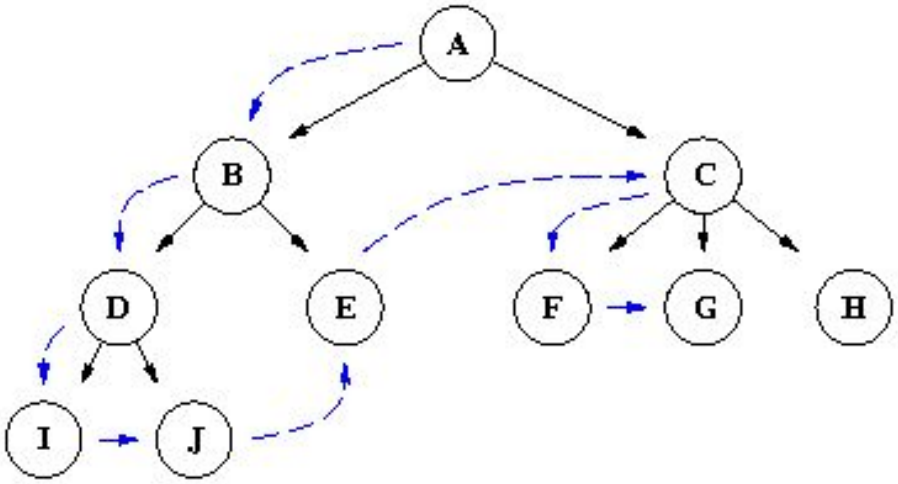


(Aunque las líneas del dibujo no tengan punta de flecha, el grafo es dirigido, de izquierda a derecha)

Una vez que se tiene este análisis de todos los recorridos, queda buscar el que es solución, es decir, aquel cuyo tamaño es el deseado. Para ello se realiza una búsqueda primero en profundidad, de manera que si un recorrido es del tamaño deseado se ha llegado a la solución, en caso contrario se realiza backtracking para seguir buscando.

También podría buscarse en el árbol aplicando búsqueda primero en anchura, aunque resultaría menos eficiente.

Como el grafo del ejemplo es demasiado extenso, emplearemos otro para explicar de forma análoga como se recorre un grafo en profundidad:



Como puede observarse, se toma el primer nodo a la izquierda hasta llegar al último nivel, momento en que se toman los nodos hermanos de los ya tomados y se vuelve a aplicar la regla de tomar el primero a la izquierda. La búsqueda primero en profundidad termina cuando se llega a un nodo solución, o cuando se recorre el grafo completo sin que exista solución.

La principal ventaja de la solución por fuerza bruta es que al generarse todos los posibles recorridos y usar búsqueda PP si existe una solución tenemos garantía de que esta siempre se encuentra.

El gran inconveniente es que el número de recorridos presenta un crecimiento exponencial, por lo que sólo es aplicable a tableros de muy pequeño tamaño, ya que en otro caso se vuelve incomputable (por ejemplo un tablero de 6x6 tiene más de 1000 millones de recorridos). Otras soluciones plantean aproximaciones de tipo divide y vencerás, de manera que se trabaje con fragmentos de tablero más pequeños.

Solución heurística.

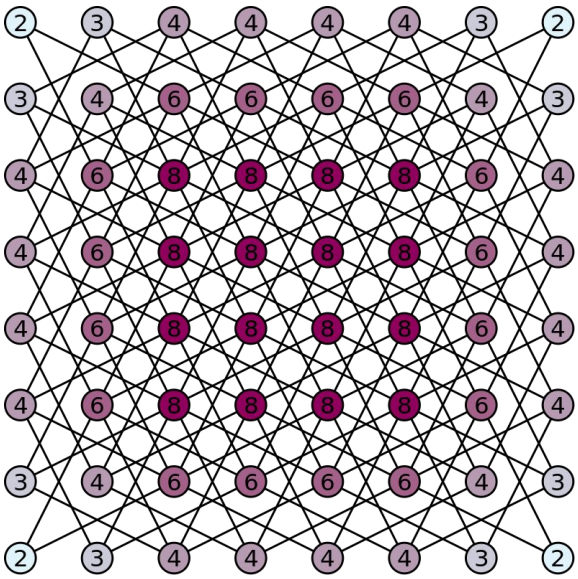
Hemos decidido implementar un algoritmo de búsqueda heurística, aplicando la conocida regla de Warnsdorff, esta heurística consiste en mover el caballo siempre hacia la casilla que tenga la menor cantidad de movimientos (futuros) posibles. Al calcular el número de futuros movimientos posibles de cada casilla no se contabilizan aquellos que visitan una casilla que ya ha sido visitada.

La regla de Warnsdorff no garantiza un recorrido completo, el problema de esta heurística se da cuando se producen empates, es decir, cuando al menos dos casillas tienen el mismo número de posibles movimientos, ya que esto puede ocasionar que no se llegue a la solución. Existen diversas propuestas para deshacer los empates, por ejemplo aplicar de nuevo la regla para las casillas empatadas (Pohl) o tomar la más alejada del centro (Roth). En nuestro caso hemos optado por escoger los nodos en orden, esto es de arriba a abajo y de izquierda a derecha,en la matriz de adyacencia.

Grafo

El grafo que hemos empleado para representar el problema tiene por nodos las casillas del tablero, cada uno de estos nodos tiene asociado un peso que viene determinado por el número de posibles movimientos desde esa casilla, siendo las aristas los posibles movimientos legales del caballo. Su tamaño es n x n, y corresponde con el tamaño del tablero. El número total de aristas es 4(n-2)(n-1). Se trata de un grafo no dirigido.

Cabe destacar que a pesar de que usamos pesos, no se trata de un problema de optimización, sino que es de búsqueda, ya que no buscamos el camino óptimo, sino el camino en sí. Los pesos simplemente permiten aplicar la heurística.



Grafo para un tablero 8x8 en su estado inicial.

Por ejemplo, el nodo de la esquina superior izquierda representa a la casilla (0,0) y tiene un peso asociado de 2, porque estos son los movimientos que el caballo puede realizar desde esta casilla.

Este grafo es adecuado porque permite la aplicación directa de la regla de Warnsdorff sobre él, de manera que dado un nodo el siguiente en el recorrido será el de menor peso.

Sobre la implementación.
Para representar el grafo hemos empleado una matriz de adyacencia, de manera que quedan representadas las conexiones de salto del caballo entre las casillas del tablero. Para inicializar la matriz, obtenemos las conexiones apoyándonos en un array con los movimientos legales del caballo, de forma que una casilla se relaciona con aquellas para las que exista un movimiento. Cuando una casilla es visitada, han de eliminarse de la matriz de adyacencia todas las conexiones que la contenían, esto se traduce poniendo a 0 la columna correspondiente.

El procedimiento principal de nuestro programa es una función recursiva llamada solve que aplica la regla de Warnsdorff para recorrer el grafo, resumidamente hace lo siguiente:

```
solve(tablero, casilla, cont)
      si cont == final
            se ha llegado a la solución
      para cada movimiento
            obtener vecino
            calcular peso del vecino
            quedarse como candidatos los de menor peso
      para cada candidato
            solve(tablero, vecino de menor peso, cont++)
```

A partir de ahora llamaremos vecinos de una casilla a aquellas casillas a las que pueda saltar el caballo mediante un movimiento legal.

Además de la ya mencionada función recursiva solve, tenemos una función que calcula el peso de una casilla, básicamente cuenta qué casillas son accesibles desde una dada, de manera que una casilla destino es contabilizada si está en rango y no ha sido visitada.

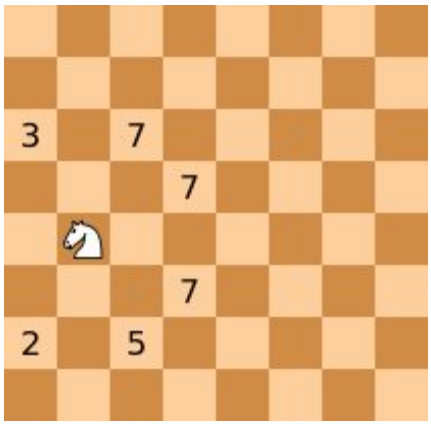
Hemos decidido llevar a cabo esta implementación por ser más óptima que otras estrategias como la ya comentada de fuerza bruta.

Posibles mejoras.
Como en este caso particular el tablero sólo puede crecer hasta un tamaño de 8x8, la matriz de adyacencia no resulta especialmente mala, ya que esta tendría un tamaño de 64x64, no obstante, si se deseara calcular tableros mayores podría resultar más óptimo implementar una lista de adyacencia, puesto que no será tan grande al no incluir conexiones que no se dan. Además, el problema también puede resolverse sin matrices o listas de adyacencia, simplemente representando el tablero mediante una matriz y limitando los posibles movimientos por la matriz, en un principio habíamos aplicado esta solución, pero debido a los objetivos de la práctica finalmente consideramos mejor solución implementar el grafo con una matriz de adyacencia.

Otra posible mejora está en el cálculo del peso de las casillas, ya que resultaría más óptimo inicializar los pesos e ir restando a medida que se producen visitas de casillas, en lugar de ir calculando, que es la estrategia que utilizamos.

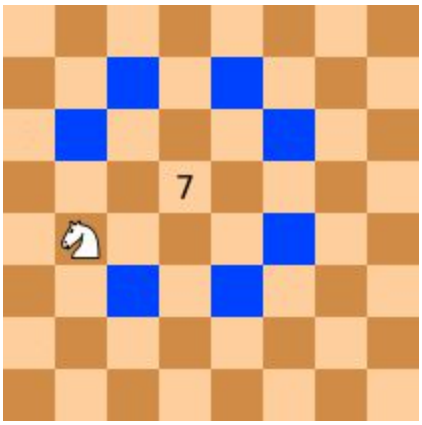
Ejemplo visual de aplicación del algoritmo.

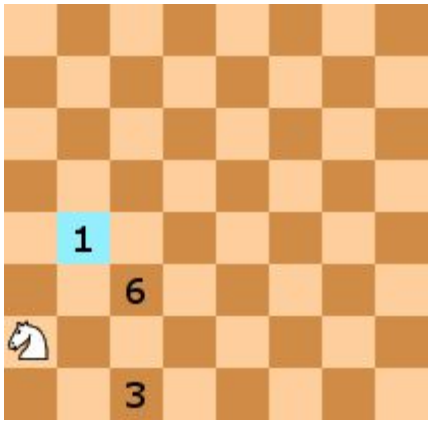
Supongamos para este ejemplo que se parte de la casilla (1,4), el primer paso es obtener los pesos de los “vecinos”, estos son los nodos que conectan con el (1,4).



Como hay que descontar las casillas visitadas, estos vecinos tienen un movimiento posible menos que en el estado inicial (porque el (1,4) ya está visitado).

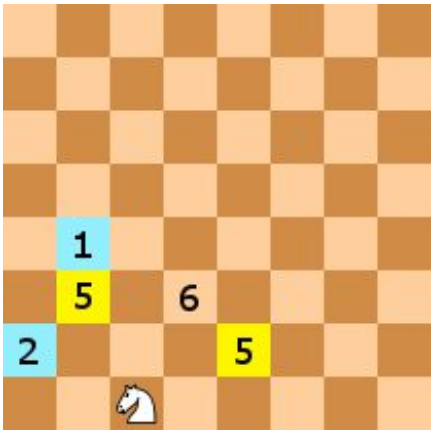
Por ejemplo el nodo (3,3) tiene 7 posibles movimientos, los marcados en azul oscuro.





Se toma de entre todos los vecinos el de menor peso, por lo que el caballo salta a la posición (0,6). Y se calculan los pesos de los nuevos vecinos.

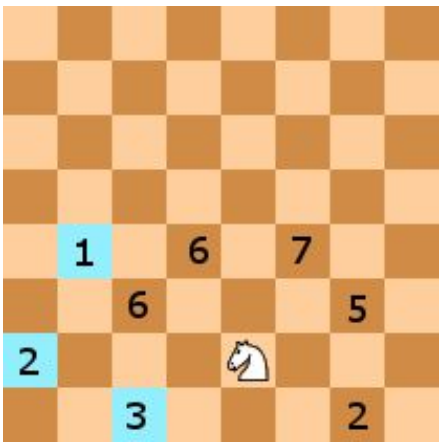
Señalamos con un sombreado azul claro y con un 1 la casilla (1,4) para indicar que este es el primer paso del recorrido.



Una vez más el caballo salta al nodo de menor peso.

Marcamos en azul los pasos del recorrido.

En este punto se produce un empate de peso entre los nodos resaltados en amarillo.



Seguimos con el mismo proceso hasta completar el tablero (Si es que contiene una solución)

14	27	16	51	12	29	34	61
17	52	13	28	55	60	11	30
26	15	54	59	50	33	62	35
53	18	25	56	63	58	31	10
24	1	64	49	32	45	36	47
19	40	21	44	57	48	9	6
2	23	42	39	4	7	46	37
41	20	3	22	43	38	5	8

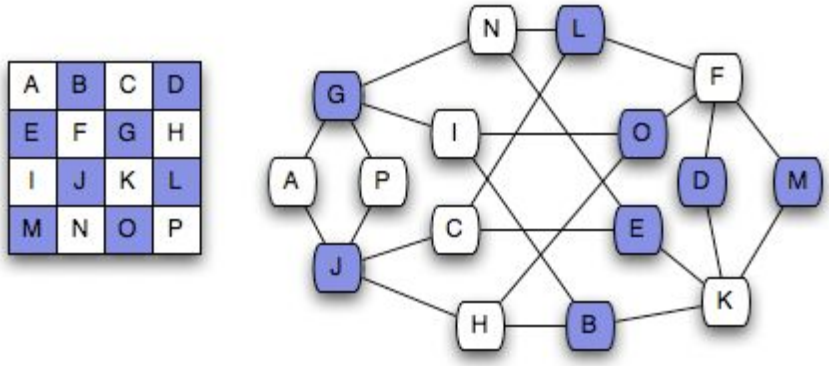
Esta es la solución para el caso ejemplo, donde podemos observar todo el recorrido.

Consideraciones sobre el tamaño del tablero:

No existe solución para tableros de tamaño 3x3 o 4x4, a continuación pasamos a detallar las causas.

En el tablero 3x3 no encontramos solución ya que, la casilla central es un nodo aislado, y no está conectado con el resto de los nodos.

Como podemos ver en la siguiente imagen, el tablero 4x4 lo podemos subdividir en 4 subgrafos conectados, 2 laterales {G-A-J-P} y {F-D-K-M} y 2 centrales {N-E-C-L} y {I-B-H-O}.



Primero nos fijamos en los 2 subgrafos laterales, se puede observar que para recorrerlos, debería empezarse en uno de ellos y acabar en el otro. El problema se encuentra en los subgrafos centrales, ya que a partir de uno no se puede llegar al otro, sino que debemos pasar al grafo lateral, por lo que no podríamos completar un recorrido completo del tablero.

En los grafos n x n, donde n es impar, no podemos encontrar solución si en la casilla inicial, la suma de fila+columna, es impar. O dicho de otra manera, si la casilla inicial es blanca. La razón es simple, en los tableros de tamaño impar, hay 1 casilla negra más que las blancas. Al ser el movimiento del caballo alternativo entre casillas blancas y negras, debemos empezar por una negra.

Fuentes consultadas.

- Referencias sobre Knights Tour:
  https://en.wikipedia.org/wiki/Knight%27s_tour
  http://faculty.olin.edu/~sadams/DM/ktpaper.pdf
  http://www.borderschess.org/KnightTour.htm
  http://functionspace.com/topic/2942/Warnsdorff--039;s-Algorithm

- Referencias sobre búsqueda PP:
  http://casimpkinsjr.radiantdolphinpress.com/pages/cogsci109/pages/search_reading/search.html

- Referencias sobre caminos hamiltonianos
  http://stackoverflow.com/questions/5766160/enumerate-all-hamiltonian-paths
  https://en.wikipedia.org/wiki/Hamiltonian_path