Algoritmos y Estructuras de Datos III

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Trabajo Practico 1

Segundo Cuatrimestre 2014

Integrante	LU	Correo electrónico
Ricardo Colombo	156/08	ricardogcolombo@gmail.com.com
Federico Suarez	610/11	elgeniofederico@gmail.com
Juan Carlos Giudici	827/06	elchudi@gmail.com
Franco Negri	893/13	franconegri2004@gmail.com

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

Contenidos

Capítulo 1

Plan de vuelo

1.0.1. Introducción

1.0.2. Ejemplos y Soluciones

1.0.3. Desarrollo

1.0.4. Complejidad

1.0.5. Experimentacion

Capítulo 2

Caballos salvajes

2.0.6. Introducción

Para este ejercicio, se nos pide encontrar un algoritmo, que, dados k caballos repartidos por un tablero de n por n casillas, encuentre cual es la casilla donde puedo reunir a todos los caballos en la menor cantidad de saltos.
Nuestra entrada será:

- Un entero **n** $\rightarrow$ Representarán el largo y el ancho del tablero.
- Un entero **k** $\rightarrow$ Representará el numero de caballos repartidos en el.
- **k** filas donde, para cada fila se tiene:
- $f\ c \rightarrow Representarnla fila y la columna de cada caballo.$

A esto nuestro algoritmo deve devolver:

- Un entero **f c** $\rightarrow$ Representará la fila y columna a donde deven converger los caballos.
- Un entero **m** $\rightarrow$ Representará el numero total de saltos que le costará a todos los caballos llegar hasta ahí.

2.0.7. Ejemplos y Soluciones

Se procede ahora a realizar un ejemplo para ilustrar el problema.
Supongamos que tenemos un tablero de 4 por 4 con un caballo en la posición 1,1 y otro en la posición 4,4. La entrada del problema luego sería:

- 4 2
- 1 1
- 4 4

Para este caso es posible encontrar una solución a mano, por ejemplo, es facil ver que en dos movimientos es posible hacer converger a amobos caballos.
En caso de querer asegurarnos de ello, podríamos hacer lo siguiente. Dibujamos en un papel dos matrices de n por n. En la primera matriz, vamos a poner cual es la cantidad minima de saltos que el primer caballo realiza para saltar a cada una de las casillas del tablero.
Para ello primero anotamos en la matriz con costo 0 la posicion donde se encuentra el primer caballo. Ahora, saltamos desde esta pocicion a todas las posibles pociciones validas del tablero. Todas estas tendran costo 1.
Ahora, desde todas las pociciones de costo 1 saltamos a todas las pociciones validas del tablero que podamos. Estas van a tener costo 2. Si seguimos realizando este procedimiento, demostraremos que

obtendremos la cantidad minima de saltos que el primer caballo realiza para saltar a cada una de las casillas del tablero, que era lo que buscabamos.
Realizamos lo mismo con el segundo caballo, marcamos la casilla donde se encuentra parado con costo 0 y empezamos a saltar a las casillas validas.
Ahora sumamos ambas matrices, y lo que obtenemos es una matriz con los costos minimos de que todos los caballos salten a cada una de las posiciones del tablero.
Buscando los minimos en esta matriz, obtenemos lo que queríamos! (me siento re paenza escribiendo estas cosas)

SE PODRIAN AGREGAR DIBUJUTOS CON LAS MATRICES PARA QUE QUEDE CLARO

Luego, algunas soluciones que el algoritmo podría devolver en este caso son:

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

2.0.8. Desarrollo

La idea general del algoritmo es sencilla, para cada caballo, confeccionamos una matriz con el costo minimo de saltar a cada uno de los casilleros de la matriz. Luego sumando estas k matrices, obtenemos el costo minimo de que cada caballo salte a cada uno de los casilleros.
Para asegurarnos de que en cada paso estamos tomando efectivamente la menor cantidad de saltos para que un caballo llegue a un casillero de la matriz, podemos pensar a la misma como un grafo, en el cual dos nodos estan conectados si y solo si un caballo puede saltar de uno a otro de manera valida.
Luego solo basta realizar un BFS para obtener el costo minimo de que un caballo llegue a esa casilla.
Cabe destacar, que por una cuestión de claridad, en la implementacion final, la idea de recorrer un grafo está implicita, la misma solo nos ayuda a ver que tanto la complegidad como la correctitud son las adecuadas en el problema dado.
En la implementacion real, simplemente creamos k matrices de enteros de n por n. Luego para cada caballo, tomamos todos los nodos de distancia j, buscamos todos los nodos validos de distancia $j+1$ y los seteamos. Realizamos esto hasta que no quedan nodos no seteados y allí pasamos de caballo.
Mas formalmente:

Algorithm 1 void FuncionPrincipal()

1: Generar k matrices de nxn todas seteadas en infinito
2: Creo dos colas: colaDeProfundidadJ,colaDeProfundidadJmasUno
3: Para cada caballo, tomo el nodo inicial y lo encolo en colaDeProfundidadJ
4: Creo un entero j igual a 0
5: Mientras colaDeProfundidadJ no este vacía.
6: Para todo nodo ∈ colaDeProfundidadJ
7: En la matriz correspondiente a este caballo, asigno j, como el valor del nodo
8: Busco los vecinos, si estan seteados en infinito los encolo en colaDeProfundidadJmasUno
9: Sumo 1 a j
10: Asigno los valores de colaDeProfundidadJ los valores de colaDeProfundidadJmasUno
11: Vacío colaDeProfundidadJmasUno
12: Sumo las k matrices
13: Busco el minimo
14: imprimo el minimo

2.0.9. Complejidad

2.0.10. Experimentacion

Capítulo 3

La comunidad del anillo

3.0.11. Introducción

3.0.12. Ejemplos y Soluciones

3.0.13. Desarrollo

3.0.14. Complejidad

3.0.15. Experimentacion

Capítulo 4

Apéndice

4.1. Medicion de los tiempos

Para este tp como trabajamos bajo el lenguaje de programacion C++, decidimos calcular los tiempos utilizando 'chrono' de la libreria standard de c++ (chrono.h) que nos permite calcular el tiempo al principio del algoritmo y al final, y devolver la resta en la unidad de tiempo que deseamos.

4.2. Código Fuente

4.2.1. Ej1.cpp

4.2.2. Ej2.cpp

4.2.3. Ej3.cpp