DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico II

subtitulo del trabajo

Algoritmos 3
Segundo Cuatrimestre de 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

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Índice

1. Ej1 **3**
1.1. Introducción . . . 3
1.2. Ejemplos y Soluciones . . . 3
1.3. Desarrollo . . . 4
1.4. Complejidad . . . 4
1.5. Experimentacion . . . 4

2. Ej2 **5**
2.1. Introducción . . . 5
2.2. Ejemplos y Soluciones . . . 5
2.3. Desarrollo . . . 6
2.4. Demostración De Correctitud . . . 7
2.5. Complejidad . . . 7
2.6. Experimentacion . . . 7

3. Ej3 **9**
3.1. Introducción . . . 9
3.2. Desarrollo . . . 9
3.2.1. Modelado . . . 9
3.2.2. Solucio'n, Correctitud y Complejidad . . . 9
3.2.3. Complejidad . . . 10
3.3. Ejemplos y Soluciones . . . 10
3.4. Experimentacion . . . 10

1. Plan de vuelo

1.1. Introducción

En este ejercicio se nos pide encontrar un algoritmo que encuentre una combinacion de vuelos entre la ciudad A y la ciudad B tal que encuentre la manera de llegar lo antes posible a destino.

La entrada del algoritmo será:

- Un string **A** $\rightarrow$ Representará la ciudad de partida.
- Un string **B** $\rightarrow$ Representará la ciudad de de llegada.
- Un entero **n** $\rightarrow$ Representará el numero total de vuelos entre todas las ciudades.
- **n** filas donde, para cada fila se tiene:
  - Un string **ori** $\rightarrow$ Representará la ciudad de partida.
  - Un string **des** $\rightarrow$ Representará la ciudad de de llegada.
  - Un entero **ini** $\rightarrow$ Representará el numero la hora de despegue de la ciudad ori
  - Un entero **fin** $\rightarrow$ Representará el numero la hora de arribo a la ciudad des

A esto nuestro algoritmo deve devolver:

- Un entero **fin** $\rightarrow$ Representará el horario de llegada a la ciudad B.
- Un entero **k** $\rightarrow$ la cantidad de vuelos del itinerario.
- k enteros **v_1, v_2 ..., v_k** $\rightarrow$ los vuelos tomados

1.2. Ejemplos y Soluciones

Se procede a generar una posible instancia del problema para ilustrar lo que se espera del algoritmo.

Supongamos que queremos ir de la ciudad de Buenos Aires a la isla de Saba (Dato curioso: en la isla de Saba se encuentra el aeropuerto mas chico del mundo (400 m)).

Lamentablemente no existen vuelos directos, por lo que tendrémos que hacer escala para poder llegar ahí. A continuaíón se muestran los posibles vuelos que podríamos tomar para llegar a destino.

- Buenos Aires - Seúl: 10 - 20
- Buenos Aires - La isla de los pitufos: 17 - 24
- Seúl - San Fransisco: 22 - 40
- La isla de los pitufos - isla de Saba: 28 - 30
- San Fransisco - isla de Saba: 40 - 43
- Buenos Aires - San Fransisco: 13 - 20

Luego, podemos observar que existen muchas maneras factibles de llegar desde Buenos Aires al objetivo. Una manera es tomar la ruta de Buenos aires - Seul, de allí ir a San Fransisco y finalmente a la isla de Saba, pero rapidamente vemos que si bien este viaje es factible, es al menos tan bueno como tomar el vuelo de Buenos aires a san fransisco y de allí continuar a Saba. De esta observación se desprende un dato interesante que utilizaremos en nuestro algoritmo, si se llegar de la manera mas barata a un grupo de ciudades v_1, v_2, ... v_s y tengo w_1, w_2,...w_i ciudades a las que puedo llegar desde el primer grupo. Si ahora tomo la ciudad w_g tal que la hora de arribo a esa ciudad desde una en v es la menor posible, sé que no es posible llegar de una manera mas barata a esa ciudad tambien.

Rapidamente esta idea nos remite al algoritmo de dijstra, el cual comparte una gran similitud en la idea de ir explorando todos los caminos más cortos que parten del vértice origen y que llevan a todos los demás vértices hasta llegar al destino, por lo que mas adelante intentaremos valernos de esta idea.

Luego de esta gran revelación algoritmica, continuamos con el problema que nos atañe. Es facil resolver a mano esta instancia del problema y descubrir que la combinación de vuelos optima es:

- Buenos Aires - La isla de los pitufos: 17 - 24
- La isla de los pitufos - isla de Saba: 28 - 30

Llegando a destino a la hora 30.

1.3. Desarrollo

Como ya hemos adelantado en el apartado anterior, la idea subyacente en este problema es, utilizando un algoritmo de dijstra levemente modificado, en cada paso, calcular la manera mas barata de llegar desde un conjuto de ciudades ya visitadas, a uno quue todavía no hemos visitado.

==COMPLETAR==

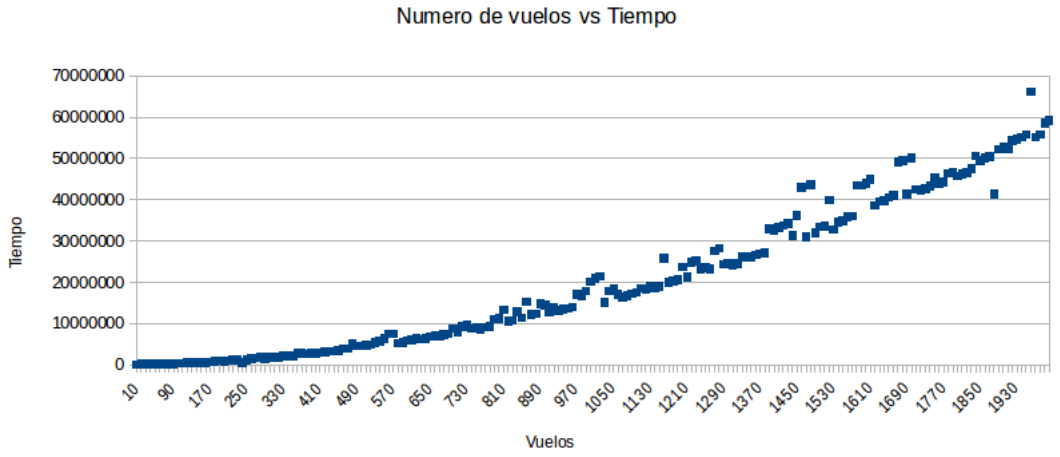
1.4. Complejidad

Dado que el algoritmo consiste en iterar por dos loops anidados, cuyo tamaño es n (osea, la cantidad de vuelos), la compelgidad del algoritmo será $O(n^2)$.

Ademas, este algoritmo para cada paso del loop mas externo, itera forzosamente por todo el arreglo para determinar el minimo, por lo que sabemos que, para el caso en que existe una solución $\omega(n^2)$.

1.5. Experimentacion

Dado que ya dijimos que no existen, 'peores casos', porque nuestro algoritmo esta acotado por ambos lados por n^2, realizamos un testeo random para comprobarlo:



Aqui puede verse claramente que nuestras hipotesis eran correctas.

2. Caballos salvajes

2.1. 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 $\rightarrow$ Representarán la 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.

2.2. 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

2.3. 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:

1: Generar k matrices de nxn todas seteadas en infinito
2: Creo dos colas: colaDeProfundidadJ,colaDeProfundidadJmasUno
3: Para cada caballo, tomo la casilla donde se encuentra y lo encolo en colaDeProfundidadJ
4: Creo un entero j igual a 0
5: Mientras colaDeProfundidadJ no este vacía.
6: Para toda casilla $\in$ 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: Encolo los valores de colaDeProfundidadJmasUno en colaDeProfundidadJ
11: Vacío colaDeProfundidadJmasUno
12: Sumo las k matrices
13: Busco el minimo
14: imprimo el minimo

Algorithm 1: void FuncionPrincipal()

2.4. Demostración De Correctitud

Para demostrar la correctitud de este algoritmo, primero, demostramos que dada una matriz de nxn, $n \geq 4$ desde cualquier casillero existe una sucesion de saltos para llegar a cualquier otro.

Por induccion en n, siendo n la cantidad de filas y de columnas.

Caso base:

DIBUJITO

Caso inductivo: Luego, si existe el tablero de nxn tal que de cualquier casillero podemos ir a cualquier casillero, quiero demostrar el tablero de $n+1$x$n+1$ tambien cumple.

Tomamos el casillero hubicado en la fila $n+1$ columna 1 (que notamos $(n+1,1)$), y vemos que existe un salto al casillero ($n-1$,2), luego, existe una sucecion de saltos desde el casillero $(n+1,1)$ a cualquiera del tablero de nxn.

Para todos los casilleros en ($n+1$,i) con $i > 1$, podemos saltar al casillero ($n-1$,$i-1$), y luego desde todos estos casilleros, tabmien podemos saltar dentro del tablero de nxn.

Ahora podemos ver que para la casilla (1,$n+1$), tambien existe el salto a (2,$n-1$), que hace que tambien cumpla.

De la misma manera, para los casilleros para (i,$n+1$) con $1 < i \geq n+1$ existe el salto a la casilla ($i-1$,$n-1$).

Con lo cual concluimos que existe un salto desde cualquier casillero de la fila $n+1$ o columna $n+1$ podemos saltar a cualquier casillero del tablero de nxn, por lo tanto, por hipotesis inductiva, existe una sucecion de saltos para llegar a cualquier casillero del tablero, en particular, puedo ir desde cualquier casillero de la fila/columna $n+1$x$n+1$ a cualquier otro que pertenezca a la misma fila/columna.

A partir de un tablero de nxn el modelado cada casillero representa un nodo, y dos nodos son adyacentes si y solo si existe un salto de caballo en el tablero.

Por la demostracion anterior, sabemoos que desde cualquier casillero existe una sucecion de saltos para llegar a cualquier otro, traducido a nuestro modelo de grafo, quiere decir que este es conexo.

Como el grafo es conexo y finito, realizando un bfs para cada caballo, podemos obtener la cantidad minima de saltos desde la pocicion en la que se encuentra el caballo hasta todas las otras del tablero.

Por lo tanto, sumando cuantos saltos debe realizar cada caballo a cada casillero, obtengo el minimo de saltos que tienen que dar todos los caballos para llegar a cualquier casillero.

Para cada casillero, sumanos los saltos de cada caballo y nos quedamos con el que tenga suma minima.

2.5. Complejidad

Para cada caballo, encolamos una sola vez cada nodo y posteriormente lo procesamos.

Entonces la complegidad es la cantidad de caballos (k) por la cantidad de nodos (n^2).

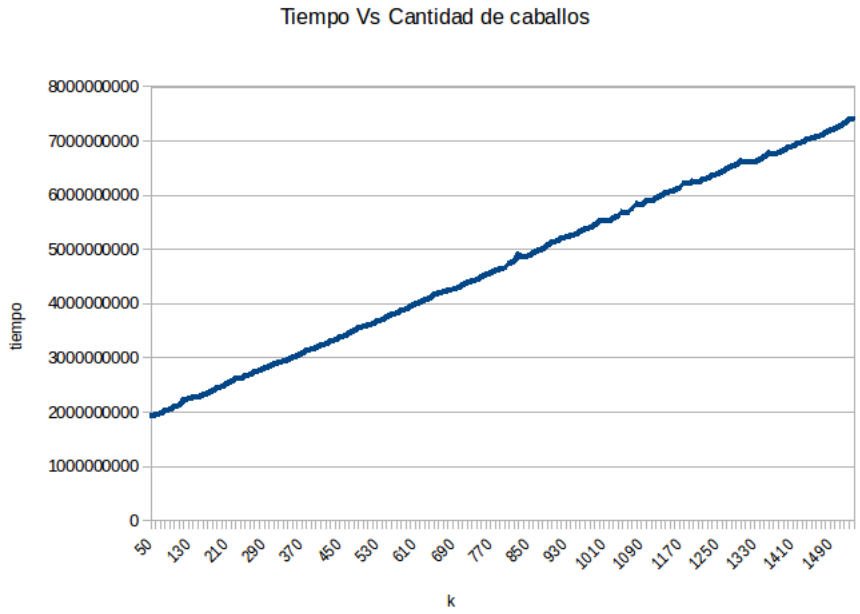
En otras palabras la complegidad es $O(kn^2)$.

2.6. Experimentacion

Para testear la performance de nuestro algoritmo se creó un generador de entradas que fabricará instancias random del problema.

Para la primera experimentación se deja el tablero fijo, y se varía el numero de caballos. Para este testeo, se tomó $n = 100$ y se hizo variar el numero de caballos entre 50 y 1500, cada uno posicionado de manera aleatoria en una casilla, y se tomaron 50 muestras de cada uno.

Los resultados obtenidos se muestran en el siguiente grafico.



Puede verse claramente que el algoritmo se comporta de manera lineal con respecto a la cantidad de caballos

3. La comunidad del anillo

3.1. Introducción

En este ejercicio se propone solucionar el problema de dado una red de existente de computadoras, con a lo sumo un enlace entre cada par de ellas y con un costo asociado al mismo, seleccionar algunas de estas para formar un backbone con topología de red tipo anillo, la cual tiene que tener como característica que conecte a todas las computadoras originales y que minimice el costo de ancho de banda de la red.

Se pide un algoritmo que genere este backbone, con un costo temporal estrictamente menor que O(n^3), este algoritmo debe detectar casos en los que no hay solucion.

3.2. Desarrollo

3.2.1. Modelado

Dada una red, la misma se puede modelar con un grafo de la siguiente manera:

1. Cada computadora se representa con un nodo.
2. Los enlaces entre cada par de computadoras se son los ejes en mi grafo, con el costo de ancho de banda como peso del mismo.

Transformamos el problema de ser uno de redes, a uno de grafos, donde encontrar un backbone con topologia de red tipo anillo que tenga costo mínimo y conecte a todas las computadoras, se transforma en encontrar subgrafo conexo cuyo costo sea mínimo y que contenga un único circuito simple, el cual se corresponde al backbone.

3.2.2. Solucio'n, Correctitud y Complejidad

Suponiendo que el grafo resultante es conexo, ya que si no lo fuese no habria soluci'on, vamos a utilizar el algoritmo de Prim, el cual dado un grafo conexo con pesos asociados a sus ejes construye un Árbol Generador Mínimo, es decir un subgrafo conexo cuya la suma de los pesos de sus ejes es mínima. Para completar el circuito, seleccionamos el eje con costo mínimo de los no elegidos por el algoritmo de Prim, el cual nos va a generar un único circuito simple.

Este último paso se justifica por lo visto en las clases teóricas donde demostramos que dado un Árbol, si agregamos un eje entre cualquier par de nodos, se forma un único circuito simple.

El algoritmo es el siguiente:

```
1: si no Es_conexo_o_no_tiene_ejes_suficientes_para_construir_un_circuito(G)
2:     devolver no
3: agm, ejes_no_seleccionados ← Prim(G)
4: eje_minimo ← Encontrar_Eje_Minimo(ejes_no_seleccionados)
5: circuito ← Construir_Circuito(agm, eje_minimo)
6: Mostrar circuito
```

Algorithm 2: EncontrarBackBone(G(E,V))

- Nuestra implementación del algoritmo de Prim, ademas del árbol generador mínimo genera una lista de ejes no seleccionados, la cual vamos a usar para encontrar el eje mínimo y completar el circuito.
- `Es_conexo_o_no_tiene_ejes_suficientes_para_construir_un_circuito(G)` recorre el grafo mediante DFS, llevando la cuenta de los nodos visitados para decidir si es conexo una vez finalizado y para decidir si es posible armar un circuito verifica que `m >= n`.
- `Encontrar_Eje_Minimo(ejes_no_seleccionados)` busca linealmente en la cantidad de ejes (a lo sumo $m = n^2$) el eje con costo mínimo.

- `Construir_Circuito(agm, eje_minimo)` Toma como punto principio y final los nodos del eje_minimo y mediante DFS construye el circuito.

3.2.3. Complejidad

La complejidad del algoritmo `EncontrarBackBone` es la siguiente:

- `Es_conexo_o_no_tiene_ejese_suficientes_para_construir_un_circuito(G)` tiene un costo de O(n^2), esta implementado mediante una variacion del algoritmo de DFS, el cual va marcando los nodos visitados.
- La implementación de `Prim(G)` que utilizamos tiene un costo O(n^2), dado que utilizamos una matriz de adyacencias.
- `Encontrar_Eje_Minimo(ejes_no_seleccionados)` tiene un costo de O(n^2)
- `Construir_Circuito(agm, eje_minimo)` tiene un costo de O(n), dado que el agm tiene $n - 1$ aristas.

Por lo tanto, el algoritmo tiene un orden temporal de O(n^2), cumpliendo con lo pedido en el enunciado.

3.3. Ejemplos y Soluciones

Aca dibujito de grafo

3.4. Experimentacion

Experimentar y tirar graficos