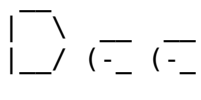
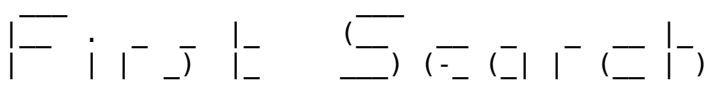
"Análisis de Algoritmos"
José Helo Guzmán





Contenido

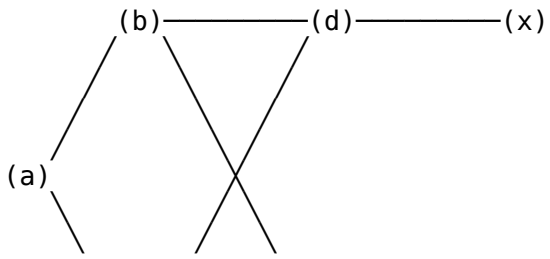
▬▬▬▬ 1. Introducción.
▬▬▬▬ 2. Definición de un grafo.
▬▬▬▬ 3. Deep First Search (DFS).
▬▬▬▬ 4. DFS para Encontrar una Ruta.
▬▬▬▬ 5. Árbol de Soluciones Para Una Ruta.
▬▬▬▬ 6. DFS para Encontrar Todas las Rutas.
▬▬▬▬ 7. Árbol de Soluciones Para Todas las Ruta.
▬▬▬▬ 8. Análisis de Complejidad.
▬▬▬▬ 9. Análisis de Complejidad para Encontrar una Ruta.
▬▬▬▬ 10. Análisis de Complejidad para Encontrar Todas las Ruta.
▬▬▬▬ 11. Resumen.
▬▬▬▬ 12. Ejercicios.

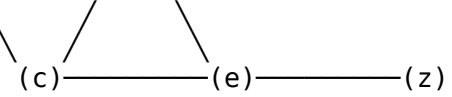
▬▬▬▬ 1. Introducción.

En muchos problemas se necesita representar un conjunto de estados y posibles formas de realizar una transición de un estado a otro. Una estructura muy usada para representar estos procesos son los grafos.

▬▬▬▬ 2. Definición de un grafo.

Supongamos que tenemos el siguiente grafo no dirigido:





Un grafo se define como un conjunto de nodos “V” (“vertex”) y un conjunto de arcos “E” (“Edges”). Por ejemplo:

```
V = [a, b, c, d, e, x, z]


E = [ [a,b], [a,c],
      [b,a], [b,d], [b,e],
      [c,a], [c,d], [c,e],
      [d,b], [d,c], [d,x],
      [e,b], [e,c], [e,x],
      [x,d],
      [z,e]
    ]
```

Otra forma más sencilla de representar el grafo consiste en enumerar el conjunto de nodos y para cada nodo, indicar con quien se conecta:

```
V = [a, b, c, d, e, x, z]

E = [ [a, [b,c]],
      [b, [a,d,e]],
      [c, [a,d,e]],
      [d, [b,c,x]],
      [e, [b,c,z]],
      [x, [d]],
      [z, [e]]
    ];
```

▬▬▬ 3. Deep First Search (DFS).

Dado el grafo anterior, estamos interesados en encontrar una ruta para ir desde el nodo “a” hasta el nodo “z”.

Para ello vamos a realizar el siguiente proceso, construyendo un conjunto de rutas.

(1) Tomamos el nodo de inicio.

(2) Vemos a qué nodos se puede transitar, los llamaremos vecinos.

(3) De la lista de posibles vecinos, se extiende la ruta, de manera que no se produzca un “loop”.

(4) Se repiten los pasos 2 y 3 hasta llegar al nodo final deseado.

▬▬▬▬ 4. DFS para Encontrar una Ruta.

Iniciamos con una lista que tiene el nodo “a”:

```
[ [a] ]
```

Este nodo tiene los siguientes arcos: [a, [b,c]]
Creamos una nueva ruta con cada arco.

```
[ [a,b],
  [a,c]
]
```

Volvemos a extender la primera ruta con los arcos: [b, [a,d,e]],

```
[ [a,b,d],
  [a,b,e],
  [a,c]
]
```

Extendemos la primera ruta con: [d, [b,c,x]],

```
[ [a,b,d,c],
  [a,b,d,x],
  [a,b,e],
  [a,c]
]
```

Extendemos la primera ruta con: [c, [a,d,e]],

```
[ [a,b,d,c,e],
  [a,b,d,x],
  [a,b,e],
  [a,c]
]
```

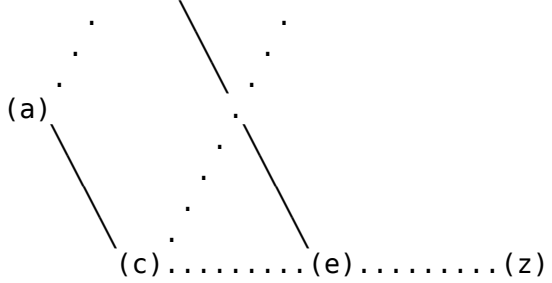
Extendemos la primera ruta con: [e, [b,c,z]],

```
[ [a,b,d,c,e,z],
  [a,b,d,x],
  [a,b,e],
  [a,c]
]
```

En este momento la primera ruta tiene un recorrido completo desde “a” hasta “z”.

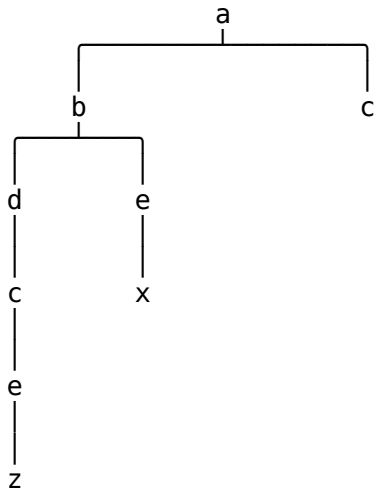
```
[a,b,d,c,e,z]
```

Aquí vemos la ruta que encontramos:



▬▬▬ 5. Árbol de Soluciones Para Una Ruta.

Veamos cómo se construyó esta solución mediante un árbol:



Podemos ver cómo para la construcción de la ruta siempre se desarolla el nodo que se encuentra más hacia abajo y a la izquierda del árbol. Este proceso de construcción se conoce como: “Deep First Search” o Profundidad Primero.

▬▬▬ 6. DFS para Encontrar Todas las Rutas.

Si continuamos el proceso de solución en deep first search en el árbol, podremos encontrar todas las rutas. Veamos este procesos

Asi encontramos la primera solución:

[[a]]

[[a,b],[a,c]]

[[a,b,d], [a,b,e], [a,c]]

[[a,b,d,c], [a,b,d,x], [a,b,e], [a,c]]

[[a,b,d,c,e], [a,b,d,x], [a,b,e], [a,c]]

[[a,b,d,c,e,z], [a,b,d,x], [a,b,e], [a,c]]

Guardamos esta solución y seguimos el proceso de búsqueda:

[[a,b,d,x], [a,b,e], [a,c]]

[[a,b,e], [a,c]]

[[a,b,e,c], [a,b,e,z], [a,c]]

[[a,b,e,c,d], [a,b,e,z], [a,c]]

[[a,b,e,c,d,x], [a,b,e,z], [a,c]]

[[a,b,e,z], [a,c]]

Quitamos esta solución y seguimos buscando:

[[a,c]]

[[a,c,d], [a,c,e]]

[[a,c,d,b], [a,c,d,x], [a,c,e]]

[[a,c,d,b,e], [a,c,d,x], [a,c,e]]

[[a,c,d,b,e,z], [a,c,d,x], [a,c,e]]

Quitamos esta solución y seguimos buscando:

[[a,c,d,x], [a,c,e]]

[[a,c,e]]

[[a,c,e,b], [a,c,e,z]]

[[a,c,e,b,d], [a,c,e,z]]

[[a,c,e,b,d,x], [a,c,e,z]]

[[a,c,e,z]]

Quitamos esta solución

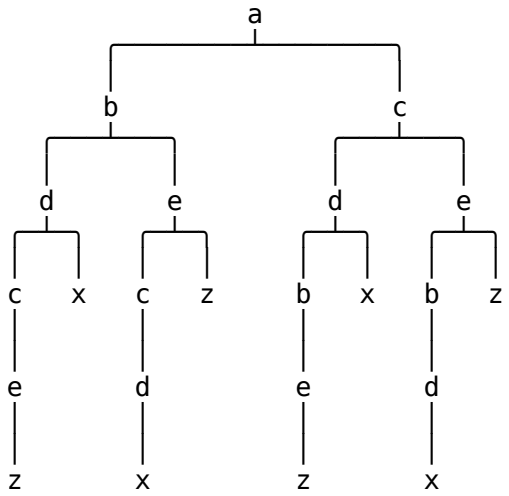
[ ]

Al obtener una lista vacía, se pueden presentar todas las soluciones que se encontraron:

```
 [ [a,b,d,c,e,z],
   [a,b,e,z],
   [a,c,d,b,e,z],
   [a,c,e,z]
]
```

▬▬▬ 7. Árbol de Soluciones Para Todas las Ruta.

Podemos ver cómo se ha construido el árbol de soluciones, siempre en profundidad primero.



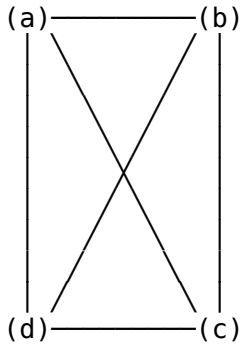
▬▬▬ 8. Análisis de Complejidad.

El pero caso se presenta cuando tenemos un grafo completamente conectado, es decir, un grafo donde cada nodo tiene un arco con todos los demás nodos. Esta suposición permite que la prueba se vuelva mucho más sencilla.

Por lo tanto, se tiene un grafo con:
n   nodos.
n-1   vértices que salen de cada nodo
n*(n-1) total de vértices.

Dado el siguiente grafo:

4   nodos
4-1 = 3 vértices que salen de cada nodo
4*3 = 12 vértices

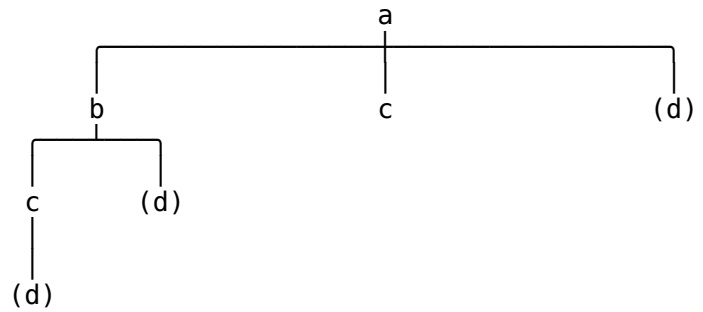


Se puede representar mediante la siguiente lista de nodos adyacentes:

```
grafo:
[ [a, [b,c,d]],
  [b, [a,c,d]],
  [c, [a,b,d]],
  [d, [a,b,c]]
];
```

▬▬▬ 9. Análisis de Complejidad para Encontrar una Ruta.

Al buscar una ruta, se genera el siguiente árbol:



Si se tienen 4 nodos.
Del primer nivel se generan 3 rutas.
Del segundo nivel se generan 2 rutas.
Del tercer nivel se genera 1 ruta.

En general si se tienen “n” nodos.
Del primer nivel se generan (n-1) ramas.
Del segundo nivel se generan (n-2) ramas.
Del tercer nivel se generan (n-3) ramas.
...
Del último nivel se genera   1   rama.

Por lo tanto la complejidad estará dada por:

$$n + (n-1) + (n-2) + \ldots + 2 + 1 = \left(\frac{n*(n+1)}{2} \right)$$

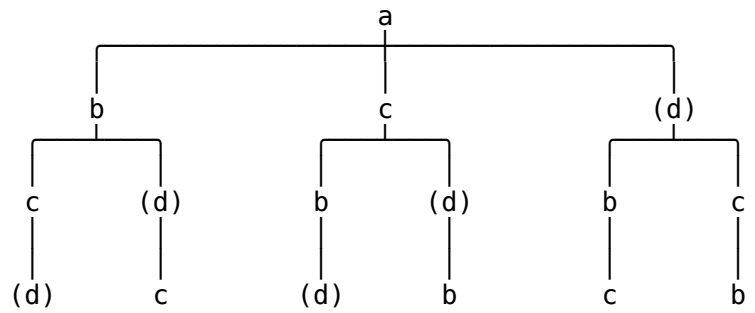
$$1 + 2 + \ldots + (n-2) + (n-1) + n = \left(\frac{n*(n+1)}{2} \right)$$

Que sabemos es equivalente a:

$O(n^2)$

▬▬▬ 10. Análisis de Complejidad para Encontrar Todas las Ruta.

Cuando se genera el árbol de búsqueda para encontrar todas las rutas, tenemos el siguiente diagrama:



Observe que el árbol con 4 nodos tiene (4-1)! = 6 rutas.

En general para “n” nodos, se tendrán (n-1)! rutas.

Por lo que el algoritmo es O(n!)

Si desean plantear la relación de recurrencia estará dada por:

T(0) = 1

T(n) = n * T(n-1) + O(n)

▬▬▬ 11. Resumen.

• Deep First Search nos indican si hay un camino en un grafo.

• Deep First Search se puede utilizar para encontrar todos los posibles caminos de un problema.

• Es relativamente fácil convertir un algoritmo en otro para estos problemas de búsqueda.

▬▬▬ 12. Ejercicios.

●● Ejercicio 1.

Bajo que condiciones utilizaría usted en un problema profundidad primero?
Bajo que condiciones utilizaría usted en un problema anchura primero?
En general, cuál de estos dos procedimientos cree que es mejor?

●● Ejercicio 2.

Reescriba el procedimiento que busca todas las soluciones en profundidad primero de manera tal que ahora en lugar de utilizar una estructura de control basada en recursión lo haga con una estructura de control basada en repetición.

●● Ejercicio 3.

Póngale un valor a cada uno de los arcos de grafo. Una vez que cuente con todas las rutas posibles para ir de i hacia f escriba un procedimiento que toma el valor de cada arco y encuentra la ruta cuya suma es la de menor valor.

Para realizar este procedimiento puede hacerlo de dos formas.

-La primera modifique la estructura del grafo para incluir el valor de ir a cada uno de los nodos vecinos. Para ello también tiene que modificar el procedimiento de vecinos y extender.

-La segunda utilice una segunda estructura donde se guarden los datos de los valores. En este caso se toman las soluciones y se operan con esta segunda estructura.

●● Ejercicio 4.

Un gran número de problemas se pueden resolver mediante la técnica anterior de extender soluciones hasta encontrar una solución final. A continuación se describe un problema clásico conocido como el problema de las tres monedas.

Se tienen monedas que pueden colocarse en corona (C) o escudo (E). El problema de las tres monedas consiste en lo siguiente:
Se tienen tres monedas que marcan (C C E)
Se busca llegar a uno de los dos siguientes estados (C C C) o (E E E)
Construya un programa que utilice un proceso de búsqueda para localizar TODAS las formas en que estas tres monedas pueden llegar al estado final.

●● Ejercicio 5.

Se tienen dos contenedores de agua a los que llamaremos A y B. Al recipiente A le caben 5 litros de agua y al recipiente B 3 litros de agua. El objetivo final del juego es dejar una cantidad 2 litros de agua en el recipiente A.

Para representar este problema se utilizará una lista de la forma (A B). Inicialmente los dos recipientes se encuentran vacíos por los que el estado inicial es (0 0). Se busca entonces iniciar en (0 0) y terminar en (2 x) con x cualquier valor.

Las únicas operaciones válidas entre estos contenedores son:
1. Llenar el recipiente A.
2. Llenar el recipiente B.
3. Vaciar el recipiente A.
4. Vaciar el recipiente B.
5. Mover el contenido de A hacia B.
6. Mover el contenido de B hacia A.

Construya una función de nombre rep(A, B, C), que en nuestro caso se invocará como:

rep(5, 3, 2)

La cual debe producir una solución para este problema. Por ejemplo:

```
> rep( 5, 3, 2)
[[0,0], [5,0], [2,3]]
```

Construya una función de nombre trep(A, B, C) que se invocará como trep(5, 3, 2) la cual debe producir TODAS las posibles soluciones para este problema.

●● Ejercicio 6.

Otro ejercicio típico es el problema de las “n” reinas. Se define un tablero de largo y anchura “n”. El problema consiste en colocar “n” reinas en dicho tablero de manera tal que ninguna de ellas ataque a las otras.

Para representar la posición de una reina dentro del tablero se utilizara un lista de dos elementos de la forma (i j), donde i representará la fila y j la columna donde se encuentra la reina.

Por ejemplo, suponga que se trata de un tablero 4x4. Las únicas maneras de colocar 4 reinas en dicho tablero sin que ninguna ataque a las otras es:

```
> reinas(4)
[ [[1,3], [2,1], [3,4], [4,2]]
  [[1,2], [2,4], [3,1], [4,3]]
]
```

●● Ejercicio 7.

En un tablero de ajedrez de tamaño NxN se coloca un caballo de ajedrez en la posición (i j). Se desea que el caballo recorra todo el tablero sin caer dos veces en una misma casilla.

Construya un programa en que indique los movimientos que debe realizar el caballo partiendo de (i j) para recorrer todo el tablero. En esta caso debe construir únicamente una solución, no todas las soluciones posibles.

●● Ejercicio 8.

Utilizando los algoritmos descritos en esta sección es posible construir todas las formas en que se puede jugar una partida de ajedrez??
Se puede realizar ese programa??
En caso afirmativo que impide realizar dicho programa??

●● Ejercicio 9.

Un puzzle (¿¿acertijo??) es un juego infantil que consta de una superficie cuadrada que ha sido subdividida en cuadrados más pequeños. Finalmente uno de los cuadrados se ha removido para movimientos entre las piezas.

Utilice un método de profundidad primero o de anchura primero para determinar todas las posibles formas de ordenar un puzzle.

La representación que se utilizará consistirá en una matriz de tamaño NxN. La solución final consistirá en tener el puzzle ordenado.

En el caso de un puzzle de 3x3 este sería el estado final:

```
[ [1, 2, 3]
  [4, 5, 6]
  [7, 8, 0]
]
```

Dos posibles estados iniciales para el problema serían:

```
[ [6, 7, 5]
  [4, 0, 2]
  [8, 1, 3]
]
```

```
[ [6, 0, 5]
  [4, 7, 2]
  [8, 1, 3]
]
```

Construya un programa que encuentre una solución para este problema.