Diseño de Algoritmos

Unidad 4: Exploración en grafos

Versión 2.2

1. Explique el concepto de orden topológico y el algoritmo de ordenación topológica por búsqueda en profundidad.

2. Dado un grafo $G = (V, A)$ orientado y acíclico, un orden topológico *inverso* es un orden lineal $<$ definido sobre V tal que si $i < j$ entonces $(i, j) \notin A$.

a) Modifique el algoritmo de ordenación topológica por búsqueda en profundidad para que produzca un orden topológico inverso.

b) Diseñe un algoritmo que produzca un orden topológico inverso preprocesando la entrada de un algoritmo de ordenación topológica convencional.

c) Diseñe un algoritmo que produzca un orden topológico inverso postprocesando la salida de un algoritmo de ordenación topológica convencional.

3. Diseñe un algoritmo que resuelva un laberinto tridimensional mediante búsqueda con retroceso. El laberinto está representado por un cubo en el que, originalmente, cada celda aparece libre (' ') o bloqueada ('X'). Se trata de encontrar, si es posible, una salida desde una posición inicial dada. A este efecto, se considera que una salida es cualquier celda libre del borde, es decir, de una de las caras del cubo. El algoritmo recibirá el laberinto y la posición inicial, que ha de ser válida y corresponder a una celda libre, y devolverá un booleano que indique si existe o no solución. Además debe marcar las celdas visitadas ('V') y las que forman parte del camino de salida ('S'), si este existe.

4. Diseñe un algoritmo de búsqueda con retroceso para colorear un grafo con un numero de colores dado. El algoritmo debe indicar si es posible encontrar una solución con dicho número de colores y, en tal caso, finalizar su ejecución proporcionando el color asignado a cada vértice.

5. Diseñe un algoritmo para calcular el *número cromático* de un grafo, es decir, el mínimo número de colores con el que se puede colorear.

6. La versión del algoritmo de búsqueda con retroceso que resuelve el problema de la mochila discreta mediante acotación superior, no comprueba explícitamente si el valor de la solución es superior al de la mejor solución obtenida hasta el momento. ¿Por qué no es necesario hacerlo? ¿Cómo se asegura el algoritmo de que el valor de la solución sea mejor?

7. A partir de la versión del algoritmo de búsqueda con retroceso que resuelve el problema de la mochila discreta mediante acotación superior, diseñe un algoritmo de ramificación y poda con expansión ordenada que emplee un montículo.

8. Dados n caracteres distintos y un número natural $m \leq n$, diseñe un algoritmo que genere todas las cadenas formadas por m caracteres distintos escogidos entre los dados.

9. Dado un tablero de ajedrez de $n \times n$ escaques y un caballo colocado en un escaque arbitrario, diseñe un algoritmo que determine, si existe, una secuencia de movimientos del caballo que visite todos los escaques del tablero exactamente una vez.

10. Repita el ejercicio anterior, esta vez exigiendo que el último movimiento devuelva el caballo a su posición inicial.

11. Se dispone de n máquinas y n trabajos a realizar. Cada máquina es capaz de realizar cualquiera de los trabajos en un tiempo que depende de la máquina y el trabajo en cuestión. Cada trabajo se asignará exactamente a una máquina, que se encargará exclusivamente de él. Diseñe un algoritmo que permita obtener una asignación óptima de trabajos a máquinas, es decir, que minimice el tiempo total necesario para completar todos los trabajos, con cada una de las siguientes técnicas:

a) Sin cotas.

b) Con cota inferior (optimista).

c) Con cota inferior (optimista) y superior (pesimista).

d) Con ambas cotas y, además, empleando montículos.

12. Repita el ejercicio anterior, pero con beneficios en vez de tiempos. En este caso, la asignación óptima será aquella que maximice el beneficio total obtenido.