SEPTIEMBRE 2018 – ORIGINAL (MODELO A)

1. Un dentista pretende dar servicio a *n* pacientes y conoce el tiempo requerido por cada uno de ellos, siendo t_i , $i= 1,2,\ldots,n$ el tiempo requerido por el paciente *i*. El objetivo es minimizar el tiempo medio de estancia de los pacientes en la consulta. En relación a este problema, ¿cuál de las siguientes afirmaciones es **falsa**?

   (a) El esquema más eficiente para resolver este problema correctamente es el esquema voraz.
   (b) El coste del algoritmo voraz que resuelve el problema es O(n log n).
   (c) Suponiendo tres pacientes con tiempos de servicio t_1=15, t_2=60 y t_3=30, el tiempo mínimo de estancia total posible es de tres horas.
   (d) Suponiendo tres pacientes con tiempos de servicio t_1=15, t_2=20 y t_3=30, el tiempo mínimo de estancia total posible es de 115 minutos.

Como se está hablando de objetos no fraccionables, el esquema o algoritmo para resolverlo es el Esquema Voraz, por lo que la respuesta A) es cierta.

Vamos a hace el cálculo de los tiempos de servicio para las respuestas C) y D), siendo el cálculo en dichos casos el siguiente:

C) 15 + (15 + 30) + (15 + 30 + 60) = 15 + 45 + 105 = 165 minutos

D) 15 + (15 + 20) + (15 + 20 + 30) = 15 + 35 + 65 = 115 minutos

Como ya se puede apreciar, la respuesta D) es cierta, por lo que vamos a pasar las 3 horas de la respuesta C) a minutos, para así saber si dicha respuesta pudiera ser o no la falsa.

1 hora → 60 minutos

3 horas → X minutos

X = 60 * 3 / 1 → X = 180 minutos

Obteniendo dicho resultado, ya se sabe con total seguridad que 180 minutos (3 horas) es distinto de los 165 minutos calculados, por lo que la respuesta B) era cierta, y la respuesta correcta buscada era:

Respuesta C)

2. Se dispone de un vector, V, que almacena n números naturales, y se desea averiguar si existe algún elemento que aparezca al menos n/2 + 1 veces en el vector. ¿Cuál sería la estrategia más adecuada para resolver el problema?

   (a) Algoritmo voraz.
   (b) Divide y vencerás.
   (c) Vuelta atrás
   (d) Ramificación y poda

A) Algoritmo Voraz

Se aplica a problemas de optimización en los que la solución se puede construir paso a paso sin necesidad de reconsiderar decisiones ya tomadas. Genéricamente el problema que se puede resolver con este tipo de esquema es: encontrar un conjunto de candidatos que constituya una solución y que optimice una función objetivo. Este esquema se utiliza principalmente en problemas de planificación de tareas y en problemas que se pueden modelar con grafos, en los que hay que realizar una búsqueda, cálculo de recorridos u optimización de pesos, entre otras tareas.

Los problemas que se pueden resolver con este esquema constan de n candidatos y se trata de encontrar una solución basada en hallar un subconjunto de esos candidatos, o una secuencia ordenada de los mismos, de manera que se optimice (maximice o minimice) una función objetivo. En este esquema se trabaja por etapas, considerando la elección de un candidato en cada etapa. Habrá que seleccionar en cada una el candidato más prometedor de los aún disponibles y decidir si se incluye o no en la solución.

- Algoritmos Voraces como procedimientos heurísticos

- Con Grafos: hallar un árbol de recubrimiento de distancia o coste mínimo y calcular el camino de coste mínimo entre un nodo y los demás.

- Árboles de recubrimiento mínimo "Algoritmo de Prim": selecciona arbitrariamente un nodo del grafo como raíz del árbol AR. En cada paso se añade al árbol una arista de coste mínimo (u, v) tal que AR U {(u, v)} sea también un árbol. El algoritmo continúa hasta que AR contiene n - 1 aristas. Nótese que la arista (u, v) que se selecciona en cada paso es tal que o bien u o v está en AR.

- Árboles de recubrimiento mínimo "Algoritmo de Kruskal": Partiendo del conjunto AR (conjunto de aristas del árbol de recubrimiento mínimo) vacío, en cada paso selecciona la arista más corta o con menos peso que todavía no haya sido añadida a AR o rechazada. Inicialmente, AR está vacío y cada nodo de G forma una componente conexa. En los pasos intermedios, el grafo parcial formado por los nodos de G y las aristas de AR consta de varias componentes conexas. Al final del algoritmo sólo queda una componente conexa, que es el árbol de recubrimiento mínimo de G. Cuando en cada paso se selecciona la arista más corta que todavía no ha sido evaluada, se determina si une dos nodos pertenecientes a dos componentes conexas distintas. En caso afirmativo la arista se añade a AR, por lo que las dos componentes conexas ahora forman una. En caso contrario, esa arista se rechaza ya que forma un ciclo al unir dos nodos de la misma componente conexa.

- Camino de coste mínimo: algoritmo de Dijkstra
- Minimización del tiempo en el sistema
- Planificación con plazos
- Almacenamiento óptimo en un soporte secuencial
- Generalización a n soportes secuenciales
- Problema de la mochila con objetos fraccionables
- Mantenimiento de la conectividad
- Problema de Mensajería urgente
- Problema del Robot desplazándose en un circuito
- Asistencia a incidencias

B) <u>Divide y Vencerás</u>

Técnica algorítmica que se basa en la descomposición de un problema en subproblemas de su mismo tipo, lo que permite disminuir la complejidad y en algunos casos, paralelizar la resolución de los mismos. La estrategia del algoritmo es la siguiente:

• Descomposición del problema en subproblemas de su mismo tipo o naturaleza.
• Resolución recursiva de los subproblemas.
• Combinación, si procede, de las soluciones de los subproblemas.
Un ejemplo sencillo de aplicación de la técnica de Divide y Vencerás es la búsqueda binaria en un vector ordenado. Otros casos de aplicación serían en:

- *Ordenación por fusión (Mergesort):* Dado un vector de enteros, lo que plantea es dividir el vector por la mitad e invocar al algoritmo para ordenar cada mitad por separado. Tras esta operación, sólo queda fusionar esos dos subvectores ordenados en uno sólo, también ordenado. La fusión (merge) se hace tomando, sucesivamente, el menor elemento de entre los que queden en cada uno de los dos subvectores.

- *El puzzle tromino:* Un tromino es una figura geométrica compuesta por 3 cuadros de tamaño 1 x 1 en forma de L. Sobre una retícula cuadrada de n x n (siendo n = 2k) se dispone un cuadrado marcado (en negro, por ejemplo) de tamaño 1 x 1 y el resto vacío. El problema consiste en rellenar el resto de la cuadrícula de trominos sin solaparlos y cubriéndola totalmente, exceptuando el cuadrado mencionado. Aunque en una primera aproximación puede pensarse que se trata de un problema resoluble únicamente mediante exploración ciega y la técnica de vuelta atrás, hay una manera de realizarlo mediante divide y vencerás si se cumple la condición de que n sea potencia de 2.

- *Ordenación rápida (Quicksort):* En la ordenación por fusión se pone el énfasis en cómo combinar los subproblemas resueltos, mientras que la operación de descomponer en subproblemas es trivial. Sin embargo, si al descomponer el vector, los elementos del primer subvector fueran todos menores o iguales que los del segundo subvector, no habría que realizar ninguna operación de combinación. En este caso, la descomposición es algo más compleja, pero la combinación es trivial. Esta es la idea que subyace al algoritmo Quicksort, realizando el siguiente proceso: 1) Toma un elemento cualquiera del vector denominado pivote; 2) Toma los valores del vector que son menores que el pivote y forma un subvector. Se procede análogamente con los valores mayores o iguales.; 3) Se invoca recursivamente al algoritmo para cada subvector.

- *Cálculo del elemento mayoritario en un vector:* Dado un vector v[l..n] de números naturales, se quiere averiguar si existe un elemento mayoritario, es decir que aparezca al menos n/2 + 1 veces en el vector.

- *Liga de Equipos:* Supongamos que n equipos (con $n = 2^k$) desean realizar una liga. Las condiciones en las que se celebra el torneo son las siguientes: cada equipo puede jugar un partido al día, y la liga debe celebrarse en n - 1 días. Damos por hecho que disponen de suficientes campos de juego. El problema plantea la realización de un calendario en el que por cada par de equipos se nos indique el día en que juega. La solución tendrá por tanto forma de matriz simétrica con M $[e_i, e_j] = d_{ij}$ indica que el equipo e_i juega el día= d_{ij} con el equipo e_j.

- *Skyline de una Ciudad:* Sobre una ciudad se alzan los edificios dibujando una línea de horizonte (conocida como skyline). Supongamos una ciudad representada por un conjunto de edificios C = {e_1, e_2, ... e_n} y cada edificio representado por un rectángulo

sobre un eje de coordenadas. El problema consiste en calcular la línea de horizonte de la ciudad, en forma de una secuencia de puntos sobre el plano.

C) Vuelta Atrás

El esquema de vuelta atrás realiza un recorrido en profundidad del grafo implícito de un problema, podando aquellas ramas para las que el algoritmo puede comprobar que no pueden alcanzar una solución al problema. Las soluciones se construyen de forma incremental, de forma que a un nodo del nivel k del árbol le corresponderá una parte de la solución construida en los k pasos dados en el grafo para llegar a dicho nodo, y que llamamos solución o secuencia k-prometedora. Si la solución parcial construida en un nodo no se puede extender o completar, decimos que se trata de un nodo de fallo.

Cuando se da esta situación, el algoritmo retrocede hasta un nodo del que quedan ramas pendientes de ser exploradas. En su forma más básica, la vuelta atrás es similar a un recorrido en profundidad dentro de un grafo dirigido. El grafo suele ser un árbol, o en todo caso un grafo sin ciclos. A medida que progresa el recorrido por el grafo implícito se van construyendo soluciones parciales.

Un recorrido tiene éxito si se llega a completar una solución. Por el contrario, no tiene éxito si en alguna etapa la solución parcial construida no se puede seguir completando. En este caso el recorrido vuelve atrás como en un recorrido en profundidad, eliminando los elementos que se hayan añadido en cada fase.

Cuando se llega a algún nodo que tiene algún vecino sin explorar prosigue el recorrido de búsqueda de una solución a partir de él. El objetivo del algoritmo puede ser encontrar una solución o encontrar todas las posibles soluciones al problema.

Algunos problemas donde podemos aplicar este algoritmo, son:
- Juego de las Reinas

- Coloreado de Grafos: asignar un color de un conjunto de m colores a cada vértice de un grafo conexo, de manera que no haya dos vértices adyacentes que tengan el mismo color.

- Ciclos Hamiltonianos: Nos planteamos encontrar todos los ciclos Hamiltonianos de un grafo. Es decir, todos aquellos caminos que pasan por cada vértice una sola vez y terminan en el vértice inicial.

- Subconjuntos de suma dada: Disponemos de un conjunto A de n números enteros sin repeticiones (tanto positivos como negativos) almacenados en una lista. Dados dos valores enteros m y C, siendo m < n se desea resolver el problema de encontrar todos los subconjuntos de A compuestos por exactamente m elementos y tal que la suma de los valores de esos m elementos sea C.

- Reparto equitativo de activos: En este problema ayudaremos a dos socios que forman una sociedad comercial a disolverla. Cada uno de los n activos de la sociedad que hay que repartir tiene un valor entero positivo. Los socios quieren repartir dichos activos a medias y, para ello, quieren conocer todas las posibles formas que tienen de dividir el

conjunto de activos en dos subconjuntos disjuntos, de forma que cada uno de ellos tenga el mismo valor entero.

- El Robot en busca del tornillo: Este problema es un ejemplo de aplicación del esquema a la búsqueda de caminos en laberintos. Un robot se mueve en un edificio en busca de un tornillo. Se trata de diseñar un algoritmo que le ayude a encontrar el tornillo y a salir después del edificio. El edificio debe representarse como una matriz de entrada a la función, cuyas casillas contienen uno de los siguientes tres valores: L para "paso libre", E para "paso estrecho" (no cabe el robot) y T para "tornillo". El robot sale de la casilla (1 ,1) y debe encontrar la casilla ocupada por el tornillo. En cada punto, el robot puede tomar la dirección Norte, Sur, Este u Oeste siempre que no sea un paso demasiado estrecho. El algoritmo debe devolver la secuencia de casillas que componen el camino de regreso desde la casilla ocupada por el tomillo hasta la casilla (1, 1). Supondremos que la distancia entre casillas adyacentes es siempre 1. Como no es necesario encontrar el camino más corto, sino encontrar un camino lo antes posible, una búsqueda en profundidad resulta más adecuada que una búsqueda en anchura. Si el edificio fuera infinito entonces una búsqueda en profundidad no sería adecuada porque no garantiza que se pueda encontrar una solución. Nosotros suponemos que el edificio es finito.

- Asignación de cursos en una escuela: En una nueva escuela se van a impartir n cursos. El equipo directivo les ha asignado n aulas y n profesores, que deben repartirse entre los cursos teniendo en cuenta que existen una serie de restricciones. No todas las aulas tienen capacidad para todos los cursos, pues estos cuentan con distinto número de alumnos. Por ello se dispone de una función booleana válida(aula, curso) que indica si el aula tiene suficiente capacidad para el curso. Además, los profesores no están especializados en los temas de todos los cursos, por lo que también se dispone de una función especialidad(prof,curso) que indica si el profesor está especializado en un determinado curso. Se pide encontrar un algoritmo que encuentre alguna forma de asignar a cada curso un aula y un profesor apropiados. Podemos representar la solución como un vector de registros, cada uno de los cuales indica la asignación de un aula y un profesor a un curso. La posición del registro en el vector indica el curso de que se trata.

D) <u>Ramificación y Poda</u>

Es un esquema para explorar un grafo dirigido implícito. En este caso lo que se busca es la solución óptima de un problema. En este esquema los nodos no se exploran siguiendo la secuencia en la que se han generado, como se hace en el esquema de vuelta atrás, sino que se utiliza la función que se quiere optimizar para establecer preferencias entre los nodos pendientes de explorar.

Ahora el recorrido se dirige por el nodo activo más prometedor, por lo que se requiere una cola de prioridad para la gestión de los nodos activos. Los montículos son particularmente adecuados para almacenar la cola de prioridad que utiliza este esquema para registrar los nodos que se han generado y aún no han sido explorados.

Una vez seleccionado un nodo se procede con una fase de ramificación, en la que se generan distintas extensiones de la solución parcial correspondiente a dicho nodo. A continuación, este esquema no sólo poda aquellas ramas que no pueden llegar a una

solución, sino también aquellas que no pueden mejorar el valor asignado a la solución por la función que se quiere optimizar.

Por ello, en cada nodo calcularemos una cota optimista del posible valor de las soluciones que pueden construirse a partir de él. Si la cota indica que estas soluciones serán con seguridad peores que una solución factible ya encontrada (o que una cota pesimista de las soluciones posibles), entonces no tiene sentido seguir explorando esa parte del grafo y se poda. Es decir, se realiza una poda por factibilidad y una poda por cota.
Algunos de los problemas donde podemos aplicar este algoritmo, son:

- Problema de la Mochila Entera: Se dispone sólo de una mochila que soporta un peso máximo P. El objetivo de este problema es hacer una selección de objetos de forma que se maximice el valor del contenido de la mochila, y cuya suma de pesos no sobrepase su capacidad. Los objetos son indivisibles, es decir, cada uno de ellos o se toma entero o se deja. Este problema no puede ser resuelto por un algoritmo voraz, como ocurre en el caso en que los objetos se pueden dividir. Como tenemos que maximizar el valor del contenido estamos ante un problema de optimización, para el que el esquema de ramificación y poda es adecuado. Podemos representar la solución al problema como un vector de booleanos que nos indique con un 1 (cierto) que el objeto de la posición i se ha incluido en la mochila, y con un O (falso) que no se ha incluido.

- Asignación de Tareas “Pastelería”: Una pastelería tiene empleados a n pasteleros, que, aunque son capaces de hacer cualquiera de los m tipos de pasteles distintos que ofrece la pastelería, tienen distinta destreza, y por tanto rendimiento, en la preparación de cada uno de ellos. Se desea asignar los próximos n pedidos, uno a cada pastelero, minimizando el coste total de la preparación de todos los pasteles. Para ello se conoce de antemano la tabla de costes C [1..n, 1..m] en la que el valor C_{ij} corresponde al coste de que el pastelero i realice el pastel j, y los tipos de pasteles correspondientes a los pedidos, pedidos[1..n].

- El Viajante de Comercio: un viajante de comercio, que partiendo de la ciudad origen, tiene que visitar todas las ciudades de su zona una y sólo una vez y volver a la ciudad de origen, minimizando el coste del recorrido. Estamos ante un problema de optimización para el que no podemos encontrar una solución voraz, y aplicamos por tanto el esquema de ramificación y poda. Podemos representar la solución como un vector en el que el contenido de la posición i indica la ciudad por la que se pasa en el orden i del recorrido. El grafo lo representamos por la matriz de adyacencia que en la posición (i,j) indica el valor del enlace entre el vértice i y el j. Cuando no existe el enlace asignamos a la correspondiente posición de la matriz un valor especial que representamos por infinito.

- Selección de Tareas “Cursos de Formación”: Un profesional autónomo, que ofrece distintos cursos de formación para empresas y se compromete a haberlos impartido antes de una fecha límite fijada por la empresa, ha recibido n solicitudes de empresas. El autónomo conoce, para cada uno de los cursos solicitados el beneficio b_i que obtendría por impartirlo. El tiempo que se tarda en impartir cada curso es también variable y viene dado por t_i. También sabe los días d_i que quedan antes de la fecha tope fijada por la empresa que ha solicitado cada curso. Se pide un algoritmo que el

autónomo pueda utilizar para decidir qué cursos debe escoger para maximizar el beneficio total obtenido. La fecha límite que fija la empresa para cada curso viene dada por el número de días desde una fecha de referencia, de forma que se trata de una lista d_1, …, d_n. Se trata de un problema de optimización en el que hay dos variables, el beneficio de cada curso y el tiempo requerido para impartirlo. Este es otro de los tipos de problemas que se resuelven con un esquema de ramificación y poda. Podemos representar la solución mediante un vector cursos, donde si la posición i tiene el valor cierto es porque el curso correspondiente a la solicitud i se impartirá. También utilizamos un montículo en el que cada nodo además de almacenar la solución parcial, etapa y cota, almacena el tiempo y beneficio acumulado.

- Distancia de edición

Respuesta B)

3. En relación a los montículos, ¿cuál de las siguientes afirmaciones es **cierta**?

   (a) La ordenación mediante el algoritmo Heapsort tiene un coste O(log n).
   (b) El vector m=[6,5,4,4,1,3,2] es un montículo de mínimos.
   (c) La función insertar tiene un coste O(n log n)
   (d) Con un montículo de mínimos disponemos de una estructura de datos en la que encontrar el mínimo es una operación de coste constante.

Los Montículos son un tipo especial de árbol binario que se implementan sobre vectores con las siguientes propiedades:

- Es un árbol balanceado y completo (los nodos internos tienen siempre dos hijos) con la posible excepción de un único nodo cuando el número de elementos es par.

- Cada nodo contiene un valor mayor o igual que el de sus nodos hijos (montículo de máximos), o menor o igual (montículo de mínimos).

A la segunda de estas propiedades se la denomina “Propiedad de Montículo”, con la cual se permite tener en la cima (el nodo raíz) del montículo el elemento mayor o menor, si se trata de un montículo de máximos o de mínimos, respectivamente, siendo esta la utilidad fundamental del montículo.

Los nodos de profundidad k están situados en las posiciones 2^k y siguientes del vector hasta la posición $2^{k+1} - 1$.

El nodo padre del elemento en la posición “i” estará en “i/2 (div., entera)”. Los nodos hijos al elemento en la posición “i” estarán en “2*i, 2*(i+1)”.

La inserción y borrado de elementos es muy eficiente, teniendo un coste de O(log n). Las operaciones que podemos realizar con los montículos, son las siguientes:

- Flotar
  Intercambiar por el nodo inmediatamente a un nivel superior.

- Hundir
  Intercambiar por el nodo hijo de menor valor (montículo de mínimos).

- Insertar
  Añadir el nodo al final y luego aplicar la operación flotar.

- Primero
  Lleva asociado un coste constante O(1).

- Obtener Cima y Borrarla
  Colocar el último elemento en la cima y hundir.

Montículo de Mínimos

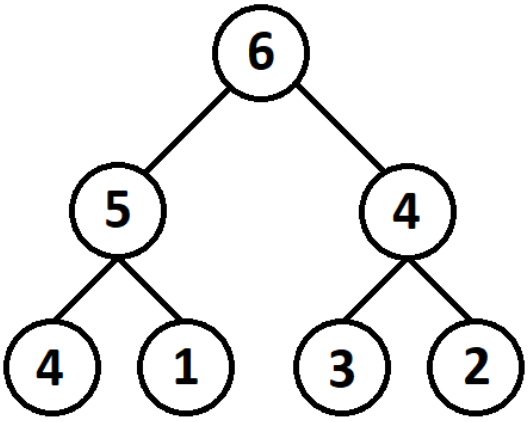
Cada nodo contiene un valor menor o igual que el de sus nodos hijos, disponiendo con él de una estructura de datos en la que, encontrar el valor mínimo, es una operación de coste constante.

Montículo de Máximos

Cada nodo contiene un valor mayor o igual que el de sus nodos hijos.

Nota: El montículo sirve de apoyo a la creación de un algoritmo de ordenación eficiente, conocido este como Heapsort.

Representamos el vector "m" de la respuesta B) como un árbol binario, quedando éste como:



La respuesta A) es falsa, ya que el coste del algoritmo Heapsort es O(n log n).

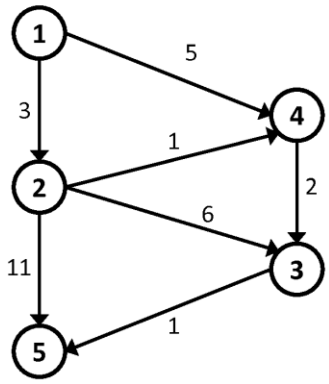
En base al árbol dibujado, vemos que podría ser un montículo de máximos, pero no de mínimos, por lo que la respuesta B) no es correcta.

En el caso de la respuesta C) tampoco es correcta, ya que la función de "insertar" tan solo añade el nodo al final, y flota a su sitio correspondiente, lo cual tiene un coste O(log n).

Tan solo nos queda una respuesta, que será para nuestro caso la correcta, por tanto:

Respuesta D)

4. Dado el siguiente grafo, indique cuáles serían, de acuerdo al algoritmo de Dijkstra, las longitudes de los caminos de coste mínimo que unen el nodo 1 con el resto de nodos del grafo:



(a) {3,4,6,7}
(b) {3,5,7,8}
(c) {3,6,5,7}
(d) Ninguna de las anteriores

Nodos que han salido	Nodos que no han salido	Vector Distancia				Predecesores o Vector Especial			
		2	3	4	5	2	3	4	5
1	2, 3, 4, 5	**3**	∞	5	∞	1	1	1	1
1, 2	3, 4, 5	3	9	**4**	14	1	2	2	2
1, 2, 4	3, 5	3	**7**	4	14	1	4	2	2
1, 2, 4, 3	5	3	7	4	**8**	1	4	2	3
1, 2, 4, 3, 5	-	3	7	4	8	1	4	2	3

Las longitudes de camino mínimo desde el nodo 1 al resto de nodos es {3, 4, 7, 8}, de donde:

Respuesta D)

5. Con respecto a las tablas y funciones Hash, indicar cuál de las siguientes afirmaciones es **cierta**:

(a) Una función Hash asocia unívocamente una clave a un elemento
(b) El recorrido lineal permite mayor dispersión de las colisiones que el cuadrático.
(c) El factor de carga se calcula como el tamaño de la tabla dividido por el número de elementos ya insertados.
(d) Es deseable mantener el factor de carga de la tabla por debajo del 50%.

A) La función Hash es una aplicación no necesariamente inyectiva entre el conjunto dominio de las claves X y el conjunto dominio de direcciones D de la estructura de datos:

$$H: X \rightarrow D$$
$$x \rightarrow h(x)$$

Una función Hash asocia una clave con una posición en la tabla Hash. Las funciones h(x) deben tener las siguientes propiedades:

- Deben repartir equiprobablemente los valores, es decir, para una distribución de probabilidad determinada de valores de X, los valores h(x) en D deben mantener dicha distribución. En este sentido, la distribución de salida no debe estar ligada a ningún patrón.

- La función debe poderse calcular de manera eficiente, de forma que en ningún caso sea el cálculo de h(x) lo que determine la complejidad del proceso de búsqueda.

- Cambios en la clave, aun siendo pequeños, deben resultar en cambios significativos en la función Hash.

B)

- Recorrido Lineal

En general, a la dirección obtenida por la función Hash h(k) se le añade un incremento lineal que proporciona otra dirección en la tabla (donde m es el tamaño de la tabla):

$$s(k,i) = (h(k) + i) \bmod m$$

- Recorrido Cuadrático

En el caso del recorrido lineal, la probabilidad de nuevas colisiones es bastante alta para determinados patrones de claves. Hay otro método basado en una expresión cuadrática g(k) que permite mayor dispersión de las colisiones por la tabla, al mismo tiempo que proporciona un recorrido completo por la misma. Así, si usamos:

$$g(i) = i^2$$

La i-ésima colisión para un valor k viene dada por la secuencia h(k), $h(k) + c_1 i^2$, $h(k) + c_2 i^2$, … es decir:

$$dir = (h(k) + c_k g(k)) \bmod m$$

Las constantes pueden ser del tipo $c_k = 1$ o diferentes. En el caso de que m= 4j + 3 con j entero, entonces se cumple que la exploración cuadrática es completa y ésta recorre todos los bloques de la tabla.

C) El factor de carga es es un parámetro que determina cuántas posiciones de la tabla se han ocupado y la probabilidad de encontrar una entrada vacía. El factor de carga se define como:

$$\delta = \frac{n}{M}$$

Siendo M el tamaño de la tabla, y n el número de índices ocupados. El factor de carga es por tanto un valor entre O (vacía) y l (llena) que mide la proporción de la tabla Hash ya ocupada.

Normalmente es necesario extender la tabla con entradas cuando se ha superado el factor 0.5. Si no es así, lo que se realiza en caso de colisión es un recorrido de entradas vacías hasta encontrar una.

D) Respuesta dada en el párrafo anterior (es correcto).

Respuesta D)

6. Dadas las matrices: *A1 (3x5)*, *A2 (5x2)*, *A3 (2x3)* y *A4 (3x2)*,y siendo *E(i,j)*, *i≤j*, el número de operaciones mínimo para resolver la operación $A_i \times A_{i+1} \times \ldots \times A_j$ mediante programación dinámica, se pide indicar cuál de las siguientes opciones es **cierta**:

   (a) E(2,3) = 15
   (b) E(1,3) = 30
   (c) E(2,4) = 32
   (d) E(2,2) = 10

El producto de un número "n" de matrices es optimizable en cuanto al número de multiplicaciones escalares requeridas. A la hora de multiplicar una serie de matrices se puede elegir en que orden queremos realizar las multiplicaciones entre estas. Se pueden realizar en un orden cualquiera, dada la propiedad asociativa de la multiplicación.

En el caso que se nos plantea, las posibilidades de realizar el cálculo son las siguientes:

1. ((A1 * A2) * A3) * A4
2. (A1 * (A2 * A3)) * A4
3. A1 * ((A2 * A3) * A4)
4. A1 * (A2 * (A3 * A4))
5. (A1 * A2) * (A3 * A4)

Decidamos el orden que decidamos, el resultado siempre es el mismo. La diferencia está en el número de multiplicaciones que implica elegir un orden u otro. Al multiplicar dos matrices A_1 de tamaño pxq y A_2 de tamaño qxr, el número de multiplicaciones escalares es p * q * r.

La cantidad total de multiplicaciones será, la suma de todas las multiplicaciones que hacen falta para multiplicar cada submatriz obtenida como resultado con la siguiente en el orden escogido.

1. ((A1 * A2) * A3) * A4 → 3 * 5 * 2 + 3 * 2 * 3 + 3 * 3 * 2 = 30 + 18 + 18 = 66
2. (A1 * (A2 * A3)) * A4 → 5 * 2 * 3 + 3 * 5 * 3 + 3 * 3 * 2 = 30 + 45 + 18 = 93
3. A1 * ((A2 * A3) * A4) → 5 * 2 * 3 + 5 * 3 * 2 + 3 * 5 * 2 = 30 + 30 + 30 = 90
4. A1 * (A2 * (A3 * A4)) → 2 * 3 * 2 + 5 * 2 * 2 + 3 * 5 * 2 = 12 + 20 + 30 = 62
5. (A1 * A2) * (A3 * A4) → 3 * 5 * 2 + 2 * 3 * 2 + 3 * 2 * 2 = 30 + 12 + 12 = **54**

Eligiendo un orden adecuado, como podemos ver, optimizamos el coste de realizar las multiplicaciones escalares necesarias para llegar al resultado. Para encontrar el modo de ordenar las multiplicaciones vamos a utilizar un algoritmo de Programación Dinámica.

Respuesta A) E(2, 3) = 15 → Falsa

A2 * A3 = 5 * 2 * 3 = **30**

Respuesta B) E(1, 3) = 30 → Falsa

1. (A1 * A2) * A3 = 3 * 5 * 2 + 3 * 2 * 3 = 30 + 18 = **48**
2. A1 * (A2 * A3) = 5 * 2 * 3 + 3 * 5 * 3 = 30 + 45 = 75

Respuesta C) E(2, 4) = 32 → Cierta

1. (A2 * A3) * A4 = 5 * 2 * 3 + 5 * 3 * 2 = 30 + 30 = 60
2. A2 * (A3 * A4) = 2 * 3 * 2 + 5 * 2 * 2 = 12 + 20 = **32**

Respuesta D) E(2, 2) = 10 → Falsa

A2 * A2 → Si i = j → E(i, j) = 0

Se concluye que, finalmente, la respuesta correcta es:

Respuesta C)

Problema (4 puntos).

Desarrollar un programa que halle todas las maneras posibles de que un caballo de ajedrez, mediante una secuencia de sus movimientos permitidos (ver tabla), recorra todas las casillas de un tablero de tamaño *N x N* (para N > 5) a partir de una determinada casilla dada como entrada y sin repetir ninguna casilla.

	*		*	
*				*
		C		
*				*
	*		*	

La resolución del problema debe incluir, por este orden:

1. Elección del esquema más apropiado, el esquema general y explicación de su aplicación al problema (0,5 puntos)

El esquema más apropiado es el de vuelta atrás. El esquema general se encuentra formulado en el libro de texto de la asignatura, en la página161.

2. Descripción de las estructuras de datos necesarias (0,5 puntos solo si el punto 1 es correcto)

Las estructuras de datos son el tablero de NxN que contiene en cada casilla valores que indiquen que el caballo no ha pasado (cero) o que ha pasado (m), donde m es un número natural que indica que en el movimiento m-ésimo, el caballo llega a dicha casilla.

El nodo contaría con dicho tablero, con el número de caballos puestos en el tablero hasta el momento, y con la posición actual del caballo (el último movimiento efectuado).

3. Algoritmo completo a partir del refinamiento del esquema general (2,5 puntos solo si el punto 1 es correcto). Si se trata del esquema voraz, debe realizarse la demostración de optimalidad. Si se trata del esquema de programación dinámica, deben proporcionarse las ecuaciones de recurrencia.

La solución completa al problema se encuentra desarrollada en el libro “Esquemas algorítmicos: enfoque metodológico y problemas resueltos” de J. Gonzalo Arroyo, Y M. Rodríguez Artacho, Cuadernos de la UNED Página 86.

El esquema de Vuelta Atrás utiliza el retroceso como técnica de exploración en grafos. En este caso, el grafo es un árbol donde los nodos son tableros. No es pertinente utilizar el algoritmo de Ramificación y Poda, ya que no existe ningún parámetro que deba ser optimizado. El esquema general de Vuelta Atrás, es el siguiente:

```
fun vuelta-atrás (ensayo)
        si válido (ensayo)
        entonces dev ensayo
        si no    para hijo € compleciones(ensayo)
                 hacer   si condiciones de poda(hijo)
                         entonces vuelta-atrás(hijo)
        fsi
ffun
```

La particularización del esquema al problema del caballo puede hacerse así:

- válido: Un ensayo será válido si no queda casilla alguna por visitar.

- compleciones: Se generarán tantos ensayos como casillas libres haya alcanzables directamente desde la última casilla visitada.

- condiciones de poda: Si sólo se generan ensayos sobre casillas alcanzables, no hay ninguna condición de poda que añadir.

Como se ha indicado, el ensayo consta de 3 elementos, por lo que la función compleciones tiene el esquema general siguiente:

```
fun compleciones(ensayo)
 lista <-- lista vacía;
        si condiciones de poda (ensayo)
        entonces hacer
                 para cada rama hacer
                  w <-- generar ensayo (rama)
                  lista <-- insertar(lista, w);
                 fpara
        fsin
 dev lista
ffun
```

(Continua, ver referencia del libro indicado).

4. Estudio del coste del algoritmo desarrollado (0,5 puntos solo si el punto 1 es correcto)

Sin tener en cuenta las reglas de colocación del caballo, n^2 casillas pueden numerarse de $(n^2)!$ maneras posibles.

Sin embargo, sabemos que el número máximo de ramificaciones del árbol es 8 por ser éstas las alternativas de movimiento de un caballo de ajedrez. Considerando esto, el tamaño del árbol puede acotarse en $8^{n\hat{}}2$, pero se puede precisar que no hay 8 ramas en todos los casos.

De cada n^2 casillas, solo desde $n^2 – 8$ (n - 2) es posible realizar 8 movimientos, y, además, los últimos movimientos no tendrán prácticamente ninguna alternativa.