UNIVERSIDAD DE JAÉN

Departamento de Informática

DISEÑO DE ALGORITMOS

Grado en Ingeniería Informática

Teoría de Algoritmos

RELACIÓN DE PROBLEMAS

• Divide y Vencerás

Problema 1: Los tornillos y las tuercas

En una habitación oscura se tienen 2 cajones. En uno de ellos hay n tornillos de varios tamaños y en el otro las n tuercas que emparejan con cada uno de los tornillos. El problema consiste en emparejar cada tornillo con su tuerca correspondiente. Pero debido a la oscuridad no se pueden comparar tornillos con tornillos ni tuercas con tuercas. La única comparación posible es la de intentar enroscar una tuerca en un tornillo para comprobar si es demasiado grande, demasiado pequeña o se ajusta perfectamente al tornillo.

- ¿Cómo sería el algoritmo clásico que se podría diseñar para resolver este problema? ¿Cuál sería su eficiencia?

- Diseñar e implementar un algoritmo basado en la técnica Divide y Vencerás para solucionar este problema. ¿Cuál sería su eficiencia?

Guía: Algoritmo de ordenación QuickSort

Problema 2: Multiplicación de Polinomios

- Algoritmo de multiplicación clásico.

- Algoritmo de multiplicación basado en la técnica Divide y Vencerás. ¿Cuál sería su eficiencia?

- Para cada algoritmo, calcular la eficiencia, a priori (teórica) y a posteriori (práctica). Para ello implementar los algoritmos y ejecutarlos con distintos valores de umbral n_0, midiendo los tiempos obtenidos en cada caso. Mostrar en una tabla los resultados obtenidos y hacer un análisis de ellos, indicando cuál es el mejor umbral.

Guía: Algoritmo de multiplicación de enteros grandes

Problema 3: Algoritmos de ordenación: Mergesort

Implementa el algoritmo de ordenación Mergesort sobre vectores de enteros de tamaño n

Problema 4: Búsqueda binaria

Implementa la búsqueda binaria sobre un vector de enteros ordenado de tamaño n.

- Si el elemento central es mayor que el valor buscado, continuar la búsqueda por la izquierda. Si es menor, continuar por la derecha.
- Continuar buscando hasta que se encuentre o se dé la condición de parada.
- ¿Cuál es el mejor umbral?
- Calcula el orden de complejidad.

Problema 5: Elemento Mayoritario de un Conjunto de Datos

Dado un conjunto C de n elementos (no necesariamente ordenables) se dice que un elemento x es mayoritario en C cuando el número de veces que aparece x en C es estrictamente mayor que n/2. Dado un conjunto de elementos se desea saber si un conjunto contiene un elemento mayoritario y devuelva tal elemento cuando exista.

- ¿Cómo sería el algoritmo clásico que se podría diseñar para resolver este problema? ¿Cuál sería su eficiencia?
- Diseñar e implementar un algoritmo basado en la técnica Divide y Vencerás para solucionar este problema. ¿Cuál sería su eficiencia?

Problema 6: El cuadrado latino

Un cuadrado latino es una matriz de n×n elementos, en la que cada casilla está ocupada por uno de los n símbolos de tal modo que cada uno de ellos aparece exactamente una vez en cada columna y en cada fila. Las siguientes matrices son cuadrados latinos:

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{bmatrix} \qquad \begin{bmatrix} a & b & d & c \\ b & c & a & d \\ c & d & b & a \\ d & a & c & b \end{bmatrix}$$

- Diseñar un algoritmo divide y vencerás que construya un cuadrado latino de tamaño n. (Asumir que n es potencia de 2)

• Algoritmos Voraces

Problema 7:

En una guerra, los ejércitos enemigos han invadido **n** ciudades. Los servicios de inteligencia están informados de que en cada una de las ciudades invadidas se encuentran $\mathbf{e_i}$ efectivos enemigos. Para contraatacar, el Grupo de Intervención Rápida dispone de **n** equipos listos para intervenir. Cada uno de ellos consta de $\mathbf{d_j}$ efectivos completamente equipados y entrenados. Para garantizar el éxito de la intervención en una ciudad es necesario que se cuenten al menos con tantos efectivos de defensa como el enemigo.

Describir e implementar al menos dos estrategias voraces (distintas funciones de selección) que indiquen qué equipo de intervención debe ir a cada ciudad, de forma que se maximice el número de éxitos garantizados.

Problema 8:

La Universidad de Jaén tiene que planificar un evento cultural que consiste en **n** conferencias. Para cada conferencia se conoce las horas de comienzo y la de finalización fijadas por los ponentes. Se ha pedido al Departamento de Informática que planifique las **n** conferencias distribuyéndolas entre las distintas salas disponibles, de forma, claro está, que no haya dos conferencias en una misma sala al mismo tiempo. El objetivo es minimizar el número de salas utilizadas, para así causar el menor trastorno posible al resto de actividades académicas.

Describir e implementar al menos dos estrategias voraces (distintas funciones de selección) para solucionar este problema.

Problema 9:

Supongamos que tenemos **n** archivos A_1, A_2, … A_n cada uno con un tamaño de $\mathbf{t_i}$ Megabytes y los queremos almacenar en un dispositivo de almacenamiento secundario con capacidad **M** Megabytes.

Si la capacidad del dispositivo de almacenamiento secundario es insuficiente para contener todos los archivos, resolver los siguientes problemas utilizando una estrategia voraz:

a) Se desea maximizar el número de archivos almacenados en el disco.

b) Se desea maximizar el espacio utilizado del disco.

Para cada uno de ellos demostrar o dar un contraejemplo de si con la estrategia elegida se alcanza siempre la solución óptima.

Problema 10:

Los profesores de Diseño de Algoritmos se ha propuesto hacer trabajar en firme a sus **n** alumnos, por lo que se ha sacado de la manga **n** trabajos. A pesar las muchas horas de botellón, todos los alumnos son capaces de hacer cualquier trabajo, aunque unos tardan más que otros y unos los hacen mejor que otros. La información al respecto se recoge en dos tablas **T[n*n]** y **E[n*n]**, donde **T[i,j]** representa el tiempo que el alumno **i** tarda en realizar el trabajo **j** y **E[i,j]** representa la eficacia con la que el alumno **i** realiza el trabajo **j**.

Siempre pensando en el bien de los alumnos, dichos profesores desean conocer la asignación óptima de trabajos a los alumnos en cada uno de los dos sentidos siguientes:

a) Que la suma total de tiempos sea mínima
b) Que la suma total de eficacias sea máxima

Diseñar dos algoritmos basados en una estrategia voraz para ayudar al pobre profesor en esta dura tarea

Problema 11:

Un fontanero necesita hacer **n** reparaciones urgentes, y sabe de antemano el tiempo que le va a llevar cada una de ellas: en la tarea **i**-ésima tardará $\mathbf{t_i}$ minutos. Como en su empresa le pagan dependiendo de la satisfacción del cliente, necesita decidir el orden en el que atenderá los avisos para minimizar el tiempo medio de espera de los clientes.

En otras palabras, si llamamos $\mathbf{E_i}$ a lo que espera el cliente **i**-ésimo hasta ver reparada su avería por completo, necesita minimizar la expresión:

$$E(n) = \sum_{i=1}^{n} E_i$$

Diseñar un algoritmo voraz que resuelva el problema y probar su validez, bien mediante demostración formal o con un contraejemplo que la refute.

Problema 12: (Heurísticas voraces) El problema del coloreado de un grafo

Sea G= <N, A> un grafo no dirigido y sea **n** el número de nodos del grafo. Un coloreado de G es una asignación de colores a los nodos de G de tal manera que no haya dos nodos adyacentes que tengan asignado el mismo color. Al mínimo número de colores que se necesitan para colorear un grafo se le denomina *número cromático del grafo*.

Diseñar una heurística voraz que resuelva el problema del coloreado de un grafo y obtenga su número cromático.

- **Programación Dinámica**

Problema 13: Memorización

La sucesión cuyo término n-ésimo cuenta la cantidad máxima de operaciones del algoritmo de Gauss para triangular una matriz de n × n es:

$$x_n = \begin{cases} 0 & si\ n = 1 \\ 3 & si\ n = 2 \\ 13 & si\ n = 3 \\ 34 & si\ n = 4 \\ 4x_{n-1} - 6x_{n-2} + 4x_{n-3} - x_{n-4} & si\ n > 4 \end{cases}$$

Implementa un algoritmo basado en Programación Dinámica que calcule el término n-ésimo de esta sucesión

Problema 14: Memorización

Implementa un algoritmo basado en PD que calcule el término n-ésimo de la siguiente función recursiva:

$$x_n = \begin{cases} 3 & si\ n = 1 \\ 3 & si\ n = 2 \\ x_{n-1} - 3 & si\ n > 2\ y\ n\ es\ par \\ x_{n-2} + 3 & si\ n > 2\ y\ n\ es\ impar \end{cases}$$

Problema 15: Tarificación Postal

En un determinado país se emiten n sellos diferentes de valores naturales positivos s_1, s_2, …, s_n. Se quiere enviar una carta y se sabe que la correspondiente tarifa postal es T. ¿De cuántas formas diferentes se puede franquear exactamente la carta, si el orden de los sellos no importa?

Guía: Para solucionar este problema se puede definir la función:

formas (n, T) = número de formas de franquear T con n tipos de sellos

A partir de aquí se plantea la recurrencia en el caso general, cuando se consideran los sellos del 1 al i y se quiere franquear una cantidad j: *formas (i,j).* (Utilizar un enfoque hacia atrás).

Una vez hecho esto habrá que determinar cuáles son los casos base (soluciones triviales) y hallar sus respectivos valores.

Problema 16: Cambio de Monedas

Vivimos en un país donde el sistema monetario está formado por una serie de monedas, cada una de ellas con un valor facial.

$$M=\{m_1, m_2, \ldots, m_n\}$$

Diseñar un algoritmo que permita devolver una cierta cantidad de dinero C utilizando el menor número de monedas posible.

Guía: Para resolver este problema podemos considerar su similitud con el problema de la mochila. Así, si $x_1, x_2, \ldots, x_{n-1}, x_n$ representa las monedas en un cambio óptimo para una cantidad C, entonces $x_1, x_2, \ldots, x_{n-1}$ debe ser también un cambio óptimo para la cantidad C-Valor_Facial (x_n) y de forma obvia x_n es un cambio óptimo para la cantidad Valor_Facial (x_n).

Si consideramos el peso de una moneda como su valor facial y la ganancia (lo que aporta una moneda para la solución final) es 1, hay que tener en cuenta que para este problema el objetivo es MINIMIZAR la ganancia o el número total de monedas

Definir una función: *monedas (n, C)* que represente el mínimo de monedas necesarias para pagar la cantidad C considerando los tipos de monedas del 1 al n.

A partir se plantea la recurrencia en el caso general, cuando se consideran los tipos de monedas del 1 al i y nos queda pagar una cantidad j: *monedas(i,j)*. (Utilizar un enfoque hacia atrás).

Una vez hecho esto habrá que determinar cuáles son los casos base (soluciones triviales) y hallar sus respectivos valores.

Problema 17: El problema del Botellón

Dos amigos, *Agonioso* y *Listillo*, salen de botellón. La nueva moda es poner una fila con n vasos (n es par). Cada vaso i, entre 1 y n contiene una cantidad de líquido c_i distinta. Los amigos beben por turnos. Cada uno, en su turno debe elegir el vaso de uno de los extremos y beberse su contenido. El vaso se retira y el turno pasa al otro amigo. La persona que comienza bebiendo se determina a priori por un procedimiento cualquiera. El objetivo de ambos amigos es beber la mayor cantidad posible de líquido.

La estrategia de *Agonioso* consiste en pensar poco y coger el vaso de los extremos que esté más lleno. En cambio *Listillo* prefiere pensárselo un poco más.

a) Demostrar, con un contraejemplo que la estrategia de *Agonioso* no es óptima, incluso cuando le toca escoger primero.
b) *Listillo* tiene unos amigos que cursan la asignatura de Diseño de Algoritmos y les pide que le diseñen un algoritmo, basado en

Programación Dinámica, para que le ayude a conseguir su objetivo, suponiendo que es él quien empieza a escoger.

Guía: Definir una función *botellón(i,j)* que represente la cantidad máxima que bebe *Listillo* con los cubos desde el i hasta el j cuando le toca empezar a beber. La solución final será *botellón(1,n)*

Problema 18: Multiplicación de una secuencia de matrices

El producto de una matriz $A_{p \times q}$ y una matriz $B_{q \times r}$ es una matriz $C_{p \times r}$ cuyos elementos son:

$$C_{i,j} = \sum_{k=1}^{q} a_{i,k} * \ b_{k,j}$$

Por tanto se necesitan p*q*r multiplicaciones escalares para calcular C. Si ahora se quiere multiplicar una secuencia de matrices M_1, M_2, …, M_n, donde cada matriz M_i tiene dimensiones d_{i-1} x d_i, el orden de las matrices no se puede alterar, pero sí el de los productos a realizar, ya que la multiplicación de matrices es asociativa. Desarrollar un algoritmo basado en Programación Dinámica que inserte paréntesis en la secuencia de matrices de forma que el número total de multiplicaciones escalares sea mínimo.

Guía: Si se decide que el último producto a realizar es el que está entre la matriz k y la k+1, los paréntesis se pondrán de la forma:

$$(M_1 \ldots M_k)_{d0 \times dk} \ (M_{k+1}, \ldots, M_n)_{dk \times dn}$$

Es decir, se multiplicará una matriz de dimensiones d_0 x d_k por otra d_k x d_n, realizando d_0 x d_k x d_n multiplicaciones escalares. El número total de multiplicaciones escalares se obtendrá sumando a d_0 x d_k x d_n el número total de multiplicaciones para calcular (M_1 … M_k) y para calcular (M_{k+1}, …, M_n), que también deben ser óptimos.

La posición k puede variar desde 1 hasta n-1, por lo que se necesita calcular el mínimo al variar k para obtener el óptimo.

Definir la función *matrices(i,j),* que represente el número mínimo de multiplicaciones escalares para realizar el producto de las matrices comprendidas entre la i y la j: (M_i, …, M_j). (Utilizar un enfoque hacia adelante).

Una vez hecho esto habrá que determinar cuáles son los casos base (soluciones triviales) y hallar sus respectivos valores.

Problema 19: Árboles Binarios de Búsqueda Óptimos

Supongamos que tenemos un conjunto ordenado de n elementos distintos:

$$C_1 < C_2 < \ldots < C_n$$

La probabilidad de que estemos interesados en buscar la clave C_i es p_i,

con $1 \leq i \leq n$ y [redacted]

Si se almacena la clave C_i en un nodo con profundidad d_i (recordemos que la profundidad de la raíz es 0, a de sus hijos 1 y así sucesivamente), entonces se necesitan $d_i + 1$ comparaciones para hallarla. Por tanto, para un árbol dado, el número medio de comparaciones viene determinado por:

[redacted]

El problema que nos planteamos es el de encontrar el árbol binario de búsqueda que minimice el número medio de comparaciones necesarias para realizar una búsqueda.

Guía: La resolución de este problema puede seguir un esquema análogo al utilizado en la multiplicación de una secuencia de matrices.

Sea T un árbol binario de búsqueda óptimo y sea C_k la clave que se encuentra en la raíz del mismo. Por las propiedades de los árboles binarios de búsqueda, se sabe que las claves C_1, .., C_{k-1} tienen que ir al hijo izquierdo y las claves C_{k+1}, .., C_n tienen que ir al hijo derecho. Para que se verifique el principio de optimalidad tenemos que las estructuras para el subárbol izquierdo T_i y el subárbol derecho T_d también deben ser optimales.