Universidad de Valladolid
Departamento de Informática

Edificio de Tecnologías de la Información | Campus Miguel Delibes, Ctra. del Cementerio s/n | Tel.: (983) 42 36 70, Fax: (983) 42 36 71

I.T. Informática de Gestión – Estructuras de Datos

Hoja de Problemas – Curso 07/08

[✍: Problema retocado. ☼: Problema nuevo]

Análisis de Algoritmos

1. Un programador ha implementado un algoritmo que tarda, en su antiguo ordenador de 100 Mhz, una milésima de segundo en resolver un problema de tamaño $n = 20$. ¿Cuánto tardaría en resolver un problema de tamaño $n = 30$ en los siguientes casos?
   a) El algoritmo es de orden $O(n^3)$
   b) El algoritmo es de orden $O(2^n)$
   c) El algoritmo es de orden $O(n!)$

   ¿Por qué factor debería multiplicar la velocidad de su ordenador para que volviera a tardar una milésima de segundo? **Nota**: 1 año = 31.557.600 segundos.

2. ¿Es posible que un algoritmo de orden $O(n^2)$ se ejecute más rápidamente que un algoritmo de orden $O(n)$ en los siguientes casos?
   a) Para un tamaño $n = 100$.
   b) Para un determinado valor de la entrada.
   c) Para el mejor caso del primer algoritmo.
   d) Para todos los valores de la entrada.
   e) Si se ejecutan en máquinas distintas y podemos elegir aquella en que se ejecuta el primer algoritmo.

3. Al medir experimentalmente un algoritmo se encuentra que nunca emplea más de $k_1 \cdot n^3$ operaciones ni menos de $k_2 \cdot n^2$, siendo n el tamaño de los datos de entrada y k_1 y k_2 constantes. ¿Cuál sería la hipótesis más adecuada al interpretar estos datos?
   - El algoritmo es $O(n^2)$ y $\Omega(n^3)$
   - El algoritmo es $O(n^3)$ y $\Omega(n^2)$
   - El algoritmo es $\Theta(n^{2.5})$
   - El algoritmo es $\Theta(n^2 + n^3)$

4. Analizar cuál sería el orden de complejidad del algoritmo que usamos habitualmente para multiplicar dos números enteros. Se supondrá que los dos números tienen el mismo número de dígitos, n, y que este valor define el tamaño de la entrada. Contar únicamente las operaciones de producto y suma de dígitos individuales.

Edificio de Tecnologías de la Información | Campus Miguel Delibes, Ctra. del Cementerio s/n | Tel.: (983) 42 36 70, Fax: (983) 42 36 71

5. Analizar el número de operaciones en las que interviene un elemento del vector en el siguiente algoritmo (ordenación por selección):

```
type TVector = array[1..N] of real;

procedure ordenar(var V: TVector);
var
  I,J,Min : integer;
  Temp : real;
begin
  for I := 1 to N-1 do
  begin
    Min := I;
    for J := I+1 to N do
      if V[J] < V[Min] then Min := J;
    Temp := V[I];
    V[I] := V[Min];
    V[Min] := Temp
  end
end
```

Encontrar las fórmulas exactas para el peor y el mejor caso, identificar cuáles son los tipos de entradas que producen esos casos y repetir el análisis utilizando notación asintótica.

6. Calcular el número de operaciones de incremento de *x* que realizan los siguientes fragmentos de programas, teniendo en cuenta que sólo deseamos conocer exactamente el término de mayor crecimiento, y el resto de términos se pueden expresar usando notación asintótica:

```
for i := 1 to n do
  for j := 1 to n do
    if j = i then
      x := x+1
```

```
for i := 1 to n do
  for j := n downto i do
    for k := i to j do
      x := x+1
```

```
for i := 1 to n do
  for j := 1 to i*i do
    for k := 1 to j*j do
      x := x+1
```

7. Calcular el orden de complejidad de las siguientes funciones recursivas (contar sólo las operaciones producto):

```
function f1(n: integer): integer;
begin
  if n < 1 then f1 := 1 else
    f1 := f1(n-3)*f1(n-3)
end;
```

```
function f2(n: integer): integer;
begin
  if n < 1 then f2 := 1 else
    f2 := f2(n div 2)+f2(n div 2)*
          f2(n div 2)
end;
```

```
function f3(n: integer): integer;
var i,x: integer;
begin
  if n < 1 then f3 := 1 else
  begin
    x := 1;
    for i := 1 to n do x := x*n;
    f3 := x+f3(n-1)
  end
end
```

```
function f4(n: integer): integer;
var i,x: integer;
begin
  if n < 1 then f4 := 1 else
  begin
    x := 1;
    for i := 1 to n*n do x := x*2;
    f4 := x*f4(n div 2)
  end
end
```

Edificio de Tecnologías de la Información | Campus Miguel Delibes Ctra. del Cementerio s/n | Tel.: (983) 42 36 70 Fax: (983) 42 36 71

8. Demostrar si los siguientes enunciados son ciertos o falsos:

   a) ¿$\log 3^n \in \Theta(n)$?

   b) ¿$2^{n+1} \in O(2^n)$?

   c) ¿$3^n \in \Omega(2^n)$?

   d) ¿$2n^2 - n + 1 \in \Omega(n)$?

   e) ¿$\log(n!) \in \Theta(n \log n)$?

   Nota: Para el último caso puede ser conveniente utilizar el método de inducción: Suponer que es cierto para un determinado valor de *n* y demostrar que eso implica necesariamente que la expresión es cierta para *n*+1.

9. Calcular el orden de complejidad respecto al tiempo (suponiendo que deseamos contar las operaciones producto) y al espacio de la siguiente función:

```
function f(n: integer) : integer;
var i,x: integer;
begin
  if n < 1 then f := 1 else
  begin
    x := n;
    for i := 1 to n do x := x+1;
    f := f(n div 2)*f(n div 2)+f(n div 3)+x
  end
end;
```

10. Al analizar la eficiencia de una operación de inserción sobre una estructura de datos se encuentra que a veces la inserción tarda un tiempo $\Theta(n^3)$ pero eso implica que las siguientes *n* inserciones tardarán un tiempo de $O(1)$. Analizar el comportamiento del algoritmo en el peor y mejor caso, el caso promedio y el tiempo amortizado.

Diseño de Algoritmos

11. Dado un número entero *n* se desea obtener todas las posibles formas de expresarlo como sumas de los enteros 1 .. *n*, permitiendo que el mismo sumando aparezca varias veces. Se considerarán iguales (y por lo tanto sólo se debe escribir una de ellas) aquellas soluciones que contengan los mismos sumandos pero ordenados de otra manera. Por ejemplo, si $n = 4$, se deberían escribir las soluciones 4=1+1+1+1, 4=2+1+1, 4=2+2, 4=3+1, 4=4.

    Crear un algoritmo que resuelva el problema. No importa el orden en que aparecen las soluciones ni el orden de los sumandos dentro de cada solución.

12. Resolver el problema de las torres de Hanoi con cuatro postes. Evaluar el orden del algoritmo y comprobar si supone una mejora respecto al problema con tres postes. Verificar si es posible aplicar una estrategia divide y vencerás al problema.

Edificio de Tecnologías de la Información | Campus Miguel Delibes, Ctra. del Cementerio s/n | Tel.: (983) 42 36 70, Fax: (983) 42 36 71

13. Resolver el problema de las torres de Hanoi con tres postes cuando se parte de cualquier posición válida. **Sugerencia**: Encontrar primero un esbozo de solución y diseñar después el tipo de datos que represente la posición de los discos que sea más adecuado para esa forma de resolver el problema. Evaluar el orden del algoritmo obtenido.

14. El problema de evaluar la expresión a^b donde *a* es un valor real y *b* un entero positivo se puede resolver mediante un bucle donde se evalúe *b* veces el producto de *a*, o bien es posible aplicar la siguiente fórmula en un esquema divide y vencerás:

$$a^b = \begin{cases} \left(a^{b \operatorname{div} 2}\right)^2 & \text{si } b \text{ es par} \\ a \cdot \left(a^{b \operatorname{div} 2}\right)^2 & \text{si } b \text{ es impar} \end{cases}$$

    Desarrollar algoritmos para ambas soluciones y evaluar el orden de complejidad de cada uno tomando como tamaño de la entrada el número de bits de *b*.

15. Se dispone de una matriz de $n \times n$ elementos que pueden tomar uno de estos tres posibles valores: *pared*, *libre* o *marca*. La matriz representa un laberinto donde desde una casilla es posible acceder a cualquier casilla adyacente (superior, inferior, izquierda y derecha) que tenga el valor *libre* o *marca*. Inicialmente se proporciona la matriz que define el laberinto donde las casillas tienen los valores *libre* o *pared*, y la fila y columna de partida.

    Desarrollar un algoritmo que indique si existe o no un camino para salir del laberinto partiendo de la posición inicial y pasando únicamente por casillas marcadas inicialmente como *libres*. Salir del laberinto significa alcanzar una casilla libre en los límites de la matriz. Si la posición inicial está fuera del laberinto o corresponde a una casilla con valor *pared*, la respuesta será falso.

    Nota: Se permite modificar el contenido de la matriz haciendo uso del valor marca con el propósito de no volver a pasar por una casilla ya recorrida.

    Modificar el algoritmo obtenido para que, en el caso de que exista una solución, imprima las filas y columnas de las casillas que se deben recorrer para salir del laberinto.

16. Crear un programa que lea una cadena de caracteres, compruebe no existe en ella ningún carácter repetido, y en ese caso escriba todas las cadenas que se pueden formar por permutaciones de los caracteres de la cadena original. Por ejemplo, si la cadena leída fuera "DATO", el programa escribiría:

DATO	DAOT	DTAO	DTOA	DOAT	DOTA
ADTO	ADOT	ATDO	ATOD	AODT	AOTD
TDAO	TDOA	TADO	TAOD	TODA	TOAD
ODAT	ODTA	OADT	OATD	OTDA	OTAD

17. Se dispone de un tablero de ajedrez de *n* x *n* casillas. El problema del **recorrido del caballo** consiste en encontrar una forma de recorrer, comenzando en la casilla (1,1), todas las casillas del tablero pasando de una a otra mediante saltos de caballo y sin pasar más de una vez por cada casilla. Crear un algoritmo que resuelva el problema.

Edificio de Tecnologías de la Información | Campus Miguel Delibes Ctra. del Cementerio s/n | Tel.: (983) 42 36 70 Fax: (983) 42 36 71

18. El problema las ***n* reinas** consiste en colocar *n* reinas sobre un tablero de ajedrez de *n* x *n* casillas de tal forma que ninguna reina pueda *amenazar* a otra. Aplicando la técnica de backtracking crear un algoritmo que, dado un valor de *n* entre 1 y 12:

    a) Presente una solución o bien indique que no existe ninguna.

    b) Presente todas las soluciones existentes.

    Diseñar los tipos de datos más adecuados para facilitar la operación de comprobar si una casilla es accesible a alguna de las reinas dispuestas en el tablero.

19. ✍ El problema del **producto de enteros grandes** consiste en calcular el producto de enteros de precisión arbitraria (representados como arrays de *n* bytes). El número que representa el producto consistirá en un array de cómo mucho 2*n* bytes. Este resultado se debe calcular usando como operación elemental el producto de bytes individuales.

    - Crear un esquema de algoritmo, basado en el método clásico de multiplicación, que resuelva el problema anterior y evaluar su eficiencia.
    - Intentar encontrar un esquema divide y vencerás que mejore el orden del algoritmo anterior.

20. ✍ Cuando en Pascal ejecutamos la instrucción **`write(x)`** donde *x* es un número entero positivo y mayor que cero, internamente se ejecuta un algoritmo parecido al siguiente:

```
procedure EscribeNumero(x: integer);
const Digitos = '0123456789';
begin
  if x > 0 then
  begin
    write(Digitos[x mod 10]);
    EscribeNumero(x div 10)
  end
end;
```

    Si analizamos el orden de éste algoritmo vemos que el número de operaciones de resta y división que se realizan es proporcional al logaritmo en base 10 de *x* (el número de veces que se puede dividir *x* por 10 hasta que valga cero). Ya que $\log_{10} x = 3.32 \cdot \log_2 x$ obtenemos que el tiempo es proporcional al número de bits de *x*.

    Supongamos que estamos trabajando con enteros de precisión arbitraria, y por lo tanto ahora *x* se representa como un array de *n* bits. En este caso las operaciones de resta y cociente no son elementales y supondremos que tienen el siguiente coste:

    - x **mod** 10, x **div** 10 : $\Theta(n)$
    - x **mod** $10^{n/2}$, x **div** $10^{n/2}$: $\Theta(n^{1.5})$

    Analizar el orden del algoritmo anterior en ésta nueva situación, expresándolo en función de *n* (número de bits de *x*). Crear un nuevo algoritmo basado en la estrategia divide y vencerás y analizar su coste, comprobando si mejora o no el del algoritmo anterior.

Edificio de Tecnologías de la Información | Campus Miguel Delibes, Ctra. del Cementerio s/n | Tel.: (983) 42 36 70, Fax: (983) 42 36 71

21. Usar la técnica de la tabla de resultados parciales para desarrollar un algoritmo que calcule el coeficiente binomial $C(n,m)$. ¿Cuál es el orden de complejidad temporal y espacial del algoritmo? Compararlo con el orden de la solución recursiva.

$$C(n,m) = \begin{cases} 1 & (m = 0) \vee (m \geq n) \\ C(n-1,m) + C(n-1,m-1) & 0 < m < n \end{cases}$$

22. ¿Es posible utilizar la técnica de la tabla de resultados parciales para calcular la función de Ackerman?

$$A(n,m) = \begin{cases} m+1 & \text{si } n = 0 \\ A(n-1,1) & \text{si } m = 0 \\ A(n-1, A(n,m-1)) & \text{caso contrario} \end{cases}$$

23. **El problema de la mochila 0/1**. En este problema disponemos de dos vectores, $v_1...v_n$, y $p_1...p_n$ que indican, respectivamente, el valor y el peso de cada uno de n objetos. El objetivo es decidir, para cada objeto, si lo incluimos o no en una mochila cuya capacidad máxima es M kilos. De entre todas las maneras distintas de llenar la mochila sin superar ese límite, debemos escoger aquella que proporcione la mochila más valiosa.

    Es decir, si expresamos la solución como el vector $x_1...x_n$ donde $x_i = 1$ indica que el objeto i se incluye en la mochila y $x_i = 0$ indica que el objeto i no se incluye, se debe cumplir que:

    $\sum_{i=1}^{n} x_i \cdot p_i \leq M$ (restricción), y que $\sum_{i=1}^{n} x_i \cdot v_i$ (valor de la mochila) es máximo.

    Resolver este problema utilizando backtracking y estimar el orden de complejidad en el peor caso, contando las operaciones en que se acceda a vectores.

24. ☼ Idear un algoritmo voraz para el problema anterior. No es necesario que proporcione la mochila óptima, tan sólo que de una solución válida (mochila no supere el límite de peso) y razonablemente "buena". ¿Podéis encontrar una cota que indique la diferencia máxima de valores entre la mochila que proporciona el algoritmo voraz y la mochila óptima?

25. ✍ El **problema de la herencia** consiste en encontrar el mejor reparto posible de n objetos en dos lotes, dados los valores de cada objeto (vector $v_1...v_n$). El mejor reparto es aquel que hace los valores de ambos lotes se diferencien en la menor cantidad posible.

    - Intentar encontrar una solución mediante programación dinámica. Verificar si se cumple el principio de suboptimalidad.
    - Reformular el problema de manera que pueda resolverse en términos del problema anterior.

26. ✍ El **problema del subconjunto suma** consiste en, dado un conjunto de n enteros positivos, $v_1...v_n$, determinar si existe algún subconjunto cuya suma sea igual al valor s (también positivo). Intentar encontrar una solución mediante programación dinámica a este problema.

Edificio de Tecnologías de la Información | Campus Miguel Delibes Ctra. del Cementerio s/n | Tel.: (983) 42 36 70 Fax: (983) 42 36 71

27. ☼ El **problema del cambio en monedas** consiste en determinar cómo dar cambio de una cantidad de dinero *c* usando la menor cantidad posible de monedas. Suposiciones:

    - Estamos en un país donde existen *n* tipos de monedas/billetes distintos, y nos proporcionan un vector, $m_1...m_n$, que indica el valor facial de cada tipo de moneda/billete. Este vector está ordenado de menor a mayor.
    - Tanto *c* como los elementos de *m* son números enteros (en nuestro caso la unidad sería el céntimo de euro y el vector *m* valdría [1,2,5,10,20,50,100,200,500,1000,2000,5000,10000,20000,50000])
    - Siempre existe un billete o moneda de valor 1 ($m_1 = 1$). Esto garantiza que siempre se pueda dar cambio.

    La solución consiste en proporcionar un vector, $v_1...v_n$, que indique el número de monedas de cada tipo que utilizamos, y que cumpla:

    $$\sum_{i=1}^{n} v_i \cdot m_i = c \text{ (damos cambio exacto de } c\text{) y que } \sum_{i=1}^{n} v_i \text{ (número de monedas) sea mínimo.}$$

    Diseñar un algoritmo que resuelva el problema. **Ejemplo**: si estuviésemos en un país con monedas de 1,6 y 10, y tuviésemos que dar cambio de 18, la solución óptima sería [0,3,0] (usar 3 monedas de 6).

28. Un cierto número de usuarios, *n*, envían simultáneamente un documento a la impresora común, la cual debe determinar su orden de impresión. Las longitudes de los documentos enviados son $l_1...l_n$, siendo l_i la longitud del documento enviado por el usuario *i* (la numeración de los usuarios es arbitraria). Suponiendo que el tiempo que se tarda en imprimir un documento es proporcional a su longitud, diseñar un algoritmo que indique el orden óptimo en que se deben imprimir de manera que se minimice el tiempo medio de espera de cada usuario.

    El tiempo de espera del usuario *i*-ésimo vendrá dado por el orden que haya establecido la impresora para su documento. Si su documento es el *j*-ésimo en imprimirse, su tiempo de espera será la suma de los tiempos de impresión de los *j* primeros documentos según ese orden (se incluye el suyo en la suma).

29. ✍ Dada una secuencia enteros, $v_1...v_n$, ordenada de menor a mayor, crear un algoritmo que encuentre la posición del elemento con un valor más cercano a cero en un tiempo mejor que $O(n)$. **Nota**: La secuencia puede contener valores negativos.

30. Dada una secuencia de valores, $X \equiv x_1 \ldots x_n$, se denomina **subsecuencia no contigua** de *X* a toda secuencia cuyos componentes pertenecen a un subconjunto de valores de *X* y están dados con el mismo orden de aparición que en la secuencia *X*. Es decir, que la secuencia $Y \equiv y_1 \ldots y_m$ es una subsecuencia no contigua de *X* si cumple:

    $$\forall\, i : 1 \le i < m : \exists\, j, k : j < k : (y_i = x_j) \wedge (y_{i+1} = x_k)$$

    Por ejemplo, una posible subsecuencia no contigua de (1,2,3,4,5,6,7,8,9) podría ser la secuencia (2,3,5,7,8,9). Resolver el problema anterior cuando no se exige que la subsecuencia sea contigua.

Universidad de Valladolid
Departamento de Informática

Edificio de Tecnologías de la Información | Campus Miguel Delibes, Ctra. del Cementerio s/n | Tel.: (983) 42 36 70, Fax: (983) 42 36 71

31. Dada la secuencia de enteros $v_1...v_n$ diseñar un algoritmo para encontrar la subsecuencia ascendente no contigua de mayor longitud (número de elementos). **Nota**: Ascendente significa que cada elemento sea estrictamente mayor que el anterior.

32. ☼ El **problema de la evaluación óptima del producto de matrices** consiste en averiguar la manera óptima de **parentizar** un producto de n matrices (de las cuales conocemos las dimensiones – filas y columnas – de cada una de ellas) de manera que al calcular el resultado se realice la menor cantidad posible de operaciones elementales (productos y sumas entre elementos de matrices).

    Al multiplicar dos matrices de dimensiones (f_1, c_1) y (f_2, c_2):

    a) Se debe cumplir que el número de columnas de la primera matriz sea igual al número de filas de la segunda ($c_1 = f_2$)

    b) Se realizan $f_1 \cdot c_1 \cdot c_2$ productos y sumas de elementos.

    c) El resultado es una matriz de dimensiones (f_1, c_2)

    La entrada del problema serán las dimensiones de las n matrices. Debido a la propiedad (a) tan sólo se necesitan n+1 valores, $d_1..d_{n+1}$, donde las dimensiones de la matriz M_i son d_i filas y d_{i+1} columnas. La salida consistirá en escribir una fórmula que indique el orden de evaluación.

    Por ejemplo, supongamos que la entrada es [10,5,1,2,100]. Esto indica que debemos multiplicar 4 matrices con las siguientes dimensiones: M_1 : [10×5], M_2 : [5×1], M_3 : [1×2] y M_4 : [2×100]. El producto $M_1 \times M_2 \times M_3 \times M_4$ se puede evaluar de 5 maneras distintas, obteniendo la misma matriz resultado a un coste en operaciones elementales muy distinto:

$\overset{10\cdot5\cdot100=5000}{\underset{5\cdot1\cdot100=500}{\underset{1\cdot2\cdot100=200}{}}}$ $M_1\times(M_2\times(M_3\times M_4))$	$\overset{10\cdot5\cdot100=5000}{\underset{5\cdot2\cdot100=1000}{\underset{5\cdot1\cdot2=10}{}}}$ $M_1\times((M_2\times M_3)\times M_4)$	$\overset{10\cdot1\cdot100=1000}{\underset{10\cdot5\cdot1=50}{}\quad\underset{1\cdot2\cdot100=200}{}}$ $(M_1\times M_2)\times(M_3\times M_4)$	$\overset{10\cdot2\cdot100=2000}{\underset{10\cdot5\cdot2=100}{\underset{5\cdot1\cdot2=10}{}}}$ $(M_1\times(M_2\times M_3))\times M_4$	$\overset{10\cdot2\cdot100=2000}{\underset{10\cdot1\cdot2=20}{\underset{10\cdot5\cdot1=50}{}}}$ $((M_1\times M_2)\times M_3)\times M_4$
Coste = 5700	Coste = 6010	Coste = 1250	Coste = 2110	Coste = 2070

    En este caso la salida debería ser el escribir `(M1xM2)x(M3xM4)` en pantalla. Es necesario darse cuenta que no deseamos *calcular* el resultado (el contenido de las matrices se desconoce), sino indicar *el orden de evaluación* del producto que es óptimo.

33. Se dispone de una secuencia de n enteros, $v_1...v_n$, y se desea hallar la subsecuencia cuya suma sea máxima. Por ejemplo, dada la secuencia -2, 10, 5, -3, 4, -6, -2, 7 la subsecuencia de suma máxima es la comprendida entre los índices 2 y 5 (10, 5, -3, 4).

    a) Obtener un algoritmo $O(n^3)$ utilizando la estrategia de fuerza bruta (comprobar las $(n^2-n)/2$ subsecuencias posibles).

    b) Refinar el algoritmo anterior para obtener un orden $O(n^2)$.

    c) Aplicar la estrategia divide y vencerás para conseguir un algoritmo $O(n \log n)$.

    d) Intentar encontrar un algoritmo heurístico con un tiempo $O(n)$.

Edificio de Tecnologías de la Información | Campus Miguel Delibes Ctra. del Cementerio s/n | Tel.: (983) 42 36 70 Fax: (983) 42 36 71

34. Se dispone de dos secuencias de valores, A ≡ $a_1 \ldots a_n$ y B ≡ $b_1 \ldots b_m$. Se desea encontrar la mayor (en el sentido de tener más elementos) secuencia C ≡ $c_1 \ldots c_p$ que cumpla ser una subsecuencia no contigua de A y ser una subsecuencia no contigua de B.

    Por ejemplo, dadas las secuencias de caracteres CGATAATTGAGA y GTTCCTAATA, una solución al problema podría ser la secuencia GTAATA, de seis caracteres de longitud (se muestran subrayados los caracteres que forman parte de la subsecuencia).

    Nota: Este problema se suele denominar **subsecuencia común maximal**, y aparece en muchos contextos distintos, por ejemplo:

    - Para obtener un índice de similitud entre dos documentos. El tamaño de la subsecuencia solución puede utilizarse para medir el parecido (las partes comunes) entre documentos.
    - Para crear ficheros de actualización de programas (más conocidos como *patches*), en los que sólo se incluya las partes que han cambiado respecto al programa original.
    - En ingeniería genética, para identificar genes similares entre especies distintas o mutaciones sobre una misma especie.

35. Se desean empaquetar n ficheros de tamaños $a_1 \ldots a_n$, para formar un único fichero que los englobe. El sistema operativo proporciona un comando para empaquetar ficheros, pero desafortunadamente sólo permite empaquetar dos ficheros a la vez. Se usará la notación **[A,B]** para representar el fichero resultante de empaquetar los ficheros **A** y **B** utilizando esta herramienta.

    Diseñar un algoritmo que implemente la solución más eficiente para empaquetar n ficheros, teniendo en cuenta que el tiempo que se tarda en empaquetar dos ficheros es igual a la suma de sus tamaños, y que el tamaño del fichero resultante también es igual a la suma de esos tamaños.

    Por ejemplo, si se tienen 3 ficheros, **A**, **B** y **C**, de tamaños 3, 1 y 5, el empaquetamiento **[A [B C]]** tardará 15 unidades de tiempo (6 de empaquetar **B** y **C** mas 9 de empaquetar el resultado con **A**), mientras que el empaquetamiento **[[A B] C]** tardaría 13 unidades de tiempo.

    Nota: El empaquetamiento debe respetar el orden original de los ficheros. Por ejemplo, **[C [A B]]** no sería un empaquetamiento válido.

Algoritmos de búsqueda y ordenación

36. El algoritmo de búsqueda ternaria es similar a la búsqueda binaria pero se diferencia en que en cada etapa el subvector se divide en **tres** partes iguales (en lugar de dos) y se examinan los dos elementos que separan estas tres partes para decidir si se ha encontrado el valor o, en caso contrario, elegir una de estas tres partes para continuar la búsqueda en ella.

    Analizar cuál sería la complejidad de este algoritmo para las operaciones en que interviene un elemento del vector (se puede suponer que el tamaño del vector es una potencia de tres) y comprobar si mejora o no el orden de la búsqueda binaria.

37. Dado un vector ordenado de enteros, $v_1 \ldots v_n$, donde no existen elementos repetidos, crear un algoritmo que determine si existe algún elemento igual a su índice (es decir, si $v_i = i$) en un tiempo mejor que $O(n)$.

Edificio de Tecnologías de la Información | Campus Miguel Delibes Ctra. del Cementerio s/n | Tel.: (983) 42 36 70 Fax: (983) 42 36 71

38. ☼ Se dispone de una matriz cuadrada, M, de *n* x *n* elementos con la propiedad de que cada fila y cada columna de la matriz están ordenadas de menor a mayor (es decir, $\forall i,j : \mathrm{M}[i,j] < \mathrm{M}[i+1,j]$ y $\forall i,j : \mathrm{M}[i,j] < \mathrm{M}[i,j+1]$).

    Diseñar un algoritmo que permita detectar si la matriz contiene un determinado valor, *x*. El objetivo es conseguir un algoritmo lo más eficiente posible.

39. Se dispone de un vector que contiene *n* elementos, todos ellos iguales. ¿Cuál sería la eficiencia del algoritmo de ordenación rápida (en sus distintas variantes) aplicado a éste vector?

40. Encontrar el orden de los enteros 1,2,3,4,5 que provoca el peor caso para el algoritmo de ordenación rápida con el segundo método de partición y cuyo elemento pivote es el medio.

41. Se desea ordenar en un tiempo de $O(n)$ una estructura de datos que almacena *n* elementos representando uno de tres colores posibles (rojo, blanco o azul). Los colores pueden compararse entre sí, siendo rojo < blanco < azul.
    - Crear un algoritmo que resuelva el problema suponiendo que la estructura es un vector.
    - Crear un algoritmo que resuelva el problema suponiendo que la estructura es un TAD donde las únicas operaciones posibles son **color(i)**, que devuelve el color i-ésimo e **intercambia(i,j)**, que intercambia los colores i y j.

42. ☼ Dado un vector A[1..*n*] de enteros (todos distintos) y un valor entero *x*, diseñar un algoritmo que imprima **todos** los pares (*i*,*j*) que cumplan que $i < j$ y $A[i]*A[j] = x$ en un tiempo $O(n \log n)$.

Identificación de TADs

En los siguientes problemas se va a plantear una tarea que se desea informatizar. El objetivo es analizar cuáles son las operaciones básicas y encontrar si alguno de los TADs básicos estudiados incorpora las operaciones básicas necesarias para llevar a cabo esas tareas. Si ninguno tiene todas las operaciones necesarias, se debe indicar cuál es el TAD que más se aproxima y que operaciones adicionales serían necesarias:

43. ☼ Un delegado de clase desea informatizar la lista de compañeros apuntados al viaje de fin de curso. Las operaciones que va a llevar a cabo con esa información son apuntar, desapuntar, responder a preguntas de si alguien está apuntado o no (los alumnos se identifican por su nombre) y sacar de vez en cuando un listado con las personas apuntadas (sin un orden concreto).

    Si ahora se exige que el listado sea por orden alfabético. ¿Cambia eso la respuesta del ejercicio?

44. ☼ Para otorgar una beca, un administrativo recibe una lista (sin duplicados) de datos de personas que pueden optar a ella. De cada persona se sabe su nombre, teléfono y calificación. La tarea a realizar consiste en introducir los datos en un programa e ir llamando a cada persona (en orden de mayor a menor calificación) para preguntarla si cumple unos determinados requisitos. La primera persona en cumplirlos recibe la beca (y eso finaliza la tarea).

Edificio de Tecnologías de la Información | Campus Miguel Delibes Ctra. del Cementerio s/n | Tel.: (983) 42 36 70 Fax: (983) 42 36 71

45. ☼ Se desea programar una agenda telefónica que muestre, en orden alfabético, los nombres de cada persona junto con su número de teléfono y permita obtener el teléfono dado el nombre de una persona, añadir una persona junto con su teléfono y borrar una persona. Sólo se permite un teléfono por persona.

    Si se permiten varios teléfonos por persona. ¿Cambia eso la respuesta del ejercicio?

Vectores y listas enlazadas

46. Se desean implementar operaciones de pila sobre un conjunto de datos almacenados en una estructura de tipo cola. Suponiendo que la cola tiene una implementación óptima y que el espacio adicional disponible es $O(1)$, crear algoritmos que implementen las operaciones de añadir y quitar elementos de pila indicando su complejidad.

47. El lenguaje de programación FORTH, que se hizo popular a finales de los años 70, disponía únicamente de dos tipos de datos: Enteros y estructuras tipo pila. Las operaciones que se podían realizar sobre las pilas eran la inserción de un elemento (que pasaba a ser la cabeza de la pila), la extracción de la cabeza de la pila y la consulta del número de valores almacenados. Todas estas operaciones se realizaban en un tiempo $O(1)$.

    Analizar cuál sería el coste temporal y espacial de implementar el algoritmo de ordenación por fusión en este lenguaje. El algoritmo partiría de una pila con n enteros almacenados y devolvería la misma pila pero con los valores organizados de manera que al extraer todos los elementos de la pila formasen una sucesión creciente. Se pueden usar todas las variables auxiliares que se requieran, teniendo en cuenta, por supuesto, que sólo pueden ser de tipo entero o pila.

48. Dar un algoritmo para imprimir el contenido de una lista simplemente enlazada en orden inverso. El algoritmo deberá tener un tiempo de ejecución de $O(n)$, utilizando un espacio adicional de $O(1)$.

    Nota: A pesar de que la operación pedida es de lectura, se permite que el algoritmo vaya modificando la estructura de datos, siempre y cuando al finalizar la deje como estaba.

49. Describa cómo almacenar matrices triangulares (inferiores y superiores) y k-diagonales de manera que no se malgaste espacio y se pueda seguir accediendo a los elementos en $O(1)$.

50. Proporcionar un algoritmo que realice la búsqueda binaria sobre una lista enlazada ordenada. Analizar su tiempo de ejecución.

51. ☼ Adaptar el algoritmo de ordenación por fusión para que ordene una lista simplemente enlazada. Analizar su complejidad temporal y espacial.

52. Dado el tipo abstracto de datos pila, dar un algoritmo para obtener el valor del menor elemento almacenado (la pila debe quedar como estaba). Se pueden usar otras pilas para almacenar datos temporalmente.

53. ✍ ¿Cuál sería la representación más eficiente para un TAD en el que las operaciones que se van a realizar son acceso al máximo e inserción por valor? (Igual número de cada una de ellas)

Edificio de Tecnologías de la Información | Campus Miguel Delibes, Ctra. del Cementerio s/n | Tel.: (983) 42 36 70, Fax: (983) 42 36 71

Implementación de TADs

54. ☼ **TAD BPila**. Se desea almacenar una conjunto de referencias a datos de alumnos, los cuales disponen de dos campos clave (DNI y NIA) sobre los que se van a realizar búsquedas. La estructura de datos que los almacena debe soportar las operaciones típicas de una pila y además búsquedas por cada uno de los campos clave. En la siguiente lista se describen las operaciones:

Nombre	Orden	Parámetro	Resultado
meter(d)	$O(\log n)$	Referencia a un dato	Inserta el dato en la estructura
cima	$O(1)$	Ninguno	Devuelve el último dato insertado
sacar	$O(\log n)$	Ninguno	Elimina el último dato insertado
buscar_dni(c)	$O(\log n)$	Un valor de tipo DNI	Una referencia al dato con clave DNI igual al proporcionado o una referencia nula si no existe
buscar_nia(c)	$O(\log n)$	Un valor de tipo NIA	Una referencia al dato con clave NIA igual al proporcionado o una referencia nula si no existe

Diseñar una estructurada de datos, usando una o varias de las representaciones contempladas en la asignatura, que pueda realizar esas operaciones con el orden deseado. ¿Es posible conseguir que todas estas operaciones se realicen en tiempo constante, $O(1)$?

55. ☼ **TAD MaxPila**. Este TAD contiene las mismas operaciones que una pila (meter, sacar, cima) junto con una operación adicional, **máximo**, que devuelve el elemento cuyo campo clave tiene el mayor valor de todos los existentes en la pila. Diseñe una implementación que pueda realizar las cuatro operaciones en tiempo constante, $O(1)$. ¿Se podría conseguir lo mismo con una cola?

56. ☼ **TAD Cola-Prioridad-Mediana**. Se desea implementar un TAD con las mismas operaciones que el TAD Cola de Prioridad pero en lugar de acceder y extraer el mínimo se desea acceder y extraer la **mediana** del conjunto de los elementos (La mediana se define como el valor del elemento que ocupa la posición **central** cuando les disponemos de forma **ordenada**. Si existen $2n+1$ elementos (número impar), la mediana será el elemento en posición $n+1$, y si existen $2n$ elementos (número par), la mediana será el elemento en posición n). Se desea que las operaciones tengan los siguientes órdenes:

Operación	Orden
Acceso mediana	$O(1)$
Extracción mediana	$O(\log n)$
Inserción	$O(\log n)$
Creación a partir de vector	$O(n)$

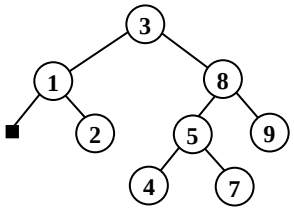
No es necesario implementarlo en código, con describir el tipo de representación que se usará y describir gráficamente como se realiza cada operación es suficiente.

57. ✍ Encontrar la implementación más eficiente posible para un TAD que represente un conjunto de elementos en el que se pueden realizar las operaciones de añadir un elemento al conjunto, comprobar la pertenencia de un elemento al conjunto y comprobar si un conjunto es igual a otro. Los elementos pueden compararse entre sí mediante el operador de igualdad.

58. En una estructura de datos se encuentran almacenados n elementos con claves numéricas, y los valores de esas n claves son, en orden ascendente, $c_1 \ldots c_n$. Se sabe que a la estructura de datos no se le van a añadir ni eliminar elementos, y que la única operación que se va a realizar sobre ella es buscar el elemento con un valor concreto de la clave (se puede suponer que todas las búsquedas son exitosas). Se conoce, además, cual es la frecuencia con que se va a preguntar sobre cada valor de la clave, y se dispone del vector que almacena esa información, $f_1 \ldots f_n$. (Nota: Se cumple que $f_1+f_2+\ldots+f_n = 1$).

    - Suponiendo que la estructura de datos es un árbol binario de búsqueda, crear un algoritmo que indique cual es la organización óptima de ese árbol, en el sentido que minimice, a largo plazo, el tiempo que se tarda en realizar búsquedas que obedezcan a la distribución de frecuencia proporcionada.
    - Indicar otra posible estructura de datos para este problema.

59. ☼ Se dispone de un árbol AVL con el contenido siguiente:



    Dibuje el árbol resultante tras el borrado del valor **2**, indicando el tipo de rotaciones que se han realizado durante el proceso.

60. ☼ Encontrar la representación más eficiente para un TAD que representa un conjunto de n enteros entre 0 y m-1 y donde se desean realizar las operaciones siguientes:

    - insertar(x): Incluye el entero $x \in [0..m\text{-}1]$ en el conjunto.
    - borrar(x): Excluye el entero x del conjunto, si pertenecía a él.
    - sucesor(x): Devuelve el menor elemento del conjunto mayor que x, o -1 si x es mayor que todos.

61. ☼ Se dispone de una red de agentes secretos, numerados del 1 al n, donde cualquier agente puede comunicarse con cualquier otro pero existe un riesgo de que la comunicación pueda ser interceptada. En concreto, se conoce la matriz $p_{i\,j}$ que indica la probabilidad de que se intercepte un mensaje enviado por el agente i al agente j.

    Supongamos que el agente i quiere enviar un mensaje al agente j. Si se lo envía directamente existe la probabilidad $p_{i\,j}$ de que lo intercepten. Es posible que usando otros agentes como intermediarios (por ejemplo enviándoselo al agente k, éste al agente m, y éste finalmente a j), la probabilidad de interceptación sea menor (en el ejemplo debería cumplirse que $p_{i\,k} + p_{km} + p_{m\,j} < p_{i\,j}$)

    a) Crear un algoritmo que reciba como entrada los índices de dos agentes, i y j, y devuelva como resultado el método más seguro para que i le envíe un mensaje a j, indicando, si los hay, que agentes (y en qué orden) se usan como intermediarios.

    b) Supongamos que el agente i quiere enviar un mensaje a **todos** los demás. Crear un algoritmo que indique la manera en que se debe propagar el mensaje (para cada agente se debe indicar de quien recibe el mensaje) de forma que se minimice la probabilidad de que sea interceptado.