Examen de Teoría de la Programación

E. U. ING. TEC. EN INFORMÁTICA DE OVIEDO

Final Febrero – Curso 2005-2006 14 de febrero de 2006

DNI _____ Nombre _____ Apellidos _____

Titulación: ☐ Gestión ☐ Sistemas

Grupo al que asistes en teoría: _____

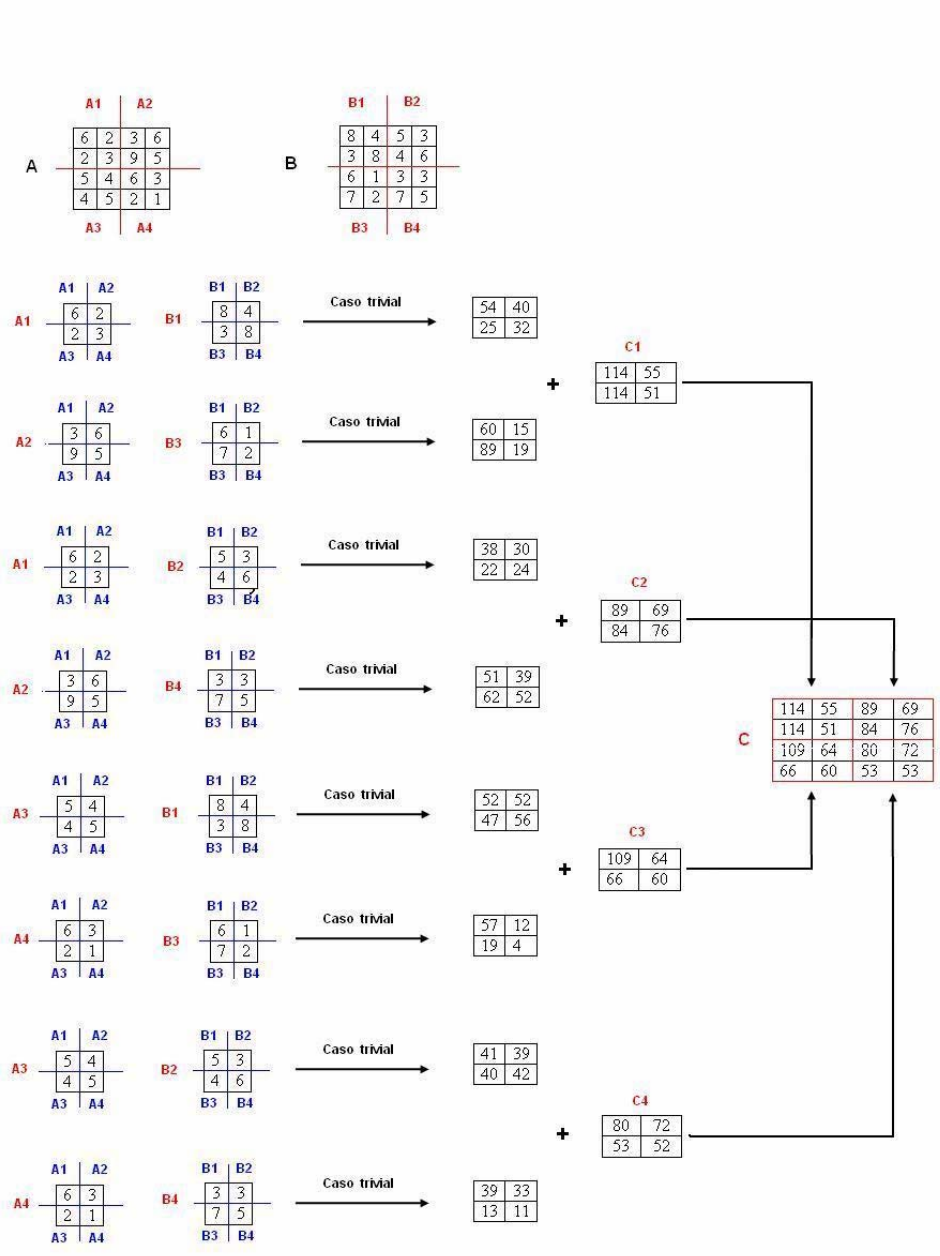
3. (1,25 puntos) La multiplicación de matrices cuadradas se puede resolver siguiendo la técnica divide y vencerás.

a) En esta técnica se diferencian tres pasos: descomposición del problema, resolución de los problemas elementales, combinación de los resultados para obtener la solución al problema inicial. Especificar que hay que hacer en cada uno de estos tres pasos en la resolución de la multiplicación de matrices cuadradas.

b) Cuál es la complejidad de esta solución por Divide y Vencerás.

(Problema expuesto en el grupo A)

a)



Descomposición del problema:
Dividimos cada una de las matrices en cuatro cuadrantes. Cada cuadrante constituye una matriz de tamaño n/2 que podemos volver a dividir hasta llegar a matrices de 2x2 que consideramos como el caso elemental.
Resolución de problemas elementales
Cuando llegamos al caso elemental aplicamos el algoritmo clásico de multiplicación de matrices:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

Combinación de resultados:
Se multiplican los cuadrantes (submatrices) como si fueran elementos y se compone la matriz resultado con los resultados de los cuatro cuadrantes.
Consideremos matrices de orden 2x.

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

Donde cada letra representa una submatriz.
La forma de multiplicar estos cuadrantes es la siguiente:
C11= A11·B11 + A12·B21
C12= A11·B12 + A12·B22
C21= A21·B11 + A22·B21
C22= A21·B12 + A22·B22

b)
Es un caso de divide y vencerás con división
a= 8
b= 2
K= 2
$O(n^{\log_2 8}) = O(n^3)$

4. (1,25 puntos) El problema del *play-off* de baloncesto consiste en dos equipos *A* y *B* que disputan una eliminatoria de la fase final de la liga de baloncesto. Jugarán 2*n*-1 partidos y el ganador será el primer equipo que consiga *n* victorias. El equipo *A* tiene una probabilidad constante *p* de ganar un partido, mientras que *B* tiene una probabilidad *q* de ganar (*q*= 1 – *p*). Queremos conocer la probabilidad de que el equipo *A* gane el *play-off*, y que esto pueda calcularse tanto antes de empezar la eliminatoria, como una vez jugados varios partidos.

La función que proporciona la solución al problema es la siguiente:
¡Error! No se pueden crear objetos modificando códigos de campo.
Donde *P(i,j)* se define, como la probabilidad de que *A* gane la eliminatoria cuando le quedan por ganar *i* partidos a *A* y *j* partidos a *B*.

Utilizaremos la técnica de Programación Dinámica para obtener la solución al problema. Suponemos que para ganar la eliminatoria un equipo debe de ganar tres partidos.

a) **Justificar, en base a la complejidad temporal, por qué es conveniente resolver este problema por *Programación Dinámica* en vez de por *Divide y Vencerás*.**
b) **Representar gráficamente el array necesario para implementar el algoritmo, señalando el significado de filas y columnas (no hace falta rellenarlo). Tener en cuenta que queremos calcular la probabilidad antes de comenzar la eliminatoria.**
c) **Representar gráficamente que celdas son necesarias para calcular el valor de la celda (2,1).**
d) **Explicar que valores podemos rellenar de forma directa en la tabla.**
e) **Plantear el orden correcto para rellenar la tabla.**

a)
Complejidad temporal de Divide y Vencerás
Es un divide y vencerás con sustracción
a= 2
b= 1 (consideramos n= i+j)
K= 0
a>1 → $O(a^{n/b})$ → $O(2^n)$

Complejidad del método de programación dinámica
Simplemente hay que tener en cuenta que para rellenar la tabla necesitamos dos bucles anidados, por tanto $O(n^2)$

El método más conveniente es programación dinámica ya que su complejidad cuadrática es mucho mejor que la complejidad exponencial de divide y vencerás.

b)

Hay que tener en cuenta que en el enunciado de la pregunta se dice que “para ganar la eliminatoria un equipo debe de ganar tres partidos”.

i \ j	0	1	2	3
0				
1				
2				
3				

c)

i \ j	0	1	2	3
0				
1				
2				
3				

	Celda que hay que rellenar
	Celdas de las que depende directamente
	Celdas que hay que rellenar previamente para obtener el valor de la celda pedida.

d)

i \ j	0	1	2	3
0		1	1	1
1	0			
2	0			
3	0			

Basándonos en la función: si i= 0 → P(i,j)= 1, si j= 0 → P(i,j)= 0

e)

La tabla se puede rellenar de diversas formas:

- Por columnas, de arriba abajo y de izquierda a derecha.
- Por filas, de izquierda a derecha y de arriba abajo.
- Por diagonales, empezando por la parte superior izquierda.

5. (1,75 puntos) Oscar quiere regalarle a su novia el CD de música perfecto, el día de San Valentín. Para ello ha estado recopilando durante los 15 años que llevan juntos, todas las canciones que le gustan a su novia y les ha asignado una puntuación, con lo que dispone de una lista de ¡1300 canciones! Además, Oscar dispone del tamaño del archivo correspondiente a cada canción. Sólo quiere realizar un único CD y, lógicamente, no caben todas las canciones, para crear el mejor disco debe maximizar la puntuación total de las canciones que contiene, teniendo en cuenta que los ficheros de canciones no se pueden partir. Como Oscar tenía que estudiar para diversos exámenes de la convocatoria de febrero, lo ha dejado para última hora y debe utilizar un algoritmo que calcule las canciones que debe incluir de forma inmediata.

a) Justificar porque debe aplicar un algoritmo voraz para resolver este problema.

b) Explicar el heurístico más adecuado para este caso.

c) Escribir el código Java que permita obtener la solución para este problema mediante un algoritmo voraz.

a)

En el enunciado del problema deja claro que se debe primar el tiempo de resolución del problema, ya que dice que debe ser inmediato. El método devorador es más rápido de ramificación y poda y sobre todo que backtracking, por tanto, debemos elegirlo para resolver este problema, aunque no tengamos ningún heurístico que asegure la solución óptima.

b)

Debemos calcular para cada canción el cociente de la puntuación entre el tamaño de cada canción y ordenar las canciones de mayor a menor por este cociente para ir seleccionándolas en este orden. Como dijimos antes este heurístico no asegura la solución óptima para este problema; pero es el más adecuado.

c)

```
public Class Cancion {
  private int identificador;
  private float tamaño;
  private int puntuación;
  private float relacion;

  public int getIdentificador() { return identificador }
  public float getTamaño() { … }
  public int getPuntuacion() { … }
   public float getRelacion() { … }
   public void setRelacion(float cociente) { relacion= cociente; }
  }

public Class SeleccionCanciones {
  Cancion[] canciones;   // array con toda la información sobre las canciones
  Cancion[] cd;          // array que almacenará la solución con las canciones escogidas
  int n= 1300;           // número total de canciones

  […]

  public void crearCD() {
    float capacidadRestante = capacidadCD;
    float puntuacionTotalCD = 0;
    int contador= 0;  // índice del array donde vamos almacenando las canciones del CD

    // Calcula la relación puntuación / tamaño para todas las canciones
    calcularRelacion(canciones);

    // Ordena canciones de mayor a menor por: puntuación / tamaño
    ordenar(canciones);

    // Ya tengo el array de canciones ordenado,
    // Voy metiendo en el CD las canciones en el orden que las tengo,
    // hasta que haya una que no quepa en el espacio restante
    for (int indice=0; indice<n && capacidadRestante>=canciones[indice].getTamaño(); indice++)
{
      if (canciones[indice].getTamaño() <= capacidadRestante)
      {
        cd[contador++]= cancion[indice];

        puntuacionTotalCD += canciones[indice].getPuntuación;
        capacidadRestante-= canciones[indice].getTamaño;
      }
    }

  }
```

La condición "`capacidadRestante>=canciones[indice].getTamaño()`" del bucle for es opcional; si queremos primar el tiempo de ejecución la debemos poner, si queremos mejorar que la solución se acerque al óptimo la quitamos.

6. (1,25 puntos) El problema de la asignación de tareas consiste en asignar *n* tareas a *n* agentes, de tal forma que cada agente realiza una única tarea y cada tarea sólo puede ser realizada por un único agente. A cada agente le cuesta realizar más unas tareas que otras, luego para cada pareja *(agente, tarea)* tendremos asociado un coste. Esto se representa en una matriz. El objetivo del problema es buscar la asignación que minimice el coste total de realizar todas las tareas.

a) Completar el código Java para dar solución a este problema con la técnica de Backtracking (escribirlo en los recuadros preparados a tal efecto).

```
public class Main
{
    private int costes[][],  // almacena lo que de cuesta a cada agente realizar una tarea
            almacenEstados[],// estado actual del problema
            mejorSolucion[]; // contiene la mejor solución en cada momento
    private boolean tareasAsignadas[];// tareas ya asignadas (las que están a true)

    private int n, // Tamaño del problema.
            coste, // Almacena el coste acumulado en el cálculo de una solución.
            mejorCoste; // El mejor coste hasta el momento de una solución.

    […]

    public void ensaya(int agente)
    {
```

```
        for (int tarea = 0; tarea < n; tarea++)
        {
            if (!tareasAsignadas[tarea])
            {
                almacenEstados[agente] = tarea;
                coste += costes[tarea][agente];
                tareasAsignadas[tarea] = true;


                if (agente < (n - 1))           // no es solución
                {
                    ensaya(agente + 1);
                 }
                else        // es una solución posible
                    if (coste < mejorCoste)
                    {
                        mejorCoste = coste;
                        System.arraycopy(almacenEstados, 0, mejorSolucion, 0, n);
                    }

                coste -= costes[tarea][agente];
                tareasAsignadas[tarea] = false;
            }
         }
    } // Fin método
```

(Problema expuesto en el grupo B)

7. (1,5 puntos) Un campo está dividido en m secciones horizontales (filas) por n secciones verticales (columnas). Los saltos del *superconejo* de una sección a otra de este campo se pueden ver como un primer movimiento de p secciones en dirección horizontal o vertical, seguida de un segundo movimiento de q secciones en dirección perpendicular a la seleccionada anteriormente. (Cuando se mueve horizontalmente lo puede hacer hacia la izquierda o hacia la derecha y cuando se mueve verticalmente lo puede hacer hacia arriba o hacia abajo).
Colocamos al *superconejo* en la sección (x,y) y queremos que llegue a otra sección distinta (u,v) realizando los saltos como se describió anteriormente. Se debe buscar el camino que conlleve la cantidad mínima de saltos para lograrlo.

- **a) El grupo que ha resuelto este problema ha utilizado un desarrollo en anchura puro, sin utilizar heurísticos. Explicar por qué este método es mejor que otras técnicas vistas: backtracking o devorador.**
- **b) Escribir el esquema del recorrido en anchura que se debe emplear para encontrar la solución a este problema.**

(Problema expuesto en el grupo C)

a)
Una de las ventajas del recorrido en anchura puro es: “Si estamos en el caso de buscar sólo la primera solución y hay estados solución cerca de la raíz.” Es más rápido que otros métodos, a no ser que tengamos un buen heurístico para el método devorador.
En este problema cada nivel que profundizamos en el árbol de estados constituye un salto más para el conejo y por tanto las soluciones óptimas estarán en la parte alta del árbol, relativamente cerca de la raíz, por otra parte no disponemos de un heurístico eficiente para la técnica del devorador; por lo que el mejor planteamiento es un recorrido en anchura aunque no tengamos heurísticos que aplicar.

b)
Esquema recorrido en anchura primera solución (según aparece en el cuaderno didáctico)

```
procedure anchura(e:estado);
var c: cola; (* cola FIFO *)
begin
  Inicializa(c);
  Meter(c,e);
  while not (Vacia(c) or hay_solucion) do
    begin
      Sacar(c,e);
      Expandir(e,hijos);
      for v:= todos los estados hijos de e do
        if solucion_final(v) then hay_solucion:= true
```

```
                                else meter(c,v);
    end;
  Destruir(c);
end;
```