Semana 1 - Ejercicios de grupo

Métodos algorítmicos en resolución de problemas II
Facultad de Informática - UCM

	Nombres y apellidos de los componentes del grupo que participan	**ID Juez**
1	Jorge Areavalo	
2	Daniel Hernández Martínez	**MAR246**
3	Miguel Verdaguer	MAR285
4		

Instrucciones:

1. Para editar este documento, es necesario hacer una copia de él. Para ello:
   - Alguien del grupo inicia sesión con la cuenta de correo de la UCM (si no la ha iniciado ya) y accede a este documento.
   - Mediante la opción *Archivo* → *Hacer una copia*, hace una copia del documento en su propia unidad de *Google Drive*.
   - Abre esta copia y, mediante el botón *Compartir* (esquina superior derecha), introduce los correos de los demás miembros del grupo para que puedan participar en la edición de la copia.

2. La entrega se realiza a través del Campus Virtual. Para ello:
   - Alguien del grupo convierte este documento a PDF (dándole como nombre el número del grupo, 1.pdf, 2.pdf, etc…). Desde *Google Docs*, puede hacerse mediante la opción *Archivo* → *Descargar* → *Documento PDF*.
   - Esta misma persona sube el fichero PDF a la tarea correspondiente deol *Campus Virtual*. Solo es necesario que uno de los componentes del grupo entregue el PDF.

Nuestro fichero debe llamarse 6.pdf

8,25

Juego de tablero

El objetivo de hoy es resolver el **problema 3 Juego de tablero**, del juez automático. Ahí podéis encontrar el enunciado completo del problema.

Se dispone de un tablero cuadrado con casillas que contienen números naturales. Un recorrido válido en el juego empieza en una casilla de la última fila y llega a una casilla de la primera fila teniendo en cuenta que desde una casilla se puede avanzar a una de las tres casillas que tiene justo encima en vertical o diagonal sin salirse del tablero. La puntuación obtenida será la suma de los valores de las casillas por las que se ha pasado en el recorrido. Se desea saber cuál es el valor del recorrido válido con mayor valor y en qué casilla de la última fila comienza.

sin hacer

Solución: (Plantead aquí la recurrencia que resuelve el problema y explicad las razones por las que es mejor resolver el problema utilizando programación dinámica que una implementación recursiva sin memoria.)
Resolvemos el problema por programación dinámica ascendente, actualizando los valores desde la segunda fila teniendo el mismo coste en tiempo para ambos algoritmos, pero el coste en memoria sería distinto

Definición de qué calcula valor(i,j)

Casos recursivos:
Fila actual != N-1
actualizamos valores:

$$valor(i,j) = \begin{cases} valor(i,j) + \max(valor(i-1,j-1), valor(i-1,j), valor(i-1,j+1)) & \text{si } j > 0 \text{ y } j < n-1 \\ valor(i,j) + \max(valor(i-1,j), valor(i-1,j+1)) & \text{si } j = 0 \\ valor(i,j) + \max(valor(i-1,j-1), valor(i-1,j)) & \text{si } j = n-1 \end{cases}$$

Casos bases:
Fila actual == N-1
actualizamos valores
devolvemos el mayor valor y su índice

Definid estos casos!

2

será 0 ¿no?

Solución: (Escribid aquí las explicaciones necesarias para contar de manera comprensible la implementación del algoritmo de programación dinámica. En particular, explicad si es posible reducir el consumo de memoria y cómo hacerlo. Incluid el código y el coste justificado de la función que resuelve el problema. Extended el espacio al que haga falta.)

```
typedef struct {
  int first;
  int second;
} Solucion;

Solucion resolver(int tablero[MAXN][MAXN], int N) {
  for(int i = 1; i < N;i++) //Recorremos desde la segunda fila
          for(int j = 0; j < N;j++) { //Recorremos cada elemento de la fila
          if(j == 0) { //Si el elemento está en el borde izquierdo
                  tablero[i][j] = tablero[i][j] + max(tablero[i-1][j], tablero[i-1][j+1]);
          } else if (j == N-1) { //Si el elemento está en el borde derecho
                  tablero[i][j] = tablero[i][j] + max(tablero[i-1][j-1], tablero[i-1][j]);
          } else { //En otro caso
                  tablero[i][j] = tablero[i][j] + max(tablero[i-1][j-1], max(tablero[i-1][j], tablero[i-1][j+1]));
          }
          }
  Solucion sol = {0,0};
//Recorremos la última fila para encontrar el mayor número, esta será la casilla por la que habría que empezar
  for(int i = 0; i < N; i++)  {
          if(tablero[N-1][i] > sol.first) {
          sol.first = tablero[N-1][i];
          sol.second = i + 1;
          }
  }
  return sol;
}
```

Coste: Primero recorremos todas las casillas de la matriz con un bucle anidado, por lo que tenemos un coste cuadrático: O(n^2). A continuación recorremos la última fila de la matriz con un for, cuyo coste es lineal: O(n). El coste total es la suma del coste de ambos bucles, que da un coste total cuadrático: O(n^2 + n) = **O(n^2)**

Resolved el problema completo del juez, que ahora debe ser ya muy sencillo.
Numero de envío del juez: s31687

Aprovechando que hemos utilizado la programación dinámica ascendente ,podemos reducir el espacio que ocupa la tabla donde guardamos los resultados, ya que en la programación dinámica mejoramos el tiempo, a costa de necesitar más espacio. Este espacio lo podríamos reducir sabiendo que en una fila(i), solo necesitamos los valores de la fila anterior (i-1) , pudiendo descartar las filas anteriores. Esta solución se podría realizar guardando el valor de las fila actual en un vector en lugar de una matriz. Para ello habría que resolverlo a la vez que se lee.