ANÁLISIS Y DISEÑO DE ALGORITMOS

Vuelta atrás

Práctica 8 de laboratorio

Entrega: Hasta el domingo 14 de mayo, 23:55h. A través de Moodle

El senderista perezoso III

Un senderista está planificando una ruta de montaña haciendo uso de un plano de coordenadas $n \times m$. En cada posición (i, j) de dicho plano se representa, mediante un número natural, el grado de dificultad de visitar esa casilla. Cuanto más elevado es dicho valor más difícil es acceder a esa posición. Por ejemplo:

$$\xrightarrow{(0,0)} \begin{array}{ccccc} 1 & 3 & 5 & 1 & 1 \\ 2 & 4 & 6 & 3 & 1 \\ 1 & 2 & 9 & 7 & 1 \\ 9 & 1 & 7 & 1 & 9 \\ 1 & 3 & 7 & 5 & 1 \\ 8 & 1 & 2 & 2 & 1 \end{array} \xrightarrow{(5,4)}$$

Se pide, aplicar el **vuelta atrás** para obtener la dificultad del camino más favorable[1] que hay desde la casilla origen $(0, 0)$ hasta la casilla destino $(n - 1, m - 1)$ asumiendo que sólo son válidos tres tipos de movimientos desde una casilla cualquiera (i, j):

1. derecha: $(i, j + 1)$,
2. abajo: $(i + 1, j)$,
3. abajo y derecha (diagonal): $(i + 1, j + 1)$.

Como es evidente, los movimientos que llevan al exterior del mapa no son válidos.

1. **Nombre del programa, opciones y sintaxis de la orden.**

   El programa a realizar se debe llamar **caminante_BT**. La orden tendrá la siguiente sintaxis:

   caminante_BT [-p] -f `fichero_entrada`

   En el siguiente apartado se describe el significado de las opciones.

2. **Salida del programa y descripción de las opciones:**

   La salida del programa (véase los ejemplos a continuación) será la dificultad del mejor camino existente y, en la línea siguiente, el tiempo de proceso[2], en milisegundos (ms), utilizado para encontrar la solución. Si además se incorpora la opción [-p] se mostrará a continuación un camino cuya dificultad coincida con la solución obtenida. Este camino se indicará de dos maneras:

   - Mediante una lista de pares (x, y) cuyo inicio y fin son, respectivamente, $(0, 0)$ y $(n-1, m-1)$, tal y como se ha hecho en prácticas anteriores.

[1] Definimos la dificultad de un camino como la suma de los grados de dificultad de las casillas que lo componen. Por lo tanto, la dificultad de un camino compuesto por una única casilla (origen y destino coinciden), es la dificultad de esa casilla.

[2] Se ha publicado a través de Moodle un pequeño programa en C++ (compatible con el estándar 11 del lenguaje) que muestra cómo obtener el tiempo de procesador utilizado por un fragmento de código.

- Mediante una representación sobre unos ejes de dos dimensiones en donde la casilla que se visita contendrá el carácter 'X' y la que no se visita un punto ('.'). Entre las casillas no hay que añadir ningún carácter delimitador, únicamente un salto de línea al finalizar cada fila de la matriz.

En cuanto a la opción -f, se utiliza para suministrar el nombre del fichero donde está el mapa a resolver (véase siguiente apartado)

3. **Entrada al programa**

   El mapa se suministrará codificado en un fichero de texto cuyo nombre se recogerá a través de la opción -f. Su formato y contenido será:

   - Línea 1 del fichero: valores n y m separados mediante un único espacio en blanco.
   - Línea 2 (y siguientes): m números naturales que componen la dificultad de la primera fila (y siguientes) del mapa, separados mediante un único espacio en blanco

   por tanto, el fichero contendrá $n + 1$ líneas que finalizarán con un salto de línea, salvo en todo caso, la última línea.

   A través de *Moodle* se puede descargar un archivo comprimido con varios ejemplos, entre ellos está `1.map` y `2.map` utilizados a continuación para describir el formato de la salida.

4. **Formato de salida. Ejemplos de ejecución.**

   Sean el fichero `6.map` y `1.map` disponibles a través de Moodle en las secciones de prácticas anteriores.

   A continuación se muestra posibles formas de utilizar en la terminal la orden descrita y la salida que el programa debe mostrar. Es imprescindible ceñirse al formato y texto de salida mostrado, incluso en lo que se refiere a los saltos de línea o carácter separador, que, de existir, en todos los casos es el espacio en blanco.

   Antes de mostrar el camino se debe incorporar una línea vacía. No debe haber más líneas vacías, es decir, a lo sumo una. La última línea debe terminar con un salto de línea (y sólo uno). En ningún caso debe añadirse texto o valores adicionales.

$caminante_BT -f 6.map
```
Backtracking : 346
CPU elapsed time: 58 ms
```

$caminante_BT -p -f 6.map
```
Backtracking : 346
CPU elapsed time: 58 ms

A possible path:
(0,0)(1,0)(2,1)(3,2)(3,3)(4,4)(5,5)(6,6)(7,6)(8,6)
(9,6)(10,6)(11,7)(12,8)(13,8)(14,9)
X.........
X.........
.X........
..XX......
....X.....
.....X....
......X...
......X...
......X...
......X...
......X...
.......X..
........X.
........X.
.........X
```

$caminante_BT -f 1.map
```
Backtracking : 10
CPU elapsed time: 0 ms
```

$caminante_BT -f 1.map -p
```
Backtracking : 10
CPU elapsed time: 0 ms

A possible path:
(0,0)
X
```

5. **Normas para la entrega.**

   a) Se debe entregar el código fuente y un *Makefile* para obtener el ejecutable. No hay que entregar nada más, en ningún caso se entregarán ficheros de test.

   b) Todos los ficheros que se entregan deben contener el nombre del autor y su DNI (o NIE) en su primera línea (entre comentarios para que no afecte a la compilación).

   c) Se comprimirán en un archivo *.tar.gz* o equivalente cuyo nombre será el DNI (o NIE) del alumno. Por ejemplo: 123456789A.tgz

   d) En el archivo comprimido no debe existir subcarpetas, es decir, al extraer su contenido no deben crearse subcarpetas.

   e) La práctica hay que subirla a *Moodle* respetando las fechas expuestas en el encabezado de este enunciado.