ANÁLISIS Y DISEÑO DE ALGORITMOS

Práctica 6 de laboratorio

Esta práctica ocupa dos entregas:

Entrega 1: Ambas versiones recursivas y gestión de argumentos.
Entrega 2: Práctica completa.

Entrega: Respectivamente, hasta los días 30 de marzo y 6 de abril, 23:55h.
A través de Moodle

El problema del laberinto I

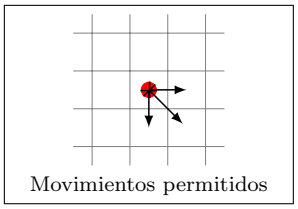
Se dispone de una cuadrícula $n \times m$ de valores $\{0, 1\}$ que representa un laberinto. Un valor 0 en una casilla cualquiera de la cuadrícula indica una posición inaccesible; por el contrario, con el valor 1 se simbolizan las casillas accesibles. Por ejemplo:

$\xrightarrow{(0,0)}$

1	1	0	0	1
1	1	1	1	1
0	1	1	0	0
1	1	0	1	1
1	1	1	0	0
0	0	0	1	1

$\xrightarrow{(5,4)}$

Un laberinto 6×5



Se pide aplicar el método *divide y vencerás* y su extensión a *programación dinámica* para calcular la longitud[1] del camino más corto que partiendo del origen $(0, 0)$ conduce al destino $(n-1, m-1)$ asumiendo solo tres posibles 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, tampoco son válidos los movimientos que llevan al exterior del laberinto ni los que conducen a casillas inaccesibles. Para resolver este ejercicio se debe implementar los siguientes algoritmos:[2]

1. Recursivo sin almacén (ineficiente —también llamada versión ingenua o *naive*—)
2. Recursivo con almacén (memoización)
3. Iterativo con almacén que hace uso de una tabla para almacenar los resultados intermedios.
4. Iterativo con almacén con complejidad espacial mejorada.
5. Por último, deberá obtenerse un camino de salida del laberinto con esa longitud.[3]

[1] Definimos *longitud del camino* como el número de casillas que lo componen. Un camino con origen y destino en la misma casilla tiene longitud 1.

[2] Como es lógico, las soluciones de todos los algoritmos que muestran la longitud del camino más corto deben coincidir.

[3] Puede existir más de un camino de salida con longitud mínima; en tal caso se mostrará uno cualquiera de esa longitud.

1. **Nombres, en el código fuente, de algunas funciones importantes.**

   Para una correcta identificación en el código de las cinco funciones que implementan los algoritmos mencionados en el apartado anterior, deberán ser nombradas de manera preestablecida. Siguiendo el mismo orden en el que se han enumerado, los nombres de las funciones de `C++` deben ser: `maze_naive`; `maze_memo`; `maze_it_matrix`; `maze_it_vector` y `maze_parser`, respectivamente.

   Se deja total libertad para añadir los parámetros que se consideren y también para escoger el tipo de datos de retorno. También se deja libertad para escoger las estructuras de datos que consideres, librerías, elementos del lenguaje, etc.

   La función `maze_naive` debe estar implementada de forma correcta. En cuanto a las restantes, para valorar esta práctica no es imprescindible que estén en el código, pero si no están (o están con otro nombre) entonces se considerará que no han sido desarrolladas, con la consiguiente merma en su calificación.

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

   El programa a realizar se debe llamar `maze`. La orden tendrá la siguiente sintaxis:

   `maze` *`[-t] [--p2D] [--ignore-naive]`* `-f` *`fichero_entrada`*

   - La opción `-f`, la única de uso obligado, se utiliza para suministrar el nombre del fichero donde está la instancia del problema a resolver. En el caso de que no se suministre o se produzca algún tipo de error al tratar de abrirlo (no existe, sin permisos para abrirlo, etc.) se advertirá con un mensaje de error. No es necesario controlar posibles errores en el contenido del fichero de entrada ya que siempre se ajustará fielmente al formato establecido, que se describe en el apartado "Entrada al programa".
   - Las opciones `--ignore-naive`; `--p2D` y `-t`, que se describen en el siguiente apartado, no son excluyentes entre sí, ninguna de ellas. Además podrán aparecer en cualquier orden.
   - En el caso de que se haga uso de la orden con una sintaxis distinta a la descrita se emitirá un mensaje de error que advierta de ello y a continuación se mostrará la sintaxis correcta. Se deben considerar en este caso únicamente las opciones inexistentes; la duplicidad de opciones puede ser ignorada y tratada por tanto como si la opción se hubiera escrito solo una vez. En el caso de que se repita la opción `-f` se tomará su última aparición ignorando las previas. No es necesario indicar todos los errores sintácticos que pueda contener la orden, basta con hacerlo solo con el primer error que se detecte.
   - Para todos los mensajes de error, incluso aquel que informa del uso apropiado de la orden, debe utilizarse la salida estándar de error, es decir, `cerr`.
   - Ante cualquier circunstancia de error en las opciones, el programa deberá terminar advirtiéndolo según se acaba de describir.

3. **Salida del programa y descripción de las opciones.**

   En todas las formas posibles de utilizar la mencionada orden, la salida del programa será siempre a la salida estándar,[4] es decir, al terminal y consistirá, en primer lugar y en la primera línea: la solución obtenida mediante los cuatro algoritmos anteriormente citados: recursivo sin almacén, memoización, iterativo con tabla e iterativo con complejidad espacial mejorada. Cada uno de los cuatro valores se mostrarán en la misma línea, separados por un espacio en blanco y siguiendo ese orden.

   Si no existe camino de salida, el valor a mostrar para cada algoritmo realizado será 0.

[4] Véanse los ejemplos de ejecución y los ficheros de *test* suministrados para conocer los detalles de la sintaxis de salida.

No obstante, si se incorpora a la orden la opción `--ignore-naive` se deberá sustituir el valor que corresponde a la solución recursiva sin almacén por el carácter guion ('`-`');[5] por lo tanto, en este caso, la primera línea contendrá, en primer lugar dicho carácter y a continuación, los tres resultados restantes.

Si alguno de los cuatro algoritmos no se implementa, se mostrará en su lugar correspondiente el carácter '?'. En ningún caso se usará el valor calculado por otra función (o cualquier otro), algo que será interpretado como un acto de mala fe y se penalizará con severidad.

Si se hace uso de la opción `--p2D` (y solo en este caso), se mostrará, a partir de la segunda línea de la salida del programa, un camino de salida del laberinto de longitud mínima. Para ello, se utilizará un formato de dos dimensiones, mostrando la misma matriz de entrada que representa el laberinto pero sustituyendo los unos de todas las casillas que componen el camino por el carácter '*'. Las demás casillas se dejarán con su valor original. No debe haber ningún carácter separador entre todos los valores de cada fila (es decir, todos los elementos de una misma fila se mostrarán juntos). Al final de cada fila de la matriz debe haber un único salto de línea. Si el laberinto no tuviera salida se mostrá únicamente el valor 0 en lugar de la mencionada matriz.

Por otra parte, si se hace uso de la opción `-t` (y solo en este caso), se mostrará, a continuación (siguiente línea) de cualquier salida previa, las tablas que almacenan los resultados intermedios en las versiones con memoización e iterativa con matriz (en este orden y con los encabezados que pueden verse en los ejemplos de salida). Los elementos de cada casilla de estas matrices pueden ser de tres tipos:

- Un valor numérico. Indica la solución para el subproblema representado en esa casilla.
- El carácter guión ('`-`'). Indica que no ha sido necesario resolver ese subproblema.
- El carácter '`X`' (en mayúsculas). Indica que se ha intentado resolver ese subproblema pero ha resultado ser una posición inaccesible (bien porque está cerrada o bien porque no es alcanzable).

El separador entre valores de una misma fila será el espacio en blanco, por lo tanto no es necesario en este caso que las columnas estén visualmente bien estructuradas. Si aun así, quieres que las columnas queden alineadas, puedes suponer que, con los ejemplos que se van a probar, estos valores numéricos nunca alcanzarán magnitudes de más de tres dígitos.

4. **Entrada al programa.**

El laberinto $n \times m$ 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 valores {0,1} que componen la primera fila (y siguientes) del laberinto, 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 (`??.maze`), junto con las soluciones (`??.maze.sol`) para el caso de que se haga uso de todas las opciones (salvo, en algunos casos, la opción `--ignore-naive`). Es importante tener en cuenta que el hecho de que el programa realizado obtenga la salida correcta para todos estos ejemplos no es garantía de que la obtenga para otros.

[5] Por su elevado coste computacional (evidentemente el programa tampoco deberá hacer la llamada a esta función).

5. **Formato de salida. Ejemplos de ejecución.**

   De entre los ejemplos de entrada publicados, está el fichero 02.maze cuyo contenido es el laberinto (6×5) utilizado en la presentación de este problema y el fichero 03.maze, cuyo contenido es un laberinto (también 6×5) que no tiene camino de salida desde el origen.

   A continuación se muestran posibles formas de utilizar en la terminal la orden descrita y la salida que el programa debe mostrar (en los ejemplos, las salidas `cout` y `cerr` están dirigidas a la terminal —siguiendo el comportamiento predeterminado—).

```
 $maze -f 02.maze
 7 7 7 7

$maze --ignore-naive -f 02.maze --p2D
- 7 7 7
*1001
*1111
0*100
1*011
11*00
000**

$maze -t -f 02.maze --p2D
7 7 7 7
*1001
*1111
0*100
1*011
11*00
000**
Memoization table:
  1  2  -  -  -
  2  2  -  -  -
  X  3  -  -  -
  X  4  X  -  -
  X  5  5  X  X
  -  -  X  6  7
Iterative table:
  1  2  X  X  X
  2  2  3  4  5
  X  3  3  X  X
  X  4  X  4  5
  X  5  5  X  X
  X  X  X  6  7
```

```
$maze -p -f 03.maze -t --p2D -t
0 0 0 0
0
Memoization table:
  -  -  -  -  -
  -  -  -  -  -
  -  -  -  -  -
  -  X  X  -  -
  -  X  X  X  X
  -  -  X  X  X
Iterative table:
  1  2  X  X  X
  2  2  3  4  5
  X  3  X  X  X
  X  X  X  X  X
  X  X  X  X  X
  X  X  X  X  X

 *ALGUNOS EJEMPLOS DE SITUACIONES DE ERROR:*

 $maze -f
 ERROR: missing filename.
 Usage:
 maze  [--p2D] [-t] [--ignore-naive] -f file


 $maze
 Usage:
 maze  [--p2D] [-t] [--ignore-naive] -f file


 $maze -f -t
 ERROR: can't open file: -t.
 Usage:
 maze  [--p2D] [-t] [--ignore-naive] -f file


 $maze -f 0.problem -t -a -b
 ERROR: unknown option -a.
 Usage:
 maze  [--p2D] [-t] [--ignore-naive] -f file


 $maze -f anonexistentfile -t
 ERROR: can't open file: anonexistentfile.
 Usage:
 maze  [--p2D] [-t] [--ignore-naive] -f file
```

6. **Normas para la entrega.**

ATENCIÓN: Estas normas son de obligado cumplimiento para que esta práctica sea evaluada.

Es imprescindible ceñirse al formato y texto de salida descrito, incluso en lo que se refiere a los saltos de línea o carácter separador, que en todos los casos es el espacio en blanco.
En ningún caso debe añadirse texto, valores o saltos de línea adicionales.
Con respecto a los ejemplos publicados, ten en cuenta que los ficheros de entrada pueden estar nombrados de cualquier otra forma; y pueden ser distintos.
Si estos requisitos no se cumplen, es posible que falle una de las fases de la evaluación de este trabajo, que se hace de manera automática; en tal caso la práctica tampoco será evaluada.

a) La función `maze_naive` debe estar implementada de forma correcta. Las demás funciones deben resolver correctamente los ejemplos básicos (como `01.maze` y `02.maze`); si esto no ocurre, trátalas como si no se hubieran implementado (escribiendo el carácter "?" en el lugar de la salida que corresponda).

b) El programa debe realizarse en un único archivo fuente con nombre `maze.cc`.

c) Se debe entregar únicamente los ficheros `maze.cc` y `makefile` (para generar el archivo ejecutable, `maze`, a través de la orden `make` sin añadir nada más). **Sigue escrupulosamente los nombres de ficheros y funciones que se citan en este enunciado. Solo hay que entregar esos dos archivos (en ningún caso se entregarán archivos de *test*).**

d) Es imprescindible que no presente errores ni de compilación ni de interpretación (según corresponda), en los ordenadores del laboratorio asignado y en el sistema operativo *GNU/Linux*.[6] Se tratará de evitar también cualquier tipo de aviso (*warning*).

e) Todos los ficheros que se entregan deben contener el nombre del autor y su DNI (o NIE) en su primera línea (entre comentarios apropiados según el tipo de archivo).

f) Se comprimirán en un archivo `.tar.gz` cuyo nombre será el DNI del alumno, compuesto de 8 dígitos y una letra (o NIE, compuesto de una letra seguida de 7 dígitos y otra letra). Por ejemplo: `12345678A.tar.gz` o `X1234567A.tar.gz`. **Solo se admite este formato de compresión y solo es válida esta forma de nombrar el archivo.**

g) En el archivo comprimido **no debe haber subcarpetas**, es decir, al extraer sus archivos estos deben quedar guardados en la misma carpeta donde está el archivo que los contiene.

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

i) En la primera entrega solo es necesario que estén implementadas ambas versiones recursivas (aunque no hay inconveniente si se realiza algo más). La gestión de opciones del programa debe estar hecha en su totalidad. El resultado que se debe mostrar, en cualquiera de las dos entregas, para los casos aún sin hacer es "`?`". Por ejemplo, la siguiente salida corresponde a lo mínimo que se pide (para obtener la máxima nota) en la primera entrega:

```
$maze --ignore-naive -t -f 02.maze --p2D
- 7 ? ?
?
?
```

j) Si se hace la práctica completa para la primera entrega, también habrá que realizar la segunda entrega.

[6] Si trabajas con tu propio ordenador o con otro sistema operativo asegúrate de que este requisito se cumple (puedes comprobarlo mediante el compilador *online* de `https://godbolt.org` seleccionando la versión "`x86-64 gcc 9.5`").