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 24 y 31 de marzo, 23:55h.
A través de Moodle

Camino de coste mínimo

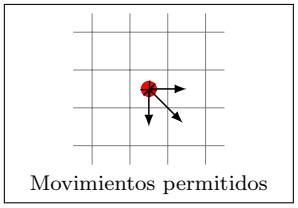
Un senderista está planificando una ruta de montaña haciendo uso de un mapa de coordenadas $n \times m$. En cada posición (i, j) de dicho mapa se representa, mediante un número natural, el grado de dificultad de visitar esa casilla. Cuanto más elevado es dicho valor más difícil es acceder a esa posición.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

$\xrightarrow{(0,0)}$

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

$\xrightarrow{(5,4)}$

Un mapa de coordenadas 6×5



Se pide, aplicar el método *divide y vencerás* y su extensión a *programación dinámica* 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 solo 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.

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 $(0, 0)$ a $(n-1, m-1)$ de esa dificultad.[3]

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

[3] Puede existir más de un camino de $(0, 0)$ a $(n-1, m-1)$ con la misma dificultad mínima; en tal caso se mostrará uno cualquiera de esa dificultad.

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: `mcp_naive`; `mcp_memo`; `mcp_it_matrix`; `mcp_it_vector` y `mcp_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.

   No es imprescindible, para valorar esta práctica, que esas cinco funciones 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. La función `mcp_naive` debe estar e implementada de forma correcta.

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

   El programa a realizar se debe llamar `mcp` (*minimum cost path*). La orden tendrá la siguiente sintaxis:

   `mcp` *`[-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` se describen en el siguiente apartado.
   - Las opciones 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 (si no se redirige) y consistirá, en primer

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

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.

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 dificultad mínima. Para ello, se utilizará un formato de dos dimensiones, mostrando una matriz, de la misma dimensión que el mapa de entrada, que contendrá el carácter 'x' para todas las casillas que componen el camino y el carácter '.' para las demás. No debe haber ningún carácter separador entre todos los caracteres 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. A continuación de esta matriz con la que se representa el camino, en la siguiente línea, debe mostrarse su dificultad calculada, exclusivamente, a partir de las celdas que lo componen. (Se trata de comprobar que la dificultad de este camino coincide con el valor devuelto por los algoritmos implementados.)

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, la tabla que almacena los resultados intermedios en la versión iterativa con matriz.

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 visualmente, estos valores numéricos nunca alcanzarán magnitudes de más de cuatro dígitos.

Al final de cada línea de la salida, sea cual sea, debe haber un único salto de línea, sin espacios en blanco ni tabuladores delante (por lo tanto, al finalizar el programa no se deben mostrar líneas visualmente vacías).

4. **Entrada al programa.**

El mapa $n \times m$ 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 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 (`??.map`), junto con las soluciones (`??.map.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.map cuyo contenido es el mapa (6×5) utilizado en la presentación de este problema y el fichero 01.map, cuyo contenido es un mapa 1×1.

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—).

```
 $mcp -f 02.map
 13 13 13 13

$mcp --ignore-naive -f 02.map --p2D
- 13 13 13
x....
x....
x....
.x...
.x...
..xxx
13

$mcp -t -f 02.map --p2D
13 13 13 13
x....
x....
x....
.x...
.x...
..xxx
13
 1  4  9 10 11
 3  5 10 12 11
 4  5 14 17 12
13  5 12 13 21
14  8 12 17 14
22  9 10 12 13

$mcp --p2D -f 01.map -t --p2D -t
10 10 10 10
x
10
10
```

```
*ALGUNOS EJEMPLOS DE SITUACIONES DE ERROR*

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

$mcp -f 01.map -f
ERROR: missing filename.
Usage:
mcp  [--p2D] [-t] [--ignore-naive] -f file

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


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


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


$mcp -f 01.map -f anonexistentfile -t
ERROR: can't open file: anonexistentfile.
Usage:
mcp  [--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 `mcp_naive` debe estar implementada de forma correcta. Las demás funciones deben resolver correctamente los ejemplos básicos (como `01.map` y `02.map`); 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 `mcp.cc`.

c) Se debe entregar únicamente los ficheros `mcp.cc` y `makefile` (para generar el archivo ejecutable, `mcp`, 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:

```
$mcp --ignore-naive -t -f 02.map --p2D
- 13 ? ?
?
?
```

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`").