Estructura de datos y de la información
Curso 2003-04

PRÁCTICA 6

Duración estimada: 2 sesiones

En esta práctica se propone escribir un algoritmo para solucionar un laberinto.

El laberinto es una matriz de posiciones, donde cada posición puede estar libre, o puede contener una barrera (pared). El laberinto lo podemos especificar en un fichero de texto. Para simplificar la lectura de datos pondremos en la primera línea del fichero el número de filas y de columnas. Para indicar una posición libre usaremos el espacio en blanco, y como barrera usaremos cualquier carácter no blanco, por ejemplo, un *.

El laberinto lo escribiremos, pues, con un editor de textos. Un ejemplo sería:

```
12 15
***************
*      *      *
****   ****   *
*  *      *   *
*  ****   *   *
*     **  *   *
*             *
* *****   *****
***        *  *
*    *   ***  *
*    *        *
***************
```

Numerando las filas y columnas de 0 a *filas*-1, y de 0 a *columnas*-1, la posición inicial, también por simplicidad, será la (1,1), y la final, donde se encuentra la salida del laberinto, será la posición (*filas*-2,*columnas*-2).

Nótese que que el laberinto está completamente rodeado por barrera, esto simplificará el algoritmo de búsqueda de la solución.

Ejercicio 1

- Crear una clase Laberinto que contenga los siguientes datos miembro:
  - El número de filas del laberinto.
  - El número de columnas del laberinto.
  - El laberinto en sí guardado como una matriz de caracteres.
- Implementar y probar dos operaciones miembro de la clase, una que lea del fichero el laberinto y la información sobre sus filas y columnas, y otra que escriba por pantalla el laberinto.

Para poder recorrer el laberinto utilizaremos asiduamente el concepto de posición en el mismo, equivalente a la posición en la matriz, dada por una fila y una columna.

Ejercicio 2

- Implementar y probar una clase posición en la que todos los datos y operaciones sean públicos. La clase contendrá las siguientes operaciones:
  - Un constructor al que se le pasen la fila y columna y construya una posición.
  - La sobrecarga de operadores == y != que permitan comparar posiciones.

A partir de las operaciones y clases ya creadas en los ejercicios anteriores estamos en condiciones de plantearnos un algoritmo para poder recorrer el laberinto.

Se pide escribir un programa que lea un laberinto contenido en un fichero de texto, lo solucione, y muestre la solución. Dicho algoritmo tendrá el siguiente aspecto:

```
int main() {
  string fichero=''laberinto.dat'';
  laberinto l;

  l.Leer(fichero);
  l.Solucionar();
  l.Mostrar();
  return 0;
}
```

A continuación se describe el algorimo que encuentra la solución al laberinto, es decir, el camino, secuencia de posiciones, a seguir desde la primera posición para llegar a la posición final:

Para recorrer el laberinto iremos avanzando por posiciones libres hasta alcanzar la posición final, o hasta que hayamos comprobado que no hay ninguna ruta hasta la salida.

Cada vez que nos situemos en una nueva posición, la marcaremos como recorrida, con el fin de no volver a avanzar por ella.

Cada vez que queramos avanzar, elegiremos una de las posiciones vecinas a la actual que estén libres y nos colocaremos en ella.

Si alcanzamos una posición que no tiene vecinas libres, tendremos que retroceder a la posición anterior. Si en la posición anterior tampoco hay vecinas libres, seguiremos retrocediendo hasta alcanzar una que sí las tenga.

Para poder retroceder, utilizaremos una pila de posiciones. Cada vez que avancemos, apilaremos la posición actual, y cuando retrocedamos, lo haremos a la posición almacenada en el tope de la pila.

Si mientras retrocedemos se vacía la pila y la posición actual no tiene vecinas libres, es que no existe ninguna ruta hasta la salida, y finalizaremos el algoritmo indicándolo.

Al final del algoritmo la solución del laberinto se encuentra en la pila de posiciones.

Ejercicio 3

- Añadir a la parte privada clase Laberinto del Ejercicio 1 un nuevo dato miembro que permita almacenar una pila de posiciones. Utilizar para ello la pila de la librería STL
- Añadir a la clase Laberinto las siguientes dos operaciones auxiliares que se utilizarán para implementar el algoritmo antes comentado:
  - Una operación que dada una posición, devuelva el número de posiciones vecinas libres. Tan sólo se considerarán cuatro vecinos: arriba, abajo, a la derecha y a la izquierda.
  - Una operación que dada una posición, devuelva una de las posiciones vecinas libres.

Como se puede observar, solucionar el laberinto implica modificar la matriz, varias veces hay que marcar ciertas posiciones como recorridas, para que no se vuelva a avanzar por una casilla por donde ya se ha pasado.

Por esto es conveniente definir dos matrices, una original y otra auxiliar, vacía en principio, y copiar la original sobre la vacía al principio del algoritmo que busca la solución. La matriz modificada por el algoritmo será la auxiliar.

Ejercicio 4

- Implementar un algoritmo que, apoyándose en las operaciones y clases implementadas hasta el momento, y siguiendo la descripción antes realizada, recorra un laberinto.

La matriz auxiliar utilizada en el algoritmo del ejercicio anterior también se puede usar para mostrar el camino solución, sacando posiciones de la pila y poniendo en cada posición de la matriz auxiliar un caracter especial, por ejemplo '0'.

Téngase en cuenta que al mostrar la solución se vacía la pila, y la solución queda en la matiz auxiliar. Si se llama a `MostrarSolucion()` una segunda vez, esta función debe saber que no hay que volver a vaciar la pila, que la solución ya está en la matriz auxliar. ¿ Como solucionarías este problema ?.

Ejercicio 5

- Añadir a la clase Laberinto una función que permita mostrar la solución del mismo utilizando las ideas anteriores.