VNIVERSITAT DE VALÈNCIA

ESTRUCTURAS DE DATOS Y ALGORITMOS
Grado en Ingeniería Multimedia (2º). **Curso 2022-23**

Práctica Nº 8: Algoritmos con Grafos
Periodo de realización: Semana del **19/12/2022 al 23/12/2022**

Objetivos

- Implementación de grafos en C++.
- Algoritmos para el recorrido de grafos.

Introducción

En los escenarios de un videojuego de mundo abierto los personajes deben desplazarse para completar sus misiones. Los personajes pueden estar gobernados por jugadores humanos o ser agentes autónomos con un comportamiento programado. Una de las formas más habituales de modelar el movimiento por estos escenarios es mediante un grafo donde los nodos corresponden a cada una de las posiciones del escenario donde se puede situar un personaje y los arcos se establecen entre posiciones vecinas a las que es posible transitar mediante un movimiento simple (un paso). En la figura 1 se muestra una representación 2D (con caracteres) de un ejemplo de este tipo de escenarios. El carácter 'X' indica una posición con obstáculo que impide que un personaje se sitúe en esa posición (p.ej., un muro). Por su parte, el carácter '**.**' representa una posición libre/transitable en la que un personaje se puede situar.

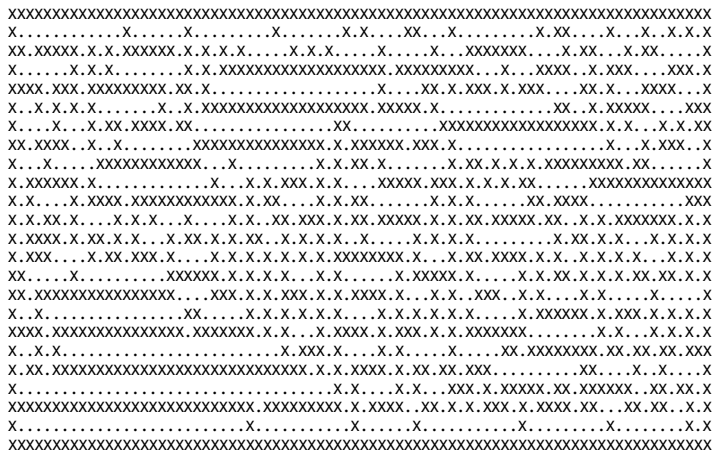


Figura 1. Ejemplo de escenario 2D representado mediante caracteres para identificar las posiciones.

Cuando un personaje se mueve por el escenario lo debe hacer mediante pasos simples, moviéndose siempre a una posición vecina que esté libre (**.**). Cuando un personaje se desplace a una nueva posición deberá seguir una ruta de posiciones libres desde su actual posición hasta el destino. Las rutas seguidas por los personajes estarán obviamente condicionadas por el diseño geométrico del escenario, pudiendo incluso haber rutas imposibles de obtener (posiciones inalcanzables).

En esta práctica se va a realizar un programa que permita representar el grafo de un escenario y se aplicará un algoritmo básico de exploración para determinar qué posiciones son alcanzables a partir de una posición origen. No es objetivo de la práctica obtener rutas entre posiciones.

Tareas

Git: Debes trabajar en la carpeta "Pr8" del repositorio que ya tienes creado para la asignatura. Descarga los archivos disponibles en Aula Virtual y realiza un primer *commit* ("`Pr.8: Versión inicial`") y *push* con ese material inicial. A partir de aquí, ya sabes que debes registrar periódicamente los cambios realizados. La política de gestión de los repositorios es responsabilidad de los estudiantes y se valorará en el apartado "Organización" de los criterios de evaluación de las prácticas. Se recomienda identificar todos los *commits* como "`Pr.8: …`" para facilitar la identificación de cambios en el repositorio.

Al igual que en anteriores prácticas, se proporciona como material adicional código fuente que servirá para desarrollar el programa requerido. En esta práctica se proporcionan los siguientes archivos:

- "`grafo.h`" y "`grafo.cpp`": Implementación de la clase *`Grafo`*[1].
- "`pr8.h`": Declaraciones de tipos y de funciones que permiten realizar las tareas descritas.

Tarea 1: Completar la clase *Grafo*

Como material para la práctica se proporcionan los archivos "`grafo.h`" y "`grafo.cpp`" con la especificación de la clase *`Grafo`* que debemos utilizar. Sin embargo, la implementación de los métodos de la clase está incompleta. Por lo tanto, en primer lugar, se deben implementar todos los métodos de la clase, cumpliendo estrictamente la especificación indicada en la documentación de cada método.

Debes tener en cuenta que el grafo se ha definido de manera que la lista de adyacencia con los arcos de un nodo se representan mediante una tabla hash (`unordered_map`)[2], lo que acelera y simplifica el proceso de búsqueda de arcos. En estas tablas la clave será el nodo destino del arco (índice que lo identifica) y el valor asociado será el peso del arco.

Al finalizar esta tarea, como siempre, es muy conveniente[3] realizar un pequeño programa de test que permita comprobar el correcto funcionamiento de la clase.

Tarea 2: Lectura del escenario

Para esta práctica se debe escribir un programa que permita crear y explorar el grafo que representa un escenario. Por lo tanto, en primer lugar, es necesario que el programa lea la información geométrica de un escenario y la almacene en un objeto del tipo *`Escenario`* (ver declaración de tipo en "`pr8.h`"). Se proporcionan dos escenarios (rectangulares) para las pruebas[4], los archivos "`escenario.dat`" y "`escenario_small.dat`", con el siguiente formato:

[1] La implementación de la clase Grafo está incompleta.
[2] El tipo ya ha sido utilizado previamente. No obstante, se puede consultar su documentación en: https://www.cplusplus.com/reference/unordered_map/unordered_map/.
[3] = necesario!!!
[4] Como siempre, se recomienda utilizar el archivo más pequeño para las pruebas iniciales.

- Primera línea: dimensiones del escenario, alto y ancho.
- Cada una de las siguientes líneas representan una fila de posiciones en el escenario. Cada carácter representa una posición. Si el carácter es 'X' la posición contiene un obstáculo, si el carácter es '`.`' la posición está libre y es transitable[5].

La información de las posiciones del escenario se representa en la estructura *`Escenario`* como un vector de valores lógicos (`true`=libre, `false`=obstáculo). El vector es unidimensional, no una matriz, de manera que las posiciones se numeran correlativamente según se van leyendo. Así, el primer carácter leído del archivo es la posición 0 (esquina superior izquierda), el siguiente la posición 1 y así sucesivamente. La última posición corresponderá con la esquina inferior derecha. Habrá tantas posiciones como establezcan las dimensiones del escenario (alto x ancho).

Verifica el proceso de lectura usando los archivos proporcionados. Lee el archivo y carga el escenario. Muestra el número de posiciones y cuántas son transitables. También puedes usar la función *`PintarEscenario`* para mostrar el escenario en pantalla. Comprueba que coincide con estos valores:

Archivo	Dimensiones	Posiciones	Transitables	Obstáculos
"`escenario_small.dat`"	5 x 10	50	18	32
"`escenario.dat`"	24 x 80	1920	917	1003

Tarea 3: Construcción del grafo de escenario

Una vez que se dispone del escenario, el programa debe construir el grafo de movilidad entre posiciones del escenario. Como se ha indicado en la introducción, se debe construir un grafo que tenga un nodo por cada posición del escenario (no importa el tipo). Cada nodo podrá tener un arco con los nodos correspondientes a posiciones vecinas en el escenario. Obviamente, una posición donde hay un obstáculo no tendrá arcos con ninguna otra posición. Las posiciones vecinas serán aquellas que pueden ser accedidas en un único paso, de acuerdo con los tipos de movimientos previstos. En este caso, vamos a suponer que solo se pueden realizar los 4 movimientos simples: 1 paso a la derecha, 1 paso a la izquierda, 1 paso arriba y 1 paso abajo. No se admiten movimientos en diagonal para no complicar innecesariamente el proceso. En la figura 2 se muestra gráficamente esta idea.

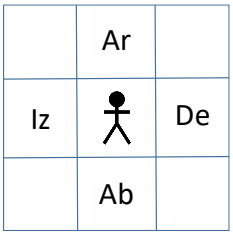


Figura 2. Movimientos admitidos en el escenario.

En el ejemplo de la figura 2 el personaje ubicado en la posición central solo puede moverse a las posiciones marcadas como `Iz`, `Ar`, `De` y `Ab`, tal como se ha indicado previamente. Por lo tanto, a la hora de construir el grafo, el nodo correspondiente a esa posición central solo podrá estar relacionado con las 4 posiciones indicadas, siempre y cuando esas posiciones estén libres, porque si hay un obstáculo en alguna de ellas no será posible realizar ese movimiento y no habrá arco en el grafo. En consecuencia,

[5] Se puede considerar que cualquier carácter que no sea '.' es un obstáculo.

en este grafo el grado máximo de cualquier nodo será 4. También consideraremos que todos los movimientos son iguales y, por tanto, todos los arcos del grafo tendrán peso 1.

El mecanismo para determinar las posiciones vecinas a una posición ya viene dado por la función *Vecinos* proporcionada en el archivo "pr8.h". Esta función devuelve las posiciones vecinas a una dada y ya tiene en cuenta que no siempre habrá 4 vecinos, puesto que hay posiciones situadas en los límites del escenario que tendrá menos vecinos (2 o 3).

Comprueba que el número de nodos del grafo creado coincide con el número de posiciones del escenario. También puedes mostrar el grafo por pantalla para comprobar que los arcos son correctos.

Tarea 4: Exploración del grafo

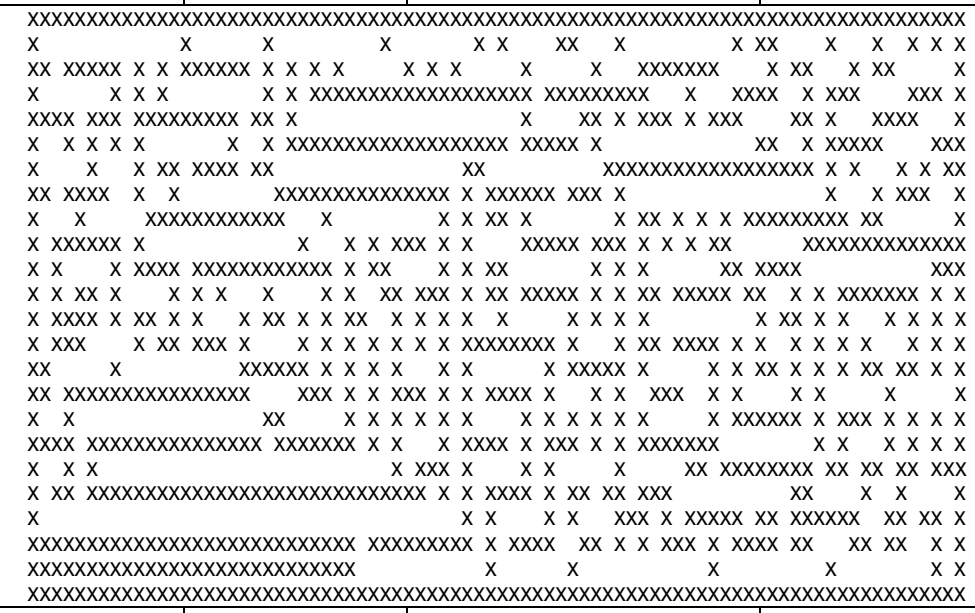
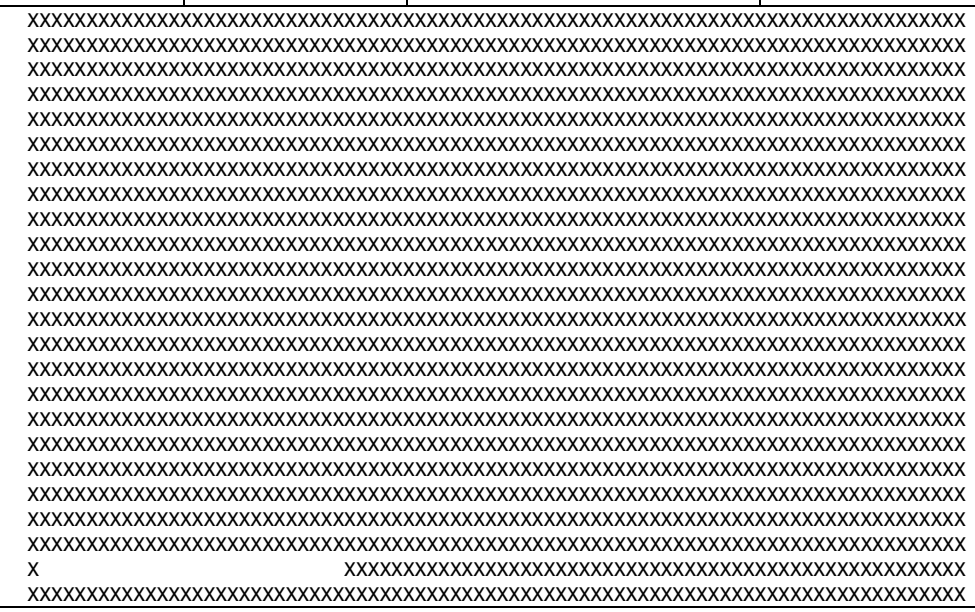
Una vez que se dispone del grafo que representa el escenario, ya es posible explorar el grafo/escenario para conocer, por ejemplo, qué posiciones serían alcanzables desde una posición origen. Para ello, se debe implementar en el programa el algoritmo de exploración en profundidad (*DFS*) de un grafo desde un nodo origen. El algoritmo debe devolver un vector de valores booleanos que indique, para todos los nodos del grafo, si el nodo ha sido visitado en el recorrido (*true*) o no (*false*). El índice de los nodos en este vector coincidirá con el índice del vector posiciones del escenario y, por tanto, del grafo.

La implementación del recorrido DFS se debe realizar de manera eficiente, puesto que se dispone del método *NodosAdyacentes* en la clase *Grafo*.

El resultado del recorrido DFS se puede utilizar para construir un nuevo escenario, que tendrá las mismas dimensiones (alto y ancho) del escenario original y donde las posiciones serán iguales al vector resultante del DFS. Puedes utilizar la función *PintarEscenario* para mostrar en pantalla el resultado de la exploración. En pantalla se mostrarán con el símbolo 'X' las posiciones no alcanzadas en el recorrido (sean obstáculos o no) y como espacios en blanco las posiciones que se sí se pueden alcanzar desde el origen del recorrido.

Tarea 5: Banco de pruebas

Para validar los resultados se deben realizar al menos dos pruebas, con dos orígenes diferentes, para cada escenario proporcionado. En la tabla se muestran los datos y los resultados esperados de las pruebas a realizar. Se indica el origen usado con cada escenario, el número de posiciones transitables en el grafo original y el número de posiciones transitadas como resultado del recorrido del grafo desde el origen (=nodos visitados). Cuando estos dos valores son diferentes significa que no es posible recorrer todo el escenario desde el origen. También se muestra el escenario del recorrido en cada caso.

Escenario	Origen	#posiciones transitables en original	#posiciones transitadas en el recorrido
"escenario_small.dat"	11	18	16
			
"escenario_small.dat"	28	18	2
			
"escenario.dat"	83	917	891
			
"escenario.dat"	1761	917	26
			

Tarea 5: Generación de documentación

El código del programa deberá estar correctamente documentado con Doxygen, de acuerdo con la guía de estilo. Una vez finalizado el programa, se deberá ejecutar Doxygen para verificar la correcta generación de la documentación del programa.