Examen parcial 4

Estructuras de Datos

27 mayo 2021

1º Grado en Ingeniería de Computadores
URJC (www.urjc.es)

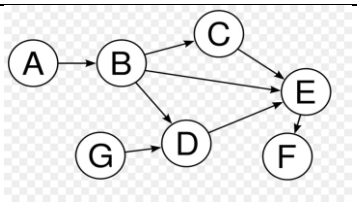
NOMBRE Y APELLIDOS: _____

DNI: _____

Marca a continuación la respuesta a todas las preguntas de test. Puedes cambiar tu respuesta en una pregunta, marcando en el siguiente "intento" tu nueva respuesta. Tienes como máximo tres intentos. Sólo se tendrá en cuenta el último intento rellenado. Te recomendamos que, antes de rellenar esta plantilla, pienses bien tu respuesta para que no te arriesgues a agotar los tres intentos. En ningún caso pongas varias "X" en un mismo intento. En caso de que esto se haga mal, se considerará que la pregunta no ha sido contestada. No lo rellenes con lápiz, sólo con bolígrafo.

	1er intento										2º intento										3er intento									
	A	B	C	D	E	F	G	H	I	J	A	B	C	D	E	F	G	H	I	J	A	B	C	D	E	F	G	H	I	J
1																														
2																														
3																														

1) (0,25 puntos) En el grafo de la figura, ¿cuál sería un recorrido topológico?



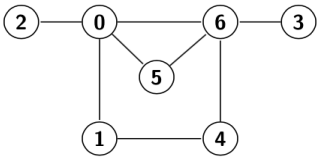
a) AGBCEFD
b) ABGDCFE
c) GABCEDF
d) CBEFDGA
e) FECDBAG
f) DGEFABC
g) EFCDBAG
h) Ninguna de las anteriores

Solución: h

2) (0,25 puntos) Indica el número de sentencias afirmativas.

- Un árbol es un grafo acíclico, además de tener otras posibles propiedades
- Un pseudografo es un grafo en el que puede haber más de un arco entre cada par de nodos
- Un grafo conexo es un grafo en el que ningún nodo está desconectado del resto
- Un camino es una lista de nodos en la que hay un arco entre cada par de elementos sucesivos
- Un ciclo es un camino en el que el primer y el último nodo son iguales, y sin arcos repetidos

a) 0.
b) 1.
c) 2.
d) 3.
e) 4.

	a) 2065341 es un recorrido en profundidad b) 2016543 es un recorrido en anchura c) 2014563 es un recorrido topológico d) 3641502 es un recorrido topológico e) "a" y "b" son correctas f) "c" y "d" son correctas g) Todas las anteriores son correctas h) Ninguna respuesta es correcta

Solución: e

4) (1,75p) Tenemos una clase Grafo implementada mediante listas de adyacencia. El elemento contenido en cada nodo es simplemente un número entero, el cual es tanto la clave como el valor del elemento. En cada nodo, guardamos la lista de sus arcos salientes (pero no los entrantes). Cada una de esas listas es simplemente un array de enteros. El conjunto de nodos se implementa mediante un array de enteros también. Todos los arrays deben tener en cada momento la memoria reservada justa para guardar los elementos (no debe sobrar memoria en ningún momento).

Se proporciona la declaración de la clase a continuación:

```
// Clase que representa a un grafo implementado con listas de adyacencia. El
// elemento de cada nodo es simplemente un número entero (dicho número es la
// clave y el valor del elemento). Asociado a cada nodo, hay una lista de
// adyacencia con los arcos salientes del nodo. Cada una de esas listas es un
// array de enteros. El conjunto de nodos está implementado mediante un
// array de enteros también. El grafo es dirigido, no multigrafo, pseudografo, no
// etiquetado.
class GrafoListasAdyacencia {
        int* nodos; // Lista de nodos del grafo
        int** arcosSalientes; // Lista de los arcos salientes de cada nodo
        int* numeroArcosSalientesPorNodo; // Num de arcos salientes de cada nodo
        int n; // Numero actual de nodos del grafo
private:
        // Obtiene la posición de un nodo en la lista de nodos del grafo
        // Parámetro: clave del nodo
        // Retorno: posición del nodo en la lista de nodos, o -1 si no existe
        // Complejidad temporal: O(n)
        int getPosicionNodo(int clave);
public:
        // Construye un grafo vacío
        GrafoListasAdyacencia();

        // Averigua si un nodo existe en el grafo o no
        // Parámetro: clave del nodo
        // Retorno: true si existe, false si no
        bool existeNodo(int clave);

        // Inserta un nuevo nodo, sin rutas entrantes ni salientes
        // Parámetro: clave del nuevo nodo
        void insertarNodo(int clave);

        // Averigua si hay un arco entre dos nodos
        // Parámetros: claves de los nodos origen y destino
        // Retorno: true si hay ruta desde el origen hasta el destino, false si no
        bool existeArco(int claveOrigen, int claveDestino);

        // Inserta un nuevo arco en el grafo
        // Parámetros:
        // - claveOrigen: entero del nodo origen
        // - claveDestino: entero del nodo destino
        void insertarArco(int claveOrigen, int claveDestino);
};
```

La implementación del constructor es la siguiente:

```
GrafoListasAdyacencia::GrafoListasAdyacencia() {
```

c) (0,4p) Método "existeArco"

d) (0,5p) Método "insertarArco"

Soluciones:

```
// Complejidad temporal: O(n)
bool GrafoListasAdyacencia::existeNodo(int clave) {
	if (getPosicionNodo(clave) == -1) return(false);
	else return(true);
}

// Complejidad temporal: O(n)
void GrafoListasAdyacencia::insertarNodo(int clave) {
	assert(!existeNodo(clave)); // Esa clave no existe previamente

	// Creamos el nuevo nodo
	nodos = (int*)realloc(nodos, (n + 1) * sizeof(int));
	nodos[n] = clave;

	// Creamos su lista de arcos salientes
	arcosSalientes = (int**)realloc(arcosSalientes, (n + 1) * sizeof(int*));
	arcosSalientes[n] = NULL;

	// Actualizamos su número de arcos salientes
	numeroArcosSalientesPorNodo = (int*)realloc(numeroArcosSalientesPorNodo, (n + 1) * sizeof(int));
	numeroArcosSalientesPorNodo[n] = 0;

	// Incrementamos n
	n++;
}

// Complejidad temporal: O(n)
bool GrafoListasAdyacencia::existeArco(int claveOrigen, int claveDestino) {
	assert(existeNodo(claveOrigen));
	assert(existeNodo(claveDestino));

	int posicionNodoOrigen = getPosicionNodo(claveOrigen);
	for (int i = 0; i < numeroArcosSalientesPorNodo[posicionNodoOrigen]; i++)
		if (arcosSalientes[posicionNodoOrigen][i] == claveDestino) return(true);
	return(false); // Si hemos llegado hasta aqui, no existe el arco
}

// Complejidad temporal: O(n)
void GrafoListasAdyacencia::insertarArco(int claveOrigen, int claveDestino) {
	assert(existeNodo(claveOrigen));
	assert(existeNodo(claveDestino));
	assert(!existeArco(claveOrigen, claveDestino));

	// Añadimos el nuevo arco saliente
	int posicionNodoOrigen = getPosicionNodo(claveOrigen);
	int m = numeroArcosSalientesPorNodo[posicionNodoOrigen]; // Numero actual de arcos salientes del nodo
	arcosSalientes[posicionNodoOrigen] = (int*)realloc(arcosSalientes[posicionNodoOrigen], (m + 1) *
sizeof(int));
	arcosSalientes[posicionNodoOrigen][m] = claveDestino;

	// Actualizamos numero de arcos salientes del nodo
	numeroArcosSalientesPorNodo[posicionNodoOrigen]++;
}
```