Problemas de examen de la E.P. de Cáceres

Junio 94

Ejercicio 1.- Se desea implementar mediante un programa la labor de un subastador. Los objetos que pueden ser subastados son **cuadros**, cuyas características son: *autor, valor, precio de salida y dimensión*; y **esculturas**, cuyas características son: *autor, valor, precio de salida, peso y material.*

Cuando al subastador le llega un objeto, se rellenan sus características convenientemente y se almacenan en una estantería, de tal forma que el objeto almacenado en último lugar será el primero en subastarse. A la hora de subastar, el subastador coge un objeto de la estantería y enumera sus características.

Se pide:

a) Declarar una estructura jerárquica que represente convenientemente los objetos a subastar. Decir brevemente qué haría cada función, sin implementarla. Si se usan métodos virtuales justificar su uso.

b) Implementar el objeto subastador, que haciendo uso de polimorfismo, tiene que realizar dos operaciones:

* RellenarEstantería ---> Llena de varios cuadros y esculturas la estantería.
* Subastar ---> Coge uno a uno los objetos y los subasta.

Justificar razonadamente la estructura elegida para la estantería y hacer un pequeño dibujo de dicha estructura indicando claramente cuál es el tipo de datos que contiene y por qué.

Ejercicio 2. Se dice que un grafo está **k-coloreado** si se cumplen 3 condiciones:

a) Todos los vértices tienen un color.
b) Ningún vértice está coloreado con un color mayor que k.
c) Dos vértices adyacentes no tienen el mismo color.

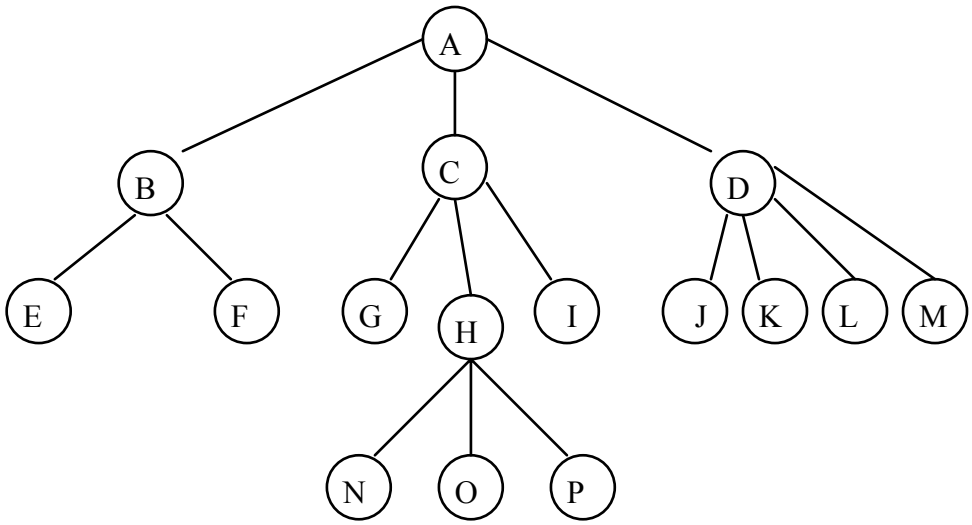
Teniendo en cuenta esta definición, se pide hacer un procedimiento que k-coloree un grafo.

Se puede usar la función *QueColor (i)* que nos dice el color de un vértice; y el procedimiento *AsignarColor (i,NuevoColor),* que asigna un color a un vértice. Con el uso de estas 2 operaciones no hace falta tocar la matriz de adyacencia.

Se **deberá** usar esquema de Backtracking. Suponed que los colores son números para simplificar.

Ejercicio 3. Sea **A** un árbol binario que es una representación de un árbol general.
Siguiendo técnicas de POO, construir un procedimiento iterativo que imprima toda la jerarquía del árbol general por niveles. Obtener la complejidad del algoritmo construido.

Ejemplo:



debe producir la salida:

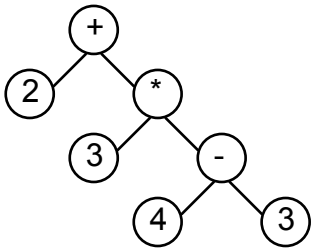
A: B,C,D
B: E,F
C: G,H,I

D: J,K,L,M
H: N,O,P

Septiembre 94

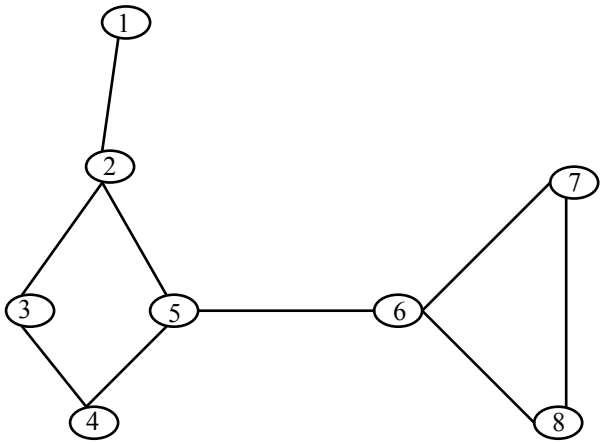
Ejercicio 1.- Dado un árbol binario que es la representación de una expresión matemática como en el ejemplo adjunto, **se pide** diseñar una función que haciendo uso de técnicas de POO obtenga el resultado de evaluar dicha expresión.

Ejemplo:



La expresión equivalente es: 2+3*(4-3)
La salida para este ejemplo sería: 5

Ejercicio 2. Sea G un grafo conexo no dirigido. Diremos que un arco (i,j) perteneciente al conjunto de los arcos de G es un puente, si al ser borrado hace a G no conexo. (Según esta definición, en el grafo de la figura, los arcos (1,2) y (5,6) son puentes)



Se pide:
a) Escribir un procedimiento que liste los puentes de un grafo conexo no dirigido,G.
b) Hallar la complejidad del algoritmo empleado.

NOTA:

Para resolver el ejercicio está permitido utilizar cualquiera de las operaciones del TAD grafo vistas en clase de teoría. Sin embargo, cualquier otra operación deberá ser implementada. Suponer que la complejidad de la operación **EsConexo** viene dada en función del número de arcos del grafo *e* , **f(e)**.

Ejercicio 3. La generalidad es la propiedad que presentan determinados lenguajes de programación, que permite definir clases donde el tipo de dato que contienen se determina desde el exterior de las clases. De todos es conocido que existen lenguajes que soportan la generalidad (e.j. ADA, C++, etc.) y otros como Pascal que, aunque no la soportan, permiten utilizar estrategias para simular la definición de clases genéricas. Se trata de idear una solución que permita definir una clase genérica LISTA en Pascal, de tal manera que podamos tener un programa con definiciones como el adjunto.

```
Program examen;
USES ListaGen;    { importa la clase Lista}
/* Habrá que cambiar algo unicamente
   en la definición de la clase Lista; no en
   la implementación de los métodos */

/*   codigo a escribir por el alumno /*

VAR
      lc    :     Lista;     {contiene una lista de caracteres }
      li    :     Lista;     {contiene una lista de enteros }
      lnombres : Lista;      {contiene una lista cuya información
                              son nombres y apellidos }
```

Una vez realizadas las definiciones oportunas, terminar de completar este programa para que inserte una colección de caracteres, enteros, nombres y apellidos en las listas respectivas y los imprima haciendo uso de los métodos convencionales de la clase Lista (primero y resto).

Febrero 97

1. En 1906, en el *Tribune* londinense, un famoso columnista inglés propuso el siguiente problema : Colocar 16 peones en un tablero de ajedrez de forma que no se encuentren 3 alineados en ninguna dirección. Por alineados se entiende cualquier recta, ya sean filas, columnas o diagonales.

Se pide :

1. Escribir el esquema general de un algoritmo de tipo Backtracking, indicando brevemente cual es la filosofía de esta familia de algoritmos.
2. Siguiendo técnicas de P.O.O., escribir en Turbo Pascal un algoritmo tipo backtracking que resuelva el problema. Para ello suponer conocido lo siguiente :
   - El tablero de juego se representa mediante la matriz M.
     M : Array [1..8,1..8] of Boolean;
   - La función *SituacionValida*, que nos dice si la situación actual del tablero es válida.
     Function SituacionValida (M : TableroDeJuego) : Boolean;

NOTA : El ejercicio NO se corregirá si no sigue un esquema de tipo backtracking.

2. Se dice que un grafo es bipartido si todos sus vértices se pueden agrupar en dos conjuntos disjuntos V_1 y V_2, de tal manera que para cualesquiera dos vértices pertenecientes a V_1 se cumple que no son adyacentes entre sí. (Igual para los miembros de V_2).

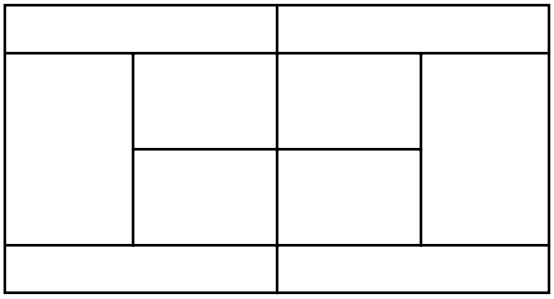
Siguiendo técnicas de P.O.O. en Pascal, desarrollar un procedimiento que tome como entrada un grafo no dirigido, compruebe si se trata de un grafo bipartido, y en caso afirmativo devuelva como salida los dos conjuntos V_1 y V_2 que haya obtenido, y en caso negativo devuelva los dos conjuntos vacíos.

Comentar brevemente la solución propuesta de forma razonada, es decir, no se trata de leer con palabras lo que pone en el código, sino dar una idea global del planteamiento dado para solucionar el problema.

Nota: si para la implementación del programa se necesitara de algún tipo de recorrido sobre el grafo, se recomienda usar un recorrido en anchura.

Junio 97

1.- La empresa *PPT* se dedica a la construcción y mantenimiento de pistas de tenis. Dispone de una máquina tiralíneas muy precisa, pero a la vez muy pesada. El método que emplean para pintar las líneas consiste en trazar primero todas las líneas exteriores, y luego todas las líneas interiores. Este método presenta un problema: hay que levantar demasiadas veces la máquina para cambiarla de dirección, y eso es complicado por el peso de la misma. *PPT* quiere encontrar la forma de pintar completamente la pista de tenis sin levantar la máquina del suelo. Para ello acude a un experto informático. Después de escuchar sus argumentos, el informático responde: "*No estoy seguro de que se pueda pintar toda la pista sin levantar la máquina, pero puedo diseñar un pequeño algoritmo que nos saque de dudas*".



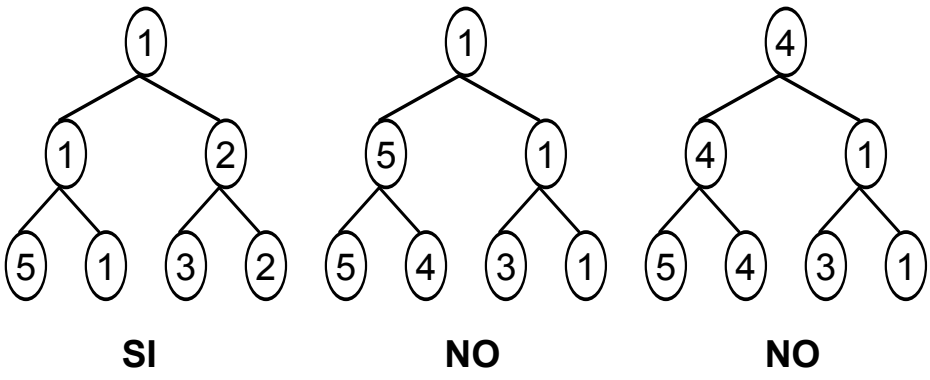
SE PIDE:

1. ¿Qué estructura de datos, de entre todas las que se han estudiado en la asignatura, sería la más adecuada para representar el problema? Haz un pequeño dibujo que lo aclare.
2. ¿Qué tipo de algoritmo empleará el informático para demostrar si tiene o no solución? Explicar el porqué de esa elección.
3. Empleando técnicas de P.O.O. en Turbo Pascal, escribir el citado algoritmo.

2.- Un árbol de selección es un árbol binario donde cada nodo representa al menor de sus dos hijos. Construir una función que, dado un árbol binario, indique si es o no un árbol de selección. El prototipo de la función debe ser:

Function ArbolSeleccion (A: Arbol) : Boolean;

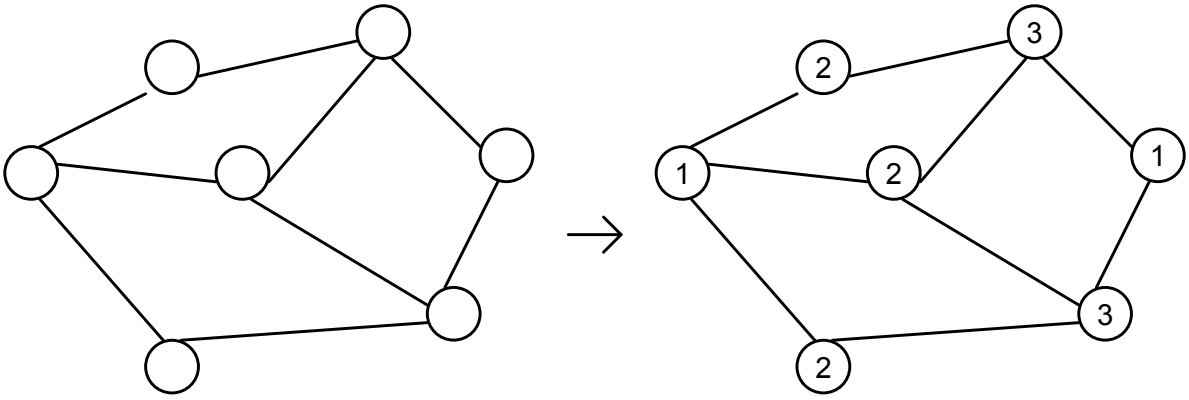
Ejemplos:



Julio 97

1.- Se dice que un grafo es *k*-coloreable si es posible colorear los vértices del mismo sin emplear más de *k* colores y de tal forma que cualesquiera dos vértices adyacentes no pueden tener el mismo color. Empleando técnicas de P.O.O. en Borland Pascal, diseñar un procedimiento que *k*-coloree un grafo G. El prototipo de la función debe ser:

Procedure kcolorear (G: Grafo; k: Integer; VAR solución);

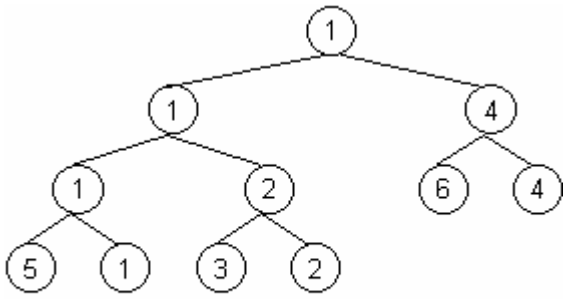


Grafo de entrada Grafo k-coloreado para k=3

2.- Un árbol de selección es un árbol binario donde cada nodo representa al menor de sus dos hijos. Siguiendo técnicas de P.O.O. en Borland Pascal, construir un procedimiento que, tome como entrada una lista que contiene una secuencia de *n* números y devuelva un árbol de selección compuesto por esos *n* números. El prototipo de la operación debe ser :

Procedure ArbolSelección (L: Lista; VAR A: ArbolBinario);

Ejemplo: Secuencia de números de entrada : 5, 1, 3, 2, 6, 4.



Árbol de decisión resultante

Septiembre 97

1.- Sea G un grafo conexo no dirigido, y *v* un vértice de G. Escribir un algoritmo que obtenga el árbol de expansión de menor altura a partir de *v* sobre el grafo G.

NOTA: Un árbol de expansión de una grafo conexo no dirigido es un árbol que contiene todos los vértices de G. Se puede hacer uso de la función ExisteCiclo respetando el siguiente prototipo:

Function ExisteCiclo (G: Grafo; v: TipoVertice) : Boolean;

Recomendación: Utilizar uno de los dos tipos de recorridos sobre grafos, justificando porqué el recorrido elegido proporciona la solución correctas.

2.- Una **tabla de símbolos** es una estructura que almacena convenientemente pares de la forma (nombre, valor), donde cada nombre puede tener asociado más de un valor. Además, el número máximo de nombres posibles es conocido a priori y viene delimitado por una constante **N**. Su especificacion es la siguiente:

TIPO: tabla

SINTAXIS:

Crear() -> tabla
Insertar(tabla, nombre, valor) -> tabla
Borrar(tabla, nombre) -> tabla
Encontrar(tabla, nombre) -> valor
Existe(tabla, nombre) ->Boolean
Vacia(tabla) -> boolean

SEMANTICA: Para todo $S \in Tabla$; $a,b \in nombre$; $r \in valor$

Vacia(crear) $\Rightarrow$ **true**
Vacia(insertar(S,a,r)) $\Rightarrow$ **false**

Existe(crear,a) $\Rightarrow$ **false**
Existe(insertar(S,a,r), b) $\Rightarrow$ **IF** a=b **THEN true ELSE** Existe(S,b)

Borrar(crear, a) $\Rightarrow$ crear
Borrar(insertar(S,a,r), b) $\Rightarrow$ **IF** a=b **THEN** S
ELSE insertar(borrar(S,b), a, r)

Encontrar(crear, a) $\Rightarrow$ *error*
Encontrar(insertar(S,a,r), b) $\Rightarrow$ **IF** a=b **THEN** r
ELSE Encontrar(S, b)

SE PIDE: Definir la estructura de clase Tabla de Simbolos, definiendo los atributos y métodos necesarios para que se cumpla la especificacion anterior. Los atributos elegidos deberán ser tales que permitan definir las operaciones CREAR y VACIA con complejidad constante. Indicar mediante un dibujo cual es la estructura de datos elegida. Explicar brevemente que hace cada uno de los métodos e implementar **exclusivamente** los métodos Existe y Encontrar.

Diciembre 97

Ejercicio 1.

Sea G un grafo conexo no dirigido. Siguiendo técnicas de POO, diseñar un algoritmo que encuentre la longitud del ciclo más corto que hay en G. El algoritmo deberá ser lo más eficiente posible.

Ejercicio 2.

Un grafo dirigido puede admitir una representación mediante matrices de adyacencia, listas de listas y también utilizando un array de listas.
Diseñar la estructura de clase GRAFO utilizando como estructura de datos un array de listas. Dar las definiciones necesarias, así como también un pequeño dibujo de la estructura utilizada.
Dar las implementaciones de los métodos adyacentes y borrarvértice.

Febrero 98

1.- Sea K la profundidad de un arbol binario. Un arbol binario se dice casi completo si todos sus nodos tienen 2 hijos y sus hojas tienen nivel K o K-1. En este último caso, si B representa a una hoja de nivel K-1, todas las hojas a la derecha de B tienen nivel K-1, necesariamente. Usando POO, escribir un algoritmo que decida si un arbol binario es o no casi-completo.

SUGERENCIA: Utilizar un recorrido en anchura.

2.- Un grafo dirigido puede admitir una representación mediante matrices de adyacencia, listas de listas y también utilizando un array de listas.

Diseñar la estructura de clase GRAFO utilizando como estructura de datos un array de listas.

Dar las definiciones necesarias, así como también un pequeño dibujo de la estructura utilizada.

Dar las implementaciones de los métodos **adyacentes** y **borrarvértice**.

Junio 98

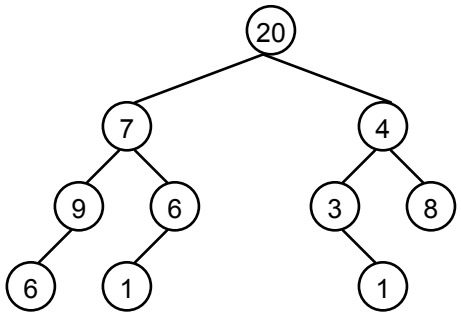
1. Hemos estudiado dos formas generales de hacer recorridos sobre árboles. Pero éstas no son las únicas. Existen otro tipo de recorridos que no siguen ninguno de esos esquemas. Concretamente, un algoritmo que llamaremos **recorrido_guiado**, permite realizar un recorrido sobre un árbol de forma que, en cada iteración, se selecciona el nodo más pequeño de entre todos los disponibles en ese momento, independientemente de en qué rama se encuentre. Se entiende por nodo disponible aquel nodo cuyo padre ya ha sido procesado (excluyendo el nodo raíz). Además, este tipo de algoritmos optimiza el tiempo de proceso, de forma que, cuando un árbol tiene nodos repetidos, y se llega a un nodo igual que uno anteriormente visitado, este último se olvida sin más.

Suponer además que tenemos implementada una clase **COLA** cuya operación de inserción coloca los datos en la cola ordenados de menor a mayor.

SE PIDE: Usando técnicas de POO en Borland Pascal, implementar una función que, dado un árbol binario, y empleando el algoritmo descrito anteriormente, realice un recorrido guiado por el árbol, mostrando en pantalla el contenido de cada nodo. El prototipo de la función será: **Procedure recorrido_guiado (A: Arbol);**

NOTA: Los nodos del árbol se deben procesar **una sola vez**.

Ejemplo de recorrido:

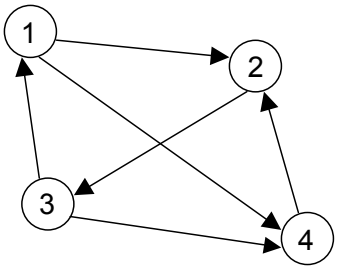


Para este árbol, el procedimiento *recorrido_guiado* debe mostrar en pantalla:

20, 4, 3, 1, 7, 6, 8, 9

2. Un grafo dirigido se dice que es IMPAR si para cada vértice del mismo la suma de los arcos de entrada y de salida es siempre un número impar. Siguiendo técnicas de POO en Borland Pascal, construir una función de complejidad cuadrática que, dado un grafo dirigido, compruebe si es impar.

Ejemplo:



Este grafo es IMPAR.

NOTA: Suponer que la clase grafo dirigido está implementada con una matriz de adyacencia.

Septiembre 98

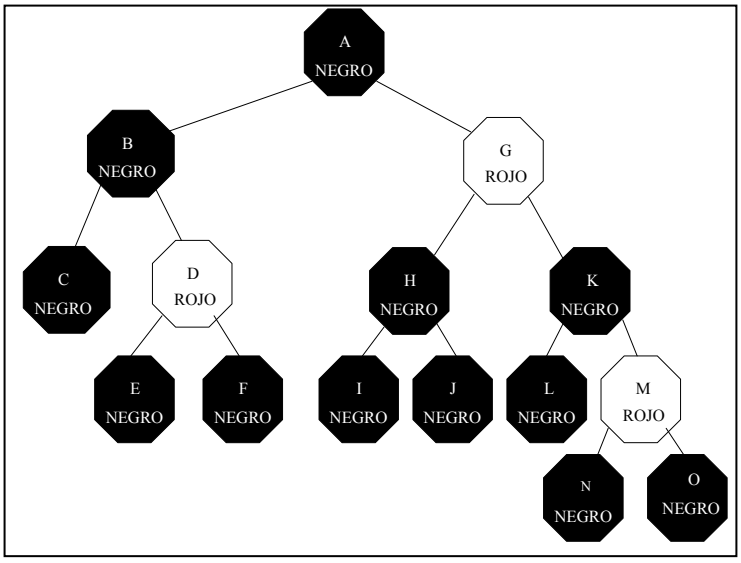
Problema 1:

Un árbol ***Rojo-Negro*** es un árbol binario coloreado (se añade un nuevo campo en cada nodo del árbol que indica su color) en el que todo nodo está coloreado de rojo o de negro, y en el que la disposición de los nodos obedece las siguientes restricciones:

1. **(Regla del Negro)** Toda hoja está coloreada de ***Negro***.
2. **(Regla del Camino)** El número de nodos **rojos** del subárbol izquierdo y el subarbol derecho del nodo raiz difiere como mucho en 1.

SE PIDE: Usando técnicas de POO en Borland Pascal, Implementar una función que, dado un árbol binario **DECIDA** si se trata de un árbol **Rojo-Negro**.

Ejemplo de Arbol **Rojo-Negro**:



Problema 2.

Un grafo conexo no dirigido se dice que es SEMI-VALORADO si existe un arco por el que se puede dividir en dos subgrafos, de tal forma que :

1. Los conjuntos de vértices de los dos subgrafos resultantes son disjuntos entre sí, es decir: Conjunto_Vértices(G_1) $\cap$ Conjunto_Vértices(G_2) = $\varnothing$, para $G_1 \subset G$ y $G_2 \subset G$.
2. La suma del contenido de los vértices del subgrafo G_1 coindice con la suma del contenido de los vértices del subgrafo G_2.

Suponiendo que se dispone de las siguientes funciones :

```
Function Conexo (G: Grafo) : Boolean;
Procedure Profundidad (G: grafo; v: TipoVértice; VAR visitados: CjtoVertices);
```

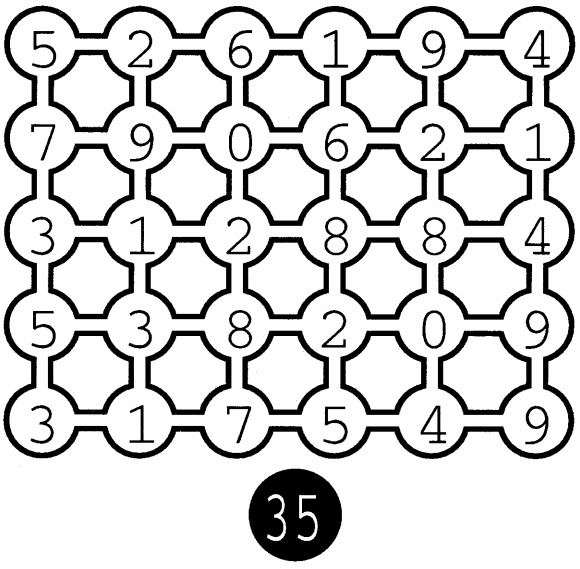
Implementar una función que dado un grafo G, y siguiendo técnicas de POO en Borland Pascal, compruebe si se trata de un grafo semi-valorado.

NOTA: Para simplificar el problema, considerar que solamente puede existir un arco que cumple las condiciones anteriores

Diciembre 98

Se nos plantea el siguiente problema es una revista de pasatiempos:

Partiendo de cualquiera de los círculos de arriba, deberá hacer un recorrido sumando todos los números por los que pase, de forma que, al salir por abajo, haya pasado por ocho círculos diferentes y sumado 35 (pueden darse diferentes soluciones).



Un algoritmo de tipo *backtracking* es el que mejor de adapta a la resolución de este problema.

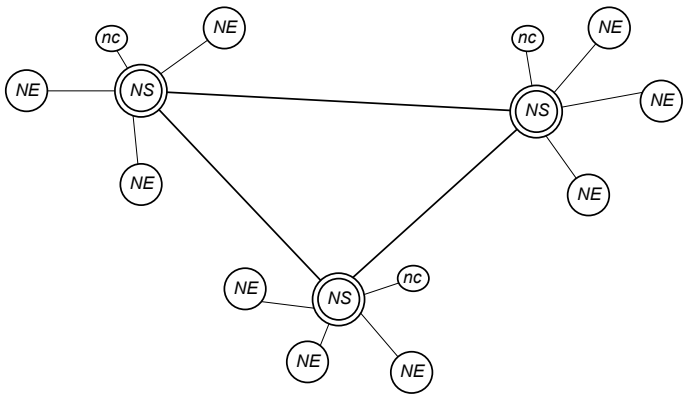
1. Definir el esquema general de un algoritmo de tipo bactracking. Indicar de forma razonada cuál será la complejidad estimada del mismo. Asimismo, comentar las estructuras de datos que se vas a emplear para implementar el algoritmo que resuelve el problema.

2. Implementar cada una de las partes que componen el esquema general definido en el paso anterior, indicando qué hace cada una de ellas y porqué se llega a la solución deseada.

3. Haciendo uso de técnicas de POO en Borland Pascal, implementar una función que resuelva el problema siguiendo el esquema backtracking descrito anteriormente.

Febrero 99

Problema nº1.



Después de terminar tus estudios te has colocado en una empresa que se dedica a confeccionar programas a medida para otras empresas. Te han asignado a un proyecto de redes, en la que pueden encontrarse 3 tipos diferentes de nodos:

NS Nodo Servidor
NE Nodo Estación de trabajo
NC Nodo Copia de seguridad

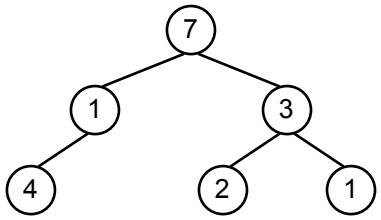
En el gráfico de la figura adjunta se muestra una posible configuración de la red. Cada nodo servidor puede ofrecer servicios a un número indeterminado de nodos estación de trabajo.

Para que la red se considere que está bien construida, cada nodo servidor debe tener al menos un nodo copia de seguridad conectado. Además, todos los nodos servidores deben estar conectados entre sí.

Tu primer trabajo será implementar en Borland Pascal, y siguiendo técnica de POO, una función que tome como entrada un grafo como el de la figura y compruebe si la red está bien construida. Además, deberá mostrar en pantalla todos los nodos estaciones de trabajo conectados a cada nodo servidor.

Problema nº2.

Se define el valor de trayectoria pesada de una hoja de un árbol binario como la suma del contenido de todos los nodos desde la raíz hasta la hoja multiplicada por el nivel en el que se encuentra.



trayectoria pesada hoja 4: 4x3 + 1x2 + 7x1 = 21
trayectoria pesada hoja 2: 2x3 + 3x2 + 7x1 = 19
trayectoria pesada hoja 1: 1x3 + 3x2 + 7x1 = 16

Implementar un procedimiento que, dado un árbol binario, muestre en pantalla el valor de trayectoria pesada de cada una de sus hojas.

NOTA: Cada nodo del árbol solo puede ser visitado UNA ÚNICA VEZ.

Junio 99

Problema nº1.

Sea G un grafo conexo, valuado y no dirigido con **n** vértices, numerados de 1 a n. Supongamos que todos los pesos son positivos. El proceso de determinar el árbol de recubrimiento **AR** de coste mínimo de G, puede realizarse de una manera eficiente mediante el siguiente algoritmo:

Sea **A**, una matriz de tipo TipoArcos, donde los arcos de G aparecen ordenados crecientemente por peso.

Utilizamos un vector **PARTICION** que indica a qué componente conexa del grafo pertenece cada vértice, puesto que lo que hacemos es asignar cada vértice a una componente conexa, adoptandose como criterio que cada componente se identifica por el valor de su menor elemento.

Al inicio del algoritmo, dispondremos en **AR** de todos los vértices y ningún arco, por lo cual, cada uno de los vértices está asignado a una partición distinta (la que constituye el propio vértice aislado), es decir, al inicio, **PARTICION[i]=i, ∀ i ∈ {1..n}.**

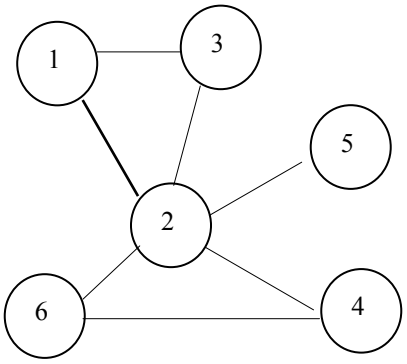
En cada paso, se procesa un arco extraido de A, el cual se va añadiendo al arbol de recubrimiento si y sólo si **PARTICION**[i] ≠ **PARTICION**[j].

Conforme se van añadiendo los arcos para formar el árbol de recubrimiento, el número de valores diferentes de **PARTICION** (particiones) va disminuyendo, hasta que al final, **PARTICION** [i]=1, **∀ i ∈ {1..n}.**

Cuando se añade un arco que conecta dos particiones, a los elementos de la mayor partición se les asigna el valor de la menor.

Se pide:

a) Explicar el proceso descrito mediante el siguiente grafo.



b) Utilizando técnicas de programación orientada a objetos, construir el algoritmo anterior, el cual, a partir de un grafo G como el descrito, obtiene un árbol de recubrimiento mínimo.

c) Razonar claramente por qué el algoritmo anterior proporciona un comportamiento correcto, y explicar cómo se consigue la no existencia de ciclos en el árbol de recubrimiento.

NOTA: Para resolver el ejercicio, pueden utilizarse todos los métodos de la clase grafo y conjunto, sin tener que implementarlos. De la misma forma, dado un arco **a**, **a.origen** y **a.destino** referencian al arco (origen, destino)

Problema nº2.

Un montículo es un arbol binario **completamente lleno, con la posible excepción del último nivel, el cual se llena de izquierda a derecha**, y en el que para cada nodo **p** ocurre que el valor de p es menor que el valor de sus dos hijos. Construir un algoritmo de **complejidad lineal** que, haciendo uso de técnicas de POO, reciba un arbol binario y retorne un valor booleano indicando si el árbol es o no un montículo.

Septiembre 99

Problema nº1.

Sea G un grafo no dirigido con **n** vértices, numerados de 1 a n. Sabemos que el proceso de determinar las componentes conexas de G puede realizarse de una manera eficiente mediante el siguiente algoritmo:

Utilizamos un vector **P**, que al inicio del algoritmo toma los valores **P[i]=i, ∀ i ∈ {1..n}.**

Al final del proceso, **P** contendrá tantos valores diferentes como componentes conexas tenga el grafo G, de modo que **P[i]=P[j]** si y sólo si **i y j** pertenecen a la misma componente conexa.

Se adopta como criterio que el valor final de P para todos los elementos de una componente conexa se corresponde con el valor del menor elemento de dicha componente.

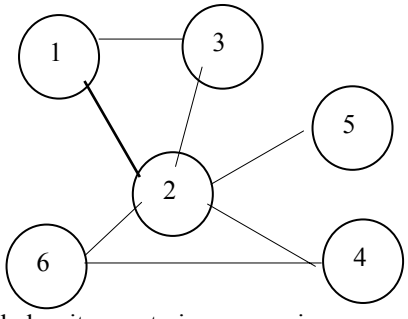
En cada paso, a medida que se procesan los arcos, el número de valores diferentes de **P** va disminuyendo.

Bien; pues resulta que siguiendo los mismos pasos que en el algoritmo anterior, en un momento dado, el algoritmo puede informar sobre la **existencia de CICLO** en el grafo G y del **arco** que provoca esa existencia. Lo importante es determinar cuál es ese momento y la condición detectada que indica la existencia de ciclo.

Se pide:

a) Utilizando técnicas de programación orientada a objetos y siguiendo el proceso anterior, construir un algoritmo que a partir de un grafo no dirigido G, indique la existencia o no de ciclo en el grafo. El algoritmo construido deberá devolver también, al **menos un arco** a partir de el cual se forme el ciclo.

b) Explicar el proceso seguido mediante un ejemplo utilizando el siguiente grafo. El arco a devolver puede ser cualquiera de los 6 que provocan la existencia de ciclo.



c) Razonar claramente porqué el algoritmo anterior proporciona un comportamiento correcto.

NOTA: Para resolver el ejercicio, pueden utilizarse todos los métodos de la clase grafo y conjunto, sin tener que implementarlos. De la misma forma, dado un arco a, a.origen y a.destino referencian al arco (origen, destino)

Problema nº2.

Frecuentemente, cuando utilizamos **conjuntos** en el diseño de algoritmo, no necesitamos operaciones tan potentes como la unión, intersección o diferencia. A menudo, únicamente necesitamos mantener un conjunto de valores con inserciones y eliminaciones periódicas, además de conocer si un elemento en particular está o no en el conjunto. Este tipo de estructuras reciben el nombre de **DICCIONARIOS**.

Además, sabemos que dentro de las posibles representaciones de conjuntos, no podemos considerar como válida un vector booleano cuando el cardinal del conjunto universal es excesivamente grande, teniéndonos que decantar por otras alternativas de ofrecen complejidad lineal en esas operaciones básicas: inserción, extracción y pertenencia.

Sin embargo, sería posible utilizar otra estructura de datos de manera que el tiempo invertido en esas operaciones básicas estuviese comprendido entre $\Omega(\log n)$ y $O(n)$.

Se pide:

a) Implementar la clase Diccionario utilizando **mecanismo de herencia** y justificar porqué la superclase elegida permite que las operaciones básicas del Diccionario mantengan complejidades comprendidas entre $\Omega(\log n)$ y $O(n)$.

b) Añadir una operación *BuscarRango* a la clase Diccionario. Esta operación debe imprimir todos aquellos valores **x** del diccionario tales que $\mathbf{k_1 \leq x \leq k_2}$, donde k_1 y k_2 son dos claves que verifican que $k_1<k_2$

Diciembre 99

Ejercicio 1.

La compañía telefónica de un determinado país dispone de una red de comunicaciones diseñada de tal forma que **hay un único nodo** que si falla hace que algunos nodos o grupos de nodos de la red no puedan comunicarse con otros nodos o grupos de nodos.

Se pide implementar una función que dado un grafo que es representación de esta red de comunicaciones nos liste los nodos o grupos de nodos han quedado aislados. Se supone que los nodos están numerados de 1 a N.

Ejemplo de listado:

Grupo 1: 1,2,3
Grupo 2: 4
Grupo 3: 6,7
...

Ejercicio 2.

En Inteligencia Artificial existe un algoritmo que propaga información hacia arriba en un árbol binario equilibrado de la siguiente forma :

- Si estamos en una hoja, se propaga el valor que devuelva una cierta función heurística H.

- Si estamos en nodos situados en niveles impares, se propaga el máximo valor de sus nodos inmediatamente sucesores.
- Si estamos en nodos situados en niveles pares, se propaga el mínimo valor de sus nodos inmediatamente sucesores.

Se pide : Haciendo uso de técnicas de P.O.O., construir una función que implemente este algoritmo y devuelva el valor que se ha propagado hasta la raíz de un cierto árbol. El prototipo debe ser:

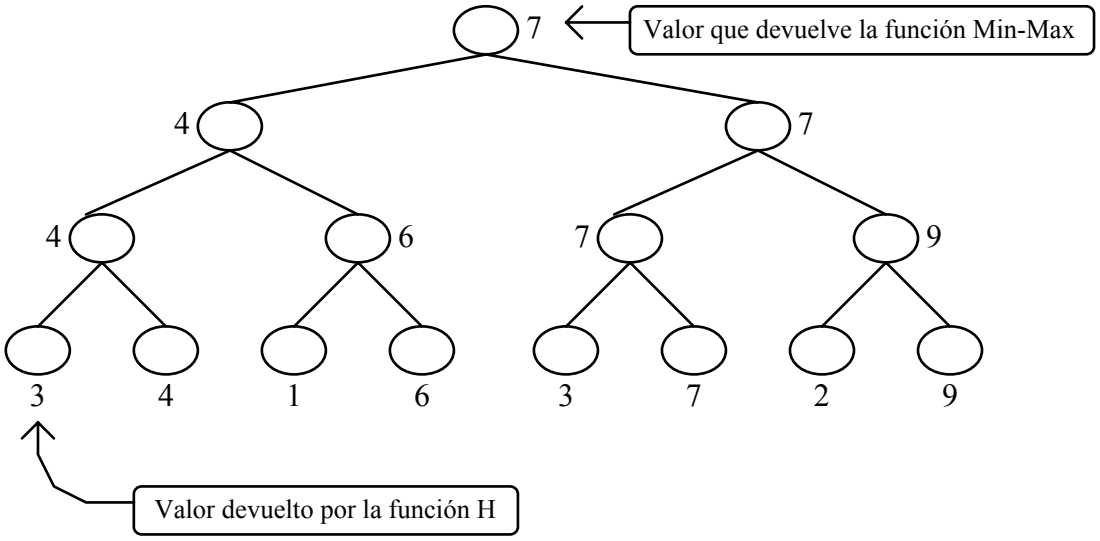
```
Function MinMax (A: Arbol; Nivel : Integer) : Integer;
```

La función construida deberá tener una complejidad LINEAL.

NOTA : Se puede hacer uso de las siguientes funciones sin necesidad de implementarlas (siempre que se respete su sintaxis) :

```
Function H (A: Arbol) : Integer;          { Devuelve el valor con que se etiqueta la hoja A }
Function Maximo (a,b: Integer) : Integer; { Devuelve el máximo de a y b }
Function Minimo (a,b: Integer) : Integer; { Devuelve el mínimo de a y b }
Function Impar (n : Integer) : Boolean;   { Devuelve true si n es un número impar }
```

Ejemplo :



La llamada a la función MinMax (A,1) para el caso del ejemplo devolverá 7.

Febrero 00

Problema nº1.

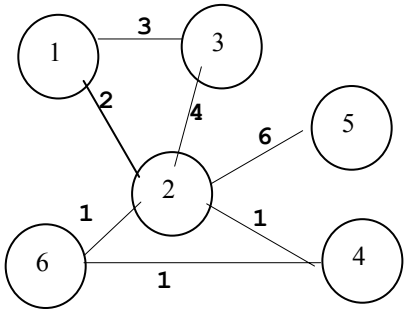
Sea G un grafo conexo, valuado y no dirigido con **n** vértices, numerados de 1 a n, y supongamos que todos los pesos son positivos. Sabemos que el proceso de determinar las componentes conexas de G puede realizarse de una manera eficiente mediante el siguiente algoritmo:

Utilizamos un vector **P**, que al inicio del algoritmo toma los valores **P[i]=i, $\forall$ i $\in$ {1..n}.**
Al final del proceso, **P** contendrá tantos valores diferentes como componentes conexas tenga el grafo G, de modo que **P[i]=P[j]** si y sólo si **i y j** pertenecen a la misma componente conexa.
Se adopta como criterio que el valor final de P para todos los elementos de una componente conexa se corresponde con el valor del menor elemento de dicha componente.
En cada paso, a medida que se procesan los arcos, el número de valores diferentes de **P** va disminuyendo.

Bien; pues resulta que siguiendo pasos análogos al algoritmo anterior, es fácil determinar el árbol de recubrimiento de coste mínimo del grafo G. Para simplificar, podemos suponer que el procesamiento de los arcos se realiza de forma creciente por pesos.

Se pide:

a) Utilizando técnicas de programación orientada a objetos, construir el algoritmo anterior el cual, a partir de un grafo G como el descrito, obtiene un árbol de recubrimiento mínimo.
b) Explicar el proceso seguido mediante un ejemplo utilizando el siguiente grafo. Observar que el árbol de recubrimiento de coste mínimo NO tiene porqué ser único.



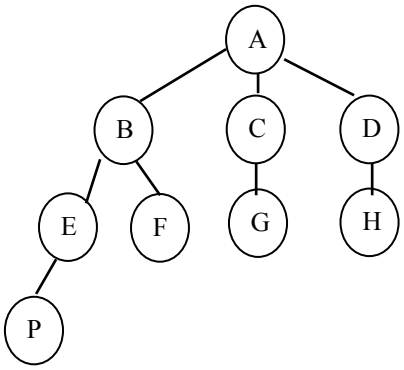
c) Razonar claramente porqué el algoritmo anterior proporciona un comportamiento correcto, y explicar cómo se consigue la no existencia de ciclos en el árbol de recubrimiento.

NOTA: Para resolver el ejercicio, pueden utilizarse todos los métodos de la clase grafo y conjunto, sin tener que implementarlos. De la misma forma, dado un arco a, a.origen y a.destino referencian al arco (origen, destino)

Problema nº2.

Sea A un árbol binario que es una representación de un árbol general G. Utilizando técnicas de POO, construir un algoritmo de complejidad lineal sobre el número de nodos que imprima el árbol G por niveles, imprimiendo además, para cada conjunto de hermanos, cuál es su padre.

Ejemplo
El nodo A tiene por hijos: B, C, D
El nodo B tiene por hijos: E, F
El nodo C tiene por hijos: G
El nodo D tiene por hijos: H
El nodo E tiene por hijos: P
El nodo F es una hoja
El nodo G es una hoja
El nodo H es una hoja
El nodo P es una hoja

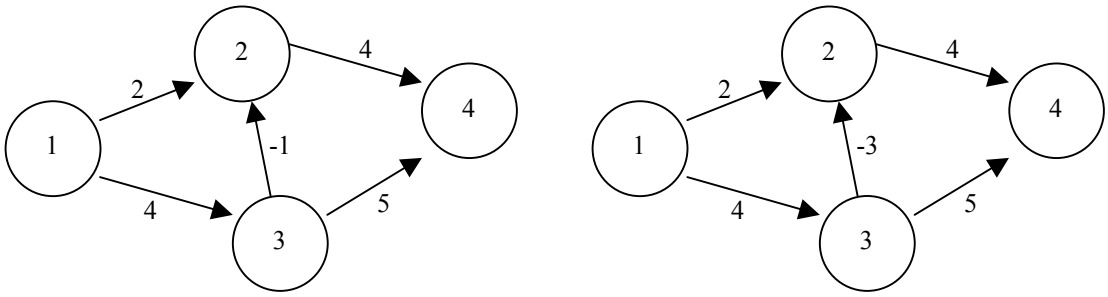


Junio 00

Problema nº1.

Ya conoces el algoritmo de Dijkstra para el cálculo de caminos mínimos en grafos. Como sabes, este algoritmo permite, con un coste $O(n^2)$, calcular el coste del camino mínimo entre un nodo determinado y cualquier otro nodo del grafo. Como también sabes, una de las limitaciones de este algoritmo es que no es apto para ser utilizado sobre grafos con pesos negativos. De cualquier forma esto no significa que Dijkstra no funciona con ningún grafo con pesos negativos. A continuación se te presentan dos ejemplos de grafos A y B con pesos negativos. Tú mismo puedes comprobar que tomando en ambos casos el vértice 1 como vértice de partida, el algoritmo funciona para el grafo A y no funciona para el grafo B.

GRAFO A	GRAFO B



El ejercicio que se te propone es modificar el algoritmo de Dijkstra para que además de proporcionar los caminos mínimos desde un vértice dado al resto de vértices, detecte si el grafo sobre el que está trabajando es apto o no para producir un resultado correcto. Para ello:

a) Realiza una traza del algoritmo de Dijkstra sobre los grafos A y B y muestra porqué el algoritmo no funciona sobre el grafo B.
b) Describe y razona cual es la condición bajo la cual puede detectarse que el grafo sobre el que se ejecuta el algoritmo no es apto para producir un resultado correcto.
c) Escribe el algoritmo modificado para que además de calcular el camino mínimo de un vértice determinado al resto de vértices detecte si el grafo al que está siendo aplicado es apto o no para producir un resultado correcto. En caso de que el grafo no sea apto el algoritmo debe devolver el vector de salida con todas las distancias a infinito.

AYUDA: Recuerda que la regla en la que se basa el algoritmo de Dijkstra es que en la iteración K-ésima ya se han visitado K nodos **y se asegura que las distancias mínimas calculadas para esos K nodos ya son definitivas**. Sin embargo esto no tiene porqué cumplirse cuando existen en el grafo arcos con pesos negativos.

Problema nº2.

Una gran cadena de centros comerciales, mantiene toda la información referente a sus empleados organizada en una estructura jerárquica, en la cabeza de la cual se encuentra el director general del que dependen los gerentes provinciales, cada uno de los cuales tiene a su cargo varios jefes de sección, que a su vez se encargan de organizar el trabajo de otros empleados y así sucesivamente. Para cada empleado, la información que se almacena en la estructura es, exclusivamente, nombre y apellidos del empleado y sueldo base.

El sueldo definitivo que cobra cada empleado es proporcional al sueldo base. La empresa mantiene en una tabla indexada del 1 a n el factor de corrección a aplicar a cada categoría profesional. Se entiende 1 como la categoría más alta (director general), 2 la siguiente (gerentes provinciales), y así sucesivamente.

El programa que calcula las nóminas a fin de mes empieza calculando la del director general, a continuación la de cada gerente provincial, luego las de los jefes de sección, etc., de forma que sólo se empiezan a calcular las nóminas de los empleados de una categoría cuando se han calculado las nóminas de todos los jefes inmediatamente superiores.

SE PIDE: Haciendo uso de técnicas de POO, construir un algoritmo de generación de nóminas. Razona qué estructura de datos sería la idónea para almacenar la información, y el método seguido para obtener el sueldo definitivo de cada empleado.

Septiembre 00

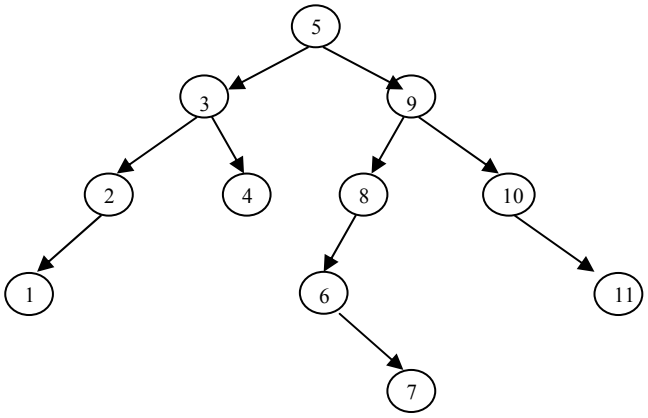
Problema nº1.

Una comunidad autónoma decide instaurar una nueva escuela para cubrir las necesidades escolares de un conjunto de pueblos. Todos los alcaldes desean que la nueva escuela se edifique en su pueblo, pues esto supondría que los niños de ese pueblo, no tendrían que desplazarse a ningún otro sitio. Sin embargo, se opta por la solución más justa: la nueva escuela se construirá en aquél pueblo que esté menos lejos de todos los restantes.

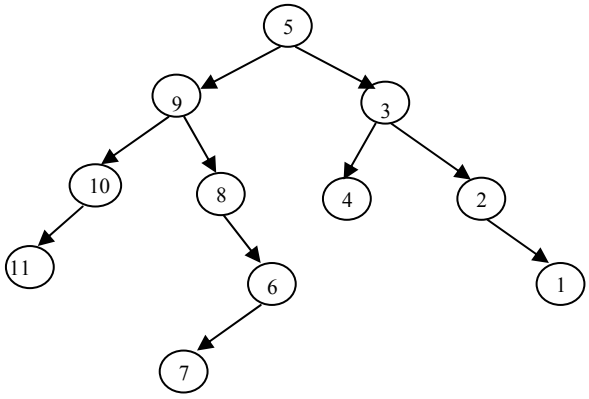
SE PIDE: Construir un algoritmo que permite decidir cuál es el pueblo que cumple ese requisito. Justifica tu solución indicando claramente cuál es el criterio que se ha seguido para elegir un pueblo frente a otro.

Problema nº2.

Podemos decir que todo árbol binario ordenado puede tener su correspondiente árbol binario "odanedro" (ordenado al revés). Llamamos árbol "odanedro" de un árbol binario ordenado a aquel árbol que lo refleja como si de un espejo se tratara.
SE PIDE: implementar el algoritmo que crea el árbol "odanedro" de un árbol binario ordenado.
Implementar además un algoritmo que nos permita recorrer el árbol "odanedro" de forma que visitemos todos sus nodos en orden ascendente.



Arbol Binario Ordenado



Arbol odanedro

Calcular el coste para los dos algoritmos implementados.

Junio 01 Problema 1

Te proponemos que intentes recomponer tu genealogía, aplicando los conocimientos adquiridos en la asignatura. **Sólo** te interesa conocer el nombre de tus ascendientes directos (padres, abuelos, bisabuelos, etc.), **no** otro tipo de relaciones (matrimonios, hermanos, primos, consuegras, etc.). Claro, toda esta información debes almacenarla en una estructura de datos que te permita, por ejemplo, mostrar todos tus ascendientes, buscar uno determinado, buscar el abuelo de tu bisabuela, etc.

En principio, piensas hacerlo sólo para ti, pero como ves que el asunto le interesa a otros familiares, estudias la posibilidad de utilizar una estructura más general, donde puedas englobar las genealogías de tus familiares y amigos (y así podríais saber si tenéis algún tatarabuelo común).

Algunas consideraciones para seleccionar la estructura adecuada:

- Las operaciones normales se basan en la obtención o búsquedas de ascendientes, no de descendientes.
- No hay que conocer a todos los ascendientes necesariamente. Por ejemplo, por la rama paterna podemos tener sólo información de nuestros abuelos, pero por la materna podemos conocer hasta 5 generaciones atrás.
- Puede darse el caso de que una determinada persona sea ascendiente tuyo por dos motivos. Por ejemplo, si tus padres fueran primos, tus abuelos serían hermanos, y por tanto, tendrías bisabuelos comunes por parte materna y paterna. . Es decir, tus padres tienen un ascendiente común. A este tipo de ascendientes los llamaremos ***"biascendientes"***, ya que podemos llegar a ellos por dos (o más) ramas familiares distintas.

1. Explica brevemente qué estructura de datos emplearías, según los dos siguientes casos:
   a) Genealogía de una sola persona, suponiendo que no hay posibilidad de *"biascendientes"*.
   b) Genealogía que nos permita trabajar con más de una persona, con *"biascendientes"*.

Nota: caracterizar la estructura tanto como sea posible. Es decir, si la estructura fuera una lista, decir el tipo de los elementos, si la inserción es ordenada, etc.; si fuera un árbol, el tipo de los elementos, el grado, si es o no ordenado, etc; y así para el resto de las estructuras. Ayúdate de un ejemplo si es necesario.

2. Para el caso **b**, escribir un procedimiento (empleando técnicas de POO) que, dado el nombre de una persona **p**, diga si p tiene, entre los antepasados conocidos, algún *"biascendiente"*.

Junio 01 Problema 2

Frecuentemente, cuando utilizamos **conjuntos** en el diseño de algoritmos, no necesitamos operaciones tan potentes como la unión, intersección o diferencia. A menudo, únicamente necesitamos mantener un conjunto de valores con inserciones y eliminaciones periódicas, además de conocer si un elemento en particular está o no en el conjunto. Como todos sabemos, este tipo de estructuras reciben el nombre de **DICCIONARIOS**.
Además, sabemos que existen muchas alternativas distintas para representar Conjuntos y, por consiguiente, diccionarios. Entre ellas, destacan el uso de un vector booleano, listas enlazadas o incluso un árbol binario ordenado. Este ejercicio trata de una nueva representación: el uso de "hashing abierto", el cuál se describe a continuación.

La idea esencial que subyace en el hashing abierto es que los (posiblemente infinitos) miembros potenciales del conjunto se particionan en un número finito de clases disjuntas. Si queremos tener **N** clases numeradas **0, 1, 2, …, n-1**, se utiliza una función hash **h** tal que a cada elemento ***x*** del tipo de datos de los miembros del conjunto, **h(*x*)** es un entero entre **0** y **n-1**. Como podemos observar, la función **h** es una función sobreyectiva. A ***x*** se le suele denominar *clave* y **h(*x*)** es el valor hash de *x*. Así, por ejemplo, si el tipo de datos de los miembros del conjunto son los números naturales, $\aleph$, una función hash típica podría ser el **módulo,** donde $\forall x \in \aleph$, $h(x) = x \bmod n$, proporciona valores entre *0* y *n-1*. Como podemos observar, el valor de **n** es conocido a priori.

Se pide:

c) Decide y justifica convenientemente cuál sería la estructura de datos que utilizarías para representar diccionarios mediante funciones hash. Razona en términos de eficiencia en tiempo y espacio.

d) Haciendo uso de clases estudiadas en la asignatura, proporciona la definición de la clase Diccionario, e implementa las operaciones de pertenencia y borrado, de acuerdo a la siguiente especificación.

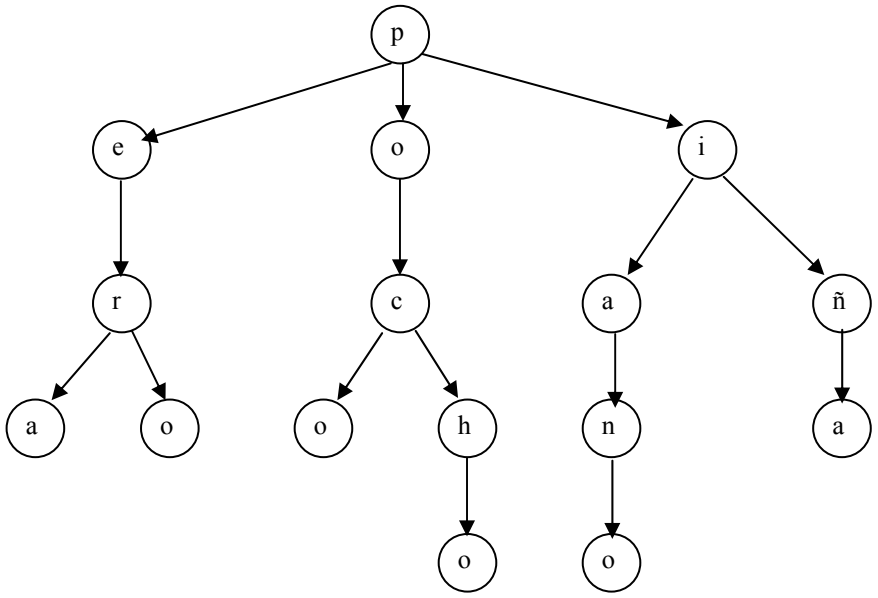
$\forall$ A, B$\in$ Conjunto; $\forall$ i,j $\in$ Elemento

Pertenece (Crear, i)	$\Rightarrow$	Falso
Pertenece (Insertar (A,i), j)	$\Rightarrow$	Si i=j → Cierto
si no → Pertenece (A, j)		
Borrar (Crear, i)	$\Rightarrow$	Crear
Borrar (Insertar (A,i), j)	$\Rightarrow$	Si i=j → Borrar (A, j)
si no → Insertar (Borrar (A, j), i)		

Utilizar una función hash genérica H para resolver el problema, donde
$\forall$ x $\in$ Elemento, h(x) $\in$ {0..n-1}

Septiembre 01 Problema 1

Vamos a trabajar con árboles ternarios, es decir, con árboles de grado 3. En general, son similares a los árboles binarios, pero con tres hijos en lugar de dos.



a) Escribir la especificación algebráica de las siguiente operaciones del TAD ArbolTernario:

TIPO: ArbolT (Elemento)
SINTAXIS

Crear ( ) ⇒ ArbolT
Vacio (ArbolT) ⇒ Booleano
Construir(Elemento, ArbolT, ArbolT, ArbolT) ⇒ ArbolT
Raiz (ArbolT) ⇒Elemento
HijoIzqdo(ArbolT) ⇒ ArbolT
HijoCentral(ArbolT) ⇒ ArbolT
HijoDerecho(ArbolT)⇒ ArbolT

b) Dado un árbol ternario de caracteres, escribir un algoritmo (utilizando POO) que devuelva una lista con las palabras formadas a partir de todos los caminos posibles desde la raíz hasta cada una de las hojas. Por ejemplo, para el árbol ternario de la figura anterior, el algoritmo debería devolver la lista [‘pera’, ‘pero’, ‘poco’, ‘pocho’, ‘piano’, ‘piña’]. (El orden en que aparezcan las palabras en la lista no tiene importancia.)

Se pueden utilizar, sin necesidad de implementarlos, el TAD Lista y/o String (Cadena de caracteres) con las operaciones habituales.

Septiembre 01 Problema 2

Queremos definir una estructura adecuada para almacenar elementos sobre los que hay definida una relación de equivalencia ≡. Las operaciones necesarias son las siguientes:

Crear(S)	Devuelve la estructura S vacía.
Añadir(x,y,S)	Inserta x en S (si ya pertenecía, no lo vuelve a insertar), Inserta y en S (si ya pertenecía, no lo vuelve a insertar), Hace que x e y sean equivalentes.
Pertenece(x,S)	Devuelve TRUE si x pertenece a S.

Equivalencia(x,S) Devuelve $\{y \in S \mid x \equiv y\}$,

Es decir, la operación Equivalencia(x,S) devuelve en un conjunto todos los elementos de S que son equivalentes a x. Dicho de otra forma, devuelve todos los elementos que se encuentran en la misma clase de equivalencia que x.

a) Explica razonadamente qué estructura de datos se considera más adecuada para almacenar estos grupos de elementos con relaciones de equivalencia. Pon un ejemplo.

b) ¿A qué métodos del TAD elegido se corresponden las operaciones Crear, Añadir y Pertenece?

c) Explica e implementa la operación Equivalencia.

d) Según la estructura de datos elegida, ¿se podría determinar cuántas clases de equivalencia hay en un grupo dado? Explícalo sin implementarlo.

Febrero 02

Problema 1

Se está pensando en volver a cambiar los planes de estudio de las carreras de la Universidad, y te han encargado que analices el problema para que todo salga bien.

En estos nuevos planes, todas las asignaturas son cuatrimestrales, y muchas asignaturas tienen prerequisitos, es decir, para poder matricularse de algunas asignaturas, hay que aprobar antes otras. Por ejemplo, para matricularse de EDA habría que haber aprobado Álgebra y Programación 2. Y para aprobar Programación 2, hay que tener Programación1.

Para representar un plan de estudios como éste, parece que un grafo es una estructura de datos apropiada.

1. Caracteriza cómo debería ser un grafo para representar un plan de estudios: dirigido o no, etiquetado o no, qué representan los vértices y las aristas, etc.
2. ¿Debe ser un grafo cíclico o acíclico? ¿Por qué?

3. Utilizando técnicas de programación orientada a objetos, escribir una operación que devuelva una cola con todas las asignaturas ordenadas de manera que se cumplan todos los prerequisitos (es lo que se conoce como *orden topológico*).
   Para el ejemplo anterior, un posible orden sería:
   Álgebra, Programación 1, Programación 2, EDA
   y también:
   Programación 1, Álgebra, Programación 2, EDA
   o también:
   Programación 1, Programación 2, Álgebra, EDA
   (Un recorrido en anchura puro **no** es adecuado. En el ejemplo anterior, un recorrido en anchura podría dar como resultado Álgebra, EDA, Programación 1, Programación2.)
4. Si para el apartado anterior se necesita una o más funciones auxiliares que no pertenezcan al TAD Grafo visto en clase, explica cómo habría que implementarlas si estamos usando una matriz de adyacencia para representar el grafo.

Problema 2

Dados dos vectores A1 y A2, y sabiendo que A1 contiene los valores resultantes de recorrer un árbol binario en preorden y A2 contiene el resultado de un recorrido en inorden del mismo árbol. Utilizando técnicas de programación orientada a objetos, desarrollar un algoritmo que nos permita construir el árbol binario a partir de los vectores A1 y A2, de tamaño n, siendo n el número de nodos del árbol. (Suponemos que no hay valores repetidos en el árbol.)

Ejemplo: Dados dos vectores,
PREorden=[7, 8, 3, 18, 5, 4, 15, 11, 9, 2, 1, 10] ,
INorden=[18, 3, 5, 8, 4, 7, 9, 11, 2, 15, 1, 10]
se debería construir el siguiente árbol: