**PREGUNTAS DE SIMULACIÓN PARA EL EXAMEN FINAL DE ESTRUCTURA DE DATOS**

## ÁRBOLES

1) Reconstruir un árbol binario a partir de los siguientes recorridos:

   Preorden: 4, 7, 2, 1, 5, 3, 8, 6

   Postorden: 3, 4, 2, 7, 5, 6, 1

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

2) Supongamos que tenemos una función *valor* tal que dado un valor de tipo *char* (una letra del alfabeto) devuelve un valor entero asociado a dicho identificador. Supongamos también la existencia de un árbol de expresión *T* cuyos nodos hoja son letras del alfabeto y cuyos nodos interiores son los caracteres *, +, -, /. Diseñar un programa en Python que tome como parámetros un nodo y un árbol binario y devuelva el resultado entero de la evaluación de la expresión representada.

   Algoritmo sugerido:

```
int Evalua(NodoB n,ArbolB T)
{
  char ident;

  EtiquetaArbolB(&c,n,T);
  switch(c){
    case '+':
      return Evalua(HijoIzqdaB(n,T),T)+Evalua(HijoDrchaB(n,T),T);
      break;
    case '-':
      return Evalua(HijoIzqdaB(n,T),T)-Evalua(HijoDrchaB(n,T),T);
      break;
    case '*':
      return Evalua(HijoIzqdaB(n,T),T)*Evalua(HijoDrchaB(n,T),T);
      break;
    case '/':
      return Evalua(HijoIzqdaB(n,T),T)/Evalua(HijoDrchaB(n,T),T);
      break;
    default:
      return valor(c);
  }
```

3) Problema a Resolver: Repaso de conceptos previo a la pregunta a resolver.

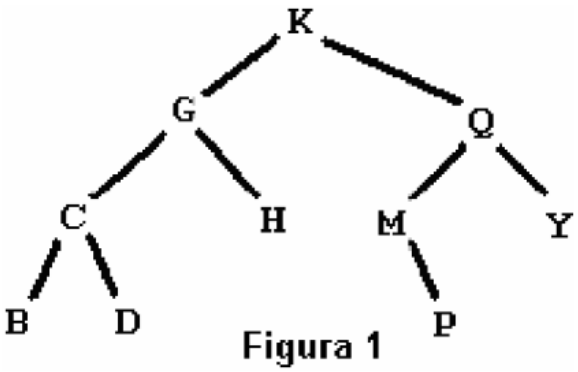


Figura 1

La Figura 1 muestra la representación gráfica de un árbol binario de letras. Lo familiarizados con los árboles binarios pueden saltarse la definición de árbol binario de letras, hojas de un árbol binario, y búsqueda en un árbol binario de letras, e ir directo al problema.

**Definición**.

Un *árbol binario de letras* puede ser una de dos cosas:

1. Puede estar vacía.
2. Puede tener un nodo raíz. Un nodo tiene una letra como dato y hace referencia a subárboles izquierdo y derecho. Los subárboles izquierdo y derecho son también árboles binarios de letras.

En la representación gráfica de un árbol binario de letras:

1. Un árbol vacío es omitido completamente.
2. Cada nodo esta indicado por
   - Su dato letra,
   - Un segmento de línea abajo a la izquierda hacia su subárbol izquierdo, si el subárbol izquierdo no está vacío,
   - Un segmento de línea abajo a la derecha hacia su subárbol derecho, si el subárbol derecho no esta vacío.

Una *hoja* en un árbol binario es un nodo donde ambos subárboles están vacíos. En el ejemplo en la Figura 1, tiene cinco nodos con datos B, D, H, P, y Y.

El *recorrido preorder de un árbol de letras* satisface las propiedades:

1. Si el árbol esta vacío, entonces el recorrido preorder está vacío.
2. Si el árbol no esta vacío, entonces el recorrido preorder consiste en lo siguiente, en orden:
   - El dato del nodo raíz,
   - El recorrido preorder del subárbol izquierdo del nodo raíz,
   - El recorrido preorder del subárbol derecho del nodo raíz.

El recorrido preorder del árbol de la Figura 1 es KGCBDHQMPY.

Un árbol como el de la Figura 1 es también un árbol binario de búsqueda de letras. Un árbol binario de búsqueda de letras es un árbol de letras en el cual cada nodo satisface:

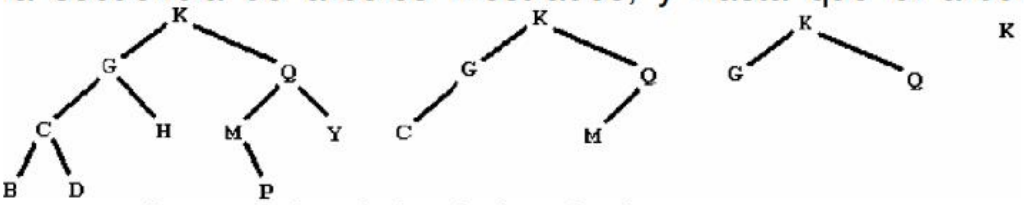
1. Los datos raíz vienen después en el alfabeto que todos los datos en los nodos en el subárbol izquierdo.
2. Los datos raíz vienen antes en el alfabeto que todos los datos en los nodos en el subárbol derecho.

**EL PROBLEMA A RESOLVER ES EL SIGUIENTE:**

Considere la siguiente secuencia de operaciones en un árbol binario de búsqueda de letras:

Borrar las hojas y listar los datos removidos
Repetir este proceso hasta que el árbol este vacío.

Empezando por el árbol de abajo a la izquierda, producimos la secuencia de árboles mostrados, y hasta que el árbol




este vacío removiendo las hojas de datos

BDHPY
CM
GQ
K

**EXPLICACIÓN ADICIONAL:**

Tu problema es empezar con tales secuencias de líneas de hojas de un árbol binario de búsqueda de letras y desplegar el recorrido preorder del árbol.
La entrada contiene uno o más sets de datos. Cada set de datos es una secuencia de uno o más líneas con letras mayúsculas. Las líneas contienen las hojas removidas del árbol binario de búsqueda de la forma descrita anteriormente. Las letras en una línea están listados en orden alfabético. Los sets de datos están separados por una línea que contiene un asterisco ('*'). El último set de datos está seguido por un signo de dólar ('$'). No hay espacios en blanco ni líneas vacías en la entrada.

Por cada set de datos de entrada, hay un único árbol binario de búsqueda que puede ser producido con la secuencia de hojas. La salida es una línea que contiene solo el recorrido preorder del árbol, sin blancos.

## Ejemplo de entrada

```
BDHPY
CM
GQ
K
*
AC
B
$
```

## Ejemplo de salida

```
KGCBDHQMPY
BAC
```

**PILAS Y COLAS**

1) Atención a los postulantes a un trabajo en una empresa con varios entrevistadores.

    1. Escribir una clase ColaDePostulantes, con los métodos nuevo_postulante, que reciba el nombre del postulante y lo encole, y un método proximo_postulante que devuelva el primer postulante en la cola y lo desencole.
    2. Escribir una clase Entrevista, que contenga el método nuevo_postulante que reciba como parámetros: el nombre del postulante, nombre de la persona que lo entrevistará, y la oficina en la cual será entrevistado; y proximo_postulante, que reciba el nombre de la persona liberada y devuelva el próximo postulante en espera.
    3. Escribir un programa en Python que permita al usuario ingresar nuevos postulantes o indicar que oficina se ha liberado, y en ese caso imprima el próximo postulante en espera.

2) Desarrollar un programa en Python que cargue por teclado una cadena de caracteres y use una pila para determinar si esa cadena es capicúa: esto es, queremos determinar si una cadena de caracteres que se recibe para analizar, puede leerse igual en dirección izquierda-derecha que en dirección derecha-izquierda.

    **Sugerencia:** Usando una pila la solución es directa: la idea es que se guardan en la pila los caracteres de mitad izquierda de la palabra, uno a uno tomados en el mismo orden en que aparecen en la cadena. Al guardarlas en la pila, el orden se invierte: el carácter que entró primero se va al fondo de la pila, y el que entró último queda al frente. Luego se recorre la segunda mitad de la cadena, y por cada carácter que se tenga, se extrae a su vez un carácter de la pila. Si la cadena era capicúa, los caracteres extraídos de la pila deberán coincidir siempre con los caracteres que se recorren en la segunda mitad de la cadena. Si algún par de caracteres no coincide, la palabra no es CAPICÚA, y el proceso se detiene.

**LISTAS**

1) Se dispone de dos listas enlazadas construidas dinámicamente con nodos del mismo tipo base y de igual longitud. Realizar un programa en Python que de ambas listas solo imprima los números enteros. Y además también saque el promedio de la primera lista y que muestre de la segunda lista solo aquellos que sean mayores al promedio de la primera lista.

2) Se dispone de dos listas enlazadas construidas dinámicamente con nodos del mismo tipo base y de igual longitud. Realizar un programa en Python que de ambas listas elimine solo aquellos valores que sean iguales a un número dado. Y que también muestre de cada lista el número de nodos eliminados.