Reto 4: Árboles

Francisco David Charte Luque Ignacio Cordón Castillo

1 Inorden no recursivo

Diseñar un procedimiento inorden no recursivo a imagen y semejanza del procedimiento preorden no recursivo que el profesor diseñó en la clase.

```
void inordenNR(const ArbolBinario<int>& a){
  ArbolBinario<int>::Nodo actual;
  stack<ArbolBinario<int>::Nodo> p;
  bool subiendo = false;

  actual=a.raiz();
  p.push(0);
  p.push(actual);

  while (actual != 0){
      if (a.izquierda(actual) != 0 && !subiendo){
          // Pasamos a manejar el hijo izquierdo
          actual = a.izquierda(actual);
          p.push(actual);
      } else {
          cout << a.etiqueta(actual) << ' ';
          p.pop();
          subiendo = true;

          if (a.derecha(actual) != 0) {
              // Pasamos a manejar el hijo derecho
              actual = a.derecha(actual);
              p.push(actual);
              subiendo = false;
          } else {
              // Trataremos de saltar al hermano
              actual = p.top();
          }
      }
```

```
    }
}
```

2 Codificación de árbol binario

Dar un procedimiento para guardar un árbol binario en disco de forma que se recupere la estructura jerárquica de forma unívoca usando el mínimo número de centinelas que veais posible.

Se propone lo siguiente:

Deseamos conocer para cada nodo del árbol si tiene dos hijos, sólo el izquierdo, solo el derecho o ninguno. Se empleará el preorden del árbol binario, haciendo que a cada nodo le preceda, caso de ser necesario, uno de los tres centinelas siguientes:

i. `<` Si le falta el hijo izquierdo

ii. `>` Si le falta el hijo derecho

iii. `-` Si no tiene hijos

No se hará empleo de ningún centinela si el nodo tiene ambos hijos.

Algoritmo 1 ALGORITMO DE RECUPERADO DEL ÁRBOL

Entrada:

`bin_tree`, Arból binario leído codificado

LLevamos una estructura pila llamada `nodos` y otra llamada `hijos`

para Cada uno de los caracteres de bin_tree
  si El carácter actual es `-`,`>` o `<` **entonces**
    Lo introducimos en `hijos`
  en otro caso
    Lo introducimos en `nodos`
    Introducimos * en `hijos`
  fin si
fin para

Creamos un nodo `raiz`
LLamamos al siguiente algoritmo con `raiz` como `actual`

si Tope `hijos` $\neq$ `-` **and** Tope `hijos` $\neq$ `<` **entonces**
  Hijo izquierda de `actual` es el tope de `hijos`
  Sacamos el último elemento de `hijos` y de `nodos`
  LLamamos al algoritmo para hijo izquierda de `actual`
en otro caso
  Hijo izquierda de `actual` no existe
fin si

si Tope `hijos` $\neq$ `-` **and** Tope `hijos` $\neq$ `>` **entonces**
  Hijo derecha de `actual` es el tope de `nodos`
  Sacamos el último elemento de `hijos` y de `nodos`
  LLamamos al algoritmo para hijo derecha de `actual`
en otro caso
  Hijo derecha de `actual` no existe
fin si