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 Recuperado del árbol (I)

Entrada:
`bin_tree`, árbol binario leído codificado

```
1: Crear pila nodos, pila hijos
2: centinelas = {-,<,>}

3: para todo elemento en bin_tree
4:    si elemento ∈ centinelas entonces
5:       Apilar elemento en hijos
6:       anterior ← actual
7:    en otro caso
8:       Apilar elemento en nodos
9:       si anterior ∉ centinelas entonces
10:         Apilar * en hijos
11:      fin si
12:   fin si
13: fin para

14: Crear arbol nuevo
15: raiz ← raíz de nuevo
16: Llamar al algoritmo 2 con parámetros raiz, nodos, hijos

17: devolver nuevo
```

Algoritmo 2 Recuperado del árbol (II)

Entrada:

`actual`, nodo sobre el que añadir descendientes
`nodos`, pila de nodos
`hijos`, pila de centinelas

1: **si** Tope `hijos` $\neq$ `-` **and** Tope `hijos` $\neq$ `<` **entonces**
2: Hijo izquierda de `actual` $\leftarrow$ tope de `hijos`
3: Sacar el tope de `hijos`
4: Sacar el tope de `nodos`
5: Llamar al algoritmo 2 con parámetros `hijo izquierda de actual`, `nodos`, `hijos`
6: **en otro caso**
7: Hijo izquierda de `actual` $\leftarrow \emptyset$
8: **fin si**

9: **si** Tope `hijos` $\neq$ `-` **and** Tope `hijos` $\neq$`>` **entonces**
10: Hijo derecha de `actual` $\leftarrow$ tope de `nodos`
11: Sacar el tope de `hijos`
12: Sacar el tope de `nodos`
13: Llamar al algoritmo 2 con parámetros `hijo derecha de actual`, `nodos`, `hijos`
14: **en otro caso**
15: Hijo derecha de `actual` $\leftarrow \emptyset$
16: **fin si**