ESTRUCTURAS DE DATOS

TIPOS ABSTRACTOS DE DATOS ARBORESCENTES

Parametrizando el recorrido de un árbol

Manuel Montenegro Montes
Departamento de Sistemas Informáticos y Computación
Facultad de Informática – Universidad Complutense de Madrid

Recorrer un árbol

Vamos a mezclar todo lo visto anterior con nuestros árboles binarios.

- Recorrer un árbol significa *visitar* todos sus nodos.

Cada uno lo visitamos exactamente UNA sola vez.

- **Visitar** un nodo significa realizar una acción que dependa del valor contenido en dicho nodo.

Hasta ahora:

```
template<typename T>
void BinTree<T>::preorder(const NodePointer &node) {
  if (node ≠ nullptr) {
    std::cout << node→elem << " ";
    preorder(node→left);
    preorder(node→right);
  }
}
```

Queremos parametrizar la ACCIÓN QUE SE HACE AL VISITAR CADA NODO

Parametrizar el recorrido

- Podemos parametrizar el recorrido con respecto a la acción a realizar en cada nodo.

```
template<typename T>
void BinTree<T>::preorder(const NodePointer &node) {
  if (node ≠ nullptr) {
    std::cout << node→elem << " ";
    preorder(node→left);
    preorder(node→right);
  }
}
```

Parametrizar consiste en hacer que la función reciba como parámetro lo que se hace cuando tenemos el elemento.

Parametrizar el recorrido

- Podemos parametrizar el recorrido con respecto a la acción a realizar en cada nodo.

```
template<typename T>
template<typename U>
void BinTree<T>::preorder(const NodePointer &node, U func) {
  if (node ≠ nullptr) {
    func(node→elem);
    preorder(node→left, func);
    preorder(node→right, func);
  }
}
```

Función que realiza las acciones al visitar el nodo.

Cambiamos el mostrar el nodo por algo que recibe.

de forma que si hacemos:
preorder(node, [](int x){

cout << x << " ";

}
)

Parametrizar el recorrido

- Modificamos también el método `preorden()` de la clase, que realiza la llamada inicial a la función recursiva:

```
template<class T>
class BinTree {
public:

   ...

   template <typename U>
   void preorder(U func) const {
     preorder(root_node, func);
   }

   ...

};
```

Esto también se puede hacer para el inorder y postorder.

Ejemplos

- Supongamos que tenemos el siguiente árbol:

```
BinTree<int> tree {{{ 9 }, 4, { 5 }}, 7, {{ 10 }, 4, { 6 }}};
```

- Imprimir el recorrido en preorden:

```
tree.preorder([] (int x) { std::cout << x << " "; });
```

- Imprimir solamente los elementos pares:

```
tree.preorder([] (int x) {
  if (x % 2 == 0) {
    std::cout << x << " ";
  }
});
```

Ejemplos

- Supongamos que tenemos el siguiente árbol:

```
BinTree<int> tree {{{ 9 }, 4, { 5 }}, 7, {{ 10 }, 4, { 6 }}};
```

- Sumar los elementos del árbol:

Aquí acumulamos la suma

```
int acum = 0;
tree.preorder([&acum](int x) { acum += x; });
std::cout << acum << std::endl;
```

- Contar el número de elementos de un árbol:

```
int num_elems = 0;
tree.preorder([&num_elems](int x) { num_elems++; });
std::cout << num_elems << std::endl;
```

Ejemplos

- Supongamos que tenemos el siguiente árbol:

```
BinTree<int> tree {{{ 9 }, 4, { 5 }}, 7, {{ 10 }, 4, { 6 }}};
```

- Añadir los elementos del árbol a una lista:

```
std::vector<int> v;
tree.preorder([&v](int x) { v.push_back(x); });
```

¿Se podrán modifica más parámetros que se pasan en los corchetes?