Iteradores sobre árboles binarios.

Prof. Isabel Pita

Explicación de las transparencias de clase

Código desarrollado por el profesor A. Verdejo para la asignatura de Estructuras de Datos de la Facultad de Informática de la UCM.

5 de mayo de 2020

1. Recorrido de un árbol binario con un iterador.

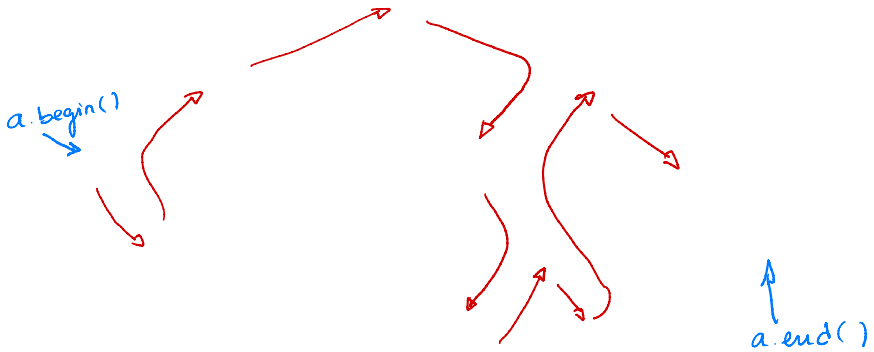
Los iteradores se utilizan para recorrer los elementos de una estructura de datos. En el caso de un árbol binario, el iterador puede realizar el recorrido en cualquiera de los órdenes vistos en la sesión anterior: preorden, inorden, postorden o en anchura.

En este curso vemos la implementación de un iterador que realiza el recorrido en inorden de un árbol. Para manejar el iterador necesitamos:

- Funciones `begin()` y `end()` que devuelvan un iterador al primer elemento en inorden y a una posición detrás del último respectivamente.
- También una operación `++` para avanzar el iterador.
  - Si el iterador está apuntando a un nodo con hijo derecho, el siguiente elemento será el primero del hijo derecho (el de más a la izquierda del hijo derecho). Observa que si el nodo al que apunta el iterador tuviese hijo izquierdo, éste ya habría sido recorrido por el iterador, por eso el siguiente a recorrer es el hijo derecho.
  - Si no existe hijo derecho, el siguiente, si existe, está por encima en el árbol. Hay que retroceder hasta el primer antecesor no visitado aún (aquel nodo en cuyo subárbol izquierdo nos encontramos).

Por ejemplo, en el siguiente árbol, el recorrido empieza en el nodo 4, que es primer elemento del recorrido en inorden. A continuación se visita el 8 y luego el 2. Cuando el iterador está en el nodo 2, todo su hijo izquierdo ya ha sido visitado, y queda por visitar su hijo derecho si lo tuviera. Como no lo tiene retrocedemos hasta el primer antecesor no visitado que es el 1.

Desde el 1 visitamos el 6. Observamos que hemos pasado por el 3, pero no lo hemos visitado. Por lo tanto el nodo 3 lo debemos guardar en una estructura para visitarlo posteriormente. Después del 6 visitamos el 9 y debemos guardar el 11 para visitarlo posteriormente. Cuando acaba el 9 retrocedemos hasta el primer antecesor no visitado, que es el 11. Guardaremos los antecesores no visitados en una pila, ya que el último que se guarda es el primero que se debe visitar.



El iterador del árbol binario se encuentra implementado en la parte pública de la clase `bintree` (justo debajo de la implementación recursiva de los recorridos). En la parte pública se declaran las operaciones que ofrece el iterador: el operador `*` y el operador `->` permiten acceder al elemento apuntado por el iterador, los operadores de igualdad `==` y desigualdad, `!=` permiten comparar dos iteradores para comprobar si apuntan al mismo elemento, y el operador `++` permite avanzar el iterador al elemento siguiente en el orden definido.

```
public:
   class const_iterator {
   public:
      T const& operator*() const {
         if (ptr == nullptr) throw std::out_of_range("fuera del arbol");
```

```
        return ptr->elem;
    }

    T const* operator->() const {
        return &operator*();
    }

    bool operator==(const_iterator const& other) const {
        return ptr == other.ptr;
    }
    bool operator!=(const_iterator const& other) const {
        return !(*this == other);
    }

    const_iterator & operator++() {  // ++ prefijo
        next();
        return *this;
    }
    .....
  };
```

En la parte privada del iterador se encuentran los tipos necesarios para mantener el estado del iterador. Se declara un `Link` que apunta al nodo que estamos visitando y una *pila de Links* donde se guardan los nodos por los que se ha pasado pero no se han visitado todavía. Las operaciones `begin()`, y `end()` son métodos de la clase `bintree`, no son del iterador.

Se declara la clase `bintree` como amiga para que se pueda acceder desde ella a los datos privados del iterador. En concreto la clase `bintree` necesita poder acceder a la constructora privada para implementar los métodos `begin()` y `end()`.

```
public:
    class const_iterator {
    public: ...
    private:
        friend class bintree;
        Link ptr;
        std::stack<Link> ancestros;
        ....
```

Las operaciones que tenemos son:

- Un constructor privado que recibe un `Link`. Observad que el valor del `Link` del iterador no es este puntero sino el resultado de aplicar la función `first` a ese puntero. La función `first` es una función privada del iterador que desciende por el árbol con un bucle (se implementa de forma iterativa) hasta alcanzar el nodo más a la izquierda. Almacena los nodos por los que pasa en la pila del iterador, ya que estos nodos todavía no han sido visitados y debemos guardarlos para visitarlos posteriormente cuando se retroceda en el recorrido.

- Un constructor privado sin parámetros. Este constructor pone el puntero del Link a nulo. Se utiliza para implementar el método `end()` que apunta al elemento siguiente al último.

- El método `first` ya se explicó anteriormente, devuelve un puntero (`Link`) al nodo más a la izquierda del árbol cuyo nodo recibe de entrada.

- El método `next` que se utiliza para implementar el operador `++` del iterador. Este método tiene una implementación iterativa. En su implementación diferencia los casos en que se puede encontrar el iterador.

  - Si el iterador esta a `nullptr` es que se ha terminado el recorrido del árbol y por lo tanto no se puede seguir avanzando. Se produce un error.
  - Si el nodo al que apunta el iterador tiene hijo derecho, se busca el nodo más a la izquierda del hijo derecho y ese es el siguiente nodo a visitar. Para buscar el nodo más a la izquierda se utiliza la función `first` vista antes.

- Si no existe nodo a la derecha del nodo al que apunta el iterador y la pila está vacía es que hemos acabado el recorrido y se devolverá un puntero `nullptr` para indicar que estamos en el final.
- Si no existen nodos a la derecha y la pila no está vacía, es que debemos retroceder en el árbol y visitar el nodo apuntado por el `Link` de la cima de la pila.

```
public:
   class const_iterator {
   public: ...
   private:
        .....
      // constructores privados
      // construye el iterador al primero
      const_iterator(Link raiz) { ptr = first(raiz); }

      // construye el iterador al ultimo
      const_iterator() : ptr(nullptr) {}

      Link first(Link act) {
         if (act == nullptr) {
            return nullptr;
         } else {
            while (act->left != nullptr) {
               ancestros.push(act);
               act = act->left;
            }
            return act;
         }
      }

      void next() {
         if (ptr == nullptr) {
            throw std::range_error("El iterador no puede avanzar");
         } else if (ptr->right != nullptr) { // primero del hijo derecho
            ptr = first(ptr->right);
         } else if (ancestros.empty()) { // hemos llegado al final
            ptr = nullptr;
         } else { // podemos retroceder
            ptr = ancestros.top();
            ancestros.pop();
         }
      }

   };  // fin de la clase iterator
```

Por último en la implementación de la clase `bintree` aparece la implementación de los métodos `begin()` y `end()`. En el primero se construye un iterador con el constructor que recibe un `Link`. El link utilizado para llamar al constructor es la raíz del árbol. El iterador que se devuelve apunta al nodo mas a la izquierda de la raíz, que es el primero en el recorrido en inorden.

Para la implementación del método `end()`, se utiliza el constructor del iterador sin parámetros, que devuelve un iterador cuyo puntero apunta a `nullptr` para indicar que se ha acabado el recorrido del árbol.

```
   // Continua la parte publica de la clase bintree
   const_iterator begin() const {
      return const_iterator(raiz);
   }
   const_iterator end() const {
      return const_iterator();
   }

};
```

El iterador definido para el árbol binario es un iterador constante, que no permite modificar el valor de los nodos apuntados por el iterador. Esto es coherente con la implementación que hemos realizado de los árboles binarios que tampoco ofrecen ningún método para modificar el contenido del árbol.

Por lo tanto el iterador sólo puede utilizarse para recorrer el árbol y obtener información del contenido del mismo. Por ejemplo la siguiente función recorre un árbol con un iterador contando el número de nodos que tienen un asterisco (*) y el número de nodos que tienen un signo mas (+). Esto mismo se podría hacer con una función función recursiva.

```
struct tSol {
    int numAst, numSum;
};

tSol resolver(bintree<char> const& arbol){
    tSol s = {0,0};
    auto it = arbol.begin();
    while (it != arbol.end()){
        if (*it == '*') ++s.numAst;
        else if (*it == '+') ++s.numSum;
        ++it;
    }
    return s;
}

void resuelveCaso() {
    bintree<char> arbol = leerArbol('.');
    tSol s = resolver(arbol);
    std::cout << s.numAst << ' ' << s.numSum  << '\n';

}
```