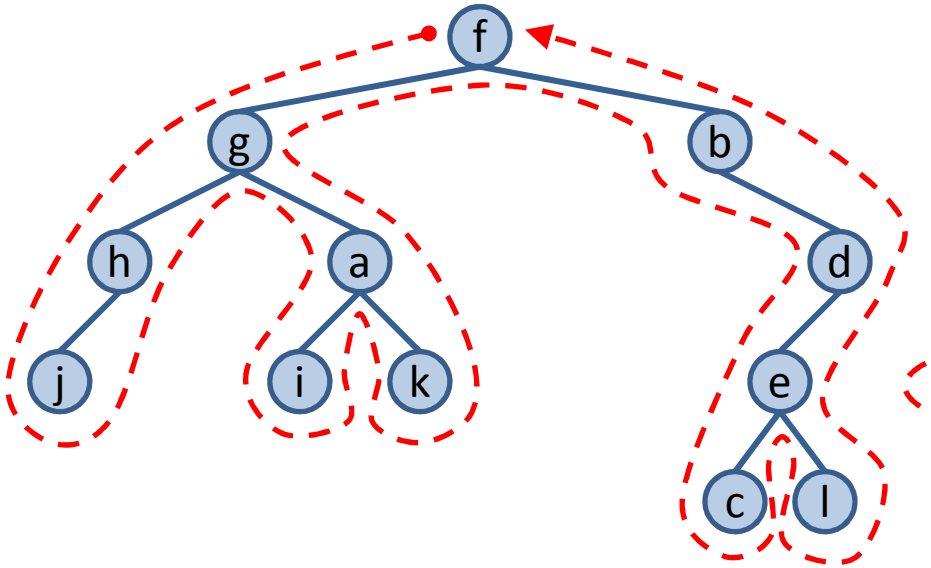
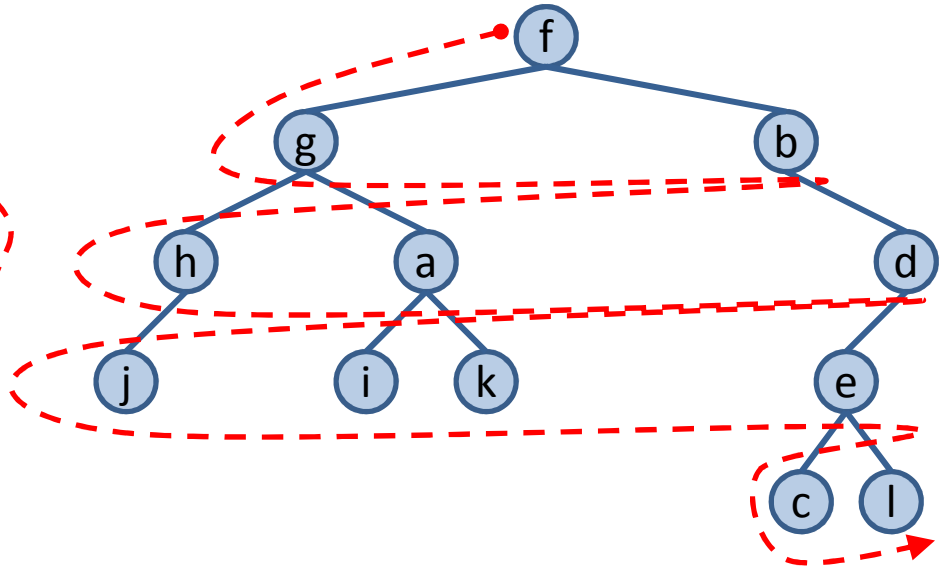
Recorridos de árboles binarios

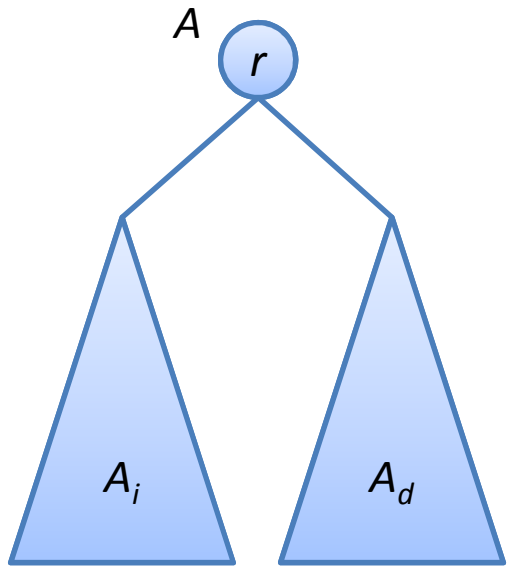
En profundidad



En anchura

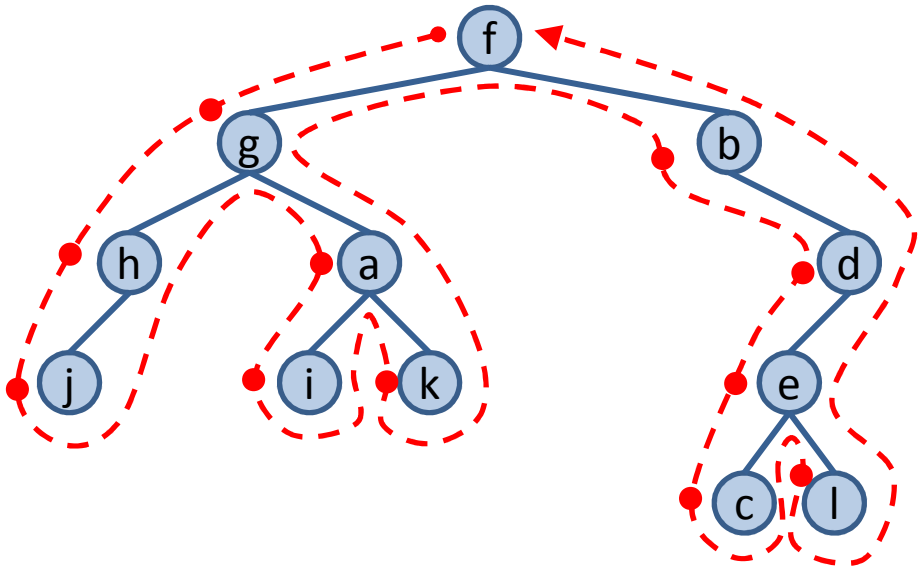


Recorridos en profundidad de árboles binarios



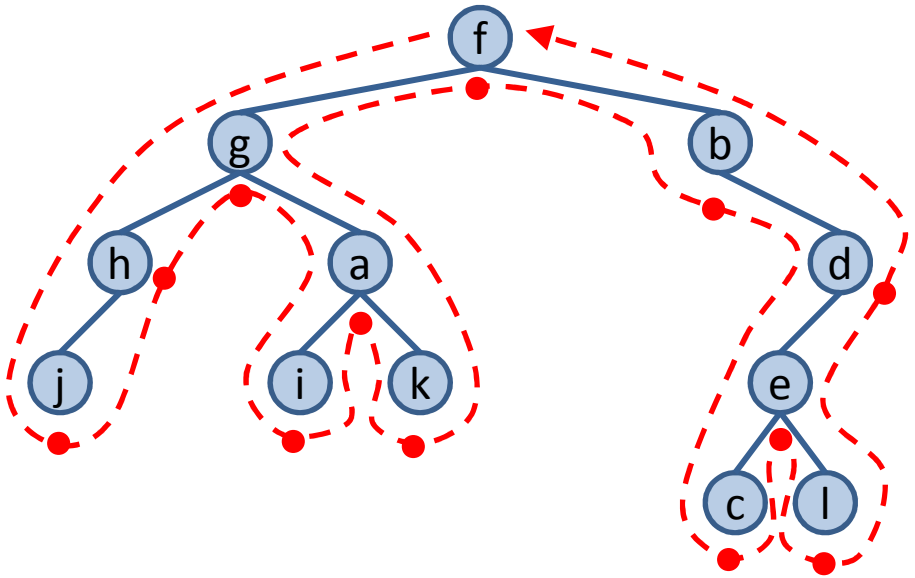
Preorden(A) = *r* Preorden(A_i) Preorden(A_d)
Inorden(A) = Inorden(A_i) *r* Inorden(A_d)
Postorden(A) = Postorden(A_i) Postorden(A_d) *r*

Recorrido en preorden de un árbol binario



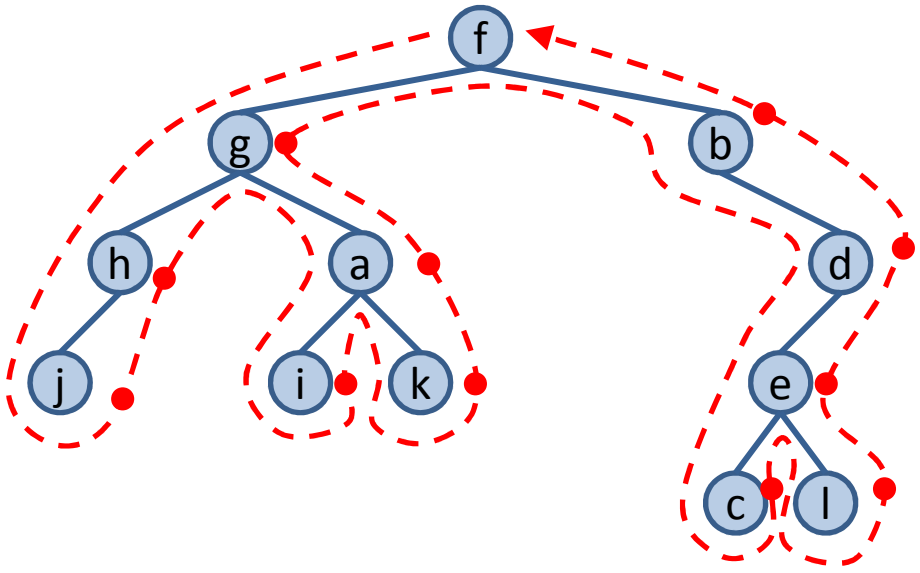
Preorden(f) = f Preorden(g) Preorden(b) =
f g Preorden(h) Preorden(a) Preorden(b) =
f g h Preorden(j) Preorden(a) Preorden(b) =
f g h j Preorden(a) Preorden(b) =
f g h j a Preorden(i) Preorden(k) Preorden(b) =
f g h j a i Preorden(k) Preorden(b) =
f g h j a i k Preorden(b) =
f g h j a i k b Preorden(d) =
f g h j a i k b d Preorden(e) =
f g h j a i k b d e Preorden(c) Preorden(l) =
f g h j a i k b d e c Preorden(l) =
f g h j a i k b d e c l

Recorrido en inorden de un árbol binario



Inorden(f) = Inorden(g) f Inorden(b) =
Inorden(h) g Inorden(a) f Inorden(b) =
Inorden(j) h g Inorden(a) f Inorden(b) =
j h g Inorden(a) f Inorden(b) =
j h g Inorden(i) a Inorden(k) f Inorden(b) =
j h g i a Inorden(k) f Inorden(b) =
j h g i a k f Inorden(b) =
j h g i a k f b Inorden(d) =
j h g i a k f b Inorden(e) d =
j h g i a k f b Inorden(c) e Inorden(l) d =
j h g i a k f b c e Inorden(l) d =
j h g i a k f b c e l d

Recorrido en postorden de un árbol binario



Postorden(f) = Postorden(g) Postorden(b) f =
Postorden(h) Postorden(a) g Postorden(b) f =
Postorden(j) h Postorden(a) g Postorden(b) f =
j h Postorden(a) g Postorden(b) f =
j h Postorden(i) Postorden(k) a g Postorden(b) f =
j h i Postorden(k) a g Postorden(b) f =
j h i k a g Postorden(b) f =
j h i k a g Postorden(d) b f =
j h i k a g Postorden(e) d b f =
j h i k a g Postorden(c) Postorden(l) e d b f =
j h i k a g c Postorden(l) e d b f =
j h i k a g c l e d b f

Implementación recursiva de recorridos en profundidad de árboles binarios

```
template <typename T>
void preordenAbin (typename Abin<T>::nodo n, const Abin<T>& A,
           void (*procesar)(typename Abin<T>::nodo, const Abin<T>&))
// Recorrido en preorden del subárbol cuya raíz
// es el nodo n perteneciente al árbol binario A.
// Cada nodo visitado se procesa mediante la
// función procesar().
{
   if (n != Abin<T>::NODO_NULO)
   {
        procesar(n, A);
        preordenAbin(A.hijoIzqdoB(n), A, procesar);
        preordenAbin(A.hijoDrchoB(n), A, procesar);
   }
}
```

```
template <typename T>
void inordenAbin (typename Abin<T>::nodo n, const Abin<T>& A,
        void (*procesar)(typename Abin<T>::nodo, const Abin<T>&))
// Recorrido en inorden del subárbol cuya raíz
// es el nodo n perteneciente al árbol binario A.
// Cada nodo visitado se procesa mediante la
// función procesar().
{
   if (n != Abin<T>::NODO_NULO)
   {
        inordenAbin(A.hijoIzqdoB(n), A, procesar);
        procesar(n, A);
        inordenAbin(A.hijoDrchoB(n), A, procesar);
   }
}
```

```
template <typename T>
void postordenAbin (typename Abin<T>::nodo n, const Abin<T>& A,
        void (*procesar)(typename Abin<T>::nodo, const Abin<T>&))
// Recorrido en postorden del subárbol cuya raíz
// es el nodo n perteneciente al árbol binario A.
// Cada nodo visitado se procesa mediante la función procesar().
{
   if (n != Abin<T>::NODO_NULO)
   {
        postordenAbin(A.hijoIzqdoB(n), A, procesar);
        postordenAbin(A.hijoDrchoB(n), A, procesar);
        procesar(n, A);
   }
}

template <typename T>
void escribirNodo (typename Abin<T>::nodo n, const Abin<T>& A)
{
   if (n != Abin<T>::NODO_NULO)
      std::cout << A.elemento(n) << ' ';
}
```

Implementación iterativa del recorrido en preorden de árboles binarios

```
#include "pilaenla.h"
template <typename T>
void preordenAbin2 (typename Abin<T>::nodo n, const Abin<T>& A,
         void (*procesar)(typename Abin<T>::nodo, const Abin<T>&))
{  // Algoritmo básico
   Pila<typename Abin<T>::nodo> P; // pila de nodos de un árbol binario

   P.push(n);
   while (!P.vacia())
   {
      n = P.tope(); P.pop();
      procesar(n, A);
      if (n != Abin<T>::NODO_NULO)
      {
         P.push(A.hijoDrchoB(n));
         P.push(A.hijoIzqdoB(n));
      }
   }
}
```

```
template <typename T>
void preordenAbin2 (typename Abin<T>::nodo n, const Abin<T>& A,
         void (*procesar)(typename Abin<T>::nodo, const Abin<T>&))
// Recorrido en preorden del subárbol cuya raíz es el nodo n
// perteneciente al árbol binario A. Cada nodo visitado se procesa
// mediante la función procesar().
{  // Algoritmo mejorado
   Pila<typename Abin<T>::nodo> P; // pila de nodos de árbol binario

   do {
      if (n != Abin<T>::NODO_NULO)
      {
         procesar(n, A);
         if (A.hijoDrchoB(n) != Abin<T>::NODO_NULO)
            P.push(A.hijoDrchoB(n));
         n = A.hijoIzqdoB(n);
      }
      else if (!P.vacia())
      {
         n = P.tope();
         P.pop();
      }
   } while (!(n == Abin<T>::NODO_NULO && P.vacia()));
}
```

Implementación del recorrido en anchura o por niveles de árboles binarios

```
#include "colaenla.h"
template <typename T>
void recNivelesAbin (typename Abin<T>::nodo n, const Abin<T>& A,
                     void (*procesar)(typename Abin<T>::nodo, const Abin<T>&))
{  // Algoritmo básico
   Cola<typename Abin<T>::nodo> C; // cola de nodos de un árbol binario

   C.push(n);
   while (!C.vacia())
   {
      n = C.frente(); C.pop();
      procesar(n, A);
      if (n != Abin<T>::NODO_NULO)
      {
         C.push(A.hijoIzqdoB(n));
         C.push(A.hijoDrchoB(n));
      }
   }
}
```

```
template <typename T>
void recNivelesAbin (typename Abin<T>::nodo n, const Abin<T>& A,
        void (*procesar)(typename Abin<T>::nodo, const Abin<T>&))
// Recorrido por niveles del subárbol cuya raíz es el nodo n
// perteneciente al árbol binario A. Cada nodo visitado se
// procesa mediante la función procesar().
{  // Algoritmo mejorado
   Cola<typename Abin<T>::nodo> C; // cola de nodos de árbol binario
   if (n != Abin<T>::NODO_NULO)
   {
      do {
         if (!C.vacia())
         {
            n = C.frente();
            C.pop();
         }
         procesar(n, A);
         if (A.hijoIzqdoB(n) != Abin<T>::NODO_NULO)
            C.push(A.hijoIzqdoB(n));
         if (A.hijoDrchoB(n) != Abin<T>::NODO_NULO)
            C.push(A.hijoDrchoB(n));
      } while (!C.vacia());
   }
}
```