Algoritmos y estructuras de datos

Árboles 2-3-4

Francisco Javier Zaragoza Martínez

Universidad Autónoma Metropolitana Unidad Azcapotzalco
Departamento de Sistemas

2 de septiembre de 2024

Citas

Eurípides

Lo mejor y más seguro es mantener el balance en tu vida.

Hazrat Inayat Khan

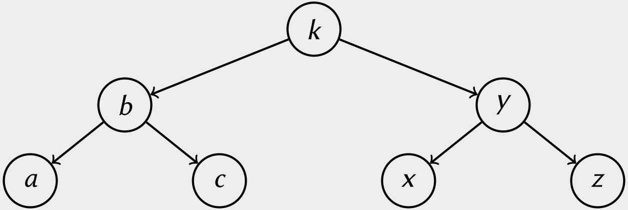
Debe haber un balance en todas tus acciones, ser extremo o tibio es igualmente malo.

Elizabeth Gilbert

Para encontrar el balance que desea, esto es en lo que debe convertirse. Debe tener los pies tan firmemente en la tierra que sea como si tuviera cuatro piernas en lugar de dos.

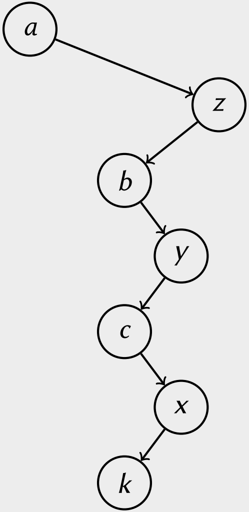
Problema de los árboles binarios de búsqueda

Si nos va bien: altura logarítmica



Problema de los árboles binarios de búsqueda

Si nos va mal: altura lineal



Árboles balanceados

Diremos que un árbol enraizado de orden n es α-balanceado si todas sus hojas tienen profundidad $\leq 1 + \alpha \log_2 n$. Idealmente $\alpha = 1$.

¿Cómo se logra esto?

Una solución al problema de crear árboles desbalanceados es la de reorganizar sus nodos conforme se hagan inserciones y borrados. Algunos ejemplos son:

Árboles AVL árboles binarios de búsqueda 1.44-balanceados.

Árboles 2-3-4 árboles cuaternarios de búsqueda 1-balanceados.

Árboles rojinegros árboles binarios de búsqueda 2-balanceados.

Otros ejemplos incluyen a los árboles B, los árboles *biselados* y los *treaps*.

Árboles 2-3-4

Un árbol 2-3-4 es un árbol balanceado en el que todas sus hojas tienen la misma profundidad y donde sus nodos pueden:

1. contener una clave x_1 y dos árboles A_0 y A_1.
2. contener dos claves $x_1 < x_2$ y tres árboles A_0, A_1 y A_2.
3. contener tres claves $x_1 < x_2 < x_3$ y cuatro árboles A_0, A_1, A_2 y A_3.

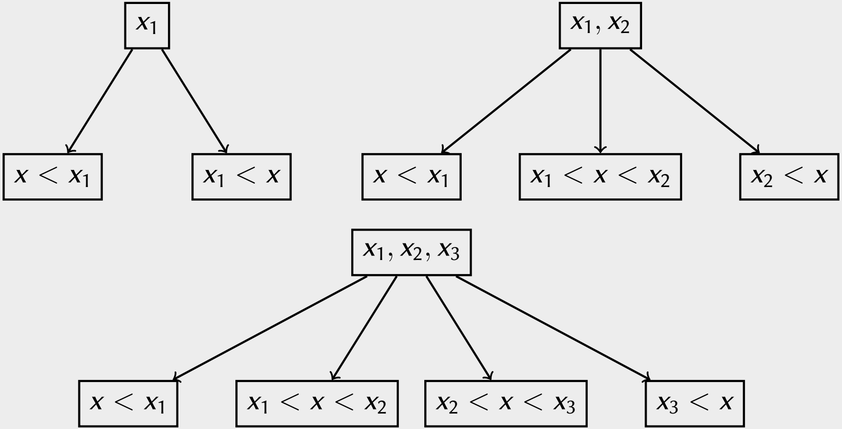
En cada caso se cumple que si a es una clave en el árbol A_{i-1} y c es una clave en el árbol A_i entonces $a < x_i$ y $x_i < c$.

Tipos de nodos

Los nodos con dos, tres y cuatro árboles se llaman 2 nodos (binarios), 3 nodos (ternarios) y 4 nodos (cuaternarios), respectivamente.

Árboles 2-3-4

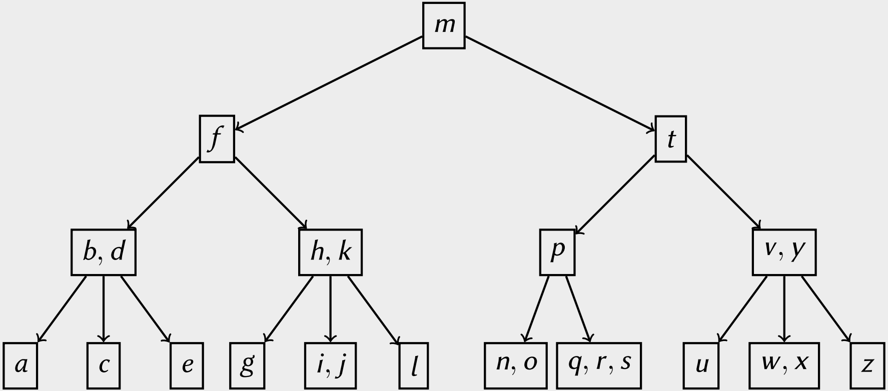
Nodos binarios, ternarios y cuaternarios



Árboles 2-3-4

Ejemplo

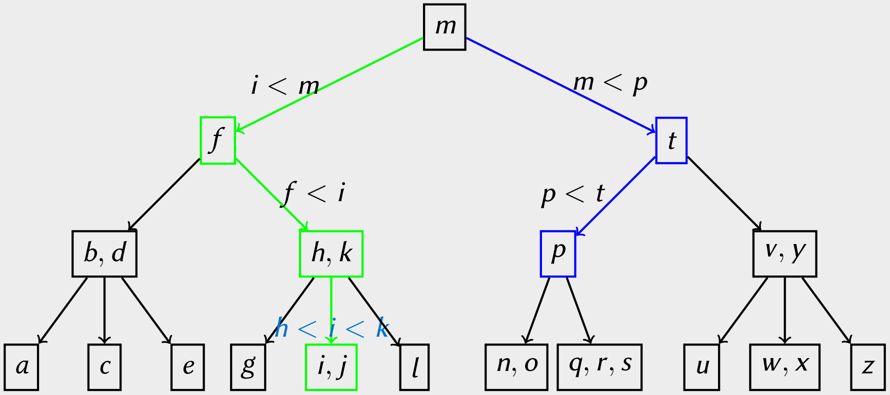
Un árbol 2-3-4 con 11 nodos binarios, 6 ternarios, 1 cuaternario y 11 hojas.



Árboles 2-3-4

Ejemplo de búsqueda

Buscar la i y buscar la p.



Inserción de una clave

1. Si la raíz está vacía, se agrega la clave a la raíz y se termina.
2. Se hace la búsqueda de la clave con dos salvedades:
   1. Si durante la búsqueda se ve un nodo cuaternario, se reorganiza y se continúa.
   2. Si la clave se encuentra, se termina.
3. La clave se agrega a la hoja que se haya llegado. ¿Por qué cabe?

Reorganizar un nodo cuaternario

1. Se manda la clave x_2 al nodo precursor (si era la raíz, se crea una nueva raíz). ¿Por qué cabe x_2 en el precursor?
2. Se sustituye el nodo cuaternario por dos nodos binarios, con claves x_1 y x_3 y con árboles A_0, A_1 y A_2, A_3 respectivamente.

Ejemplo de inserción

Queremos insertar claves en el orden q, w, e, r, t, y, u, i, o, p.

$$\boxed{q}$$

Las claves q, w entran a la raíz.

Árboles 2-3-4

Ejemplo de inserción

Queremos insertar claves en el orden $q, w, e, r, t, y, u, i, o, p$.

$$\boxed{q, w}$$

Las claves q, w entran a la raíz.

Árboles 2-3-4

Ejemplo de inserción

Queremos insertar claves en el orden $q, w, e, r, t, y, u, i, o, p$.

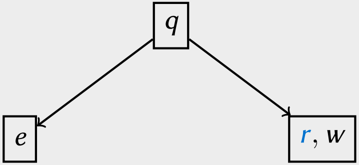
$$\boxed{e, q, w}$$

La clave e entra a la raíz, que se vuelve cuaternaria.

Árboles 2-3-4

Ejemplo de inserción

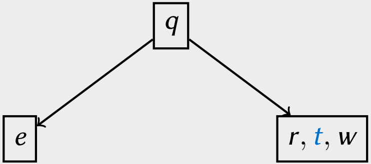
Queremos insertar claves en el orden $q, w, e, r, t, y, u, i, o, p$.



La clave r obliga a reorganizar la raíz.

Ejemplo de inserción

Queremos insertar claves en el orden $q, w, e, r, t, y, u, i, o, p$.

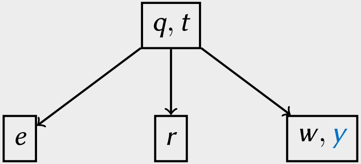


La clave t crea una hoja cuaternaria.

Árboles 2-3-4

Ejemplo de inserción

Queremos insertar claves en el orden $q, w, e, r, t, y, u, i, o, p$.

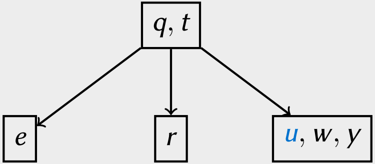


La clave y obliga a reorganizar esa hoja.

Árboles 2-3-4

Ejemplo de inserción

Queremos insertar claves en el orden $q, w, e, r, t, y, u, i, o, p$.

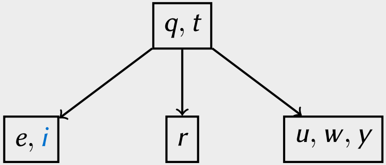


La clave u crea una hoja cuaternaria.

Árboles 2-3-4

Ejemplo de inserción

Queremos insertar claves en el orden q, w, e, r, t, y, u, i, o, p.

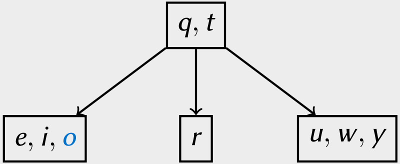


Las claves i, o entran en hojas.

Árboles 2-3-4

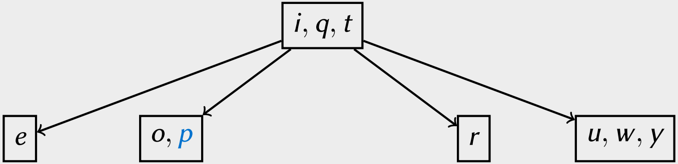
Ejemplo de inserción

Queremos insertar claves en el orden q, w, e, r, t, y, u, i, o, p.



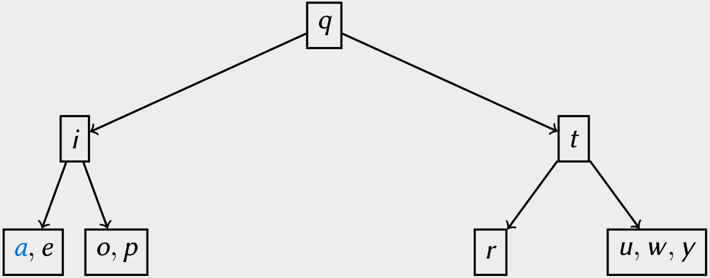
Las claves i, o entran en hojas.

Queremos insertar claves en el orden q, w, e, r, t, y, u, i, o, p.



La clave p obliga a reorganizar una hoja. La raíz es cuaternaria.

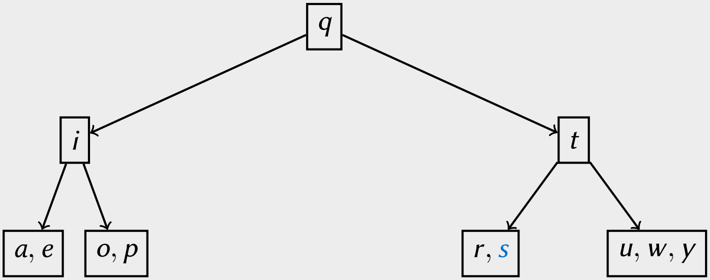
Ahora queremos insertar claves en el orden a, s, d, f, g, h, j, k, l, z, x, c, v, b.
La clave a obliga a reorganizar la raíz.



Árboles 2-3-4

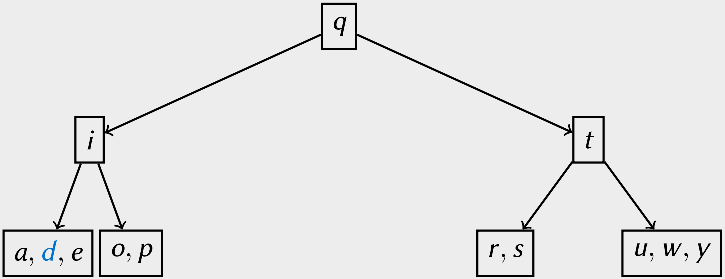
Ejemplo de inserción

Ahora queremos insertar claves en el orden a, s, d, f, g, h, j, k, l, z, x, c, v, b.
La clave s entra en una hoja.



Ejemplo de inserción

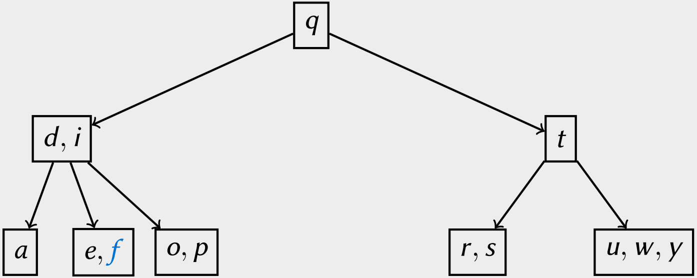
Ahora queremos insertar claves en el orden a, s, d, f, g, h, j, k, l, z, x, c, v, b.
La clave d crea una hoja cuaternaria.



Árboles 2-3-4

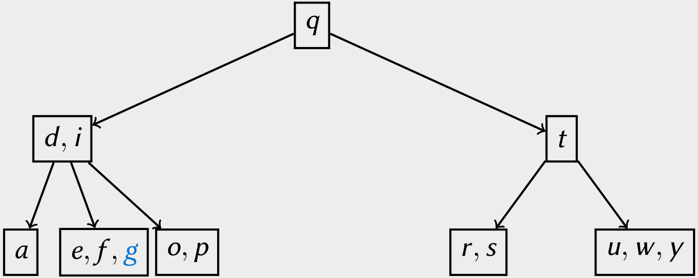
Ejemplo de inserción

Ahora queremos insertar claves en el orden $a, s, d, f, g, h, j, k, l, z, x, c, v, b$.
La clave f obliga a reorganizar esa hoja.



Ejemplo de inserción

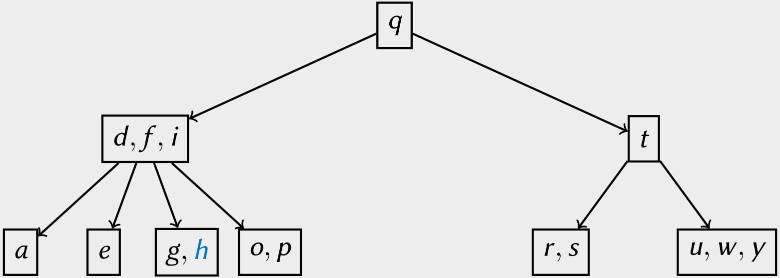
Ahora queremos insertar claves en el orden $a, s, d, f, g, h, j, k, l, z, x, c, v, b$.
La clave g crea una hoja cuaternaria.



Árboles 2-3-4

Ejemplo de inserción

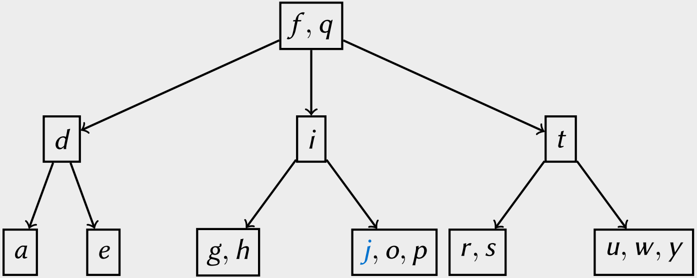
Ahora queremos insertar claves en el orden a, s, d, f, g, h, j, k, l, z, x, c, v, b.
La clave h obliga a reorganizar esa hoja y crea un nodo cuaternario.



Árboles 2-3-4

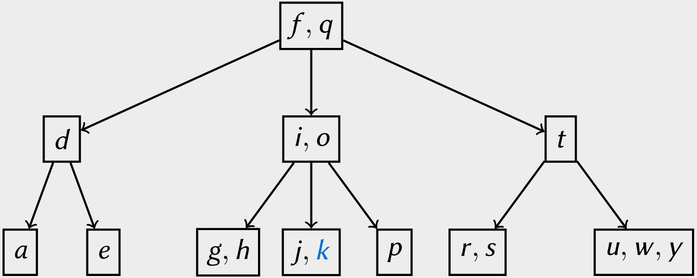
Ejemplo de inserción

Ahora queremos insertar claves en el orden $a, s, d, f, g, h, j, k, l, z, x, c, v, b$.
La clave j obliga a reorganizar ese nodo y crea una hoja cuaternaria.



Árboles 2-3-4
Ejemplo de inserción

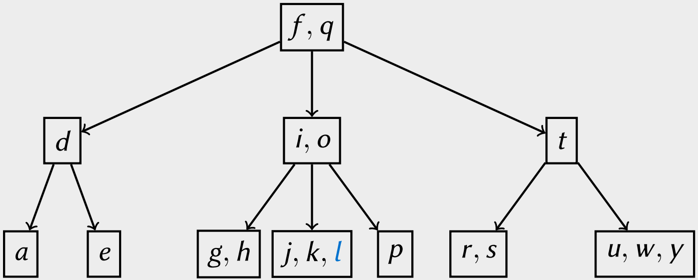
Ahora queremos insertar claves en el orden $a, s, d, f, g, h, j, k, l, z, x, c, v, b$.
La clave k obliga a reorganizar esa hoja.



Árboles 2-3-4

Ejemplo de inserción

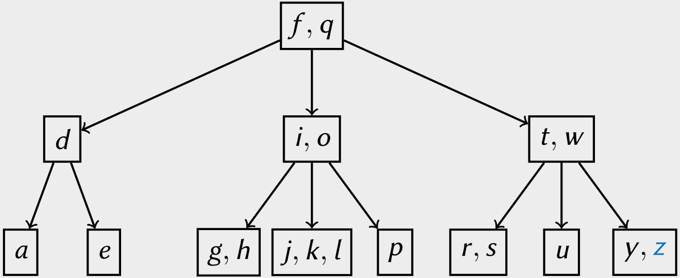
Ahora queremos insertar claves en el orden $a, s, d, f, g, h, j, k, l, z, x, c, v, b$.
La clave l crea una hoja cuaternaria.



Árboles 2-3-4

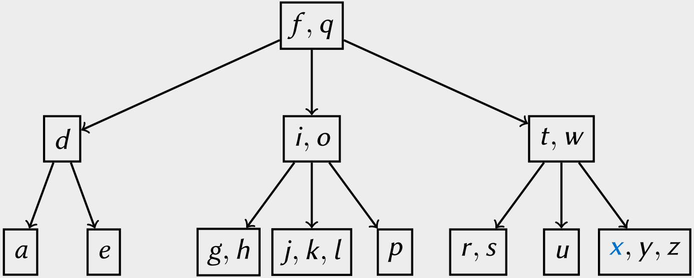
Ejemplo de inserción

Ahora queremos insertar claves en el orden $a, s, d, f, g, h, j, k, l, z, x, c, v, b$.
La clave z obliga a reorganizar una hoja cuaternaria.



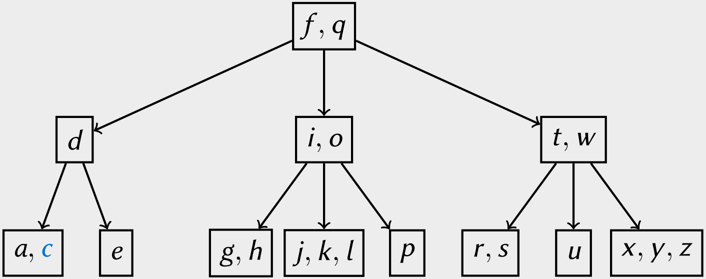
Ejemplo de inserción

Ahora queremos insertar claves en el orden $a, s, d, f, g, h, j, k, l, z, x, c, v, b$.
Las claves x, c, v entran en hojas.

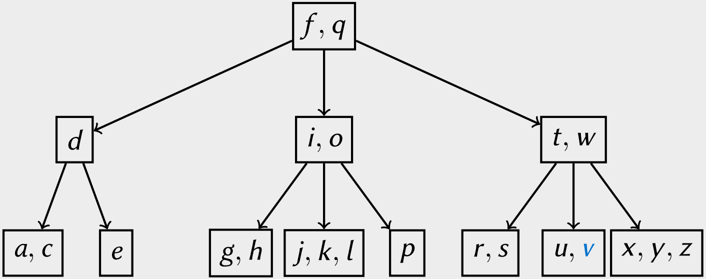


Ejemplo de inserción

Ahora queremos insertar claves en el orden $a, s, d, f, g, h, j, k, l, z, x, c, v, b$.
Las claves x, c, v entran en hojas.



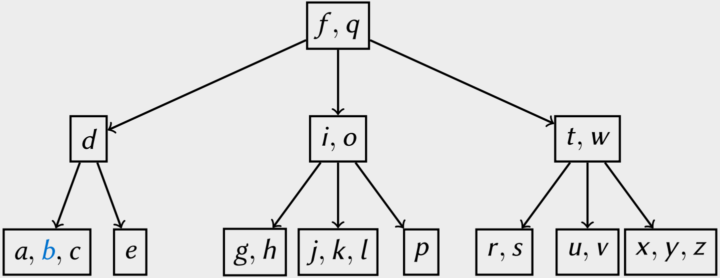
Ahora queremos insertar claves en el orden a, s, d, f, g, h, j, k, l, z, x, c, v, b.
Las claves x, c, v entran en hojas.



Árboles 2-3-4

Ejemplo de inserción

Ahora queremos insertar claves en el orden $a, s, d, f, g, h, j, k, l, z, x, c, v, b$.
La clave b crea una hoja cuaternaria.



Balance perfecto

Los árboles balanceados que tienen todas sus hojas a la misma profundidad se dicen perfectamente balanceados. Los árboles 2-3-4 son 1-balanceados en el peor caso (si todos los nodos fueran binarios) y 0.5-balanceados en el mejor caso (si todos los nodos fueran cuaternarios).

Balance de árboles 2-3-4

Ocurre automáticamente: La profundidad de todas las hojas aumenta simultáneamente cuando se reorganiza la raíz (y nunca más).

¿Por qué no se usan?

Las implementaciones son lentas debido a los diferentes tipos de nodos.

Tipos asociados a un árbol 2-3-4

Definiremos un tipo estructurado `nodo234` para representar un nodo y un tipo `arbol234` para representar un árbol 2-3-4. El tipo `nodo234` consiste de un contador, tres datos y cuatro apuntadores a sus sucesores (valdrán `NULL` si son vacíos).

```
typedef struct nodo234 {
  int tam;                   // contador de claves
  int a[3];                  // datos almacenados
  struct nodo234 *suc[4];    // enlaces a sucesores
} nodo234;
```

Por otro lado, el tipo `arbol234` es un apuntador a `nodo234`.

```
typedef nodo234 *arbol234;
```

Note que los tipos `nodo234 *`, **`struct`** `nodo234 *` y `arbol234` son equivalentes.

Árboles 2-3-4

Implementación: busca clave en un nodo y en un árbol

```
int nomenor234(arbol234 p, int x) {
  int i;
  for (i = 0; i < p->tam && x > p->a[i]; i++);
  return i;
}

nodo234 *busca234(arbol234 s, int x) {
  while (s != NULL) {              // mientras haya nodo
    int i = nomenor234(s, x); // busca x en ese nodo
    if (i < s->tam && x == s->a[i]) // si x esta
      return s;                    // en el nodo *s
    s = s->suc[i];                 // si no avanza
  }
  return NULL;                     // x no esta en arbol
}
```

Árboles 2-3-4

Implementación: inserta clave en nodo y crea un nodo con clave

```
void clave234(nodo234 *p, int x, nodo234 *izq, nodo234 *der) {
  int i;                 // busca la posicion i para la clave x
  for (i = p->tam; i > 0 && x < p->a[i-1]; i--)
      p->a[i] = p->a[i-1], p->suc[i+1] = p->suc[i];
  p->a[i] = x;           // inserta x en su posicion
  p->suc[i] = izq;       // inserta sus dos sucesores
  p->suc[i+1] = der;
  p->tam++;              // actualiza el contador
}

nodo234 *nuevo234(int x, nodo234 *izq, nodo234 *der) {
  nodo234 *t = (nodo234 *) calloc(1, sizeof(nodo234));
  clave234(t, x, izq, der);
  return t;
}
```

Árboles 2-3-4

Implementación: reorganiza un nodo cuaternario

```
arbol234 *parte234(arbol234 *r, arbol234 *prec) {
  int x = (*r)->a[1];          // x es la clave central
  nodo234 *izq = nuevo234((*r)->a[0], (*r)->suc[0], (*r)->suc[1]);
  nodo234 *der = nuevo234((*r)->a[2], (*r)->suc[2], (*r)->suc[3]);
  if (prec != NULL) {          // si *r no es la raiz
    free(*r);                  // libera *r
    r = prec;                  // x se insertara en el precursor
  } else (*r)->tam = 0;        // si no vacia *r
  clave234(*r, x, izq, der); // inserta x donde corresponde
  return t;
}
```

Observación: Esta función siempre crea dos nodos nuevos para la primera y la última claves del nodo a reorganizar, que es lo más simple pero no lo más eficiente.

Árboles 2-3-4

Implementación: inserta una clave en el árbol 2-3-4

```
void inserta234(arbol234 *r, int x) {
  if (*r != NULL) {
    arbol234 *prec = NULL;
    while (*r != NULL) {
      if ((*r)->tam == 3) r = parte234(r, prec);
      int i = nomenor234(*r, x);
      if (i < (*r)->tam && x == (*r)->a[i]) return;
      prec = r;
      r = &((*r)->suc[i]);
    }
    clave234(*prec, x, NULL, NULL);
  } else *r = nuevo234(x, NULL, NULL);
}
```

Árboles 2-3-4

Implementación: recorrido en orden de un árbol 2-3-4

```
void enorden234(arbol234 s) {
  if (s != NULL) {
    for (int i = 0; i < s->tam; i++) {
      enorden234(s->suc[i]);
      printf("%d ", s->a[i]);
    }
    enorden234(s->suc[s->tam]);
  }
}
```

Número de pasos en el peor caso, sobre un conjunto A de n elementos y un elemento x. Abajo $\log_2 n \leq h \leq n$ es la altura del árbol binario de búsqueda.

Operación	Símbolo	Bit	AD	AO	LD	LO	ABB	A234
crear	$\emptyset$	n	1	1	1	1	1	1
destruir		1	1	1	n	n	n	n
cardinalidad	$\lvert A \rvert$	1	1	1	1	1	1	1
pertenencia	$x \in A$	1	n	$\log_2 n$	n	n	h	$\log_2 n$
agregar	$A \cup x$	1	n	n	n	n	h	$\log_2 n$
eliminar	$A \setminus x$	1	n	n	n	n	h	$\log_2 n$

Objetivo casi cumplido: pertenencia, agregar y eliminar en $\approx \log_2 n$ pasos.