Tablas Hash y árboles binarios

Algoritmos

- Tablas hash
- Árboles Binarios
- Árboles Balanceados

Tablas Hash

Introducción

- Las tablas hash son estructuras tipo vector que ayudan a asociar claves con valores o datos
  - Estructura preferida para la implementación de diccionarios
  - Proveen un tiempo constante de búsqueda (O(1))
- Concepto clave:
  - No buscar directamente el valor deseado, sino a través de una función de dispersión h(x) localizar la posición del valor buscado

Ejemplo

- Considere que nos dan un conjunto de palabras y nos piden contar el numero de veces que parece cada palabra
  - Se requiere crear un diccionario que represente a cada palabra válida y a su ves, nos indique en número de instancias por palabra

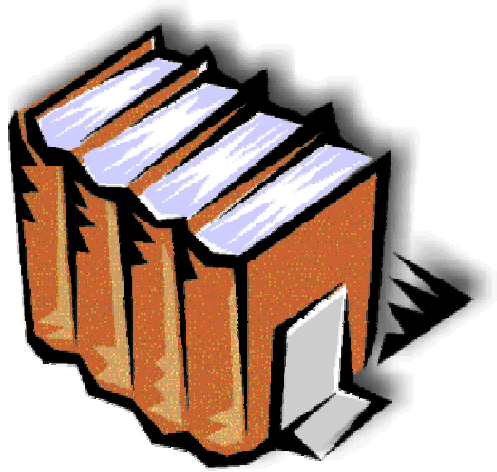
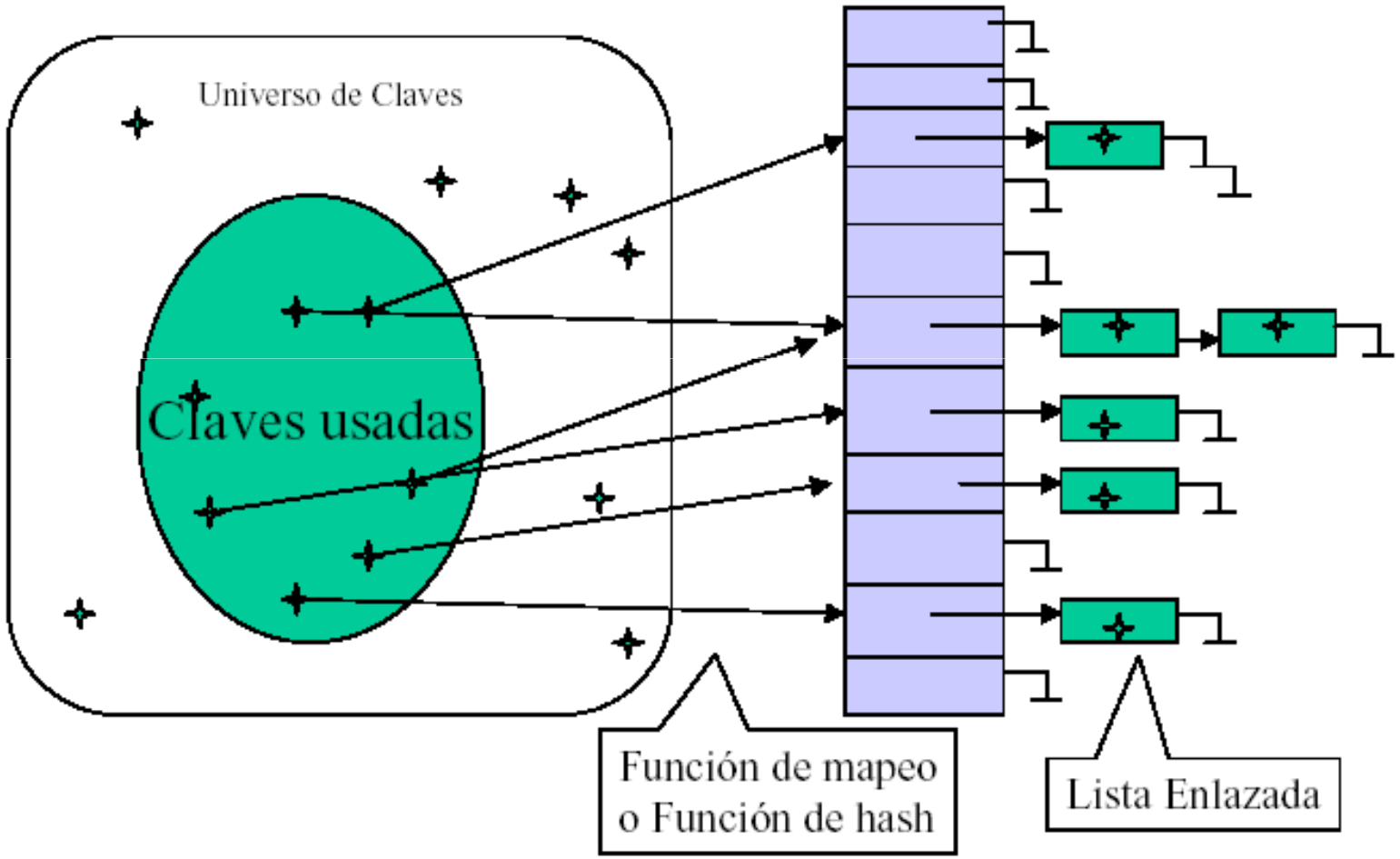


Tabla Hash

- Una tabla hash se construye con tres elementos básicos:
  - Un vector capaz de almacenar “m” elementos
  - Función de dispersión que permita a partir de los datos (llamados clave) obtener el índice donde estará el dato en el arreglo
  - Una función de resolución de colisiones

Tablas Hash


Universo de Claves
Claves usadas
Función de mapeo
o Función de hash
Lista Enlazada

Tablas Hash

- En el diseño de una tabla hash, es de gran importancia la elección de la función de dispersión
  - Función sencilla para una evaluación rápida
  - Distribuir uniformemente en el espacio de almacenamiento
  - Evitar (si es posible) la aparición de sinónimos o colisiones
  - Para dos claves similares, generar posiciones distantes

Tablas Hash

- Retornando al ejemplo de conteo de palabras
  - ¿como se podría definir una tabla hash para la búsqueda de una palabra en un diccionario?
- Idea
  - Transformar una palabra en un valor numérico entre [0, …, m) donde m representa el número de entradas en el arreglo que almacenará el dato

Tablas Hash

- Solución
  - Dado que se requiere construir una función que para dos valores similares, den resultados alejados entre si, se puede usar la función de Bernstain

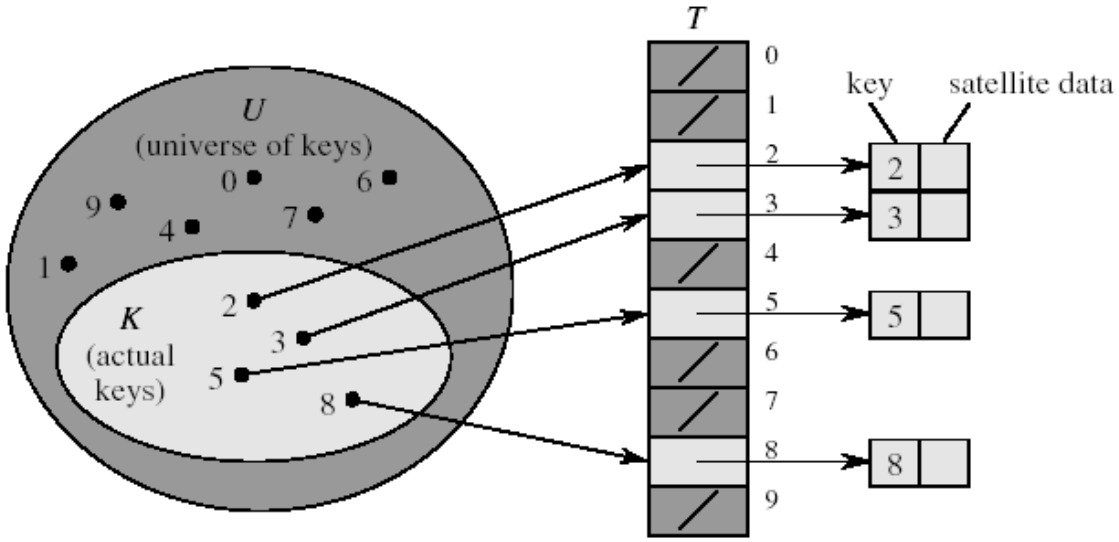
$$h = 33 * h + p[i]$$

Tablas Hash

- Ejemplo del conteo de cadenas:
  - Un string se puede interpretar como una secuencia de valores ASCII entre 0 – 255
  - Un string sería un número en base 255
  - Por ejemplo, la clave “pt” equivale a la entrada
    - 112 * 33 + 116

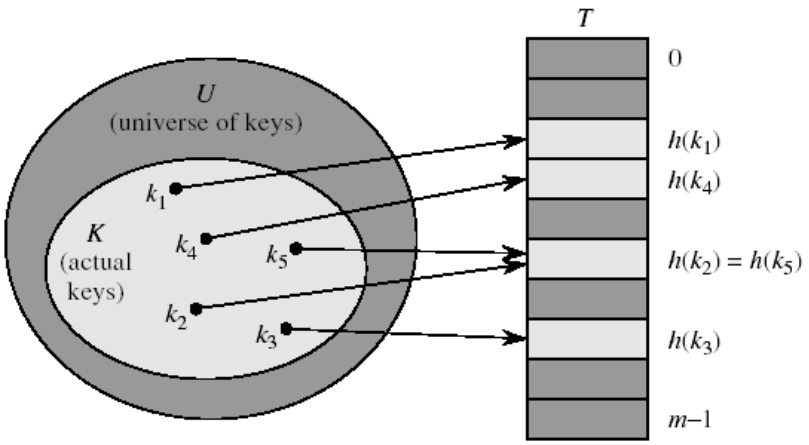
Tipos de Tablas Hash

- Existen diferentes tipos de tablas hash:
  - Tablas de dirección directa: cuando el universo de objetos U es relativamente pequeño
    Elemento con llave k → almacenado en el slot k



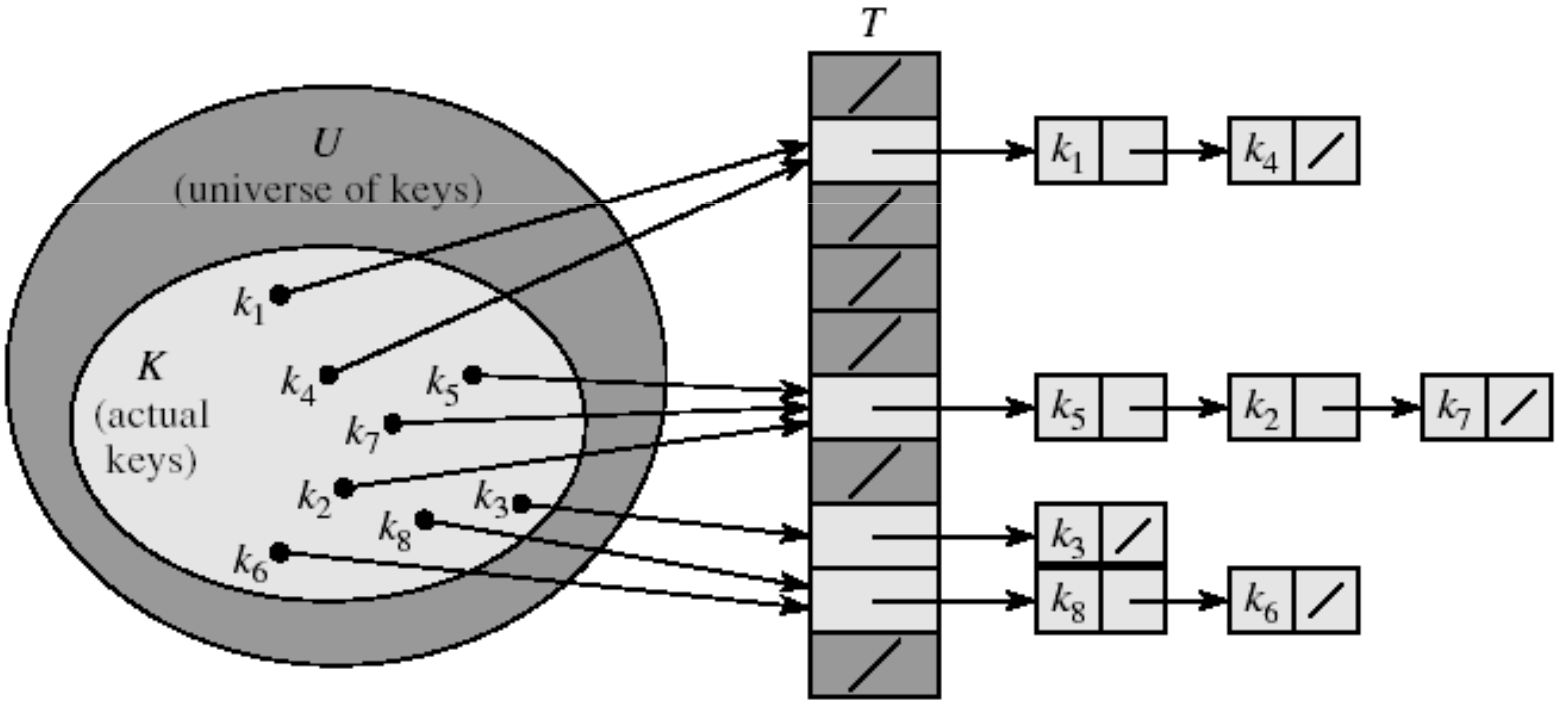
Tipos de Tablas Hash

- … tipos de tablas hash:
  - Tablas de dirección indirecta: Cuando el universo de objetos U es muy grande, se crea un vector T de dimensión m, tal que |U| > m
    Elemento con llave k → almacenado en el slot h(k)
    |U| > m → $\exists\ k_i, k_j \in U: h(k_i) = h(k_j)$ ← COLISIONES



Tipos de Tablas Hash

- Solución de colisiones a través de encadenamiento



Funciones Hash

- Una función hash h: U → {0, 1, …, m-1} (considerando un arreglo de dimensión m) debe de cumplir algunas condiciones:
  - Cada llave k_i debe tener la misma probabilidad de ser asignada a cualquiera de los “m” slots
- NOTA: esta condición es difícil de garantizar

Funciones Hash

- Método de división:
  - h(k) = k mod m
  - P.E. si m = 12 y k = 100 → h(k) = 4
- Si se utiliza el método de la división, se debe evitar que “m” sea una potencia de 2
  - Se recomienda utilizar un número primo “no cercano” a una potencia de 2

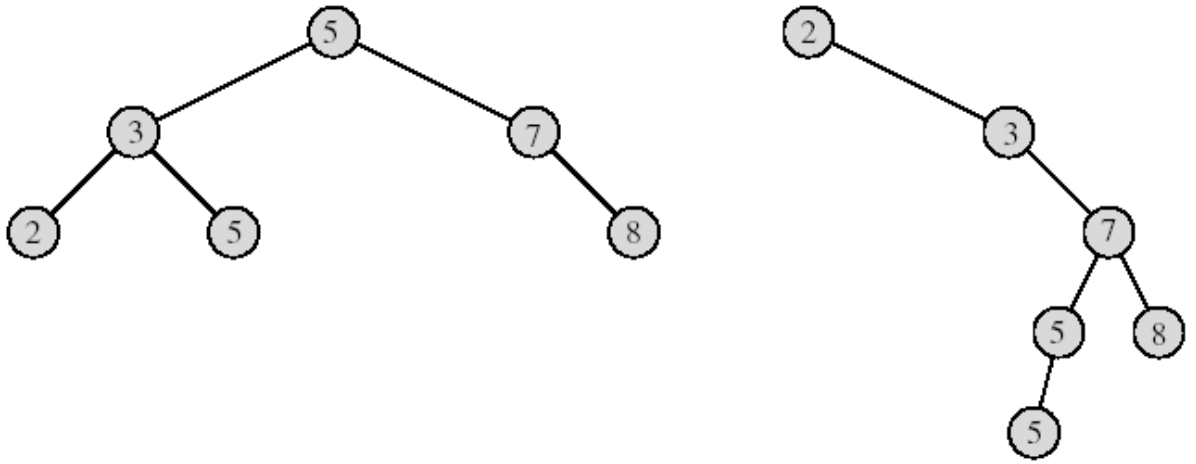
Funciones Hash

- Método de Multiplicación
  - Se multiplica la clave k por A, $0 < A < 1$ y se extrae la parte fraccionaria de k*A
  - Se multiplica el resultado por el número de entradas de la tabla y se toma el piso o techo
  - h(k) = int(m*(k*A – int(k*A)))
- Para este método, no se imponen restricciones sobre el valor de “m”

Árboles Binarios

Introducción

- Un árbol binario es una estructura tipo árbol donde cada nodo tiene los apuntadores:
  - Left: árbol izquierdo
  - Rigth: árbol derecho
  - p: padre



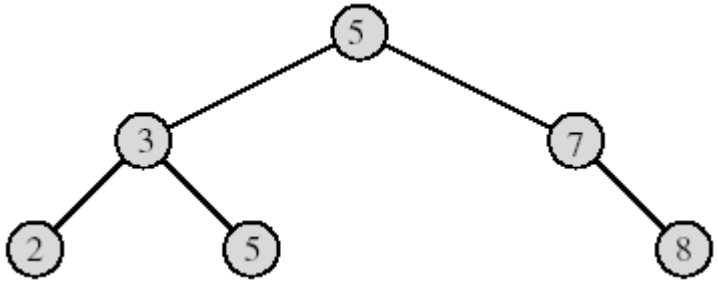
Introducción

- Regla general para un árbol binario
  - Sea “x” un nodo en un árbol binario
    - Si “y” es un nodo en el subárbol izquierdo de x → key[y] ≤ key[x]
    - Si “y” es un nodo en el subárbol derecho de x → key[x] ≤ key[y]

Recorrido de un Árbol Binario

- Para recorrer un árbol binario T, se puede utilizar una estrategia “inorder”, donde el llamado a la función es: INORDER-TREE-WALK(root[T])

```
INORDER-TREE-WALK(x)
1  if x ≠ NIL
2     then INORDER-TREE-WALK(left[x])
3          print key[x]
4          INORDER-TREE-WALK(right[x])
```



¿Recorrido del árbol?

Búsqueda en un Árbol Binario

- Se puede implementar una versión recursiva o iterativa:
  - Entrada: x (puntero a un nodo del árbol) – para el llamado inicial, es la raiz del árbol
  - Salida: puntero al nodo que contiene la llave
- Complejidad de la búsqueda:

  O(h)

  h: altura del árbol

```
TREE-SEARCH(x, k)
1  if x = NIL or k = key[x]
2     then return x
3  if k < key[x]
4     then return TREE-SEARCH(left[x], k)
5     else  return TREE-SEARCH(right[x], k)
```

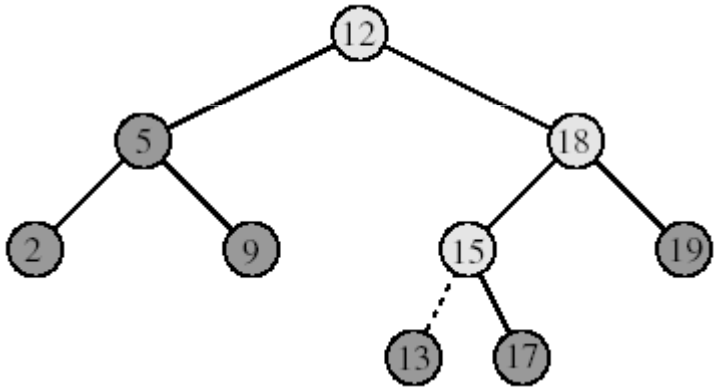
```
ITERATIVE-TREE-SEARCH(x, k)
1  while x ≠ NIL and k ≠ key[x]
2        do if k < key[x]
3              then x ← left[x]
4              else  x ← right[x]
5  return x
```

Construcción de un Árbol Binario

- Existen dos operaciones básicas en la construcción de un árbol binario:
  - Insertar
  - Borrar
- Ambas reglas deben de mantener la reglas generales de los árboles binarios

Inserción

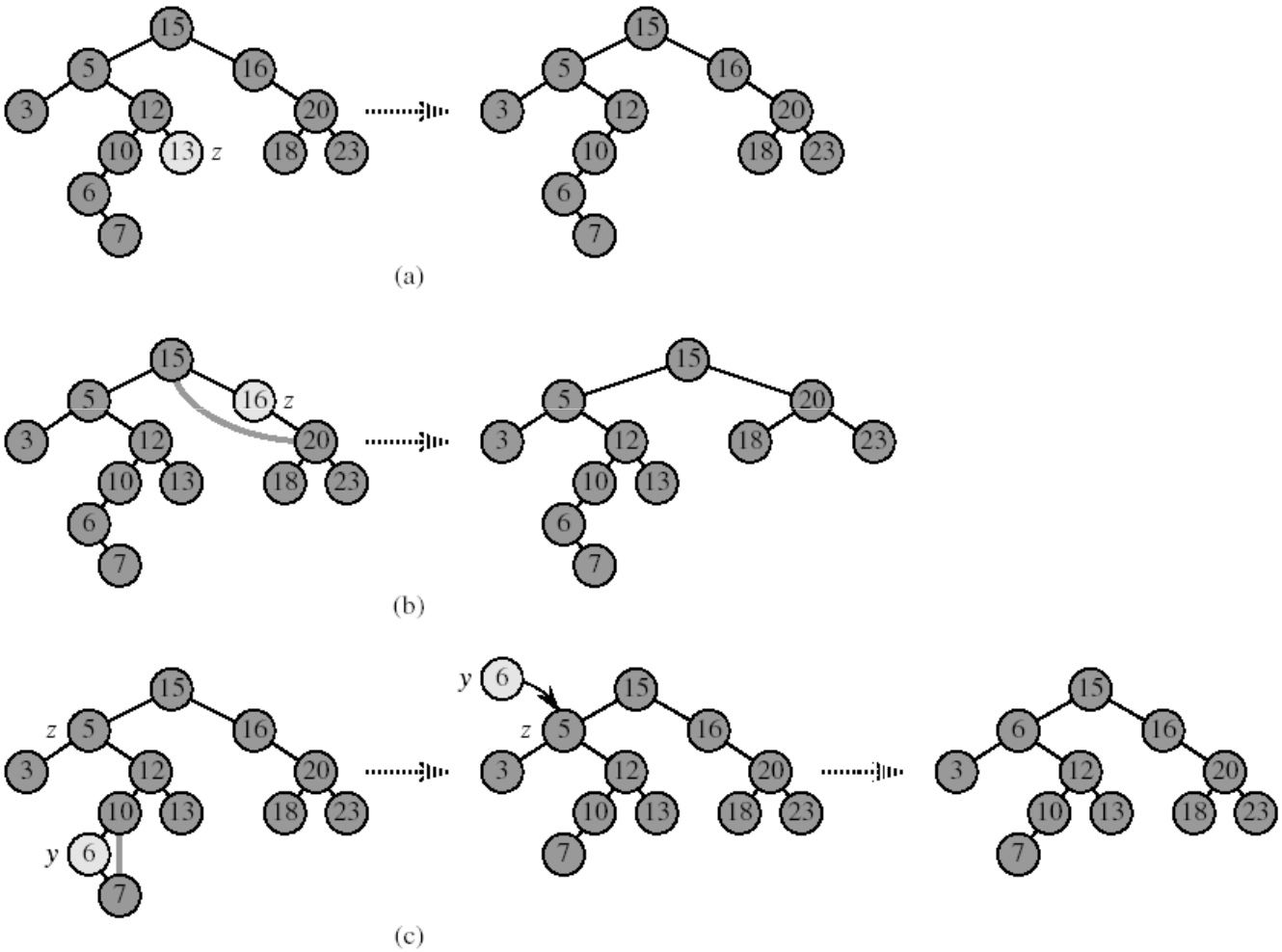
```
TREE-INSERT(T, z)
 1  y ← NIL
 2  x ← root[T]
 3  while x ≠ NIL
 4        do y ← x
 5           if key[z] < key[x]
 6              then x ← left[x]
 7              else  x ← right[x]
 8  p[z] ← y
 9  if y = NIL
10     then root[T] ← z
11     else  if key[z] < key[y]
12              then left[y] ← z
13              else  right[y] ← z
```



Borrar

```
TREE-DELETE(T, z)
 1  if left[z] = NIL or right[z] = NIL
 2     then y ← z
 3     else y ← TREE-SUCCESSOR(z)
 4  if left[y] ≠ NIL
 5     then x ← left[y]
 6     else x ← right[y]
 7  if x ≠ NIL
 8     then p[x] ← p[y]
 9  if p[y] = NIL
10     then root[T] ← x
11     else if y = left[p[y]]
12             then left[p[y]] ← x
13             else right[p[y]] ← x
14  if y ≠ z
15     then key[z] ← key[y]
16          copy y's satellite data into z
17  return y
```

Borrar


15
5
16
3
12
20
10
13
z
18
23
6
7
(a)
15
5
16
3
12
20
10
18
23
6
7
15
5
16
z
3
12
20
10
13
18
23
6
7
(b)
15
5
20
3
12
18
23
10
13
6
7
15
z
5
16
3
12
20
10
13
18
23
y
6
7
y
6
15
z
5
16
3
12
20
10
13
18
23
7
15
6
16
3
12
20
10
13
18
23
7
(c)