Documentacion Arbol-B

Maycol Gonzalez-Gerardo Carvajal

19 de abril de 2018

Índice

1. Descripcion Arbol-B

Los arboles b disenados para funcionar bien en discos u otros accesos directos dispositivos de almacenamiento secundario.Los arboles B son similares a los arboles red-black, pero son mejores para minimizar las operaciones de E / S de disco. Muchos sistemas de bases de datos utilizan B-trees, o variantes de B-trees, para almacenar informacion.

La idea tras los arboles-B es que los nodos internos deben tener un numero variable de nodos hijo dentro de un rango predefinido. Cuando se inserta o se elimina un dato de la estructura, la cantidad de nodos hijo varÃa dentro de un nodo. Para que siga manteniendose el numero de nodos dentro del rango predefinido, los nodos internos se juntan o se parten. Dado que se permite un rango variable de nodos hijo, los arboles-B no necesitan Re balancearse tan frecuentemente como los arboles binarios de busqueda auto-balanceables. Pero, por otro lado, pueden desperdiciar memoria, porque los nodos no permanecen totalmente ocupados.

2. Caracteristicas Arbol-B

Los Arboles B deben cumplir las sigientes caracteristicas:

- Toda pagina tiene como maximo 2n nodos.
- Toda pagina distinta de la raiz tiene como minimo n nodos.
- La raiz tiene como minimo un nodo.
- Todas las paginas hojas estan en el ultimo nivel.

Ademas de estas caracteristicas los arboles B tienen cumplir cierto orden:

- Los nodos dentro de una pagina mantienen un orden ascendente.
- Cada nodo es mayor que los nodos situados a su izquerda.
- Cada nodo es mayor que los nodos situados a su derecha.

3. Implementacion Arbol-B

En esta seccion, presentamos los detalles de algunas de las operaciones basicas del los arboles B como los es crear,Buscar, eliminar e Insertar:

- Crear Arbol B: Para construir un T arbol B, primero usamos B-TREE-CREATE para crear un nodo raiz vacio y luego agregamos nuevas claves.

```
B-TREE-CREATE(T)
1  x ← ALLOCATE-NODE()
2  leaf[x] ← TRUE
3  n[x] ← 0
4  DISK-WRITE(x)
5  root[T] ← x
```

- Buscar: Buscar un arbol B es muy parecido a buscar en un arbol binario de busqueda, excepto que en lugar de hacer un decision de bifurcacion binaria o "bidireccional.[en] cada nodo, hacemos una bifurcacion de multiples vias decision segun el numero de hijos del nodo.

```
B-TREE-SEARCH(x, k)
1 i ← 1
2 while i ≤ n[x] and k > key_i[x]
3         do i ← i + 1
4 if i ≤ n[x] and k = key_i[x]
5     then return (x, i)
6 if leaf [x]
7     then return NIL
8     else DISK-READ(c_i[x])
9          return B-TREE-SEARCH(c_i[x], k)
```

Usando un procedimiento de busqueda lineal, las lineas 1-3 encuentran el indice mas pequeno i tal que k ¡= key i (x), o de lo contrario, establecen i a n [x] + 1. Las lineas 4-5 comprueban si ahora hemos descubierto la clave, volviendo si tenemos Las lineas 6-9 terminan la busqueda sin exito (si x es una hoja) en el busque el subarbol apropiado de x, despues de realizar el DISK-READ necesario en ese hijo

- Insertar: Las inserciones se hacen en los nodos hoja.

  - 1. Realizando una busqueda en el arbol, se halla el nodo hoja en el cual deberia ubicarse el nuevo elemento.
  - 2. Si el nodo hoja tiene menos elementos que el maximo numero de elementos legales, entonces hay lugar para uno mas. Inserte el nuevo elemento en el nodo, respetando el orden de los elementos.

  - 3. De otra forma, el nodo debe ser dividido en dos nodos. La division se realiza de la siguiente manera:
    - Se escoge el valor medio entre los elementos del nodo y el nuevo elemento.
    - Los valores menores que el valor medio se colocan en el nuevo nodo izquierdo, y los valores mayores que el valor medio se colocan en el nuevo nodo derecho; el valor medio actua como valor separador.
    - El valor separador se debe colocar en el nodo padre, lo que puede provocar que el padre sea dividido en dos, y asi sucesivamente.

```
B-TREE-INSERT(T, k)
 1  r ← root[T]
 2  if n[r] = 2t - 1
 3     then s ← ALLOCATE-NODE()
 4          root[T] ← s
 5          leaf[s] ← FALSE
 6          n[s] ← 0
 7          c1[s] ← r
 8          B-TREE-SPLIT-CHILD(s, 1, r)
 9          B-TREE-INSERT-NONFULL(s, k)
10     else B-TREE-INSERT-NONFULL(r, k)
```

Insercion normal

```
B-TREE-INSERT-NONFULL(x, k)
 1  i ← n[x]
 2  if leaf[x]
 3     then while i ≥ 1 and k < keyi[x]
 4            do keyi+1[x] ← keyi[x]
 5                  i ← i - 1
 6           keyi+1[x] ← k
 7           n[x] ← n[x] + 1
 8           DISK-WRITE(x)
 9     else while i ≥ 1 and k < keyi[x]
10            do i ← i - 1
11          i ← i + 1
12          DISK-READ(ci[x])
13          if n[ci[x]] = 2t - 1
14            then B-TREE-SPLIT-CHILD(x, i, ci[x])
15                   if k> keyi[x]
16                     then i ← i + 1
17          B-TREE-INSERT-NONFULL(ci[x], k)
```

Insercion con divicion de nodo

- Eliminar:La eliminacion de un B-tree es analoga a la insercion, pero un poco mas complicada, porque una clave puede eliminarse de cualquier nodo, no solo una hoja, y la eliminacion de un nodo interno requiere que los hijos del nodo se reorganizen. Al igual que en la insercion, debemos protegernos contra la eliminacion produciendo un arbol cuya estructura viola las propiedades del arbol B. Asi como tunemos que asegurarnos de que un nodo no se hace demasiado grande debido a la insercion, debemos asegurarnos de que un nodo no se vuelva demasiado pequeno durante la eliminacion (excepto que la raiz tiene permitido tener menos que el numero minimo t - 1 de teclas, aunque no esta permitido tener mas que el numero maximo 2t - 1 de teclas).

4. Referencias

- Libro CLRS.