CS150 Discussion IV

Jiachun Jin
jinjch@shanghaitech.edu.cn

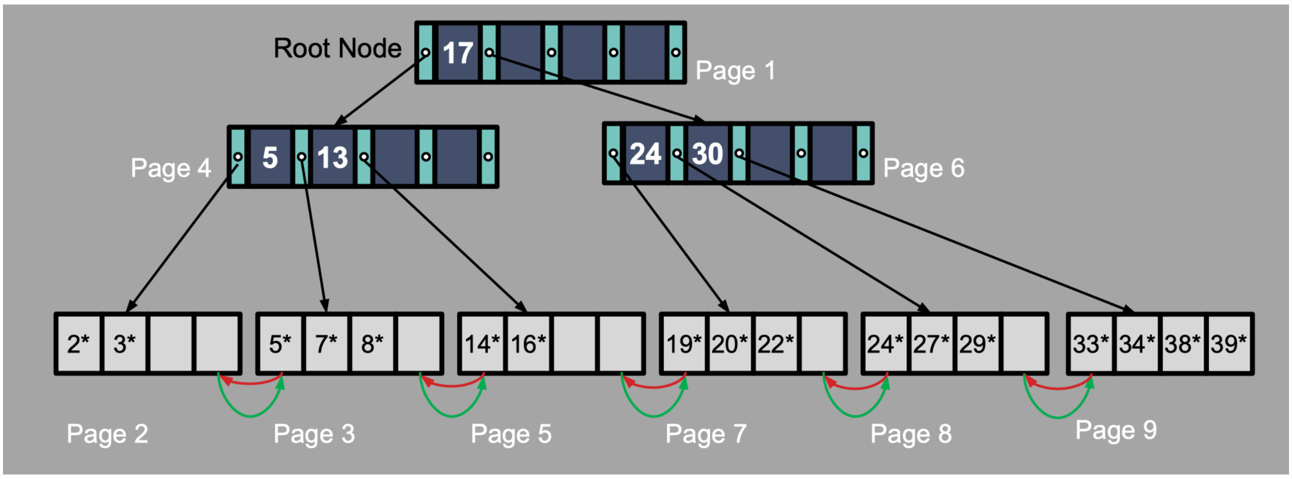
1. B^+ Tree

A quick review

- balanced tree
- internal nodes direct the search, the leaf nodes contain the data entries
- doubly linked list
- a parameter $d :=$ the order of the tree

  $d \leq \#$ entries $\leq 2d$ for interior nodes

  $1 \leq \#$ entries $\leq 2d$ for root node



Cost model for search

d := order of the B^+ Tree

f := fanout, $f \in [d+1, 2d+1]$, here assume it's constant for simplicity

N := total number of pages we'd like to index

F := fill factor (~usually $2/3$)

B := #available buffer pages

- our B^+ Tree needs to have room to index N/F pages
- what is the height h of our B^+ Tree?

  $h = \lceil \log_f \frac{N}{F} \rceil$
- L_B is the number the number of levels such that the sum of all the levels' nodes fit in the buffer

  $B \geq 1 + f + \cdots + f^{L_B - 1} = \sum_{l=0}^{L_B - 1} f^l$
- IO cost: $\lceil \log_f \frac{N}{F} \rceil - L_B + 1$, where $B \geq 1 + f + \cdots + f^{L_B - 1} = \sum_{l=0}^{L_B - 1} f^l$

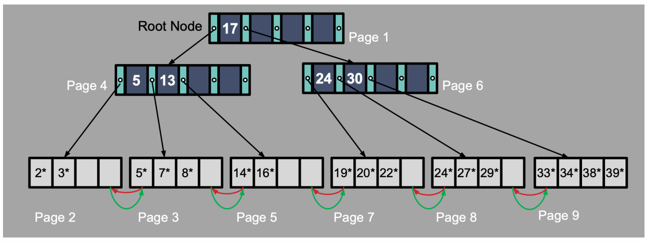
$+\lceil \log_f \frac{N}{F} \rceil$: We read in one page per level of the tree

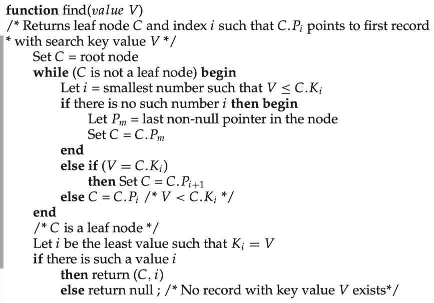
$-L_B$: However, levels that we can fit in buffer are free!

$+1$: Finally we read in the actual record

Examine the codes

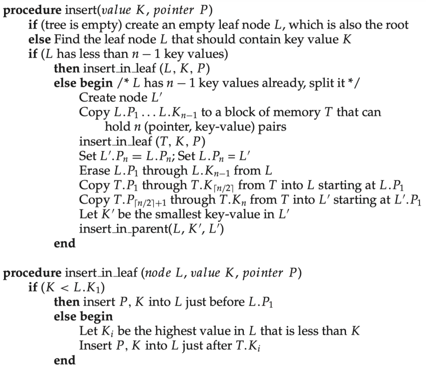
find





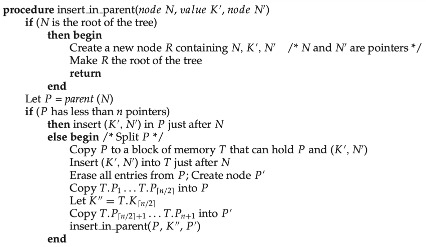
```
function find(value V)
/* Returns leaf node C and index i such that C.P_i points to first record
* with search key value V */
    Set C = root node
    while (C is not a leaf node) begin
        Let i = smallest number such that $V \leq C.K_i$
        if there is no such number i then begin
            Let $P_m$ = last non-null pointer in the node
            Set $C = C.P_m$
        end
        else if ($V = C.K_i$)
            then Set $C = C.P_{i+1}$
        else $C = C.P_i$ /* $V < C.K_i$ */
    end
    /* C is a leaf node */
    Let i be the least value such that $K_i = V$
    if there is such a value i
        then return (C, i)
        else return null ; /* No record with key value V exists*/
```

insert



```
procedure insert(value K, pointer P)
    if (tree is empty) create an empty leaf node L, which is also the root
    else Find the leaf node L that should contain key value K
    if (L has less than $n - 1$ key values)
        then insert_in_leaf (L, K, P)
        else begin /* L has $n - 1$ key values already, split it */
            Create node $L'$
            Copy $L.P_1 \ldots L.K_{n-1}$ to a block of memory T that can
                hold n (pointer, key-value) pairs
            insert_in_leaf (T, K, P)
            Set $L'.P_n = L.P_n$; Set $L.P_n = L'$
            Erase $L.P_1$ through $L.K_{n-1}$ from L
            Copy $T.P_1$ through $T.K_{\lceil n/2 \rceil}$ from T into L starting at $L.P_1$
            Copy $T.P_{\lceil n/2 \rceil + 1}$ through $T.K_n$ from T into $L'$ starting at $L'.P_1$
            Let $K'$ be the smallest key-value in $L'$
            insert_in_parent(L, $K'$, $L'$)
        end

procedure insert_in_leaf (node L, value K, pointer P)
    if ($K < L.K_1$)
        then insert P, K into L just before $L.P_1$
        else begin
            Let $K_i$ be the highest value in L that is less than K
            Insert P, K into L just after $T.K_i$
        end
```



```
procedure insert_in_parent(node N, value $K'$, node $N'$)
    if (N is the root of the tree)
        then begin
            Create a new node R containing N, $K'$, $N'$    /* N and $N'$ are pointers */
            Make R the root of the tree
            return
        end
    Let P = parent (N)
    if (P has less than n pointers)
        then insert ($K'$, $N'$) in P just after N
        else begin /* Split P */
            Copy P to a block of memory T that can hold P and ($K'$, $N'$)
            Insert ($K'$, $N'$) into T just after N
            Erase all entries from P; Create node $P'$
            Copy $T.P_1 \ldots T.P_{\lceil n/2 \rceil}$ into P
            Let $K'' = T.K_{\lceil n/2 \rceil}$
            Copy $T.P_{\lceil n/2 \rceil + 1} \ldots T.P_{n+1}$ into $P'$
            insert_in_parent(P, $K''$, $P'$)
        end
```

2. Buffer Management

Exercises

1. What does it mean to say that a page is pinned in the buffer pool? Who is responsible for pinning pages? Who is responsible for unpinning pages?

2. Name an important capability of a DBMS buffer manager that is not supported by a typical operating system's buffer manager.

3. What happens if a page is requested when all pages in the buffer pool are dirty?