CS540: Assignment 3

Ga Young Lee
Department of Electrical Engineering and Computer Science
Oregon State University
leegay@oregonstate.edu

Jeffrey M. Young
Department of Electrical Engineering and Computer Science
Oregon State University
youngjef@oregonstate.edu

February 11, 2021

1 Problem 1

Problem Description:
Assume that the block size is 30 bytes. In this question, each node in a B+ tree must fit in a block and its size must be as close as possible to the size of a block. Answer the following parts using B+ trees.

1.a

Problem Description:
Assume that all data values are integers with the fixed size of 8 bytes and each record pointer takes at most 4 bytes. Find a B+ tree whose height changes from 2 to 3 when the value 20 is inserted. Note that the height of a tree is depth of the deepest node plus one. For example, the height of a tree with a single node is 1. Show your structure before and after the insertion.

To find out the structure of a B+ tree, we need to calculate the maximum number of keys to be stored in a node. Let m be the number of entries, search keys, in a node and $(m + 1)$ be the number of pointers in a node. Since each leaf node must be able to fit m values with $(m + 1)$ pointers in a block, we can calculate the maximum number of m to be contained in a node by the following equation:

$$f(m) = \text{Search Key Size} * (m) + \text{Pointer Size} * (m + 1) \leq \text{Block Size}$$

Plugging in the given block size gives us $f(m) = 8*m+4*(m+1) \leq 30$, which is simplified to $12m \leq 26$. By solving the inequality, we infer that each node can have up to 2 entries. Additionally, by the minimum 50% occupancy rule for each node, there should be at least 1 key in a node. Given this hint, we can construct a B+ tree as follows.

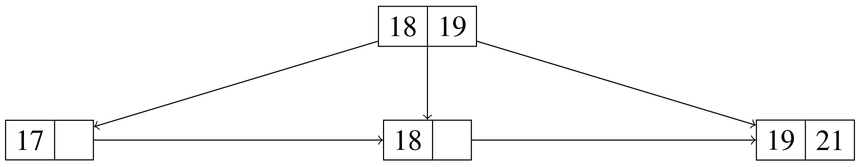

Figure 1: B+Tree before insertion of the value 20.

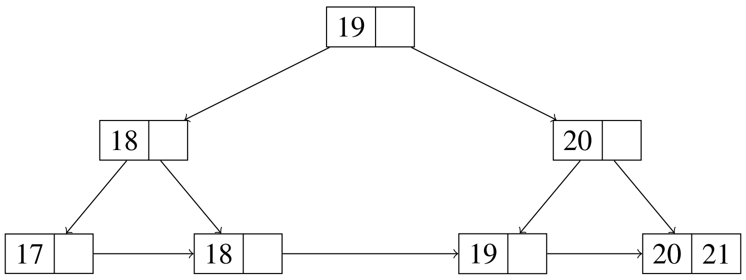

Figure 2: B+Tree after the insertion of 20. Notice the height increases from 2 to 3 and two nodes were split.

We assume a right-biased tree, thus for some index K_i, all children from pointer $P_{i-1} < P_i$. This means that all nodes pointed to by P_i are strictly $\geq K_i$. Insertion of 20 into the tree shown in Figure 1 results in two nodes being split. First we cannot simply place 20 in the 19—21 node because it is full. Thus, we must split the node, choosing the middle value—the value 20—as the key. This means that we construct a new node where we will have the $<$20 child hold 19, and the $>$19 but $\leq$20 child hold 20, and the $>$ 20 node hold 21. Unfortunately the root node is also full, thus when we split the 19—21 node in Figure 1 we must also split the root node. We choose the middle value again, in this case 19, and perform the split. Normally we would have a 19—20 node where the 20 node was created in Figure 2 however this would violate a constraint for the B+Tree, thus we use 20, and the data value of 19 is held in the $<$20 node.

1.b

Problem Description:
Assume that all data values are integers with the fixed size of 4 bytes and each record pointer takes at most 4 bytes. Find a B+ tree in which the deletion of the value 40 leads to a redistribution. Show your structure before and after the deletion. Your example could be different from the answer given for part (a).

First, we must calculate the maximum number of keys in each node by considering the maximum occupancy of a block. The equation is as follows:

$$f(m) = \text{Search Key Size} * (m) + \text{Pointer Size} * (m+1) \leq \text{Block Size}.$$

Then, we have $f(m) = 4 * m + 4 * (m+1) \leq 30$, so $m \leq 3.25$. In other words, a leave node can have at most 3 entries. Additionally, the minimum occupancy of each node is 50% of the at least 2 entries to comply with the minimum 50% occupancy rule except for the root node. Root node can hold one entry.

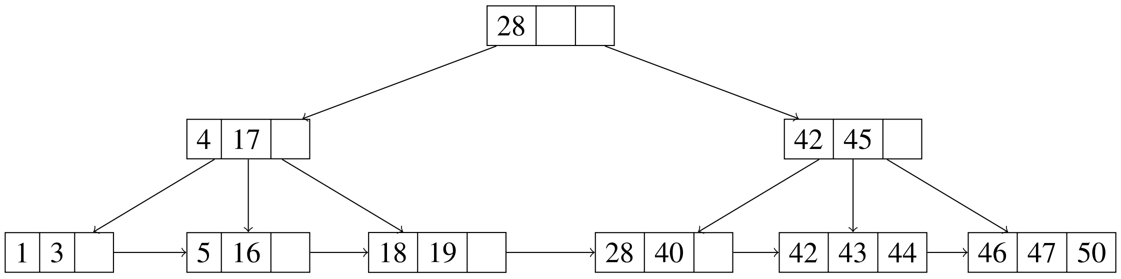

Figure 3: B+Tree before deletion of 40

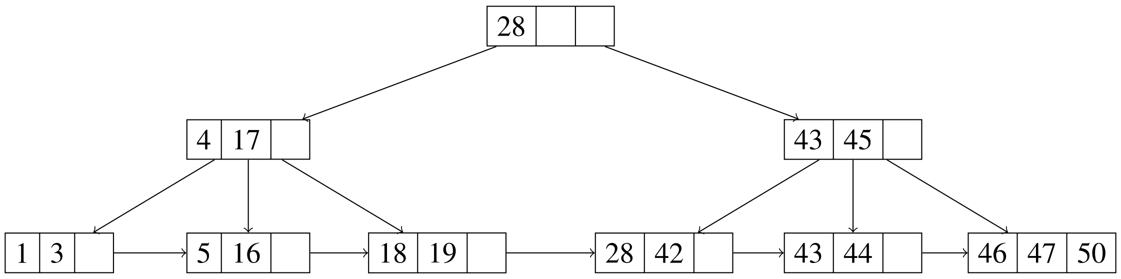

Figure 4: Resulting B+Tree after deletion

Deleting 40 requires the merge of two nodes and a re-balance through the root. We remove 40 from the data and the key, which leaves a node with only one key. This is a violation of the 50% occupancy rule, so we need to borrow from its neighbor. Thus, we borrow 42 from its neighboring node and update the key on level 2 (root is level 1, data is level 3). We rotate 42 through the root, when we perform the rotation the 28—42 node violates the BST constraint and so it must point to the new key node, indexed by 43, as the new <43 child.

2 Problem 2

2.a

Problem Description:
Consider the following relational schema:

Emp(eid:integer,ename:string,age:integer,salary:real)

The underlined attributes are keys for their relations.

Consider the following SQL query:

```
Select * From Emp
Where age = 20 and salary > 20000
```

Problem Description:
Describe the index on Emp that improves the running time of this query more than every other index over most Emp relation instances. You may explain the attributes, their order, type (B+tree or hash) of the index and whether it should be clustered.

In the general case, indexes slow down data manipulation but make search and retrieval fast. Because our example query involves *is only* search and retrieval, we can construct a clustered index on `age`, and `salary` attributes. This will form an optimal key and index *for only this query* at the cost of queries which perform manipulation.

The given query contains a range retrieval of the data over `age` and `salary`, so it is beneficial to store the data by a composite key of `age` and `salary` to accelerate the retrieval. It is safe to disregard `eid` since it is not included in the selection criterion. Thus, if we include both `age` and `salary` as a composite key and implement with a B+Tree, then the composite key will perform optimally for a retrieval and a range selection because of the linked nature of the data leaves in the B+Tree. Therefore, in this given scenario, it is recommended to use B+ tree and clustering for efficient retrieval of a range query using a composite key. So, we'll have a composite key over a cluster B+ tree as follows:

```
  CREATE INDEX Age_Salary ON EMP(age,salary);
```