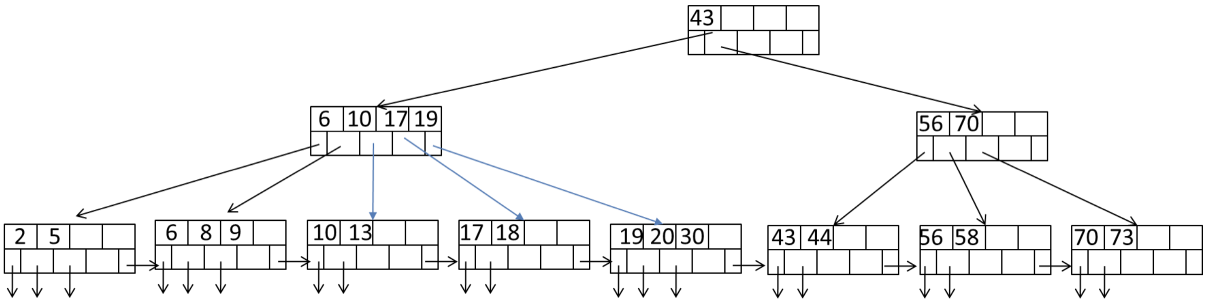
Name: Chaoyu Li
Date: 04/03/2022

DSCI551-HW4

(Indexing and Query Execution)

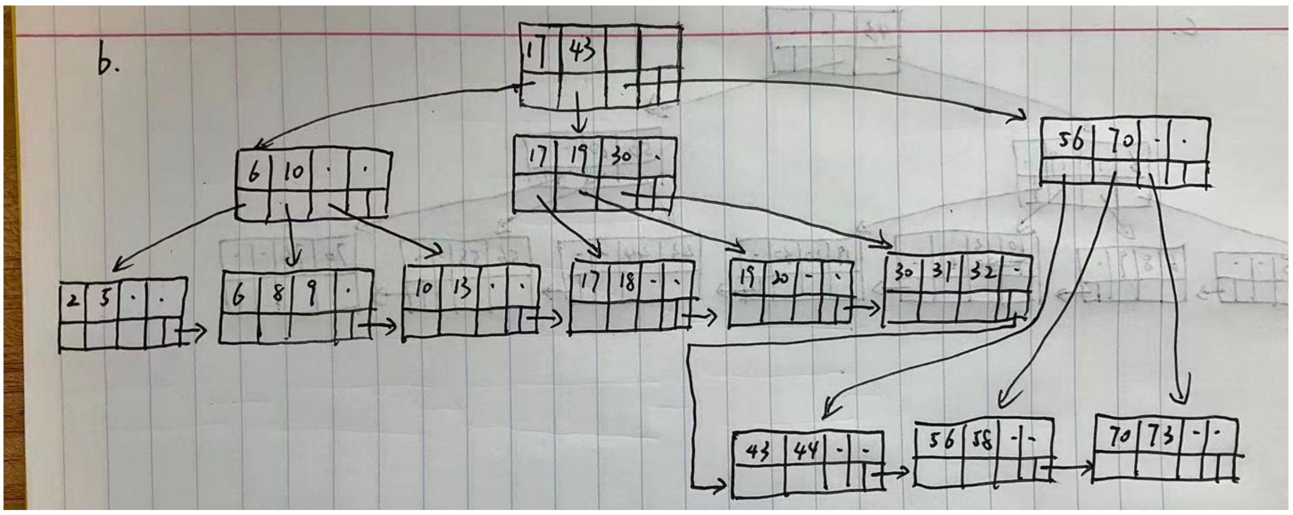
1. [40 points] Consider the following B+tree for the search key "age. Suppose the degree d of the tree = 2, that is, each node (except for root) must have at least two keys and at most 4 keys. **Note that sibling nodes are nodes with the same parent.**



a. [10 points] Describe the process of finding keys for the query condition "age >= 10 and age <= 50". How many blocks I/O's are needed for the process?

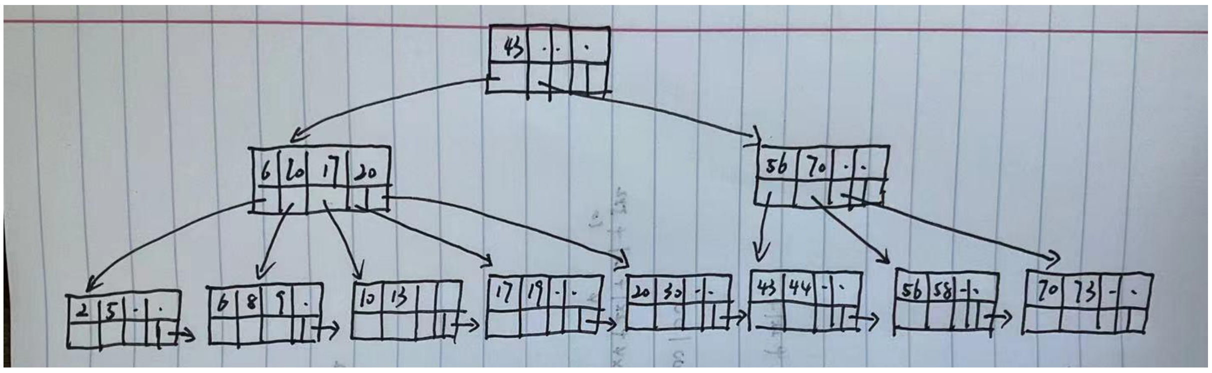
A: (1) Read the root node, find the left child node of the root node.
(2) Read the node with [6,10,17,19], find the 3rd leaf node.
(3) Read the 3rd leaf node, starting with the point 10.
(4) Sequential traversal of leaves until 50. Therefore, it will read 4nd, 5nd and 6nd **and 7nd(需要读到56才知道比50大了)** leaf nodes.
The number of blocks I/O's should be: (1+1+**5**)=**7** Read + 0 Write = **7**.

b. [15 points] Draw the B+-tree after inserting 31 and 32 into the tree. Only need to show the final tree after the insertions.



第二层的**17**应该被删掉，因为**root**层里已经包含**17**了。

c. [15 points] Draw the tree after deleting 18 from the original tree.



2. [60 points] Consider natural-joining tables R(a, b) and S(a,c). Suppose we have the following scenario.

   I. R is a clustered relation with 1000 blocks.

   II. S is a clustered relation with 500 blocks.

   III. 102 pages available in main memory for the join.

   IV. Assume the output of join is given to the next operator in the query execution plan (instead of writing to the disk) and thus the cost of writing the output is ignored.

Describe the steps (including input, output, and their sizes at each step, e.g., sizes of runs or buckets) for each of the following join algorithms. What is the total number of block I/O's needed for each algorithm? Which algorithm is most efficient?

a. [10 points] (Block-based) nested-loop join with R as the outer relation.

A: R⋈S

Pseudocode:

```
For each (102-2) blocks $b_r$ of R do:
    For each block $b_s$ of S do:
        For each tuple $t_s$ of S do:
            For each tuple $t_r$ of R do:
                If "$t_s$ and $t_r$ join" then output($t_s$, $t_r$)
```

Number of block I/O's:

(1) Read R one time.

(2) There are B(R)/100 outer loops, each loop reads S one time.

Total: B(R) + B(R)*B(S)/100 = 1000 + 1000*500/100 = 6000 block I/O's.

b. [10 points] (Block-based) nested-loop join with S as the outer relation.

A: S⋈R

Pseudocode:

```
For each (102-2) blocks $b_s$ of S do:
    For each block $b_r$ of R do:
        For each tuple $t_s$ of S do:
            For each tuple $t_r$ of R do:
                If "$t_s$ and $t_r$ join" then output($t_s$, $t_r$)
```

Number of block I/O's:

(1) Read S one time.

(2) There are B(S)/100 outer loops, each loop reads R one time.

Total: B(S) + B(S)*B(R)/100 = 500 + 1000*500/100 = 5500 block I/O's.

c. [20 points] Sort-merge join (assume only 100 pages are used for sorting and 101 pages for merging). Note that if join can not be done by using only a single merging pass, runs from one or both relations need to be further merged, in order to reduce the number of runs. Select the relation with a larger number of runs for further merging first if both have too many runs.

A: (1) Read 100 blocks of R each time, sort them and send them back to the disk. This will create 10 runs, each size is 100. Cost = 2B(R) = 2000. (100 blocks/run)

(2) Read 100 blocks of S each time, sort them and send them back to the disk. This will create 5 runs, each size is 100. Cost = 2B(S) = 1000. (100 blocks/run)

(3) Because the number of runs of R=10 < 100 and runs of S=5 < 100, we can merge them directly. Cost = B(S) + B(R) = 1500.

Total: 3B(R) + 3B(S) = 4500 block I/O's.

d. [20 points] Partitioned-hash join (assume 101 pages used in partitioning of relations and no hash table is used to lookup in joining tuples).

A: (1) Hash R into 100 buckets and send them back to disk: Cost = 2B(R) = 2000. (10 blocks/bucket)

(2) Hash S into 100 buckets and send them back to disk: Cost = 2B(S) = 1000. (5 blocks/bucket)

(3) Join corresponding buckets from R and S Cost: B(S) + B(R) = 1500.

Total: 3B(R) + 3B(S) = 4500 block I/O's.

Therefore, the Sort-merge join and the Partitioned-hash join are both the most efficient.