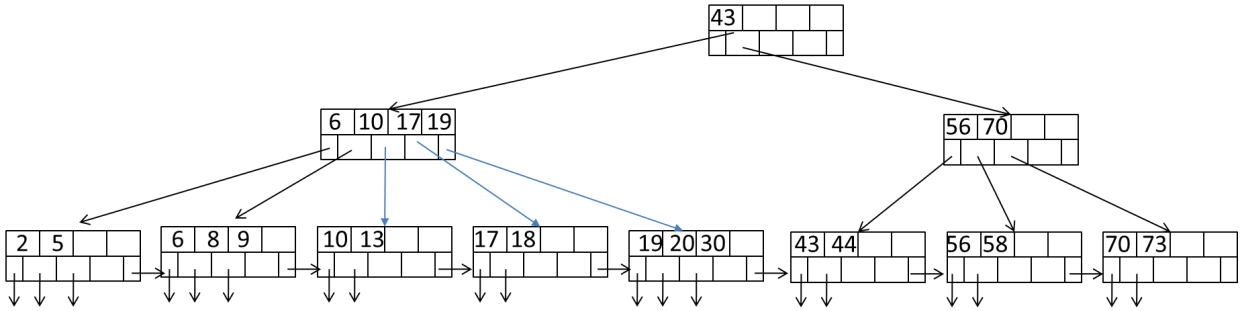
DSCI 551 – HW4

(Indexing and Query Execution)

(Fall 2022)

100 points, Due 11/14, Monday

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 >= 35 and age <= 65". How many blocks I/O's are needed for the process?

Ans:

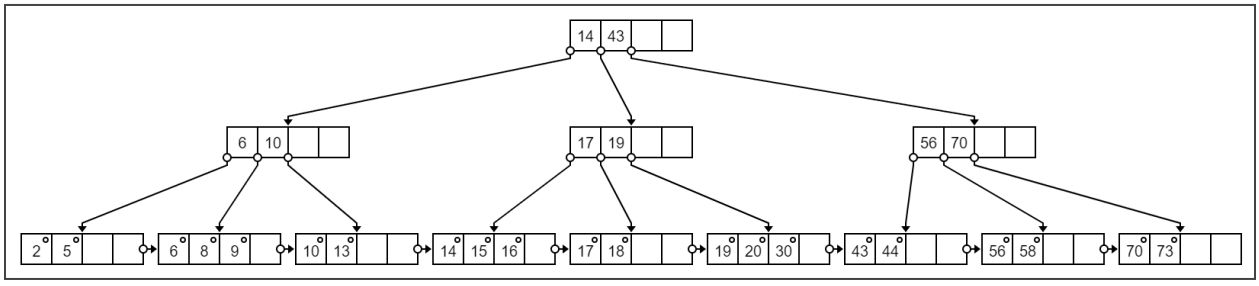
1) Read block (43)
2) Read block (6, 10, 17,19).
3) Read block (19,20,30), block (43,44), block(56,58), block(70,73).

Since 70 > 65, stop.

Total I/O's: **6**

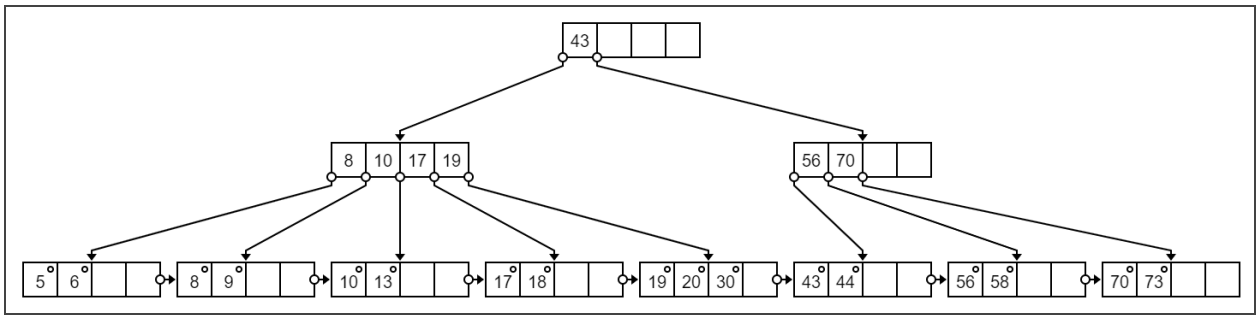
   b. [15 points] Draw the B+-tree after inserting 14, 15, and 16 into the tree. Only need to show the final tree after all insertions.

Ans:



   c. [15 points] Draw the tree after deleting 2 from the original tree.

Ans:



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 5,000 blocks.
    ii. S is a clustered relation with 20,000 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 for each of the following join algorithms. For sorting and hashing-based algorithms, also indicate the sizes of output from each step. What is the total number of block I/O's needed for each algorithm? Which algorithm is most efficient in terms of block's I/O?

    a. [10 points] (Block-based) nested-loop join with R as the outer relation.
    ANS: Here we will use 102-2 = 100 blocks for loading R. After loading R, we load one block of S into memory and merge with each block of R. Finally, the result will store in a one block memory and output.
    Cost: B(R)+B(R)/(M-2)*B(S)=5000+50*20000=1005000
    b. [10 points] (Block-based) nested-loop join with S as the outer relation.
    ANS: Here we will use 102-2 = 100 blocks for loading S. After loading S, we load one block of R into memory and merge with each block of S. Finally, the result will store in a one block memory and output.
    Cost: B(S)+B(S)/(M-2)*B(R)=20000+200*5000=1020000

    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.
    ANS: First we use 100 blocks for sorting and one buffer for output. The number of runs for R after first sorting is 5000/100=50 runs. Produce 50 runs (each run has 100 blocks). The number of runs for S after first sorting is 20000/100=200 runs. Produce 200 runs (each run has 100 blocks). Since 200 + 50 > 100, we need further sorting. Sort 200 runs of S into 200/100 = 2 runs. Produce 2 runs (each run has 10000 blocks). Finally, we could merge by loading 50+2 runs into memory.
    Cost:2*B(R) + 4*B(S)+B(R)+B(S)= 115000

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).
ANS: First hash R into 100 buckets (50 blocks/buckets) and hash S into 100 buckets (200 blocks/buckets). Write them back to storage. Note that Min(B(S),B(R))/100=50 < 99. Thus, we could join without further hashing.
Cost: 2*B(S)+2*B(R) + B(S) + B(R)= 75000

Result: By comparing the cost, we could know that Partitioned-hash join is more efficient.