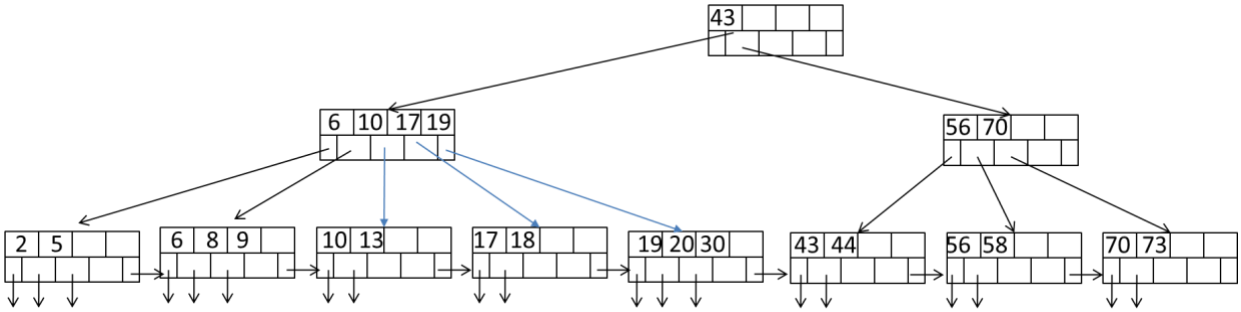
DSCI 551 – HW4

(Indexing and Query Execution)

(Fall 2020)

100 points, Due 4/18, Sunday

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.**

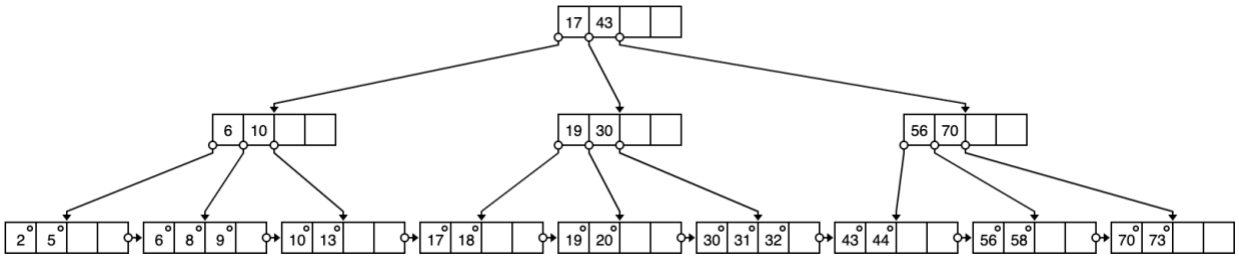


   a. Describe the process of finding keys for the query condition "age >= 10 and age <= 20". How many blocks I/O's are needed for the process?
   b. Draw the B+tree after inserting 31 and 32 into the tree. Only need to show the final tree after the two insertions.
   c. Draw the tree after deleting 43 from the original tree.

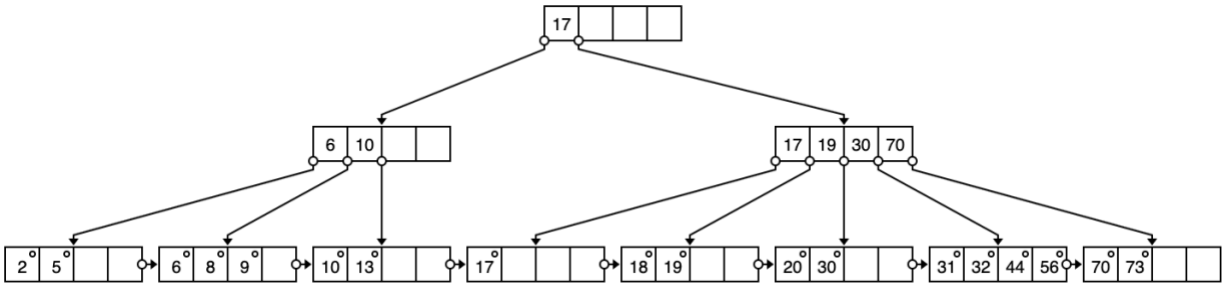
a.

- Start with reading the root node. Find the intermediate node which contains the key=10.
- Read the second leaf node to find the starting point 10.
- Continue, sequentially reading the subsequent leaves until 20 (reading 3rd, 4th and 5th node).
- We found the endpoint that is 20 in the 5th leaf node, so we stop the process.
- Total cost = 6 I/Os

b.



c.



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 2,000 blocks and 10,000 tuples
    ii. S is a clustered relation with 50,000 blocks and 100,000 tuples
    iii. S has a clustered index on the join attribute a
    iv. V(S, a) = 5 (recall that V(S, a) is the number of distinct values of a in S
    v. 102 pages available in main memory for the join.
    vi. 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. Nested-loop join with R as the outer relation

ANSWER

For each (102-2) block b_r of R do

For each block b_s of S do

For each tuple r in b_r do

For each tuple s in b_s do

If r and s join then output (r,s)

Cost: Read R once costs 2,000

Outer loop runs 2,000/ (102-2) times, each time need to read S: costs 2,000*50,000/ (102-2)

Overall cost = 2000 + 2000*50000/100 = 2000 + 100,000 = 102,000 I/O s

b. Sort-merge join (assume only 100 pages used for sorting and 101 pages for merging). Note that if join cannot 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.

- Load 100 blocks of R each time, sort them and send them back to the disk. This will create 20 runs of size 100 each.
  - Cost = 2*2000 = 4,000
- Load 100 blocks of R each time, sort them and send them back to the disk. This will create 500 runs of size 100 each.
  - Cost = 2*50,000 = 100,000

As the number of runs of R > 100 and S > 100 they cannot be merged directly, so, we merge them using ceil (20/100) = 1 and ceil (500/100) = 5 runs, respectively.

- Load 20 runs of R merge them and send them back to the disk. This will create 1 run.
  - Cost = 2*2,000 = 4,000
- Load 500 runs of R each time, merge them and send them back to the disk. This will create 5 runs.
  - Cost = 2*50,000 = 100,000
- Merge 1 run from R and 5 runs from S, join all the sorted runs.
  - Cost = 2,000 + 50,000 = 52,000

Total Cost = 4000 + 4000 + 100,000 + 100,000 + 52,000 = 260,000 I/O s.

c. Partitioned-hash join (assume 101 pages used in partitioning of relations and no hash table used for lookup in joining buckets)

- Hash R into 100 buckets and send them back to the disk.
  - Cost = 2*2000 = 4,000
- Hash S into 100 buckets and send them back to the disk.
  - Cost = 2*50,000 = 100,000
- Join the buckets from R and S.
  - Cost = 2000 + 50,000 = 52,000

Total Cost = 4000 + 50,000 + 52,000 = 106,000 I/O

d. Index join (ignore the cost of index lookup)

- Load R into pages and iterate it by tuples.
    - Cost = 2000
- For each tuple in R, get the corresponding couple from S.
    - Cost = 10000 * 50000 / 5 = 100,000
- Join R and S

Total Cost = 2000 + 100,000 = 102,000 I/O s

- Based on the costs of these different algorithms, it is clear that both Nested-loop join and the index join are equally efficient.