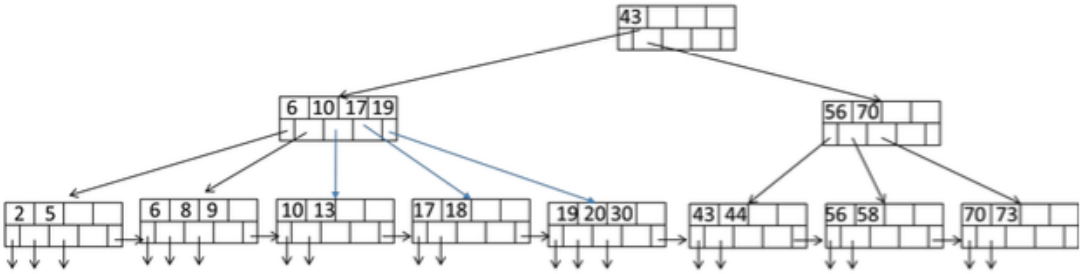
DSCI 551 Homework4
Vorapoom Thirapatarapong

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.**

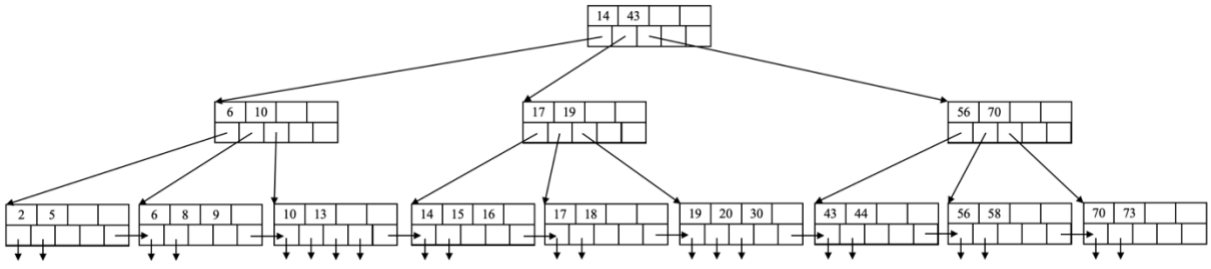


a) [10 points] Describe the process of finding keys for the query condition "age >= 35 and age <= 65". How many blocks I/O's are needed for the process?

- Read the root node, navigate to the first internal node → 1 block I/O
- Read the first internal node, navigate to the fifth leaf node → 1 block I/O
- The last data point in the fifth leaf node is still less than 35, so perform sequential traversal of leaf to sixth leaf node (under second internal node) and find the first data point 43 → 2 blocks I/O
- Sequential traversal of leaf to seventh leaf node → 1 block I/O
- Sequential traversal of leaf to eighth leaf node, the end point of query at 65 is less than the first data point of the eighth leaf node (70) so we stop → 1 block I/O
- **A total of 6 block I/O's are needed for the given query process.**

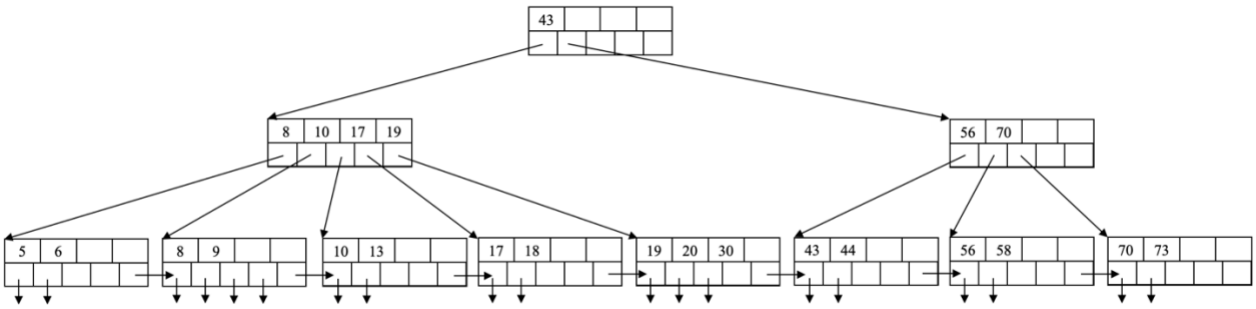
b) [15 points] Draw the B+-tree after inserting 14, 15, and 16 into the tree. Only need to show the final tree after all insertions.

- Insert 14: 14 is inserted into the third leaf of the tree, as a third element
- Insert 15: 15 is inserted into the third leaf of the tree, as a fourth element
- Insert 16: 16 is supposed to be in the third leaf but it is full after inserting the first two values, so we split the third leaf to a leaf with 10 and 13, and a leaf with 14, 15, and 16. 14 is then also inserted to its parent (first internal node). The first internal node then becomes overflow so we split the first internal node into 6 and 10, and 17 and 19; 14 which is the middle element in this internal node is inserted into root node.



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

- Deleting 2 from the first leaf node leaves the first leaf node being below the minimum capacity. Therefore, the first available option is to move 6 from its right sibling node. The first key in the first internal node (parent) also changes from 6 to 8.



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.

   - Steps of joining R as outer relation:
     For each (102-2) blocks 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 is from reading R once, reading S as outer loop B(R)/(M-2) times
   - Therefore, total cost of nested-loop join with R as outer relation = B(R) + B(R)B(S)/(M-2) = 5000 + 5000*20000/(102-2) = 1,005,000 block I/O's

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

- Steps of joining S as outer relation:
  For each (102-2) blocks b_s of S do:
  For each block b_r of R do:
  For each tuple s in b_s do:
  For each tuple r in b_r do:
  If r and s join then output(r, s)

- Cost is from reading S once, reading R as outer loop B(S)/(M-2) times
- Therefore, total cost of nested-loop join with S as outer relation = B(S) + B(S)B(R)/(M-2) = 20000 + 20000*5000/(102-2) = 1,020,000 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.

- First we sort both table R and S:
  - Sort R: number of runs = 5000/100 = 50 runs, 100 blocks per run → 2B(R)
  - Sort S: number of runs = 20000/100 = 200 runs, 100 blocks per run → 2B(S)
- Since number of runs of R and S = 250 which is more than 100, so we further merge-sort the 200 runs from S
  - Sort S: number of runs = 200/100 = 2 runs →2B(S)
- Now we have number of runs = 52, so we can merge R and S →B(R) + B(S)
- Therefore, total cost = 3B(R) + 5B(S) = 3*5000 + 5*20000 = 115,000 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).

- First we hash both R and S:
  - Hash R into 100 buckets, 50 blocks/bucket, write back to storage → 2B(R)
  - Hash S into 100 buckets, 200 blocks/bucket, write back to storage → 2B(S)
- Since min(Ri, Si) = min(50, 200) = 50 is less than M-1, so we are able to join R and S within each bucket without having to further hash/merger. → B(R) + B(S)
- Therefore, total cost = 3B(R) + 3B(S) = 3*5000 + 3*20000 = 75,000 block I/O's

Based on the given conditions, the partition-hash join algorithm is the most efficient in terms of blocks I/O when compared to nested-loop join and sort-merge join.