Homework 4

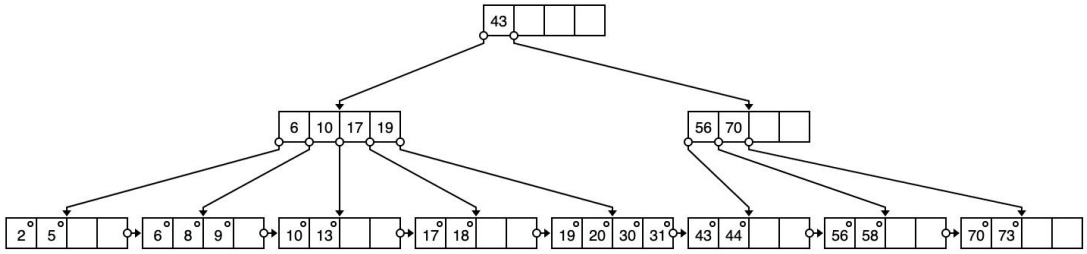
Name: Zepei Zhao USC ID: 5635405542

1a. Describe the process of finding keys for the query condition "age >= 15 and age <= 45". How many blocks I/O's are needed for the process?

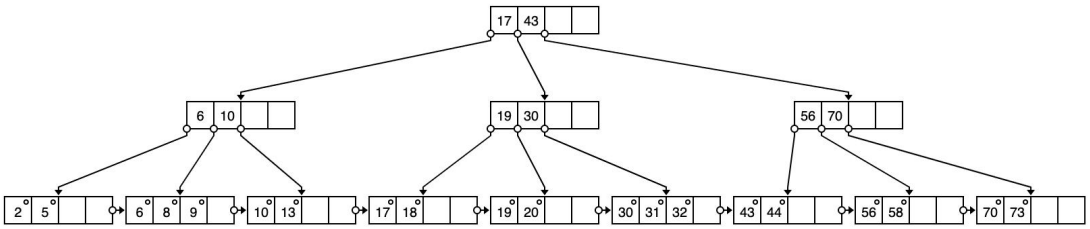
Answer: In the range of age[15,45], find the first leaf in the range, which is the left leaf with key [6,10,17,19]. Then sequential traversal of leaves until find the keys. So next, we can find the first pointer which satisfies "age>=15", pointing to keys [17,18] and linking to [19,20,30] and [43,44]. Because keys in next leaf node are beyond our range, the traversal is stopped.
Start by loading the root first in memory, follow the left pointer and load the block into memory. Then follow the forth pointer, and load this leaf into memory. So it totally costs seven blocks I/O's to complete the process.

1b. Draw the B+tree after inserting 31 and 32 into the tree.

Insert 31

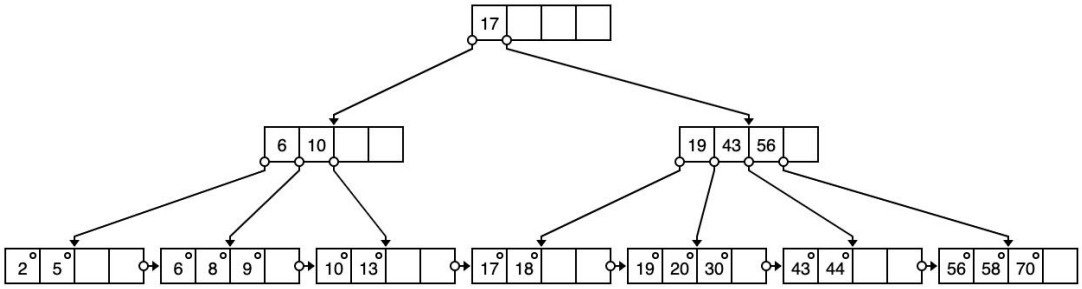


Insert 32



1c. Draw the tree after deleting 73 from the original tree.

Delete 73



2a. Nested-loop join with R as the outer relation

B(R) = 5000 blocks
B(S) = 10000 blocks
M = 102 pages
M-2 = 100 pages

for each 100 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)

- Read R once, cost B(R) = 5000 blocks
- In main memory, it can load 100 blocks at a time when outer loop is executed.
- R as the outer relation, so it will has B(R)/(M-2) = 5000/100 = 50 time iterations in outer loop. Each time, 100 blocks are loaded into main memory.
- In each iteration, make one pass through R, scanning entire S to check every block in S and join tuples to output buffer. So the cost is (B(R)/(M-2))*B(S) = 50*10000 = 500000.
- The total cost is B(R) + B(R)B(S)/(M-2) = 5000 + 500000 = 505000 block I/O's

2b. Nested-loop join with S as the outer relation

for each 100 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 s and r join then output (s,r)

- S as the outer relation, make a pass through S, cost B(S) = 10000 blocks.
- In outer loop, it will execute B(S)/(M-2) = 10000/100 = 100 times. Each time, 100 blocks are loaded into main memory.

- In inner loop, each time scanning R in input buffer to check every block in R and join tuples to output buffer. The cost is (B(S)/(M-2))*B(R) = 100*5000 = 500000 blocks.
- The total cost is B(S) + (B(S)/(M-2))*B(R) = 10000 + 500000 = 510000 block I/O's

2c. Sort-merge join (assume only 100 pages used for sorting and 101 pages for merging). When the number of runs of a relation is too large for merging, the runs will be further merged first. Select the relation with a larger number of runs for further merging if both have too many runs.

- 100 pages are used for sorting, so there are B(R)/100 = 50 runs(each run size of 100 blocks) of R and B(S)/100 = 100 runs(each run size of 100 blocks) of S. The cost (sort and write) is 2B(R)+2B(S) = 2*5000 + 2*10000 = 30000 blocks. Send them back to disk.
- M-1 = 101-1 = 100, while B(R)+B(S) = 15000 >= (M-1)M = 10000. So we cannot merge M-1 runs from R and S directly. So we need to merge S into 100/100 = 1 run.
- Load 100 runs of S and merge them into 1 run with 10000 pages, then send then to disk. Cost is 2B(S) = 2*10000 = 20000 blocks.
- Merge 50 runs of R and 1 run of S, merge them. The cost is B(R)+B(S) = 5000+10000=15000 blocks.
- Total cost is 3B(R)+5B(S) = 3*5000+5*10000 = 65000 block I/O's

2d. Simple sort-based join (using the same assumption on # of pages used for sorting and merging as above)

- Both R and S are completely sorted into a single run in two passes. So both R and S only has one run with size of 5000 blocks and 10000blocks. The cost is 4B(R)+4B(S) = 4*5000+4*10000 = 60000 blocks.
- They only need one input buffer for S, and one output buffer for the two, R and S. So there still have many buffers are available, which can load as much data as possible for the single run from R.
- Read both relations in sorted order and match tuples. The cost is B(R)+B(S) = 5000+10000 = 15000 blocks.
- Total cost is 5B(R)+5B(S) = 60000+15000 = 75000 block I/O's

2e. Partitioned-hash join (assume 101 pages used in partitioning of relations and no hash table used for lookup in joining buckets)

- Hash R into M-1 buckets (use hash function h), so there are 101-1 = 100 buckets and each bucket has B(R)/(M-1) = 5000/100 = 50 blocks. Send all buckets to disk. Input: B(R) = 5000, output: M-1 = 100 buckets with 50 blocks each. Buckets R(R1, R2...Rm-1). Cost is 2B(R) = 10000 blocks.
- Hash S into M-1 buckets (use hash function h, there are 100 buckets and each bucket has B(S)/(M-1) = 10000/100 = 100 blocks. Send all buckets to disk. Input: B(S) = 10000, output: 100 buckets with 100 blocks each. Buckets S(S1,S2,...Sm-1). Cost is 2B(S) = 20000 blocks.

- Join every pair of corresponding buckets. Partition tuples in R and S using join attributes as key for hash. Tuples in partition Ri only match tuples in partition Si. Cost is B(R) + B(S) = 5000+10000 = 15000 blocks.
- Read Si from partition of S, hash it using hash function h'.
- Load the matching partition Ri, one block at a time and join tuples to output.
- Total cost: 3B(R) + 3B(S) = 3*5000 + 3*10000 = 45000 block I/O's

2f. Index join (ignore the cost of index lookup)

- s has clustered index on the join attribute a and V(S,a) = 10. T(R) = 100000 tuples, T(S) = 200000 tuples. Load R: cost is B(R) = 5000 blocks.
- Index-based selection of S: cost is B(S)/V(S,a) = 10000/10 = 1000
- Iterate over R, for each tuple, fetch corresponding tuple(s) from S. Cost is T(R)*B(S)/V(S,a) = 100000*1000 = 100000000
- Total cost is: B(R) + T(R)B(S)/V(S,a) = 5000+100000000 = 100005000 block I/O's

To sum up, partitioned-hash join algorithm is the most efficient, cause it has the smallest cost.