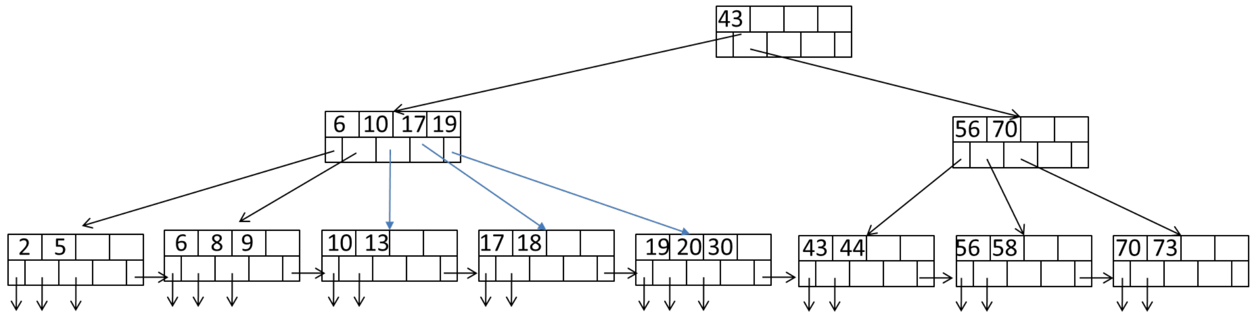
INF 551 - Spring 2020: Homework-4
Supreeth Kabbin, USC ID - 8533555714

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.



a. 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?

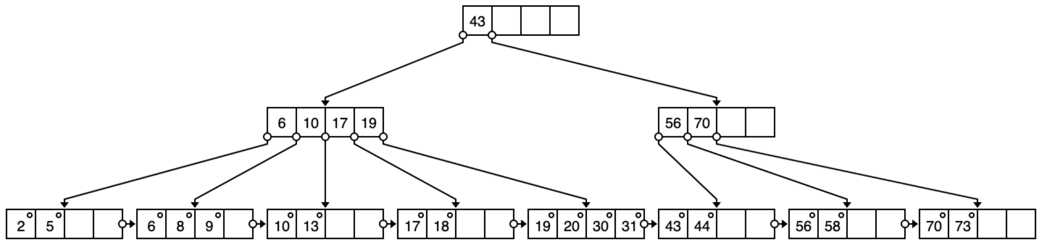
Ans:

1. Read the root node, find the first internal node.
2. Read the first internal node, find the third leaf node.
3. Read the third leaf node, to understand position of 15.
4. Perform sequential traversal of leaves 4, 5, 6 and 7 while reading them.
5. Stop traversal at leaf 7 as the upper limit is less than the first data point in 7.
   We performed 7 reads to complete this search query. Hence, 7 block I/O's are needed for the process.

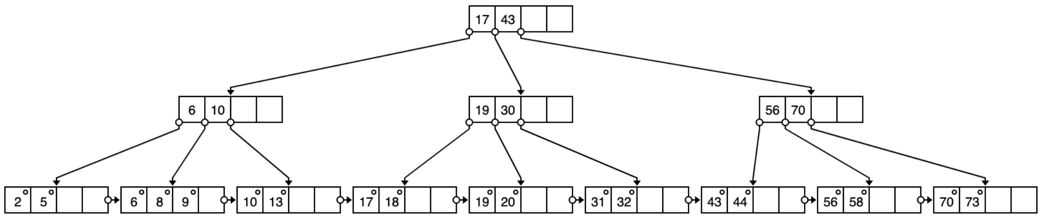
b. Draw the B+tree after inserting 31 and 32 into the tree.

Ans:

After inserting 31:

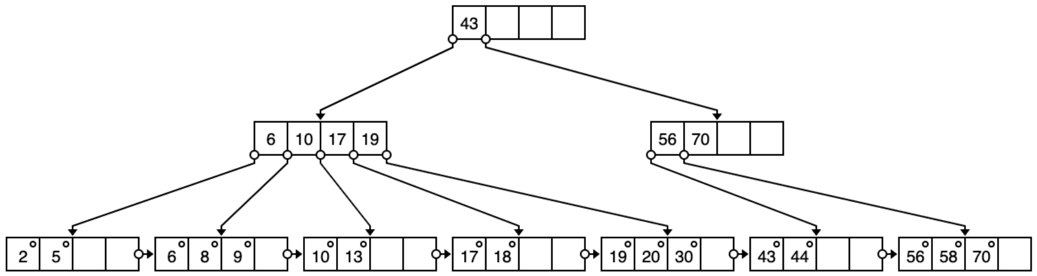


After inserting 31 and 32:



TYPE TO ENTER A CAPTION.

c. Draw the tree after deleting 73 from the original tree.
Ans:



TYPE TO ENTER A CAPTION.

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 and 100,000 tuples

ii. S is a clustered relation with 10,000 blocks and 200,000 tuples

iii. S has a clustered index on the join attribute a

iv. V(S, a) = 10 (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) 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.
Ans:

```
For each 100 blocks of $b_r$ of R do
    for each block $b_s$ of S do
        for each tuple r in br do
            for each tuple s in bs do
                if r and s join then output (r,s)
```

Cost:
- Read R: cost B(R)
- Read S for each outer loop run, B(R)/100 times: cost B(R)B(S)/100
- Total cost: B(R) + B(R)B(S)/100 = **505,000** Block I/O's.

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

```
For each 100 blocks of bs of S do
    for each block br of R do
        for each tuple s in bs do
            for each tuple r in br do
                if s and r join then output (s,r)
```

Cost:
- Read S: cost B(S)
- Read R for each outer loop run, B(S)/100 times: cost B(S)B(R)/100
- Total cost: B(S) + B(S)B(R)/100 = **510,000** Block I/O's.

c. Sort-merge join.
Ans:

1. Load 100 blocks of R at a time, perform sort and return them to disk. This results in 50 runs of size 100 each. Cost here: 2*B(R) = 10000.
2. Load 100 blocks of S at a time, perform sort and return them to disk. This results in 100 runs of size 100 each. Cost here: 2*B(S) = 20000.

Since number of runs(R) + runs(S) > 100, we cannot merge them directly. Using the ceil function, we merge them ceil(100/100) = 1 run.

3. Load 100 runs of S, perform merge and return to disk. This generates 1 run of size 10000 pages. Cost: 2*B(S) = 20000.
4. Merge 50 runs from R and 1 run from S, join all sorted runs. Cost: B(R) + B(S) = 15000.
Hence, total cost: 3*B(R) + 5*B(S) = **65000** Block I/O's.

d. Simple sort-based join.

Ans:

1. Load 100 blocks of R at a time, perform sort and return them to disk. Results in 50 runs of size 100 each. Cost: 2*B(R) = 10000.
2. Load 50 runs of R, perform merge and return them to disk. This generates 1 run of size 5000. Cost: 2*B(R) = 10000.
2. Load 100 blocks of S at a time, perform sort and return them to disk. Results in 100 runs of size 100 each. Cost: 2*B(S) = 20000.
3. Load 100 runs of S, perform merge and return to disk. This generates 1 run of size 10000 pages. Cost: 2*B(S) = 20000.
4. Join all sorted runs. Cost: B(R) + B(S) = 15000.
Hence, total cost: 5*B(R) + 3*B(S) = **55000** Block I/O's.

e. Partitioned-hash join.

Ans:

1. Hash R onto 100 buckets and return to disk. This generates 50 blocks per bucket. Cost: 2*B(R) = 10000.
2. Hash S onto 100 buckets and return to disk. This generates 100 blocks per bucket. Cost: 2*B(S) = 20000.
3. Join the corresponding buckets consisting of R and S. Cost: B(S) + B(R) = 15000.
Hence, total cost: 3*B(R) + 3*B(S) = **45000** Block I/O's.

f. Index join.

Ans:

1. Load R onto 100 pages and iterate through R on tuples. Cost: B(R) = 10000.
2. Retrieve corresponding tuples from S for each tuple in R. Cost: T(R)*B(S)/V(S,a) = 100,000,000.
3. Join R and S.
Given that R is a clustered relation, total cost comes to: B(R) + T(R)*B(S)/V(S,a) = **100,005,000** Block I/O's.

Based on the results achieved, we see that **Partitioned-hash algorithm** is the most efficient.