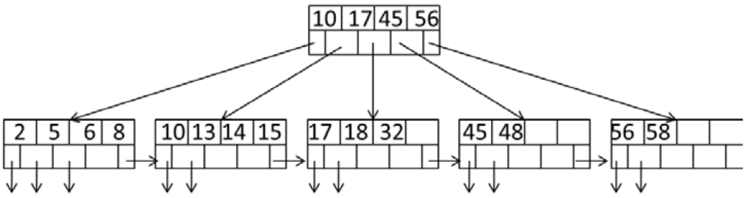
Due: November 10, Friday
100 points

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.

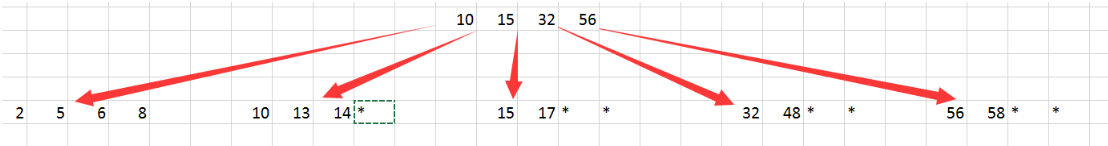


   a. Describe the process of finding keys for the query condition "age >= 10 and age != 15". How many blocks I/O's are needed for the process?

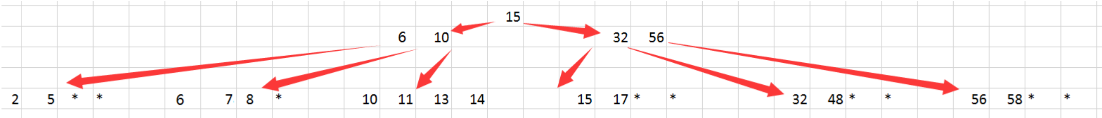
Starting from the Root block, try to find the leaf with value 10 or the smallest value available that larger than 10 (if not 15), use that as the starting point. Then, traverse the leaf from starting point to right most value (until no data left), check if the value is 15, if not, add to result.

Thus, we have to read **5 Blocks**.

   b. Draw the updated B+tree after first deleting 45 and then deleting 18 from the tree.



   c. Draw the updated tree after first inserting 7 and then inserting 11 into the tree obtained in part b.



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 500 blocks and 5,000 tuples
   ii. S is a clustered relation with 10,000 blocks and 100,000 tuples
   iii. S has a clustered index on the join attribute a
   iv. V(S, a) = 200 (recall that V(S, a) is the number of distinct values of a in S)
   v. 100 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 hence the cost of writing the output is ignored.

Describe the steps (including input, output at each step, and their sizes) in each of the following join algorithms. What is the total number of block I/O's needed for each algorithm? Which algorithm is the most efficient?

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

Steps:

for each (M-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)

Sizes:

For each step,
input = tuple s and tuple r,
output = (r, s), 1 tuple.

Cost:

– Read R once: cost B(R)
– Outer loop runs ceil(B(R)/(M-2)) =6 times, and each time need to read S: costs B(S)*6
– Total cost: B(R) + B(R)*B(S)/(M-2) = 500 + 6*10000 = **60500**

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

Steps:

for each (M-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 s and r join then output(s,r)

Sizes:

For each step,
input = tuple s and tuple r,
output = (r, s), 1 tuple.

Cost:

– Read R once: cost B(S)
– Outer loop runs ceil(B(S)/(M-2)) =103 times, and each time need to read S: costs B(R)*100
– Total cost: B(S) + B(R)*B(S)/(M-2) = 10000+ 103*500 = **61500**

c. Sort-merge join

Step 1: Perform a pass 0 on Relation R. This generates ___5___ runs each of size _____ blocks.
Cost of this step: _2B(R)____

Step 2: Perform a pass 0 on Relation S. This generates ___100___ runs each of size _____ blocks.
Cost of this step: _2B(S)____

Observe: Number of runs from Step 1 + Step 2 > Memory Pages (M – 1)

Step 3: Load ___7___ runs from S, sort them and write back to disk. (Try to minimize this number instead of blindly writing 99).
Cost of this step: ___2M*7____

Observe that # of runs of R + # of runs of S = (M - 1)

Step 4: Join the runs of R and S
Cost of this step: _B(R) + B(S)_____

Grand total cost of all the steps: _3*(B(R) + B(S)) + 14M = 3*(10500)+ 1400= **32900**

d. Simple sort-based join

Step 1: Perform a pass 0 on Relation R. This generates ___5___ runs each of size _____ blocks.
Cost of this step: _2B(R)____

Step 2: Perform a pass 0 on Relation S. This generates ___100___ runs each of size _____ blocks.
Cost of this step: _2B(S)____

Sort R runs.
Cost: 2B(R)

Sort S runs.
Since we do not have enough memory, we will merge two runs into one.
Cost: 2B(S) + 2M*2

Step 4: Merge the runs of R and S
Cost of this step: _B(R) + B(S)_____

Grand total cost of all the steps: _5*(B(R) + B(S)) + 4M = 5*(10500) + 400= **52900**

e. Partitioned-hash join

Steps:

Hash S into M – 1 buckets – send all buckets to disk

  input: S,

  output: M-1 Buckets

Hash R into M – 1 buckets – Send all buckets to disk

  input: R,

  output: M-1 Buckets

Join every pair of corresponding buckets

  input: Block in partition of Si & block in matching partition Ri

  output: joined result

Cost: 3B(R) + 3B(S) = 3*(10000+ 500) = 31500

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

Steps:

Read R one time,

Iterate over R, for each tuple, fetch corresponding tuple(s) from S

Cost:

Since R is clustered, and index is clustered,

Cost = B(R) + T(R)B(S)/V(S,a)= 500 + 5000*10000/200 = **250500**

Efficiency:

Overall, compare with all the algorithms, the one has least cost is **Partitioned-hash join**.