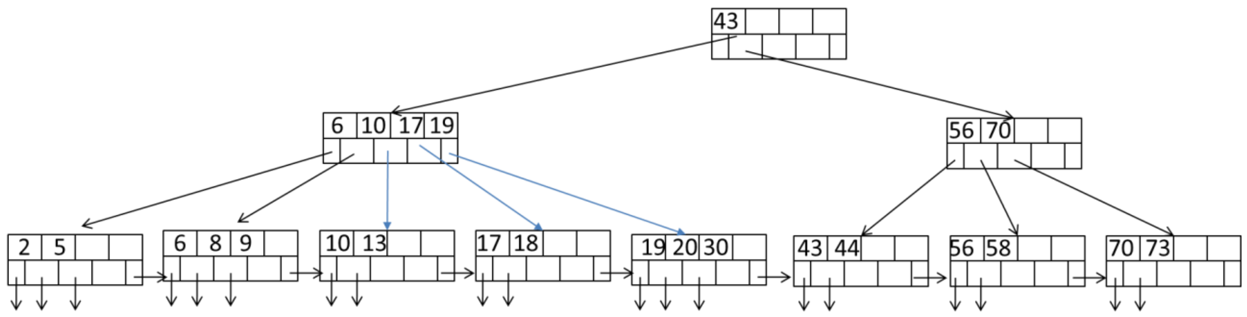
DSCI 551 – HW4 (Indexing and Query Execution)

Zhenmin Hua

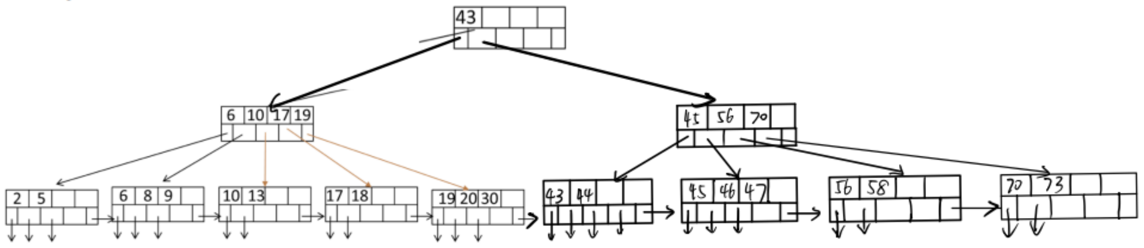
1. [40 points] Consider the following B+tree for the search key "age. Suppose the degree d of the tree = 2, that is, each node (except for root) must have at least two keys and at most 4 keys. **Note that sibling nodes are nodes with the same parent.**



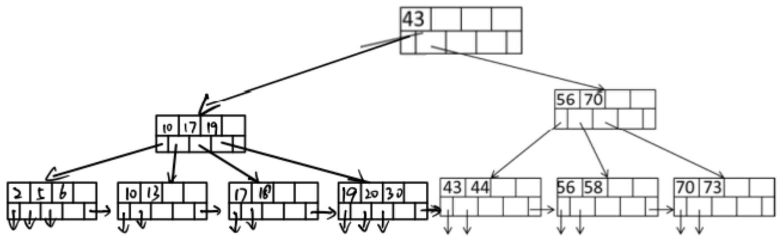
   a. [10 points] Describe the process of finding keys for the query condition "age > 20 and age <= 50". How many blocks I/O's are needed for the process?
   b. [15 points] Draw the B+-tree after inserting 45, 46, 47 into the tree. Only need to show the final tree after the insertions.
   c. [15 points] Draw the tree after deleting 8 and 9 from the original tree.

a. Start at the root, 20<50, proceed down to the left leaf [6,10,17,19], 20>19, then turn to the right most leaf [19,20,30], age >20, then start at 30, age<=50, so sequential traversal until 44. So at last we find 30, 43, 44. And, 3 blocks I/O's are needed.

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 5,000 blocks.
    ii. S is a clustered relation with 10,000 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 (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. [10 points] (Block-based) nested-loop join with R as the outer relation.
    b. [10 points] (Block-based) nested-loop join with S as the outer relation.
    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.
    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).

a.

chunk size: 102-2=100 blocks

Read R once: cost B(R); Outer loop runs B(R)/(M-2) times, and each time need to read S: costs B(R)B(S)/(M-2) ;

Total cost: B(R) + B(R)B(S)/(M-2), in this case: 5000 blocks+5000/(102-2)*10000=505,000

b.

chunk size: 102-2=100 blocks

Read S once: cost B(S); Outer loop runs B(S)/(M-2) times, and each time need to read R: costs B(S)B(R)/(M-2) ;

Total cost: B(S) + B(S)B(R)/(M-2), in this case: 5 10,000 blocks+10,000/(102-2)*5000=510,000

c.

Each step: Pass 1: sort R => 50 runs, 100 blocks/run Sort S => 100 runs, 100 blocks/run Pass 2(merge): B(R)+B(S)

Total number of I/O's needed: 3B(R)+3B(S), in this case: 3*(10000+5000) = 45000 blocks

need to sorting first, faster than the nested loop

d.

Each step: Step 1. Partition S into (M-1) buckets, i.e. hash S into 100 buckets, send all buckets to disk; Step 2. Partition R into (M-1) buckets, i.e. hash R into 100 buckets, send all buckets to disk; Step 3. Join every pair of corresponding buckets

Total number of I/O's needed: 3B(R)+3B(S)=45000 blocks

So, partitioned hash join is relatively most efficient. Sort merge join needs to sort two tables first, so its efficiency is the worst.