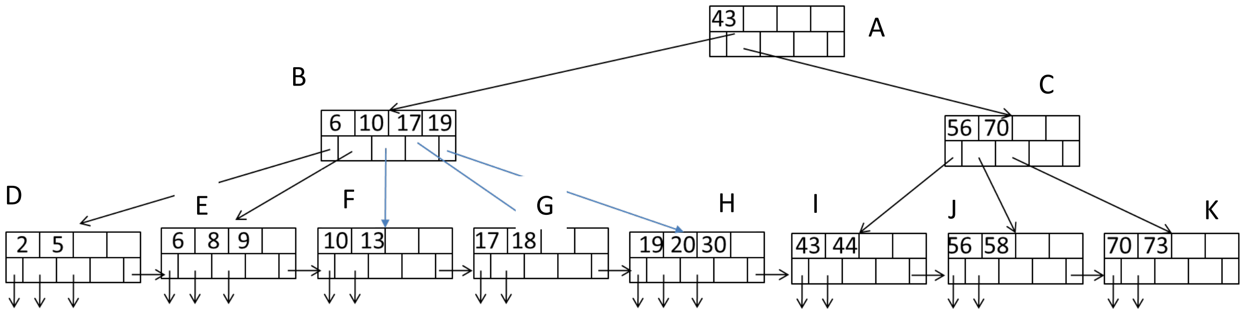
DSCI 551 – HW4

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 >= 10 and age <= 50”. How many blocks I/O’s are needed for the process?
   b. [15 points] Draw the B+-tree after inserting 31 and 32 into the tree. Only need to show the final tree after the insertions.
   c. [15 points] Draw the tree after deleting 18 from the original tree.

Answer:

a.

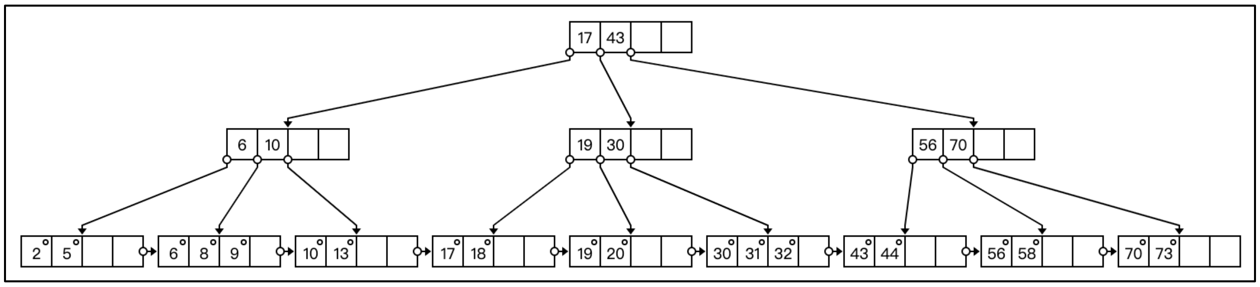
Name the block at level1 as A, two blocks at level2 from left to right as B and C, and blocks at level3 as D, E, F, G, H, I, J, K, from left to right respectively.

The basic principle of finding keys is finding the starting leaf with the minimum value of the range and then traveling along the pointers to find the specific keys we want until to the maximum value of the range

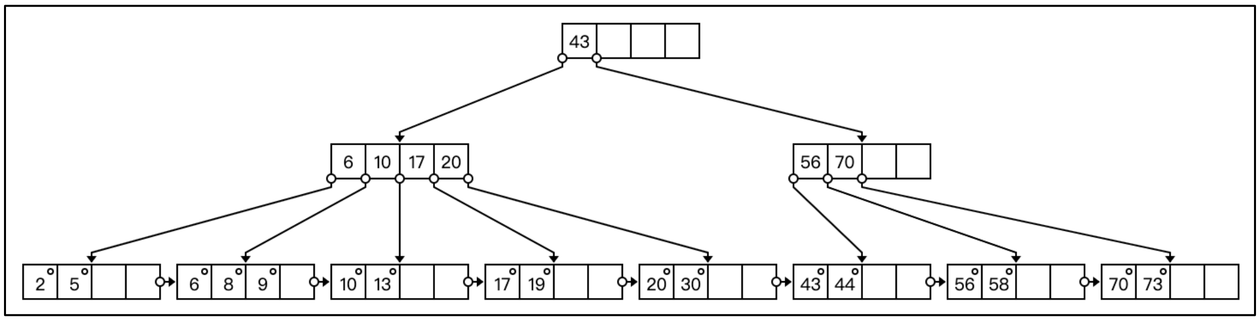
So we start from the block A, then turn left down to block B since 10, the minimum value of the range “age >= 10 and age <= 50”, is less than 43. And go down to the block F, and then search along the sequential traversal until meet value which is larger than 50, meaning ending at block J.

So 7 blocks needed for the process and they are block A, B, F, G, H, I and block J.

b. Inserting 31 and 32



c. **Deleting 18**



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 1000 blocks.
   ii. S is a clustered relation with 500 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, 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).

Answer:

a:

According to the given information, we use one block as input buffer and one block as output buffer, so there are 100 blocks can be used for nested-loop join. So when R is the outer one:

- Step1: we should load 100 blocks of R at once .
- Step2: and then load one pass through S, which means load one block for 500 times. By doing so we will gain joined blocks which are satisfied the join condition.
- Step3: repeat operations above 10 times (1000/(102-2)=10).
- Therefore, the total cost (total number of block I/O's needed) of nested-loop join with R as the outer relation is 6000 blocks (1000+500*(1000/102-2) = 6000).

b:

According to the given information, we use one block as input buffer and one block as output buffer, so there are 100 blocks can be used for nested-loop join. So when S is the outer one:

- Step1: we should load 100 blocks of S at once.
- Step2: and then load one pass through R, which means load one block for 1000 times. By doing so we will gain joined blocks which are satisfied the join condition.
- Step3: repeat this operation 5 times (500/(102-2)=5).
- Therefore, the total cost of nested-loop join with S as the outer relation is 5500 blocks (500+1000*(500/102-2) = 5500).

c:

According to the given information, there are 100 pages can be used for sorting and 101 pages can be used for merging. So :

- Step1: sort R into 10 runs and then sort S into 5 runs, each run has 100 blocks.
  And the number of block I/O's needed is 3000 (2*(1000+500)=3000)
- Step2: merge 15 runs into one relation. To do this we need to input first block in each run and then compare and merge the satisfied pair of blocks into output buffer.
  And the cost is 1500 blocks (1000+500=1500) since the cost of writing the output is ignored.
- So the total number of block I/O's needed 4500.

d:

According to the given information, there are 101 pages used in partitioning of relations. So :

- Step1: read blocks of R and hash them into 100 buckets ($R_1, R_2, \dots, R_{100}$) and each bucket has 10 blocks. Then read blocks of S and then hash them into 100 buckets ($S_1, S_2, \dots, S_{100}$) and each bucket has 5 blocks.
  And the cost is 3000 (2*(1000+500) = 3000) blocks. Since we do read and write for each relation.
- Step2: join R_i with the corresponding bucket S_i.
  And the cost is 1500 blocks.
- So the total number of block I/O's needed 4500.

So according to discussion above, the most efficient algorithm is sort-merge join (c) and partitioned-hash join.