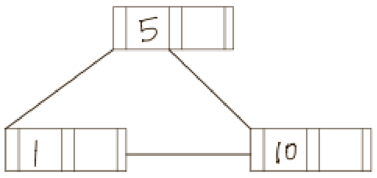
CS 540 Midterm 2
Ga Young Lee (Student ID: 933-961-022)

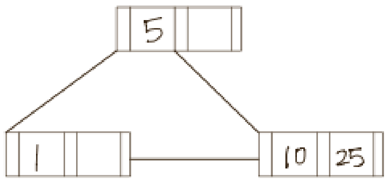
Q1. B+ Tree Index

1. Original Tree:

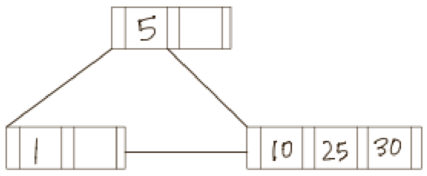


To insert “25,” find the appropriate location of the key, which is the node where “10” is located in and check if it complies with the maximum number of keys allowed in a node. The resulting node after inserting “10” contains two keys, “10” and “25”, so we don’t have to change the structure of the tree.

2. After inserting “25”



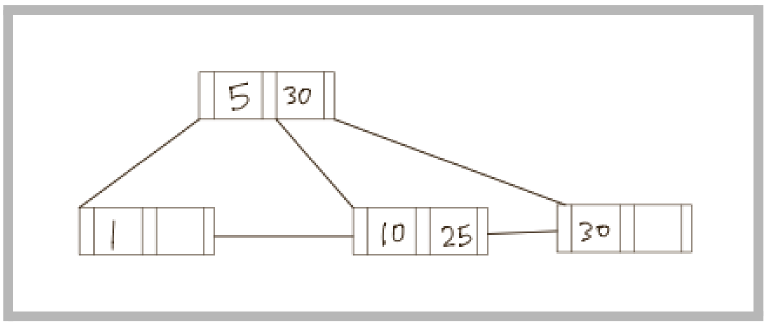
3. While inserting “30”



However, when we’re inserting “30” to the leaf node, it exceeds the number of allowed keys in a node, 2, so we need to split the node with the three keys. Therefore, we create a new leaf node.

Since we now have three leaf nodes, we need an additional pointer coming out of its parent node, so we send the smallest key value of the newly created node, “30”, to its parent node. This results in the final B+ tree as shown below.

4. After inserting “30” → Final B+ Tree



Q2. Index Design

In the given query, there are two inequalities in the ‘WHERE’ clause, so it’s beneficial to use the “**clustered B+ tree index**” on the “**salary**” attribute to improve the running time. The main reason for choosing this type of index is that clustered B+ tree index supports range queries efficiently. In other words, when retrieving data using a range query on clustered B+ tree index, we can find the lowest or highest point in the range and follow the links between the nodes to get the data that falls into a specific range. Considering this property, using the “clustered B+ tree index” is recommended in this case.

Q3. Query Processing: Block-based Nested Loop Join

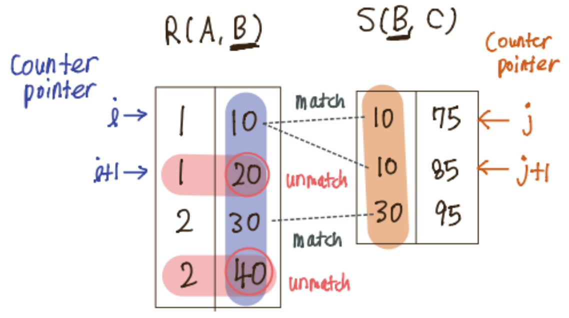
$$B(Student) = 40,000$$
$$B(Enroll) = 400$$
$$M = 4$$

Given this condition, it is impossible to use the internal memory join since we can’t fit both relations into the main memory, so we cross out the internal memory join algorithms. For a similar reason, we eliminate the sort-merge join with two-pass multi-way merge-sort algorithm because the requirement for this algorithm, B(Student) <= M^2 and B(Enroll) <= M^2, is not satisfied.

Therefore, the only available option is the “**Block-based Improved Nested Loop**” algorithm that requires M blocks, 4 blocks in this case (2 blocks for reading relations, 1 block for join, and 1 block for output), which takes a quadratic number of I/O accesses.
Thus, the cost of this join algorithm is $B(Enroll) + [B(Enroll) * \frac{B(Student)}{(M-2)}] = 8,000,400$ which is approximated to $\frac{B(Student)*B(Enroll)}{M} = \frac{40,000*400}{4} = 4,000,000.$

Q4. Query Processing: Left Anti-join

First, we need to sort R and S using the Two-pass Multi-way Merge Sort for both relations, which takes the cost of 3B(R) and 3B(S). Then, we can apply a modified sort-merge join algorithm on the sorted relations, R and S. The modification is described below:



match if R.B = S.B
skip to the next pointer (tuple)

unmatch if R.B ≠ S.B
return the tuple in R(A, B)

Let’s say the join attribute of relation R and S is B. We assign counter pointers, ‘i’ to R.B and ‘j’ to S.B. Then, since both relations are sorted, we scan R.B and S.B from the beginning to search for a match. Once we scan the relations and find a match where R.B = S.B, we skip to the next tuple by moving the pointers. On the other hand, when there’s a unmatch where R.B != S.B, we return the corresponding tuple in R(A, B). To handle duplicates, we look ahead by keeping track of ‘i+1’ and ‘j+1’ to check the end of duplicate tuples.