1 File Organizations

If you were about to create an index on a relation, what considerations would guide your choice? Discuss:

1. Direct vs. indirect index.
2. Clustered vs. unclustered indexes.
3. Hash vs. tree indexes. In particular, discuss how equality and range searches work.
4. The use of a sorted file rather than a tree-based index.

2 B+ tree Operations

Consider the B+ tree index of order d= 2 shown in Figure 1.

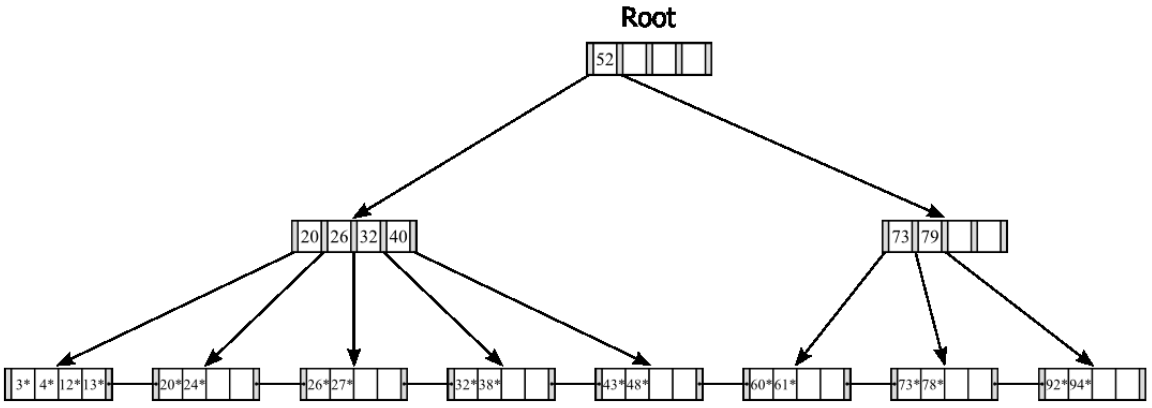

Figure 1: Example of a B+ tree.

1. Show the tree that would result from inserting a data entry with key 23 into this tree.
2. Show the B+ tree that would result from inserting a data entry with key 6 into the original tree. How many page reads and page writes will the insertion require?
3. Show the B+ tree that would result from deleting the data entry with key 20 from the original tree, assuming that the left sibling is checked for possible redistribution.
4. Show the B+ tree that would result from deleting the data entry with key 20 from the original tree, assuming that the right sibling is checked for possible redistribution.
5. Show the B+ tree that would result from starting with the original tree, inserting a data entry with key 50 and then deleting the data entry with key 60.
6. Show the B+ tree that would result from deleting the data entry with key 92 from the original tree.
7. Show the B+ tree that would result from starting with the original tree, inserting a data entry with key 63, and then deleting the data entry with key 92.
8. Show the B+ tree that would result from successively deleting the data entries with keys 32, 38, 43, 48, and 73 from the original tree.

3 Shared scans vs. B+ trees

Concurrent queries with common sequences of disk accesses can make maximal use of a buffer pool by using *shared scans*. The idea of shared scans is the following: given a batch of k queries, it sequentially scans the table with the data records and, for each record, it evaluates all k queries and stores the intermediate results. If the end of the table is reached the scan resumes from the beginning of the table (one can see it as a "cyclic scan"). If the table is stored on disk, shared scans improves I/O since only sequential accesses are performed.

Given a table with 1 Mio. records sequentially stored on disk, each record of the size of 100B. Assume the hard disk has a sequential read performance of 100MB/sec. k queries are evaluated at a time.

1. How long does it take until all queries are processed? Do not consider CPU time for now only consider I/O cost.
2. An alternative to shared scans is to use a B+ tree. Assume that a random access on disk can be done in 5ms and all intermediate index nodes of the tree are available from the DB-buffer. How long does it take to process k queries on the B+-tree?
3. Which solution is better in terms of I/O-time? How would you choose k for shared scans?
4. Now, in addition to the I/O cost, let's also consider CPU cost. Assume that it takes 100 nanoseconds for the CPU to process a record for each query during the scan. Similarly, assume that it takes 20 nanoseconds to perform a key comparison in main memory inside any tree node. For which values of k will the shared scan be better than the B+-tree?
5. What other gain can you imagine by using a scan instead of a tree-structure?

4 Project: Main-Memory Efficient Tree-index

This task must be performed in small teams from 2-4 people. Please make sure to register your group ASAP. This task is part of the project as defined in the first week of this lecture.

Task: Develop a cache-efficient main-memory resident tree-index. The index should be able to perform point queries, range queries, inserts, updates, and deletes. Assume a key range 0,..., 10 million. Values are any 4 byte integer. Please do not use any multi-threading for the moment. Persisting changed entries is not necessary.

1. Take the B+-tree framework available from the lecture Web-site as a starting point. Alternatively, you may develop your index from scratch (whatever suits you best).
2. Implement at least one of the techniques presented in the lecture including CSB+-trees, pB+-trees, fpB+-trees, or CSS-trees.
3. Ensure functional correctness of your index by comparing the result set produced by your index against a reference set.
4. Execute the workload available from the lecture web-site. Try to optimize performance (wall clock time) of your method for that workload.

Keep in mind that the goal of this exercise is to further use your index for the final task as it will be defined by the ACM programming contest.

Your code has to run on a linux server. Only Java, C, C++ may be used as programming languages. Your code should be handed-in on Thursday, Nov 27, midnight latest. This will allow us to run all code

submissions on the same server. Note that the benchmark used by us for this evaluation will slightly differ from the published benchmark. However, the characteristics will be the same.