

CMPS 340 File Processing
Estimating Running Times for Operations on a B⁺-Tree: A Sample Problem

Suppose that you are planning to create a file of approximately four million (4,000,000) records, each 300 bytes in length. The file's *key* field is also its *ordering* field. You intend to provide indexed sequential access to the file by organizing it as a B⁺-tree. That is, the buckets holding the file's records will be the leaves of a B⁺-tree; "above" will be a B-tree that provides a *primary* index on the key/ordering field.

The disk drive at your computer installation has the following characteristics:

sector size = 3000 bytes s = 15 ms (seek time)
stt = 1 ms (sector transfer time) r = 5 ms (rotational delay)

1. Assuming space utilization of 80% (which, according to empirical evidence, is a reasonable estimate for a B⁺-tree where redistribution is used whenever possible) and leaf nodes of four sectors in length, estimate the number of leaves you would expect to find in the B⁺-tree. (That is, estimate the number of leaves/buckets holding the four million records in the file.) Use this estimate as a basis for deriving an estimate of the time t_x needed to read all the records in the file, in order with respect to the key field. (Assume that each leaf includes a pointer to its logical successor, but that there is no particular relationship between the location (on the disk) of a leaf and the location of its logical successor.)

Solution: Sectors are 3000 bytes and leaf nodes are four (4) sectors; thus, each leaf node is 12,000 bytes in length. As there are 300 bytes per record, up to 40 records fit into a leaf:¹

$$(12,000 \text{ bytes/leaf}) / (300 \text{ bytes/record}) = 40 \text{ records/leaf}$$

Under the assumption that nodes are, on average, full to 80% of their capacity, each leaf would hold, on average, $40 \cdot 0.8$, or 32, records. We now calculate the number of leaves:

$$4,000,000 \text{ records} / 32 \text{ records/leaf} = 125,000 \text{ leaves}$$

To read all the leaves in the file would require that, for each leaf, we seek to it and read it in. (Each leaf points to the next, so no extra disk accesses are required for determining the address of any of the leaves.) As each leaf is 4 sectors in length, Ltt (leaf transfer time) is taken to be $4 \cdot stt$, or 4 ms. We get

$$\begin{aligned} t_x &= 125,000(s + r + Ltt) \\ &= 125,000(20ms + 4ms) \\ &= 125,000 \cdot 24ms \end{aligned}$$

¹To be more accurate, we should allow for the fact that a little space is needed in each leaf to record how many records it currently holds and/or the records' starting locations, plus the pointer to the next leaf. Thus, there is probably room for only 39 records in a leaf.

$$\begin{aligned}
&= 3,000,000ms \\
&= 3000sec \\
&= 50min
\end{aligned}$$

2. Assuming that interior nodes are each one sector in length and that a key requires 10 bytes and a pointer 6 bytes, calculate the greatest possible order m you could choose for the B-tree that forms the index. Taking the m you just computed to be the order of the tree and assuming, again, 80% space utilization (so that, on the average, an interior node has $.8m$ children), estimate of the number of nodes that occur on the parent-of-leaf, grandparent-of-leaf, etc., levels of the B⁺-tree. (This depends upon your estimate, from (1), of the number of leaves.)

Solution: In a B-tree of order m , each node must be large enough to hold $m - 1$ keys and m pointers to children. Here, this means that m must be small enough so that $10(m - 1)$ bytes for keys and $6m$ bytes for pointers do not exceed the space in a 3000-byte node:

$$\begin{aligned}
10(m - 1) + 6m &\leq 3000 \\
16m - 10 &\leq 3000 \\
16m &\leq 3010 \\
m &\leq 188.125
\end{aligned}$$

We see that the largest possible order for the B-tree is 188. Under the assumption that, on average, nodes have 80% of the maximum number of children, they have, on average, $188 \cdot 80\% \approx 150$ children.

Thus, we would expect the parent-of-leaf level of the B⁺-tree to contain about

$$125,000 \text{ children} / 150 \text{ children/node} \approx 833 \text{ nodes}$$

Similarly, the grandparent-of-leaf level would contain about $833/150$, or 6 or 7, nodes. Finally, the root node would be the greatgrandparent of the leaves.

3. Assuming that we had one megabyte (approximately 1,000,000 bytes) of primary memory in which to store *interior nodes* (i.e., non-leaf nodes) of the B⁺-tree while the file was in use, tell how many interior nodes you could store in primary memory and, more precisely, how many nodes from each level you would store there. Based upon your answer, estimate (to the nearest tenth) the number of disk accesses required, on the average, to fetch a record from the file. (For example, if there is enough RAM to hold all the interior nodes of the B⁺-tree except for 60% of those on the parent-of-leaf level, the average number of disk accesses to fetch a record would be 1.6: one access to a leaf and 0.6 accesses (on average) to nodes on the parent-of-leaf level.) Use this as a basis to estimate the average running time t_f for a single-record fetch in your file.

Solution: A megabyte of RAM is large enough to hold about 333 interior nodes of the B⁺-tree (10^6 bytes / 3000 bytes/node). Which 333 nodes ought to be stored in RAM? Clearly, the ones that are accessed most often, which are those closest to the root. Thus, we would store the

root and its 6 or 7 children in RAM, and use the space left over for storing as many of the nodes on the parent-of-leaf level as possible. Thus, about 325 nodes out of the 833 nodes on the parent-of-leaf level (or about 40% of them) are stored in RAM. Assuming that each node on the parent-of-leaf level is as likely as any other to be retrieved in performing a search, this means that about 40% of searches require no disk accesses to fetch interior nodes and about 60% require only one. That is, on the average, a search requires 0.6 accesses to the disk to retrieve interior nodes and one access to the disk to retrieve a leaf node. Letting Itt denote “interior node transfer time” (which is equal to stt , because interior nodes are one sector in length), we get

$$\begin{aligned}
 t_f &= 0.6(s + r + Itt) + 1(s + r + Ltt) \\
 &= 0.6(20ms + 1ms) + (20ms + 4ms) \\
 &= 0.6(21ms) + (24ms) \\
 &= 15.6ms + 24ms \\
 &= 39.6ms
 \end{aligned}$$