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 ten million (10^7) records, each 400 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" the leaves will be a B-tree that provides a *primary* index on the key/ordering field.

The disk drive/operating system platform at your computer installation has the following characteristics:

```
block size = 2048 bytes                 s = 5 ms (seek time)
bltt = 0.1 ms  (block 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 blocks 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 10^7 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 the cost incurred in determining the locations (on the disk) of the leaves, in proper order, is insignificant. One possibility is for each leaf to include a pointer to its logical successor. Another is to do an inorder traversal of the B-tree. The "bottom" level of B-tree nodes contain pointers to all the leaves, of course.)

Solution: Blocks are 2048 bytes and leaf nodes are four (4) blocks; thus, each leaf node is 8192 bytes in length. As there are 400 bytes per record, up to 20 records fit into a leaf, with 192 bytes left over for metadata (e.g., a pointer to the next leaf):

$$\lfloor (8192 \text{ bytes/leaf})/(400 \text{ bytes/record}) \rfloor = 20 \text{ records/leaf}$$

Under the assumption that nodes are, on average, full to 80% of their capacity, each leaf would hold, on average, $20 \cdot 0.8$, or 16, records. We now calculate the number of leaves:

$$10,000,000 \text{ records } / \text{ 16 records/leaf} = 625,000 \text{ leaves}$$

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 four blocks in length, *Ltt* (leaf transfer time) is taken to be $4 \cdot bltt$, or 0.4 ms. We get

$$\begin{aligned} t_x &= 625,000(s + r + Ltt) \\ &= 625,000(10ms + 0.4ms) \\ &= 625,000 \cdot 10.4ms \\ &= 6,500,000ms \end{aligned}$$

$$
\begin{aligned}
&= 6500sec \\
&\approx 1.8\text{hours}
\end{aligned}
$$

2. Assuming that interior nodes are each one block 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 2048-byte node:

$$
\begin{aligned}
10(m-1)+6m &\leq 2048 \\
16m-10 &\leq 2048 \\
16m &\leq 2058 \\
m &\leq 128.6
\end{aligned}
$$

We see that the largest possible order for the B-tree is 128. Under the assumption that, on average, nodes have 80% of the maximum number of children, they have, on average, $128 \cdot 80\% \approx 102$ children.

For the sake of making our calculations a little easier, let's be pessimistic and assume that, on average, our B-tree nodes will have 100 children. Thus, we would expect the parent-of-leaf level of the B^+-tree to contain about

$$625,000 \text{ children / } 100 \text{ children/node} \approx 6250 \text{ nodes}$$

Similarly, the grandparent-of-leaf level would contain about 6250/100, or about 63, nodes. Finally, the root node would be the greatgrandparent of the leaves.

3. Assuming that we had two megabytes (2^30 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: Two megabytes of RAM is large enough to hold about 1024 interior nodes of the B^+-tree (2^31 bytes / 2^11 bytes/node). Which 1024 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 63 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 960 $(1024 - 64)$ nodes out of the 6250 nodes on the parent-of-leaf level (or about 15% 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 15% of searches require no disk accesses to fetch interior nodes and about 85% require only one. That is, on the average, a search requires 0.85 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 *bltt*, because interior nodes are one block in length), we get

$$\begin{aligned} t_f &\approx 0.85(s + r + Itt) + 1(s + r + Ltt) \\ &\approx 0.85(10ms + 0.1ms) + (10ms + 0.4ms) \\ &\approx 0.85(10.1ms) + (10.4ms) \\ &\approx 8.6ms + 10.4ms \\ &\approx 19ms \end{aligned}$$