Exercise 5.2

Saturday, 23 September 2023 22:17

1. The replacement policy would work as follows: for each block, store whether or not it was accessed for an entry in one of the last two rows. Then:
   a. If there is a block which will never be used anymore, evict that block.
   b. If there is no such block, evict a block which was accessed for an entry in the last two rows.

   Effectively, this will lead to the first m-2 rows being handled optimally (without duplicate fetching), and the last two rows being fetched continuously. The first m-2 rows will take $O(n/B)$ I/os, whereas the last two rows will take $m = \sqrt{n}$ I/Os. Together, this gives $O\left(\frac{n}{B} + \sqrt{n}\right)$ I/Os.
2. For simplicity of notation, let $\frac{M}{B} = z$. We note that $\frac{M}{B}$ denotes the number of blocks which fit into memory. Because we have that $m > 2z$, we have that no single row can fit into memory at a given time. *(This effectively allows us to ignore any cases where a block wraps around to the next row; by the time the algorithm works on the last column, it cannot have that blocks from the first column (which may overlap with those from the last column) are still in memory. In particular: we can state that, for each column, all entries in that column are placed in separate blocks.)* Now, we have that the number of blocks which have to be read to iterate over a single column is given by $m > 2z$. Since only z blocks fit into memory, this means that, for each column over which the algorithm iterates, at least $m - z$ blocks need to be read from memory. Additionally, to make place for these $m - z$ blocks, at least $m - z$ blocks need to be evicted from memory. Hence, we have that any replacement policy must, for each column over which the algorithm iterates, perform at least $2(m - z) = 2m - 2z$ I/Os. Since we have that $m > 2z$, it follows that $2m - 2z > m$, which implies the number of I/Os for each column is at least m. Now, since the algorithm iterates over m columns, we have that the total number of I/Os must be at least $m \cdot m = m^2 = n$; in other words, the number of I/Os will be $\Omega(n)$, for any replacement policy.
3. Probably not. 1024 bytes equals 1 kilobyte, and having 100.000 rows with such size would give rise to 100.000 kilobyte = 100 megabyte. This will clearly fit in the internal memory of most modern computers. However, it should be noted that this does not fit in the CPU cache (and, therefore, the problem of I/O being slow is still relevant).
4. The running times of the algorithms would basically be the same (at least asymptotically), although there would be some small differences. In particular, it should be noted that the column-by-column algorithm would only need to fetch a block twice is when that block 'wraps around' from the end of one row to the next. The number of such occurrences is at most $m - 1$ (since the last row cannot have a block which wraps around). Hence, the number of blocks which need to be read increases from $\frac{n}{B}$ (in the row-by-row algorithm) to $\frac{n}{B} + O(m)$ (in the column-by-column algorithm). With $m = \frac{M}{2B}$, this gives us $O\left(\frac{n}{B}\right) + O\left(\frac{M}{2B}\right)$ I/Os for the column-by-column algorithm. Using that $n = m^2 = \frac{M^2}{4B^2}$, this gives us a total number of I/Os which is given as $O\left(\frac{n}{B} + \frac{M}{2B}\right) = O\left(\frac{M^2}{4B^3} + \frac{M}{2B}\right) = O\left(\frac{M}{B}\left(\frac{M}{B^2} + 1\right)\right)$. Using the tall-cache assumption ($M \geq B^2$), we then find that this is equal to $O\left(\frac{M}{B} \cdot \frac{M}{B^2}\right) = O\left(\frac{M^2}{B^3}\right)$, which is equal to the bound one would obtain for the row-by-row algorithm. *(However, the running time would most likely be larger for the column-by-column algorithm; just not by a term/factor which changes the asymptotic bounds.)*
   a. ~~Note: still need to check whether the right blocks are being evicted (i.e. we might need space for two blocks per row).~~
   b. ~~Note: check whether there are issues if blocks wrap around to the next row. (Then, some blocks may need to be fetched twice.)~~

do not forget about the cache, which is much smaller!