Exercise 5.10

maandag 16 oktober 2023 15:16

(i) Each iteration of the outermost loop needs $O\left(\frac{n}{B}\right)$ I/Os to load each of the $\frac{n}{B}$ block that make up the array Y. Thus, the total number of I/Os performed by the algorithm is $O\left(\frac{n^2}{B}\right)$, as the outer loop runs for n iterations.

Alternatively:

X is scanned once: $O(\text{scan}(n)) = O\left(\frac{n}{B}\right)$ I/Os

Y is scanned n times: $O(n \cdot \text{scan}(n)) = O\left(\frac{n^2}{B}\right)$ I/Os

total: $O\left(\frac{n^2}{B}\right)$ I/Os

(ii) idea of the cache-aware algorithm: instead of scanning over the entire array Y for each element of X, scan over tiles of X-Y-combinations which fit entirely in memory. More precisely scan over $\frac{M}{2} - 2(B-1)$ items of X and $\frac{M}{2} - 2(B-1)$ items of Y at the same time, which allows for using $\frac{2\eta}{B}$ I/Os for analyzing η^2 X-Y combinations, where $\eta = \frac{M}{2} - 2(B-1)$

FindMin-CacheAware(X, Y, M, B)

1. $x \leftarrow +\infty$
2. $\eta \leftarrow \left\lfloor \frac{M}{2} - 2(B-1) \right\rfloor$
3. for $i \leftarrow 0$ to $\left\lceil \frac{n-1}{\eta} \right\rceil$
4. for $j \leftarrow 0$ to $\left\lceil \frac{n-1}{\eta} \right\rceil$
5. $x \leftarrow \min(x, \text{FindMin}(X[i \cdot \eta \ldots \min(n-1, (i+1)\cdot\eta)], Y[i\cdot\eta \ldots \min(n-1, (i+1)\cdot\eta)]))$
6. end for
7. end for
8. return x

For each time line 5 is executed, we note that all elements needed to execute FindMin fit into memory; this means each such execution requires at most $\frac{2\eta}{B} = O\left(\frac{M}{B}\right)$ I/Os.
The number of times line 5 is executed is

$$\left(\left\lceil \frac{n-1}{\eta} \right\rceil\right)^2 \leq \left(\left(\frac{n-1}{\eta}\right) + 1\right)^2 = \left(\frac{n-1}{\eta}\right)^2 + 2\frac{n-1}{\eta} + 1$$
$$= O\left(\frac{n^2}{\eta^2}\right) = O\left(\frac{n^2}{M^2}\right)$$

Thus, the total number of I/Os scales as $O\left(\frac{n^2}{M^2} \cdot \frac{M}{B}\right) = O\left(\frac{n^2}{MB}\right)$ I/Os.

(iii) idea of the cache-oblivious algorithm: we use a recursive approach, where we continuously split up the problem until it fits into memory.

FindMin-CacheOblivious$(X, Y, i_1, i_2, j_1, j_2)$

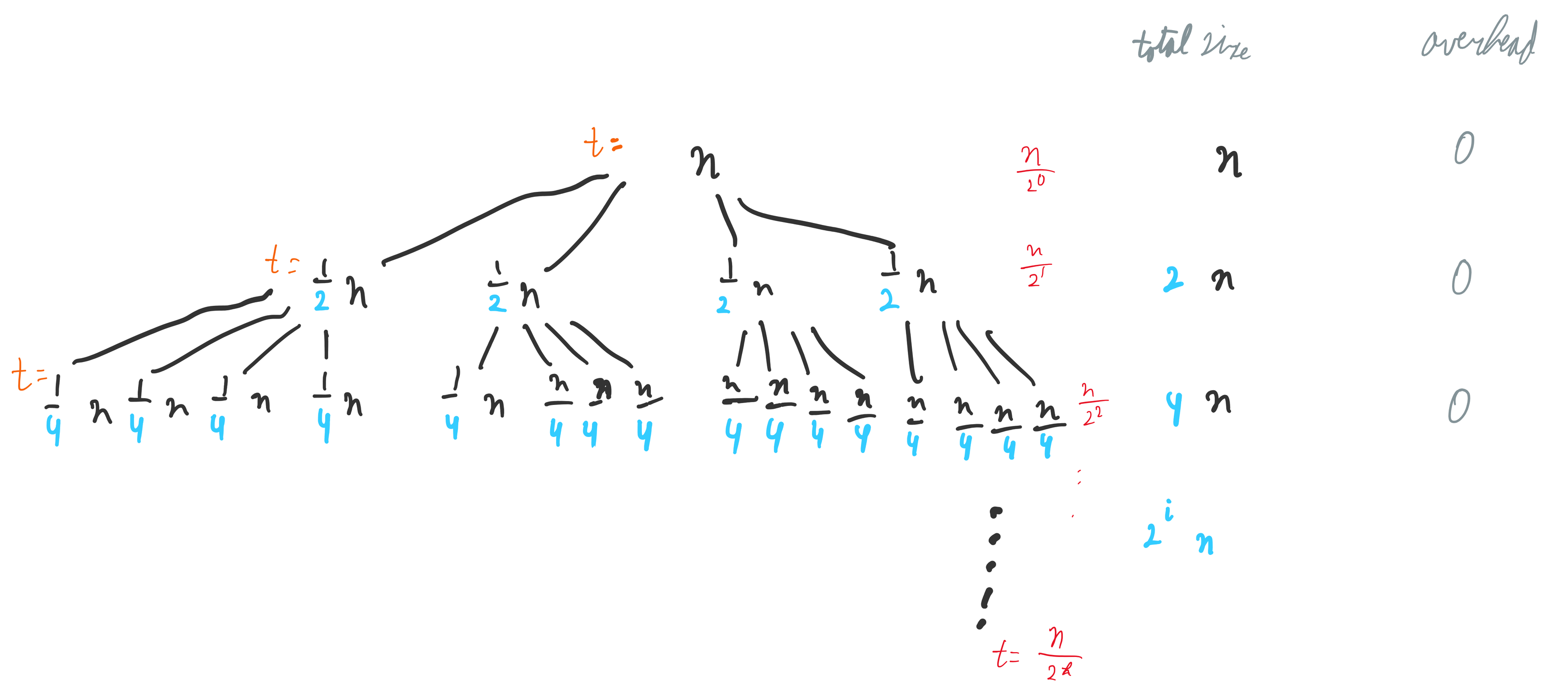
1. if $i_1 = i_2$ or $j_1 = j_2$ — note: we do not know exactly when the problem fits into memory; hence, we need to split it until the problem size becomes 1.
2. return FindMin$(X[i_1 \ldots i_2], Y[j_1 \ldots j_2])$
3. else
4. $i_{mid} \leftarrow \left\lfloor \frac{i_1 + i_2}{2} \right\rfloor$
5. $j_{mid} \leftarrow \left\lfloor \frac{j_1 + j_2}{2} \right\rfloor$
6. $x_1 \leftarrow$ FindMin-CacheOblivious$(X, Y, i_1, i_{mid}, j_1, j_{mid})$
7. $x_2 \leftarrow$ FindMin-CacheOblivious$(X, Y, i_1, i_{mid}, j_{mid}+1, j_2)$
8. $x_3 \leftarrow$ FindMin-CacheOblivious$(X, Y, i_{mid}+1, i_2, j_1, j_{mid})$
9. $x_4 \leftarrow$ FindMin-CacheOblivious$(X, Y, i_{mid}+1, i_2, j_{mid}+1, j_2)$
10. return $\min(x_1, x_2, x_3, x_4)$
11. endif

The number of I/Os $T_{IO}(t)$ for running this algorithm with problem size t, is given by

where t is the size of storing one subarray (we need two of those), we ignore rounding issues

$$T_{IO}(t) = \begin{cases} O\left(\frac{M}{B}\right) & \text{if the problem fits entirely into memory} \\ 4\,T_{IO}\left(\frac{t}{2}\right) & \text{otherwise} \end{cases} = \begin{cases} O\left(\frac{M}{B}\right) & \text{if } 2(t + (B-1)) \leq M \\ 4\,T_{IO}\left(\frac{t}{2}\right) & \text{otherwise} \end{cases}$$

Solving this recurrence for $t = n$ gives us the following recursion tree



The total number of I/Os for the tree is the

usually: $\sum_{i=0}^{k-1}$ overhead on level i — but here, this is zero, as overhead $= 0$ on all levels

$$T_{IO}(n) = \text{number of base cases} \cdot \text{number of I/Os per base case} + \text{total overhead}$$
$$= \#\text{ base cases} \cdot O\left(\frac{M}{B}\right)$$
$$= 4^k \cdot O\left(\frac{M}{B}\right)$$

We solve a simplified recurrence where the condition is replaced by $2t = M$

at the base cases, we have

based on the recurrence: $2t = M$ — because of using the tall-cache assumption to ignore blocks sticking out

$t = \frac{M}{2}$

at the base cases, we have that, for the height of the tree

$t = \frac{n}{2^k}$

$\frac{M}{2} = \frac{n}{2^k}$

$2^k = \frac{2n}{M}$

$$= (2^k)^2 \cdot O\left(\frac{M}{B}\right)$$
$$= \left(\frac{2n}{M}\right)^2 \cdot O\left(\frac{M}{B}\right)$$
$$= O\left(\frac{n^2}{M^2}\right) \cdot O\left(\frac{M}{B}\right)$$
$$= O\left(\frac{n^2}{MB}\right)$$