6.851: Advanced Data Structures, Spring 2021

Prof. Erik Demaine, Josh Brunner, Dylan Hendrickson, Yevhenii Diomidov

Problem Set 5 Solutions

Due: Thursday, March 25, 2021

Problem 5.1 [Walking the Matrix].

Solution: If N isn't a power of two, make the matrix at most 4 times bigger by extending it to the next power of two and filling with any value. Now assume N is a power of two.

We store the $N \times N$ matrix by dividing it into four equal quadrants and recursively storing each quadrant consecutively. Equivalently, the location in memory at which we store the entry at (i, j) computed by interleaving the bits of i and j. The key property of this order is that, for any k, grid-aligned blocks of size 2^k are consecutive.

We implement **teleport** and each **move** operation by simply maintaining the position of the finger, and **get** by looking up the entry corresponding to the current position of the finger.

Choose k such that $2^k \leq \sqrt{B} < 2^{k+1}$. Conceptually divide the matrix into squares of side length 2^k. Our layout ensures each square is consecutive in memory, so each square is contained in at most 2 cache blocks.

The cache-oblivious model follows an optimal replacement strategy, which will be at least as good as the following cache strategy:

- **teleport**: Clear the cache, and load the 9 squares (that is, the at most 18 cache blocks containing them) surrounding the square containing the new position.
- Any **move**: If the new position is in the cache, do nothing. Otherwise perform a **teleport**.
- **get**: Do nothing; the current position is guaranteed to be in the cache.

Clearly **teleport** uses at most $18 = O(1)$ memory transfers, and **get** uses 0. After a **teleport**, the nearest unloaded matrix entry is at least $2^k > \frac{1}{2}\sqrt{B}$ away from the current position, so we need to perform at least 2^k **move** operations before they trigger a **teleport**. So the amortized cost of each **move** operation is at most $18/2^k < 36/\sqrt{B} = O(1/\sqrt{B})$. We store up to $c = 18$ cache blocks simultaneously.