Can you get rotating an array to run faster than O(n²)?

devblogs.microsoft.com/oldnewthing/20100106-00

January 6, 2010

Raymond Chen

Some follow-up remarks to my old posting on rotating a two-dimensional array:

Some people noticed that the article I linked to purporting to rotate the array actually transposes it. I was wondering how many people would pick up on that.

I was surprised that people confused rotating an array (or matrix) with creating a rotation matrix. They are unrelated operations; the only thing they have in common are the letters r-o-t-a-t-i. A matrix is a representation of a linear transformation, and a rotation matrix is a linear transformation which rotates *vectors*. In other words, applying the rotation matrix to a vector produces a new vector which is a rotated version of the original vector. The linear transformation is a *function* of one parameter: It takes a vector and produces a new vector. A rotation matrix is a matrix which *rotates other things*. Whereas rotating an array is something you do *to the array*. The array is the thing being rotated, not the thing doing the rotating. It didn't even occur to me that people would confuse the two. It's the difference a phone dial and dialing a phone.

Showing that you cannot rotate an array via matrix multiplication is straightforward. Suppose there were a matrix R which rotated an array (laid out in the form of a matrix) clockwise. The result of rotating the identity matrix would be a a matrix with 1's along the diagonal from upper right to lower left, let's call that matrix J. Then we have $RI = J$, and therefore $R = J$. Now apply R to both sides: $RRI = RJ = I$ and therefore $R^2 = I$. But clearly rotating clockwise twice is not the identity for $n \geq 2$. (Rotating clockwise twice is turning upside-down.)

A more mechanical way to see this is to take the equation $R = J$ and show that J does not perform the desired operation; just try it on the matrix with 1 in the upper left entry and 0's everywhere else.

And since it's one of those geeky math pastimes to see how many differents proofs you can come up with for a single result, the third way to show that rotation cannot be effected by matrix multiplication is to observe that the transformation is not linear. (That's the magical algebra-theoretical way of showing it, which is either *so obvious you can tell just by looking*

at it or *so obscure it defies comprehension*.) [The transformation viewed as a transformation on matrices rather than a transformation on column vectors is indeed linear, but the matrix for that would be an $n^2 \times n^2$ matrix, and the operation wouldn't be matrix multiplication, so that doesn't help us here.]

The last question raised by this exercise was whether you could do better than $O(n^2)$. Computer science students spend so much time trying to push the complexity of an algorithm down that they neglect to learn how to tell that you can't go any lower. In this case, you obviously can't do better than $O(n^2)$ because every single one of the n^2 entries in the array needs to move (except of course the center element if n is odd). If you did less than $O(n^2)$ of work, then for sufficiently large n, you will end up not moving some array elements, which would be a failure to complete the required operation.

Bonus chatter: Mind you, you can do better than $O(n^2)$ if you change the rules of the problem. For example, if you allow *pretending* to move the elements, say by overloading the `[]` operator, then you can perform the rotation in $O(1)$ time by just writing a wrapper:

```
struct IArray
{
  virtual int& Element(int x, int y) = 0;
  virtual ~IArray() = 0;
};
class RotatedArray : public IArray {
public:
 RotatedArray(IArray *p) : m_p(p) { }
 ~RotatedArray() { delete m_p; }
 int& Element(int x, int y) {
  return m_p->Element(y, x);
 }
private:
 IArray *m_p;
};
void RotateInPlace(IArray *& p, int N)
{
 p = new RotatedArray(p);
}
```

This pseudo-rotates the elements by changing the accessor. Cute but doesn't actually address the original problem, which said that you were passed an array, not an interface that simulates an array.

Raymond Chen

Follow