The Power Method

Franck van Breugel

November 16, 2007

Consider the following recursive method.

```
public class Math
{
  /**
   * Returns the base raised to the power exponent.
   *
   * @param base the base.
   * @param exponent the exponent.
   * @pre. exponent >= 0
   * @impl. performs O(exponent) multiplications
   */
  public int pow(int base, int exponent)
  {
    if (exponent == 0)
    {
      return 1;
    }
    else
    {
      return base * Math.pow(base, exponent - 1);
  }
}
```

Let us first convince ourselves that the above recursive method returns the desired result. In the base case, when `exponent` is 0, the method returns 1 which is correct since $\texttt{base}^0 = 1$. For the recursive case, assume that the recursive call is correct, that is, assume that `Math.pow(base, exponent - 1)` returns $\texttt{base}^{\texttt{exponent}-1}$. In this case, the method returns $\texttt{base * base}^{\texttt{exponent}-1}$ which equals $\texttt{base}^{\texttt{exponent}}$. Hence, the method returns the correct result in this case as well.

Next, let us check that the above recursive method always terminates. We define the "size" of the problem solved by `Math.pow(base, exponent)` as `exponent`. Clearly, `exponent` is a non-negative integer. Since the recursive call solves a problem of smaller size (obviously, $\texttt{exponent} - 1 < \texttt{exponent}$), we can conclude that the method terminates.

Finally, let us analyze how many multiplications are performed by `Math.pow(base, exponent)`. Let *count* be a function that for a given `exponent` returns the number of multiplications performed by Math.pow(base, exponent). By inspecting the code of the above method we can conclude that

$$count(\texttt{exponent}) = \begin{cases} 0 & \text{if } \texttt{exponent} = 0 \\ 1 + count(\texttt{exponent} - 1) & \text{otherwise} \end{cases}$$

Next, we prove by induction on exponent that $count(\texttt{exponent}) = \texttt{exponent}$ for all $\texttt{exponent} \geq 0$. In the base case, when $\texttt{exponent} = 0$, the property is vacuously true. Assume that $\texttt{exponent} > 0$ and suppose that $count(\texttt{exponent} - 1) = \texttt{exponent} - 1$ (induction hypothesis). Then

$$\begin{aligned} count(\texttt{exponent}) &= 1 + count(\texttt{exponent} - 1) \\ &= 1 + (\texttt{exponent} - 1) \quad \text{[induction hypothesis]} \\ &= \texttt{exponent}. \end{aligned}$$

To conclude that $count \in O(\texttt{exponent})$, we pick the "minimal size" M to be 0 and the "factor" F to be 1. Then it remains to show that

$$\forall \texttt{exponent} \geq 0 \; count(\texttt{exponent}) \leq \texttt{exponent}$$

which is obviously true.

Consider the following recursive method.

```
public class Math
{
  /**
   * Returns the base raised to the power exponent.
   *
   * @param base the base.
   * @param exponent the exponent.
   * @pre. exponent >= 0
   * @impl. performs O(log(exponent)) multiplications
   */
  public int pow(int base, int exponent)
  {
    if (exponent == 0)
    {
      return 1;
    }
    else
    {
      if (exponent % 2 == 0)
      {
        int temp = Math.pow(base, exponent / 2);
        return temp * temp;
      }
      else
      {
        return base * Math.pow(base, exponent - 1);
```

```
            }
        }
    }
}
```

Again, first we convince ourselves that the above recursive method returns the desired result. The base case is the same as before. For this method, there are two recursive cases. Let us first consider the case that `exponent` is even. Assume that `Math.pow(base, exponent / 2)` returns $\texttt{base}^{\texttt{exponent}/2}$. In this case, the method returns $\texttt{base}^{\texttt{exponent}/2} \times \texttt{base}^{\texttt{exponent}/2}$ which equals $\texttt{base}^{\texttt{exponent}}$. The case that `exponent` is odd is the same as before.

Next, let us check that the above recursive method always terminates. We define the "size" of the problem solved by `Math.pow(base, exponent)` as `exponent`. Clearly, `exponent` is a non-negative integer. Since the recursive call solves a problem of smaller size (obviously, $\texttt{exponent}/2 < \texttt{exponent}$ and $\texttt{exponent} - 1 < \texttt{exponent}$), we can conclude that the method terminates.

Finally, let us analyze how many multiplications are performed by `Math.pow(base, exponent)`. Let *count* be a function that for a given `exponent` returns the number of multiplications performed by `Math.pow(base, exponent)`. By inspecting the code of the above method we can conclude that

$$\mathit{count}(\texttt{exponent}) = \begin{cases} 0 & \text{if } \texttt{exponent} = 0 \\ 1 + \mathit{count}(\texttt{exponent}/2) & \text{if } \texttt{exponent} \text{ is even} \\ 1 + \mathit{count}(\texttt{exponent} - 1) & \text{if } \texttt{exponent} \text{ is odd} \end{cases}$$

If `exponent` is odd then

$$\begin{aligned} \mathit{count}(\texttt{exponent}) &= 1 + \mathit{count}(\texttt{exponent} - 1) \\ &= 2 + \mathit{count}((\texttt{exponent} - 1)/2) \quad [\texttt{exponent - 1} \text{ is even}] \end{aligned}$$

Next, we prove by induction on `exponent` that $\mathit{count}(\texttt{exponent}) \leq 1 + 2\log_2(\texttt{exponent})$ for all $\texttt{exponent} \geq 1$. In the base case, when $\texttt{exponent} = 1$, we have that

$$\begin{aligned} \mathit{count}(1) &= 1 \\ &= 1 + 2\log_2(1) \end{aligned}$$

and, hence, the property holds in this case. Next we consider that $\texttt{exponent} > 1$. Assume that $\mathit{count}(e) \leq 1 + 2\log_2(e)$ for all $e < \texttt{exponent}$ (induction hypothesis). We distinguish between the cases that `exponent` is even and odd. Assume that `exponent` is even. Then

$$\begin{aligned} \mathit{count}(\texttt{exponent}) &= 1 + \mathit{count}(\texttt{exponent}/2) \\ &\leq 1 + 1 + 2\log_2(\texttt{exponent}/2) \quad [\text{induction hypothesis}] \\ &= 2 + 2(\log_2(\texttt{exponent}) - 1) \\ &= 2\log_2(\texttt{exponent}) \\ &\leq 1 + 2\log_2(\texttt{exponent}). \end{aligned}$$

Assume that `exponent` is odd. Then

$$\begin{aligned} \mathit{count}(\texttt{exponent}) &= 2 + \mathit{count}((\texttt{exponent} - 1)/2) \\ &\leq 2 + 1 + 2\log_2((\texttt{exponent} - 1)/2) \quad [\text{induction hypothesis}] \end{aligned}$$

$$
\begin{aligned}
&= 2+1+2(\log_2(\texttt{exponent}-1)-1) \\
&= 1+2\log_2(\texttt{exponent}-1) \\
&\leq 1+2\log_2(\texttt{exponent}).
\end{aligned}
$$

To conclude that $\mathit{count} \in O(\log_2(\texttt{exponent}))$, we pick the "minimal size" M to be 2 and the "factor" F to be 3. Then it remains to show that

$$\forall \texttt{exponent} \geq 2 \;\; \mathit{count}(\texttt{exponent}) \leq 3\log_2(\texttt{exponent})$$

which follows from the observation that for all $\texttt{exponent} \geq 2$,

$$
\begin{aligned}
\mathit{count}(\texttt{exponent}) &\leq 1+2\log_2(\texttt{exponent}) \\
&\leq \log_2(\texttt{exponent})+2\log_2(\texttt{exponent}) \quad [\texttt{exponent} \geq 2 \text{ and hence } \log_2(\texttt{exponent}) \geq 1] \\
&= 3\log_2(\texttt{exponent}).
\end{aligned}
$$

One may wonder whether the local variable `temp` in the above method is needed. That is, one may wonder if there is any change in the number of multiplications if

```
int temp = Math.pow(base, exponent / 2);
return temp * temp;
```

is replaced with

```
return Math.pow(base, exponent / 2) * Math.pow(base, exponent / 2);
```

In this case, we get

$$\mathit{count}(\texttt{exponent}) = 2\mathit{count}(\texttt{exponent}/2)$$

if $\texttt{exponent} > 0$ and `exponent` is even. As a consequence, $\mathit{count}(\texttt{exponent}) = \texttt{exponent}$ and $\mathit{count} \in O(\texttt{exponent})$. We leave the proofs of these facts to the reader. From the above we can conclude that the local variable `temp` is essential for the efficiency of the `pow` method.