Exponentiation by Squaring

April 2, 2020

Concepts to take away

- Usefulness of divide and conquer algorithms
- How the exponentiation function is implemented
- How seemingly disjoint problems are connected

Short slide on Divide and Conquer

Reduce repeated work by recursively splitting the problem.

1. Splitting the problem
2. Combining the solutions of the two sub-problems

Introduction

$$2^8 = \overbrace{2 \times 2 \times \cdots \times 2 \times 2}^{8}$$

$$a^n = \overbrace{a \times a \times \cdots \times a \times a}^{n}$$

```
def pow(a, n):
    # ???
```

Iterative Solution

$$a^n = 1 \times \overbrace{a \times a \cdots a \times a}^{n}$$

```
def pow(a, n):
    ret = 1
    for i in range(n):
        ret *= a
    return ret
```

Iterative Solution

$$a^n = 1 \times \overbrace{a \times a \cdots a \times a}^{n}$$

```python
def pow(a, n):
    ret = 1
    for i in range(n):
        ret *= a
    return ret
```

There are n multiplications here so the function is proportional to input size n.

Key Observation

$$2^8 = \overbrace{2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2}^{8}$$

Key Observation

$$2^8 = \overbrace{2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2}^{8}$$

$$2^8 = \overbrace{2 \cdot 2 \cdot 2 \cdot 2}^{4} \cdot \overbrace{2 \cdot 2 \cdot 2 \cdot 2}^{4}$$

Key Observation

$$2^8 = \overbrace{2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2}^{8}$$

$$2^8 = \overbrace{2 \cdot 2 \cdot 2 \cdot 2}^{4} \cdot \overbrace{2 \cdot 2 \cdot 2 \cdot 2}^{4}$$

$$2^8 = \overbrace{2 \cdot 2}^{2} \cdot \overbrace{2 \cdot 2}^{2} \cdot \overbrace{2 \cdot 2}^{2} \cdot \overbrace{2 \cdot 2}^{2}$$

We can use this observation to reduce the computational speed!

Developing the Equation

$$2^8 = (2 \cdot 2)^{\frac{8}{2}} = 4^4 \qquad \left(x^2\right)^{\frac{n}{2}} = x^{2 \cdot \frac{n}{2}} = x^n$$

Developing the Equation

$$\begin{aligned} 2^8 &= (2\cdot 2)^{\frac{8}{2}} = 4^4 \\ &= (4\cdot 4)^{\frac{4}{2}} = 16^2 \end{aligned}$$

$$\left(x^2\right)^{\frac{n}{2}} = x^{2\cdot\frac{n}{2}} = x^n$$

Developing the Equation

$$\begin{aligned} 2^8 &= (2 \cdot 2)^{\frac{8}{2}} = 4^4 \\ &= (4 \cdot 4)^{\frac{4}{2}} = 16^2 \\ &= 16 \cdot 16 = 4096 \end{aligned}$$

$$\left(x^2\right)^{\frac{n}{2}} = x^{2 \cdot \frac{n}{2}} = x^n$$

Only 3 multiplications needed!

What about odd exponents?

$$3^{15} = \overbrace{3 \times 3 \times \cdots \times 3 \times 3}^{14} \times 3$$

$$3^{15} = (3 \cdot 3)^{\frac{14}{2}} \cdot 3 = 9^7 \cdot 3$$

$$\left(x^2\right)^{\frac{n-1}{2}} \cdot x = x^n$$

What about odd exponents?

$$3^{15} = \overbrace{3 \times 3 \times \cdots \times 3 \times 3}^{14} \times 3$$

$$3^{15} = (3 \cdot 3)^{\frac{14}{2}} \cdot 3 = 9^7 \cdot 3$$

$$= (9 \cdot 9)^{\frac{6}{2}} \cdot 9 \cdot 3 = 81^3 \cdot 9 \cdot 3$$

$$\left(x^2\right)^{\frac{n-1}{2}} \cdot x = x^n$$

What about odd exponents?

$$3^{15} = \overbrace{3 \times 3 \times \cdots \times 3 \times 3}^{14} \times 3$$

$$3^{15} = (3 \cdot 3)^{\frac{14}{2}} \cdot 3 = 9^7 \cdot 3$$

$$= (9 \cdot 9)^{\frac{6}{2}} \cdot 9 \cdot 3 = 81^3 \cdot 9 \cdot 3$$

$$= 81 \cdot 81 \cdot 81 \cdot 9 \cdot 3 = 14348907$$

$$\left(x^2\right)^{\frac{n-1}{2}} \cdot x = x^n$$

All together now

$$a^n = \begin{cases} (a^2)^{\frac{n}{2}} & n \text{ is even} \\ (a^2)^{\frac{n-1}{2}} \times a & n \text{ is odd} \end{cases}, \qquad a^0 = 1$$

Intuitively, n is being halved in every call - resulting in $\log n$ multiplications to be done.

Recursive Code

$$a^n = \begin{cases} (a^2)^{\frac{n}{2}} & n \text{ is even} \\ (a^2)^{\frac{n-1}{2}} \times a & n \text{ is odd} \end{cases}, \qquad a^0 = 1$$

```
def pow(a, n):
    if n == 0: return 1
    if n % 2 == 1:
        return pow(a * a, n // 2) * a
    else:
        return pow(a * a, n / 2)
```

Fibonacci Sequence

$$f(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ f(n-1) + f(n-2) & n > 1 \end{cases}$$

```python
def fib(n):
    if n <= 1: return n
    return fib(n - 1) + fib(n - 2)
```

The number of additions in this function is approximately 2^n.*

Can we go faster?

```python
def fib(n):
    y = 0
    x = 1
    for i in range(n):
        y, x = x, x + y
    return y
```

In this example, y holds the value of $f(n)$, x holds the value of $f(n+1)$. Each iteration updates x and y according to the equations above. This implementation involves n additions, which is $<< 2^n$.

Review of Matrices

A matrix is a 2-D rectangular array of numbers.

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

A vector is a 1-D rectangular array of numbers.

$$v = \begin{bmatrix} x \\ y \end{bmatrix}$$

You can multiply them together.

$$Av = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax + by \\ cx + dy \end{bmatrix} \qquad \text{Matrix * Vector}$$

$$AB = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x & p \\ y & q \end{bmatrix} = \begin{bmatrix} ax + by & ap + bq \\ cx + dy & cp + dq \end{bmatrix} \qquad \text{Matrix * Matrix}$$

More Matrix Properties

$$A(BC) = (AB)C \qquad \text{Associative Property}$$

$$A(Bv) = (AB)v \qquad \text{Associative Property}$$

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \text{Identity Matrix}$$

$$AI = IA = A$$

Modelling Recursive Functions Using Matrices

```python
def fib(n):
    y = 0
    x = 1
    for i in range(n):
        y, x = x, x + y
    return y
```

$$\overbrace{\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}}^{T} \overbrace{\begin{bmatrix} x \\ y \end{bmatrix}}^{v^{[i]}} = \overbrace{\begin{bmatrix} x + y \\ x \end{bmatrix}}^{v^{[i+1]}}$$

We define a **transition matrix**, $T = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$, to go from the i iteration to $i + 1$ iteration, or from $v^{[i]}$ to $v^{[i+1]}$.

Modelling Recursive Functions Using Matrices

```python
def fib(n):
    y = 0
    x = 1
    for i in range(n):
        y, x = x, x + y
    return y
```

$$\overbrace{\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}}^{T} \overbrace{\begin{bmatrix} \mathrm{x} \\ y \end{bmatrix}}^{v^{[i]}} = \overbrace{\begin{bmatrix} \mathrm{x} + \mathrm{y} \\ x \end{bmatrix}}^{v^{[i+1]}}$$

We define a **transition matrix**, $T = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$, to go from the i iteration to $i+1$ iteration, or from $v^{[i]}$ to $v^{[i+1]}$.

Modelling Recursive Functions Using Matrices

```
def fib(n):
    y = 0
    x = 1
    for i in range(n):
        y, x = x, x + y
    return y
```

$$\overbrace{\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}}^{T} \overbrace{\begin{bmatrix} x \\ \mathrm{y} \end{bmatrix}}^{v^{[i]}} = \overbrace{\begin{bmatrix} x + y \\ \mathrm{x} \end{bmatrix}}^{v^{[i+1]}}$$

We define a **transition matrix**, $T = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$, to go from the i iteration to $i + 1$ iteration, or from $v^{[i]}$ to $v^{[i+1]}$.

Some examples

Start with $T = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$, $v^{[0]} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$.

$$v^{[1]} = Tv^{[0]} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1+0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$v^{[2]} = Tv^{[1]} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1+1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

$$v^{[3]} = Tv^{[2]} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 2+1 \\ 2 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

$$v^{[4]} = Tv^{[3]} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 3 \\ 2 \end{bmatrix} = \begin{bmatrix} 3+2 \\ 3 \end{bmatrix} = \begin{bmatrix} 5 \\ 3 \end{bmatrix}$$

Some examples

Start with $T = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$, $v^{[0]} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$.

$$v^{[1]} = Tv^{[0]} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1+0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$v^{[2]} = Tv^{[1]} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1+1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

$$v^{[3]} = Tv^{[2]} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 2+1 \\ 2 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

$$v^{[4]} = Tv^{[3]} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 3 \\ 2 \end{bmatrix} = \begin{bmatrix} 3+2 \\ 3 \end{bmatrix} = \begin{bmatrix} 5 \\ 3 \end{bmatrix}$$

Highlighted numbers are $f(n)$.

Key Observation

$$v^{[1]} = Tv^{[0]}$$

Key Observation

$$v^{[1]} = Tv^{[0]}$$

$$v^{[2]} = Tv^{[1]} = T \cdot Tv^{[0]}$$

Key Observation

$$
\begin{aligned}
v^{[1]} &= Tv^{[0]} \\
v^{[2]} &= Tv^{[1]} = T \cdot Tv^{[0]} \\
v^{[3]} &= Tv^{[2]} = T \cdot T \cdot Tv^{[0]}
\end{aligned}
$$

Key Observation

$$
\begin{aligned}
v^{[1]} &= Tv^{[0]} \\
v^{[2]} &= Tv^{[1]} = T \cdot Tv^{[0]} \\
v^{[3]} &= Tv^{[2]} = T \cdot T \cdot Tv^{[0]} \\
&\vdots \\
v^{[n]} &= \overbrace{T \cdot T \cdots T \cdot T}^{n} v^{[0]}
\end{aligned}
$$

Key Observation

$$
\begin{aligned}
v^{[1]} &= Tv^{[0]} \\
v^{[2]} &= Tv^{[1]} = T \cdot Tv^{[0]} \\
v^{[3]} &= Tv^{[2]} = T \cdot T \cdot Tv^{[0]} \\
&\vdots \\
v^{[n]} &= \overbrace{T \cdot T \cdots T \cdot T}^{n} v^{[0]} \\
v^{[n]} &= T^n v^{[0]}
\end{aligned}
$$

Computing Fibonacci

$$f(n) = v_y^{[n]}, v^{[n]} = T^n v^{[0]}$$

```
def pow(a, n):
    if n == 0: return np.eye(2) # Identity
    if n % 2 == 1:
        return pow(a * a, n // 2) * a
    else:
        return pow(a * a, n / 2)

def fib(n):
    T = np.matrix([[1,1], [1,0]])
    v_0 = np.matrix([[1], [0]])
    T_n = pow(T, n)
    v_n = T_n * v_0
    return v_n[1,0] # Returning the second value
```

Computing Fibonacci

```python
def fib(n):
    T = np.matrix([[1,1], [1,0]])
    v_0 = np.matrix([[1], [0]])
    T_n = pow(T, n) # Log n operation
    v_n = T_n * v_0
    return v_n[1,0] # Returning the second value
```

The number of multiplications for matrix multiplication is dim^3 but dimension is constant here.
Thus, the computation is on order of $\log n$.
This strategy works for any linear recurrence!

Extra: Deriving Binet's Formula

$$f(n) = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}}$$

Diagonalization of $T = PDP^{-1}$. $T^n = PD^nP^{-1}$

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} \frac{1+\sqrt{5}}{2} & \frac{1-\sqrt{5}}{2} \\ 1 & 1 \end{bmatrix} \begin{bmatrix} \frac{1+\sqrt{5}}{2} & 0 \\ 0 & \frac{1-\sqrt{5}}{2} \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{5}} & -\frac{1-\sqrt{5}}{2\sqrt{5}} \\ -\frac{1}{\sqrt{5}} & \frac{1+\sqrt{5}}{2\sqrt{5}} \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n = \begin{bmatrix} \frac{1+\sqrt{5}}{2} & \frac{1-\sqrt{5}}{2} \\ 1 & 1 \end{bmatrix} \begin{bmatrix} \left(\frac{1+\sqrt{5}}{2}\right)^n & 0 \\ 0 & \left(\frac{1-\sqrt{5}}{2}\right)^n \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{5}} & -\frac{1-\sqrt{5}}{2\sqrt{5}} \\ -\frac{1}{\sqrt{5}} & \frac{1+\sqrt{5}}{2\sqrt{5}} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{\left(\frac{1+\sqrt{5}}{2}\right)^{n+1} - \left(\frac{1-\sqrt{5}}{2}\right)^{n+1}}{\sqrt{5}} & (\cdots) \\ \frac{\left(\frac{1+\sqrt{5}}{2}\right)^{n} - \left(\frac{1-\sqrt{5}}{2}\right)^{n}}{\sqrt{5}} & (\cdots) \end{bmatrix} = \begin{bmatrix} f(n+1) & (\cdots) \\ f(n) & (\cdots) \end{bmatrix}$$