

Fast Exponentiation

CS 491 – Competitive Programming

Dr. Mattox Beckman

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
DEPARTMENT OF COMPUTER SCIENCE

Spring 2025

Objectives

- ▶ Use binary representation to compute exponentials in logarithmic time.
- ▶ Use a similar technique with matrices to compute Fibonacci numbers.

Naïve Exponent

Remember the equation for n^m :

$$b^0 = 1$$

$$b^n = b * b^{n-1}$$

Recursive Implementation

```
1 int exponent(int base, int n) {  
2     if (n == 0)  
3         return 1;  
4     return n * exponent(base, n-1);  
5 }
```

Naïve Factorial, II

Iterative Implementation

```
1 int exponent(int n) {
2     int out = 1;
3     while (n>0) {
4         out *= base;
5         --n;
6     }
7     return out;
8 }
```

- ▶ What is the time complexity?

Trick: Use a Binary Representation of the Exponent

- ▶ Q: What is 3^{22} ?
 - ▶ Remember $a^x a^y = a^{x+y}$
- ▶ A: $3^{22} = 3^{10110_2} = 3^{10000_2} 3^{00100_2} 3^{00010_2} = 3^{16} 3^4 3^2$
 $= 1 \cdot 3^{10000_2} \times 1 \cdot 3^{00100_2} \times 1 \cdot 3^{00010_2}$

So, for example ...

- ▶ $3^4 \times 3^4 = 3^8$
- ▶ So, we can keep track of the current factor and get the next factor by squaring....

Implementation

Call with base=3 and n=22

```
1 int fexp(int b, int n) {
2     int out = 1;
3     int fact = b;
4
5     while (n>0) {
6         if (n & 1)
7             out *= fact;
8         fact *= fact;
9         n >>= 1;
10    }
11
12    return out;
13 }
```

► Initialize: out=1, fact=3, n=22

Implementation

Call with base=3 and n=22

```
1 int fexp(int b, int n) {
2     int out = 1;
3     int fact = b;
4
5     while (n>0) {
6         if (n & 1)
7             out *= fact;
8         fact *= fact;
9         n >>= 1;
10    }
11
12    return out;
13 }
```

- ▶ Initialize: out=1, fact=3, n=22
- ▶ n (22) & 1 is 0:
out=1, fact=9, n=11

Implementation

Call with base=3 and n=22

```
1 int fexp(int b, int n) {
2     int out = 1;
3     int fact = b;
4
5     while (n>0) {
6         if (n & 1)
7             out *= fact;
8         fact *= fact;
9         n >>= 1;
10    }
11
12    return out;
13 }
```

- ▶ Initialize: out=1, fact=3, n=22
- ▶ n (22) & 1 is 0:
out=1, fact=9, n=11
- ▶ n (11) & 1 is 1:
out=9, fact=81, n=5

Implementation

Call with base=3 and n=22

```
1 int fexp(int b, int n) {
2     int out = 1;
3     int fact = b;
4
5     while (n>0) {
6         if (n & 1)
7             out *= fact;
8         fact *= fact;
9         n >>= 1;
10    }
11
12    return out;
13 }
```

- ▶ Initialize: out=1, fact=3, n=22
- ▶ n (22) & 1 is 0:
out=1, fact=9, n=11
- ▶ n (11) & 1 is 1:
out=9, fact=81, n=5
- ▶ n (5) & 1 is 1:
out=729, fact=6561, n=2

Implementation

Call with base=3 and n=22

```
1 int fexp(int b, int n) {
2     int out = 1;
3     int fact = b;
4
5     while (n>0) {
6         if (n & 1)
7             out *= fact;
8         fact *= fact;
9         n >>= 1;
10    }
11
12    return out;
13 }
```

- ▶ Initialize: out=1, fact=3, n=22
- ▶ n (22) & 1 is 0:
out=1, fact=9, n=11
- ▶ n (11) & 1 is 1:
out=9, fact=81, n=5
- ▶ n (5) & 1 is 1:
out=729, fact=6561, n=2
- ▶ n (2) & 1 is 0:
out=729, fact=43046721, n=1

Implementation

Call with base=3 and n=22

```
1 int fexp(int b, int n) {
2     int out = 1;
3     int fact = b;
4
5     while (n>0) {
6         if (n & 1)
7             out *= fact;
8         fact *= fact;
9         n >>= 1;
10    }
11
12    return out;
13 }
```

- ▶ Initialize: out=1, fact=3, n=22
- ▶ n (22) & 1 is 0:
out=1, fact=9, n=11
- ▶ n (11) & 1 is 1:
out=9, fact=81, n=5
- ▶ n (5) & 1 is 1:
out=729, fact=6561, n=2
- ▶ n (2) & 1 is 0:
out=729, fact=43046721, n=1
- ▶ n (1) & 1 is 1:
out=31381059609, n=0

Calculating Fibonacci Numbers

Wrong Way

```
1 int fib(int n) {
2     if (n<2) return 1;
3     return fib(n-1) + fib(n-2)
4 }
```

Reasonable way

```
1 int fib(int n, int a=1, int b=1) {
2     if (n==1)
3         return a;
4     else
5         return fib(n-1,b,a+b);
6 }
```

Fast Fibonacci

- ▶ We can do even better! Consider this equation:

$$\begin{aligned}f'_1 &= f_1 + f_2 \\f'_2 &= f_1\end{aligned}$$

- ▶ We can represent this in matrix form.

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

- ▶ Repeating

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

- ▶ Again...!

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 3 \\ 2 \end{bmatrix} = \begin{bmatrix} 5 \\ 3 \end{bmatrix}$$

Square the Matrix

- ▶ Square the matrix to do multiple steps:

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}$$

- ▶ Then...

$$\begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 5 \\ 3 \end{bmatrix}$$

- ▶ You can use the same technique as with fast exponents to “power up” this matrix and compute large Fibonacci numbers in logarithmic time.