Fast Exponentiation

CS 491 – Competitive Programming

Dr. Mattox Beckman

University of Illinois at Urbana-Champaign
Department of Computer Science

Fall 2024

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

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

Naïve Factorial, II

Iterative Implementation

```
int fact(int n) {
  int out = 1;
  while (n>0) {
    out *= base;
    --n;
  }
  return out;
}
```

- 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: $3^{22} = 3^{10110_2} = 3^{10000_2} 3^{00100_2} 3^{00010_2} = 3^{16} 3^4 3^2$
  $= 1 \cdot 3^{10000_2} \times 1 \cdot 3^{00100_2} \times 1 \cdot 3^{00010_2}$

Implementation

Call with base=3 and n=22

```
int fexp(int b, int n) {
  int out = 1;
  int fact = b;

  while (n>0) {
    if (n & 1)
      out *= fact;
    fact *= fact;
    n >>= 1;
  }

  return out;
}
```

- Initialize: out=1, fact=3, n=22

Implementation

Call with base=3 and n=22

```
int fexp(int b, int n) {
  int out = 1;
  int fact = b;

  while (n>0) {
    if (n & 1)
      out *= fact;
    fact *= fact;
    n >>= 1;
  }

  return out;
}
```

- 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

```
int fexp(int b, int n) {
  int out = 1;
  int fact = b;

  while (n>0) {
    if (n & 1)
      out *= fact;
    fact *= fact;
    n >>= 1;
  }

  return out;
}
```

- 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

```
int fexp(int b, int n) {
  int out = 1;
  int fact = b;

  while (n>0) {
    if (n & 1)
      out *= fact;
    fact *= fact;
    n >>= 1;
  }

  return out;
}
```

- 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

```
int fexp(int b, int n) {
  int out = 1;
  int fact = b;

  while (n>0) {
    if (n & 1)
      out *= fact;
    fact *= fact;
    n >>= 1;
  }

  return out;
}
```

- 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

```
int fexp(int b, int n) {
  int out = 1;
  int fact = b;

  while (n>0) {
    if (n & 1)
      out *= fact;
    fact *= fact;
    n >>= 1;
  }

  return out;
}
```

- 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

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

Reasonable way

```
int fib(int n, int a=1, int b=1) {
  if (n==1)
    return a;
  else
    return fib(n-1,b,a+b);
}
```

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.