1 Recursive Formulae

Suppose that we want to compute \( 2^n \mod M \) for some numbers \( n \geq 0 \) and \( M \geq 2 \). This can require a lot of digits to write down for large \( n \), and we want to avoid that, since the end result is \( \leq M \).

Our first attempt avoids multiplication and only uses addition modulo \( M \). We use the fact that \( 2^{n-1} \cdot 2^n \equiv 2^n \mod M \).

```python
function PowerOfTwo(x, M):
    if x == 0 then
        return 1
    return (PowerOfTwo(x - 1, M) + PowerOfTwo(x - 1, M)) mod M
```

What is the runtime of the above algorithm?
- \( \Theta(n^2) \)
- \( \Theta(\log^2 n) \)
- \( \Theta(2^n) \)

Correct: \( \Theta(\log^2 n) \)

Now let us replace this algorithm with an iterative one that stores the results:

\[
A[0] = 1
\]

for \( i = 1 \ldots \log n \) do

\[
A[i] = (A[i-1] + A[i-1]) \mod M
\]

return \( A[\log n] \)

What is the runtime of the above algorithm?
- \( \Theta(\log n) \)
- \( \Theta(2^n) \)
- \( \Theta(\log^2 n) \)

Correct: \( \Theta(\log n) \)

What if we are allowed to use multiplication? Suppose that \( n \) is a power of two.

\[
B[0] = 1
\]

for \( i = 1 \ldots \log n \) do

\[
B[i] = (B[i-1] + B[i-1]) \mod M
\]

return \( B[\log n] \)

What is the value of \( B[i] \) in the above algorithm?
- \( 2^n \mod M \)
- \( 2^{\log n} \mod M \)
- \( 2^m \mod M \)

Correct: \( 2^n \mod M \)

What is the runtime of the above algorithm?
- \( \Theta(n) \)
- \( \Theta(2^n) \)
- \( \Theta(\log n) \)

Correct: \( \Theta(n) \)

What if \( n \) is not a power of two? We can run the following slightly modified algorithm:

\[
B[0] = 1
\]

for \( i = 1 \ldots \log n \) do

\[
B[i] = (B[i-1] + B[i-1]) \mod M
\]

Let the binary representation of \( n \) be \((x_{log(n)}\ldots x_0)\).

\[
B[\log n] = (B[\log (n - 1)] + B[\log (n - 1)]) \mod M
\]

What is the runtime of this algorithm?
- \( \Theta(n) \)
- \( \Theta(2^n) \)
- \( \Theta(\log n) \)

Correct: \( \Theta(\log n) \)

Remark: A clever algorithm inspired by the above can compute \( \text{Fibonacci}(n) \) modulo a desired number \( M \), in time \( O(\log(n)) \). As a challenge, try to use the following identity involving Fibonacci numbers and matrix multiplication, to come up with this \( O(\log(n)) \) algorithm:

\[
\text{Fibonacci}(n) = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n \begin{bmatrix} 1 \\ 0 \end{bmatrix}
\]

2 Shortest Paths

Suppose that we have a weighted graph with \( n \) vertices and \( m \) edges and no negative cycles (so shortest paths are well-defined). Suppose for the below questions that our implementation of Dijkstra uses \( \min(\text{distance}, \text{distance}) \) and is not Fibonacci heaps.

If \( m = \Theta(n^2) \), and we want to find the shortest path between some \( s \) and \( v \) which algorithm should we use? We prefer algorithms with the smallest worst-case runtime.

- Dijkstra
- \( \Theta(n \cdot \log(n)) \)
- Floyd-Warshall

Correct: Dijkstra

If \( m = \Theta(n^2) \), and we want to find the shortest path between some \( s \) and \( v \) which algorithm should we use when \( n > \Theta(2^{\log(n)}) \)?

- \( \Theta(n \cdot \log(n)) \)
- \( \Theta(n \cdot \log(n)) \)
- Floyd-Warshall

Correct: Floyd-Warshall

Two or more of the above algorithms are correct and have the smallest worst-case runtime.

Suppose that we have a graph with \( m = \Theta(n^3) \) edges that all have nonnegative weights. Which algorithm should we use to find the shortest path between all pairs of vertices?

- \( \Theta(n \cdot \text{rows of Dijkstra}) \)
- \( \Theta(n \cdot \text{rows of Dijkstra}) \)
- \( \Theta(n \cdot \text{rows of Dijkstra}) \)
- \( \Theta(n \cdot \text{rows of Dijkstra}) \)
- \( \Theta(n \cdot \text{rows of Dijkstra}) \)

Correct: \( \Theta(n \cdot \text{rows of Dijkstra}) \)

Two or more of the above algorithms are correct and have the smallest worst-case runtime.

Suppose that we have a graph with \( m = \Theta(n^3) \) edges that all have nonnegative weights. Which algorithm should we use to find the shortest path between all pairs of vertices?

- \( \Theta(n \cdot \text{rows of Dijkstra}) \)
- \( \Theta(n \cdot \text{rows of Dijkstra}) \)
- \( \Theta(n \cdot \text{rows of Dijkstra}) \)
- \( \Theta(n \cdot \text{rows of Dijkstra}) \)
- \( \Theta(n \cdot \text{rows of Dijkstra}) \)

Correct: \( \Theta(n \cdot \text{rows of Dijkstra}) \)

Two or more of the above algorithms are correct and have the smallest worst-case runtime.