Dynamic Programming I

Reset Progress Reveal Solutions

1 Recursive Formulae

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

Our first attempt avoids multiplication and only uses addition modulo M. We use the fact that $2^n = 2^{n-1} + 2^{n-1} \pmod{M}$.

function *PowerOfTwo*(n, M)**:**
  if $n = 0$ **then**
    return 1
  return $(\mathit{PowerOfTwo}(n-1, M) + \mathit{PowerOfTwo}(n-1, M)) \mod M$

What is the runtime of the above algorithm?

○ $\Theta(n)$
● $\Theta(2^n)$
○ $\Theta(\log n)$

Correct

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

$A \leftarrow$ array indexed with $0, \ldots, n$
$A[0] \leftarrow 1$
for $i = 1, \ldots, n$ **do**
  $A[i] \leftarrow (A[i-1] + A[i-1]) \mod M$
return $A[n]$

What is the runtime of the above algorithm?

● $\Theta(n)$
○ $\Theta(2^n)$
○ $\Theta(\log n)$

Correct

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

$B \leftarrow$ array indexed with $0, \ldots, \log n$
$B[0] \leftarrow 2$
for $i = 1, \ldots, \log n$ **do**
  $B[i] \leftarrow (B[i-1] \times B[i-1]) \mod M$
return $B[\log n]$

What is the value of $B[i]$ in the above algorithm?

● $2^{2^i} \mod M$
○ $2^i \mod M$
○ $2^{i^2} \mod M$

Correct

What is the runtime of this algorithm?

○ $\Theta(n)$
○ $\Theta(2^n)$
● $\Theta(\log n)$

Correct

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

$B \leftarrow$ array indexed with $0, \ldots, \lfloor \log n \rfloor$
$B[0] \leftarrow 2$
for $i = 1, \ldots, \lfloor \log n \rfloor$ **do**
  $B[i] \leftarrow (B[i-1] \times B[i-1]) \mod M$
Let the binary representation of n be $(x_{\lfloor \log n \rfloor} x_{\lfloor \log n \rfloor - 1} \cdots x_0)$.
$R \leftarrow 1$
for $i = 0, \ldots, \lfloor \log n \rfloor$ **do**
  if $x_i = 1$ **then**
    $R \leftarrow (R \times B[i]) \mod M$
return R

What is the runtime of this algorithm?

○ $\Theta(n)$
○ $\Theta(2^n)$
● $\Theta(\log n)$

Correct

Remark: A clever algorithm inspired by the above can compute 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.

$$\begin{bmatrix} \text{Fibonacci}(n) \\ \text{Fibonacci}(n-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \text{Fibonacci}(n-1) \\ \text{Fibonacci}(n-2) \end{bmatrix}$$

2 Shorest 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 red-black trees (and not Fibonacci heaps).

If $m = n^{1.5}$, and we want to find the shortest path between some u and v which algorithm should we use? We prefer algorithms with the smallest worst-case runtime.

○ Dijkstra
● Bellman-Ford
○ Floyd-Warshall
○ Two or more of the above algorithms are correct and have the smallest worst-case runtime.

Correct

What if all the edges have nonnegative weight?

● Dijkstra
○ Bellman-Ford
○ Floyd-Warshall
○ Two or more of the above algorithms are correct and have the smallest worst-case runtime.

Correct

Suppose that we have a graph with $m = n^{1.5}$ edges that all have nonnegative weights. Which algorithm should we use to find the shortest path between all pairs of vertices?

○ n^2 runs of Dijkstra
● n runs of Dijkstra
○ n^2 runs of Bellman-Ford
○ n runs of Bellman-Ford
○ Floyd-Warshall
○ Two or more of the above algorithms are correct and have the smallest worst-case runtime.

Correct

Suppose that we have a graph with $m = \Theta(n^2)$ edges that all have nonnegative weights. Which algorithm should we use to find the shortest path between all pairs of vertices?

○ n^2 runs of Dijkstra
○ n runs of Dijkstra
○ n^2 runs of Bellman-Ford
○ n runs of Bellman-Ford
● Floyd-Warshall
○ Two or more of the above algorithms are correct and have the smallest worst-case runtime.

Correct