Binary Sequence Problem

Frank Liu

August 29, 2023

Problem.
Design and justify an algorithm that takes two natural numbers n and k and outputs $M_k(n)$, the number of possible binary strings of length n where the 1's must be no closer than k apart.

Dynamic Programming Approach
First we'll consider a DP, an array with the following recurrence

$$DP[i] = \begin{cases} i & \text{if, } i \leq k \\ DP[i-1] + DP[i-k-1] & \text{otherwise} \end{cases}$$

Algorithm 1 Returns $M_k(n)$ given $n, k \in \mathbb{N}$

Require: $n, k \in \mathbb{N}$
  VALIDSTRINGS(n, k) :
  if $n \leq k$ **then**
    return n;
  end if
  $DP \leftarrow \text{int}[n]$

  for $i \leftarrow 1$ **to** k **do**
    $DP[i] = i + 1$
  end for

  for $i \leftarrow k + 1$ **to** n **do**
    $DP[i] = [i - 1] + D[i - k - 1]$
  end for

  return $DP[n]$

Correctness

Claim: $DP[i] = M_k(i), \forall i \in \mathbb{N}$.

Proof. Suppose $k \in \mathbb{N}$
Base Cases:
When $0 < i \leq k$, $DP[i] = i + 1 = \binom{i}{1} + 1 = M_k(i)$
Note that this is because there is at most a single 1, so we consider the number possible permutations with a single 1, $\binom{i}{1}$, as well as the all 0 string.

Induction Hypothesis:
Suppose that $\forall n < i$, $DP[n] = M_k(n)$, $i > k$

Induction Step:
Let us consider the two cases when we have a string of length i:

1. The last number is 1, this tells us that the next k last numbers must 0. Thus the number of possible binary strings that end with 1 is the same as $M_k(i - k - 1)$

2. The last number is 0, this tells us that the first $i - 1$ elements can be any binary string of length $i - 1$ that satisfies the spacing conditions or $M_k(i - 1)$.

Thus $M_k(i) = M_k(i - 1) + M_k(i - k - 1)$, since these two possibilities have no overlap and their union is all possible binary strings. Thus by our hypothesis:

$$DP[i] = DP[i - 1] + DP[i - k - 1] = M_k(i - 1) + M_k(i - k - 1) = M_k(i)$$

Thus we conclude that $DP[n] = M_k(n), \forall n \in \mathbb{N}$ ◁

The correctness of our algorithm follows.

Runtime Analysis

1. The if statement runs in $O(1)$ time

2. Allocating space for an array of size n takes $O(1)$ time

3. The first for loop runs k times with each iteration taking $O(1)$ time

4. The second for loop runs $n - (k + 1)$ times with each iteration taking $O(1)$ time. Note that the for loops run only if $n > k$, thus the runtime of our algorithm $T(n)$ given an input of n and k is:

$$T(n, k) = O(1) + O(1) + O(k) + O(n - (k + 1)) = O(n)$$

Analytic Solution From part 1 we saw that $M_k(n)$ follows the following recurrence:

$$M_k(n) = \begin{cases} n & \text{if, } n \leq k \\ M_k(n-1) + M_k(n-k-1) & \text{otherwise} \end{cases}$$

Calculating $M_k(n)$ when $k \geq n$ takes constant time so let $k < n$. Let us consider the following matrix representation of the recurrence:

$$\mathfrak{M}_{n+1} = \begin{pmatrix} M_k(n+1) \\ M_k(n) \\ M_k(n-1) \\ \vdots \\ M_k(n+2-k) \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 0 & \dots & 0 & 1 \\ & & & & 0 \\ & I_{k-1} & & & \vdots \\ & & & & 0 \end{pmatrix}}_{k \times k \text{ Matrix}} \begin{pmatrix} M_k(n) \\ M_k(n-1) \\ M_k(n-2) \\ \vdots \\ M_k(n+1-k) \end{pmatrix} = B_k \mathfrak{M}_n$$

Thus we can see that $\mathfrak{M}_n = B_k^{n-k} \mathfrak{M}_k$. Where:

$$\mathfrak{M}_k = \begin{pmatrix} k+1 \\ k \\ k-1 \\ \vdots \\ 2 \end{pmatrix}$$

Using the naive definition of Matrix multiplication of $k \times k$ matrices takes $O(k^3)$. We know that we can reduce exponentiation problems to run in $O(\log_2(n)M)$, where M is the runtime at each multiplication. Thus our total runtime is:

$$T(n,k) = O(\log_2(n)k^3)$$

Which is much more efficient than our first algorithm when n is large.
(When $k^3 < \frac{2^n}{n}$).