

## Question 1

## Using prime numbers for 2-D pattern matching

Some Results that we know are:

$$\bullet M(\varepsilon) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad (i)$$

$$\bullet M(0) = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \quad (ii)$$

$$\bullet M(1) = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \quad (iii)$$

$$\bullet \text{For non-empty strings } x \text{ and } y, M(xy) = M(x)M(y) \quad (iv)$$

$$\bullet M(x) \text{ is defined for all } x \in \{0, 1\}^* \quad (v)$$

$$\bullet M(x) = M(y) \Rightarrow x = y \quad (vi)$$

(a)

Note that  $M(z)$  a string  $z$  of length  $k$  can be calculated recursively as:

---

**Algorithm 1:**


---

```

1 def compute_M(z, k):
2     if(k == 1)
3         if(z[0] == 0)
4             return M(0)
5         else
6             return M(1)
7     else
8         return M(z[0])*compute_M(z+1, k-1)
```

Time Complexity of above algorithm =  $O(k)$ . So now we have a method to compute fingerprint in form of  $2 \times 2$  matrix, in  $O(k)$  where  $k$  is length of string. First we compute fingerprint of pattern in  $O(m)$  time, and then we compute fingerprint for all continuous strings of length  $m$  ( $n - m + 1$  such strings) in text. and then we compare these  $n - m + 1$  fingerprints with fingerprint of pattern. By result (vi) we know that if fingerprint are equal, then so is the actual string. Now we need to find an efficient way to compute fingerprint of all the strings of length  $m$  in text.

**Computation of M for all string in text of length  $m$** 

let text  $(t) = t_1 t_2 \dots t_n$ ,  $M_i$  be the fingerprint( $M$ ) of  $t[i, m + i - 1]$ . Now how can we compute  $M_{i+1}$  efficiently from  $M_i$ ? Note that  $M_{i+1}$  is fingerprint of  $t[i + 1, m + i] = t_{i+1} t_{i+2} \dots t_{m+i}$ , and  $t[i, m + i - 1] = t_i t_{i+1} \dots t_{m+i-1}$ .

$$M_{i+1} = M(t_{i+1})M(t_{i+2}) \dots M(t_{m+i})$$

$$M_i = M(t_i)M(t_{i+1}) \dots M(t_{m+i-1})$$

hence we get:

$$M_{i+1} = M(t_i)^{-1} * M_i * M(t_{m+i})$$

Since any operation involving aithematics is assumed to be in  $O(1)$ , the above computation can be done in constant time, with memoization. When we have the fingerprint for all  $M_i$ , we just compare them with fingerprint of pattern, and report if we find a collision.

**Algorithm 2:**


---

```

1 T := text of length n
2 P := pattern of length m
3 M[1:n-m+1] := list to store fingerprints of T
4 N[0:1] := list to store fingerprint of base case
5
6 N[0] = {1, 0, 1, 1} # base case
7 N[1] = {1, 1, 0, 1} # base case
8
9 M_P := compute_M(P, m)
10 M[1] = compute_M(T, m)
11
12 for i in (2, n-m+1)
13     M[i] = matrix_inverse(N[ T[ i - 1 ] ]) * M[i - 1] * N[ T[ m + i - 1 ] ]
14
15 for i in (1, n-m+1)
16     if (M[i] = M_P)
17         return i

```

**Time Complexity:**

Note that `compute_M` takes  $O(k)$  time for a string of length  $k$ . So computation of  $M_P$  would take  $O(m)$  time, similar to this computation of  $M[1]$  would also take  $O(m)$  time. Since computation of  $M[i]$  from  $M[i - 1]$  takes  $O(1)$  time, time complexity of computing  $M[i]$  is  $O(n - m)$ , and in the end we just do a  $O(1)$  comparison  $n - m + 1$  time.

$$T_{n,m} = O(m) + O(m) + O(n - m) + O(n - m + 1)$$

$$T_{n,m} = O(2n + 1) \approx O(n + m) \text{ (since } m < n \text{)}$$

**(b)**

Let  $M(P)$  be the fingerprint of pattern. The only case when the algorithm in (b) will give wrong answer, is when the pattern and string don't match, but  $M_i = M(P)$ . Or  $M_i - M(P) = 0 \pmod p$ , which means that all the 4 entries of the matrix  $M_i - M(P) = M(E)$  are zero modulo  $p$ .

So the event when error will occur is defined as:  $M(E)_{11}, M(E)_{12}, M(E)_{21}, M(E)_{22} = 0 \pmod p$  and  $P, T[i:i+m-1]$  don't match.

Let  $P(\delta)$  denote the error probability (note that below equalities are given  $\pmod p$ )

$$P(\delta) = P(M(E)_{11} = 0 \cap M(E)_{12} = 0 \cap M(E)_{21} = 0 \cap M(E)_{22} = 0)$$

$$P(\delta) \leq P(M(E)_{11} = 0) \text{ since } \delta \text{ is a subset of this event}$$

Let  $\pi(x)$  denotes the no of primes less than or equal to  $x$ .

Let  $X(a)$  be the no of primes which divide  $a$ .

Clearly,

$$a \geq 2^{X(a)}$$

Thus,

$$X(a) \leq \log(a)$$



**Lemma 1:** Probability that  $A \bmod p = 0$  when  $p$  is a random prime no from  $(2, t)$  is less than or equal to  $\frac{\log(A)}{\pi(t)}$ .

*Proof.* Since  $p$  is selected randomly uniformly from all primes from  $(2, t)$ ,

$$P(A \bmod p = 0) = \frac{X(A)}{\pi(t)}$$

Thus,

$$P(A \bmod p = 0) = \frac{\log(A)}{\pi(t)}$$

□

Note that  $F_n < 2^n$  where  $F_n$  is  $n^{th}$  fibonacci number, also the terms of  $M(Z)$  are bounded by  $F_n$  (given in problem).  
we know:

$$M(Z)_{11} < 2^n$$

By above lemma we know that:

$$P(\delta) \leq \frac{\log(2^n)}{\pi(t)}$$

$$P(\delta) \leq \frac{n}{\pi(t)}$$

Now,

$$\pi(t) \approx \frac{t}{\log(t)}$$

Thus,

$$P(\delta) \leq \frac{n * \log(t)}{t}$$

Take  $t > 5 * n^5 \log(n)$ . Then,

$$P(\delta) \leq n^{-4}$$

Thus for  $t > 5 * n^5 \log(n)$ , probability that fingerprint will match, provided actual string don't match, would be less than  $n^{-4}$ .

### (c)

In this problem we use the fingerprinting approach, we compute fingerprint of matrices of size  $m \times m$ , same way as we did above. First we flatten the 2D matrix into a 1D matrix of size  $m^2$ , by appending  $2^{nd}$  row after  $1^{st}$  row, and so on. This computation for each matrix in text of size  $m \times m$ , needs to be computed efficiently, to achieve  $O(n^2)$  Complexity.

Note that here also our fingerprint is a 2D matrix of size  $2 \times 2$ .

---

#### Algorithm 1:

---

- 1  $T := n \times n$  matrix in which we need to search for pattern
- 2  $P :=$  pattern (dimension  $m \times m$ )
- 3  $p :=$  prime number choosen uniformly and randomly from  $(2, t)$
- 4  $N[0:1] :=$  list to store fingerprint of base case
- 5
- 6  $N[0] = \{1, 0, 1, 1\}$  # base case
- 7  $N[1] = \{1, 1, 0, 1\}$  # base case
- 8
- 9 # first we compute the fingerprint of all continuous strings of length  $m$  in
- 10 # each row ( $n-m+1$  such string in each row)
- 11  $F :=$  2D list to store above fingerprint

```

12 # F[i][j] stores fingerprint of string in ith row starting at index j
13
14 for i in (1,n)
15     F[i][1] = compute_M(T[i], m) mod p
16 for i in (1,n)
17     for j in (2,n-m+1)
18         F[i][j] = matrix_inverse(N[T[i][j-1]]) * F[i][j-1] * N[T[i][m+j-1]] mod p
19
20 # store fingerprint of pattern
21 P_new = flattened(P)
22 M_P = compute_M(P_new, m*m) mod p
23
24 # Now we compute fingerprint of all continuous matrices of size m x m using F[i][j]
25 G := 2D list to store this fingerprint
26
27 # G[i][j] denote fingerprint of matrix with left top at i,j and right bottom at i+m-1, j+m-1
28 # Since fingerprint is calculated by flattening, hence G[i][j] = F[i][j]*F[i+1][j]*...F[i+m-1][j]
29
30 # we do a column wise traversal
31 for j in (1, n-m+1)
32     G[1][j] = F[1][j]*F[2][j]*...F[m][j] mod p
33 for j in (1, n-m+1)
34     for i in (2, n-m+1)
35         G[i][j] = matrix_inverse(F[i-1][j]) * G[i-1][j] * F[i+m-1][j] mod p
36
37 # fingerprint matching
38 for i in (1, n-m+1)
39     for j in (1, n-m+1)
40         if (G[i][j] = M_P)
41             return (i,j)

```

### Time Complexity

Computing matrix  $F[i][1]$  for all  $i \in (1, n)$  would take  $O(m) * n$  time. Computing rest of the entries of  $F$  would take  $O(1)$  time per entry, so a total of  $O(n * (n - m))$ . Calculation of  $M$  for  $P$  would take  $O(m^2)$  time. Computing  $G[1][j]$  for all  $j \in (1, n - m + 1)$  would take  $O(m) * (n - m + 1)$  time. And lastly computation of remaining entries of  $G$  would take  $O(1)$ , for each entry, so total  $O((n - m) * (n - m + 1))$ . So in total we get

$$T_{n,m} = O(mn) + O(n(n - m)) + O(m^2) + O(m * (n - m + 1)) + O((n - m) * (n - m + 1))$$

Or,

$$T_{n,m} = O(2n^2 + m^2 - mn + n - 2m) \approx O(n^2) \text{ since } m < n$$

### Error Probability

Error analysis is similar to previous problem, the only thing that changes is that now  $M[i][j]_{11}$  would be bounded by  $F_n^2$ , since we are flattening 2D square matrix.

Note that  $F_n < 2^n$  where  $F_n$  is  $n^{th}$  fibonacci number, also the terms of  $M[i][j]$  are bounded by  $F_n^2$ . we know:

$$M(Z)_{11} < 2^{2n}$$

By above lemma we know that:

$$P(\delta) \leq \frac{\log(2^{2n})}{\pi(t)}$$

$$P(\delta) \leq \frac{2n}{\pi(t)}$$

Now,

$$\pi(t) \approx \frac{t}{\log(t)}$$

Thus,

$$P(\delta) \leq \frac{2n * \log(t)}{t}$$

Take  $t > 10 * n^5 \log(n)$ . Then,

$$P(\delta) \leq n^{-4}$$

Thus for  $t > 10 * n^5 \log(n)$ , probability that fingerprint will match, provided actual matrices don't match, would be less than  $n^{-4}$ .