2-Dimentional Pattern Matching

What is given:

a. $M(\epsilon) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ $\quad M(0) = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$ $\quad M(1) = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$
b. $M(xy) = M(x) \times \mathrm{M(y)}$; where x and y are non-empty strings
c. $M(x)$ is well defined for $x \in \{0,1\}^*$
d. $M(x) = M(y) \Rightarrow x = y$

First, we chalk out a way to compute the finger-print of a bit-string efficiently, by definition, the finger print of a substring from x of length m beginning from index i, denoted by $x_{i,m}$ would be:

$$M(x_{i,m}) = M(b_i) \cdot M(b_{i+1}) \cdots M(b_{i+m-1}) \quad \rightarrow \; eq(0)$$

Similarly, at index $i + 1$, the fingerprint of x_{i+1} is:

$$M(x_{i+1,m}) = M(b_{i+1}) \cdot M(b_{i+2}) \cdots\cdot M(b_{i+m})$$

We can see that $M(x_{i+1,m})$ can be obtained from the precomputed $M(x_{i,m})$ by using the relation:

$$M(x_{i+1,m}) = M(b_i)^{-1} \cdot M(x_{i,m}) \cdot M(b_{i+m}) \qquad \rightarrow \; eq(1)$$

Since $b_i \in \{0,1\}$ we can precompute $M(0)^{-1}$ and $M(1)^{-1}$. Note that for the first m bits the fingerprint $M(x_{0,m})$ will take about $0(m)$ time assuming word-ram model.

1. Assuming that any arithmetic operation involving two numbers of arbitrary length can be executed in $0(1)$ time. We can state that $M(x_{i+1,m})$ can be computed from $M(x_{i,m})$ in $O(1)$ time since it requires a constant number of multiplications and additions.

   Taking advantage of the above consequence, we can conjure the following algorithm:

   a. We will compute the fingerprint as stated in $eq(1)$
   b. Precompute the inverses, fingerprints $M(P_{0,m})$ and $M(T_{0,m})$.

<u>Algorithm-1:</u>

$T \rightarrow$ a bit string of length n
$P \rightarrow$ a pattern of length m

Precompute $M(0)^{-1}, M(1)^{-1}$
Compute $M(P_{0,m})$ // takes $O(m)$ time
Compute $M(T_{0,m})$ // takes $O(m)$ time

for i in 0 to $n - m + 1$:
  if $i > 0$:
    $M(x_{i,m}) = M(b_{i-1})^{-1} \cdot M(x_{i-1,m}) \cdot M(b_{i+m-1})$ // takes $O(1)$ time
  if $M(P_{0,m}) = M(x_{i,m})$: // takes $O(1)$ time
    return i

Analysis of time-complexity:

The computation of $M(P_{0,m})$ and $M(T_{0,m})$ both take $0(m)$ time since we will loop over all the m bits according to $eq(0)$ and for every matrix multiplication we only take constant time. We cannot use $eq(1)$ for computation at $i = 0$ since $M(T_{-1,m})$ and $M(P_{-1,m})$ do not exist.

Due to the word-ram model assumption, computation of $M(T_{i,m})$ takes $O(1)$ time for all $i > 0$. Note that the loop runs from $i = 0$ to $i = n - m - 1$ in the worst case (when pattern is not in T), and for every iteration takes constant time. Thus, the loop can be bounded by $O(n - m)$

$$t(n, m) = 2 \cdot O(m) + O(n - m) + constant = O(n + m)$$

Thus algorithm-1 takes $O(n + m)$ time under the stated assumption.

2. Given that the elements of $M(x)$ are bounded by F_n, where F_n is the n^{th} Fibonacci number and x is a bit-string of length n. We will use the following bound on the n^{th} Fibonacci number where ϕ is the golden ratio. And since $\phi < 2$:

$$\phi^{n-2} \leq F_n \leq \phi^{n-1} < 2^n$$

Let p be a prime number in range $[0, t]$. The number of primes less than t is $\pi(t) \cong \frac{t}{\log t}$
And the number of prime factors of n is $\leq \log n$.

We know that an error occurs only if $\left(M(T_{i,m}) - M(P)\right) mod\ p = 0$, but $T_{i,m} \neq P$.

This is possible only if p divides all the entries of $M(T_{i,m}) - M(P)$. Let the entries of $M(T_{i,m}) - M(P)$ be denoted as M_{ij}, and E_{ij} be the event that $M_{ij}\ mod\ p = 0$. Then probability that an error has occurred is:

$$P(error) = P(E_{11} \cap E_{12} \cap E_{21} \cap E_{22}) \leq P(E_{ij})$$

Now, since E_{ij} is also bounded by F_n, $E_{ij} \leq F_n < 2^n$.
Hence,

$$P(error) \leq P(E_{ij}) \leq \frac{\log F_n}{\pi(t)}$$

$$P(error) < \frac{\log 2^n}{\pi(t)} = \frac{n}{\pi(t)}$$

Taking $t > 6n^5 \log n$:

$$\pi(t) \cong \frac{t}{\log t} \cong \frac{6n^5 \log n}{\log 6 + 5 \log n + \log \log n} > \frac{6n^5 \log n}{6 \log n} = n^5$$

$$P(error) < \frac{n}{\pi(t)} < \frac{n}{n^5} = n^{-4}$$

$$P(error) < n^{-4}$$

Choosing prime number from the range $[0, t]$ for some $t > 6n^5 \log n$ gives an error probability bounded by n^{-4}.

However, a better bound would be $t > 6 \log_2 \phi\ n^5 \log_2 n$

$$P(error) \leq \frac{\log F_n}{\pi(t)} \leq \frac{(n-1) \log \phi \cdot \log t}{t} < n^{-4}$$

3. We are given an $n \times n$ text matrix T and a $m \times m$ pattern matrix P. To find a matching submatrix in $O(n^2)$ time, we need to be able to compute the fingerprint of submatrices of size $m \times m$ in the matrix T efficiently in $O(1)$ time.

Define $A_{i,j}$ as a submatrix of shape $m \times m$ in T, with the top-left corner at i, j.
Define S_j^i as a bit-string of length m starting at (i, j) and ending at $(i, j + m - 1)$.
Define $b_{i,j} \in \{0,1\}$ as the bit at (i, j) location in T.

Then the fingerprint of $A_{i,j}$ is $M(A_{i,j}) = M(S_j^i) \cdot M(S_j^{i+1}) \cdots M(S_j^{i+m-1})$.
And the fingerprint of S_j^i is $M(S_j^i) = M(b_{i,j}) \cdot M(b_{i,j+1}) \cdots M(b_{i,j+m-1})$

$M(A_{i,j})$ can be computed cheaply by the following relation:

$$M(A_{i+1,j}) = M(S_j^i)^{-1} \cdot M(A_{i,j}) \cdot M(S_j^{i+m})$$

With the base case $M(A_{0,j}) = \prod_{k=0}^{m-1} M(S_j^k)$

$M(S_j^i)$ can also be computed efficiently by the following relation:

$$M(S_{j+1}^i) = M(b_{i,j})^{-1} \cdot M(S_j^i) \cdot M(b_{ij+m})$$

With the base case $M(S_0^i) = \prod_{k=0}^{m-1} M(b_{i,k})$

In a similar way, the fingerprint of the pattern matrix P is $M(P) = \prod_{i=0}^{m-1}[\prod_{j=0}^{m-1} M(b_{i,j})]$

We can say that a pattern has been found if $M(A_{i,j}) = M(P)$.

Text	Pattern
10001111011111110111	100
01111100010011100010	010
01110010100011010110	100
10100100000100011100	
00011001100101010000	

As in the example on left, $i \in \{1,2,3\}$ and $j = 6$. We find that $M(A_{1,6}) = M(P)$

For the conciseness, in the following algorithm '(…) mod p' should be read as mod over all the operations in the parenthesis.

$M(0)$ and $M(1)$ are the same as that in the 1D case. And we precompute and store the value of $M(0)^{-1}$ and $M(1)^{-1}$. To support $O(n^2)$ running time, we used $O(8n^2)$ space as outlined:

- Matrix X to store $M(S_j^i)$ for the efficient computation of $M(A_{i,j})$ and $M(S_{j+1}^i)$.
- Matrix Y to store $M(A_{i,j})$ for efficient computation of $M(A_{i+1,j})$.

Algorithm-2:

$T \rightarrow n \times n$ matrix of bits
$P \rightarrow m \times m$ matrix of bits denoting the pattern to search
$p \rightarrow$ prime number chosen at random from range $[2, t]$

Compute and store $M_P = \left(\prod_{i=0}^{m-1}\left[\prod_{j=0}^{m-1} M\left(b_{i,j}\right)\right]\right) \mathrm{mod}\ p$
Compute and store $M(0)^{-1}$ and $M(1)^{-1}$

$X[i,j] \rightarrow$ to store all $M\left(S_j^i\right)$

for $i = 0,1,\dots,n-1$:
    $M\left(S_0^i\right) = \left(M\left(b_{i,0}\right) \cdot M\left(b_{i,1}\right) \cdots M\left(b_{i,m-1}\right)\right) \mathrm{mod}\ p$
    $X[i,0] = M\left(S_0^i\right)$

for $j = 1,\dots,n-m$:
    for $i = 0,1,\dots,m-1$:
        $M\left(S_j^i\right) = \left(M\left(b_{i,j-1}\right)^{-1} \cdot M\left(S_{j-1}^i\right) \cdot M\left(b_{i,j+m-1}\right)\right) \mathrm{mod}\ p$
        $X[i,j] = M\left(S_j^i\right)$

$Y[i,j] \rightarrow M\left(A_{i,j}\right)$the fingerprint of a contiguous submatrix of size $m \times m$ with the top-right corner at i,j. $Y[i,j] = \prod_{k=0}^{m-1} X[i+k,j]$

for $j = 0,1,\dots,n-m$:
    $Y[0,j] = \left(\prod_{i=0}^{m-1} X[i,j]\right) \mathrm{mod}\ p$

for $i = 1,2,\dots,n-m$:
    for $j = 0,1,\dots,n-m$:
        $Y[i,j] = \left(X[i-1,j]^{-1} \cdot Y[i-1,j] \cdot X[i+m-1,j]\right) \mathrm{mod}\ p$
        if $(Y[i,j] - M_P)\ mod\ p = 0$:
            return (i,j)

Analysis of Time-complexity:

Computation of $X[i,0] = M\left(S_0^i\right)$ takes $O(m)$ time for all $i = 0,1,\dots,n-1$.
Computation of $X[i,j] = M(S_j^i)$ takes $O(1)$ time $\forall j = 1,..,n-m; i = 0,..,n-1$
Therefore the computation of the matrix X takes $0(n*(n-m-1)+n) + O(mn)$

Computation of $Y[0,j] = M\left(A_{0,j}\right)$ takes $0(m)$ time for $j = 0,1,\dots,n-m$
Computation of $Y[i,j] = M\left(A_{i,j}\right)$ takes $O(1)$ time $\forall i = 1,..,n-m; j = 0,..,n-m$
Hence, computation of the matrix Y takes $O\left((n-m)*(n-m+1) + m(n-m)\right)$

Computation of $M(P)$ takes $O(m^2)$ time.

Note that we exit computing $Y[i,j]$ when we find a possible match, thus:

$$t(n,m) = O(n*(n-m-1)+n) + O(mn) + O\big((n-m)*(n-m+1)+m(n-m)\big) + O(m^2)$$

$$t(n,m) = O(n^2 - nm + nm + n^2 - nm + n - m + m^2)$$
$$= O(2n^2 + m^2 - nm + n - m)$$

$t(n,m) = O(n^2)$ assuming that $n > m$

Analysis of Error-probability:

Similar to the previous 1D example, the elements of the fingerprints will be bounded by Fn. Therefore, denoting the element of the computation $M(A_{i,j}) - M(P)$ as M_{ij}:

$$M_{ij} < F_n \leq \phi^{n-1} < 2^n$$

The algorithm makes an error if $\big(M(A_{i,j}) - M(P)\big)\, mod\ p = 0$, but $A_{i,j} \neq P$.

Let E_{ij} be the event $M_{ij}\ mod\ p = 0$. Then probability that an error has occurred is:

$$P(error) = P(E_{11} \cap E_{12} \cap E_{21} \cap E_{22}) \leq P(E_{ij})$$

Hence,

$$P(error) \leq P(E_{ij}) \leq \frac{\log F_n}{\pi(t)}$$
$$P(error) < \frac{\log 2^n}{\pi(t)} = \frac{n}{\pi(t)}$$

Taking $t > 6n^5 \log n$:

$$\pi(t) \cong \frac{t}{\log t} \cong \frac{6n^5 \log n}{\log 6 + 5\log n + \log\log n} > \frac{6n^5 \log n}{6\log n} = n^5$$

$$P(error) < \frac{n}{\pi(t)} < \frac{n}{n^5} = n^{-4}$$
$$P(error) < n^{-4}$$

Choosing prime number from the range $[0, t]$ for some $t > 6n^5 \log n$ gives an error probability bounded by n^{-4}.

Q2 Making an intelligent guess

Given-

1) $F : \{O, ..., n - 1\} \rightarrow \{O, ..., m - 1\}$ $\quad 0 \leq x, y \leq n - 1,$
2) $F((x + y) \bmod n) = (F(x) + F(y)) \bmod m$
3) The only way we have for evaluating F is to use a lookup table and an evil has changed $1/5^{th}$ of table entries

Solution-

Pick x uniformly from $\{0, . . . , n - 1\}$ and let $y = z - x \bmod n$. Then output the value F(z) as $F(z) = F(x) + F(y) \bmod m$. Note that x and $y = z-x$ are both a uniformly random number in the range $\{0, . . . , n-1\}$, but they are not independent. Thus, we have Pr[F(y) is corrupted] = Pr[F(x) is corrupted] = 1/5. Then by union bound P[error] = P[F(y) is corrupted ∪ F(x) is corrupted] $\leq 2/5$

If we repeat the algorithm k times, take the majority vote (or the first if no majority exists). Then the probability of error requires at least (k-1) of the runs to go wrong. Suppose the probability that the algorithm runs once, on input z is wrong is exactly p_z. We know that $p_z \leq 2/5$. Then the probability that when the algorithm is run k times that at least (k-1) runs are wrong is exactly p_z. We know that $p_z \leq 2/5$. Then the probability that when the algorithm is run k times that at least k-1 runs are wrong is exactly ${}^kC_{k-1}\, p_z^{k-1}\,(1+(1-p_z)^k p_z^{k-1})+\ldots+{}^kC_2 p_z^2$.