CSCI 5454: Algorithms: Homework 2

Ashutosh Gandhi

February 13, 2023

Problem 1

```
procedure KMP_MATCHER($txt, pat, \pi$)
    $n \leftarrow len(txt)$
    $m \leftarrow len(pat)$
    $i \leftarrow -1$    # i is the last index processed in text
    $j \leftarrow -1$    # j is the last index processed in pattern
    $max\_len \leftarrow 0$
    $ans \leftarrow 0$
    while $i < n - 1$ do
        if $pat[j + 1] == txt[i + 1]$ then
            $i \leftarrow i + 1$
            $j \leftarrow j + 1$
        else
            if $j == -1$ then
                $i \leftarrow i + 1$
            else
                $j \leftarrow \pi(j)$
        if $j+1 > max\_len$ then    # j has the number of characters in the pattern that are matched
            $max\_len \leftarrow j + 1$
            $ans \leftarrow i$
        if $j == m - 1$ then    # once the complete pattern has been matched reset j
            $j \leftarrow \pi(j)$
    print(f"Longest prefix ends at index: {ans} and is of length: {max_len}")
```

To check for the longest matching pattern prefix we keep track of the maximum value of j in max_len and the corresponding value of i in ans. j points to the last character in the pattern that is matched and hence at any point, it would longest prefix matched up until that point. Thus storing the maximum value of j would give us the longest prefix of the pattern that matches the text.

Problem 2

2.1 Part A

Firstly, each array is considered to be a polynomial with elements $a_i x^j$ with i being the index and j the value at the index i. Each of the elements is added to get the complete polynomial. Now to find the number of triplets (i_1, i_2, i_3) from (S_1, S_2, S_3) such that $i_1 + i_2 + i_3 = T$, we perform the polynomial multiplication using Fast Fourier Transform (FFT) Algorithm for the 3 arrays. Then the coefficient of the x^T in the result would be the ans. The FFT algorithm takes $O(nlogn)$ time for the multiplication and so the triplet count would also take $O(nlogn)$.

2.2 Part B

Similar to the previous part S_1 and S_2 are assumed to be polynomial with elements $a_i x^j$ with i being the index and j the value at the index i. For S_3 we first find the maximum value j in it, which can in done O(n). Then in S_3 each value is negated and then added with j so that all the powers are positive and can be considered a polynomial. Next, S_1, S_2, and modified S_3 are multiplied using the FFT algorithm. The coefficient of x^j would then give the number of triplets (i_1, i_2, i_3) from (S_1, S_2, S_3) such that $i_1 + i_2 - i_3 = 0$. Since FFT takes $O(nlogn)$ the above algorithm would also take $O(nlogn)$.

Problem 3

$$\begin{aligned}
T(n) &= 2T(n/2) + C_0(nlog(n)) \quad \# \text{ expand} \quad T(n/2) \\
&= 2[2T(n/2^2) + C_0(n/2(log(n/2)))] + C_0(nlogn) \\
&= 2^2T(n/2^2) + C_0(nlog(n) - log(2)) + C_0(nlogn) \\
&= 2^2T(n/2^2) + C_0(nlog(n) - 1) + C_0(nlogn) \\
&= 2^2T(n/2^2) + 2C_0(nlog(n)) - nC_0 \quad \# \text{ expand} \quad T(n/2^2) \\
&= 2^2[2T(n/2^3) + C_0(n/4(log(n/4)))] + 2C_0(nlog(n)) - nC_0 \\
&= 2^3T(n/2^3) + nC_0[log(n) - log(4)] + 2C_0(nlog(n)) - nC_0 \\
&= 2^3T(n/2^3) + C_0(nlog(n)) - 2nC_0 + 2C_0(nlog(n)) - nC_0 \quad \# \; log4 = 2 \\
&= 2^3T(n/2^3) + 3C_0(nlog(n)) - nC_0(1+2) \quad \# \text{ expand} \quad T(n/2^3) \\
&= 2^3[2T(n/2^4) + C_0(n/2^3(log(n/2^3)))] + 3C_0(nlog(n)) - nC_0(1+2) \\
&= 2^4T(n/2^4) + nC_0(log(n) - 3log(2)) + 3C_0(nlog(n)) - nC_0(1+2) \\
&= 2^4T(n/2^4) + 4C_0(nlog(n)) - nC_0(1+2+3)
\end{aligned}$$

For the j^{th} expansion we would get

$$T(n) = 2^jT(n/2^j) + jC_0(nlog(n)) - (j(j+1)/2)nC_0$$

To get $T(1)$ substitute $j = log(n)$, we then get

$$\begin{aligned}
&T(n) = 2^{(log(n))}T(1) + (log(n))C_0(nlog(n)) - (log(n)(log(n)+1)/2)nC_0 \\
&T(n) = nC_1 + C_0n(log(n))^2 - C_0/2(nlog(n))^2 - C_0/2(nlog(n)) \\
&T(n) = nC_1 + (C_0/2)n(log(n))^2 - C_0/2(nlog(n))
\end{aligned}$$

Since $(C_0/2)n(log(n))^2$ is the dominant term we can say that $T(n) = O(n(log(n))^2)$

Next, lets assume $T(n) = \Omega(n(log(n))^2)$ and prove it through induction

For **base case** consider $n = 2$

$$\begin{aligned}
&T(2) = 2T(1) + C_0(2log(2)) \\
&T(2) = 2C_1 + C_0(2log(2)) \\
&T(2) \geq C_0(2log(2)) \quad \# \quad for \quad C_1 \geq 1
\end{aligned}$$

Assume the bound is valid for $n/2$ such that

$$T(n/2) \geq k(n/2)[log(n/2)]^2$$

Now proving for some n

$$\begin{aligned}
T(n) &= 2T(n/2) + C_0(nlogn) \\
&\geq 2[k(n/2)[log(n/2)]^2] + C_0(nlog(n)) \quad \# \quad substitute \quad T(n/2) \\
&\geq kn[log(n) - log(2)]^2 + C_0(nlog(n)) \\
&\geq kn[log(n)^2 + 1 - 2log(n)] + C_0(nlog(n)) \\
&\geq knlog(n)^2 + kn - 2k(nlog(n)) + C_0(nlog(n)) \\
&\geq knlog(n)^2 + kn + \lambda(nlog(n)) \quad \# \quad for \quad \lambda = C_0 - 2k \\
&\geq knlog(n)^2 \quad \# \quad for \quad k\&\lambda \geq 0
\end{aligned}$$

Therefore, we can say $T(n) = \Omega(n(log(n))^2)$
Since $T(n) = \Omega(n(log(n))^2)$ and $T(n) = O(n(log(n))^2)$ hence $T(n) = \Theta(n(log(n))^2)$
Thus proved.