Second Level Algorithms

Prof. Palash Dey

Department of Computer Science and Engineering

Indian Institute of Technology, Kharagpur

Week – 11

Lecture 54

Welcome to the 54th lecture of the second-level algorithm course. In the last lecture, we started studying the string-matching algorithm. We have seen a pseudocode of the KMP string-matching algorithm, assuming that we can compute the prefix function in efficient time. In this lecture, we will continue, analyze the runtime, and also see pseudocode for computing the prefix function, OK? So, let us begin.

So, to finish the pseudocode of the algorithm, let us first write down the pseudocode of the prefix function computation algorithm. So, that function, let us call it compute_prefix_function. So, we first initialize an array called pi where we will store the function values. So, we know $\pi[1]$ is 0 because the only strict prefix of a string of length 1 is the empty string. Now we will The algorithm, on a high level, is very similar to the KMP matcher. So, we will use a partial match in a variable K and read the pattern P one character at a time. So, in the q-th iteration, we will compute the value of $\pi[q]$. $\pi[1]$ we already have computed; we know it is 0.

So, that is why we are starting from q equal to 2 to m. Now, again, we are seeing whether we can extend the partial match we have—a k-size partial. That means the first k characters of the string $P[1,\ldots,q-1]$ match with the last k characters of the string $P[1,\ldots,q-1]$, and we are seeing whether we can extend that matching. So, if we cannot extend the matching, then instead of matching with k, we match with $\pi[k]$, ok? Again, this is very similar to the KMP matcher algorithm. We see: can we extend it? When the while loop exits, either k equals 0 or we can extend it.

So, first, we see if we can extend it. If we can, then we make k equal to $k+1$. Otherwise, k remains $\pi[k]$, and $\pi[q]$ becomes k. So, when the while loop exits, either k equals 0 or we can extend. If we can extend, then we have updated the k value. k is the partial match of the prefix of the first $k+1$ characters of the pattern string P that matches with its suffix.

On the other hand, if k is 0 and we cannot extend, then $\pi[q]$ is 0. So, that is it, and then we return π.

So, this is the compute prefix function. Now, let us do the runtime analysis of the KMP algorithm. For that, let us briefly recall the pseudocode of the KMP matcher that we need. So, KMP So, in the first step, we computed the prefix function and then setting q 2 0 which is the partial match we read one character at a time. So, we are checking if we can extend the partial match if we cannot then we keep updating the partial match. So, if q greater than 0 and we cannot extend the partial match. then instead of q we try with pi q and then if the while when the while loop exits either we can extend or q equal to 0 or both. So, first we are checking whether we can extend or not and then we are checking if we have a match or not that means q is m or not if q is m then we return i minus m that is the shift what is needed otherwise we return -1 encoding the fact that there is no match. our first claim is the following. So, let us first see a high level idea, let us first forget about the time spent on while loop. So, if I ignore the time spent on while loop the run time of the for loop is $O(n)$. So, let me write this way.

So, this fact we will use which we will prove later that the time complexity of compute prefix function is $O(m)$. So, let us use this fact in the running time analysis of KMP matcher and then we will prove this later. So, first observe that the time complexity of the for loop excluding the time complexity of the while loop is $O(n)$. So, in the for loop you exclude the time spent on the while loop and the rest of the operations takes $O(1)$ time the for loop iterates for n iterations.

So, the time complexity of the for loop is $O(n)$, excluding the time complexity of the while loop. Now, we will compute the total time spent on the while loop using some accounting method or aggregate analysis, whatever you can say. So, that is a crucial claim. The total time spent in the while loop is $O(m)$, ok.

So, let us prove it, but once we prove it, you first see that assuming this claim, the time complexity of KMP matcher is $O(m+n)$ So, assuming the claim, the time complexity of the KMP matching algorithm is $O(m+n)$, ok. So, now, let us prove the claim. So, a crucial first observation is in the while loop, q strictly decreases because q is greater than or equal to $\pi[q]$. Since $\pi[q]$ is less than q, the variable q decreases by at least 1. in every iteration of the while loop, ok. q starts with 0 before execution of the for loop and never becomes negative throughout the run of the algorithm.

On the other hand if you look at the pseudo code the value of q can only decrease in the while loop and it can only increase here in the body of if only once. So, the value of q can increase by at most one in every iteration. because there are $n-1$ iterations or let us see n iterations. the value of q increases by at most 1, n times.

The value of q decreases by at most 1 at most n times. ok, but whenever while loop executes body of the while loop executes q value decreases. Therefore, the total number of iterations of the while loop is at most n since every iteration of the by loop decreases the value of q by at least 1.

the value of q decreases by at most by at least 1 this should be at least n times ok. So, this shows that the number of iterations of the while loop is at most n. So, to finish the time complexity analysis of the Knuth-Morris-Pratt algorithm we need to prove this fact that the time complexity of prefix function is $O(m)$. So, now, let us see the prefix function here again we have a for loop which iterates for m iterations and if we ignore the while loop the time complexity of the for loop is $O(m)$. So, this analysis is also very similar to the analysis of KMP measure the time complexity of the for loop

Now we are talking about the compute prefix function method that also has a for loop and has a nested while loop inside. So, the time complexity of the for loop excluding the time spent in while loop is $O(m)$. The for loop iterates for $m-1$ iterations and if we ignore the while loop the rest of the computation takes $O(1)$ time. So, time complexity of the for loop is $O(m)$ excluding the time complexity of while loop and again we will show that the total time spent on the while loop is $O(m)$ again.

So, we claim the total number of iterations of the while loop is $O(m)$. proof again the same kind of proof for KMP matcher we use the variable q. Now, we use the variable K here you see k started with 0 and k never becomes negative. every iteration the value of k increases by at least once and every iteration of the while loop the value of k decreases. So, the proof of let us call it claim 1.

ah shows or proves this proves this claim also by replacing the role of q with k ok. So, this shows that the compute prefix function takes $O(n)$ time hence the runtime of the Knuth-Morris-Platt algorithm is $O(n+m)$. Now, the proof of correctness proof of correctness is quite straight forward and that I leave it as a homework to you. prove the correctness of first you have to prove the correctness of compute prefix function and then the KMP string matching algorithm ok.

So, let us stop here. Thank you.