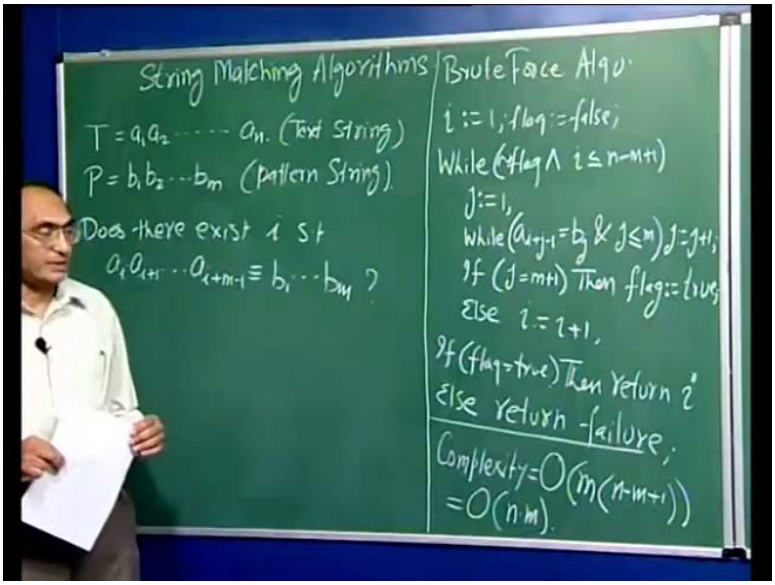
Computer Algorithms - 2
Dr. Shashank K. Mehta
Department of Computer Science and Engineering
Indian Institute of Science, Kanpur

Lecture - 14
Knuth Morris Pratt Algo

So, from today, we will discuss some string matching algorithms.

(Refer Slide Time: 00:22)



So, let us describe the problem. Suppose you are given a string T of say length n. So, we call it a text string and another string P. We will call pattern string; as given b 2, b n. Our goal is and usually n is greater than equal to m. Our problem is to see if there exists some position in this string T, from where continuous m symbols match these symbols. So, we want to know does they exist i, say that a i plus 1 i plus m minus 1 is same as b m. If possible, we would like to compute that i. Now, indeed of course, say if m is greater than n, there is no solution to this.

The trivial solution to this problem is to start matching the pattern string symbol by symbol starting from the first position. If we succeed all the way up to a n then, we have a solution else we repeat the entire process starting at a 2. We continue to do that. If at all we ever succeed then, we do have an affirmative answer to this else there is no way. There is a spring substring of T, which matches with T. So, let us write down the simple or so called brute force algorithm. So, we begin with i at 1. Then, we are going to so let

us say, we will have some flags later on. So, while not of flag; this flag will indicate that we have succeeded in finding such a string. So, initially flag is false. Now, sorry, we have not gone too far on this string. If we reach a point beyond this, there are not enough symbols left.

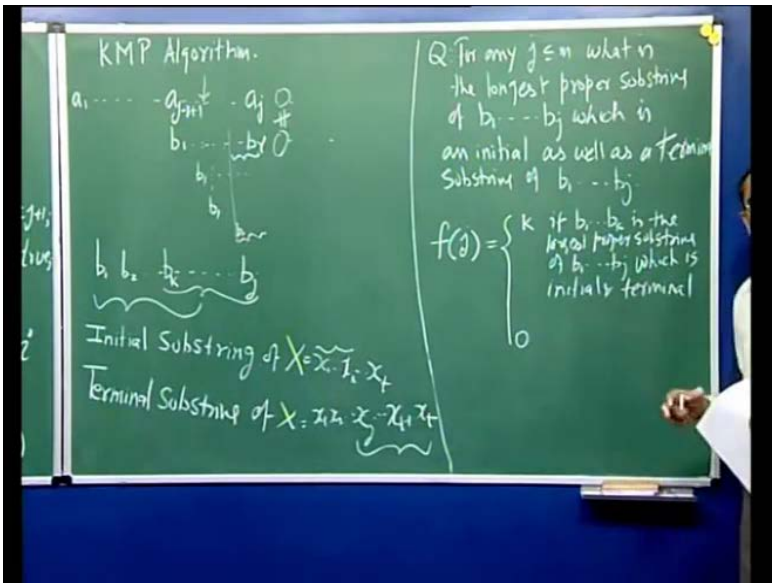
Then, there is no point in matching. So, the last point where we will try to match will be n minus m plus 1. So, i is less than or equal to n minus m plus 1. While both these conditions hold then, there is a need for checking for such a substance. So, what we will do is we will begin with j equal to 1 and try to match a i with b 1 that is b j, a i plus 1 with b 2 and so on. So, this is say while a i plus j minus 1 is b j. b j is still not m. Then, we set j equal to j plus 1; we have succeeded in matching up to j value. We are extending it to j plus 1. Yes, we have to match up to the m eth symbol.

When we come out of this and if j value is m plus 1, then we know we have matched all the m symbols. So, when we come out if j is n plus 1, then we set our flag to true else we have to start a fresh search starting from the next value namely i plus 1. So, we will set our i to be i plus 1. This loop will terminate eventually either with a success that is flag would be true or i would have exceeded this last limit. So, we will say if flag is true then, return i. i is the position from where we found the match else. Now, notice that it is possible that the success was at the very end of the string.

But, if the flag is not true that means we have not found the matching substring. So, we will simply state that a failure. This algorithm in the worst case; suppose there is no such string. There is no place in the text string, where we find these symbols together. Then, we are going to match these m symbols here starting from position 1. That will account for order m. then, again we will match m symbols. Again, we will match m symbols.

So, the total cost the time complexity is going to be order m times the total number of symbols starting, which we are going to do this search which is n minus m plus 1. So, it will be n minus m plus 1. In general, m is significantly larger than n. This is going to be n times m. now, what we will see next is that in this process we have done lot of duplication of work. We will discuss a way to avoid that and improve the performance of the algorithm. Now, this improved algorithm is due to Knuth Morris and Pratt.

(Refer Slide Time: 09:32)



The algorithm is named up to them. Let us try to analyze the situation when we find a theory. Let us suppose we have our text string. Let us suppose at some stage, we started matching b 1 with this and proceeded up to this point. So, first r symbols of the r is less than m. First r symbols of the pattern string do match with these successive symbols of the text string. Here, we will find that there is a mismatch. We find that this is not proceeding any further. At this stage, we will begin matching with the next symbol with b 1. Proceed again trying to find whether this matches with b 1, the next one matches with b 2 and so on whether a j matches with b r minus 1.

Then, we will see whether the next symbol matches with b r. This is what we will be doing. Notice that these symbols are precisely with these because we have already done the matching. So, the question is b 1 equal to b 2 because the next symbol which is b j minus r plus 2 is same as b 2. So, we would be asking is b 1 equal to b 2 is b 2 equal to b 3 and so on. Now, clearly this tests of course; after that we will worry about the next symbol. But at up to this point whatever we want to know is information which is exclusively available by analyzing the pattern string al1.

We do not really need to know what this is. We can very well ask the same question just looking at the pattern string and say if is b 2 equal to b 1 is b 3 equal to b 2 and so, b r equal to b r. Suppose, we find that is not working, then I will go to the next symbol and I will ask, whether b 1 is equal to b 3 and b 2 is equal to b 4 and so 1. I will continue to

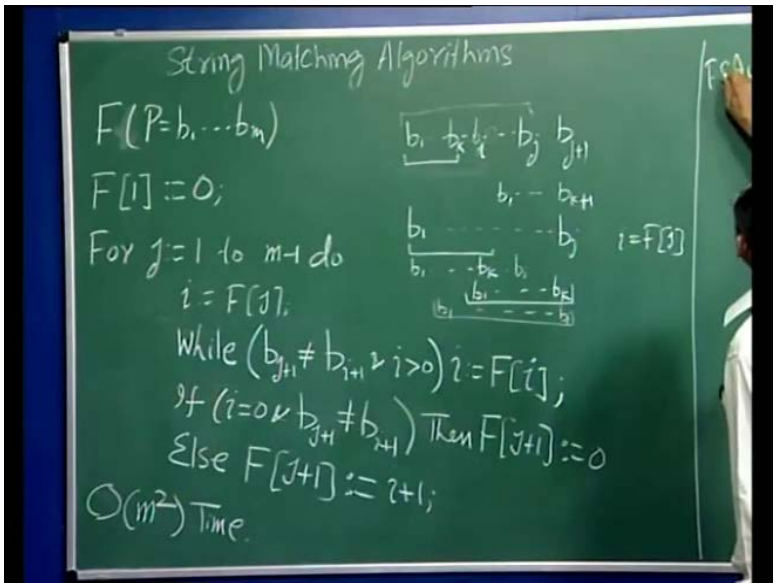
search. Maybe I will find eventually some place where the so certain number of symbols here are same as the same number of symbols at the start of the b string. If we do not find any of these then of course, we will have to move on and start searching from here.

This suggests that we could do certain initial processing before we even begin with searching with text string. This suggests the following computation that is required. What this says is that here is b 1, b 2, b 3 say b j. I would like to know what the earliest symbol say b k is such that this string is same as this string. Isn't that what we want? So, we would like to stay to or left as far as possible and locate a string that is to say as long as a string is possible. It is equal to the last symbols as well as the first so many symbols of the pattern. So here, let us use a term which is called an initial substring.

An initial substring of a given string in a substring or of some string X. X is let us say x 1 through x t; any contiguous string that is to say x 1, x 2, x 3 up to x i. Any contiguous string from 1 to i is called an initial substring of X. Similarly, a terminal substring of X would be a string starting from any x j to x t; this contiguous substring. This will be called a terminal substring of X. Now, what is our objective right now? We would like to find a longest possible initial substring of b 1 through b j, which is also a terminal substring of b 1 through b j.

So, question is for any j, what is the longest proper substring? We do not want the whole thing, which is obviously the case that is not of our interest. We want a string shorter than j. What is a longest proper substring of b 1 through b j, which is an initial as well as a terminal substring of this? So, we will define a simple function f. This would be k if b 1 through b k is the largest proper substring initial and terminal. So, k has to be strictly less than j. If you find nothing there is no string, which is both initial and terminal then, we will output 0. So, what we will do is we will first describe an efficient method to conclude this. Then, we will grow on to device an algorithm based on this information.

(Refer Slide Time: 18:29)



So now, let us device an algorithm to compute the value of the function we have just defined. Let us call this F. We just call it algorithm F. This is of course, a function of the pattern P, which is b 1 through b m, function F. The proper substring of a single symbol string is empty string. So, there is nothing there we are going to build from lower to higher argument of F. So, let us suppose. So, let us first put this in a for j equal to 1 to m minus 1. We have the value of F j. In each path, we want to compute the value of f j plus 1. Now, it is clear that if let us just keep the picture here that we have our b 1 through b j.

This is b j plus 1. They want a longest substring, which is initial and terminal in this. Now, what we have is the longest initial string, which is also terminal of this part. Let us say that string is this big. It goes from b 1 to b k that means F of j is k. Now, if it so happens that b k plus 1 is same as b j plus 1; that is say if F. Well we will say, i equal to F of j. Now, if b of i plus 1 is equal to b j plus 1; let us just call it i. So, if b i plus 1 the next symbol is same as this then, F, j plus 1 will be i plus 1. This symbol next to this; F is indicating the index. So, if you are lucky that the very next symbol matches with the next symbol here.

Then, the longest string, which is initial and terminal for the bigger string b is actually 1 more than the F of j, but if it is not? If this is not the case then, there will be a smaller string hopefully. Whatever that string is say up to this point let us suppose, this is b k such that b 1 through b k plus 1 is also a terminal string. Let us suppose that is the case.

Right. So here, we find b k plus 1 through some b 1. This is a terminal string as well as b k plus 1. b 1 through b k plus 1 is a also the initial string. Now, in that case b 1 through b k is also an initial and terminal string of b 1 through b j. If b 1 through b k plus 1 is initial and terminal string of b 1 through b j plus 1 then, b 1 through b k is initial and terminal substring of b 1 through b j. Now, let us go back and look at b 1 through b i. So, we know that b 1 through b j has this string b 1 through b k.

It is also matching here b 1 through b k. But, we have already a longer string b 1 through b i. It is initial and terminal in b j because my i is equal to F of j. i is F of j. So, there is this up to b i this string and over here, it is b 1 through b i. What we notice is that this b 1 through b k is also initial and terminal substring of b 1 through b i. Notice that this is b 1 through b i and b 1 through b k is an initial substring. This b 1 through b i and b 1 through b k is its terminal substring. So, whatever we will be searching we would also be initial and terminal substring of b 1 through b i. So, what we notice is that we need in our search in this entire space.

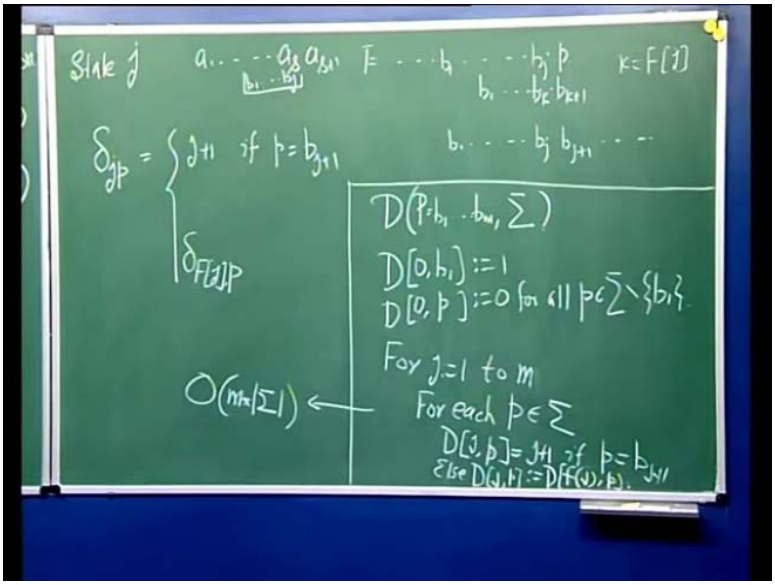
We can restrict our search to b 1 through b i and look for the largest possible string we can locate which is initial and terminal. So, what we will do is if we fail, we would search in the space which is F of i from 1 through F of I. Again, we will try to verify whether the next symbol. Again if we fail then, we will again search in F of F of i. We will continue. If we succeed then, we have guarded otherwise there is no string. So, for that, what we will do is the following. Since, these may take iterations; we will have to put this in a loop. So, let us just say while b j plus 1 is so, here is the final test. We are not succeeding in finding b i plus 1.

If i is still positive, this search strings are gradually shrinking. As long as it is still positive and not succeeding, we will set i to F of i. If we come out of it there are 2 possibilities. Either this condition satisfies or we have shrunk the search space to empty. There is no more to look for something. So, we will then check a condition. If the condition would be that; if i is 0 then obviously, we have not found any string, which would mean F of j plus 1 is 0. This space was from 1 through i that you may verify this first time. Let us just check the first time. This is a f i. It turns out for a given i plus 1 does not match f i.

So, that may happen that we have nothing here. But, the first symbol matches. So, we have to check one more condition. If this is equal and b j plus 1 is not equal to b i plus 1 because maybe the whole string does not match. But, the next symbol matches then, we set F of j plus 1 equal to 0 else F of j plus 1 should be; let us see. If i is F j and we find that. So, it will be this is so; it will be I plus 1. That computes the desired function F. The complexity of this computation is that in the worst case this loop goes all the way.

It may take at most m steps certainly less. But, no more than m steps, this loop also goes through m times. So, this we can compute in order m square times. Now, we are ready with we have done. Some precomputation and we have enough information to do a more efficient search. What we will do is we will device an automaton; a finite state automaton.

(Refer Slide Time: 30:56)



A finite state automaton, in which we will have states long 0, 1 through m. The interpretation will be the following, if the system is in state j. Let us say, we say here a state j. If you are currently at state j and you are searching through the text string up to this; you reach say input will be the text string to this automaton at the entry of this input. If you are in state j, it would mean that the string b 1 through b j. The string, the pattern, initial string of j of the pattern right matches with the last j symbols of the text. Then, you will go to the next symbol. If it turns out to be symbol b j plus 1, then I would like to

make a transition from j to j plus 1. So, let us define our transition function delta j and a symbol let us say we use symbol p; p is a symbol in the symbol set.

This should be j plus 1 if p is same as b j plus 1. We have found matching from b 1 to b j here. If the next symbol comes which is same as symbol at j plus position then, we have got p equal to the new state to be j plus 1. Now, in case p does not match with the next symbol namely b j plus 1 so, we have b 1 through b j, b j plus 1 and so on. Currently, in our text string, we had symbols, which were equal to b 1 through b j and followed by p. Now, indeed I am not going to find this matching with this. So, there should be a smaller spring as should be looking for which matches the terminating point at p. So, suppose we have some other location; let this goes up to some b k plus 1.

In that case, b 1 through b k must be which is already an initial substring of the pattern string; is also a terminal substring of b 1 through b j. But, we know the longest such string is known to us. So, in that case, we will be looking forward through matching the symbol b k plus 1. If we succeed then, we have found the best string the longest possible string initial substring of the pattern which matches up to this point. This is of course, our text string. These are the symbols matching with the text string. But, this problem is exactly the question of where does my automaton jump when it was in state k?

Suppose my k is F of j because there that is the longest string, which is initial and terminal in b 1 to b j. I am trying to see after that if is found then, what is the best thing I can find which matches. So, we can directly say that in this case, you should go to the same state that you would have gone had you been in state F j and you had encountered symbol p. Once again if we compute this from smaller to higher values of j, then, this value would have been already known to us because F of j is strictly smaller than j. So then, that gives us a simple algorithm to compute the transition function.

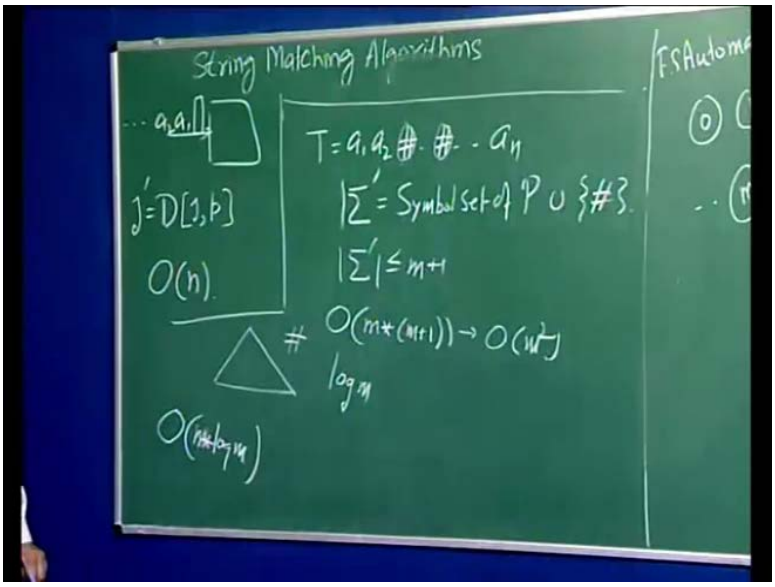
So, we say D. This depends on the pattern string and the symbol set. For every symbol, we have to do this. We will define D 0, b 1 to be equal to 1. Initially, when there is no, nothing matching and the first symbol, we encounter is b 1. Then, we have found this string of length 1, which is initial and initial of p and terminal of that length of p. D for all other symbols p equal to 0 for all p in sigma minus b 1. So, we have now complete description of the transition function starting from state 0. Then, we can put this in a loop put j equal to 1 to m minus 1. Notice that, once we reached at m, which means we have

found a position, where the previous m symbols of the text string match with the first m symbol of p.

But, p is only m symbols to be a found a complete matching. This is where we can stop. So, we need to worry about the transitions from these. But, in case if you wish to continue and search for all occurrences, you can actually compute this for all the way up to m. Then here, you will say if we will capture this. If here we are going to consider for all for each p in sigma, we are going to have D of j, p. If p is b j plus 1 then that would be so. We will say p equal to j plus 1 if p is b j plus 1 else D of j, p should be D of F j, p. D of F j, p. So, this is just capturing this definition of the function delta.

The cost of this computation for each of j we have to stand through each symbol. The complexity of this is order m times mod of sigma the set symbol sigma. You have to keep in mind that this cost and the cost of computing function F, which we have used here, is a onetime cost. Subsequently, for every text string we can use this transition function. So, this does not have to computed more than 1. Now, let us talk about the automaton.

(Refer Slide Time: 41:36)



So, here we have the automaton and we are going to enter the text string. Whenever the system reaches the state m that should be taken as a terminal state. We can output the fact that we have successfully found a match for the string. The cost of computing the transition is unique because we have stored the transition in the array. So, for a symbol p

and if you are currently in state j, you go to this state j prime. Hence, this is a unit cost. So, the entire search will take order n time; be linear in the length of the text stream subsequently.

It has one drawback that is it is dependent on the size of the symbol set. Imagine this is much bigger than m. It is very significantly bigger. It is a closer to n that is to say there are lots of symbols in the text string, which do not occur in the patterns. In this case, you will end up paying heavy cost of computing the transition function. In such extreme situation once again, in case a onetime cost such as this is to be paid, we can pay if we have to repeatedly use the same automaton. So, still it is alright. But if you insist that this should be reduced, then there is one trick you can exploit.

The trick would be that in the text string every symbol set that does not belong to the symbol set of p, so any symbol, so this symbol is not in p. This symbol is not in p and so on. It does not really play any role in the matching. We know that this is as good as a symbol which is alliance to p. So, if I replace this by a single symbol let us say, a new symbol then, we can actually still perform the search. We will still get the same answer. The advantage of this substitution would be that this the new set sigma. The sigma set would be this symbol set of p, symbol set of p union a special symbol.

So, the size of this is nothing m plus at less than equal to n plus 1 because at most m symbols can be present here. That can reduce the cost to m times m plus 1, which is order m square which is also the cost of computing function F. But, there is a price to pay for this. You will have to make these substitutions. So, as you are inputting here, somewhere here, somebody switches the symbol. If it is found to be alliance symbol, not a p symbol; it switches it to that special symbol. What is the cost of doing that? So, in general, if we form a binary such tree storing those m symbols, you run through this.

We, if we do not find it then, we plug in the sharp symbol for it. So, this transition will take log m time for each symbol to detect whether it is a new symbol of p or it is an alliance. So, in that case the cost will rise through m times log n. This is the price we pay. This will be better this, because this if this is huge this is the order of n. Then, we are talking about m times n as against log m times n. So, that is the advantage of such a trick. So, we close this discussion. We will discuss another algorithm for the spring matching in the next lecture.