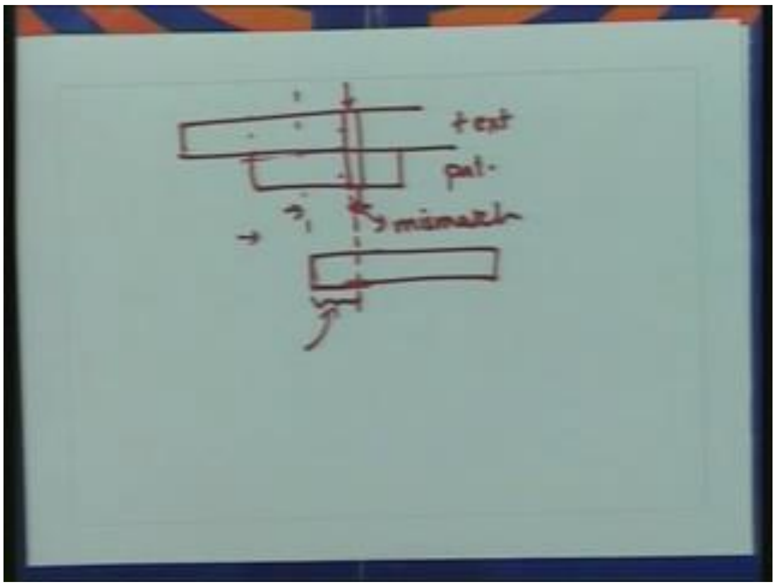
Design and Analysis of Algorithms
Prof. Sunder Vishwanathan
Department of Computer Science Engineering
Indian Institution of Technology, Bombay

Lecture – 15
Pattern Matching – II

We were discussing Pattern Matching in the last lecture. The problem was given a pattern and text. You want to find let us say the first occurrence of this pattern in the text. And we were on our way to finding linear time algorithm for this problem. So, the first thing was that the notice that, what we would like to do on. So, we start matching the pattern to the text and once this mismatch. We would like to shift the pattern. The worst case was algorithm the proof of the algorithm shifted the pattern by 1. And started comparing from the beginning again. We would like to do much better than this.

(Refer Slide Time: 1:52)



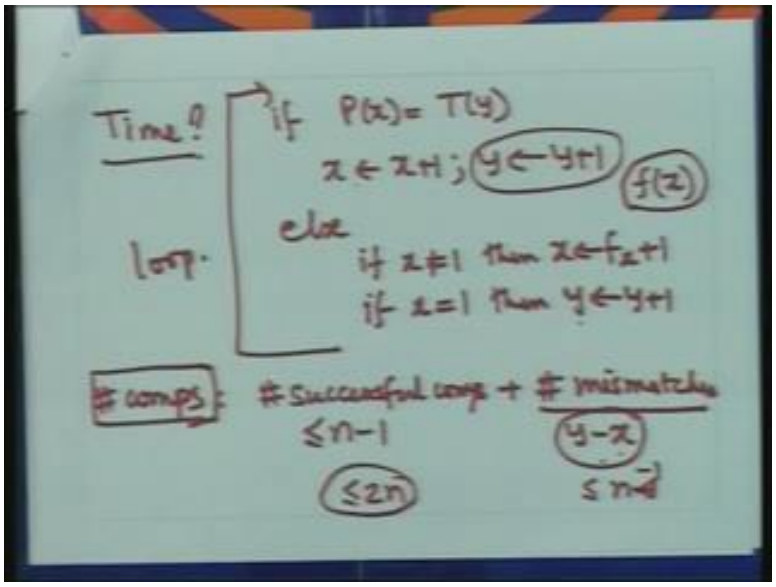
And we will let us quickly go away what we did last time. So, you have the text and you have let us say the pattern and let us say you are comparing this position starting from this position. Now, you compare till you find a mismatch. Let us say that is mismatch and your matches is all over this place, what you like to use is the fact that. The text in this portion exactly equally pattern in this portion. This is the information that we would like to use. And on a mismatch we would like to push the pattern as much as possible.

So, we would like to push the pattern the maximum possible without missing and occurrence of the pattern in the text that is the goal. And for this, we set that what we want to know is for each position in the pattern. So, what is the longest prefix of this, this portion of the pattern which is also suffix of this portion of the pattern. So, this is something that we want to compute; because if there is a mismatch and we shift the pattern across.

Let us, say to this position. Then we notice that up to here. This is the prefix of the pattern which is a suffix of this portion of the pattern. There is a mismatch here. So this portion, if this has to match prefix of the pattern has to be equal to the suffix of the pattern. And since, we do not want to miss out and occurrence of the pattern in the text. We find the maximum search, I mean in the maximum search which means a minimum shift which gives you this.

So, we find the maximum length of a prefix of the pattern, which is also a suffix of this portion of the pattern, the mismatch that occurred here. So, this is what we want to find. And the algorithm was once we determined this. This is a function of the pattern and can be pre-computed. So, once this is computed, then the algorithm was clear, I keep matching. Once I have a mismatch I move the pattern over as much as I can. And I start comparing at the same position starting from the same position on the text. The pattern is of course shifted.

(Refer Slide Time: 04:53)



<u>Time?</u>

loop.

```
if $P(x) = T(y)$
    $x \leftarrow x+1$; $\boxed{y \leftarrow y+1}$        $\boxed{f(x)}$
else
    if $x \neq 1$ then $x \leftarrow f_x + 1$
    if $x = 1$ then $y \leftarrow y+1$
```

$\boxed{\#\text{comps}}$: $\underset{\le n-1}{\#\text{successful comps}} + \underset{\le n-1}{\underset{\boxed{y-x}}{\underline{\#\text{mismatches}}}}$

$\boxed{\le 2n}$

And we argue that this let us look at this algorithm once again. This is from yesterdays notes. So, pointer to the text is y the pointer to the pattern is x. The generic step is this you check if these are equal if there is a match. If there is a match then you go on to the next character. You go on to the next character in the pattern and next character in the text. That is what this portion does x is x plus 1 y is y plus 1.

If there is a mismatch ignore the x equal to 1 part. Well if x equal to 1, then you are the first position in the pattern which means your pattern shift and the text also and the pointer on the text also shift. The pattern shifting does not make any changes, because x remains 1. But, y increases by 1, because now we are going to start comparing the next character in the text.

Now, if x is not equal to 1 that is the crucial stack that we set. Then, this is where you shift the pattern why remains the same. So, you start comparing from the mismatch onwards this is what happens ((Refer Time: 1:52)). This y remains the same, but now the pattern shifts. So, initially x was pointing here now x will point here. This is a new x this is the old x this a new x.

And how do you find this new x, new x is nothing but this length plus 1 and that length we have calculated already and which is f x. So, f is an array then you want to write you know f of x. If f is stored in an array, then you would write want to find a prefix? So, this gives you the longest prefix of the pattern up to x minus 1 which is also suffix of the pattern up to x minus 1, only you need the prefix to be a proper prefix.

So, let us say the prefix of p 1 up to p x minus 2 prefix of this which is the suffix of p 1 up to p x minus 1. So that is what we wanted. So, this moves the pattern up that is what this says. And you start comparing again this is in a loop. The question was what is the number of comparisons we make, because this dictate the order of the algorithm.

Now, this I can write as number of successful comparisons in number of unsuccessful, which mean number of times I take the, if else number of times I take the else close. Well in the, if close we notice that each time, this is a match y moves up by 1. The pointer in the text moves by 1 increases. We also notice that the pointer in the text never moves back it always moves forward.

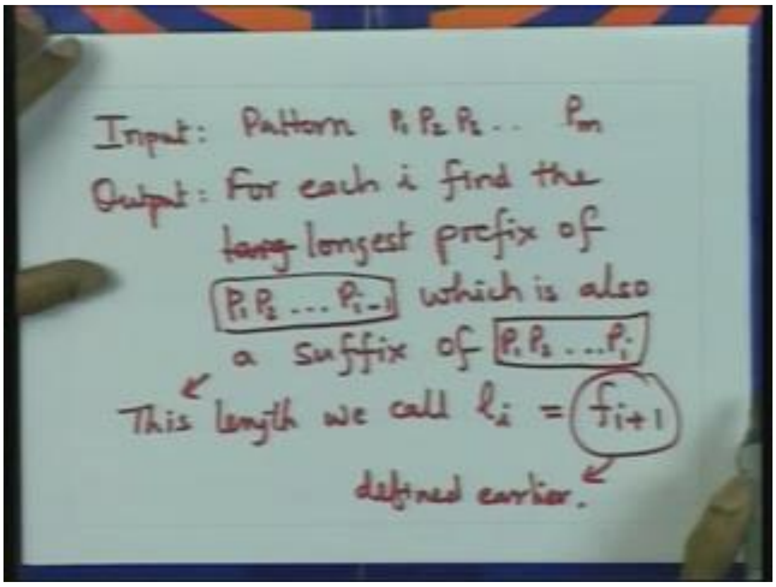
So, the total number of comparisons can be at most n minus 1. Total number of successful comparison can be at most n minus 1, because why increases each time. Now, what about unsuccessful comparisons here? Well again let us look at only x not equal to 1, when x is not equal to 1. Then, the pattern shift the text pointer remain as it is with the pattern shift to the by at least 1 unit.

So, the total number of times a pattern, which shift is at most n minus 1 and well that is why this is also bounded by n minus 1. And y minus x plus 1 actually gives you the position of the head of the pattern. So, if you look at y minus x it points to the position before the pattern. So, x points here y points here, this position is actually y minus x. So, this is giving me y minus x.

So, if I look at y minus x since the pattern shift this will strictly increase. In this case in fact, in both this in that you can see that, this strictly increase. Why increase in the first case the second case. I mean in the first case x decreases in the second case here when x equal to 1 y increases. So, this thing always goes up it never goes down. Even when there is a match it does not go down.

The pattern remains as it is that is what that is what this is. So, this follow initially starts out being 0 and the largest value it can be is n minus 1. So, maybe this is n. So, the largest value is n. Actually, x will be 1. So, this is n minus 1. So, the total number of comparisons is order n, it is let us than equal to 2 n clearly which is what we wanted to show. So once we compute this fx, which is a which is a property of this. If the pattern we can do these pattern matching in linear timing? So, we still left with the problem of computing this function f. So that is something we are to do?

(Refer Slide Time: 10:48)



Input: Pattern $p_1 p_2 p_3 \ldots p_m$

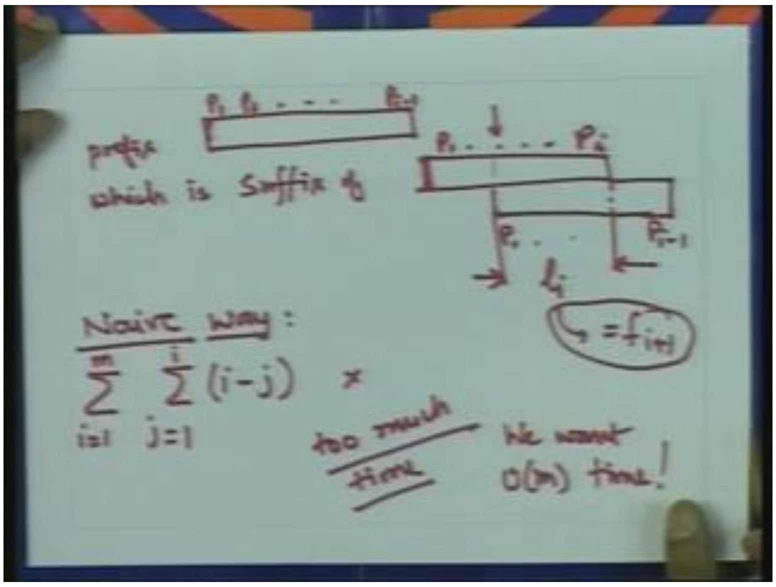
Output: For each i find the ~~long~~ longest prefix of $\boxed{p_1 p_2 \ldots p_{i-1}}$ which is also a suffix of $\boxed{p_1 p_2 \ldots p_i}$

This length we call $l_i = f_{i+1}$

defined earlier.

And we will. in fact, do this do it now. So, what is it that we want? So, we have given so input is the pattern p 1 p 2 p 3 p m. It is m characters long now what we want as output is for each i, find the largest prefix largest or longest maybe longest prefix of p 1 p 2 up to p i minus 1, which is also a suffix p 1 p 2 up to pi. So this length, we call say li. And this is nothing but f of i plus 1 from the previous this is the f which was defined earlier. So, for each i for each position I want to find the longest prefix of this string, which is the sub string of this, which is also a suffix of this.

(Refer Slide Time: 12:42)



Naive way:

$$\sum_{i=1}^{m} \sum_{j=1}^{i} (i-j) \times$$

too much time

We want $O(m)$ time!

So, again pictorially let me draw this. So, I have this is p 1 p 2 up to pi pi minus 1. So, I want the longest prefix which is suffix of p 1 up to pi which means the longest prefix will look like this p 1 this is pi minus 1. So that the minimum, I have to shift, so that there is a match up to this place. This length is what we want and this length is called li it is also equal to f i plus 1 as of the previous.

So, let us focus on li will not look at fi as if not. So, this is what I want shift the pattern a minimum. So that, there is a match here suffix of this string is the prefix of that string this is a prefix and that this is the suffix. This is what we want this is what we want to compute and this is a proof force way of doing this. So, what the naïve way, well I just shifted by 1 and c. If it matches shifted by 2 and c if it matches shifted by 3 and stop the first I am I get the complete match.

If it does not match at all the value is 0, this length is 0. I mean if at no stage you are find the match here; that means, this value is 0. So, the naïve way is for each position up to i, I do these comparisons and it looks like I have to do about for position j I need do i minus j comparisons. So, this will just be sigma i minus j, j going from this is the position j.

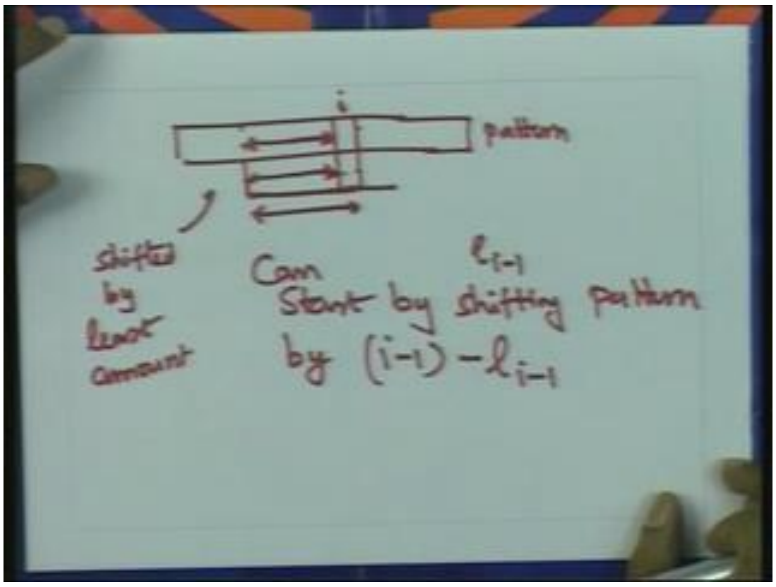
So, this is firstly for i going from 1 to m. For each I, now I need to compare for each i I need to start this off from j, j going from 1 to i it is i minus j. So, this is too much well I let you figure out what this sum is, but this is suddenly not linear in the size of the pattern this is too much time.

So, I guess you can write this it needed, but the time taken by this just too much. It is suddenly not linear in the size of the pattern. So, we want o m time. So, this is not so. Well then, what we do? So, we will put over algorithm design sort principles to use some extend. So, we would like we again. So, this is the problem on arrays roughly it is look like a problem on arrays or less.

So, we will use our favorite inductive kind of approach will use our inductive approach and assuming which means assuming that I have d1 it for you first i minus 1 places, how do I compute the ith value? So, the value of l I have computed let us say up to the first I minus 1 places and I want to compute li, l1 l2 up to li minus 1 is d1. So, how do we do this? The question also is can be use this l 1 l 2 up to li minus 1.

So, let us see, what this is? So, you thing is to keep working at the problem and trying and understanding what it really means sometime thing added. Some point and time thinks just start to click.

(Refer Slide Time: 17:30)



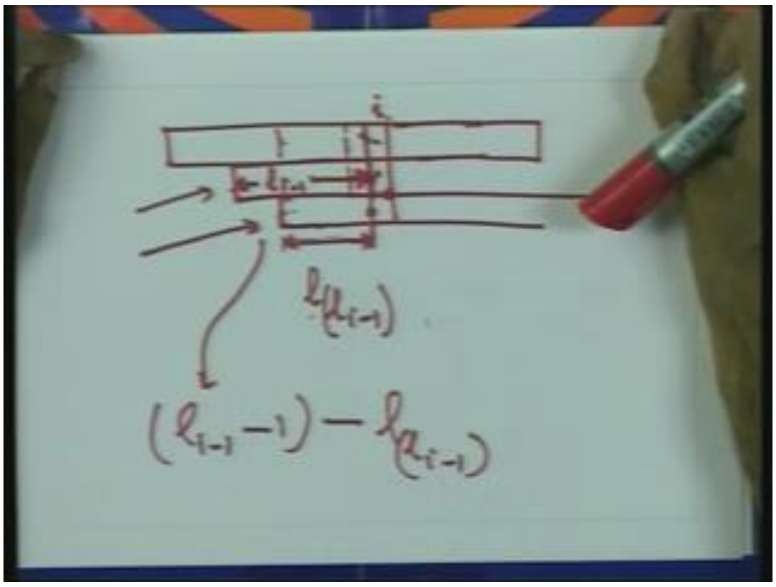
So, here is my pattern this is the ith thing which I want to calculate li is what I want to calculate. So, what I want is shifted by the least amount. So that, it matches algorithm this matches up to here it all matches upwards, what is this is what I want. So, shifted by least amount or the longest prefix whichever way you want to look at it.

So, the first thing to notice is that, when if this portion matches this portion exactly. This portion up to the previous character also matches this portion. If this big portion matches all the way up to i this portion up to the previous character also matches up to this portion which means if I want a prefix to be a suffix up to i. I want to prefix also to be suffix up to i minus 1.

So, I want that to happen, I know this length is at most l of i minus 1. So, I can start of by shifting. So, I can start by shifting pattern by i minus 1 minus l of i minus 1. These have already computed by induction, I have already computed. So, I do not need to start of here I can start of by shifting it by this much and now I start comparing. I know that there is a match up to this place; I only need to compare these two.

If there is a match here also, then I know what li is just li minus 1 plus 1. If there is no match here, then what we do to shift this pattern further. But again, we use we make use of the fact that if calculate the l calculated l values previously.

(Refer Slide Time: 20:13)



So, what happens here? So, here is my pattern. So, here is the same pattern shifted. And this is the ith position, ith position in the top. So, this is something else. Now, this I know is l of i minus 1, this is my first shift day shift this is my first shift. And now I am comparing these two I know that shifting it by any less is useless.

So, shift by this and now I am comparing these two. If there is a match or found li if this does not match. Now what should I do well I need to shift this further. I need to shift this pattern further, why by how much how much do I shifted. Well again we notice that if you shifted more. Then this is the i minus 1 at position, if I look at this pattern. And this I need a match from here. Forget this for the timing focus on these two focus on these two.

Now, I know that these two patterns must match in this portion which means this prefix of the pattern must be a suffix of this part of the portion, this part of the pattern. This index is l of i minus 1. So, this is nothing but l of l of i minus 1. So that is, what this length is. So, let us this is just 1 level of because itself. So let us see what is exactly going on.

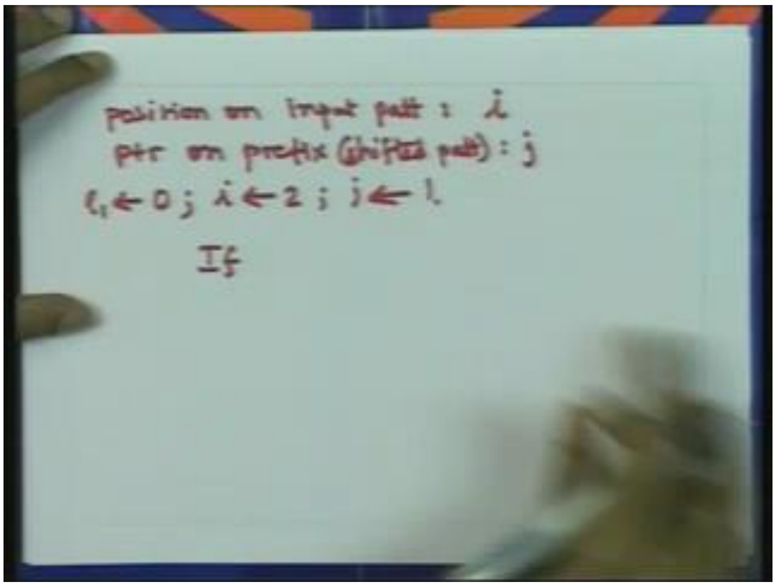
So, I have a mismatch here. Now, I want to shift the pattern further. This is the original pattern now I want to shift this pattern. Supposing this is the correct shift after this, this is the correct shift. Now, we observe that this portion of the pattern must match this portion which is what we want the middle part for the timing. This portion must match this portion correct.

Now, put the middle portion back this portion of the middle part match this portion of the text. And so, must match this portion. So, if I look at only the middle part and the bottom part, this portion of the pattern must match write above it. So, if I omit the top part what is see is, this is a similar phenomenon, what I want, is the longest prefix of this portion which is also suffix.

So, this initially this length was i and I got l of i minus 1. Now, this length is l of i minus 1. So, what I get is l of l of i minus 1. So, essentially I use the l values to shifted further that is what this length is. So, I know what the shift is? So, this shift will be nothing but l of i minus 1 minus 1 minus l of l of i minus 1 that is what I shift. Now, again if I find the mismatch again I do the shift only now I work with this rather than l of i minus 1 and so on.

So, you can see even though it is sounds bit complicated see roughly, how we use the previous values of previous l values, to shift the pattern forward. So, let us write what happens well it could so happen that you keep shifting and you shift all the way to 0. So, it will happen that you shift it all the way across, there is no place to you get any you get complete match which case l of i could well turn out to be 0.

(Refer Slide Time: 24:39)



position on input patt : i

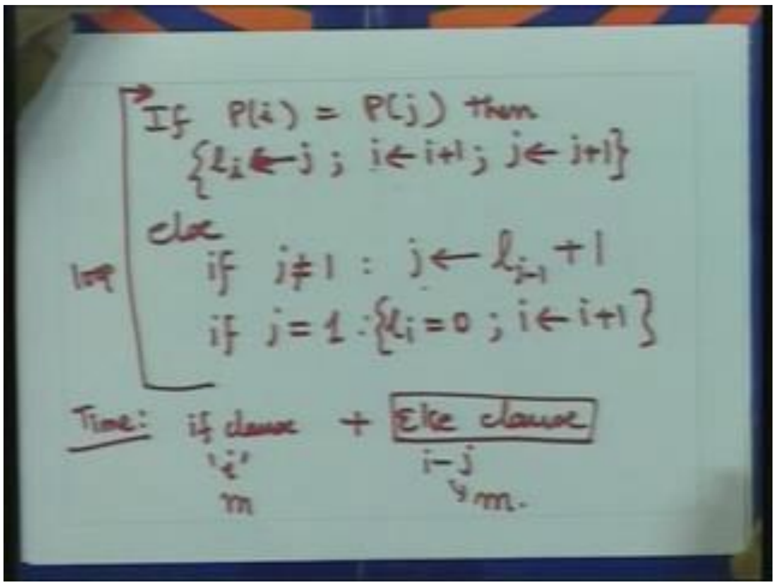
ptr on prefix (shifted patt) : j

$l_1 \leftarrow 0$; $i \leftarrow 2$; $j \leftarrow 1$

If

So, let us write this algorithm down. So, the position on the on input pattern let us say is i. So, li is what I want to fill l 1 l 2 up to li minus 1, I have filled. So, pointer on the prefix, this is the shifted, this I will call j. So, this is my initial pattern that is i and this will be j. So, initially j for instance here is li minus 1 plus 1, then it becomes l of li minus 1 plus 1. So that is what j the shifted pattern. So, now what do you have, so again it is let us see l 1 is 0. Let us do the initialization first l 1 is 0, i as set to 2 j i set to 1, this is my initialization. So, I have to find l 2 l 3 and so on.

(Refer Slide Time: 26:10)



```
loop
    If $P(i) = P(j)$ then
        {$l_i \leftarrow j$ ; $i \leftarrow i+1$ ; $j \leftarrow j+1$}
    else
        if $j \neq 1$ : $j \leftarrow l_{j-1} + 1$
        if $j = 1$ : {$l_i = 0$ ; $i \leftarrow i+1$}
```

Time: if clause + $\boxed{\text{Else clause}}$

'i'   i−j

m   ≤ m.

Now if so here is the procedure, if let us say note down separate things and need the space. If pattern at i equals pattern at j. Then, well here there is a match so both pointers move forward. So also, where was this, if there is a match at any time, then I know what the l value is. If there is a match at any time I know what l value is its nothing, but j. So, li equals j, this is the longest prefix that I can now increment i and j.

Now, I want to find it for the next value of i this is inside if else this is the crucial part. Well again, if j is not equal to 1 which means you are not looking at the first character in the pattern in the shifted pattern then what can we say. Then, we just shift the pattern some more and what is the shifted value it is nothing but l of j minus 1 plus 1. So, this is what this is exactly what we had.

So, let me recall this portion it was l of we will looking at i. (Refer Slide Time: 20:13) So, it is l of i minus 1 plus 1. Similarly again here, it will be l of i minus 1 minus 1 l of that is what we want. So, j will be l of j minus 1 plus 1, this is j is not equal to 1. If j equal to 1 this is an exceptional case. Then, which means we are looking at the first we shifted the pattern all the way.

So that, the first character match is looking is matches is below the ith character of the pattern and this does not match which means li must be 0 li equal to 0 and I can move forward. So, i is i plus 1. So, I calculated li and I move forward. This now goes into a loop this thing goes into a loop and that is it. So, we just keep calculating this way. So, what is the time taken? Well again, we break it into two parts 1 is the, if and else the successful match and unsuccessful comparison.

So, pi equals pj and pi is not equal to pj. So, plus number of times you execute the l close. So, how many times do you execute this well again we notice that here I goes up by 1. So, look at I, i goes up by 1 time which is the pattern you fill out this entry. You filled out 1 entry in the pattern is li set to j and i goes up i never decreases anywhere. So, the number of times, this will happen is m. So, each time this happens i is incremented by 1 the largest value of i is n.

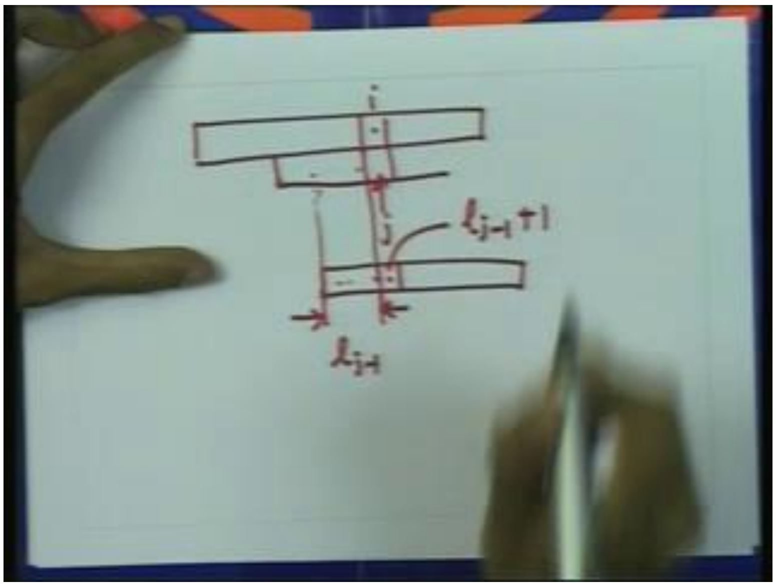
So, this is at most m, how about this else clause well either i is incremented or j is decremented which means pattern below its shifts again it is the same argument. So, I look at i minus j the maximum value, this is always incremented here. It remains the same here; this is always incremented in this case. So, the maximum time in the time taken here is m. So, the total time taken by this procedure is twice m. I just realize that I made will small mistake let me correct that.

So, everything was light except this. So, just focus on this. Here I have I wrote this minus 1 which is not true what I said earlier was perfectly correct. The reason the j value is actually n of I minus 1 plus 1. So, what I have here is l of j minus 1. When I take l of j minus 1, now it becomes l of l of i minus 1, this minus 1. Now, the new j value will be this plus 1 and so on. So, what I earlier said was correct, this length there is no minus 1, here this length is exactly l of i minus 1. I am sorry for this for this mistake.

So, let us go back to the algorithm, we just finished looking at ((Refer Time: 32:59)) y the time is twice m. So, in linear time we have actually computed the l values, well if you want to write l as an array I have written as subscripts. I have this is the way I describe this algorithm you can of course, write l of j minus 1.

If you are storing the l values and array and this is what this, how you write it. I just written it as a subscript. So, if there is a match then I have computed the l value for i and i go ahead and I am going to try and compute the l value for i plus 1. That is what this portion says; this portion says if there is a mismatch. Then, I need to shift the pattern. And how much do, I shifted by, what do I shifted by well, that is why this j comes in j i set to l of j minus 1 plus 1.

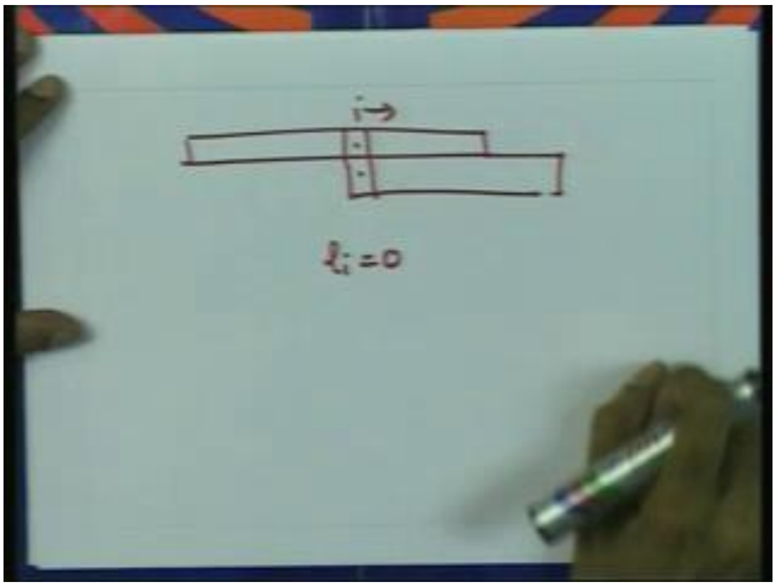
(Refer Slide Time: 33:06)



The reasoning is exactly what we did earlier, because so here is my pattern. The ith position is what I am trying to compare and I have shifted and this is the jth position on this bottom pattern. This is i n that is j and there is a mismatch, how much you are shifted by well, if I shifted by some amount.

So, let us say I shifted by this amount and this is the value then up to here this portion of the pattern matches this portion of the pattern. And I need to shift by minimum. So, that this happens which means this length is nothing but l of j minus 1 this length is l of j minus 1. So, the new j will be here, I am going to compare start comparing this with i. So, this is nothing but l of j minus 1 plus 1 which is exactly what we have. So, if j is not equal to 1 j is l of j minus 1 plus 1. If j is 1 which means I have comparing the first character, if which means it looks like this.

(Refer Slide Time: 34:25)



So, let us do 1 figure. So, what happens when so this is i and the pattern actually looks like this j is 1.Now, if there is a mismatch, then I know there is no prefix which is the suffix till of this portion. So, li equal to 0 and I shift I now move everything over. So, I move i 1 and I move the pattern by 1 and I start comparing from this same position. That is what this is the whole thing is in a loop and you end when you finish scanning the entire pattern. The total time is m, because the number of time this is executed is at most m, number of time this is executed is also at most m. So, the total time is 2 m and in 2 m steps, we can compute this length l.

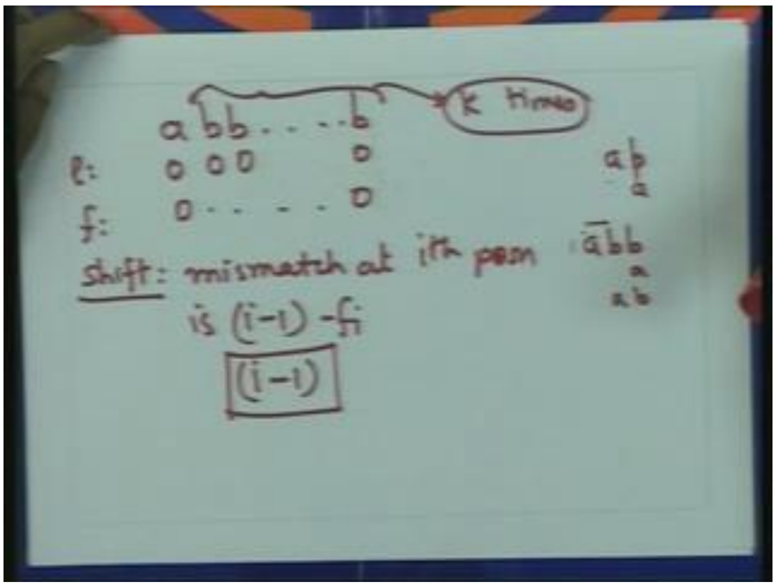
And once we have the length l, we can compute the f that we wanted by just shifting the l values 1 step to the right that is all the risk. Because, fi plus 1 is li and once we have computed f for the pattern. Now, we can scan any text for this pattern using the algorithm that we saw last time. And that also works in linear time.

This algorithm is due to ((Refer Time: 36:57)). This is not actually use in practice an algorithm, even though this is efficient. And algorithm that is found to be more efficient in practice is due to Boyer and Moore and that what is used in ((Refer Time: 36:20). And Boyer Moore, the idea is very similar you want to shift by as much as possible and use the fact that has been a match in some portion of the pattern in text.

The difference between Boyer Moore and this Growth Moore is track is there Boyer Moore starts comparing from the last pattern I mean in the last character of the pattern downwards. Here, you start from the first character of the pattern upwards and Boyer Moore you look at the last character of the pattern downwards that is what they do. But, the concepts are very very similar.

Let us before, we windup this thing. Let us go back and look at those two examples that we looked at write at the beginning. And see what happens to the f function and how this pattern matching algorithm works on that. They were two strings that we worked at we looked at the beginning. So, let us look at them and see what we have.

(Refer Slide Time: 37:30)



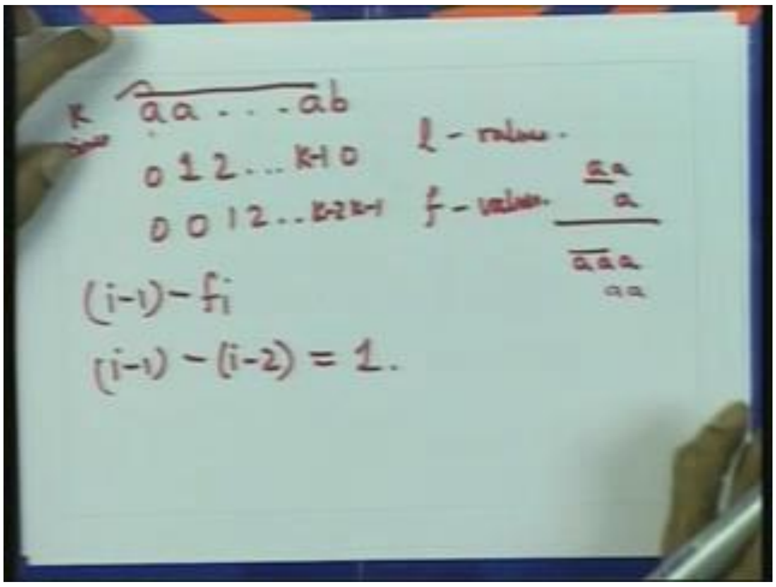
So, the two strings first string was a b b b this is k times. In other words it is a b to the k. So, what happens in this case? Suppose this is the pattern what we do. So, let what are the l values. So of course, for a it is 0. If I look at a b, what is the longest prefix, which is also a suffix, while if I look at a b the only prefix proper prefix I have is a.

And certainly it is not a suffix. So, this value is also 0, how about a b b? Well the two proper prefix is a and a b and meet the algorithm is the suffix here a is not a suffix ab is also not a suffix I will this matches this does not matches. So, there is no proper prefix of a b b which is also a suffix of this is 0. And. In fact, all of them as zero for n1 of them is that a proper prefix which is also a suffix, so that is the l value. So, f values are also 0.

Now, what is so when there is a mismatch, what is what does 1 shift by let us recall this. So the shift value, if there is a mismatch at the ith position is i minus 1 minus fi. In this case it is i minus 1. If there is mismatch of the ith position I shift the pattern to the right by i minus 1 position this is exactly what we did earlier.

So, we basically shift the pattern all the way up to the position, where this a mismatch almost to the position, where there is a mismatch. So that was for that pattern and this is exactly what we did earlier before we did all this you know analysis about prefix and then suffixes. But, this is the idea that the sort of intuitively used use then.

(Refer Slide Time: 40:07)


$\overbrace{k\ \ aa\ldots a}b$

$0\ 1\ 2\ldots k\text{-}1\ 0$ l - values.

$0\ 0\ 1\ 2\ldots k\text{-}2\ k\text{-}1$ f - values

$(i-1)-f_i$

$(i-1)-(i-2)=1.$

aa
a
aaa
aa

So, let us look at the second 1 which was a a a b this is k times. The string of a, so this is the second pattern that we looked at. So, what are the l values? Well it is 0 for 1 the first 1 if I look at a a the longest prefix which is also a suffix well the only proper prefix is a that is also a suffix. So, this value is 1, how about a a a? Well the longest prefix which is also a suffix is a a longest proper prefix which is also a suffix is a. So, this value is 2 and so on.

This value will be k minus 1 for this it will be 0. These are the l values what are the f values? Well the f values are just all of these shifted to the right. So, I have 0 0 1 2 so on k minus 2 k minus 1. These are the f values. And if there is a mismatch, then you shifted by i minus 1 minus fi and in this case and this case this will be i minus 1 and fi you can see its nothing but i minus 2. So, this is 1.

So, you actually shift the pattern by just 1 unit which is also what we did there. If there is a mismatch we only shifted it by 1 unit. But, the crucial thing is we start compare now we start comparing where there was a mismatch. We do not go back and compare everything again. This was the other lesion be we had we are learned, we also done this earlier.

That once the pattern once you have matched up to some level of the pattern. We know the previous things are all a’s in this case. Some we do not have to again sort of match all over them. We know that the shift that portion will always match with the previous thing we just need to look at the new position in the text. So that is it for pattern matching from us I would recommend strangely that.

You take more examples and do this yourself take examples of text strings take examples of pattern take these patterns compute the l values compute the f values see, how these values you know sit in. Then, take text and see how the algorithms runs with this pattern and takes what happens to the pointers how the pointers move how do you shift, what exactly this 1 mean by shifting a pattern etcetera, etcetera. So, I would like you to run this algorithm. You let you hopefully understood on a few examples that you can cook up yourself. So that you get a better hang of what exactly is going on.

Thank you.