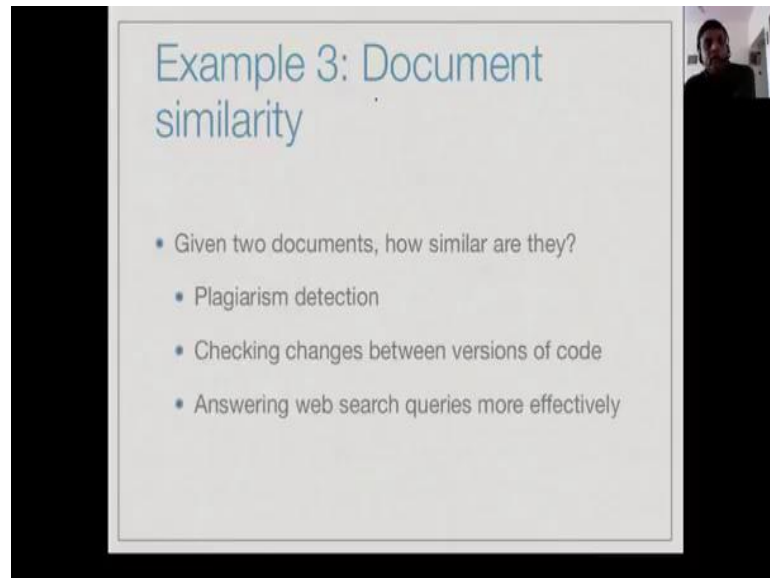


Design and Analysis of Algorithms
Prof. Madhavan Mukund
Chennai Mathematical Institute

Week - 01
Module - 01
Lecture - 04

(Refer Slide Time: 00:09)



Example 3: Document similarity

- Given two documents, how similar are they?
- Plagiarism detection
- Checking changes between versions of code
- Answering web search queries more effectively

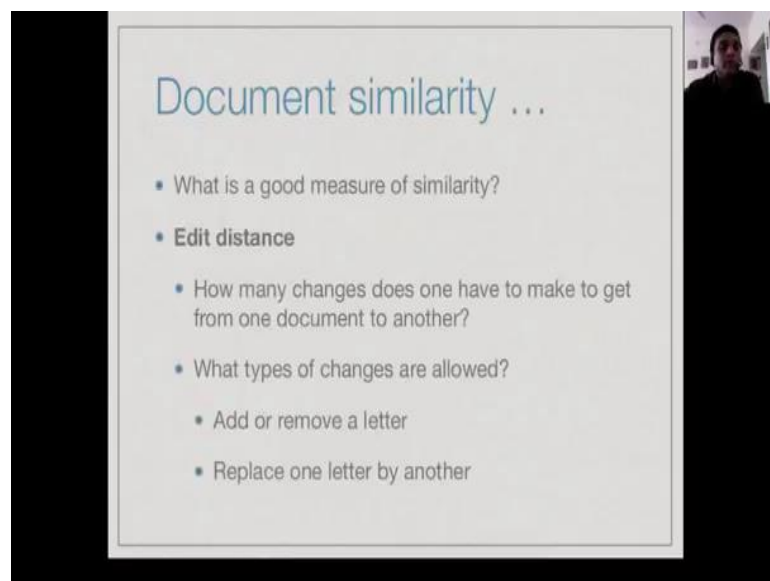
So far at final example before we delegate in this course, let us look at a problem involving documents. So, we have two documents and our goal is to find out how similar they are, right. So, these two documents really variations of the same field. Now, there may be many different scenarios where this problem is interesting. So, one question may be for **plagiarism detection**. So, it could be that somebody has forced to an article in a newspaper or on a website and you believe that this author has not really written the article themselves. They have copied these articles from somewhere else or if you are a teacher in a course, you might be worried that the student, two students have submitted the same assignments or one student has copied an assignment from some source, some detail. So, while looking at how similar, if you can measure or similar two documents are, you can try to quantify this notion that is somebody has copied from somebody else.

Now, it may not always have a negative connotation like this. It might also be to look at some kind of things when some people are writing code typically writing programs for some application, over the period of time documents evolve with in this sense the

programs evolves, right. So, people add features. Now, you might want to look at two different pieces of code and try to figure out what are the changes that had happened. How similar they are, how different they are, what the actual changes that had happened.

Another place where there is positive notion towards documents similarity is to look for web search. If you ask a question to a search engine and it reports results, typically it tries to group together result which is similar because they are not really different answers. Now, if there are 10 different copies or similar copies of a document saying more or less the same thing and these show up as your first 10 search results, then another document will be highly relevant and quite different from these will now be lost because it will be of the first page of searches. So, it is useful to be able to group together the results of a search query by similarity, so that the user is actually presented by an effective choice between different answers to the search query and not just the large number of variations of the same answers.

(Refer Slide Time: 02:13)



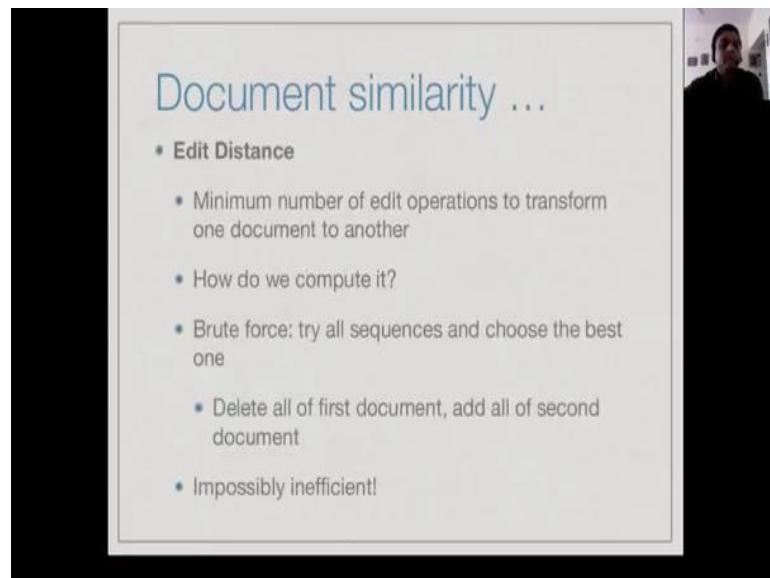
Document similarity ...

- What is a good measure of similarity?
- Edit distance
 - How many changes does one have to make to get from one document to another?
 - What types of changes are allowed?
 - Add or remove a letter
 - Replace one letter by another

So, if this is our motivation, we need a way of comparing documents what is the good measure of similarity of documents. Now, there are many different notions that people have come up with. Obviously, it has to do something with the order towards and the choice of letters and so on, but one way of quantifying the distance looking to document is to use what is called the edit distance, namely how many changes to you have to make to transform one document in to another document, and the edit we mean supposing you

actually loaded the document in a text editor or word processor, what would be the kind of things that you could do when we could limit because you got of course block out and delete the entire document and then cut and paste another document and say edit in two steps, but this could be kind of cheating. So, we have to limit what operations you do, so that we have a uniform way of counting this. So, we could say that edit involves how many characters which changing. So, each step of editing will either add or remove a letter and perhaps you can allow you to replace one letter by another letter and call that one change. So, now we want to count these as are basic steps adding or removing the letter or replacing one letter by another, and find out how many steps it takes to edit one document make it to make it to another document.

(Refer Slide Time: 03:42)

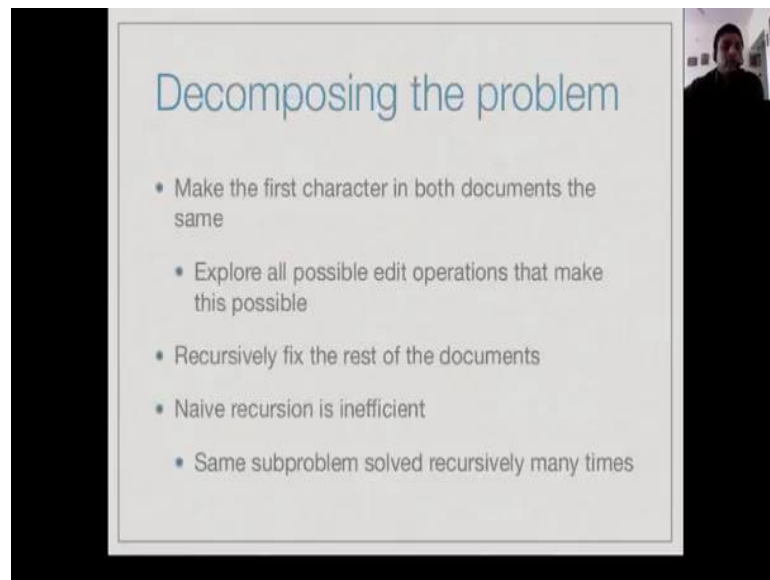


Document similarity ...

- **Edit Distance**
 - Minimum number of edit operations to transform one document to another
 - How do we compute it?
 - Brute force: try all sequences and choose the best one
 - Delete all of first document, add all of second document
 - Impossibly inefficient!

So, the minimum number of edit operations will then return the distance. Now, the question that we have as an algorithm problem is how do compute this minimum distance, right. How do you decide what is the best way to edit one document and make it another document. Of course, there is always the trivial solution like that block cut and block space. You can just delete all the letters and then type in the new documents. So, there is a brute force way of doing it, but this is not likely to be the best possibility, right. So, you can also try out all possible delete and insert sequences and see which among them gives you the best solutions, but all of these are very inefficient kind of solutions.

(Refer Slide Time: 04:26)



Decomposing the problem

- Make the first character in both documents the same
- Explore all possible edit operations that make this possible
- Recursively fix the rest of the documents
- Naive recursion is inefficient
- Same subproblem solved recursively many times

So, again we can go to this **question is decomposing the problem**. So, supposing our first goal is just make the first character of the two documents say, if they are already the same, we leave and go on. If they are not the same, well then we have two options, right. We can either transform the character, the first character to be equal or we can insert a character in one of the two documents. So, supposing the document, first document start with an x and the second document have the z. Either we can say we do one operation to make x into z or z into x or we can insert x before the z insert before the x or insert the z before the x, but then we do not necessarily get the same answer. Then, once we have done this, once we have made the first character the same, then we can recursively try to fix the rest of the document.

(Refer Slide Time: 05:27)

Naive recursion can be inefficient

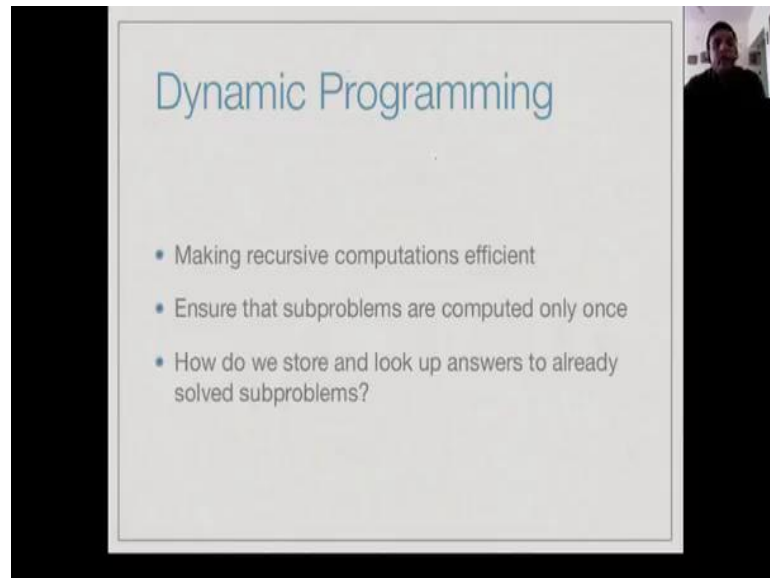
- Fibonacci numbers:
 - $F(n) = F(n-1) + F(n-2)$, $F(1) = 1$, $F(2) = 1$
 - Sequence is 1, 1, 2, 3, 5, 8, 13, 21, ...
 - Computing recursively
 - $F(7) = F(6) + F(5) = (F(5) + F(4)) + (F(4) + F(3)) = (F(4) + F(3) + F(3) + F(2)) + (F(3) + F(2) + F(2) + F(1)) = \dots$

So, now one of the difficulties we face when we do recursion in this manner is that the same sub-problem becomes up for solutions many times. So, a typical example of this is the recursive solution to finding the n Fibonacci. So, the Fibonacci numbers are defined, it is a very classical sequence. So, the first two Fibonacci numbers are 1 and 1. After this you get the next Fibonacci number adding the previous tools. So, after 1 and 1, the next one is 2 which is 1 plus 1. The next one is 3 which is 1 plus 2 and so on. 5 is 2 plus 3 and so on. So, in general the recursive relationship is given by the fact that f_n is the sum of the previous two numbers, n minus 1 n minus 2, and then you have as a base case that the first two numbers f_1 and f_2 for which n minus 1 and n minus 2 may not be defined for these two numbers, the values 1.

Now, the problem is that when you apply the recursion directly, so if try to compute the seventh Fibonacci number for example, it will say for this I need to compute f_6 plus f_5 , but by the excursively apply this f_6 and f_5 , we find things like f_4 coming up twice. So, we have an f_4 which comes here and then f_4 comes here because when I do f_5 , I need to apply this written, and when I do f_6 , I need to apply recursively. So, If I do it, I do f depth forth twice and in fact, I compute this f_5 , I actually get another f_4 . This f_4 I am computing a number of times, f_3 a number of times and so on. So, this is really an inefficient way to calculate it whereas, I just do it for in since here, I get 1 1 2 3 5 8 13 21 and I find that the seventh Fibonacci number is actually you sequence not use that it, right. So, there is intuitively a very fast way to do this. The recurrence is respected, but if

I do it recursively, I end up solving a lot of problems again and again. So, how do we get around it and this is what we call dynamic program.

(Refer Slide Time: 07:17)

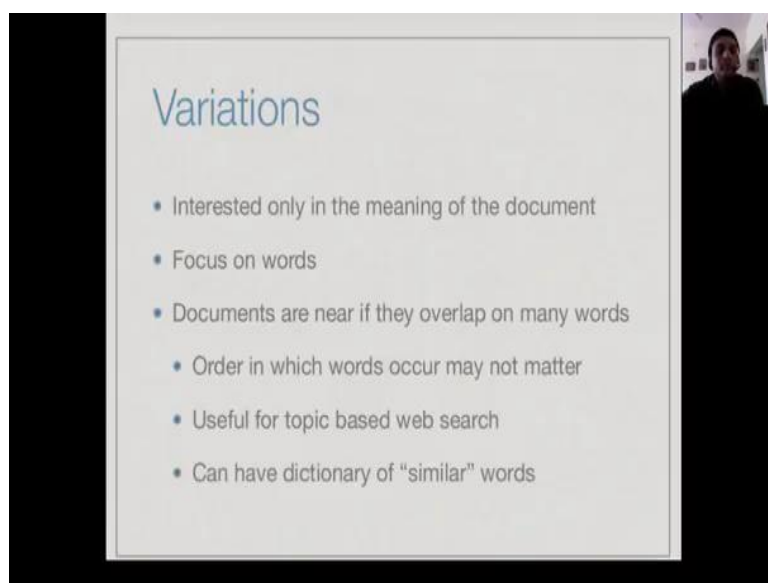


Dynamic Programming

- Making recursive computations efficient
- Ensure that subproblems are computed only once
- How do we store and look up answers to already solved subproblems?

So, **dynamic programming says do not compute same sub-problems twice.** Whenever we solve the problems, if have found f of 4, just look it up, store it somewhere, look it up and make sure that you do not do f 4 again. So, this is one of the techniques that we have seen in beginning as that we are going to do in this course, and it is important that when break-up problems into sub-problems, it is not always the case that sub problems can be solved efficiently unless we look slightly more deeply to this structure of sub-problem and make sure we solve them in an effective sequence.

(Refer Slide Time: 07:55)



Variations

- Interested only in the meaning of the document
- Focus on words
- Documents are near if they overlap on many words
 - Order in which words occur may not matter
- Useful for topic based web search
- Can have dictionary of "similar" words

Now, as usual this problem of, the difference or similarity between two documents can be at many different levels. So, we are focused on the words, the actual text, but if we do not really look at the sequence of words, we just want to set of words, then we might the for apart in terms edit distance because we need to rearrange the words, but the content if you just measure in terms of what types of words are there, this might give us an accurate understanding of the meaning of the documents. So, if you actually search for a document in a typical search engine, you will often find that the words that you ask for may not occur together, may not occur in the sequence that you mention. It will find just document which have that collection of words. So, this is very useful for web search and the other thing that you might want to do is, **measure similarity of words and terms of meanings**.

So, if you search for a document which contains the word car and there is another document which contains the words automobile, it might to be a good idea for the search engine to go to documents containing automobile, because automobile and car is essentially the same thing. So, like the other example you seen before, there can be variations on the problem and the solution you have for the regional problem, may or may not be valid for these variations. So, there is always a whole space of new and interesting problem to solve ((Refer Time: 09:25)).