

Natural Language Processing
Prof. Pawan Goyal
Department of Computer Science and Engineering
Indian Institute Technology, Kharagpur

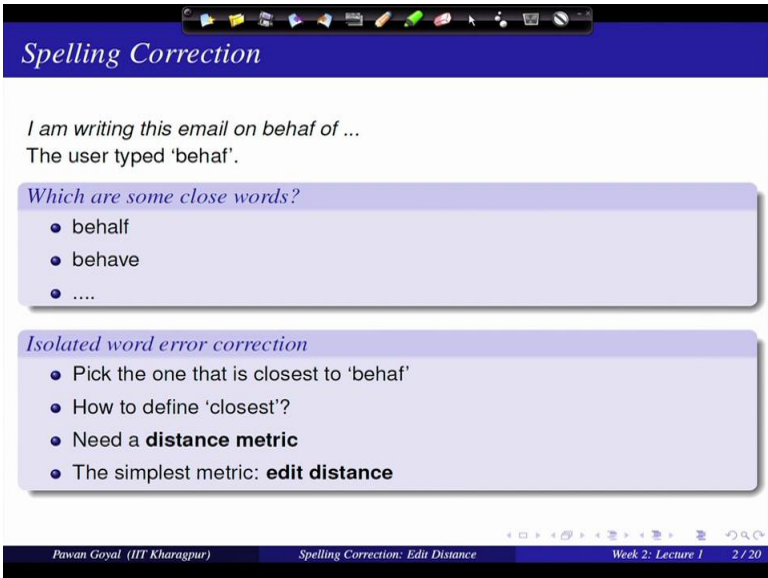
Lecture - 06
Spelling Correction: Edit Distance

Welcome back to the second week of this course. So, last week we handled with some the concept in pre-processing of the text, so what are the concepts that we covered? We covered given is sentence or document in general; a text document, how do I cyber into a sentences, how do I segment into words; we saw what are specific issues that might arise because of various languages and we talked about normalization, case folding and other aspects like lemmatization and stemming.

So these are some very very standard pre-processing tasks that you might have to do on a given text in a given language. But there is another task that you might have do, in certain cases, if the data that you are obtaining is bit noisy; what I mean noisy? There are certain spelling errors and you want to correct them for doing your analysis. So this task is called spelling correction. So, what are the various problems that involved in spelling correction and what are the different algorithm that we will be using?

So, we will be start with the very very simple algorithm of how do you use at the distance for spelling correction in this particular lecture.

(Refer Slide Time: 01:42)



Spelling Correction

I am writing this email on behalf of ...
The user typed 'behaf'.

Which are some close words?

- behalf
- behave
-

Isolated word error correction

- Pick the one that is closest to 'behaf'
- How to define 'closest'?
- Need a **distance metric**
- The simplest metric: **edit distance**

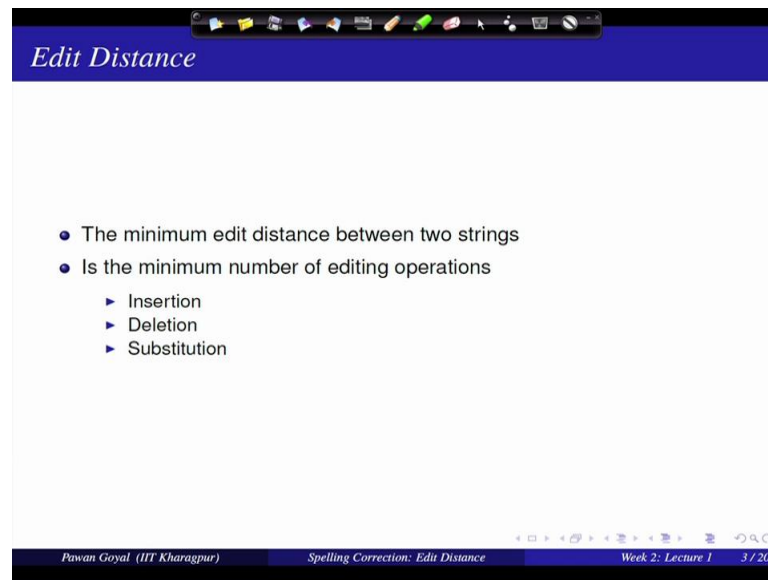
Pawan Goyal (IIT Kharagpur) Spelling Correction: Edit Distance Week 2: Lecture 1 2 / 20

So, what is the problem of start a spelling correction; suppose you are reading or you are seeing this text in your data and the sentence is I am writing on behalf of, but in behalf you see only b e h a f and l is missing. So, now you want to correct this particular word to the actual word behalf, now what is the algorithm approach of solving this problem? So, in general if you see this word like behalf, so ideally we would want to find out what are the closest words in English that might come in place of this. So, other words you can think of; I can think of probably behalf and I can think of probably behave yes and there might be some other words.

Now so when I will be talking about the spelling correction, I will talk about two different scenarios; one is isolated word error correction; that is I am trying to correct a word that is incorrect, but I am not trying to use the context around it. So, given the word b e h a f, I want to correct it without choosing what is there before and after this word and this is called isolated word error detection. So now, in this approach I will try to find the word that are closest to behalf; you see maybe behalf and behave will be two words.

Now so immediately I will crop them, so how do I define what do I mean by closest. So, ideally we say where you have not some sort of match between this word and that word, but how do I further define it. So, I need some sort of distance metric, how do I measure distance between one word to another word and as per this metric whichever two words are coming out to be closer will be called as closest, if they were coming too far I will not even consider as a candidate for the correction. So, I really want distant metric that can give me the b e h a f and b e h a l f are very very closed together. So, we will start and we will actually use the metric called edit distance and this is one of the simplest metric that we can think of in this case. Now what is edit distance?

(Refer Slide Time: 04:04)



So what is my problem? In general I am given two strings; one that is incorrect another might be correct and I have to find out what is the minimum distance between these two strings, so that is what I want to define by my concept of edit distance. Now how is it defined, so minimum edit distance is defined as the minimum number of edit operations that I have to do for going from one string to another one, as a name say edit distance; how many edits you are making to go from one string to another string; that is how it is defined.

Now immediately I will have the next question; what are all the various edit operations I can make for going from one string to another string. So, we will start with three basic operations that are insertion; that means, I insert a character in the first string, deletion; that means, I delete a character in the first string or substitution; I delete or I substitute one character in the first string by another character. So, these are three different operations that we will consider. So, now my question would be how many of these operations are needed to go from one string to another string that is my edit distance between two strings.

(Refer Slide Time: 05:30)

Minimum Edit Distance

Example
Edit distance from 'intention' to 'execution'

I N T E * N T I O N
| | | | | | | | |
del * E X E C U T I O N

Pawan Goyal (IIT Kharagpur) Spelling Correction: Edit Distance Week 2: Le

So, let us take a simple example, so I have the initial word is intention and the final word is execution and I want to find out how many of these operations are required to go from intention to execution. So, now if I try to see that, so what am I doing to go from intention to execution; I am first doing a deletion, I deleted I. Now I substitute it N with E, again I substitute it T with X; I kept E as it is, then I inserted C.

So, here star means I went to null that is the deletion; here null went to C that is insertion. Then again I did a substitution and with U and T, I, O, N remain as it is. So, how many operations I needed? I needed five different operations; one was deletion, one intentions and three substitutions, so here the various operations that I did. Now I need to define for these operations what will be the edit distance.

(Refer Slide Time: 07:04)

Minimum Edit Distance

INTENTION
| | | | | | | | |
* EXECUTION
d s s i s

- If each operation has a cost of 1 (Levenshtein)
 - ▶ Distance between these is 5
- If substitution costs 2 (alternate version)
 - ▶ Distance between these is 8

Pawan Goyal (IIT Kharagpur) Spelling Correction: Edit Distance Week 2: L

So, there are two difference variations that I can use; in one variation I might say that each operation has cost of 1. So, whether I am doing deletion intention or substitution each has a cost of 1. So, in this case what will be the edit distance between the two strings because I have done five operations; each has a cost of 1, so distance will be 5, so this is called Levenshtein distance.

Now I may also use another variant where the substitution has a cost of 2 because this is one insertion, one deletion. So, if I take substitution cost as 2; what will be the edit distance between the two strings? So, when I have 1 plus 1 for one insertion, one deletion plus 3 times 2 for 3 substitutions that will be 8.

(Refer Slide Time: 08:09)

How to find the Minimum Edit Distance?

Searching for a path (sequence of edits) from the *start string* to the *final string*:

- **Initial state:** the word we are transforming
- **Operators:** insert, delete, substitute
- **Goal state:** the word we are trying to get to
- **Path cost:** what we want to minimize: the number of edits

Diagram illustrating edit operations on the word "intention":

```
graph TD; intention -- Del --> ntention; intention -- Ins --> eintention; intention -- Sub --> entention;
```

Pawan Goyal (IIT Kharagpur) | Spelling Correction: Edit Distance | Week 2: Le

So, now my question is that how do we find the minimum edit distance between two strings, so what will be the approach for doing metric. So let us define it formally what are we trying to achieve, so I am trying to search for a path, by a path I mean what are the sequence of edit operations that I have to make for going from the start string to my final string. So now I can define what is my initial state; that is what is the word that I am trying to transform?

So in the previous example intention, so this is my initial state then what are the various operators that I am allowed, what various operators that I am allowed to use over my over any of the strings; I am allowed to use insertion, deletion or substitution; these are my three operators and there is a one goal state that is the final string that I am trying to get. So, it will be execution in the previous case and I want to minimise the path cost; number of edits, I want to find out what is the minimum number of operations or minimum path cost by which I can go from initial strings to final string. So, now if you think of it in a very naive manner, what will be an approach that you might try? So, I am starting with intention and I have to reach execution. So, one very naive method might be you start with intention and try out all possible edit operations until unless you reach execution.

So, that is I shall intention; if I delete one character; I can delete i, I go to n intention. Similarly you might delete other characters, you might insert one character in the

beginning say e or you can insert in any of the places or you might try to substitute one character like e can be substituted by; I can be substituted with e and that gives you another word and you might keep on trying all these operations until there is a point when you are at the goal state. So, this might be one possible approach; now what is even problem that you see where is approach. So, don't you think you might have to travel a lot of paths that are not at all relevant for my problem, why should I keep deleting i n t e n t i o n all together in one path when this will never give me execution.

(Refer Slide Time: 10:44)

The slide is titled "Minimum Edit as Search" in a blue header. Below the header, there is a light purple box containing the text "How to navigate?" followed by a bulleted list. At the bottom of the slide, there is a footer with navigation icons and text.

Minimum Edit as Search

How to navigate?

- The space of all edit sequences is huge
- Lot of distinct paths end up at the same state
- Don't have to keep track of all of them
- Keep track of the shortest path to each state

Pawan Goyal (IIT Kharagpur) Spelling Correction: Edit Distance Week 2: Lecture 1 7 / 20

So here I am trying to navigate all the possible edit operations to achieve the goal state, but this is gets me huge space, we will try to work out one simple example that how many even if I try to find words with the addition one or two, it might be the number of operation, the number of possibility might be huge, so how can I optimise this. So, in this case what would also happen; lot of paths might end up with the same state.

So, I might end up with doing some insertion and another deletion that might end up in the same state I was starting with or it might have a insertion, deletion or another substitution. So, again this is unavoidable that kind of approach, but we will like to avoid it. Further do I really need to keep track of all of them, I need only those paths that will come between intention to execution. So the idea would be; I define some sort of the states and keep track what are shortest paths to each state because ideally I want to find it what is the shortest distance between two strings. So, there is no reason that I should

store all possible ways of arriving from one string to another that are having a higher distance because I want to keep the minimum distance path.

(Refer Slide Time: 12:21)

Defining Minimum Edit Distance Matrix

For two strings

- X of length n
- Y of length m

We define $D(i,j)$

- the edit distance between $X[1..i]$ and $Y[1..j]$
- i.e., the first i characters of X and the first j characters of Y

Thus, the edit distance between X and Y is $D(n,m)$

Pawan Goyal (IIT Kharagpur) Spelling Correction: Edit Distance Week 2: Lecture 1 8 / 20

So what is an approach that I will use? So for that let us try to define it more formally. So, I have two strings X and Y ; X is of length n and Y is of length m and I am trying to find out the edit distance from X to Y , so in general I can define some edit distance metrics.

So how will I define it, so let us see my definition $D(i,j)$; where an element $D(i,j)$ denotes the edit distance between the first i characters of X and first j characters of Y . So, how many operations I need to make to go from i characters of X to j characters of Y . So, now as per this definition what will be the edit distance between X and Y ; this will be simply $D(n,m)$; that means, the n characters of X and m characters of Y . So now, question is what kind of algorithm I should use for obtaining the element of this matrix.

(Refer Slide Time: 13:30)

Computing Minimum Edit Distance

Dynamic Programming

- A tabular computation of $D(n,m)$
- Solving problems by combining solutions to subproblems
- Bottom-up
 - Compute $D(i,j)$ for small i,j
 - Compute larger $D(i,j)$ based on previously computed smaller values
 - Compute $D(i,j)$ for all i and j till you get to $D(n,m)$

Pawan Goyal (IIT Kharagpur) Spelling Correction: Edit Distance Week 2: Lecture 1 9 / 20

So, there we will try to see this problem as a dynamic programming approach, so we will compute $D(n,m)$ in a tabular manner. In general what do we do in dynamic programming, so whenever we have to solve the problem; we try to divide it in such a manner that I first solve the smaller problems and try to use their solutions to solve the bigger problem so that I can avoid certain repetitions in my computations. So, I will first start by solving $D(i,j)$ for small values of i and j and I will implemently use that to find out finally, my $D(n,m)$; by using the smaller values that I have already computed.

So, this is the bottom of approach; I first solve the very smalls of problems and then I use it to solve the bigger problems. So, I compute $D(i,j)$ for small i and j and based on the smaller values that I have computed, I will compute larger values of $D(i,j)$ and I keep on doing it until I get $D(n,m)$, so this will be our basic idea.

(Refer Slide Time: 14:56)

Dynamic Programming Algorithm

Initialization
 $D(i, 0) = i$
 $D(0, j) = j$

Recurrence Relation:
For each $i = 1 \dots M$
For each $j = 1 \dots N$
$$D(i, j) = \min \begin{cases} D(i-1, j) + 1 \\ D(i, j-1) + 1 \\ D(i-1, j-1) + \begin{cases} 2, & \text{if } X(i) \neq Y(j) \\ 0, & \text{if } X(i) = Y(j) \end{cases} \end{cases}$$

Termination:
 $D(N, M)$ is distance

Handwritten red annotations:
- A string 'X' is shown with a bracket from the first character to the last, labeled 'del' with an arrow, indicating a deletion operation.
- A string 'Y' is shown below it.

Footer: Pawan Goyal (IIT Kharagpur) | Spelling Correction: Edit Distance | Week 2: Lec. 2

So, now what we will need to see that; how should I will be using the smaller computations to get the value for the large amounts? So, I can define; so here I am I can define my dynamic programming algorithm for finding out the edit distance between two strings. So, first we are doing the initialization, so there we are saying $D(i, 0)$ is i and $D(0, j)$ is j now does that make sense, what is $D(i, 0)$ that is the distance between the i characters of X and 0 characters of Y . Now what will be the distance, how many operations I have to make for going from i characters of X to 0 characters of Y and how can I do that, the only possible way of doing it efficiently is I keep on deleting each of the character. So, there will be i operations of deletions, the cost will be i . Similarly how we go from 0 character of X to j characters of Y , I can do j different intention. So, that is why I can initialize it as $D(i, 0)$ is i and $D(0, j)$ is j ; that is fine.

Now, in general how do I define $D(i, j)$ in terms of the previous the other short elements that I have already filled up, so here are 3 (Refer Time: 16:13) by which $D(i, j)$ might be filled. So, suppose I am trying to find out the edit distance between X that is having; right now i characters of X and Y that is having j characters. Now the idea is that how go I compute $D(i, j)$ using the smaller values of i and j . So, that is how I use $D(i-1, j)$; $D(i, j-1)$ and $D(i-1, j-1)$, so let us see in this example. So, I want to use $D(i-1, j)$, so what is $D(i-1, j)$.

So, this is suppose one character, so this is my i minus 1; distance between X_{i-1} to Y_j and suppose we have already compute it, what is the minimum cost between X_{i-1} characters and Y_j characters. Now how do I use to find out what might be the minimum possible edit distance between X and Y_j . So, for the i characters in X ; I can do a deletion, I go to X_{i-1} and then I use the distance between X_{i-1} Y_j . So, how can I write the distance; I can say that same distance that much would be $D_{i-1, j}$; plus 1 because I deleted one character from X ; is that clear; in X_i I delete one character, I went to X_{i-1} and I have already computed distance between X_{i-1} and Y_j , so that is one for deletion and this cost we already have; similarly how do you use the use $D_{i, j-1}$; same idea.

(Refer Slide Time: 18:19)

Dynamic Programming Algorithm

Initialization
 $D(i, 0) = i$
 $D(0, j) = j$

Recurrence Relation:
 For each $i = 1 \dots M$
 For each $j = 1 \dots N$
 $D(i, j) = \min \begin{cases} D(i-1, j) + 1 \\ D(i, j-1) + 1 \\ D(i-1, j-1) + 2, \text{ if } X(i) \neq Y(j) \\ 0, \text{ if } X(i) = Y(j) \end{cases}$

Termination:
 $D(N, M)$ is distance

So, here I have X and Y ; X is i ; Y is j and suppose I have already computed i, j minus 1. So, how will I compute the distance between i and j , i is the same distance that I computed between i and j minus 1 plus I inserted the j th character. So, this you can think of it as insertion, so distance will be i, j minus 1 and 1 and what might be the third possibility; third possibility would be that I know what is the distance between i minus 1 j minus 1 and I am seeing whether I am substituting i with j .

So there might be two ways; if each character is the same; that means I am not having any cost, but if there are different characters, I will have a cost corresponding to the substitution. So, $D_{i-1, j-1}$ plus 2 if the characters are not same and 0 if they

are same. So, there are three ways in which I can write $D[i][j]$; it can be $D[i-1][j]$ plus 1; $D[i][j-1]$ plus 1 or $D[i-1][j-1]$ plus 2; if the characters are different or 0 if the character are same, so there is three different ways I can define these cost. Now the cost we are only store in the minimum cost, I will see what is the minimum of all three cost and that is what I will store as $D[i][j]$ and that I keep on doing this until I arrive at $D[n][m]$. So, that is the simple dynamic programming approach for solving this.

(Refer Slide Time: 20:13)

The Edit Distance Table

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1									
#	0	1	2	3	4	5	6	7	8	9
#		E	X	E	C	U	T	I	O	N

Handwritten notes on the slide: Red arrows point to the first column (index 0) and the first row (index 1). A red circle is drawn around the cell at (1,0) which contains the value 1. Below the circle, the text "1" is written. To the left of the circle, the text "D[1][0]" is written. To the right of the circle, the text "1" is written. Below the first row, the text "1" is written. Below the first column, the text "1" is written. Below the first row and first column, the text "1" is written. Below the first row and first column, the text "1" is written. Below the first row and first column, the text "1" is written.

So, let us see once one example; now so same example intention and execution. So, we have already filled up some entries here; now can you relate to this, what is this entry this is my $D[1][0]$. So, initialization we saw that $D[1][0]$ will be simply 1 so that means I can simply delete; I can delete this character and I arrive at the final sub history; starting from i; how do I go to null; I delete i; so this is 1.

Similarly this will be 2; I can delete i and n; one at a time, this will be 3 i n t and all so on. Similarly why this is 1 from null to e; that means, I am inserting one character, null to e X i am inserting two characters. So, this will 1, 2 and so on, so that explains all these initial Levenshtein distance in this step. Now my problem is how do I fill up the other (Refer Time: 21:25) in this table, so for example, how I fill this particular (Refer Time: 21:29).

(Refer Slide Time: 21:35)

The Edit Distance Table

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1									
#	0	1	2	3	4	5	6	7	8	9
#	E	X	E	C	U	T	I	O	N	

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$

i: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
 j: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

So now, how did we define my dynamic programming approach, I said I filled $D[i][j]$ based on $D[i-1][j]$, $D[i][j-1]$ and $D[i-1][j-1]$ and j minus 1. Now in this case what is $D[i-1][j]$, suppose I want to fill this element; this my $D[i][j]$, what is $D[i-1][j]$; this is my i minus 1 j , what is my $D[i][j-1]$; this is my i minus 1 j minus 1, this is my i minus 1 j minus 1. Now what will be the cost here; it will be minimum of all these three operations. So, let us find out for individual $D[i-1][j]$ is 1 plus 1, so this gives me 2; $D[i][j-1]$ cost between null and I am sorry I think I wrote it incorrectly, it should be $D[i][j-1]$ and there should be the $D[i-1][j]$.

So, what is $D[i-1][j]$ that is from null to E, what is the cost 1 plus 1 that will be 2; $D[i][j-1]$ is i to null that is 1, plus 1 this will again give me 2 and $D[i-1][j-1]$ is j 0; plus 2 if the two characters are not the same. So, here the two characters i and E there are not the same; so this will be 2. So, that is I can go from any of these paths and I have a cost of 2, so minimum will again be 2. So, I enter here, so now suppose you want to use that to find out the value here. So, now one thing we can see, I need three different entries this, this and this. So, what will be the cost minimum of 3, 3, 3 again this will be 3.

So, like that I will keep on filling up all the table until I arrive at this element; that is my $D[n][m]$. So, in general there might be more than one ways of arriving at the minimum cost, so we will see how do we store that, but right now we can just put that cost as it is.

(Refer Slide Time: 24:28)

The Edit Distance Table

N	9	8	9	10	11	12	11	10	9	8
O	8	7	8	9	10	11	10	9	8	9
I	7	6	7	8	9	10	9	8	9	10
T	6	5	6	7	8	9	8	9	10	11
N	5	4	5	6	7	8	9	10	11	10
E	4	3	4	5	6	7	8	9	10	9
T	3	4	5	6	7	8	7	8	9	8
N	2	3	4	5	6	7	8	7	8	7
I	1	2	3	4	5	6	7	6	7	8
#	0	1	2	3	4	5	6	7	8	9
#	E	X	E	C	U	T	I	O	N	

Pawan Goyal (IIT Kharagpur) Spelling Correction: Edit Distance Week 2: Lec. 1

So we fill up this complete matrix, this is what I will get and you see the distance between intentions, execution; in this case is 8. But you also see in this slide that for certain strings, the edit distance might go up to 12 that is possible, but finally the edit distance between intention, execution comes up to be 8. So, this approach gives me very very efficiently what is the edit distance between one string and another string and we know how to use this. Now so there is one further thing that we can do with this algorithm, so that is again depending on my algorithm; my particular task.

(Refer Slide Time: 25:17)

Computing Alignments

- Computing edit distance may not be sufficient for some applications
 - We often need to align characters of the two strings to each other
- We do this by keeping a "backtrace"
- Every time we enter a cell, remember where we came from
- When we reach the end,
 - Trace back the path from the upper right corner to read off the alignment

Pawan Goyal (IIT Kharagpur) Spelling Correction: Edit Distance Week 2: Lecture 1 13 / 20

That is about computing alignments, so for some applications we just want to know what is the edit distance between two strings but for other applications, you might also want to find out what are all the places where the edit took place, so what is the alignment. Remember the way we did the case of intention versus execution, we found out what are the actual places where insertion, deletion, substitution are taking place. So, this is what I mean by alignment which parts are line to the other part in the string.

So, now the question is can I modify the same algorithm to also find out what will be the various alignments in the string, and you will see it is very very easy by use the previous algorithm that we have seen. For some application, we might need to align characters of the two strings to each other. Now we do this by keeping a sort of backtrace, so what do you mean by backtrace. So whenever I am filling out the value of $D[i][j]$; i can come off from either $D[i-1][j]$; $D[i][j-1]$ or $D[i-1][j-1]$.

So whichever gives me the minimum cost, I will put a pointer that I came from either from left, from bottom or diagonal element. So whenever I am at $D[n][m]$, I will again start doing a backtrace from there which element I came here from and I keep on doing it from started from there until I arrive at the null characters. So every time we enter a cell, we remember where we came from and when we reach the end, we trace back the path from the upper right corner to read all the alignment.

(Refer Slide Time: 27:08)

The Edit Distance Table

N	9									
O	8									
I	7									
T	6									
N	5									
E	4	3	4							
T	3	4	5							
N	2	3	4							
I	1	2	3							
#	0	1	2	3	4	5	6	7	8	9
#	E	X	E	C	U	T	I	O	N	

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$

Pawan Goyal (IIT Kharagpur) Spelling Correction: Edit Distance Week 2: Lec

(Refer Slide Time: 28:14)

Minimum Edit with Backtrace

n	9	↓ 8	↖ 9	↖ 10	↖ 11	↖ 12	↓ 11	↓ 10	↓ 9	↖ 8	
o	8	↓ 7	↖ 8	↖ 9	↖ 10	↖ 11	↓ 10	↓ 9	↖ 8	← 9	
i	7	↓ 6	↖ 7	↖ 8	↖ 9	↖ 10	↓ 9	↖ 8	← 9	← 10	
t	6	↓ 5	↖ 6	↖ 7	↖ 8	↖ 9	↖ 8	← 9	← 10	← 11	
n	5	↓ 4	↖ 5	↖ 6	↖ 7	↖ 8	↖ 9	↖ 10	↖ 11	↖ 10	
e	4	↖ 3	← 4	↖ 5	← 6	← 7	← 8	↖ 9	↖ 10	↓ 9	
t	3	↖ 4	↖ 5	↖ 6	↖ 7	↖ 8	↖ 7	← 8	↖ 9	↓ 8	
n	2	↖ 3	↖ 4	↖ 5	↖ 6	↖ 7	↖ 8	↓ 7	↖ 8	↖ 7	
i	1	↖ 2	↖ 3	↖ 4	↖ 5	↖ 6	↖ 7	↖ 6	← 7	← 8	
#	0	1	2	3	4	5	6	7	8	9	
	#	e	x	e	c	u	t	i	o	n	

Pawan Goyal (IIT Kharagpur) Spelling Correction: Edit Distance Week 2: Lecture 1 16 / 20

So, let us see how we will do that in this case, so suppose I was filling up this element and I know I can come from either here or here or here; all three are equally likely. So, I might put all three points, but suppose when I was here the best possible way was this one. So, I should have come from the diagonal to this element, so this is what I will store, similarly here I will store another back point and so on. So, now once my edit operation is over; all you have these back point is each and every cell, I start from D n m and keep on following by back point.

So, suppose I follow this one, I reach this cell again I follow point I reach this cell this one and so on until I reach the null characters and that will give me the alignment of two strings. So you will have alignment starting from D n m, so we will see from here I can go to this element, this element, this element. Now when I am here, it could have come from either of the three elements. So, you might have some sort of preference; you might say I whenever there equal to equal possibility among all three, I will always diagonal this you might say. So, you come to this element then you go to left, then again you go down based on preference, down based on preference, down based on preference and then that is where you get your alignment. So, this is some way you can compute which characters in intention are aligned to which character in executions.

(Refer Slide Time: 29:03)

Adding Backtrace to Minimum Edit


Base conditions:
 $D(i, 0) = i$ $D(0, j) = j$

Termination:
 $D(N, M)$ is distance

Recurrence Relation:
 For each $i = 1 \dots M$
 For each $j = 1 \dots N$

$$D(i, j) = \min \begin{cases} D(i-1, j) + 1 & \text{deletion} \\ D(i, j-1) + 1 & \text{insertion} \\ D(i-1, j-1) + \begin{cases} 2 & \text{if } X(i) \neq Y(j) \\ 0 & \text{if } X(i) = Y(j) \end{cases} & \text{substitution} \end{cases}$$

$\text{ptr}(i, j) = \begin{cases} \text{LEFT} & \text{insertion} \\ \text{DOWN} & \text{deletion} \\ \text{DIAG} & \text{substitution} \end{cases}$

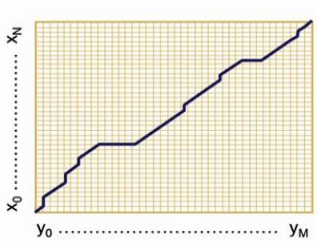


Pawan Goyal (IIT Kharagpur) Spelling Correction: Edit Distance Week 2: Lec

So, simple idea whenever so you can say these are deletion, insertion and substitution. So whichever gives you the minimum cost; you will add that pointer if all three are giving you the same cost, you might have all the three point.

(Refer Slide Time: 29:24)

The distance matrix



Every non-decreasing path from (0,0) to (M,N) corresponds to an alignment of two sequences.

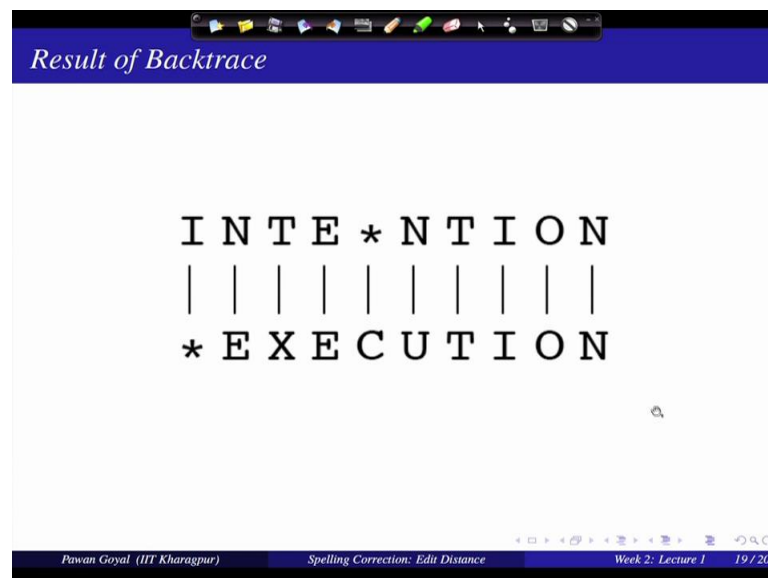
An optimal alignment is composed of optimal sub-alignments.

Pawan Goyal (IIT Kharagpur) Spelling Correction: Edit Distance Week 2: Lecture 1 18 / 20

So, in general suppose I start with a word X_0 to X_N , so 0 means null X_N is the complete word X and I have the output string Y_0 to Y_M . So, if I see this matrix of size N cross M , so every possible path from 0_0 to M_N is a possible alignment. So, what I am trying to find out which path gives me the minimum possible edit distance and this as per

dynamic programming approach and optimal alignment should be composed optimal sub alignments so; that means, whenever I am going from I to j, I can use always the optimal sub align alignment would be I minus 1 j and. so on, so that is how this problem algorithm is designed.

(Refer Slide Time: 30:24)



So, if you do the victories; you can find out from the intention, you go to executions and what are all the places where you did insertion, deletion and substitutions and which words you did not have to change at all, now what is the complexity of this algorithm. So, let us talk about the time complicity; how much time we took. So, you see you have to fill all the N cross M entries of the matrix. So, there are N cross and different things you have to do and for each entry, you need a constant number of operation you will find out minimum of three different distances.

(Refer Slide Time: 31:11)

The slide is titled "Performance" in a blue header. It contains three light purple boxes with the following text:

- Time*
 $O(nm)$
- Space*
 $O(nm)$
- Backtrace*
 $O(n + m)$

At the bottom, there is a footer with the text: "Pawan Goyal (IIT Kharagpur) Spelling Correction: Edit Distance Week 2: Lecture 1 20 / 20".

So there are N cross M times some constant, so the time complexity can be simply order of $N M$, what would the space complicity again the space complexity also the same you are using only N cross M different element that must the space that you need you are not using any other space remember for each element you are using some other elements from the same matrix. So, you not any edit distance space. So, again a space complexity is N cross M ; now what about the backtrace.

So, that is I am at $D_{n m}$ and I want to find backtrace, the worst case can be I have to go through all the N elements in my column N elements sorry N elements in my top and M elements in my first column. So in the worst case, I might have to do N plus M different operations, so this complexity is again N plus M . So, you will see this is very very efficient in compared to the approach that favour the naive approach that you have seen earlier you start with the input strings and try out all possibility this will be become explanation very very soon.

So, the first lecture for this week and in the next lecture, we will start talking about certain variation of edit distance. Why should I even think of some variations and how do we use that for finding edit distance.