Edit Distance

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Programming, Data Structures and Algorithms using Python

Week 9

Document similarity

- "The students were able to appreciate the concept optimal substructure property and its use in designing algorithms"
- "The lecture taught the students to appreciate how the concept of optimal substructures can be used in designing algorithms"

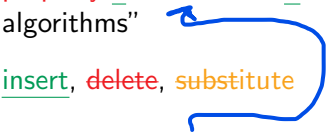
Document similarity

- "The students were able to appreciate the concept optimal substructure property and its use in designing algorithms"
- "The lecture taught the students to appreciate how the concept of optimal substructures can be used in designing algorithms"
- Edit operations to transform documents
  - Insert a character
  - Delete a character
  - Substitute one character by another

Document similarity

- "The students were able to appreciate the concept optimal substructure property and its use in designing algorithms"
- "The lecture taught the students to appreciate how the concept of optimal substructures can be used in designing algorithms"
- Edit operations to transform documents
  - Insert a character
  - Delete a character
  - Substitute one character by another

- "The lecture taught the students ~~were able~~ to appreciate how the concept of optimal substructures ~~property~~ cand ~~itbse~~ used in designing algorithms"
- insert, ~~delete~~, ~~substitute~~



Document similarity

- "The students were able to appreciate the concept optimal substructure property and its use in designing algorithms"
- "The lecture taught the students to appreciate how the concept of optimal substructures can be used in designing algorithms"
- Edit operations to transform documents
  - Insert a character
  - Delete a character
  - Substitute one character by another

- "The lecture taught the students ~~were able~~ to appreciate how the concept of optimal substructure~~s~~ ~~property~~ cand ~~itbse~~ used in designing algorithms"
- insert, ~~delete~~, ~~substitute~~

Edit distance

Document similarity

- "The students were able to appreciate the concept optimal substructure property and its use in designing algorithms"
- "The lecture taught the students to appreciate how the concept of optimal substructures can be used in designing algorithms"
- Edit operations to transform documents
  - Insert a character
  - Delete a character
  - Substitute one character by another

- "The lecture taught the students ~~were able~~ to appreciate how the concept of optimal substructures ~~property~~ cand ~~itbse~~ used in designing algorithms"
- insert, ~~delete~~, ~~substitute~~

Edit distance

- Minimum number of edit operations needed

converting "its" to "be" above can be done in several ways.
1. deleting "its" then adding "be"
2. Substitute "ts" with "be" and delete "i" (as done above)

We need to find minimum number of edit operations to achieve the same result

Document similarity

- "The students were able to appreciate the concept optimal substructure property and its use in designing algorithms"
- "The lecture taught the students to appreciate how the concept of optimal substructures can be used in designing algorithms"
- Edit operations to transform documents
  - Insert a character
  - Delete a character
  - Substitute one character by another

- "The lecture taught the students ~~were able~~ to appreciate how the concept of optimal substructure~~s~~ ~~property~~ can~~d~~ ~~itbse~~ used in designing algorithms"
- insert, ~~delete~~, ~~substitute~~

Edit distance

- Minimum number of edit operations needed
- In our example, 24 characters inserted, 18 ~~deleted~~, 2 ~~substituted~~

Document similarity

- "The students were able to appreciate the concept optimal substructure property and its use in designing algorithms"
- "The lecture taught the students to appreciate how the concept of optimal substructures can be used in designing algorithms"
- Edit operations to transform documents
  - Insert a character
  - Delete a character
  - Substitute one character by another

- "The lecture taught the students ~~were able~~ to appreciate how the concept of optimal substructure~~s~~ ~~property~~ can~~d~~ ~~itbse~~ used in designing algorithms"
- insert, ~~delete~~, ~~substitute~~

Edit distance

- Minimum number of edit operations needed
- In our example, 24 characters inserted, 18 ~~deleted~~, 2 ~~substituted~~
- Edit distance is at most 44

Edit distance

- Minimum number of editing operations needed to transform one document to the other
  - Insert a character
  - Delete a character
  - Substitute one character by another

Edit distance

- Minimum number of editing operations needed to transform one document to the other
  - Insert a character
  - Delete a character
  - Substitute one character by another
- Also called Levenshtein distance
  - Vladimir Levenshtein, 1965

Edit distance

- Minimum number of editing operations needed to transform one document to the other
  - Insert a character
  - Delete a character
  - Substitute one character by another
- Also called Levenshtein distance
  - Vladimir Levenshtein, 1965
- Applications
  - Suggestions for spelling correction
  - Genetic similarity of species

What is Edit Distance?

Measure of similarity between two strings based on the minimum number of operations (insertions, deletions, and substitutions) required to transform one string into the other.

Edit distance

- Minimum number of editing operations needed to transform one document to the other
  - Insert a character
  - Delete a character
  - Substitute one character by another
- Also called Levenshtein distance
  - Vladimir Levenshtein, 1965
- Applications
  - Suggestions for spelling correction
  - Genetic similarity of species

Edit distance and LCS

- Longest common subsequence of u, v

Edit distance

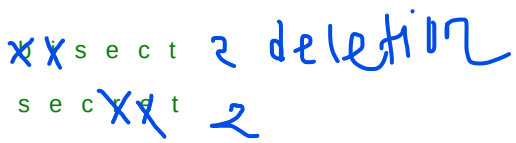
- Minimum number of editing operations needed to transform one document to the other
  - Insert a character
  - Delete a character
  - Substitute one character by another
- Also called Levenshtein distance
  - Vladimir Levenshtein, 1965
- Applications
  - Suggestions for spelling correction
  - Genetic similarity of species

Edit distance and LCS

- Longest common subsequence of u (m), v (n)
  - Minimum number of deletes needed to make them equal

Suppose LCS(u, v) = k

then deletes required to make both equal will be (m-k) for u and (n-k) for v



Edit distance

- Minimum number of editing operations needed to transform one document to the other
  - Insert a character
  - Delete a character
  - Substitute one character by another
- Also called Levenshtein distance
  - Vladimir Levenshtein, 1965
- Applications
  - Suggestions for spelling correction
  - Genetic similarity of species

Edit distance and LCS

- Longest common subsequence of u, v
  - Minimum number of deletes needed to make them equal
- Deleting a letter from u is equivalent to inserting it in v

Edit distance

- Minimum number of editing operations needed to transform one document to the other
  - Insert a character
  - Delete a character
  - Substitute one character by another
- Also called Levenshtein distance
  - Vladimir Levenshtein, 1965
- Applications
  - Suggestions for spelling correction
  - Genetic similarity of species

Edit distance and LCS

- Longest common subsequence of u, v
  - Minimum number of deletes needed to make them equal
- Deleting a letter from u is equivalent to inserting it in v
  - `bisect`, `secret` — LCS is `sect`

Edit distance

- Minimum number of editing operations needed to transform one document to the other
  - Insert a character
  - Delete a character
  - Substitute one character by another
- Also called Levenshtein distance
  - Vladimir Levenshtein, 1965
- Applications
  - Suggestions for spelling correction
  - Genetic similarity of species

Edit distance and LCS

- Longest common subsequence of u, v
  - Minimum number of deletes needed to make them equal
- Deleting a letter from u is equivalent to inserting it in v
  - `bisect`, `secret` — LCS is `sect`
  - Delete `b`, `i` in `bisect` and `r`, `e` in `secret`

Edit distance

- Minimum number of editing operations needed to transform one document to the other
  - Insert a character
  - Delete a character
  - Substitute one character by another
- Also called Levenshtein distance
  - Vladimir Levenshtein, 1965
- Applications
  - Suggestions for spelling correction
  - Genetic similarity of species

Edit distance and LCS

- Longest common subsequence of u, v
  - Minimum number of deletes needed to make them equal
- Deleting a letter from u is equivalent to inserting it in v in terms of edit distance
  - `bisect`, `secret` — LCS is `sect`

method 1
- Delete `b`, `i` in `bisect` and `r`, `e` in `secret` — 4 operations

method 2
- Delete `b`, `i` and then insert `r`, `e` in ~~`b`~~`sect` — 4 operations

As focusing and changing one string (or document) is easier, we focus on method 2

Edit distance

sti ual to the edit distance
between those strings when only
ins s are
cons stitutions.

In other words, if you exclude
is
an orm
one into another will be the
gth of the LCS of the
two strings

Edit distance and LCS

- Longest common subsequence of u, v
  - Minimum number of deletes needed to make them equal
- Deleting a letter from u is equivalent to inserting it in v
  - `bisect`, `secret` — LCS is `sect`
  - Delete `b`, `i` in `bisect` and `r`, `e` in `secret`
  - Delete `b`, `i` and then insert `r`, `e` in `bisect`
- LCS equivalent to edit distance without substitution

Inductive structure for edit distance

- $u = a_0 a_1 \ldots a_{m-1}$
- $v = b_0 b_1 \ldots b_{n-1}$

Inductive structure for edit distance

- $u = a_0 a_1 \ldots a_{m-1}$
- $v = b_0 b_1 \ldots b_{n-1}$

- Recall LCS

Inductive structure for edit distance

- $u = a_0 a_1 \ldots a_{m-1}$
- $v = b_0 b_1 \ldots b_{n-1}$

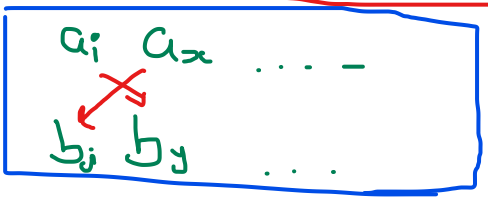
Remember you cannot have both because if you have both a_i and b_j then the LCS looks like this:

u = a_i...a_x...

v = b_j...b_y...

But we know a_x = b_j this means LCS is a_i....a_x... b_j...b_y...
But we also know a_i = b_y this means LCS is a_i...a_x... and b_j...b_y...

Thus both cannot occur toghther



- Recall LCS
- If $a_i = b_j$,
  $LCS(i,j) = 1 + LCS(i{+}1, j{+}1)$
- If $a_i \neq b_j$,
  $LCS(i,j) = \max[\ LCS(i, j{+}1),\ LCS(i{+}1, j)\ ]$

Inductive structure for edit distance

- $u = a_0 a_1 \ldots a_{m-1}$
- $v = b_0 b_1 \ldots b_{n-1}$

- Recall LCS
- If $a_i = b_j$,
  $LCS(i,j) = 1 + LCS(i{+}1, j{+}1)$
- If $a_i \neq b_j$,
  $LCS(i,j) = \max[\ LCS(i, j{+}1),$
  $LCS(i{+}1, j)\ ]$

- Edit distance — aim is to transform u to v

So instead of editing both u and v to make them equal, we will do uni-direction editing. i.e, we will transform u to v to find the EDIT DISTANCE

Inductive structure for edit distance

- $u = a_0 a_1 \ldots a_{m-1}$
- $v = b_0 b_1 \ldots b_{n-1}$

- Recall LCS
- If $a_i = b_j$,
  $LCS(i,j) = 1 + LCS(i{+}1, j{+}1)$
- If $a_i \neq b_j$,
  $LCS(i,j) = \max[\ LCS(i, j{+}1),$
  $LCS(i{+}1, j)\ ]$

- Edit distance — aim is to transform u to v
- If $a_i = b_j$, nothing to be done

Inductive structure for edit distance

- $u = a_0 a_1 \ldots a_{m-1}$
- $v = b_0 b_1 \ldots b_{n-1}$

- Recall LCS
- If $a_i = b_j$,
  $LCS(i,j) = 1 + LCS(i{+}1, j{+}1)$
- If $a_i \neq b_j$,
  $LCS(i,j) = \max[\ LCS(i, j{+}1),$
  $LCS(i{+}1, j)\ ]$

- Edit distance — aim is to transform u to v
- If $a_i = b_j$, nothing to be done
- If $a_i \neq b_j$, best among

Inductive structure for edit distance

- $u = a_0 a_1 \ldots a_{m-1}$
- $v = b_0 b_1 \ldots b_{n-1}$

- Recall LCS
- If $a_i = b_j$,
  $LCS(i,j) = 1 + LCS(i{+}1, j{+}1)$
- If $a_i \neq b_j$,
  $LCS(i,j) = \max[\ LCS(i, j{+}1),\ LCS(i{+}1, j)\ ]$

- Edit distance — aim is to transform u to v
- If $a_i = b_j$, nothing to be done
- If $a_i \neq b_j$, best among
  - Substitute a_i by b_j

Inductive structure for edit distance

- $u = a_0 a_1 \ldots a_{m-1}$
- $v = b_0 b_1 \ldots b_{n-1}$

- Recall LCS
- If $a_i = b_j$,
  $LCS(i,j) = 1 + LCS(i{+}1, j{+}1)$
- If $a_i \neq b_j$,
  $LCS(i,j) = \max[\ LCS(i, j{+}1),\ LCS(i{+}1, j)\ ]$

- Edit distance — aim is to transform u to v
- If $a_i = b_j$, nothing to be done
- If $a_i \neq b_j$, best among
  - Substitute a_i by b_j
  - Delete a_i

Inductive structure for edit distance

- $u = a_0 a_1 \ldots a_{m-1}$
- $v = b_0 b_1 \ldots b_{n-1}$

- Recall LCS
- If $a_i = b_j$,
  $LCS(i,j) = 1 + LCS(i{+}1, j{+}1)$
- If $a_i \neq b_j$,
  $LCS(i,j) = \max[\ LCS(i,j{+}1),\ LCS(i{+}1,j)\ ]$

- Edit distance — aim is to transform u to v
- If $a_i = b_j$, nothing to be done
- If $a_i \neq b_j$, best among
  - Substitute a_i by b_j
  - Delete a_i
  - Insert b_j before a_i

Inductive structure for edit distance

- $u = a_0 a_1 \ldots a_{m-1}$
- $v = b_0 b_1 \ldots b_{n-1}$
- Edit distance — transform u to v
- If $a_i = b_j$, nothing to be done
- If $a_i \neq b_j$, best among
  - Substitute a_i by b_j
  - Delete a_i
  - Insert b_j before a_i

Inductive structure for edit distance

- $u = a_0 a_1 \ldots a_{m-1}$
- $v = b_0 b_1 \ldots b_{n-1}$
- Edit distance — transform u to v
- If $a_i = b_j$, nothing to be done
- If $a_i \neq b_j$, best among
  - Substitute a_i by b_j
  - Delete a_i
  - Insert b_j before a_i
- $ED(i, j)$ — edit distance for $a_i a_{i+1} \ldots a_{m-1}$, $b_j b_{j+1} \ldots b_{n-1}$

Inductive structure for edit distance

- $u = a_0 a_1 \ldots a_{m-1}$
- $v = b_0 b_1 \ldots b_{n-1}$
- Edit distance — transform u to v
- If $a_i = b_j$, nothing to be done
- If $a_i \neq b_j$, best among
  - Substitute a_i by b_j
  - Delete a_i
  - Insert b_j before a_i
- $ED(i, j)$ — edit distance for $a_i a_{i+1} \ldots a_{m-1}$, $b_j b_{j+1} \ldots b_{n-1}$

- If $a_i = b_j$, $ED(i, j) = ED(i+1, j+1)$

Inductive structure for edit distance

- $u = a_0 a_1 \ldots a_{m-1}$
- $v = b_0 b_1 \ldots b_{n-1}$
- Edit distance — transform u to v
- If $a_i = b_j$, nothing to be done
- If $a_i \neq b_j$, best among
  - Substitute a_i by b_j
  - Delete a_i
  - Insert b_j before a_i
- $ED(i,j)$ — edit distance for $a_i a_{i+1} \ldots a_{m-1}$, $b_j b_{j+1} \ldots b_{n-1}$

- If $a_i = b_j$,
  $ED(i,j) = ED(i{+}1, j{+}1)$
- If $a_i \neq b_j$,
  $ED(i,j) = 1 + \min[\ ED(i{+}1, j{+}1),\ ED(i{+}1, j),\ ED(i, j{+}1)\ ]$

Inductive structure for edit distance

- $u = a_0 a_1 \ldots a_{m-1}$
- $v = b_0 b_1 \ldots b_{n-1}$
- Edit distance — transform u to v
- If $a_i = b_j$, nothing to be done
- If $a_i \neq b_j$, best among
  - Substitute a_i by b_j
  - Delete a_i
  - Insert b_j before a_i
- $ED(i,j)$ — edit distance for $a_i a_{i+1} \ldots a_{m-1}$, $b_j b_{j+1} \ldots b_{n-1}$

- If $a_i = b_j$,
  $ED(i,j) = ED(i{+}1, j{+}1)$
- If $a_i \neq b_j$,
  $ED(i,j) = 1 + \min[\ ED(i{+}1, j{+}1),\ ED(i{+}1, j),\ ED(i, j{+}1)\ ]$
- Base cases
  - $ED(m,n) = 0$

Inductive structure for edit distance

- $u = a_0 a_1 \ldots a_{m-1}$
- $v = b_0 b_1 \ldots b_{n-1}$
- Edit distance — transform u to v
- If $a_i = b_j$, nothing to be done
- If $a_i \neq b_j$, best among
  - Substitute a_i by b_j
  - Delete a_i
  - Insert b_j before a_i
- $ED(i,j)$ — edit distance for $a_i a_{i+1} \ldots a_{m-1}$, $b_j b_{j+1} \ldots b_{n-1}$

- If $a_i = b_j$,
  $ED(i,j) = ED(i{+}1, j{+}1)$
- If $a_i \neq b_j$,
  $ED(i,j) = 1 + \min[\ ED(i{+}1, j{+}1),\ ED(i{+}1, j),\ ED(i, j{+}1)\ ]$
- Base cases
  - $ED(m,n) = 0$
  - $ED(i,n) = m - i$ for all $0 \leq i \leq m$
    Delete $a_i a_{i+1} \ldots a_{m-1}$ from u

Inductive structure for edit distance

- $u = a_0 a_1 \ldots a_{m-1}$
- $v = b_0 b_1 \ldots b_{n-1}$
- Edit distance — transform u to v
- If $a_i = b_j$, nothing to be done
- If $a_i \neq b_j$, best among
  - Substitute a_i by b_j
  - Delete a_i
  - Insert b_j before a_i
- $ED(i,j)$ — edit distance for $a_i a_{i+1} \ldots a_{m-1}$, $b_j b_{j+1} \ldots b_{n-1}$

SEE NOTEBOOK FOR NOTES OF THIS SLIDE

- If $a_i = b_j$,
  $ED(i,j) = ED(i+1, j+1)$
- If $a_i \neq b_j$,
  $ED(i,j) = 1 + \min[\ ED(i+1,j+1),\ ED(i+1,j),\ ED(i,j+1)\ ]$
- Base cases
  - $ED(m,n) = 0$
  - $ED(i,n) = m - i$ for all $0 \leq i \leq m$
    Delete $a_i a_{i+1} \ldots a_{m-1}$ from u
  - $ED(m,j) = n - j$ for all $0 \leq j \leq n$
    Insert $b_j b_{j+1} \ldots b_{n-1}$ into u

Subproblem dependency

- Subproblems are $ED(i, j)$, for $0 \leq i \leq m$, $0 \leq j \leq n$

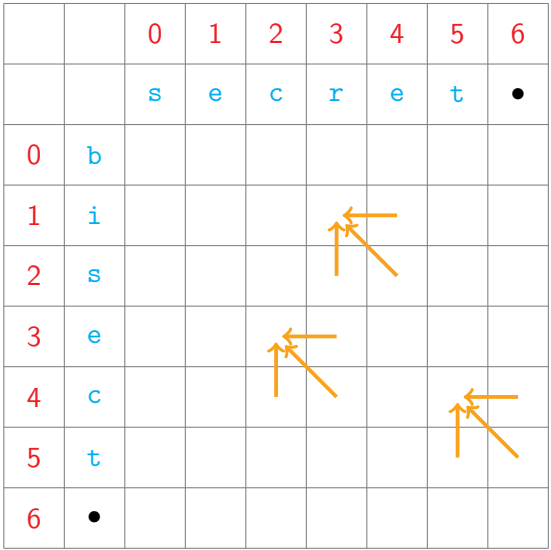
Subproblem dependency

- Subproblems are $ED(i, j)$, for $0 \leq i \leq m$, $0 \leq j \leq n$
- Table of $(m+1) \cdot (n+1)$ values

		0	1	2	3	4	5	6
		s	e	c	r	e	t	•
0	b							
1	i							
2	s							
3	e							
4	c							
5	t							
6	•							

Subproblem dependency

- Subproblems are $ED(i, j)$, for $0 \leq i \leq m$, $0 \leq j \leq n$
- Table of $(m + 1) \cdot (n + 1)$ values
- Like LCS, $ED(i, j)$ depends on $ED(i+1, j+1)$, $ED(i, j+1)$, $ED(i+1, j)$



Subproblem dependency

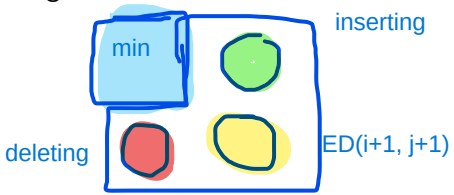
- Subproblems are $ED(i, j)$, for $0 \leq i \leq m$, $0 \leq j \leq n$
- Table of $(m+1) \cdot (n+1)$ values
- Like LCS, $ED(i, j)$ depends on $ED(i+1, j+1)$, $ED(i, j+1)$, $ED(i+1, j)$
- No dependency for $ED(m, n)$ — start at bottom right and fill by row, column or diagonal

We are going from BISECT to SECRET

		0	1	2	3	4	5	6
		v						
	u	s	e	c	r	e	t	•
0	b	u = "bisect" v= "", delete " bisect" ED=6						6
1	i							5
2	s							4
3	e							3
4	c	u = "ct" v= "", delete "ct" ED=2						2
5	t	u = "t" v= "", delete "t" ED=1						1
6	•	u = "", v = "" => ED() = 0						0

Subproblem dependency

- Subproblems are $ED(i, j)$, for $0 \leq i \leq m$, $0 \leq j \leq n$
- Table of $(m + 1) \cdot (n + 1)$ values
- Like LCS, $ED(i, j)$ depends on $ED(i+1, j+1)$, $ED(i, j+1)$, $ED(i+1, j)$
- No dependency for $ED(m, n)$ — start at bottom right and fill by row, column or diagonal



		0	1	2	3	4	5	6
		s	e	c	r	e	t	•
0	b						5	6
1	i						4	5
2	s						3	4
3	e						2	3
4	c	u="ct", v="t" delete "c" OR insert "t", u = "tct"					1	2
5	t	ED("t", "t") = ED("", "")					0	1
6	•	u = "", v = "t" insert "t"					1	0

min

Subproblem dependency

- Subproblems are $ED(i, j)$, for $0 \leq i \leq m$, $0 \leq j \leq n$
- Table of $(m + 1) \cdot (n + 1)$ values
- Like LCS, $ED(i, j)$ depends on $ED(i+1, j+1)$, $ED(i, j+1)$, $ED(i+1, j)$
- No dependency for $ED(m, n)$ — start at bottom right and fill by row, column or diagonal

u = "t" v = "et"
inserting "e", u = "et" takes 1 operation
deleting we need to make u="", v= "" takes 3 operations in total (remember all operations are done on u)

		0	1	2	3	4	5	6
		s	e	c	r	e	t	•
0	b					**4**	**5**	**6**
1	i					**3**	**4**	**5**
2	s					**2**	**3**	**4**
3	e					**1**	**2**	**3**
4	c					**1**	**1**	**2**
5	t					**1**	**0**	**1**
6	•					**2**	**1**	**0**

Subproblem dependency

- Subproblems are $ED(i,j)$, for $0 \leq i \leq m$, $0 \leq j \leq n$
- Table of $(m+1) \cdot (n+1)$ values
- Like LCS, $ED(i,j)$ depends on $ED(i{+}1,j{+}1)$, $ED(i,j{+}1)$, $ED(i{+}1,j)$
- No dependency for $ED(m,n)$ — start at bottom right and fill by row, column or diagonal

		0	1	2	3	4	5	6
		s	e	c	r	e	t	•
0	b				**4**	**4**	**5**	**6**
1	i				**3**	**3**	**4**	**5**
2	s				**2**	**2**	**3**	**4**
3	e				**2**	**1**	**2**	**3**
4	c				**2**	**1**	**1**	**2**
5	t				**2**	**1**	**0**	**1**
6	•				**3**	**2**	**1**	**0**

Subproblem dependency

- Subproblems are $ED(i, j)$, for $0 \leq i \leq m$, $0 \leq j \leq n$
- Table of $(m + 1) \cdot (n + 1)$ values
- Like LCS, $ED(i, j)$ depends on $ED(i+1, j+1)$, $ED(i, j+1)$, $ED(i+1, j)$
- No dependency for $ED(m, n)$ — start at bottom right and fill by row, column or diagonal

		0	1	2	3	4	5	6
		s	e	c	r	e	t	•
0	b			**4**	**4**	**4**	**5**	**6**
1	i			**3**	**3**	**3**	**4**	**5**
2	s			**3**	**2**	**2**	**3**	**4**
3	e			**3**	**2**	**1**	**2**	**3**
4	c			**2**	**2**	**1**	**1**	**2**
5	t			**3**	**2**	**1**	**0**	**1**
6	•			**4**	**3**	**2**	**1**	**0**

Subproblem dependency

- Subproblems are $ED(i, j)$, for $0 \leq i \leq m$, $0 \leq j \leq n$
- Table of $(m + 1) \cdot (n + 1)$ values
- Like LCS, $ED(i, j)$ depends on $ED(i+1, j+1)$, $ED(i, j+1)$, $ED(i+1, j)$
- No dependency for $ED(m, n)$ — start at bottom right and fill by row, column or diagonal

		0	1	2	3	4	5	6
		s	e	c	r	e	t	•
0	b		4	4	4	4	5	6
1	i		4	3	3	3	4	5
2	s		3	3	2	2	3	4
3	e		2	3	2	1	2	3
4	c		3	2	2	1	1	2
5	t		4	3	2	1	0	1
6	•		5	4	3	2	1	0

Subproblem dependency

- Subproblems are $ED(i, j)$, for $0 \leq i \leq m$, $0 \leq j \leq n$
- Table of $(m + 1) \cdot (n + 1)$ values
- Like LCS, $ED(i, j)$ depends on $ED(i+1, j+1)$, $ED(i, j+1)$, $ED(i+1, j)$
- No dependency for $ED(m, n)$ — start at bottom right and fill by row, column or diagonal

		0	1	2	3	4	5	6
		s	e	c	r	e	t	•
0	b	4	4	4	4	4	5	6
1	i	3	4	3	3	3	4	5
2	s	2	3	3	2	2	3	4
3	e	3	2	3	2	1	2	3
4	c	4	3	2	2	1	1	2
5	t	5	4	3	2	1	0	1
6	•	6	5	4	3	2	1	0

For entire string bisect and secret we need to make minimum of 4 operations to make them equal

Subproblem dependency

- Subproblems are $ED(i,j)$, for $0 \leq i \leq m$, $0 \leq j \leq n$
- Table of $(m+1) \cdot (n+1)$ values
- Like LCS, $ED(i,j)$ depends on $ED(i{+}1,j{+}1)$, $ED(i,j{+}1)$, $ED(i{+}1,j)$
- No dependency for $ED(m,n)$ — start at bottom right and fill by row, column or diagonal

Reading off the solution

- Transform `bisect` to `secret`

		0	1	2	3	4	5	6
		s	e	c	r	e	t	•
0	b	4	4	4	4	4	5	6
1	i	3	4	3	3	3	4	5
2	s	2	3	3	2	2	3	4
3	e	3	2	3	2	1	2	3
4	c	4	3	2	2	1	1	2
5	t	5	4	3	2	1	0	1
6	•	6	5	4	3	2	1	0

Subproblem dependency

- Subproblems are $ED(i, j)$, for $0 \leq i \leq m$, $0 \leq j \leq n$
- Table of $(m+1) \cdot (n+1)$ values
- Like LCS, $ED(i, j)$ depends on $ED(i{+}1, j{+}1)$, $ED(i, j{+}1)$, $ED(i{+}1, j)$
- No dependency for $ED(m, n)$ — start at bottom right and fill by row, column or diagonal

Reading off the solution

- Transform `bisect` to `secret`
- Delete `b`

		0	1	2	3	4	5	6
		s	e	c	r	e	t	•
0	b	4	4	4	4	4	5	6
1	i	3	4	3	3	3	4	5
2	s	2	3	3	2	2	3	4
3	e	3	2	3	2	1	2	3
4	c	4	3	2	2	1	1	2
5	t	5	4	3	2	1	0	1
6	•	6	5	4	3	2	1	0

Subproblem dependency

- Subproblems are $ED(i,j)$, for $0 \leq i \leq m$, $0 \leq j \leq n$
- Table of $(m+1) \cdot (n+1)$ values
- Like LCS, $ED(i,j)$ depends on $ED(i{+}1,j{+}1)$, $ED(i,j{+}1)$, $ED(i{+}1,j)$
- No dependency for $ED(m,n)$ — start at bottom right and fill by row, column or diagonal

Reading off the solution

- Transform `bisect` to `secret`
- Delete `b` , Delete `i`

		0	1	2	3	4	5	6
		s	e	c	r	e	t	•
0	b	4	4	4	4	4	5	6
1	i	3	4	3	3	3	4	5
2	s	2	3	3	2	2	3	4
3	e	3	2	3	2	1	2	3
4	c	4	3	2	2	1	1	2
5	t	5	4	3	2	1	0	1
6	•	6	5	4	3	2	1	0

Subproblem dependency

- Subproblems are $ED(i,j)$, for $0 \leq i \leq m$, $0 \leq j \leq n$
- Table of $(m+1) \cdot (n+1)$ values
- Like LCS, $ED(i,j)$ depends on $ED(i{+}1,j{+}1)$, $ED(i,j{+}1)$, $ED(i{+}1,j)$
- No dependency for $ED(m,n)$ — start at bottom right and fill by row, column or diagonal

Reading off the solution

- Transform `bisect` to `secret`
- Delete `b` , Delete `i` , Insert `r`

		0	1	2	3	4	5	6
		s	e	c	r	e	t	•
0	b	4	4	4	4	4	5	6
1	i	3	4	3	3	3	4	5
2	s	2	3	3	2	2	3	4
3	e	3	2	3	2	1	2	3
4	c	4	3	2	2	1	1	2
5	t	5	4	3	2	1	0	1
6	•	6	5	4	3	2	1	0

Subproblem dependency

- Subproblems are $ED(i, j)$, for $0 \leq i \leq m$, $0 \leq j \leq n$
- Table of $(m+1) \cdot (n+1)$ values
- Like LCS, $ED(i, j)$ depends on $ED(i+1, j+1)$, $ED(i, j+1)$, $ED(i+1, j)$
- No dependency for $ED(m, n)$ — start at bottom right and fill by row, column or diagonal

Reading off the solution

diagonal line substitution or when a[i] = b[j]

- Transform `bisect` to `secret`
- Delete `b` , Delete `i` , Insert `r` , Insert `e`

		0	1	2	3	4	5	6
		s	e	c	r	e	t	•
0	b	4	4	4	4	4	5	6
1	i	3	4	3	3	3	4	5
2	s	2	3	3	2	2	3	4
3	e	3	2	3	2	1	2	3
4	c	4	3	2	2	1	1	2
5	t	5	4	3	2	1	0	1
6	•	6	5	4	3	2	1	0

Implementation

```
def ED(u,v):
  import numpy as np
  (m,n) = (len(u),len(v))
  ed = np.zeros((m+1,n+1))

  for i in range(m-1,-1,-1):
    ed[i,n] = m-i
  for j in range(n-1,-1,-1):
    ed[m,j] = n-j

  for j in range(n-1,-1,-1):
    for i in range(m-1,-1,-1):
      if u[i] == v[j]:
        ed[i,j] = ed[i+1,j+1]
      else:
        ed[i,j] = 1 + min(ed[i+1,j+1],
                          ed[i,j+1],
                          ed[i+1,j])
  return(ed[0,0])
```

Implementation

```
def ED(u,v):
  import numpy as np
  (m,n) = (len(u),len(v))
  ed = np.zeros((m+1,n+1))

  for i in range(m-1,-1,-1):
    ed[i,n] = m-i
  for j in range(n-1,-1,-1):
    ed[m,j] = n-j

  for j in range(n-1,-1,-1):
    for i in range(m-1,-1,-1):
      if u[i] == v[j]:
        ed[i,j] = ed[i+1,j+1]
      else:
        ed[i,j] = 1 + min(ed[i+1,j+1],
                          ed[i,j+1],
                          ed[i+1,j])
  return(ed[0,0])
```

Complexity

Implementation

```
def ED(u,v):
  import numpy as np
  (m,n) = (len(u),len(v))
  ed = np.zeros((m+1,n+1))

  for i in range(m-1,-1,-1):
    ed[i,n] = m-i
  for j in range(n-1,-1,-1):
    ed[m,j] = n-j

  for j in range(n-1,-1,-1):
    for i in range(m-1,-1,-1):
      if u[i] == v[j]:
        ed[i,j] = ed[i+1,j+1]
      else:
        ed[i,j] = 1 + min(ed[i+1,j+1],
                          ed[i,j+1],
                          ed[i+1,j])
  return(ed[0,0])
```

Complexity

- Again $O(mn)$, using dynamic programming or memoization

Implementation

```
def ED(u,v):
  import numpy as np
  (m,n) = (len(u),len(v))
  ed = np.zeros((m+1,n+1))

  for i in range(m-1,-1,-1):
    ed[i,n] = m-i
  for j in range(n-1,-1,-1):
    ed[m,j] = n-j

  for j in range(n-1,-1,-1):
    for i in range(m-1,-1,-1):
      if u[i] == v[j]:
        ed[i,j] = ed[i+1,j+1]
      else:
        ed[i,j] = 1 + min(ed[i+1,j+1],
                          ed[i,j+1],
                          ed[i+1,j])
  return(ed[0,0])
```

Complexity

- Again $O(mn)$, using dynamic programming or memoization
  - Fill a table of size $O(mn)$
  - Each table entry takes constant time to compute