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~~ can~~d~~ ~~it~~b~~s~~e used in designing algorithms"
- insert, ~~delete~~, ~~subs~~t~~i~~tute

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~~ can~~d~~ ~~it~~b~~s~~e 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~~ can~~d~~ ~~it~~b~~s~~e used in designing algorithms”
- insert, ~~delete~~, ~~subst~~itute

Edit distance

- Minimum number of edit operations needed

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~~ can~~d~~ ~~it~~b~~s~~e used in designing algorithms"
- insert, ~~delete~~, ~~subst~~itute

Edit distance

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

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~~ can~~d~~ ~~it~~b~~s~~e 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

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, v
  - Minimum number of deletes needed to make them equal

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
  - `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`

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`
  - 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

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

- Recall LCS

Inductive structure for edit distance

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

- 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

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

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

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

- 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 $a_i a_{i+1} \ldots a_{m-1}$, $b_j b_{j+1} \ldots b_{n-1}$

- 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 $ED(i+1,j+1)$, $ED(i,j+1)$, $ED(i+1,j)$

		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 $ED(i{+}1,j{+}1)$, $ED(i,j{+}1)$, $ED(i{+}1,j)$
- 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							**6**
1	i							**5**
2	s							**4**
3	e							**3**
4	c							**2**
5	t							**1**
6	•							**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						**1**	**2**
5	t						**0**	**1**
6	•						**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	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

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

- 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