CSE 202: Design and Analysis of Algorithms

Lecture 6

Instructor: Kamalika Chaudhuri

Announcements

- Homework 1 solutions are up!
- Homework 2 is out, due in class Feb 2nd

Last Class: Three steps of Dynamic Programming

Main Steps:

1. Divide the problem into **subtasks**
2. Define the subtasks **recursively** (express larger subtasks in terms of smaller ones)
3. Find the **right order** for solving the subtasks (but do not solve them recursively!)

String reconstruction

Given: document x[1..n] : an array of characters
dictionary function dict(w): returns true if w is a valid word

Is x a sequence of valid words ?

Example:

x = anonymousarrayofletters : **True**

x = anhuymousarrayofhetters : **False**

String reconstruction

Given: document x[1..n] : an array of characters
dictionary function dict(w): returns true if w is a valid word

Is x a sequence of valid words ?

Example:

x = anonymousarrayofletters : **True**

x = anhuymousarrayofhetters : **False**

STEP 1: Define subtask

S(k) = True if x[1..k] is a valid sequence of words

False otherwise

Output of algorithm = S(n)

String reconstruction

Given: document x[1..n] : an array of characters
dictionary function dict(w): returns true if w is a valid word

Is x a sequence of valid words ?

Example:

x = anonymousarrayofletters : **True**

x = anhuymousarrayofhetters : **False**

STEP 1: Define subtask

S(k) = True if x[1..k] is a valid sequence of words

False otherwise

Output of algorithm = S(n)

x		A	N	O	N	Y	M	O	U	S	A	R	R	A	Y	O	F	L	E	T	T	E	R	S
k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
S																								

String reconstruction

Given: document x[1..n] : an array of characters
dictionary function dict(w): returns true if w is a valid word

Is x a sequence of valid words ?

STEP 1: Define Subtask

S(k) = True if x[1..k] is a valid sequence of words
False otherwise

x		A	N	O	N	Y	M	O	U	S	A	R	R	A	Y	O	F	L	E	T	T	E	R	S
k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
S																								

String reconstruction

Given: document x[1..n] : an array of characters
dictionary function dict(w): returns true if w is a valid word

Is x a sequence of valid words ?

STEP 1: Define Subtask

S(k) = True if x[1..k] is a valid sequence of words
False otherwise

STEP 2: Express Recursively

S(k) = True iff $\exists$ j < k s.t. S(j) is True, and x[j+1..k] is a valid word

x		A	N	O	N	Y	M	O	U	S	A	R	R	A	Y	O	F	L	E	T	T	E	R	S
k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
S																								

String reconstruction

Given: document x[1..n] : an array of characters
dictionary function dict(w): returns true if w is a valid word

Is x a sequence of valid words ?

STEP 1: Define Subtask

S(k) = True if x[1..k] is a valid sequence of words
False otherwise

STEP 2: Express Recursively

S(k) = True iff $\exists$ j < k s.t. S(j) is True, and x[j+1..k] is a valid word

STEP 3: Order of Subtasks

S(1), S(2), S(3), ..., S(n) [Do not solve recursively!]

x		A	N	O	N	Y	M	O	U	S	A	R	R	A	Y	O	F	L	E	T	T	E	R	S
k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
S																								

String reconstruction

Given: document x[1..n] : an array of characters
dictionary function dict(w): returns true if w is a valid word

Is x a sequence of valid words ?

STEP 1: Define Subtask

S(k) = True if x[1..k] is a valid sequence of words
False otherwise

STEP 2: Express Recursively

S(k) = True iff $\exists$ j < k s.t. S(j) is True, and x[j+1..k] is a valid word

STEP 3: Order of Subtasks

S(1), S(2), S(3), ..., S(n) [Do not solve recursively!]

x		A	N	O	N	Y	M	O	U	S	A	R	R	A	Y	O	F	L	E	T	T	E	R	S
k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
S		T																						

String reconstruction

Given: document x[1..n] : an array of characters
dictionary function dict(w): returns true if w is a valid word

Is x a sequence of valid words ?

STEP 1: Define Subtask

S(k) = True if x[1..k] is a valid sequence of words
False otherwise

STEP 2: Express Recursively

S(k) = True iff $\exists$ j < k s.t. S(j) is True, and x[j+1..k] is a valid word

STEP 3: Order of Subtasks

S(1), S(2), S(3), ..., S(n) [Do not solve recursively!]

x		A	N	O	N	Y	M	O	U	S	A	R	R	A	Y	O	F	L	E	T	T	E	R	S
k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
S		T	T																					

String reconstruction

Given: document x[1..n] : an array of characters
dictionary function dict(w): returns true if w is a valid word

Is x a sequence of valid words ?

STEP 1: Define Subtask

S(k) = True if x[1..k] is a valid sequence of words
False otherwise

STEP 2: Express Recursively

S(k) = True iff $\exists$ j < k s.t. S(j) is True, and x[j+1..k] is a valid word

STEP 3: Order of Subtasks

S(1), S(2), S(3), ..., S(n) [Do not solve recursively!]

x		A	N	O	N	Y	M	O	U	S	A	R	R	A	Y	O	F	L	E	T	T	E	R	S
k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
S		T	T	T																				

String reconstruction

Given: document x[1..n] : an array of characters
dictionary function dict(w): returns true if w is a valid word

Is x a sequence of valid words ?

STEP 1: Define Subtask

S(k) = True if x[1..k] is a valid sequence of words
False otherwise

STEP 2: Express Recursively

S(k) = True iff $\exists$ j < k s.t. S(j) is True, and x[j+1..k] is a valid word

STEP 3: Order of Subtasks

S(1), S(2), S(3), ..., S(n) [Do not solve recursively!]

x		A	N	O	N	Y	M	O	U	S	A	R	R	A	Y	O	F	L	E	T	T	E	R	S
k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
S		T	T	T	T																			

String reconstruction

Given: document x[1..n] : an array of characters
dictionary function dict(w): returns true if w is a valid word

Is x a sequence of valid words ?

STEP 1: Define Subtask

S(k) = True if x[1..k] is a valid sequence of words
False otherwise

STEP 2: Express Recursively

S(k) = True iff $\exists$ j < k s.t. S(j) is True, and x[j+1..k] is a valid word

STEP 3: Order of Subtasks

S(1), S(2), S(3), ..., S(n) [Do not solve recursively!]

x		A	N	O	N	Y	M	O	U	S	A	R	R	A	Y	O	F	L	E	T	T	E	R	S
k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
S		T	T	T	T	F																		

String reconstruction

Given: document x[1..n] : an array of characters
dictionary function dict(w): returns true if w is a valid word

Is x a sequence of valid words ?

STEP 1: Define Subtask

S(k) = True if x[1..k] is a valid sequence of words
False otherwise

STEP 2: Express Recursively

S(k) = True iff $\exists$ j < k s.t. S(j) is True, and x[j+1..k] is a valid word

STEP 3: Order of Subtasks

S(1), S(2), S(3), ..., S(n) [Do not solve recursively!]

x		A	N	O	N	Y	M	O	U	S	A	R	R	A	Y	O	F	L	E	T	T	E	R	S
k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
S		T	T	T	T	F	F																	

String reconstruction

Given: document x[1..n] : an array of characters
dictionary function dict(w): returns true if w is a valid word

Is x a sequence of valid words ?

STEP 1: Define Subtask

S(k) = True if x[1..k] is a valid sequence of words
False otherwise

STEP 2: Express Recursively

S(k) = True iff $\exists$ j < k s.t. S(j) is True, and x[j+1..k] is a valid word

STEP 3: Order of Subtasks

S(1), S(2), S(3), ..., S(n) [Do not solve recursively!]

x		A	N	O	N	Y	M	O	U	S	A	R	R	A	Y	O	F	L	E	T	T	E	R	S
k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
S		T	T	T	T	F	F	F																

String reconstruction

Given: document x[1..n] : an array of characters
dictionary function dict(w): returns true if w is a valid word

Is x a sequence of valid words ?

STEP 1: Define Subtask

S(k) = True if x[1..k] is a valid sequence of words
False otherwise

STEP 2: Express Recursively

S(k) = True iff $\exists$ j < k s.t. S(j) is True, and x[j+1..k] is a valid word

STEP 3: Order of Subtasks

S(1), S(2), S(3), ..., S(n) [Do not solve recursively!]

x		A	N	O	N	Y	M	O	U	S	A	R	R	A	Y	O	F	L	E	T	T	E	R	S
k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
S		T	T	T	T	F	F	F	F															

String reconstruction

Given: document x[1..n] : an array of characters
dictionary function dict(w): returns true if w is a valid word

Is x a sequence of valid words ?

STEP 1: Define Subtask

S(k) = True if x[1..k] is a valid sequence of words
False otherwise

STEP 2: Express Recursively

S(k) = True iff $\exists$ j < k s.t. S(j) is True, and x[j+1..k] is a valid word

STEP 3: Order of Subtasks

S(1), S(2), S(3), ..., S(n) [Do not solve recursively!]

x		A	N	O	N	Y	M	O	U	S	A	R	R	A	Y	O	F	L	E	T	T	E	R	S
k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
S		T	T	T	T	F	F	F	F	T														

String reconstruction

Given: document x[1..n] : an array of characters
dictionary function dict(w): returns true if w is a valid word

Is x a sequence of valid words ?

STEP 1: Define Subtask

S(k) = True if x[1..k] is a valid sequence of words
False otherwise

STEP 2: Express Recursively

S(k) = True iff $\exists$ j < k s.t. S(j) is True, and x[j+1..k] is a valid word

STEP 3: Order of Subtasks

S(1), S(2), S(3), ..., S(n) [Do not solve recursively!]

x		A	N	O	N	Y	M	O	U	S	A	R	R	A	Y	O	F	L	E	T	T	E	R	S
k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
S		T	T	T	T	F	F	F	F	T	T													

String reconstruction

Given: document x[1..n] : an array of characters
dictionary function dict(w): returns true if w is a valid word

Is x a sequence of valid words ?

STEP 1: Define Subtask

S(k) = True if x[1..k] is a valid sequence of words
False otherwise

STEP 2: Express Recursively

S(k) = True iff $\exists$ j < k s.t. S(j) is True, and x[j+1..k] is a valid word

STEP 3: Order of Subtasks

S(1), S(2), S(3), ..., S(n) [Do not solve recursively!]

x		A	N	O	N	Y	M	O	U	S	A	R	R	A	Y	O	F	L	E	T	T	E	R	S
k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
S		T	T	T	T	F	F	F	F	T	T	F												

String reconstruction

Given: document $x[1..n]$: an array of characters
dictionary function dict(w): returns true if w is a valid word

Is x a sequence of valid words ?

STEP 1: Define Subtask

S(k) = True   if x[1..k] is a valid sequence of words
False   otherwise

STEP 2: Express Recursively

S(k) = True iff $\exists$ j < k s.t. S(j) is True, and x[j+1..k] is a valid word

STEP 3: Order of Subtasks

S(1), S(2), S(3), ..., S(n) [Do not solve recursively!]

x		A	N	O	N	Y	M	O	U	S	A	R	R	A	Y	O	F	L	E	T	T	E	R	S
k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
S		T	T	T	T	F	F	F	F	T	T	F	F											

String reconstruction

Given: document x[1..n] : an array of characters
dictionary function dict(w): returns true if w is a valid word

Is x a sequence of valid words ?

STEP 1: Define Subtask

S(k) = True if x[1..k] is a valid sequence of words
False otherwise

STEP 2: Express Recursively

S(k) = True iff $\exists$ j < k s.t. S(j) is True, and x[j+1..k] is a valid word

STEP 3: Order of Subtasks

S(1), S(2), S(3), ..., S(n) [Do not solve recursively!]

x		A	N	O	N	Y	M	O	U	S	A	R	R	A	Y	O	F	L	E	T	T	E	R	S
k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
S		T	T	T	T	F	F	F	F	T	T	F	F	F										

String reconstruction

Given: document x[1..n] : an array of characters
dictionary function dict(w): returns true if w is a valid word

Is x a sequence of valid words ?

STEP 1: Define Subtask

S(k) = True if x[1..k] is a valid sequence of words
False otherwise

STEP 2: Express Recursively

S(k) = True iff $\exists$ j < k s.t. S(j) is True, and x[j+1..k] is a valid word

STEP 3: Order of Subtasks

S(1), S(2), S(3), ..., S(n) [Do not solve recursively!]

x		A	N	O	N	Y	M	O	U	S	A	R	R	A	Y	O	F	L	E	T	T	E	R	S
k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
S		T	T	T	T	F	F	F	F	T	T	F	F	F	T									

String reconstruction

Given: document x[1..n] : an array of characters
dictionary function dict(w): returns true if w is a valid word

Is x a sequence of valid words ?

STEP 1: Define Subtask

S(k) = True if x[1..k] is a valid sequence of words
False otherwise

STEP 2: Express Recursively

S(k) = True iff $\exists$ j < k s.t. S(j) is True, and x[j+1..k] is a valid word

STEP 3: Order of Subtasks

S(1), S(2), S(3), ..., S(n) [Do not solve recursively!]

x		A	N	O	N	Y	M	O	U	S	A	R	R	A	Y	O	F	L	E	T	T	E	R	S
k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
S		T	T	T	T	F	F	F	F	T	T	F	F	F	T	F	T	F	F	T	F	F	T	T

String reconstruction

Given: document x[1..n] : an array of characters
dictionary function dict(w): returns true if w is a valid word

Is x a sequence of valid words ?

STEP 1: Define Subtask

S(k) = True if x[1..k] is a valid sequence of words
False otherwise

STEP 2: Express Recursively

S(k) = True iff $\exists$ j < k s.t. S(j) is True, and x[j+1..k] is a valid word

STEP 3: Order of Subtasks

S(1), S(2), S(3), ..., S(n)

Algorithm:

```
S[0] = true
for k = 1 to n:
  S[k] = false
  for j = 1 to k:
    if S[j-1] and dict(x[j..k])
      S[k] = true
```

Reconstructing Document:

Define array D(1,..n):
If S(k) = true, then D(k) = starting position of the word that ends at x[k]

Reconstruct text by following these pointers.

String reconstruction

Given: document x[1..n] : an array of characters
dictionary function dict(w): returns true if w is a valid word

Is x a sequence of valid words ?

STEP 1: Define Subtask

S(k) = True if x[1..k] is a valid sequence of words
= False otherwise

STEP 2: Express Recursively

S(k) = True iff there is j < k s.t. S(j) is True, and x[j+1..k] is a valid word

STEP 3: Order of Subtasks

S(1), S(2), S(3), ..., S(n)

Reconstructing Document:

Define array D(1,..n):
If S(k) = True, then D(k) = starting position of the word that ends at x[k]

Reconstruct text by following these pointers.

x		A	N	O	N	Y	M	O	U	S	A	R	R	A	Y	O	F	L	E	T	T	E	R	S
k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
S		T	T	T	T	F	F	F	F	T	T	F	F	F	T	F	T	F	F	T	F	F	T	T
D		1				-	-	-	-			-	-	-		-		-	-		-	-		

String reconstruction

Given: document x[1..n] : an array of characters
dictionary function dict(w): returns true if w is a valid word

Is x a sequence of valid words ?

STEP 1: Define Subtask

S(k) = True if x[1..k] is a valid sequence of words
= False otherwise

STEP 2: Express Recursively

S(k) = True iff there is $j < k$ s.t. S(j) is True, and x[j+1..k] is a valid word

STEP 3: Order of Subtasks

S(1), S(2), S(3), ..., S(n)

Reconstructing Document:

Define array D(1,..n):
If S(k) = True, then D(k) = starting position of the word that ends at x[k]

Reconstruct text by following these pointers.

x		A	N	O	N	Y	M	O	U	S	A	R	R	A	Y	O	F	L	E	T	T	E	R	S
k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
S		T	T	T	T	F	F	F	F	T	T	F	F	F	T	F	T	F	F	T	F	F	T	T
D		1	1			-	-	-	-			-	-	-		-		-	-		-	-		

String reconstruction

Given: document x[1..n] : an array of characters
dictionary function dict(w): returns true if w is a valid word

Is x a sequence of valid words ?

STEP 1: Define Subtask

S(k) = True if x[1..k] is a valid sequence of words
= False otherwise

STEP 2: Express Recursively

S(k) = True iff there is j < k s.t. S(j) is True, and x[j+1..k] is a valid word

STEP 3: Order of Subtasks

S(1), S(2), S(3), ..., S(n)

Reconstructing Document:

Define array D(1,..n):
If S(k) = True, then D(k) = starting position of the word that ends at x[k]

Reconstruct text by following these pointers.

x		A	N	O	N	Y	M	O	U	S	A	R	R	A	Y	O	F	L	E	T	T	E	R	S
k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
S		T	T	T	T	F	F	F	F	T	T	F	F	F	T	F	T	F	F	T	F	F	T	T
D		1	1	2		-	-	-	-			-	-	-		-		-	-		-	-		

String reconstruction

Given: document x[1..n] : an array of characters
dictionary function dict(w): returns true if w is a valid word

Is x a sequence of valid words ?

STEP 1: Define Subtask

S(k) = True if x[1..k] is a valid sequence of words
= False otherwise

STEP 2: Express Recursively

S(k) = True iff there is j < k s.t. S(j) is True, and x[j+1..k] is a valid word

STEP 3: Order of Subtasks

S(1), S(2), S(3), ..., S(n)

Reconstructing Document:

Define array D(1,..n):
If S(k) = True, then D(k) = starting position of the word that ends at x[k]

Reconstruct text by following these pointers.

x		A	N	O	N	Y	M	O	U	S	A	R	R	A	Y	O	F	L	E	T	T	E	R	S
k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
S		T	T	T	T	F	F	F	F	T	T	F	F	F	T	F	T	F	F	T	F	F	T	T
D		1	1	2	3	-	-	-	-			-	-	-		-		-	-		-	-		

String reconstruction

Given: document x[1..n] : an array of characters
dictionary function dict(w): returns true if w is a valid word

Is x a sequence of valid words ?

STEP 1: Define Subtask

S(k) = True if x[1..k] is a valid sequence of words
= False otherwise

STEP 2: Express Recursively

S(k) = True iff there is j < k s.t. S(j) is True, and x[j+1..k] is a valid word

STEP 3: Order of Subtasks

S(1), S(2), S(3), ..., S(n)

Reconstructing Document:

Define array D(1,..n):
If S(k) = True, then D(k) = starting position of the word that ends at x[k]

Reconstruct text by following these pointers.

x		A	N	O	N	Y	M	O	U	S	A	R	R	A	Y	O	F	L	E	T	T	E	R	S
k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
S		T	T	T	T	F	F	F	F	T	T	F	F	F	T	F	T	F	F	T	F	F	T	T
D		1	1	2	3	-	-	-	-	1		-	-	-		-		-	-		-	-		

String reconstruction

Given: document x[1..n] : an array of characters
dictionary function dict(w): returns true if w is a valid word

Is x a sequence of valid words ?

STEP 1: Define Subtask

S(k) = True if x[1..k] is a valid sequence of words
= False otherwise

STEP 2: Express Recursively

S(k) = True iff there is j < k s.t. S(j) is True,
and x[j+1..k] is a valid word

STEP 3: Order of Subtasks

S(1), S(2), S(3), ..., S(n)

Reconstructing Document:

Define array D(1,..n):
If S(k) = True, then D(k) = starting position of the word that ends at x[k]

Reconstruct text by following these pointers.

x		A	N	O	N	Y	M	O	U	S	A	R	R	A	Y	O	F	L	E	T	T	E	R	S
k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
S		T	T	T	T	F	F	F	F	T	T	F	F	F	T	F	T	F	F	T	F	F	T	T
D		1	1	2	3	-	-	-	-	1	10	-	-	-	10	-	15	-	-	17	-	-	17	17

String reconstruction

Given: document x[1..n] : an array of characters
dictionary function dict(w): returns true if w is a valid word

Is x a sequence of valid words ?

STEP 1: Define Subtask

S(k) = True if x[1..k] is a valid sequence of words
= False otherwise

STEP 2: Express Recursively

S(k) = True iff there is j < k s.t. S(j) is True, and x[j+1..k] is a valid word

STEP 3: Order of Subtasks

S(1), S(2), S(3), ..., S(n)

Reconstructing Document:

Define array D(1,..n):
If S(k) = True, then D(k) = starting position of the word that ends at x[k]

Reconstruct text by following these pointers.

x		A	N	O	N	Y	M	O	U	S	A	R	R	A	Y	O	F	L	E	T	T	E	R	S
k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
S		T	T	T	T	F	F	F	F	T	T	F	F	F	T	F	T	F	F	T	F	F	T	T
D		1	1	2	3	-	-	-	-	1	10	-	-	-	10	-	15	-	-	17	-	-	17	17

Dynamic Programming

- String Reconstruction
- Longest Common Subsequence
- ...

Longest Common Subsequence (LCS)

Problem: Given two sequences x[1..m] and y[1..n], find their longest common subsequence

Example:

x = A,C,G,T,A,G
y = G,T, C,C,A,C

LCS(x, y) = G,T,A

Longest Common Subsequence (LCS)

Problem: Given two sequences x[1..m] and y[1..n], find their longest common subsequence

Example:

x = A,C,G,T,A,G
y = G,T, C,C,A,C

LCS(x, y) = G,T,A

Structure:

x = A,C,G, **T**,
y = G,**T**,

Longest Common Subsequence (LCS)

Problem: Given two sequences x[1..m] and y[1..n], find their longest common subsequence

Example:

x = A,C,G,T,A,G
y = G,T, C,C,A,C

LCS(x, y) = G,T,A

Structure:

x = A,C,G, **T**,
y = G,**T**,

If x[i] = y[j], then
LCS(x[1..i], y[1..j]) = LCS(x[1..i-1], y[1..j-1]) + x[i]

Longest Common Subsequence (LCS)

Problem: Given two sequences x[1..m] and y[1..n], find their longest common subsequence

Example:

x = A,C,G,T,A,G
y = G,T, C,C,A,C

LCS(x, y) = G,T,A

Structure:

x = A,C,G, **T**,
y = G,**T**,

If x[i] = y[j], then
LCS(x[1..i], y[1..j]) = LCS(x[1..i-1], y[1..j-1]) + x[i]

x = A,C,G,T, **A**
y = G,**T**,

Longest Common Subsequence (LCS)

Problem: Given two sequences x[1..m] and y[1..n], find their longest common subsequence

Example:

x = A,C,G,T,A,G
y = G,T, C,C,A,C

LCS(x, y) = G,T,A

Structure:

x = A,C,G, **T**,
y = G,**T**,

If x[i] = y[j], then
LCS(x[1..i], y[1..j]) = LCS(x[1..i-1], y[1..j-1]) + x[i]

x = A,C,G,T, **A**
y = G,**T**,

Otherwise,
LCS(x[1..i], y[1..j]) = max(LCS(x[1..i-1], y[1..j]),
LCS(x[1..i], y[1..j-1]))

Longest Common Subsequence (LCS)

Problem: Given two sequences x[1..m] and y[1..n], find their longest common subsequence

Example:

x = A,C,G,T,A,G
y = G,T, C,C,A,C

LCS(x, y) = G,T,A

STEP 1: Define subtasks

S(i,j) = Length of LCS of x[1..i] and y[1..j]

Output of algorithm = S(m,n)

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1									
T	2									
G	3									
A	4									
C	5									
A	6									
G	7									
T	8									
T	9									

Longest Common Subsequence (LCS)

Problem: Given two sequences x[1..m] and y[1..n], find their longest common subsequence

Example:

x = A,C,G,T,A,G
y = G,T, C,C,A,C

LCS(x, y) = G,T,A

STEP 1: Define subtasks

S(i,j) = Length of LCS of x[1..i] and y[1..j]

Output of algorithm = S(m,n)

STEP 2: Express recursively

S(i,j) = S(i-1,j-1) + 1, if x[i] = y[j]
= max(S(i-1,j), S(i,j-1)), ow

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1									
T	2									
G	3									
A	4									
C	5									
A	6									
G	7									
T	8									
T	9									

Longest Common Subsequence (LCS)

Problem: Given two sequences x[1..m] and y[1..n], find their longest common subsequence

Example:

x = A,C,G,T,A,G
y = G,T, C,C,A,C

LCS(x, y) = G,T,A

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1									
T	2									
G	3									
A	4									
C	5									
A	6									
G	7									
T	8									
T	9									

STEP 1: Define subtasks

S(i,j) = Length of LCS of x[1..i] and y[1..j]

Output of algorithm = S(m,n)

STEP 2: Express recursively

S(i,j) = S(i-1,j-1) + 1, if x[i] = y[j]
= max(S(i-1,j), S(i,j-1)), ow

STEP 3: Order of subtasks

Row by row, left to right

Longest Common Subsequence (LCS)

Problem: Given two sequences x[1..m] and y[1..n], find their longest common subsequence

Example:

x = A,C,G,T,A,G
y = G,T, C,C,A,C

LCS(x, y) = G,T,A

STEP 1: Define subtasks

S(i,j) = Length of LCS of x[1..i] and y[1..j]

Output of algorithm = S(m,n)

STEP 2: Express recursively

S(i,j) = S(i-1,j-1) + 1, if x[i] = y[j]
= max(S(i-1,j), S(i,j-1)), ow

STEP 3: Order of subtasks

Row by row, left to right

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1									
T	2									
G	3									
A	4									
C	5									
A	6									
G	7									
T	8									
T	9									

Base Case: Row 0, Column 0

Longest Common Subsequence (LCS)

Problem: Given two sequences x[1..m] and y[1..n], find their longest common subsequence

Example:

x = A,C,G,T,A,G
y = G,T, C,C,A,C

LCS(x, y) = G,T,A

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1	0	0	0						
T	2									
G	3									
A	4									
C	5									
A	6									
G	7									
T	8									
T	9									

STEP 1: Define subtasks

S(i,j) = Length of LCS of x[1..i] and y[1..j]

Output of algorithm = S(m,n)

STEP 2: Express recursively

S(i,j) = S(i-1,j-1) + 1, if x[i] = y[j]
= max(S(i-1,j), S(i,j-1)), ow

STEP 3: Order of subtasks

Row by row, left to right

Base Case: Row 0, Column 0

Longest Common Subsequence (LCS)

Problem: Given two sequences x[1..m] and y[1..n], find their longest common subsequence

Example:

x = A,C,G,T,A,G
y = G,T, C,C,A,C

LCS(x, y) = G,T,A

STEP 1: Define subtasks

S(i,j) = Length of LCS of x[1..i] and y[1..j]

Output of algorithm = S(m,n)

STEP 2: Express recursively

S(i,j) = S(i-1,j-1) + 1, if x[i] = y[j]
= max(S(i-1,j), S(i,j-1)), ow

STEP 3: Order of subtasks

Row by row, left to right

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1	0	0	0	1					
T	2									
G	3									
A	4									
C	5									
A	6									
G	7									
T	8									
T	9									

Base Case: Row 0, Column 0

Longest Common Subsequence (LCS)

Problem: Given two sequences x[1..m] and y[1..n], find their longest common subsequence

Example:

x = A,C,G,T,A,G
y = G,T, C,C,A,C

LCS(x, y) = G,T,A

		A	**C**	**T**	**G**	**G**	**C**	**T**	**A**	**G**
	0	1	2	3	4	5	6	7	8	9
G	1	0	0	0	1	1	1	1	1	1
T	2									
G	3									
A	4									
C	5									
A	6									
G	7									
T	8									
T	9									

STEP 1: Define subtasks

S(i,j) = Length of LCS of x[1..i] and y[1..j]

Output of algorithm = S(m,n)

STEP 2: Express recursively

S(i,j) = S(i-1,j-1) + 1, if x[i] = y[j]
= max(S(i-1,j), S(i,j-1)), ow

STEP 3: Order of subtasks

Row by row, left to right

Base Case: Row 0, Column 0

Longest Common Subsequence (LCS)

Problem: Given two sequences x[1..m] and y[1..n], find their longest common subsequence

Example:

x = A,C,G,T,A,G
y = G,T, C,C,A,C

LCS(x, y) = G,T,A

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1	0	0	0	1	1	1	1	1	1
T	2	0	0							
G	3									
A	4									
C	5									
A	6									
G	7									
T	8									
T	9									

STEP 1: Define subtasks

S(i,j) = Length of LCS of x[1..i] and y[1..j]

Output of algorithm = S(m,n)

STEP 2: Express recursively

S(i,j) = S(i-1,j-1) + 1, if x[i] = y[j]
= max(S(i-1,j), S(i,j-1)), ow

STEP 3: Order of subtasks

Row by row, left to right

Base Case: Row 0, Column 0

Longest Common Subsequence (LCS)

Problem: Given two sequences x[1..m] and y[1..n], find their longest common subsequence

Example:

x = A,C,G,T,A,G
y = G,T, C,C,A,C

LCS(x, y) = G,T,A

STEP 1: Define subtasks

S(i,j) = Length of LCS of x[1..i] and y[1..j]

Output of algorithm = S(m,n)

STEP 2: Express recursively

S(i,j) = S(i-1,j-1) + 1, if x[i] = y[j]
= max(S(i-1,j), S(i,j-1)), ow

STEP 3: Order of subtasks

Row by row, left to right

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1	0	0	0	1	1	1	1	1	1
T	2	0	0	1						
G	3									
A	4									
C	5									
A	6									
G	7									
T	8									
T	9									

Base Case: Row 0, Column 0

Longest Common Subsequence (LCS)

Problem: Given two sequences x[1..m] and y[1..n], find their longest common subsequence

Example:

x = A,C,G,T,A,G
y = G,T, C,C,A,C

LCS(x, y) = G,T,A

STEP 1: Define subtasks

S(i,j) = Length of LCS of x[1..i] and y[1..j]

Output of algorithm = S(m,n)

STEP 2: Express recursively

S(i,j) = S(i-1,j-1) + 1, if x[i] = y[j]
= max(S(i-1,j), S(i,j-1)), ow

STEP 3: Order of subtasks

Row by row, left to right

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1	0	0	0	1	1	1	1	1	1
T	2	0	0	1	1					
G	3									
A	4									
C	5									
A	6									
G	7									
T	8									
T	9									

Base Case: Row 0, Column 0

Longest Common Subsequence (LCS)

Problem: Given two sequences x[1..m] and y[1..n], find their longest common subsequence

Example:

x = A,C,G,T,A,G
y = G,T, C,C,A,C

LCS(x, y) = G,T,A

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1	0	0	0	1	1	1	1	1	1
T	2	0	0	1	1	1	1			
G	3									
A	4									
C	5									
A	6									
G	7									
T	8									
T	9									

STEP 1: Define subtasks

S(i,j) = Length of LCS of x[1..i] and y[1..j]

Output of algorithm = S(m,n)

STEP 2: Express recursively

S(i,j) = S(i-1,j-1) + 1, if x[i] = y[j]
= max(S(i-1,j), S(i,j-1)), ow

STEP 3: Order of subtasks

Row by row, left to right

Base Case: Row 0, Column 0

Longest Common Subsequence (LCS)

Problem: Given two sequences x[1..m] and y[1..n], find their longest common subsequence

Example:

x = A,C,G,T,A,G
y = G,T, C,C,A,C

LCS(x, y) = G,T,A

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1	0	0	0	1	1	1	1	1	1
T	2	0	0	1	1	1	1	2		
G	3									
A	4									
C	5									
A	6									
G	7									
T	8									
T	9									

STEP 1: Define subtasks

S(i,j) = Length of LCS of x[1..i] and y[1..j]

Output of algorithm = S(m,n)

STEP 2: Express recursively

$$S(i,j) = S(i-1,j-1) + 1, \text{ if } x[i] = y[j]$$
$$= \max(S(i-1,j), S(i,j-1)), \text{ ow}$$

STEP 3: Order of subtasks

Row by row, left to right

Base Case: Row 0, Column 0

Longest Common Subsequence (LCS)

Problem: Given two sequences x[1..m] and y[1..n], find their longest common subsequence

Example:

x = A,C,G,T,A,G
y = G,T, C,C,A,C

LCS(x, y) = G,T,A

STEP 1: Define subtasks

S(i,j) = Length of LCS of x[1..i] and y[1..j]

Output of algorithm = S(m,n)

STEP 2: Express recursively

S(i,j) = S(i-1,j-1) + 1, if x[i] = y[j]
= max(S(i-1,j), S(i,j-1)), ow

STEP 3: Order of subtasks

Row by row, left to right

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1	0	0	0	1	1	1	1	1	1
T	2	0	0	1	1	1	1	2	2	2
G	3	0	0	1	2	2	2	2	2	3
A	4	1	1	1	2	2	2	2	3	3
C	5									
A	6									
G	7									
T	8									
T	9									

Base Case: Row 0, Column 0

Longest Common Subsequence (LCS)

Problem: Given two sequences x[1..m] and y[1..n], find their longest common subsequence

STEP 1: Define subtasks

S(i,j) = Length of LCS of x[1..i] and y[1..j]

Output of algorithm = S(m,n)

STEP 2: Express recursively

$$S(i,j) = \begin{cases} S(i-1,j-1) + 1, & \text{if } x[i] = y[j] \\ \max(S(i-1,j), S(i,j-1)), & \text{ow} \end{cases}$$

STEP 3: Order of subtasks

Row by row, left to right

Algorithm:

```
for i = 0 to n: S[i,0] = 0
for j = 0 to m: S[0,j] = 0
for i = 1 to n:
  for j = 1 to m:
    if x[i] = y[j]:
      S[i,j] =
        S[i-1,j-1] + 1
    else:
      S[i,j] = max{
        S[i-1,j], S[i,j-1]}
return S[n,m]
```

Longest Common Subsequence (LCS)

Problem: Given two sequences x[1..m] and y[1..n], find their longest common subsequence

STEP 1: Define subtasks

S(i,j) = Length of LCS of x[1..i] and y[1..j]

Output of algorithm = S(m,n)

STEP 2: Express recursively

$$S(i,j) = S(i-1,j-1) + 1, \text{ if } x[i] = y[j]$$
$$= \max(S(i-1,j), S(i,j-1)), \text{ ow}$$

STEP 3: Order of subtasks

Row by row, left to right

Running Time: O(mn)

Algorithm:

```
for i = 0 to n: S[i,0] = 0
for j = 0 to m: S[0,j] = 0
for i = 1 to n:
  for j = 1 to m:
    if x[i] = y[j]:
      S[i,j] =
        S[i-1,j-1] + 1
    else:
      S[i,j] = max{
        S[i-1,j], S[i,j-1]}
return S[n,m]
```

Longest Common Subsequence (LCS)

Problem: Given two sequences x[1..m] and y[1..n], find their longest common subsequence

STEP 1: Define subtasks

S(i,j) = Length of LCS of x[1..i] and y[1..j]

Output of algorithm = S(m,n)

STEP 2: Express recursively

S(i,j) = S(i-1,j-1) + 1, if x[i] = y[j]
= max(S(i-1,j), S(i,j-1)), ow

STEP 3: Order of subtasks

Row by row, left to right

Running Time: O(mn)

How to reconstruct the actual subsequence?

Algorithm:

```
for i = 0 to n: S[i,0] = 0
for j = 0 to m: S[0,j] = 0
for i = 1 to n:
  for j = 1 to m:
    if x[i] = y[j]:
      S[i,j] =
        S[i-1,j-1] + 1
    else:
      S[i,j] = max{
        S[i-1,j], S[i,j-1]}
return S[n,m]
```

Longest Common Subsequence (LCS)

Problem: Given two sequences x[1..m] and y[1..n], find their longest common subsequence

STEP 1: Define subtasks

S(i,j) = Length of LCS of x[1..i] and y[1..j]

STEP 2: Express recursively

S(i,j) = S(i-1,j-1) + 1, if x[i] = y[j]
= max(S(i-1,j), S(i,j-1)), ow

To reconstruct LCS:

Define L(i, j):

L(i, j) = (i - 1, j - 1), if x[i] = y[j]
= (i - 1, j), ow if S(i-1,j) > S(i, j-1)
= (i, j - 1), ow

S

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1	0	0	0	1	1	1	1	1	1
T	2	0	0	1	1	1	1	2	2	2
G	3	0	0	1	2	2	2	2	2	3
A	4	1	1	1	2	2	2	2	3	3

L

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1	←								
T	2									
G	3									
A	4									

Reconstruct LCS by following the L(i,j) pointers, starting with L(m,n)

Recall: Row 0 and column 0: Base Case

Longest Common Subsequence (LCS)

Problem: Given two sequences x[1..m] and y[1..n], find their longest common subsequence

STEP 1: Define subtasks

S(i,j) = Length of LCS of x[1..i] and y[1..j]

STEP 2: Express recursively

S(i,j) = S(i-1,j-1) + 1, if x[i] = y[j]
= max(S(i-1,j), S(i,j-1)), ow

To reconstruct LCS:

Define L(i, j):

L(i, j) = (i - 1, j - 1), if x[i] = y[j]
= (i - 1, j), ow if S(i-1,j) > S(i, j-1)
= (i, j - 1), ow

S

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1	0	0	0	1	1	1	1	1	1
T	2	0	0	1	1	1	1	2	2	2
G	3	0	0	1	2	2	2	2	2	3
A	4	1	1	1	2	2	2	2	3	3

L

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1	←	←	←						
T	2									
G	3									
A	4									

Reconstruct LCS by following the L(i,j) pointers, starting with L(m,n)

Recall: Row 0 and column 0: Base Case

Longest Common Subsequence (LCS)

Problem: Given two sequences x[1..m] and y[1..n], find their longest common subsequence

STEP 1: Define subtasks

S(i,j) = Length of LCS of x[1..i] and y[1..j]

STEP 2: Express recursively

S(i,j) = S(i-1,j-1) + 1, if x[i] = y[j]
= max(S(i-1,j), S(i,j-1)), ow

To reconstruct LCS:

Define L(i, j):

L(i, j) = (i - 1, j - 1), if x[i] = y[j]
= (i - 1, j), ow if S(i-1,j) > S(i, j-1)
= (i, j - 1), ow

S

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1	0	0	0	1	1	1	1	1	1
T	2	0	0	1	1	1	1	2	2	2
G	3	0	0	1	2	2	2	2	2	3
A	4	1	1	1	2	2	2	2	3	3

L

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1	←	←	←	↖					
T	2									
G	3									
A	4									

Reconstruct LCS by following the L(i,j) pointers, starting with L(m,n)

Recall: Row 0 and column 0: Base Case

Longest Common Subsequence (LCS)

Problem: Given two sequences x[1..m] and y[1..n], find their longest common subsequence

STEP 1: Define subtasks

S(i,j) = Length of LCS of x[1..i] and y[1..j]

STEP 2: Express recursively

$$S(i,j) = S(i-1,j-1) + 1, \text{ if } x[i] = y[j]$$
$$= \max(S(i-1,j), S(i,j-1)), \text{ ow}$$

To reconstruct LCS:

Define L(i, j):

$$L(i, j) = (i - 1, j - 1), \text{ if } x[i] = y[j]$$
$$= (i - 1, j), \quad \text{ow if } S(i-1,j) > S(i, j-1)$$
$$= (i, j - 1), \quad \text{ow}$$

S

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1	0	0	0	1	1	1	1	1	1
T	2	0	0	1	1	1	1	2	2	2
G	3	0	0	1	2	2	2	2	2	3
A	4	1	1	1	2	2	2	2	3	3

L

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1	←	←	←	↖	↖				
T	2									
G	3									
A	4									

Reconstruct LCS by following the L(i,j) pointers, starting with L(m,n)

Recall: Row 0 and column 0: Base Case

Longest Common Subsequence (LCS)

Problem: Given two sequences x[1..m] and y[1..n], find their longest common subsequence

STEP 1: Define subtasks

S(i,j) = Length of LCS of x[1..i] and y[1..j]

STEP 2: Express recursively

S(i,j) = S(i-1,j-1) + 1, if x[i] = y[j]
= max(S(i-1,j), S(i,j-1)), ow

To reconstruct LCS:

Define L(i, j):

L(i, j) = (i - 1, j - 1), if x[i] = y[j]
= (i - 1, j), ow if S(i-1,j) > S(i, j-1)
= (i, j - 1), ow

S

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1	0	0	0	1	1	1	1	1	1
T	2	0	0	1	1	1	1	2	2	2
G	3	0	0	1	2	2	2	2	2	3
A	4	1	1	1	2	2	2	2	3	3

L

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1	←	←	←	↖	↖	←			
T	2									
G	3									
A	4									

Reconstruct LCS by following the L(i,j) pointers, starting with L(m,n)

Recall: Row 0 and column 0: Base Case

Longest Common Subsequence (LCS)

Problem: Given two sequences x[1..m] and y[1..n], find their longest common subsequence

STEP 1: Define subtasks

S(i,j) = Length of LCS of x[1..i] and y[1..j]

STEP 2: Express recursively

S(i,j) = S(i-1,j-1) + 1, if x[i] = y[j]
= max(S(i-1,j), S(i,j-1)), ow

To reconstruct LCS:

Define L(i, j):

L(i, j) = (i - 1, j - 1), if x[i] = y[j]
= (i - 1, j), ow if S(i-1,j) > S(i, j-1)
= (i, j - 1), ow

S

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1	0	0	0	1	1	1	1	1	1
T	2	0	0	1	1	1	1	2	2	2
G	3	0	0	1	2	2	2	2	2	3
A	4	1	1	1	2	2	2	2	3	3

L

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1	←	←	←	↖	↖	←	←	←	↖
T	2									
G	3									
A	4									

Reconstruct LCS by following the L(i,j) pointers, starting with L(m,n)

Recall: Row 0 and column 0: Base Case

Longest Common Subsequence (LCS)

Problem: Given two sequences x[1..m] and y[1..n], find their longest common subsequence

STEP 1: Define subtasks

S(i,j) = Length of LCS of x[1..i] and y[1..j]

STEP 2: Express recursively

$$S(i,j) = S(i-1,j-1) + 1, \text{ if } x[i] = y[j]$$
$$= \max(S(i-1,j), S(i,j-1)), \text{ ow}$$

To reconstruct LCS:

Define L(i, j):

$$L(i, j) = (i - 1, j - 1), \text{ if } x[i] = y[j]$$
$$= (i - 1, j), \quad \text{ow if } S(i-1,j) > S(i, j-1)$$
$$= (i, j - 1), \quad \text{ow}$$

S

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1	0	0	0	1	1	1	1	1	1
T	2	0	0	1	1	1	1	2	2	2
G	3	0	0	1	2	2	2	2	2	3
A	4	1	1	1	2	2	2	2	3	3

L

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1	←	←	←	↖	↖	←	←	←	↖
T	2	←	←	↖	←	←	←	↖	←	←
G	3	←	←	↑	↖	↖	←	←	←	↖
A	4	↖	←	←	←	←	←	←	↖	←

Reconstruct LCS by following the L(i,j) pointers, starting with L(m,n)

Recall: Row 0 and column 0: Base Case

Longest Common Subsequence (LCS)

Problem: Given two sequences x[1..m] and y[1..n], find their longest common subsequence

STEP 1: Define subtasks

S(i,j) = Length of LCS of x[1..i] and y[1..j]

STEP 2: Express recursively

$S(i,j) = S(i-1,j-1) + 1$, if $x[i] = y[j]$
$= \max(S(i-1,j), S(i,j-1))$, ow

To reconstruct LCS:

Define L(i, j):

$L(i, j) = (i - 1, j - 1)$, if $x[i] = y[j]$
$= (i - 1, j)$, ow if $S(i-1,j) > S(i, j-1)$
$= (i, j - 1)$, ow

S

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1	0	0	0	1	1	1	1	1	1
T	2	0	0	1	1	1	1	2	2	2
G	3	0	0	1	2	2	2	2	2	3
A	4	1	1	1	2	2	2	2	3	3

L

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1	←	←	←	↖	↖	←	←	←	↖
T	2	←	←	↖	←	←	←	↖	←	←
G	3	←	←	↑	↖	↖	←	←	←	↖
A	4	↖	←	←	←	←	←	←	↖	←

Reconstruct LCS by following the L(i,j) pointers, starting with L(m,n)

Recall: Row 0 and column 0: Base Case

Longest Common Subsequence (LCS)

Problem: Given two sequences x[1..m] and y[1..n], find their longest common subsequence

STEP 1: Define subtasks

S(i,j) = Length of LCS of x[1..i] and y[1..j]

STEP 2: Express recursively

$$S(i,j) = S(i-1,j-1) + 1, \text{ if } x[i] = y[j]$$
$$= \max(S(i-1,j), S(i,j-1)), \text{ ow}$$

To reconstruct LCS:

Define L(i, j):

$$L(i, j) = (i - 1, j - 1), \text{ if } x[i] = y[j]$$
$$= (i - 1, j), \quad \text{ow if } S(i-1,j) > S(i, j-1)$$
$$= (i, j - 1), \quad \text{ow}$$

S

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1	0	0	0	1	1	1	1	1	1
T	2	0	0	1	1	1	1	2	2	2
G	3	0	0	1	2	2	2	2	2	3
A	4	1	1	1	2	2	2	2	3	3

L

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1	←	←	←	↖	↖	←	←	←	↖
T	2	←	←	↖	←	←	←	↖	←	←
G	3	←	←	↑	↖	↖	←	←	←	↖
A	4	↖	←	←	←	←	←	←	↖	←

Reconstruct LCS by following the L(i,j) pointers, starting with L(m,n)

Recall: Row 0 and column 0: Base Case

Longest Common Subsequence (LCS)

Problem: Given two sequences x[1..m] and y[1..n], find their longest common subsequence

STEP 1: Define subtasks

S(i,j) = Length of LCS of x[1..i] and y[1..j]

STEP 2: Express recursively

S(i,j) = S(i-1,j-1) + 1, if x[i] = y[j]
= max(S(i-1,j), S(i,j-1)), ow

To reconstruct LCS:

Define L(i, j):

L(i, j) = (i - 1, j - 1), if x[i] = y[j]
= (i - 1, j), ow if S(i-1,j) > S(i, j-1)
= (i, j - 1), ow

S

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1	0	0	0	1	1	1	1	1	1
T	2	0	0	1	1	1	1	2	2	2
G	3	0	0	1	2	2	2	2	2	3
A	4	1	1	1	2	2	2	2	3	3

L

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1	←	←	←	↖	↖	←	←	←	↖
T	2	←	←	↖	←	←	←	↖	←	←
G	3	←	←	↑	↖	↖	←	←	←	↖
A	4	↖	←	←	←	←	←	←	↖	←

Reconstruct LCS by following the L(i,j) pointers, starting with L(m,n)

Recall: Row 0 and column 0: Base Case

Longest Common Subsequence (LCS)

Problem: Given two sequences x[1..m] and y[1..n], find their longest common subsequence

STEP 1: Define subtasks

S(i,j) = Length of LCS of x[1..i] and y[1..j]

STEP 2: Express recursively

S(i,j) = S(i-1,j-1) + 1, if x[i] = y[j]
= max(S(i-1,j), S(i,j-1)), ow

To reconstruct LCS:

Define L(i, j):

L(i, j) = (i - 1, j - 1), if x[i] = y[j]
= (i - 1, j), ow if S(i-1,j) > S(i, j-1)
= (i, j - 1), ow

S

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1	0	0	0	1	1	1	1	1	1
T	2	0	0	1	1	1	1	2	2	2
G	3	0	0	1	2	2	2	2	2	3
A	4	1	1	1	2	2	2	2	3	3

L

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1	←	←	←	↖	↖	←	←	←	↖
T	2	←	←	↖	←	←	←	↖	←	←
G	3	←	←	↑	↖	↖	←	←	←	↖
A	4	↖	←	←	←	←	←	←	↖	←

Reconstruct LCS by following the L(i,j) pointers, starting with L(m,n)

Recall: Row 0 and column 0: Base Case

Longest Common Subsequence (LCS)

Problem: Given two sequences x[1..m] and y[1..n], find their longest common subsequence

STEP 1: Define subtasks

S(i,j) = Length of LCS of x[1..i] and y[1..j]

STEP 2: Express recursively

S(i,j) = S(i-1,j-1) + 1, if x[i] = y[j]
= max(S(i-1,j), S(i,j-1)), ow

To reconstruct LCS:

Define L(i, j):

L(i, j) = (i - 1, j - 1), if x[i] = y[j]
= (i - 1, j), ow if S(i-1,j) > S(i, j-1)
= (i, j - 1), ow

S

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1	0	0	0	1	1	1	1	1	1
T	2	0	0	1	1	1	1	2	2	2
G	3	0	0	1	2	2	2	2	2	3
A	4	1	1	1	2	2	2	2	3	3

L

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1	←	←	←	↖	↖	←	←	←	↖
T	2	←	←	↖	←	←	←	↖	←	←
G	3	←	←	↑	↖	↖	←	←	←	↖
A	4	↖	←	←	←	←	←	←	↖	←

Reconstruct LCS by following the L(i,j) pointers, starting with L(m,n)

Recall: Row 0 and column 0: Base Case

Longest Common Subsequence (LCS)

Problem: Given two sequences x[1..m] and y[1..n], find their longest common subsequence

STEP 1: Define subtasks

S(i,j) = Length of LCS of x[1..i] and y[1..j]

STEP 2: Express recursively

$$S(i,j) = S(i-1,j-1) + 1, \text{ if } x[i] = y[j]$$
$$= \max(S(i-1,j), S(i,j-1)), \text{ ow}$$

To reconstruct LCS:

Define L(i, j):

$$L(i, j) = (i - 1, j - 1), \text{ if } x[i] = y[j]$$
$$= (i - 1, j), \quad \text{ow if } S(i-1,j) > S(i, j-1)$$
$$= (i, j - 1), \quad \text{ow}$$

S

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1	0	0	0	1	1	1	1	1	1
T	2	0	0	1	1	1	1	2	2	2
G	3	0	0	1	2	2	2	2	2	3
A	4	1	1	1	2	2	2	2	3	3

L

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1	←	←	←	↖	↖	←	←	←	↖
T	2	←	←	↖	←	←	←	↖	←	←
G	3	←	←	↑	↖	↖	←	←	←	↖
A	4	↖	←	←	←	←	←	←	↖	←

Reconstruct LCS by following the L(i,j) pointers, starting with L(m,n)

Recall: Row 0 and column 0: Base Case

Longest Common Subsequence (LCS)

Problem: Given two sequences x[1..m] and y[1..n], find their longest common subsequence

STEP 1: Define subtasks

S(i,j) = Length of LCS of x[1..i] and y[1..j]

STEP 2: Express recursively

S(i,j) = S(i-1,j-1) + 1, if x[i] = y[j]
= max(S(i-1,j), S(i,j-1)), ow

To reconstruct LCS:

Define L(i, j):

L(i, j) = (i - 1, j - 1), if x[i] = y[j]
= (i - 1, j), ow if S(i-1,j) > S(i, j-1)
= (i, j - 1), ow

S

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1	0	0	0	1	1	1	1	1	1
T	2	0	0	1	1	1	1	2	2	2
G	3	0	0	1	2	2	2	2	2	3
A	4	1	1	1	2	2	2	2	3	3

L

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1	←	←	←	↖	↖	←	←	←	↖
T	2	←	←	↖	←	←	←	↖	←	←
G	3	←	←	↑	↖	↖	←	←	←	↖
A	4	↖	←	←	←	←	←	←	↖	←

Reconstruct LCS by following the L(i,j) pointers, starting with L(m,n)

Recall: Row 0 and column 0: Base Case

Longest Common Subsequence (LCS)

Problem: Given two sequences x[1..m] and y[1..n], find their longest common subsequence

STEP 1: Define subtasks

S(i,j) = Length of LCS of x[1..i] and y[1..j]

STEP 2: Express recursively

$$S(i,j) = S(i-1,j-1) + 1, \text{ if } x[i] = y[j]$$
$$= \max(S(i-1,j), S(i,j-1)), \text{ ow}$$

To reconstruct LCS:

Define L(i, j):

$$L(i, j) = (i - 1, j - 1), \text{ if } x[i] = y[j]$$
$$= (i - 1, j), \quad \text{ow if } S(i-1,j) > S(i, j-1)$$
$$= (i, j - 1), \quad \text{ow}$$

S

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1	0	0	0	1	1	1	1	1	1
T	2	0	0	1	1	1	1	2	2	2
G	3	0	0	1	2	2	2	2	2	3
A	4	1	1	1	2	2	2	2	3	3

L

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1	←	←	←	↖	↖	←	←	←	↖
T	2	←	←	↖	←	←	←	↖	←	←
G	3	←	←	↑	↖	↖	←	←	←	↖
A	4	↖	←	←	←	←	←	←	↖	←

Reconstruct LCS by following the L(i,j) pointers, starting with L(m,n)

Recall: Row 0 and column 0: Base Case

Longest Common Subsequence (LCS)

Problem: Given two sequences x[1..m] and y[1..n], find their longest common subsequence

STEP 1: Define subtasks

S(i,j) = Length of LCS of x[1..i] and y[1..j]

STEP 2: Express recursively

S(i,j) = S(i-1,j-1) + 1, if x[i] = y[j]
= max(S(i-1,j), S(i,j-1)), ow

To reconstruct LCS:

Define L(i, j):

L(i, j) = (i - 1, j - 1), if x[i] = y[j]
= (i - 1, j), ow if S(i-1,j) > S(i, j-1)
= (i, j - 1), ow

S

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1	0	0	0	1	1	1	1	1	1
T	2	0	0	1	1	1	1	2	2	2
G	3	0	0	1	2	2	2	2	2	3
A	4	1	1	1	2	2	2	2	3	3

L

		A	C	T	G	G	C	T	A	G
	0	1	2	3	4	5	6	7	8	9
G	1	←	←	←	↖	↖	←	←	←	↖
T	2	←	←	↖	←	←	←	↖	←	←
G	3	←	←	↑	↖	↖	←	←	←	↖
A	4	↖	←	←	←	←	←	←	↖	←

Reconstruct LCS by following the L(i,j) pointers, starting with L(m,n)

Recall: Row 0 and column 0: Base Case

LCS = T, G, A

Longest Common Subsequence (LCS)

Problem: Given two sequences x[1..m] and y[1..n], find their longest common subsequence

STEP 1: Define subtasks

S(i,j) = Length of LCS of x[1..i] and y[1..j]

Output of algorithm = S(m,n)

STEP 2: Express recursively

$$S(i,j) = \begin{cases} S(i-1,j-1) + 1, & \text{if } x[i] = y[j] \\ \max(S(i-1,j), S(i,j-1)), & \text{ow} \end{cases}$$

STEP 3: Order of subtasks

Row by row, left to right

To reconstruct LCS:

Define L(i, j):

$$L(i, j) = \begin{cases} (i - 1, j - 1), & \text{if } x[i] = y[j] \\ (i - 1, j), & \text{ow if } S(i-1,j) > S(i, j-1) \\ (i, j - 1), & \text{ow} \end{cases}$$

Reconstruct LCS by following the L(i,j) pointers, starting with L(m,n)

Running Time: O(mn)

Dynamic Programming

- String Reconstruction
- Longest Common Subsequence
- ...

Dynamic Programming vs Divide and Conquer

Divide-and-conquer

A problem of size n is decomposed into a few subproblems which are significantly smaller (e.g. n/2, 3n/4,...)

Therefore, size of subproblems decreases geometrically.

eg. n, n/2, n/4, n/8, etc

Use a recursive algorithm.

Dynamic programming

A problem of size n is expressed in terms of subproblems that are not much smaller (e.g. n-1, n-2,...)

A recursive algorithm would take exp. time.

Saving grace: in total, there are only polynomially many subproblems.

Avoid recursion and instead solve the subproblems one-by-one, saving the answers in a table, in a clever explicit order.

DP: Common Subtasks

Case 1: Input: $x_1, x_2,..,x_n$ Subproblem: $x_1, .., x_i$.

$\boxed{x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6} \quad x_7 \quad x_8 \quad x_9 \quad x_{10}$

DP: Common Subtasks

Case 1: Input: $x_1, x_2, .., x_n$ Subproblem: $x_1, .., x_i$.

$\boxed{x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6} \; x_7 \quad x_8 \quad x_9 \quad x_{10}$

Case 2: Input: $x_1, x_2, .., x_n$ and $y_1, y_2, .., y_m$ Subproblem: $x_1, .., x_i$ and $y_1, y_2, .., y_j$

$\boxed{x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6} \; x_7 \quad x_8 \quad x_9 \quad x_{10}$

$\boxed{y_1 \quad y_2 \quad y_3 \quad y_4 \quad y_5} \; y_6 \quad y_7 \quad y_8$

DP: Common Subtasks

Case 1: Input: $x_1, x_2,..,x_n$ Subproblem: $x_1, .., x_i$.

$\boxed{x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6}\ x_7 \quad x_8 \quad x_9 \quad x_{10}$

Case 2: Input: $x_1, x_2,..,x_n$ and $y_1, y_2,..,y_m$ Subproblem: $x_1, .., x_i$ and $y_1, y_2,..,y_j$

$\boxed{x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6}\ x_7 \quad x_8 \quad x_9 \quad x_{10}$

$\boxed{y_1 \quad y_2 \quad y_3 \quad y_4 \quad y_5}\ y_6 \quad y_7 \quad y_8$

Case 3: Input: $x_1, x_2,..,x_n$. Subproblem: $x_i, .., x_j$

$x_1 \quad x_2\ \boxed{x_3 \quad x_4 \quad x_5 \quad x_6}\ x_7 \quad x_8 \quad x_9 \quad x_{10}$

DP: Common Subtasks

Case 1: Input: $x_1, x_2, .., x_n$ Subproblem: $x_1, .., x_i$.

$$\boxed{x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6} \; x_7 \quad x_8 \quad x_9 \quad x_{10}$$

Case 2: Input: $x_1, x_2, .., x_n$ and $y_1, y_2, .., y_m$ Subproblem: $x_1, .., x_i$ and $y_1, y_2, .., y_j$

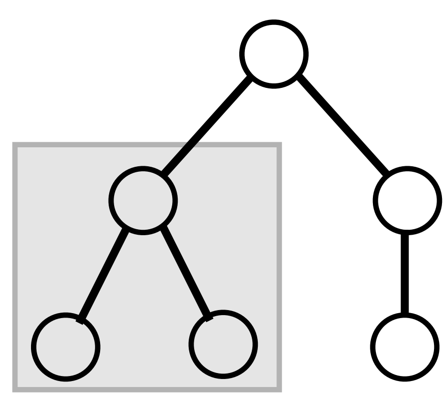
$$\boxed{x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6} \; x_7 \quad x_8 \quad x_9 \quad x_{10}$$

$$\boxed{y_1 \quad y_2 \quad y_3 \quad y_4 \quad y_5} \; y_6 \quad y_7 \quad y_8$$

Case 3: Input: $x_1, x_2, .., x_n$. Subproblem: $x_i, .., x_j$

$$x_1 \quad x_2 \; \boxed{x_3 \quad x_4 \quad x_5 \quad x_6} \; x_7 \quad x_8 \quad x_9 \quad x_{10}$$

Case 4: Input: a rooted tree. Subproblem: a subtree



Dynamic Programming

- String Reconstruction
- Longest Common Subsequence
- Edit Distance

Edit Distance: String Alignment

Alignment: Convert one string to another using insertions, deletions and substitutions.

S	U	N	N	-	Y
S	-	N	O	W	Y

Alignment 1
Cost = 3

Edit Distance: String Alignment

Alignment: Convert one string to another using insertions, deletions and substitutions.

S	U	N	N	-	Y
S	-	N	O	W	Y

Alignment 1
Cost = 3

S	U	N	-	-	N	Y
-	S	N	O	W	-	Y

Alignment 2
Cost = 5

Edit Distance: String Alignment

Alignment: Convert one string to another using insertions, deletions and substitutions.

S	U	N	N	-	Y
S	-	N	O	W	Y

Alignment 1
Cost = 3

S	U	N	-	-	N	Y
-	S	N	O	W	-	Y

Alignment 2
Cost = 5

Edit Distance(x, y): minimum # of insertions, deletions and substitutions required to convert x to y

Edit Distance: String Alignment

Alignment: Convert one string to another using insertions, deletions and substitutions.

S	U	N	N	-	Y
S	-	N	O	W	Y

Alignment 1
Cost = 3

S	U	N	-	-	N	Y
-	S	N	O	W	-	Y

Alignment 2
Cost = 5

Edit Distance(x, y): minimum # of insertions, deletions and substitutions required to convert x to y

Edit Distance(SUNNY, SNOWY) = 3

Edit Distance: String Alignment

Alignment: Convert one string to another using insertions, deletions and substitutions.

S	U	N	N	-	Y
S	-	N	O	W	Y

Alignment 1
Cost = 3

S	U	N	-	-	N	Y
-	S	N	O	W	-	Y

Alignment 2
Cost = 5

Edit Distance(x, y): minimum # of insertions, deletions and substitutions required to convert x to y

Edit Distance(SUNNY, SNOWY) = 3

Is Edit Distance(x, y) = Edit Distance(y, x)?

Edit Distance

Problem: Given two strings x[1..n] and y[1..m], compute edit-distance(x, y)

Example:

S	U	N	N	-	Y
S	-	N	O	W	Y

Cost = 3

Structure:
Three cases in the last column of alignment between x[1..i] and y[1..j]:

x[i]	-	x[i]
-	y[j]	y[j]

Case 1. Align x[1..i-1] and y[1..j], delete x[i]
Case 2. Align x[1..i] and y[1..j-1], insert y[j]
Case 3. Align x[1..i-1] and y[1..j-1]. Substitute x[i], y[j] if different.

Edit Distance

Problem: Given two strings x[1..n] and y[1..m], compute edit-distance(x, y)

Example:

x = SUNNY
y = SNOWY

Edit-distance(x, y) = 3

			S	U	N	N	Y
	-	0	1	2	3	4	5
	0						
S	1						
N	2						
O	3						
W	4						
Y	5						

STEP 1: Define subtasks

E(i,j) = Edit-distance(x[1..i], y[1..j])

Output of algorithm = E(n,m)

STEP 2: Express recursively

E(i,j) = min(E(i-1,j) + 1, E(i, j-1) + 1,
E(i-1,j-1) + diff(x[i], y[j]))

diff(a, b) = 0, if a=b
= 1, o.w.

STEP 3: Order of subtasks

Row by row, left to right

Edit Distance

Problem: Given two strings x[1..n] and y[1..m], compute edit-distance(x, y)

Example:

x = SUNNY
y = SNOWY

Edit-distance(x, y) = 3

			S	U	N	N	Y
	-	0	1	2	3	4	5
	0	0	1	2	3	4	5
S	1	1					
N	2	2					
O	3	3					
W	4	4					
Y	5	5					

STEP 1: Define subtasks

E(i,j) = Edit-distance(x[1..i], y[1..j])

Output of algorithm = E(n,m)

STEP 2: Express recursively

E(i,j) = min(E(i-1,j) + 1, E(i, j-1) + 1,
E(i-1,j-1) + diff(x[i], y[j]))

diff(a, b) = 0, if a=b
= 1, o.w.

STEP 3: Order of subtasks

Row by row, left to right

Edit Distance

Problem: Given two strings x[1..n] and y[1..m], compute edit-distance(x, y)

Example:

x = SUNNY
y = SNOWY

Edit-distance(x, y) = 3

			S	**U**	**N**	**N**	**Y**
	-	0	1	2	3	4	5
	0	0	1	2	3	4	5
S	1	1	0				
N	2	2					
O	3	3					
W	4	4					
Y	5	5					

STEP 1: Define subtasks

E(i,j) = Edit-distance(x[1..i], y[1..j])

Output of algorithm = E(n,m)

STEP 2: Express recursively

$$E(i,j) = \min(E(i-1,j) + 1, E(i, j-1) + 1, E(i-1,j-1) + \mathrm{diff}(x[i], y[j]))$$

$$\mathrm{diff}(a, b) = 0, \text{ if } a=b; \quad = 1, \text{ o.w.}$$

STEP 3: Order of subtasks

Row by row, left to right

Edit Distance

Problem: Given two strings x[1..n] and y[1..m], compute edit-distance(x, y)

Example:

x = SUNNY
y = SNOWY

Edit-distance(x, y) = 3

			S	**U**	**N**	**N**	**Y**
	-	0	1	2	3	4	5
	0	0	1	2	3	4	5
S	1	1	0	1			
N	2	2					
O	3	3					
W	4	4					
Y	5	5					

STEP 1: Define subtasks

E(i,j) = Edit-distance(x[1..i], y[1..j])

Output of algorithm = E(n,m)

STEP 2: Express recursively

$$E(i,j) = \min(E(i-1,j) + 1, E(i, j-1) + 1, E(i-1,j-1) + \mathrm{diff}(x[i], y[j]))$$

$$\mathrm{diff}(a, b) = 0, \text{ if } a=b$$
$$= 1, \text{ o.w.}$$

STEP 3: Order of subtasks

Row by row, left to right

Edit Distance

Problem: Given two strings x[1..n] and y[1..m], compute edit-distance(x, y)

Example:

x = SUNNY
y = SNOWY

Edit-distance(x, y) = 3

			S	U	N	N	Y
	-	0	1	2	3	4	5
	0	0	1	2	3	4	5
S	1	1	0	1	2	3	4
N	2	2					
O	3	3					
W	4	4					
Y	5	5					

STEP 1: Define subtasks

E(i,j) = Edit-distance(x[1..i], y[1..j])

Output of algorithm = E(n,m)

STEP 2: Express recursively

$$E(i,j) = \min(E(i-1,j) + 1, E(i, j-1) + 1, E(i-1,j-1) + \text{diff}(x[i], y[j]))$$

$$\text{diff}(a, b) = 0, \text{ if } a=b$$
$$= 1, \text{ o.w.}$$

STEP 3: Order of subtasks

Row by row, left to right

Edit Distance

Problem: Given two strings x[1..n] and y[1..m], compute edit-distance(x, y)

Example:

x = SUNNY
y = SNOWY

Edit-distance(x, y) = 3

			S	**U**	**N**	**N**	**Y**
	-	0	1	2	3	4	5
	0	0	1	2	3	4	5
S	1	1	0	1	2	3	4
N	2	2	1				
O	3	3					
W	4	4					
Y	5	5					

STEP 1: Define subtasks

E(i,j) = Edit-distance(x[1..i], y[1..j])

Output of algorithm = E(n,m)

STEP 2: Express recursively

$$E(i,j) = \min(E(i-1,j) + 1, E(i, j-1) + 1, E(i-1,j-1) + \mathrm{diff}(x[i], y[j]))$$

$$\mathrm{diff}(a, b) = 0, \text{ if } a=b$$
$$= 1, \text{ o.w.}$$

STEP 3: Order of subtasks

Row by row, left to right

Edit Distance

Problem: Given two strings x[1..n] and y[1..m], compute edit-distance(x, y)

Example:

x = SUNNY
y = SNOWY

Edit-distance(x, y) = 3

			S	U	N	N	Y
	-	0	1	2	3	4	5
	0	0	1	2	3	4	5
S	1	1	0	1	2	3	4
N	2	2	1	1			
O	3	3					
W	4	4					
Y	5	5					

STEP 1: Define subtasks

E(i,j) = Edit-distance(x[1..i], y[1..j])

Output of algorithm = E(n,m)

STEP 2: Express recursively

$$E(i,j) = \min(E(i-1,j) + 1, E(i, j-1) + 1, E(i-1,j-1) + \mathrm{diff}(x[i], y[j]))$$

$$\mathrm{diff}(a, b) = 0, \text{ if } a=b$$
$$= 1, \text{ o.w.}$$

STEP 3: Order of subtasks

Row by row, left to right

Edit Distance

Problem: Given two strings x[1..n] and y[1..m], compute edit-distance(x, y)

Example:

x = SUNNY
y = SNOWY

Edit-distance(x, y) = 3

			S	**U**	**N**	**N**	**Y**
	-	0	1	2	3	4	5
	0	0	1	2	3	4	5
S	1	1	0	1	2	3	4
N	2	2	1	1	1		
O	3	3					
W	4	4					
Y	5	5					

STEP 1: Define subtasks

E(i,j) = Edit-distance(x[1..i], y[1..j])

Output of algorithm = E(n,m)

STEP 2: Express recursively

E(i,j) = min(E(i-1,j) + 1, E(i, j-1) + 1,
E(i-1,j-1) + diff(x[i], y[j]))

diff(a, b) = 0, if a=b
= 1, o.w.

STEP 3: Order of subtasks

Row by row, left to right

Edit Distance

Problem: Given two strings x[1..n] and y[1..m], compute edit-distance(x, y)

Example:

x = SUNNY
y = SNOWY

Edit-distance(x, y) = 3

			S	**U**	**N**	**N**	**Y**
	-	0	1	2	3	4	5
	0	0	1	2	3	4	5
S	1	1	0	1	2	3	4
N	2	2	1	1	1	2	
O	3	3					
W	4	4					
Y	5	5					

STEP 1: Define subtasks

E(i,j) = Edit-distance(x[1..i], y[1..j])

Output of algorithm = E(n,m)

STEP 2: Express recursively

$$E(i,j) = \min(E(i-1,j) + 1, E(i, j-1) + 1, E(i-1,j-1) + \text{diff}(x[i], y[j]))$$

$$\text{diff}(a, b) = 0, \text{ if } a=b; \quad = 1, \text{ o.w.}$$

STEP 3: Order of subtasks

Row by row, left to right

Edit Distance

Problem: Given two strings x[1..n] and y[1..m], compute edit-distance(x, y)

Example:

x = SUNNY
y = SNOWY

Edit-distance(x, y) = 3

			S	**U**	**N**	**N**	**Y**
	-	0	1	2	3	4	5
	0	0	1	2	3	4	5
S	1	1	0	1	2	3	4
N	2	2	1	1	1	2	3
O	3	3					
W	4	4					
Y	5	5					

STEP 1: Define subtasks

E(i,j) = Edit-distance(x[1..i], y[1..j])

Output of algorithm = E(n,m)

STEP 2: Express recursively

$$E(i,j) = \min(E(i-1,j) + 1, E(i, j-1) + 1, E(i-1,j-1) + \text{diff}(x[i], y[j]))$$

$$\text{diff}(a, b) = 0, \text{ if } a=b; \quad = 1, \text{ o.w.}$$

STEP 3: Order of subtasks

Row by row, left to right

Edit Distance

Problem: Given two strings x[1..n] and y[1..m], compute edit-distance(x, y)

Example:

x = SUNNY
y = SNOWY

Edit-distance(x, y) = 3

			S	U	N	N	Y
	-	0	1	2	3	4	5
	0	0	1	2	3	4	5
S	1	1	0	1	2	3	4
N	2	2	1	1	1	2	3
O	3	3					
W	4	4					
Y	5	5					

STEP 1: Define subtasks

E(i,j) = Edit-distance(x[1..i], y[1..j])

Output of algorithm = E(n,m)

STEP 2: Express recursively

$$E(i,j) = \min(E(i-1,j) + 1, E(i, j-1) + 1, E(i-1,j-1) + \mathrm{diff}(x[i], y[j]))$$

STEP 3: Order of subtasks

Row by row, left to right

Running Time = O(mn)

How to reconstruct the best alignment?