# CSE 202: Design and Analysis of Algorithms

# Lecture 6

Instructor: Kamalika Chaudhuri

# **Announcements**

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

# Last Class: Three steps of Dynamic Programming

# **Main Steps:**

- I. 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!)

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

# **Example:**

x = anonymousarrayofletters : **True** 

x = anhuymousarrayofhetters : **False** 

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

# **Example:**

x = anonymousarrayofletters : **True**x = anhuymousarrayofhetters : **False** 

## **STEP I: Define subtask**

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

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 I: Define subtask**

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

False otherwise

Output of algorithm = S(n)

X		Α	Ν	0	Ν	Y	M	0	U	S	Α	R	R	Α	Y	0	F	L	Ε	Т	Т	Е	R	S
k	0	_	2	3	4	5	6	7	8	9	10	П	12	13	14	15	16	17	18	19	20	21	22	23
S																								

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 I: Define Subtask**

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

False otherwise

X		Α	Ν	O	Ν	Υ	M	O	U	S	Α	R	R	Α	Υ	0	F	L	Ε	Т	Т	Ε	R	S
k	0	_	2	3	4	5	6	7	8	9	10	П	12	13	14	15	16	17	18	19	20	21	22	23
S																								

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 I: 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		Α	Ν	0	Z	Υ	M	0	U	S	Α	R	R	Α	Υ	0	F	L	Е	Т	Т	Е	R	S
k	0	Ι	2	3	4	5	6	7	8	9	10	П	12	13	14	15	16	17	18	19	20	21	22	23
S																								

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 I: 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**

X		Α	Z	0	Ν	Y	M	O	U	S	Α	R	R	Α	Y	0	F	L	Ε	Т	Т	Ε	R	S
k	0		2	3	4	5	6	7	8	9	10	П	12	13	14	15	16	17	18	19	20	21	22	23
S																								

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 I: 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**

X		Α	Ν	O	Ν	Y	M	0	U	S	Α	R	R	Α	Υ	0	F	L	Ε	Т	Т	Ε	R	S
k	0	_	2	3	4	5	6	7	8	9	10	П	12	13	14	15	16	17	18	19	20	21	22	23
S		T																						

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 I: 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**

X		Α	Z	0	Ν	Y	M	0	U	S	Α	R	R	Α	Υ	0	F	L	Ε	Т	Т	Ε	R	S
k	0	—	2	3	4	5	6	7	8	9	10	Ш	12	13	14	15	16	17	18	19	20	21	22	23
S		Т	Т																					

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 I: 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**

x		Α	Ν	0	Ν	Y	M	0	U	S	Α	R	R	Α	Υ	0	F	L	Е	Т	Т	Ε	R	S
k	0	1	2	3	4	5	6	7	8	9	10	П	12	13	14	15	16	17	18	19	20	21	22	23
S		Т	Т	Т																				

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 I: 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**

X		Α	Z	0	Ν	Y	Μ	0	U	S	Α	R	R	Α	Υ	0	F	L	Ε	Т	Т	Ε	R	S
k	0	—	2	3	4	5	6	7	8	9	10	Ш	12	13	14	15	16	17	18	19	20	21	22	23
S		Т	Т	Т	Т																			

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 I: 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**

X		Α	Ν	O	Ζ	Y	M	0	U	S	Α	R	R	Α	Υ	0	F	L	Ε	Т	Т	Ε	R	S
k	0		2	3	4	5	6	7	8	9	10	П	12	13	14	15	16	17	18	19	20	21	22	23
S		Т	T	Т	Т	F																		

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 I: 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**

X		Α	Ν	O	Ζ	Y	M	O	U	S	Α	R	R	Α	Y	O	F	L	Ε	Т	Т	Ε	R	S
k	0	-	2	3	4	5	6	7	8	9	10	Ш	12	13	14	15	16	17	18	19	20	21	22	23
S		Т	Т	Т	Т	F	F																	

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 I: 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**

X		Α	Ν	0	Ν	Y	M	O	U	S	Α	R	R	Α	Υ	0	F	L	Ε	Т	Т	Ε	R	S
k	0	_	2	3	4	5	6	7	8	9	10	П	12	13	14	15	16	17	18	19	20	21	22	23
S		Т	Т	Т	Т	F	F	F																

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 I: 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**

X		Α	Z	0	Ν	Y	M	O	U	S	Α	R	R	Α	Y	0	F	L	Ε	Т	Т	Ε	R	S
k	0	_	2	3	4	5	6	7	8	9	10	П	12	13	14	15	16	17	18	19	20	21	22	23
S		Т	Т	Т	Т	F	F	F	F															

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 I: 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**

X		Α	Z	0	Ν	Y	М	O	U	S	Α	R	R	Α	Υ	0	F	L	Ε	Т	Т	Ε	R	S
k	0	_	2	3	4	5	6	7	8	9	10	П	12	13	14	15	16	17	18	19	20	21	22	23
S		Т	Т	Т	Т	F	F	F	F	Т														

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 I: 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**

X		Α	Ν	0	Ν	Y	M	O	U	S	Α	R	R	Α	Y	0	F	L	Ε	Т	Т	Ε	R	S
k	0	_	2	3	4	5	6	7	8	9	10	П	12	13	14	15	16	17	18	19	20	21	22	23
S		Т	Т	Т	Т	F	F	F	F	Т	Т													

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 I: 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**

X		Α	Ν	O	Ν	Y	M	O	J	S	Α	R	R	Α	Y	0	F	L	Ε	Т	Т	Ε	R	S
k	0	—	2	3	4	5	6	7	8	9	10	П	12	13	14	15	16	17	18	19	20	21	22	23
S		Т	Т	Т	Т	F	F	F	F	Т	Т	F												

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 I: 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**

X		Α	Z	0	Ν	Y	M	0	U	S	Α	R	R	Α	Y	0	F	L	Е	Т	Т	Ε	R	S
k	0	1	2	3	4	5	6	7	8	9	10	П	12	13	14	15	16	17	18	19	20	21	22	23
S		Т	Т	Т	Т	F	F	F	F	Т	Т	F	F											

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 I: 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**

X		Α	Ν	0	Ν	Y	M	0	J	S	Α	R	R	Α	Y	0	F	L	Е	Т	Т	Ε	R	S
k	0	I	2	3	4	5	6	7	8	9	10	П	12	13	14	15	16	17	18	19	20	21	22	23
S		Т	Т	Т	Т	F	F	F	F	Т	Т	F	F	F										

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 I: 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**

x		Α	Ν	0	Ν	Y	M	0	U	S	Α	R	R	Α	Υ	0	F	L	Е	Т	Т	Ε	R	S
k	0	1	2	3	4	5	6	7	8	9	10	П	12	13	14	15	16	17	18	19	20	21	22	23
S		Т	Т	Т	Τ	F	F	F	F	Т	Т	F	F	F	Т									

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 I: 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**

X		Α	Z	0	Z	Υ	M	0	U	S	Α	R	R	Α	Y	0	F	L	Е	Т	Т	Е	R	S
k	0	I	2	3	4	5	6	7	8	9	10	П	12	13	14	15	16	17	18	19	20	21	22	23
S		Т	Т	Т	Т	F	F	F	F	Т	Т	F	F	F	Т	F	$\dashv$	F	F	Т	F	F	Т	Т

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

## **STEP I: 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(I,..n): If S(k) = true, then D(k) = starting position of the word that ends at x[k]

Reconstruct text by following these pointers.

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 I: 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

# **Reconstructing Document:**

Define array D(I,..n):

If S(k) = True, then D(k) = starting position of the word that ends at <math>x[k]

Reconstruct text by following these pointers.

#### **STEP 3: Order of Subtasks**

X		Α	Z	0	Ν	Υ	M	0	U	S	Α	R	R	Α	Y	0	F	L	Ε	Т	Т	Е	R	S
k	0		2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
S		Τ	Т	Т	Т	F	F	F	F	Т	Т	F	F	F	Т	F	Т	F	F	Т	F	F	Т	Т
D						•						1	1			1		1	1		1	1		

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 I: 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

# **Reconstructing Document:**

Define array D(I,..n):

If S(k) = True, then D(k) = starting position of the word that ends at <math>x[k]

Reconstruct text by following these pointers.

#### **STEP 3: Order of Subtasks**

X		Α	Z	0	Z	Υ	M	0	U	S	Α	R	R	Α	Y	0	F	L	Ε	Т	Т	Ε	R	S
k	0		2	3	4	5	6	7	8	9	10	П	12	13	14	15	16	17	18	19	20	21	22	23
																			1				1	_
S				T	Т	╆	F	F	F	Т	T	F	F	F	T	F	T	H	F		F	F		

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 I: 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

# **Reconstructing Document:**

Define array D(I,..n):

If S(k) = True, then D(k) = starting position of the word that ends at <math>x[k]

Reconstruct text by following these pointers.

#### **STEP 3: Order of Subtasks**

X		Α	Z	0	Ν	Υ	M	0	U	S	Α	R	R	Α	Y	0	F	L	Ε	Т	Т	Е	R	S
k	0		2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
S		H	Т	Т	Т	F	F	F	F	Т	Т	F	F	F	Т	F	Т	F	F	Т	F	F	Т	Т
D		I		2		•	•	•	-			-	1			•		-				1		

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 I: 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

# **Reconstructing Document:**

Define array D(I,..n):

If S(k) = True, then D(k) = starting position of the word that ends at x[k]

Reconstruct text by following these pointers.

#### **STEP 3: Order of Subtasks**

X		Α	N	0	Z	Υ	M	0	U	S	Α	R	R	Α	Y	0	F	L	Ε	Т	Т	Е	R	S
k	0	I	2	3	4	5	6	7	8	9	10	П	12	13	14	15	16	17	18	19	20	21	22	23
S		Т	Т	Т	Т	F	F	F	F	Τ	Т	F	F	F	$\dashv$	F	Т	F	F	T	F	귀	Т	Т

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 I: 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

# **Reconstructing Document:**

Define array D(I,..n):

If S(k) = True, then D(k) = starting position of the word that ends at <math>x[k]

Reconstruct text by following these pointers.

#### **STEP 3: Order of Subtasks**

X		Α	Z	0	Z	Υ	M	0	U	S	Α	R	R	Α	Υ	0	F	L	Ε	Т	Т	Ε	R	S
k	0		2	3	4	5	6	7	8	9	10	П	12	13	14	15	16	17	18	19	20	21	22	23
S			Т	7	7	П	П	П	П	4	4	П		П	4	П	1		П	4		П	$\perp$	Т
				ı		ı	ı	I	Г	I	I	Г	Г	Г	I	Г		Г	Г		Г		I	I

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 I: 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

# **Reconstructing Document:**

Define array D(I,..n):

If S(k) = True, then D(k) = starting position of the word that ends at <math>x[k]

Reconstruct text by following these pointers.

#### **STEP 3: Order of Subtasks**

X		Α	Z	0	Z	Υ	M	0	J	S	Α	R	R	Α	Υ	0	F	L	Ε	Т	Т	Ε	R	S
k	0		2	3	4	5	6	7	8	9	10	П	12	13	14	15	16	17	18	19	20	21	22	23
S		$\vdash$	T	Т	Т	F	F	F	F	Т	Т	F	F	F	Т	F	T	F	F	Т	F	F	Т	Т
D			1	2	3		_	_			10			_	10	-	15		_	17	_	_	17	17

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 I: 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

# **Reconstructing Document:**

Define array D(I,..n):

If S(k) = True, then D(k) = starting position of the word that ends at <math>x[k]

Reconstruct text by following these pointers.

#### **STEP 3: Order of Subtasks**

X		Α	Z	0	Z	Υ	M	0	U	S	Α	R	R	Α	Y	0	F	L	Е	Т	Т	Е	R	S
k	0		2	3	4	5	6	7	8	9	10	П	12	13	14	15	16	17	18	19	20	21	22	23
S		Т	Т	Т	Т	F	F	F	F	Т	Т	F	F	F	Τ	F	Т	F	F	Т	F	F	Т	Т

# **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:**

$$x = A,C,G, T,$$
 If  $x[i] = y[j]$ , then  $y = G,T,$  LCS( $x[1..i-1], y[1..j-1]$ ) +  $x[i]$ 

**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:**

$$x = A,C,G, T,$$
 If  $x[i] = y[j]$ , then  
 $y = G,T,$  LCS( $x[1..i], y[1..j]$ ) = LCS( $x[1..i-1], y[1..j-1]$ ) +  $x[i]$   
 $x = A,C,G,T, A$   
 $y = G,T,$ 

**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:

$$x = A,C,G, T,$$
 If  $x[i] = y[j]$ , then  $y = G,T,$  LCS( $x[1..i],y[1..j]$ ) = LCS( $x[1..i-1],y[1..j-1]$ ) +  $x[i]$   $x = A,C,G,T,A$  Otherwise,  $y = G,T,$  LCS( $x[1..i],y[1..j]$ ) = max(LCS( $x[1..i-1],y[1..j]$ ), LCS( $x[1..i],y[1..j-1]$ ))

**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 I: Define subtasks**

		Α	C	T	G	G	C	T	A	G
	0		2	3	4	5	6	7	8	9
G	I									
4	2									
G	3									
A	4									
O	5									
A	6									
G	7									
T	8									
T	9									

**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 I: Define subtasks**

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

Output of algorithm = S(m,n)

### **STEP 2: Express recursively**

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

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

**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 I: Define subtasks**

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

Output of algorithm = S(m,n)

### **STEP 2: Express recursively**

$$S(i,j) = S(i-1,j-1) + I, if x[i] = y[j]$$
  
= 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	Ι	2	3	4	5	6	7	8	9
G										
T	2									
G	3									
A	4									
C	5									
Α	6									
G	7									
T	8									
T	9									

**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 I: Define subtasks**

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

Output of algorithm = S(m,n)

## **STEP 2: Express recursively**

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

### **STEP 3: Order of subtasks**

Row by row, left to right

		Α	C	T	G	G	C	T	Α	G
	0		2	3	4	5	6	7	8	9
G										
<b>–</b>	2									
G	3									
A	4									
C	5									
A	6									
G	7									
T	8									
T	9									

**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 I: Define subtasks**

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

Output of algorithm = S(m,n)

## **STEP 2: Express recursively**

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

### **STEP 3: Order of subtasks**

Row by row, left to right

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

**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 I: Define subtasks**

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

Output of algorithm = S(m,n)

## **STEP 2: Express recursively**

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

### **STEP 3: Order of subtasks**

Row by row, left to right

		Α	C	T	G	G	C	T	A	G
	0		2	3	4	5	6	7	8	9
G		0	0	0	—					
<b>–</b>	2									
G	3									
Α	4									
С	5									
A	6									
G	7									
T	8									
T	9									

**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 I: Define subtasks**

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

Output of algorithm = S(m,n)

## **STEP 2: Express recursively**

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

### **STEP 3: Order of subtasks**

Row by row, left to right

		A	U	T	G	G	U	H	A	G
	0		2	3	4	5	6	7	8	9
G	I	0	0	0			_	I		
T	2									
G	3									
A	4									
С	5									
A	6									
G	7									
T	8									
T	9									

**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 I: Define subtasks**

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

Output of algorithm = S(m,n)

## **STEP 2: Express recursively**

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

### **STEP 3: Order of subtasks**

Row by row, left to right

		Α	C	T	G	G	C	T	Α	G
	0		2	3	4	5	6	7	8	9
G		0	0	0	_	_	_		-	I
H	2	0	0							
G	3									
A	4									
C	5									
A	6									
G	7									
T	8									
T	9									

**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 I: Define subtasks**

Output of algorithm = S(m,n)

## **STEP 2: Express recursively**

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

### **STEP 3: Order of subtasks**

Row by row, left to right

		Α	C	T	G	G	C	T	A	G
	0		2	3	4	5	6	7	8	9
G		0	0	0	_	_				I
<b>–</b>	2	0	0	_						
G	3									
A	4									
C	5									
A	6									
G	7									
T	8									
T	9									

**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 I: Define subtasks**

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

Output of algorithm = S(m,n)

## **STEP 2: Express recursively**

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

### **STEP 3: Order of subtasks**

Row by row, left to right

		Α	C	T	G	G	C	T	Α	G
	0		2	3	4	5	6	7	8	9
G	I	0	0	0		_		-	ı	
7	2	0	0	I						
G	3									
A	4									
C	5									
A	6									
G	7									
-	8									
T	9									

**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 I: Define subtasks**

Output of algorithm = S(m,n)

## **STEP 2: Express recursively**

$$S(i,j) = S(i-1,j-1) + I, if x[i] = y[j]$$
  
= 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		2	3	4	5	6	7	8	9
G		0	0	0	_	_	_			I
T	2	0	0	_	_	_				
G	3									
Α	4									
C	5									
A	6									
G	7									
T	8									
T	9									

**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 I: Define subtasks**

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

Output of algorithm = S(m,n)

## **STEP 2: Express recursively**

$$S(i,j) = S(i-1,j-1) + I, if x[i] = y[j]$$
  
= 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	_	2	3	4	5	6	7	8	9
G	-	0	0	0	_	_				
T	2	0	0	_	_	_		2		
G	3									
A	4									
C	5									
A	6									
G	7									
T	8									
T	9									

**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 I: Define subtasks**

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

Output of algorithm = S(m,n)

## **STEP 2: Express recursively**

$$S(i,j) = S(i-1,j-1) + I, if x[i] = y[j]$$
  
= 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	_	2	3	4	5	6	7	8	9
G		0	0	0	_				I	
T	2	0	0			I		2	2	2
G	3	0	0	_	2	2	2	2	2	3
Α	4	I	I	ı	2	2	2	2	3	3
C	5									
A	6									
G	7									
T	8									
T	9									

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

#### **STEP I: Define subtasks**

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

Output of algorithm = S(m,n)

## **STEP 2: Express recursively**

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

### **STEP 3: Order of subtasks**

Row by row, left to right

### **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]
```

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

#### **STEP I: Define subtasks**

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

Output of algorithm = S(m,n)

## **STEP 2: Express recursively**

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

### **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,j], S[i,j-1]}
return S[n,m]
```

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

### **STEP I: Define subtasks**

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

Output of algorithm = S(m,n)

## **STEP 2: Express recursively**

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

### **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]
```

How to reconstruct the actual subsequence?

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

### **STEP I: Define subtasks**

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

## **STEP 2: Express recursively**

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

#### To reconstruct LCS:

Define L(i, j):

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

		A	C	H	G	G	U	T	A	G
	0	_	2	3	4	5	6	7	8	9
G	I	0	0	0	ı	ı	ı	ı	ı	
T	2	0	0	I	ı	ı	ı	2	2	2
G	3	0	0		2	2	2	2	2	3
Α	4				2	2	2	2	3	3

		A	C	T	G	G	C	T	A	G
	0	_	2	3	4	5	6	7	8	9
G	ı	<b>\</b>								
4	2									
G	3									
Α	4									

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

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

### **STEP I: Define subtasks**

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

## **STEP 2: Express recursively**

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

#### To reconstruct LCS:

Define L(i, j):

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

		A	C	H	G	G	U	T	A	G
	0	_	2	3	4	5	6	7	8	9
G		0	0	0						
T	2	0	0					2	2	2
G	3	0	0	I	2	2	2	2	2	3
A	4				2	2	2	2	3	3

		Α	C	T	G	G	C	T	Α	G
	0	—	2	3	4	5	6	7	8	9
G		<b>\</b>	<b>\</b>	<b>\</b>						
H	2									
G	3									
A	4									

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

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

### **STEP I: Define subtasks**

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

## **STEP 2: Express recursively**

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

#### To reconstruct LCS:

Define L(i, j):

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

		Α	C	H	G	G	U	H	A	G
	0	I	2	3	4	5	6	7	8	9
G	П	0	0	0		ı				
T	2	0	0	I				2	2	2
G	3	0	0	I	2	2	2	2	2	3
A	4	Ī			2	2	2	2	3	3

		A	C	T	G	G	C	T	A	G
	0	_	2	3	4	5	6	7	8	9
G	ı	<b>\</b>	<b>\</b>	<b>\</b>	K					
T	2									
G	3									
A	4									

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

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

### **STEP I: Define subtasks**

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

## **STEP 2: Express recursively**

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

#### To reconstruct LCS:

Define L(i, j):

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

		A	C	H	G	G	U	T	A	G
	0	_	2	3	4	5	6	7	8	9
G		0	0	0						ı
T	2	0	0		I	ı	ı	2	2	2
G	3	0	0	I	2	2	2	2	2	3
A	4				2	2	2	2	3	3

		Α	C	T	G	G	C	T	A	G
	0	_	2	3	4	5	6	7	8	9
G	ı	<b>\</b>	<b>\</b>	<b>\</b>	K	K				
-	2									
G	3									
Α	4									

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

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

### **STEP I: Define subtasks**

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

## **STEP 2: Express recursively**

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

#### To reconstruct LCS:

Define L(i, j):

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

		A	C	H	G	G	U	T	A	G
	0	_	2	3	4	5	6	7	8	9
G	I	0	0	0	ı	ı	ı	ı	ı	
T	2	0	0	I	ı	ı	ı	2	2	2
G	3	0	0		2	2	2	2	2	3
Α	4				2	2	2	2	3	3

		A	C	T	G	G	C	T	A	G
	0	_	2	3	4	5	6	7	8	9
G	ı	<b>\</b>	<b>\</b>	<b>\</b>	K	K	<b>\</b>			
4	2									
G	3									
Α	4									

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

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

### **STEP I: Define subtasks**

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

## **STEP 2: Express recursively**

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

#### To reconstruct LCS:

Define L(i, j):

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

		Α	C	H	G	G	U	H	A	G
	0	I	2	3	4	5	6	7	8	9
G	П	0	0	0		-				
T	2	0	0	I				2	2	2
G	3	0	0	I	2	2	2	2	2	3
A	4	Ī			2	2	2	2	3	3

		A	С	T	G	G	C	T	A	G
	0	_	2	3	4	5	6	7	8	9
G	ı	<b>\</b>	<b>\</b>	<b>\</b>	K	K	<b>\</b>	<b>\</b>	<b>+</b>	X
4	2									
G	3									
A	4									

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

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

### **STEP I: Define subtasks**

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

## **STEP 2: Express recursively**

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

#### To reconstruct LCS:

Define L(i, j):

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

		4	U	H	G	G	U	H	A	G
	0	_	2	3	4	5	6	7	8	9
G		0	0	0		I				1
7	2	0	0	I	ı	ı		2	2	2
G	3	0	0		2	2	2	2	2	3
A	4				2	2	2	2	3	3

		Α	C	T	G	G	C	T	Α	G
	0	_	2	3	4	5	6	7	8	9
G		+	<b>\</b>	<b>\</b>	K	K	<b>\</b>	<b>\</b>	<b>+</b>	X
T	2	+	<b>+</b>	X	<b>\</b>	<b>+</b>	+	X	<b>+</b>	<b>+</b>
G	3	<b>+</b>	<b>+</b>	<b>↑</b>	K	X	<b>\</b>	<b>\</b>	1	×
Α	4	K	<b>+</b>	<b>+</b>	<b>+</b>	<b>\</b>	<b>+</b>	<b>+</b>	K	<b>+</b>

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

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

### **STEP I: Define subtasks**

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

## **STEP 2: Express recursively**

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

#### To reconstruct LCS:

Define L(i, j):

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

			A	C	H	G	G	C	<b>T</b>	A	G
	0	)	_	2	3	4	5	6	7	8	9
G			0	0	0		ı				ı
T	' 2	<b>)</b>	0	0	ı	ı	I	ı	2	2	2
G	i 3	}	0	0	I	2	2	2	2	2	3
A	4	ŀ				2	2	2	2	3	3

		A	C	T	G	G	C	T	Α	G
	0	_	2	3	4	5	6	7	8	9
G		+	<b>\</b>	<b>\</b>	K	K	<b>\</b>	<b>\</b>	<b>+</b>	X
T	2	+	<b>+</b>	X	<b>\</b>	<b>→</b>	+	X	<b>+</b>	<b>+</b>
G	3	<b>+</b>	<b>+</b>	<b>1</b>	K	K	<b>→</b>	<b>+</b>	<b>+</b>	X
Α	4	K	<b>+</b>	<b>+</b>	<b>+</b>	<b>+</b>	<b>+</b>	<b>+</b>	K	<b>+</b>

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

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

### **STEP I: Define subtasks**

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

## **STEP 2: Express recursively**

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

#### To reconstruct LCS:

Define L(i, j):

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

		4	U	H	G	G	U	H	4	G
	0	_	2	3	4	5	6	7	8	9
G		0	0	0					I	
7	2	0	0	I	I	ı	ı	2	2	2
G	3	0	0		2	2	2	2	2	3
Α	4				2	2	2	2	3	3

		Α	С	T	G	G	С	T	A	G
	0	_	2	3	4	5	6	7	8	9
G		<b>+</b>	<b>\</b>	<b>\</b>	K	K	<b>\</b>	<b>\</b>	<b>+</b>	X
4	2	<b>+</b>	<b>\</b>	X	<b>\</b>	<b>\</b>	<b>+</b>	X	<b>1</b>	<b>\</b>
G	3	+	<b>+</b>	<b>1</b>	K	K	<b>\</b>	+	<b>+</b>	X
A	4	K	+	+	+	+	+	+	K	<b>+</b>

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

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

### **STEP I: Define subtasks**

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

## **STEP 2: Express recursively**

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

#### To reconstruct LCS:

Define L(i, j):

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

		A	U	H	G	G	U	H	A	G
	0	_	2	3	4	5	6	7	8	9
G		0	0	0						1
7	2	0	0	I	I	ı	ı	2	2	2
G	3	0	0		2	2	2	2	2	3
Α	4				2	2	2	2	3	3

		A	C	T	G	G	C	T	A	G
	0		2	3	4	5	6	7	8	9
G		+	<b>\</b>	<b>\</b>	K	K	<b>\</b>	<b>\</b>	<b>+</b>	X
T	2	+	<b>+</b>	X	<b>\</b>	<b>+</b>	<b>+</b>	X	<b>+</b>	<b>+</b>
G	3	<b>+</b>	<b>→</b>	<b>1</b>	K	K	<b>+</b>	<b>+</b>	<b>+</b>	X
A	4	K	+	+	+	+	<b>+</b>	+	K	<b>+</b>

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

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

### **STEP I: Define subtasks**

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

## **STEP 2: Express recursively**

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

#### To reconstruct LCS:

Define L(i, j):

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

		A	C	H	G	G	U	T	A	G
	0	_	2	3	4	5	6	7	8	9
G		0	0	0						ı
T	2	0	0	I	ı	ı	ı	2	2	2
G	3	0	0	I	2	2	2	2	2	3
Α	4	I	I		2	2	2	2	3	3

		A	С	T	G	G	C	T	A	G
	0		2	3	4	5	6	7	8	9
G		+	<b>\</b>	<b>\</b>	K	K	<b>\</b>	<b>\</b>	<b>+</b>	X
4	2	+	<b>\</b>	X	<b>\</b>	<b>+</b>	<b>+</b>	X	<b>+</b>	<b>+</b>
G	3	+	<b>+</b>	<b>1</b>	K	K	<b>+</b>	<b>+</b>	<b>+</b>	X
A	4	K	+	+	+	+	+	+	K	<b>+</b>

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

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

### **STEP I: Define subtasks**

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

## **STEP 2: Express recursively**

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

#### To reconstruct LCS:

Define L(i, j):

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

		4	U	H	G	G	U	H	A	G
	0	_	2	3	4	5	6	7	8	9
G	I	0	0	0						1
T	2	0	0		I	ı	ı	2	2	2
G	3	0	0	I	2	2	2	2	2	3
A	4				2	2	2	2	3	3

		Α	С	T	G	G	C	T	Α	G
	0	_	2	3	4	5	6	7	8	9
G		+	<b>\</b>	<b>\</b>	K	K	<b>\</b>	<b>\</b>	<b>+</b>	X
4	2	+	<b>\</b>	X	<b>\</b>	<b>→</b>	<b>+</b>	X	<b>+</b>	<b>+</b>
G	3	<b>+</b>	<b>\</b>	<b>↑</b>	K	K	<b>+</b>	<b>+</b>	1	X
Α	4	K	<b>+</b>	<b>+</b>	<b>+</b>	<b>+</b>	<b>+</b>	+	K	<b>+</b>

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

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

### **STEP I: Define subtasks**

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

## **STEP 2: Express recursively**

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

#### To reconstruct LCS:

Define L(i, j):

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

			A	U	H	G	G	U	H	A	G
		0	_	2	3	4	5	6	7	8	9
	<b>L</b> )		0	0	0		ı				
٦	7	2	0	0					2	2	2
	<b>[</b> ]	3	0	0		2	2	2	2	2	3
1	1	4				2	2	2	2	3	3

		A	С	T	G	G	C	T	A	G
	0	_	2	3	4	5	6	7	8	9
G	I	+	<b>+</b>	<b>+</b>	K	X	<b>\</b>	<b>+</b>	<b>+</b>	×
4	2	<b>\</b>	<b>\</b>	X	<b>\</b>	<b>\</b>	<b>\</b>	X	<b>1</b>	<b>\</b>
G	3	+	<b>+</b>	<b>1</b>	K	K	<b>+</b>	<b>+</b>	<b>+</b>	X
A	4	K	+	+	+	+	+	+	K	<b>+</b>

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

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

### **STEP I: Define subtasks**

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

## **STEP 2: Express recursively**

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

#### To reconstruct LCS:

Define L(i, j):

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

		A	U	H	G	G	C	H	A	G
	0	_	2	3	4	5	6	7	8	9
G	I	0	0	0	ı	ı	ı	ı	I	
_	2	0	0	ı	I	ı	ı	2	2	2
G	3	0	0		2	2	2	2	2	3
Α	4				2	2	2	2	3	3

		A	C	T	G	G	C	T	A	G
	0		2	3	4	5	6	7	8	9
G	I	+	+	+	K	X	<b>\</b>	<b>\</b>	<b>+</b>	×
T	2	+	<b>+</b>	X	<b>\</b>	<b>\</b>	<b>\</b>	X	<b>+</b>	<b>+</b>
G	3	+	<b>→</b>	<b>1</b>	K	K	<b>+</b>	<b>+</b>	+	X
A	4	K	+	+	+	+	+	+	K	<b>+</b>

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

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

### **STEP I: Define subtasks**

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

## **STEP 2: Express recursively**

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

#### To reconstruct LCS:

Define L(i, j):

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

		4	U	H	G	G	U	H	A	G
	0	_	2	3	4	5	6	7	8	9
G		0	0	0			I	I	I	
T	2	0	0	I	ı	ı	ı	2	2	2
G	3	0	0		2	2	2	2	2	3
A	4				2	2	2	2	3	3

		A	C	T	G	G	C	T	A	G
	0	_	2	3	4	5	6	7	8	9
G		+	+	<b>\</b>	K	K	<b>\</b>	<b>\</b>	<b>+</b>	X
T	2	+	<b>+</b>	X	<b>\</b>	<b>+</b>	<b>\</b>	X	<b>+</b>	<b>+</b>
G	3	+	<b>+</b>	<b>1</b>	K	K	<b>+</b>	<b>+</b>	<b>+</b>	X
Α	4	K	<b>+</b>	<b>+</b>	<b>+</b>	<b>+</b>	<b>+</b>	<b>+</b>	K	<b>+</b>

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

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

### **STEP I: Define subtasks**

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

## **STEP 2: Express recursively**

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

#### To reconstruct LCS:

Define L(i, j):

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

		A	C	H	G	G	U	T	A	G
	0	_	2	3	4	5	6	7	8	9
G	I	0	0	0	ı	ı	ı	ı	ı	
T	2	0	0	I	ı	ı	ı	2	2	2
G	3	0	0	I	2	2	2	2	2	3
Α	4	I			2	2	2	2	3	3

		A	С	T	G	G	C	T	A	G
	0	_	2	3	4	5	6	7	8	9
G	I	+	<b>+</b>	<b>+</b>	K	X	<b>\</b>	<b>+</b>	<b>+</b>	×
T	2	+	<b>\</b>	K	<b>\</b>	<b>→</b>	<b>\</b>	K	<b>\</b>	<b>\</b>
G	3	+	<b>+</b>	<b>1</b>	K	K	<b>+</b>	<b>+</b>	<b>+</b>	X
Α	4	K	+	+	+	+	+	+	K	<b>+</b>

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

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

#### **STEP I: Define subtasks**

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

Output of algorithm = S(m,n)

## **STEP 2: Express recursively**

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

### **STEP 3: Order of subtasks**

Row by row, left to right

#### To reconstruct LCS:

Define L(i, j):

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

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.

**Case I:** Input:  $x_1, x_2,...,x_n$  Subproblem:  $x_1, ..., x_i$ .

X1 X2 X3 X4 X5 X6 X7 X8 X9 X10

**Case I:** Input:  $x_1, x_2,...,x_n$  Subproblem:  $x_1, ..., x_i$ .

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

**Case I:** Input:  $x_1, x_2,...,x_n$  Subproblem:  $x_1, ..., x_i$ .

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

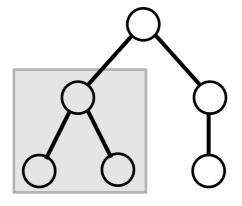
Case 3: Input:  $x_1, x_2,...,x_n$ . Subproblem:  $x_i, ..., x_j$ 

**Case I:** Input:  $x_1, x_2,...,x_n$  Subproblem:  $x_1, ..., x_i$ .

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

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

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

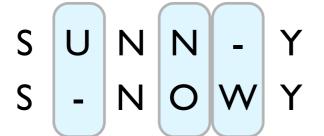


# **Dynamic Programming**

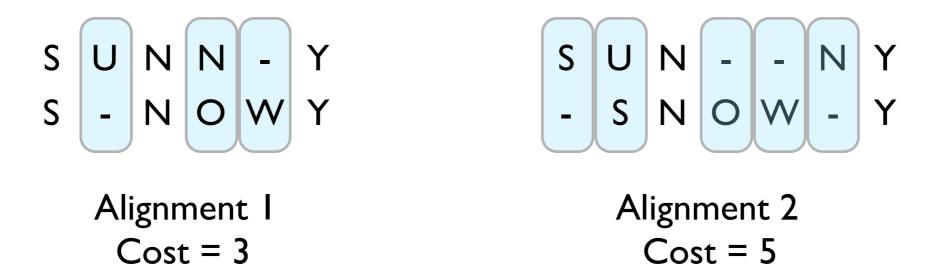
- String Reconstruction
- Longest Common Subsequence
- Edit Distance

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

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

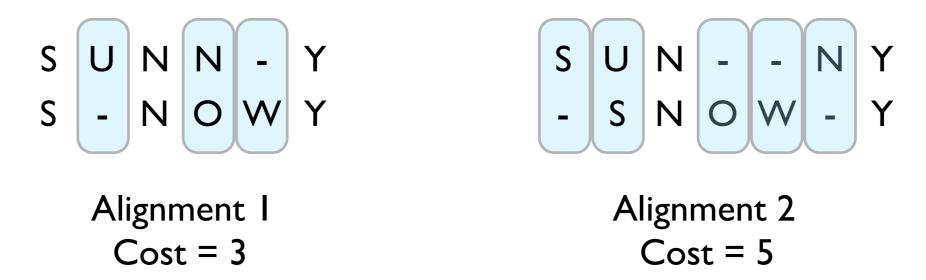


**Alignment:** Convert one string to another using insertions, deletions and substitutions.



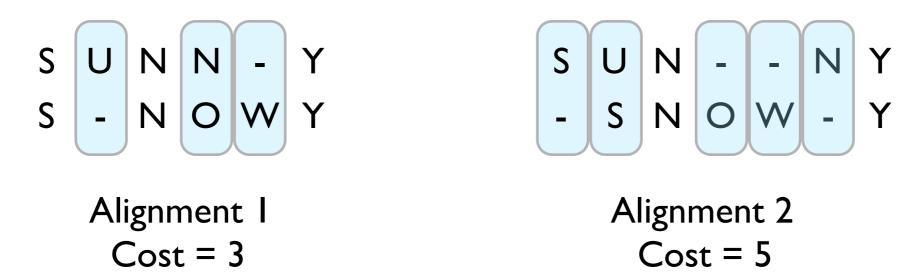
**Edit Distance(x, y):** minimum # of insertions, deletions and substitutions required to convert x to y

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



**Edit Distance(x, y):** minimum # of insertions, deletions and substitutions required to convert x to y Edit Distance(SUNNY, SNOWY) = 3

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



**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)?

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

#### **Example:**

#### **Structure:**

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

- Case I. Align x[I..i-I] and y[I..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.

**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

#### **STEP I: Define subtasks**

#### **STEP 2: Express recursively**

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

#### **STEP 3: Order of subtasks**

			S	U	N	N	Y
	1	0		2	3	4	5
	0						
S	-						
N	2						
<b>0 %</b>	3						
W	4						
Y	5						

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

**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

#### **STEP I: Define subtasks**

#### **STEP 2: Express recursively**

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

#### **STEP 3: Order of subtasks**

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

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

**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

#### **STEP I: Define subtasks**

#### **STEP 2: Express recursively**

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

#### **STEP 3: Order of subtasks**

			S	U	N	Ν	Y
	•	0		2	3	4	5
	0	0		2	3	4	5
S	-		0				
Z	2	2					
0	3	3					
W	4	4					
Y	5	5					

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

**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

#### **STEP I: Define subtasks**

#### **STEP 2: Express recursively**

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

#### **STEP 3: Order of subtasks**

			S	U	N	N	Y
	-	0		2	3	4	5
	0	0	_	2	3	4	5
S	_		0	_			
Z	2	2					
0	3	3					
W	4	4					
Y	5	5					

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

**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

#### **STEP I: Define subtasks**

#### **STEP 2: Express recursively**

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

#### **STEP 3: Order of subtasks**

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

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

**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

#### **STEP I: Define subtasks**

#### **STEP 2: Express recursively**

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

#### **STEP 3: Order of subtasks**

			S	U	N	N	Y
	-	0		2	3	4	5
	0	0	_	2	3	4	5
S	_		0		2	3	4
Z	2	2	_				
0	3	3					
W	4	4					
Y	5	5					

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

**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

#### **STEP I: Define subtasks**

#### **STEP 2: Express recursively**

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

#### **STEP 3: Order of subtasks**

			S	U	N	N	Y
	ı	0		2	3	4	5
	0	0	_	2	3	4	5
S	_		0		2	3	4
Z	2	2	_	_			
0	3	3					
W	4	4					
Y	5	5					

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

**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

#### **STEP I: Define subtasks**

#### **STEP 2: Express recursively**

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

#### **STEP 3: Order of subtasks**

			S	U	N	N	Y
	-	0		2	3	4	5
	0	0	_	2	3	4	5
S	_		0		2	3	4
Z	2	2	_	_	ı		
0	3	3					
W	4	4					
Y	5	5					

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

**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

#### **STEP I: Define subtasks**

#### **STEP 2: Express recursively**

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

#### **STEP 3: Order of subtasks**

			S	U	N	N	Y
	1	0		2	3	4	5
	0	0		2	3	4	5
S	-		0	_	2	3	4
Z	2	2		_	ı	2	
0	3	3					
W	4	4					
Y	5	5					

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

**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

#### **STEP I: Define subtasks**

#### **STEP 2: Express recursively**

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

#### **STEP 3: Order of subtasks**

			S	U	N	N	Y
	ı	0		2	3	4	5
	0	0	_	2	3	4	5
S	-		0	_	2	3	4
Z	2	2	_			2	3
0	3	3					
W	4	4					
Y	5	5					

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

**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

#### **STEP I: Define subtasks**

#### **STEP 2: Express recursively**

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

#### **STEP 3: Order of subtasks**

Row by row, left to right

			S	U	Z	N	Y
	-	0		2	3	4	5
	0	0	I	2	3	4	5
S	_		0	_	2	3	4
Z	2	2	_	_		2	3
0	3	3					
W	4	4					
Y	5	5					

**Running Time** = O(mn)

How to reconstruct the best alignment?