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!)
Given: document \( x[1..n] \) : an array of characters

dictionary function \( \text{dict}(w) \): returns true if \( w \) is a valid word

Is \( x \) a sequence of valid words?

**Example:**
\( x = \text{anonymousarrayofletters} \) : \textbf{True}
\( x = \text{anhuymousarrayofhetters} \) : \textbf{False}
String reconstruction

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

Example:
$x = \text{anonymousarrayofletters} : \text{True}$
$x = \text{anhuymousarrayofhetters} : \text{False}$

STEP 1: Define subtask
$S(k) = \text{True} \quad \text{if } x[1..k] \text{ is a valid sequence of words}$
$\quad \text{False} \quad \text{otherwise}$
Output of algorithm = $S(n)$
String reconstruction

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

Example:
$x = \text{anynymousarrayofletters}$ : True
$x = \text{anhuymousarrayofhetters}$ : False

STEP 1: Define subtask
$S(k) = \text{True}$ if $x[1..k]$ is a valid sequence of words
 otherwise $S(k) = \text{False}$
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 $\text{dict}(w)$: returns true if $w$ is a valid word  
Is $x$ a sequence of valid words?

#### STEP 1: Define Subtask

$S(k) = \begin{cases} 
\text{True} & \text{if } x[1..k] \text{ is a valid sequence of words} \\
\text{False} & \text{otherwise} 
\end{cases}$

| $x$  | 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 $\text{dict}(w)$: returns true if $w$ is a valid word
Is $x$ a sequence of valid words?

**STEP 1: Define Subtask**

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

**STEP 2: Express Recursively**

$S(k) = \text{True}$ 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  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
Given: document $x[1..n]$ : an array of characters
dictionary function $\text{dict}(w)$: returns true if $w$ is a valid word
Is $x$ a sequence of valid words?

**STEP 1: Define Subtask**

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

**STEP 2: Express Recursively**

$S(k) = \text{True}$ 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!]
String reconstruction

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

**STEP 1: Define Subtask**

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

$S(k) = \text{False} \quad \text{otherwise}$

**STEP 2: Express Recursively**

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

**STEP 3: Order of Subtasks**

$S(1), S(2), S(3), ..., S(n) \ [ \text{Do not solve recursively!} \ ]$

<table>
<thead>
<tr>
<th>$k$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
String reconstruction

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

**STEP 1: Define Subtask**

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

**STEP 2: Express Recursively**

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

**STEP 3: Order of Subtasks**

$S(1), S(2), S(3), ..., S(n) \ [\text{Do not solve recursively!}]$

| $x$  | 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
$S(k) = \text{True} \quad \text{if } x[1..k] \text{ is a valid sequence of words}$
$\text{False} \quad \text{otherwise}$

STEP 2: Express Recursively
$S(k) = \text{True if } \exists j < k \text{ s.t. } S(j) \text{ is True, and } x[j+1..k] \text{ is a valid word}$

STEP 3: Order of Subtasks
$S(1), S(2), S(3), ..., S(n) \ [ \text{Do not solve recursively!} \ ]$

<table>
<thead>
<tr>
<th>$x$</th>
<th>ANONY MOU S A R R A Y O F L E T T E R S</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23</td>
</tr>
<tr>
<td>$S$</td>
<td>T T T</td>
</tr>
</tbody>
</table>
String reconstruction

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

**STEP 1: Define Subtask**
$S(k) = \text{True} \quad \text{if } x[1..k] \text{ is a valid sequence of words}$
    $\text{False} \quad \text{otherwise}$

**STEP 2: Express Recursively**
$S(k) = \text{True} \iff \exists \ j < k \text{ s.t. } S(j) \text{ is True, and } x[j+1..k] \text{ is a valid word}$

**STEP 3: Order of Subtasks**
$S(1), S(2), S(3), \ldots, S(n) \ [ \text{Do not solve recursively!} \ ]$

| $x$ | 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 | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T |
## String reconstruction

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

### STEP 1: Define Subtask

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

### STEP 2: Express Recursively

$S(k) = \text{True}$ 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 | 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 ∃ 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! ]
### String reconstruction

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

#### STEP 1: Define Subtask

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

#### STEP 2: Express Recursively

$S(k) = \text{True}$ 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! ]

<table>
<thead>
<tr>
<th>$x$</th>
<th>ANONYMOUS</th>
<th>ARRAY</th>
<th>LETTERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$S$</td>
<td>T T T T T F F F</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
String reconstruction

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

**STEP 1: Define Subtask**

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

$S(k) = \text{False}$ otherwise

**STEP 2: Express Recursively**

$S(k) = \text{True}$ 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! ]

|   | A | N | O | N | Y | M | O | U | S | A | R | R | A | Y | O | F | L | E | T | T | E | R | S |
|x | T | T | T | T | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F |
|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|
String reconstruction

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

**STEP 1: Define Subtask**

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

**STEP 2: Express Recursively**

$S(k) = \text{True}$ iff $\exists j < k \text{ s.t. } S(j)$ 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 | T | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F |
String reconstruction

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

**STEP 1: Define Subtask**

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

False otherwise

**STEP 2: Express Recursively**

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

**STEP 3: Order of Subtasks**

$S(1), S(2), S(3), \ldots, S(n)$ [ Do not solve recursively! ]
String reconstruction

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

**STEP 1: Define Subtask**

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

**STEP 2: Express Recursively**

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

**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 |
| $S$   | F | T | T | T | T | F | F | F | F | F | F | T | T | T | T |   |   |   |   |   |   |   |   |
String reconstruction

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

**STEP 1: Define Subtask**

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

**STEP 2: Express Recursively**

$S(k) = \text{True}$ 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! ]

<table>
<thead>
<tr>
<th>$x$</th>
<th>ANONYMYS</th>
<th>MOUS</th>
<th>ARRAYY</th>
<th>O</th>
<th>F</th>
<th>LETT</th>
<th>E</th>
<th>R</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$S$</td>
<td>T T T T F F F F T T F F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
String reconstruction

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

Is \( x \) a sequence of valid words?

**STEP 1: Define Subtask**

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

**STEP 2: Express Recursively**

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

**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 | T | F | F | F | F | F | F | T | T | F | F | F | F | | | | | | | |
### String reconstruction

**STEP 1: Define Subtask**

\[ S(k) = \text{True} \quad \text{if} \; x[1..k] \; \text{is a valid sequence of words} \]
\[ \text{False} \quad \text{otherwise} \]

**STEP 2: Express Recursively**

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

**STEP 3: Order of Subtasks**

\[ S(1), S(2), S(3), ..., S(n) \; \text{[ Do not solve recursively! ]} \]

---

**Given:**
- Document \( x[1..n] \): an array of characters
- Dictionary function \( \text{dict}(w) \): returns true if \( w \) is a valid word

Is \( x \) a sequence of valid words?
String reconstruction

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

**STEP 1: Define Subtask**

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

**STEP 2: Express Recursively**

$S(k) = \text{True}$ 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! ]

<table>
<thead>
<tr>
<th>$x$</th>
<th>ANONYMOS</th>
<th>ARRAY</th>
<th>OFLETTERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$</td>
<td>0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$S$</td>
<td>T T T T F F F F T T F F F F T F F F F F T T</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
String reconstruction

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

**STEP 1: Define Subtask**
\[ S(k) = \text{True} \quad \text{if} \quad x[1..k] \text{ is a valid sequence of words} \]
\[ S(k) = \text{False} \quad \text{otherwise} \]

**STEP 2: Express Recursively**
\[ S(k) = \text{True} \iff \exists \; j < k \; \text{s.t.} \; S(j) \text{ is True,} \]
\[ \text{and} \quad x[j+1..k] \text{ is a valid word} \]

**STEP 3: Order of Subtasks**
\( S(1), S(2), S(3), \ldots, S(n) \)

**Algorithm:**
\[ S[0] = \text{true} \]
for \( k = 1 \) to \( n \):
\[ S[k] = \text{false} \]
for \( j = 1 \) to \( k \):
\[ \text{if} \; S[j-1] \; \text{and} \; \text{dict}(x[j..k]) \]
\[ S[k] = \text{true} \]

**Reconstructing Document:**
Define array \( D(1,..n) \):
If \( S(k) = \text{true} \), 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 \( \text{dict}(w) \): returns true if \( w \) is a valid word

Is \( x \) a sequence of valid words?

**STEP 1: Define Subtask**

\[ S(k) = \begin{cases} 
  \text{True} & \text{if } x[1..k] \text{ is a valid sequence of words} \\
  \text{False} & \text{otherwise} 
\end{cases} \]

**STEP 2: Express Recursively**

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

**STEP 3: Order of Subtasks**

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

---

**Reconstructing Document:**

Define array \( D(1,\ldots,n) \):

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

Reconstruct text by following these pointers.

---

<table>
<thead>
<tr>
<th>( x )</th>
<th>ANONYMOUS</th>
<th>SAR</th>
<th>RAY</th>
<th>OF</th>
<th>LETTERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>( k )</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>( S )</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>( D )</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
String reconstruction

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

Is $x$ a sequence of valid words?

**STEP 1: Define Subtask**

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

$= \text{False}$ otherwise

**STEP 2: Express Recursively**

$S(k) = \text{True}$ 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 $S(k) = \text{True}$, then $D(k) = \text{starting position of the word that ends at } x[k]$

Reconstruct text by following these pointers.

**Reconstructing Document:**

<table>
<thead>
<tr>
<th>$x$</th>
<th>ANONYMOUS</th>
<th>S</th>
<th>K</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ANONY</td>
<td>T</td>
<td>1</td>
<td>I</td>
</tr>
<tr>
<td>1</td>
<td>MOUS</td>
<td>T</td>
<td>2</td>
<td>I</td>
</tr>
<tr>
<td>2</td>
<td>S</td>
<td>T</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>AR</td>
<td>T</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>ARAY</td>
<td>F</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>YO</td>
<td>F</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>OF</td>
<td>F</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>LETTERS</td>
<td>F</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>E</td>
<td>F</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>R</td>
<td>F</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>S</td>
<td>T</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>A</td>
<td>F</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>R</td>
<td>F</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>A</td>
<td>F</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Y</td>
<td>F</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>O</td>
<td>F</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>F</td>
<td>F</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>L</td>
<td>F</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>E</td>
<td>F</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>T</td>
<td>F</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>T</td>
<td>F</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>E</td>
<td>F</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>R</td>
<td>F</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>S</td>
<td>T</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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.

---

| 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 |
| x | A | N | O | N | Y | M | O | U | S | A | R | R | A | Y | O | F | L | E | T | T | E | R | S |
| S | T | T | T | T | F | F | F | F | F | T | T | F | F | F | T | F | T | F | F | T | F | F | T | T |
| D | 1 | 1 | 2 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |

---
Given: document $x[1..n]$ : an array of characters
dictionary function $\text{dict}(w)$: returns true if $w$ is a valid word
Is $x$ a sequence of valid words?

**STEP 1: Define Subtask**

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

$= \text{False}$ otherwise

**STEP 2: Express Recursively**

$S(k) = \text{True}$ if 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 $S(k) = \text{True}$, then $D(k) = \text{starting position}$ 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 | F | T | F | F | F | T | F | F | F | T | T |
| D   | 1 | 1 | 2 | 3 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
String reconstruction

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

**STEP 1: Define Subtask**

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

**STEP 2: Express Recursively**

$S(k) = \text{True}$ 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 $S(k) = \text{True}$, then $D(k) =$ starting position
of the word that ends at $x[k]$
Reconstruct text by following these pointers.

---

|   | 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 | T | F | F | F | F | F | T | T | F | F | F | F | T | F | F | F | F | T | T |
| $D$ | 1 | 1 | 2 | 3 | - | - | - | - | 1 | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
String reconstruction

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

STEP 1: Define Subtask
$S(k) = \text{True}$ if $x[1..k]$ is a valid sequence of words
$= \text{False}$ otherwise

STEP 2: Express Recursively
$S(k) = \text{True}$ 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 $S(k) = \text{True}$, then $D(k) = \text{starting position}$
of the word that ends at $x[k]$
Reconstruct text by following these pointers.

<table>
<thead>
<tr>
<th>$x$</th>
<th>ANONYMOUS</th>
<th>S</th>
<th>A</th>
<th>R</th>
<th>R</th>
<th>A</th>
<th>Y</th>
<th>O</th>
<th>F</th>
<th>L</th>
<th>E</th>
<th>T</th>
<th>T</th>
<th>E</th>
<th>R</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$S$</td>
<td>T T T T F F F F T T F F F T F T F F F T F F T F F T F T T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$D$</td>
<td>1 1 2 3 - - - - - - - 1 10 - - - 10 - 15 - - 17 - - 17 17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
String reconstruction

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

**STEP 1: Define Subtask**

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

**STEP 2: Express Recursively**

$S(k) = \text{True}$ 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 $S(k) = \text{True}$, then $D(k) =$ starting position of the word that ends at $x[k]$.
Reconstruct text by following these pointers.

Reconstructing Document:

| $x$  | A | N | O | N | Y | M | O | U | S | A | R | A | R | 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 | T | F | F | F | F | T | T | T | F | F | F | T | F | F | F | F | T | 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:**

\[\begin{align*}
x & = A, C, G, T, A, G \\
y & = G, T, C, C, A, C
\end{align*}\]

\[
LCS(x, y) = G, T, A
\]
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:**

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:**

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

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:**

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

If $x[i] = y[j]$, then

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

Otherwise,

$LCS(x[1..i], y[1..j]) = \max(LCS(x[1..i-1], y[1..j]), LCS(x[1..i], y[1..j-1]))$
**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**

$S(i,j) = \text{Length of LCS of } x[1..i] \text{ and } y[1..j]$

Output of algorithm = $S(m,n)$
**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 $x[i] = y[j]$

$= \max(S(i-1,j), S(i,j-1))$, otherwise

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>T</th>
<th>G</th>
<th>G</th>
<th>C</th>
<th>T</th>
<th>A</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Longest Common Subsequence (LCS)

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

**Example:**

\( x = A,C,G,T,A,G \)
\( y = G,T,C,C,A,C \)

\( \text{LCS}(x, y) = G,T,A \)

**STEP 1: Define subtasks**

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

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:**

$x = \text{A, C, G, T, A, G}$

$y = \text{G, T, C, C, A, C}$

$LCS(x, y) = \text{G, T, A}$

**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

**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:**
\[
x = A, C, G, T, A, G
\]
\[
y = G, T, C, C, A, C
\]
\[
\text{LCS}(x, y) = G, T, A
\]

**STEP 1: Define subtasks**
\[
S(i,j) = \text{Length of LCS of } x[1..i] \\
\text{and } y[1..j]
\]
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**

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

**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:**

\(x = A,C,G,T,A,G\)  
\(y = G,T,C,C,A,C\)  
\(\text{LCS}(x, y) = G,T,A\)

**STEP 1: Define subtasks**

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

Output of algorithm = \(S(m,n)\)

**STEP 2: Express recursively**

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

\(= \max(S(i-1,j), S(i,j-1)), \text{ otherwise}\)

**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**

$S(i,j) = \text{Length of LCS of } x[1..i] \text{ and } y[1..j]$

Output of algorithm = $S(m,n)$

**STEP 2: Express recursively**

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

$= \max(S(i-1,j), S(i,j-1))$, otherwise

**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**

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

Output of algorithm = \( S(m,n) \)

**STEP 2: Express recursively**

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

**STEP 3: Order of subtasks**

Row by row, left to right

**Base Case:** Row 0, Column 0
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

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

**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

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**

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

\(\text{Output of algorithm } = S(m,n)\)

**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

**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

**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**
$S(i,j) = \text{Length of LCS of } x[1..i] \text{ and } y[1..j]$

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:** $O(mn)$

**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

**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 $S[n,m]$

**Running Time:** $O(mn)$

How to reconstruct the actual subsequence?
**Longest Common Subsequence (LCS)**

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

**STEP 1: Define subtasks**

$S(i,j) = \text{Length of LCS of } x[1..i] \text{ and } y[1..j]$

**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

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

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

**To reconstruct LCS:**

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

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

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**

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

**STEP 2: Express recursively**

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

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 \( L(m,n) \)

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 $x[i] = y[j]$

$= \max(S(i-1,j), S(i,j-1))$, otherwise

**To reconstruct LCS:**

Define $L(i,j)$:

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

$= (i-1, j)$, otherwise if $S(i-1,j) > S(i,j-1)$

$= (i, j-1)$, otherwise

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**

$S(i,j) = \text{Length of LCS of } x[1..i] \text{ and } y[1..j]$

**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

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**

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

**STEP 2: Express recursively**

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

To reconstruct LCS:

Define L(i, j):

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

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**

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

**STEP 2: Express recursively**

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

To reconstruct LCS:

Define \( L(i,j) \):

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

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**
\[
S(i,j) = \text{Length of LCS of } x[1..i] \text{ and } y[1..j]
\]

**STEP 2: Express recursively**
\[
S(i,j) = \begin{cases} 
S(i-1,j-1) + 1, & \text{if } x[i] = y[j] \\
\max(S(i-1,j), S(i,j-1)), & \text{otherwise}
\end{cases}
\]

To reconstruct LCS:
Define \( L(i, j) \):
\[
L(i, j) = \begin{cases} 
(i-1, j-1), & \text{if } x[i] = y[j] \\
(i-1, j), & \text{if } S(i-1,j) > S(i,j-1) \\
(i, j-1), & \text{otherwise}
\end{cases}
\]

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**

\[ S(i,j) = \text{Length of LCS of } x[1..i] \]
\[ \quad \text{and } y[1..j] \]

**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 \(L(i, j)\):

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

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**

$S(i,j) = \text{Length of LCS of } x[1..i] \\
\text{and } y[1..j]$

**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

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

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

Recall: Row 0 and column 0: Base Case

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>T</th>
<th>G</th>
<th>G</th>
<th>C</th>
<th>T</th>
<th>A</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>G</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>T</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>G</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>A</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
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

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**

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

**STEP 2: Express recursively**

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

To reconstruct LCS:

Define L(i, j):

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

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

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**
- $S(i, j) =$ $S(i-1, j-1) + 1$, 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

**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
- 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

<table>
<thead>
<tr>
<th>Divide-and-conquer</th>
<th>Dynamic programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>A problem of size n is decomposed into a few subproblems which are significantly smaller (e.g. n/2, 3n/4,...)</td>
<td>A problem of size n is expressed in terms of subproblems that are not much smaller (e.g. n-1, n-2,...)</td>
</tr>
<tr>
<td>Therefore, size of subproblems decreases geometrically. eg. n, n/2, n/4, n/8, etc</td>
<td>A recursive algorithm would take exp. time.</td>
</tr>
<tr>
<td>Use a recursive algorithm.</td>
<td>Saving grace: in total, there are only polynomially many subproblems.</td>
</tr>
<tr>
<td></td>
<td>Avoid recursion and instead solve the subproblems one-by-one, saving the answers in a table, in a clever explicit order.</td>
</tr>
</tbody>
</table>
**Case I:** Input: $x_1, x_2, \ldots, x_n$ Subproblem: $x_1, \ldots, x_i$. 

\[
\begin{array}{cccccccc}
X_1 & X_2 & X_3 & X_4 & X_5 & X_6 & X_7 & X_8 & X_9 & X_{10} \\
\end{array}
\]
DP: Common Subtasks

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

**Case 2:** Input: $x_1, x_2, \ldots, x_n$ and $y_1, y_2, \ldots, y_m$ Subproblem: $x_1, \ldots, x_i$ and $y_1, y_2, \ldots, y_j$
**DP: Common Subtasks**

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

![Case 1 Diagram]

**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 2 Diagram]

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

![Case 3 Diagram]
DP: Common Subtasks

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

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

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

- $x_1 \ x_2 \ x_3 \ x_4 \ x_5 \ x_6 \ x_7 \ x_8 \ x_9 \ x_{10}$
- $y_1 \ y_2 \ y_3 \ y_4 \ y_5 \ y_6 \ y_7 \ y_8$

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

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

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

- [Diagram of a rooted tree with a subtree highlighted]
Dynamic Programming

• String Reconstruction
• Longest Common Subsequence
• Edit Distance
**Edit Distance: String Alignment**

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

![Alignment Diagram]

Alignment 1
Cost = 3
Edit Distance: String Alignment

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

Alignment 1
Cost = 3

Alignment 2
Cost = 5
Edit Distance: String Alignment

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

\[ \begin{array}{c}
S & U & N & N & Y \\
S & - & N & O & W & Y
\end{array} \]

Alignment 1
Cost = 3

\[ \begin{array}{c}
S & U & N & - & N & Y \\
- & S & N & O & W & - & Y
\end{array} \]

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.

Alignment 1
Cost = 3

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.

![Alignment Diagram]

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

\[
\text{Edit Distance}(\text{SUNNY}, \text{SNOWY}) = 3
\]

Is \( \text{Edit Distance}(x, y) = \text{Edit Distance}(y, x) \)?
**Edit Distance**

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

**Example:**

$$
\begin{array}{cccc}
S & U & N & - \\
S & - & N & O \\
\end{array}
\begin{array}{c}
Y \\
Y \\
\end{array}
\quad \text{Cost} = 3
$$

**Structure:**

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

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

**Example:**

$x = \text{SUNNY}$  
$y = \text{SNOWY}$  
$\text{Edit-distance}(x, y) = 3$

**STEP 1: Define subtasks**

$E(i, j) = \text{Edit-distance}(x[1..i], y[1..j])$

Output of algorithm = $E(n,m)$

**STEP 2: Express recursively**

$E(i, j) = \min(E(i-1, j) + 1, E(i, j-1) + 1,$

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

**STEP 3: Order of subtasks**

Row by row, left to right

$\text{diff}(a, b) = 0$, if $a = b$

$= 1$, o.w.
**Edit Distance**

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

**Example:**

$x = \text{SUNNY}$ \hspace{1cm} $y = \text{SNOWY}$

$\text{edit-distance}(x, y) = 3$

**STEP 1: Define subtasks**

$E(i,j) = \text{Edit-distance}(x[1..i], y[1..j])$

Output of algorithm = $E(n,m)$

**STEP 2: Express recursively**

$E(i,j) = \min(E(i-1,j) + 1, E(i, j-1) + 1,$

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

$\text{diff}(a, b) = 0$, if $a=b$

$= 1$, o.w.

**STEP 3: Order of subtasks**

Row by row, left to right
**Problem:** Given two strings \( x[1..n] \) and \( y[1..m] \), compute \( \text{edit-distance}(x, y) \)

**Example:**

\[ x = \text{SUNNY} \quad \quad \text{Edit-distance}(x, y) = 3 \]
\[ y = \text{SNOWY} \]

**STEP 1: Define subtasks**

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

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 $\text{edit-distance}(x, y)$

**Example:**

$x = \text{SUNNY}$  
$y = \text{SNOWY}$  

$\text{edit-distance}(x, y) = 3$

**STEP 1: Define subtasks**

$E(i,j) = \text{Edit-distance}(x[1..i], y[1..j])$

Output of algorithm = $E(n,m)$

**STEP 2: Express recursively**

$E(i,j) = \min(E(i-1,j) + 1, E(i, j-1) + 1,$

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

$\text{diff}(a,b) = 0$, if $a=b$

$= 1$, o.w.

**STEP 3: Order of subtasks**

Row by row, left to right
Problem: Given two strings $x[1..n]$ and $y[1..m]$, compute $\text{edit-distance}(x, y)$

Example:

$x = \text{SUNNY}$ \hspace{1cm} $y = \text{SNOWY}$ \hspace{1cm} $\text{edit-distance}(x, y) = 3$

STEP 1: Define subtasks

$E(i,j) = \text{Edit-distance}(x[1..i], y[1..j])$

Output of algorithm = $E(n,m)$

STEP 2: Express recursively

$E(i,j) = \min(E(i-1,j) + 1, E(i, j-1) + 1,$

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

STEP 3: Order of subtasks

Row by row, left to right

$$\text{diff}(a, b) = 0, \text{ if } a=b$$
$$= 1, \text{ o.w.}$$
Problem: Given two strings x[1..n] and y[1..m], compute edit-distance(x, y)

Example:

x = SUNNY    Edit-distance(x, y) = 3
y = SNOWY

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
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 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
**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 1: Define subtasks**

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

Output of algorithm = $E(n,m)$

**STEP 2: Express recursively**

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

**STEP 3: Order of subtasks**

Row by row, left to right

$\text{diff}(a, b) = 0, \text{ if } a=b$

$= 1, \text{o.w.}$
**Problem:** Given two strings \(x[1..n]\) and \(y[1..m]\), compute \(\text{edit-distance}(x, y)\)

**Example:**
- \(x = \text{SUNNY}\)
- \(y = \text{SNOWY}\)

\(\text{edit-distance}(x, y) = 3\)

**STEP 1: Define subtasks**

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

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

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

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>U</th>
<th>N</th>
<th>N</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>S</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>N</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>O</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>4</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Problem:** Given two strings $x[1..n]$ and $y[1..m]$, compute $\text{edit-distance}(x, y)$

**Example:**

$x = \text{SUNNY}$ \\
y = $\text{SNOWY}$ \\
$\text{edit-distance}(x, y) = 3$

**STEP 1: Define subtasks**

$E(i,j) = \text{Edit-distance}(x[1..i], y[1..j])$

Output of algorithm = $E(n,m)$

**STEP 2: Express recursively**

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

**Running Time** = $O(mn)$

**How to reconstruct the best alignment?**