13.3
Checking if a string is in $L^*$
Problem

Input  A string $w \in \Sigma^*$ and access to a language $L \subseteq \Sigma^*$ via function $\text{IsInL}(\text{string } x)$ that decides whether $x$ is in $L$

Goal  Decide if $w \in L^*$ using $\text{IsInL}(\text{string } x)$ as a black box sub-routine
Problem

Input  A string $w \in \Sigma^*$ and access to a language $L \subseteq \Sigma^*$ via function $\text{IsInL}(\text{string } x)$ that decides whether $x$ is in $L$

Goal  Decide if $w \in L$ using $\text{IsInL}(\text{string } x)$ as a black box sub-routine
Problem

**Input** A string \( w \in \Sigma^* \) and access to a language \( L \subseteq \Sigma^* \) via function \( \text{IsInL}(\text{string } x) \) that decides whether \( x \) is in \( L \)

**Goal** Decide if \( w \in L \) using \( \text{IsInL}(\text{string } x) \) as a black box sub-routine
Problem

Input A string \( w \in \Sigma^* \) and access to a language \( L \subseteq \Sigma^* \) via function \( \text{IsInL}(\text{string } x) \) that decides whether \( x \) is in \( L \)

Goal Decide if \( w \in L \) using \( \text{IsInL}(\text{string } x) \) as a black box sub-routine
Problem

Input  A string $w \in \Sigma^*$ and access to a language $L \subseteq \Sigma^*$ via function

$\text{IsInL}(\text{string } x)$ that decides whether $x$ is in $L$
Problem

Input  A string $w \in \Sigma^*$ and access to a language $L \subseteq \Sigma^*$ via function $\text{IsInL}(\text{string } x)$ that decides whether $x$ is in $L$

Goal  Decide if $w \in L^*$ using $\text{IsInL}(\text{string } x)$ as a black box sub-routine
Problem

Input  A string $w \in \Sigma^*$ and access to a language $L \subseteq \Sigma^*$ via function $\text{IsInL}(\text{string } x)$ that decides whether $x$ is in $L$

Goal  Decide if using $\text{IsInL}(\text{string } x)$ as a black box sub-routine

Example 13.1.

Suppose $L$ is $\text{English}$ and we have a procedure to check whether a string/word is in the $\text{English}$ dictionary.

- Is the string “isthisanenglishsentence” in $\text{English}$*?
- Is “stampstamp” in $\text{English}$*?
- Is “zibzzzad” in $\text{English}$*?
Recursive Solution

When is \( w \in L^* \)?

\[
\begin{align*}
w \in L^* & \iff w \in L \text{ or if } w = uv \text{ where } u \in L^* \text{ and } v \in L, |v| \geq 1. 
\end{align*}
\]

Assume \( w \) is stored in array \( A[1..n] \)

```plaintext
IsInL*(A[1..n]):
    If (n = 0) Output YES
    If (IsInL(A[1..n]))
        Output YES
    Else
        For (i = 1 to n - 1) do
            If IsInL*(A[1..i]) and IsInL(A[i + 1..n])
                Output YES
        Output NO
```
Recursive Solution

When is $w \in L^*$?

\[ w \in L^* \iff w \in L \text{ or if } w = uv \text{ where } u \in L^* \text{ and } v \in L, |v| \geq 1. \]

Assume $w$ is stored in array $A[1..n]$

```plaintext
IsInL*(A[1..n]):
  If (n = 0) Output YES
  If (IsInL(A[1..n]))
    Output YES
  Else
    For (i = 1 to n - 1) do
      If IsInL*(A[1..i]) and IsInL(A[i + 1..n])
        Output YES
    Output NO
```
Recursive Solution

When is $w \in L^*$?

$$w \in L^* \iff w \in L \text{ or if } w = uv \text{ where } u \in L^* \text{ and } v \in L, \; |v| \geq 1.$$ 

Assume $w$ is stored in array $A[1..n]$

```
IsInL*(A[1..n]):
    If (n = 0) Output YES
    If (IsInL(A[1..n]))
        Output YES
    Else
        For (i = 1 to n - 1) do
            If IsInL*(A[1..i]) and IsInL(A[i+1..n])
                Output YES
        Output NO
```
Recursive Solution

Assume \( w \) is stored in array \( A[1..n] \)

\[
\text{IsInL}^*(A[1..n]):
\]
If \( n = 0 \) Output YES
If \( \text{IsInL}(A[1..n]) \)
\hspace{1em} Output YES
Else
\hspace{1em} For \( i = 1 \) to \( n - 1 \) do
\hspace{2em} If \( \text{IsInL}^*(A[1..i]) \) and \( \text{IsInL}(A[i + 1..n]) \)
\hspace{3em} Output YES

Output NO

Question: How many distinct sub-problems does \( \text{IsInL}^*(A[1..n]) \) generate? \( O(n) \)
Recursive Solution

Assume $w$ is stored in array $A[1..n]$ 

```python
def IsInL*(A[1..n]):
    if (n == 0) Output YES
    if (IsInL(A[1..n]))
        Output YES
    else
        for (i = 1 to n - 1) do
            if IsInL*(A[1..i]) and IsInL(A[i + 1..n])
                Output YES
        Output NO
```

Question: How many distinct sub-problems does $\text{IsInL}^*(A[1..n])$ generate? $O(n)$
Recursive Solution

Assume $w$ is stored in array $A[1..n]$.

IsInL$(A[1..n])$:
- If ($n = 0$) Output YES
- If (IsInL$(A[1..n])$)
  - Output YES
- Else
  - For ($i = 1$ to $n - 1$) do
    - If IsInL$(A[1..i])$ and IsInL$(A[i + 1..n])$
      - Output YES
  - Output NO

Question: How many distinct sub-problems does IsInL$(A[1..n])$ generate? $O(n)$
Example

Consider string *samiam*
Naming subproblems and recursive equation

After seeing that number of subproblems is $O(n)$ we name them to help us understand the structure better.

**ISL***(i)**: a boolean which is 1 if $A[1..i]$ is in $L^*$, 0 otherwise

**Base case:** $ISL^*(0) = 1$ interpreting $A[1..0]$ as $\epsilon$

**Recursive relation:**

- $ISL^*(i) = 1$ if $\exists j, 0 \leq j < i$ s.t. $ISL^*(j)$ and $IsInL(A[j+1..i])$
- $ISL^*(i) = 0$ otherwise

**Output:** $ISL^*(n)$
Naming subproblems and recursive equation

After seeing that number of subproblems is $O(n)$ we name them to help us understand the structure better.

$\text{ISL}^*(i)$: a boolean which is 1 if $A[1..i]$ is in $L^*$, 0 otherwise

**Base case:** $\text{ISL}^*(0) = 1$ interpreting $A[1..0]$ as $\epsilon$

**Recursive relation:**

- $\text{ISL}^*(i) = 1$ if
  $\exists j, 0 \leq j < i$ s.t $\text{ISL}^*(j)$ and $\text{IslL}(A[j + 1..i])$
- $\text{ISL}^*(i) = 0$ otherwise

**Output:** $\text{ISL}^*(n)$
Naming subproblems and recursive equation

After seeing that number of subproblems is $O(n)$ we name them to help us understand the structure better.

**ISL***(i)**: a boolean which is 1 if $A[1..i]$ is in $L^*$, 0 otherwise

**Base case:** $\text{ISL}^*(0) = 1$ interpreting $A[1..0]$ as $\epsilon$

**Recursive relation:**

- $\text{ISL}^*(i) = 1$ if
  $\exists j, \ 0 \leq j < i \ s.t \ \text{ISL}^*(j)$ and $\text{IsInL}(A[j + 1..i])$
- $\text{ISL}^*(i) = 0$ otherwise

**Output:** $\text{ISL}^*(n)$
Removing recursion to obtain iterative algorithm

Typically, after finding a dynamic programming recursion, we often convert the recursive algorithm into an **iterative** algorithm via **explicit memoization** and **bottom up** computation.

**Why?** Mainly for further optimization of running time and space.

**How?**
- First, allocate a data structure (usually an array or a multi-dimensional array that can hold values for each of the subproblems)
- Figure out a way to order the computation of the sub-problems starting from the base case.

**Caveat:** Dynamic programming is not about filling tables. It is about finding a smart recursion. First, find the correct recursion.
Removing recursion to obtain iterative algorithm

Typically, after finding a dynamic programming recursion, we often convert the recursive algorithm into an iterative algorithm via explicit memoization and bottom up computation.

Why? Mainly for further optimization of running time and space.

How?
- First, allocate a data structure (usually an array or a multi-dimensional array that can hold values for each of the subproblems)
- Figure out a way to order the computation of the sub-problems starting from the base case.

Caveat: Dynamic programming is not about filling tables. It is about finding a smart recursion. First, find the correct recursion.
Removing recursion to obtain iterative algorithm

Typically, after finding a dynamic programming recursion, we often convert the recursive algorithm into an iterative algorithm via explicit memoization and bottom up computation.

Why? Mainly for further optimization of running time and space.

How?
- First, allocate a data structure (usually an array or a multi-dimensional array that can hold values for each of the subproblems)
- Figure out a way to order the computation of the sub-problems starting from the base case.

Caveat: Dynamic programming is not about filling tables. It is about finding a smart recursion. First, find the correct recursion.
Removing recursion to obtain iterative algorithm

Typically, after finding a dynamic programming recursion, we often convert the recursive algorithm into an iterative algorithm via explicit memoization and bottom up computation.

Why? Mainly for further optimization of running time and space.

How?
- First, allocate a data structure (usually an array or a multi-dimensional array that can hold values for each of the subproblems)
- Figure out a way to order the computation of the sub-problems starting from the base case.

Caveat: Dynamic programming is not about filling tables. It is about finding a smart recursion. First, find the correct recursion.
Iterative Algorithm

**IsStringinLstar-Iterative**($A[1..n]$):

boolean $ISL^*[0..(n+1)]$

$ISL^*[0] = TRUE$

for $i = 1$ to $n$ do

  for $j = 0$ to $i - 1$ do
  
    if ($ISL^*[j]$ and $IsInL(A[j+1..i])$)
      $ISL^*[i] = TRUE$
      break

  if ($ISL^*[n] = 1$) Output YES
else Output NO

- Running time: $O(n^2)$ (assuming call to $IsInL$ is $O(1)$ time)
- Space: $O(n)$
Iterative Algorithm

**IsStringinLstar-Iterative**($A[1..n]$):

1. boolean $ISL^*[0..(n + 1)]$
2. $ISL^*[0] = TRUE$
3. for $i = 1$ to $n$ do
   1. for $j = 0$ to $i - 1$ do
      1. if ($ISL^*[j]$ and $IsInL(A[j + 1..i])$)
         1. $ISL^*[i] = TRUE$
         2. break
4. if ($ISL^*[n] = 1$) Output YES
   else Output NO

- **Running time:** $O(n^2)$ (assuming call to $IsInL$ is $O(1)$ time)
- **Space:** $O(n)$
Iterative Algorithm

IsStringinLstar-Iterative($A[1..n]$):
  boolean $ISL^*[0..(n + 1)]$
  $ISL^*[0] = TRUE$
  for $i = 1$ to $n$ do
    for $j = 0$ to $i - 1$ do
      if ($ISL^*[j]$ and $IsInL(A[j + 1..i])$)
        $ISL^*[i] = TRUE$
        break
  if ($ISL^*[n] = 1$) Output YES
  else Output NO

- Running time: $O(n^2)$ (assuming call to $IsInL$ is $O(1)$ time)
- Space: $O(n)$
Iterative Algorithm

(defun IsStringinLstar-Iterative (A [1..n])
  (boolean ISL*[0..(n + 1)])
  ISL*[0] = TRUE
  for i = 1 to n do
    for j = 0 to i - 1 do
      if (ISL*[j] and IsInL(A[j + 1..i]))
        ISL*[i] = TRUE
        break
  if (ISL*[n] = 1) Output YES
  else Output NO

- Running time: \(O(n^2)\) (assuming call to IsInL is \(O(1)\) time)
- Space: \(O(n)\)
Iterative Algorithm

\textbf{IsStringInLstar-Iterative}(A[1..n]):

\begin{verbatim}
boolean ISL*[0..(n + 1)]
ISL*[0] = TRUE
for i = 1 to n do
    for j = 0 to i - 1 do
        if (ISL*[j] and IsInL(A[j + 1..i]))
            ISL*[i] = TRUE
            break

if (ISL*[n] = 1) Output YES
else Output NO
\end{verbatim}

- Running time: $O(n^2)$ (assuming call to IsInL is $O(1)$ time)
- Space: $O(n)$
Example

Consider string *samiam*
THE END

... (for now)