13.3

Checking if a string is in L^*

L^*

Problem

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

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

Problem

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

Goal Decide if $w \in L^{*}$ using **IsInL**(*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 **IsInL**(***string*** x) that decides whether x is in L

*

Goal Decide if $w \in L$ using **IsInL**(***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 **IsInL**(*string x*) that decides whether x is in L

*

Goal Decide if $w \in L$ using **IsInL**(*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 **IsInL**($\textit{string}\ \ x$) that decides whether x is in L

*

Problem

Input	A string $w \in \Sigma^*$ and access to a language $L \subseteq \Sigma^*$ via function **IsInL**$(string\ \ x)$ that decides whether x is in L
Goal	Decide if $w \in L^*$ using **IsInL**$(string\ \ x)$ as a black box sub-routine

Problem

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

Goal Decide if using **IsInL**(*string* x) as a black box sub-routine

Example 13.1.

Suppose L is ***English*** and we have a procedure to check whether a string/word is in the ***English*** dictionary.

- Is the string “isthisanenglishsentence” in ***English****?
- Is “stampstamp” in ***English****?
- Is “zibzzzad” in ***English****?

Recursive Solution

When is $w \in L^*$?

$w \in L^* \iff w \in L$ or if $w = uv$ where $u \in L^*$ 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

When is $w \in L^*$?

$w \in L^* \iff w \in L$ or if $w = uv$ where $u \in L^*$ 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

When is $w \in L^*$?

$w \in L^* \iff w \in L$ or if $w = uv$ where $u \in L^*$ 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 $A[1..n]$

```
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 $\textbf{IsInL*}(A[1..n])$ 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? $O(n)$

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 ϵ

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.

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 ϵ
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.

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

Base case: $\textsf{ISL}^*(0) = 1$ interpreting $A[1..0]$ as ϵ
Recursive relation:

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

Output: $\textsf{ISL}^*(n)$

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($A[1..n]$):
    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($A[1..n]$):
    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($A[1..n]$):
    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($A[1..n]$):
    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:** $O(n)$

Example

Consider string ***samiam***

THE END

…

(for now)