CSCI-UA 480.4: APS
Algorithmic Problem Solving

Some String Algorithms

Instructor: Joanna Klukowska

created based on materials for this class by
Bowen Yu and materials shared by the authors of
the textbook Steven and Felix Halim

String Definitions

a **string** of length consists of characters , , ...,

a **substring** is a sequence of consecutive characters in a string that starts at position and ends at position (inclusive on both ends)

a **prefix** is a substring for which

a **suffix** is a substring for which

a **subsequence** is any sequence of characters in a string in their original order (not necesserily consecutive)

Longest Common Subsequence

The **longest common subsequence** (lcs) of two strings is the longest string that appears as a subsequence in both strings.

Examples:

- "floor" and "door", the lcs is "oor"
- "caged" and "rage", the lcs is "age"
- "capsule" and "recaps", the lcs is "caps"

Longest Common Subsequence

The **longest common subsequence** (lcs) of two strings is the longest string that appears as a subsequence in both strings.

Examples:

- "floor" and "door", the lcs is "oor"
- "caged" and "rage", the lcs is "age"
- "capsule" and "recaps", the lcs is "caps"

Solution

given: two strings and

- function that returns length of the longest common subsequence of the prefixes and

Longest Common Subsequence

The **longest common subsequence** (lcs) of two strings is the longest string that appears as a subsequence in both strings.

Examples:

- "floor" and "door", the lcs is "oor"
- "caged" and "rage", the lcs is "age"
- "capsule" and "recaps", the lcs is "caps"

Solution

given: two strings and

- function that returns length of the longest common subsequence of the prefixes and

[Visualization of the algorithm]

Edit Distance

The **edit distance** between two strings is defined as the minimum number of editing operations that transform one string into the other.

The allowed operations may vary, but are often

- insert a character, "ABC" -> "ABCA"
- remove a character, "ABC" -> "AC"
- replace a character, "ABC" -> "ADC"
  (this one can be thought of as two separate operations of remove followed by insert)

Edit Distance

The **edit distance** between two strings is defined as the minimum number of editing operations that transform one string into the other.

The allowed operations may vary, but are often

- insert a character, "ABC" -> "ABCA"
- remove a character, "ABC" -> "AC"
- replace a character, "ABC" -> "ADC"
  (this one can be thought of as two separate operations of remove followed by insert)

Solution

given: two strings and

- function that returns the edit distance between the prefixes and

Edit Distance

The **edit distance** between two strings is defined as the minimum number of editing operations that transform one string into the other.

The allowed operations may vary, but are often

- insert a character, "ABC" -> "ABCA"
- remove a character, "ABC" -> "AC"
- replace a character, "ABC" -> "ADC"
  (this one can be thought of as two separate operations of remove followed by insert)

Solution

given: two strings and

- function that returns the edit distance between the prefixes and

where when and , otherwise.

Edit Distance

The **edit distance** between two strings is defined as the minimum number of editing operations that transform one string into the other.

The allowed operations may vary, but are often

- insert a character, "ABC" -> "ABCA"
- remove a character, "ABC" -> "AC"
- replace a character, "ABC" -> "ADC"
  (this one can be thought of as two separate operations of remove followed by insert)

Solution

given: two strings and

- function that returns the edit distance between the prefixes and

where when and , otherwise.

Z-Array / Z-Algorithm

What is a Z-array?

Z-Array / Z-Algorithm

What is a Z-array?

The Z-array for a string of length is an array of length in which stores the legnth of the longest substring starting at that is also a prefix of .

Example:

Z-Array / Z-Algorithm

What is a Z-array?

The Z-array for a string of length is an array of length in which stores the legnth of the longest substring starting at that is also a prefix of .

Example:

What is a Z-Algorithm?

Z-Array / Z-Algorithm

What is a Z-array?

The Z-array for a string of length is an array of length in which stores the legnth of the longest substring starting at that is also a prefix of .

Example:

What is a Z-Algorithm?

It's an algorithm that computes the Z-Array for a given string .

Z-Algorithm

Idea: mantain a range such that is a prefix of , the value of has been calculated and is as large as possible.

Observation: is the same as

Challenge: Pattern Matching

Find all locations of a pattern string in a given string .

Challenge: Pattern Matching

Find all locations of a pattern string in a given string .

Solution

- create a new string in which is a special character that does not occur in neither nor
- create the z-array for the new string
- the locations in the z-array for which the value is equal to the length of the pattern string are the location of the pattern in (adjust indexes by subtracting the)

Challenge: Pattern Matching

Find all locations of a pattern string in a given string .

Solution

- create a new string in which is a special character that does not occur in neither nor
- create the z-array for the new string
- the locations in the z-array for which the value is equal to the length of the pattern string are the location of the pattern in (adjust indexes by subtracting the)

Visualization of the solution

Challenge: Finding Borders

A **border** in a string is a substring that is both a prefix and a sufix of that string (but not the entire string, i.e., proper prefix and proper suffix).

Example:

the borders are

Challenge: Finding Borders

Solution

- create the z-array for
- boarders are all suffixes , such that

Challenge: Finding Borders

Solution

- create the z-array for
- boarders are all suffixes , such that

Knuth-Morris-Pratt's (KMP) Algorithm

Another algorithm for pattern matching in strings.

It uses the observation that when a mismatch occurs, the pattern itself contains sufficient information to determine where the next match could begin. This allows to skip re-examination of previously matched characters.

Example of the idea:

Knuth-Morris-Pratt's (KMP) Algorithm

Another algorithm for pattern matching in strings.

It uses the observation that when a mismatch occurs, the pattern itself contains sufficient information to determine where the next match could begin. This allows to skip re-examination of previously matched characters.

Example of the idea:

Knuth-Morris-Pratt's (KMP) Algorithm

Another algorithm for pattern matching in strings.

It uses the observation that when a mismatch occurs, the pattern itself contains sufficient information to determine where the next match could begin. This allows to skip re-examination of previously matched characters.

Example of the idea:

Knuth-Morris-Pratt's (KMP) Algorithm

Another algorithm for pattern matching in strings.

It uses the observation that when a mismatch occurs, the pattern itself contains sufficient information to determine where the next match could begin. This allows to skip re-examination of previously matched characters.

Example of the idea:

Knuth-Morris-Pratt's (KMP) Algorithm

Another algorithm for pattern matching in strings.

It uses the observation that when a mismatch occurs, the pattern itself contains sufficient information to determine where the next match could begin. This allows to skip re-examination of previously matched characters.

Example of the idea:

Knuth-Morris-Pratt's (KMP) Algorithm

Another algorithm for pattern matching in strings.

It uses the observation that when a mismatch occurs, the pattern itself contains sufficient information to determine where the next match could begin. This allows to skip re-examination of previously matched characters.

Example of the idea:

Knuth-Morris-Pratt's (KMP) Algorithm

Another algorithm for pattern matching in strings.

It uses the observation that when a mismatch occurs, the pattern itself contains sufficient information to determine where the next match could begin. This allows to skip re-examination of previously matched characters.

Example of the idea:

Knuth-Morris-Pratt's (KMP) Algorithm

Another algorithm for pattern matching in strings.

It uses the observation that when a mismatch occurs, the pattern itself contains sufficient information to determine where the next match could begin. This allows to skip re-examination of previously matched characters.

Example of the idea:

Knuth-Morris-Pratt's (KMP) Algorithm

Another algorithm for pattern matching in strings.

It uses the observation that when a mismatch occurs, the pattern itself contains sufficient information to determine where the next match could begin. This allows to skip re-examination of previously matched characters.

Example of the idea:

Knuth-Morris-Pratt's (KMP) Algorithm

Another algorithm for pattern matching in strings.

It uses the observation that when a mismatch occurs, the pattern itself contains sufficient information to determine where the next match could begin. This allows to skip re-examination of previously matched characters.

Example of the idea:

Visualization of KMP

another visualization of KMP

Souce code implementation: cpp, java