Algorithms and Data Structures

String Searching

Robert Horvick

www.pluralsight.com

Outline

- **API Overview**
- **Naïve Search Algorithm**
  - Algorithm
  - Performance Considerations
- **Boyer-Moore-Horspool Algorithm**
  - Algorithm
  - Performance Considerations
- **Demo: Search and Replace**

API Overview

- **Search Algorithm**

```
public interface IStringSearchAlgorithm
{
    IEnumerable<ISearchMatch> Search(string toFind, string toSearch);
}
```

- **Search Results**

```
public interface ISearchMatch
{
    int Start { get; }
    int Length { get; }
}
```

Naïve Search Algorithm

- **For startIndex= 0; startIndex < toSearch.Length; startIndex++**
  - matchCount = 0
  - While toSearch[startIndex + matchCount] == toFind[matchCount]
    - matchCount++
    - If toFind.Length == matchCount
      - Match Found!
- **Find “DROP” in “PHIL DROPPED HIS PHONE”**

Naïve Search Code

```
public IEnumerable<ISearchMatch> Search(string toFind, string toSearch)
{
    Validate Parameters are not null

    if (toFind.Length > 0 && toSearch.Length > 0)
    {
        for (int startIndex = 0; startIndex <= toSearch.Length - toFind.Length; startIndex++)
        {
            int matchCount = 0;

            while (Compare(toFind[matchCount], toSearch[startIndex + matchCount]) == 0)
            {
                matchCount++;

                if (toFind.Length == matchCount)
                {
                    yield return new StringSearchMatch(startIndex, matchCount);

                    startIndex += matchCount - 1;
                    break;
                }
            }
        }
    }
}
```

Naïve Search Performance

- **Average Performance**
  - **O(n+m)**
  - n = length of string to search
  - m = length of string to find
- **Worst Case Performance**
  - **O(nm)**
- **Naïve string search requires no pre-processing overhead**
- **Appropriate when the string to search and find are both small**

Boyer-Moore-Horspool Overview

- **A simplification of the Boyer-Moore algorithm developed by Robert Boyer and J Strother Moore.**
- **Two-stage algorithm**
- **Stage 1**
  - Preprocesses the string being searched for to build a table that contains the length to shift when a bad match occurs
  - Classic Boyer-Moore creates a second “good suffix” table
- **Stage 2**
  - The string to find is searched from the last character to the first
  - The bad match table is used to skip characters when a mismatch occurs

Bad Match Table

- **Algorithm**
  - 1. Store the length of the search string as the default shift length
  - 2. For-Each character in the search string
    - Set the shift index for the current character value
- **Example Pattern: "TRUTH"**

```
public BadMatchTable(string pattern)
{
    _defaultValue = pattern.Length;
    _distances = new Dictionary<int, int>();

    for (int i = 0; i < pattern.Length - 1; i++)
    {
        _distances[pattern[i]] = pattern.Length - i - 1;
    }
}
```

Index	Value
?	5
T	1
R	3
U	2

Boyer-Moore-Horspool Algorithm

```
WE HOLD THESE TRUTHS TO BE SELF-EVIDENT
               TRUTH
```

Index	Value
?	5
T	1
R	3
U	2

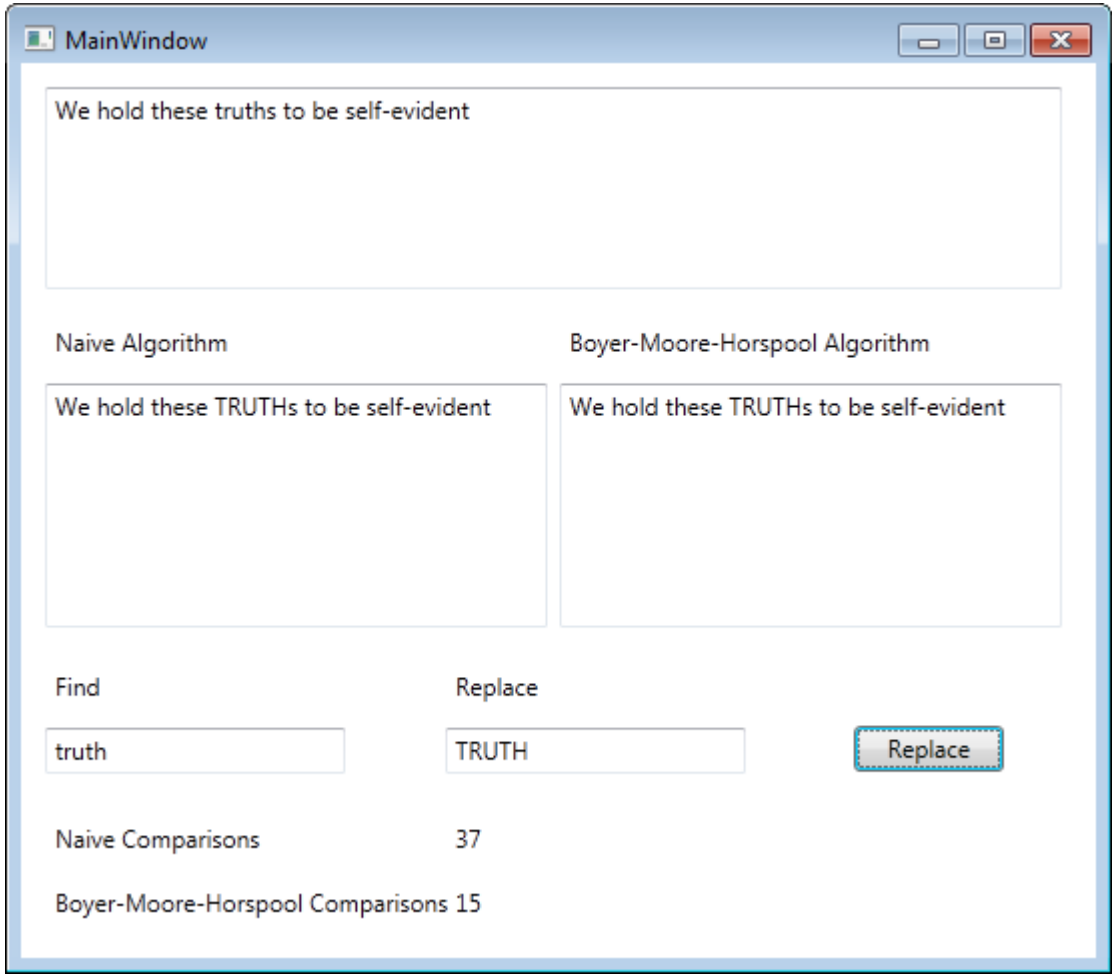
Boyer-Moore-Horspool Code

```
BadMatchTable badMatchTable = new BadMatchTable(pattern);

int currentStartIndex = 0;
while (currentStartIndex <= toSearch.Length - pattern.Length)
{
    int charactersLeftToMatch = pattern.Length - 1;

    while (charactersLeftToMatch >= 0 &&
           Compare(pattern[charactersLeftToMatch], toSearch[currentStartIndex + charactersLeftToMatch]) == 0)
    {
        charactersLeftToMatch--;
    }

    if (charactersLeftToMatch < 0)
    {
        yield return new StringSearchMatch(currentStartIndex, pattern.Length);
        currentStartIndex += pattern.Length;
    }
    else
    {
        currentStartIndex += badMatchTable[toSearch[currentStartIndex + pattern.Length - 1]];
    }
}
```

Boyer-Moore-Horspool Performance

- **Performance improves with the length of the search string**
  - The larger the bad match table, the further the search string can be shifted
  - Less overhead than Boyer-Moore because only one table is created
- **Best Case**
  - **O(n/m)**
  - n = length of string to search
  - m = length of string to find
- **Worst Case Performance**
  - **O(nm)**
- **Appropriate as a general purpose string search algorithm**

Demo: Search and Replace



Summary

- **API Overview**
- **Naïve Search Algorithm**
  - Algorithm
  - Performance Considerations
- **Boyer-Moore-Horspool Algorithm**
  - Algorithm
  - Performance Considerations
- **Demo: Search and Replace**

References

- **String Searching Algorithms**
  - http://en.wikipedia.org/wiki/String_searching_algorithm
- **Boyer-Moore-Horspool Search Algorithm**
  - http://en.wikipedia.org/wiki/Boyer-Moore-Horspool_algorithm
- **Boyer-Moore Search Algorithm**
  - http://en.wikipedia.org/wiki/Boyer–Moore_string_search_algorithm