String Searching Algorithms

Jerry Cimo & Michael Roefs
CS375 Final Project | Fall 2015
Binghamton University

Objective

- ❏ Given a base string, find every instance of a given substring (key string)
- ❏ Return list of integers representing each index of the base string which is the first char of an instance of the key

```
                1         2
0123456789012345678901234567 89
ABCDEFGHELLOIJKLMNHELLOAHJJWJS
        ^         ^
```

Objective

Look at three approaches to solving this problem:

- **Brute Force Approach**
- **Boyer-Moore String Searching Algorithm**
- **Knuth Morris Pratt Algorithm**

We will analyze and compare their process and performance

Brute Force Algorithm (Naive Approach)

- For each character in the base string
  - Check if it is first character of an instance of the key
  - If an instance of key, add index to list of finds

This algorithm reads over the same character many more times than it may need to.

Preprocesses can result in a ‘smarter’ way to search

Boyer-Moore Algorithm

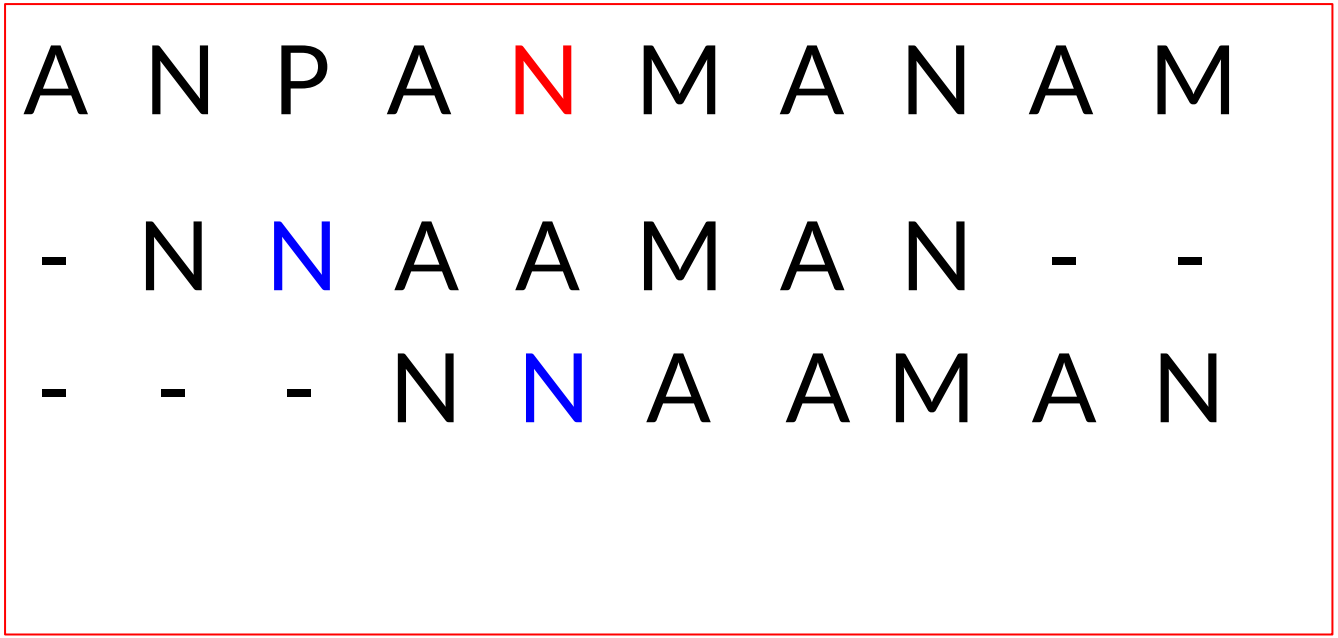
Preprocess:

- Create the “bad character table”

Searching:

- Iterate through the base string in reverse
  - If a mismatch is detected, consult the bad character table to determine how to shift the string
  - This means that unlike the Brute Force and KMP algorithms, we do not examine every character in the base string

Boyer-Moore Algorithm



Knuth-Morris-Pratt Algorithm

Preprocess:

- Create a table (size of key) that recognizes repeat patterns in the key string to be used in the search

Searching:

- Iterate through the base string
  - Check if element of key string
  - Move up a number of characters specified in the preprocessed table

Knuth-Morris-Pratt Algorithm

BASE: ABABCD
KEY: ABC

```
      012345
step 1 ABABCD
       A

      012345
step 2 ABABCD
       AB

      012345
step 3 ABABCD
       ABC

      012345
step 4 ABABCD
        AB

      012345
step 5 ABABCD
         ABC

      012345
step 6 ABABCD
           A
```

Time Complexity Analysis

n - length of key
m - length of base (searchable text)
k - number of unique characters (size of alphabet)

	Brute Force	Boyer-Moore	Knuth-Morris-Pratt
preprocessing time	0	$\Theta(m+k)$	$\Theta(m)$
matching time	$\Theta((n-m)m)$	best: $\Omega(n/m)$ worst: $O(n)$	$\Theta(n)$

Space Complexity Analysis

Brute Force

- No significant additional space

Boyer-Moore

- An additional table of size 256. (The bad character table)

Knuth-Morris-Pratt

- An additional table with size of the length of the key string is used

OIUWQOIUASDLKJALSIKJD
AKSJHALSDHKJHABSKDJH
GALKSDJHLKAJSHDLKJHG
KLA**DEMO**AJKUWASDAKSJ
DHLKJAHSDLKJSHDSLIUH
LKJASLDKJOIWPOQPDAPQ

Conclusions

- The added preprocessing time allowed for a more efficient search
- The Boyer-Moore string searching algorithm has the fastest possible time complexity compared to the other algorithms we looked at, and is the recommended method for string searching

OIUPOJHAKSLDJHKLJHKJD
LKAJHSDIKLJHASKDJHLKA
JSHDIULHAWUIAJHAISDHL
MK**QUESTIONS?**OEUASPQH
REILKJHALKSJDHKJHKJAH
LKJHASLDKJHLKJHASDKIO