Knuth-Morris-Pratt Algorithm

Robin Visser

IOI Training Camp
University of Cape Town

18 April 2015

Overview

Background

- The KMP algorithm searches for occurrences of a word **W** (*needle*) within a main text string **S** (*haystack*).
- Does pre-processing on needle such that, when mismatch occurs, bypasses re-examination of previously matched characters

Background

- The KMP algorithm searches for occurrences of a word **W** (*needle*) within a main text string **S** (*haystack*).
- Does pre-processing on needle such that, when mismatch occurs, bypasses re-examination of previously matched characters

Brute Force

- We could do naive check of whether needle occurs in haystack.
- For random data, this is often sufficient.
  Expected runtime: O($|S|$)
- There are test cases where this gives poor performance.

Example

Find all occurrences of **AAAAAB** in **AAA..AAA**

- Therefore, not useful for contests as worst case runtime is O($|S||W|$)

Brute Force

- We could do naive check of whether needle occurs in haystack.
- For random data, this is often sufficient.
  Expected runtime: $O(|S|)$
- There are test cases where this gives poor performance.

Example

Find all occurrences of **AAAAAB** in **AAA..AAA**

- Therefore, not useful for contests as worst case runtime is $O(|S||W|)$

Brute Force

- We could do naive check of whether needle occurs in haystack.
- For random data, this is often sufficient.
  Expected runtime: $O(|S|)$
- There are test cases where this gives poor performance.

Example

Find all occurrences of **AAAAAB** in **AAA..AAA**

- Therefore, not useful for contests as worst case runtime is $O(|S||W|)$

Brute Force

- We could do naive check of whether needle occurs in haystack.
- For random data, this is often sufficient.
  Expected runtime: $O(|S|)$
- There are test cases where this gives poor performance.

Example

Find all occurrences of **AAAAAB** in **AAA..AAA**

- Therefore, not useful for contests as worst case runtime is $O(|S||W|)$

Brute Force

- We could do naive check of whether needle occurs in haystack.
- For random data, this is often sufficient.
  Expected runtime: O($|S|$)
- There are test cases where this gives poor performance.

Example

Find all occurrences of **AAAAAB** in **AAA..AAA**

- Therefore, not useful for contests as worst case runtime is O($|S||W|$)

Knuth-Morris-Pratt

- Process each character of S one at a time. We keep track of the longest prefix that currently matches.
- When mismatch occurs, fall back to shorter prefix.

Example

Find all occurrences of **ABACABAD** in **ABACABABACABAD**

Knuth-Morris-Pratt

- Process each character of S one at a time. We keep track of the longest prefix that currently matches.
- When mismatch occurs, fall back to shorter prefix.

Example

Find all occurrences of **ABACABAD** in **ABACABABACABAD**

Knuth-Morris-Pratt

- Process each character of S one at a time. We keep track of the longest prefix that currently matches.
- When mismatch occurs, fall back to shorter prefix.

Example

Find all occurrences of **ABACABAD** in **ABACABABACABAD**

Knuth-Morris-Pratt

- Process each character of S one at a time. We keep track of the longest prefix that currently matches.
- When mismatch occurs, fall back to shorter prefix.

Example

Find all occurrences of **ABACABAD** in **ABACABABACABAD**

A	B	A	C	A	B	A	B	A	C	A	B	A	D
A	B	A	C	A	B	A	D						

Knuth-Morris-Pratt

- Process each character of S one at a time. We keep track of the longest prefix that currently matches.
- When mismatch occurs, fall back to shorter prefix.

Example

Find all occurrences of **ABACABAD** in **ABACABABACABAD**

A	B	A	C	A	B	A	B	A	C	A	B	A	D
A	B	A	C	A	B	A	D						

Knuth-Morris-Pratt

- Process each character of S one at a time. We keep track of the longest prefix that currently matches.
- When mismatch occurs, fall back to shorter prefix.

Example

Find all occurrences of **ABACABAD** in **ABACABABACABAD**

A	B	A	C	A	B	A	B	A	C	A	B	A	D
A	B	A	C	A	B	A	D						

Knuth-Morris-Pratt

- Process each character of S one at a time. We keep track of the longest prefix that currently matches.
- When mismatch occurs, fall back to shorter prefix.

Example

Find all occurrences of **ABACABAD** in **ABACABABACABAD**

A B A C A B A B A C A B A D
A B A C A B A D

Knuth-Morris-Pratt

- Process each character of S one at a time. We keep track of the longest prefix that currently matches.
- When mismatch occurs, fall back to shorter prefix.

Example

Find all occurrences of **ABACABAD** in **ABACABABACABAD**

A	B	A	C	A	B	A	B	A	C	A	B	A	D
A	B	A	C	A	B	A	D						

Knuth-Morris-Pratt

- Process each character of S one at a time. We keep track of the longest prefix that currently matches.
- When mismatch occurs, fall back to shorter prefix.

Example

Find all occurrences of **ABACABAD** in **ABACABABACABAD**

A	B	A	C	A	B	A	B	A	C	A	B	A	D
A	B	A	C	A	B	A	D						

Knuth-Morris-Pratt

- Process each character of S one at a time. We keep track of the longest prefix that currently matches.
- When mismatch occurs, fall back to shorter prefix.

Example

Find all occurrences of **ABACABAD** in **ABACABABACABAD**

A	B	A	C	A	B	A	B	A	C	A	B	A	D
A	B	A	C	A	B	A	D						

Knuth-Morris-Pratt

- Process each character of S one at a time. We keep track of the longest prefix that currently matches.
- When mismatch occurs, fall back to shorter prefix.

Example

Find all occurrences of **ABACABAD** in **ABACABABACABAD**

A B A C A B A B A C A B A D
A B A C A B A D

Knuth-Morris-Pratt

- Process each character of S one at a time. We keep track of the longest prefix that currently matches.
- When mismatch occurs, fall back to shorter prefix.

Example

Find all occurrences of **ABACABAD** in **ABACABABACABAD**

A	B	A	C	A	B	A	B	A	C	A	B	A	D
				A	B	A	C	A	B	A	D		

Knuth-Morris-Pratt

- Process each character of S one at a time. We keep track of the longest prefix that currently matches.
- When mismatch occurs, fall back to shorter prefix.

Example

Find all occurrences of **ABACABAD** in **ABACABABACABAD**

A	B	A	C	A	B	A	B	A	C	A	B	A	D
						A	B	A	C	A	B	A	D

Knuth-Morris-Pratt

- Process each character of S one at a time. We keep track of the longest prefix that currently matches.
- When mismatch occurs, fall back to shorter prefix.

Example

Find all occurrences of **ABACABAD** in **ABACABABACABAD**

A	B	A	C	A	B	A	B	A	C	A	B	A	D
						A	B	A	C	A	B	A	D

Knuth-Morris-Pratt

- Process each character of S one at a time. We keep track of the longest prefix that currently matches.
- When mismatch occurs, fall back to shorter prefix.

Example

Find all occurrences of **ABACABAD** in **ABACABABACABAD**

A	B	A	C	A	B	A	B	A	C	A	B	A	D
						A	B	A	C	A	B	A	D

Knuth-Morris-Pratt

- Process each character of S one at a time. We keep track of the longest prefix that currently matches.
- When mismatch occurs, fall back to shorter prefix.

Example

Find all occurrences of **ABACABAD** in **ABACABABACABAD**

A	B	A	C	A	B	A	B	A	C	A	B	A	D
						A	B	A	C	A	B	A	D

Knuth-Morris-Pratt

- Process each character of S one at a time. We keep track of the longest prefix that currently matches.
- When mismatch occurs, fall back to shorter prefix.

Example

Find all occurrences of **ABACABAD** in **ABACABABACABAD**

A	B	A	C	A	B	A	B	A	C	A	B	A	D
						A	B	A	C	A	B	A	D

Knuth-Morris-Pratt

- Process each character of S one at a time. We keep track of the longest prefix that currently matches.
- When mismatch occurs, fall back to shorter prefix.

Example

Find all occurrences of **ABACABAD** in **ABACABABACABAD**

A	B	A	C	A	B	A	B	A	C	A	B	A	D
						A	B	A	C	A	B	A	D

Knuth-Morris-Pratt

- Process each character of S one at a time. We keep track of the longest prefix that currently matches.
- When mismatch occurs, fall back to shorter prefix.

Example

Find all occurrences of **ABACABAD** in **ABACABABACABAD**

A	B	A	C	A	B	A	B	A	C	A	B	A	D
						A	B	A	C	A	B	A	D

Knuth-Morris-Pratt

- Process each character of S one at a time. We keep track of the longest prefix that currently matches.
- When mismatch occurs, fall back to shorter prefix.

Example

Find all occurrences of **ABACABAD** in **ABACABABACABAD**

A	B	A	C	A	B	A	B	A	C	A	B	A	D
						A	B	A	C	A	B	A	D

Finding the Overlap

- We do not wish to match any character of **S** more than once.
- We can pre-search **W** to determine all possible fallback positions which will prevent us from checking positions we know already.
- We require a table **T** where **T**[i] will give us the longest prefix of **W** which is also a suffix of **W**[:i].
- We can consider a DP approach to the problem.

Finding the Overlap

- We do not wish to match any character of **S** more than once.
- We can pre-search **W** to determine all possible fallback positions which will prevent us from checking positions we know already.
- We require a table **T** where **T**[i] will give us the longest prefix of **W** which is also a suffix of **W**[:i].
- We can consider a DP approach to the problem.

Finding the Overlap

- We do not wish to match any character of **S** more than once.
- We can pre-search **W** to determine all possible fallback positions which will prevent us from checking positions we know already.
- We require a table **T** where **T**[i] will give us the longest prefix of **W** which is also a suffix of **W**[:i].
- We can consider a DP approach to the problem.

Finding the Overlap

- We do not wish to match any character of **S** more than once.
- We can pre-search **W** to determine all possible fallback positions which will prevent us from checking positions we know already.
- We require a table **T** where **T**[i] will give us the longest prefix of **W** which is also a suffix of **W**[:i].
- We can consider a DP approach to the problem.

Finding the Overlap

- If **T**[i] = k and **W**[i] = **W**[k], then **T**[i+1] = k+1
- If **W**[i] $\neq$ **W**[k], then we fall back to the next valid prefix/suffix: k $\leftarrow$ **T**[k]
- If we can't fall back any further, we simply set the table function of that position to 0, and increment our position index.

Finding the Overlap

- If **T**[i] = k and **W**[i] = **W**[k], then **T**[i+1] = k+1
- If **W**[i] $\neq$ **W**[k], then we fall back to the next valid prefix/suffix: k $\leftarrow$ **T**[k]
- If we can't fall back any further, we simply set the table function of that position to 0, and increment our position index.

Finding the Overlap

- If **T**[i] = k and **W**[i] = **W**[k], then **T**[i+1] = k+1
- If **W**[i] $\neq$ **W**[k], then we fall back to the next valid prefix/suffix: k $\leftarrow$ **T**[k]
- If we can't fall back any further, we simply set the table function of that position to 0, and increment our position index.

Partial match table

Pseudocode:

```
def table(w):
    t[0] = -1,  t[1] = 0
    pos=2,  cnd=0
    while (pos<len(w)):
        if w[pos-1] == w[cnd]:
            cnd += 1,  t[pos] = cnd,  pos += 1
        elif (cnd > 0):
            cnd = t[cnd]
        else:
            t[pos] = 0,  pos += 1
    return t
```

Algorithm

- Once we have our partial match table, we can do the search.
- If **W**[i] = **S**[m+i], we increment i.
- Else, we simply shift m and i by the fall back value: **T**[i]
- If we can't fall back, we set i to 0 and increment m.

Algorithm

- Once we have our partial match table, we can do the search.
- If **W**[i] = **S**[m+i], we increment i.
- Else, we simply shift m and i by the fall back value: **T**[i]
- If we can't fall back, we set i to 0 and increment m.

Algorithm

- Once we have our partial match table, we can do the search.
- If **W**[i] = **S**[m+i], we increment i.
- Else, we simply shift m and i by the fall back value: **T**[i]
- If we can't fall back, we set i to 0 and increment m.

Algorithm

- Once we have our partial match table, we can do the search.
- If **W**[i] = **S**[m+i], we increment i.
- Else, we simply shift m and i by the fall back value: **T**[i]
- If we can't fall back, we set i to 0 and increment m.

Search Algorithm

Pseudocode:

```
def kmp(w, s):
    m = 0,  i = 0
    while(m+i < len(s)):
        if (w[i] == s[m+i]):
            if (i == len(w)-1):
                return m
            i += 1
        else:
            if (t[i] > -1):
                m = m+i-t[i],  i = t[i]
            else:
                i = 0,  m += 1
```

Efficiency

- For the partial match table, we note that both *pos* and *pos-cnd* are non-decreasing, and for every iteration, one of the two must strictly increase. Since both are bounded by $|W|$, it will take at most $2|W|$ steps.
- Similarly, one can prove that the search algorithm will take at most $2|S|$ steps.
- We therefore get a total linear runtime of $O(|S| + |W|)$, which is the best that can be obtained.

Efficiency

- For the partial match table, we note that both *pos* and *pos-cnd* are non-decreasing, and for every iteration, one of the two must strictly increase. Since both are bounded by $|W|$, it will take at most $2|W|$ steps.
- Similarly, one can prove that the search algorithm will take at most $2|S|$ steps.
- We therefore get a total linear runtime of $O(|S| + |W|)$, which is the best that can be obtained.

Efficiency

- For the partial match table, we note that both *pos* and *pos-cnd* are non-decreasing, and for every iteration, one of the two must strictly increase. Since both are bounded by $|W|$, it will take at most $2|W|$ steps.
- Similarly, one can prove that the search algorithm will take at most $2|S|$ steps.
- We therefore get a total linear runtime of O($|S| + |W|$), which is the best that can be obtained.