Advanced Algorithms & Data Structures

Department of CSE

ADVANCED ALGORITHMS AND DATA STRUCTURES
23CS03HF

Topic:

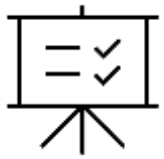
Knuth Morris Pratt Algorithm

Session - 30

AIM OF THE SESSION

To familiarize students with the concept of the Knuth Morris Pratt Algorithm.

INSTRUCTIONAL OBJECTIVES



This Session is designed to:

1. Demonstrate :- Knuth Morris Pratt Algorithm.
2. Describe :- Sequence of steps for Knuth Morris Pratt Algorithm

LEARNING OUTCOMES

At the end of this session, you should be able to:

1. Define :- Knuth Morris Pratt Algorithm.
2. Describe :- Sequence of steps for Knuth Morris Pratt Algorithm
3. Summarize:- Build a LPS(Longest Proper Prefix which is also a Suffix) array and then perform string matching.

Knuth Morris Pratt Algorithm

- Knuth Morris Pratt (KMP) is an algorithm, which checks the characters from left to right.
- When a pattern has a sub-pattern which appears more than one in the sub-pattern,
- it uses that property to improve the time complexity, also for in the worst case.
- The time complexity of KMP is O(n) which is achieved by avoiding comparison with an element of 'S' that have previously been involved in comparison with some element of the pattern 'p' to be matched. i.e., backtracking on the string 'S' never occurs

Components of KMP Algorithm

- **PREFIX Function**
  - **Prefix[i]** is the length of the longest proper prefix of the substring **s[0…i]** which is also a suffix of this substring.
  - By Definition,
    - **Prefix[0]=0**
    - **Prefix[0…k-1] = s[i-(k-1)….i]**

Why???

- Pattern: a b c d a b c
- 1 2 3 4 5 6 7

- Prefix: a, ab, abc, abcd

- Sufix: c, bc, abc, dabc

- In this way, KMP is reducing the Number of Comparisons.
- Like the beginning part of the pattern is again appeared again in the Pattern or not. That's what we need to observe.

KMP Algorithm

- In KMP algorithm, we have “π” Table which is also called as “Longest Prefix which is same as some of suffix” which is referred as “lps[]”.
- Eg: 1 2 3 4 5 6 7 8 9
  - P1: a b c d a b e a b f
    0 0 0 0 1 2 0 1 2 0
  - P2: a b c d e a b f a b c
    0 0 0 0 0 1 2 0 1 2 3
  - P3: a a b c a d a a b e
    0 1 0 0 1 0 1 2 3 0
  - P4: a a a a b a a c d
    0 1 2 3 0 1 2 0 0

This is the base of KMP algorithm to prepare the table. Once the Table is prepared, then Algorithm is very simple.

How KMP Algorithm Works

- String: a b a b c a b c a b a b a b d

  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

- Pattern: a b a b d
- First, we will prepare the “π” Table

0 1 2 3 4 5

a	b	a	b	d
0	0	1	2	0

How KMP Algorithm works

i

- String: a b a b c a b c a b a b a b d

'i' - S [index] ->1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

j

'j' – P [index] -> 0 1 2 3 4 5

Pattern ->

Pat-Alpha	a	b	a	b	d
Pat-Num	0	0	1	2	0

Now, Compare, String[i] is matched with the Pattern[j+1].

If Matched, Increment both 'j' & 'i' by 1.

Else if not matched, move the 'j' to the index numbered by P [j]..

How KMP Algorithm works

i

- String: a b a b c a b c a b a b a b d

'i' - S [index] -> 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

j

'j' – P [index] -> 0 1 2 3 4 5

Pattern -> Pat-Alpha	a	b	a	b	d
Pat-Num	0	0	1	2	0

Now, Compare, String[i] is matched with the Pattern[j+1].

If Matched, Increment both 'j' & 'i' by 1.

Else if not matched, move the 'j' to the index numbered by P [j]..

How KMP Algorithm works

- String: a b a b c a b c a b a b a b d

i

'i' - S [index] ->1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

j

'j' – P [index] -> 0 1 2 3 4 5

Pattern ->						
	Pat-Alpha	a	b	a	b	d
	Pat-Num	0	0	1	2	0

Now, Compare, String[i] is matched with the Pattern[j+1].

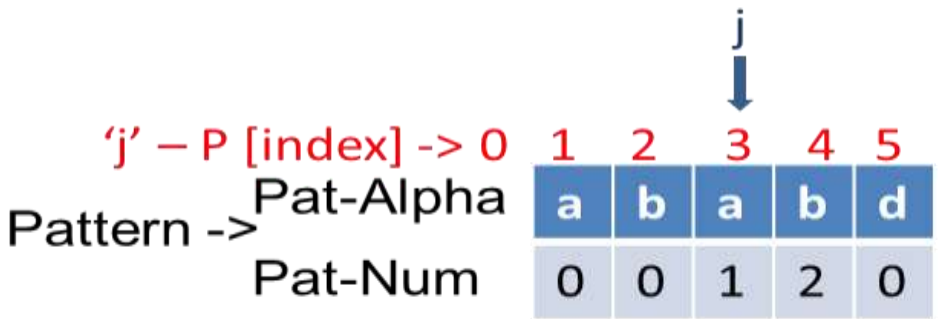
If Matched, Increment both 'j' & 'i' by 1.

Else if not matched, move the 'j' to the index numbered by P [j]..

How KMP Algorithm works

- String: a b a b c a b c a b a b a b d

'i' - S [index] -> 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15



Now, Compare, String[i] is matched with the Pattern[j+1].

If Matched, Increment both 'j' & 'i' by 1.

Else if not matched, move the 'j' to the index numbered by P [j]..

How KMP Algorithm works

- String: a b a b c a b c a b a b a b d

i

‘i’ - S [index] ->1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

j

‘j’ – P [index] -> 0 1 2 3 4 5

Pattern ->						
	Pat-Alpha	a	b	a	b	d
	Pat-Num	0	0	1	2	0

Now, Compare, String[i] is matched with the Pattern[j+1].

If Matched, Increment both ‘j’ & ‘i’ by 1.

Else if not matched, move the ‘j’ to the index numbered by P [j]..

How KMP Algorithm works

i

- String: a b a b c a b c a b a b a b d

'i' - S [index] ->1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

j

'j' – P [index] -> 0 1 2 3 4 5

Pattern ->	Pat-Alpha	a	b	a	b	d
	Pat-Num	0	0	1	2	0

Now, Compare, String[i] is matched with the Pattern[j+1].

If Matched, Increment both 'j' & 'i' by 1.

Else if not matched, move the 'j' to the index numbered by P [j]..

And continue the same process.

How KMP Algorithm works

- String: a b a b c a b c a b a b a b d

'i' - S [index] ->1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

'j' – P [index] -> 0 1 2 3 4 5

Pattern ->						
	Pat-Alpha	a	b	a	b	d
	Pat-Num	0	0	1	2	0

If 'j' reaches the end of pattern & 'i' reaches the n+1, then we can say that the given pattern is present in the given string.

The Basic Idea in KMP is If a substring of a pattern is again appeared in the pattern, then no need to compare again and again. In that way, we can reduce the number of comparisons.

Here, we are not backtracking the 'i', only 'j' is backtracking.

The Time Complexity of KMP is

m + n

Table + Search

O(m+n) – This is the one of the Fastest algorithm in Pattern Matching Algorithm.

SUMMARY

- A string matching/searching algorithm that is used to search for a pattern in a given string. Conceptually, there are two steps to implement KMP algorithm, build a LPS(Longest Proper Prefix which is also a Suffix) array and then perform string matching.

SELF-ASSESSMENT QUESTIONS

In the KMP algorithm, if the pattern and text characters mismatch after some matches, what does the algorithm do next?

(a) Moves the pattern one position to the right

(b) Uses the π array to determine the next position to continue matching

(c) Restarts the search from the beginning of the text

(d) Uses a different pattern for matching

What is the main advantage of the KMP algorithm over the naive string matching algorithm?

(a) Lower space complexity

(b) Simplicity in implementation

(c) Ability to handle multiple patterns simultaneously

(d) Avoidance of unnecessary comparisons

TERMINAL QUESTIONS

1. How does the KMP algorithm handle mismatches during the search phase? What role does the π array play in this process?
2. In the context of the KMP algorithm, what is a "proper prefix"? How is it different from a regular prefix?

REFERENCES FOR FURTHER LEARNING OF THE SESSION

Reference Books :

1. Introduction to Algorithms, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein., 3rd, 2009, The MIT Press.
2 Algorithm Design Manual, Steven S. Skiena., 2nd, 2008, Springer.
3 Data Structures and Algorithms in Python, Michael T. Goodrich, Roberto Tamassia, and Michael H. Goldwasser., 2nd, 2013, Wiley.
4 The Art of Computer Programming, Donald E. Knuth, 3rd, 1997, Addison-Wesley Professiona.

MOOCS :

1. https://www.coursera.org/specializations/algorithms?=
2.https://www.coursera.org/learn/dynamic-programming-greedy-algorithms#modules

THANK YOU