Abstract

Karp-Rabin Algorithm is an average case analysis of string matching algorithm. This algorithm uses hash technique for string searching. In other words we can say that Karp-Rabin uses finger printing technique. The initial hashes are called Finger Prints.

This algorithm updates these finger prints in constant time. Hashing technique adds ASCII values for each letter and mod it by some prime number. Good hash function need to look at every character in the string. After calculating the hash function, If the values are same then we need to do normal string comparison. If both the strings are same then the search completed, Else we have to continue. Now shift over a character in the string and calculate the hash value of it, Continue the process as above until the actual string is found or we reach to the end of text string.
1 Introduction

Karp-Rabin Algorithm is one of the most effective string matching algorithms. To find a numeric pattern ‘p’ from a given text ‘q’. It firstly divides the pattern with a predefined prime number ‘r’ to calculate the remainder of pattern ‘p’. Then it takes the first ‘m’ characters here ‘m’ is the length of pattern ‘p’ from text ‘q’ at first shift ‘S’ to compute reminder of ‘m’ characters from text ‘q’. If the remainder of the Pattern and the remainder of the text ‘q’ are equal only when we compare the text with the pattern otherwise there is no need for comparison. The reason is that if the remainders of the two numbers are not equal then these numbers cannot be equal in any case. We will repeat the process for next set of characters from text for all possible shifts.

2 History

The Karp-Rabin algorithm is a string searching algorithm created by Miachel O.Rabin and Richard M.Karp in 1987 that uses hashing to find any one of a set of pattern strings in a text. For text of length ‘n’ and ‘p’ patterns of combined length ‘m’, Its average and best case running times is $O(n + m)$ in space $O(p)$, but its worst case is $O(nm)$.

3 Karp-Rabin Algorithm

3.1 Statement of the problem

The problem with this algorithm is to perform a hash function. In this algorithm every substring are closely related to next substring, The way of obtaining next substring is simply by means of remove the first character in substring and new character will be added to the other end of the substring. Now, we have to calculate hash for the substring of input and instead of comparing two string values. We have to compare integer values after calculating hash function.

3.2 Complexity

For the length n and patterns ‘p’ of combined length ‘m’
• It uses hashing function
• Preprocessing $O(m)$ time complexity with constant space. The preprocessing phase of the Karp-Rabin algorithm consists in computing $\text{hash}(q)$.
• Searching in $O(nm)$ time. During searching phase, it is enough to compare $\text{hash}(q)$ with $\text{SrchStr}$. If an equality is found, it is still necessary to check character by character.
• Expected run time is $O(n + m)$ that is by number of text character comparisons.

4 Comparisons

4.1 Knuth-Morris-Pratt

4.1.1 Statement of the problem
Given a string ‘$S$’, The problem of string matching deals with finding whether a pattern ‘$p$’ occur in ‘$S$’ and if ‘$p$’ does occur then return the position in ‘$S$’.

4.1.2 Complexity
• It performs comparisons from left to right.
• Preprocessing $O(m)$ time complexity as the length of the longest border of the input string followed by a character.
• Searching phase in $O(n + m)$. The Knuth-Morris-Pratt algorithm performs at most $2n - 1$ text character comparisons during the searching phase and the delay (maximal number of comparisons for a single text character). It is independent from the alphabet size.

4.2 Boyer-Moore

4.2.1 Statement of the problem
The main problem of the Boyer-Moore algorithm is the preprocessing time and the space required, Which depends on the alphabet size or the pattern size.
4.2.2 Complexity

- Preprocessing \( O(m + n) \) time complexity with constant space. The searching phase time complexity is quadratic but at most \( 3n \) text character comparisons are performed when searching for a non periodic pattern. On large alphabets (relatively to the length of the pattern) the algorithm is extremely fast.

- Searching in \( O(nm) \) time. It is the worst case.

- \( 3n \) text character comparisons in the worst case when searching for a non-periodic pattern.

- The best case is \( O(n/m) \), which is the absolute minimum for any string-matching algorithm in the model where the pattern only is preprocessed.

5 pseudocode

\[
p ← inputstring
q ← searchstring
n ← length(p)
m ← length(q)
hSrchStr := hash(q)
for i = 0 \rightarrow n - m do
    hStr ← hash(p[i \rightarrow m])
    if p[i \rightarrow m] = q then
        return "FOUND"
    else
        return "NOTFOUND"
end if
end for
\]

Calculating the hash

\[
Str ← inputstring
sum ← 0
y ← length(Str)
x[] ← Str
for i = 0 \rightarrow y do
    sum = sum + str[i]
end for
return sum \% 3
\]
6 How to calculate hash value of the given string

The hash value of aba is 1.

7 Example of Karb-Rabin Algorithm

Input String

K A R P R A B I N

Search String

R A B

1. Calculate hash of SearchString.

hash of SearchString is 0.

2. Find the SearchString from the input String.

Step1: Take First three letters from input String

K A R P R A B I N

Figure 2: Considering first 3 characters

Calculate the hash of selected String

• The hash of SelectedString is 0.
• Here hash of SearchString matches with the SelectedString.

• Now we will check the SearchString matches to the Selected String. When SearchString not matches with SelectedString.

Step2: Shift the Character to right

Figure 3: Shifting the character

Calculate the hash of SelectedString

• The hash of SelectedString is 2.

• Here hash of SearchString not matches with the SelectedString.

Step3: Shift the Character to right

Figure 4: Shifting the character

Calculate the hash of SelectedString

• The hash of SelectedString is 1.

• Here hash of SearchString not matches with the SelectedString.

Step4: Shift the Character to right

Figure 5: Shifting the character
Calculate the hash of SelectedString

- The hash of SelectedString is 2.
- Here hash of SearchString not matches with the SelectedString.

Step 5: Shift the character to right

![Shifting the character](image)

Figure 6: Shifting the character

Calculate the hash of Selected String

- The hash of SelectedString is 0.
- Here hash of SearchString matches with the SelectedString.
- Now we will check the SearchString matches to the SelectedString. SearchingString matches with SelectedString.

Step 6: Shift the character to right

![Shifting the character](image)

Figure 7: Shifting the character

Calculate the hash of SelectedString

- The hash of SelectedString is 0.
- Here hash of SearchString matches with the SelectedString.
- Now we will check the SearchString matches to the SelectedString. SearchString not matches with SelectedString.

Step 7: Shift the character to right.

Calculate the hash of SelectedString

- The hash of SelectedString is 0.
Here hash of SearchString not matches with the SelectedString.
3. Result is 4,3.

8 Features

• It is independent of length of pattern
• The algorithms require a constant number of storage locations, and essentially run in real time. They are conceptually simple and easy to implement. The method readily generalizes to higher-dimensional pattern-matching problems.
• Karp-Rabin algorithm is used for multiple pattern search
• Application of Karp-Rabin is detecting Plagiarism

References