Ishita Mehta
16csu153
CS-5
CSE-C

Rabin-Karp Algorithm

21st November 2017

INTRODUCTION

It is a commonly used pattern search algorithm. It is actually a "Naive Pattern Search" algorithm enhanced by the use of a hash function(A hash function is any function that can be used to map a data set of an arbitrary size to a data set of a fixed size[1]). Given a text and a pattern to match, we have to print all the occurrences of the given pattern in the text. By finding the hash value of the pattern of say length *l*, we then slide over the text to and compute hash value for substrings of length *l.* If the hash value of the pattern and the substring matches we then conduct a Naive Pattern Search by comparing each character of the pattern with each pattern of the substring to avoid any collisions in the hash function. Formula[2] for computing:

*hash(txt[s+1 .. s+l]) = d (hash(txt[s .. s+l-1]) – txt[s]*h) + txt[s + l]) mod q*

hash(txt[s .. s+l-1]) : Hash value at shift *s*.

hash(txt[s+1 .. s+l]) : Hash value at next shift (or shift *s*+1)

d: Number of characters in the alphabet

q: A prime number

h: d^(m-1)

HOW TO SOLVE

Assume length of text string to be *n*, length of pattern string to be *l*, given string of text to be *txt* and given string of pattern to be *p*.

[1] Definition of Hash Function Source: www.hackerearth.com
[2] Formula for Hash Function cited from www.geeksforgeeks.com

1. Compute Hash Functions for the given text and given pattern, h_t and h_p respectively.
2. For an int i←1 to *n- l*
3. *If both the hash values match i.e. if* $h_t == h_p$
4. We match *txt[i… i+l] with pattern p, if they match we return 1.*
5. *Else* we slide our *txt* to the next substring and calculate the hash function for this new substring using the formula

   $h_t = (d(h_t - t[i + 1] .d^{l-1}) + t[m + i + 1]) \bmod q$

6. *End*

IMPLEMENTATION

```
#include<stdio.h>

#include<string.h>

#define num 256

void rabinSearch(char pattern[], char text[], int prime);

void main()

{

   char text[] = "This is my Data Structures and Algorithms Project";

   char pattern[] = "Algorithms";

   int prime = 101; // A prime number

   rabinSearch(pattern, text, prime);

}

void rabinSearch(char pattern[], char text[], int prime)

{

   int s1 = strlen(pattern);

   int s2 = strlen(text);

   int i, j;
```

```
int hashP = 0; // hash value for pattern
int hashT = 0; // hash value for text
int q = 1;

for (i = 0; i < s1-1; i++)
    q = (q*num)%prime;

for (i = 0; i < s1; i++)
{
    hashP = (num*hashP + pattern[i])%prime;
    hashT = (num*hashT + text[i])%prime;
}

for (i = 0; i <= s2 - s1; i++)
{      if ( hashP == hashT )
   {
       for (j = 0; j < s1; j++)
       {
           if (text[i+j] != pattern[j])
               break;
       }

       if (j == s1)
           printf("Pattern found at index %d \n", i);
```

```
        }


        if ( i < s2-s1)
        {
            hashT = (num*(hashT - text[i]*q) + text[i+s1])%prime;
            if (hashT < 0)
            hashT = (hashT + prime);
        }
    }
}
```

OUTPUT

Code compiled in GCC version 7.2.0