

# Advanced Computer Architecture - Project Report

## Substring Matching with application to Genomics

Diego Mastella (matricola) - Matteo Ragni (520038)  
University of Pavia - Master's Degree in Computer Engineering

### 1 Serial algorithm analysis

**The Algorithm** The substring search algorithm implemented in this project is the Rabin-Karp fingerprint search. It is based on hashing: we compute a hash function for the pattern and then look for a match by using the same hash function for each possible M-character substring of the text. If we find a text substring with the same hash value as the pattern, we can check for a match. Rabin and Karp showed that it is easy to compute hash functions for M-character substrings in constant time (after some preprocessing), which leads to a *linear-time* substring search  $O(M+N)$  in practical situations.

		pat.charAt(j)																							
j		0	1	2	3	4																			
		2	6	5	3	5	% 997 = 613																		
		txt.charAt(i)																							
i		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15								
		3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3								
0		3	1	4	1	5	% 997 = 508																		
1			1	4	1	5	9	% 997 = 201																	
2				4	1	5	9	2	% 997 = 715																
3					1	5	9	2	6	% 997 = 971															
4						5	9	2	6	5	% 997 = 442														
5							9	2	6	5	3	% 997 = 929													
6	← return i = 6							2	6	5	3	5	% 997 = 613												match

Figure 1: **Example** - Rabin-Karp substring search

**Key idea** The Rabin-Karp method is based on efficiently computing the hash function for position  $i+1$  in the text, given its value for position  $i$ . It follows directly from a simple mathematical formulation. Using the notation  $t_i$  for `txt.charAt(i)`, the number corresponding to the M-character substring of `txt` that starts at position  $i$  is

$$x_i = t_i R^{M-1} + t_{i+1} R^{M-2} + \dots + t_{i+M-1} R^0 \quad (1)$$

and we can assume that we know the value of  $h(x_i) = x_i \% Q$ . Shifting one position right in the text corresponds to replacing  $x_i$  by

$$x_{i+1} = (x_i - t_i R^{M-1}) R + t_{i+M} \quad (2)$$

We subtract off the leading digit, multiply by  $R$ , then add the trailing digit. The result is that we can effectively move right one position in the text in constant time, whether  $M$  is 5 or 100 or 1,000.

**Monte Carlo correctness** When we find a match, we have to ensure that it is a true match, not just a hash collision. We can make the hash table “size”  $Q$  as large as we wish using a long value greater than  $10^{20}$ , making the probability that a random key hashes to the same value as our pattern less than  $10^{-20}$ , an exceedingly small value. This algorithm is an example of a Monte Carlo algorithm that has a guaranteed completion time but fails to output a correct answer with a small probability.

Parameter	Meaning
txt	Text string
pat	Pattern string
$N$	Text length
$M$	Pattern length
$R$	Base (26)
$h(x)$	Hash function
$Q$	Large prime number

Table 1: Cited parameters

	pat.charAt(j)				
i	0	1	2	3	4
	2	6	5	3	5
0	2	% 997 = 2			
1	2	6	% 997 = (2*10 + 6) % 997 = 26		
2	2	6	5	% 997 = (26*10 + 5) % 997 = 265	
3	2	6	5	3	% 997 = (265*10 + 3) % 997 = 659
4	2	6	5	3	5 % 997 = (659*10 + 5) % 997 = 613

Figure 2: **Example** - Computing hash value for the pattern

## 2 A-priori study of parallelism

## 3 MPI parallel implementation

## 4 Performance and scalability analysis

## 5 Individual contribution

## References