UNIVERSITÀ DI PAVIA

Advanced Computer Architecture - Project Report

Substring Matching with apllication to Genomics

Diego Mastella (525163) - 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.

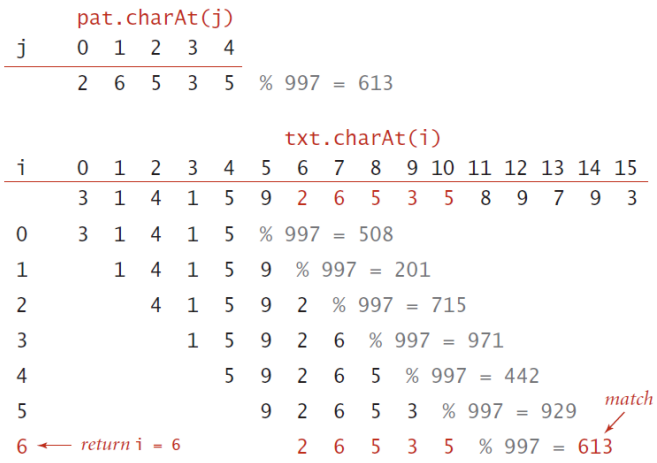

Figure 1: **Example** - Rabin-Karp substring search

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

$$x_i = t_i R^{M-1} + t_{i+1} R^{M-2} + \cdots + t_{i+M-1} R^0 \tag{1}$$

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

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

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

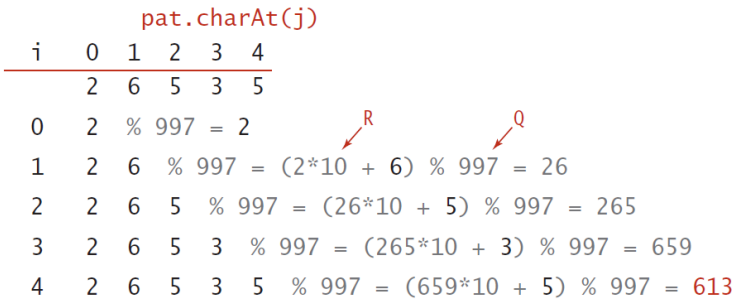

Figure 2: **Example** - Computing hash value for the pattern

2 A-priori study of parallelism

The approach we choose to design this algorithm in parallel mode is the following: divide the `text` string between multiple cores and allow each individual core to search for `pat` string in allocated part of `text` string. Once all calculations are done, we combine their result to get final count of that pattern. Another approach could be to assign individual core to individual pattern (in case of multi-pattern search) and then search for that single pattern in the `text` string. Second approach does not use efficiently *MPI Functions* because it does not require communication between cores, as each individual core would count their own pattern in the given main string.

3 MPI parallel implementation

4 Performance and scalibility analysis

5 Individual contribution

References