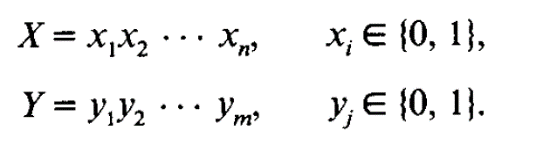
|  |
| --- |
| *\\samba1\vpandey\dcengr\Desktop\School_of_Engineering_Red_Logo.jpgCOEN 379: Advanced DesIGN & ANALYSIS OF ALGORITHM* |
| *Project* |
| *Rabin \_ Karp Fingerprint: A randomized approach to Pattern Matching problem* |
|  |
| ***Aaraadhya Nara, Himanshu Gupta & Varun Pandey*** |
| ***5/10/2015*** |

|  |
| --- |
| ***The project documentation contains following sub-sections: -*** |
| 1. C:\Users\Varun\Downloads\qr_code(1)\qr_code_without_logo.jpg*Universe outside Rabin Karp* 2. *Details of Rabin-Karp Algorithms* 3. *Mathematical Analysis* 4. *References* |

**Some Important Notations**



n <= m



**Overview: -**

We started with the analysis of other string matching techniques. String matching can be done in linear time by many algorithms. So what’s great about Rabin Karp algorithm? A beautiful analysis about Rabin Karp algorithm (aka RK) is that we convert the pattern matching problem to an arithmetic problem. This helps us to use the full capacity of processing unit and ALU operations. Use of limited hardware registers is an important benefits of RK.

We went on to read the actual RK paper and studied three different algorithm proposed by RK to have a heuristic/definitive match. We went on to analyze the algorithms to find the probability of false hit. We also implemented the algorithm. As a novelty, we implemented a parallel algorithm that uses RK (implementation in CUDA).

**Universe Outside Rabin Karp Algorithm: -**

Three Algorithms: -

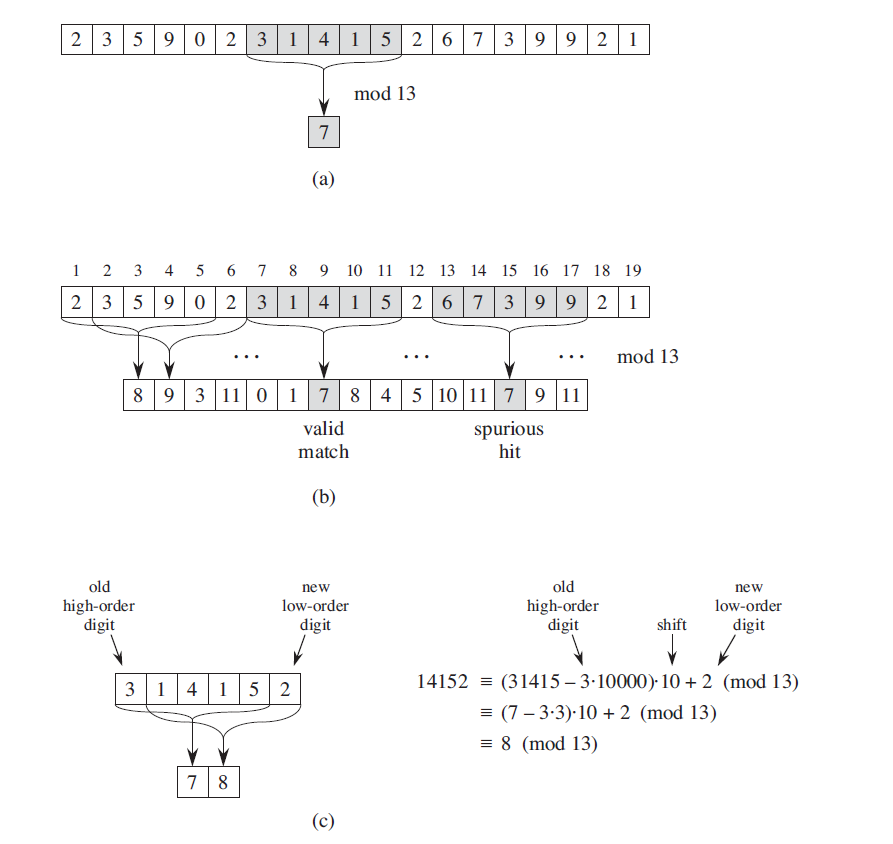
1. Naïve Algorithm
2. KMP
3. Boyers et. al.

* Comparison based deterministic algorithms
* Requires preprocessing except Naive Algorithm
* Stores portions of text Y in main memory
* Offline Algorithms
* Require O(n) registers to store table of pointers
* Same theoretical time bounds as the previous algorithms, but fewer number of registers
* Simple and conceptually easy to program
* Better than previous algorithms for long patterns.
* Applicable to patterns involving multi-dimensional rectangular array of symbols or even irregularly shaped arrangements of symbol
* Faster because randomized algorithm

**Rabin Karp Algorithm: -**

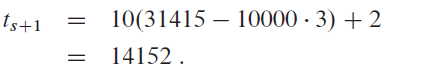
* Convert the String Comparison to an Arithmetic problem.
* Convert the pattern(X) of length n and Text of length(y) into Binary strings.
* Calculate the integer value and then compare the integers.

Stream of Text



31415

Pattern to search



|  |  |  |
| --- | --- | --- |
| Algorithm 1 | Algorithm 2 | Algorithm 3 |
| * Online * Uses k Randomly generated primes at a time | * Not Online * Not Requires one to one pattern checking on HIT * Considers only 1 prime | * Not Online * Not Requires one to one pattern checking on HIT * Considers a new prime every time we get a bad seed |

**Mathematical Analysis**

Expected Number of Trials to get the prime is ln M. The time to perform each primality test is O ((log M) 2) if we use Solovay and Strassen.

**Lemma 1:**

Proof: - Rosser and Schoenfeld in their paper stated that the

**Corollary 1:**

In short, we are saying here that if we can upper bound a power of 2 say u, then the number can have at most u - 1 prime divisors. We will use this later to proofs by counting the number of possible prime divisors

**Central Theorem:**

For a **fixed**  a single false can occur if –

This also means that,

**p |**

Literally translated to “p completely divides the quantity ”

So, for 1 run of algorithm, for a single false match,

For each r, assuming that we are considering only binary alphabets. Max difference between the

Cosnsidering t shifts and we have maximum difference possible on each iteration,

By corollary 1, this product has < and it is for these primes will the false hit is expected. Total prime to choose from is at most. Therefore, the probability that false match will be –

This proves for how many primes out of M, a false match could occur. It does NOT give any idea about the entire run of the algorithm

QED

**Extension of Central Theorem**

If Algorithm 1 is executed, the

Why? We are picking k random independent prime, so probability of false match is equal to

… k times

YET! This wouldn’t provide the tight bound. Why? Because here we are considering that all of the primes resulted in false match. This may not happen – only 1 position out of t choices may have a false match. So probability that one of the hash is a false hit is

,

**So, # of choice for prime dividing this value at every position of r is**

Therefore, the probability that one of the primes result in false match at every for one run of the algorithm is,

…t times

**What is a good value of M?**

1. Large enough
2. Must support register operation
3. Good balance
4. Must support the Ram Model

How about if we take

With choice of largest number, we can have overflow operation at which requires bits. This is in order of number of bits require to

Applying Lemma 2,

This is good because it gives us a good probability to work with, depending only upon the available number of shifts.

Example: - Consider n = 250, t = 4000, M = nt2 = 4x109 < 232. Then, for any p <= M, the bits required to store p is less than 32. So it can be stored in a register. Applying above formula, the probability of false match < 1000-1. For algorithm 1, if k = 4, Pr ≤ 2 x 10-22.

**Storage Requirement**

p is drawn from M ≤ nt2. So will require . Following are the operations needed for RK.

* 4 Fetch
* 3 Add
* 3 Compare
* 2 Subtraction
* 1 store

Hash computation is in constant time from Hr-1 to Hr.

Algorithm 1 requires (4k + 2) Registers

Algorithm 2, 3 requires 6 Registers

**Expected Time Spend in detecting the False Match:**

Algorithm 1 is real-time. So the complexity is O(m). Algorithm 2 & 3 requires confirmation and thus, the complexity is O(m+n).

So the expected time spend on detecting a false match is

Thus the complexity of Rabi Karp Algorithm is *O(m+n)*

**Novelty**

We implemented RK in three algorithms of Rabin Karp algorithm: -

1. A simple algorithm without caring about the overflow of fingerprint.
2. With mod prime restriction to fingerprint.
3. A parallel implementation of RK that essentially uses overlapping chunks of data writeen in CUDA. This algorithm is shelved currently as I am attempting to implement online version of the parallel algorithm.

*Pattern size n*

**Fig: -** Overlapping chunks

**References:**

1. Karp, Richard M., and Michael O. Rabin. "Efficient randomized pattern-matching algorithms." *IBM Journal of Research and Development* 31.2 (1987): 249-260.
2. Gusfield, Dan. *Algorithms on strings, trees and sequences: computer science and computational biology*. Cambridge university press, 1997.
3. Thomas H. Cormen et al. *Introduction to Algorithms*. 3nd edition. McGraw-Hill Higher Education, 2001. ISBN: 0070131511
4. Gonnet, Gaston H., and Ricardo A. Baeza-Yates. "An analysis of the Karp-Rabin string matching algorithm." *Information Processing Letters* 34.5 (1990): 271-274.
5. Barkley Rosser, J., and Lowell Schoenfeld. "Approximate formulas for some functions of prime numbers." *Illinois Journal of Mathematics* 6 (1962): 6z-9z.
6. Góngora-Blandón, Mario, and Miguel Vargas-Lombardo. "State of the art for string analysis and pattern search using cpu and gpu based programming." *Journal of Information Security* 3.04 (2012): 314.
7. Aronovich, Lior, et al. "The design of a similarity based deduplication system." *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*. ACM, 2009.
8. Some discrete youtube videos and articles.