Dictionary Matching with Fingerprints

Student: Dominic Moylett, Supervisor: Dr. Raphael Clifford and Dr. Benjamin Sach, Project Type: research

University of Bristol, Department of Computer Science

Introduction

'Big data' is a common term thrown about nowadays. As computers have become more and more powerful, the amount of information we want to process has also grown in size. As a result, time and space efficiency is a significant problem.

A common method for reducing space is the streaming model, where parts of the input come in at a time instead of the whole input. This is a good choice for pattern matching, where the text can come in one character at a time.

This project specifically looks at the area of dictionary matching, where you are trying to match one text to many patterns. The Algorithms team have devised a method of solving this problem in $O(\log m)$ time per character and $O(k \log m)$ space, based on Porat and Porat's solution to exact pattern matching in $O(\log m)$ time per character and $O(\log m)$ space. My work is on implementing this algorithm and compare it to other solutions for dictionary matching to see how it performs in practice.

Dictionary Matching Formally

We have a text T of n characters and a set P of k patterns $p_1, \ldots, p_k$ with lengths $m_1, \ldots, m_k$. For each index of the text, we output an occurance at j if $\exists i \in \{1, \ldots, k\}$ such that $t_{j-m_i}, \ldots, t_j = p_i$.

The typical solution to dictionary matching in the streaming model is the Aho-Corasick algorithm, which solves the problem based on a generalisation of Knuth-Morris-Pratt. It takes $O(1)$ time per character and $O(\sum_{i=1}^{k} m_i)$ space.

Pattern Matching in Less Space than the Pattern?

Sublinear space can be achieved by using a fingerprint function developed by Karp and Rabin for a string of characters $t_1, \ldots, t_n$, a prime number p and a randomly selected r:

$$\Phi_{p,r}(t_1, \ldots, t_n) = \sum_{i=1}^{n} r^i t_i (\mathrm{mod} p)$$

These fingerprints compress the amount of space required to store text, and can be modified to match changes in the underlying strings.

Current Progress

- The Aho-Corasick algorithm has been implemented.
- This new algorithm has been implemented for:
  - Patterns whose lengths are a power of 2
  - and patterns which are shorter than k
- Current work is focusing on long patterns which are repetitive.

Future Work

- Finish the work on long patterns which are repetitive.
- Implement the algorithm for long patterns.
- Compare the performance of the new algorithm to Aho-Corasick.
- Implement optimisations to the algorithm.