Available online at www.sciencedirect.com
ScienceDirect
Discrete Applied Mathematics 155 (2007) 386–393
DISCRETE APPLIED MATHEMATICS
www.elsevier.com/locate/dam

Decompositions of graphs of functions and fast iterations of lookup tables☆

Boaz Tsaban

Department of Mathematics, The Weizmann Institute of Science, Rehovot 76100, Israel

Received 9 October 2005; received in revised form 27 June 2006; accepted 29 June 2006
Available online 17 August 2006

Abstract

We show that every function f implemented as a lookup table can be implemented such that the computational complexity of evaluating $f^m(x)$ is small, independently of m and x. The implementation only increases the storage space by a small *constant* factor.

Keywords: Fast forward functions; Fast forward permutations; Cycle decomposition; Cycle structure

1. Introduction and motivation

According to Naor and Reingold [2], a function $f : \{0, \dots, N-1\} \to \{0, \dots, N-1\}$ is *fast forward* if for each natural number m which is polynomial in N, and each $x = 0, \dots, N-1$, the computational complexity of evaluating $f^m(x)$—the mth iterate of f at x—is small (polynomial in $\log N$). This is useful in simulations and cryptographic applications, and for the study of dynamic-theoretic properties of the function f.

Originally this notion was studied in the context of pseudorandomness, where N is very large—see [2,3,1]. Here we consider the remainder of the scale, where N is not too large, so that the function $f : \{0, \dots, N-1\} \to \{0, \dots, N-1\}$ is or can be implemented by a lookup table of size N. Implementations as lookup tables are standard for several reasons, e.g., in the case where the evaluation $f(x)$ is required to be efficient, or in the case that f is a random function, so that f has no shorter definition than just specifying its values for all possible inputs. We describe a simple way to implement a given function f such that it becomes fast forward. The implementation only increases the storage space by a small constant factor.

The case that f is a permutation is of special importance and is easier to treat. This is done in Section 2. In Section 3 we treat the general case.

2. Making a permutation fast forward

We recall two definitions from [3].

☆ Supported by the Koshland Fellowship.
E-mail address: boaz.tsaban@weizmann.ac.il
URL: http://www.cs.biu.ac.il/∼ tsaban.

0166-218X/$ - see front matter
doi:10.1016/j.dam.2006.06.006

Definition 1. Assume that f is a permutation on $\{0, \ldots, N-1\}$. The *ordered cycle decomposition* of f is the sequence $(C_0, \ldots, C_{\ell-1})$ consisting of all (distinct) cycles of f, such that for each $i, j \in \{0, \ldots, \ell-1\}$ with $i < j$, $\min C_i < \min C_j$. The *ordered cycle structure* of f is the sequence $(|C_0|, \ldots, |C_{\ell-1}|)$.

The ordered cycle decomposition of f can be computed in time N: find C_0, the cycle of 0. Then find C_1, the cycle of the first element not in C_0, etc. In particular, the ordered cycle *structure* of f can be computed in time N.

Definition 2. Assume that $(m_0, m_1, \ldots, m_{\ell-1})$ is the ordered cycle structure of a permutation f on $\{0, \ldots, N-1\}$. For each $i = 0, \ldots, \ell-1$, let $s_i = m_0 + \cdots + m_i$. The *fast forward permutation coded by* $(m_0, m_1, \ldots, m_{\ell-1})$ is the permutation π on $\{0, \ldots, N-1\}$ such that for each $x \in \{0, \ldots, N-1\}$,

$$\pi(x) = s_i + (x - s_i + 1 \bmod m_{i+1}) \quad \text{where } s_i \leqslant x < s_{i+1}.$$

In other words, π is the permutation whose ordered cycle decomposition is

$$\pi = (\underbrace{0 \ldots s_0 - 1}_{m_0})(\underbrace{s_0 \ldots s_1 - 1}_{m_1})(\underbrace{s_1 \ldots s_2 - 1}_{m_2}) \cdots (\underbrace{s_{\ell-2} \ldots N - 1}_{m_{\ell-1}}).$$

The assignment $x \mapsto i(x)$ such that $s_{i(x)} \leqslant x < s_{i(x)+1}$ can be implemented (in time N) as a lookup table of size N. As

$$\pi^m(x) = s_{i(x)} + (x - s_{i(x)} + m \bmod (s_{i(x)+1} - s_{i(x)})),$$

π is fast forward.

Coding 3. To code a given permutation f on $\{0, \ldots, N-1\}$ as a fast forward permutation, do the following.

(1) Compute the ordered cycle decomposition of f:

$$f = (\underbrace{b_0 \ldots b_{s_0-1}}_{m_0})(\underbrace{b_{s_0} \ldots b_{s_1-1}}_{m_1})(\underbrace{b_{s_1} \ldots b_{s_2-1}}_{m_2}) \cdots (\underbrace{b_{s_{\ell-2}} \ldots b_{N-1}}_{m_{\ell-1}}).$$

(2) Define a permutation σ on $\{0, \ldots, N-1\}$ by $\sigma(x) = b_x$ for each $x = 0, \ldots, N-1$.
(3) Store in memory the following tables: σ, σ^{-1}, the list $s_0, \ldots, s_{\ell-1}$ (where $s_k = m_0 + \cdots + m_k$ for each k), and the assignment $x \mapsto i(x)$.

Let π be the fast forward permutation coded by $(m_0, m_1, \ldots, m_{\ell-1})$. Then

$$f = \sigma \circ \pi \circ \sigma^{-1}.$$

For each m and x, $f^m(x)$ is equal to $\sigma(\pi^m(\sigma^{-1}(x)))$, which is computed by five invocations of the stored lookup tables and five elementary arithmetic operations (addition, subtraction, or modular reduction). We therefore have the following.

Theorem 4. *Every permutation f on $\{0, \ldots, N-1\}$ can be coded by four lookup tables of size N each, such that each evaluation $f^m(x)$ can be carried using five invocations of lookup tables and five elementary arithmetic operations, independently of the size of m.*

Remark 5.

(1) For random permutations, $\ell \approx \log N$ and therefore the total amount of memory is about $3N + \log N$.
(2) Instead of storing the assignment $x \mapsto i(x)$, we can compute it online. This is a search in an ordered list and takes $\log_2(\ell)$ in the worst case. For a typical permutation this is about $\log_2(\log(N))$ additional operations in the worst case (e.g., for $N = 2^{32}$, this is about four additional operations per evaluation). This reduces the memory to $2N + \log N$.