A trade-off for deterministic dictionaries

Rasmus Pagh
BRICS
Department of Computer Science
University of Aarhus

SWAT 2000, Bergen, July 5, 2000
Detennunistic dictionaries

Space constraint: \( \text{Use } O(n) \text{ words.} \)

- Free access to \( O(1) \) words computed at "compile time".
- \( m \{0, 1\} = \Omega \)
- Unit cost RAM with word size \( m \) and a standard instruction set.

Model of computation:

Complexity expressed in terms of \( n \):
- \( |S| = n \)
- Query for membership, and retrieval of any satellite information.
- \([ \) Deletion of an element.
- \( ] \) Insertion of an element, with satellite information.

Store a set \( \cap \subseteq S \) under the following operations:

The dictionary problem.
What can be done

Can be extended using Dietzel(binder) & Meyer & der Heide, '90.

Shown in the literature for $m = \log n$ (Willard, '00)

\[ \frac{m}{u(I_1)} - 1 = \log \frac{1}{1 - \text{constant}}. \]

What is a dictionary with

What randomization, very good expected bounds can be achieved.
Previous deterministic dictionaries

(update time)

update time

log log n

log n

(1, n^ε)

(\sqrt{\log n / \log \log n}, \sqrt{\log n / \log \log n})

lower bound

query time

Deterministic dictionaries
Amortized – can be made worst-case

Update time: $O(\log^2 n)$

Query time: $O(\log \log \log n)$

... and the result of this talk.
Such a function can be found in time $O(\log n \log \log n \log \log \log n)$. Let the problem to a smaller universe $S'$ on and constant-time evaluable, ‘trans-
that is $\{0\} \leftarrow \{0\}$ :
simultaneously in constant time.
If words are not short, many \( h \)'s can be evaluated.

Supports predecessor (and longest common prefix) queries.
Problem 1: How do we evaluate $y$ quickly?

Problem 2: Is "highly non-constant" over time.

Maintaining $y$ in $O \left( \log_2 n \right)$ amortized time/insertion:

$$\{ \emptyset, \{ z \} \} \subseteq |S| \text{ and } S' \cap |S| = S$$

(\text{where } y \text{ is 1-to-1 on } S')

$$\{ (x)^0 y \ldots (x)^{z-1} y (x)^1 \ldots (x)^1 \} \leftrightarrow x : y$$

Idea: Map to strings over $\{ 0, 1 \}$, length $O \left( \log_2 n \right)$.
Dealing with changes – the idea

**The key:** For $x \in S_i$, use the key

$$k_x = h_{l-1}(x) \ h_{l-2}(x) \ldots \ h_i(x)$$

The key is fixed, unique to $x$, and a prefix of $h(x)$.

**Example:**

Query for $x$:

- Search for $k$ that is prefix of $h(x)$ (if none exists, then $x \notin S$).
\{ x^y \mid S \subseteq x \}\text{ is a prefix of } y\}\}

Search for } x \text{ in } y\text{'s attached list, } \bullet \text{ the longest}

\{ x^s, x^s, x^s \}
\{ x^s, x^s, x^s \}
\{ x^s, x^s \}
\{ x^s, x^s \}
Is there a "smoother" range of trade-offs than presently known?

- Near-linear-time computation of reduction function wanted!
- Trees is a promising approach.
- Using dynamic universe reduction with exponential search.

Can Sundar's lower bound be matched (up to a polynomial)?

Open Questions: ...

Linear space?
- Maintaining the associated lists at no additional overhead.
- Predecessor (binary longest common prefix) data structure.

The actual details ...