Faster deterministic dictionaries

Rasmus Pagh

BRICS

Aarhus University, Denmark
A dictionary stores a set $S$ of $n$ elements from a finite universe $U$.

[Updates: Insertion and deletion of elements.]
Performance issues

Model of computation:

- Unit cost RAM with word size \( w \) and a standard instruction set, including multiplication.
- We allow compile-time computation (weak non-uniformity).
- \( U = \{0, 1\}^w \).

We consider the asymptotics of:

- Lookup time.
- Space usage.
- Construction time/update time.

... as functions of \( n \) [and \( w \)].
Fredman, Komlós and Szemerédi (1982) designed a static dictionary with:

**Lookup time:** Worst case $O(1)$.  

**Space usage:** $O(n)$ machine words.  

**Construction time:** Expected $O(n)$ by a randomized algorithm.  

Dietzfelbinger et al. (1988): Updates in $O(1)$ expected amortized time.
Removing randomness

Theme of this talk:
Using $O(n)$ words of space, how well can we do without random bits?

Motivation:

- Theoretical understanding.
- Random bits may not be available.
- Random bits may be expensive.
- No guarantee against bad performance.

We consider first the static case, then (briefly) the dynamic case.
Some deterministic dictionaries

<table>
<thead>
<tr>
<th>Reference</th>
<th>Lookup time</th>
<th>Construction time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alon-Naor '94</td>
<td>$O(w / \log n)$</td>
<td>$O(nw \log^4 n)$</td>
</tr>
<tr>
<td>Andersson '96</td>
<td>$O(\log w \log \log n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Miltersen '98</td>
<td>$O(1)$</td>
<td>$O(n^{1+\epsilon})$</td>
</tr>
<tr>
<td>Hagerup '99</td>
<td>$O(\log \log n)$</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>This talk</td>
<td>$O(1)$</td>
<td>$O(n \log n)$</td>
</tr>
</tbody>
</table>

[Dictionaries with lookup time $(\log n)^{\Omega(1)}$ not included]
Tarjan and Yao (1979) defined a class of hash functions for $|U| = O(n^2)$.

Look at $U$ as an $O(n) \times O(n)$ grid. The “double displacement” process is:

1. Displace each column (circularly), and then
2. Displace each row (circularly).

The final rows define the hash function.

Fact: Displacement values can always be chosen such that this function is 1-1 on $S$. 
Algorithms for finding displacement values

Tarjan and Yao’s algorithm:

**First-Fit-Decreasing search:** $O(n^2)$ time, deterministically.

Our two-step approach: **First randomize, then derandomize**

1. **Random-Decreasing search:** $O(n)$ time using $O(n \log n)$ random bits (exp.)
2. **Derandomized-Random-Decreasing search:** $O(n \log n)$ deterministic time.
1. What is a “good” displacement value?

— one that does not result in (many) more collisions than what could be expected by a random choice
2. Derandomization

The essential facts:

- We can (tentatively) fix bits of a displacement value and efficiently compute the \textit{conditionally} expected number of collisions when choosing the remaining bits at random.

- This is particularly simple if we exchange the cyclic shift with a permutation based on bit-wise exclusive or.

Theorem 1 \textit{For} $|\mathcal{U}| = O(n^2)$ \textit{there is a static dictionary occupying} $O(n)$ \textit{words of space and with constant lookup time, which can be constructed deterministically in time} $O(n \log n)$. 
Universe reduction

What remains is to reduce the general problem to the case $|U| = O(n^2)$. 

$\{0,1\}^{2 \log(n)+O(1)}$ 

Tarjan-Yao (trie) 

$\{0,1\}^{O(\log n)}$ 

This paper (Fredman-Willard technique) 

$U_2$ (size poly(n)) 

Miltersen, Hagerup (error-correcting codes) 

Weakly non-uniform 

$U$
Main theorem

Theorem 2  There is a static dictionary occupying $O(n)$ words of space and with constant lookup time, which can be constructed deterministically in time $O(n \log n)$ by a weakly non-uniform algorithm.
Removing randomness, dynamically

Miltersen’s dynamic dictionary
Hagerup’s trade-off
New trade-off
Dynamic universe reduction
Exponential search tree d.s.
Some open problems

**Static dictionaries:**
Does randomness help at all?

**Dynamic dictionaries:**
Better deterministic upper/lower bounds.