Strongly History-Independent Hashing with Applications

Guy Blelloch, Daniel Golovin
Carnegie Mellon University
Pittsburgh, PA USA

Why be History Independent?

- Information stored by an implementation of some *abstract data type* (ADT) is a superset of that demanded by the ADT
- Implementation may store undesirable clues of past use of the data structure.
  - File systems
  - Databases
  - Voting logs

Strong History-Independence (SHI)

- Store exactly the information required by the ADT, and no more.
- Impossible to learn more from the machine state than via the legitimate interface.
- For *reversible* data structures, equivalent to unique representation [Hartline *et al*. '05]:
  For every ADT state there is exactly one machine state that represents it.

Previous Work

- Pointer Machine Models & Comparison-based Models
  - Snyder ('77):
  - Sundar & Tarjan ('90):
  - Andersson & Ottmann ('95):
  - Buchbinder & Petrank ('06):

  Very strong lower bounds: $\Omega(n^{1/3})$ or worse for dictionaries, $\Omega(n)$ for heaps & queues
- Characterizing History Independence
  - Micciancio ('97): Oblivious data structures
  - Naor & Teague ('01): Weak & Strong History Independence
  - Hartline *et al.* ('05): SHI vs. Unique representation
- Strongly History Independent Data Structures
  - Amble & Knuth ('74): Hash tables (without deletions)
  - Naor & Teague ('01): Hash table (without deletions) with limited randomness
  - Acar *et al.* ('04): Dynamic trees (via dynamization)

Our Contributions

In a RAM we can build efficient SHI data structures. Hashing is the key.

	Time	Space
☆ Hash Table	Expected O(1) lookup, insert, & delete	Linear
☆ Perfect Hash Table	Expected O(1) updates Worst case O(1) lookup	Linear
BST	Expected O(log(n))	Linear
Ordered Dictionaries	Expected O(log(n)) [comparisons] Expected O(log log (n)) [Integer keys]	Linear
Order Maintenance	Expected O(1) updates Worst case O(1) compare	Linear

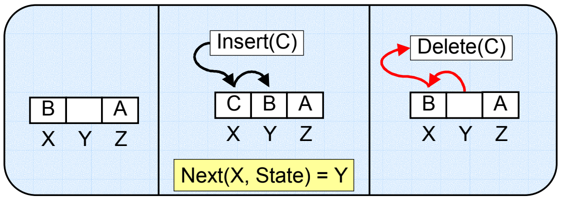
SHI Hashing (with Deletions)

- Based on correspondence with Gale-Shapley stable matching algorithm

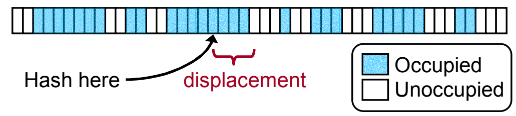
Gale-Shapley		Hashing
Men	⟷	Keys
Women		Memory slots
Male pref. lists		Probe sequences
Female pref. lists		Eviction policies
Matchings		Memory Configurations

Theorem [GS '62]: Every valid execution of the GS stable matching algorithm outputs the same stable matching.

- Probe(K, i): i^{th} slot in K's probe sequence
- Rank(K, X) = i if Probe(K, i) = X
- Next(X, State) is the slot containing X's favorite key that prefers X to its current slot.



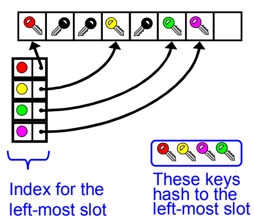
- With linear probing, and uniform eviction policy, we can implement operations in O(displacement) time.



Theorem [PPR '07]: Linear probing with 5-wise independent hash functions yields expected O(1) time operations (and hence expected O(1) displacement).

Dynamic Perfect Hashing

- Label each key with labels in $\{1,\ldots,(\log(n))^3\}$ using a hash function
- For all slots x, indices on {(label(k), displacement(k)) : k hashed to x}



Assume (for now):
(1) Every slot has $O(\frac{\log(n)}{\log\log(n)})$ keys hashed to it.
(2) Every key has displacement O(log(n))
(3) For all slots x, the keys hashed to x all get distinct labels

Assume (for now):
(1) Every slot has $O(\frac{\log(n)}{\log\log(n)})$ keys hashed to it.
(2) Every key has displacement O(log(n))
(3) For all slots x, the keys hashed to x all get distinct labels

Then each index is O(log n) bits.

Updates & queries in O(1) worst case time!

- Removing the assumptions:
  - If you don't really need to be SHI, just resample random bits "on the fly".
  - Otherwise, sample several hash functions on initialization, but use only what you need.

Other Results

- BSTs using treaps and hash table for memory allocation
- Ordered Dictionaries using treaps (comparison based) or van Emde Boas structures (integer keys)
- Order Maintenance

Conclusions

- Very small overhead for many fundamental SHI data structures in a RAM (unlike in pointer machines).
- Fast SHI hashing is a crucial enabling factor.

Future Work/Open Problems

- SHI versions of various other ADTs
- Develop techniques to automate the creation of SHI versions of various ADTs
- lower bounds in a RAM

Thank You