CS 420: Advanced Algorithm Design and Analysis
Spring 2015 – Lecture 4

Department of Computer Science
University of British Columbia

January 15, 2015

Announcements

Assignments...

- Asst1... due today
- Asst2....out (due next Thursday)

Readings...

- review CS320 (and earlier) notes, particularly material on ranking and selection as well as basic data structures: know where you can find what you may need to revisit
- basic material on dictionaries: [Cormen, Chapt. 11-15]
- material on treaps [EricksonNotes, Chapt 10]
- material on x-fast and y-fast tries [on line]
- material on hashing [Kleinberg, 13.6; Erickson, chapt 12]

Last class...

Begin unit on issues related to construction, search and application of *dictionaries*

- brief review of basic definitions concerning dictionary structures
- recall that the worst-case cost of $\mathrm{search}(S, x)$ is $O(\lg |S|)$ comparisons, even in a dynamic setting.
  - another *randomized* dictionary structure: treaps
  - expected running time of all basic operations is $O(\lg n)$ for an n-node treap
- recall that the cost of $\mathrm{search}(S, x)$ is $\Omega(\lg |S|)$, on a comparison-based model
  - information theoretic (leaf-counting) argument

Looking ahead...

Our goal, in the next few lectures is to understand how we might circumvent this lower bound, by *stepping outside the abstract comparison-based model*. We will consider:

- exploiting assumptions about the structure/size of the key space $\mathcal{U}$
- exploiting assumptions about the distribution of keys in S
- exploiting assumptions about the pattern of successive queries
- (if time permits) other issues: randomization, error tolerance...

Exploiting assumptions about the structure/size of the key space $\mathcal{U}$

Inputs are drawn from a restricted *universe* $\mathcal{U} = \{0, 1, \ldots m-1\}$

- direct access tables...properties and limitations
- ..... fast initialization
  - cost is $O(|S|)$
  - but it requires *even more space*

Today...

Continue exploiting assumptions about the structure/size of the key space $\mathcal{U}$

- inputs are drawn from a restricted *universe* $\mathcal{U} = \{0, 1, \ldots m-1\}$ (cont.)
  - finding the predecessor and successor keys using auxiliary structures (*x-fast tries*)
  - handling updates efficiently (*y-fast tries*)

Inputs are drawn from a restricted *universe* $\mathcal{U} = \{0, 1, \ldots m-1\}$

Extending the functionality of direct access tables to include successor & predecessor

- un-augmented direct access tables
  - worst-case cost of successor & predecessor is $\Theta(m)$
- augmented direct access tables
  - add a *tree directory* whose internal nodes are *marked* if their subtree contains a leaf in S
  - insert/delete requires updates to marks along the ancestor path
  - can find successor (predecessor) of $x \notin S$ by (i) walking up tree to *lowest ancestor* of x with a marked right (left) sibling z, then (ii) walking back down to the leftmost (rightmost) marked leaf in the subtree rooted at z.
  - worst-case cost of insert/delete and successor/predecessor is $\Theta(\lg m)$ (the height of the directory)

Inputs are drawn from a restricted *universe* $\mathcal{U} = \{0, 1, \ldots m-1\}$

Extending the functionality of direct access tables to include successor & predecessor

- augmented direct access tables
  - add a *tree directory* whose internal nodes are marked if their subtree contains a leaf in S
    - simple improvements:
      (i) explicit pred/succ links at leaf level;
      (ii) direct link (at every internal node) to left/rightmost marked descendent leaf
  - find both successor/predecessor of $x \notin S$ by (i) walking up tree to *lowest marked ancestor* z, (ii) jumping back down to leftmost or rightmost marked leaf in the subtree rooted at z, and (iii) following leaf-level pred/succ links.
  - worst-case cost remains $\Theta(\lg m)$ (the height of the directory)

Inputs are drawn from a restricted *universe* $\mathcal{U} = \{0, 1, \ldots m-1\}$

Extending the functionality of direct access tables to include successor & predecessor

- augmented direct access tables: *x-fast trie*
  - add a *tree directory* whose internal nodes are marked if their subtree contains a leaf in S with (i) explicit pred/succ links at leaf level; (ii) direct link to left/rightmost marked child
  - instead of walking up tree from x to its *lowest marked ancestor* z, find z by *binary search* of the ancestors of x.
  - cost (of predecessor and successor)is reduced to $O(\lg \lg m)$
  - however, cost of insert/delete remains $\Theta(\lg m)$.

Inputs are drawn from a restricted *universe* $\mathcal{U} = \{0, 1, \ldots m-1\}$

How do you do binary search on a path in a tree?

- represent the tree *implicitly*, like an *implicit heap* with parent pointers replaced by indexing
  - the parent of node with index i is the node with index $\lfloor i/2 \rfloor$
  - the j-th ancestor of the node with index i is the node with index $\lfloor i/2^j \rfloor$

Inputs are drawn from a restricted *universe* $\mathcal{U} = \{0, 1, \ldots m - 1\}$

Extending the functionality of direct access tables to include successor & predecessor, as well as fast updates

- augmented direct access tables: *y-fast tries*
  - key ideas: (i) *partition* keys in S into subsets of size about $\lg m$, (ii) represent each subset as a standard balanced binary search tree, and (iii) store one *representative* from each subset in an x-fast trie.

Inputs are drawn from a restricted *universe* $\mathcal{U} = \{0, 1, \ldots m-1\}$

Extending the functionality of direct access tables to include successor & predecessor, as well as fast updates

- augmented direct access tables: *y-fast tries*
  - successor/predecessor is *more complicated*: it involves search in the x-fast tree to find the closest representative, followed by search in $O(1)$ of the subtrees to find the true successor/predecessor. But the cost remains $O(\lg \lg m)$.
  - insert/delete involves (i) finding the correct partition (using successor/predecessor) and (ii) updating the representation of that partition. Only rarely (when partitions become too large or too small, or the representtive no longer "represents") is it necessary to add or delete partition representatives in the x-fast trie. So the *amortized* cost of insert/delete is $O(\lg \lg m)$.

Inputs are drawn from a restricted *universe*
$\mathcal{U} = \{0, 1, \ldots m-1\}$

But...what about the space requirements!

Next time...

Stepping away from the most general (comparison-based) dictionary model...different possibilities

- inputs are drawn from a restricted *universe* $\mathcal{U} = \{0, 1, \ldots m-1\}$ (cont.)
  - overcoming space concerns with previous structures
    - hashing (the role of randomization)
    - universal hashing
    - perfect hashing