CS 420: Advanced Algorithm Design and Analysis
Spring 2015 – Lecture 3

Department of Computer Science
University of British Columbia

January 13, 2015

Announcements

Assignments...

- Asst1... due Thursday .... Questions? Clarifications?

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...

Continue case study on finding extrema (reviewing basic issues & previewing others)

- taking the cost of other operations/resources into account
  - *auxiliary space* in finding the max and second largest; *streaming algorithms*; *time-space tradeoffs*
  - *update costs* in finding the maximum (the iterative and on-line hiring problems); *randomized algorithms*
- finding extrema in other computation models
  - parallel algorithms
  - distributed algorithms; communication complexity
- finding extrema in more restricted or more general input domains
  - inputs are drawn from $\mathcal{U} = \{0, 1, \ldots m - 1\}$
  - inputs are specified *implicitly*; *linear programming*
  - inputs are points in two (or higher) dimensions; *computational geometry*

Today...

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

- brief review of basic definitions and results concerning dictionary structures
- another *randomized* dictionary structure: treaps
- stepping away from the most general (comparison-based) model...different possibilities
  - inputs are drawn from a restricted *universe* $\mathcal{U} = \{0, 1, \ldots m - 1\}$
    - direct access tables...properties and limitations
    - ..... fast initialization
    - ..... finding the closest key

Issues related to construction, search and application of *dictionaries*

Brief review of basic definitions and results concerning dictionary structures...

- A *(static) dictionary* is a data structure that represents a finite set $S = \{a_1, a_2, \ldots, a_n\}$ of *keys* drawn from a *universe* (or *key space*) $\mathcal{U}$, that facilitates *membership queries* of the form $\mathrm{member}(S, x)$ that return `true`, if $x \in S$, and `false`, otherwise. (More generally, it returns a pointer to an element of S, to facilitate access to associated information.)
- If $\mathcal{U}$ is a *metric space* (with distance function $d(\cdot, \cdot)$) then the function $\mathrm{closest}(S, x)$ returns a key a in S that minimizes $d(a, x)$. Depending on context, the function $\mathrm{search}(S, x)$ could be interpreted as either $\mathrm{member}(S, x)$ or $\mathrm{closest}(S, x)$.

Issues related to construction, search and application of *dictionaries*

Brief review of basic definitions and results concerning dictionary structures...

- If $\mathcal{U}$ admits a *total ordering*, typically denoted $\leq$, then it is natural to include the additional operations $\max(S)$, $\min(S)$, $\operatorname{successor}(S, x)$, and $\operatorname{predecessor}(S, x)$ (with the natural interpretations).
- If in addition the dictionary structure services the operations $\operatorname{insert}(S, x)$ and $\operatorname{delete}(S, x)$, the structure is said to be *dynamic*.

Issues related to construction, search and application of *dictionaries*

Brief review of basic definitions and results concerning dictionary structures...

- Recall that the worst-case cost of $\mathrm{search}(S, x)$ is $O(\lg |S|)$ comparisons, even in a dynamic setting.
- exhibited by a variety of balanced binary search tree structures: AVL trees, red-black trees, B-trees...
- exhibited in expected case: skip lists

Treaps: another randomized implementation of *dictionaries*

A *treap* is a binary tree T each node of which has both a search *key* and a (distinct) *priority*. T is *simultaneously*:

- A *binary search tree* with respect to the search keys; and
- a *min-heap* with respect to the priorities

Note that

- every subtree of a treap is a treap
- a treap is completely determined by the keys and priorities
- treaps were first introduced (as *Cartesian trees*) by McCreight [C.ACM 1980] and later in their randomized form by Seidel and Aragon [Algorithmica, 1996]

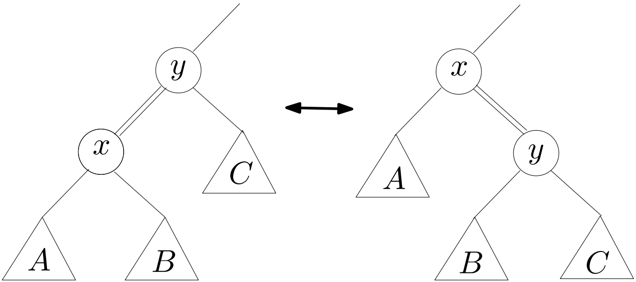
Treaps: another randomized implementation of *dictionaries*

Treaps support all of the following operations in time proportional to the depth of some node in the structure:

- search
  - standard BST search
- insert and delete
  - insert: (unsuccessful) search followed by upward *rotations* to heapify
  - delete: replace priority by ∞, heapify, and prune
- split and join
  - split: insert a splitting key with priority $-\infty$, and extra the two subtrees of this (new) root
  - join: combine with an artificial root (with priority $-\infty$), then delete root

Tree rotation restructuring primitive

Rotation of edge (x, y)



- Preserves in-order of nodes
- reduces depth of child node by 1

Treaps: another randomized implementation of *dictionaries*

Treaps are particularly interesting when the priorities associated with nodes are chosen *uniformly at random* from some continuous domain (like $[0, 1)$). In this case the *expected depth* of any node in an n-node treap T is $O(\lg n)$, and so the expected running time of all of the operations is also $O(\lg n)$.
The critical observation is that, for all $i < k$:

- the probability that node i is an ancestor of node k is exactly $\frac{1}{k-i+1}$; and
- the probability that node k is an ancestor of node i is exactly $\frac{1}{k-i+1}$.

Given this, we compute the expected depth of a node j by summing, over all $i < j$ and all $k > j$, the probability that node i (or k) is an ancestor of node j:

$$\sum_{i<j} \frac{1}{j-i+1} + \sum_{k>j} \frac{1}{k-j+1}$$

which is

$$H_j - 1 + H_{n-j+1} - 1 < 2\ln n - 2.$$

Treaps: another randomized implementation of *dictionaries*

Question: What is another name for the algorithm sorts n keys as follows: (i) associate a random priority with each key;
(ii) build a treap on the resulting set, by successive insertions
(iii) output the keys of T by doing an in-order traversal

Issues related to construction, search and application of *dictionaries*

Returning to our review of basic results concerning dictionary structures...

- Recall that the cost of $\mathrm{search}(S, x)$ is $\Omega(\lg |S|)$, on a comparison-based model
- 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
  - other issues: randomization, error tolerance...

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

- direct access tables
  - represent set S as a *characteristic vector* (bit array) $A[0, m-1]$, where $A[i] = i$, if $i \in S$, and $A[i] = 0$, otherwise.
  - insert, delete and member operations all have cost $O(1)$!
  - What's not to like?
    - initialization cost
    - successor & predecessor cost
    - space requirement

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

- direct access tables
  - *lazy initialization*
    - cost is $O(|S|)$
    - but it requires *even more space*
  - extending functionality to include successor & predecessor
    - un-augmented direct access tables
    - augmented direct access tables

Next time...

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

- stepping away from the most general (comparison-based) model...different possibilities
  - inputs are drawn from a restricted *universe* $\mathcal{U} = \{0, 1, \ldots m - 1\}$ (cont.)
    - finding the closest key using auxiliary structures (x-fast tries)
    - handling updates efficiently (y-fast tries)
    - space considerations; hashing (more randomization)