COMP 3170 - Analysis of Algorithms & Data Structures

Shahin Kamali

Lecture 8 - Jan. 19, 2018
CLRS 12.2, 12.3, 13.2, read problem 13-3
University of Manitoba
Dictionary ADT

Definition

A dictionary is a collection $S$ of items, each of which contains a key and some data, and is called a key-value pair (KVP).

- Keys can be compared and are (typically) unique.
- We often focus on keys; associating data with keys is easy.

Operations:

- $search(x)$: return true iff $x \in S$
- $insert(x, v)$: $S \leftarrow S \cup \{x\}$
- $delete(x)$: $S \leftarrow S \setminus \{x\}$
- additional: $join$, $isEmpty$, $size$, etc.
Optional Operations

In addition to the main operations (search, insert, delete), the followings are useful:

- \textit{predecessor}(x): return the largest \( y \in S \) such that \( y < x \)
- \textit{successor}(x): return the smallest \( y \in S \) such that \( y > x \)
- \textit{rank}(x): return the index of \( x \) in the sorted array
- \textit{select}(i): return the key at index \( i \) in the sorted array \( \rightarrow i \)’th order statistic
- \textit{isEmpty}(x): return true if \( S \) is empty
Dictionaries

- Dictionary is a collection of key-value pairs with the support of search, insert, delete (and possibly some other operations).
- There is a total ordering of elements, i.e., keys are comparable.
- Is dictionary an abstract data type or a data structure?
  - It is an abstract data type; we did not discuss implementation.
  - Different data structures can be used to implement dictionaries.
Elementary Implementations

- **Common assumptions:**
  - Dictionary has $n$ KVPs
  - Each KVP uses constant space
  - Comparing keys takes constant time

- **Unsorted array or linked list**
  - $\text{search } \Theta(n)$
  - $\text{insert } \Theta(1)$
  - $\text{delete } \Theta(n)$ (need to search)

- **Sorted array**
  - $\text{search } \Theta(\log n)$
  - $\text{insert } \Theta(n)$
  - $\text{delete } \Theta(n)$
# Data Structures for Dictionaries

<table>
<thead>
<tr>
<th></th>
<th>space</th>
<th>search</th>
<th>insert/delete</th>
<th>predecessor</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsorted array, linked list</td>
<td>$\Theta(n + a)$</td>
<td>$\Theta(n)$</td>
<td>$\Theta(1)/\Theta(n)$</td>
<td>$\Theta(n)$</td>
</tr>
<tr>
<td>sorted array</td>
<td>$\Theta(n + a)$</td>
<td>$\Theta(\log n)$</td>
<td>$\Theta(n)$</td>
<td>$\Theta(\log n)$</td>
</tr>
<tr>
<td>sorted linked-list</td>
<td>$\Theta(n)$</td>
<td>$\Theta(n)$</td>
<td>$\Theta(n)$</td>
<td>$\Theta(n)$</td>
</tr>
<tr>
<td>unbalanced BST</td>
<td>$\Theta(n)$</td>
<td>$\Theta(n)$</td>
<td>$\Theta(n)$</td>
<td>$\Theta(n)$</td>
</tr>
<tr>
<td>balanced BST</td>
<td>$\Theta(n)$</td>
<td>$\Theta(\log n)$</td>
<td>$\Theta(\log n)$</td>
<td>$\Theta(\log n)$</td>
</tr>
<tr>
<td>hash tables</td>
<td>$\Theta(n + a)$</td>
<td>$\Theta(1)^*$</td>
<td>$\Theta(1)^*$</td>
<td>$\Theta(n + a)$</td>
</tr>
<tr>
<td>skip list</td>
<td>$\Theta(n)^*$</td>
<td>$\Theta(\log n)^*$</td>
<td>$\Theta(\log n)^*$</td>
<td>$\Theta(\log n)^*$</td>
</tr>
</tbody>
</table>

- $n$: number of KVPs.
- $a$: the length of array; when we use sorted/unordered arrays, $a \geq n$.
- $^*$: expected time/space
Binary Search Trees (review)

**Structure** A BST is either empty or contains a KVP, left child BST, and right child BST.

**Ordering** Every key \( k \) in \( T.left \) is less than the root key.
Every key \( k \) in \( T.right \) is greater than the root key.
BST Search and Insert

\[\text{search}(k)\] Compare \(k\) to current node, stop if found, else recurse on subtree unless it’s empty

\[\text{insert}(k, v)\] Search for \(k\), then insert \((k, v)\) as new node

Example:
BST Delete

- If node is a leaf, just delete it.
- If node has one child, move child up
- Else, swap with successor or predecessor node and then delete
**Height of a BST**

*search, insert, delete* all have cost $\Theta(h)$, where $h = \text{height of the tree} = \text{max. path length from root to leaf}$

If $n$ items are *inserted* one-at-a-time, how big is $h$?

- **Worst-case:**
- **Best-case:**
- **Average-case:**