**Lesson 20 – Tables I**

**Reading: Chapter 12, Sections 1-2 of the text.**

**Learning Objectives:**

* Define the operations of ADT Table.
* Describe various implementations of ADT Table.
* Compare the time complexity of various implementations of ADT Table.
* Describe the structure and operations of a heap.

**Heaps:**

* A **heap** is an ADT that is similar to a binary search tree, although it differs from a binary search tree in two significant ways.
  + First, while you can view a BST as sorted, a heap is ordered in a much weaker sense.
  + Second, while BSTs come in many different shapes, heaps are always **complete binary trees**.
* So, whereas a complete binary tree is not ordered in and of itself, a heap is a complete binary tree such that:
  + It is empty; or
  + The root contains a search key greater than or equal to the search key in each of its children, and
  + Each child is the root of a subtree that is also a heap.
* Give an example of a heap.
* Insert a new value:
  + Traverse the tree to the next open slot.
  + Add the new item/node.
  + Trickle up new item as long as child key is greater than parent key.

9

/ \

5 6

/ \ .

3 2 .

add 15

9

/ \

5 6

/ \ / .

3 2 15 .

9

/ \

5 15

/ \ / .

3 2 6 .

15

/ \

5 9

/ \ / .

3 2 6 .

* Since a heap is balanced, insert can be done in, worst case, O(log n) time.
* Delete root:
  + Find last item in complete binary tree and replace root value with found value.
  + Delete last item.
  + Trickle down as long as parent key is smaller by swapping with larger child value.

10

/ \

9 6

/ \ / .

3 2 5 .

delete 10

5

/ \

9 6

/ \ / .

3 2 5 .

5

/ \

9 6

/ \ .

3 2 .

9

/ \

5 6

/ \ .

3 2 .

* The replacement value is found in O(log n) time. The **re-heapify** operation can also be done in O(log n) time. Thus, overall time complexity of delete root is O(2 log n) = O(log n).
* What we describe here is a max-heap. You could also implement a min-heap. What would be the min-heap property?
* Consider the heap where the key is the priority. Thus, a heap can be used as a priority queue.

**The ADT Table:**

* A **table** is a value-oriented ADT whose operations are of the form:
  + Insert a data item containing the value x.
  + Delete a data item containing the value x.
  + Ask a question about a data item containing the value x.
* Another name for a table is a **dictionary**. For example, Python has a built-in dictionary data-structure.
* Since the items in a table are value-oriented, the way the information is stored **may be organized** to facilitate search.
* Operations of the ADT Table:
  + Create an empty table.
  + Determine whether a table is empty.
  + Determine the number of items in a table.
  + Insert a new item into a table.
  + Delete the item with a given search key from a table.
  + Retrieve the item with a given search key from a table.
  + Traverse the items in a table in sorted search-key order.
* For simplicity, we assume that all search keys are unique.
* Linear implementations of a table:
  + Unsorted, array-based.
  + Unsorted, reference-based.
  + Sorted (by search key), array-based
  + Sorted (by search key), reference-based.
* Non-linear implementations of a table.
  + Binary search tree.
* Comparison of the time-complexity of the operations for ADT Table.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Insertion | Deletion | Retrieval | Traversal |
| Unsorted array-based | O(1) | O(n) | O(n) | O(n) |
| Unsorted reference-based | O(1) | O(n) | O(n) | O(n) |
| Sorted array-based | O(n) | O(n) | O(log n) | O(n) |
| Sorted reference-based | O(n) | O(n) | O(n) | O(n) |
| Binary Search Tree | O(h) | O(h) | O(h) | O(h) |

import java.util.HashMap;

public class **DictionaryExample** {

    public static void main(String[] args) {

        HashMap<Integer, String> courses = new HashMap<>();

        // add courses to the dictionary

        courses.put(1302, "Principles");

        courses.put(2302, "Programming");

        courses.put(3302, "Data Structures");

        courses.put(2311, "Event Driven");

        courses.put(2314, "Architecture");

        courses.put(3333, "Discrete");

        // construction

        System.out.println(courses);

        // contains

        System.out.println("---------------CONTAINS");

        for (int i = 2000; i <= 3000; i++)

            if (courses.containsKey(i))

                System.out.println("Number: " + i + " Name " + courses.get(i));

        // remove

        System.out.println("---------------REMOVE");

        courses.remove(2314);

        // traverse

        for (Integer i : courses.keySet())

            System.out.println("Number: " + i + " Name " + courses.get(i));

        // traverse values

        for (String s : courses.values())

            System.out.println(s);

        // remove all

        courses.clear();

        System.out.println(courses);

    }

}