Sets, Dictionaries, Maps, Oh My!

Introduction

- In this module, we will study the notion of the **set** abstract data type. Sets are as fundamental to computer science as they are to mathematics. In computer science, unlike in mathematics, sets are **dynamic** as they can change over time.
- A set that only needs to support the ability to insert elements, delete elements, and test for membership is known as a **dictionary**.
- A closely related term is **map** that often implies an associative data structure. By associative, we mean that there is a *one to one mapping* from a key to a value.
  - Note that Java uses the term Map to mean the same thing as a dictionary.
  - Python has dictionary as built-in type! (**Note**: Python uses map for something entirely different – it is used for *functional programming* where we want to apply a function to each element in a list.)
- Some algorithms require additional operations such as *minimum, maximum*, *successor*, and *predecessor* operations.
- The elements of a set will be represented by an object. Some types of sets assume that each object has a **key** value. Note that a key may be a single field in the object or a more complex combination of fields. In the latter case, we will need to use some type of `compareTo` method to compare two keys.

A key to understanding keys!

- A key is any information that we can use to compare two elements. For example, it could simply be an integer, or be double, or be more complex, like a String, Color, or a Task.
- An element may contain just the key and nothing else. Like an array of integers. More commonly, an element has a key and associated data (we call this satellite data). For example, a row in a spreadsheet may have a student id as key and associated information like class, address, email etc. An address object may have a name, house/apt number, a street address, a city name, and a zip code. Here name may be the key and the rest satellite data.
- When programming, we often consider an object and an element to be the same thing (except for primitive elements such as ints)
- For a more complex element, we need to define a `compareTo` method that allows us to compare two elements. That's what we are doing in Project 2. In order to compare two Task objects, we first compare them by priority and if that is equal, then we compare them by hourCreated. The pseudocode often just uses $<$ to compare two elements but we have to use `compareTo` to implement the more complex way of comparing often found in applications.

- Some sets assume that the keys are drawn from a **totally ordered set**, such as integers, real numbers, strings, or a combination of fields. A total ordering allows to define the minimum (or maximum) or to speak of the next larger (or smaller) element than a given element in the set.
- The object may have **satellite data** that is carried around in other object attributes but the satellite data is typically not used in the set implementation.

Operations on dynamic sets

Here are some typical operations on a set. Any specific application may require only some of these to be implemented.

SEARCH(S, K)
Given a set S and a key value k, returns a pointer x to an element in S such that $x.key == k$ or NIL if no such element belongs to S.

INSERT(S, X)
Adds an element x to the set S. Here x is a pointer to the element.

DELETE(S, X)
Given a pointer x to an element in S, removes x from S. Note that this operation takes a pointer to an element, not a key value.

MINIMUM(S) and MAXIMUM(S)
Assuming a totally ordered set, returns a pointer to an element with smallest (or largest) key value.

SUCCESSOR(S, X)
Assuming a totally ordered set, returns a pointer to an element with the next larger element in S, or NIL if x is the maximum element.

PREDECESSOR(S, X)
Assuming a totally ordered set, returns a pointer to an element with the next larger element in S, or NIL if x is the minimum element.

Recommended Exercises

Think-Pair-Share: What is the output look like if we call MINIMUM followed by $n-1$ calls to SUCCESSOR?

Think-Pair-Share: What is the output look like if we call MAXIMUM followed by $n-1$ calls to PREDECESSOR?

Exercise 1: How would you implement the set ADT using unsorted arrays?

Exercise 2: How would you implement the set ADT using sorted arrays?