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 are known as a **dictionary**.
- A closely related term is **map** that often implies an associative data structures where there is one to one mapping from a key to a value.
  - Note that Java uses Map to mean the same thing as a dictionary.
  - Pythin 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 more complicated operations such as minimum, maximum, successor, and predecessor operations.
- The elements of a set will be represented by object. Some types of sets assume that each object has a **key** value. The object may have **satellite data** that is carried around in other object attributes but is typically no used in the set implementation.
- Some sets assume that the keys are drawn from a **totally ordered set**, such as integers, real numbers or strings. 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.

Operations on dynamic sets

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

SEARCH(S, K)
Given a set S and a key value k, returns a pointer to 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 pointer to by x to the set S.

DELETE(S, X)
Given a points 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?

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

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