Data structures and Algorithms

Note04-Data Structures

Edited By

Arshia Gharooni

2023

0.1 Data structures

Data structures are a fundamental concept in computer science that is essential for efficient programming. In simple terms, a data structure is a way to organize and store data, with algorithms that support operations on the data. The collection of supported operations is called an interface, also known as an API or ADT. In this note, we will discuss two main interfaces of data structures, which are Sequence and Set. We will also discuss the special case interfaces of stack, queue, and dictionary.

0.2 Data Structure Interfaces

A data structure interface is a specification of the operations that a data structure should support. The interface defines what operations are supported and what parameters and data types are expected. On the other hand, a data structure is a representation of the actual implementation of those operations. The data structure specifies how the operations are supported and how the data is stored.

0.2.1 Basic Functionality

The basic functionality of a data structure refers to the operations it can perform, such as adding, removing, searching, and retrieving elements. The functionality of a data structure is defined by its interface, which includes its methods and properties. For example, the basic functionality of an array includes adding and removing elements, accessing elements by index, and determining the length of the array.

0.2.2 Properties

Properties are the attributes of a data structure that describe its state. They can be used to determine the size, capacity, and other characteristics of the data structure. Examples of properties include:

- size – the number of elements in the data structure
- capacity – the maximum number of elements that can be stored in the data structure
- isEmpty – a Boolean value that indicates whether the data structure is empty or not

0.2.3 Implementation of Interfaces

The implementation of the interfaces of data structures is an important consideration when designing and developing software. There are several factors

that can influence the implementation of data structure interfaces, including performance, memory usage, and ease of use.

Performance

Performance is a critical factor when implementing data structure interfaces. The performance of a data structure is determined by its efficiency in terms of time and space complexity. For example, an array has O(1) time complexity for accessing elements by index, but O(n) time complexity for inserting or deleting elements in the middle of the array.

Memory Usage

Memory usage is another important consideration when implementing data structure interfaces. The memory requirements of a data structure can affect the overall performance of a program, as well as its scalability. For example, a linked list uses more memory than an array, but it can be more efficient for inserting or deleting elements in the middle of the list.

Ease of Use

Ease of use is also a factor when implementing data structure interfaces. The interface of a data structure should be intuitive and easy to use, even for developers who are not familiar with the specific implementation. For example, the interface of an ArrayList in Java is similar to that of an array, making it easy for developers to switch between the two.

0.2.4 Types of Interfaces

There are two main interfaces of data structures, which are Sequence and Set. We will also discuss the special case interfaces of stack, queue, and dictionary.

Set Interface

Sequence Interface

The sequence interface maintains a sequence of items where order is extrinsic. This means that the order is explicitly specified and is not inherent in the data. An example of a sequence is (x0, x1, x2, . . . , xn-1), where zero indexing is used. The sequence interface supports the following sequence operations:

- Container build(X): Given an iterable X, build a sequence from the items in X.
- len(): Return the number of stored items.

- Static iter seq(): Return the stored items one-by-one in sequence order.
- get at(i): Return the i-th item.
- set at(i, x): Replace the i-th item with x.
- Dynamic insert at(i, x): Add x as the i-th item.
- delete at(i): Remove and return the i-th item.
- insert first(x): Add x as the first item.
- delete first(): Remove and return the first item.
- insert last(x): Add x as the last item.
- delete last(): Remove and return the last item.

The sequence interface can be used to implement other data structures such as stacks and queues. A stack is a data structure that supports inserting and deleting elements from the top of the stack, while a queue supports inserting elements at the rear and deleting elements from the front.

Set Interface

The set interface maintains a set of items where order is intrinsic. This means that the order is inherent in the data and not explicitly specified. The set interface supports the following set operations:

- Container build(X): Given an iterable X, build a set from the items in X.
- len(): Return the number of stored items.
- Static find(k): Return the stored item with key k.
- Dynamic insert(x): Add x to the set (replace item with key x.key if one already exists).
- delete(k): Remove and return the stored item with key k.
- Order iter ord(): Return the stored items one-by-one in key order.
- find min(): Return the stored item with the smallest key.
- find max(): Return the stored item with the largest key.
- find next(k): Return the stored item with the smallest key larger than k.

- find prev(k): Return the stored item with the largest key smaller than k.

The set interface can be used to implement a dictionary, which is a data structure that supports storing key-value pairs. A dictionary is similar to a set, but instead of storing only keys, it stores both keys and their associated values.