Data Structures and Algorithms

Section: Introduction to DS and Algorithms

Devere Anthony Weaver

Data Structures

"A **data structure** is a way of organizing, storing, and performing operations on data."

Some examples of these operations include:

- updating stored data
- searching
- inserting
- removing data

Examples of basic data structures include:

- Record - stores subitems (fields) with a name associated with each subitem
- Array - stores an ordered list of items where each is accessible by a positional index
- Linked List - stored an ordered list of items in nodes, where each node stores data and has a pointer to the next node
- Binary Tree - each nodes stores data and has up to two children
- Hash Table - stores unordered items by mapping each item to a location in the array
- Heap - max heap is a tree that maintains a property that a node's key is greater than or equal to the node's children's keys; min-heap is a tree that maintains the property that a node's key is less than or equal to the node's children's keys
- Graph - represents connections among items and consists of vertices connected by edges

When choosing a data structure for a program, the best data structure requires determining which DS provides a balance given expected uses. In other words, the selection of a DS depends on the type of data being stored and the operations the program may need to be performed on the data.

Introduction to Algorithms

"An **algorithm** describes a sequence of steps to solve a computational problem or perform a calculation."

A **computational problem** specifies an input, a desired output and a question about the input.

Efficient Algorithms and Hard Problems

Algorithm efficiency is most commonly measured by the algorithm runtime and an efficient algorithm is one whose runtime increases no more than polynomially with respect to input size.

NP-complete problems are the set of problems for which no known efficient algorithm exists yet. They have the following characteristics:

- No efficient algorithm has been found
- No one has proven an efficient algorithm is impossible
- If an efficient algorithm exists for one NP-complete, then all NP-complete problems can be solved efficiently

Knowing beforehand that a problem is NP-complete allows the implementer to find an algorithm that is good but non-optimal. Again, when we say optimal, we mean that we can find a polynomial-time algorithm.

Abstract Data Types

An **abstract data type (ADT)** is a data types described by predefined user operations without indicating how each operation is implemented. The idea here is that a programmer can use an ADT without having any knowledge of how the ADT is implemented.

An common example of a ADT is a **list**. Lists are used for storing ordered data and come with a set of operations defined on the data. However, a programmer using the list need not know whether the list is implemented using an array or maybe a linked-list.

Some common ADTs are:

- List - ADT for ordered data (usually implemented with an array or a linked list)
- Dynamic Array - holding ordered data and allowing indexed access (usually implemented with an array)
- Stack - items are only inserted on or removed from the top of the stack (linked list)
- Queue - items are inserted at the end of the queue and removed from the front of the queue (linked list)
- Deque - doubled-ended queue, items can be inserted and removed from the front and back (linked list)
- Bag - storing items in which the order doesn't matter and duplicate items are allowed (array, linked list)
- Set - collection of distinct elements (binary search tree, hash table)
- Priority queue - each item has a priority and items with higher priority are closer to the front (heap)
- Dictionary - maps keys with values (hash table, binary search tree)

This completes the section. It was simply a brief introduction to the concepts of data structures, algorithms, and the relation between the two ideas. More in-depth examination of particular data structures and algorithms will be covered later.