College of Engineering & Applied Sciences

CSPB 2270

Data Structures

Class Notes

University Of Colorado

2024

Sections

Week 1

C++ Review, Debugging, Unit Testing

1.0.1 Activities

The following are the activities that are planned for Week 1 of this course.

- Take the C++ assessment
- Read the C++ refresher or access other resources to improve your skills (book activities are graded but the grades are not included in your final grade for this course)
- Read the zyBook chapter(s) assigned and complete the reading quiz(s) by next Monday
- Access the GitHub Classroom and get your Assignment-0 repository created, cloned, edited, and graded by next Tuesday
- Watch the videos for Cloning GitHub Classroom Assignments, Setting up an IDE in Jupytherhub, and Unit Testing

1.0.2 Lectures

Here are the lectures that can be found for this week:

- Course Concepts
- GitHub Classroom
- GitHub Security
- Accepting an Assignment
- Accessing Git Files
- Cloning Into JupyterHub
- VSCode in JupyterHub
- Multi File Programming
- Unit Testing Basics

1.0.3 Programming Assignment

The programming assignment for Week 1 - **Using GitHub and GitHub Classroom**.

1.0.4 Chapter Summary

The first chapter of this week was **Chapter 1 - Introduction to Data Structures**.

Section 1.1 - Data Structures

We define Data Structures to be the following:

- A data structure is a method of organizing, storing, and performing operations on data.
- Operations performed on data structures include accessing or updating stored data, searching for specific

data, inserting new data, and removing data.

- Understanding data structures is crucial for effectively managing and manipulating data.

To summarize, data structures are methods of organizing, storing, and manipulating data, including arrays, linked lists, stacks, queues, trees, graphs, hash tables, and heaps.

Arrays

Arrays - Sequential collections of elements with efficient access and modification.

- Sequential collection of elements with unique indices. Indexes from zero.
- Efficient access and modification of elements at specific locations.
- Less efficient for inserting or removing elements in the middle.

Linked Lists

Linked Lists - Chain of nodes allowing efficient insertion and removal.

- Chain of nodes where each node contains data and a reference to the next node.
- Efficient insertion and removal of elements.
- Sequential traversal required for access specific elements.

Stacks

Stacks - Follows Last-In-First-Out (LIFO) principle for efficient insertion and removal from the top.

- Follows Last-In-First-Out (LIFO) principle.
- Insert and remove elements from the top of the stack.
- Useful for tasks like function class and undo operations.

Queues

Queues - Follows First-In-First-Out (FIFO) principle for efficient insertion, and removal from the front and rear.

- Follows First-In-First-Out (FIFO) principle.
- Insert elements at the rear and remove elements from the front.
- Useful for tasks like process scheduling.

Queues

Trees

Trees - Hierarchical structure for enabling efficient searching, insertion, and deletion.

- Hierarchical structure consisting of nodes connected by edges.
- Efficient searching, insertion, and deletion operations.
- Suitable for organizing file systems or representing hierarchical relationships.

Basic Data Structures

Graphs - Collection of nodes connected by edges, useful for representing complex relationships.

- Collection of nodes connected by edges.
- Each node can have multiple connections.
- Used to represent complex relationships like social networks or computer networks.

Basic Data Structures

Hash Tables - Data structure that uses hashing for efficient insertion, retrieval, and deletion of key-value pairs.

- Efficient data structure using hashing for fast key-value pair operations.
- Uses a hash function to convert keys into indices.
- Provides quick access to elements and handes collisions for proper storage.

Heaps

Heaps - Binary-tree based structure that ensures efficient retrieval of the minimum or maximum element.

- Binary tree-based structure for efficient retrieval of minimum or maximum element.
- Maintains a partial order property, such as the min-heap or max-heap property.
- Supports fast insertion and deletion of elements while preserving the heap property.

In the study of data structures, we explore various methods of organizing, storing, and manipulating data.

Arrays are sequential collections of elements, allowing efficient access and modification. Linked Lists from a chain of nodes, facilitating efficient insertion and removal. Stacks follow the Last-In-First-Out principle and are useful tasks like function calls and undo opertions. Queues follow the First-In-First-Out principle and are suitable for process scheduling.

Trees, consisting of nodes connected by edges, provide a hierarchical organization, enabling efficient searching, insertion, and deletion. Graphs are collections of nodes connected by edges and represent complex relationships like social networks or computer networks.Hash Tables emply hashing for efficient insertion, retrieval, and deletion of key-value pairs. They use a hash function to convert keys into indices, providing fast access to elements while handling collisions. Heaps, based on binary trees, allow efficient retrieval of the minimum or maximum element. They maintain a partial order property and support fast insertion and deletion while preserving the heap property.

Understanding these data structures and their characteristics is essential for problem-solving and designing efficient algorithms in data-oriented scenarios.

Section 1.2 - Abstract Data Types

Abstract Data Types (ADT) can be summarized as:

- Abstract Data Types (ADTs) define a set of operations and behavior for manipulating data without specifying the implementation details.
- ADTs provide a logical representation of data and operations, focusing on the "what" rather than the "how" of data manipulation.
- ADTs promote code abstraction and modularity, allowing for reusable and maintable code by encapsulating dtat and providing a clear interference for interaction.

To summarize, Abstract Data Types (ADTs) provide a high-level, logical representation of data and operations, focusing on the "what" rather than the "how", enabling code abstraction and modularity for reusable and maintable programming.

List

List - A basic data structure that represents an ordered collection of elements, allowing for efficient insertion, deletion, and retrieval operations.

- Lists are a versatile data structure that can store elements of any type and maintain their order, allowing for easy access and modification.
- They offere efficient insertion and deletion operations at both ends, making them suitable for scenarios where elements need to be dynamically added or removed.
- Lists can be implemented using various techniques such as arrays or linked lists, each with its own trade-offs in terms of memory usage and performance.

Dynamic Array

Dynamic Array - A dynamic array is a resizable data structure that provides the flexibility to dynamically adjust its size to accommodate the changing needs of a program.

- Dynamic arrays are resizable data structures that can grow or shrink in size based on the program's needs, allowing for efficient memory management.
- They provide the benefits of random access like traditional arrays, enabling constant-time access to elements using indices.
- Dynamic arrays allocate contiguous memory blocks, and when the array size exceeds its capacity, a larger memory block is allocated, and elements are copied over, ensuring efficient insertion and deletion operations while maintaining order.

Stack

Stack - A stack is a Last-In-First-Out (LIFO) data structure that allows efficient insertion and removal of elements from one end, commonly used in scenarios involving function calls, memory management, and undo operations.

- A stack is a linear data structure that follows the Last-In-First-Out (LIFO) principle, where the last element added is the first one to be removed.
- It supports two primary operations, push, which adds an element to the top of the stack, and pop, which removes the top most element from the stack.
- Stacks are commonly used in tasks that require tracking function calls, managing memory, and undo operations, providing efficient insertion and deletion of elements from a single end.

Queue

Queue - A queue is a Firs-In-First-Out (FIFO) data structure that enables efficient insertion at one end and removal at the other, commonly used for managing processes, task scheduling, and breadth-first search algorithms.

- Queues follow the First-In-First-Out (FIFO) principle, ensuring that the element inserted first is the first one to be removed.
- They support two primary operations: enqueue, which adds an element to the rear of the queue, and dequeue, which removes the element from the front.
- Queues are frequently utilized for process management, task scheduling, and breadth-first search algorithms, as they maintain the order of elements and provide efficient insertion and removal at both ends.

Deque

Deque - A deque (double-ended queue) is a data structure that allows efficient insertion and removal of elements at both ends, providing flexibility in managing data from the front or the rear.

- Deques support insertion and removal of elements at both ends, allowing for efficient operations at the front and rear of the data structure.
- They provide flexibility in managing data by enabling operations like push and pop at both ends, as well as accessing elements from either end.
- Deques are useful in scenarios where elements need to be added or removed from both ends, such as implementing algorithms like breadth-first search, implementing a queue with additional functionalities, or managing a sliding window in algorithms like dynamic programming.

Bag

Bag - A bad, also known as a multiset or a collection, is an unordered data structure that allows storing multiple occurrences of elements, providing efficient insertion and retrieval operations.

- Bags allow for the insertion of elements without enforcing any particular order, making them suitable for scenarios where maintaining the order is not necessary.
- Unlike other data structures, bags can store duplicate elements, allowing for multiple occurrences of the same item.
- Bags are commonly used when it is important to count or track the frequency of elements, such as in data analytics text processing, or certain types of machine learning algorithms.

Set

Set - A set is an unordered data structure that stores a collection of unique elements, providing efficient membership testing and set operations.

- Sets contain only unique elements, ensuring that duplicates are automatically removed, making the suitable for tasks that require uniqueness, such as maintaining a distinct list of items.
- Sets provide efficient membership testing, allowing for quick checks to determine if an element is present or absent.
- Sets support common set operations like union, intersection, and difference, enabling efficient manipulation and comparison of multiple sets, often used in tasks like data deduplication, finding common elements, or checking for similarities across multiple datasets.

Priority Queue

Priority Queue - A priority queue is an abstract data type that stores elements with associated priorities, allowing efficient retrieval of the highest priority element.

- Priority queues store elements with priorities, where the element with the highest priority can be efficiently retrieved.
- Elements in a priority queue are typically ordered on their priority, allowing for operations such as insertion and removal according to their priority level.
- Priority queues are commonly used in various applications like task scheduling, event-driven simulations, graph algorithms, and data compression, where efficient handling of elements based on their priority is essential for optimizing performance.

Dictionary (Map)

Dictionary (Map) - A dicitionary, also known as a map or associative array, is a data structure that stores key-value pairs, providing efficient lookup and retrieval of values based on their associated keys.

- Dicitionaries store key-value pairs, allowing efficient retrieval of values based on their associated keys.
- Keys in a dictionary are unique, enabling fast and direct access to the corresponding values.
- Dicitionaries are commonly used in situations that require fast lookup, such as data indexing, caching, symbol tables, and implementing algorithms like graph traversal or dynamic programming.

Lists provide ordered collections of elements with efficient insertion, deletion, and retrieval operations. They offer flexibility in managing data and are widely used in various applications that require maintaining a specific order. Dynamic arrays, on the other hand, offer resizable storage that adjusts to the needs of the program. They provide random access to elements and efficient memory management by reallocating memory blocks as the array size changes.

Stacks adhere to the Last-In-First-Out (LIFO) principle and are commonly used for tracking function calls, managing memory, and implementing undo operations. They offer efficient insertion and removal of elements from one end. Queues, on the other hand, follow the Firs-In-First-Out (FIFO) principle. They are employed for process management, task scheduling, and breadth-first search algorithms. Queues enable efficient insertion at one end and removal at the other.

Bags are a type of data structure that stores unordered elements. They allow duplicates and enable frequency tracking. Bags are useful in scenarios where maintaining a distinct collection of items is not necessary, but counting occurrences or tracking frequency is essential. Sets, on the other hand, maintain unique elements. They provide efficient membership testing and support common set operations like union, intersection, and difference. Sets are utilized in various applications that require distinct elements and set manipulation.

Priority queues are data structures that store elements with associated priorities. They allow efficient retrieval of the highest priority element. Priority queues are commonly used in tasks like task scheduling, event-driven simulations, and graph algorithms. Finally, dictionaries (maps) store key-value pairs and provide efficient lookup and retrieval based on keys. They are extensively used for data indexing, symbol tables, and implementing algorithms that require fast access to values based on their associated keys.

Sec. 1.3 - Applications of ADTs

Abstract Data Types (ADTs) find applications across various domains, offering versatile solutions to address computational challenges. One common application of ADTs is in data storage and retrieval. ADTs like lists, arrays, and dictionaries (maps) provide flexible structures that enable efficient organization and access to data with different requirements for ordering, uniqueness, or key-value associations. These ADTs are used in databases, file systems, and data-driven applications to store and retrieve information in a structured and optimized manner.

ADTs play a crucial role in algorithm design. They are fundamental in solving computational problems efficiently. Stacks and queues, for example, are essential for managing program flow and data manipulation. They are used in areas such as compiler design, expression evaluation, and depth-first or breadth-first traversals. Priority queues are particularly useful in optimization algorithms and event-driven simulations, where elements with associated priorities need to be processed in a specific order.

Memory management in programming languages relies on ADTs for efficient memory allocation and deallocation. Dynamic arrays, for instance, are used to dynamically allocate and resize memory blocks as needed. Stacks are instrumental in tracking function calls and managing runtime memory, ensuring efficient resource utilization. ADTs help manage memory effectively, preventing issues like memory leaks or excessive memory fragmentation.

ADTs have applications in various fields, including simulation modeling, task scheduling, and graph manipulation. Simulation models often rely on ADTs for modeling and analyzing complex systems. Queues are used for process scheduling, while bags and sets assist in statistical analysis, data sampling, and randomness generation. In task scheduling, ADTs like queues help manage process execution and prioritize tasks based on their priority levels or time constraints. In graph manipulation and network analysis, ADTs such as dictionaries (maps) provide efficient storage and retrieval of graph elements and properties, while priority queues can aid in

graph algorithms like Dijkstra's algorithm for finding the shortest path.