1.1 Data structures

Data structures

A ***data structure*** is a way of organizing, storing, and performing operations on data. Operations performed on a data structure include accessing or updating stored data, searching for specific data, inserting new data, and removing data. The following provides a list of basic data structures.

Table 1.1.1: Basic data structures.

Data structure	Description
Record	A ***record*** is the data structure that stores subitems, often called fields, with a name associated with each subitem.
Array	An ***array*** is a data structure that stores an ordered list of items, where each item is directly accessible by a positional index.
Linked list	A ***linked list*** is a data structure that stores an ordered list of items in nodes, where each node stores data and has a pointer to the next node.
Binary tree	A ***binary tree*** is a data structure in which each node stores data and has up to two children, known as a left child and a right child.
Hash table	A ***hash table*** is a data structure that stores unordered items by mapping (or hashing) each item to a location in an array.
Heap	A ***max-heap*** is a tree that maintains the simple property that a node's key is greater than or equal to the node's childrens' keys. A ***min-heap*** is a tree that maintains the simple property that a node's key is less than or equal to the node's childrens' keys.
Graph	A ***graph*** is a data structure for representing connections among items, and consists of vertices connected by edges. A ***vertex*** represents an item in a graph. An ***edge*** represents a connection between two vertices in a graph.

PARTICIPATION ACTIVITY 1.1.1: Basic data structures.

1) A linked list stores items in an unspecified order.

- True
- False

2) A node in binary tree can have zero, one, or two children.

- True
- False

3) A list node's data can store a record with multiple subitems.

- True
- False

4) Items stored in an array can be accessed using a positional index.

- True
- False

Choosing data structures

The selection of data structures used in a program depends on both the type of data being stored and the operations the program may need to perform on that data. Choosing the best data structure often requires determining which data structure provides a good balance given expected uses. Ex: If a program requires fast insertion of new data, a linked list may be a better choice than an array.

PARTICIPATION ACTIVITY 1.1.2: A linked list avoids the shifting problem.

Animation content:

undefined

Animation captions:

1. Inserting an item at a specific location in an array requires making room for the item by shifting higher-indexed items.
2. Once the higher index items have been shifted, the new item can be inserted at the desired index.
3. To insert new item in a linked list, a list node for the new item is first created.

4. Item B's next pointer is assigned to point to item C. Item A's next pointer is updated to point to item B. No shifting of other items was required.

PARTICIPATION ACTIVITY 1.1.3: Basic data structures.

1) Inserting an item at the end of a 999-item array requires how many items to be shifted?

Check Show answer

2) Inserting an item at the end of a 999-item linked list requires how many items to be shifted?

Check Show answer

3) Inserting an item at the beginning of a 999-item array requires how many items to be shifted?

Check Show answer

4) Inserting an item at the beginning of a 999-item linked list requires how many items to be shifted?

Check Show answer

©zyBooks 05/24/23 23:21 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023

1.2 Abstract data types

Abstract data types (ADTs)

An ***abstract data type*** (***ADT***) is a data type described by predefined user operations, such as "insert data at rear," without indicating how each operation is implemented. An ADT can be implemented using different underlying data structures. However, a programmer need not have knowledge of the underlying implementation to use an ADT.

Ex: A list is a common ADT for holding ordered data, having operations like append a data item, remove a data item, search whether a data item exists, and print the list. A list ADT is commonly implemented using arrays or linked list data structures.

PARTICIPATION ACTIVITY 1.2.1: List ADT using array and linked lists data structures.

Animation captions:

1. A new list named agesList is created. Items can be appended to the list. The items are ordered.
2. Printing the list prints the items in order.
3. A list ADT is commonly implemented using array and linked list data structures. But, a programmer need not have knowledge of which data structure is used to use the list ADT.

PARTICIPATION ACTIVITY 1.2.2: Abstract data types.

1) Starting with an empty list, what is the list contents after the following operations?

   Append(list, 11)
   Append(list, 4)
   Append(list, 7)

   - 4, 7, 11
   - 7, 4, 11
   - 11, 4, 7

2) A remove operation for a list ADT removes the specified item. Given a list with contents: 2, 20, 30, what is the list contents after the following operation?

   Remove(list, item 2)

   - 2, 30

- 2, 20, 30
- 20, 30

3) A programmer must know the underlying implementation of the list ADT in order to use a list.

- True
- False

4) A list ADT's underlying data structure has no impact on the program's execution.

- True
- False

Common ADTs

Table 1.2.1: Common ADTs.

Abstract data type	Description	Common underlying data structures
List	A ***list*** is an ADT for holding ordered data.	Array, linked list
Dynamic array	A ***dynamic array*** is an ADT for holding ordered data and allowing indexed access.	Array
Stack	A ***stack*** is an ADT in which items are only inserted on or removed from the top of a stack.	Linked list
Queue	A ***queue*** is an ADT in which items are inserted at the end of the queue and removed from the front of the queue.	Linked list
Deque	A ***deque*** (pronounced "deck" and short for double-ended queue) is an ADT in which items can be inserted and removed at both the front and back.	Linked list
Bag	A ***bag*** is an ADT for storing items in which the order does not matter and duplicate items are allowed.	Array, linked list

Set	A ***set*** is an ADT for a collection of distinct items.	Binary search tree, hash table
Priority queue	A ***priority queue*** is a queue where each item has a priority, and items with higher priority are closer to the front of the queue than items with lower priority.	Heap
Dictionary (Map)	A ***dictionary*** is an ADT that associates (or maps) keys with values.	Hash table, binary search tree

PARTICIPATION ACTIVITY 1.2.3: Common ADTs.

Consider the ADTs listed in the table above. Match the ADT with the description of the order and uniqueness of items in the ADT.

If unable to drag and drop, refresh the page.

Bag **Set** **Priority queue** **List**

	Items are ordered based on how items are added. Duplicate items are allowed.
	Items are not ordered. Duplicate items are not allowed.
	Items are ordered based on items' priority. Duplicate items are allowed.
	Items are not ordered. Duplicate items are allowed.

Reset

1.3 Applications of ADTs

Abstraction and optimization

Abstraction means to have a user interact with an item at a high-level, with lower-level internal details hidden from the user. ADTs support abstraction by hiding the underlying implementation details and providing a well-defined set of operations for using the ADT.

Using abstract data types enables programmers or algorithm designers to focus on higher-level operations and algorithms, thus improving programmer efficiency. However, knowledge of the underlying implementation is needed to analyze or improve the runtime efficiency.

PARTICIPATION ACTIVITY 1.3.1: Programming using ADTs.

Animation content:

undefined

Animation captions:

1. Abstraction simplifies programming. ADTs allow programmers to focus on choosing which ADTs best match a program's needs.
2. Both the List and Queue ADTs support efficient interfaces for removing items from one end (removing oldest entry) and adding items to the other end (adding new entries).
3. The list ADT supports iterating through list contents in reverse order, but the queue ADT does not.
4. To use the List (or Queue) ADT, the programmer does not need to know the List's underlying implementation.

PARTICIPATION ACTIVITY 1.3.2: Programming with ADTs.

Consider the example in the animation above.

1) The _____ ADT is the better match for the program's requirements.
   - ○ queue
   - ○ list

2) The list ADT _____.
   - ○ can only be implemented using an array
   - ○ can only be implemented using a linked list

- can be implemented in numerous ways

3) Knowledge of an ADT's underlying implementation is needed to analyze the runtime efficiency.
   - True
   - False

ADTs in standard libraries

Most programming languages provide standard libraries that implement common abstract data types. Some languages allow programmers to choose the underlying data structure used for the ADTs. Other programming languages may use a specific data structure to implement each ADT, or may automatically choose the underlying data-structure.

Table 1.3.1: Standard libraries in various programming languages.

Programming language	Library	Common supported ADTs
Python	Python standard library	list, set, dict, deque
C++	Standard template library (STL)	vector, list, deque, queue, stack, set, map
Java	Java collections framework (JCF)	Collection, Set, List, Map, Queue, Deque

PARTICIPATION ACTIVITY 1.3.3: ADTs in standard libraries.

1) Python, C++, and Java all provide built-in support for a deque ADT.
   - True
   - False

2) The underlying data structure for a list data structure is the same for all programming languages.

- True
- False

3) ADTs are only supported in standard libraries.

- True
- False

©zyBooks 05/24/23 23:21 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023