**Arrays and Linked Lists**

viernes, 21 de abril de 2023

1:20 p. m.

Programiz Arrays: <https://www.programiz.com/python-programming/array>

Illustrated Data Series - roadmap.io | Arrays: [Array Data Structure | Illustrated Data Structures](https://www.youtube.com/watch?v=QJNwK2uJyGs&list=PLkZYeFmDuaN2-KUIv-mvbjfKszIGJ4FaY&index=3&t=4s)

Geek for geeks - Arrays: <https://www.geeksforgeeks.org/python-arrays/>

Edureka: <https://www.edureka.co/blog/arrays-in-python/>

Freecodecamp Arrays: <https://www.freecodecamp.org/news/python-array-tutorial-define-index-methods/>

Geek for geeks - Linked list vs Arrays: <https://www.geeksforgeeks.org/linked-list-vs-array/>

Illustrated Data Series - roadmap.io | Linked lists: [Linked List Data Structure | Illustrated Data Structures](https://www.youtube.com/watch?v=odW9FU8jPRQ)

Programiz - Linked lists: <https://www.programiz.com/dsa/linked-list>

Arrays

Programiz Arrays: <https://www.programiz.com/python-programming/array>

Illustrated Data Series - roadmap.io | Arrays: [Array Data Structure | Illustrated Data Structures](https://www.youtube.com/watch?v=QJNwK2uJyGs&list=PLkZYeFmDuaN2-KUIv-mvbjfKszIGJ4FaY&index=3&t=4s)

Geek for geeks - Arrays: <https://www.geeksforgeeks.org/python-arrays/>

Edureka: <https://www.edureka.co/blog/arrays-in-python/>

Freecodecamp Arrays: <https://www.freecodecamp.org/news/python-array-tutorial-define-index-methods/>

A data structure that is a collection of elements following this characteristics:

1. All the data stored in arrays must have the same type.

1. All elements of the Array are stored in adjacent memory location, allowing constant-time i.e. O(1) access to any element base on index.

1. The Arrays has fixed size which declared upon creation and can't change dynamically.

1. Arrays indexing is zero-based.

Why sized-fixed? Arrays have fixed sizes is because for memory optimization it reserves only what is required for the data initially passed and other programs or functions could be using whatever memory locations, so if a new element would be added to the Array it may or may not have a conflict with other elements, and on the other hand if more than initially required space is reserved for the array, it will be space that may or may not be used, then a memory inefficiency will raise, so that's why they're fixed in size.

And why same type? Arrays has to have same type of data intrinsically for the same reason of size, because different data types use different memory space, for example ints use 2 bytes for 1 int, and booleans only 1. So, if different data types are indexed there's no way to efficiently allocate enough space in memory.

Array Operations

Algorithmic Complexity: Time Complexity (TC) / Space or Memory Complexity (SC)

* Element accessing: reading the value of the array on a certain index - TC: O(1) / SC: O(n)

* Element Inserting: works the same as *list*.insert(*index*, *item*) - TC: O(n) / SC: O(n)

* Element Removing: works the same as the del *list*[index] statement - TC: O(n) / SC: O(n)

* Element Updating: same as assigning items to a list by index - TC: O(n) / SC: O(1)

* Array Searching: To look up within the array - TC: O(n) / SC: O(n)

Note: The above is true for certain languages, but for Python and JavaScript the mixed type storing and the dynamic size are implemented.

Now, if the module Array is used in python, the same type restriction is applied.

In Python there is no need for implementation of this data structure, depending on the case, the module or the built-in list would serve as Arrays implementations.

Python Array Module

Source: <https://docs.python.org/es/3/library/array.html>

Here the array module is to be imported and the only thing to consider different from List management is that the type of the data stored must be declared when creting the array following this characters:

1. 'b': Represents signed integers of size 1 byte (-128 to 127).
2. 'B': Represents unsigned integers of size 1 byte (0 to 255).
3. 'h': Represents signed integers of size 2 bytes (-32,768 to 32,767).
4. 'H': Represents unsigned integers of size 2 bytes (0 to 65,535).
5. 'i': Represents signed integers of size 4 bytes (-2,147,483,648 to 2,147,483,647).
6. 'I': Represents unsigned integers of size 4 bytes (0 to 4,294,967,295).
7. 'l': Represents signed integers of size 4 bytes (same as 'i').
8. 'L': Represents unsigned integers of size 4 bytes (same as 'I').
9. 'f': Represents floating-point numbers of size 4 bytes.
10. 'd': Represents floating-point numbers of size 8 bytes.
11. 'c': Represents characters of size 1 byte.
12. 's': Represents strings of variable length (each character is 1 byte).
13. 'u': Represents Unicode characters of size 2 bytes.

Array Use Cases

Aside of the usual data storing or numerical computations (for which python Arrays are not optimized but still functional in simple math calculation) and data storing, there are some couple other implementation of Arrays like:

1. Algorithms and Data Structures: For example implementing Stacks, Queues and Hash Tables, or for implementing sorting and searching algorithms.

1. Image processing and computer vision: Arrays can be used to represent and manipulate images in computer vision applications. Each pixel in the image can be stored as an element in a multidimensional array. Libraries like OpenCV provide powerful tools for image processing and analysis.

Linked Lists

Illustrated Data Series - roadmap.io | Linked lists: [Linked List Data Structure | Illustrated Data Structures](https://www.youtube.com/watch?v=odW9FU8jPRQ)

Programiz - Linked lists: <https://www.programiz.com/dsa/linked-list>

A linear data structure that is different from Arrays in the way that data does not has to be stored contiguously nor has to be same type data, and nor even has to have a fixed sized.

This data structure consists in a connected objects called Nodes which are the ones that contains the data, being the first of the list the 'head' (having no preceding node) and being the last of the list the 'tail' (having no succeeding node).

The Nodes are essentially objects with two (and on a very specific case: doubly linked lists, three) attributes: The actual data stored and the pointer of the next node, except for the tail node, which has the pointer of the next node is set to None.

Linked Lists Operations

* Element accessing / Search / Updating: To access an element in a linked list traversing the whole list until finding the element is required, hence, the same process of searching. The process would be checking one by one each node would be the process to access information. - TC: O(n) / SC: O(n).

* Element Inserting: Is just matter of pointing the next node for the inserted one, and if it's gonna be the tail, just set to None, and being careful to have the preceding node be pointing to the inserted one, or just leave it as it is if it's gonna be the head, it'd only be matter of reassigning the head to the new node and point it to the former head. - TC: O(1) / SC: O(n).

* Element Removing: To remove a node, in the Head case, is only necessary to reassign the head to the second node; and in the case of the tail, is just to point the second last node to None. In any other case the preceding node must be pointed to the succeeding node of the erased one. - TC: O(1) / SC: O(n).

Types of Linked Lists

* Singly Linked List: Nodes with only two attributes - Data & Next Node. / The tail next is None.

HEAD 
NULL 
DATA NEXT 
DATA NEXT 
NEXT 
ona NEXT 
DATA NEXT 

* Doubly Linked List: Nodes with three attributes - Preceding Node, Data & Next Node. / The head preceding is None, as well as tail next is None.

Link: <https://www.programiz.com/dsa/doubly-linked-list>

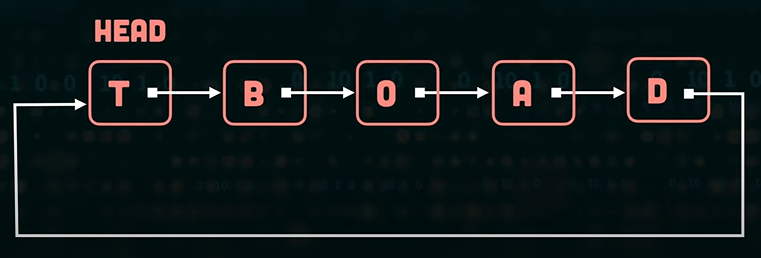
م-م-9بSي 
11M 
Ob3H 

Doubly Linked List Applications

* Redo and undo functionality in software.
* Forward and backward navigation in browsers.
* For navigation systems where forward and backward navigation is required.

* Circular Linked List: This one is a singly linked list with the difference that the tail's next node point to the head node.

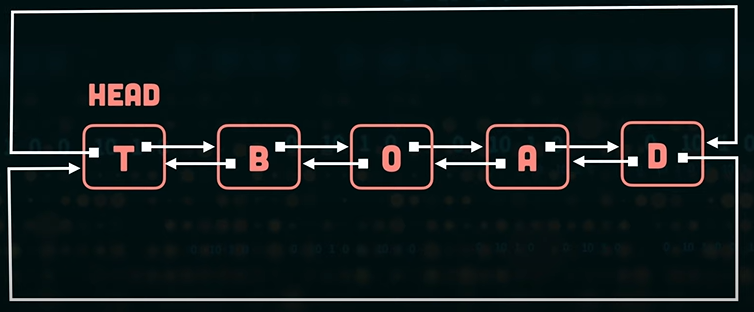
Link: <https://www.programiz.com/dsa/circular-linked-list>



Circular Linked List Applications

* It is used in multiplayer games to give a chance to each player to play the game.
* Multiple running applications can be placed in a circular linked list on an operating system. The os keeps on iterating over these applications.

* Circular Doubly Linked List: This is a doubly linked list but the head's preceding node is the tail and the tail's next node is the head.



Linked Lists Use Cases

1. Implementing Data Structures: For example implementing Stacks, Queues and Hash Tables.

1. Memory management: Linked lists can be used in memory management systems to keep track of allocated and deallocated memory blocks.

1. File systems: Linked lists are utilized in file systems to represent directories and files. Each node in the linked list can correspond to a file or directory and contain information about the name, size, permissions, and a reference to the next node in the list.

1. Music and Video Playlist: Linked lists can be used to create playlists in music or video players. Each node in the linked list represents a song or video and contains information about the media file and a reference to the next node in the playlist. This allows for sequential playback and easy insertion or removal of songs/videos.

1. Undo/Redo Functionality: Linked lists can be used to implement undo/redo functionality in applications. Each node in the linked list represents a state or action, and by traversing the list backward or forward, the application can undo or redo previous actions.

1. Graphs and Trees: Linked lists can be used to represent adjacency lists in graph and tree data structures. Each node in the linked list represents a neighboring vertex or child node, allowing for efficient graph traversal and manipulation.

Arrays vs Linked Lists / Use Cases

Geek for geeks - Linked list vs Arrays: <https://www.geeksforgeeks.org/linked-list-vs-array/>

Tabla

Descripción generada automáticamente con confianza media