Array

Array is by far **the most widely used data structure** in computer science. Why? Because its structure is easy to understand and it comes built-in with most of the programming languages.

What it is? A data structure of multiple elements each defined with an index. Consecutive indexes are placed in a consecutive way in memory. Thus by knowing an index of an element you can access it directly.

Why there are other data structures, than arrays? Yeah, arrays are great, who need something else? Well, arrays are great indeed, but there are cases where they are not optimal. Arrays require certain amount of memory and linked lists could be much more effective for instance.

Tips

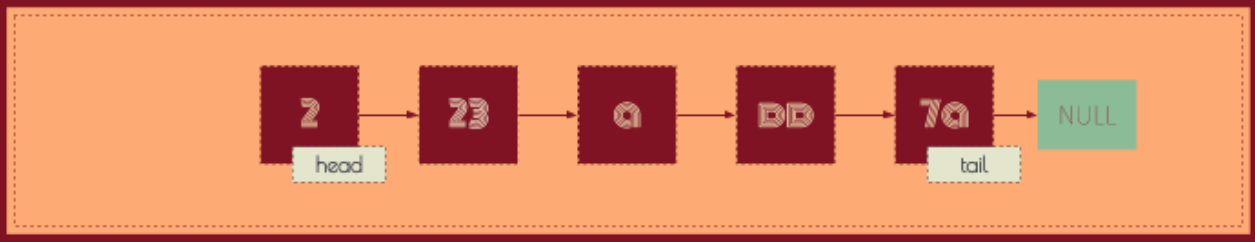
Don't rely only on arrays. Yep, they are easy to use, but with the application growing will grow its memory fingerprint

Pros & Cons

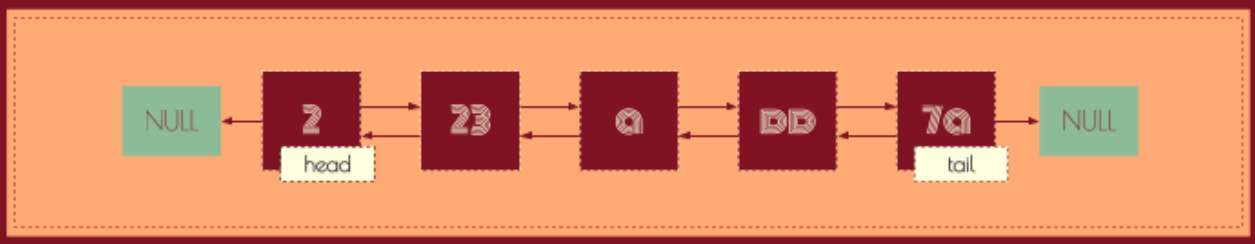
PROS: direct access to memory

CONS: Require fixed amount of memory and it's by far the most memory-optimal solution

linked list



In single linked lists (above) the elements are not next to each other in the memory and each item holds a pointer to the next. Direct access to elements is not possible. The last (tail) points to NULL to indicate the end. Element access is done by walking through the pointers of the elements preceding it.



The double-linked list (above) is facilitating walk through the elements in both forward and backward directions. Linked lists could be very handy for entire class of software problems and are a very basic private case of another widely used data structure: trees!

tips

If you're often inserting, deleting or reversing the order of a list better use linked lists than arrays.
When searching in a non-sorted list is often performed - your data structure of choice should be an array!

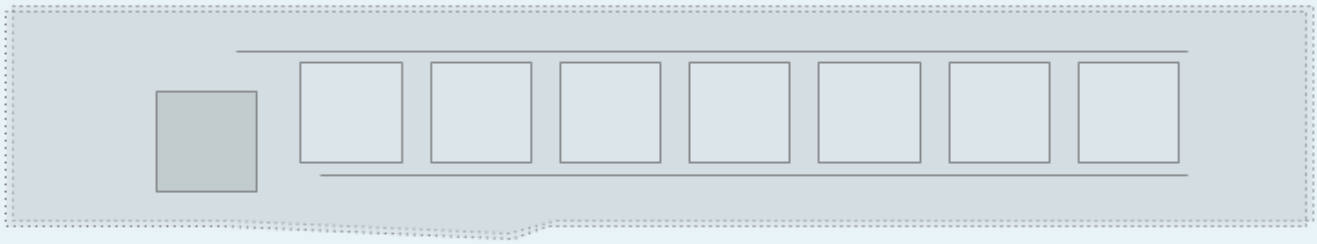
pros & cons

PROS: More memory efficient than arrays and very helpful in certain type of problems i.e. reversing a list or merging two lists.

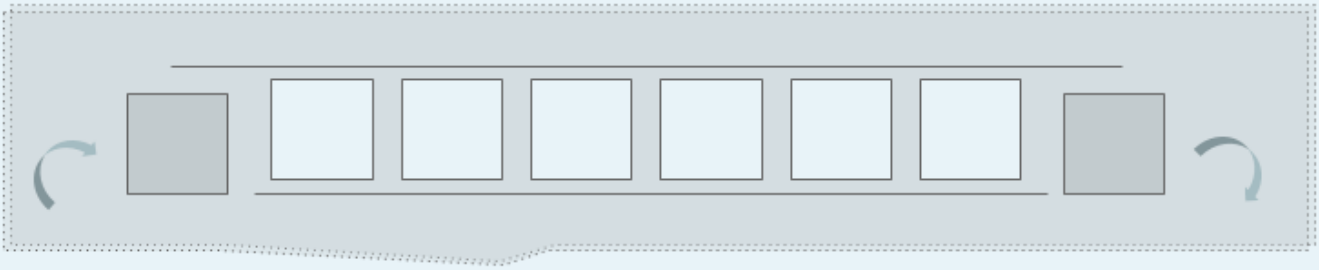
BAD: Lists lack the big advantage of arrays - the direct access to the elements by a given index, thus searching in a list could be quite time consuming.

data structures

stack & queue



The stack data structure models the real-world stack being an abstract implementation and not physical. You can think of it as stack of boxes one above the other. Thus the only way to put another item into the stack is to put it above all other items (on its top). This operation is often called “push”. In the other hand taking an item from the stack is called pop, and also only the highest item can be “popped”.



As mentioned above the queue is somehow related to the stack data structure. However it follows a different principle – FIFO (First In First Out), which means that the item that has been in the queue for the longest time is retrieved first.

tips

Queues and stacks are abstract terms, not physical. Thus implementation could be either a linked list or an array. Normally it’s implemented as a linked list, because of its memory efficiency. However there is no restriction to implement it as an array where there are already most of the functions like pop or push.

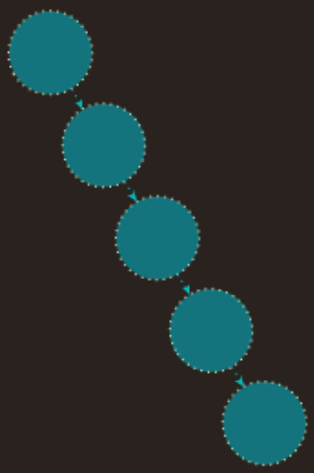
pros & cons

Can’t just talk on pros and cons. Specific software cases need a usage of stacks and queues, so ... go for it!

Data structures

tree

A tree is ... a tree, a data structure much like a natural tree with a root, branches and leaves. There are no cycles - to each leaf there's *only one path* from the root. Each leaf has *only one parent* unlike graphs. Widely used for indexing large data and very handy for fast search and discovery trees are among the most popular data structures.

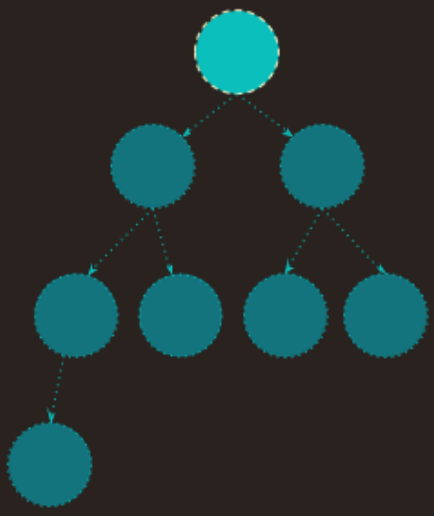


Is this a tree? Or a linked list?!

Well, the single linked list could be as well a valid tree, since every element has only one parent. In practice this is always considered a linked list.

Only the root doesn't have a parent.

There are NO cycles in trees and a tree without nodes is called empty.



Binary, quad or ..

Each element can have as many children as possible - typically two for binary trees, but not necessarily.

tip

The most common implementation of a tree - the binary tree in many cases might not be the most optimal. There are many ***different implementations*** among which quad tree, red-black tree, B-Tree, R-Tree, trie and suffix tree. The choice must be guided by exploration of the data.

application

PROS: Very cool for searching

BAD: Depending on the type of tree some implementation might be hard. Should not be overused and must be carefully applied to certain class of problems and data.