Introduction to Custom Data Structures

Let’s go over the Custom Data Structures pattern, its real-world applications, and some problems we can solve with it.

We'll cover the following...

About the pattern

Although many coding problems can be solved using existing data structures like arrays, linked lists, stacks, queues, trees, and hash tables, sometimes these structures may not perfectly fit the requirements of a given problem or may not provide the desired efficiency. This is where we need custom data structures. These structures can be implemented using basic data structures as building blocks and incorporate unique features or behaviors specific to the problem domain. In easier words, a custom data structure is the modified version of an existing data structure.

For example, we have to build a web crawler. It starts with a set of seed URLs, visits each page, finds links on each page, and then follows those links to find more pages. Crawling the web means dealing with lots of pages and handling many URLs. Additionally, storing and managing the URLs efficiently while ensuring uniqueness and prioritizing certain pages (e.g., based on relevance or importance) are critical challenges. Basic data structures like arrays or hash tables might not be sufficient to handle the scale and complexity of the web. To address the challenges of this task, a custom data structure, such as a URL queue, can be designed to

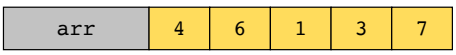
manage the URLs to be crawled. It is responsible for maintaining a queue of URLs to visit, ensuring uniqueness, and potentially prioritizing URLs based on various criteria.

Using custom data structures makes it easier and more efficient to solve problems that would otherwise be difficult with the existing data structures.

Each custom data structure can be effectively implemented as a class in programming languages like Python, Java, or C++. Classes facilitate abstraction, enabling users to interact with the data structure through well-defined methods and properties without needing to understand the underlying implementation details. Moreover, custom data structures represented as classes allow code reuse.

The illustration below shows some commonly used data structures that can be used to make a custom data structure:

Array



Properties

The most significant properties of arrays are:

1. Arrays store a fixed-size collection of elements of the same data type.
2. The elements in an array are stored in a contiguous block of memory.
3. Due to these two properties, the elements in the array can be accessed by index in O(1) and are iterable.

1 of

Linked list

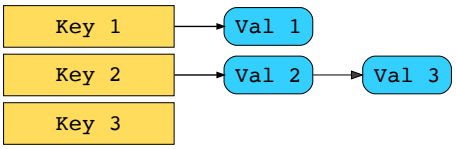
Properties

The most significant properties of a linked list are:

1. Linked lists are used to implement stacks, queues, graphs, etc.
2. Linked lists are dynamic in nature – memory is only allocated when needed.
3. As the addresses of linked list nodes cannot be calculated directly, random access by index is not supported. Instead, accessing an element takes O(n) time.

2 of 4

Hash map



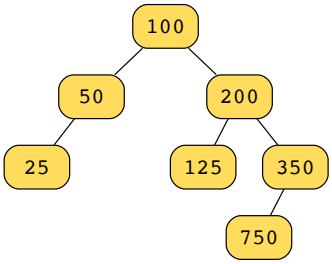
Properties

The most significant properties of hash maps are:

1. Access by key, insertion and deletion take on average O(1) time.
2. Hash maps store data in the form of key-value pairs.
3. Unlike arrays, hash maps can store values of more than one data type.

3 of 4

Trees



Properties

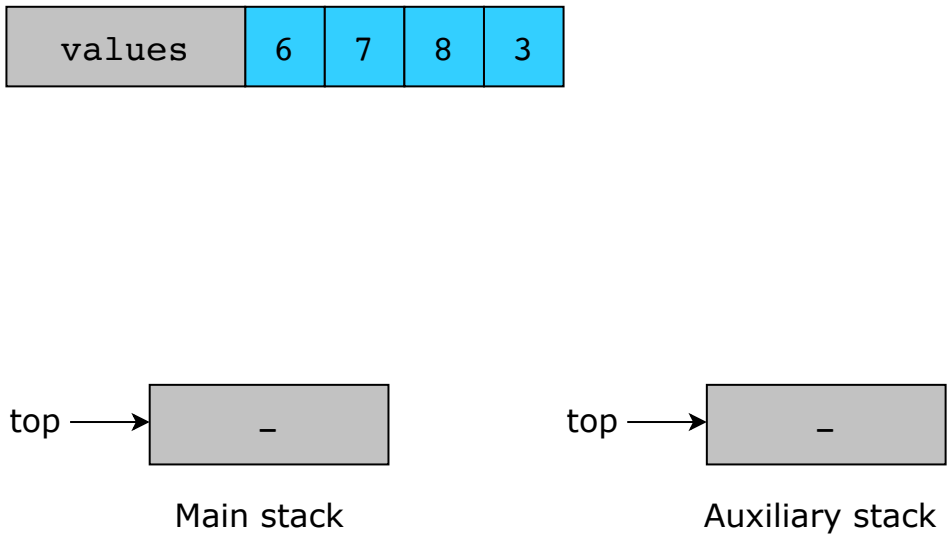
The most significant properties of trees are:

1. Trees can be seen as undirected graphs with no cycles.
2. In a tree, every node has a parent, except the root node.
3. A node without children is called a leaf node.
4. If the parent node has at most two children, it is a binary tree as shown in the diagram. A tree can have more than two children, such as an n-ary tree.
5. Due to their hierarchical structure, trees naturally support recursion.
6. The most commonly used traversals through trees are breadth-first search and depth-first search.
7. Powerful and widely used subtypes include: binary search trees, heaps, tries and n-ary trees.

Examples

The following examples illustrates a problem that can be solved with this approach:

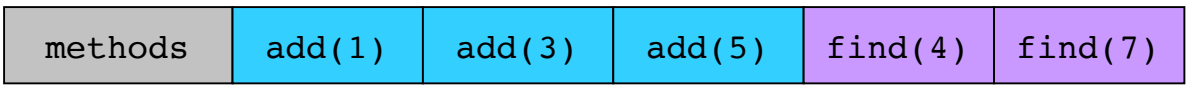
1. **Custom stack with getMin() in O(1) complexity: `getMin()` in O(1) complexity:** Design a stack data structure to retrieve the minimum value in O(1) time.



Let's add the following stream of values to the stack while maintaining the min value.

2. **Two sum III:** Design a data structure that accepts a stream of integers and checks if it has a pair of integers that sum up to a particular value. This data structure will have two main methods:

- `add(number)`: Adds `number` to the data structure.
- `find(value)`: Returns TRUE if there exists any pair of numbers whose sum is equal to `value`, otherwise, it returns FALSE.



We'll use hash map to create this data structure.

For **add()** method, the relevant number will be added to the stack along with its count.

For **find()** method, a compliment for each value in hash map is calculated to see if any pair exists whose sum equals the given value.

1 of 8

Does your problem match this pattern?

Yes, if either of these conditions is fulfilled:

- **Modification of an existing data structure:** The problem requires customizing an existing data structure, that is, adding a feature to it or modifying an existing feature. Examples include min stack and maximum frequency stack.

- **Multiple data structures involved:** The problem requires combining one or more data structures to solve the problem efficiently. An example would be implementing a Least Recently Used (LRU) cache.

Real-world problems

Many problems in the real world share the custom data structure pattern. Let's look at some examples.

- **Video games:** By modifying/combining the standard data structures, we can maintain the state of the players, levels, and other relevant game details efficiently.
- **Customizing search engines:** Search engines use custom data structures such as customized trees and arrays to quickly search and display data.
- **Managing car parking:** A custom data structure can be utilized to allow efficient tracking of available parking spots, dynamically allocating spaces, and handling reservations and payments seamlessly in multi-level parking garages.

Strategy time!

Match the problems that can be solved using the custom data structures pattern.

> **Note:** Select a problem in the left-hand column by clicking it, and then click one of the two options in the right-hand column.

Match The Answer

Select an option from the left-hand side

Find the k^{th} largest element in an array.

Custom Data Structures

Implement arbitrary state persistence of multiple workers nodes to improve the fault tolerance of a distributed processing system.

Some other pattern

Enhance a stack to enable popping the highest value in O(1).

Detect a cycle in a linked list.

Reset | Show Solution | Submit

Back lesson | Mark As Completed | Next