### Programming Guide for Dynamic Array

#### Overview of Dynamic Array

A Dynamic Array is a data structure that provides a way to store elements in a contiguous block of memory. Unlike a static array, it can change size dynamically to accommodate the addition or removal of elements.

#### Key Operations

* **Add:** Insert an item at the end of the array.
* **Get:** Retrieve an item at a specified index.
* **Remove:** Delete an item from a specified index.
* **Resize:** Increase or decrease the array’s capacity as needed.

#### Pseudo Code for Dynamic Array

CLASS DynamicArray:  
 FUNCTION \_\_init\_\_():  
 Initialize an array with a predefined capacity  
 Set count to 0  
  
 FUNCTION add(item):  
 IF count equals capacity:  
 Resize the array to double its size  
 Add item to the array at the current count index  
 Increment the count  
  
 FUNCTION get(index):  
 IF index is within the bounds of the array:  
 RETURN the item at the index  
 ELSE:  
 Raise an IndexError  
  
 FUNCTION remove(index):  
 IF index is within the bounds of the array:  
 FOR each element starting from index to end:  
 Shift elements to the left  
 Decrement the count  
 IF count is much smaller than capacity:  
 Resize the array to half its size  
 ELSE:  
 Raise an IndexError  
  
 FUNCTION \_resize(new\_capacity):  
 Create a new array with new\_capacity  
 Copy elements from the old array to the new one  
 Update the array reference and capacity

#### Implementation Tips

* Initialize the array with a fixed initial capacity.
* Implement a private \_resize method that creates a new array with the desired capacity and copies elements from the old array to the new one.
* In the add method, if the array is full (i.e., count equals capacity), call \_resize to double the size of the array before adding the new element.
* In the remove method, after removing an element, if the number of elements is much less than the capacity (e.g., a quarter of the capacity), reduce the size of the array by half to save space.
* Always check for index out-of-bounds in the get and remove methods.

#### Applications

* **Resizable storage:** Dynamic arrays are used in languages like Python, Java, and C++ (as vectors) to provide resizable array-like storage.
* **Building blocks:** They are foundational for higher-level data structures like strings, stacks, queues, and arraylists.
* **Space efficiency:** By resizing, dynamic arrays use memory efficiently and offer flexibility for various applications that require array-like storage.

### Programming Guide for Specialized Data Structures

#### 1. Disjoint Set (Union-Find)

**Overview:** A Disjoint Set data structure, also known as Union-Find, is used to keep track of a set of elements partitioned into a number of disjoint (non-overlapping) subsets. It’s particularly useful in graph algorithms to determine whether a graph contains a cycle.

**Key Operations:** - **Find:** Determine which subset a particular element is in. Often used for determining if two elements are in the same subset. - **Union:** Join two subsets into a single subset.

**Pseudo Code for Disjoint Set:**

CLASS DisjointSet:  
 FUNCTION \_\_init\_\_(size):  
 Initialize an array where each element's parent is itself  
  
 FUNCTION find(x):  
 IF x is not the parent of itself:  
 Recursively find and update the parent of x  
 RETURN the parent of x  
  
 FUNCTION union(x, y):  
 Find the root of x and y  
 Set the root of x as the root of y (or vice versa)

#### 2. Bloom Filter

**Overview:** A Bloom Filter is a space-efficient probabilistic data structure that is used to test whether an element is a member of a set. It can lead to false positives but never false negatives.

**Key Operations:** - **Add:** Add an item to the Bloom Filter. - **Check:** Check if an item is in the Bloom Filter.

**Pseudo Code for Bloom Filter:**

CLASS BloomFilter:  
 FUNCTION \_\_init\_\_(size):  
 Initialize a bit array of given size  
 Choose suitable hash functions  
  
 FUNCTION add(item):  
 FOR each hash function:  
 Compute the hash of the item  
 Set the bit at the hash index in the bit array  
  
 FUNCTION check(item):  
 FOR each hash function:  
 Compute the hash of the item  
 IF the bit at the hash index is not set:  
 RETURN False (item not present)  
 RETURN True (item possibly present)

#### Applications and Integration into Projects

* **Disjoint Set:**
  + **Graph Algorithms:** Efficiently used in algorithms like Kruskal’s for finding Minimum Spanning Tree.
  + **Network Connectivity:** Determines whether points in a network are connected.
* **Bloom Filter:**
  + **Database Indexing:** Quickly check if a record exists without storing the whole dataset.
  + **Web Caching:** Web browsers use Bloom filters to check if a URL is in a blacklist.
  + **Networking:** For quick lookups in routers and switches.

#### Project Integration Insights

* When integrating these structures into projects, focus on their specific advantages: Disjoint Sets for efficient set operations and Bloom Filters for space-efficient presence checking.
* Analyze the trade-offs: Bloom Filters save space but introduce a probability of false positives.

During project design, consider how these structures impact the overall system performance, especially in scenarios requiring frequent membership tests or set unions.