Dearray

Introduction

Dearray is an innovative data structure that seamlessly combines the capabilities of both an array (vector) and a deque (double-ended queue). Dearray is a data structure inspired by a dequeue and an array. Also **Dearray requires 2 times the space required by Array (Vector) or Dequeue.**

Justification for the new data structure

Before coming towards the Dearray, we will first go through the need of a new data structure(Dearray),

An **array (vector)** exhibits **O(1) time complexity** for the following operations:

- Accessing an element through its index
- Updating an element an any index
- Erasing an element at the last index
- Inserting an element at the last index

However, the same **array (vector)** requires **O(n) time complexity** to execute the following two operations, where n represents the number of elements in the array:

- Erasing an element at the first index
- Inserting an element at the first index.

Similarly,
A **dequeue** exhibits **O(1) time complexity** for the following operations:

- Erasing an element at the last index
- Inserting an element at the last index
- Erasing an element at the first index
- Inserting an element at the first index.

However, the same **dequeue** requires **O(n) time complexity** to execute the following two operations, where n represents the number of elements in the dequeue:

- Accessing an element through its index
- Updating an element an any index

At present, there is no data structure available that can execute all the mentioned above operations in O(1) complexity. So here comes a new innovative data structure **(Dearray)** that

seamlessly combines the capabilities of both an **array (vector) and a deque (double-ended queue)**. **Dearray** can perform below mentioned operations in just **O(1) complexity.**

- Erasing an element at the last index
- Inserting an element at the last index
- Erasing an element at the first index
- Inserting an element at the first index.
- Accessing an element through its index
- Updating an element an any index
- Rotate the order of the elements

Basically **Dearray** is an upgraded version of Array (Vector) and Dequeue, which solves the problem of inefficiency in some operations of Array (Vector) and Dequeue.
Dearray can be a **perfect replacement** for all these mentioned below data structure with additional functionalities:

- **Array (vector)**
- **Stack**
- **Queue**
- **Dequeue**

Results

Comparison of Dearray with Array (vector) and Dequeue:

Performance test 1:
To perform **1000000 operations** from the given below operations in random order, where each operation have equal probability of getting chosen in the **i^{th} operation**:

- Erasing an element at the last index
- Inserting an element at the last index
- Erasing an element at the first index
- Inserting an element at the first index.
- Accessing an element through its index
- Updating an element an any index

Array (Vector) takes **17 sec** and **Dequeue** takes **more than 23 sec** in **performance test 1** whereas for the same performance test the **Dearray** takes **less than 1 sec** to perform the same **1000000 operations.**

Performance test 2:
To perform **1000000 operations** from the given below operations in random order, where each operation have equal probability of getting chosen in the **i^{th} operation**:

- Erasing an element at the last index
- Inserting an element at the last index
- Erasing an element at the first index
- Inserting an element at the first index.

Array (Vector) takes more than **21 sec** in **performance test 2** whereas for the same performance test the **Dequeue** and **Dearray** takes **less than 1 sec** to perform the same **1000000 operations.**

Performance test 2:
To perform **1000000 operations** from the given below operations in random order, where each operation have equal probability of getting chosen in the **i^{th} operation**:

- Erasing an element at the last index
- Inserting an element at the last index
- Accessing an element through its index
- Updating an element an any index

Dequeue takes more than **30 sec** in **performance test 2** whereas for the same performance test the **Array (Vector)** and **Dearray** takes **less than 1 sec** to perform the same **1000000 operations.**

Result Table

Data Structure	Array (Vector)	Dequeue	Dearray
Performance test 1	17 sec	23 sec	Less than 1 sec
Performance test 2	21 sec	Less than 1 sec	Less than 1 sec
Performance test 3	Less than 1 sec	30 sec	Less than 1 sec

Conclusion

Based on the 3 performance test we conclude that (the new data structure) **Dearray is the best option among Array (Vector), Dequeue and Dearray** for performing these above operations in terms of time complexity. However **Dearray requires 2 times the space required by Array (Vector) or Dequeue.**