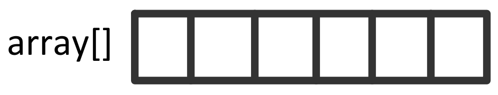
Static array

Friday, April 29, 2022 12:26 AM

Static array - Data structure & algorithms



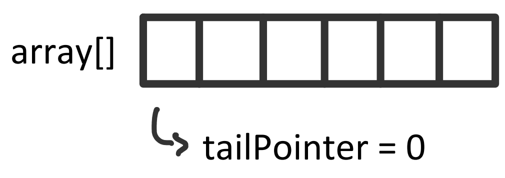
Initial length/size = 6 Initial tailPointer = -1

Time complexity:

- **InsertAtHead() -** O(n) linear-time
- **RemoveFromHead() -** O(n) linear-time
- **InsertAtTail() -** O(1) constant-time
- **RemoveFromTail() -** O(1) constant-time

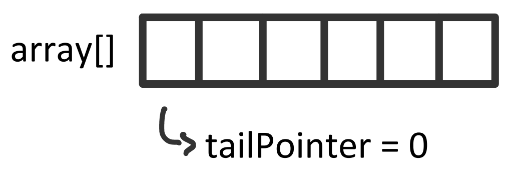
- **Shifting right*****
  - If we have one element in our array, we will simply swap the first two elements (initial shift).
  - Otherwise, perform shifting algorithm.
- **Shifting left*****
  - Shift elements to the left & set array[tailPointer] = 0.

Initial state (initialized)

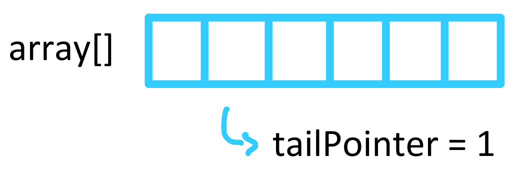


insertAtHead(int data)

- Check if the array is uninitialized. We cannot insertAtHead() in an uninitialized array.
  - If the array length/size == 0, then we will not even initialize the array.
- If the array is initialized and not full capacity, then we can insertAtHead().
  - Check if tailPointer == -1 AND the array is initialized, then we can insertAtHead().

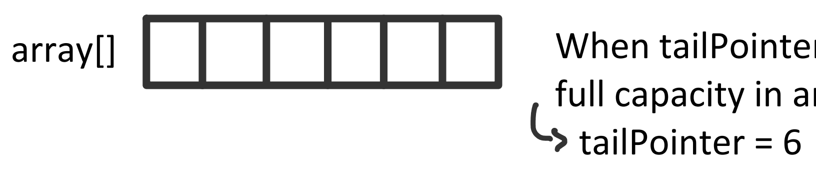


Because tailPointer == 0 after successful initialization, we can insertAtHead(data).
We can then increment tailPointer++.



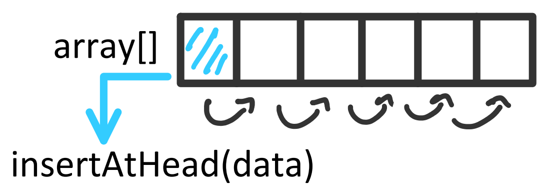
Now tailPointer == 1 so we correctly insertAtTail() and keep a reference to how many elements are in our static array.
This allows us to do adequate shifting of elements and future insertions/deletions.

- If tailPointer == array.length, then the array has full capacity and will no longer accept insertions.



When tailPointer == array.length, then we have reached full capacity in array and cannot further insert.

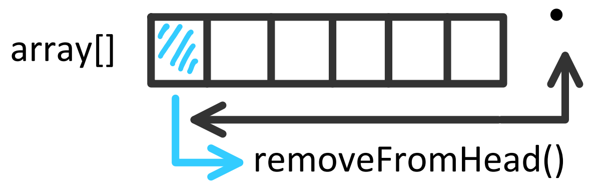
- After the first element, any insertions at head must require current element(s) to be shifted to the right one space to allocate space for new data to be inserted at the head of the array.



- Shift elements one space to the right…
- If tailPointer == array.length, then no shifting will be done and no further insertions permitted.

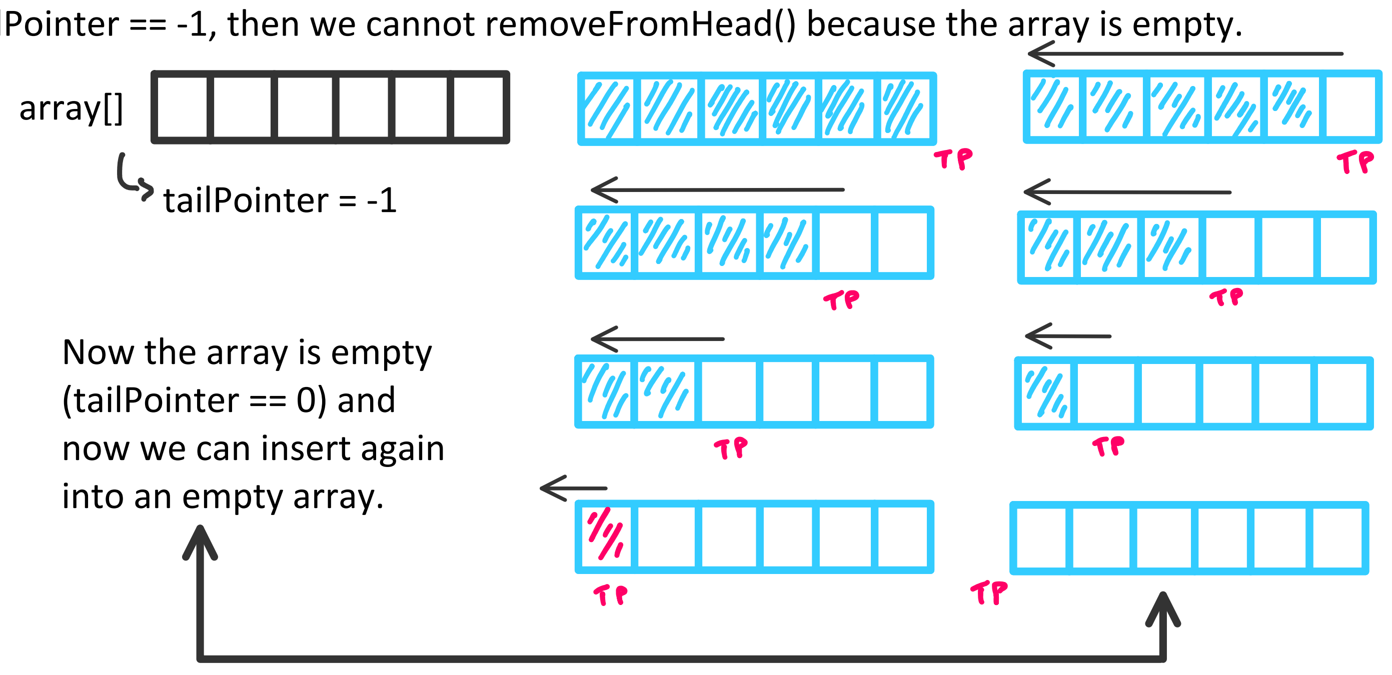
removeFromHead()

- Check if the array is uninitialized. We cannot removeFromHead() from an uninitialized array.
  - If the array length/size == 0, then we will not even initialize the array.
- Now we can successfully removeFromHead()…
  - Shift all elements one space to the left to “remove” the head element (or overwrite it).
  - Decrement tailPointer—.



- Shift all elements to the left to overwrite or “delete” the element at the head position in array.
  - Decrement tailPointer— to keep correct track of number of total elements (capacity) and keep O(1) insertion/deletion operations at tail.

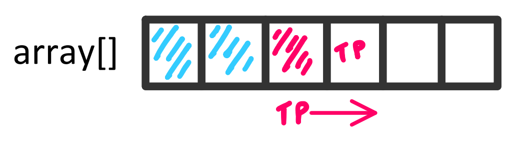
- If tailPointer == -1, then we cannot removeFromHead() because the array is empty.



Now the array is empty (tailPointer == 0) and now we can insert again into an empty array.

insertAtTail(int data)

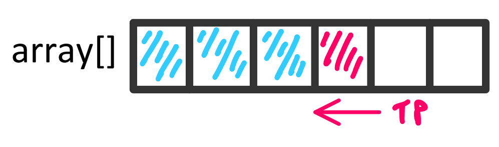
- Check if the array is uninitialized. We cannot insertAtTail(data) in an uninitialized array.
  - If the array length/size == 0, then we will not even initialize the array.
- Now we need to check if there is enough space (if capacity full) to insert a new element.
  - Check if tailPointer == array.length —> If not, then we can insertAtTail(data).



array[tailPointer] = data
tailPointer++

removeFromTail()

- Check if the array is uninitialized. We cannot removeFromTail() from an uninitialized array.
  - If the array length/size == 0, then we will not even initialize the array.
- Now we need to check if the the array is empty because we cannot remove any element from an already empty array.
  - Check if tailPointer == -1 —> throw ERROR for removing from empty array.
  - If array is not empty, then set array[tailPointer - 1] = 0. Setting or nulling out the element at the tail to 0 and decrement tailPointer—.



array[tailPointer - 1] = 0
tailPointer—