CSE 230: Data Structures

Lecture 5:Sequences: Vectors and Iterators

Dr. Vidhya Balasubramanian

Sequences

- In linear data structure each object comes before another
- Represent relationship of “next” and “previous” between related objects
  - Examples
    - Packets in a network
    - Order of instructions in a program

Vector ADT: An Overview

- Extends the array data structure by providing a sequence of objects
- Provides fast random access while maintaining ability to automatically resize when needed
- Provides accessor functions
  - Index into middle of a sequence
  - Add and remove elements by their indices
  - The index is referred to as rank

Vector: Concept of Rank

- Given a sequence S of n elements, each element e of S can be uniquely referred by an integer in the range [0,n-1]
- Rank of an element e in S
  - Number is elements that are before e in S
  - First element has rank 0, and last one has rank n-1
- Rank r does not have to mean that element e is stored at index r in the array
  - Refers to the index without referring to the actual implementation

Concept of rank

- If an element is the rth element then its rank is r-1
  - Previous element has rank r-1
- Rank may change whenever the sequence is updated
  - When a new element is inserted at the beginning of the sequence, rank of all other elements increase by 1
- Vector
  - Linear sequence that supports access by their ranks
  - Can specify where to insert or remove an element

Vector ADT: Main Operations

- elemAtRank(r)
  - Returns element S at rank r
  - Input::Integer; Output:: object;
  - Error if r<0 or r>n-1
- replaceAtRank(r,e)
  - Replace with e the element at rank r.
  - Input: integer r and object e; Output: None
  - Error if r<0 or r>n-1

Vector ADT: Main Operations

- insertAtRank(r,e)
  - Insert a new element into S so that it has rank r
  - Input:: Integer r and object e; Output:none;
  - Error if r<0 or r>n
- removeAtRank(r)
  - Remove element from S at rank r.
  - Input: integer; Output: None
  - Error if r<0 or r>n-1
- Also supports size() and isEmpty() operations

Vector Example

Operation	Output	Vector Contents
Insert 7 at rank 0	-	(7)
Insert 4 at rank 0	-	(4,7)
Return the element at rank 1	7	(4,7)
Insert 2 at rank 2	-	(4,7,2)
Return the element at rank 3	“error”	(4,7,2)
Remove the element at rank 1	-	(4,2)
Insert 5 at rank 1	-	(4,5,2)
Insert 3 at rank 1	-	(4,3,5,2)
Insert 9 at rank 4	-	(4,3,5,2,9)
Return element at rank 2	5	(4,3,5,2,9)
size()	5	(4,3,5,2,9)

Applications of Vectors

- Direct application
  - Sorted collection of objects
  - Serves as a simple database
- Indirect application
  - Component of other data structures
  - Widely used for implementing many algorithms

Realization of a deque using a vector

Deque Function	Realization with Vector Functions
size()	size()
isEmpty()	isEmpty()
first()	elemAtRank(0)
last()	elemAtRank(size()-1)
insertFirst(e)	InsertAtRank(0,e)
insertLast(e)	InsertAtRank(size(),e)
removeFirst()	removeAtRank(0)
removeLast()	removeAtRank(size()-1)

Exercise

- How can the functions of a stack be realized using a vector
- Consider a vector V. Consider the sequence of operations. Show the state of the vector and output an error if the operation is illegal.
  - For i= 1 to 10
    - Insert i at rank I-1
    - If i is even delete element at rank i-1

Vector Interface (Java)

```
interface Vector<E> {
    int size(): //returns number of objects in the vector
    bool isEmpty(); //returns true if vector is empty, false otherwise
    E elemAtRank(int r); //access element at rank r
    void replaceAtRank(int r, E obj): //replace element at given rank
    void removAtRank(int r); //remove element at given rank
    void insertAtRank(int r, E obj); //insert element at given rank
}
```

Array based implementation of a Vector

- Here A[i] stores element at rank i , and maintain a value n to keep track of size
- **Algorithm** elemAtRank(r)

  Return A[r]
- **Algorithm** replaceAtRank(r,e)

  A[r] = e
- **Algorithm** size()

  Return n-1 // the number of elements inserted must be tracked
- **Algorithm** isEmpty()

  Return (n==0)

Array ADT Functions

- **Algorithm** insertAtRank(r,e)

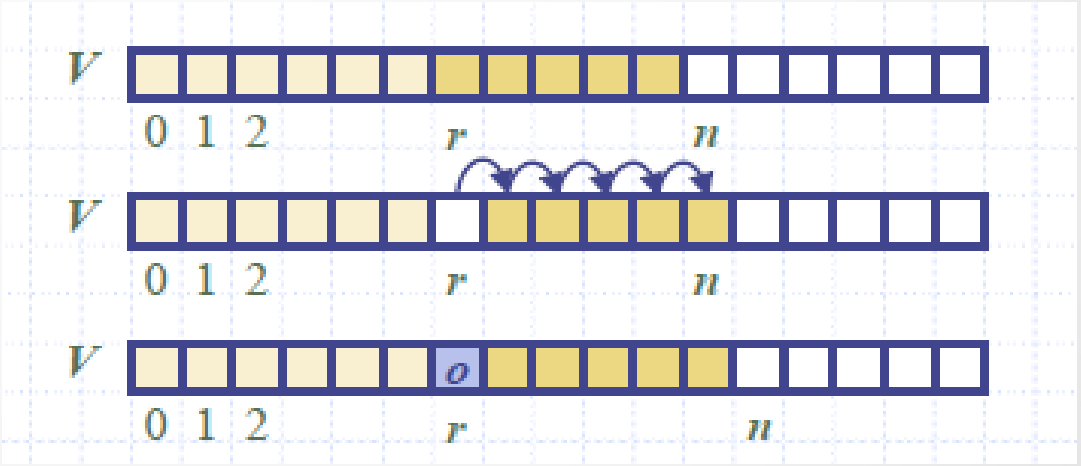
  for i = n-1, n-2, …,r **do**

  A[i+1] = A[i] // make room for new element

  A[r] ← e

  n ← n+1

- Worst case running time
  - O(n)
  - When r=0



Array ADT Functions

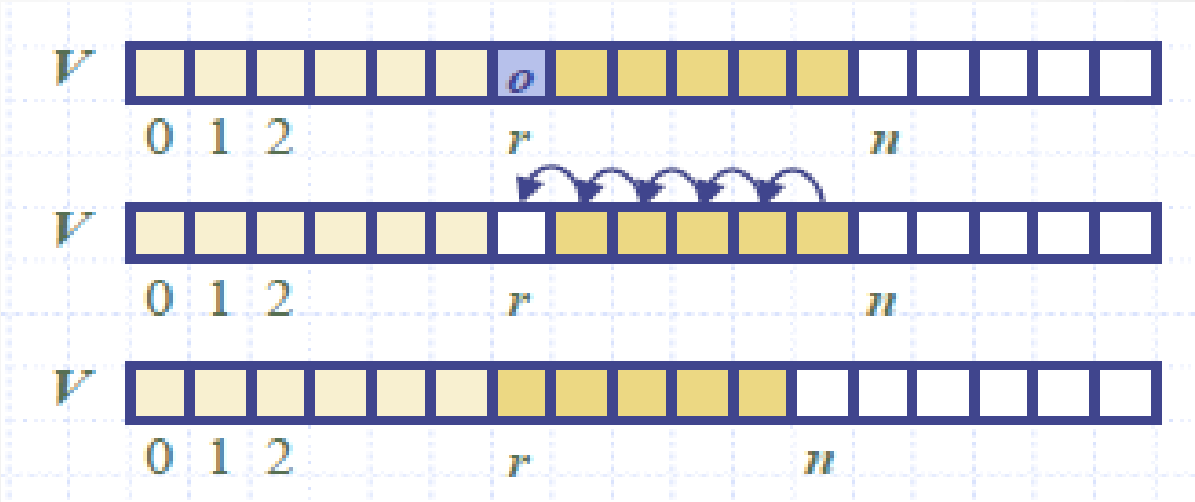
- **Algorithm** removeAtRank(r)

  for i = r, r+1, … , n-2 **do**

  A[i] ← A[i+1]

  n ← n-1

- Worst case running time
  - O(n)
  - When r = 0



Complexity Analysis

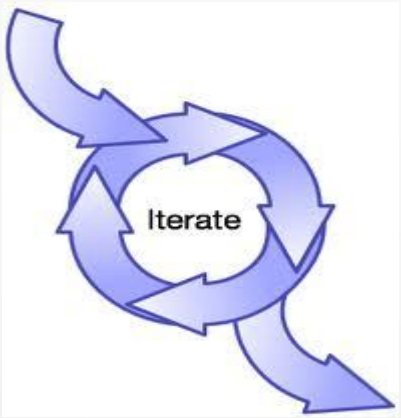
- Time Complexity
  - size – *O*(1)
  - isEmpty – *O*(1)
  - elemAtRank– *O*(1)
  - replaceAtRank – *O*(1)
  - removeAtRank – *O*(n)
  - InsertAtRank - O(n)
- Space Complexity
  - *O*(N)

Circular Array Implementation

- The restriction that element at rank r is stored at index r causes higher cost for insertions and deletions
  - Can be rectified using circular array like the one used for a queue

Iterators

- abstracts the process of scanning through a collection of elements one element at a time
- Consists of
  - a sequence S
  - a current position in S
  - a way of stepping to the next position in S making it the current position



ObjectIterator ADT

- hasNext():
  - Test whether there are elements left in the iterator
  - Input:None; Output: Boolean
- next():
  - Return the next element in the iterator and step to the next position
  - Input: None; Output: Object
  - Can be implemented by using a pointer to the current element

Iterators

- Provides a unified scheme to access all the elements of a container
  - Independent from the specific organization of the collection
  - Can be augmented with can augment the Stack, Queue, Vector, List and Sequence ADTS
    - May be implemented with an array or singly linked list
    - e.g next() in a stack can be implemented using pop()
- Extends the concept of position by adding a traversal capability
  - Returns objects according to their linear ordering

Notions and Types

- Snapshot
  - freezes the contents of the data structure at a given time
- Dynamic
  - follows changes to the data structure
- Forward and Backward Iterators
  - Allows movement in both directions on a certain ordering of elements
  - Can also support repeated access to current element