Data Structure

Recitation III
Topic

- Binary Search
- Abstract Data types
- Java Interface
- Linked List
Binary search

- Searching a sorted collection is a common task. A dictionary is a sorted list of word definitions. Given a word, one can find its definition. A telephone book is a sorted list of people's names, addresses, and telephone numbers. Knowing someone's name allows one to quickly find their telephone number and address.

- **Binary search** algorithm finds the position of a specified value (the input "key") within a sorted array

- A binary search is a divide and conquer search algorithm.
Binary search

- Example:

- The binary search algorithm can be expressed iteratively with two index limits that progressively narrow the search range:
  http://video.franklin.edu/Franklin/Math/170/common/mod01/binarySearchAlg.html

- And if it is known that some items will be searched for much more often than others, and it can be arranged that these items are at the start of the list, then a linear search may be the best.
Abstract Data Types

- Object-oriented programming (OOP): data abstraction, encapsulation, polymorphism, and inheritance.

- Separating the abstract properties from the implementation is called **data abstraction**

- The abstract properties: visible to client that makes use of the data type (the interface to the data type)

- Implementation detailed code hide from user.
Java Interface

- An interface in the Java programming language is an abstract type that is used to specify an interface (in the generic sense of the term) that classes must implement.

- Interfaces are declared using the interface keyword, and may only contain method signature and constant declarations (variable declarations that are declared to be both static and final).

- An interface may never contain method definitions.
interface Animal {
    public void eat();
    public void travel();
}

public class MammalInt implements Animal{
    public void eat(){
        System.out.println("Mammal eats");
    }
    public void travel(){
        System.out.println("Mammal travels");
    }
    public int noOfLegs(){
        return 0;
    }
    public static void main(String args[]){
        MammalInt m = new MammalInt();
        m.eat();
        m.travel();
    }
}
An interface is not a class. Writing an interface is similar to writing a class, but they are two different concepts. A class describes the attributes and behaviors of an object. An interface specify the method declaration, contains behaviors that a class implements.
An interface is different from a class in several ways, including:

- You cannot instantiate an interface.
- An interface does not contain any constructors.
- All of the methods in an interface are abstract.
- An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
- An interface is not extended by a class; it is implemented by a class.
Java Interface Adv Topic

- When implementation interfaces there are several rules:
  - A class can implement more than one interface at a time.
  - A class can extend only one class, but implement many interface.
  - An interface can extend another interface, similarly to the way that a class can extend another class.
An interface can extend another interface, similarly to the way that a class can extend another class. The extends keyword is used to extend an interface, and the child interface inherits the methods of the parent interface.

The Hockey interface has four methods, but it inherits two from Sports; thus, a class that implements Hockey needs to implement all six methods.

Similarly, a class that implements Football needs to define the three methods from Football and the two methods from Sports.
• Extending Multiple Interfaces:
  • A Java class can only extend one parent class. Multiple inheritance is not allowed.
  • Interfaces are not classes, however, and an interface can extend more than one parent interface.
  • For example, if the Hockey interface extended both Sports and Event, it would be declared as:

```java
public interface Hockey extends Sports, Event
```
interface Fruit {
    public boolean hasAPeel();
    //has a peel must be implemented in any class implementing Fruit
    //methods in interfaces must be public
}

interface Vegetable {
    public boolean isARoot();
    //is a root must be implemented in any class implementing Vegetable
    //methods in interfaces must be public
}
A class can extend only one class, but implement many interface.

class Tomato implements Fruit, Vegetable {

    boolean peel = false;
    boolean root = false;

    public Tomato() {} 

    public boolean hasAPeel()
        // must have this method, 
        // because Fruit declared it
    {
        return peel;
    }

    public boolean isARoot()
        // must have this method, 
        // because Vegetable declared it
    {
        return root;
    }
}
Linked List

- Linked list is a data structure consisting of a group of nodes which together represent a sequence. Under the simplest form, each node is composed of a data area and a reference (a link) to the next node in the sequence.
- This structure allows for efficient insertion or removal of elements from any position in the sequence.
- More complex variants add additional links: double linked list.
Linked List Operations

Traversal

Deletion

Insertion
Traversals

- Traversal of a singly linked list is simple, beginning at the first node and following each `next` link until we come to the end:

```plaintext
node := list.firstNode
while node not null
    (do something with node.data)
    node := node.next
```
Insert a new node

```plaintext
function insertAfter(Node node, Node newNode) // insert newNode after node
    newNode.next := node.next
    node.next := newNode
```
Delete a node

To find and remove a particular node, one must again keep track of the previous element.

```javascript
function removeAfter(node node) { // remove node past this one
    obsoleteNode := node.next
    node.next := node.next.next
    destroy obsoleteNode
}
```
Adv Linked List Operations

- How to reverse a linked list:
Adv Linked List Operations

- How to reverse a linked list:
  - Initially take three pointers: PrevNode, CurrNode, NextNode
  - Let CurrNode point to HeaderNode of the list. And let PrevNode and NextNode points to null
  - Now iterate through the linked list until CurrNode is null
  - In the loop, we need to change NextNode to PrevNode, PrevNode to CurrNode and CurrNode to NextNode
public ListNode reverseList(ListNode headerNode) {
    ListNode prevNode = null;
    ListNode currNode = headerNode;
    ListNode nextNode = null;

    while (currNode != null) {
        nextNode = currNode.next;
        currNode.next = prevNode;
        prevNode = currNode;
        currNode = nextNode;
    }

    return prevNode;
}
Linked List VS Array

- Access: Random / Sequential
- Memory Structure
- Insertion / Deletion
- Memory Allocation
Linked List VS Array

1. Access : Random / Sequential
   - Array elements can be randomly Accessed using Subscript Variable
   - e.g. a[0], a[1], a[3] can be randomly accessed
   - While In Linked List We have to Traverse Through the Linked List for Accessing Element. So O(n) Time required for Accessing Element.
   - Generally In linked List Elements are accessed Sequentially.
Linked List VS Array

2. Memory Structure:
   - Stack is stored in contiguous Memory Locations, i.e., Suppose first element is stored at 2000 then second integer element will be stored at 2002.
   - But it is not necessary to store next element at the consecutive memory location.
   - Element is stored at any available location, but the pointer to that memory location is stored in previous node.
Linked List VS Array

3. Insertion / Deletion

- As the Array elements are stored in consecutive memory locations, so while inserting elements, we have to create space for insertion.
- So more time required for creating space and inserting element.
- Similarly, we have to delete the element from given location and then shift all successive elements up by 1 position.
- In Linked List, we have to just change the pointer address field (Pointer), so insertion and deletion operations are quite easy to implement.
Linked List VS Array

- 4. Memory Allocation:
  - Memory Should be allocated at Compile-Time in Stack, i.e. at the time when Programmer is Writing Program.
  - In Linked list memory can be allocated at Run-Time, i.e. After executing Program.
  - Stack uses Static Memory Allocation and Linked List Uses Dynamic Memory Allocation.
  - Dynamic Memory allocation functions – new, malloc, calloc, delete etc...
More Linked List...

- Circular linked lists

![Circular linked list](image)

- Doubly linked list

![Doubly linked list](image)