CS211. DATA STRUCTURES AND ALGORITHMS

Lab Packets- Solutions

Modhi AlSobeihy

Table of Contents

Lab 01. Review Arrays

Examples

1. The following program shows how an array structure can be used.

```
public class Lab1 {
public static void main(String[] args) {
	/**
	 * Demonstrates Java arrays
	 **/
	Scanner read = new Scanner(System.in);// input from keyboard
	int[] arr; // reference
	arr = new int[100]; // make array
	int nElems = 0; // number of items
	int j; // loop counter
	int searchKey; // key of item to search for

	// ------------------------------------------------------------
	// reads a set of 10 integer numbers
	System.out.println("Enter 10 integer numbers.");

	for (j = 0; j < 10; j++)
		arr[nElems++] = read.nextInt(); // insert number into arr
	// now 10 items in array
	System.out.println("The set contains: ");
	for (j = 0; j < nElems; j++) // display items
		System.out.print(arr[j] + " ");
	System.out.println("");
	// ------------------------------------------------------------
	System.out.print("Search for key: ");
	searchKey = read.nextInt(); // find item with the given key
	for (j = 0; j < nElems; j++) // for each element,
		if (arr[j] == searchKey) // found item?
			break; // yes, exit before end
	if (j == nElems) // at the end?
		System.out.println("Can't find " + searchKey); // yes
	else
		System.out.println("Found " + searchKey); // no
	// ------------------------------------------------------------
	System.out.print("Delete key: ");
	searchKey = read.nextInt(); // delete item with the given key
	boolean found = false;
	for (j = 0; j < nElems; j++) // look for it
		if (arr[j] == searchKey) {

			found = true;
			break;
		}
	if (found) {
		System.out.println("Deleting " + searchKey); // yes
		for (int k = j; k < nElems; k++) // move higher ones down
			arr[k] = arr[k + 1];
		nElems--; // decrement size
	} else
		System.out.println("Can't find " + searchKey); // yes
```

```
            // ------------------------------------------------------------
            System.out.println("The set contains: ");
            for (j = 0; j < nElems; j++) // display items
                  System.out.print(arr[j] + " ");
            System.out.println("");

            read.close(); // close read file
      } // end main
} // end class
```

```
Sample Run
Enter 10 integer numbers.
22 55 88 66 44 11 99 33 77 0
The set contains:
22 55 88 66 44 11 99 33 77 0
Search for key: 33
Found 33
Delete key: 22
Deleting 22
The set contains:
55 88 66 44 11 99 33 77 0
```

Exercises

1. The example program is written in old-fashioned procedural approach, write the equivalent object-oriented approach.

```
public static void main(String[] args)
          int maxSize = 100; // array size
          ObjOrtArray arr; // reference to array
          int j; // loop counter
          Scanner read = new Scanner(System.in);// input from keyboard
          arr = new ObjOrtArray(maxSize); // create the array
          System.out.println("Enter 10 double numbers.");
          for (j = 0; j < 10; j++)
                arr.insert(read.nextDouble());// insert 10 items
          arr.display(); // display items
          System.out.print("Search for key: ");
          int searchKey = read.nextInt(); // find item with the given key
          if (arr.find(searchKey))
                System.out.println("Found " + searchKey);
          else
                System.out.println("Can't find " + searchKey);
          System.out.print("Delete key: ");
          searchKey = read.nextInt(); // delete item with the given key
          if(arr.delete(searchKey))
                System.out.println(searchKey+" Deleted successfully!");
          else
                System.out.println("Can't find " + searchKey);
          System.out.print("Delete key: ");
          searchKey = read.nextInt(); // delete item with the given key
          if(arr.delete(searchKey))
                System.out.println(searchKey+" Deleted successfully!");
          else
                System.out.println("Can't find " + searchKey);
          arr.display(); // display items again
    }
}
class ObjOrtArray {
    private double[] a; // ref to array a
    private int nElems; // number of data items
//--------------------------------------------------------------
    public ObjOrtArray(int max) // constructor
    {
          a = new double[max]; // create the array
          nElems = 0; // no items yet
    }
//--------------------------------------------------------------
    public boolean find(double searchKey) { // find specified value
          int j;
          for (j = 0; j < nElems; j++) // for each element,
                if (a[j] == searchKey) // found item?
                      break; // exit loop before end
          if (j == nElems) // gone to end?
                return false; // yes, can't find it
          else
                return true; // no, found it
```

```
    } // end find()
//-------------------------------------------------------------
    public void insert(double value) // put element into array
    {
          a[nElems] = value; // insert it
          nElems++; // increment size
    }
//-------------------------------------------------------------
    public boolean delete(double value) {
          int j;
          for (j = 0; j < nElems; j++) // look for it
                if (value == a[j])
                      break;
          if (j == nElems) // can't find it
                return false;
          else // found it
          {
                for (int k = j; k < nElems; k++)// move higher ones down
                      a[k] = a[k + 1];
                nElems--; // decrement size
                return true;
          }
    } // end delete()
//-------------------------------------------------------------
    public void display() // displays array contents
    {
          for (int j = 0; j < nElems; j++) // for each element,
                System.out.print(a[j] + " "); // display it
          System.out.println("");
    }
//-------------------------------------------------------------
```

2. Overload the method `display()` so it displays a predefined part of the array

```
public void display(int start, int end) // displays part of the array
    {
          if (!(start <= end & end <= nElems)) {
                System.out.println("Invalid indecies!");
                return;
          }
          for (int j = start; j <= end; j++)
                System.out.print(a[j] + " ");
          System.out.println("");
    }
```

Homework

1. Write `insertSorted(double key)` member method, which inserts items into the array according to their natural order.

```
public void insertSorted(double key) {
		int i;
		for (i = nElems - 1; (i >= 0 && a[i] > key); i--)
			a[i + 1] = a[i];
		a[i + 1] = key;
		nElems++;
	}
```

2. Write `rotate(int k)` member method, which rotates an array to the right by k positions(Hint. Use another array structure).

```
public void rotate(int k) {
		if (k > nElems)
			k = k % nElems;
		double[] result = new double[nElems];
		for (int i = 0; i < k; i++) {
			result[i] = a[nElems - k + i];
		}
		int j = 0;
		for (int i = k; i < nElems; i++) {
			result[i] = a[j];
			j++;
		}
		System.arraycopy(result, 0, a, 0, nElems);
	}
```

Challenge Question

1. Rewrite the `rotate(int k)` member method so no auxiliary array structure is needed.

```
public void rotateInPlace(int k) {
		if (a == null || k < 0) {
		    throw new IllegalArgumentException("Illegal argument!");
		}

		for (int i = 0; i < k; i++) {
			for (int j = nElems - 1; j > 0; j--) {
				double temp = a[j];
				a[j] = a[j - 1];
				a[j - 1] = temp;
			}
		}
```

Lab 02. Analysis of Algorithms

Examples

2. You are given an array of N records that is sorted with respect to some key field contained in each record.
    a. Give two different algorithms for searching the array to find the record with a specified key value.
    b. Which one do you consider "better"?
    c. In worst case scenario, how many comparisons do you perform in each algorithm?

a.1 Algorithm1.
Since the array is ordered ($R_0 \leq R_1 \ldots \leq R_{n-1}$), then we can conclude that the target record does not exists once R_i exceeds the target. We will start from the first record and check until the target record is found or a record with a larger key has been reached,

1. Traverse the array sequentially.
2. In every iteration, compare the `target` key with the current key value of the array.
    1. If the values match, return the current index of the array.
    2. If the values do not match, move on to the next array element.
    3. If key value > `target`, return `-1`.

a.2 Algorithm2.
Since the array is ordered ($R_0 \leq R_1 \ldots \leq R_{n-1}$), we will start by dividing the array into 2 halves and exclude the half that contains larger keys. We will repeat the dividing process until the target record is found or we can no longer divide the remaining half.

1. Start with the middle element:
    - If the `target` value is equal to the middle element of the array, then return the index of the middle element.
    - If not, then compare the middle element with the target value,
        - If the target value is greater than the number in the middle index, then pick the elements to the right of the middle index and start with Step 1.
        - If the target value is less than the number in the middle index, then pick the elements to the left of the middle index and start with Step 1.
2. When a match is found, return the index of the element matched.
3. If no match is found, then return `-1`

b. Algorithm2 is better because we perform less comparisons hence less time is consumed to identify whether the target element exists.

c. For N elements:
algorithm 1 : N
algorithm2: log N

Exercises

1. Imagine that you are a programmer who must write a function to sort an array of about 1000 integers from lowest value to highest value. Write down at least four approaches to sorting the array. Do not write algorithms in Java or pseudocode. Just write a sentence or two for each approach to describe how it would work.

```
Lab instructor can discuss the ideas of various sorting algorithms such as
Bubble sort; Selection sort; Insertion sort; Quick sort and Merge sort
```

2. What is the order of the following growth functions?

a. $10n^2 + 100n + 1000$	**a.** *$O(n^2)$*
b. $10n^3 - 7$	**b.** *$O(n^3)$*
c. $2^n + 100n^3$	**c.** *$O(2^n)$*
d. $n^2 \log n$	**d.** *$O(n^2 \log n)$*

3. For each of the following code snippets, give an analysis of the running time (Big-Oh will do).

```
(1)  sum = 0;
     for( i = 0; i < n; i++ )
           sum++;

(2)  sum = 0;
     for( i = 0; i < n; i++ )
        for( j = 0; j < n; j++ )
           sum++;

(3)  sum = 0;
     for( i = 0; i < n; i++ )
        for( j = 0; j < n * n; j++ )
           sum++;

(4)  sum = 0;
     for( i = 0; i < n; i++ )
        for( j = 0; j < i * i; j++ )
           for( k = 0; k < j; k++ )
                 sum++;
```

(1) The running time is $O(N)$.

(2) The running time is $O(N^2)$.

(3) The running time is $O(N^3)$.

(4) j can be as large as i^2, which could be as large as N^2. k can be as large as j, which is N^2. The running time is thus proportional to $N \cdot N^2 \cdot N^2$, which is $O(N^5)$.

Homework

1. Order the following functions by growth rate:

   N, $\sqrt{N}$, $N^{1.5}$, N^2, $N \log N$, $N \log \log N$, $N \log^2 N$, $N \log(N^2)$, $2/N$, 2^N, $2^{N/2}$, 37, $N^2 \log N$, N^3.

   Indicate which functions grow at the same rate.

   $2/N$, 37, $\sqrt{N}$, N, $N \log \log N$, $N \log N$, $N \log(N^2)$, $N \log^2 N$, $N^{1.5}$, N^2, $N^2 \log N$, N^3, $2^{N/2}$, 2^N.

   $N \log N$ and N log (N^2) grow at the same rate.

2. Solving a problem requires running an *O(N)* algorithm and then afterwards a second *O(N)* algorithm. What is the total cost of solving the problem?

   The total cost is O(N)

3. Solving a problem requires running an $O(N^2)$ algorithm and then afterwards an *O(N)* algorithm. What is the total cost of solving the problem?

   The total cost is $O(N^2)$

Challenge Question

1. An algorithm takes 0.5 ms for input size 100. How long will it take for input size 500 (if low-order terms are negligible) if the running time is
   a. **linear**
   b. ***O (N log N)***
   c. **quadratic**
   d. **cubic**

(a) 2.5 milliseconds (5 times longer);

0.5 * 5

(b) 3.5 milliseconds (about 7 times longer; 5 * log(500)/log(100));

Slightly more than five times as long 0.5 * log(5)

(c) 12.5 milliseconds (25 times longer);

$0.5 * 5^2$

(d) 62.5 milliseconds (125 times)

$0.5 * 5^3$

Lab 03. Recursion

Examples

1. Computing a positive integer power of a number is easily seen as a recursive process. Consider a^n : If n = 0, a^n is 1 (by definition)
   If n > 0, a^n is a * a^{n-1}.
   Apply the following code segment which reads in integers base and exp and calls method power to compute $base^{exp}$

```
public static void main(String[] args) {
		int base, exp;
		int answer;
		Scanner read = new Scanner(System.in);
		System.out.println("Use integers only.");
		// get base
		System.out.print("Base: ");
		base = read.nextInt();
		// get exponent
		System.out.print("Power: ");
		exp = read.nextInt();
		answer = power(base, exp);
		System.out.println(base + " raised to the " + exp + " is " + answer);
	}
public static int power(int base, int exp) {
		int pow;
		// if the exponent is 0, set pow to 1
		// otherwise set pow to base*base^(exp-1)
		if (exp == 0)
			return 1;
		return base * power(base, --exp);
	}
```

2. Printing a string backwards can be done iteratively or recursively. To do it recursively, think of the following specification:
   If s contains any characters
   print the last character in s
   print s' backwards, where s' is s without its last character
   Apply the following code segment which reads in a string and displays in reverse.

```
public static void main(String[] args) {
	String str;
	Scanner read = new Scanner(System.in);
	System.out.println("Enter a string: ");
	str = read.nextLine();
	System.out.println("original: "+str + "; reversed: " + reverseString(str));
}
public static String reverseString(String str){
	if(str.isEmpty()){
		return str;
	} else {
		return reverseString(str.substring(1))+str.charAt(0);
	}
}
```

Exercises

1. Write a recursive method `digitSum` that takes a non-negative integer and returns the sum of its digits. For example, `digitSum(1234)` returns `1 + 2 + 3 + 4 = 10`. Hint. It is easy to break a number into two smaller pieces by dividing by 10 (i.e., `1234/10 = 123` and `1234%10 = 4`). Write the prober Java statements to test your code.

```
static int digitSum(int n)
    {
        if (n == 0)
            return 0;
        return (n % 10 + digitSum(n / 10));
    }
```

2. Write a recursive method `isPalindrome` that takes a string and returns true if it is the same when read forwards or backwards. For example,

```
isPalindrome("home")  → false

isPalindrome("madam") → true
```

```
static boolean isPalindrome(String str, int s, int e) {
            int n = str.length();

            // An empty string is palindrome
            if (n == 0)
                  return true;
      // If there is only one character
            if (s == e)
                  return true;
            // If first and last characters do not match
            if ((str.charAt(s)) != (str.charAt(e)))
                  return false;

            // If there are more than two characters, check if middle substring
is also palindrome.
            if (s < e + 1)
                  return isPalindrome(str, s + 1, e - 1);

            return true;
      }
```

Homework

1. Write `int crazySum(int n)`, a recursive method that calculates the sum $1^1 + 2^2 + 3^3 + ... + n^n$, given an integer value of n in between 1 and 9. Make sure to call the method power from example one in your implementation below:

```
public static int crazySum(int n) {
          if (n == 1)
          return 1;
          else
          return power(n,n) + crazySum(n-1);
          }
```

2. Given an unsorted array and an element e, write recursive method search() which returns true if the element is in the array, false otherwise.

```
static boolean search(int arr[], int l, int r, int x) {
          if (r < l)
                return false;
          if (arr[l] == x)
                return true;
          if (arr[r] == x)
                return true;
          return search(arr, l + 1, r - 1, x);
    }
```

Challenge Question

1. Subset Sum is an important and classic problem in computer theory. Given a set of integers and a target number, the goal is to find a subset of those numbers that sums to the target number. For example, given the set {3, 7, 1, 8, -3} and the target sum 4, the subset {3, 1} sums to 4. On the other hand, if the target sum were 2, the result is false since there is no subset that sums to 2. Write a recursive method `boolean haveSubsetSum(int subset[], int n, int target)`. Assume that the array contains is completely full. Note that you are not asked to print the subset members, just return true or false.

```
public static void main(String args[]) {
          int set[] = { 5, 7, 25, 3, 5, 12, 30, 1 };
          int sum = 15;
          int n = set.length;
          if (haveSubsetSum(set, n, sum) == true)
                System.out.println("Found a subset" + " with the sum of " +
sum);
          else
                System.out.println("No subset with" + " the sum of " + sum);
   }

   static boolean haveSubsetSum(int subset[], int n, int target) {
          // Base Cases
          if (target == 0)
                return true;
          if (n == 0 && target != 0)
                return false;
          // If last element is greater than sum, then ignore it
          if (subset[n - 1] > target)
                return haveSubsetSum(subset, n - 1, target);

          /*
           * else, check if sum can be obtained by any of the following (a)
including the
           * last element (b) excluding the last element
           */
          return haveSubsetSum(subset, n - 1, target) || haveSubsetSum(subset,
n - 1, target - subset[n - 1]);
   }
```

Lab 04. Single Linked List

Examples

1. Run the following code and check the output.

```
public class SingleLinkedList {

	Node head;

	/*
	 * this is an example implementation of some of the basic operations of the
	 * simple linked list. the implementation of double linked list operations is
	 * similar, just remember that you are dealing with two pointers in each node object.
	 */

	// you can add a tail pointer to point always to the last node in the linked list
	void insertFirst(int data) {// creates the node within the method
		if (head == null)
			head = new Node(data);
		else {
			Node newNode = new Node(data);
			newNode.next = head;
			head = newNode;
		}
	}
	void insertLast(int data) {
		if (head == null)
			head = new Node(data);
		else {
			Node cur = head;
			for (; cur.next != null; cur = cur.next)
				;
			// this for segment does nothing but traversing to the last node in
			// the list
			// if we keep a tail pointer, we will directly write t.next = newNode;
			Node newNode = new Node(data);
			cur.next = newNode;
		}
	}
	void insertAfterNode(int prevData, int newData) {
		Node cur = head;
		// we need to search until we find the right location to insert the node
		while (cur.next != null && cur.data != prevData)
			cur = cur.next;
		Node newNode = new Node(newData);
		newNode.next = cur.next;
		cur.next = newNode;
	}
	void deleteFirst() {
		if (head != null)
			head = head.next;
		else
			System.out.println("List is empty!!");
	}
```

```
	void deleteLast() {
		if (head == null)// another way to test
		{
			System.out.println("List is empty!!");
			return;
		}
		Node cur = head;
		for (; cur.next.next != null; cur = cur.next)
			;
		// this for segment does nothing but traversing to the node
		// before the last node in the list
		cur.next = null;
	}

	void deleteAfterNode(int prevData) {

		if (head == null) {
			System.out.println("List is empty!!");
			return;
		}
		Node cur = head;
		while (cur.next.next != null && cur.data != prevData)
			cur = cur.next;
		if (cur.data != prevData)
			System.out.println("List does not contain the target node or the target
node is the last node!");
		else
			cur.next = cur.next.next;
	}

	void displayList() {
		if (head == null) {
			System.out.println("List is empty!!");
			return;
		}
		Node cur = head;
		while (cur != null) {
			System.out.print(cur.data + " -> ");
			cur = cur.next;
		}
		System.out.println("/");
	}
}

class Node {
	int data;
	Node next = null;

	Node(int d) {
		data = d;
	}
}


public class SingleLinkedListApplication {
```

```
	public static void main(String[] args) {
		SingleLinkedList sll = new SingleLinkedList();
		sll.insertFirst(15);
		sll.displayList();
		sll.insertFirst(25);
		sll.displayList();
		sll.insertLast(35);
		sll.displayList();
		sll.insertLast(45);
		sll.displayList();
		sll.displayList();
		sll.displayList();
		sll.displayList();
		sll.insertFirst(75);
		sll.displayList();
		sll.deleteFirst();
		sll.displayList();
		sll.deleteLast();
		sll.displayList();
		sll.deleteAfterNode(55);
		sll.displayList();
		sll.deleteAfterNode(25);
		sll.displayList();
	}
}
```

Exercises

1. Write the following member methods, and add the appropriate code in the main application to invoke them:
    a. count() : returns the number of nodes in the linked list
    b. `displayPreSucc(T val)`: displays the data values of the predecessor and successor nodes of val in the list

```
a. int count() {
        Node current = head;
        int count = 0;
        while (current != null) {
            count++;
            current = current.next;
        }
        return count;
    }

b. void displayPreSucc(int val) {
        Node current = head;
        Node succ = null;
        Node pre = null;
        if (current.data == val) {
            succ = current.next;
        } else
            while (current != null && current.data != val) {
                pre = current;
                succ = current.next.next;
                current = current.next;
            }
        if (pre != null && succ != null)
            System.out.println(pre.data + " -> " + current.data + " -> "
                    + succ.data);
        else if (pre == null && succ != null)
            System.out.println(current.data + " -> " + succ.data);
        else if (pre != null && succ == null)
            System.out.println(pre.data + " -> " + current.data);
        else
            System.out.println(current.data);
    }
```

Homework

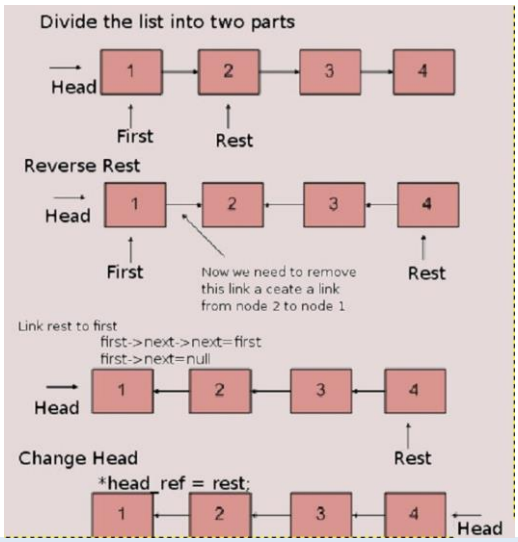
1. Find Union and Intersection of two sets. Given two sets (elements in a set are distinct), write a method that finds the union and intersection of two sets using linked list. E.g., given A= {6,1,3, 20,7 }, B={2,6,8,3,5} your algorithm should print Union as {1, 2, 3, 5, 6, 7, 8, 20} and Intersection as {3, 6}. Note that the elements of union and intersection can be printed in any order

```
public void union(SingleLinkedList set2) {
SingleLinkedList temp;
if (set2.head == null) {
displayList();
return;
}
if (head == null) {
set2.displayList();
return;
}

/* Merge the two lists */
Node cur = head;
while (cur.next != null) {
cur = cur.next;
}
cur.next = set2.head;
/* Remove the duplicates */
Node ptr1 = null, ptr2 = null, dup = null;
ptr1 = head;

/* Pick elements one by one */
while (ptr1 != null && ptr1.next != null) {
ptr2 = ptr1;

/*
Compare the picked element with rest of the elements
*/
while (ptr2.next != null) {

/* If duplicate then delete it */
if (ptr1.data == ptr2.next.data) {

/* sequence of steps is important here */
dup = ptr2.next;
ptr2.next = ptr2.next.next;
System.gc();
} else /* This is tricky */ {
ptr2 = ptr2.next;
}
}
ptr1 = ptr1.next;
}
displayList();
}
void intersection(SingleLinkedList set2) {
Node set1Cur;
```

```
    Node set2Cur;
    SingleLinkedList intersectionSet = new SingleLinkedList();
    for (set1Cur = head; set1Cur != null; set1Cur = set1Cur.next) {
    for (set2Cur = set2.head; set2Cur != null; set2Cur = set2Cur.next)
    if (set1Cur.data == set2Cur.data) {
    intersectionSet.insertFirst(set1Cur.data);
    }
    }
    intersectionSet.displayList();
    }
```

Challenge Question

1. Reverse a singly linked list. Write reverseIteratively() method which reverses a single linked list using a three-pointers approach and using loops. It traverses through the linked list and adds nodes at the beginning of the singly linked list in each iteration. It uses three reference variables (pointers) to keep track of previous, current, and next nodes.




```
void reverseIteratively() {
            Node current = head;
            Node previous = null;
            Node forward = null;
            // traversing linked list until there is no more element
            while (current.next != null) {
                  // Saving reference of next node, since we are changing current node
                  forward = current.next;
                  // Inserting node at start of new list
                  current.next = previous;
                  previous = current;
                  // Advancing to next node
                  current = forward;
            }
            head = current;
            head.next = previous;
            displayList();
      }
```

Lab 05. Doubly Linked List

Examples

1. A doubly linked list is a data structure that consists of a sequence of nodes. Each node stores some element value and two links, one link to the previous node and one link to the next node. The following code shows the implementation of doubly linked list. Apply the code and check the output.

```
package LinkedListLab;

public class DoublyLinkedListApplication {
   public static void main(String[] args) {
         /* Start with the empty list */
         DoublyLinkedList dll = new DoublyLinkedList();

         dll.insertLast("One");
         dll.insertLast("Two");
         dll.insertFirst("Three");
         dll.insertFirst("Four");
         dll.InsertAfter("Three", "Five");
         dll.insertFirst("Six");
         dll.insertLast("Seven");
         dll.printlist(dll.head);
         System.out.println(dll.deleteNode(target)?target+" Found and deleted
successfuly!":target+" Not Found!");
         target = "Seven";
         System.out.println(dll.deleteNode(target)?target+" Found and deleted
successfuly!":target+" Not Found!");
         target = "Nine";
         System.out.println(dll.deleteNode(target)?target+" Found and deleted
successfuly!":target+" Not Found!");
               target = "One";
         System.out.println(dll.deleteNode(target)?target+" Found and deleted
successfuly!":target+" Not Found!");
         dll.printlist(dll.head);  }
}

class DoublyLinkedList {
   DNode head; // head of list

   // Adding a node at the front of the list
   public void insertFirst(String new_data) {
         /*
          * 1. allocate node 2. put in the data
          */
         DNode new_Node = new DNode(new_data);

         /* 3. Make next of new node as head and previous as NULL */
         new_Node.next = head;
         new_Node.prev = null;

         /* 4. change prev of head node to new node */
         if (head != null)
               head.prev = new_Node;
```

```
            /* 5. move the head to point to the new node */
            head = new_Node;
    }

    public void InsertAfter(String string, String string2) {
            //implement this part
    }

    // Add a node at the end of the list
    void insertLast(String new_data) {
            //implement this part
    }

    public void printlist(DNode node) {
            DNode last = null;
            System.out.println("Traversing the Doubly Linked List in forward Direction");
            while (node != null) {
                    System.out.print(node.element + " ");
                    last = node;
                    node = node.next;
            }
            System.out.println();
            System.out.println("Traversing the Doubly Linked List in reverse direction");
            // implement this part
    }
    // delete a node in a Doubly Linked List.
    boolean deleteNode(String target) {
            // implement this part
    }
}

/* DNode of a doubly linked list of strings */
class DNode {
    protected String element; // String element stored by a node

    protected DNode next, prev; // Pointers to next and previous nodes

    /* Constructor that creates a node with given fields */

    public DNode(String e) {

            element = e;

            prev = null;

            next = null;
    }

    /* Returns the element of this node */

    public String getElement() {
            return element;
    }

    /* Returns the previous node of this node */
```

```
    public DNode getPrev() {
           return prev;
    }

    /* Returns the next node of this node */

    public DNode getNext() {
           return next;
    }

    /* Sets the element of this node */

    public void setElement(String newElem) {
           element = newElem;
    }

    /* Sets the previous node of this node */

    public void setPrev(DNode newPrev) {
           prev = newPrev;
    }

    /* Sets the next node of this node */

    public void setNext(DNode newNext) {
           next = newNext;
    }

}
```

Exercises

1. Write the proper Java code to implement `insertLast(String new_data)` member methods, which inserts a node at the end for the list

```
/*
           * 1. allocate node 2. put in the data
           */
          DNode new_node = new DNode(new_data);

          DNode last = head; /* used in step 5 */

          /*
           * 3. This new node is going to be the last node, so make next of it
as NULL
           */
          new_node.next = null;

          /*
           * 4. If the Linked List is empty, then make the new node as head
           */
          if (head == null) {
                new_node.prev = null;
                head = new_node;
                return;
          }

          /* 5. Else traverse till the last node */
          while (last.next != null)
                last = last.next;

          /* 6. Change the next of last node */
          last.next = new_node;

          /* 7. Make last node as previous of new node */
          new_node.prev = last;
```

2. Complete the missing code in `printlist(DNode node)` which should traverse the list backwards.

```
while (last != null) {
                    System.out.print(last.element + " ");
                    last = last.prev;
               }
               System.out.println();
```

Homework

1. Write the proper java code to implement **boolean** `deleteNode(String target)` member method.

```
{
        if (head == null || target == null) {
            return false;
        }
        // If node to be deleted is head node
        if (head.element.equals(target)) {
            head = head.next;
            head.prev = null;
            return true;
        }

        DNode current = head;
        while (current.next != null && !current.element.equals(target))
            current = current.next;
        // If node to be deleted is last node
        if (current.next == null && current.element.equals(target)) {
            current.prev.next = current.next;
            return true;
        }
        // If node to be deleted is neither head nor last node
        if (current.next != null && current.element.equals(target)) {
            current.next.prev = current.prev;
            current.prev.next = current.next;
            return true;
        }
        // If no node with such value
        return false;
    }
```

2. Write the proper Java code to implement `InsertAfter(String preValue, String newValue)` member method.

```
{
        /* 1. allocate a new node put in the new value */
        DNode new_node = new DNode(newValue);
        /* 2. traverse the list to get to the node with preValue */
        DNode current = head;
        while (current.next != null && !current.element.equals(preValue))
            current = current.next;
        if (current.element.equals(preValue)) {
            /* 3. insert new_node after current */
            current.next.prev = new_node;
            new_node.next = current.next;
            new_node.prev = current;
            current.next = new_node;
        }
    }
```

Challenge Question

1. Write the proper java code to implement a recursive method that returns the size of a given doubly linked list .

```
static int getListSize(DoublyLinkedList dll) {
		if(dll == null)
			return 0;
		return getListSize(dll.head);
	}

	static int getListSize(DNode node) {
		if(node == null)
			return 0;
		return 1+ getListSize(node.next);
	}
```

Lab 06. Stack ADT

Examples

1. Stacks provide FILO (First-in-last-out) or LIFO (Last-in-fist-out) access where the first item in the set is accessed last. Regardless of the data structure we use to implement a stack, stacks have two main operations: push() and pop() and two supporting operations: isEmpty() and isFull().
The following code implement a stack using an array. Apply and test the code.

```
// Stack.java: stack implementation
class Stack {
	private int maxStack;
	private int emptyStack;
	private int top;
	private char[] items;
	public Stack(int size) {
		maxStack = size;
		emptyStack = -1;
		top = emptyStack;
		items = new char[maxStack];
	}
	public void push(char c) {
		items[++top] = c;
	}
	public char pop() {
		return items[top--];
	}
	public boolean isFull() {
		return top + 1 == maxStack;
	}
	public boolean isEmpty() {
		return top == emptyStack;
	}
}
// Stackmain.java: use the stack
import java.io.*;
public class StackMain {
  final static int STACK_SIZE = 10;
	public static void main(String[] args) throws IOException {
		Scanner read = new Scanner(System.in);
		Stack s = new Stack(STACK_SIZE); // 10 chars
		char ch;
		System.out.println("Enter 10 characters:");
		for (int i = 0; i < STACK_SIZE; i++) {
			ch = read.next().charAt(0);
			if (!s.isFull())
				s.push(ch);
		}
		while (!s.isEmpty())
			System.out.print(s.pop());
		System.out.println();
	}
}
```

2. Apply and test the following code which creates a stack interface and a class that implements the stack interface using an array. However, the data type is generic, so it accepts any type of data (i.e. int, double, long, char, String… etc.). Note. the array expands to fit any number of elements; use the same Stackmain.java; just change the type of the stack object and remove the isFull() call.

```
//Stack.java: stack implementation using generic parameterized type E
public interface StackInterface<E> {
    public boolean isEmpty();
    public E peek();
    public E pop();
    public void push(E element);
    public int size();

}
// ArrayStack.java: stack implements the Stack<E>
class ArrayStack<E> implements StackInterface<E> {

    private E[] elements;
    private int size;
    private static final int INITIAL_CAPACITY = 100;

    public ArrayStack() {
        elements = (E[]) new Object[INITIAL_CAPACITY];
    }

    public ArrayStack(int capacity) {
        elements = (E[]) new Object[capacity];
    }

    public boolean isEmpty() {
        return (size == 0);
    }

    public E peek() {
        if (size == 0) {
            throw new java.util.EmptyStackException();
        }
        return elements[size - 1]; // top of stack
    }

    public E pop() {
        if (size == 0) {
            throw new java.util.EmptyStackException();
        }
        E element = elements[--size];
        elements[size] = null;
        return element;
    }

    public void push(E element) {
        if (size == elements.length) {
            resize();
        }
        elements[size++] = element;
    }
```

```
    public int size() {
        return size;
    }

    private void resize() {
        assert size == elements.length;
        Object[] a = new Object[2 * size];
        System.arraycopy(elements, 0, a, 0, size);
        elements = (E[]) a;
    }
}
```

Exercises

1. Using Linked list operations, write LinkedStack class which implements the previous Stack interface using a linked list. Note. Use the same Stackmain.java, just change the type of the stack object and do any necessary changes.

```
public class LinkedStack<E> implements StackInterface<E> {

	private SingleLinkedList stackList;
	private Node top; // check the appendix for the required classes
	private int size;

	public LinkedStack() { // constructor
		stackList = new SingleLinkedList();
	}

	public boolean isEmpty() {
		return (size == 0);
	}

	public E peek() {
		if (size == 0) {
			throw new java.util.EmptyStackException();
		}
		return (E) top.data; // top of stack
	}

	public E pop() {
		if (isEmpty()) {
			throw new java.util.EmptyStackException();
		}
		E element = (E) top.data;
		stackList.deleteFirst();
		top = stackList.head;
		--size;
		return element;
	}

	public void push(E element) {
		stackList.insertFirst(element);
		top = stackList.head;
```

```
            ++size;
        }

        public int size() {
            return size;
        }

        public void display() {
            Node c = top;
            while (c != null) {
                System.out.print(c.data);
                c = c.next;
            }
            System.out.println();

        }
    }
```

Homework

1. Write `reverseStack(Stack stack)` method, which reverse the items of a given stack so the top is now at the bottom of the stack

```
//proper header
{
    // declare a new stack rs
    while (!stack.isEmpty())
        rs.push(stack.pop());
    //display the new reversed stack
}
```

2. Write `displayStack(Stack stack)` method, which displays the items of a given stack without emptying the stack.

```
    static void displayStack(LinkedStack stack) {
            LinkedStack tempStack = new LinkedStack();
            if (stack.isEmpty()) {
                System.out.println("Stack is empty!");
                return;
            }
            while (!stack.isEmpty()) {
                Object item = stack.pop();
                System.out.print(item);
                tempStack.push(item);
            }
    // another solution: you may return the tempStack after calling the reverse
    method
            while (!tempStack.isEmpty())
                stack.push(tempStack.pop());
            System.out.println();

        }
```

Challenge Question

1. One common use for stacks is to parse certain kinds of text strings. Write a java program that uses a stack object to check the delimiters in a line of text typed by the user. The delimiters are the braces '{'and'}', brackets '['and']', and parentheses '('and')'. Each opening or left delimiter should be matched by a closing or right delimiter; that is, every '{' should be followed by a matching '}' and so on. Also, opening delimiters that occur later in the string should be closed before those occurring earlier. Examples:
    - c[d] // correct
    - a{b[c]d}e // correct
    - a{b(c]d}e // not correct;] doesn't match (
    - a[b{c}d]e} // not correct; nothing matches final }
    - a{b(c) // not correct; Nothing matches opening {

```
import java.io.*;

//stacks used to check matching brackets
class BracketChecker {
   private String input; // input string

   public BracketChecker(String in) // constructor
   {
          input = in;
   }
//-------------------------------------------------------------

   public void check() {
          LinkedStack<Character> theStack = new LinkedStack<Character>(); // make
stack
          int errorCount = 0;
          for (int j = 0; j < input.length(); j++) // get chars in turn
          {

                char ch = input.charAt(j); // get char
                switch (ch) {
                case '{': // opening symbols
                case '[':
                case '(':
                      theStack.push(ch); // push them

                      break;
                case '}': // closing symbols
                case ']':
                case ')':

                      if (!theStack.isEmpty()) // if stack not empty
                      {
                            char chx = (char) theStack.pop(); // pop and check
                            if ((ch == '}' && chx != '{') || (ch == ']' && chx
!= '[') || (ch == ')' && chx != '(')) {
```

```
                                    System.out.println("Error: " + ch + " at " +
j);
                                    errorCount++;
                              }

                        } else // prematurely empty
                        {
                              System.out.println("Error: " + ch + " at " + j);
                              errorCount++;
                        }
                        break;
                  default: // no action on other characters
                        break;
                  } // end switch
            } // end for
// at this point, all characters have been processed
            if (!theStack.isEmpty())
                  System.out.println("Error: missing right delimiter");
            else if (errorCount == 0)
                  System.out.println("All delimiters match");
      } // end check()
//--------------------------------------------------------------

} // end class BracketChecker
////////////////////////////////////////////////////////////////

public class Brackets {
      public static void main(String[] args) throws IOException {
            String input;
            while (true) {
                  System.out.print("Enter string containing delimiters [empty
strting to quit]: ");
                  System.out.flush();
                  input = getString(); // read a string from kbd
                  if (input.equals("")) // quit if [Enter]
                        break;
// make a BracketChecker
                  BracketChecker theChecker = new BracketChecker(input);
                  theChecker.check(); // check brackets
            } // end while
      } // end main()
//--------------------------------------------------------------

      public static String getString() throws IOException {
            InputStreamReader isr = new InputStreamReader(System.in);
            BufferedReader br = new BufferedReader(isr);
            String s = br.readLine();
            return s;
      }
//--------------------------------------------------------------

} // end class Brackets
```

Lab 07. Queue ADT

Examples

1. Queues provide FIFO (First-in-first-out) access where the first item in the set is accessed first. Regardless of the data structure we use to implement a queue, queues have two main operations: enqueue() and dequeue() . The following code implement a queue using an array. Apply and test the code.

```
//Queue.java: is a queue implementation using generic parameterized type E
    public interface Queue<E> {
        public void enqueue(E element);

        public E element();

        public boolean isEmpty();

        public E dequeue();

        public int size();

    }
// ArrayQueue.java: An ArrayQueue Implementation
    public class ArrayQueue<E> implements Queue<E> {
        private E[] elements;
        private int front;
        private int back;
        private static final int INITIAL_CAPACITY = 4;

        public ArrayQueue() {
               elements = (E[]) new Object[INITIAL_CAPACITY];
        }

        public ArrayQueue(int capacity) {
               elements = (E[]) new Object[capacity];
        }

        public void enqueue(E element) {
               if (size() == elements.length - 1) {
                      resize();
               }
               elements[back] = element;
               if (back < elements.length - 1) {
                      ++back;
               } else {
                      back = 0; // wrap
               }
        }

        public E element() {
               if (size() == 0) {
                      throw new java.util.NoSuchElementException();
               }
               return elements[front];
        }
```

```
	public boolean isEmpty() {
		return (size() == 0);
	}

	public E dequeue() {
		if (size() == 0) {
			throw new java.util.NoSuchElementException();
		}
		E element = elements[front];
		elements[front] = null;
		++front;
		if (front == back) { // queue is empty
			front = back = 0;
		}
		if (front == elements.length) { // wrap
			front = 0;
		}
		return element;
	}

	public int size() {
		if (front <= back) {
			return back - front;
		} else {
			return back - front + elements.length;
		}
	}

	private void resize() {
		int size = size();
		int len = elements.length;
		assert size == len;
		Object[] a = new Object[2 * len];
		System.arraycopy(elements, front, a, 0, len - front);
		System.arraycopy(elements, 0, a, len - front, back);
		elements = (E[]) a;
		front = 0;
		back = size;
	}

	public void printQueue() {
		System.out.print("[");
		String str = "";
		if (front <= back) {
			for (int i = front; i < back; i++)
				str += elements[i] + ",";
		} else {
			for (int i = front; i < elements.length; i++)
				str += elements[i] + ",";

			if (back != 0)
				for (int i = 0; i < back; i++)
					str += elements[i] + ",";
		}
		if (str.length() != 0)
```

```
                    str = str.substring(0, str.length() - 1);

                System.out.print(str + "]\n");

        }
    }
// QueueMain.java class: is the main class where we test a String Queue
    public class QueueMain{
        public static void main(String[] args) {
                ArrayQueue<String> queue = new ArrayQueue<String>();
                queue.enqueue("GB");
                queue.enqueue("DE");
                queue.enqueue("FR");
                queue.enqueue("ES");
                queue.printQueue();
                System.out.println("queue.element(): " + queue.element());
                System.out.println("queue.dequeue(): " + queue.dequeue());
                queue.printQueue();
                System.out.println("queue.dequeue(): " + queue.dequeue());
                queue.printQueue();
                System.out.println("queue.enqueue(\"IE\"): ");
                queue.enqueue("IE");
                queue.printQueue();
                System.out.println("queue.dequeue(): " + queue.dequeue());
                queue.printQueue();
        }

    }
```

Exercises

1. Using Linked list operations, write LinkedQueue class which implements the previous Queue interface using a linked list. Note. Use the same QueueMain.java, just change the type of the queue object.

```
public class LinkedQueue<E> implements Queue<E> {

	private SingleLinkedList<E> lnkdQ = new SingleLinkedList<E>() ;
	private Node<E> front;
	private Node<E> rare; // check the appendix for the required classes
	private int size;

	public void enqueue(E element) {

		if (size == 0) {
			lnkdQ.insertLast(element);
			front = rare = lnkdQ.head;
		} else {
			lnkdQ.insertLast(element);
			rare = lnkdQ.tail;
		}
		++size;
	}

	public E element() {// front of queue
		if (size == 0) {
			throw new java.util.EmptyStackException();
		}
		return front.data;
	}

	public boolean isEmpty() {
		return (size == 0);
	}

	public E dequeue() {
		if (size == 0) {
			throw new java.util.EmptyStackException();
		}
		E element = front.data;
		lnkdQ.deleteFirst();
		front = lnkdQ.head;
		--size;
		return element;
	}

	public int size() {
		return size;
	}


	public void printQueue() {

		System.out.print("[");
```

```
            String str = "";
            if (!isEmpty()) {
                  Node<E> current = front;
                  for (int i = 0; i < size; i++) {
                        str += current.data + ",";
                        current = current.next;
                  }
            }
            if (str.length() != 0)
                  str = str.substring(0, str.length() - 1);
            System.out.print(str + "]\n");

      }
}
```

Homework

1. Write `StackQueue.java`, a class that implements a queue of integer numbers using the Stack structure. Write the prober java code to test your code. You may use `java.util.Stack class`

```
import java.util.Stack;

public class StackQueue {

    Stack<Integer> s1 = new Stack<Integer>();
    Stack<Integer> s2 = new Stack<Integer>();

    void enQueue(int x) {
          // Move all elements from s1 to s2
          while (!s1.isEmpty()) {
                s2.push(s1.pop());

          }

          // Push item into s1
          s1.push(x);

          // Push everything back to s1
          while (!s2.isEmpty()) {
                s1.push(s2.pop());

          }
    }

    // Dequeue an item from the queue
     int deQueue() {
          // if first stack is empty
          if (s1.isEmpty()) {
                System.out.println("Q is Empty");
                System.exit(0);
          }

          // Return top of s1
          int x = s1.peek();
          s1.pop();
```

```
            return x;
      }
      // Driver code
      public static void main(String[] args) {
            StackQueue q = new StackQueue();
            q.enQueue(1);
            q.enQueue(2);
            q.enQueue(3);

            System.out.println(q.deQueue());
            System.out.println(q.deQueue());
            System.out.println(q.deQueue());
      }
}
```

2. Write `displayQueue(Queue queue)` method, which displays the items of a given queue without emptying the queue.

```
static void displayQueue(LinkedQueue queue) {
            LinkedQueue tempQueue = new LinkedQueue();
            if (queue.isEmpty()) {
                  System.out.println("Queue is empty!");
                  return;
            }
            while (!queue.isEmpty()) {
                  Object item = queue.dequeue();
                  System.out.print(item);
                  tempQueue.enqueue(item);
            }
            while (!tempQueue.isEmpty())
                  queue.enqueue(tempQueue.dequeue());
            System.out.println();

      }
```

Challenge Question

1. Write `Queue reverseQueue(Queue queue)`, a static method which returns a queue in which the order of the elements is reversed. Write a suitable main() to test your code.

```
static LinkedQueue reverseQueue(LinkedQueue queue) {
            Stack<Object> s = new Stack<Object>();
            LinkedQueue tempQueue = new LinkedQueue();
            if (queue.isEmpty()) {
                  System.out.println("Queue is empty!");
                  return null;
            }
            while (!queue.isEmpty()) {
                  s.push(queue.dequeue());
            }
            while (!s.isEmpty())
                  tempQueue.enqueue(s.pop());
            return tempQueue;
      }
```

Lab 08. Tree ADT

Examples

1. The following program shows how a binary tree can be implemented.

```
File: TestBST.java
public class TestBST {

	public static void main(String[] args) {

		BST T = new BST();
		T.insert(5);
		T.insert(3);
		T.insert(9);
		T.insert(1);
		T.insert(4);
		T.insert(6);
		System.out.println("The root of Bi-Tree is: " + (T.root()));
		System.out.println("In-order traversal sequence :");
		T.inOrder();
		System.out.println("Pre-order traversal sequence :");
		T.preOrder();
		System.out.println("Post-order traversal sequence :");
		T.postOrder();
		System.out.println("Level-order traversal sequence :");
		T.levelOrder();
		System.out.println("In-order traversal sequence (after mirroring) :");
		T.mirror();
		T.inOrder();
	}
}

File: BST.java
public class BST {
	// Root node pointer. Will be null for an empty tree.
	private Node root;
	/*
	 * --Node-- The binary tree is built using this nested node class. Each node
	 * stores one data element, and has left and right sub-tree pointer which may be
	 * null. The node is a "dumb" nested class -- we just use it for storage; it
	 * does not have any methods.
	 */
	public static class Node {
		Node left;
		Node right;
		int data;
		Node(int newData) {
			left = null;
			right = null;
			data = newData;
		}
	}
	/**
	 * Creates an empty binary tree -- a null root pointer.
	 */
```

```
	public void BST() {
		root = null;
	}
	/**
	 * Inserts the given data into the binary tree. Uses a recursive helper.
	 */
	public void insert(int data) {
		root = insert(root, data);
	}
	/**
	 * Recursive insert -- given a node pointer, recur down and insert the given
	 * data into the tree. Returns the new node pointer (the standard way to
	 * communicate a changed pointer back to the caller).
	 */
	private Node insert(Node node, int data) {
		if (node == null) {
			node = new Node(data);
		} else {
			if (data <= node.data) {
				node.left = insert(node.left, data);
			} else {
				node.right = insert(node.right, data);
			}
		}
		return (node); // in any case, return the new pointer to the caller
	}
	/**
	 * Returns the number of nodes in the tree. Uses a recursive helper that recurs
	 * down the tree and counts the nodes.
	 */
	public int size() {
		return (size(root));
	}

	private int size(Node node) {
		return 0;

		// Complete the code here

	}
	/**
	 * Returns true if the given target is in the binary tree. Uses a recursive
	 * helper.
	 */
	public boolean lookup(int data) {
		return (lookup(root, data));
	}
	/**
	 * Recursive lookup -- given a node, recur down searching for the given data.
	 */
	private boolean lookup(Node node, int data) {
		if (node == null) {
			return (false);
		}
		if (data == node.data) {
```

```
                return (true);
        } else if (data < node.data) {
                return (lookup(node.left, data));
        } else {
                return (lookup(node.right, data));
        }
}
/**
 * Prints the node values in the "inorder" order. Uses a recursive helper to do
 * the traversal.
 */
public void inOrder() {
        inorderTree(root);
        System.out.println();
}
private void inorderTree(Node node) {
        if (node == null) {
                return;
        }
        // left, node itself, right
        inorderTree(node.left);
        System.out.print(node.data + "  ");
        inorderTree(node.right);
}

/**
 * Prints the node values in the "preOrder" order. Uses a recursive helper to do
 * the traversal.
 */
public void preOrder() {
        preOrder(root);
        System.out.println();
}

public void preOrder(Node node) {

        // Complete the code here

}
/**
 * Prints the node values in the "postOrder" order. Uses a recursive helper to
 * do the traversal.
 */
public void postOrder() {
        postOrder(root);
        System.out.println();
}
public void postOrder(Node node) {

        // Complete the code here

}
/**
 * Prints the node values in the "levelOrder" order. Uses a helper to do the
 * traversal.
```

```
	 */
	public void levelOrder() {
		levelOrder(root);
		System.out.println();
	}
	public void levelOrder(Node node) {
		if (node != null) {
			Queue<Node> q = new ArrayDeque<Node>();
			q.add(node);
			while (q.size() != 0) {
				Node currentNode = q.remove();
				System.out.print(currentNode.data + "  ");
				if (currentNode.left != null) {
					q.add(currentNode.left);
				}
				if (currentNode.right != null) {
					q.add(currentNode.right);
				}
			}
		}
	}
	/**
	 * Changes the tree into its mirror image. Uses a recursive helper that recurs
	 * over the tree, swapping the left/right pointers.
	 */
	public void mirror() {

		// write the code here

	}
	private Node mirror(Node node) {

		// write the code here

	}
	public int root() {
		// TODO Auto-generated method stub
		return root.data;
	}
}
```

Exercises

1. Write the proper java code to implement the unfinished methods in the previous program.

```
private int size(Node node) {
             if (node == null)
               return 0;
           else
               return(size(node.left) + 1 + size(node.right));

      }

public void preOrder(Node node) {

             if (node == null) {
                   return;
             }
             System.out.print(node.data + "  ");
             preOrder(node.left);
             preOrder(node.right);
      }
public void postOrder(Node node) {
             if (node == null) {
                   return;
             }
             postOrder(node.left);
             postOrder(node.right);
             System.out.print(node.data + "  ");
      }
public void mirror() {

             root = mirror(root);

      }

private Node mirror(Node node) {

             if (node == null)
                   return node;

             /* do the subtrees */
             Node left = mirror(node.left);
             Node right = mirror(node.right);

             /* swap the left and right pointers */
             node.left = right;
             node.right = left;

             return node;

      }
```

Homework

1. Write `deleteNode(int key)` member method, which deletes a node from the BST. If the node to be deleted has 2 subtrees, then the candidate replacement is the minimum of the right subtree. Write any helper methods needed.

```
public void deleteNode(int key) {
		deleteNode(root, key);
	}

	public Node deleteNode(Node n, int key) {
		if (n == null)
			return n;
		if (key < n.data)
			n.left = deleteNode(n.left, key);
		else if (key > n.data)
			n.right = deleteNode(n.right, key);
		else {
			if (n.left == null)
				return n.right;
			else if (n.right == null)
				return n.left;
			n.data = minimum(n.right);
			n.right = deleteNode(n.right, n.data);
		}
		return n;
	}


	private int minimum(Node node) {
		if (node.left == null)
			return node.data;

		return minimum(node.left);

	}
```

2. Write int findMax(BST t) which finds the maximum value of a BST

```
private int findMax(Node node) {
		if (node.right == null)
			return node.data;

		return findMax(node.right);

	}
```

3. Write `insertByLevel(int i)` which inserts nodes in a binary tree level by level

```
public void insertByLevel(int i) {
          if (root == null) {
                root = new Node(i);
          } else insertByLevel(root,i);
}

private void insertByLevel(Node temp, int key)
{
        Queue<Node> q = new LinkedList<Node>();
        q.add(temp);
        // Do level order traversal until we find an empty place.
        while (!q.isEmpty()) {
            temp = q.peek();
            q.remove();

            if (temp.left == null) {
                temp.left = new Node(key);
                break;
            } else
                q.add(temp.left);

            if (temp.right == null) {
                temp.right = new Node(key);
                break;
            } else
                q.add(temp.right);
        }
    }
```

Challenge Question

1. Write `int` findMax(BT t) which finds the maximum value of a Binary tree.

```
public int findMax(Node node)
{
            if (node == null)
                return Integer.MIN_VALUE; //dummy value

            int currentValue = node.data;
            int maxLeft = findMax(node.left);
            int maxRight = findMax(node.right);

            if (maxLeft > currentValue)
                currentValue= maxLeft;
            if (maxRight > currentValue)
                currentValue= maxRight;
            return res;
}
public int findMax() {
              return findMax(root);
}
```

Lab 09. Graphs

Examples

1. The following program shows how the di-graph can be implemented using adjacency lists. Run the program and make sure you understand how it works.

```
public class GraphTraversal {
    // Representation of a directed graph using adjacency list

    public static void main(String[] args) {
           DiGraph g = new DiGraph(10);

           g.addEdge(1, 0);
           g.addEdge(0, 2);
           g.addEdge(2, 1);
           g.addEdge(0, 3);
           g.addEdge(1, 4);
           g.addEdge(4, 5);
           g.addEdge(4, 6);
           g.addEdge(4, 7);
           g.addEdge(6, 7);
           g.addEdge(6, 8);
           g.addEdge(8, 9);
           g.addEdge(9, 9);
           g.printGraph();
           System.out.println("Depth First Search visit order: ");
           g.DFS(0);
           System.out.println();
           System.out.println("Breadth First Search visit order: ");
           g.BFS(0);
    }
}

class DiGraph {
    int V; // Number of Vertices

    LinkedList<Integer> adj[]; // adjacency lists

    // Constructor
    DiGraph(int V) {
           this.V = V;
           adj = new LinkedList[V];

           for (int i = 0; i < adj.length; i++)
                  adj[i] = new LinkedList<Integer>();

    }

    // To add an edge to graph
    void addEdge(int v, int w) {
           adj[v].add(w); // Add w to v's list.
    }

    void DFS(int s) {
// Write the proper code for DFS
```

```
    }

    void BFS(int s) {
// Write the proper code for BFS
    }

    public void printGraph() {

        for (int src = 0; src < adj.length; src++) {
            System.out.print(src);
            for (Integer dest : adj[src]) {
                System.out.print(" -> " + dest);
            }
            System.out.println();
        }
    }
}
```

Exercises

1. Write the proper code for DFS method in the program of the previous section.

```
    void DFS(int s) {
              // Initially mark all vertices as not visited
              Vector<Boolean> visited = new Vector<Boolean>(V);
              for (int i = 0; i < V; i++)
                    visited.add(false);

              // Create a stack for DFS
              Stack<Integer> stack = new Stack<>();

              // Push the current source node
              stack.push(s);

              while (stack.empty() == false) {
                    // Pop a vertex from stack and print it
                    s = stack.peek();
                    stack.pop();

                    /*
                     * Stack may contain same vertex twice. So we print the popped
    item only if it
                     * is not visited.
                     */
                    if (visited.get(s) == false) {
                          System.out.print(s + " ");
                          visited.set(s, true);
                    }

                    /*
                     * Get all adjacent vertices of the popped vertex s If a
    adjacent has not been
                     * visited, then push it into the stack.
                     */
                    Iterator<Integer> itr = adj[s].iterator();

                    while (itr.hasNext()) {
                          int v = itr.next();
                          if (!visited.get(v))
                                stack.push(v);
                    }

              }
       }
```

Homework

1. Write the proper code to implement BFS method in the program of the Example section

```
void BFS(int s) {

		// Mark all the vertices as not visited(By default set as false)

		boolean visited[] = new boolean[V];

		// Create a queue for BFS
		LinkedList<Integer> queue = new LinkedList<Integer>();

		// Mark the current node as visited and enqueue it
		visited[s] = true;
		queue.add(s);

		while (queue.size() != 0) {
			// Dequeue a vertex from queue and print it
			s = queue.poll();
			System.out.print(s + " ");

			/*
			 * Get all adjacent vertices of the dequeued vertex s If a
adjacent has not been
			 * visited, then mark it visited and enqueue it
			 */
			Iterator<Integer> i = adj[s].listIterator();
			while (i.hasNext()) {
				int n = i.next();
				if (!visited[n]) {
					visited[n] = true;
					queue.add(n);
				}
			}
		}
	}
```

2. Rewrite the above program so it can represent undirected graphs.

Just add the following line in the method `adj[w].add(v);`

Challenge Question

1. Is it possible to use the same graph program to implement a k-ary tree? Explain!

By definition, a tree is an acyclic connected directed graph. We can implement tree structures using DAGs.
Example.
Straightforward application using the graph program

Lab 10. Searching and Sorting Algorithms

Examples

3. The following program shows how linear searching and binary searching can be implemented. Apply and check the code and the output.

```
public static void main(String[] args) {
		// TODO Auto-generated method stub
		String[] Names = { "YAHYA", "HASAN", "EID", "MANAL", "BDOOR", "RAWIAH", "HAIFA",
"AMJAD", "FAHAD", "OHOOD","MAHA", "SARA", "AMAL", "MEHAD", "ZAIN" };
		System.out.println(linearSearch(Names, 0, Names.length, "AMAL") ? "Found!" : "Not
Found!");

		Integer[] ID = { 3692098, 3692102, 3692104, 3692107, 3692112, 3692256, 3692449,
3692643, 3693199, 3757224,3757225, 3757622, 3757623, 3757625, 3757628, 3757629, 3757631 };
	System.out.println(linearSearch(ID, 0, ID.length,3692107 ) ? "Found!" : "Not Found!");
	System.out.println(binarySearch(ID, 0, ID.length,3692107 ) ? "Found!" : "Not Found!");
	System.out.println(binarySearch(ID, 0, ID.length,3692999 ) ? "Found!" : "Not Found!");
	}

	public static <T> boolean linearSearch(T[] data, int min, int max, T target) {
		int index = min;
		boolean found = false;
		while (!found && index <= max) {
			found = data[index].equals(target);
			index++;
		}
		return found;
	}

	public static <T extends Comparable<T>> boolean binarySearch(T[] data, int min, int max,
T target) {
		boolean found = false;
		int midpoint = (min + max) / 2; // determine the midpoint
		if (data[midpoint].compareTo(target) == 0)
			found = true;
		else if (data[midpoint].compareTo(target) > 0) {
			if (min <= midpoint - 1)
				found = binarySearch(data, min, midpoint - 1, target);
		} else if (midpoint + 1 <= max)
			found = binarySearch(data, midpoint + 1, max, target);
		return found;
	}

}
```

Exercises

1. In an unordered list of items, you must check every item until you find a match. How can you optimize linear search if applied on an ordered list of items? Show your implementation.

```
   Idea. If the list is ordered ascendingly, then once a larger item is reached
   the search should terminate and vice versa.

public static <T extends Comparable<T>> boolean linearSearch(T[] data, int min, int
max, T target) {
              int index = min;
              boolean found = false;
              while (!found && index <= max) {
                     if (data[index].compareTo(target) > 0)
                            return false;
                     found = data[index].equals(target);
                     index++;
              }
              return found;
       }
```

2. Write the proper java code to implement Bubble sort. Write a suitable main to test your code.

```
   public static <T extends Comparable<T>> void bubbleSort(T a[], int length) {
              for (int i = 0; i < (length - 1); i++) {
                     for (int j = length - 1; j > i; j--) {
                            if (a[j - 1].compareTo( a[j])>0) {
                                   T temp = a[j - 1];
                                   a[j - 1] = a[j];
                                   a[j] = temp;
                            }
                     }
              }
       }
```

3. Write the proper java code to implement Selection sort. Write a suitable main to test your code.

```
public static <T extends Comparable<T>> void selectionSort(T[] data) {
              int min;
              T temp;
              for (int index = 0; index < data.length - 1; index++) {
                     min = index;
                     for (int scan = index + 1; scan < data.length; scan++)
                            if (data[scan].compareTo(data[min]) < 0)
                                   min = scan;
                     swap(data, min, index);
              }
       }
private static <T extends Comparable<T>> void swap(T[] data, int index1, int index2) {
              T temp = data[index1];
              data[index1] = data[index2];
              data[index2] = temp;
       }
```

Homework

Write the proper java code to implement the following sorting algorithms. Use the array

```
String[] Names = { "YAHYA", "HASAN", "EID", "MANAL", "BDOOR", "RAWIAH", "HAIFA", "AMJAD",
                   "FAHAD", "OHOOD","MAHA", "SARA", "AMAL", "MEHAD", "ZAIN" };
```

to test your code. Display the content of the array before and after the sorting process.

1. Insertion sort.

```
public static <T extends Comparable<T>> void insertionSort(T[] data) {
          for (int index = 1; index < data.length; index++) {
                T key = data[index];
                int position = index;
                // shift larger values to the right
                while (position > 0 && data[position - 1].compareTo(key) > 0) {
                      data[position] = data[position - 1];
                      position--;
                }
                data[position] = key;
          }
    }
}
```

2. Quick sort.

```
public static <T extends Comparable<T>> void quickSort(T[] data) {
          quickSort(data, 0, data.length - 1);
    }

    private static <T extends Comparable<T>> void quickSort(T[] data, int min,
int max) {
          if (min < max) {
                // create partitions
                int indexofpartition = partition(data, min, max);
                // sort the left partition (lower values)
                quickSort(data, min, indexofpartition - 1);
                // sort the right partition (higher values)
                quickSort(data, indexofpartition + 1, max);
          }
    }

    private static <T extends Comparable<T>> int partition(T[] data, int min,
int max) {
          T partitionelement;
          int left, right;
          int middle = (min + max) / 2;
          // use the middle data value as the partition element
          partitionelement = data[middle];
          // move it out of the way for now
          swap(data, middle, min);
          left = min;
          right = max;
          while (left < right) {
                // search for an element that is > the partition element
                while (left < right && data[left].compareTo(partitionelement)
<= 0)
```

```
                        left++;
                // search for an element that is < the partition element
                while (data[right].compareTo(partitionelement) > 0)
                        right--;
                // swap the elements
                if (left < right)
                        swap(data, left, right);
        }
        // move the partition element into place
        swap(data, min, right);
        return right;
    }
```

3. Merge sort.

```
    public static <T extends Comparable<T>> void mergeSort(T[] data) {
        mergeSort(data, 0, data.length - 1);
    }

    private static <T extends Comparable<T>> void mergeSort(T[] data, int min,
 int max) {
        if (min < max) {
            int mid = (min + max) / 2;
            mergeSort(data, min, mid);
            mergeSort(data, mid + 1, max);
            merge(data, min, mid, max);
        }
    }

    private static <T extends Comparable<T>> void merge(T[] data, int first, int
 mid, int last) {
        T[] temp = (T[]) (new Comparable[data.length]);
        int first1 = first, last1 = mid; // endpoints of first subarray
        int first2 = mid + 1, last2 = last; // endpoints of second subarray
        int index = first1; // next index open in temp array
        // Copy smaller item from each subarray into temp until one
        // of the subarrays is exhausted
        while (first1 <= last1 && first2 <= last2) {
            if (data[first1].compareTo(data[first2]) < 0) {
                temp[index] = data[first1];
                first1++;
            } else {

                temp[index] = data[first2];
                first2++;

            }
            index++;
        }
        // Copy remaining elements from first subarray, if any
        while (first1 <= last1) {
            temp[index] = data[first1];
            first1++;
            index++;
        }
        // Copy remaining elements from second subarray, if any
```

```
            while (first2 <= last2) {
                  temp[index] = data[first2];
                  first2++;
                  index++;
            }
            // Copy merged data into original array
            for (index = first; index <= last; index++)
                  data[index] = temp[index];
      }
```

Challenge Question

1. Banks often record transactions on an account in order of the times of the transactions, but many people like to receive their bank statements with checks listed in order by check numbers. People usually write checks in order by check number, and merchants usually cash them with reasonable dispatch. The problem of converting time-of-transaction ordering to check-number ordering is therefore the problem of sorting almost-sorted input. Argue that insertion sorting would be more efficient than quick sorting on this problem.

The more sorted the array is, the less work insertion sort will do. Namely, INSERTION-SORT is Θ(n+d), where d is the number of inversions in the array. In the example above the number of inversions tends to be small so insertion sort will be close to linear.
On the other hand, if PARTITION does pick a pivot that does not participate in an inversion, it will produce an empty partition. Since there is a small number of inversions, QUICKSORT is very likely to produce empty partitions.

Appendix

File: SingleLinkedList.java

```
public class SingleLinkedList<E> {

	public Node<E> head;
	public Node<E> tail;

	public void insertFirst(E data) {// creates the node within the method
		Node<E> newNode = new Node<E>(data);
		if (head == null)
			head = tail = newNode;
		else {

			newNode.next = head;
			head = newNode;
		}
	}

	public void insertLast(E data) {
		Node<E> newNode = new Node<E>(data);
		if (head == null)
			head = tail = newNode;
		else {

			tail.next = newNode;
			tail = newNode;
		}
	}

	public void deleteFirst() {
		if (head == null)
			System.out.println("List is empty!!");
		else if (head == tail)
			head = tail = null;
		else
			head = head.next;

	}

	public void deleteLast() {
		if (head == null)
			System.out.println("List is empty!!");
		else if (head == tail)
			head = tail = null;
		else {
			Node<E> cur = head;
			for (; cur.next != tail; cur = cur.next)
				;
			cur.next = null;
			tail = cur;
		}
	}
```

```
	public void displayList() {
		if (head == null) {
			System.out.println("List is empty!!");
			return;
		}
		Node<E> cur = head;
		while (cur != null) {
			System.out.print(cur.data + " -> ");
			cur = cur.next;
		}
		System.out.println("/");
	}

}
```

File: Node.java

```
public class Node<E> {
	public E data;
	public Node<E> next = null;

	Node(E data2) {
		data = data2;
	}
}
```