**Data Structures Lab**

This document contains the functional test cases of Data Structures Lab. It includes testing of internal and external links availability, also the expected output with valid inputs.

No. of Experiments: 9

**Test Environment:**

Operating Systems: Windows, CentOS, Ubuntu

Browsers: Chrome, Firefox

Landing page: All the links given on landing page of lab are working fine.

**Experiment wise Test Cases**

1. **Number System**

**Test Summary**:

To understand how various data structures work. To understand some important applications of various data structures.

**Test Procedure:**

Click on procedure tab.

Follow the instructions given to perform the experiment by giving valid input.

Simulation should give the expected output to pass the test case.

**Test Data:**

<http://cse01-iiith.vlabs.ac.in/exp1/UnarySystem.html>

<http://cse01-iiith.vlabs.ac.in/exp1/RomanSystem.html>

<http://cse01-iiith.vlabs.ac.in/exp1/BinarySystem.html>

**Links Availability:**

All links are working fine.

**Procedure**

Procedure is to follow the “instructions” given in the lab to run “simulations” by giving valid input to get the expected output.

**Input**

Click on the Unary System link. Give unary values and click on add

1111 1111

The two numbers will concatenate

**Expected output:**

11111111

**Test pass or Fail?**

**PASS**

**Comments:**

1. Instructions given in simulation are not clear. Needs to define properly.

**2.** [**Expression Evaluation using Stacks**](http://cse01-iiith.vlabs.ac.in/exp2/Introduction.html?domain=Computer%20Science&lab=Data%20Structures) **:**

**Test Summary**:

This experiment guides about stacks in a practical example. Stacks are an important data structure and finds applications where essentially the data item that is stored latest is what needs to be accessed first.

**Test Procedure:**

Click on procedure tab.

Follow the instructions given to perform the experiment by giving valid input.

Simulation should give the expected output to pass the test case.

**Test Data:**

<http://cse01-iiith.vlabs.ac.in/exp2/calc3.html>

**Links Availability:**

All other links are working fine. Not a single broken link found.

**Procedure**

Procedure is to follow the “instructions” given in the lab to run “simulations” by giving valid input to get the expected output.

**Input**

2+3

**Procedure:**

After giving input, it will append the expression to post fix

**Expected output:**

Postfix exp = 2 3 +

**Test pass or Fail?**

**PASS**

**Comments:**

No Comments

**3. Sorting Using Arrays :**

**Test Summary**:

1.  There are several sorting algorithms.

2.  This experiment will cover merge sort, insertion sort, and quick sort .

**Test Procedure:**

Click on procedure tab.

Follow the instructions given to perform the experiment by giving valid input.

Simulation should give the expected output to pass the test case.

**Test Data:**

<http://cse01-iiith.vlabs.ac.in/exp3/quick.html>

<http://cse01-iiith.vlabs.ac.in/exp3/merge.html>

<http://cse01-iiith.vlabs.ac.in/exp3/insertion.html>

**Links Availability:**

All other links are working fine. Not a single broken link found.

**Procedure**

Procedure is to follow the “instructions” given in the lab to run “simulations” by giving valid input to get the expected output.

**Input**

Click on Quick Sort, Merge Sort, Insertion Sort . Then we can see the best case, worst case, random input.

Quick Sort :

1. Select the best case and worst case.
2. Click on play button.
3. It will show how the quick sort algorithm works.
4. Click on Random Input, give values and click on play button.
5. It will show how the quick sort algorithm works.
6. It will place the array of values in ascending order.

**Sample input to test simulation:**

12, 3, 1, 5, 7, 4, 2

**Function**:

It will select the elements from pivot and then swap the elements, replaces the array in order.

**Expected output:**

**1, 2,3,4,5,7,12**

**Test pass or Fail?**

**PASS**

**Comments:**

There are no proper instructions to perform the experiment.

**4. Polynomials via Linked Lists :**

**Test Summary**:

1. To understand linked lists or if you want to see a realistic, applied example of pointer-intensive code.

2. To learn how to use linked lists to add and multiply (sparse) polynomials.

**Test Procedure:**

Click on procedure tab.

Follow the instructions given to perform the experiment by giving valid input.

Simulation should give the expected output to pass the test case.

**Test Data:**

<http://cse01-iiith.vlabs.ac.in/exp4/polynomial.html>

**Links Availability:**

All other links are working fine. Not a single broken link found.

**Procedure**

Procedure is to follow the “instructions” given in the lab to run “simulations” by giving valid input to get the expected output.

**Input**

1. Enter values

**Sample input to test simulation:**

D, 3::a, 2::b, 1::c, 0

D, 4::a, 3::b, 2::c , 0

**Procedure:**

After giving input, user needs to click on enter button then the simulation runs.

**Expected output:**

C , 0 => null

**Test pass or Fail?**

**PASS**

**Comments:**

No Comments

5. **Expression Trees**

**Test Summary**:

1. To know how to construct expression trees

2. To know how to evaluate an expression using the tree.

.

**Test Procedure:**

To generate an expression tree, given an expression, we first convert the expression to its postfix form. From the postfix expression, we read one symbol at a time from left to right. If the current symbol is an operand, then we push a tree of one node consisting of the operator onto a stack.

**Test Data:**

<http://cse01-iiith.vlabs.ac.in/exp6/exptree.html>

**Links Availability:**

All other links are working fine. Not a single broken link found.

**Procedure**

Procedure is to follow the “instructions” given in the lab to run “simulations” by giving valid input to get the expected output.

**Input**

Give an expression and click on play button, then it constructs an expression tree

Then click on pre-order,

Function:

private void preOrder(node localRoot){

if(localRoot != null){

localRoot.displayNode();

preOrder(localRoot.leftChild);

preOrder(localRoot.rightChild);

} }

Example Input:

4+5

**Expected output:**

+ 4 5

**Test pass or Fail?**

**PASS**

**Comments:**

No Comments

**6. Search Trees :**

**Test Summary**:

1.To know how to use trees for hierarchial data .

2.To know trees applications and its importance.

3. To know search trees efficient operations like insert, delete and find.

**Test Procedure:**

Procedure is to follow the “instructions” given in the lab to run “simulations” by giving valid input to get the expected output.

**Test Data :**

<http://cse01-iiith.vlabs.ac.in/exp5/binarySearchTree.html>

**Links Availability:**

All other links are working fine. Not a single broken link found.

**Input:**

1.Construct a tree by giving values and click on insert button.

2.If we want to search for a value at which position it is placed , give the value and click on search button.

3. We can insert or delete the values using same process.

**Expected output:**

It should insert or delete or search a value.

**Test pass or Fail?**

**PASS**

**Comments:**

No Comments

**7. Graph Traversals :**

**Test Summary**:

1. Perform a Breadth First Traversal on a graph

2. Perform a Depth First Traversal on a graph

3. Classify edges of the graph based on the traversal

**Test Procedure:**

Procedure is to follow the “instructions” given in the lab to run “simulations” by giving valid input to get the expected output.

**Test Data:**

<http://cse01-iiith.vlabs.ac.in/exp7/DFS_BFS.html>

**Links Availability:**

All other links are working fine. Not a single broken link found.

**Input**

1. Construct a graph.

2. Select algorithm.

3. Then follow the instructions to perform the experiment.

Function:

 for each v in V(G) do  
   from[v] = NIL; state[v] = NOT\_VISITED; d(v) = ∞  
 end-for  
  
 ds[S] = 0; state[S] = IN\_PROCESS; from[s] = NIL;  
 Q = EMPTY; Q.Enqueue(S);  
  
 While Q is not empty do  
   v = Q.Dequeue();  
   for each neighbour w of v do  
    if state[w] == NOT\_VISITED then  
     state[w] = IN\_PROCESS; from[w] = v; ds[w] = ds[v] + 1; Q.Enqueue(w);  
    end-if  
   state[v] = VISITED  
   end-for  
 end-whil*e*

**Expected output:**

In the DFS Graph start vertex should visit all the vertices.

**Test pass or Fail?**

**PASS**

**Comments:**

No Comments

**8. Shortest Paths in Graphs**

**Test Summary**:

1. To understand what a shortest path is

2. To perform Bellman-Ford single source shortest path on a graph

3. To perform Dijkstra single source shortest path on a graph

**Test Procedure:**

Procedure is to follow the “instructions” given in the lab to run “simulations” by giving valid input to get the expected output.

**Test Data:**

<http://cse01-iiith.vlabs.ac.in/exp8/Dijkstra.html>

**Links Availability:**

All other links are working fine. Not a single broken link found.

**Input**

1. Construct a graph.

2. Select algorithm.

3. Then follow the instructions to perform the experiment

**Function:**

 for all vertices v do   
   ds(v) = ∞; from(v) = NIL;  
 end-for  
 for n-1 iterations do  
   for each edge (v,w) do   
      if ds(w) > ds(v) + W(v,w) then  
        ds(w) = ds(v) + W(v,w); from(w) = v;  
      end-if  
   end-for  
 end-for

**Expected output:**

It should find the shortest path from the graph

**Test pass or Fail?**

**PASS**

**Comments:**

No Comments

**9. Minimum Spanning Tree**

**Test Summary**:

1. To know the concept of spanning trees and minimum spanning trees.
2. To understand algorithmic approaches to finding minimum spanning trees in graphs.
3. To understand the data structures required to efficiently implement algorithms for minimum spanning trees..

**Test Procedure:**

Procedure is to follow the “instructions” given in the lab to run “simulations” by giving valid input to get the expected output.

**Test Data:**

[https://cse02-iiith.vlabs.ac.in/exp10/index.html](https://cse02-iiith.vlabs.ac.in/exp2/index.html)

**Links Availability:**

All other links are working fine. Not a single broken link found.

**Input**

1. Construct a graph.

2. Select algorithm.

3. Select the edge we want to include in minimum spanning tree.

**Function:**

  sort the edges of G in increasing order of weight as e1, e2, ..., em  
  i = 1; E(T) = Φ  
  while |E(T)| < n-1 do   
   if E(T) ∪ ei does not have a cycle then  
     E(T) = E(T) ∪ ei  
   end-if  
   i = i + 1;  
  end-while

**Expected output:**

It should find a spanning tree of the edge that has the least cost.

**Test pass or Fail?**

**PASS**

**Comments:**

No comments.