DIGITAL CONTENT DEVELOPMENT FOR STUDYING DATA STRUCTURES AND ALGORITHMS

Submitted in partial fulfilment of the requirements for the degree of

BACHELOR OF ENGINEERING

in

INFORMATION TECHNOLOGY

by

Group No: 5

Roll No.	Name of Student
92	Nikhil Chandak
93	Darshil Shah
94	Akshay Desai
95	Rohan Fagwani

Supervisor:

Dr. Gopakumaran T. Thampi

(Principal, Thadomal Shahani Engineering College)

Department of Information Technology
Thadomal Shahani Engineering College
University Of Mumbai. 2014-2015

CERTIFICATE

This is to certify that the project entitled **"Digital Content Development for Data Structures and Algorithms"** is a bonafide work of Nikhil Chandak (92)**,** Darshil Shah (93), Akshay Desai (94), Rohan Fagwani (95) submitted to the University of Mumbai in partial fulfillment of the requirement for the award of the degree of **"Undergraduate"** in **"Bachelor of Engineering in Information Technology"**.

(Name and sign)
Supervisor/Guide

(Name and sign)
Co-Supervisor/Guide

(Name and sign)
Head of Department

(Name and sign)
Principal

Project Report Approval for B. E.

This project report entitled **"Digital Content Development For Data Structures And Algorithms"** by **Nikhil Chandak, Darshil Shah, Akshay Desai, Rohan Fagwani** is approved for the degree of **Bachelor of Engineering in Information Technology** under **University of Mumbai.**

Examiners

1.----------

2.----------

Date:

Place:

Department of Information Technology

Thadomal Shahani Engineering College

Bandra(W), Mumbai-400050

DECLARATION

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

(Signature)

ol

(Name of student and Roll No)

Abstract

Data Structures and Algorithms is a fundamental course in Computer Science and related fields. However, many students find it difficult because it requires abstract thinking. The best way to understand complex data structures is to see them in action. It would be very helpful if there was a visualization tool of data structures such as arrays, queues, stacks, trees and graphs for students to experiment with. This project is intended to create such an exploration environment, in which students can learn through experimentation. We've developed interactive animations for a variety of data structures and algorithms. Along with that we have provided simple and precise explanation of every data structure which goes with the animation for getting hold of the terms.

TABLE OF CONTENTS

LIST OF FIGURES

1 INTRODUCTION

1.1 Problem Statement

We present a visualization tool for the interactive learning of data structures and algorithmic schemes which can be used as a useful educational environment in Computer Science. The different parts of this tool intend to transmit to the students the importance of separating the specification and the implementation of data structures, facilitating the intuitive comprehension of the most typical implementations by using algorithmic schemes, and providing examples where such structures are used. The tool has been designed to allow teachers to monitor the whole educational process of the students, providing a personal training assistant and a range of tutoring techniques according to the student's response.

1.2 Scope of the Project

The application allows users to understand data structures and algorithm more easily and in a simplified manner. The user can select which data structure and algorithm is to be learnt from the given set of algorithms. Difficult algorithms can now be easily understood. With the huge potential in uses of data structures and algorithms, the applications of this system are extensive in the field of computer science.

1.3 Relevance Of Project

Real-World Data Storage:

By real-world data, we mean data that describes physical entities external to the computer. As some examples, a personnel record describes an actual human being, an inventory record describes an existing car part or grocery item, and a financial transaction record describes, say, an actual check written to pay the electric bill. A non-computer example of real-world data storage is a stack of 3-by-5 index cards. These cards can be used for a variety of purposes. If each card holds a person's name, address, and phone number, the result is an address book. If each card holds the name, location, and value of a household possession, the result is a home inventory.

Programmer's Tools

Not all data storage structures are used to store real-world data. Typically, real-world data is accessed more or less directly by a program's user. Some data storage structures, however, are not meant to be accessed by the user, but by the program itself. A programmer uses such structures as tools to facilitate some other operation. Stacks, queues, and priority queues are often used in this way. We'll see examples as we go along.

Real-World Modelling

Some data structures directly model real-world situations. The most important data structure of this type is the graph. You can use graphs to represent airline routes between cities or connections in an electric circuit or tasks in a project. Other data structures, such as stacks and queues, may also be used in simulations. A queue, for example, can model customers waiting in line at a bank or cars waiting at a toll booth.

Chapter 2

Review Of Literature

2.1 What is a Visualization of Data Structures and Algorithms?

Data Structures and Algorithms is a fundamental course in Computer Science. However,

many students find it difficult because it requires abstract thinking. It would be very helpful if there was a visualization tool of data structures such as arrays, queues, stacks, trees and graphs for students to experiment with. The tool would allow students to see how an element is inserted into or deleted from different data structures, how a tree is traversed in different order (pre-order, in-order, post-order, level-order), etc. Moreover, this tool would provide

a simple language, by which students can write their own algorithms so that the execution of the algorithm is animated. This project is intended to create such an exploration environment, in which students can learn through experimentation. This tool can be used as an effective supplement to the traditional classroom education and textbooks for Data Structures and Algorithms courses. The software package presented in this paper has the following functionality.

a. Provides complete visualization for the widely used data structures such as array,

stack, queue, tree, heap, graph, etc.

b. Provides the animation of common operations associated with the data

structures, such as inserting an element into and deleting an element from array, stack,

and queue.

c. Provides animation of simple user-defined algorithms.

2.1.1 State of the current Methodology

As we already know that study and proper understanding of Data Structures and Algorithms is very vital. Thus, In me self made virtualization tool. But still in most of the countries, Data Structures are being countries like USA and other European countries, Data Structures are being studied using so studied using traditional books and other such static resources. Thus it becomes very difficult for many students to comprehend the Data structures and its algorithms easily.

Data structure and algorithm visualizations and animations have a long history in computer science education. While the 1981 video “Sorting out Sorting” by Ronald Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Baeker was the first well-known visualization, ad hoc visualizations existed long before. The first recognized system for creating algorithm animations was BALSA in 1984.

Since then, hundreds of algorithm visualizations have been implemented and provided freely to educators, and scores (or hundreds) of papers have been written about them. It is widely perceived that algorithm visualizations can provide a powerful alternative to static written presentations (from textbooks) or verbal descriptions supported by illustrations (from lectures). There has been some debate in the literature as to whether algorithm visualizations are effective in practice. Some studies have shown the classic dismissal that is the downfall of most technological interventions in education: “no significant difference”. Other studies have shown that algorithm visualizations can indeed improve understanding of the fundamental data structures and algorithms that are part of a traditional computer science curriculum.

Certainly, many visualizations exist and are widely (and freely) available via the Internet. Unfortunately, the vast majority of those currently available serve no useful teaching purpose. So we see that (a) many algorithm visualizations exist, yet relatively few are of true value, and (b) algorithm visualizations can be demonstrated to have pedagogical value, yet it is also quite possible to use them in ways that have no teaching effect. These facts seem to imply that creating and deploying effective algorithm visualizations is difficult.

2.2 Description of Project

2.2.1 Design Details

The proposed visualization system is designed. The design of the website platform and the user terminal are described as below.

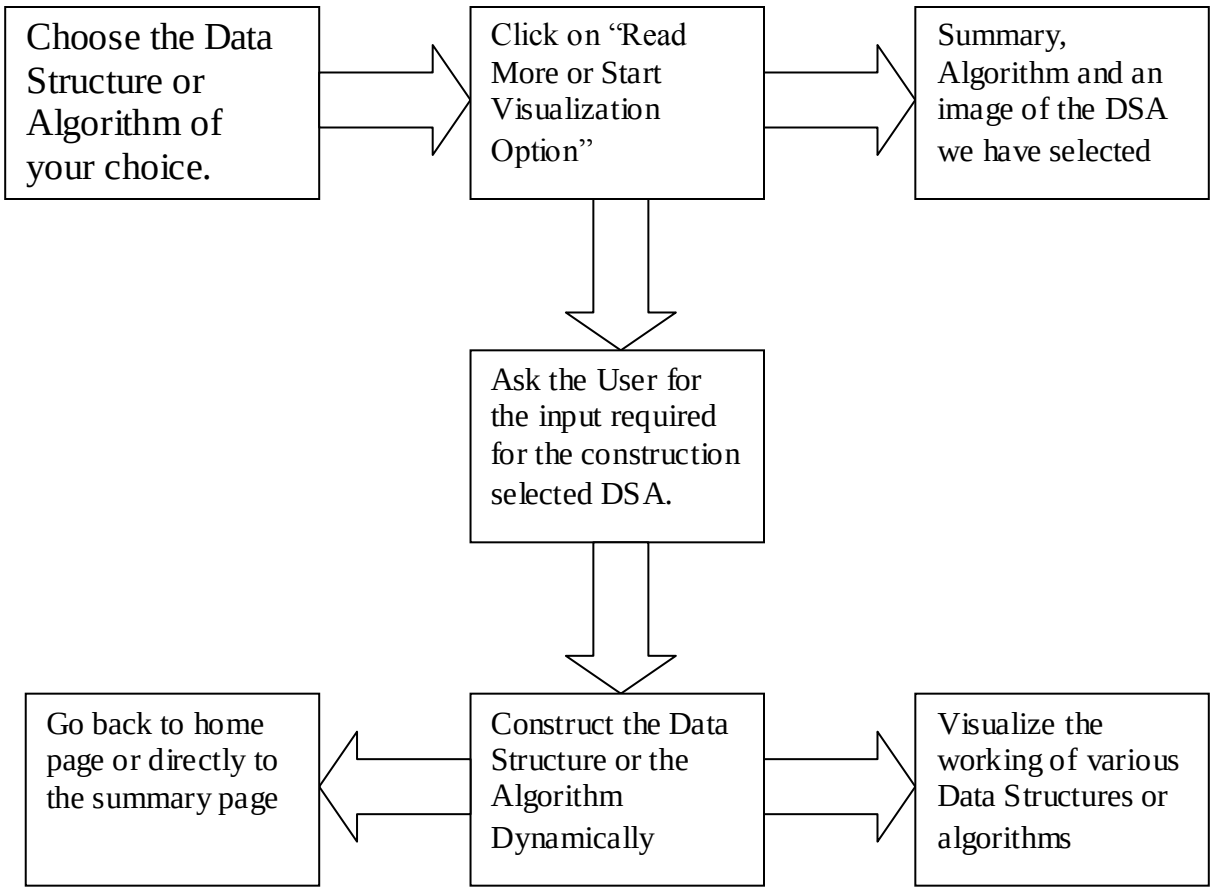


Fig 2.1 Overview of project

In general, within the website user has to choose from various Data structures and algorithms options to work out for visualization. The user has two choices; first one is to read a brief summary of the selected Data structure or Algorithm. In the second option, the user can redirect to the page which shows visualization of the same selected Data Structure or Algorithm. After clicking on the desired option the browser fetches summary or visualization from the database depending on what is clicked. On the summary page, there is a brief description, followed by the algorithm and an image or gif showing the Data Structure. On the Visualization page, we need to create the Data structure by providing the input so that the working can be visually be shown.

2.3. SYSTEM RESPONSE TO DIFFERENT INPUTS

2.3.1 CASE 1:

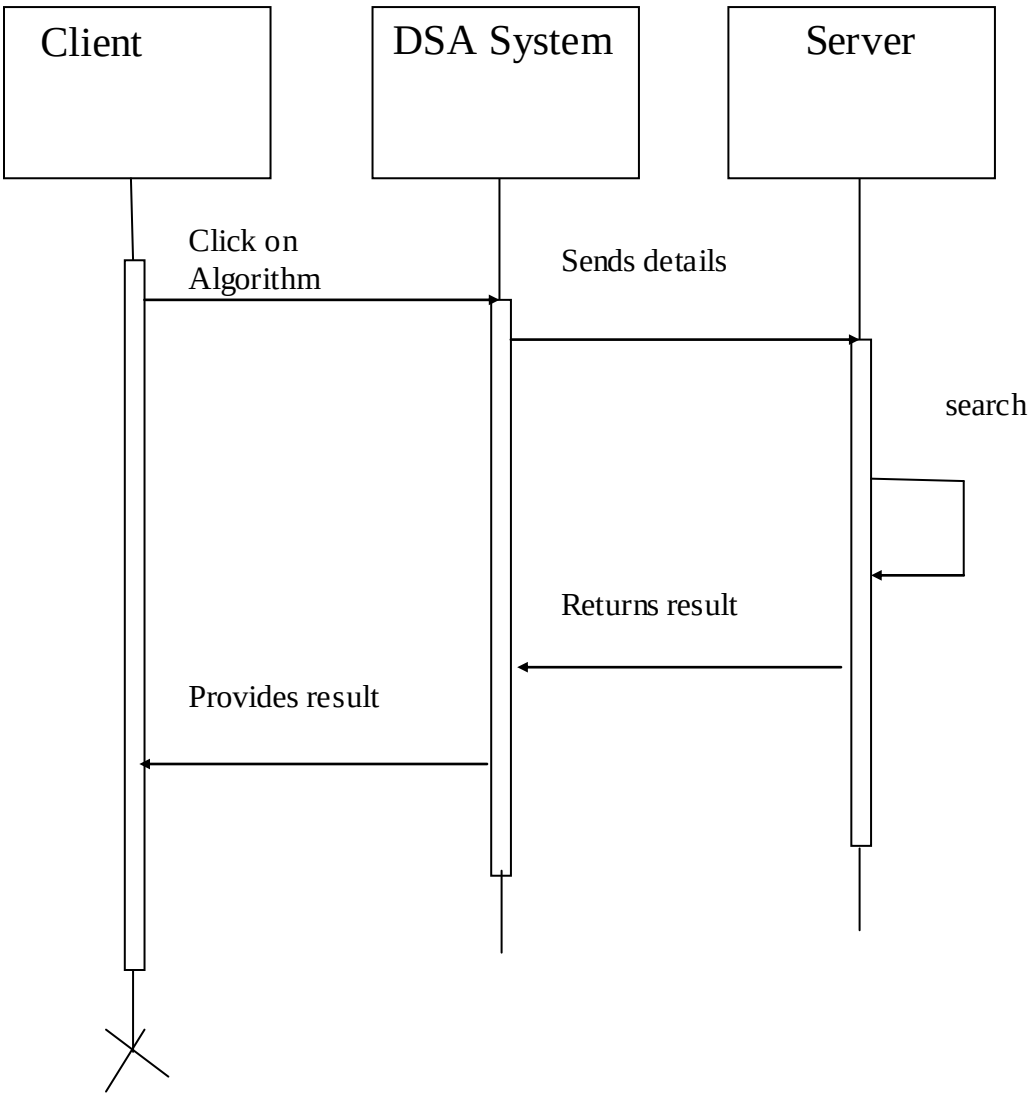


Fig 2.2 CASE 1: When the user selects the data structure or algorithm

In this case when user enters the website, the system then displays the various options he has at his disposal. These options are the various Data Structures and Algorithms. The application then asks the user about Data Structure or Algorithm the user wants to choose from the list of options. This will provide it with two results.

A. Read in detail about the Data Structure

B. Visualize the working of the Data Structure

2.3.2: CASE 2

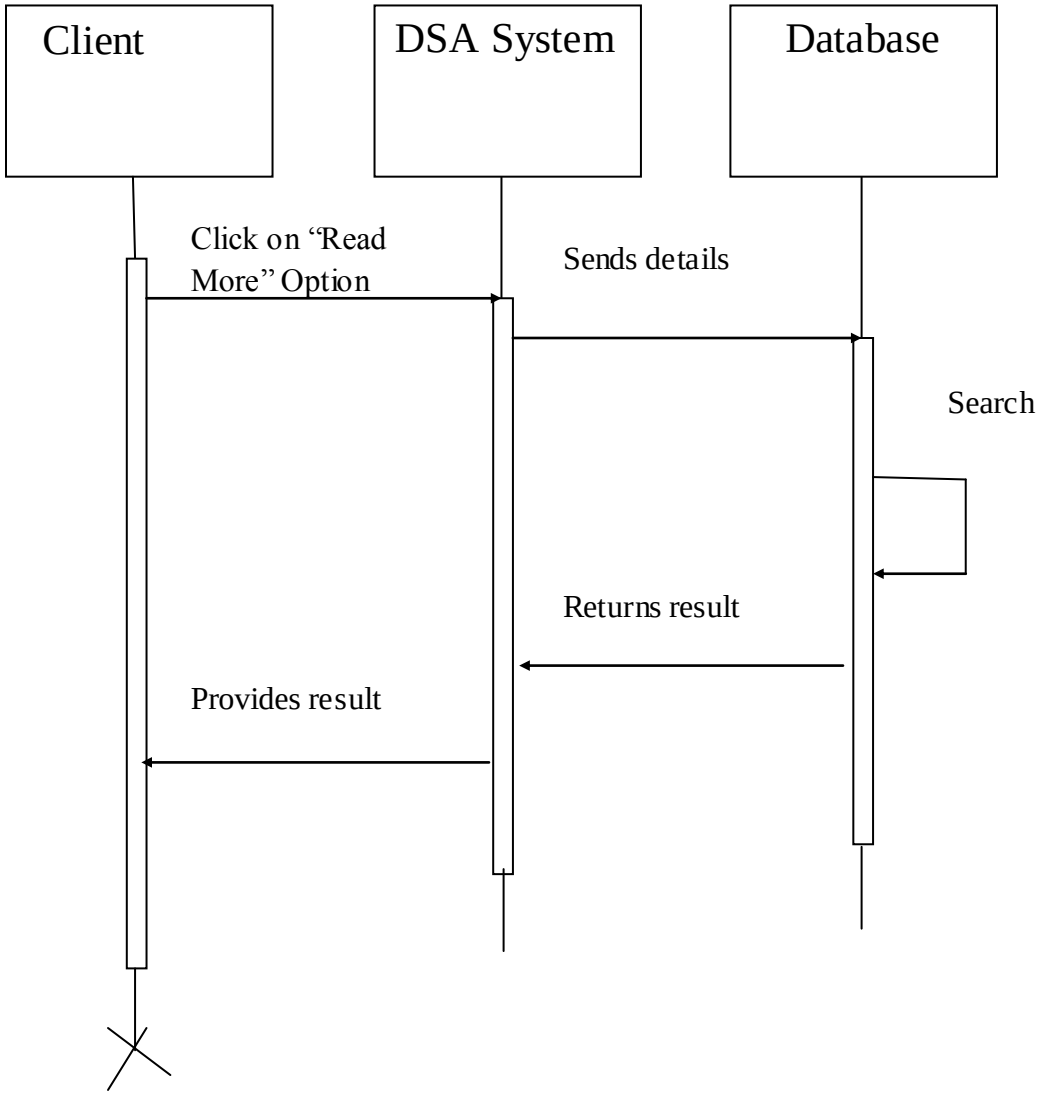


Fig 2.3 CASE 2: When the user selects the Read Option

In this case when user selects the read option, the website will provide it with a detailed summary of the Data Structure. The application will also provide us with the algorithm and also a small image of the algorithm. It will also give us the option of going back to the home page or going to the visualization page of the Data Structure.

2.3.3. CASE 3:

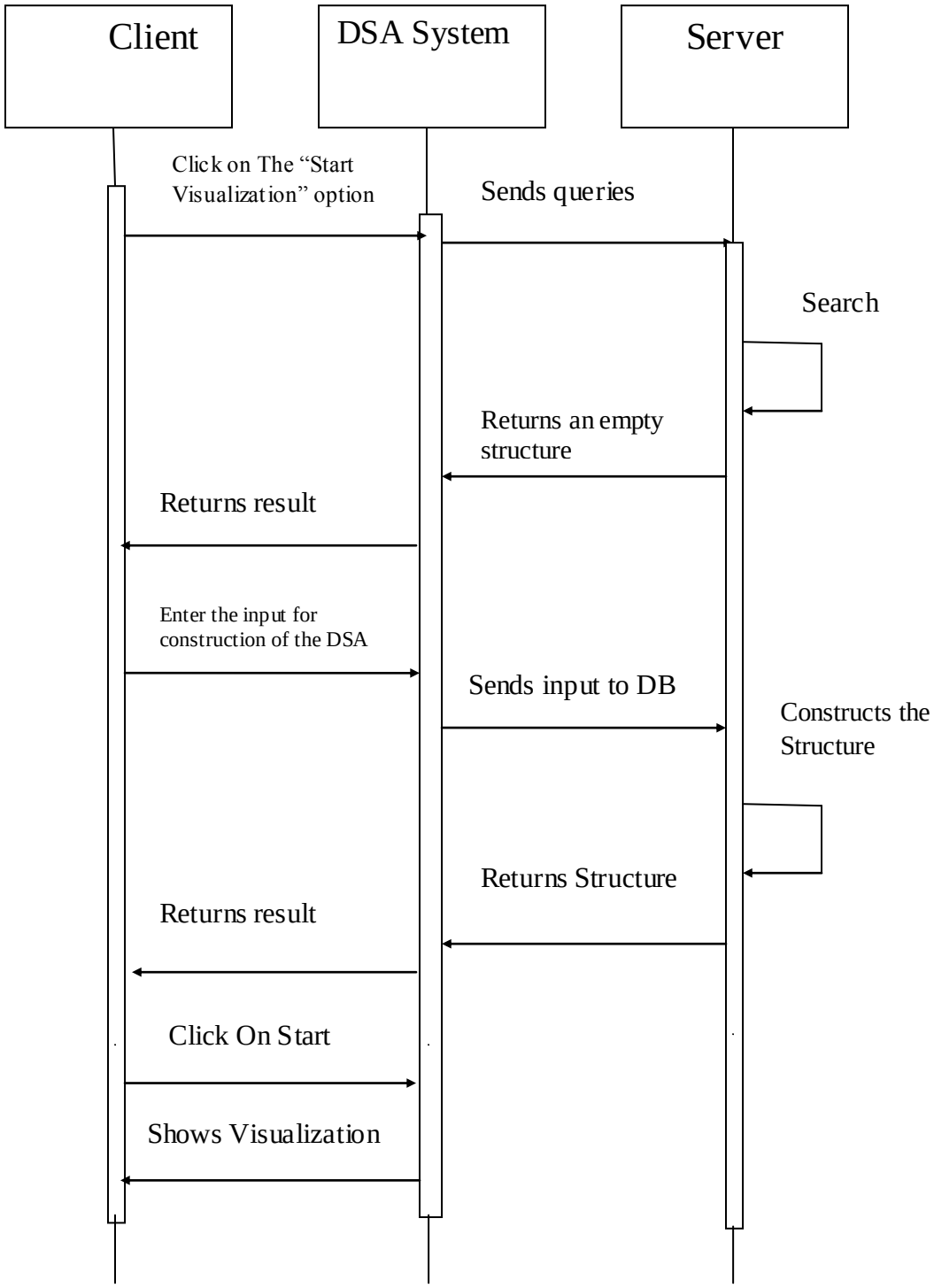


Fig 2.4 CASE 3: When User selects the Visualization Option

In this case the user enters the specific input the system asks for, for the construction of the Data Structure. After the user enters the input the system sends them to the database. In the database, the input is combined with the algorithm for construction of the DSA. The construction of the DSA is dynamic. The user can at any point ask for visualization by clicking on the "Start visualization" option.

2.4 Website Interface

For the website we have developed interactive animations for a variety of data structures and algorithms. Our visualization tool is written in java script using the HTML5 canvas element, and run in just about any modern browser. We have made the website more interactive by adding easy to understand tools and buttons for working. The website using javascript for the algorithms and a small amount of memory for storing the content needed for every database.

2.5 General Animation Controls

- *Skip Back* If you are in the middle of an animation, this button will completely undo the current command. If you are not in the middle of an animation, this button will undo the previous command
- *Step Back* This button is only active if you have paused the current animation (using the play/pause button). Step back one step in the current animation. If you are not currently animating, step back into the previous command. You could use this button (with sufficient key presses) to back up through the entire history of everything you've done
- *Play/Pause* Toggle between play mode (in which the algorithm runs free until it completes) and paused mode (where you need to press the Step Forward or Step Back button to advance the animation)
- *Step Forward* This button is only active if you have paused the current animation (using the play/pause button), and the current animation has not yet completed. Step forward one step in the current animation.
- *Skip Forward* This button is only active if the current animation has not completed. Skip to the end of the current animation

- *Animation Speed (slider)* Change the speed of the animation. The value of this slider is saved in a cookie, so you should only need to set it once if you have a preferred speed
- *w, h, Change Canvas Size* Change the width / height of the display area. While the change is immediate, the animations will not be centered in this new field until you reload the page. This will clear out the current animation, but the width and height values are also saved in a cookie, so you should only need to change this field once, and then everything should work well if you are on a smart phone or a desktop with loads of screen real estate.
- *Move Controls* Toggle between the general controls being at the top or bottom of the webpage

2.6 Methodologies and techniques implemented

I. Netbeans

Net Beans is a software development platform written in Java. The Net Beans Platform allows applications to be developed from a set of modular software components called modules. Applications based on the Net Beans Platform, including the Net Beans integrated development environment (IDE), can be extended by third party developers. The Net Beans IDE is primarily intended for development in Java, but also supports other languages, in particular PHP, C/C++and HTML5.

II. Notepad++

Notepad++ is a text editor and source code editor for use with Microsoft Windows. Unlike Notepad, the built-in Windows text editor, it supports tabbed editing, which allows working with multiple open files in a single window. The project's name comes from the C increment operator.

2.7 HARDWARE REQUIREMENTS

Processor :Intel Pentium 4 or higher

Hard disk :20 GB or higher

RAM :512 MB or higher

Input and Output :Standard

2.8 SOFTWARE REQUIREMENTS

Operating System :Microsoft Windows XP or higher, Linux

Software :JDK

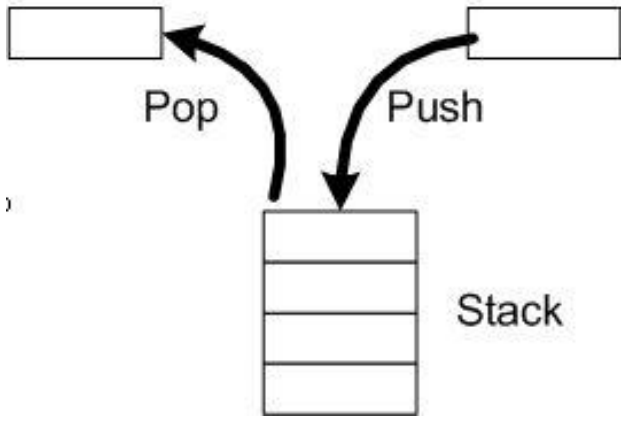
IDE :Netbeans 6.9.1

Chapter 3

Description

3.1 Stack

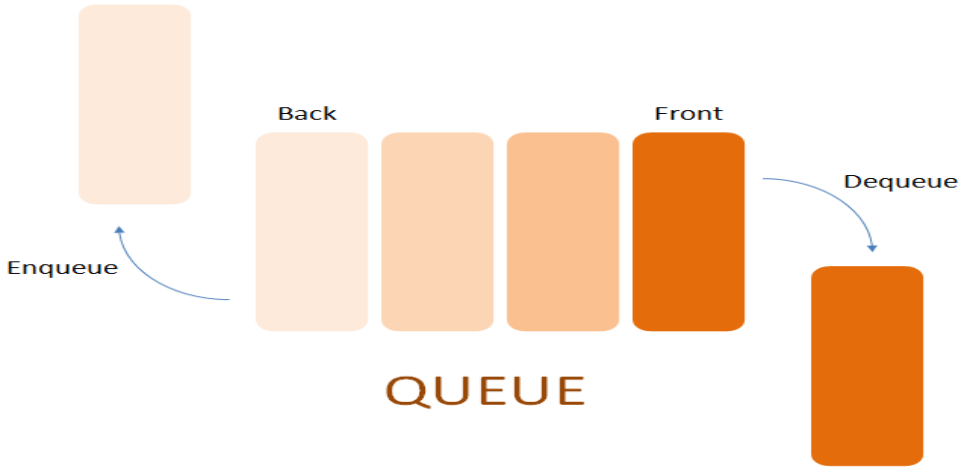
A stack is a container of objects that are inserted and removed according to the last-in first-out (LIFO) principle. In the pushdown stacks only two operations are allowed: **push** the item into the stack, and **pop** the item out of the stack. A stack is a limited access data structure - elements can be added and removed from the stack only at the top. **Push** adds an item to the top of the stack, **pop** removes the item from the top. A helpful analogy is to think of a stack of books; you can remove only the top book, also you can add a new book on the top.



3.2 Queue

A queue is a container of objects (a linear collection) that are inserted and removed according to the first-in first-out (FIFO) principle. An excellent example of a queue is a line of students in the food court of the UC. New additions to a line made to the back of the queue, while removal (or serving) happens in the front. In the queue only two operations are allowed **enqueue** and **dequeue**. Enqueue means to insert an item into the back of the queue, dequeue means removing the front item. The picture demonstrates the FIFO access.

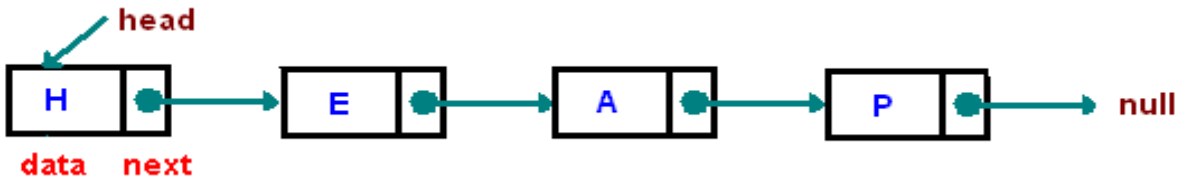
The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added.



3.3 Linked List

A linked list is a linear data structure where each element is a separate object. Each element (we will call it a **node**) of a list is comprising of two items - the data and a reference to the next node. The last node has a reference to null. The entry point into a linked list is called the **head** of the list. It should be noted that head is not a separate node, but the reference to the first node. If the list is empty then the head is a null reference.

A linked list is a dynamic data structure. The number of nodes in a list is not fixed and can grow and shrink on demand. Any application which has to deal with an unknown number of objects will need to use a linked list.

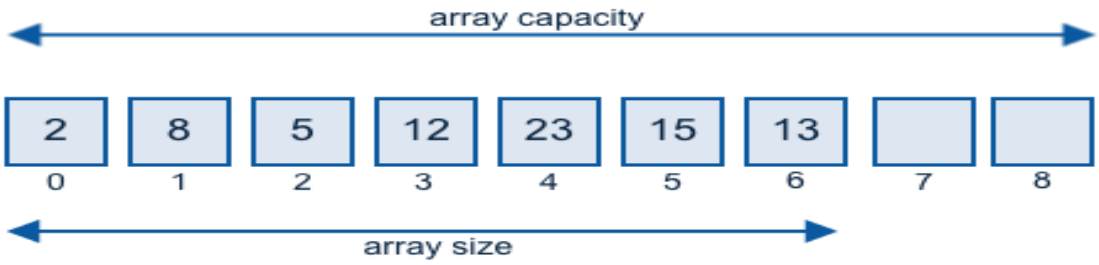


3.4 Array

An array is an aggregate data structure that is designed to store a group of objects of the same or different types. Arrays can hold primitives as well as references. The array is the most efficient data structure for storing and accessing a sequence of objects.

Here is the list of most important array features:

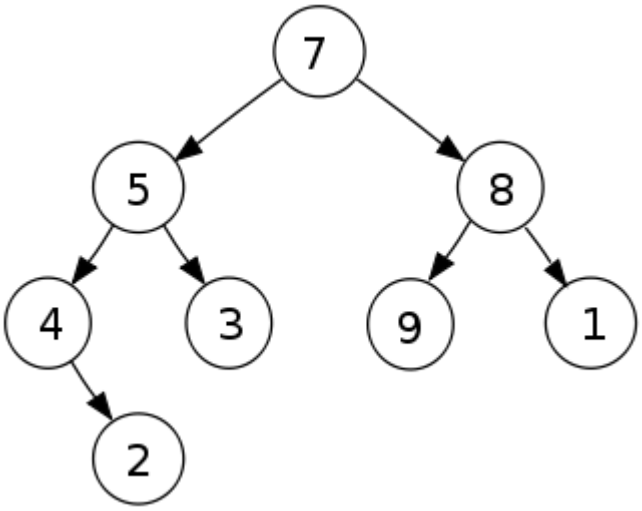
- copying and cloning
- insertion and deletion
- searching and sorting



3.5 Binary Search Tree

A binary tree is made of nodes, where each node contains a "left" pointer, a "right" pointer, and a data element. The "root" pointer points to the topmost node in the tree. The left and right pointers recursively point to smaller "subtrees" on either side. A null pointer represents a binary tree with no elements - the empty tree. The formal recursive definition is: a binary tree is either empty (represented by a null pointer), or is made of a single node, where the left and right pointers (recursive definition ahead) each point to a binary tree.

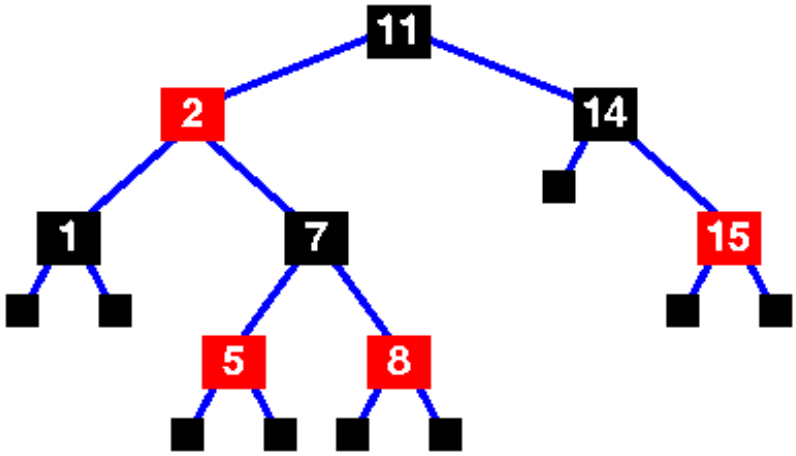
Binary search trees keep their keys in sorted order, so that lookup and other operations can use the principle of binary search: when looking for a key in a tree (or a place to insert a new key), they traverse the tree from root to leaf, making comparisons to keys stored in the nodes of the tree and deciding, based on the comparison, to continue searching in the left or right subtrees.



3.6 Red Black Tree

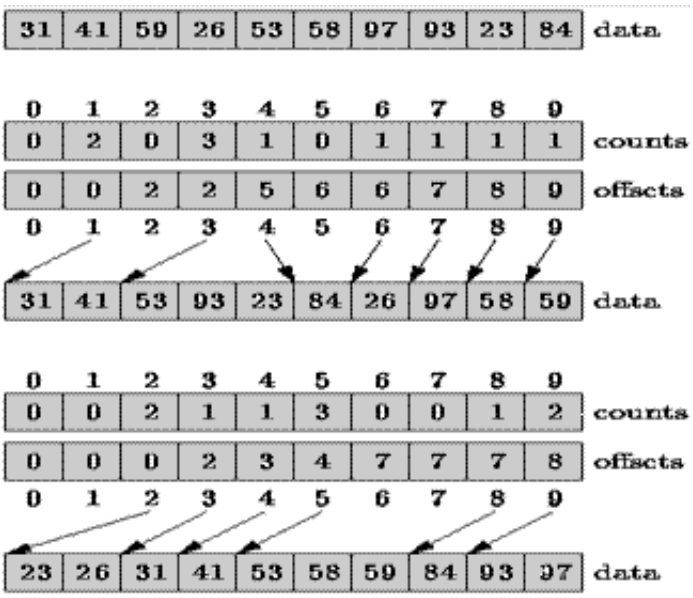
A red–black tree is a binary search tree with an extra bit of data per node, its color, which can be either red or black. The extra bit of storage ensures an approximately balanced tree by constraining how nodes are colored from any path from the root to the leaf. Thus, it is a data structure which is a type of self-balancing binary search tree.

Balance is preserved by painting each node of the tree with one of two colors (typically called 'red' and 'black') in a way that satisfies certain properties, which collectively constrain how unbalanced the tree can become in the worst case. When the tree is modified, the new tree is subsequently rearranged and repainted to restore the coloring properties. The properties are designed in such a way that this rearranging and recoloring can be performed efficiently.



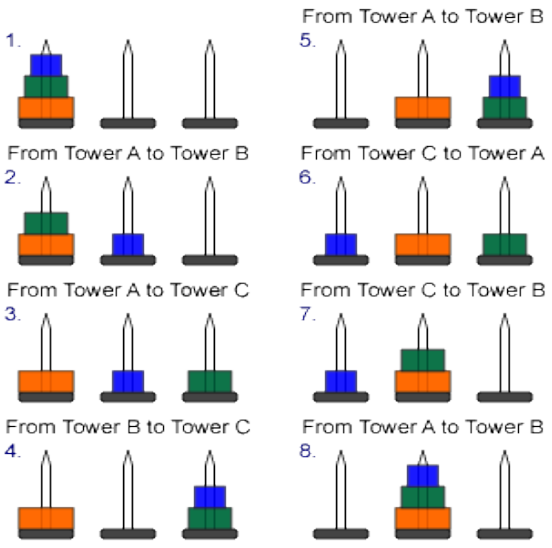
3.7 Radix Sort

Radix sort is a non-comparative integer sorting algorithm that sorts data with integer keys by grouping keys by the individual digits which share the same significant position and value. A positional notation is required, but because integers can represent strings of characters (e.g., names or dates) and specially formatted floating point numbers, radix sort is not limited to integers.



3.8 Tower of Hanoi

There are 3 rods where discs can be stacked. All the discs are having different diameters. The goal is to move all discs from one tower to another using the third tower. The rule is that only smaller disc can go on top of the bigger discs. So the discs will be sorted according to their sizes and bigger one will be at the bottom. Only one disc can be moved at a time. The removed disc must be placed in some tower and it cannot be left in any other places. At the beginning, there are some discs in tower one. The goal is to move them to tower two following the above rules. With three disks, it can be solved in seven moves. The minimum number of moves required to solve a Tower of Hanoi is 2^n - 1, where n is the number of disks.



Chapter 4

Results And Discussion

4.1 Module 1:

In general when the user opens the web browser and enters the url for the website, the request from the browser is sent to the DNS to search for the ip address of the server. After successfully locating the ip on the server, the browser establishes a TCP connection. Browser then sends a HTTP request

GET http://dsa.tsec.com/ HTTP/1.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-Agent: Mozilla/5.0 (X11; Windows x86_64; rv:29.0) Gecko/20100101 Firefox/29.0
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
Host: google.com

After getting response, the browser displays the HTML content.

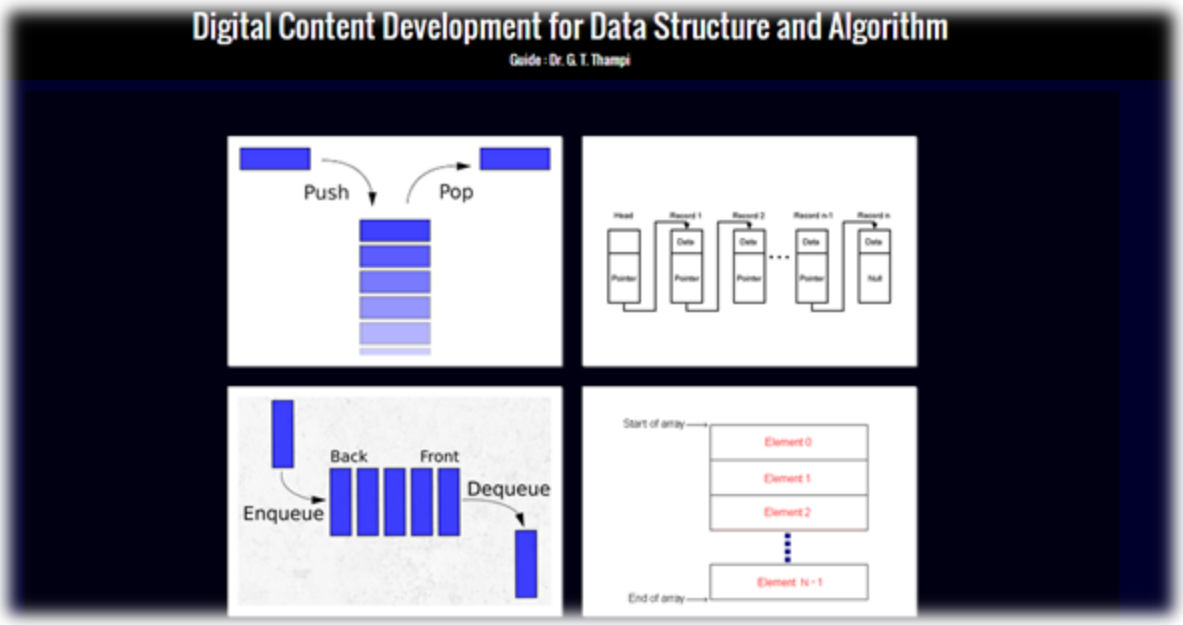


Fig 4.1 The Home UI of the application

4.2 Module 2:

In this case when user fetches the HTML content, the system initially displays the list of all the Data Structures and Algorithms which are available on the website. Then the user chooses the Data Structure or Algorithm he wants to view. On hovering above the DSA there is a small animation. Then the user gets two options to choose from

a. Run Code
b. Read More

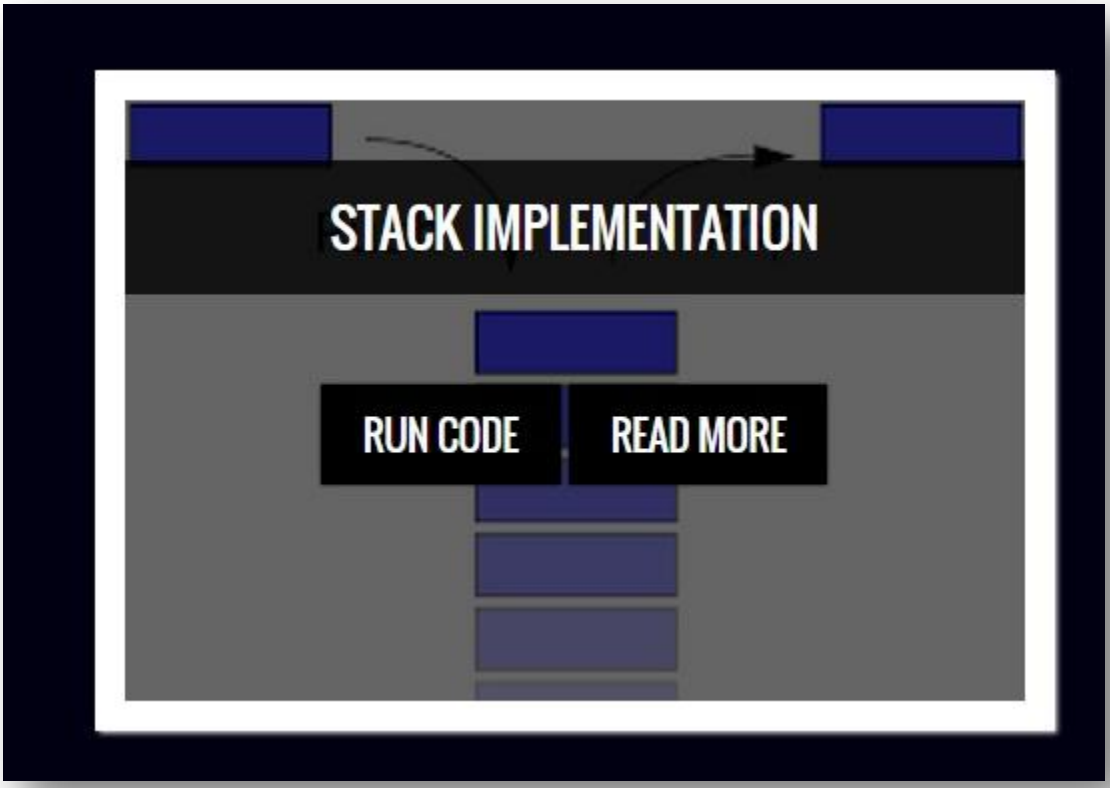
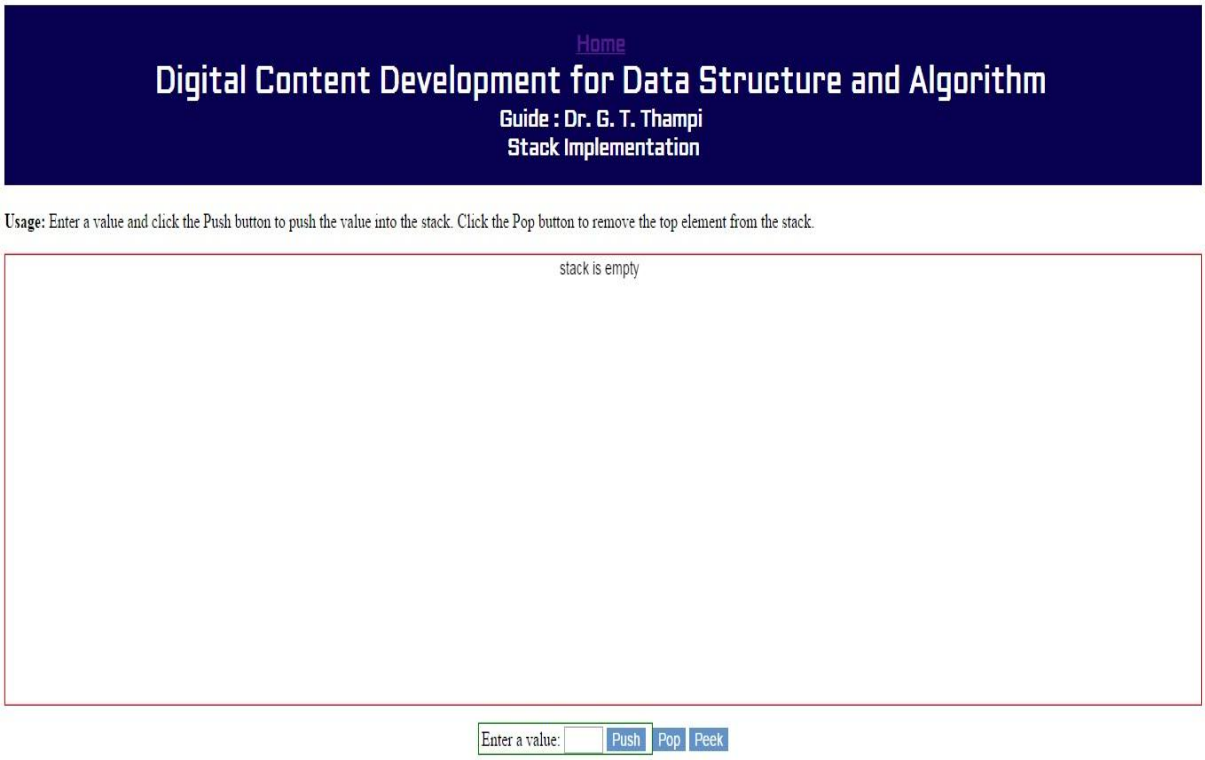


Fig 4.2 After choosing a desired DSA

4.3 Selection of "Run Code" Option:

In this case when user chooses the "Run Code" option, the browser redirects the website to the specifically chosen Data Structure or Algorithm. In this case we are showing the implementation of a Stack. Here we can see an empty stack. Now we need to provide input to the stack for the construction.



4.3.1 UI for the Stack implementation

```
<body onload="draw()" onresize="draw()">
  <h3 style="text-align:center; font: bold"> <a href="../index.html" >Home</a>
  <br />
  <span style="font-size:30px">Digital Content Development for Data Structure and Algorithm</span>
<br />
    Guide : Dr. G. T. Thampi
    <br /><b>Stack Implementation</b>
    </h3>

  <p>

    <b>Usage:</b> Enter a value and click the Push button to push the value into the stack.
    Click the Pop button to remove the top element from the stack.
  </p>

  <div style="margin: 0px auto; border: 1px solid red; text-align: right">
    <canvas id="canvas" width="200" height="300"></canvas>
  </div>
  <div style="text-align: center; margin-top: 1em">
    <span style="border: 1px solid green; padding: 3px">
      Enter a value: <input type="text" value="" id="value" style="width: 3em; text-align: right">
      <button type="button" class="button" onclick="push()">Push</button>
    </span>
    <button type="button" class="button" onclick="pop()">Pop</button>
    <button type="button" class="button" onclick="peek()">Peek</button>
  </div>
</body>
```

4.3.2 HTML code for “Stack” UI

```
<script src="Stack.js"></script>
<script >
  stack = new Stack();

  function draw() {
      var canvas = document.getElementById('canvas');
      var context = canvas.getContext("2d");

      // Reset size will clear the canvas, but not for IE9
      canvas.width = window.innerWidth - 20;
      canvas.height = window.innerHeight - 300;
      context.clearRect(0, 0, canvas.width, canvas.height-160); // For IE 9

      context.font = "14px sans-serif";
      context.strokeStyle = "#100"; // Set a pen color

      if (stack.isEmpty()) {
          context.fillText("stack is empty", canvas.width / 2 - 50, 15);
      }
      else {
          x = canvas.width / 2 - 30;
          y = canvas.height - 30;
          rectWidth = 40;
          rectHeight = 20;

          list = stack.list;
          for (var i = 0; i < list.length; i++) {
              value = list[i];
              context.fillText(list[i] + "", x + 15, y + 15);
              context.rect(x, y, rectWidth, rectHeight);
              y -= rectHeight;
          }
          context.fillText("Top", x - 85, y + rectHeight + 10);
          drawArrowLine(context, x - 45, y + rectHeight + 10,
                  x, y + rectHeight + 10);
      }

      context.stroke();
  }
```

4.3.3. javascript code for "Stack" creation

4.3.1 Providing the Input for construction of stack

In this case when user enters the input, i.e. pushes an element into the stack, a dynamic stack is created. The HTML script calls the javascript which contains predefined way in which Stack implementation works.

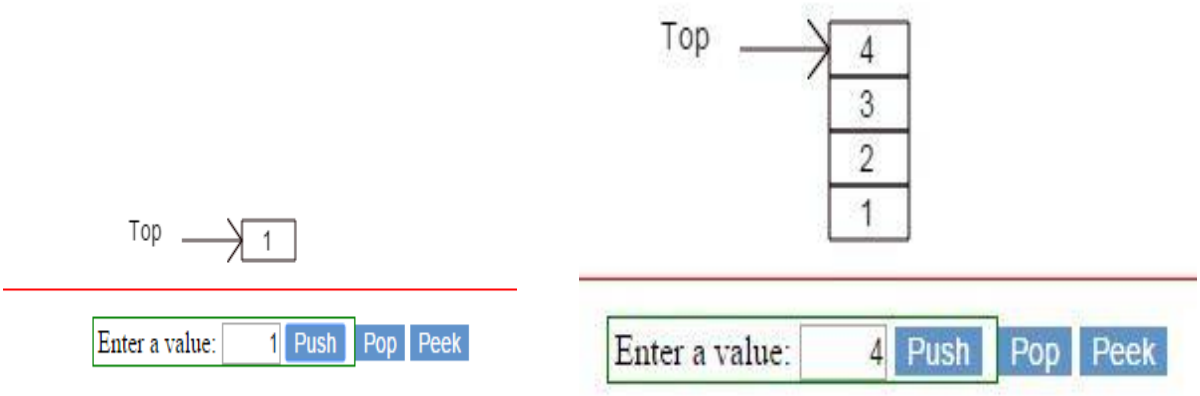


Fig 4.3.4 Dynamic Construction of Stack

4.3.2 Performing the various functions:

After the construction of the stack, the user gets an option for pushing further, popping or peeking. According to the user choice the application performs the function.

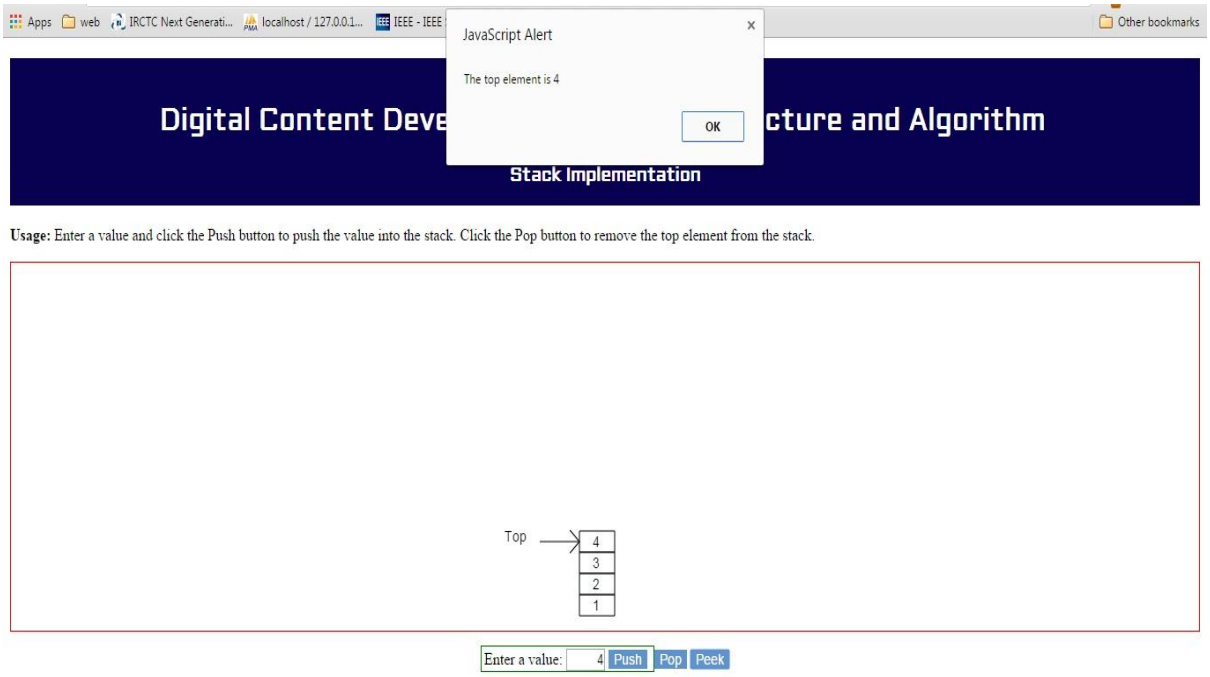


Fig 4.3.5 Working Peek function

```
function peek() {
    if (stack.isEmpty()) {
        alert("The stack is empty");
    }
    else {
        alert("The top element is " + stack.peek());
    }
}

function pop() {
    if (stack.isEmpty()) {
        alert("The stack is empty");
    }
    else {
        stack.pop();
        draw();
    }
}

function push() {
    var value = document.getElementById('value').value.trim();
    if (value == "") {
        alert("no value entered");
    }
    else {
        stack.push(value);
    }

    draw();
}
```

4.3.6. javascript code for “Push”, “Pop” and “Peek” functions

```
1   // Constructor for Stack
2   function Stack() {
3     this.list = [];
4
5     // Push an element to the stack
6     this.push = function(e) {
7       this.list.push(e);
8     }
9
10    // Pop the top  from the stack
11    this.pop = function() {
12      return this.list.pop();
13    }
14
15    // Peek the top from the stack
16    this.peek = function() {
17      return this.list[this.getSize() - 1];
18    }
19
20    // Return the stack size
21    this.getSize = function() {
22      return this.list.length;
23    }
24
25    // Return true if stack is empty
26    this.isEmpty = function() {
27      return this.list.length == 0;
28    }
29  }
```

4.3.7. Functions in javascript calling the Push, pop and seek

4.4 Selection of the “Read More” option

On the selection of the “Read more” option on the main page, the website redirects the user to a new page which consists of detailed description about the Algorithm. Along with it the user can see some images and animations of the data structure or the algorithm

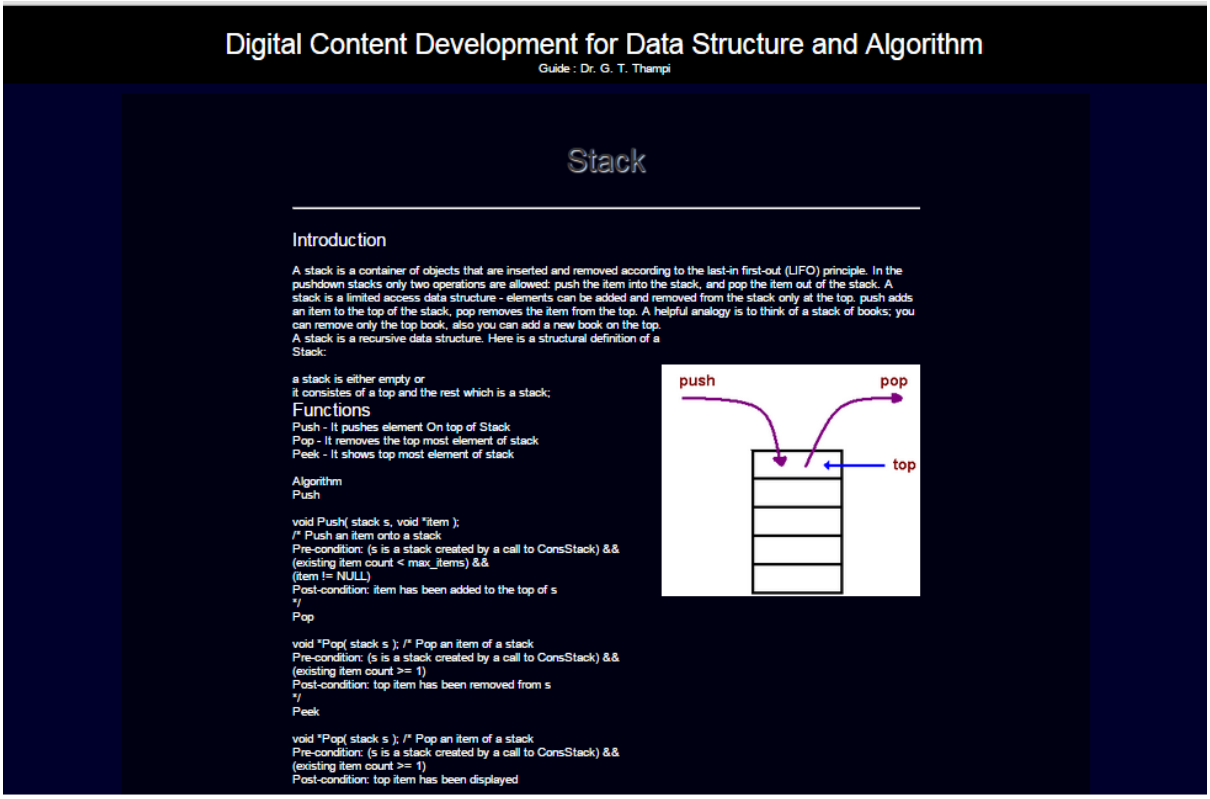


Fig 4.4 “Read More” page of Stack

4.5 Other Data Structures and Algorithms

In this example we have only shown the working of Stacks. Thus we have only displayed and the functions of Stack (Push, Pop, Peek). But in general there are other Data Structures and they are having their own different functions and different inputs. The interface and pattern of working of all the other Data Structures and Algorithms is similar to that of Stacks.

The functions of other data structures and algorithms are as follows:

1. Stack :
   Inputs: a. Enter a value
   Functions: a. Push b. Pop c. Peek functions
2. Queue :
   Inputs: a. Enter a value
   Functions: a. Enqueue b. Dequeue c. Start Over
3. Linked List:
   Inputs: a. Enter a value b. Enter an Index
   Functions: a. Search b. Insert c. Delete
4. Array:
   Inputs: a. Enter a value b. Enter an Index
   Functions: a. Search b. Insert c. Delete d. Trim to size
5. Binary Search Tree:
   Inputs: a. Enter a key
   Functions: a. Search b. Insert c. Remove d. Inorder e. Preorder f. Postorder
6. Red Black Tree:
   Inputs: a. Enter a key
   Functions: a. Search b. Insert c. Remove
7. Radix Sort:
   Inputs: -
   Functions: a. Step b. Reset
8. Tower Of Hanoi:
   Inputs: a. Number of Disks
   Functions: a. Start b. Reset

5. TESTING

The main philosophy behind testing is to discover errors and improve existing system. We used following software engineering methods for testing our application.

5.1 CODE TESTING

Code testing means checking correctness of code fragment by examining its logic using test cases.

Following are some of the test cases that we have used:

Case 1: 34,56,21,67

Case 2: ; , ^ , 7 , $

Case 3: a, b, c, d

These three test cases were given as input to stack, queue and BST. Case 1 did not cause any trouble when given to stack, queue and BST. When case 2 was given as input to stack, it caused the application to die. This was rectified by changing data type to string. Case 3 also caused the same problem and was corrected by the above step. In the case of BST, decision to do lexicographic sorting solved problem of comparison to some extent.

5.2 UNIT TESTING

In Unit testing, we concentrated on individual modules. Each of the modules like panels, stack, queue, BST and sorting algorithms were individually tested and errors were corrected.

5.3 INTEGRATION TESTING

In integration testing we tested our application by combining the different modules. Initially compilation error regarding missing packages were shown which was corrected by giving correct class path names.

5.4 SYSTEM TESTING

In this phase we tested our application in different platforms, for example Windows and Linux. Our application was found to work correctly in both platforms.

6. CONCLUSION AND FURTHER WORK

Conclusion:

In this project, we have presented a visualization tool designed to aid computer science, Information Technology and other students from similar fields to learn Data Structures and Algorithms. This tool lets students visualize the commonly used data structures. We believe this tool will be an effective supplement to traditional instruction. Because of the time imitation, only the most commonly used data structures are implemented in this version of the software package, which include arrays, stacks, queues, binary search tree and sorting algorithms like insertion sort, selection sort and bubble sort.

Further Work:

This application can be expanded by including more data structures and algorithms. This can also be included with visualization of user defined algorithms. There are two ways to add more observable data structures to this software such as directed graph, weighted graph, AVL tree, Red Black Tree, AAtree, splay tree, hash table, etc. One way is to implement these data structures in the software. Another approach would be to develop and implement a mechanism for the software package to recognize the user-defined observable data structures, and leave the implementation to the user. This approach will allow users to use their own observable data structures, hence add more flexibility to the software. Another possible future enhancement for the software is to highlight the executing command line of the user-defined algorithm file. This would help the user to better follow the execution of the algorithm. This software package can be included with time and space complexities as well as algorithms.

Chapter 7

REFERENCES

1. https://www.cs.auckland.ac.nz/~jmor159/PLDS210/ds_ToC.html

2. http://people.cs.vt.edu/shaffer/Papers/Cooper07.pdf

3. T. Chen and T. Sobh, A tool for data structure visualization and user-defined algorithm animation. Frontiers in Education Conference, 2001.

4. http://arxiv.org/pdf/1403.4423.pdf

5. T. H. Cormen, C.E. Leiserson, and R. L. Rivest. Introduction to Algorithms, The MIT Press/McGraw-Hill, 2001.

6. http://www.dsalgo.com/2013/02/towers-of-hanoi.html

7. http://www.tutorialspoint.com/java/java_data_structures.htm

8. C. Kehoe, J. Stasko, and A. Taylor. Rethinking the evaluation of algorithm animations as learning aids: an observational study. In International Journal of Human-Computer Studies, 54(2), pp. 265-284, 2001.

9. http://en.wikipedia.org/wiki/List_of_data_structures

10. M. A. Weiss. Data Structures and Problem Solving Using Java. Addison-Wesley, 3rd Edition, 2005. 191

11. R. Fleischer and L. Kucera. Algorithm Animation for Teaching. Software Visualization, International Seminar Dagstuhl Castle 2001, Revised Lectures, Springer LNCS 2269, pp. 113-128, 2002.

ACKNOWLEDGEMENT

We feel great pleasure in presenting the report for our project "DIGITAL CONTENT DEVELOPMENT FOR DATA STRUCTURES AND ALGORITHMS". We wholeheartedly thank DR.G.T.THAMPI, Principal, Thadomal Shahani Engineering College, first for letting us choose the project of our interest and secondly for co-operating with us in every possible manner. He has also provided with encouragement and she has been the motivational force for us. We would also like to thank him for providing us with the infrastructure and the facilities in this college.

We would also like to thank Mr.ARUN KULKARNI, Head, Information Technology Department, Thadomal Shahani Engineering College, for providing us with a wide range of domains to choose our project from.

We would also like to thank all the Faculty members as well as the non teaching staff of Thadomal Shahani Engineering College for giving us their valuable time and being co-operative. Last but not the least we would like to thank our families and friends who supported us in this due course.

Nikhil Chandak 92

Darshil Shah 93

Akshay Desai 97

Rohan Fagwani 95