Job No: 01

Job Name: **Write a program to implement a simple chatbot.**

Theory:

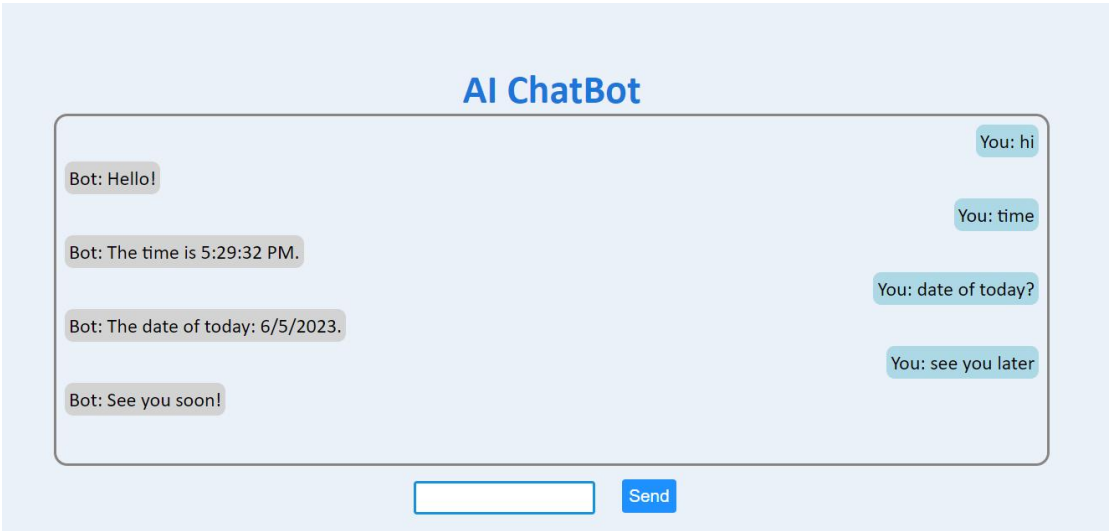
The purpose of a chatbot is to simulate a conversation with a user and provide responses based on predefined rules or patterns. It can be used to answer questions, provide information, or assist with tasks. A chatbot takes user input, typically in the form of text, and generates an appropriate response.

Code:

```
<!DOCTYPE html>
<html>
  <head>
    <title>ChatBot</title>
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
    <h1 style="margin: auto; color: #1e76da;">AI ChatBot</h1>
    <div id="chat-container"></div>
    <input type="text" id="user-input" />
    <button id="submit-btn">Send</button>

    <script>
      const userInput = document.getElementById("user-input");
      const submitBtn = document.getElementById("submit-btn");
      const chatContainer = document.getElementById("chat-container");
      let dateTime = new Date();

      const responses = {
          "hello": "Hello!",
          "hi": "Hello!",
          "how are you": "I'm doing well, What about you?",
          "good": "Great!",
          "see you later": "See you soon!",
          "what's your name?": "I am Jhon(AI Chat Bot)",
          "who are you?": "I am Jhon(AI Chat Bot)",
          "what's the time now?": `The time is
                                ${dateTime.toLocaleTimeString('en-US')}.`,
```

```
        "what's the date of today?": `The date of today:
                                    ${dateTime.toLocaleDateString()}.`
    }

    function getResponse(input) {
      for (let key in responses) {
        if ((input.toLowerCase().includes(key)) ||
            (key.includes(input.toLowerCase()))) {
          return responses[key];
        }
      }
      return responses["default"];
    }

    function Message(message,className,person) {
      const messageElement = document.createElement("div");
      messageElement.classList.add(className);
      messageElement.innerText = `${person}: ${message}`;
      chatContainer.appendChild(messageElement);
    }

    submitBtn.addEventListener("click", () => {
      const userInputValue = userInput.value.trim();
      Message(userInputValue,"user-message","You");
      const chatbotResponse = getResponse(userInputValue);
      Message(chatbotResponse, "chatbot-message","Bot");
      userInput.value = "";
    });
  </script>
 </body>
</html>
```

Input/Output:



Job No: 02

Job Name: **Write a program for Breadth First Search algorithm.**

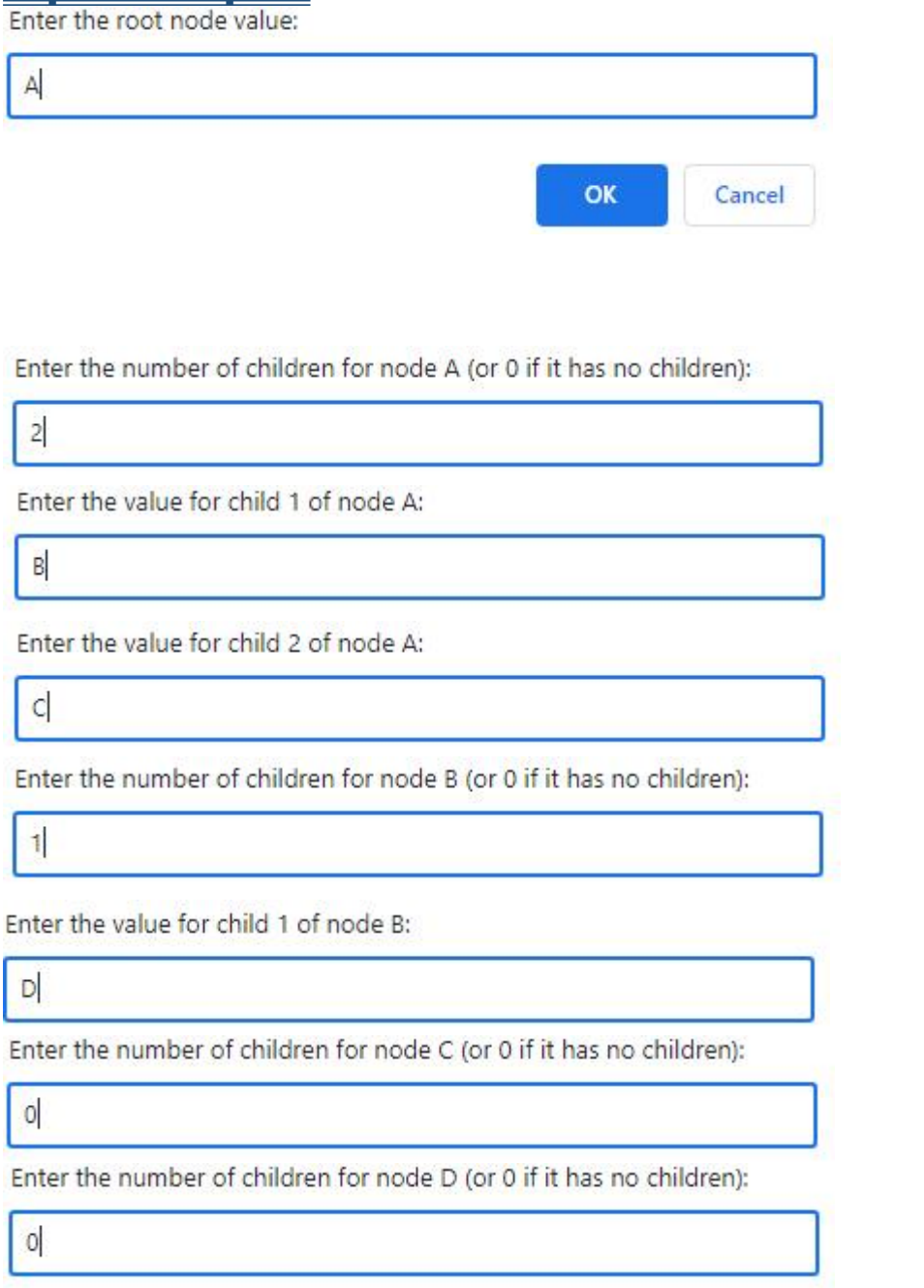
Theory: Breadth First Search (BFS) is a tree traversal algorithm used to search a tree or graph in a breadthward motion. It starts at the root node and visits all the nodes of the tree in breadth-first order, i.e., visiting all the immediate children of a node before moving to the next level children.

Code:

```
class Node {
  constructor(value) {
    this.value = value;
    this.children = [];
  }
}

function breadFirstTraversal(root) {
  const queue = [root];
  const result = [];

  while (queue.length) {
    const node = queue.shift();
    result.push(node.value);
    queue.push(...node.children);
  }
  return result;
}

function buildTree() {
  const root = new Node(prompt("Enter the root node value:"));
  const queue = [root];
  while (queue.length > 0) {
    const node = queue.shift();
    const numChildren = +prompt(`Enter the number of children for node
                        ${node.value} (or 0 if it has no children):`)
    for (let i = 1; i <= numChildren; i++) {
      const childNode = new Node(prompt(`Enter the value for child ${i} of
                                        node ${node.value}:`));
      node.children.push(childNode);
      queue.push(childNode);
    }
```

```
  }
  return root;
}

console.log(`The result of BFS Search for given tree is :
            ${breadFirstTraversal(buildTree())}`);
```

Input/Output:




```
The result of BFS Search for given tree is : A,B,C,D
```

Job No: 03

Job Name: **Write a program for Depth First Search algorithm.**

Theory: Depth-first search is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node and explores as far as possible along each branch before backtracking.

Code:

```
class Node {
  constructor(value) {
    this.value = value;
    this.children = [];
  }
}

function dfs(node,values=[]) {
  node.children.forEach((child) => {
    values = values.concat(dfs(child));
  });

  return values.concat(node.value);
}

function buildTree() {
  const root = new Node(prompt("Enter the root node value:"));
  const queue = [root];

  while (queue.length > 0) {
    const node = queue.shift();
    const numChildren = +prompt(`Enter the number of children for node
                                ${node.value} or (0: No children):`);
    for (let i = 1; i <= numChildren; i++) {
      const childNode = new Node(prompt(`Enter the value for child ${i} of
                                        node ${node.value}:`));
      node.children.push(childNode);
      queue.push(childNode);
    }
  }
  return root;
}
```

```
const root = buildTree();
console.log(`The result of DFS Search for the given tree is: ${dfs(root)}`);
```

Input/Output:

Enter the root node value:

A

Enter the number of children for node A or (0: No children):

2

Enter the value for child 1 of node A:

B

Enter the value for child 2 of node A:

C

Enter the number of children for node B or (0: No children):

2

Enter the value for child 1 of node B:

D

Enter the value for child 2 of node B:

E

Enter the number of children for node C or (0: No children):

0

Enter the number of children for node D or (0: No children):

0

Enter the number of children for node E or (0: No children):

0

```
The result of DFS Search for the given tree is: D,E,B,C,A
```