DAA -Assig - Yejsjdkd hskslao hsksksk

Design Analysis and Algorithm (Chandigarh University)

DEPARTMENT OF
COMPUTER SCIENCE &
ENGINEERING

Discover. Learn. Empower.

Assignment

Student: Name: Abhishek Pathak **UID:** 22BCS16643

Branch: BE-CSE **Section/Group:** 638-B

Semester: 5th **Date of Performance:** 22/10/24

Subject Name: Design and Analysis of Algorithms **Subject Code:** 22CSH-311

Problem 1

1. **Aim:**Given the pointer to the head node of a linked list, change the next pointers of the nodes so that their order is reversed. The head pointer given may be null meaning that the initial list is empty.

2. **Objective:** The goal is to reverse the `next` pointers of each node to point to the previous node.

3. **Algorithm**

reverse function:

- We initialize `prev` to `NULL`, `curr` to the head, and then iteratively reverse the `next` pointers while traversing the list.
- After the loop, `prev` will be the new head of the reversed linked list.

printLinkedList function:

- This prints the linked list in the format `1->2->3->NULL`.

insertNodeAtEnd function:

- It inserts a node at the end of the list for each element in the input.

4. **Implementation/Code:**

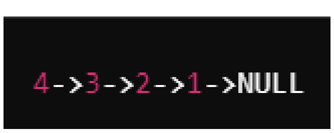
```
#include <iostream>
using namespace std;
struct SinglyLinkedListNode {
    int data;
    SinglyLinkedListNode* next;
```

```
    SinglyLinkedListNode(int node_data) {
        data = node_data;
        next = nullptr;
    }
};
SinglyLinkedListNode* reverse(SinglyLinkedListNode* head) {
    SinglyLinkedListNode* prev = nullptr;
    SinglyLinkedListNode* curr = head;
    SinglyLinkedListNode* next_node = nullptr;

    while (curr != nullptr) {
        next_node = curr->next; // Store next node
        curr->next = prev;      // Reverse current node's pointer
        prev = curr;            // Move prev to current
        curr = next_node;       // Move current to next
    }
    return prev; // prev will be the new head of the reversed list
}
void printLinkedList(SinglyLinkedListNode* head) {
    SinglyLinkedListNode* node = head;
    while (node != nullptr) {
        cout << node->data;
        node = node->next;
        if (node != nullptr) cout << "->";
    }
    cout << "->NULL" << endl;
}
SinglyLinkedListNode* insertNodeAtEnd(SinglyLinkedListNode* head, int data) {
    SinglyLinkedListNode* new_node = new SinglyLinkedListNode(data);
    if (head == nullptr) {
        return new_node;
```

```
    }
    SinglyLinkedListNode* temp = head;
    while (temp->next != nullptr) {
        temp = temp->next;
    }
    temp->next = new_node;
    return head;
}
int main() {
    int t;
    cin >> t; // Number of test cases
    while (t--) {
        int n;
        cin >> n; // Number of elements in the linked list
        SinglyLinkedListNode* head = nullptr;
        for (int i = 0; i < n; ++i) {
            int data;
            cin >> data;
            head = insertNodeAtEnd(head, data);
        }
        head = reverse(head);
        printLinkedList(head);
    }
    return 0;
}
```

5. Output:



Problem 2

1. **Aim:** Write the postOrder function. It received 1 parameter: a pointer to the root of a binary tree. It must print the values in the tree's postorder traversal as a single line of space-separated values.

2. **Objective:** Perform a postorder traversal of a binary tree

3. **Algorithm**

Node structure: Each node of the binary tree stores an integer value (`data`) and has two pointers, `left` and `right`, which point to its left and right children, respectively.

postOrder function:

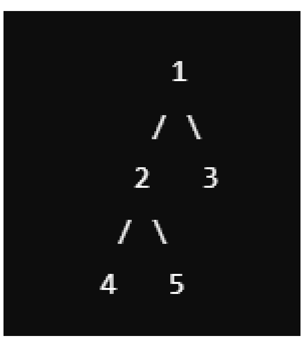
- It first checks if the current node is `NULL` (base case).
- Then it recursively calls `postOrder` on the left child and right child.
- Finally, it prints the data of the current node after traversing both subtrees.

4. **Implementation/Code:**

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* left;
    Node* right;
    Node(int val) {
        data = val;
        left = nullptr;
        right = nullptr;
    }
};
void postOrder(Node* root) {
    if (root == nullptr) {
        return; // Base case: if the node is null, do nothing
```

```
    }
    postOrder(root->left);
    postOrder(root->right);
    cout << root->data << " ";
}
int main() {
    Node* root = new Node(1);
    root->left = new Node(2);
    root->right = new Node(3);
    root->left->left = new Node(4);
    root->left->right = new Node(5);
    postOrder(root);
    return 0;
}
```

5. Output:



Problem 3

1. **Aim:** You are given a pointer to the root of a binary search tree and values to be inserted into the tree. Insert the values into their appropriate position in the binary search tree and return the root of the updated binary tree.

2. **Objective:** To insert a node in a Binary Search Tree (BST)

3. **Algorithm**

Node Structure: Each node contains an integer `data`, and two pointers `left` and `right` that point to its left and right children, respectively.

insert Function:

- If the tree is empty (i.e., `root == nullptr`), create a new node with the given data and return it.
- If the tree is not empty:
  - Recursively insert the data into the left subtree if `data < root->data`.
  - Recursively insert the data into the right subtree if `data > root->data`.
- Return the root after insertion to maintain the link to the new node.

inorder Function: This function prints the nodes in an inorder traversal, which for a BST will produce a sorted sequence of values. It is used to check if the tree is built correctly.

4. **Implementation/Code:**

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* left;
    Node* right;
    Node(int val) {
        data = val;
        left = nullptr;
        right = nullptr;
    }
```

```
};
Node* insert(Node* root, int data) {
    // Base case: If the tree is empty, return a new node
    if (root == nullptr) {
        return new Node(data);
    }
    if (data < root->data) {
        root->left = insert(root->left, data);
    } else if (data > root->data) {
        root->right = insert(root->right, data);
    }
    return root;
}
void inorder(Node* root) {
    if (root != nullptr) {
        inorder(root->left); // Visit left subtree
        cout << root->data << " "; // Print the root node
        inorder(root->right); // Visit right subtree
    }
}
int main() {
    Node* root = nullptr; // Start with an empty tree

    int n, data;
    cout << "Enter number of nodes to insert: ";
    cin >> n; // Number of nodes to insert

    for (int i = 0; i < n; i++) {
        cout << "Enter value " << i + 1 << ": ";
        cin >> data;
        root = insert(root, data); // Insert each value into the BST
```

```
    }
    cout << "Inorder traversal of the BST: ";
    inorder(root);
    cout << endl;
    return 0;
}
```

5. Output:

```
Enter number of nodes to insert: 5
Enter value 1: 4
Enter value 2: 2
Enter value 3: 7
Enter value 4: 1
Enter value 5: 3
```

```
Inorder traversal of the BST: 1 2 3 4 7
```

Problem 4

1. **Aim:**We're covering a *divide-and-conquer* algorithm called Quicksort (also known as *Partition Sort*). This challenge is a modified version of the algorithm that only addresses partitioning. It is implemented as follows:

2. **Objective:** Perform a postorder traversal of a binary tree

3. **Algorithm**

Initialize three empty arrays: `left`, `equal`, and `right`.

Traverse the array starting from the second element (`arr[1]` to `arr[n-1]`):

- If `arr[i] < pivot`, append it to `left`.
- If `arr[i] > pivot`, append it to `right`.

Return the concatenation of `left`, `equal`, and `right`.

4. **Implementation/Code:**

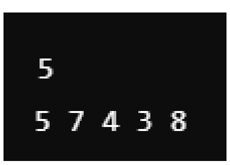
```
#include <iostream>
#include <vector>
using namespace std;

vector<int> quickSort(vector<int>& arr) {
    int pivot = arr[0]; // The pivot element is always the first element
    vector<int> left, equal, right;
    for (int i = 0; i < arr.size(); i++) {
        if (arr[i] < pivot) {
            left.push_back(arr[i]);  // Elements less than the pivot
        } else if (arr[i] == pivot) {
            equal.push_back(arr[i]); // Elements equal to the pivot
        } else {
            right.push_back(arr[i]); // Elements greater than the pivot
        }
```

```
    }
    vector<int> result;
    result.insert(result.end(), left.begin(), left.end());
    result.insert(result.end(), equal.begin(), equal.end());
    result.insert(result.end(), right.begin(), right.end());
    return result;
}
int main() {
    int n;
    cin >> n; // Size of the array
    vector<int> arr(n);
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }
    vector<int> sorted_arr = quickSort(arr);
    for (int i = 0; i < sorted_arr.size(); i++) {
        cout << sorted_arr[i] << " ";
    }
    cout << endl;

    return 0;
}
```

5. Output:



Problem 5

1. **Aim:**Samantha and Sam are playing a numbers game. Given a number as a string, no leading zeros, determine the sum of all integer values of substrings of the string.

Given an integer as a string, sum all of its substrings cast as integers. As the number may become large, return the value modulo $10^9 + 7$.

2. **Objective:** find the sum of all substrings that can be formed from the given string `n`

3. **Algorithm**

Input:

- We first read the input `n` which is a string representing the integer.

substrings Function:

- We initialize two variables:
  - `total_sum`: To store the total sum of all substrings (modulo 109+710^9 + 7109+7).
  - `current_sum`: To store the sum of all substrings ending at the current position `i`.
- For each digit in the string:
  - We compute its contribution to all substrings that include it. This is done using the formula `current_sum = (current_sum * 10 + (i + 1) * num) % MOD`.
  - We add `current_sum` to `total_sum`.
- The final result is `total_sum % MOD`.

Output:

4. **Implementation/Code:**

```
#include <iostream>

#include <string>

using namespace std;


const int MOD = 1000000007;

int substrings(string n) {

    long long total_sum = 0;

    long long current_sum = 0;
```

```
    for (int i = 0; i < n.size(); i++) {
        int num = n[i] - '0';
        current_sum = (current_sum * 10 + (i + 1) * num) % MOD;
        total_sum = (total_sum + current_sum) % MOD;
    }
    return total_sum;
}
int main() {
    string n;
    cin >> n;
    cout << substrings(n) << endl;
    return 0;
}
```

5. Output: