TECHNICAL TRAINING DSA- CODING PRACTICE PROBLEMS

Name: Sachin A

Dept: CSBS

Date: 19-11-2024

Minimum Path SubProgram

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

class MinimumPathSum {
public:
    static int minPathSum(vector<vector<int>>& grid) {
        int m = grid.size(), n = grid[0].size();
        for (int j = 1; j < n; j++)
            grid[0][j] += grid[0][j - 1];

        for (int i = 1; i < m; i++)
            grid[i][0] += grid[i - 1][0];

        for (int i = 1; i < m; i++) {
            for (int j = 1; j < n; j++) {
                grid[i][j] += min(grid[i - 1][j], grid[i][j - 1]);
            }
        }
        return grid[m - 1][n - 1];
    }
```

```
};

int main() {
    int m, n;
    cout << "Enter the number of rows (m): ";
    cin >> m;
    cout << "Enter the number of columns (n): ";
    cin >> n;

    vector<vector<int>> grid(m, vector<int>(n));
    cout << "Enter the grid values row by row:" << endl;

    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            cin >> grid[i][j];
        }
    }

    int result = MinimumPathSum::minPathSum(grid);
    cout << "The minimum path sum is: " << result << endl;

    return 0;
}
```

OUTPUT:

```
Enter the number of rows (m):
3
Enter the number of columns (n):
3
Enter the grid values row by row:
1 3 1
1 5 1
4 2 1
The minimum path sum is: 7
```

Validate Binary search tree:

```
#include <iostream>

#include <stack>

#include <queue>

#include <string>

#include <vector>


using namespace std;


struct TreeNode {

    int val;

    TreeNode* left;

    TreeNode* right;

    TreeNode() : val(0), left(nullptr), right(nullptr) {}

    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}

    TreeNode(int x, TreeNode* left, TreeNode* right) : val(x), left(left), right(right) {}

};


class Solution {

public:

    bool isValidBST(TreeNode* root) {

        if (!root) return true;
```

```
        stack<TreeNode*> st;
        TreeNode* previous = nullptr;
        while (root || !st.empty()) {
            while (root) {
                st.push(root);
                root = root->left;
            }
            root = st.top();
            st.pop();
            if (previous && root->val <= previous->val) return false;
            previous = root;
            root = root->right;
        }
        return true;
    }
};

TreeNode* buildTree(const vector<string>& values) {
    if (values.empty() || values[0] == "null") return nullptr;
    TreeNode* root = new TreeNode(stoi(values[0]));
    queue<TreeNode*> q;
    q.push(root);
    int i = 1;
    while (i < values.size()) {
        TreeNode* current = q.front();
        q.pop();
        if (i < values.size() && values[i] != "null") {
            current->left = new TreeNode(stoi(values[i]));
            q.push(current->left);
```

```
        }
        i++;
        if (i < values.size() && values[i] != "null") {
            current->right = new TreeNode(stoi(values[i]));
            q.push(current->right);
        }
        i++;
    }
    return root;
}

int main() {
    vector<string> values = {"2", "1", "3"};
    TreeNode* root = buildTree(values);
    Solution solution;
    bool isValid = solution.isValidBST(root);
    cout << "Is the binary tree a valid BST? " << (isValid ? "true" : "false") << endl;
    return 0;
}
```

OUTPUT

```
Is the binary tree a valid BST? true
```

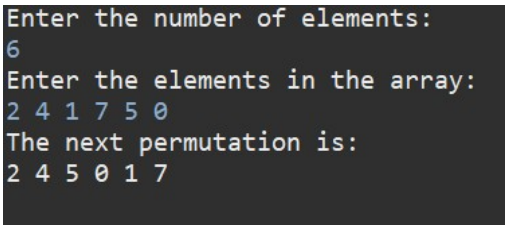
Next permutation

```
#include <iostream>
#include <vector>
#include <algorithm>
```

```
using namespace std;

void swap(vector<int>& arr, int i, int j) {
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
}

void permutations(vector<vector<int>>& res, vector<int>& arr, int idx) {
    if (idx == arr.size() - 1) {
        res.push_back(arr);
        return;
    }
    for (int i = idx; i < arr.size(); i++) {
        swap(arr, idx, i);
        permutations(res, arr, idx + 1);
        swap(arr, idx, i);
    }
}

void nextPermutation(vector<int>& arr) {
    vector<vector<int>> res;
    permutations(res, arr, 0);
    sort(res.begin(), res.end());
    for (int i = 0; i < res.size(); i++) {
        if (res[i] == arr) {
            if (i < res.size() - 1) {
                arr = res[i + 1];
            } else {
```

```
            arr = res[0];
        }
        break;
      }
    }
}

int main() {
    int n;
    cout << "Enter the number of elements: ";
    cin >> n;

    vector<int> arr(n);
    cout << "Enter the elements in the array: ";
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    nextPermutation(arr);

    cout << "Next permutation: ";
    for (int x : arr) {
        cout << x << " ";
    }

    return 0;
}
```

OUTPUT:



SPIRAL MATRIX

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

class SpiralMatrix {
public:
    vector<int> spiralOrder(vector<vector<int>>& matrix) {
        if (matrix.empty()) return {};

        int m = matrix.size(), n = matrix[0].size();
        vector<int> ans;
        int r1 = 0, c1 = 0, r2 = m - 1, c2 = n - 1;

        while (ans.size() < m * n) {
            for (int j = c1; j <= c2 && ans.size() < m * n; ++j) ans.push_back(matrix[r1][j]);
            for (int i = r1 + 1; i <= r2 - 1 && ans.size() < m * n; ++i) ans.push_back(matrix[i][c2]);
            for (int j = c2; j >= c1 && ans.size() < m * n; --j) ans.push_back(matrix[r2][j]);
            for (int i = r2 - 1; i >= r1 + 1 && ans.size() < m * n; --i) ans.push_back(matrix[i][c1]);
            ++r1; ++c1; --r2; --c2;
        }
```

```
        return ans;
    }
};

int main() {
    int m, n;
    cout << "Enter the number of rows: ";
    cin >> m;
    cout << "Enter the number of columns: ";
    cin >> n;

    vector<vector<int>> matrix(m, vector<int>(n));
    cout << "Enter the elements of the matrix row by row:\n";
    for (int i = 0; i < m; ++i)
        for (int j = 0; j < n; ++j)
            cin >> matrix[i][j];

    SpiralMatrix solution;
    vector<int> result = solution.spiralOrder(matrix);

    cout << "Spiral Order:\n";
    for (int num : result)
        cout << num << " ";
    return 0;
}
```

OUTPUT:

```
Enter the number of rows:
3
Enter the number of columns:
3
Enter the elements of the matrix row by row:
1 2 3
4 5 6
7 8 9
Spiral Order:
1 2 3 6 9 8 7 4 5
```

Longest substring without repeating characters

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

int longestUniqueSubstr(string s) {
    int n = s.length();
    int res = 0;

    for (int i = 0; i < n; i++) {
        vector<bool> visited(256, false);

        for (int j = i; j < n; j++) {
            if (visited[s[j]]) {
                break;
            } else {
                res = max(res, j - i + 1);
                visited[s[j]] = true;
```

```
        }
      }
    }
    return res;
}

int main() {
    string s;
    cout << "Enter the String: ";
    getline(cin, s);

    cout << "The length of the longest substring is " << longestUniqueSubstr(s) << endl;
    return 0;
}
```

OUTPUT:

```
Enter the String:
aabccc
The length of the longest substring is 3
```

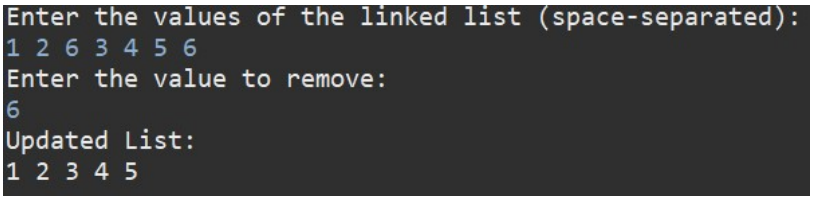
Remove linked list element

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

struct ListNode {
    int val;
    ListNode* next;
```

```
    ListNode() : val(0), next(nullptr) {}
    ListNode(int x) : val(x), next(nullptr) {}
    ListNode(int x, ListNode* next) : val(x), next(next) {}
};

class RemoveLinkedListElement {
public:
    ListNode* removeElements(ListNode* head, int val) {
        ListNode* res = new ListNode(0, head);
        ListNode* temp = res;
        while (temp != nullptr) {
            while (temp->next != nullptr && temp->next->val == val) {
                temp->next = temp->next->next;
            }
            temp = temp->next;
        }
        return res->next;
    }

    void printList(ListNode* head) {
        while (head != nullptr) {
            cout << head->val;
            head = head->next;
            if (head != nullptr) cout << " ";
        }
    }

    ListNode* createList(vector<int>& values) {
        if (values.empty()) return nullptr;
```

```
        ListNode* head = new ListNode(values[0]);
        ListNode* current = head;
        for (int i = 1; i < values.size(); ++i) {
            current->next = new ListNode(values[i]);
            current = current->next;
        }
        return head;
    }
};

int main() {
    string input;
    getline(cin, input);
    stringstream ss(input);
    vector<int> values;
    int val, temp;

    while (ss >> temp) values.push_back(temp);
    cin >> val;

    RemoveLinkedListElement solution;
    ListNode* head = solution.createList(values);
    ListNode* updatedList = solution.removeElements(head, val);
    solution.printList(updatedList);

    return 0;
}
```

OUTPUT:



Palindrome linked list

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

struct Node {
    int data;
    Node* next;
    Node(int d) : data(d), next(nullptr) {}
};

class PalindromeLinkedList {
public:
    static bool isPalindrome(Node* head) {
        Node* currNode = head;
        stack<int> s;

        while (currNode != nullptr) {
            s.push(currNode->data);
            currNode = currNode->next;
        }

        while (head != nullptr) {
```

```
            int c = s.top();
            s.pop();

            if (head->data != c) {
                return false;
            }

            head = head->next;
        }

        return true;
    }
};

int main() {
    Node* head = new Node(1);
    head->next = new Node(2);
    head->next->next = new Node(3);
    head->next->next->next = new Node(2);
    head->next->next->next->next = new Node(1);

    bool result = PalindromeLinkedList::isPalindrome(head);

    if (result)
        cout << "It is a palindrome" << endl;
    else
        cout << "It is not a palindrome" << endl;

    return 0;
```

}

OUTPUT:

```
It is a palindrome
```