TECHNICAL TRAINING DSA- CODING PRACTICE PROBLEMS

Name: Sachin A

Dept: CSBS

Date: 11-11-2024

0-1 knapsack problem

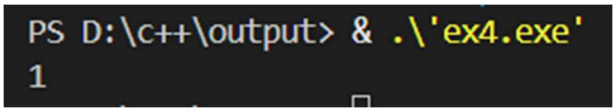
```
#include <bits/stdc++.h>
using namespace std;

int maximizeValue(int capacity, int weights[], int values[], int itemCount) {
    if (itemCount == 0 || capacity == 0)
        return 0;

    if (weights[itemCount - 1] > capacity)
        return maximizeValue(capacity, weights, values, itemCount - 1);

    return max(maximizeValue(capacity, weights, values, itemCount - 1),
            values[itemCount - 1] + maximizeValue(capacity - weights[itemCount - 1], weights, values, itemCount - 1));
}

int main() {
    int values[] = { 60, 100, 120 };
    int weights[] = { 10, 20, 30 };
    int capacity = 50;
    int itemCount = sizeof(values) / sizeof(values[0]);
    cout << maximizeValue(capacity, weights, values, itemCount);
    return 0;
}
```

OUTPUT:

Floor in sorted array

```
#include <bits/stdc++.h>
using namespace std;
int searchInsert(vector<int>& nums, int target) {
    int low = 0;
    int high = nums.size() - 1;
    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (nums[mid] == target) {
            return mid;
        } else if (nums[mid] < target) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }
    return low;
}

int main(){
    vector<int>arr={1,3,5,6};
    int target=2;
    int a=searchInsert(arr,target);
    cout<<a;
}
```

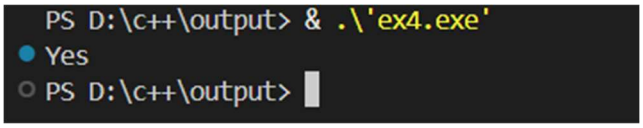
Output:



Check equal arrays

```
#include <bits/stdc++.h>
using namespace std;

bool arraysAreEqual(vector<int>& array1, vector<int>& array2) {
    int size1 = array1.size(), size2 = array2.size();
    if (size1 != size2)
        return false;

    sort(array1.begin(), array1.end());
    sort(array2.begin(), array2.end());

    for (int i = 0; i < size1; i++)
        if (array1[i] != array2[i])
            return false;

    return true;
}

int main() {
    vector<int> array1 = {3, 5, 2, 5, 2};
    vector<int> array2 = {2, 3, 5, 5, 2};

    if (arraysAreEqual(array1, array2))
        cout << "Yes";
    else
        cout << "No";
    return 0;
}
```

OUTPUT:



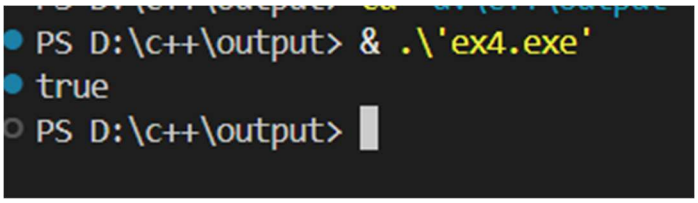
Palindrome linked list

```
#include <iostream>

using namespace std;


class ListNode {

public:

   int value;

   ListNode* next;

   ListNode(int val) {

      value = val;

      next = nullptr;

   }

};


ListNode* reverseList(ListNode* head) {

   ListNode* prevNode = nullptr;

   ListNode* currentNode = head;

   ListNode* nextNode;


   while (currentNode) {

      nextNode = currentNode->next;

      currentNode->next = prevNode;

      prevNode = currentNode;

      currentNode = nextNode;

   }

   return prevNode;

}
```

```
bool areListsEqual(ListNode* list1, ListNode* list2) {
    while (list1 && list2) {
        if (list1->value != list2->value)
            return false;
        list1 = list1->next;
        list2 = list2->next;
    }
    return true;
}

bool checkPalindrome(ListNode* head) {
    if (!head || !head->next)
        return true;

    ListNode* slowPointer = head;
    ListNode* fastPointer = head;

    while (fastPointer->next && fastPointer->next->next) {
        slowPointer = slowPointer->next;
        fastPointer = fastPointer->next->next;
    }

    ListNode* secondHalf = reverseList(slowPointer->next);
    slowPointer->next = nullptr;

    bool isPalindrome = areListsEqual(head, secondHalf);

    secondHalf = reverseList(secondHalf);
    slowPointer->next = secondHalf;

    return isPalindrome;
```

```
}

int main() {
    ListNode head(1);
    head.next = new ListNode(2);
    head.next->next = new ListNode(3);
    head.next->next->next = new ListNode(2);
    head.next->next->next->next = new ListNode(1);

    bool result = checkPalindrome(&head);

    if (result)
        cout << "true\n";
    else
        cout << "false\n";

    return 0;
}
```

OUTPUT:



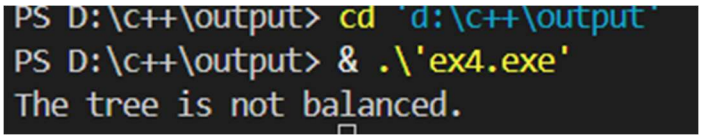
Balanced tree check

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

```
struct Node {
    int data;
    Node* left;
    Node* right;
    Node(int val) : data(val), left(nullptr), right(nullptr) {}
};

class Solution {
public:
    bool isBalanced(Node* root) {
        return dfsHeight(root) != -1;
    }
    int dfsHeight(Node* root) {
        if (root == NULL) return 0;
        int leftHeight = dfsHeight(root->left);
        if (leftHeight == -1)
            return -1;
        int rightHeight = dfsHeight(root->right);
        if (rightHeight == -1)
            return -1;
        if (abs(leftHeight - rightHeight) > 1)
            return -1;
        return max(leftHeight, rightHeight) + 1;
    }
};

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);
    root->left->right->right = new Node(6);
    root->left->right->right->right = new Node(7);

    Solution solution;

    if (solution.isBalanced(root)) {
        cout << "The tree is balanced." << endl;
    } else {
        cout << "The tree is not balanced." << endl;
    }

    return 0;
}
```

OUTPUT:



3sum

```
#include <bits/stdc++.h>
using namespace std;

vector<vector<int>> threeSum(vector<int>& nums) {
    vector<vector<int>> result;
    int n = nums.size();
    if (n < 3) return result;

    sort(nums.begin(), nums.end());

    for (int i = 0; i < n - 2; ++i) {
```

```
        if (i > 0 && nums[i] == nums[i - 1]) continue;

        int j = i + 1;
        int k = n - 1;

        while (j < k) {
            int sum = nums[i] + nums[j] + nums[k];
            if (sum == 0) {
                result.push_back({nums[i], nums[j], nums[k]});
                while (j < k && nums[j] == nums[j + 1]) ++j;
                while (j < k && nums[k] == nums[k - 1]) --k;
                ++j;
                --k;
            } else if (sum < 0) {
                ++j;
            } else {
                --k;
            }
        }
    }

    return result;
}

int main() {
    vector<int> arr = {-1, 0, 1, 2, -1, -4};
    vector<vector<int>> result = threeSum(arr);

    for (const auto& triplet : result) {
        for (int num : triplet) {
            cout << num << " ";
        }
```

```
        cout << endl;
    }


    return 0;
}
```

Output: