TECHNICAL TRAINING DSA - CODING PRACTICEPROBLEMS

Name: Rishi Kumar S
Dept: CSBS
Date: 10-11-2024

Question 1: 0-1 knapsack problem

Answer:

```
#include <bits/stdc++.h>
using namespace std;
int helper(int W, int wt[], int val[], int index, int** dp){
    if (index < 0)
        return 0;
    if (dp[index][W] != -1)
        return dp[index][W];

    if (wt[index] > W) {
        dp[index][W] = helper(W, wt, val, index - 1, dp);
        return dp[index][W];
    }
    else {
        dp[index][W] = max(val[index]+helper(W - wt[index], wt, val,index
- 1, dp),helper(W, wt, val, index - 1, dp));
        return dp[index][W];
    }
}

int knapSack(int W, int wt[], int val[], int n){
    int** dp;
    dp = new int*[n];
    for (int i = 0; i < n; i++)
        dp[i] = new int[W + 1];
    for (int i = 0; i < n; i++)
        for (int j = 0; j < W + 1; j++)
            dp[i][j] = -1;
    return helper(W, wt, val, n - 1, dp);
}
int main()
{
```

```
    int profit[] = { 60, 100, 120 };
    int weight[] = { 10, 20, 30 };
    int W = 50;
    int n = sizeof(profit) / sizeof(profit[0]);
    cout << knapSack(W, weight, profit, n);
    return 0;
}
```

Time Complexity: O(2^N)
Space Complexity:O(N) Rec Stack

Question 2: Floor in sorted array

Answer:

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

int searchInsert(vector<int>& nums, int target) {
        int n = nums.size();
        int low = 0, high = n - 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> nums;
    int a;
      while (std::cin >> a) {
        nums.push_back(a);
      }
      int target;
      cout<<"Enter Target: ";
```

```
    cin>>target;
    int ans=searchInsert(nums,target);
    cout<<ans;

  return 0;
}
```

Time Complexity: O(log N)
Space Complexity:O(1)

Question3: Check equal arrays

Answer:

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

  bool check(vector<int>& arr1, vector<int>& arr2) {
    int n=arr1.size(),m=arr2.size();
    if(m!=n) return false;
    unordered_map<int,int> map;
    for(int i=0;i<n;i++){
      map[arr1[i]]++;
    }
    for(int i=0;i<n;i++){
      if(map.find(arr2[i])!=map.end()){
        if(map[arr2[i]]==1){
          map.erase(arr2[i]);
        }
        else {map[arr2[i]]--;}
      }
    }
    return map.size()==0;
  }
// Driver code
int main() {
  int n, element;
  vector<int> arr1, arr2;
  cout << "Enter the number of elements in the arrays: ";
  cin >> n;

  cout << "Enter elements of the first array: ";
```

```
    for (int i = 0; i < n; ++i) {
        cin >> element;
        arr1.push_back(element);
    }
    cout << "Enter elements of the second array: ";
    for (int i = 0; i < n; ++i) {
        cin >> element;
        arr2.push_back(element);
    }
    if (check(arr1, arr2)) {
        cout << "The arrays are equal" << endl;
    } else {
        cout << "The arrays are not equal" << endl;
    }

    return 0;
}
```

Time Complexity: O(N)
Space Complexity:O(N)

Question 4: Palindrome linked list

Answer:

```
#include <bits/stdc++.h>
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) {}
};

bool isPalindrome(ListNode* head) {
    string s;
    ListNode* temp = head;
    s += to_string(temp->val);
    while (temp->next != nullptr) {
        s += to_string(temp->next->val);
        temp = temp->next;
```

```
    }
    cout << "Original string: " << s << endl;
    string rev = s;
    reverse(rev.begin(), rev.end());
    return s == rev;
}

ListNode* create(const vector<int>& arr) {
    if (arr.empty()) return nullptr;
    ListNode* head = new ListNode(arr[0]);
    ListNode* current = head;
    for (size_t i = 1; i < arr.size(); ++i) {
        current->next = new ListNode(arr[i]);
        current = current->next;
    }
    return head;
}

int main() {
    vector<int> values = {1,2,2,1};
    ListNode* head = create(values);
    if (isPalindrome(head)) {
        cout << "The linked list is a palindrome." << endl;
    } else {
        cout << "The linked list is not a palindrome." << endl;
    }

    return 0;
}
```

Time Complexity: O(N)
Space Complexity:O(N)

Question5: Balanced tree check

Answer:

```
#include <bits/stdc++.h>
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) {}
};

int finder(TreeNode* root){
    if(root==nullptr) return 0;
    int l=finder(root->left);
    if(l==-1) return -1;
    int r=finder(root->right);
    if(r==-1) return -1;
    if(abs(l-r)>1) return -1;
    return 1+max(l,r);
}

bool isBalanced(TreeNode* root) {
    return(finder(root)!=-1);
}

TreeNode* newNode(int value) {
    return new TreeNode(value);
}


int main() {
    TreeNode* root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    root->right->right = newNode(6);
    if (isBalanced(root)) {
        cout << "The binary tree is balanced." << endl;
    } else {
        cout << "The binary tree is not balanced." << endl;
    }
    delete root->left->left;
    delete root->left->right;
    delete root->right->right;
    delete root->left;
    delete root->right;
```

```
    delete root;

    return 0;
}
```

Time Complexity: O(2^N)
Space Complexity: O(N)

Question 6: Triplet sum in array

Answer:

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

vector<vector<int>> threeSum(vector<int>& nums) {
        vector<vector<int>> ans;
        sort(nums.begin(),nums.end());

        for(int i=0;i<nums.size();i++){
            if(i>0 && nums[i]==nums[i-1]){
                continue;
            }
            int j=i+1;
            int k=nums.size()-1;
            while(j<k){
                int total=nums[i]+nums[j]+nums[k];
                if(total>0){
                    k--;
                }
                else if(total<0){
                    j++;
                }
                else{
                    ans.push_back({nums[i],nums[j],nums[k]});
                    j++;
                    while(nums[j]==nums[j-1] && j<k){
                        j++;
                    }
                }
```

```
            }
        }
        return ans;

}

int main(){
    vector<int> nums;
    int a;
      while (std::cin >> a) {
        nums.push_back(a);
      }
      vector<vector<int>> ans=threeSum(nums);
      for(vector<int> a:ans){
        for(int b:a){
            cout<<b<<" ";
        }
        cout<<endl;
      }
    return 0;

}
```

Time Complexity: O(N^2)
Space Complexity:O(1)