**10 Marks Coding Question**

**Problem Statement**

Given an array of integers **nums** and an integer **target**, find two numbers in the array such that they add up to **target**. Return the indices of the two numbers.

**Input Format:**

Sixe = 4

nums = [2,7,11,15]

Target = 9

**Output Format:**

0 1 (Because nums[0] + nums[1] == 9, we return [0, 1].)

**Code Constraints:**

2 <= nums.length <= 104

-109 <= nums[i] <= 109

-109 <= target <= 109

**Solution: (in required language)**

#include <iostream>

#include <vector>

#include <unordered\_map>

std::vector<int> twoSum(const std::vector<int>& nums, int target) {

std::unordered\_map<int, int> numMap;

std::vector<int> indices;

for (int i = 0; i < nums.size(); ++i) {

int complement = target - nums[i];

if (numMap.find(complement) != numMap.end()) {

indices.push\_back(numMap[complement]);

indices.push\_back(i);

return indices;

}

numMap[nums[i]] = i;

}

return indices;

}

int main() {

// Input size of the array

int size;

std::cout << "Enter the size of the array: ";

std::cin >> size;

// Input array elements

std::vector<int> nums(size);

std::cout << "Enter the elements of the array: ";

for (int i = 0; i < size; ++i) {

std::cin >> nums[i];

}

// Input target value

int target;

std::cout << "Enter the target value: ";

std::cin >> target;

// Find indices

std::vector<int> result = twoSum(nums, target);

// Print indices if found

if (!result.empty()) {

std::cout << "Indices of the two numbers that add up to " << target << " are: ";

std::cout << result[0] << " and " << result[1] << std::endl;

} else {

std::cout << "No two numbers found that add up to the target." << std::endl;

}

return 0;

}

**Sample Testcase 1**

Input: 4

2 7 11 15

9

Output: 0 1

**Sample Testcase 2**

Input:3

3 2 4

6

Output: 1 2

**Hidden Testcase 1 - (Easy) Weightage 10%**

Input: 2

3 3

6

Output:0 1

**Hidden Testcase 2 - (Easy) Weightage 10%**

Input:3

3 2 4

6

Output: 1 2

**Hidden Testcase 3 - (Medium) Weightage 15%**

Input: 4

2 7 11 15

6

Output: 0 1

**Hidden Testcase 4 - (Medium) Weightage 15%**

Input: 4

2 9 7 15

9

Output: 0 1

**Hidden Testcase 5 - (Hard) Weightage 25%**

Input:4

1 7 9 15

8

Output: 0 1

**Hidden Testcase 6 - (Hard) Weightage 25%**

Input: 4

1 7 9 8

8

Output:0 1

**10 Marks Coding Question**

**Problem Statement**

Given two integer arrays nums1 and nums2, find the intersection of the two arrays. The intersection is a set of elements that are common to both arrays. Return an array containing the unique elements of the intersection.

**Input Format:**

Size1=4

Size2=2

nums1 = [1,2,3,4]

nums2 = [2,6]

**Output Format:**

2

**Code Constraints:**

1 <= nums1.length, nums2.length <= 1000

0 <= nums1[i], nums2[i] <= 1000

**Solution: (in required language)**

#include <iostream>

#include <vector>

#include <unordered\_map>

void printCommonElements(const std::vector<int>& nums1, const std::vector<int>& nums2) {

std::unordered\_map<int, int> countMap;

for (int num : nums1) {

countMap[num]++;

}

std::cout << "Common elements in both arrays are: ";

for (int num : nums2) {

if (countMap[num] > 0) {

std::cout << num << " ";

countMap[num]--;

}

}

std::cout << std::endl;

}

int main() {

// Input size of the first array

int size1;

std::cout << "Enter the size of the first array: ";

std::cin >> size1;

// Input elements of the first array

std::vector<int> nums1(size1);

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

for (int i = 0; i < size1; ++i) {

std::cin >> nums1[i];

}

// Input size of the second array

int size2;

std::cout << "Enter the size of the second array: ";

std::cin >> size2;

// Input elements of the second array

std::vector<int> nums2(size2);

std::cout << "Enter the elements of the second array: ";

for (int i = 0; i < size2; ++i) {

std::cin >> nums2[i];

}

// Print common elements

printCommonElements(nums1, nums2);

return 0;

}

**Sample Testcase 1**

Input: 4

4

1 2 3 4

4 5 6 7

Output: 4

**Sample Testcase 2**

Input:2

2

1 2

2 3

Output: 2

**Hidden Testcase 1 - (Easy) Weightage 10%**

Input: 3

3

1 2 3

2 3 4

Output: 2 3

**Hidden Testcase 2 - (Easy) Weightage 10%**

Input: 3

3

1 2 4

4 5 6

Output: 4

**Hidden Testcase 3 - (Medium) Weightage 15%**

Input: 5

5

1 2 3 4 5

3 4 5 6 7

Output: 3 4 5

**Hidden Testcase 4 - (Medium) Weightage 15%**

Input:

2

2

1 3

3 5

Output: 3

**Hidden Testcase 5 - (Hard) Weightage 25%**

Input:4

4

1 2 3 5

4 5 6 7

Output: 5

**Hidden Testcase 6 - (Hard) Weightage 25%**

Input: 1

1

1

1

Output: 1

**10 Marks Coding Question**

**Problem Statement**

**You are given an array representing the structure of a binary tree where each element is a string containing either an integer value or the string "null". The array represents the level-order traversal of the binary tree (from left to right, level by level). Write a function to return the preorder traversal of the values of the nodes in the binary tree.**

**Input:**

**Input Format:**

**4**

[“1”,”null”,”2”,”3”]

**Output Format:**

**1 2 3**

**Code Constraints:**

**1<=N<=10**

**Solution: (in required language)**

**#include <iostream>**

**#include <vector>**

**#include <queue>**

**// Definition for a binary tree node.**

**struct TreeNode {**

**int val;**

**TreeNode \*left;**

**TreeNode \*right;**

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

**};**

**// Function to convert array to binary tree**

**TreeNode\* arrayToTree(const std::vector<int>& arr) {**

**if (arr.empty()) {**

**return nullptr;**

**}**

**TreeNode\* root = new TreeNode(arr[0]);**

**std::queue<TreeNode\*> q;**

**q.push(root);**

**for (int i = 1; i < arr.size(); i += 2) {**

**TreeNode\* current = q.front();**

**q.pop();**

**if (arr[i] != NULL) {**

**current->left = new TreeNode(arr[i]);**

**q.push(current->left);**

**}**

**if (i + 1 < arr.size() && arr[i + 1] != NULL) {**

**current->right = new TreeNode(arr[i + 1]);**

**q.push(current->right);**

**}**

**}**

**return root;**

**}**

**// Function to perform a preorder traversal of the binary tree**

**void preorderTraversal(TreeNode\* root) {**

**if (root == nullptr) {**

**return;**

**}**

**std::cout << root->val << " ";**

**preorderTraversal(root->left);**

**preorderTraversal(root->right);**

**}**

**int main() {**

**// Input the array representation from the user**

**std::cout << "Enter the size of the array: ";**

**int size;**

**std::cin >> size;**

**std::vector<int> arr(size);**

**std::cout << "Enter the array elements (use 'null' for NULL): ";**

**for (int i = 0; i < size; ++i) {**

**std::string element;**

**std::cin >> element;**

**if (element == "null") {**

**arr[i] = NULL;**

**} else {**

**arr[i] = std::stoi(element);**

**}**

**}**

**// Convert array to binary tree**

**TreeNode\* root = arrayToTree(arr);**

**// Perform preorder traversal to verify the tree structure**

**std::cout << "Preorder Traversal: ";**

**preorderTraversal(root);**

**// Clean up allocated memory**

**// You can implement a function to delete the tree nodes**

**return 0;**

**}**

**Sample Testcase 1**

Input: 4

1 null 2 3

Output: 1 2 3

**Sample Testcase 2**

Input:1

1

Output: 1

**Hidden Testcase 1 - (Easy) Weightage 10%**

Input:4

1 2 2 3

Output: 1 2 3 2

**Hidden Testcase 2 - (Easy) Weightage 10%**

Input:7

1 2 2 3 4 4 5

Output: 1 2 3 4 2 4 5

**Hidden Testcase 3 - (Medium) Weightage 15%**

Input:4

1 null 2 3

Output: 1 2 3

**Hidden Testcase 4 - (Medium) Weightage 15%**

Input:7

1 2 2 3 null 9 5

Output: 1 2 3 2 9 5

**Hidden Testcase 5 - (Hard) Weightage 25%**

Input:7

1 2 2 3 null null 5

Output: 1 2 3 2 5

**Hidden Testcase 6 - (Hard) Weightage 25%**

Input: 7

1 2 2 10 null null 5

Output: 1 2 10 2 5

**10 Marks Coding Question**

**Problem Statement**

**You are given an array representing the structure of a binary tree where each element is a string containing either an integer value or the string "null". The array represents the level-order traversal of the binary tree (from left to right, level by level). Write a function to return the postorder traversal of the values of the nodes in the binary tree.**

**Input:Input Format:**

**4**

[“1”,”null”,”2”,”3”]

**Output Format:**

3 2 1

**Code Constraints:**

**1<=N<=10**

**Solution: (in required language)**

**#include <iostream>**

**#include <vector>**

**#include <queue>**

**// Definition for a binary tree node.**

**struct TreeNode {**

**int val;**

**TreeNode \*left;**

**TreeNode \*right;**

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

**};**

**// Function to convert array to binary tree**

**TreeNode\* arrayToTree(const std::vector<int>& arr) {**

**if (arr.empty()) {**

**return nullptr;**

**}**

**TreeNode\* root = new TreeNode(arr[0]);**

**std::queue<TreeNode\*> q;**

**q.push(root);**

**for (int i = 1; i < arr.size(); i += 2) {**

**TreeNode\* current = q.front();**

**q.pop();**

**if (arr[i] != NULL) {**

**current->left = new TreeNode(arr[i]);**

**q.push(current->left);**

**}**

**if (i + 1 < arr.size() && arr[i + 1] != NULL) {**

**current->right = new TreeNode(arr[i + 1]);**

**q.push(current->right);**

**}**

**}**

**return root;**

**}**

**// Function to perform a postorder traversal of the binary tree**

**void postorderTraversal(TreeNode\* root) {**

**if (root == nullptr) {**

**return;**

**}**

**postorderTraversal(root->left);**

**postorderTraversal(root->right);**

**std::cout << root->val << " ";**

**}**

**int main() {**

**// Input the array representation from the user**

**std::cout << "Enter the size of the array: ";**

**int size;**

**std::cin >> size;**

**std::vector<int> arr(size);**

**std::cout << "Enter the array elements (use 'null' for NULL): ";**

**for (int i = 0; i < size; ++i) {**

**std::string element;**

**std::cin >> element;**

**if (element == "null") {**

**arr[i] = NULL;**

**} else {**

**arr[i] = std::stoi(element);**

**}**

**}**

**// Convert array to binary tree**

**TreeNode\* root = arrayToTree(arr);**

**// Perform postorder traversal**

**std::cout << "Postorder Traversal: ";**

**postorderTraversal(root);**

**std::cout << std::endl;**

**// Clean up allocated memory**

**// You can implement a function to delete the tree nodes**

**return 0;**

**}**

**Sample Testcase 1**

Input: 4

1 null 2 3

Output: 3 2 1

**Sample Testcase 2**

Input:1

1

Output: 1

**Hidden Testcase 1 - (Easy) Weightage 10%**

Input:4

1 null 4 3

Output: 3 4 1

**Hidden Testcase 2 - (Easy) Weightage 10%**

Input:6

1 null 4 3 null 5

Output: 5 3 4 1

**Hidden Testcase 3 - (Medium) Weightage 15%**

Input: 6

1 2 4 3 null 5

Output: 3 2 5 4 1

**Hidden Testcase 4 - (Medium) Weightage 15%**

Input: 6

1 2 4 3 5 6

Output: 3 6 2 5 4 1

**Hidden Testcase 5 - (Hard) Weightage 25%**

Input:7

1 2 4 3 6 5 7

Output: 3 6 2 5 7 4 1

**Hidden Testcase 6 - (Hard) Weightage 25%**

Input: 4

1 null 2 3

Output: 3 2 1

**10 Marks Coding Question**

**Problem Statement**

**You are given an array representing the structure of a binary tree where each element is a string containing either an integer value or the string "null". The array represents the level-order traversal of the binary tree (from left to right, level by level). Write a function to return the inorder traversal of the values of the nodes in the binary tree.**

**Input:Input Format:**

**4**

[“1”,”null”,”2”,”3”]

**Output Format:**

1 3 2

**Code Constraints:**

**1<=N<=10**

**Solution: (in required language)**

#include <iostream>

#include <vector>

#include <queue>

// Definition for a binary tree node.

struct TreeNode {

int val;

TreeNode \*left;

TreeNode \*right;

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

};

// Function to convert array to binary tree

TreeNode\* arrayToTree(const std::vector<int>& arr) {

if (arr.empty()) {

return nullptr;

}

TreeNode\* root = new TreeNode(arr[0]);

std::queue<TreeNode\*> q;

q.push(root);

for (int i = 1; i < arr.size(); i += 2) {

TreeNode\* current = q.front();

q.pop();

if (arr[i] != NULL) {

current->left = new TreeNode(arr[i]);

q.push(current->left);

}

if (i + 1 < arr.size() && arr[i + 1] != NULL) {

current->right = new TreeNode(arr[i + 1]);

q.push(current->right);

}

}

return root;

}

// Function to perform an inorder traversal of the binary tree

void inorderTraversal(TreeNode\* root) {

if (root == nullptr) {

return;

}

inorderTraversal(root->left);

std::cout << root->val << " ";

inorderTraversal(root->right);

}

int main() {

// Input the array representation from the user

std::cout << "Enter the size of the array: ";

int size;

std::cin >> size;

std::vector<int> arr(size);

std::cout << "Enter the array elements (use 'null' for NULL): ";

for (int i = 0; i < size; ++i) {

std::string element;

std::cin >> element;

if (element == "null") {

arr[i] = NULL;

} else {

arr[i] = std::stoi(element);

}

}

// Convert array to binary tree

TreeNode\* root = arrayToTree(arr);

// Perform inorder traversal

std::cout << "Inorder Traversal: ";

inorderTraversal(root);

std::cout << std::endl;

// Clean up allocated memory

// You can implement a function to delete the tree nodes

return 0;

}

**Sample Testcase 1**

Input: **4**

**1 null 2 3**

Output: 1 3 2

**Sample Testcase 2**

Input:5

1 null 2 3 5

Output: 1 3 2 5

**Hidden Testcase 1 - (Easy) Weightage 10%**

Input: 6

1 null 2 3 5 7

Output: 1 7 3 2 5

**Hidden Testcase 2 - (Easy) Weightage 10%**

Input: 6

1 2 2 3 5 7

Output: 3 2 5 1 7 2

**Hidden Testcase 3 - (Medium) Weightage 15%**

Input: 6

1 2 2 3 9 7

Output: 3 2 9 1 7 2

**Hidden Testcase 4 - (Medium) Weightage 15%**

Input: 7

1 2 2 3 9 7 8

Output: 3 2 9 1 7 2 8

**Hidden Testcase 5 - (Hard) Weightage 25%**

Input:7

1 5 2 3 9 7 8

Output: 3 5 9 1 7 2 8

**Hidden Testcase 6 - (Hard) Weightage 25%**

Input: 7

1 5 2 3 null 7 8

Output: 3 5 1 7 2 8

**10 Marks Coding Question**

**Problem Statement**

**You are given an array representing the structure of a binary tree where each element is a string containing either an integer value or the string "null". The array represents the level-order traversal of the binary tree (from left to right, level by level). Write a function to return the level order traversal of the values of the nodes in the binary tree.**

**Input:4**

1 null 2 3

**Output Format:**

1 2 3

**Code Constraints:**

**1<=N<=10**

**Solution: (in required language)**

**#include <iostream>**

**#include <vector>**

**#include <queue>**

**// Definition for a binary tree node.**

**struct TreeNode {**

**int val;**

**TreeNode \*left;**

**TreeNode \*right;**

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

**};**

**// Function to convert array to binary tree**

**TreeNode\* arrayToTree(const std::vector<int>& arr) {**

**if (arr.empty()) {**

**return nullptr;**

**}**

**TreeNode\* root = new TreeNode(arr[0]);**

**std::queue<TreeNode\*> q;**

**q.push(root);**

**for (int i = 1; i < arr.size(); i += 2) {**

**TreeNode\* current = q.front();**

**q.pop();**

**if (arr[i] != NULL) {**

**current->left = new TreeNode(arr[i]);**

**q.push(current->left);**

**}**

**if (i + 1 < arr.size() && arr[i + 1] != NULL) {**

**current->right = new TreeNode(arr[i + 1]);**

**q.push(current->right);**

**}**

**}**

**return root;**

**}**

**// Function to perform a level order traversal of the binary tree**

**void levelOrderTraversal(TreeNode\* root) {**

**if (root == nullptr) {**

**return;**

**}**

**std::queue<TreeNode\*> q;**

**q.push(root);**

**while (!q.empty()) {**

**TreeNode\* current = q.front();**

**q.pop();**

**std::cout << current->val << " ";**

**if (current->left != nullptr) {**

**q.push(current->left);**

**}**

**if (current->right != nullptr) {**

**q.push(current->right);**

**}**

**}**

**}**

**int main() {**

**// Input the array representation from the user**

**std::cout << "Enter the size of the array: ";**

**int size;**

**std::cin >> size;**

**std::vector<int> arr(size);**

**std::cout << "Enter the array elements (use 'null' for NULL): ";**

**for (int i = 0; i < size; ++i) {**

**std::string element;**

**std::cin >> element;**

**if (element == "null") {**

**arr[i] = NULL;**

**} else {**

**arr[i] = std::stoi(element);**

**}**

**}**

**// Convert array to binary tree**

**TreeNode\* root = arrayToTree(arr);**

**// Perform level order traversal**

**std::cout << "Level Order Traversal: ";**

**levelOrderTraversal(root);**

**std::cout << std::endl;**

**// Clean up allocated memory**

**// You can implement a function to delete the tree nodes**

**return 0;**

**}**

**Sample Testcase 1**

Input: 7

3 9 20 null null 15 7

Output:

3 9 20 15 7

**Sample Testcase 2**

Input:1

1

Output: 1

**Hidden Testcase 1 - (Easy) Weightage 10%**

Input:4

1 null 2 3

Output: 1 2 3

**Hidden Testcase 2 - (Easy) Weightage 10%**

Input:4

1 2 2 3

Output: 1 2 2 3

**Hidden Testcase 3 - (Medium) Weightage 15%**

Input: 1 2 null 3

Output: 1 2 3

**Hidden Testcase 4 - (Medium) Weightage 15%**

Input: 4

1 2 5 3

Output: 1 2 5 3

**Hidden Testcase 5 - (Hard) Weightage 25%**

Input:5

1 2 5 3 6

Output: 1 5 3 6

**Hidden Testcase 6 - (Hard) Weightage 25%**

Input: 6

1 2 5 3 6 9

Output: 1 2 5 3 6 9