Q-1: You have given an unsorted integer array the task is to find a pair of elements in array with the given sum in it. Make use of hashing to solve this problem.

Sample test case:

|  |
| --- |
| Input: nums = [8, 7, 2, 5, 3, 1]  target = 10  Output:  (8, 2) or (7, 3) |

Solution:

#include <iostream>

#include <unordered\_map>

using namespace std;

// Function to find a pair in an array with a given sum using hashing

void findPair(int nums[], int n, int target)

{

// create an empty map

unordered\_map<int, int> map;

// do for each element

for (int i = 0; i < n; i++)

{

// check if pair (nums[i], target - nums[i]) exists

// if the difference is seen before, print the pair

if (map.find(target - nums[i]) != map.end())

{

cout << "(" << nums[map[target - nums[i]]] << ", "

<< nums[i] << ")\n";

return;

}

// store index of the current element in the map

map[nums[i]] = i;

}

// we reach here if the pair is not found

cout << "Pair not found";

}

int main()

{

int nums[] = { 8, 7, 2, 5, 3, 1 };

int target = 10;

int n = sizeof(nums)/sizeof(nums[0]);

findPair(nums, n, target);

return 0;

}

Q-2: Given a binary tree, write a cpp program using hashing to print right view of binary tree using hashing technique.

Sample test case:

|  |
| --- |
| Input:  1  / \  2 3  \ / \  4 5 6  / \  7 8  Output: 1 3 6 8 |

Solution:

#include <iostream>

#include <unordered\_map>

using namespace std;

// Data structure to store a binary tree node

struct Node

{

int key;

Node \*left, \*right;

Node(int key)

{

this->key = key;

this->left = this->right = nullptr;

}

};

// Traverse nodes in reverse preorder fashion

void printRightView(Node\* root, int level, auto &map)

{

if (root == nullptr) {

return;

}

// insert the current node and level information into the map

map[level] = root->key;

// recur for the left subtree before the right subtree

printRightView(root->left, level + 1, map);

printRightView(root->right, level + 1, map);

}

// Function to print the right view of a given binary tree

int printRightView(Node\* root)

{

// create an empty map to store the last node for each level

unordered\_map<int, int> map;

// traverse the tree and fill the map

printRightView(root, 1, map);

// iterate through the map and print the right view

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

cout << map[i] << " ";

}

}

int main()

{

Node\* root = new Node(1);

root->left = new Node(2);

root->right = new Node(3);

root->left->right = new Node(4);

root->right->left = new Node(5);

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

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

root->right->left->right = new Node(8);

printRightView(root);

return 0;

}

Q-3: You have given an array of distinct elements and a range [low, high], find all numbers that are in a range, but not present array. Print missing elements in sorted order. Use hashing to achieve same.

Sample test case:

|  |
| --- |
| Input: arr[] = {10, 12, 11, 15},  low = 10, high = 15  Output: 13, 14 |

Solution:

#include <bits/stdc++.h>

using namespace std;

// Function to find and print missing elements in the given range

void findMissing(int arr[], int n, int low, int high) {

// Loop through the range of numbers from low to high

for (int i = low; i <= high; i++) {

bool found = false;

// Loop through the array to check if i exists in it

for (int j = 0; j < n; j++) {

if (arr[j] == i) {

found = true;

break;

}

}

// If i is not found in the array, print it

if (!found) {

cout << i << " ";

}

}

}

int main() {

// Input array

int arr[] = { 1, 3, 5, 4 };

int n = sizeof(arr) / sizeof(arr[0]);

int low = 1, high = 10;

// Function call

findMissing(arr, n, low, high);

return 0;

}