#include <cstdlib>

#include <string>

#include <vector>

#include <iostream>

#include <unordered\_map>

#include <algorithm>

using namespace std;

class BinaryNode {

public:

int value;

BinaryNode \*left = nullptr;

BinaryNode \*right = nullptr;

BinaryNode(int v = 0)

{

value = v;

}

};

// Helper function: expects a hashtable containing frequencies for

// each of its keys; returns the key with the highest frequency.

template<class T>

T findMaxKey(unordered\_map<T, int> hashtbl)

{

T max\_key{};

for (auto kvp : hashtbl)

{

if (kvp.second > hashtbl[max\_key])

{

max\_key = kvp.first;

}

}

return max\_key;

}

// helper function for creating BST

BinaryNode\* treeInsert(BinaryNode\* node, int val)

{

if (node == nullptr)

{

BinaryNode\* new\_node = new BinaryNode{};

new\_node->value = val;

return new\_node;

}

if (val < node->value)

{

BinaryNode\* left = treeInsert(node->left, val);

node->left = left;

}

else

{

BinaryNode\* right = treeInsert(node->right, val);

node->right = right;

}

return node;

}

// Task 1: Write a function that sorts an array using Bubble Sort.

void bubbleSort(int\* numbers, int size)

{

// i represents the number of elements that have been put in

// their proper place so far

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

{

// j represents how many comparisons have been made in the

// current pass, up to size - i - 1 (where size = total

// number of elements, i = number of elements that have

// already been sorted, and minus 1 because to sort n items

// we do n-1 comparisons)

for (int j = 0; j < size - i - 1; j++)

{

// if numbers[j] > numbers[j+1], swap so that we bubble

// the largest number to the end of the array

if (numbers[j] > numbers[j + 1])

{

int temp = numbers[j + 1];

numbers[j + 1] = numbers[j];

numbers[j] = temp;

}

}

}

return;

}

/\* Task 2: Implement a function that parses the supplied vector

of integers representing rainfall data and returns the day (as

a string) that has the most rain. The input vector is formatted

such that index 0 represents Sunday and every 7th item is

the same day. Thus:

\* 0, 7, 14... is a Sunday

\* 1, 8, 15... is a Monday

\* 2, 9, 16... is a Tuesday

etc.

\*/

string dayWithMostRain(vector<int> rainfall)

{

// Set up a hashtable containing strings corresponding to

// each day of the week.

unordered\_map<int, string> days\_of\_week{};

days\_of\_week[0] = "Sunday";

days\_of\_week[1] = "Monday";

days\_of\_week[2] = "Tuesday";

days\_of\_week[3] = "Wednesday";

days\_of\_week[4] = "Thursday";

days\_of\_week[5] = "Friday";

days\_of\_week[6] = "Saturday";

// Set up a hashtable to contain the rainfall data for each

// day of the week.

unordered\_map<int, int> rain\_per\_day;

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

{

// Add the rainfall corresponding to i's weekday to the

// running total for that weekday.

rain\_per\_day[i % 7] += rainfall[i];

}

// Determine which day of the week had the most rainfall

// out of all 7 days in the hashtable.

int highest\_rain\_day = findMaxKey<int>(rain\_per\_day);

// Return the string corresponding to that day.

return days\_of\_week[highest\_rain\_day];

}

/\* Task 3: Write a function called bstToVector that returns the

supplied binary search tree into a sorted STL vector.

\*/

// helper function for bstToVector.

// expects a vector and a pointer to the root of a binary search tree

// returns nothing

// pushes values from the binary search tree into the vector via an

// in-order traversal so that the vector is in sorted order

void inOrderTraversal(vector<int>& sorted, BinaryNode\* node)

{

// check for nullness

if (node == nullptr)

{

return;

}

// left

inOrderTraversal(sorted, node->left);

// us

sorted.push\_back(node->value);

// right

inOrderTraversal(sorted, node->right);

}

vector<int> bstToVector(BinaryNode\* node)

{

vector<int> sortedBst{};

// use an in-order traversal to get the values of the tree into

// the vector in sorted order

inOrderTraversal(sortedBst, node);

// then sort the resulting vector, just in case the tree we

// were passed wasn't an actual BST

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

return sortedBst;

}

/\* Task 4: Complete the function isBst() such that the function

returns TRUE when the supplied node represents a valid BST,

and FALSE otherwise.

\*/

// helper function for task 4, using a recursive solution

bool isBstHelper(BinaryNode\* node, int min, int max)

{

// check for nullness

if (node == nullptr)

{

return true;

}

// check for numerical validity-- e.g. NONE of the values on

// the left side of the tree should be >= the root;

// none of the values on the right side of the tree should be

// < the root

if (node->value < min || node->value > max)

{

return false;

}

return isBstHelper(node->left, min, node->value - 1)

&& isBstHelper(node->right, node->value, max);

}

bool isBst(BinaryNode\* node)

{

// A binary tree is a BST if all left children are less than

// their parent, and if all right children are greater than or

// equal to their parent.

return isBstHelper(node, INT\_MIN, INT\_MAX);

}

/\* Task 5: Write a function called findMostCommon that finds the

most commonly used character in the supplied string.

\*/

char findMostCommon(string text)

{

// set up hashtable to contain frequencies of characters in this

// string

unordered\_map<char, int> char\_freq{};

// get the characters from the string and put their frequencies

// into the hashtable

for (auto i : text)

{

char\_freq[i]++;

}

// return the character with the highest frequency

return findMaxKey<char>(char\_freq);

}

int main(void)

{

cout << boolalpha;

// 1: bubbleSort

int numbers[10] = { 4, 6, 1, 4, 66, 2, 16, 4, 6, 1 };

bubbleSort(numbers, 10);

cout << "Problem 1: " << endl;

for (int i = 0; i < 10; i++)

{

cout << numbers[i] << " ";

}

cout << endl;

// 2: rainfall data

cout << "Problem 2: " << endl;

vector<int> sample\_rainfall{};

for (int i = 2; i < 16; i++)

{

sample\_rainfall.push\_back(i);

}

sample\_rainfall.push\_back(0);

sample\_rainfall.push\_back(0);

sample\_rainfall.push\_back(2364);

cout << "Most rainfall happened on Tuesday: "

<< dayWithMostRain(sample\_rainfall) << endl;

// 3: BST to sorted vector

cout << "Problem 3: " << endl;

// null tree

BinaryNode\* root{};

cout << "Null tree: ";

vector<int> sorted = bstToVector(root);

for (auto i : sorted)

{

cout << i << " ";

}

cout << endl;

// one element

root = treeInsert(root, 5);

cout << "One element: ";

sorted = bstToVector(root);

for (auto i : sorted)

{

cout << i << " ";

}

cout << endl;

// 2+ elements

root = treeInsert(root, 10);

root = treeInsert(root, 2);

root = treeInsert(root, 9);

root = treeInsert(root, 1);

cout << "2+ elements: ";

sorted = bstToVector(root);

for (auto i : sorted)

{

cout << i << " ";

}

cout << endl;

// problem 4

cout << "Problem 4 " << endl;

// null tree

BinaryNode\* root4{};

cout << "Is a null tree a BST? Should be true: ";

cout << isBst(root4) << endl;

// one element

root4 = treeInsert(root4, 5);

cout << "Is a one-element tree a BST? Should be true: ";

cout << isBst(root4) << endl;

// 2+ elements

root4 = treeInsert(root4, 10);

root4 = treeInsert(root4, 2);

root4 = treeInsert(root4, 9);

root4 = treeInsert(root4, 1);

cout << "Is a 2+ element BST a BST? Should be true: ";

cout << isBst(root4) << endl;

cout << "Is a wack-ass tree a BST: Should be false: ";

BinaryNode\* wack\_tree = new BinaryNode(5);

wack\_tree->left = new BinaryNode(2);

wack\_tree->left->left = new BinaryNode(1);

wack\_tree->left->right = new BinaryNode(4);

wack\_tree->left->right->left = new BinaryNode(3);

wack\_tree->left->right->right = new BinaryNode(6);

wack\_tree->right = new BinaryNode(8);

wack\_tree->right->left = new BinaryNode(7);

wack\_tree->right->right = new BinaryNode(10);

cout << isBst(wack\_tree) << endl;

// Problem 5

cout << "Problem 5: " << endl;

cout << "Apple: " << findMostCommon("apple") << endl;

cout << "Banana: " << findMostCommon("banana") << endl;

cout << "Empty string: " << findMostCommon("") << endl;

cout << "kahwoiehoiadsfkcxxxxx: " << findMostCommon("kahwoiehoiadsfkcxxxxx") << endl;

return 0;

}