Reto Semestral – Ronda 2

Layla Tame A0119234

**Problema 169 – Majority Element (Easy, LeetCode)**

//

// main.cpp

// problem1

//

// Created by Layla Tame on 2/27/19.

// Copyright © 2019 Layla Tame. All rights reserved.

//

#include <iostream>

#include <algorithm>

#include <vector>

#include <sstream>

using namespace std;

//Reto Semestral - Problema 1. Majority Element

int input, iSize = 0;

string line;

vector<int> numbers;

void readVector()

{

cout << "Enter array: " << endl;

getline(cin, line);

istringstream stream(line);

while(stream >> input)

{

numbers.push\_back(input);

iSize++;

}

}

int majorityElement(vector<int> numbers, int iSize)

{

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

return numbers[iSize/2];

}

int main() {

readVector();

cout << majorityElement(numbers, iSize);

}

A screenshot of a cell phone

Description automatically generated

**Problema 2 – Tree Postorder Traversal (Easy, hackerrank)**

//

// main.cpp

// problem1

//

// Created by Layla Tame on 2/27/19.

// Copyright © 2019 Layla Tame. All rights reserved.

//

#include <bits/stdc++.h>

using namespace std;

class Node {

public:

int data;

Node \*left;

Node \*right;

Node(int d) {

data = d;

left = NULL;

right = NULL;

}

};

class Solution {

public:

      Node\* insert(Node\* root, int data) {

if(root == NULL) {

return new Node(data);

} else {

Node\* cur;

if(data <= root->data) {

cur = insert(root->left, data);

root->left = cur;

} else {

cur = insert(root->right, data);

root->right = cur;

}

return root;

}

}

/\* you only have to complete the function given below.

Node is defined as

class Node {

public:

int data;

Node \*left;

Node \*right;

Node(int d) {

data = d;

left = NULL;

right = NULL;

}

};

\*/

void postOrder(Node \*root) {

if(root == NULL){

return;

}

postOrder(root->left);

postOrder(root->right);

cout << root->data << " ";

}

}; //End of Solution

A screenshot of a cell phone

Description automatically generated

**Problema 3 – Is this a binary search tree? (Medium – hackerrank)**

bool checkLeft(Node\* root, int iRoot)

{

if (root == NULL)

{

return true;

}

if (root->data > iRoot)

{

return false;

}

return checkLeft(root->left, root->data) && checkRight(root->right, root->data) && checkLeft(root->left, iRoot) && checkLeft(root->right, iRoot);

}

bool checkRight(Node\* root, int iRoot)

{

if (root == NULL)

{

return true;

}

if (root->data <= iRoot)

{

return false;

}

return checkRight(root->right, root->data) && checkLeft(root->left, root->data) && checkRight(root->right, iRoot) && checkRight(root->left, iRoot);

}

bool checkBST(Node\* root)

{

return checkLeft(root->left, root->data) && checkRight(root->right, root->data);

}

A screenshot of a cell phone

Description automatically generated

**Problema 4 (938) – Range Sum of BST (Medium, leetcode)**

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

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

\* };

\*/

class Solution {

public:

int rangeSumBST(TreeNode\* root, int L, int R) {

if (root != NULL)

{

if(root->val >= L && root->val <= R)

{

return root->val + rangeSumBST(root->left, L, R) + rangeSumBST(root->right, L, R);

}

else if(root->val < L)

{

return rangeSumBST(root->right, L, R);

}

else if(root->val > R)

{

return rangeSumBST(root->left, L, R);

}

}

return 0;

}

};

A screenshot of a cell phone

Description automatically generated

**Problema 5 – Closest points (Hard, tarea 5)**

//

// main.cpp

// Tarea 5

//

// Created by Layla Tame on 2/20/19.

// Copyright © 2019 Layla Tame. All rights reserved.

//

// A divide and conquer program in C++ to find the smallest distance from a

// given set of points.

#include <iostream>

#include <float.h>

#include <math.h>

#include <fstream>

#include <vector>

#include <algorithm>

using namespace std;

//Stores id and coordenates read from file

struct Point

{

string id;

int x, y;

};

//Stores distance between two planes and their ids

struct Planes

{

string id1, id2;

float dist;

};

int iSize;

Planes fDistance = {"AM0000", "AM0000", FLT\_MAX};

vector<Point> vPoints;

Planes closestPlanes;

//Reads data from several files and stores it in a Point vector

//Counts number of elements read

void readFile() {

int iX, iY, cont = -1;

iSize = 0;

string sId, sArchivo;

ifstream fVuelos;

cout << "Nombre del archivo a leer: ";

cin >> sArchivo;

sArchivo = sArchivo + ".txt";

while (cont != 0)

{

fVuelos.open(sArchivo);

while (!fVuelos.eof())

{

fVuelos >> sId >> iX >> iY;

vPoints.push\_back({sId, iX, iY});

iSize++;

}

fVuelos.close();

cout << "Si desea procesar otro archivo?" << endl;

cout << "1. Sí" << endl;

cout << "2. No" << endl;

cin >> cont;

if (cont == 1)

{

cout << "Nombre del archivo a leer: ";

cin >> sArchivo;

sArchivo = sArchivo + ".txt";

}

if (cont == 2)

{

cont = 0;

}

}

}

//Compares two points and returns the minimum

bool compareX(Point p1, Point p2)

{

return p1.x < p2.x;

}

//Compares two points and returns the minimum

bool compareY(Point p1, Point p2)

{

return p1.y < p2.y;

}

void print(vector<Point> vPoints, int iSize)

{

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

{

cout << vPoints[i].id;

cout << " " << vPoints[i].x;

cout << " " << vPoints[i].y;

cout << endl;

}

}

//Returns two planes with the minimun distance and their ids

Planes min(Planes p1, Planes p2)

{

if (p1.dist < p2.dist)

{

return closestPlanes = {p1.id1, p1.id2, p1.dist};

}

else

{

return closestPlanes = {p2.id1, p2.id2, p2.dist};

}

}

//Calculates distance between two points

float distTwoPoints(Point p1, Point p2)

{

return sqrt(abs((p1.x - p2.x)\*(p1.x - p2.x)) + abs((p1.y - p2.y)\*(p1.y - p2.y)));

}

//Divides original vector into smaller ones following divide and conquer technique until vector is smaller than 4

//Obtains the two points with the least distance between them

//Returns structure Plane with the ids of the two closest planes and the distance between them

Planes divideConquer(vector<Point> vPointsX, vector<Point> vPointsY, int iSize)

{

sort(vPointsX.begin(), vPointsX.end(), compareX);

vector<Point> pointsLeft;

vector<Point> pointsRight;

Point midPoint;

Planes dist1, dist2;

if (iSize < 4) {

Planes tempDist1, tempDist2, tempDist3;

tempDist1 = {vPointsX[0].id, vPointsX[1].id, distTwoPoints(vPointsX[0], vPointsX[1])};

if(iSize == 3)

{

tempDist2 = {vPointsX[0].id, vPointsX[2].id, distTwoPoints(vPointsX[0], vPointsX[2])};

tempDist3 = {vPointsX[1].id, vPointsX[2].id, distTwoPoints(vPointsX[1], vPointsX[2])};

tempDist2 = min(tempDist2, tempDist3);

tempDist1 = min(tempDist2, tempDist1);

}

fDistance = min(fDistance, tempDist1);

}

else {

int mid = iSize/2;

midPoint = {vPointsX[mid].id, vPointsX[mid].x, vPointsX[mid].y};

int iLeft = 0, iRight = 0;

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

{

if (vPointsY[i].x < midPoint.x) {

pointsLeft.push\_back({vPointsY[i].id, vPointsY[i].x, vPointsY[i].y});

iLeft++;

}

else {

pointsRight.push\_back({vPointsY[i].id, vPointsY[i].x, vPointsY[i].y});

iRight++;

}

}

dist1 = divideConquer(pointsLeft, pointsLeft, iLeft);

dist2 = divideConquer(pointsRight, pointsRight, iRight);

fDistance = min(dist1, dist2);

//Create a vector to check between points separated into two different vectors determined by the smaller distance obtained of each.

vector<Point> midDistance;

int distX, iCount = 0;

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

{

distX = abs(midPoint.x - vPointsX[i].x);

if (distX < fDistance.dist)

{

midDistance.push\_back({vPointsX[i].id, vPointsX[i].x, vPointsX[i].y});

iCount++;

}

}

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

{

for (int j = i+1; j<iCount; j++)

{

if(distTwoPoints(midDistance[i], midDistance[j]) < fDistance.dist)

{

fDistance = {midDistance[i].id, midDistance[j].id, distTwoPoints(midDistance[i], midDistance[j])};

}

}

}

}

return fDistance;

}

int main() {

readFile();

vector<Point> vPointsX = vPoints, vPointsY = vPoints;

sort(vPointsX.begin(), vPointsX.end(), compareX);

sort(vPointsY.begin(), vPointsY.end(), compareY);

divideConquer(vPointsX, vPointsY, iSize);

cout << "Closest planes are " << fDistance.id1 << " and " << fDistance.id2 << " with a distance of " << fDistance.dist << endl;

}

A screenshot of a cell phone

Description automatically generated

\*No funciona con el archive grande, solamente con los archivos prueba.