Tree with Hashing/ Tree view from many angle

Used Node and Class/Interface in single link list

**public** **class** HTreeNode {

**public** **int** dt;

**public** HTreeNode lt, rt;

**public** **int** index; // index lt is root-- and rt=root++

**public** **int** hIndex; // height index root at 0 lt, rt=root+1

**public** HTreeNode(**int** dt) {

**this**.dt = dt;

**this**.lt = **this**.rt = **null**;

**this**.index = 0;

**this**.hIndex = 0;

}

}

===========================

**public** **interface** DAIHashingWithTree {

**public** HTreeNode createBstIndexTree(HTreeNode node, **int** dt, **int** index);

/\* arrangement of node in index/height as key of map by inorder traverse \*/

**public** **void** nodeOnIndex(Map<Integer, List<HTreeNode>> map, HTreeNode node);

**public** **void** nodeOnLabel(Map<Integer, List<HTreeNode>> map, HTreeNode node);

**public** **void** nodeOnLabelMinusIndex(Map<Integer, List<HTreeNode>> map, HTreeNode node);

**public** **void** nodeOnLabelPlusIndex(Map<Integer, List<HTreeNode>> map, HTreeNode node);

**public** Map<Integer, List<HTreeNode>> printColumWise(HTreeNode node);

/\*\* side view of tree top, bottom, left and right \*\*/

**public** List<Integer> printBottomView(HTreeNode node);

**public** List<Integer> printTomView(HTreeNode node);

/\* this is also the same of +90 degree rotational view \*/

**public** List<Integer> printLeftView(HTreeNode node);

/\* this is also the same of -90 degree rotational view \*/

**public** List<Integer> printRightView(HTreeNode node);

/\*\* corner view of tree top-left, bottom-left, bottom-right and top-right \*\*/

**public** List<Integer> topLeftCornerView(HTreeNode node);

**public** List<Integer> bottomRightCornerView(HTreeNode node);

**public** List<Integer> bottomLeftCornerView(HTreeNode node);

**public** List<Integer> topRightCornerView(HTreeNode node);

**public** **void** printInorder(HTreeNode node);

**public** **void** printHashMap(Map<Integer, Integer> map);

}

=====================================================================================Implementation of interface

-------------------------------------------------------------------------------------

**public** **class** DAIHashingWithTreeImpl **implements** DAIHashingWithTree {

Map<Integer, Integer> map = **new** HashMap<>();

/\* comparator used for sort the list index wise or height wise \*/

**public** **static** Comparator<HTreeNode> *heightComp* = **new** Comparator<HTreeNode>() {

@Override

**public** **int** compare(HTreeNode o1, HTreeNode o2) {

**return** o1.hIndex - o2.hIndex;

}

};

**public** **static** Comparator<HTreeNode> *indexComp* = **new** Comparator<HTreeNode>() {

@Override

**public** **int** compare(HTreeNode o1, HTreeNode o2) {

**return** o1.index - o2.index;

}

};

@Override

**public** HTreeNode createBstIndexTree(HTreeNode node, **int** dt, **int** index) {

**if** (node == **null**) {

node = **new** HTreeNode(dt);

node.index = index;

} **else** {

**if** (dt <= node.dt) {

node.lt = createBstIndexTree(node.lt, dt, --index);

node.lt.hIndex = Math.*abs*(node.hIndex) + 1;

} **else** {

node.rt = createBstIndexTree(node.rt, dt, ++index);

node.rt.hIndex = Math.*abs*(node.hIndex) + 1;

}

}

**return** node;

}

@Override

**public** **void** printInorder(HTreeNode node) {

**if** (node != **null**) {

printInorder(node.lt);

System.***out***.print(node.dt + "[" + node.index + "]->");

printInorder(node.rt);

}

}

@Override

**public** **void** printHashMap(Map<Integer, Integer> map) {

**for** (Map.Entry<Integer, Integer> entry : map.entrySet()) {

System.***out***.print(entry.getValue() + "->");

}

}

@Override

**public** **void** nodeOnIndex(Map<Integer, List<HTreeNode>> map, HTreeNode node) {

**if** (map == **null**)

**return**;

**if** (node != **null**) {

nodeOnIndex(map, node.lt);

**if** (map.containsKey(node.index)) {

List<HTreeNode> list = (List<HTreeNode>) map.get(node.index);

list.add(node);

map.put(node.index, list);

} **else** {

List<HTreeNode> list = **new** ArrayList<>();

list.add(node);

map.put(node.index, list);

}

nodeOnIndex(map, node.rt);

}

}

@Override

**public** **void** nodeOnLabel(Map<Integer, List<HTreeNode>> map, HTreeNode node) {

**if** (map == **null**)

**return**;

**if** (node != **null**) {

nodeOnLabel(map, node.lt);

**if** (map.containsKey(node.hIndex)) {

List<HTreeNode> list = (List<HTreeNode>) map.get(node.hIndex);

list.add(node);

map.put(node.hIndex, list);

} **else** {

List<HTreeNode> list = **new** ArrayList<>();

list.add(node);

map.put(node.hIndex, list);

}

nodeOnLabel(map, node.rt);

}

}

@Override

**public** **void** nodeOnLabelMinusIndex(Map<Integer, List<HTreeNode>> map, HTreeNode node) {

**if** (map == **null**)

**return**;

**if** (node != **null**) {

nodeOnLabelMinusIndex(map, node.lt);

**if** (map.containsKey(node.hIndex - node.index)) {

List<HTreeNode> list = (List<HTreeNode>) map.get(node.hIndex - node.index);

list.add(node);

map.put(node.hIndex - node.index, list);

} **else** {

List<HTreeNode> list = **new** ArrayList<>();

list.add(node);

map.put(node.hIndex - node.index, list);

}

nodeOnLabelMinusIndex(map, node.rt);

}

}

@Override

**public** **void** nodeOnLabelPlusIndex(Map<Integer, List<HTreeNode>> map, HTreeNode node) {

**if** (map == **null**)

**return**;

**if** (node != **null**) {

nodeOnLabelPlusIndex(map, node.lt);

**if** (map.containsKey(node.hIndex + node.index)) {

List<HTreeNode> list = (List<HTreeNode>) map.get(node.hIndex + node.index);

list.add(node);

map.put(node.hIndex + node.index, list);

} **else** {

List<HTreeNode> list = **new** ArrayList<>();

list.add(node);

map.put(node.hIndex + node.index, list);

}

nodeOnLabelPlusIndex(map, node.rt);

}

}

@Override

**public** Map<Integer, List<HTreeNode>> printColumWise(HTreeNode node) {

Map<Integer, List<HTreeNode>> mapColumnWiseprintMap = **new** TreeMap<Integer, List<HTreeNode>>();

nodeOnIndex(mapColumnWiseprintMap, node);

**return** mapColumnWiseprintMap;

}

@Override

**public** List<Integer> printBottomView(HTreeNode node) {

Map<Integer, List<HTreeNode>> botmViewMap = **new** TreeMap<Integer, List<HTreeNode>>();

nodeOnIndex(botmViewMap, node);

List<Integer> rsList = **new** ArrayList<>();

**for** (Map.Entry<Integer, List<HTreeNode>> mapEntry : botmViewMap.entrySet()) {

List<HTreeNode> list = mapEntry.getValue();

Collections.*sort*(list, *heightComp*);

rsList.add(list.get(list.size() - 1).dt);

}

**return** rsList;

}

@Override

**public** List<Integer> printTomView(HTreeNode node) {

Map<Integer, List<HTreeNode>> topViewMap = **new** TreeMap<Integer, List<HTreeNode>>();

nodeOnIndex(topViewMap, node);

List<Integer> rsList = **new** ArrayList<>();

**for** (Map.Entry<Integer, List<HTreeNode>> mapEntry : topViewMap.entrySet()) {

List<HTreeNode> list = mapEntry.getValue();

Collections.*sort*(list, *heightComp*);

rsList.add(list.get(0).dt);

}

**return** rsList;

}

@Override

**public** List<Integer> printLeftView(HTreeNode node) {

Map<Integer, List<HTreeNode>> leftViewMap = **new** TreeMap<Integer, List<HTreeNode>>();

nodeOnLabel(leftViewMap, node);

List<Integer> rsList = **new** ArrayList<>();

**for** (Map.Entry<Integer, List<HTreeNode>> mapEntry : leftViewMap.entrySet()) {

List<HTreeNode> list = mapEntry.getValue();

Collections.*sort*(list, *indexComp*);

rsList.add(list.get(0).dt);

}

**return** rsList;

}

**public** Map<Integer, List<HTreeNode>> rightViewMap = **null**;

@Override

**public** List<Integer> printRightView(HTreeNode node) {

Map<Integer, List<HTreeNode>> rightViewMap = **new** TreeMap<Integer, List<HTreeNode>>();

nodeOnLabel(rightViewMap, node);

List<Integer> rsList = **new** ArrayList<>();

**for** (Map.Entry<Integer, List<HTreeNode>> mapEntry : rightViewMap.entrySet()) {

List<HTreeNode> list = mapEntry.getValue();

Collections.*sort*(list, *indexComp*);

rsList.add(list.get(list.size() - 1).dt);

}

**return** rsList;

}

/\*\*

\* square corner view of tree top-left, bottom-left, bottom-right and top-right

\*\*/

@Override

**public** List<Integer> topLeftCornerView(HTreeNode node) {

Map<Integer, List<HTreeNode>> topLeftCornerViewMap = **new** TreeMap<Integer, List<HTreeNode>>();

nodeOnLabelMinusIndex(topLeftCornerViewMap, node);

List<Integer> rsList = **new** ArrayList<>();

**for** (Map.Entry<Integer, List<HTreeNode>> mapEntry : topLeftCornerViewMap.entrySet()) {

List<HTreeNode> list = mapEntry.getValue();

Collections.*sort*(list, *indexComp*);

rsList.add(list.get(0).dt);

}

**return** rsList;

}

@Override

**public** List<Integer> bottomRightCornerView(HTreeNode node) {

Map<Integer, List<HTreeNode>> topLeftCornerViewMap = **new** TreeMap<Integer, List<HTreeNode>>();

nodeOnLabelMinusIndex(topLeftCornerViewMap, node);

List<Integer> rsList = **new** ArrayList<>();

**for** (Map.Entry<Integer, List<HTreeNode>> mapEntry : topLeftCornerViewMap.entrySet()) {

List<HTreeNode> list = mapEntry.getValue();

Collections.*sort*(list, *indexComp*);

rsList.add(list.get(list.size() - 1).dt);

}

**return** rsList;

}

@Override

**public** List<Integer> bottomLeftCornerView(HTreeNode node) {

Map<Integer, List<HTreeNode>> bottomLeftCornerViewMap = **new** TreeMap<Integer, List<HTreeNode>>();

nodeOnLabelPlusIndex(bottomLeftCornerViewMap, node);

List<Integer> rsList = **new** ArrayList<>();

**for** (Map.Entry<Integer, List<HTreeNode>> mapEntry : bottomLeftCornerViewMap.entrySet()) {

List<HTreeNode> list = mapEntry.getValue();

Collections.*sort*(list, *heightComp*);

rsList.add(list.get(list.size() - 1).dt);

}

**return** rsList;

}

@Override

**public** List<Integer> topRightCornerView(HTreeNode node) {

Map<Integer, List<HTreeNode>> topRightCornerViewMap = **new** TreeMap<Integer, List<HTreeNode>>();

nodeOnLabelPlusIndex(topRightCornerViewMap, node);

List<Integer> rsList = **new** ArrayList<>();

**for** (Map.Entry<Integer, List<HTreeNode>> mapEntry : topRightCornerViewMap.entrySet()) {

List<HTreeNode> list = mapEntry.getValue();

Collections.*sort*(list, *heightComp*);

rsList.add(list.get(0).dt);

}

**return** rsList;

}

}

=====================================================================================

Test case:

-----------------------------------------------------------------

**public** **class** DAIHashingWithTreeTest {

HTreeNode root = **null**;

DAIHashingWithTree iht = **null**;

@Before

**public** **void** init() {

iht = **new** DAIHashingWithTreeImpl();

}

@Test

**public** **void** createBstIndexTreeTest() {

**int** arr[] = { 5, 3, 8, 2, 4, 6, 7, 0, 1, 10, 9 };

**for** (**int** i = 0; i < arr.length; i++) {

root = iht.createBstIndexTree(root, arr[i], 0);

}

System.***out***.println("\nInorder print start");

iht.printInorder(root);

System.***out***.println("\nInorder print end");

}

@Test

**public** **void** printColumWiseTest() {

**int** arr[] = { 3, 9, 8, 7, 6, 5, 4 };// { 5, 3, 8, 2, 4, 6, 7, 0, 1, 10, 9 };

**for** (**int** i = 0; i < arr.length; i++) {

root = iht.createBstIndexTree(root, arr[i], 0);

}

System.***out***.println("\nColumn wise print start");

Map<Integer, List<HTreeNode>> map = iht.printColumWise(root);

**for** (Map.Entry<Integer, List<HTreeNode>> entry : map.entrySet()) {

System.***out***.println();

List<HTreeNode> list = map.get(entry.getKey());

**for** (HTreeNode integer : list) {

System.***out***.print(integer.dt + "->");

}

}

System.***out***.println("\nColumn print end");

}

@Test

**public** **void** printBottomImageTest() {

**int** arr[] = { 5, 3, 8, 2, 4, 6, 7, 0, 1, 10, 9 };// { 3, 9, 8, 7, 6, 5, 4 }

**for** (**int** i = 0; i < arr.length; i++) {

root = iht.createBstIndexTree(root, arr[i], 0);

}

System.***out***.println("\nBottom wise print start");

List<Integer> listRs = iht.printBottomView(root);

**for** (Integer inte : listRs)

System.***out***.print(inte + "->");

System.***out***.println("\nBottom view print end");

}

@Test

**public** **void** printTomImageTest() {

**int** arr[] = { 5, 3, 8, 2, 4, 6, 7, 0, 1, 10, 9 };// { 3, 9, 8, 7, 6, 5, 4 }

**for** (**int** i = 0; i < arr.length; i++) {

root = iht.createBstIndexTree(root, arr[i], 0);

}

System.***out***.println("\nTop wise view print start");

List<Integer> listRs = iht.printTomView(root);

**for** (Integer inte : listRs)

System.***out***.print(inte + "->");

System.***out***.println("\nTop wise view print end");

}

@Test

**public** **void** printLeftViewTest() {

**int** arr[] = { 5, 3, 8, 2, 4, 6, 7, 0, 1, 10, 9 };// { 3, 9, 8, 7, 6, 5, 4 };

**for** (**int** i = 0; i < arr.length; i++) {

root = iht.createBstIndexTree(root, arr[i], 0);

}

System.***out***.println("\nleft wise view print start");

List<Integer> listRs = iht.printLeftView(root);

**for** (Integer inte : listRs)

System.***out***.print(inte + "->");

System.***out***.println("\nleft wise view print end");

}

@Test

**public** **void** printRightViewTest() {

**int** arr[] = { 5, 3, 8, 2, 4, 6, 7, 0, 1, 10, 9 };// { 3, 9, 8, 7, 6, 5, 4 };

**for** (**int** i = 0; i < arr.length; i++) {

root = iht.createBstIndexTree(root, arr[i], 0);

}

System.***out***.println("\n Riht wise view print start");

List<Integer> listRs = iht.printRightView(root);

**for** (Integer inte : listRs)

System.***out***.print(inte + "->");

System.***out***.println("\n Right wise view print end");

}

/\*\*

\* square corner view of tree top-left, bottom-left, bottom-right and top-right

\*\*/

@Test

**public** **void** topLeftCornerViewTest() {

**int** arr[] = { 5, 3, 8, 2, 4, 6, 7, 0, 1, 10, 9 };// { 3, 9, 8, 7, 6, 5, 4 };

**for** (**int** i = 0; i < arr.length; i++) {

root = iht.createBstIndexTree(root, arr[i], 0);

}

System.***out***.println("\n Top Left corner view print start");

List<Integer> listRs = iht.topLeftCornerView(root);

**for** (Integer inte : listRs)

System.***out***.print(inte + "->");

System.***out***.println("\n Top Left corner view print end");

}

@Test

**public** **void** bottomRightCornerViewTest() {

**int** arr[] = { 5, 3, 8, 2, 4, 6, 7, 0, 1, 10, 9 };// { 3, 9, 8, 7, 6, 5, 4 };

**for** (**int** i = 0; i < arr.length; i++) {

root = iht.createBstIndexTree(root, arr[i], 0);

}

System.***out***.println("\n Bottom Right Corner view print start");

List<Integer> listRs = iht.bottomRightCornerView(root);

**for** (Integer inte : listRs)

System.***out***.print(inte + "->");

System.***out***.println("\n Bottom Right Corner view print end");

}

@Test

**public** **void** bottomLeftCornerViewTest() {

**int** arr[] = { 5, 3, 8, 2, 4, 6, 7, 0, 1, 10, 9 };// { 3, 9, 8, 7, 6, 5, 4 };

**for** (**int** i = 0; i < arr.length; i++) {

root = iht.createBstIndexTree(root, arr[i], 0);

}

System.***out***.println("\n Bottom Left Corner view print start");

List<Integer> listRs = iht.bottomLeftCornerView(root);

**for** (Integer inte : listRs)

System.***out***.print(inte + "->");

System.***out***.println("\n Bottom Left Corner view print end");

}

@Test

**public** **void** topRightCornerViewTest() {

**int** arr[] = { 5, 3, 8, 2, 4, 6, 7, 0, 1, 10, 9 };// { 3, 9, 8, 7, 6, 5, 4 };

**for** (**int** i = 0; i < arr.length; i++) {

root = iht.createBstIndexTree(root, arr[i], 0);

}

System.***out***.println("\n Top Right Corner view print start");

List<Integer> listRs = iht.topRightCornerView(root);

**for** (Integer inte : listRs)

System.***out***.print(inte + "->");

System.***out***.println("\n Top Right Corner view print end");

}

}

}