1. 霍夫曼树（HashMap实现）
2. 霍夫曼树（Node实现）
3. 二叉排序树BST\_Sort
4. 二叉查找树AVL\_Tree
5. 霍夫曼树（哈希表实现）

import java.io.\*;  
import java.util.HashMap;  
import java.util.Map;  
  
class Node *{* Node left;  
 Node right;  
 Character character;  
  
 public Node*(*Character character*) {* this.character = character;  
 *}* public void setChar*(*Character character*) {* this.character = character;  
 *}* @Override  
 public String toString*() {* return "Node{" +  
 "character=" + character +  
 '}';  
 *}  
  
 // Using the preOrder to output the string* public void preOreder*() {* System.*out*.println*(*this*)*;  
 if *(*this.left != null*) {* this.left.preOreder*()*;  
 *}* if *(*this.right != null*) {* this.right.preOreder*()*;  
 *}  
 }* public void constructHuffmanTree*(*String codewords*) {  
 //Use HashMap to put the corresponding password and letter* Map*<*String, Character*>* list = new HashMap*<>()*;  
 StringBuilder stringBuilder = new StringBuilder*()*;  
  
 for *(*int i = 0; i < codewords.length*()*; i++*) {* if *(*codewords.charAt*(*i*)* == '1' || codewords.charAt*(*i*)* == '0'*) {* stringBuilder.append*(*codewords.charAt*(*i*))*;  
 *}* else *{ //Store in map to form corresponding key-value pairs* list.put*(*stringBuilder.toString*()*, codewords.charAt*(*i*))*;  
 *//clean up* stringBuilder = new StringBuilder*()*;  
 *}  
 }* for *(*Map.Entry*<*String, Character*>* entry : list.entrySet*()) {  
// To find the password* String code = entry.getKey*()*;  
 Node node = this;  
 for *(*int i = 0; i < code.length*()*; i++*) {* if *(*code.charAt*(*i*)* == '0'*) {  
 //If it is empty, then a new node is created* if *(*node.left == null*) {* Node left = new Node*(*null*)*;  
 node.left = left;  
 node = left;  
 *}* else *{* node = node.left;  
 *}  
 }* else *{* if *(*node.right == null*) {* Node right = new Node*(*null*)*;  
 node.right = right;  
 node = right;  
 *}* else *{* node = node.right;  
 *}  
 }  
 }* char c = entry.getValue*()*;  
 node.setChar*(*c*)*;  
 *}  
  
 }  
  
 /\*  
 Use the constructed binary tree to decode the cipher text and print the decoded text on screen.  
 NOTE: you must use the binary tree to decode the cipher text. The decoding method must be named:  
 “public String decode(String cipher)”  
 Provide comment for the method to explain how your decoding algorithm works.  
 \*/* public String decode*(*String cipher*) {* StringBuilder stringBuilder = new StringBuilder*()*;  
 StringBuilder result = new StringBuilder*()*;  
 for *(*int i = 0; i < cipher.length*()*; i++*) {* stringBuilder.append*(*cipher.charAt*(*i*))*;  
 Node node = this;  
  
 */\*  
 In theory, we don't need to check the first few password bits  
 because they must be empty, and just go to the last one  
 But then you still have to do things outside of the loop  
 and the code is still there, so I'm just going to go through each loop and look it up  
 \*/* for *(*int j = 0; j < stringBuilder.length*()*; j++*) {* if *(*stringBuilder.charAt*(*j*)* == '0'*) {* node = node.left;  
 if *(*node.character != null*) {* result.append*(*node.character*)*;  
 stringBuilder = new StringBuilder*()*;  
 *}  
 }* else *{* node = node.right;  
 if *(*node.character != null*) {* result.append*(*node.character*)*;  
 stringBuilder = new StringBuilder*()*;  
 *}  
 }  
 }  
 }* return result.toString*()*;  
 *}  
  
}  
// This code is contributed by Lance Cai 832002117*public class HuffmanDecoder *{* public static void main*(*String*[]* args*)* throws IOException *{* File f = new File*(*"/Users/lancecai/Desktop/Lab-2022S/CS211\_Algorithms&Data Structure 2/Lab1/Cipher.txt"*)*;  
 FileReader fre = new FileReader*(*f*)*;  
 BufferedReader bre = new BufferedReader*(*fre*)*;  
 String line1 = bre.readLine*()*;  
 String line2 = bre.readLine*()*;  
 Node node = new Node*(*null*)*;  
 node.constructHuffmanTree*(*line1*)*;  
 String codewordsDecoder = node.decode*(*line2*)*;  
 System.*out*.println*(*codewordsDecoder*)*;  
 *}  
}*

1. 霍夫曼树（Node实现）

import java.io.File;  
import java.io.FileNotFoundException;  
import java.util.Scanner;  
  
public class HuffmanDecoderDemo *{* public NodeDemo huffmanTree;  
  
 public HuffmanDecoderDemo*(){* huffmanTree = new NodeDemo*(*-1*)*;  
 *}* public class NodeDemo *{* int key;  
 char value;  
 NodeDemo left, right;  
  
 NodeDemo*(*int key*) {* this.key = key;  
 *}  
 }* public void constructHuffmanTreeDemo*(*String codewords*) {* NodeDemo currentRoot = huffmanTree;  
 for *(*int i = 0; i < codewords.length*()*; i++*) {* if *(*codewords.charAt*(*i*)* == '0'*) {* if *(*currentRoot.left == null*) {* currentRoot.left = new NodeDemo*(*0*)*;  
 *}* currentRoot = currentRoot.left;  
 *}* else if *(*codewords.charAt*(*i*)* == '1'*) {* if *(*currentRoot.right == null*) {* currentRoot.right = new NodeDemo*(*1*)*;  
 *}* currentRoot = currentRoot.right;  
 *}* if*(*i<*(*codewords.length*()*-1*)* && codewords.charAt*(*i+1*)* != '0' && codewords.charAt*(*i+1*)* != '1'*){* currentRoot.value = codewords.charAt*(*++i*)*;  
 currentRoot = huffmanTree;  
 *}  
 }  
 }* public String decode*(*String cipher*){* StringBuilder sb = new StringBuilder*()*;  
 for *(*int i = 0; i < cipher.length*()*; i++*) {* NodeDemo root = huffmanTree;  
 while *(*root.left != null && root.right != null && i<cipher.length*()){* if *(*cipher.charAt*(*i*)* == '1'*){* root = root.right;  
 *}* else *{* root = root.left;  
 *}* i++;  
 *}* if *(*root != null*){* sb.append*(*root.value*)*;  
 *}  
  
 }* return sb.toString*()*;  
 *}* public static void main*(*String*[]* args*) {*

HuffmanDecoderDemo hd = new HuffmanDecoderDemo*()*;  
 Scanner sc;  
 try *{* sc = new Scanner*(*new File*(*"/Users/lancecai/Desktop/Lab-2022S/CS211\_Algorithms&Data Structure 2/Lab1/Cipher.txt"*))*;  
 String codewords = sc.nextLine*()*;  
 String ciper = sc.nextLine*()*;  
  
 hd.constructHuffmanTreeDemo*(*codewords*)*;  
 System.*out*.println*(*hd.decode*(*ciper*))*;  
  
 *}* catch *(*FileNotFoundException e*) {* e.printStackTrace*()*;  
 *}  
 }  
  
}*

1. 二叉排序树BSTSorting

import java.io.BufferedReader;  
import java.io.File;  
import java.io.FileReader;  
import java.io.IOException;  
  
class BSTSorting *{  
  
 // Driver Code* public static void main*(*String*[]* args*)* throws IOException *{* BSTSorting tree = new BSTSorting*()*;  
  
 *// This code is contributed by Lance Cai 832002117* File f = new File*(*"/Users/lancecai/Desktop/Lab-2022S/CS211\_Algorithms&Data Structure 2/Lab1/RandomNumbers.txt"*)*;  
 FileReader fre = new FileReader*(*f*)*;  
 BufferedReader bre = new BufferedReader*(*fre*)*;  
 String str = "";  
 while *((*str = bre.readLine*())* != null*) {* System.*out*.println*(*str*)*;  
 String*[]* string = str.split*(*" "*)*;  
 int*[]* numsInt1 = new int*[*string.length*]*;  
 for *(*int i = 0; i < string.length; i++*) {* numsInt1*[*i*]* = Integer.*parseInt(*string*[*i*])*;  
 *//System.out.print(numsInt1[i]+" ");  
 }* for *(*int i = 0; i < string.length; i++*) {* tree.insert*(*numsInt1*[*i*])*;  
 *}  
 }* bre.close*()*;  
 fre.close*()*;  
  
 *// print inorder traversal of the BST by sortAscending() && sortDescending()* tree.sortAscending*()*;  
 *//tree.sortDescending();  
  
 // This code is contributed by Lance Cai  
  
 }  
  
  
 /\* Class containing left  
 and right child of current node  
 \* and key value\*/* class Node *{* int key;  
 Node left, right;  
  
 public Node*(*int item*) {* key = item;  
 left = right = null;  
 *}  
 }  
  
 // Root of BST* Node root;  
  
 *// Constructor* BSTSorting*() {* root = null;  
 *}* BSTSorting*(*int value*) {* root = new Node*(*value*)*;  
 *}  
  
 // This method mainly calls insertRec()* void insert*(*int key*) {* root = insertRec*(*root, key*)*;  
 *}  
  
 /\* A recursive function to  
 insert a new key in BST \*/* Node insertRec*(*Node root, int key*) {  
  
 /\* If the tree is empty,  
 return a new node \*/* if *(*root == null*) {* root = new Node*(*key*)*;  
 return root;  
 *}  
  
 /\* Otherwise, recur down the tree \*/* if *(*key < root.key*)* root.left = insertRec*(*root.left, key*)*;  
 else if *(*key > root.key*)* root.right = insertRec*(*root.right, key*)*;  
  
 */\* return the (unchanged) node pointer \*/* return root;  
 *}  
  
  
 // This method mainly calls InorderRec()* void sortAscending*() {* sortAscendingRec*(*root*)*;  
 *}  
  
 // A utility function to  
 // do inorder traversal of BST* void sortAscendingRec*(*Node root*) {* if *(*root != null*) {* sortAscendingRec*(*root.left*)*;  
 System.*out*.println*(*root.key*)*;  
 sortAscendingRec*(*root.right*)*;  
 *}  
 }* void sortDescending*() {* sortDescendingRec*(*root*)*;  
 *}  
  
 // A utility function to  
 // do inorder traversal of BST* void sortDescendingRec*(*Node root*) {* if *(*root != null*) {* sortDescendingRec*(*root.right*)*;  
 System.*out*.println*(*root.key*)*;  
 sortDescendingRec*(*root.left*)*;  
  
 *}  
 }  
  
}*

1. 平衡树AVLTree

import java.io.\*;  
  
public class SpecialAVLTree {  
 private static Node root;  
 static Node flag;  
  
 *//Create the Node* private static class Node {  
 private int key;  
 private String value;  
 Node leftChild, rightChild;  
 int height;  
 boolean isLazyRemove;  
  
 public Node(int key, String value) {  
 this.key = key;  
 this.value = value;  
 }  
  
 }  
  
 private static int height(Node node) {  
 return node == null ? -1 : node.height;  
 }  
  
 *//Rotation method* private static Node rotateWithLeftChild(Node k2) {  
 Node k1 = k2.leftChild;  
 k2.leftChild = k1.rightChild;  
 k1.rightChild = k2;  
 k2.height = Math.max(height(k2.leftChild), height(k2.rightChild)) + 1;  
 k1.height = Math.max(height(k1.leftChild), k2.height) + 1;  
  
 return k1;  
 }  
 *//Rotation method* private static Node rotateWithRightChild(Node k2) {  
 Node k1 = k2.rightChild;  
 k2.rightChild = k1.leftChild;  
 k1.leftChild = k2;  
 k2.height = Math.max(height(k2.leftChild), height(k2.rightChild)) + 1;  
 k1.height = Math.max(height(k1.rightChild), k2.height) + 1;  
  
 return k1;  
 }  
 *//doubleRotation method* private static Node doubleWithLeftChildRightSubtree(Node k3) {  
 k3.leftChild = rotateWithRightChild(k3.leftChild);  
 return rotateWithLeftChild(k3);  
 }  
 *//doubleRotation method* private static Node doubleWithRightChildLeftSubtree(Node k3) {  
 k3.rightChild = rotateWithLeftChild(k3.rightChild);  
 return rotateWithRightChild(k3);  
 }  
  
  
 */\*  
 If there are not nodes, directly return  
 If (Height of left subtree - height of right subtree) > 1, then right rotation  
 If (Height of right subtree - height of left subtree) > 1, then left rotation  
 Finally，return the height of the current node, itself also need to calculate, so（+1）.  
  
 This comment is written by Hanlin Cai.  
 \*/* private static Node balance(Node node) {  
 *//If there are not nodes* if (node == null)  
 return node;  
 *//If (Height of left subtree - height of right subtree) > 1, then right rotation* if (height(node.leftChild) - height(node.rightChild) > 1) {  
 if (height(node.leftChild.leftChild) >= height(node.leftChild.rightChild)) {  
 node = rotateWithLeftChild(node);  
 } else {  
 node = doubleWithLeftChildRightSubtree(node);  
 }  
 } else if (height(node.rightChild) - height(node.leftChild) > 1) {  
 *//If (Height of right subtree - height of left subtree) > 1, then left rotation* if (height(node.rightChild.rightChild) >= height(node.rightChild.leftChild)) {  
 node = rotateWithRightChild(node);  
 } else {  
 node = doubleWithRightChildLeftSubtree(node);  
 *}  
 }  
 //Finally，return the height of the current node, itself also need to calculate, so（+1）.* node.height = Math.*max(height(*node.leftChild*)*, *height(*node.rightChild*))* + 1;  
 return node;  
 *}  
  
 // insert a new node into the AVL tree* private static Node insert*(*int key, String value, Node currentRoot*) {* if *(*currentRoot == null*)* return new Node*(*key, value*)*;  
  
 if *(*key < currentRoot.key*) {  
 // the position is less than current position, then insert it into the left child* currentRoot.leftChild = *insert(*key, value, currentRoot.leftChild*)*;  
 *}* else if *(*key > currentRoot.key*) {  
 // the position is greater than current position, then insert it into the right child* currentRoot.rightChild = insert(key, value, currentRoot.rightChild);  
 }  
 *//Rebalance* return balance(currentRoot);  
 }  
  
 public static void insert(int key, String value) {  
 root = insert(key, value, root);  
 }  
  
 public void remove(int key) {  
 root = remove(key, root);  
 }  
  
 *//Remove Node* private Node remove(int toBeRemoved, Node currentRoot) {  
 if (currentRoot == null)  
 return currentRoot;  
  
 if (toBeRemoved < currentRoot.key) {  
 currentRoot.leftChild = remove(toBeRemoved, currentRoot.leftChild);  
 } else if (toBeRemoved > currentRoot.key) {  
 currentRoot.rightChild = remove(toBeRemoved, currentRoot.rightChild);  
 } else if (currentRoot.leftChild != null && currentRoot.rightChild != null) {  
 currentRoot.key = findMin(currentRoot.rightChild).key;  
 currentRoot.rightChild = remove(currentRoot.key, currentRoot.rightChild);  
 } else {  
 currentRoot = (currentRoot.leftChild != null) ? currentRoot.leftChild : currentRoot.rightChild;  
 }  
  
 return balance(currentRoot);  
 }  
  
 *// get the MIN value in AVLTree* private Node findMin(Node currentRoot) {  
 if (currentRoot == null)  
 return currentRoot;  
  
 while (currentRoot.leftChild != null)  
 currentRoot = currentRoot.leftChild;  
  
 return currentRoot;  
 }  
  
 *//This code is created by Hanlin Cai in 2022/04/08.  
 //sort the AVLTree in sortAscending().* void sortAscending() {  
 sortAscendingRec(root);  
 }  
  
 *// A utility function to  
 // do inorder traversal of AVLtree* void sortAscendingRec(Node root) {  
 if *(*root != null*) {* sortAscendingRec*(*root.leftChild*)*;  
 System.*out*.println*(*root.key + " " + root.value);  
 sortAscendingRec(root.rightChild);  
 }  
 }  
  
  
 static boolean search(int key) {  
 if (flag == null) {  
 return false;  
 }  
 if (flag.key == key) {  
 if (flag.isLazyRemove){  
 return false;  
 }  
 System.out.println(Integer.toString(flag.key) + " " + flag.value);  
 return true;  
 }  
 if (key < flag.key) {  
 flag = flag.leftChild;  
 } else {  
 flag = flag.rightChild;  
 }  
 return search(key);  
 }  
  
 static void lazyRemove(int key) {  
 if (flag == null) {  
 return;  
 }  
 if (flag.key == key) {  
 flag.isLazyRemove = true;  
 return;  
 }  
 if (key < flag.key) {  
 flag = flag.leftChild;  
 } else {  
 flag = flag.rightChild;  
 }  
 lazyRemove(key);  
 }  
  
 *//This code is created by Hanlin Cai in 2022/04/08.* public static void main(String[] args) throws IOException {  
  
 *//Use InputStreamReader to read the data.* SpecialAVLTree avlTree = new SpecialAVLTree();  
 InputStreamReader ISR = new InputStreamReader(new FileInputStream("/Users/lancecai/Desktop/Lab-2022S/CS211\_Algorithms&Data Structure 2/Lab2/Lab2\_PopularNames.csv"));  
 char[] chars = new char[100000];  
 ISR.read(chars);  
 String[] names = (new String(chars)).split("\n");  
 for (int i = 0; i < names.length; i++) {  
 int id = Integer.parseInt(names[i].split(",")[0]);  
 String name = names[i].split(",")[1];  
 insert(id, name);  
 }  
  
 System.out.println("--Task 1--");  
 System.out.println("I have written the comment in my code.");  
  
  
 System.out.println(" ");  
 System.out.println("--Task 2--");  
 *//Print all the records line-by-line in an ascending order* avlTree.sortAscending();  
  
  
 System.out.println(" ");  
 System.out.println("--Task 3--");  
*// search(1999);  
// search(2001);* flag = root;  
 if (!search(1492)){  
 System.out.println("No match records were found!");  
 }  
 flag = root;  
 if (!search(2001)) {  
 System.out.println("No match records were found!");  
 }  
  
  
 System.out.println(" ");  
 System.out.println("--Task 4--");  
 lazyRemove(1492);  
 if (!search(1492)) {  
 System.out.println("No match records were found! Because you have lazyRemove() it.");  
 }  
 *//avlTree.sortAscending();* System.out.println(" ");  
 System.out.println("--Task 5--");  
 flag = root;  
 insert(1492, "Kora");  
 search(1492);  
  
 flag = root;  
 insert(1492, "Kora1");  
 search(1492);  
 System.out.println("//Print the same key with two values on the screen.");  
 System.out.println("//Thank you! Have a nice day~ :D");  
  
 ISR.close();  
 }  
 *//This code is created by Hanlin Cai in 2022/04/08 16:32*}