# 实验报告

### 17042127 陶逸群

## 实验总结

这次实验我采用了Java语言实现了FP-tree建树算法，并应用所给数据构建了FP-tree。并且完成了所要挖掘的全部数据。

### 程序运行环境

Windows10+jdk1.8。

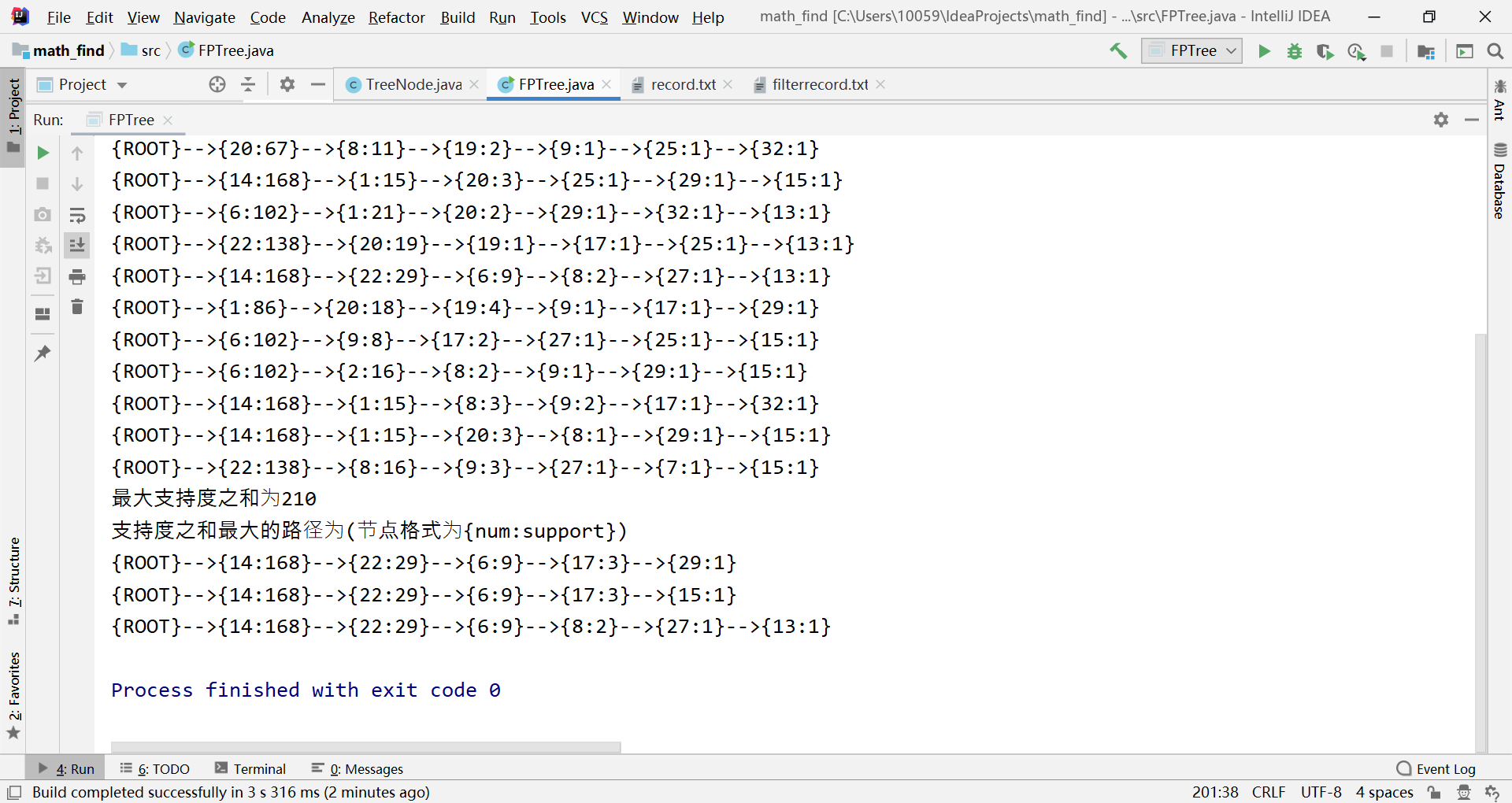
### 程序相关配置

* 首先将源数据去除表头（期号 红球1 红球2 红球3 红球4 红球5 红球6），重命名为record.txt，放入工程文件夹下。
* 随后运行程序，会提示输入最小支持度阈值，根据提示输入后，程序自动建立FP-tree，并且给出要求B、C、D的结果打印出来。
* 在建立FP-tree的过程中，程序会自动生成文件” filterrecord.txt”，表示过滤掉小于支持度阈值并且按照全局支持度排序后的数据。

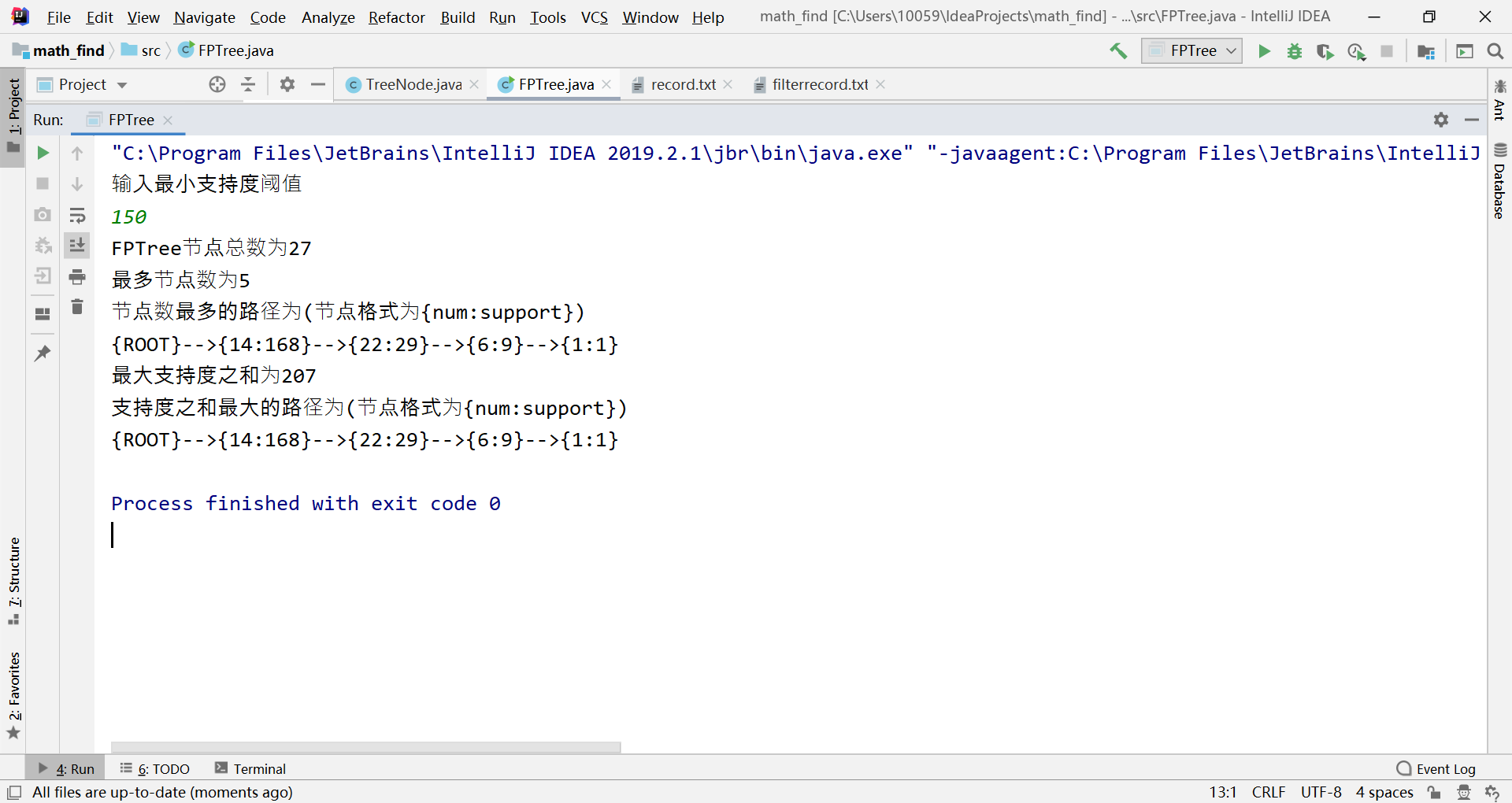
### 程序运行截图

* 最小支持度阈值为140时

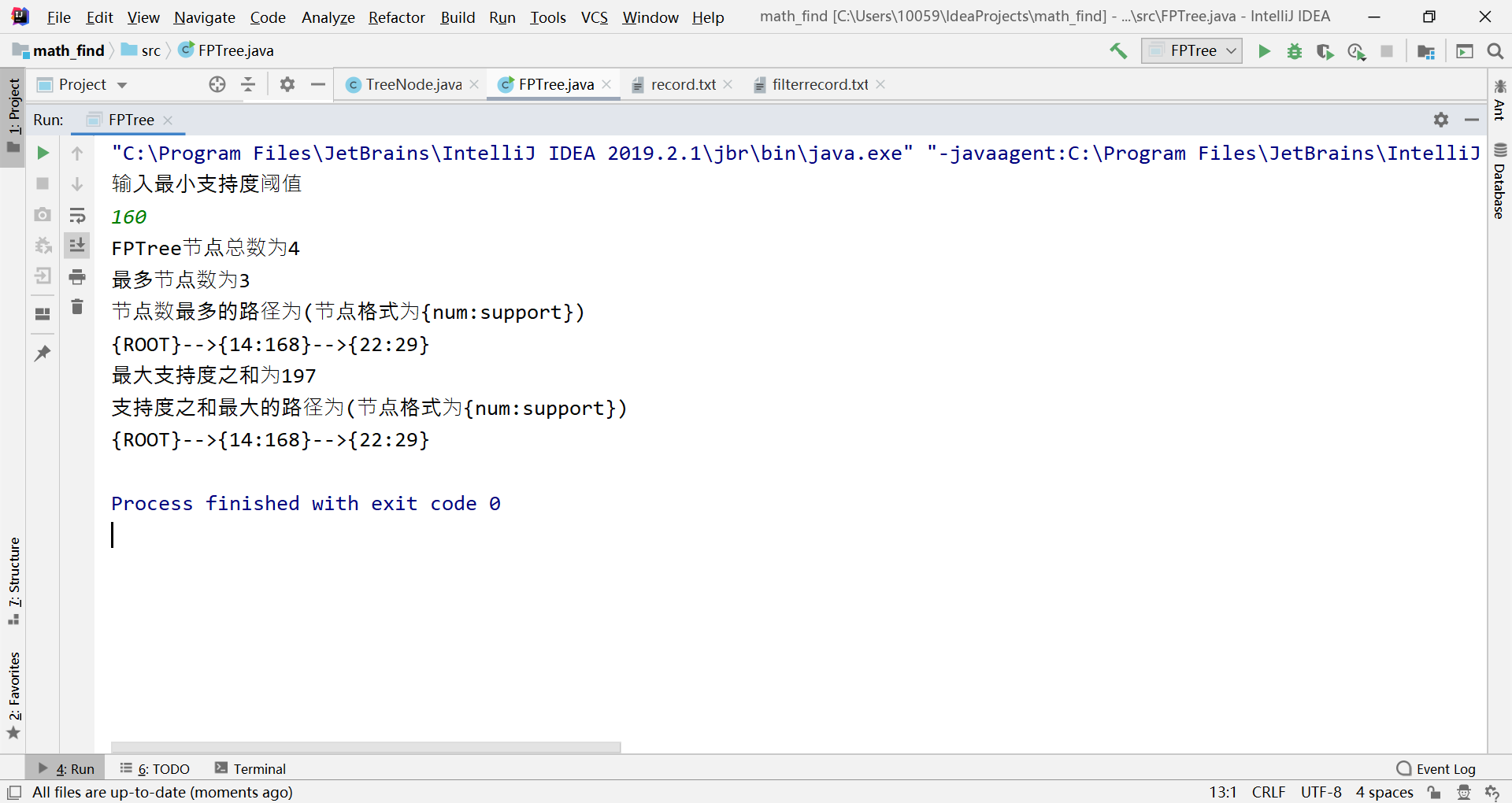




* 最小支持度值为150时



* 最小支持度为160时



### 代码

工程文件打包作为附件发送，下面仅贴代码。

树的节点类：

**import** **java.util.ArrayList**;

**import** **java.util.List**;

**class** **TreeNode** {

**private** int num;*//节点*

**private** int support;*//频数*

**private** int layer;*//层数*

**private** int sum;*//支持度之和*

**private** TreeNode parent;

**private** List<TreeNode> children;

**private** TreeNode nextHomonym;*//下一个节点（由表头项维护的那个链表）*

**private** TreeNode tail;*//末节点（由表头项维护的那个链表)*

**private** **static** **final** int ROOT = -99;

**public** TreeNode() {

}

**public** TreeNode(int ballNumber) {

**this**.num = ballNumber;

}

**@Override**

**public** String toString() {

**return** (num+"频数:"+support);

}

**public** int getNode() {

**return** **this**.num;

}

**public** void setNode(int ballNumber) {

**this**.num = ballNumber;

}

**public** int getSupport() {

**return** **this**.support;

}

**public** void setSupport(int support) {

**this**.support = support;

}

**public** TreeNode getParent() {

**return** **this**.parent;

}

**public** void setParent(TreeNode parent) {

**this**.parent = parent;

}

**public** List<TreeNode> getChildren() {

**return** **this**.children;

}

**public** void addChild(TreeNode child) {

**if** (getChildren() == **null**) {

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

list.add(child);

setChildren(list);

} **else** {

getChildren().add(child);

}

}

**public** TreeNode findChild(int ballNumber) {

List<TreeNode> children = getChildren();

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

**for** (TreeNode child : children) {

**if** (child.getNode()==ballNumber) {

**return** child;

}

}

}

**return** **null**;

}

**public** void setChildren(List<TreeNode> children) {

**this**.children = children;

}

**public** **static** String printParent(TreeNode treeNode) {

List<String> str = **new** ArrayList<>();

StringBuilder res = **new** StringBuilder();

**while** (treeNode.getNode()!=ROOT){

str.add("-->{"+treeNode.getNode()+":"+treeNode.getSupport()+"}");

treeNode = treeNode.getParent();

}

str.add("{ROOT}");

**for** (int i =str.size()-1;i>=0;i--){

res.append(str.get(i));

}

**return** res.toString();

}

**public** TreeNode getNextHomonym() {

**return** **this**.nextHomonym;

}

**public** void setNextHomonym(TreeNode nextHomonym) {

**this**.nextHomonym = nextHomonym;

}

**public** void SupportIncrement(int n) {

**this**.support += n;

}

**public** void SumIncrement(int n) {

**this**.sum += n;

}

**public** TreeNode getTail() {

**return** tail;

}

**public** void setTail(TreeNode tail) {

**this**.tail = tail;

}

**public** void setLayer(int layer) {

**this**.layer = layer;

}

**public** void setSum(int sum) {

**this**.sum = sum;

}

**public** int getLayer() {

**return** layer;

}

**public** int getSum() {

**return** sum;

}

}

FP-tree类以及主函数：

**import** **java.io.\***;

**import** **java.util.\***;

**public** **class** **FPTree** {

**private** int totalcount;*//节点的总数*

**private** TreeNode root;

**private** Map<Integer,Integer> frequency;*//支持度表*

**private** Map<Integer, TreeNode> headers;*//表头*

**private** int max\_layer;

**private** int max\_support;

**private** List<TreeNode> path\_max\_layer\_nodes;

**private** List<TreeNode> path\_max\_support\_nodes;

**private** **static** **final** int ROOT = -99;

**private** int minSuport;*//最小支持度。*

**public** int getTotalcount() {

**return** totalcount;

}

**public** TreeNode getRoot() {

**return** root;

}

**public** int getMax\_layer() {

**return** max\_layer;

}

**public** int getMax\_support() {

**return** max\_support;

}

**public** List<TreeNode> getPath\_max\_layer\_nodes() {

**return** path\_max\_layer\_nodes;

}

**public** List<TreeNode> getPath\_max\_support\_nodes() {

**return** path\_max\_support\_nodes;

}

**public** FPTree(int minSuport,String filename) **throws** IOException {

max\_layer = -1;

max\_support = -1;

path\_max\_support\_nodes = **new** ArrayList<>();

path\_max\_layer\_nodes = **new** ArrayList<>();

setMinSuport(minSuport);

buildFPTree(filename);

setSum(root);

}

**public** void setMinSuport(int minSuport) {

**this**.minSuport = minSuport;

}

**private** Map<Integer, Integer> getFrequency(String filename) **throws** IOException {

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

BufferedReader bufferedReader =**new** BufferedReader(**new** FileReader(filename));

String line = **null**;

**while** ((line = bufferedReader.readLine()) != **null**){

String[] str = line.split("\t");

**for** (int i = 1;i<7;i++){

Integer item = Integer.parseInt(str[i]);

Integer cnt = freq.get(item);

**if** (cnt == **null**) {

cnt = 0;

}

cnt++;

freq.put(item, cnt);

}

}

bufferedReader.close();

**return** freq;

}

**private** Map<Integer, TreeNode> getHeaders(Map<Integer,Integer> frequency,String filename) **throws** IOException {

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

BufferedWriter bufferedWriter =**new** BufferedWriter(**new** FileWriter("filterrecord.txt"));

BufferedReader bufferedReader =**new** BufferedReader(**new** FileReader(filename));

String line = **null**;

**while** ((line = bufferedReader.readLine()) != **null**){

String[] str = line.split("\t");

List<SupportNum> supportNums = **new** ArrayList<>();

**for** (int i = 1;i<7;i++){

int num = Integer.parseInt(str[i]);

int cnt = frequency.get(num);

**if**(minSuport <=cnt){

supportNums.add(**new** SupportNum(num,cnt));

}

}

Collections.sort(supportNums);

**for** (SupportNum supportnum:supportNums) {

bufferedWriter.write(supportnum.toString()+"\t");

}

bufferedWriter.write("\n");

}

bufferedWriter.flush();

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

int num = entry.getKey();

int cnt = entry.getValue();

**if** (cnt >= minSuport) {

TreeNode node = **new** TreeNode(num);

node.setSupport(cnt);

heads.put(num, node);

}

}

bufferedReader.close();

bufferedWriter.close();

**return** heads;

}

**private** void buildFPTree(String filename) **throws** IOException {

frequency = getFrequency(filename);

headers = getHeaders(frequency,filename);

root = **new** TreeNode(ROOT);

root.setSum(0);

root.setLayer(1);

root.setSupport(0);

totalcount = 1;

List<Integer> transaction = **null**;

BufferedReader bufferedReader =**new** BufferedReader(**new** FileReader("filterrecord.txt"));

String line = **null**;

**while** ((line = bufferedReader.readLine()) != **null**){

String[] str = line.split("\t");

int count = str.length;

**if** (str[0].equals("")){

count =0;

}

int i = 0;

LinkedList<Integer> record = **new** LinkedList<>();

**while** (i<count){

int num = Integer.parseInt(str[i]);

record.add(num);

i++;

}

TreeNode subTreeRoot = root;

TreeNode tmpRoot = **null**;

**if** (root.getChildren() != **null**) {

*//延已有的分支，令各节点support加1*

**while** (!record.isEmpty() *//record不为空*

&& (tmpRoot = subTreeRoot.findChild(record.peek())) != **null**) {*//寻找相同子节点*

tmpRoot.SupportIncrement(1);

subTreeRoot = tmpRoot;

record.poll();

}

}

*//长出新的节点*

addNodes(subTreeRoot, record, headers);

}

}

**private** void addNodes(TreeNode ancestor, LinkedList<Integer> record,

**final** Map<Integer, TreeNode> headers) {

**while** (!record.isEmpty()) {

Integer item = record.poll();

**if** (headers.containsKey(item)) {

TreeNode leafnode = **new** TreeNode(item);

leafnode.setSupport(1);

leafnode.setParent(ancestor);

ancestor.addChild(leafnode);

leafnode.setLayer(ancestor.getLayer()+1);

TreeNode header = headers.get(item);

TreeNode tail=header.getTail();

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

tail.setNextHomonym(leafnode);

}**else**{

header.setNextHomonym(leafnode);

}

header.setTail(leafnode);

**if**(leafnode.getLayer()>max\_layer){

path\_max\_layer\_nodes.clear();

path\_max\_layer\_nodes.add(leafnode);

max\_layer = leafnode.getLayer();

}**else** **if**(leafnode.getLayer()==max\_layer){

path\_max\_layer\_nodes.add(leafnode);

}

totalcount++;

addNodes(leafnode, record, headers);

}

}

}

**private** void setSum(TreeNode Parent){

**if**(Parent.getChildren()==**null**){

**if**(Parent.getSum()>max\_support){

path\_max\_support\_nodes.clear();

path\_max\_support\_nodes.add(Parent);

max\_support = Parent.getSum();

}**else** **if**(Parent.getSum()==max\_support){

path\_max\_support\_nodes.add(Parent);

}

**return**;

}

**for**(TreeNode Child:Parent.getChildren()){

Child.SumIncrement(Child.getSupport()+Parent.getSum());

setSum(Child);

}

}

**public** **static** void main(String[] args) **throws** IOException {

int my\_minSupport;

System.out.println("输入最小支持度阈值");

Scanner scanner = **new** Scanner(System.in);

my\_minSupport=scanner.nextInt();

FPTree fpTree = **new** FPTree(my\_minSupport,"record.txt");

System.out.println("FPTree节点总数为"+fpTree.getTotalcount());

System.out.println("最多节点数为"+fpTree.getMax\_layer());

System.out.println("节点数最多的路径为(节点格式为{num:support})");

**for**(TreeNode treeNode:fpTree.getPath\_max\_layer\_nodes()){

System.out.println(TreeNode.printParent(treeNode));

}

System.out.println("最大支持度之和为"+fpTree.getMax\_support());

System.out.println("支持度之和最大的路径为(节点格式为{num:support})");

**for**(TreeNode treeNode:fpTree.getPath\_max\_support\_nodes()){

System.out.println(TreeNode.printParent(treeNode));

}

}

}

**class** **SupportNum** **implements** Comparable<SupportNum>{

**private** int num;

**private** int support;

**public** SupportNum(int num,int support){

**this**.num = num;

**this**.support = support;

}

**public** int getNum() {

**return** num;

}

**public** int getSupport() {

**return** support;

}

**@Override**

**public** int compareTo(SupportNum o) {

**return** ((o.support)-(**this**.support));

}

**@Override**

**public** String toString() {

**return** **this**.num+"";

}

}