**实验四、从知晓到行动的模拟**

输入：给定m个人对n个项目按排序的投票。

输出：

1）确定其中是否隐含有孔多塞悖论（涉及到在有向图上尝试节点的拓扑排序）。

2）如果没有，就直接给出群体序，如果有，就按照一个特定的属性序，指出哪些投票是不满足单峰性质的，认为它们是“废票”，剔除后按照中位项定理给出群体排序。

算法流程如下：

* VoteSorter(int[][] votes)：构造函数，接受一个二维数组 votes 作为参数，该数组表示每个选民的投票结果。构造函数会初始化一些必要的变量，如 visited、onStack 和 ranking 等。
* public boolean hasCycle()：检查投票结果是否存在环路。如果存在环路，说明存在一个或多个候选人得票数相等，无法对它们进行排名。该方法使用 DFS 算法实现。
* private boolean dfs(int v)：DFS 算法的实现，用于检查从顶点 v 开始的路径是否存在环路。
* public int[] getRanking()：计算每个候选人的排名。如果投票结果存在环路，则该方法会先调用 removeInvalidVotes() 方法，删除无效投票，然后重新计算排名。该方法使用哈希表记录每个候选人在每个选民的投票中出现的次数，并选择出现次数最多的候选人作为该选民的投票结果。
* private void removeInvalidVotes()：删除无效投票。如果某个选民的投票结果不合法（即存在两个或更多个候选人得票数相等），则该投票结果为无效投票。该方法会将所有无效投票删除，并调整投票结果数组 votes 的大小。
* private boolean isInvalid(int[] vote)：检查某个选民的投票结果是否为无效投票。

代码实现(java):

import java.util.\*;  
  
public class Experiment5\_vote\_problem {  
 private int[][] votes;  
 private boolean[] visited;  
 private boolean[] onStack;  
 private int[] ranking;  
 private int invalidVotes;  
  
 public Experiment5\_vote\_problem(int[][] votes) {  
 this.votes = votes;  
 visited = new boolean[votes.length];  
 onStack = new boolean[votes.length];  
 ranking = new int[votes[0].length];  
 invalidVotes = 0;  
 }  
  
 public boolean hasCycle() {  
 for (int i = 0; i < votes.length; i++) {  
 if (!visited[i]) {  
 if (dfs(i)) {  
 return true;  
 }  
 }  
 }  
 return false;  
 }  
  
 private boolean dfs(int v) {  
 visited[v] = true;  
 onStack[v] = true;  
 for (int i = 0; i < votes[0].length; i++) {  
 int w = votes[v][i] - 1;  
 if (!visited[w]) {  
 if (dfs(w)) {  
 return true;  
 }  
 } else if (onStack[w]) {  
 return true;  
 }  
 }  
 onStack[v] = false;  
 return false;  
 }  
  
 public int[] getRanking() {  
 if (hasCycle()) {  
 removeInvalidVotes();  
 ranking = new int[votes[0].length];  
 }  
 for (int i = 0; i < votes[0].length; i++) {  
 Map<Integer, Integer> map = new HashMap<>();  
 for (int j = 0; j < votes.length; j++) {  
 int vote = votes[j][i];  
 if (!map.containsKey(vote)) {  
 map.put(vote, 0);  
 }  
 map.put(vote, map.get(vote) + 1);  
 }  
 int maxCount = 0;  
 int maxVote = 0;  
 for (int vote : map.keySet()) {  
 int count = map.get(vote);  
 if (count > maxCount) {  
 maxCount = count;  
 maxVote = vote;  
 }  
 }  
 ranking[i] = maxVote;  
 }  
 return ranking;  
 }  
  
 private void removeInvalidVotes() {  
 for (int i = 0; i < votes.length; i++) {  
 if (isInvalid(votes[i])) {  
 invalidVotes++;  
 votes[i] = null;  
 }  
 }  
 int[][] validVotes = new int[votes.length - invalidVotes][votes[0].length];  
 int index = 0;  
 for (int[] vote : votes) {  
 if (vote != null) {  
 validVotes[index] = vote;  
 index++;  
 }  
 }  
 votes = validVotes;  
 invalidVotes = 0;  
 }  
  
 private boolean isInvalid(int[] vote) {  
 int maxIndex = 0;  
 int maxVote = vote[0];  
 for (int i = 1; i < vote.length; i++) {  
 if (vote[i] > maxVote) {  
 maxIndex = i;  
 maxVote = vote[i];  
 }  
 }  
 for (int i = maxIndex + 1; i < vote.length; i++) {  
 if (vote[i] > maxVote) {  
 return true;  
 }  
 }  
 return false;  
 }  
  
 public static void main(String[] args) {  
 int[][] votes = {{1, 2, 3, 4, 5}, {3, 2, 1, 5, 4}, {2, 1, 3, 4, 5}, {1, 2, 3, 4, 5}, {1, 3, 2, 5, 4}, {1, 5, 4, 2, 3}, {2, 3, 4, 5, 1}, {3, 2, 1, 5, 4}, {5, 4, 3, 2, 1}, {4, 2, 3, 1, 5}};  
 Experiment5\_vote\_problem sorter = new Experiment5\_vote\_problem(votes);  
 int[] ranking = sorter.getRanking();  
 System.*out*.println(Arrays.*toString*(ranking));  
 }  
}

运行结果：

