CS 6100 Program 3: Social Choice (40 points)

Purpose

The reasons I'm asking you to do this lab are:
- I want you to play with social welfare functions so that you get familiar with the concept.
- I want you to see that the various voting mechanisms choose very different candidates (so that you do not mistakenly believe that all of them will select the same "most preferred" candidate).
- I want you to try to develop a social choice mechanism yourself.

The Problem

Arrow's impossibility theorem says that there is no social choice mechanism that takes individual preference patterns and generates a fair societal preference pattern. Arrow defined fairness according to axioms, and showed that all the axioms could not be simultaneously satisfied. Voting methods are attempts to take individual preference patterns and create a "fair" societal preference pattern. We should be able to identify situations where the voting mechanism breaks down. Since there is no way for voting to be fair, the task of somebody who is designing a voting mechanism is to minimize the unfairness.

Consider a society of seven voters (A-G) who are trying to reach a consensus on which of the alternatives they want (say, Hamiltons, Coppermill, Firehouse, Olive Garden, Indian Oven) for the title “CS Department Favorite Restaurant”. Each agent ranks them as 1 (meaning the best) and 5 (meaning the worst).

Each voting method will utilize a confidence in the vote. Each individual has a varying confidence in his/her vote. We will use confidence as follows: Since voter B has confidence 4 (conf 4) in its ranking, we will treat voter B’s first place vote for Hamiltons as if 4 different voters all ranked Hamiltons first. (This same concept exists when different parties have a different number of votes, like in the electoral college.)

To facilitate experimentation, let the following values be the initial values. Also, allow the user to request random (1) preferences and (2) confidences be used.

<table>
<thead>
<tr>
<th>Choice/Agent</th>
<th>A (conf 5)</th>
<th>B (conf 4)</th>
<th>C (conf 3)</th>
<th>D (conf 3)</th>
<th>E (conf 4)</th>
<th>F (conf 2)</th>
<th>G (conf 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hamiltons</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Coppermill</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>Firehouse</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Olive Garden</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Indian Oven</td>
<td>3</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>
Your job is to help these agents (as a group) choose one of the alternatives from the set of candidates.

The Experiment

I want you to **determine what choice is made** when the following social choice functions are applied.

1. Slater Ranking: find an ordering (no cycles) which has the fewest disagreements with the majority graph. In output, label choices in Slater Ranking as 1, 2, …5.
2. Kemeny ranking; create an overall ranking of the candidates that has as few **disagreements** as possible (where a disagreement is with a vote on a pair of candidates). In output, label choices in Kemeny Ranking as 1, 2, …5.
3. Bucklin ranking: start with k=1 and increase k gradually until some candidate is among the top k candidates in more than half the votes; that candidate wins. Identify the winner and the k required.
4. Cumulative voting: Each voter is given k votes which can be cast arbitrarily (voting for any set of candidates he wants). The candidate with the most votes is selected. For our case, assume k=6 and the voter gives 3 to first choice, 2 to second choice, and 1 to third choice.
5. Identify the Smith Set: the smallest nonempty set such that every member of the set pairwise defeats every member outside the set. In output, label choices in Smith Set with an asterisk.
6. Single Transferable Vote (STV, aka. Instant Runoff): candidate with lowest plurality score drops out; if you voted for that candidate (as your first choice), your vote transfers to the next (live) candidate on your list; repeat until one candidate remains. In the output, indicate the order they drop out with (5) being used to label the first to drop out.
7. Try a voting scheme you make up (or one that you find in the literature). **Explain your method** (in the readme file). Indicate the winner (and any other important results) using this method.

The Output:

Allow the user to see the results for individual voting methods and easily compare the results. I suggest you arrange the results something like the following (with winner highlighted and other information put in the corresponding boxes)

<table>
<thead>
<tr>
<th>Slater</th>
<th>Kemeny</th>
<th>Bucklin</th>
<th>Cumulative Voting</th>
<th>Smith Set</th>
<th>Single Transferable Vote</th>
<th>My Own</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hamiltons</td>
<td>Hamiltons</td>
<td>Hamiltons</td>
<td>Hamiltons</td>
<td>Hamiltons</td>
<td>Hamiltons</td>
<td>Hamiltons</td>
</tr>
<tr>
<td>Coppermill</td>
<td>Coppermill</td>
<td>Coppermill</td>
<td>Coppermill</td>
<td>Coppermill</td>
<td>Coppermill</td>
<td>Coppermill</td>
</tr>
<tr>
<td>Firehouse</td>
<td>Firehouse</td>
<td>Firehouse</td>
<td>Firehouse</td>
<td>Firehouse</td>
<td>Firehouse</td>
<td>Firehouse</td>
</tr>
</tbody>
</table>
The actual winners are not as important as how you analyze your results. Submit your code and a README report file. Use the scientific method (observe a phenomena, generate a hypothesis, test your hypothesis, and present supporting data). You should submit a report that summarizes the social choice functions you tested and an analysis of the results (what was good about each scheme, what was bad, why were the schemes good/bad). Feel free to experiment with different preference data.

**Hint:**
This is a trickier than it first appears. I would recommend thinking a bit about the design before you start coding. While you must do your own work, I would recommend comparing results with others.

Use all your good coding skills: meaningful variable names, print outs of current data, easy subscripting. For example, I had something called ctRank[i][j] which was the number of times product i was ranked j. Since j goes from 1 to 5, instead of 0-4, I just created the array to allow an extra 0th column that wasn’t used. That was simpler for me to think about than to always have to subtract 1 from the rank. I had something called stillIn[i] which told me whether or not item i was still in the running. That way I could ignore an item without having to rebuild the tables as much.

So my function to remove the worst from consideration and rebuild the ranks looked like:

```c
void remove(int worst, int rank[PROD][VOTER], bool stillIn[PROD])
{
    stillIn[worst] = false;  //"remove " candidateName[worst]
    for (int i=0; i < prodCt; i++)
    {
        if (stillIn[i])
            for (int j=0; j < voterCt; j++)
                if (rank[i][j] > rank[worst][j])
                    rank[i][j]--;
    }
    return rank;
}
```