MODULE *CandidateRanking*

This module formally specifies two systems of ranking candidates in an election–the *Borda* ranking and the *Condorcet* ranking. The specifications can be executed by *TLC* to compute the results for very small elections (on the order of 10 voters and 10 candidates). The module begins by importing operators defined in some standard modules.

EXTENDS *Integers*, *Sequences*, *FiniteSets*

The *Borda* system assigns each candidate a score and ranks them by score. This requires sorting the candidates by score, so we must define a sorting operator. For simplicity, we assume the items to be sorted are records with a *key* component, and they are to be sorted by that component. We sort a set of such records, defining a sorting of them to be a sequence of the set's elements in non-decreasing order of key. Since different elements can have the same key value, there is no unique way to sort them. We first define *Sortings*(S) to be the set of all sortings of the set S of records.

$$
\begin{array}{l}
Sortings(S) \triangleq \\
\quad \text{LET } D \triangleq 1 \,..\, Cardinality(S) \\
\quad \text{IN } \; \{seq \in [D \to S] : \\
\qquad\qquad \land\; S \subseteq \{seq[i] : i \in D\} \\
\qquad\qquad \land\; \forall\, i, j \in D : (i < j) \Rightarrow (seq[i].key \leq seq[j].key)\}
\end{array}
$$

We now define *SortSet*(S) to be some element of *Sortings*(S). We can define it quite simply to choose an arbitrary element of *Sortings*(S), but *TLC* could execute that definition in a reasonable time only for very tiny sets. Here's a definition that *TLC* can execute efficiently enough.

$$
\begin{array}{l}
\text{RECURSIVE } SortSet(\_) \\
SortSet(S) \triangleq \\
\quad \text{IF } S = \{\} \text{ THEN } \langle\rangle \\
\qquad\qquad\qquad \text{ELSE LET } s \triangleq \text{CHOOSE } ss \in S : \forall\, t \in S : ss.key \leq t.key \\
\qquad\qquad\qquad\qquad\quad \text{IN } \; \langle s \rangle \circ SortSet(S \setminus \{s\})
\end{array}
$$

We now declare the constant parameter *Votes*, which is the input. It is assumed to be a sequence of rankings, where each ranking is a sequence of candidate names in preference order. We assume that each voter ranks all the candidates.

CONSTANT *Votes*

The following defines *Cand* to be the set of all candidates in the first voter's ranking, which we assume is the set of all candidates.

$$Cand \triangleq \{Votes[1][i] : i \in 1 \,..\, Len(Votes[1])\}$$

The following asserts what we assume about *Votes*: that is it a sequence of rankings, each of which is a sequence of candidates, and that each candidate appears in each ranking exactly once.

$$
\begin{array}{l}
\text{ASSUME } \land\; Votes \in Seq(Seq(Cand)) \\
\qquad\qquad\; \land\; \forall\, j \in 1 \,..\, Len(Votes) : \\
\qquad\qquad\qquad \land\; Len(Votes[j]) = Cardinality(Cand) \\
\qquad\qquad\qquad \land\; \{Votes[j][i] : i \in 1 \,..\, Len(Votes[j])\} = Cand
\end{array}
$$

We now define the *Borda* ranking. Each ranking assigns a value to the candidates, where an i-th place ranking has a value of $N - i$. A candidate's score is the sum of the values assigned to it by the rankings. The *Borda* system ranks candidates according to their scores. We define an operator *Borda* whose value is a sequence of records, each having a candidate's name and score, sorted by score.

We start by defining $SumOfSeq(s)$ so that, if s is a sequence of numbers, then $SumOfSeq(s)$ is the sum of those numbers.

$$\textsc{recursive}\ SumSeq(\_)$$
$$SumSeq(s) \triangleq \textsc{if}\ s = \langle\rangle\ \textsc{then}\ 0 \ \textsc{else}\ Head(s) + SumSeq(Tail(s))$$

For convenience, we define N to be the number of candidates and V the number of voters.

$$N \triangleq Cardinality(Cand)$$
$$V \triangleq Len(Votes)$$

$RankBy(c, i)$ is the rank (a number from 1 to N) that voter number i assigns to candidate c.

$$RankBy(c, i) \triangleq \textsc{choose}\ r \in 1 \,..\, N : Votes[i][r] = c$$

$Score(c)$ is the score of candidate c.

$$Score(c) \triangleq SumSeq([i \in 1 \,..\, V \mapsto N - RankBy(c, i)])$$

To define *Borda*, we first define *ReverseBorda* to be a sequence containing the candidates' records in increasing order of their scores, and then define *Borda* to be the sequence obtained by reversing the sequence *ReverseBorda*.

$$ReverseBorda \triangleq SortSet(\{[name \mapsto c, key \mapsto Score(c)] : c \in Cand\})$$

$$Borda \triangleq [i \in 1 \,..\, N \mapsto ReverseBorda[N - i + 1]]$$

We now define the *Condorcet* ranking. We first define $\succ$ so that $c \succ d$ is true iff more voters prefer c to d (rank c before d) than prefer d to c.

$$c \succ d \triangleq$$
$$\quad \textsc{let}\ NumberPreferring(a, b) \triangleq$$

The number of voters who prefer candidate a to candidate b.

$$\qquad Cardinality(\{v \in 1 \,..\, V : RankBy(a, v) < RankBy(b, v)\})$$
$$\quad \textsc{in}\ NumberPreferring(c, d) > NumberPreferring(d, c)$$

We now define the *Condorcet* ranking to be the sequence

$\langle C\_1, \ldots, C\_m \rangle$

of disjoint sets of candidates such that

- $C\_1 \cup \ \ldots\ \cup C\_m$ is the set of all candidates.

- For each i and j in $1 \,..\, m$, if $i > j$ then $c \succ d$ for each c in C_i and d in C_j.

- The sets C_i are as small as possible.

$$CondorcetRanking \triangleq$$
$$\quad \textsc{let}\ IsDominatingSet(D, C) \triangleq$$

If D and C are sets of candidates, then this is true iff $d \succ e$ is true for all $d \in D$ and all e in C but not in D.

$$\begin{aligned}
&\land\; D \neq \{\} \\
&\land\; \forall\, d \in D : \forall\, e \in C \setminus D : d \succ e
\end{aligned}$$

$$CWinners(C) \;\triangleq$$

The set of *Condorcet* winners in the election for the set C of candidates, meaning that it is the smallest nonempty subset D of C such that $IsDominateSet(D,\, C)$ is true.

$$\begin{aligned}
&\textsc{choose}\; D \in \textsc{subset}\; C : \\
&\quad \land\; IsDominatingSet(D,\, C) \\
&\quad \land\; \forall\, E \in \textsc{subset}\; C : IsDominatingSet(E,\, C) \Rightarrow (D \subseteq E)
\end{aligned}$$

We now inductively define $CRanking(C)$ for sets C of candidates such that the *Condorcet* ranking is $CRanking(Cand)$.

$$\begin{aligned}
&\textsc{recursive}\; CRanking(_) \\
&CRanking(C) \;\triangleq\; \textsc{if}\; C = \{\}\; \textsc{then}\; \langle\rangle \\
&\qquad\qquad\qquad\qquad\qquad \textsc{else}\; \textsc{let}\; CW \;\triangleq\; CWinners(C) \\
&\qquad\qquad\qquad\qquad\qquad\qquad\quad \textsc{in}\;\;\; \langle CW \rangle \circ CRanking(C \setminus CW)
\end{aligned}$$

$$\textsc{in}\quad CRanking(Cand)$$

We now write another definition of the *Condorcet* ranking that *TLC* can compute more efficiently. To do that, we first define the transitive closure of a relation, where a relation R is a set of ordered pairs. We informally write $c\; R\; d$ to mean $\langle c,\, d\rangle \in R$. We say that R is a relation *on* a set S if R is a subset of $S \times S$.

We think of a relation R as a directed graph, where there is an edge from c to d iff $c\; R\; d$ holds. We then define $NodesOf(R)$ to be the set of nodes of this graph. (The set $NodesOf(R)$ is the smallest set S such that R is a relation on S.)

$$NodesOf(R) \;\triangleq\; \{r[1] : r \in R\} \cup \{r[2] : r \in R\}$$

The transitive closure of a relation R is the relation R^+ such that $c\; R^+\; d$ holds iff there is a path from c to d in the graph of R. It is not hard to see that if there is a path from c to d in R, then there is a path from c to d in R whose length (number of nodes) is at most one greater than the number of nodes in R. We now define $PathsOfLen(R,\, j)$ to be the set of all paths in R of length exactly j.

$$\begin{aligned}
PathsOfLen(R,\, j) \;\triangleq\; &\{p \in [1 \,..\, j \rightarrow NodesOf(R)] : \\
&\quad \forall\, i \in 1 \,..\, (j-1) : \langle p[i],\, p[i+1]\rangle \in R\}
\end{aligned}$$

We define $ShortPaths(R)$ to be the set of paths in R of length between 2 and 1 plus the number of nodes in R.

$$ShortPaths(R) \;\triangleq\; \textsc{union}\; \{PathsOfLen(R,\, j) : j \in 2 \,..\, (Cardinality(NodesOf(R)) + 1)\}$$

Finally, we define $TC(R)$, the transitive closure of R, to be the set of all pairs $\langle c,\, d\rangle$ of nodes of R that are joined by a path in $ShortPaths(R)$.

$$TC(R) \;\triangleq\; \{\langle p[1],\, p[Len(p)]\rangle : p \in ShortPaths(R)\}$$

This definition of the transitive closure is mathematically elegant, but it can't be computed efficiently by *TLC*. (The time it takes *TLC* to compute $TC(R)$ is exponential in the number of nodes of R.) We therefore write an equivalent definition that *TLC* can compute faster. To do this, we first define $R ** S$ to be the composition of relations R and S, which is the set of all pairs $\langle r, s\rangle$ such that $r\ R\ t$ and $t\ R\ s$ hold for some t.

$$
\begin{aligned}
R ** S \triangleq\ & \text{LET } T \triangleq \{rs \in R \times S : rs[1][2] = rs[2][1]\} \\
& \text{IN } \quad \{\langle x[1][1], x[2][2]\rangle : x \in T\}
\end{aligned}
$$

It is not hard to show that the transitive closure of a relation R equals

$R \cup R ** R \cup \ \ldots\ \cup R ** \ \ldots\ ** R$

where the number of sets in the union is the number of nodes in R. The following alternative definition of the transitive closure is based on this observation.

$$
\begin{aligned}
& SimpleTC(R) \triangleq \\
& \quad \text{LET RECURSIVE } STC(\_) \\
& \quad\quad\quad STC(n) \triangleq \text{IF } n = 1 \text{ THEN } R \\
& \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \text{ELSE } STC(n-1) \cup STC(n-1) ** R \\
& \quad \text{IN} \quad \text{IF} \quad R = \{\} \quad \text{THEN } \{\} \quad \text{ELSE } STC(Cardinality(NodesOf(R)))
\end{aligned}
$$

TLC has checked that $SimpleTC(R)$ equals $TC(R)$ for all relations R on a set of 4 elements.

We now write a definition of the *Condorcet* ranking that *TLC* can compute more efficiently than the definition *CondorcetRanking* above. It is based on the following observation. Let $\succeq$ be the relation on the set of candidates such that $c \succeq d$ holds iff $d \succ c$ does *not* hold. (It's not hard to see that $c \succeq d$ means that at least as many voters prefer c to d as prefer d to c.) Let $\succeq^+$ be the transitive closure of $\succeq$. Then the *Condorcet* ranking is the unique sequence

$\langle C\_1, \ldots, C\_m\rangle$

of sets of candidates such that

– $C\_1 \cup \ \ldots\ \cup C\_m$ equals the set of all candidates.

- For all i and j in $1 \,..\, m$, if $i \geq j$ then $c \succeq^+ d$ for all $c \in C\_i$ and $d \in C\_j$.

In the following definition, *DomEq* is the relation $\succeq$ and *DomEqPlus* is its transitive closure $\succeq^+$.

$$
\begin{aligned}
& CRanking \triangleq \\
& \quad \text{LET } DomEq \triangleq \{r \in Cand \times Cand : \neg(r[2] \succ r[1])\} \\
& \quad\quad\quad DomEqPlus \triangleq SimpleTC(DomEq) \\
& \quad\quad\quad CWinners(C) \triangleq \{c \in C : \forall d \in C : \langle c, d\rangle \in DomEqPlus\} \\
& \quad\quad\quad \text{RECURSIVE } CRanking(\_) \\
& \quad\quad\quad CRanking(C) \triangleq \text{IF } C = \{\} \text{ THEN } \langle\rangle \\
& \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \text{ELSE LET } CW \triangleq CWinners(C) \\
& \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \text{IN } \quad \langle CW\rangle \circ CRanking(C \setminus CW) \\
& \quad \text{IN} \quad CRanking(Cand)
\end{aligned}
$$

TLC has checked the equivalence of *CondorcetRanking* and *CRanking* on all possible values of *Votes* for a set of 4 candidates and 3 or 4 voters, as well as on a number of randomly chosen values of *Votes* with more candidates and voters.