FOREST-SIZE GAMES

by

Darren Lim
Moravian College
Bethlehem, PA 18018-6650

June-August 1992

The research was made at a Research Experience for Undergraduates program held at Drew University during the noted time. The author would like to acknowledge Carol Brennan, Mary Dana Morrill, and Dr. David Housman, who have all contributed to this report.

I. Introduction and Notation

A Cooperative Game is an ordered pair (N, w), where $N = \{1, 2, \ldots, n\}$ represents players in a game, and w is a real-valued function, called the worth function, which maps subsets of N onto real numbers, with the condition that $w(\emptyset) = 0$. A subset of N, denoted S, is called a coalition; when $S = N$, we refer to it as the grand coalition. We will be looking at a cooperative game with the property of superadditivity; a game is superadditive if $\forall\ S,T \subseteq N : S \bigcap T = \emptyset,\ \ w(S \bigcup T) \geq w(S) + w(T)$.

An allocation for a cooperative game is a vector $x = (x_1, x_2, \ldots, x_n)$, where x_i represents an individual allocation, or payoff, to player i. An allocation method is a function which maps cooperative games to an allocation. The allocation method which will be used in this study will have the following four properties:

(1) Efficiency: $\sum_{i=1}^{n} x_i = w(N)$.

(2) Equal Treatment: If $w(S \bigcup \{i\}) = w(S \bigcup \{j\})$ for all $S \subseteq N - \{i, j\}$, then $x_i(N, w) = x_j(N, w)$.

(3) No Free Lunch: Let $i \in N$. If $w(S \bigcup \{i\}) = w(S) + w(i)$ for all S s.t. $i \notin S$, then $x_i(N, w) = w(i)$.

(4) Additivity: $x_i(N, v+u) = x_i(N, v) + x_i(N, u)$ for all games (N, v) and (N, u).

There is a unique allocation method which satisfies the preceding four properties; it is called the Shapley value. A formula for finding the Shapley value of a cooperative game is given:

$$\phi_i(N, w) = \sum_{S \subseteq N} \frac{(s-1)!\,(n-s)!}{n!}\ [w(S)\ -\ w(S - \{i\})], \text{ where } s = |S|.$$

This project will look at the Shapley value to a specially-defined game called a forest-size game. A forest-size game is represented as a graph, $G=(V, E)$ with n vertices. $V(G)$ represents the set of vertices in the graph G, and $E(G)$ represents the set of symmetric pairs of an irreflexive, symmetric relation on V. The following terms will be used throughout the report:

(1) Adjacency: Two vertices u, v are set to be adjacent, if the edge uv $\in E(G)$. Similarly, if two edges, uv and uw are distinct edges of a graph G, then uv and uw are adjacent.

(2) Incidence: If an edge uv $\in E(G)$, then vertices u and v are said to be incident with edge uv.

(3) Degree: The degree of a vertex v is the number of edges incident with v.

(4) Isolated Vertex: An isolated vertex is a vertex with degree zero.

(5) Order: The order of a graph, denoted p, is the number of vertices in the graph.

(6) Size: The size of a graph, denoted q, is the number of edges in the graph.

(7) Path: A u-v path, where u and v are vertices of a graph G, is an alternating sequence of vertices and edges beginning with u and ending with v, such that every edge joins its preceding and succeeding vertex, and that no vertex or edge is repeated in the sequence.

(8) Subdivision: A subdivision of a graph G is a graph obtained by inserting vertices of degree two into edges of G.

(9) Connectiveness: A graph is said to be connected if for every two vertices u,v in G s.t. $u \neq v$, there exists a u-v path in G.

(10) Subgraph: A subgraph of a graph G is a graph H s.t. $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$.

(11) Induced Subgraph: An induced subgraph of a graph G on a subset of vertices S is a subgraph $G'=(S, E')$, where E' is the set of all edges of G which are incident with two vertices in S.

(12) Component: A component of a graph G is a connected subgraph H s.t. no connected subgraph of G with either more edges or vertices than H contains the subgraph H.

(13) Cycle: A cycle is a u-v path, where $u = v$ and the path contains at least three edges.

(14) Forest: A forest is a graph with no cycles.

(15) Spanning forest: A spanning forest of a graph G is a forest which contains every vertex in G.

The players of the game itself are represented by vertices, and the worth function is defined as follows: $w(S) = |S| - k$, where k is the number of components in the induced subgraph on the vertices in the coalition S.

II. Proof Techniques

This section deals with two methods of proving many theorems about Forest-Size games. Using the properties of the Shapley value allocation method and the Shapley value formula was not sufficient for many of the more complicated graphs which have been examined during this study. The first method, Forest-Size Decomposition, involves the creation of a number of subgraphs of the original graph. The second method, The Difference Game, involves removing or adding an edge to the original graph. The basic idea behind both of these methods is to look at graph games, determined by a starting graph, whose Shapley value is easily calculated.

A. Forest-Size Decomposition

The original graph game, $W(G = (V, E))$, is represented by a number of subgraph games. These games are constructed so that the Shapley value can be easily determined for one or more players in the subgraph game. Finally, the Shapley value of a player in the original graph game can be determined by summing up the Shapley values found in the subgraph games for that particular player.

The initial step in Forest-Size Decomposition is the creation of the subgraph games. A subgraph game will consist of a subgraph of the original game, The order of the subgraph will be the same as the order of the original graph. However, the edges in the subgraph are determined by a chosen set of vertices, called X. The method of choosing X for each subgraph will depend on the theorem which is being proved. After X is chosen, the edges in the induced subgraph on X are added to the subgraph; no additional edges are added. Once a subgraph is constructed, the process may continue using a different subset X, creating another subgraph.

The object of this procedure is to create a collection of graph games which will produce the same worth function as the original graph game, when the worth functions of the subgraph games are summed together. Therefore, every edge in the original graph will be present in at least one of the subgraphs; it is possible that a edge can be present more than once. When computing the Shapley values for each subgraph game, a negative multiplying factor will be necessary to account for edges represented in more than one subgraph. Since the Shapley value allocation method is additive, finding the Shapley value of a player represented in the original graph is done by adding up the Shapley values of that same player in all of the subgraph games.

B. The Difference Game

This technique involves three different graph games named Edge-in, Edge-out, and the Difference game $DG(N, u)$. The original game is either the Edge-in or Edge-out game, and an edge is removed from the graph or an edge is added to the graph to create the other game. The Difference game's worth function is equal to the difference of Edge-in's and Edge-out's worth functions.

For example, look at the following two games.

Game 1	Game 2
$w(1) = w(2) = w(3) = 0$	$w(1) = w(2) = w(3) = w(12) = 0$
$w(12) = w(23) = w(13) = 1$	$w(23) = w(13) = 1$
$w(123) = 2$	$w(123) = 2$

Game 1 would represent the Edge-in game, and Game 2 would represent the Edge-out game, where the edge removed from Game 1 was the edge 1-2. The difference game would be the following:

$u(1) = 0 - 0 = 0$

$u(2) = 0 - 0 = 0$

$u(3) = 0 - 0 = 0$

$u(12) = 1 - 0 = 1$

$u(23) = 1 - 1 = 0$

$u(13) = 1 - 1 = 0$

$u(123) = 2 - 2 = 0$

The Shapley value for the Difference Game can be usually calculated by using the Shapley value formula. After this is done, the Shapley value for the game in question can be found in terms of the Shapley values of the Difference Game and an edge-added or and edge-removed game, using the Difference Game Formula: $\phi(\text{Edge-in}) - \phi(\text{Difference Game}) = \phi(\text{Edge-out})$.

III. Results

This section is divided into two sections: base graphs and constructions. Base graph results involve formulas for specific types of graphs, such as cycles and complete graphs. Construction results involve formulas for graphs which are "created"; creation methods include adjoining graphs and subdividing edges.

A. Base graphs.

Formulas for certain graph games can be easily determined because of the symmetry of the players.

Theorem 1: Let G be a cycle C_n. The Shapley value for each player is $\frac{n-1}{n}$.

Proof: There are n players in the graph game $W(G)$. The worth for the grand coalition equals $n-1$. By equal treatment and efficiency, each player shall receive $\frac{n-1}{n}$, according to the Shapley value. ★

Theorem 2: Let G be a complete graph K_n. The Shapley value for each player is $\frac{n-1}{n}$.

Proof: A graph is complete if every vertex is adjacent to every other vertex. The proof is exactly the same as the proof for Theorem 1. ★

Theorem 3: Let G be a complete bipartite graph $K_{m,m}$, where $2 \cdot m = n$. The Shapley value for each player is $\frac{n-1}{n}$.

Proof: A complete bipartite graph is a graph whose vertices are divided into two sets and every vertex in a set is adjacent to all of the vertices in the other set, and none of the vertices in its own set. The proof is exactly the same as the proof for Theorem 1. ★

Other graphs need more proving machinery.

Theorem 4: For a wheel graph W_n, where n equals order(W_n), the Shapley values for each player in the graph game is defined as follows:

Let i be a player.

(1) If i is represented by a vertex on the rim of the wheel, then $\phi_i = \frac{5}{6} - \frac{1}{n(n-1)}$

(2) If i is represented by the hub vertex, then $\phi_i = \frac{(n-1)}{6} + \frac{1}{n}$

Proof: By The Difference Game.

Let ab be a rim edge of the original graph.

Let the Edge-in game be the original graph game.

Let the Edge-out game be the game on the graph $W_n - (ab)$.

Analysis of the Edge-out game:

The graph $W_n - (ab)$ is a series of C_3 graphs edge adjoined together. {See Theorem ~~[illegible]~~ 9}

Spoke vertex players who are neither a nor b have the Shapley value $\frac{5}{6}$.

Analysis of the Difference Game:

The worth function values for the Difference Game equals 1 for all coalitions of size $< n-1$ which contain both players a and b, but not ~~g.~~ the hub vertex. All other coalitions will have worth zero. Therefore, the only coalition S for which a player $i \neq a$ or b to have a non-zero marginal $u(S) - u(S-i)$ is when $S{=}N-h$, where h is the hub vertex. In this case, the marginal value equals -1. Therefore, the Shapley value equals $(-1)\cdot\dfrac{(n-(n-1))!(n-1-1)!}{n!} = \dfrac{-1}{(n)(n-1)}$.

Analysis of the Edge-in Game:

Using the Difference Game formula, the Shapley value for a rim vertex player other than players a or b equals $\frac{5}{6} - \frac{1}{(n)(n-1)}$. However, by an equal treatment argument, players a and b also have that value.

Finally, by efficiency ~~and equal treatment~~, the Shapley value for the hub vertex equals the following:

$$(n-1) \ -(n-1)[\,\tfrac{5}{6} - \frac{1}{(n)(n-1)}\,] \ = \ (n-1) - \frac{5(n-1)}{6} + \frac{1}{n} \ = \ \frac{(n-1)}{6} + \frac{1}{n}.\bigstar$$

Theorem 5: Let G be a complete bipartite graph $K_{2,x}$, where $n = 2 + x$. Let the two vertices of degree x be called a and b. The Shapley value for players a and b equals $\frac{n+1}{6} - \frac{1}{n(n-1)}$. The Shapley value for all other players equals $\frac{2}{3} + \frac{2}{n(n-1)(n-2)}$.

Proof: By the Difference Game.

Let the Edge-out game be the original graph game.

Let the Edge-in game be the game on the graph ~~W_n~~ $K_{2,x}$ $+ (ab)$, where (ab) is the edge a-b.

Analysis of the Edge-in game:

The graph ~~W_n~~ $K_{2,x}$ $+ (ab)$ is a series of C_3 graphs edge adjoined onto the edge a-b. {See Theorem ~~8~~ 9}

Therefore, the Shapley value for players a and b $= \dfrac{2(n-2)}{3} - \dfrac{(n-3)}{2}$

$$= \frac{2n-4}{3} - \frac{n-3}{2}$$

$$= \frac{4n-8-3n+9}{6}$$

$$= \frac{n+1}{6}.$$

Analysis of the Difference Game:

The worth function values for the Difference Game equals one for only the coalition $\{a, b\}$. All other coalitions have the worth zero. This is because the edge a-b only affects coalitions which do not

In each of the subgraphs generated by Forest-Size Decomposition, the Shapley value of a is $\frac{1}{2}$. Therefore, the Shapley value for a in the original graph is $\frac{n}{2}$. ★

The following three theorems pertain to a class of graphs called Adjoining Graphs.

Disconnected graphs are comprised of a number of components which are graphs themselves. Certain graphs can be also formed by bringing together, or adjoining, two distinct graphs at a common edge or vertex. The Shapley value for players in games on these types of graphs can be determined by looking at the games on these "subgraphs."

Theorem 7: Let G be a graph with k components. The Shapley value of a player, a, in $W(G)$ is equal to the Shapley value of a found in the forest-size subgraph game on the component containing a.

Proof: Perform a Forest-Size Decomposition on G, producing k subgraphs. The subset of vertices, X, for each subgraph is determined by choosing all vertices in a distinct component.

The sum of the subgraph game worth functions is the same as the worth function of the original graph game. Indeed, every edge is accounted for exactly once. Any maximal spanning forest can be exactly represented in the collection of subgraph games by developing spanning forests using the exact edges from the original spanning forests. These forests are maximal; if they were not, then this would imply that the original spanning forest is not maximal.

Therefore, the Shapley value for any player a is equal to its Shapley value in the subgraph game constructed on its component. But all of the isolated vertices in the subgraph act as dummies in the subgraph game. Therefore, player a has the same Shapley value in the subgraph game only on the component of a. ★

Notation:

Let A and B be two distinct graphs, with $a \in V(A)$ and $b \in V(B)$.

Let a and b be called point vertices, and let p be the vertex of adjoinment, called a pivot.

$G = (A \cdot B)(a,b)$ is the vertex adjoined graph with respect to vertices a in A and b in B, where $V(G) = V(A) \cup V(B) - \{a,b\} \cup \{p\}$ and $E(G)=E(A) \cup E(B)$. with a and b identified with p.

Theorem 8: Let G and H be distinct starting graphs, and let g be a vertex in G, and let h be a vertex in H. Let K be the vertex adjoined graph $(G \cdot H)(g,h)$. The Shapley value for a player other than the pivot player in $W(K)$ equals the Shapley value for that player in $W(G)$ or $W(H)$. The Shapley value for the pivot player equals the sum of the Shapley values for player g and h.

Proof: Perform a Forest-Size Decomposition on the vertex adjoined graph game. The subset of vertices, X, for each subgraph is chosen by selecting all of the vertices of one of the two starting graphs. The second subgraph is created by selecting the vertices of the other starting graphs.

The sum of the two subgraph games equals the worth function of the vertex adjoined game. Indeed, coalitions consisting of players from one of the starting graph games have the same worth as the same coalitions in $(G \cdot H)(g,h)$, because edges for spanning forests are chosen exactly for the original game and the subgraph games. Coalitions consisting of players from both of the starting graph games have the same worth as the same coalitions in the vertex adjoined game; spanning forests are created using the same edges in both games. All of these spanning forests in the subgraph games are maximal; if edges need to be added, then this would imply that the spanning forest in the vertex adjoined game is also not maximal.

The Shapley value for each non-pivot player equals the Shapley value of the player in the subgraph game constructed on the set of vertices in its starting graph. Therefore, its value does not change after the adjoining process. The Shapley value of the pivot equals the sum of the Shapley value for the pivot player in the two subgraph games. ★

Notation:

Let A and B be two distinct graphs, with a_1, $a_2 \in V(A)$ and $b_1, b_2 \in V(B)$, such that $a_1a_2 \in E(A)$ and $b_1b_2 \in E(B)$.

Let a_1 and b_1 be called upper point vertices and let a_2 and b_2 be called lower point vertices.

$(A - B)(a_1a_2, b_1b_2)$ is the edge adjoined graph at edges with respect to edges a_1a_2 in A and b_1b_2 in B.

When two graphs are adjoined at an edge, called a squeeze edge, the upper point vertices are adjoined, as well as the lower point vertices. All edges incident to a point vertex are now incident to the respective vertices incident to the squeeze edge. The rest of the edges remain unchanged.

Theorem 9: Let G and H be distinct starting graphs, with $g_1g_2 \in E(G)$ and $h_1h_2 \in E(H)$. Let K be the edge adjoined graph. The Shapley value for each player represented by a vertex not on the squeeze edge is not changed after the adjoining process. The Shapley value for players represented by vertices on the squeeze edge equals the sum of the Shapley values on $W(G)$ and $W(H)$ minus $\frac{1}{2}$.

Proof: Perform a Forest-Size Decomposition on $(G - H)(g_1g_2, b_1b_2)$. The subset of vertices, X, consists of all of the vertices in one of the starting graphs. Create two subgraphs by choosing the vertices of the two starting graphs in this manner. Finally, a subset consisting of the two vertices incident with the squeeze edge is chosen for the last subgraph.

The three subgraph games determine a worth function which is the same as the worth function of the edge adjoined game. Coalitions consisting of players from one of the starting graph games having at most one point vertex player have the same worth as the same coalitions from the collection of subgraph games: spanning forests are constructed exactly the same way in both cases. Coalitions consisting of players from both starting graph games having at most one point vertex player have the same worth as the same coalitions from the collection of subgraph games: spanning forests are constructed exactly the same way in both cases. Finally, coalitions containing both point vertex players have the same worth as the same coalitions from the collection of subgraph games. The spanning forest for these coalitions is created by first adding the squeeze edge to the forest, and then building around it. The spanning forests in the collection of subgraph games are constructed in the same manner. In the first two subgraph games, the squeeze edge is counted twice; therefore, the subgraph game consisting of only the squeeze edge counts as a negative worth.

The Shapley value for a non-vertex point player equals the Shapley value the player would have received its respective starting game. The Shapley value for a vertex point player equals the sum of the Shapley values for each subgraph game minus $\frac{1}{2}$ (from the third game). ★

The next theorem involve subdivisions of graphs.

Theorem 10: Let G be a cycle C_n. A subdivision of an edge results in an increase in the Shapley value for each original player by $\frac{1}{(n)(n+1)}$.

Proof: A subdivision of a C_n produces a C_{n+1} graph. Therefore, the change in the Shapley values is equal to the following:

$$\Delta\phi = \frac{n}{n+1} - \frac{n-1}{n} = \frac{1}{(n)(n+1)} \quad \bigstar$$

IV. *Conjectures, Open Questions and Ideas*

(1) Subdivisions on a chord of a cycle.

Let G be a cycle C_4. Label two non-adjacent vertices a and b. Add the edge a-b to the graph. All two-degree vertices will be added to this "chord." The resulting graph G' is of order n. The Shapley value for an added player equals $\frac{n-1}{n} - \frac{2}{n(n-1)}$. The Shapley value of the two non-labeled original vertices equals $\frac{3}{4} - \frac{1}{n(n-1)}$.

(2) Two-Form adjoinings.

Attach m Two-Forms to a cycle C_k at the same two vertices. The Shapley value for players on the outer loop of the cycle equals the following:

$$\frac{k-1}{k} - \sum_{j=1}^{m} \frac{j!}{\prod_{i=0}^{j} (k+i)}$$

(3) Proof Conjecture: All theorems dealing with Forest-Size Games can be proved using either Forest-Size Decomposition or the Difference Game.

V. Appendix

I. Shapley Value/Banzhaf Value program

The following is a Turbo Pascal ver6.0 program written in order to facilitate the process of finding these results.

```
program Shapprogram(input,output);

{BY DARREN LIM}
{JULY 22, 1992}
{VERSION 1.2}

{DESCRIPTION: THIS PROGRAM WILL CALCULATE THE SHAPLEY VALUE OR THE ABSOLUTE
              BANZHAF VALUE FOR FOREST-SIZE COOPERATIVE GAMES.}
{PROCEDURE: THIS PROGRAM WILL FIRST CALCULATE THE WORTHS FOR EACH COALITION.
            IT WILL THEN CALCULATE THE APPROPRIATE ALLOCATION.}
{INPUT: NUMBER OF VERTICES, EDGES}
{OUTPUT: ALLOCATION VECTOR}

type
  dset=set of char;
  set2=set of 1..15;
  intset=array[1..15] of boolean;
  graphtype=array[1..15,1..15] of boolean;
  arraytype=array[0..11000] of shortint;
  array2type=array[1..15] of real;
  settype=set of 1..200;

const
  digits:dset=['0'..'9'];

{****************************************************************************}

function coalitionsize(b:intset;num:integer):integer;
  {Description: returns the number of elements in the coalition}
  {Input: binary coalition}
  {Output: number of elements in the coalition}

var
  temp,count:integer;

begin
  temp:=0;
  for count:=1 to num do
    if b[count]
      then Inc(temp);
  coalitionsize:=temp
end;
{****************************************************************************}

function fact(n:integer):real;
  {Description: returns n!}
  {Input: an integer n}
  {Output: n!}

begin
  if n<=1
    then fact:=1.0
    else fact:=n*fact(n-1)
end;
{****************************************************************************}

procedure binvector(number:integer;var inter:intset);
  {Description: returns the binary vector for a number}
  {Input: a number}
  {Output: its number in binary vector form}
```

```
var
  c,rem:integer;
  int:intset;

begin
  for c:=1 to 15 do
    int[c]:=false;
  c:=0;
  while number>0 do
    begin
      Inc(c);
      rem:=number mod 2;
      if rem=1
        then int[c]:=true;
      number:=number div 2
    end;
  inter:=int
end;
{***************************************************************************}

function findvalue(g:graphtype;num:integer;int:intset):integer;
  {Description: finds and returns the worth for a coalition}
  {Input: adjacency matrix, number of vertices, binary vector for a coalition}
  {Output: worth of a coalition}

var
  temp,count,index,c2:integer;
  dset,markedset,copyset:set2;

begin                                       {This is the most complicated procedure}
  dset:=[];                                 {in the program.  The Depth-First}
  temp:=coalitionsize(int,num);             {search algorithm is used to determine}
  for count:=1 to num do                    {components in the induced subgraph.}
    if int[count]
      then dset:=dset+[count];
  while dset<>[] do
    begin
      markedset:=[];
      index:=1;
      while not int[index] do
        Inc(index);
      markedset:=markedset+[index];
      repeat
        copyset:=markedset;
        for count:=1 to num do
          if (count in markedset)
            then for c2:=1 to num do
                   if not(c2 in markedset) and int[c2] and g[c2,count]
                     then markedset:=markedset+[c2];
      until markedset=copyset;
      dset:=dset-markedset;
      for count:=1 to num do
        if count in markedset
          then int[count]:=false;
      Dec(temp)
    end;
  findvalue:=temp
end;
{***************************************************************************}
```

```
function value(g:graphtype;num:integer;count:integer):integer;
  {Description: converts a number into a binary vector and finds the
                corresponding coalition worth}
  {Input: adjacency matrix, order of graph, coalition number}
  {Output: worth of the coalition}

var
  int:intset;

begin
  binvector(count,int);
  value:=findvalue(g,num,int)
end;
{***************************************************************************}

function twopower(n:integer):integer;
  {Description recursively returns 2 raised to the nth power}
  {Input: exponent}
  {Output: 2^N}

begin
  if n=0
    then twopower:=1
    else if n=1
           then twopower:=2
           else twopower:=2*twopower(n-1)
end;
{***************************************************************************}

function computeshap(s,n,marg:integer):real;
  {Description: computes Shapley value}
  {Input: array of worths, n, s, number}
  {Output: contribution to Shapley value}

var
  marginal:integer;
  temp:real;

begin
  marginal:=marg;
  temp:=marginal*fact(n-s)*fact(s-1)/fact(n);
  computeshap:=temp
end;
{***************************************************************************}

function computeban(n,marg:integer):real;
  {Description: computes Banzhaf value}
  {Input: array of worths, n, s, number}
  {Output: contribution to Banzhaf value}

var
  marginal:integer;
  temp:real;

begin
  marginal:=marg;
  temp:=marginal/twopower(n-1);
  computeban:=temp
end;
```

```
{**************************************************************************}


procedure coalition(g:graphtype;num:integer);
  {Description: calculates the worth function for all coalitions and prints
                allocation vector}
  {Input: adjacency matrix, order of the graph}
  {Output: none}

var
  binarray:intset;
  conum,count,marg,shapindex,result:integer;
  arr:arraytype;
  allocation:array2type;
  tempstr:string;

begin
  repeat
    writeln(output,'Allocation method options');
    writeln(output);
    writeln(output,'1 - Shapley value');
    writeln(output,'2 - Banzhaf value');
    writeln(output);
    write(output,'Choose an option (1-2) =>');
    readln(input,result)
  until (result>0) and (result<3);
  for count:=1 to 15 do
    allocation[count]:=0.0;
  arr[0]:=0;
  for count:=1 to twopower(num)-1 do
    begin
      arr[count]:=value(g,num,count);
      binvector(count,binarray);
      conum:=coalitionsize(binarray,num);
      for shapindex:=1 to num do
        if binarray[shapindex]
          then
            begin
              marg:=arr[count]-arr[count-twopower(shapindex-1)];
              case result of
                1:allocation[shapindex]:=allocation[shapindex]+
                                         computeshap(conum,num,marg);
                2:allocation[shapindex]:=allocation[shapindex]+
                                         computeban(num,marg)
              end;{Case}
            end
    end;
  case result of
    1:tempstr:='Shapley(';
    2:tempstr:='Banzhaf('
  end;{Case}
  for count:=1 to num do
    writeln(output,tempstr,count,')=',allocation[count]:12:9);
  readln(input)
end;
{**************************************************************************}

procedure process(str:string;var numset:settype);
  {Description: interprets an input string of edges}
  {Input: sequence of integers and commas}
  {Output: updated set of numbers for the adjacency matrix}

var
  tstr:string;
  tempint,code:integer;

begin
  while Length(str)>0 do
    begin
      tstr:='';
      while (str[1] in digits) and (Length(str)>0) do
        begin
          tstr:=tstr + str[1];
          str:=copy(str,2,Length(str)-1)
        end;
      if str<>''
        then str:=copy(str,2,Length(str)-1);
      val(tstr,tempint,code);
      if code=0
        then numset:=numset+[tempint]
        else
          begin
            writeln(output,'Error in string');
            readln(input,str);
            Halt(1)
          end
    end
end;
{****************************************************************************}

procedure globalfilter;

var
  numset:settype;
  count,index,Vertnum:integer;
  answer:string;
  graphmat:graphtype;

begin
  digits:=['0'..'9'];
  for count:=1 to 15 do
    for index:=1 to 15 do
      graphmat[count,index]:=false;
  Write(output,'Enter the number of vertices =>');
  readln(input,Vertnum);
  for count:=1 to Vertnum do
    begin
      numset:=[];
      Write(output,'Type the vertices adjacent to vertex ',count);
      Write(output,' separated by commas =>');
      Readln(input,answer);
      process(answer,numset);
      for index:=count+1 to Vertnum do
        if index in numset
          then
            begin
              graphmat[index,count]:=true;
              graphmat[count,index]:=true
            end
    end;
  coalition(graphmat,vertnum)
end;
{******************************************************************************}

begin
  globalfilter
end.
```

Citations

Chartrand, Gary. Introductory Graph Theory. New York: Dover Publications, 1977.

Hartsfield, Nora and Gerhard Ringel. Pearls in Graph Theory: A Comprehensive Introduction. New York: Academic Press, 1990.

Housman, David. "Fair Allocation." Manuscript.

Housman, David and Lori Jew. "Allocation Methods for Cooperative Games: An Axiomatic Approach."
Manuscript.