```
> # Authors: D. Offner and K. Ojakian
> # First we define a number of functions.
> # Then we define a number of graphs.  For each graph we call a function that displays the graph
      with its corner ranking.
> # WARNING: The drawingss show that the graphs are cop-win and indicate the ranks of the
      vertices.  However, the edges are not drawn well, so the edges as drawn should be viewed as
      approximations ...
>
>
>
> with(GraphTheory) :
> with(ListTools) :
>
> CorneredBy := proc(G, v)

      description "Returns list of vertices in G that corner (not neccessarily strictly) v and empty
      list if nothing does";
                  local N, L, u;
                  N := Neighborhood(G, v, closed);
                  L := [ ];
                   for u in Neighborhood(G, v, open) do
      # Range over neighbors of v
                        if {op(N)} subset {op(Neighborhood(G, u, closed))} then
      # If a neighbor of v contains the neighborhood of v, add it to the list of veritces that corner v
                              L := [op(L), u];
                        fi;

                   od;
                   return L;
                  end proc;

CorneredBy := proc(G, v)                                                                    (1)
    local N, L, u;
    description
    "Returns list of vertices in G that corner (not neccessarily strictly) v and empty list if nothing
    does";
    N := GraphTheory:-Neighborhood(G, v, closed);
    L := [ ];
    for u in GraphTheory:-Neighborhood(G, v, open) do
        if {op(N)}subset{op(GraphTheory:-Neighborhood(G, u, closed))} then
            L := [op(L), u]
        end if
    end do;
    return L
end proc
>
```

```
>
>
> AreTwins := proc(G, u, v)
                    description "Returns True if u and v are twins and False otherwise";
                    return Neighborhood(G, v, closed) = Neighborhood(G, u, closed);
    end proc;
```

```
AreTwins := proc(G, u, v)                                                    (2)
    description "Returns True if u and v are twins and False otherwise";
    return GraphTheory:-Neighborhood(G, v, closed) = GraphTheory:-Neighborhood(G, u,
    closed)
end proc
```

```
>
>
>
> StrictCorners := proc(G)
      description "Returns list of strict corners in G";
      local vertexlist, u, v, strict;
       vertexlist := [ ];
      for v in Vertices(G) do
       # Range over all the vertices in G
         strict := false;
         for u in CorneredBy(G, v) do
       # For each vertex, check if some other vertex strictly corners it
              if not (AreTwins(G, u, v)) then strict := true fi;
         od;
         if strict then vertexlist := [op(vertexlist), v] fi;
       # If there is some other vertex strictly cornering it, then add that vertex to the list
      od;
      return vertexlist;
    end proc;
```

```
StrictCorners := proc(G)                                                     (3)
    local vertexlist, u, v, strict;
    description "Returns list of strict corners in G";
    vertexlist := [ ];
    for v in GraphTheory:-Vertices(G) do
        strict := false;
        for u in CorneredBy(G, v) do
            if not AreTwins(G, u, v) then strict := true end if
        end do;
        if strict then vertexlist := [op(vertexlist), v] end if
    end do;
    return vertexlist
end proc
```

```
>
>
>
```

```
> RankVertices := proc(G, vlist, rank)

      description "Ranks the given list of vertices (vlist) so that they all have the rank equal to the
      value of 'rank'  ";
       local v;
       for v in vlist do
                SetVertexAttribute(G, v, "rank" = rank);
        od;

   end proc;
```

```
RankVertices := proc(G, vlist, rank)                                                    (4)
    local v;
    description
    "Ranks the given list of vertices (vlist) so that they all have the rank equal to the value of
    'rank'  ";
    for v in vlist do
        GraphTheory:-SetVertexAttribute(G, v, "rank" = rank)
    end do
end proc
```

```
>
>
>
```

```
> RankGraph := proc(G)

      description "The given graph has its vertex 'rank' attribute set to its numerical rank; unranked
      vertices assigned -1";
      local H, corners, rank;
      H := G;

      corners := StrictCorners(H);
      rank := 1;
      while (corners ≠ [ ]) do                    # Loop until there are NO strict corners.
            RankVertices(G, corners, rank);       # Rank the current strict corners
            H := DeleteVertex(H, corners);        # Delete the current strict corners
            corners := StrictCorners(H);          # Find the new strict corners in the smaller graph
            rank := rank + 1;                     # Increment the rank
       od;

      if IsClique(H) then RankVertices(G, Vertices(H), rank)
      # Assign the final vertices their appropriate ranks, using "-1" to refer to infinite rank vertices.
      else RankVertices(G, Vertices(H), -1) fi;

   end proc;
```

```
RankGraph := proc(G)                                                                    (5)
    local H, corners, rank;
    description
    "The given graph has its vertex 'rank' attribute set to its numerical rank; unranked vertices
    assigned -1";
```

```
    H := G;
    corners := StrictCorners(H);
    rank := 1;
    while corners <> [ ] do
        RankVertices(G, corners, rank);
        H := GraphTheory:-DeleteVertex(H, corners);
        corners := StrictCorners(H);
        rank := rank + 1
    end do;
    if GraphTheory:-IsClique(H) then
        RankVertices(G, GraphTheory:-Vertices(H), rank)
    else
        RankVertices(G, GraphTheory:-Vertices(H), -1)
    end if
end proc
```

```
>
>
>
> LabelByRank := proc(G)

    description "Takes a ranked graph and labels the vertices for printing, by putting their rank in
    parentheses";
     local new_labels, v, next_label, rank;

     new_labels := [ ];
    for v in Vertices(G) do
    # Ranges over all the vertices in the graph creating a list of corresponding string labels
        rank := GetVertexAttribute(G, v, "rank");
        next_label := cat(convert(v, string), " (", convert(rank, string), ")");
        new_labels := [op(new_labels), next_label];
    od;

    return RelabelVertices(G, new_labels);

  end proc;
```

```
LabelByRank := proc(G)                                                    (6)
    local new_labels, v, next_label, rank;
    description
    "Takes a ranked graph and labels the vertices for printing, by putting their rank in
    parentheses";
    new_labels := [ ];
    for v in GraphTheory:-Vertices(G) do
        rank := GraphTheory:-GetVertexAttribute(G, v, "rank");
        next_label := cat(convert(v, string), " (", convert(rank, string), ")");
        new_labels := [op(new_labels), next_label]
```

```
    end do;
    return GraphTheory:-RelabelVertices(G, new_labels)
end proc
```

```
> RankVector := proc(G)
        description "Given a ranked, unlabeled graph, returns a vector of the vertex ranks";
        local rank_list, v, next_rank;

         rank_list := [ ];
         for v in Vertices(G) do
        # Ranges over all he vertices, collecting the ranks into a list
              next_rank := GetVertexAttribute(G, v, "rank");
              rank_list := [op(rank_list), next_rank];
         od:

        return rank_list;

    end proc
```

```
RankVector := proc(G)                                                    (7)
    local rank_list, v, next_rank;
    description "Given a ranked, unlabeled graph, returns a vector of the vertex ranks";
    rank_list := [ ];
    for v in GraphTheory:-Vertices(G) do
        next_rank := GraphTheory:-GetVertexAttribute(G, v, "rank");
        rank_list := [op(rank_list), next_rank]
    end do;
    return rank_list
end proc
```

```
> GraphRankValue := proc(G)

        description "Given a ranked, unlabeled graph. Returns its maximum FINITE rank; returns -1
        if NO finite rank";

           return FindMaximalElement(RankVector(G));

    end proc;
```

```
GraphRankValue := proc(G)                                                (8)
    description
    "Given a ranked, unlabeled graph. Returns its maximum FINITE rank; returns -1 if NO
    finite rank";
    return ListTools:-FindMaximalElement(RankVector(G))
```

```
end proc
>
>
>
> PositionVertices := proc(G)

      description "Given a graph with numbered vertices and rank attribute, it outputs a position
      vector to be used by the function:  SetVertexPositions";
       local current_left, v, rank, next_position, position_list, graph_rank, top, row;

      position_list := [ ];
      graph_rank := GraphRankValue(G);
      if graph_rank = -1 then
      # Determines "top" - the number of rows of vertices and the value of the top row
           top := 1;
      else
           top := graph_rank + 1;
       fi;
       current_left := Vector(top, 0);
      # Creates a vector of zeros of length "top"

      for v in Vertices(G) do
      # Ranges over vertices
              row := GetVertexAttribute(G, v, "rank");
      # Chooses the row of a vertex to correspond to its rank
              if row = -1 then row := top  fi;
              next_position := [ current_left[row],  row];
      # Creates a coordinate position for the vertex
              position_list := [op(position_list),  next_position ];
      # Adds the position to the list
              current_left[row] := current_left[row] + 1;
      # Updates the leftmost position for vertices of a given rank
      od;
      return position_list;

   end proc;
PositionVertices := proc(G)                                                    (9)
    local current_left, v, rank, next_position, position_list, graph_rank, top, row;
    description
    "Given a graph with numbered vertices and rank attribute, it outputs a position vector to be
    used by the function:  SetVertexPositions";
    position_list := [ ];
    graph_rank := GraphRankValue(G);
    if graph_rank = -1 then top := 1 else top := graph_rank + 1 end if;
    current_left := Vector(top, 0);
    for v in GraphTheory:-Vertices(G) do
        row := GraphTheory:-GetVertexAttribute(G, v, "rank");
        if row = -1 then row := top end if;
```

```
        next_position := [current_left[row], row];
        position_list := [op(position_list), next_position];
        current_left[row] := current_left[row] + 1
    end do;
    return position_list
end proc
```

```
> DrawRankedGraph := proc(G)
      description "Draws graph G as a corner ranked graph";
      local H;
       H := G;
      RankGraph(H);
      SetVertexPositions(H, PositionVertices(H));
      DrawGraph(LabelByRank(H));

  end proc;
```

```
DrawRankedGraph := proc(G)                                                    (10)
    local H;
    description "Draws graph G as a corner ranked graph";
    H := G;
    RankGraph(H);
    GraphTheory:-SetVertexPositions(H, PositionVertices(H));
    GraphTheory:-DrawGraph(LabelByRank(H))
end proc
```

```
> isCopWin := proc(G)

      description "Given a ranked, unlabeled graph, returns True if cop-win (i.e. finite ranked),
      False otherwise";

          return not member(-1, RankVector(G));
  end proc;
```

```
isCopWin := proc(G)                                                           (11)
    description
    "Given a ranked, unlabeled graph, returns True if cop-win (i.e. finite ranked), False
    otherwise";
    return not member(-1, RankVector(G))
end proc
```

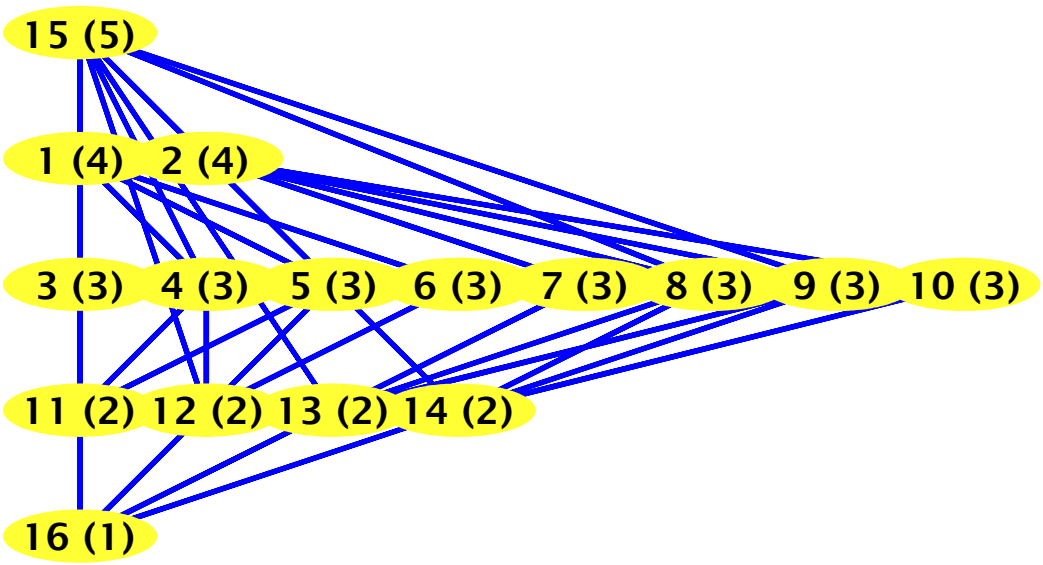
```
> ##################################################################
> ################### BEGIN 1-COP-WIN GRAPHS ###################
> ##################################################################
>
>
>
> GraphOne12841 := Graph(16)
```

GraphOne12841 := Graph 1: an undirected unweighted graph with 16 vertices and 0 edge(s) **(12)**

```
> AddEdge( GraphOne12841, { {15, 4}, {15, 5}, {15, 8}, {15, 9}, {3, 4}, {4, 5}, {5, 6}, {6, 3},
      {7, 8}, {8, 9}, {9, 10}, {10, 7}, {3, 11}, {4, 11}, {5, 11}, {15, 11}, {4, 12}, {5, 12}, {6,
     12}, {15, 12}, {7, 13}, {9, 13}, {8, 14}, {10, 14}, {15, 13}, {15, 14}, {11, 16}, {12, 16},
      {13, 16}, {14, 16}, {15, 1}, {15, 2}, {1, 3}, {1, 4}, {1, 5}, {1, 6}, {2, 7}, {2, 8}, {2, 9},
      {2, 10}, {8, 13}, {9, 14}, {15, 16} } )
```

Graph 1: an undirected unweighted graph with 16 vertices and 43 edge(s) **(13)**

```
> DrawRankedGraph(GraphOne12841)
```




```
>
>
>
> GraphOne126421 := Graph(16)
```

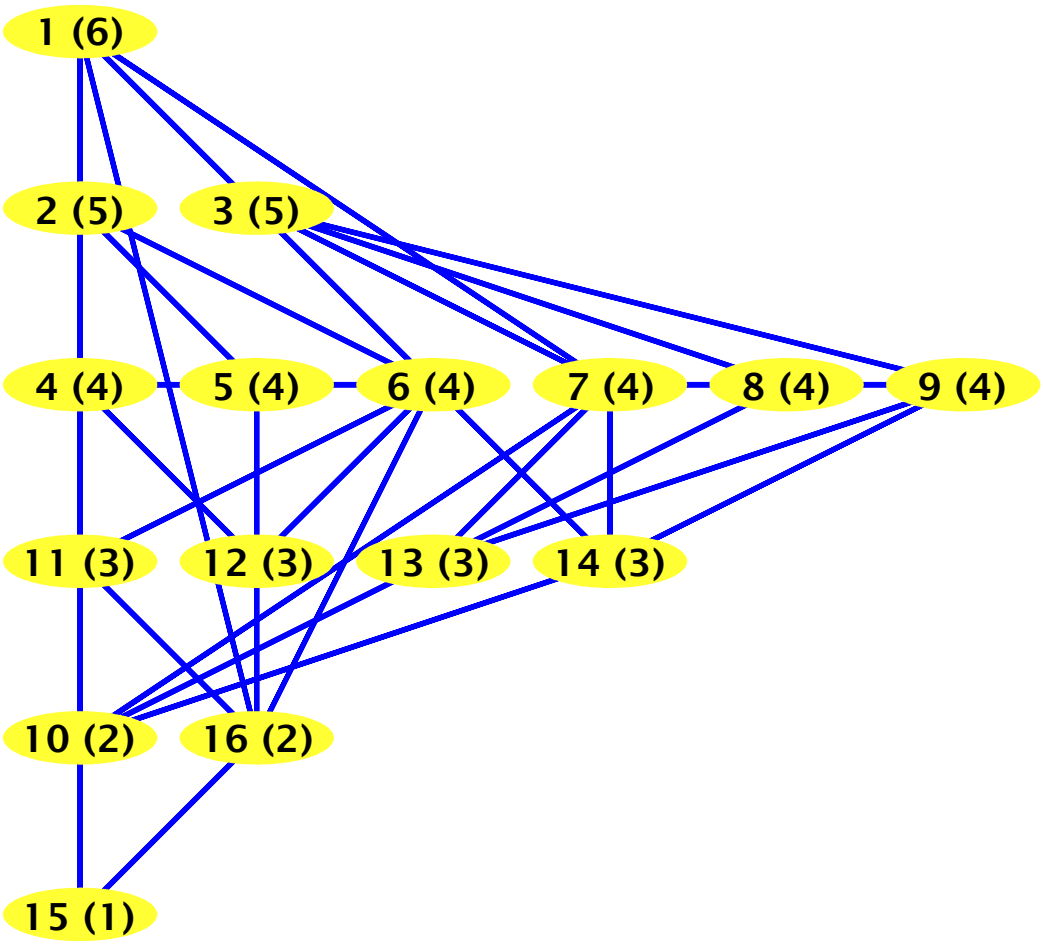
(14)

GraphOne126421 := *Graph 2: an undirected unweighted graph with 16 vertices and 0 edge(s)* **(14)**

```
> AddEdge( GraphOne126421, { {1, 2}, {1, 3}, {2, 4}, {2, 5}, {2, 6}, {3, 7}, {3, 8}, {3, 9}, {4,
    5}, {5, 6}, {7, 8}, {8, 9}, {2, 11}, {4, 11}, {6, 11}, {4, 12}, {5, 12}, {6, 12}, {7, 14}, {9,
    14}, {3, 14}, {8, 13}, {9, 13}, {7, 13}, {13, 10}, {14, 10}, {12, 16}, {11, 16}, {16, 15},
     {10, 15}, {1, 15}, {1, 16}, {1, 10}, {1, 6}, {1, 7}, {6, 16}, {7, 10} } )
```

Graph 2: an undirected unweighted graph with 16 vertices and 37 edge(s) **(15)**

```
> DrawRankedGraph(GraphOne126421)
```



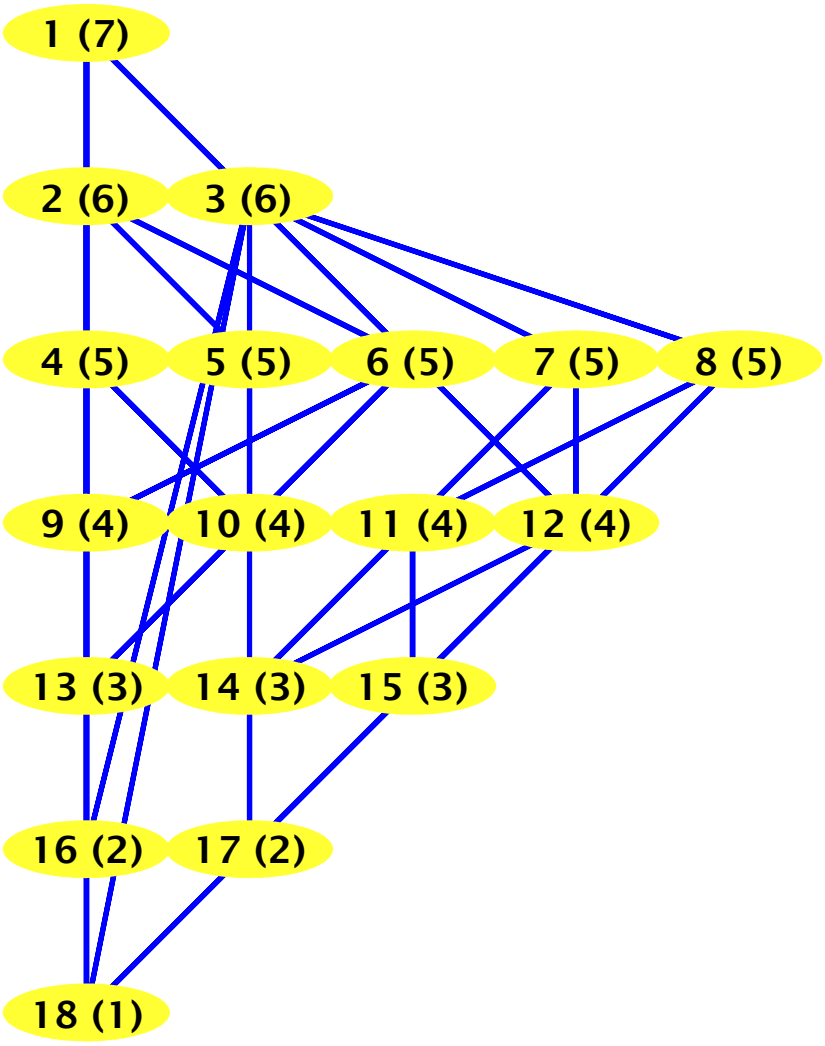

```
>
>
>
> GraphOne1254321 := Graph(18)
```

GraphOne1254321 := *Graph 3: an undirected unweighted graph with 18 vertices and 0 edge(s)* **(16)**

```
> AddEdge( GraphOne1254321, { {1, 2}, {1, 3}, {1, 6}, {1, 16}, {2, 4}, {2, 5}, {2, 6}, {2, 9},
    {3, 7}, {3, 8}, {3, 12}, {3, 17}, {3, 18}, {3, 16}, {4, 5}, {5, 6}, {7, 8}, {4, 9}, {4, 10},
    {5, 10}, {6, 9}, {6, 10}, {6, 13}, {9, 13}, {10, 13}, {13, 16}, {16, 18}, {7, 11}, {7, 12},
    {7, 14}, {8, 11}, {8, 12}, {8, 15}, {11, 14}, {11, 15}, {12, 14}, {12, 15}, {12, 17}, {14,
    17}, {15, 17}, {17, 18}, {1, 13} } )
```

Graph 3: an undirected unweighted graph with 18 vertices and 42 edge(s) **(17)**

```
> DrawRankedGraph(GraphOne1254321)
```




```
>
>
>
```

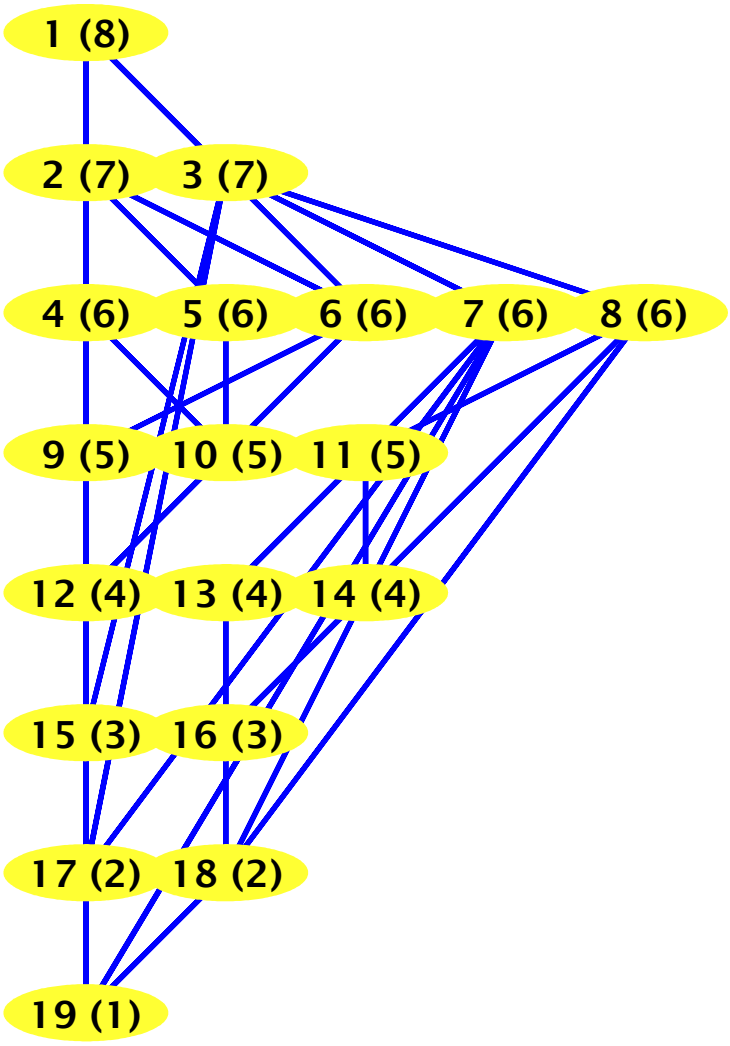
```
> GraphOne12533221 := Graph(19)
```

GraphOne12533221 := Graph 4: an undirected unweighted graph with 19 vertices and 0 edge(s) **(18)**

```
> AddEdge( GraphOne12533221, { {1, 2}, {1, 3}, {1, 12}, {1, 15}, {1, 6}, {2, 4}, {2, 5}, {2,
      6}, {3, 7}, {3, 8}, {3, 15}, {3, 17}, {2, 9}, {4, 5}, {5, 6}, {7, 8}, {4, 9}, {4, 10}, {6, 12},
       {5, 10}, {6, 9}, {6, 10}, {7, 11}, {7, 13}, {7, 17}, {7, 19}, {7, 18}, {8, 11}, {8, 14}, {8,
      16}, {8, 18}, {9, 12}, {10, 12}, {11, 13}, {11, 14}, {12, 15}, {13, 14}, {13, 16}, {14, 16},
       {15, 17}, {16, 18}, {17, 19}, {18, 19} } )
```

Graph 4: an undirected unweighted graph with 19 vertices and 43 edge(s) **(19)**

```
> DrawRankedGraph(GraphOne12533221 )
```




```
>
>
>
```

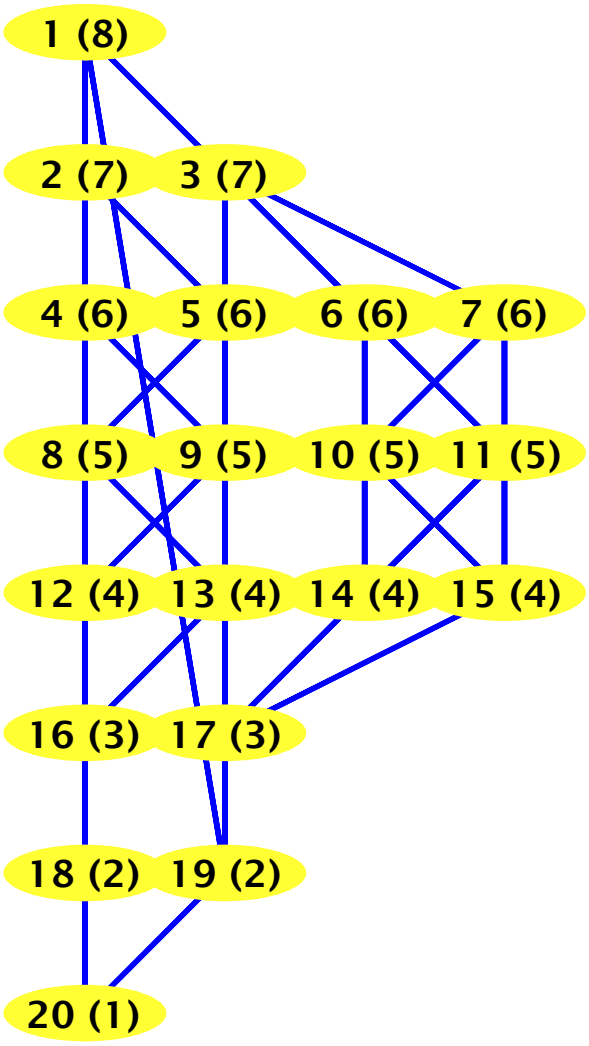
```
> GraphOne12444221 := Graph(20)
```

GraphOne12444221 := Graph 5: an undirected unweighted graph with 20 vertices and 0 edge (s) **(20)**

```
> AddEdge( GraphOne12444221, { {1, 2}, {1, 3}, {1, 18}, {1, 19}, {1, 20}, {2, 4}, {2, 5}, {2,
      8}, {2, 16}, {2, 18}, {3, 6}, {3, 7}, {3, 11}, {3, 17}, {3, 19}, {4, 5}, {4, 8}, {4, 12}, {4,
      9}, {5, 8}, {5, 9}, {5, 13}, {6, 7}, {6, 10}, {6, 14}, {6, 11}, {7, 10}, {7, 11}, {7, 15}, {8,
      12}, {8, 16}, {8, 13}, {9, 12}, {9, 13}, {10, 14}, {10, 15}, {11, 14}, {11, 15}, {11, 17},
       {12, 16}, {13, 16}, {14, 17}, {15, 17}, {16, 18}, {17, 19}, {18, 20}, {19, 20} } )
```

Graph 5: an undirected unweighted graph with 20 vertices and 47 edge(s) **(21)**

```
> DrawRankedGraph(GraphOne12444221)
```




```
>
>
>
```

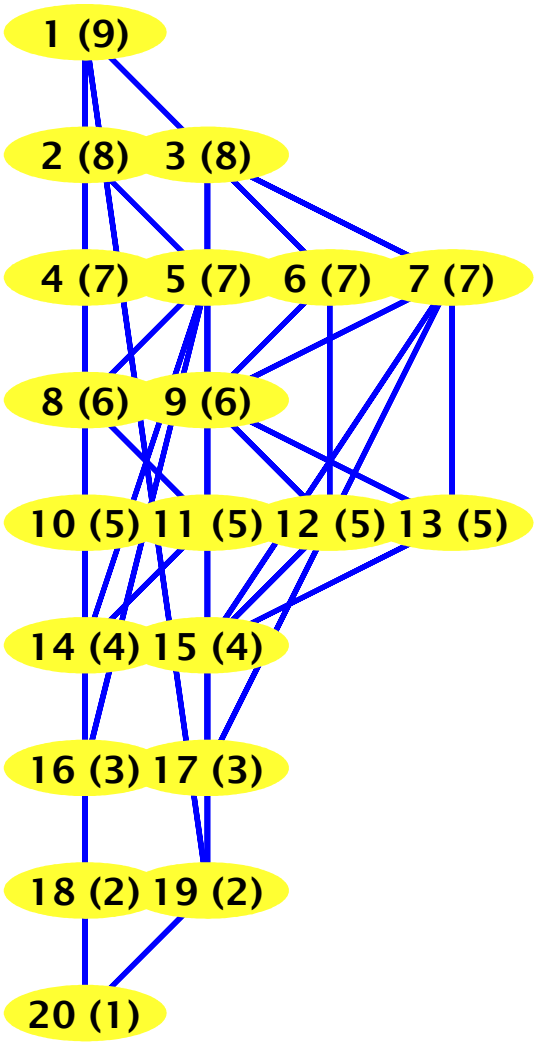
```
> GraphOne124242221 := Graph(20)
```

GraphOne124242221 := Graph 6: an undirected unweighted graph with 20 vertices and 0 edge (s) **(22)**

```
> AddEdge( GraphOne124242221, { {1, 2}, {1, 3}, {1, 18}, {1, 19}, {1, 20}, {2, 4}, {2, 5}, {2,
      16}, {2, 18}, {3, 6}, {3, 7}, {3, 17}, {3, 19}, {4, 5}, {4, 10}, {4, 8}, {5, 8}, {5, 11}, {5,
      14}, {5, 16}, {6, 7}, {6, 12}, {6, 9}, {7, 9}, {7, 13}, {7, 15}, {7, 17}, {8, 10}, {8, 11},
       {9, 12}, {9, 13}, {10, 11}, {10, 14}, {11, 14}, {12, 13}, {12, 15}, {13, 15}, {14, 16},
       {15, 17}, {16, 18}, {17, 19}, {18, 20}, {19, 20} } )
```

Graph 6: an undirected unweighted graph with 20 vertices and 43 edge(s) **(23)**

```
> DrawRankedGraph(GraphOne124242221)
```




```
>
>
>
```

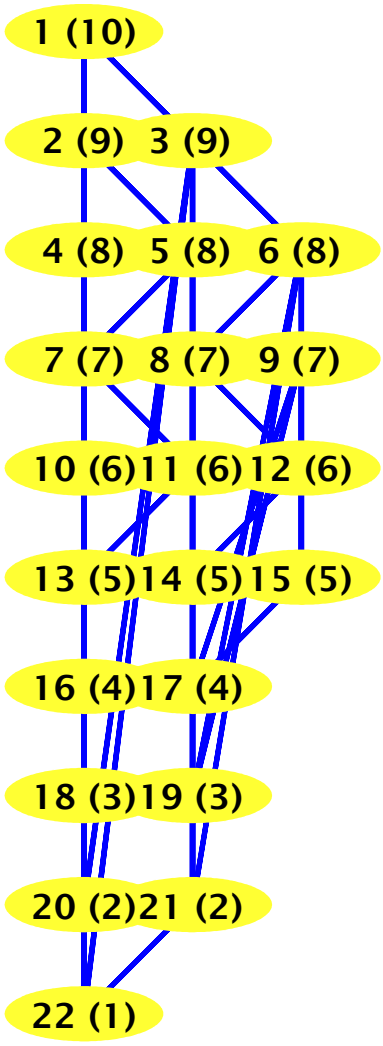
```
> GraphOne1233332221 := Graph(22)
```

GraphOne1233332221 := Graph 7: an undirected unweighted graph with 22 vertices and 0 edge(s) **(24)**

```
> AddEdge( GraphOne1233332221, { {20, 22}, {21, 22}, {17, 19}, {18, 20}, {19, 21}, {15, 17},
      {13, 16}, {16, 18}, {12, 14}, {12, 15}, {14, 15}, {14, 17}, {10, 11}, {10, 13}, {11, 13},
      {9, 12}, {9, 15}, {9, 17}, {9, 19}, {7, 10}, {7, 11}, {8, 9}, {8, 12}, {8, 14}, {6, 8}, {6,
      9}, {6, 19}, {6, 21}, {5, 7}, {5, 11}, {4, 13}, {4, 16}, {4, 5}, {4, 10}, {4, 7}, {1, 2}, {1,
      3}, {1, 20}, {1, 18}, {2, 4}, {2, 5}, {2, 16}, {2, 18}, {3, 6}, {3, 20}, {3, 21}, {3, 22} } )
```

Graph 7: an undirected unweighted graph with 22 vertices and 47 edge(s) **(25)**

```
> DrawRankedGraph(GraphOne1233332221)
```




```
>
>
>
```

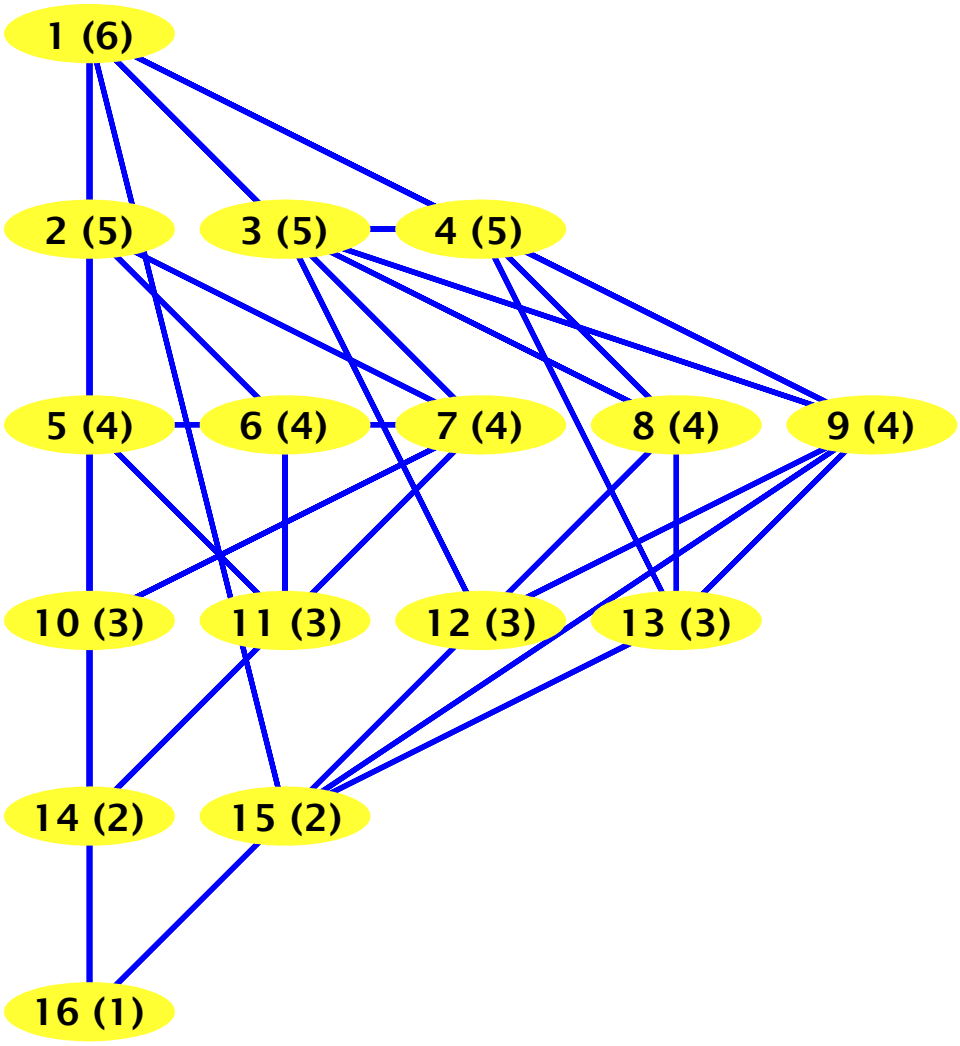
```
> GraphOne135421 := Graph(16)
```

GraphOne135421 := Graph 8: an undirected unweighted graph with 16 vertices and 0 edge(s) **(26)**

```
> AddEdge( GraphOne135421, { {1, 2}, {1, 3}, {1, 4}, {1, 7}, {1, 9}, {1, 14}, {1, 15}, {1, 16},
      {2, 5}, {2, 6}, {2, 7}, {2, 10}, {5, 6}, {6, 7}, {5, 10}, {5, 11}, {6, 11}, {7, 10}, {7, 11},
      {7, 14}, {10, 14}, {11, 14}, {14, 16}, {3, 4}, {3, 8}, {3, 9}, {3, 12}, {4, 8}, {4, 9}, {4,
     13}, {8, 12}, {8, 13}, {9, 12}, {9, 13}, {9, 15}, {12, 15}, {13, 15}, {15, 16} } )
```

Graph 8: an undirected unweighted graph with 16 vertices and 38 edge(s) **(27)**

```
> DrawRankedGraph(GraphOne135421)
```




```
>
>
>
```

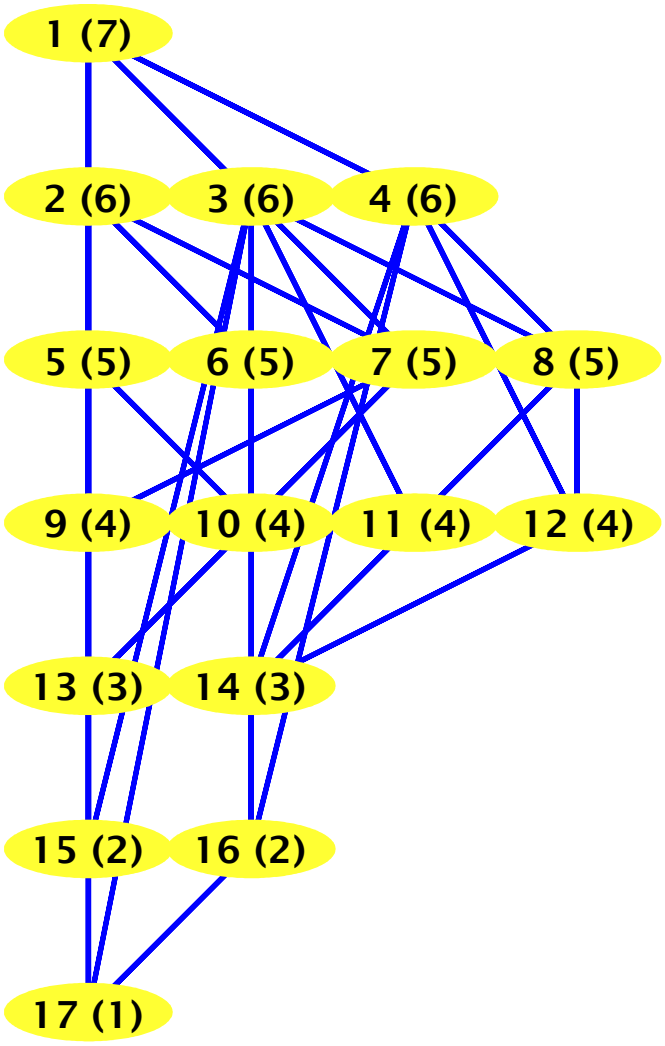
```
> GraphOne1344221 := Graph(17)
```

GraphOne1344221 := Graph 9: an undirected unweighted graph with 17 vertices and 0 edge(s) **(28)**

```
> AddEdge( GraphOne1344221, { {1, 2}, {1, 3}, {1, 4}, {1, 13}, {1, 15}, {1, 7}, {2, 5}, {2, 6},
      {2, 7}, {2, 9}, {5, 6}, {6, 7}, {5, 9}, {5, 10}, {6, 10}, {7, 9}, {7, 10}, {7, 13}, {9, 13},
      {10, 13}, {13, 15}, {15, 17}, {3, 4}, {3, 8}, {3, 11}, {3, 16}, {3, 17}, {3, 15}, {4, 8},
      {4, 12}, {4, 14}, {4, 16}, {8, 11}, {8, 12}, {11, 12}, {11, 14}, {12, 14}, {14, 16}, {16,
      17} } )
```

Graph 9: an undirected unweighted graph with 17 vertices and 39 edge(s) **(29)**

```
> DrawRankedGraph(GraphOne1344221)
```




```
>
>
>
```

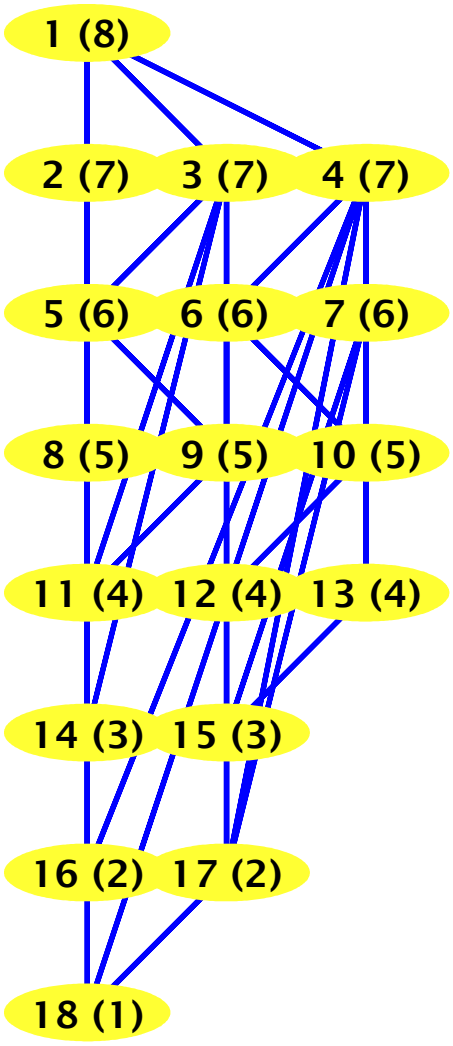
```
> GraphOne13333221 := Graph(18)
```

GraphOne13333221 := Graph 10: an undirected unweighted graph with 18 vertices and 0 edge (s) **(30)**

```
> AddEdge( GraphOne13333221, { {1, 2}, {1, 3}, {1, 4}, {1, 14}, {1, 16}, {2, 3}, {2, 5}, {2,
      8}, {3, 5}, {3, 9}, {3, 11}, {3, 14}, {4, 16}, {4, 6}, {4, 7}, {4, 17}, {4, 18}, {5, 8}, {5,
      9}, {6, 7}, {6, 12}, {6, 10}, {7, 10}, {7, 13}, {7, 15}, {7, 17}, {8, 9}, {8, 11}, {9, 11},
       {10, 12}, {10, 13}, {11, 14}, {12, 13}, {12, 15}, {13, 15}, {14, 16}, {15, 17}, {16, 18},
       {17, 18} } )
```

Graph 10: an undirected unweighted graph with 18 vertices and 39 edge(s) **(31)**

```
> DrawRankedGraph(GraphOne13333221)
```



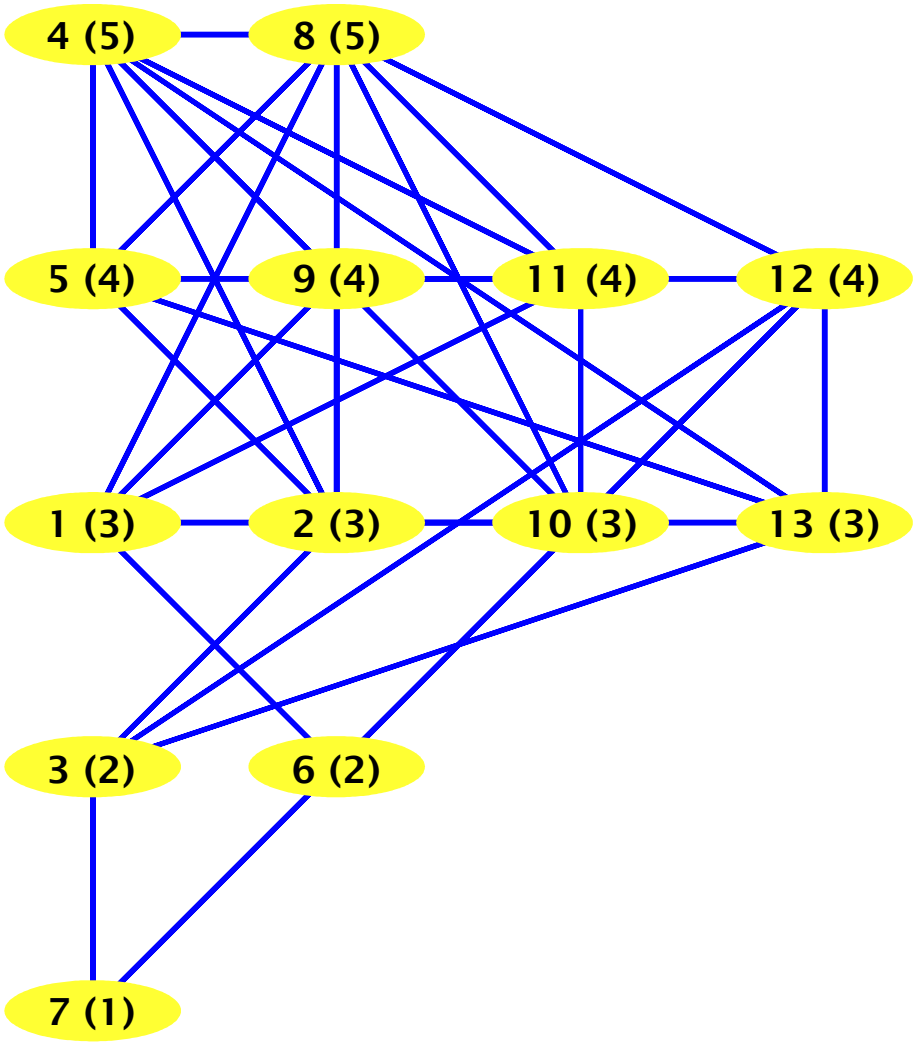

```
>
>
>
> ##################################################################
> ################### BEGIN 0-COP-WIN GRAPHS ###################
> ##################################################################
>
>
>
> GraphZero24421 := Graph(13)
```

GraphZero24421 := Graph 11: an undirected unweighted graph with 13 vertices and 0 edge(s) **(32)**

```
> AddEdge( GraphZero24421, { {4, 8}, {4, 9}, {4, 11}, {4, 5}, {8, 11}, {8, 5}, {8, 12}, {9, 11},
      {5, 12}, {4, 13}, {4, 2}, {8, 2}, {8, 1}, {5, 2}, {9, 2}, {9, 1}, {11, 1}, {5, 13}, {12, 13},
      {12, 10}, {8, 10}, {4, 10}, {11, 10}, {2, 13}, {1, 10}, {2, 3}, {13, 3}, {12, 3}, {1, 6},
      {10, 6}, {12, 6}, {3, 7}, {6, 7}, {12, 7} } )
```

Graph 11: an undirected unweighted graph with 13 vertices and 34 edge(s) **(33)**

```
> DrawRankedGraph(GraphZero24421)
```




```
>
>
>
```

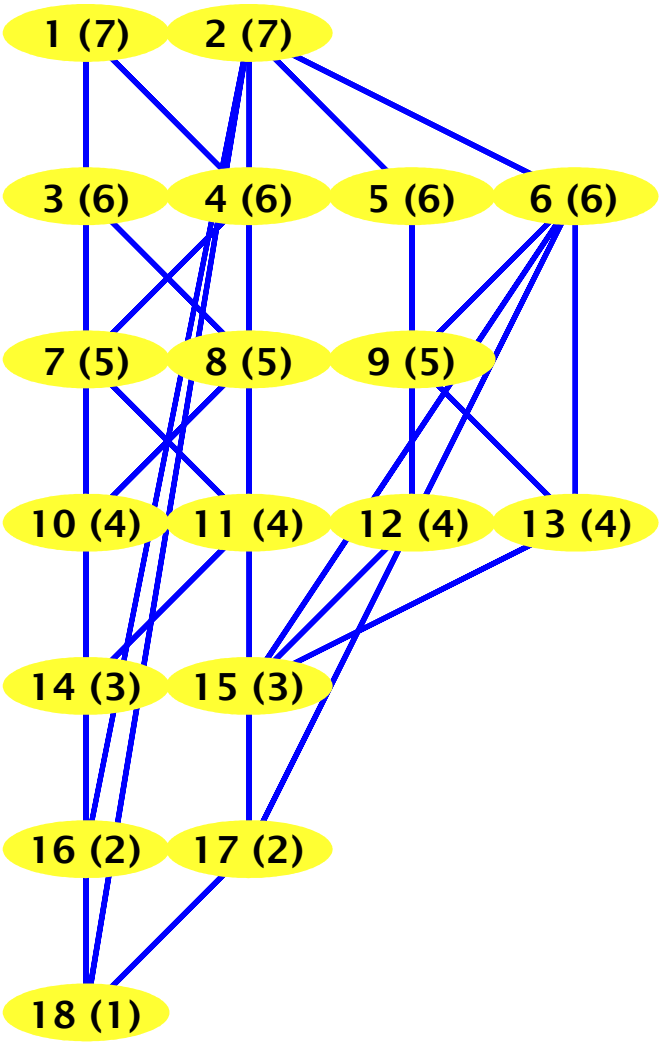
```
> GraphZero2434221 := Graph(18)
```

GraphZero2434221 := Graph 12: an undirected unweighted graph with 18 vertices and 0 edge(s) **(34)**

```
> AddEdge( GraphZero2434221, { {1, 2}, {1, 3}, {1, 4}, {1, 7}, {1, 14}, {1, 16}, {2, 5}, {2, 6},
      {2, 16}, {2, 17}, {2, 18}, {3, 4}, {3, 7}, {3, 10}, {3, 8}, {4, 7}, {4, 8}, {4, 11}, {5, 6},
      {5, 12}, {5, 9}, {6, 9}, {6, 13}, {6, 15}, {6, 17}, {7, 10}, {7, 11}, {7, 14}, {8, 10}, {8,
     11}, {9, 12}, {9, 13}, {10, 14}, {11, 14}, {12, 13}, {12, 15}, {13, 15}, {14, 16}, {15, 17},
      {16, 18}, {17, 18} } )
```

Graph 12: an undirected unweighted graph with 18 vertices and 41 edge(s) **(35)**

```
> DrawRankedGraph(GraphZero2434221)
```




```
>
>
>
```

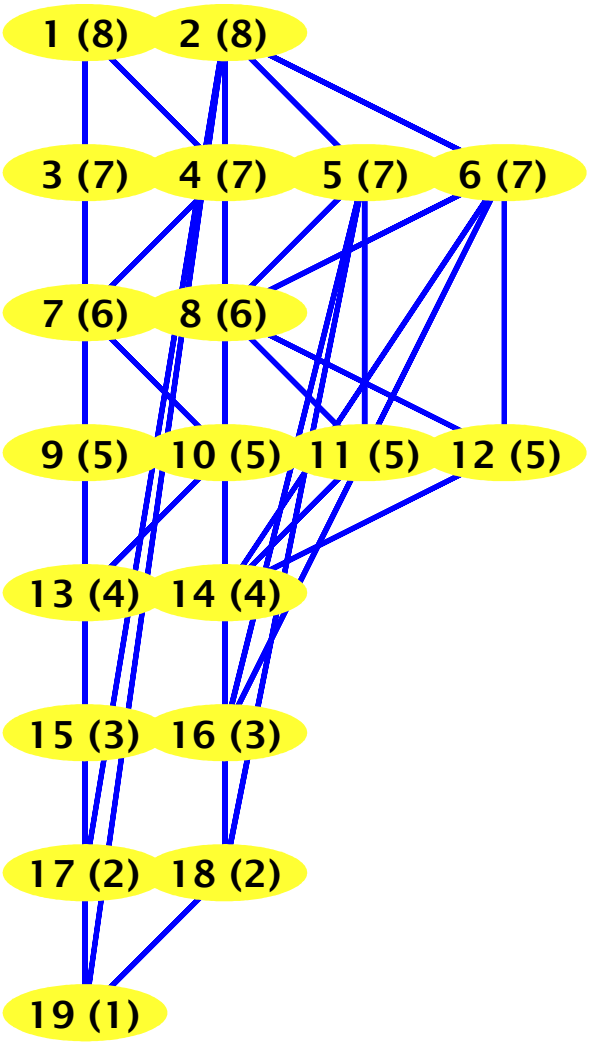
```
> GraphZero24242221 := Graph(19)
```

GraphZero24242221 := Graph 13: an undirected unweighted graph with 19 vertices and 0 edge(s) **(36)**

```
> AddEdge( GraphZero24242221, { {1, 2}, {1, 3}, {1, 4}, {1, 15}, {1, 17}, {2, 5}, {2, 6}, {2,
      17}, {2, 18}, {2, 19}, {3, 4}, {3, 7}, {3, 9}, {3, 13}, {3, 15}, {4, 7}, {4, 10}, {5, 6}, {5,
      8}, {5, 11}, {5, 16}, {5, 18}, {6, 8}, {6, 12}, {6, 14}, {6, 16}, {7, 9}, {7, 10}, {8, 11},
       {8, 12}, {9, 13}, {10, 13}, {11, 14}, {12, 14}, {9, 10}, {11, 12}, {13, 15}, {14, 16},
       {15, 17}, {16, 18}, {17, 19}, {18, 19} } )
```

Graph 13: an undirected unweighted graph with 19 vertices and 42 edge(s) **(37)**

```
> DrawRankedGraph(GraphZero24242221)
```



>

>

>

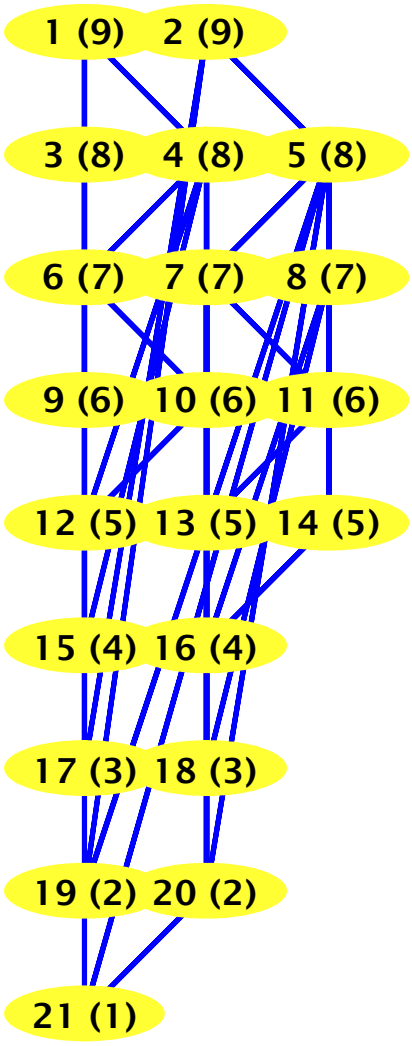
> *GraphZero233332221* := *Graph*(21)

GraphZero233332221 := *Graph 14: an undirected unweighted graph with 21 vertices and 0 edge(s)* **(38)**

> *AddEdge*(*GraphZero233332221*, { {1, 2}, {1, 3}, {1, 4}, {1, 15}, {1, 17}, {2, 5}, {2, 17}, {2, 19}, {3, 4}, {3, 6}, {3, 9}, {4, 6}, {4, 12}, {4, 15}, {4, 10}, {5, 7}, {5, 8}, {5, 19}, {5, 20}, {5, 21}, {6, 9}, {6, 10}, {7, 8}, {7, 13}, {7, 11}, {8, 11}, {8, 14}, {7, 18}, {7, 20}, {8, 16}, {8, 18}, {9, 12}, {10, 12}, {9, 10}, {11, 13}, {11, 14}, {12, 15}, {13, 14}, {13, 16}, {14, 16}, {15, 17}, {16, 18}, {17, 19}, {18, 20}, {19, 21}, {20, 21} })

Graph 14: an undirected unweighted graph with 21 vertices and 46 edge(s) **(39)**

> *DrawRankedGraph*(*GraphZero233332221*)



>

>

>

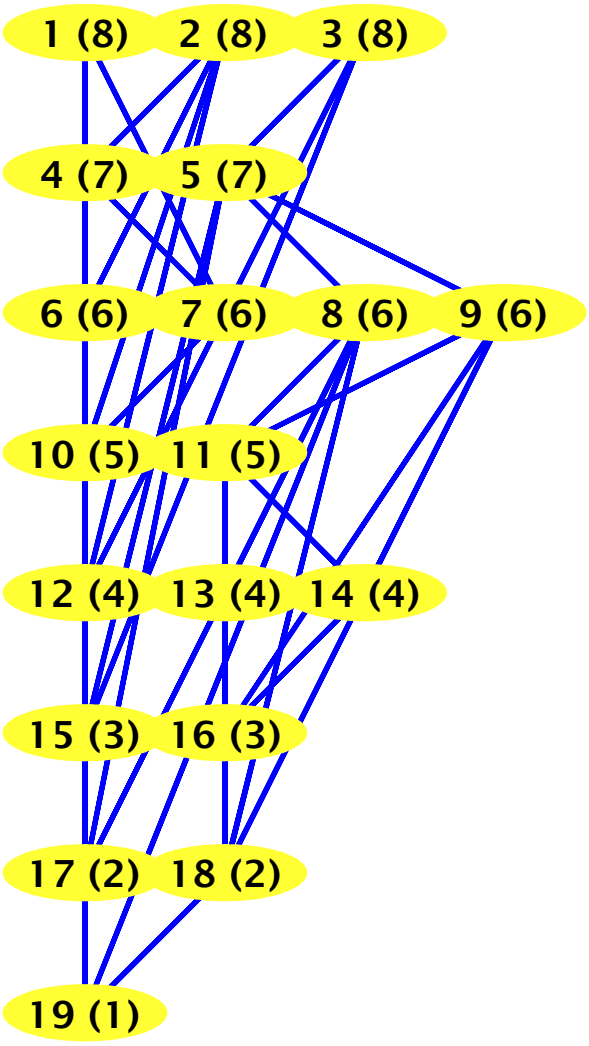
> *GraphZero32423221* := *Graph*(19)

GraphZero32423221 := *Graph 15: an undirected unweighted graph with 19 vertices and 0 edge(s)* **(40)**

> *AddEdge*(*GraphZero32423221*, { {1, 2}, {1, 3}, {1, 4}, {1, 7}, {2, 4}, {2, 3}, {2, 10}, {2, 12}, {2, 6}, {4, 6}, {4, 7}, {6, 7}, {6, 10}, {7, 10}, {10, 12}, {12, 15}, {15, 17}, {17, 19}, {3, 5}, {3, 12}, {3, 15}, {5, 8}, {5, 9}, {5, 15}, {5, 17}, {8, 9}, {8, 13}, {8, 17}, {8, 18}, {8, 19}, {8, 11}, {9, 11}, {9, 14}, {9, 16}, {9, 18}, {11, 13}, {11, 14}, {13, 14}, {13, 16}, {14, 16}, {16, 18}, {18, 19} })

Graph 15: an undirected unweighted graph with 19 vertices and 42 edge(s) **(41)**

> *DrawRankedGraph*(*GraphZero32423221*)



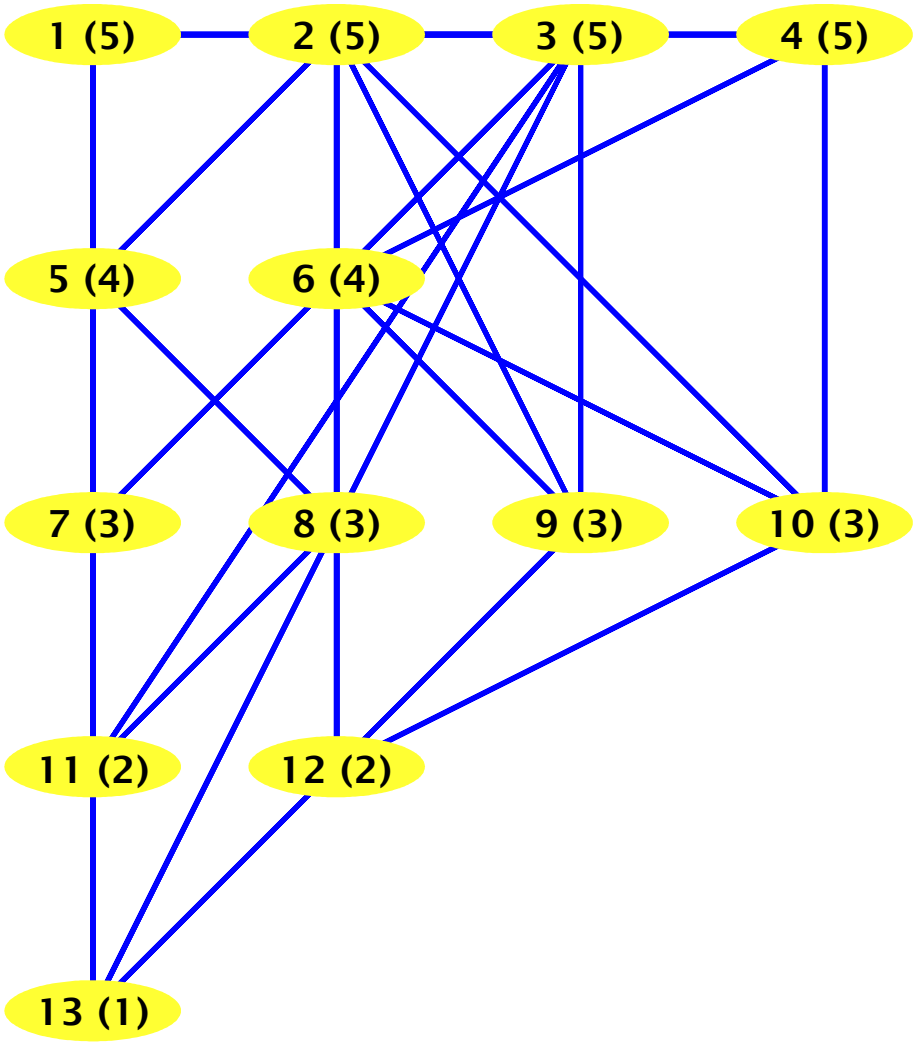

```
>
>
>
> GraphZero42421 := Graph(13)
```

GraphZero42421 := Graph 16: an undirected unweighted graph with 13 vertices and 0 edge(s) **(42)**

```
> AddEdge( GraphZero42421, { {1, 2}, {1, 3}, {1, 4}, {2, 3}, {2, 4}, {3, 4}, {1, 5}, {1, 7}, {2,
      5}, {2, 8}, {2, 9}, {2, 10}, {2, 12}, {5, 7}, {5, 8}, {3, 6}, {3, 9}, {3, 7}, {3, 8}, {3, 11},
       {4, 6}, {4, 10}, {6, 9}, {6, 10}, {7, 11}, {8, 11}, {8, 12}, {9, 12}, {10, 12}, {11, 13},
       {12, 13}, {8, 13} } )
```

Graph 16: an undirected unweighted graph with 13 vertices and 32 edge(s) **(43)**

```
> DrawRankedGraph(GraphZero42421)
```



>

> ########################## *THE END* ##########################