Rooted forests without odd symmetries

Lise Millerjord

March 14, 2019

Abstract

A rooted forest is called even if every automorphism induces an even permutation of the edges. This project aims to determine the number of even rooted forests with n edges. Two different strategies have been adopted, yielding the same results and a surprising link between the number of even rooted trees with n edges and chiral n–ominoes in $(n-1)$–space with one cell marked was observed. This leads to the conjecture that there is a bijection between these different objects and hence a generating function for the even rooted trees. The number of even rooted forests can be found using combinatorics, but a generating function has not yet been obtained.

1 Introduction

In this project, the symmetries of trees and forests are discussed. A rooted forest is called even if every automorphism induces an even permutation of the edges. The aim of this report is to give the details of the strategy adopted to find the number of even rooted forests with n edges and the results obtained. The ultimate goal is to find a generating function of the number of even rooted forests.

To find all the forests matching all of the criteria, two different strategies were adopted:

1. The first strategy is a brute force method of producing all possible forests with n edges and then removing from the list all forests that have odd symmetries. This method is quite resource intensive, but was nevertheless useful as it produces accurate results.

2. The second strategy was an application of combinatorics that is less resource intensive but demands an accurate grasp of how to determine whether or not a forest will have odd symmetries. This method also requires knowledge about the number of rooted trees without odd symmetries that have less than or equal to n edges.

The two methods yielding the same results gives confidence to the results obtained. Further details on the different approaches is included in section **??**.

1.1 Numerical results

The numerical results obtained are shown in Table **??**.

Table 1: Number of even trees and forests with n edges

n	Even trees	Even forests
0	1	1
1	1	1
2	1	1
3	2	3
4	4	7
5	8	15
6	16	32
7	34	74
8	75	175
9	166	405
10	370	939

1.2 Graphical results

Figures of the even rooted trees (up to $n = 6$) and even rooted forests (up to $n = 6$) found are included in Appendix A.1 and Appendix A.2 respectively.

1.3 Generating series

When writing t_n for the numbers on the left hand side of Table **??**, and f_n for those on the right, then the relation between these is

$$\sum_{n=0} f_n x^n = \prod_{m\geqslant 1} \left(1 + (-1)^{m+1} x^m\right)^{(-1)^{m+1} t_m} .$$

This means

$$\begin{aligned} &1 + x + x^2 + 3x^3 + 7x^4 + 15x^5 + 32x^6 + 74x^7 + 175x^8 + 405x^9 + \dots \\ &= \frac{(1+x)(1+x^3)^2(1+x^5)^8(1+x^7)^{34}(1+x^9)^{166}\dots}{(1-x^2)(1+x^4)^4(1+x^6)^{16}(1+x^8)^{75}\dots} . \end{aligned}$$

1.4 Connection to n–ominoes

To see if these integer sequences have appeared before or are known, the *The On-Line Encyclopedia of Integer Sequences* [**?**] was used. The integer sequence for even trees found one match (for the sequence up to $n = 10$ as this is all the data we have): A045648, and it corresponded to the number of n–ominoes

in $(n-1)$–space with one cell labelled. The integer sequence for even forests found no matches.

These results lead to the following conjecture:

Conjecture 1 *There exists a bijection between the even rooted trees with n edges and the chiral n–ominoes in $(n-1)$–space with one cell labelled [**?**].*

Definition 1 *An n–**omino** is an object consisting of exactly n cells. A **chiral** object is asymmetric in such a way that the object and its mirror image are not superimposable. **Chiral** n–**ominoes in** $(n-1)$–**space with one cell labelled**[**?**] are therefore objects with exactly n $(n-1)$–dimensional cells, glued together at their $(n-2)$–dimensional boundaries, in $(n-1)$–space, where one of the cells is distinguished from the others and where the object and its mirror image are not superimposable.*

The one-dimensional cells are lines, the two-dimensional cells are squares and the three-dimensional cells are cubes. A 4–omino will therefore consist of four cubes glued together at the sides (one of the six squares that make up its boundary), existing in 3–space.

Acknowledgements

I would like to express my special thanks and gratitude to my advisor, Markus, for all his patience and support. The time spent in fruitful discussion has been invaluable to my progress. Also thanks to the department for providing this opportunity through the StudForsk project.

2 Background

It is assumed that the reader has some knowledge of basic abstract algebra and other elementary branches of mathematics, but this section intends to define and explain the specific notation and terms used in this project. Some elementary definitions:

2.1 Graphs

Definition 2 *A **graph** is a pair (V, E) consisting of a set V of vertices and a set E of edges. Edges are pairs of exactly two distinct vertices and are visualised as connecting these two vertices.*

Definition 3 *A **tree** is a connected, non-cyclic graph, so the path from one vertex to another is unique. A **rooted tree** is a tree in which one vertex is distinguished from the others, the root.*

Definition 4 *A **forest** is a set of trees, that is a graph consisting of one or more trees. A **rooted forest** is a forest in which every tree is rooted.*

Note that there is a canonical bijection between the set of edges and the set of non-roots: it is given by assigning to an edge the vertex that is farther away from the root.

Definition 5 *A* ***child*** *of a specific vertex in a rooted tree is a vertex that is directly connected to this specific vertex by an edge and is farther away from the root than this vertex. In particular, a* ***child of the root*** *is any vertex that is directly connected to the root by an edge.*

2.2 Symmetries

This project concerns itself with symmetries of trees and forests. Some definitions are included to describe what is meant by the different terms in this context.

Definition 6 *An* ***automorphism*** *of a tree or forest is a permutation of the vertices that sends edges to edges and the roots are pointwise preserved.*

Since an automorphism permutes vertices while respecting edges, it is also possible to view it as permuting edges while respecting vertices. In other words, the automorphism group of the tree or forest acts on both the vertices and the edges.

There are two ways of counting the transpositions, as there are two actions defined on the trees, either on the vertices or on the edges.

In this case, the action on edges are of interest and therefore the decision of whether or not a symmetry is even will be based on counting the number of transpositions of edges. Because of the canonical bijection between the set of edges and the set of non-roots, it does not matter.

So we view the automorphism group $\mathrm{Aut}(G)$ of the graph (in our case a tree or a forest) as a subgroup of the symmetry group $\mathrm{Sym}(V)$ of the set of vertices. The symmetry group also acts on the set E of edges.

Definition 7 *A symmetry of a tree or forest is called* ***odd*** *if it induces an odd number of transpositions on the edges. It is called* ***even*** *if it induces an even number of transpositions on the edges.*

In order to check if a tree or forest is even, its automorphism group must be inspected.

Definition 8 *A tree or forest is called* ***even*** *if all its symmetries are even, that is it has no odd symmetries.*

In particular, we say that the graph G is even if the composition

$$\mathrm{Aut}(G) \longrightarrow \mathrm{Sym}(E) \xrightarrow{\text{sign}} \{\pm 1\}$$

is trivial.

It is known how many rooted trees and forests there are. But what happens if we restrict the symmetries of these? This project aims to find the number of even rooted forests, that is forests that have no odd symmetries.

3 Method

The results presented in section ?? were obtained by implementing the approaches explained in this section.

The open-source software and programming language GAP [?] was used to perform computations and implement the algorithms described in this report. All computation is performed using GAP [?] and the code used for implementing the first two approaches is included in Appendix B.

The three strategies are the following:

3.1 Method for finding even rooted trees

In order to find all even rooted trees with exactly n, the algorithm described by Beyer and Hedetniemi [?] was used to produce all trees with exactly n vertices. We have that a tree with exactly n vertices has exactly $n-1$ edges and hence the list generated by using the Beyer Hedetniemi procedure for $n-1$ vertices will give us all rooted trees with n edges.

The automorphism group for each of the trees produced is inspected. If it contains any automorphism that induces an odd number of transpositions of the edges of the tree, the tree is discarded. Otherwise, the tree is even and it is hence added to the list of even rooted trees with n edges.

3.2 Method for finding even rooted forests

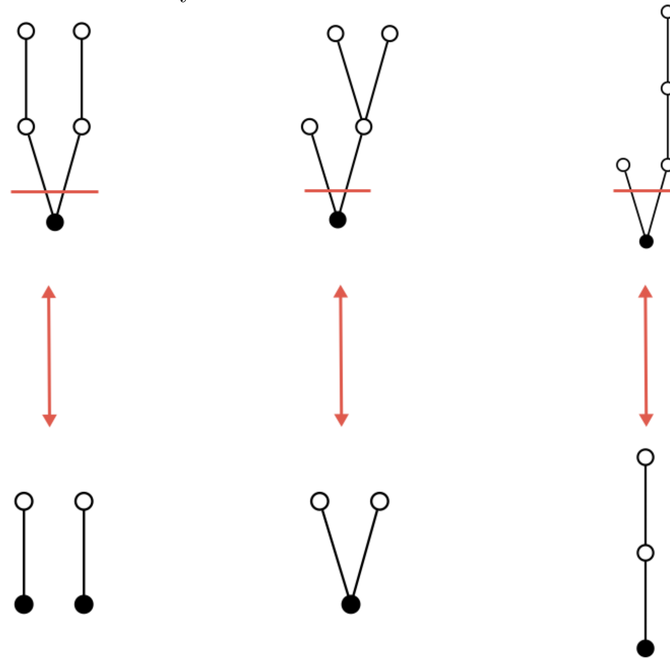
To find all even rooted trees with n edges, first all rooted trees with $2n$ edges are found. From this list, all trees for which the root has exactly n children, that is the trees have exactly n vertices next to the root, are chosen. For each of these trees do the following:

- Cut off the root and its edges. Now we are left with a forest containing exactly n edges since we cut off exactly n edges in this process.
- Make all the vertices next to the original root a new root if they have any children. This means if the new root is the root of a tree containing nothing but the root, we ignore it, but if this root has children, then we keep it.
- We now have a forest containing at most n trees with exactly n edges in total.
- Lastly, the automorphism group of this forest is computed. If there is no automorphism of the forest that induces an odd number of permutations of edges, the forest is even rooted and hence added to the list of even rooted forests. Otherwise, the forest is discarded.

At this point, the list of even rooted forests is complete because there is a bijection between all forests with n edges and all trees with $2n$ edges where

the root has exactly n children. This bijection is shown in figure **??** for forests with 2 edges.

Figure 1: The bijection between forests with 2 edges and trees with 4 edges where the root has exactly 2 children.



3.3 A combinatorial approach for finding even rooted forests

Wilson [**?**] has described a combinatorial approach that uses the fact that forests are made up of one or more trees, and that the sum of the edges of the individual trees is the number of edges of the forest. This approach requires that we already know the number of even rooted trees of with i edges for $1 \leq i \leq n$. The approach consist of the following steps:

1. Find the maximum number of trees the forest can consist of: An even rooted forest of n edges can contain at most $\lceil \frac{n}{2} \rceil$ trees.

   For $n = 4$, the maximum number of trees the forest can contain is 2.

2. Find the combinations of sizes of trees (that is, the number of edges for each tree) that will make a forest of size n.

   For $n = 4$ we have $0 + 4 = 4, 1 + 3 = 4, 2 + 2 = 4$.

3. Count how many forests there are for each of the combinations in the previous step and sum these.

For $n = 4$ we have $0 + 4$ gives $1 \times 4 = 4$ different forests (since there is 1 tree with 0 edges and 4 trees with 4 edges), $1 + 3$ gives $1 \times 2 = 2$ different forests and $2 + 2$ gives $1 \times 1 = 1$ forest. $4 + 2 + 1 = 7$ forests in total.

In the original method presented by Wilson [**?**], there was one adjustment when the combinations from step 2 had repeated terms: When the repeated term is odd, because we cannot have two of the same forest with an odd number of edges, the binomial coefficient is used: If we have $3 + 3 = 6$, we have $\binom{2}{2} = 1$ forest.

This method apparently leads to an error when the repeated term is even. When $4 + 4 = 8$ we do not have $4 \times 4 = 16$ forests as this would mean counting some forests twice. In fact, this is a combination with repetition and hence if we have n objects and should choose r objects with repetition and order does not matter, we have $\binom{n+r-1}{r}$ different choices. So for $n = 8$ and the combination $4 + 4$ we have $\binom{4+2-1}{2} = \binom{5}{2} = 10$ forests. This explains the discrepancy between our table and the one in Wilson for $n = 8$.

A Appendix

A.1 Trees

This appendix contains figures of rooted trees up to $n = 6$ edges.

Figure 2: $n = 1$: 1 even rooted tree with 1 edge.

Figure 3: $n = 2$: 1 even rooted tree with 2 edges.

Figure 4: $n = 3$: 2 even rooted trees with 3 edges.

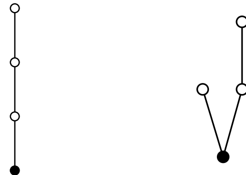


Figure 5: $n = 4$: 4 even rooted trees with 4 edges.

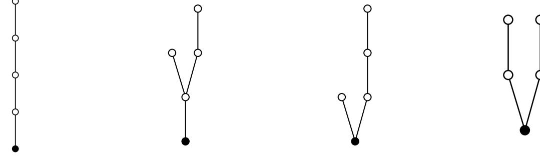


Figure 6: $n = 5$: 8 even rooted trees with 5 edges.

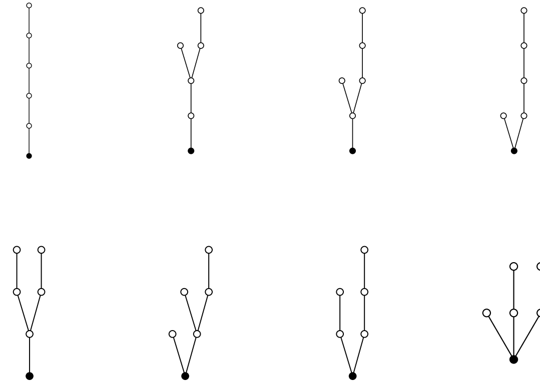
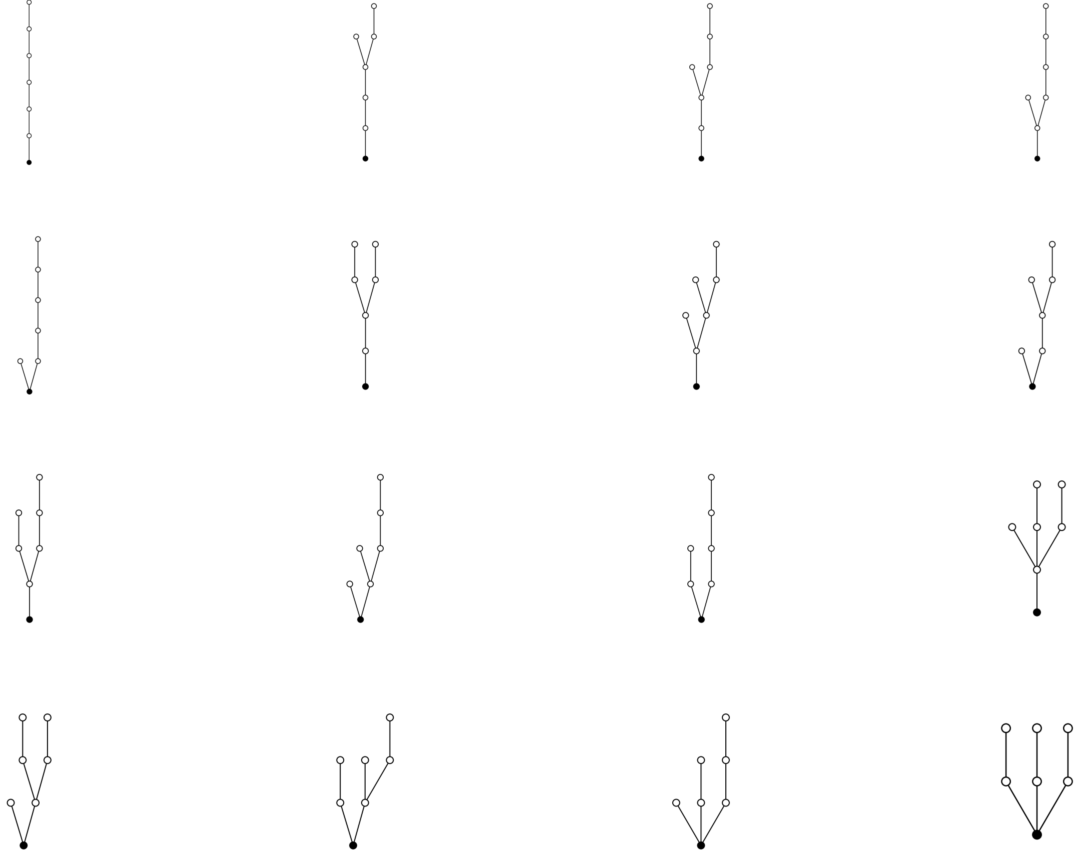


Figure 7: $n = 6$: 16 even rooted trees with 6 edges.



A.2 Forests

This appendix contains figures of even rooted forests up to $n = 6$ edges.

Figure 8: $n = 1$: 1 even rooted forest with 1 edge.

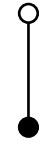


Figure 9: $n = 2$: 1 even rooted forest with 2 edges.

Figure 10: $n = 3$: 3 even rooted forests with 3 edges.

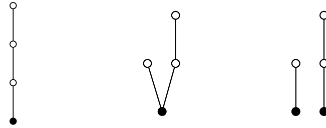


Figure 11: $n = 4$: 7 even rooted forests with 4 edges.

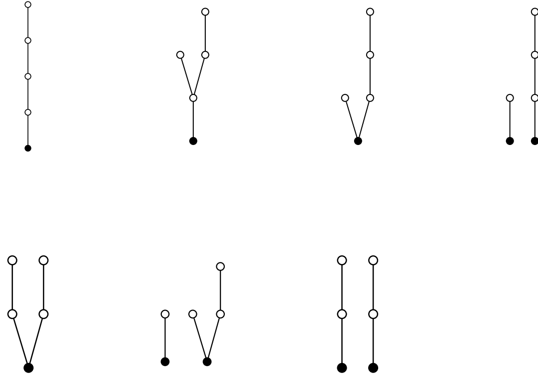


Figure 12: $n = 5$: 15 even rooted forests with 5 edges.

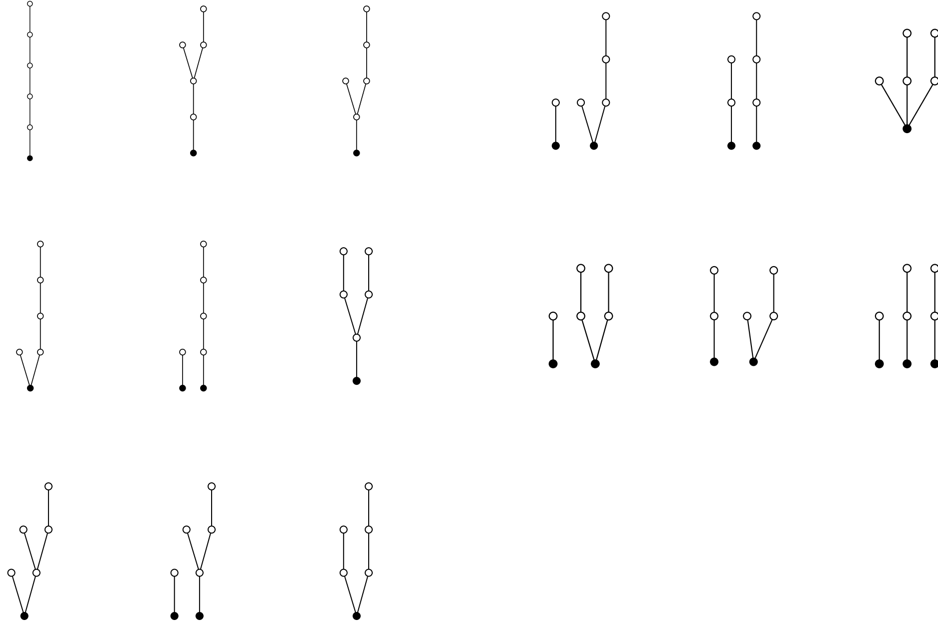
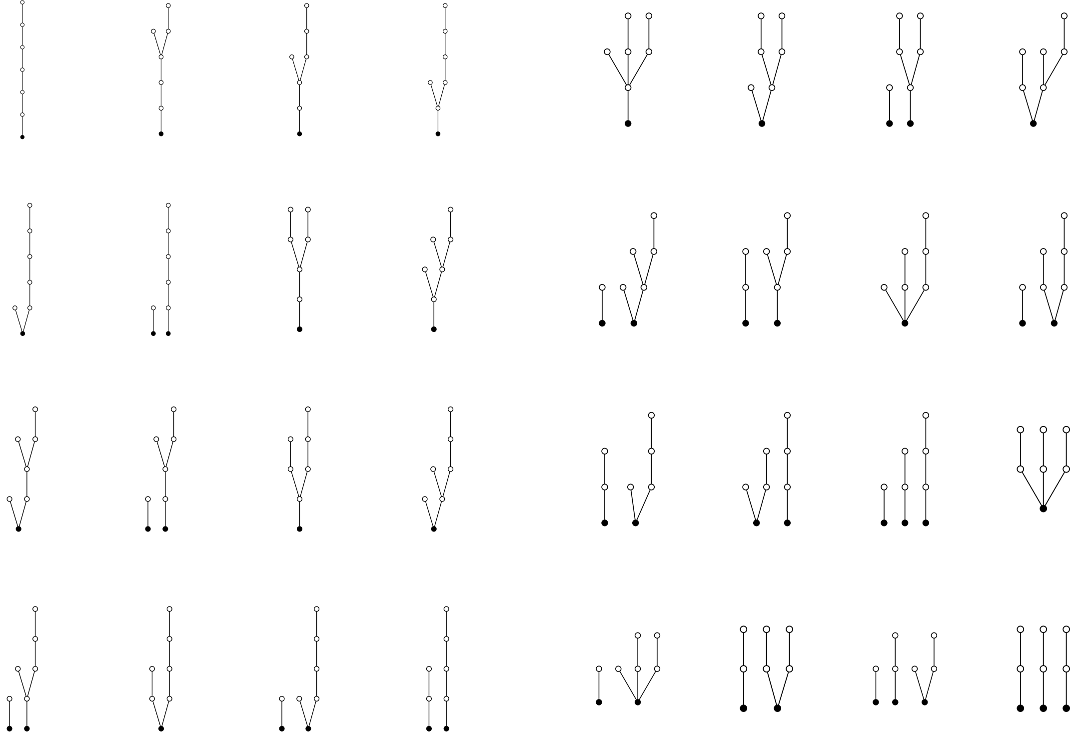


Figure 13: $n = 6$: 32 even rooted forests with 6 edges.



B Appendix

This ontains the GAP code used to produce the number of and list of even rooted trees and forests with n edges.

B.1 Trees

```
LevelSequenceToTree := function( L )
        local T, i, j;
        T := [];
        for i in [1..Length(L)-1] do
                j := i+1;
                while j<=Length(L) and L[i]<L[j] do
                        if L[j] = L[i]+1 then
                                Add(T,[i,j]);
                        fi;
                        j := j+1;
```

```
                od;
        od;
        return( T );
end;

###

BeyerHedetniemi := function ( L )
        local S, T, p, q, i;
        S := L;
        p := Maximum(PositionsProperty( L, l -> l>2 ));
        T := List([1..p], x->L[x]);
         q := Maximum(Positions( T, L[p]-1 ));
        for i in [1..Length( L )] do
        if i>=p then
                        S[i]:=S[i-(p-q)];
                fi;
        od;
        return S;
end;

###

ListOfRootedTrees := function( n )
        local First, Last, Current, Out, L, t;
        L:=[];

        First:=List([1..n],x->x);
        #Print(First,"\n");
        #Print(LevelSequenceToTree(First),"\n\n");
        Append(L,[LevelSequenceToTree(First)]);

        Last:=List([1..n],x->2); Last[1]:=1;

        Current:= First;
        while not Current = Last do
        Current := BeyerHedetniemi (Current);
                #Print(Current,"\n");
                #Print(LevelSequenceToTree(Current),"\n\n");
                t:=LevelSequenceToTree(Current);
                Append(L,[t]);
        od;
        return L;
end;

#Print("\nListOfRootedTrees( n );\n");
```

```
###

NumberOfRootedTrees := function( n )
        return Length( ListOfRootedTrees( n ) );
end;

#Print("\nNumberOfRootedTrees( n );\n");

###

LoadPackage("GRAPE");

TreeFromListOfEdges:= function( L )
        local v, G, e;
        v:=Maximum(Flat(L)); # number of vertices
        G:=NullGraph( Group(()), v );

        for e in L do
                AddEdgeOrbit( G, e );
        od;
        return G;
end;

#Print("\nTreeFromListOfEdges( L );\n");

###

RootedTreeFromListOfEdges:= function( L )
        local T, v;
        T := TreeFromListOfEdges( L );
        v := T.order;
        return rec( graph:=T, colourClasses:=[[1],[2..v]] );
end;

#Print("\nRootedTreeFromListOfEdges( L );\n");

###

ShowRootedTreesAndAutomorphismGroups := function( n )
        local T, t, r, G;
        T:=ListOfRootedTrees( n );
        Print("\n");
        for t in T do
                r := RootedTreeFromListOfEdges( t );
                Print(t,"\n");
```

```
                G:=AutGroupGraph(r);
                Print(G,"\n\n");
        od;
end;

#Print("\nShowRootedTreesAndAutomorphismGroups( n );\n");

###

ContainsOdd := function( G )
        local L;
        L:=List(Elements( G ),g->SignPerm( g ));
        return (-1 in L);
end;

#Print("\nContainsOdd( G );\n");

###

RootedTreesWOS := function( n )
        local T, t, r, number, List;
        List:=[];
        number := 0;
        Print("\n");
        if n=1 then Print("[ [  ] ]\n\n"); return; fi;
        T:=ListOfRootedTrees( n );
        for t in T do
                r := RootedTreeFromListOfEdges( t );
                if not ContainsOdd(AutGroupGraph(r)) then
                        Add(List, t);
                        number := number + 1;
                fi;
        od;
        Print("Number of trees: ");
        Print(number) ;
        Print("\n");
        return List;
end;


IsNodeRoot := function(Edges, node)
        local i;
        if node = 1 then
                return true ;
        fi;
        return false ;
```

```
end ;


NewListOfEdgesToDotNodes := function( Edges )
        local NumberOfNodes, NodesString, Roots, NonRoots, i, rootCheck, vertex;

        NumberOfNodes := Length(Unique(Flat(Edges)));
        NodesString := "" ;

        if NumberOfNodes = 1 then
                return "1 [label=\"\" shape=circle fixedsize=true height=.1
                                               style=filled fillcolor=black];" ;
        fi ;

        Roots := [] ;
        NonRoots := [] ;

        for i in [1..(NumberOfNodes)] do
                if IsNodeRoot(Edges, i) then
                        Add(Roots, i) ;
                else
                        Add(NonRoots, i) ;
                fi ;
        od ;

        for i in [1..(Length(Roots)-1)] do
                vertex := Roots[i];
                Append(NodesString,String(vertex));
                Append(NodesString,", ");
        od ;
        Append(NodesString,String(Roots[Length(Roots)]));
        Append(NodesString," [label=\"\" shape=circle fixedsize=true
                      height=.1 style=filled fillcolor=black];\n");

        for i in [1..(Length(NonRoots)-1)] do
                vertex := NonRoots[i];
                Append(NodesString,String(vertex));
                Append(NodesString,", ");
        od ;
        Append(NodesString,String(NonRoots[Length(NonRoots)]));
        Append(NodesString," [label=\"\" shape=circle fixedsize=true height=.1];\n");

        return NodesString ;
```

```
end;

ListOfEdgesToDotEdges := function( E )
        local EdgesString, e;
        EdgesString := "";
        for e in E do
                Append(EdgesString, String(e[1]));
                Append(EdgesString, " -- ");
                Append(EdgesString, String(e[2]));
                Append(EdgesString, ";\n");
        od;
        return EdgesString;
end;

ListOfEdgesToDotGraph := function( E )
        local BeginGraph, EndGraph, GraphString;
        BeginGraph := "graph {\n";
        EndGraph := "}\n";
        GraphString := "";
        Append(GraphString,BeginGraph);
        Append(GraphString,NewListOfEdgesToDotNodes( E ));
        Append(GraphString,ListOfEdgesToDotEdges( E ));
        Append(GraphString,EndGraph);
        return GraphString;
end;


FileStringBase := "<your_desktop_path>/graph";

NewListOfEdgesToDotFile := function( E, n )
        local FileStringDot;
        FileStringDot:="";
        Append(FileStringDot,FileStringBase);
        Append(FileStringDot,"_");
        Append(FileStringDot,String(n));
        Append(FileStringDot,".dot");
        PrintTo(FileStringDot,ListOfEdgesToDotGraph( E ));
end;

NewListOfEdgesToPdfFile := function( E, n )
        local FileStringDot, FileStringPdf, CommandString;
        NewListOfEdgesToDotFile( E, n );

        FileStringDot:="";
        Append(FileStringDot,FileStringBase);
```

```
        Append(FileStringDot,"_");
        Append(FileStringDot,String(n));
        Append(FileStringDot,".dot");

        FileStringPdf:="";
        Append(FileStringPdf,FileStringBase);
        Append(FileStringPdf,"_");
        Append(FileStringPdf,String(n));
        Append(FileStringPdf,".pdf");

        CommandString:="dot -Tpdf ";
        Append(CommandString,FileStringDot);
        Append(CommandString," -o ");
        Append(CommandString,FileStringPdf);

        Exec(CommandString);
end;


#Print("\nNewListOfEdgesToPdfFile (Edges,ForestNumber) \nProduces the picture of the
#forest represented by its list of edges and given the number it has in the list of all
#the forests.");


# The Trees function takes n and produces the trees with n edges
#that have no symmetries and produces the pictures of these and
#saves them to the Desktop.


Trees := function(n)
        local numberoftrees, i, m, trees ;
        m := n + 1;
        trees := RootedTreesWOS(m) ; #RootedTreesWOS returns a list of the trees
        numberoftrees := Length(trees);

        for i in [1..numberoftrees] do
                NewListOfEdgesToPdfFile(trees[i],i);
        od ;
end ;


Print("\"Trees( n );\" produces the number of trees with n edges
                        and pdf and dot files of these trees.");
```

B.2 Forests

```
# Take the trees of 2*n edges.
#Choose all trees where the root has n children,
#that is it has n vertices in the second position.
#Cut off the root, that is make
#all vertices in the second position roots,
#and give all other vertices their
#current number in the sequence minus 1.
#Each tree will become a forest of n edges
#and the trees of 2*n edges will give all forests of n edges.
#Compute the automorphism groups of each of these forests
#and exclude any that have odd symmetries.
#We are now left with all forests without
#odd symmetries of n edges.

LoadPackage("GRAPE") ; # Need this for functions later

# The BeyerHedetniemi function takes a tree as argument and
# returns the next tree, from the first one which is [1,2,3?n]
# to the last one which is [1,2,2?2].
# It does this one at the time.

BeyerHedetniemi := function ( L )
        local S, T, p, q, i;
        S := L;
        p := Maximum(PositionsProperty( L, l -> l>2 ));
        T := List([1..p], x->L[x]);
         q := Maximum(Positions( T, L[p]-1 ));
        for i in [1..Length( L )] do
                if i>=p then
                        S[i]:=S[i-(p-q)];
                fi;
        od;
        return S;
end;


# The MakeForest function takes a tree as argument and
# Takes away its root and gives all other vertices a lower
# position in the sequence.

MakeForest := function( L )
        local F, i ;
        F := [] ;
```

```
        for i in [1..Length(L)] do
                if not L[i] = 1 then
                        Add(F,(L[i]-1)) ;
                fi ;
        od ;
        return F ;
end ;



# The SequenceToEdges function takes a tree or a forest as
# argument and produces the list of edges in the tree/forest.

SequenceToEdges := function( L )
        local T, i, j;
        T:=[];
        for i in [1..Length(L)-1] do
                j := i+1;
                while j<=Length(L) and L[i]<L[j] do
                        if L[j] = L[i]+1 then
                                Add(T,[i,j]);
                        fi;
                        j := j+1;
                od;
od;
return( T );
end;

# The ForestFromEdges function takes a list of edges as argument
# and returns a forest in the grape format.

ForestFromEdges := function(Edges)
        local vertices, Forest, edge ;

        vertices := Maximum(Flat(Edges)) ; # vertices
        Forest := NullGraph( Group(()), vertices) ;

        for edge in Edges do
AddEdgeOrbit(Forest, edge) ;
        od ;

        return Forest;
end ;

# The isRoot function takes a forest and a vertex as argument
# and checks if the vertex is a root in the forest.
```

```
isRoot := function(Forest, vertex)
        local i ;
        if vertex = 1 then
                return true ;
        fi ;
        for i in [1..(vertex-1)] do
                if IsEdge(Forest,[i,vertex]) then
                        return false ;
                fi ;
        od ;
return true ;
end ;


# The RootedForestFromEdges function takes a list of edges as argument
# and returns a rooted forest in the grape format.

RootedForestFromEdges := function( Edges )
        local Forest, vertices, vertex, rootCheck, Roots, NonRoots;
        Forest := ForestFromEdges(Edges) ;
        vertices := Forest.order ; # number of vertices
        Roots := [] ;
        NonRoots := [] ;
        for vertex in [1..vertices] do
                rootCheck := isRoot(Forest,vertex) ;
                if rootCheck then
                        Add(Roots,vertex) ;
                else
                        Add(NonRoots, vertex) ;
                fi ;
        od ;
        return rec(graph := Forest, colourClasses := [Roots, NonRoots]) ;
end ;


PermutationsOfEdges := function( RootedForest, Edges )
        local PermsofVertices, PermsofEdges, perm, permedEdges, EdgesofForest ;

        PermsofVertices := List(Elements(AutGroupGraph(RootedForest))) ;
        PermsofEdges := [] ;

        for perm in PermsofVertices do
                permedEdges := OnTuplesTuples(Edges,perm) ;
                Add(PermsofEdges,PermListList(Edges,permedEdges)) ;
        od ;
```

```
        return PermsofEdges ;
        # returns a list of the permutations of the edges of the forest.
end ;



# The HasOddSymmetries function takes a list of edges as input and returns
# whether or not the forest has any odd symmetries.

HasOddSymmetries := function( Forest )
        local RootedForest, Edges, SignOfPerms ;

        Edges := SequenceToEdges(Forest) ;
        # Make a forest from the list of edges
        RootedForest := RootedForestFromEdges(Edges) ;

        # Lists the sign of each permutation of the rooted forest:

        SignOfPerms := List(PermutationsOfEdges(RootedForest, Edges), x->SignPerm(x) ) ;

        return -1 in SignOfPerms ;
end ;


# The PrintForest function takes a forest as argument and
# prints the sequence and the list of edges of the forest:

PrintForest := function(forest)

        local edges ;

        Print("The corresponding forest:\n",forest,"\n") ;
        Print(SequenceToEdges(forest),"\n\n") ;


end ;

# The PrintTree function takes a tree as argument and prints
# the sequence and the list of edges of the tree:

PrintTree := function(tree)
        Print("The current tree:\n",tree,"\n") ;
        Print(SequenceToEdges(tree),"\n\n") ;
end ;
```

```
# The RootedForestsWOS function produces all rooted forests with n edges
# that have NO ODD SYMMETRIES.

RootedForestsWOS := function( n )

        local FirstTree, verticesOfTrees, Last, CurrentTree, countForests;
        local CurrentForest, i, no, count, ListofForests, f ;

        ListofForests := [] ;

        no := n ;
        verticesOfTrees := (2*n)+1 ;
        countForests := 0 ;

        FirstTree := List([1..(verticesOfTrees)],x->x) ;

        Last := List([1..(verticesOfTrees)],x->2) ;
        Last[1] := 1 ;

        CurrentTree := FirstTree ;

        while not CurrentTree = Last do

                count := 0 ; # Resets the count of the
# children of the root

                for i in [1..Length(CurrentTree)] do
                        if CurrentTree[i] = 2 then
                                count := count + 1 ;          # Counts the number of
                                                              # children of the root
                        fi ;
                od ;

                if count = n then           # If tree has n children of the
                                            # root, make a forest:

                        # CurrentForest is the sequence of this forest.
                        CurrentForest := MakeForest(CurrentTree) ;

                        #Check if forest has any odd symmetries:
                        if not HasOddSymmetries(CurrentForest) then
                                # This forest has no odd symmetries
                                # Add forest to a list
                                countForests := countForests + 1 ;
```

```
                                #PrintTree(CurrentTree) ;

                                Add(ListofForests, SequenceToEdges(CurrentForest));
                        fi ;
                fi ;

                # You are finished with making a forest (and printing it/adding it
                # if it is appropriate), now create the next tree to do it all again.

                CurrentTree := BeyerHedetniemi (CurrentTree) ;

        od ;


        Print("\nThe number of Forests without odd symmetries with ",n," edges:
                                        " ,countForests,"\n\n") ;


        return ListofForests ;
end ;

IsNodeRoot := function(Edges, node)
        local i;
        if node = 1 then
                return true ;
        fi;
        for i in [1..(node-1)] do
                if ([i,node] in Edges) then
                        return false ;
                fi ;
        od ;
        return true ;
end ;


NewListOfEdgesToDotNodes := function( Edges )
        local NumberOfNodes, NodesString, Roots, NonRoots, i, rootCheck, vertex;

        NumberOfNodes := Length(Unique(Flat(Edges)));
        NodesString := "" ;

        if NumberOfNodes = 1 then
                return "1 [label=\"\" shape=circle fixedsize=true
                                height=.1 style=filled fillcolor=black];" ;
        fi ;
```

```
        Roots := [] ;
        NonRoots := [] ;

        for i in [1..(NumberOfNodes)] do
                if IsNodeRoot(Edges, i) then
                        Add(Roots, i) ;
                else
                        Add(NonRoots, i) ;
                fi ;
        od ;


        for i in [1..(Length(Roots)-1)] do
                vertex := Roots[i];
                Append(NodesString,String(vertex));
                Append(NodesString,", ");
        od ;
        Append(NodesString,String(Roots[Length(Roots)]));
        Append(NodesString," [label=\"\" shape=circle fixedsize=true
                        height=.1 style=filled fillcolor=black];\n");

        for i in [1..(Length(NonRoots)-1)] do
                vertex := NonRoots[i];
                Append(NodesString,String(vertex));
                Append(NodesString,", ");
        od ;
        Append(NodesString,String(NonRoots[Length(NonRoots)]));
        Append(NodesString," [label=\"\" shape=circle fixedsize=true height=.1];\n");

        return NodesString ;




end;

ListOfEdgesToDotEdges := function( E )
        local EdgesString, e;
        EdgesString := "";
        for e in E do
                Append(EdgesString, String(e[1]));
                Append(EdgesString, " -- ");
                Append(EdgesString, String(e[2]));
                Append(EdgesString, ";\n");
        od;
        return EdgesString;
```

```
end;

ListOfEdgesToDotGraph := function( E )
        local BeginGraph, EndGraph, GraphString;
        BeginGraph := "graph {\n";
        EndGraph := "}\n";
        GraphString := "";
        Append(GraphString,BeginGraph);
        Append(GraphString,NewListOfEdgesToDotNodes( E ));
        Append(GraphString,ListOfEdgesToDotEdges( E ));
        Append(GraphString,EndGraph);
        return GraphString;
end;

# note that the filestringbase contains a hardcoded path to your desired location
# e.g. your desktop.
FileStringBase := "<your desktop path>/graph";

NewListOfEdgesToDotFile := function( E, n )
        local FileStringDot;
        FileStringDot:="";
        Append(FileStringDot,FileStringBase);
        Append(FileStringDot,"_");
        Append(FileStringDot,String(n));
        Append(FileStringDot,".dot");
        PrintTo(FileStringDot,ListOfEdgesToDotGraph( E ));
end;

NewListOfEdgesToPdfFile := function( E, n )
        local FileStringDot, FileStringPdf, CommandString;
        NewListOfEdgesToDotFile( E, n );

        FileStringDot:="";
        Append(FileStringDot,FileStringBase);
        Append(FileStringDot,"_");
        Append(FileStringDot,String(n));
        Append(FileStringDot,".dot");

        FileStringPdf:="";
        Append(FileStringPdf,FileStringBase);
        Append(FileStringPdf,"_");
        Append(FileStringPdf,String(n));
        Append(FileStringPdf,".pdf");

        CommandString:="dot -Tpdf ";
        Append(CommandString,FileStringDot);
```

```
        Append(CommandString," -o ");
        Append(CommandString,FileStringPdf);

        Exec(CommandString);
end;


#Print("\n NewListOfEdgesToPdfFile (Edges,ForestNumber) \n
             Produces the picture of the #forest represented by its list of edges
                   and given the number it has in the list of all #the forests.");


# The Forests function takes n and produces the forests with n edges
         that have no symmetries and produces the pictures of these
         and saves them to the Desktop (or your chosed location).

Forests := function(n)
        local Edges, numberofforests, i ;
        Edges := RootedForestsWOS(n) ; #RootedForestsWOS returns a list of the forests
        numberofforests := Length(Edges);

        for i in [1..numberofforests] do
                NewListOfEdgesToPdfFile(Edges[i],i);
        od ;
end ;


Print("\"Forests( n );\" produces the number of forests with n edges
                                        and pdf and dot files with these forests.");
```

References

[GAP] The GAP Group. GAP – Groups, Algorithms, and Programming. www.gap-system.org

[OEIS] *The On-Line Encyclopedia of Integer Sequences*, Published electronically at https://oeis.org, https://bit.ly/2FbWAmw, Retrieved: 2018, Sequence A045648

[Wil] J. Wilson. Private communication. February 6th, 2018.

[BH80] Beyer, T. and Hedetniemi, S.M., 1980. Constant time generation of rooted trees. *SIAM Journal on Computing*, 9(4), pp.706-712.

[Lun75] Lunnon, W. F. Counting multidimensional polyominoes. Comput. J. 18 (1975) 366–367.