
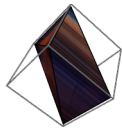


In[*]:= Quit[]

In[*]:= With[{pts = {{0, 0, 0}, {1, 0, 0}, {0, 1, 0}, {1, 0, 1}, {0, 1, 1}, {1, 1, 1}}, fcs =
 {{1, 2, 3}, {4, 5, 6}, {1, 2, 4}, {1, 3, 5}, {2, 3, 6}, {1, 4, 5}, {3, 5, 6}, {2, 4, 6}}},
 Graphics3D[{Opacity[0.9], Texture[, Polyhedron[pts[[Sequence@#]] & /@ fcs,
 VertexTextureCoordinates -> (pts[[Sequence@#]] & /@ fcs)]}]]

Out[*]=



In[*]:= With[{pts = {{0, 0, 0}, {1, 0, 0}, {0, 1, 0}, {1, 0, 1}, {0, 1, 1}, {1, 1, 1}},
 fcs = {{1, 2, 3}, {4, 5, 6}, {1, 2, 4}, {1, 3, 5}, {2, 3, 6},
 {1, 4, 5}, {3, 5, 6}, {2, 4, 6}}}, pts[[Sequence@#]] & /@ fcs]

Out[*]=

```
{{{0, 0, 0}, {1, 0, 0}, {0, 1, 0}}, {{1, 0, 1}, {0, 1, 1}, {1, 1, 1}},  

  {{0, 0, 0}, {1, 0, 0}, {1, 0, 1}}, {{0, 0, 0}, {0, 1, 0}, {0, 1, 1}},  

  {{1, 0, 0}, {0, 1, 0}, {1, 1, 1}}, {{0, 0, 0}, {1, 0, 1}, {0, 1, 1}},  

  {{0, 1, 0}, {0, 1, 1}, {1, 1, 1}}, {{1, 0, 0}, {1, 0, 1}, {1, 1, 1}}}
```

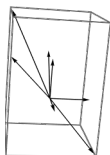
In[*]:= Inverse[{{1, 0, 1}, {0, 1, 1}, {1, 1, 1}}^T]

Out[*]=

```
{{0, -1, 1}, {-1, 0, 1}, {1, 1, -1}}
```

In[*]:= Graphics3D[Arrow[{{0, 0, 0}, #}] & /@ (Tuples[{0, 1}, 3].%32)]

Out[*]=



Are there any arrangement acheiving maximum product except for cube-with-basis? achNumber counts number of such arrangements given a non-degenerate 0-1 matrix -- basis of the second set in the coordinates dual to some basis of the second set. If maximum is only acheived with cube+ba-sis, the function will return 2

In[*]:= \$HistoryLength = 1

Out[*]=

1

```

In[*]:= achNumber[B01_List] := With[{Cpl = Complement, Len = Length,
  d = Length@B01, cube = Tuples[{0, 1}, Length@B01], inv = Inverse[B01]},
  With[{acan = Select[Subsets[Cpl[cube, B01]], Divisible[2d (d + 1), (Len@# + d)] &]},
    bcan = Select[Subsets[Cpl[cube, B01T]], Divisible[2d (d + 1), (Len@# + d)] &]},
    Total@Flatten@Table[Boole[Cpl[Flatten[Join[B01, a].inv.(Join[B01T, b]T)],
      {0, 1}] == {}], {a, acan}, {b, Select[bcan, Len@# ==  $\frac{2^d (d + 1)}{d + \text{Len@a}} - d$  &]}]]]

```

Test on all invertible 2 x2 matrices :

```

In[*]:= Tally[achNumber /@ Select[Tuples[{0, 1}, {2, 2}], MatrixRank@# == 2 &]] // AbsoluteTiming
Out[*]= {0.0036872, {{2, 6}}}

```

Test on all invertible 3x3 matrices:

```

In[*]:= Tally[achNumber /@ Select[Tuples[{0, 1}, {3, 3}], MatrixRank@# == 3 &]] // AbsoluteTiming
Out[*]= {0.140387, {{2, 174}}}

```

Number of 4x4 bases ignoring order:

```

In[*]:= Select[DeleteDuplicates[Sort /@ Tuples[{0, 1}, {4, 4}]], MatrixRank@# == 4 &]] // Length
Out[*]= 940

```

```

In[*]:= achNumber[IdentityMatrix[4]] // AbsoluteTiming
Out[*]= {15.8582, 2}

```

```

In[*]:= 35 *  $\frac{940}{3600}$ .
Out[*]= 9.13889

```

```

In[*]:= SeedRandom[42];
fourByFourSample = Transpose /@ RandomSample[
  Select[DeleteDuplicates[Sort /@ Tuples[{0, 1}, {4, 4}]], MatrixRank@# == 4 &], 4]
Out[*]= {{{0, 0, 1, 1}, {0, 1, 0, 0}, {1, 0, 0, 1}, {1, 0, 1, 0}},
  {{0, 0, 0, 1}, {0, 1, 1, 0}, {0, 0, 1, 1}, {1, 1, 0, 0}},
  {{0, 0, 0, 1}, {0, 0, 1, 1}, {0, 1, 0, 0}, {1, 1, 1, 0}},
  {{0, 0, 1, 1}, {0, 1, 0, 1}, {0, 1, 1, 0}, {1, 1, 1, 0}}}

```

```

In[*]:= (achNumber /@ fourByFourSample) // AbsoluteTiming
Out[*]= {565.683, {2, 2, 2, 2}}

```

```

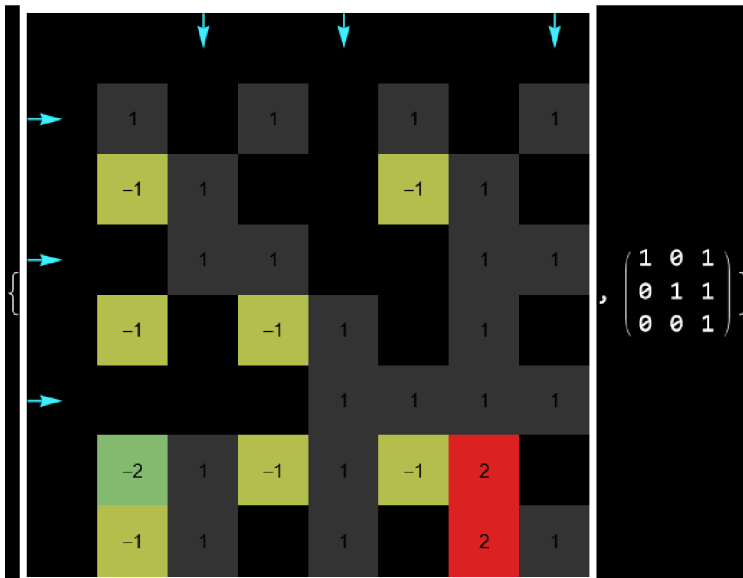
In[ ]:= matrixPicture[B01_] :=
Module[{d = Length@B01, cube, final, pos1, pos2}, cube = Tuples[{0, 1}, d];
pos1 = Flatten[Position[cube, #] & /@ (B01^T)];
pos2 = Flatten[Position[cube, #] & /@ B01];
final = cube.Inverse@B01.(cube^T);
Style[{ArrayPlot[final, ColorRules -> {0 -> Black, 1 -> White},
ColorFunction -> {ColorData["Rainbow",  $\frac{\# + 1}{2}$ ] &},
Epilog -> {Black, MapIndexed[Text[#1, Reverse[#2 - 0.5]] &, Reverse[final], {2}],
■, Arrow[{ $\{\# - 0.5, 2^d\}$ ,  $\{\# - 0.5, 2^d - \frac{1}{2}\}$ ]}] & /@ pos1,
Arrow[{ $\{0, 2^d - \# + 0.5\}$ ,  $\{0.5, 2^d - \# + 0.5\}$ ]}] & /@ pos2},
ImageSize -> 300, Frame -> None, Background -> White, PlotRangePadding -> 0.1],
Style[B01 // MatrixForm, White]}, White, Background -> Black]]

```

```

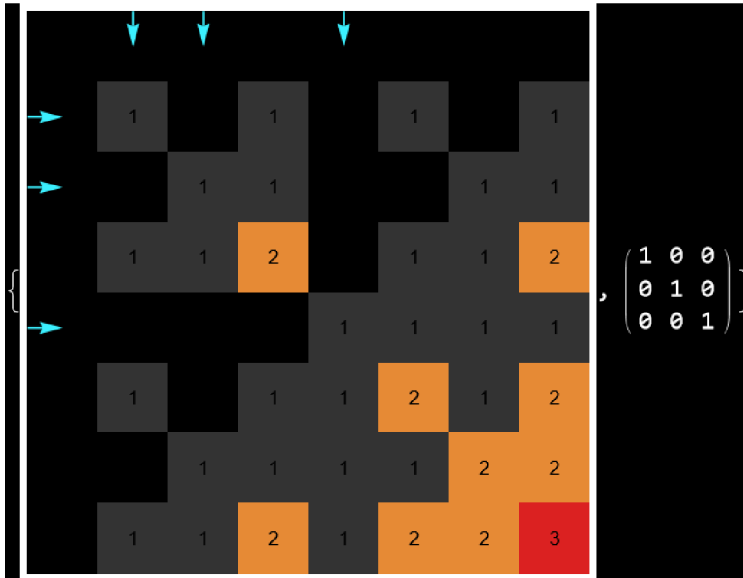
In[ ]:= matrixPicture[{{1, 0, 1}, {0, 1, 1}, {0, 0, 1}}] // Rasterize
Out[ ]:=

```



```
In[*]:= matrixPicture[IdentityMatrix@3] // Rasterize
```

```
Out[*]=
```



```
In[*]:= matrixPicture@First[
  RandomChoice[Select[Tuples[{0, 1}, {3, 3}], MatrixRank@# == 3 &], 1]] // Rasterize
imlist = Rasterize /@ matrixPicture /@ Select[Tuples[{0, 1}, {3, 3}], MatrixRank@# == 3 &];
```

```
In[*]:= matrixPicture@IdentityMatrix[4] // Rasterize
```

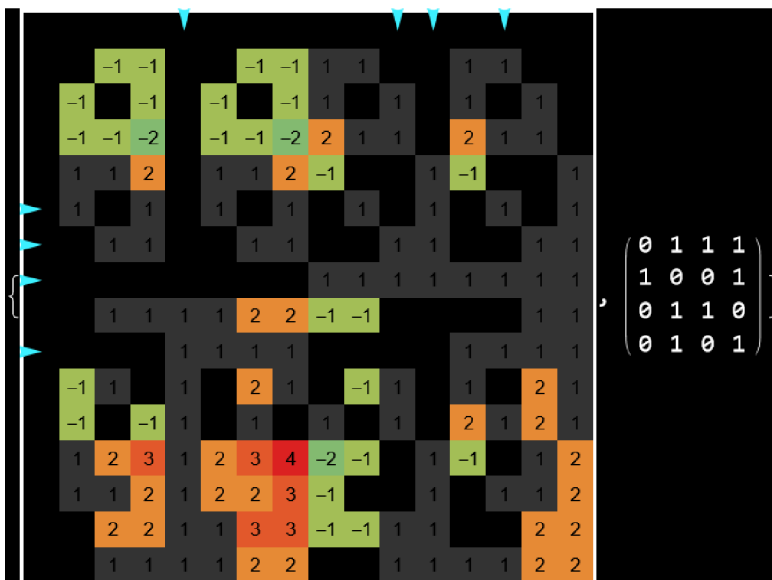
```
In[*]:= NestWhile[RandomChoice[{0, 1}, {4, 4}] &, {{0}}, MatrixRank[#] != 4 &]
```

```
Out[*]=
```

```
{{0, 1, 1, 1}, {1, 0, 0, 1}, {0, 1, 1, 0}, {0, 1, 0, 1}}
```

```
In[*]:= matrixPicture@% // Rasterize
```

```
Out[*]=
```



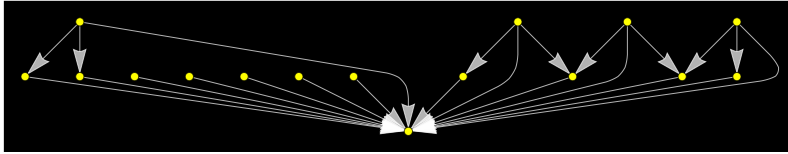
```
In[*]:= Inverse@{{0, 1, 1, 1}, {1, 0, 0, 1}, {0, 1, 1, 0}, {0, 1, 0, 1}}
```

```
Out[*]=
```

```
{{-1, 1, 1, 0}, {-1, 0, 1, 1}, {1, 0, 0, -1}, {1, 0, -1, 0}}
```

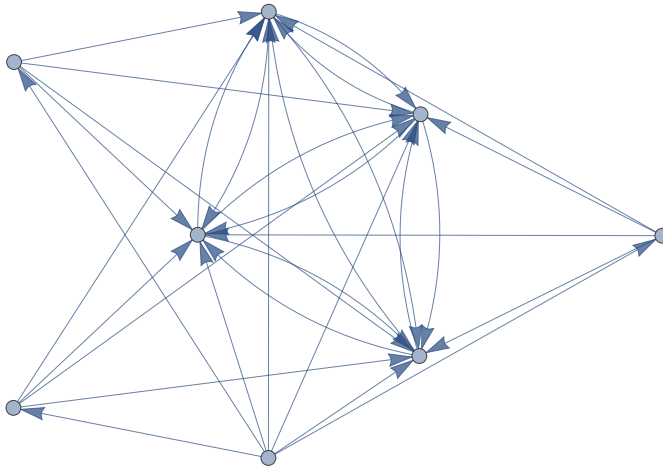
```
In[ ]:= Graph[graphPicture@{{0, 1, 1, 1}, {1, 0, 0, 1}, {0, 1, 1, 0}, {0, 1, 0, 1}},
  Background → Black, VertexStyle → Yellow, EdgeStyle → White]
```

```
Out[ ]:=
```



```
Graph[graphFullSym@IdentityMatrix[3]]
```

```
Out[ ]:=
```



Check whether sets of “skewed dot products” coincide with all nonbinary inputs treated as same and no distinction between 0 and 1:

```
In[ ]:= coincidingSets[B01_] := Module[{d = Length@B01, cube, final}, cube = Tuples[{0, 1}, d];
  final = cube.Inverse@B01.(cube^T) /. {1 → 0, x_ /; x ≠ 0 → 2};
  Map[Sort, final, {0, -2}] == Map[Sort, final^T, {0, -2}]]
```

Check on invertible $d \times d$ matrices for d from 2 to 4:

```
In[ ]:= Table[
  And @@ coincidingSets /@ Select[Tuples[{0, 1}, {d, d}], MatrixRank[#] == d &], {d, 2, 4}]
```

```
Out[ ]:=
```

```
{True, True, True}
```

```
In[ ]:= randomInvertible[d_] :=
  NestWhile[RandomChoice[{0, 1}, {d, d}] &, {{0}}, MatrixRank[#] ≠ d &]
```

```
In[ ]:= randomInvertible[5]
```

```
Out[ ]:=
```

```
{{1, 1, 1, 0, 1}, {1, 0, 0, 1, 0}, {0, 1, 0, 0, 0}, {1, 0, 0, 1, 1}, {1, 0, 0, 0, 1}}
```

```
In[ ]:= NotebookDelete[temp]
```

```
Out[ ]:=
```

```
NotebookDelete[temp]
```

```

In[*]:= cnt = 0;
SeedRandom[42];
NestWhile[If[Mod[cnt, 1] == 0, NotebookDelete[temp];
temp = PrintTemporary[cnt]];
cnt++;
randomInvertible[4] &, {{1}}, coincidingSets[#] &]

Out[*]:=
$Aborted

In[*]:= counterexample5 =
{{0, 0, 1, 1, 1}, {0, 1, 0, 0, 0}, {1, 0, 0, 1, 1}, {1, 1, 0, 1, 0}, {1, 0, 0, 0, 0}}

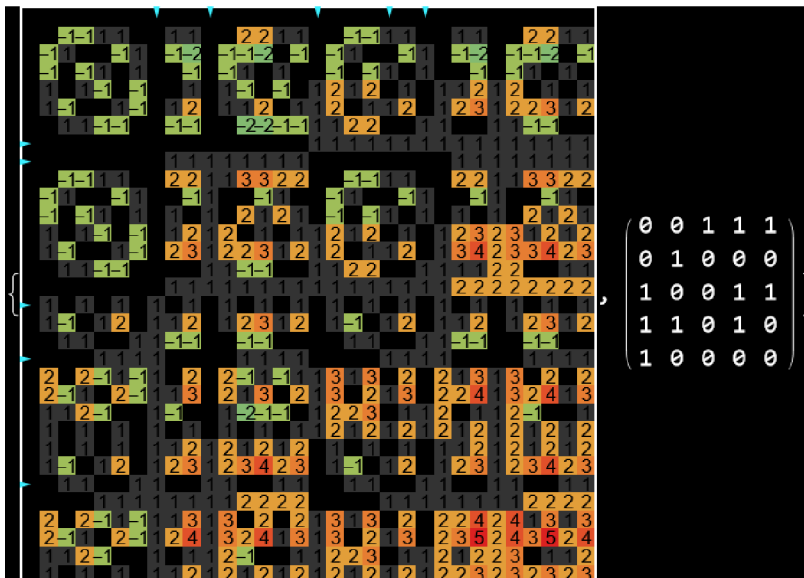
Out[*]:=
{{0, 0, 1, 1, 1}, {0, 1, 0, 0, 0}, {1, 0, 0, 1, 1}, {1, 1, 0, 1, 0}, {1, 0, 0, 0, 0}}

In[*]:= coincidingSets@counterexample5

Out[*]:=
False

In[*]:= matrixPicture@counterexample5 // Rasterize
Out[*]:=

```



```

In[*]:= cube01[d_] := Tuples[{0, 1}, d]

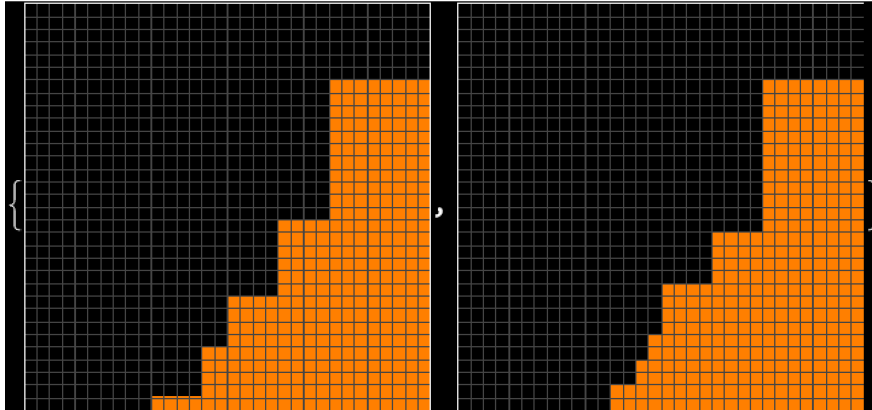
```

```

In[ ]:= Style[With[{cube = Tuples[{0, 1}, 5]},
  ArrayPlot[Map[Sort, (cube.Inverse[#].(cube^T)) /. {1 -> 0, x_ /; x != 0 -> 2}, {0, -2}],
  Frame -> True, FrameStyle -> White, Background -> Black, PlotRangePadding -> None,
  Mesh -> True, MeshStyle -> Directive[■, Thickness[10^-4]],
  ColorRules -> {0 -> Black, 2 -> Orange}}] & /@
  {counterexample5, counterexample5^T} // Rasterize

```

Out[]:=



```

In[ ]:= Inverse@counterexample5

```

Out[]:=

```

{{0, 0, 0, 0, 1}, {0, 1, 0, 0, 0}, {1, 0, -1, 0, 1}, {0, -1, 0, 1, -1}, {0, 1, 1, -1, 0}}

```

Function to produce graph of allowable relations given symmetric invertible 01 matrix:

```

In[ ]:= graphFullSym[B01_] := Module[{d = Length@B01, cube, matr}, cube = Tuples[{0, 1}, d];
  matr = (cube.Inverse@B01.(cube^T)) /. {1 -> 0, x_ /; x != 0 -> 2};
  Flatten@Table[If[i != j && And@@ NonNegative[matr[[i]] - matr[[j]]],
    cube[[i]] -> cube[[j]], Nothing], {i, 2^d}, {j, 2^d}]

```

All symmetric invertible (without row permutations!):

```

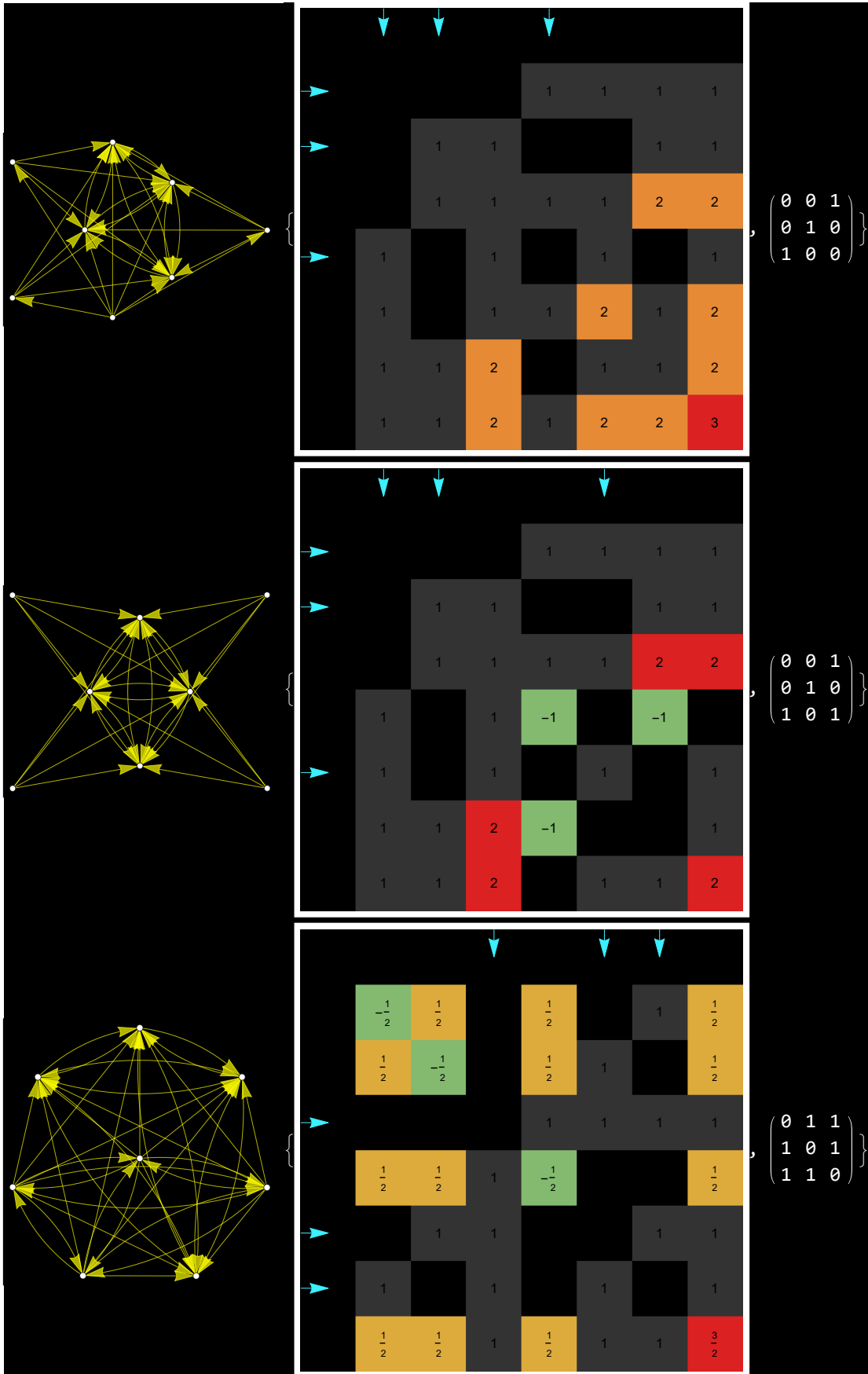
In[ ]:= allSymInvertible[d_] :=
  DeleteDuplicatesBy[Select[(# + LowerTriangularize[#^T, -1]) & /@
    PadLeft /@ (Internal`PartitionRagged[#, Range[d, 1, -1]] &) /@
    Tuples[{0, 1},  $\frac{d(d+1)}{2}$ ], MatrixRank@# == d &], Sort]

```

```

In[ ]:= Style[
  Grid@DeleteDuplicates[{Graph[graphFullSym@#, Background -> Black, VertexStyle -> White,
    EdgeStyle -> Yellow], matrixPicture@#} & /@ allSymInvertible[3],
  IsomorphicGraphQ[#1[[1]], #2[[1]]] &], Background -> Black, White]

```

$Out[*]=$



```
In[*]:= Inverse@
$$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$$

```

```
Out[*]= {{-1, 0, 1}, {0, 1, 0}, {1, 0, 0}}
```

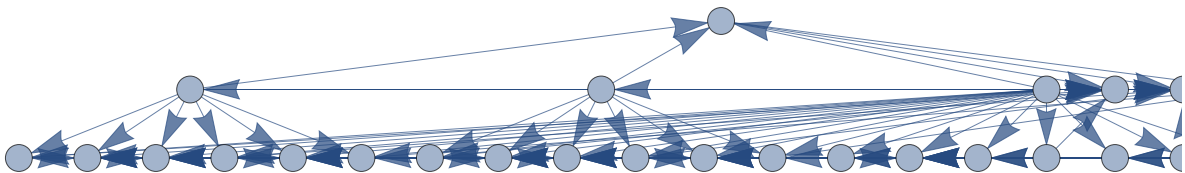
```
In[*]:= Style[
  Grid@DeleteDuplicates[{Graph[graphFullSym@#, Background → Black, VertexStyle → White,
    EdgeStyle → Yellow], matrixPicture@#} & /@ allSymInvertible[4],
    IsomorphicGraphQ[#1[[1]], #2[[1]]] &], Background → Black, White]
```

```
In[*]:= {Graph[graphFullSym[counterexample5]], Graph[graphFullSym[counterexample5T]]}
```

```
In[*]:= graphSymNoSinks[B01_] := Module[{d = Length@B01, filtcube, matr, pos},
  filtcube = Rest@DeleteCases[Tuples[{0, 1}, d], Alternatives@@B01];
  matr = (filtcube.Inverse@B01.(filtcubeT)) /. {1 → 0, x_ /; x ≠ 0 → 2};
  Flatten@Table[If[i ≠ j && And@@NonNegative[matr[[i]] - matr[[j]]],
    filtcube[[i]] → filtcube[[j]], Nothing], {i, 2d - d - 1}, {j, 2d - d - 1}]]
```

```
In[*]:= Graph[graphSymNoSinks@IdentityMatrix@5, GraphLayout → "LayeredEmbedding"]
```

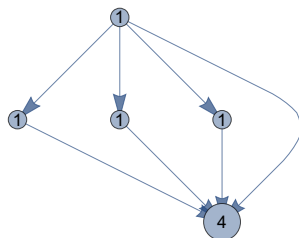
```
Out[*]=
```



```
In[*]:= graphSymFactorizedAndSzs[B01_] :=
  Module[{d = Length@B01, cube, grps, filt = (# /. {1 → 0, x_ /; x ≠ 0 → 2} &)},
    cube = Tuples[{0, 1}, d];
    grps = {#, filt[#[[1]]]} & /@ GatherBy[cube.Inverse@B01.(cubeT), filt];
    {Flatten@Table[If[i ≠ j && And@@NonNegative[grps[[i], 2] - grps[[j], 2]],
      grps[[i], 1] → grps[[j], 1], Nothing], {i, Length@grps}, {j, Length@grps}},
      Normal@AssociationMap[Length, grps[[All, 1]]]}
```


```
In[*]:= With[{gsf = graphSymFactorizedAndSzs[IdentityMatrix[3]]},
  Graph[First@gsf, VertexLabels → MapAt[Placed[#, Center] &, gsf[[2]], {All, 2}],
    VertexSize → MapAt[{ "Scaled",  $\frac{\text{Sqrt}[\#]}{20}$  } &, gsf[[2]], {All, 2}]]]
```

```
Out[*]=
```



```
In[*]:= Get["https://raw.githubusercontent.com/szhorvat/IGraphM/master/IGInstaller.m"]
```

The currently installed versions of IGraph/M are: {}

Installing IGraph/M is complete: PacletObject[ Name: IGraphM
Version: 0.6.5].

It can now be loaded using the command << IGraphM`

In[]:= << IGraphM`

Out[]:=

IGraph/M 0.6.5 (December 21, 2022)
Evaluate IGDokumentation[] to get started.

Remove[Global`IGVertexMap]

Remove[Global`IGVertexProp]

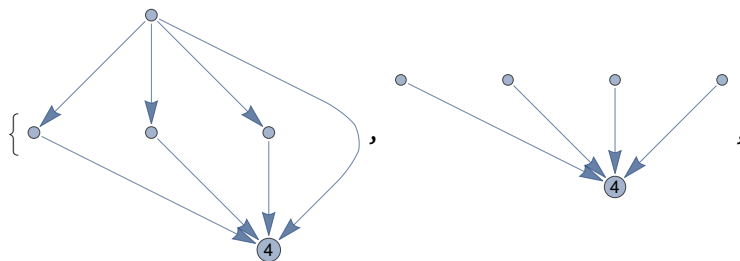
In[]:= graphSymFactorizedRaw[B01_] :=
Module[{d = Length@B01, cube, grps, filt = (# /. {1 → 0, x_ /; x ≠ 0 → 2} &)},
cube = Tuples[{0, 1}, d];
grps = {#, filt[#[[1]]]} & /@ GatherBy[cube.Inverse@B01.(cube^T), filt];
Flatten@Table[If[i ≠ j && And@@NonNegative[grps[[i, 2]] - grps[[j, 2]]],
grps[[i, 1]] → grps[[j, 1]], Nothing], {i, Length@grps}, {j, Length@grps}]]
hasseDiagram[gr_] := With[{f = EdgeQ[gr, #1 → #2] &, s = VertexList[gr]},
ResourceFunction["HasseDiagram"][f, s, PerformanceGoal → "Quality"]]

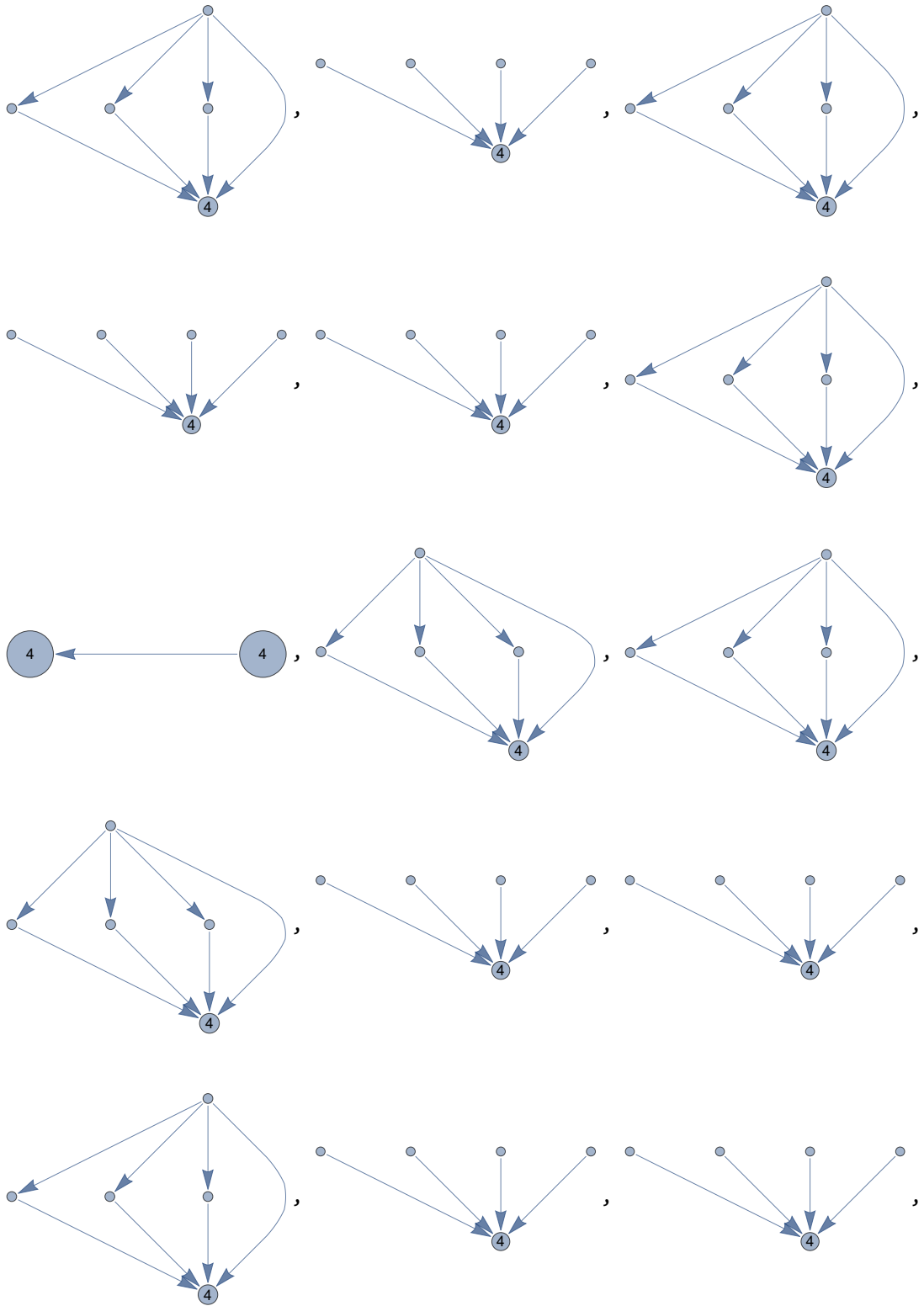
In[]:= addGraphProps[gr_] := Module[{filt = (# /. {1 → 0, x_ /; x ≠ 0 → 2} &)},
gr // IGVertexMap[Sqrt@*Length, "SqLength" → VertexList] /*
IGVertexMap[Length, "Length" → VertexList] /* IGVertexMap[filt[#[[1]]] &,
"Filter" → VertexList] /* IGVertexMap[MatrixForm[#^T] &, Tooltip → VertexList]]
grSymFact[B01_] :=
addGraphProps[Graph[graphSymFactorizedRaw[B01], PerformanceGoal → "Quality"]]
hasseSymFact[B01_] := addGraphProps[
hasseDiagram@Graph[graphSymFactorizedRaw[B01], PerformanceGoal → "Speed"]]

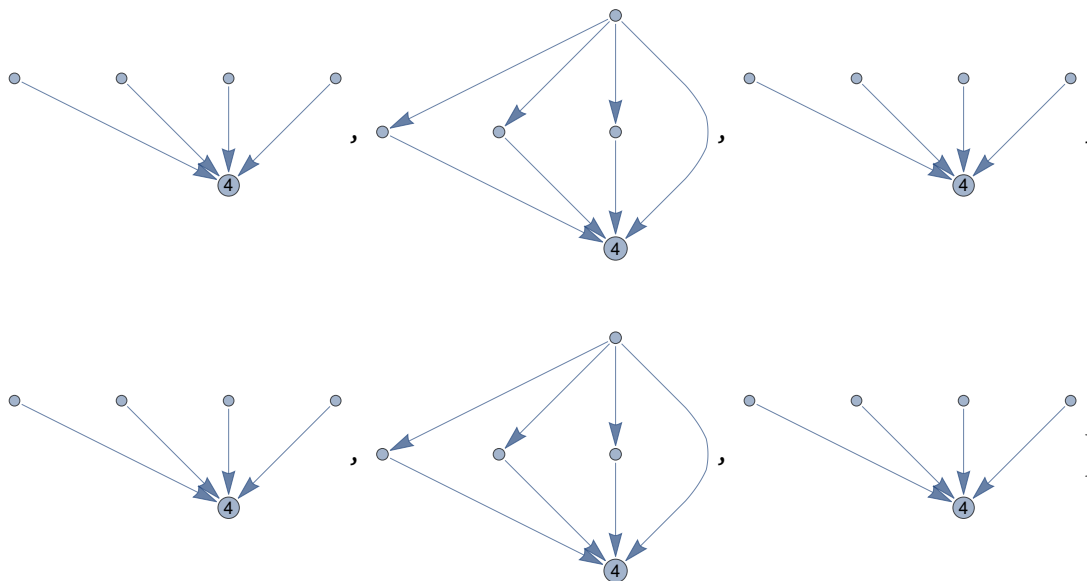
In[]:= dispFact[gr_] :=
gr // IGVertexMap[0.1 # &, VertexSize → IGVertexProp["SqLength"]] /* IGVertexMap[
Placed[If[# == 1, "", #], Center] &, VertexLabels → IGVertexProp["Length"]]

In[]:= dispFact[grSymFact[#]] & /@ allSymInvertible[3]

Out[]:=

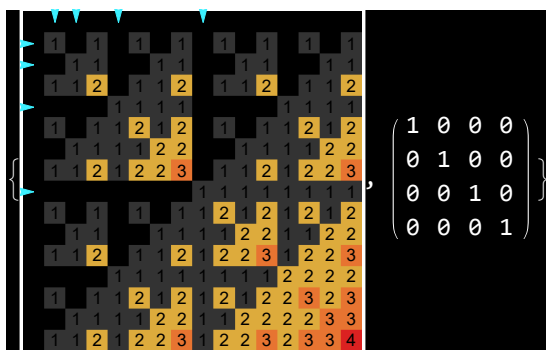






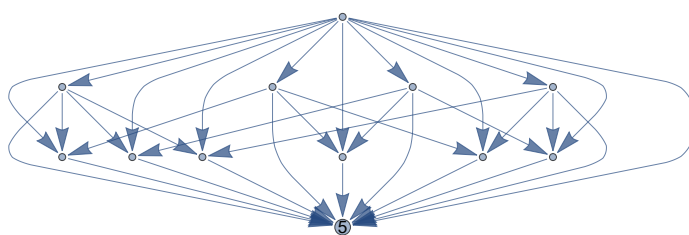
```
In[ ]:= matrixPicture@IdentityMatrix[4]
```

```
Out[ ]:=
```



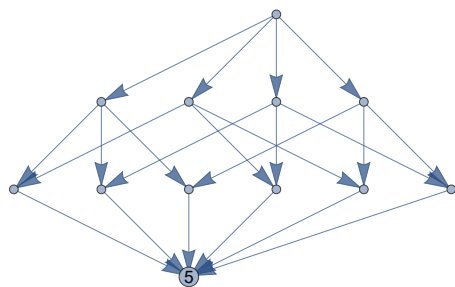
```
In[ ]:= dispFact[grSymFact[IdentityMatrix[4]]]
```

```
Out[ ]:=
```



```
In[ ]:= dispFact[hasseSymFact[IdentityMatrix@4]]
```

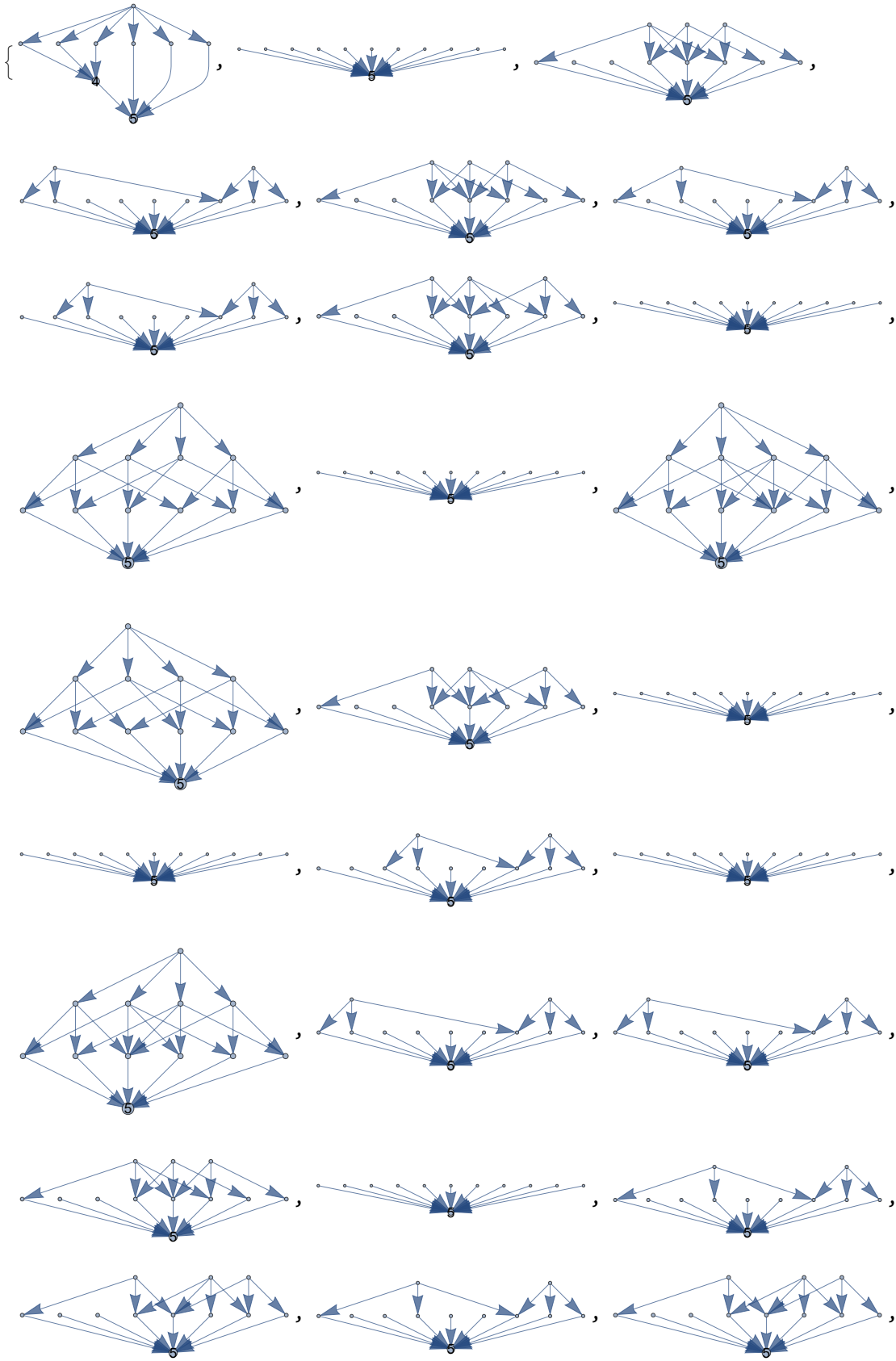
```
Out[ ]:=
```

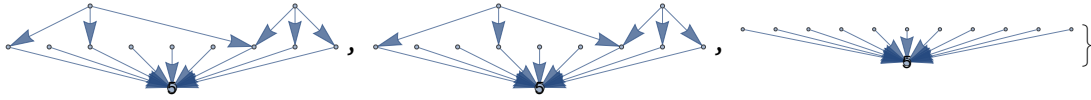


```

In[ ]:= (dispFact@*hasseSymFact) /@ RandomSample[allSymInvertible[4], 30]
Out[ ]:=

```



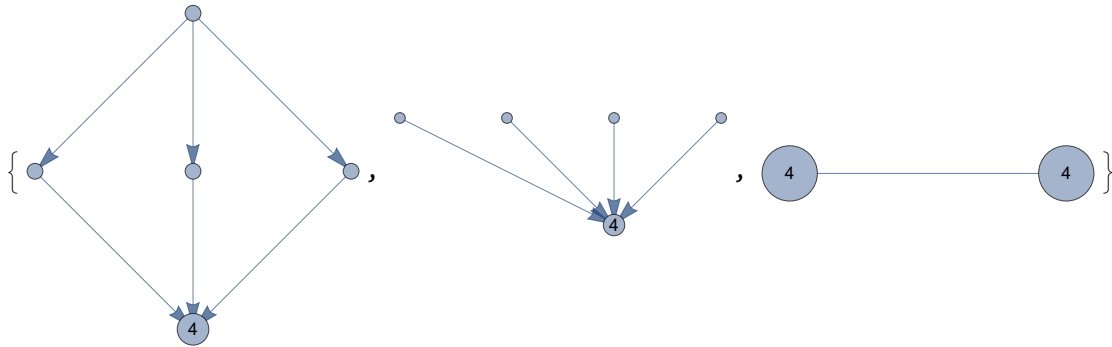


```
In[*]:= deepSort[expr_] := Map[Sort, expr, {0, -2}]
```

```
In[*]:= allSymInvertibleDiffGraph[d_] :=  
  First /@ DeleteDuplicates[{#, Graph@graphFullSym@#} & /@ allSymInvertible[d],  
    IsomorphicGraphQ[#1[[2]], #2[[2]]] &]
```

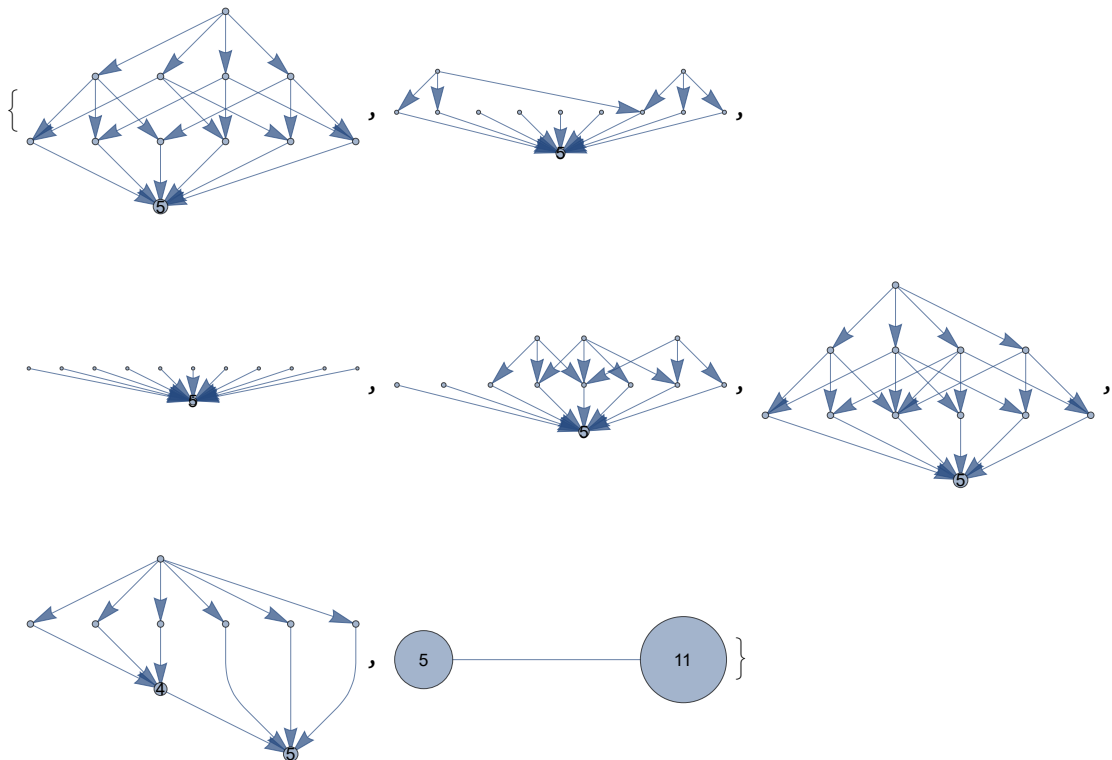
```
In[*]:= allThreeHasse = dispFact /@ hasseSymFact /@ allSymInvertibleDiffGraph[3]
```

```
Out[*]:=
```



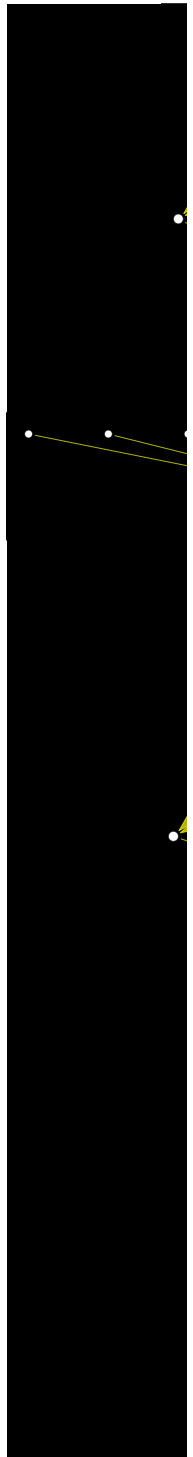
```
In[*]:= allFourHasse = dispFact /@ hasseSymFact /@ allSymInvertibleDiffGraph[4]
```

```
Out[*]:=
```



```
In[*]:= setGraphStyle[graph_Graph] := Graph[graph, EdgeStyle -> Yellow,  
  VertexStyle -> White, Background -> Black, PerformanceGoal -> "Quality"]
```

```
Grid[ArrayReshape[setGraphStyle /@ (allFourHasse ~ Join ~ allThreeHasse), {5, 2}],
      Background → Black]
```



```
In[*]:= Export[FileNameJoin[{NotebookDirectory[], "All-Hasse-3-4-diagrams.jpg"}],
```

```
In[*]:= allHasse5withDup = ParallelMap[hasseSymFact, allSymInvertible[5]];
```

```
In[*]:= allFiveHasse = DeleteDuplicates[allHasse5withDup, IsomorphicGraphQ];
```

```
Grid[ArrayReshape[setGraphStyle /@ (allFourHasse ~ Join ~ allThreeHasse), {5, 2}],
      Background → Black]
```

```
In[ ]:= DumpSave[FileNameJoin[{NotebookDirectory[], "all-sym-hasse-3.mx"}], allThreeHasse];
DumpSave[FileNameJoin[{NotebookDirectory[], "all-sym-hasse-4.mx"}], allFourHasse];
DumpSave[FileNameJoin[{NotebookDirectory[], "all-sym-hasse-5.mx"}], allFiveHasse];
```

```
In[ ]:= MapIndexed[Export[FileNameJoin[{NotebookDirectory[],
    "all-sym-hasse-5-pictures", ToString[#2[[1]]] <> ".jpg"}], #1] &,
    setGraphStyle[Graph[dispFact@#, ImageSize → 800]] & /@ allFiveHasse];
```

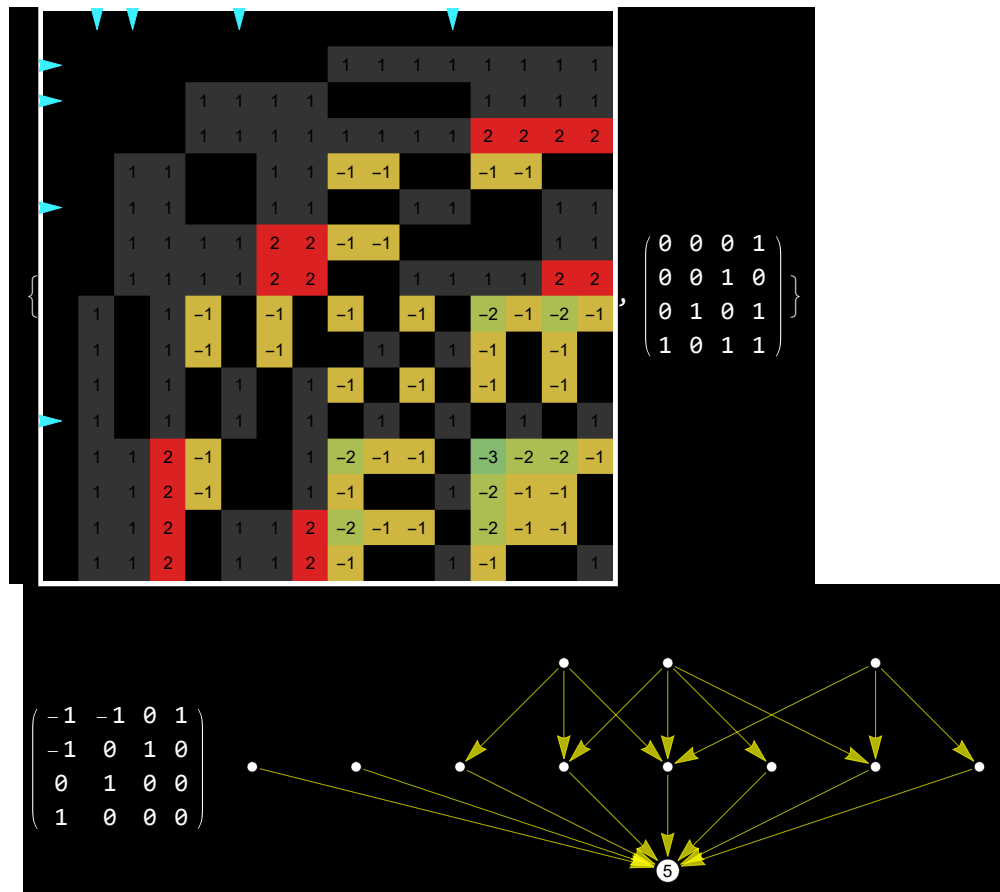
There is an example that is not even a semilattice among 4 by 4 matrices

```
In[ ]:= nonSemilatticecCounter = allSymInvertibleDiffGraph[4][4]
```

```
Out[ ]:= {{0, 0, 0, 1}, {0, 0, 1, 0}, {0, 1, 0, 1}, {1, 0, 1, 1}}
```

```
In[ ]:= Style[{matrixPicture[nonSemilatticecCounter],
    Style[MatrixForm@Inverse@nonSemilatticecCounter, White],
    Graph[setGraphStyle@dispFact@hasseSymFact@nonSemilatticecCounter,
        ImageSize → 400]}, Background → Black]
```

```
Out[ ]:=
```



```
In[ ]:= Export[FileNameJoin[{NotebookDirectory[], "non-semilattice-4.jpg"}], %];
```

```
In[ ]:= With[{c = cube01[4], ctr = Inverse@nonSemilatticecCounter},
    {MatrixForm[c.ctr.(c^T)], MatrixForm@c, MatrixForm@ctr, MatrixForm@(c^T)}]
```

Unlike the simple cube case there might be more than $d+1$ vectors that can lie in intersection (meet) of A and B; the following function counts the number of such vectors.


```
In[*]:= selfCompatibleCount[B01_] :=  
  With[{c = Tuples[{0, 1}, Length@B01]}, Count[Diagonal[c.Inverse[B01].(cT)], 0 | 1]]
```

```
In[*]:= Union[selfCompatibleCount /@ allSymInvertible[3]]  
Out[*]= {4, 6}
```

```
In[*]:= Union[selfCompatibleCount /@ allSymInvertible[4]]  
Out[*]= {5, 6, 8, 9, 10}
```

```
In[*]:= Union[selfCompatibleCount /@ allSymInvertible[5]]  
Out[*]= {6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 24}
```

In case of 4 by 4 matrices numbers of possible self-compatible vectors reduces when only considering matrices that produce non-isomorphic graphs. So there are matrices producing isomorphic graphs but having different-sized self-compatible sets.

A matrix with many self-compatible vectors:

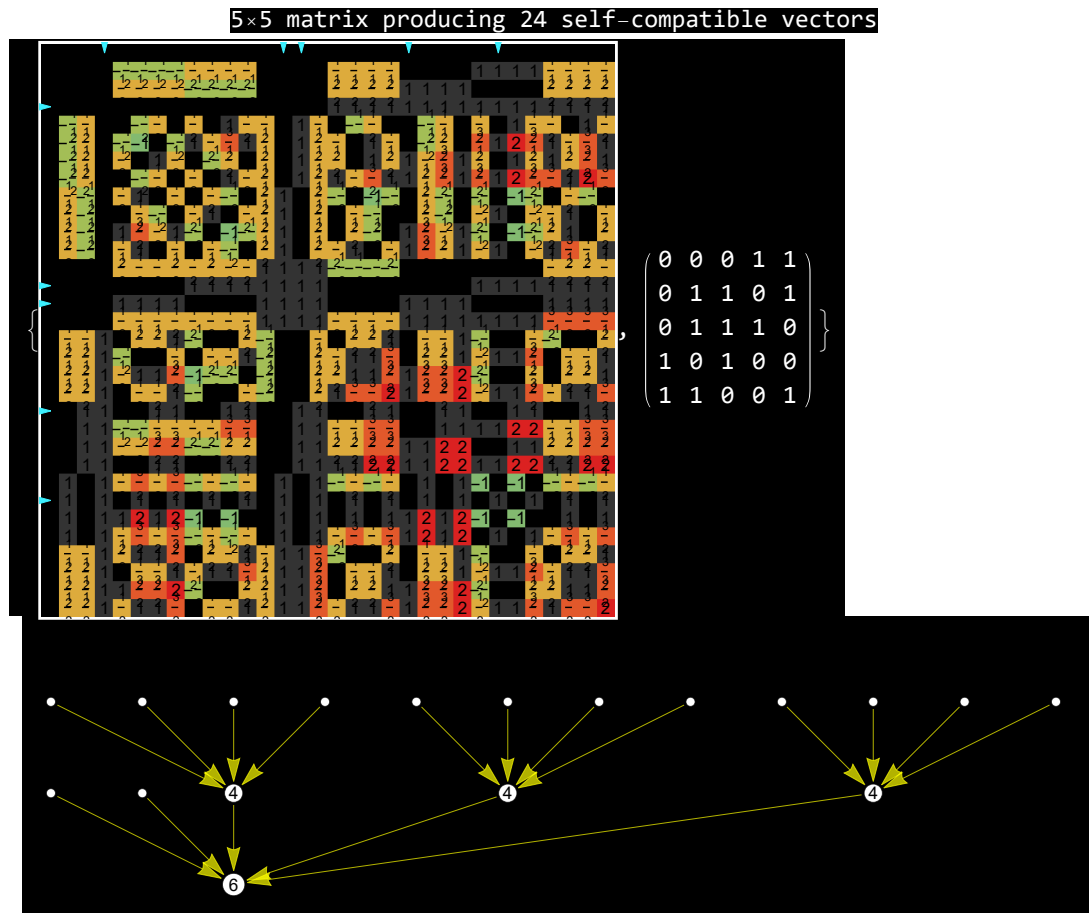
```
In[*]:= manySelfCompatibleExample =  
  Select[allSymInvertible[5], selfCompatibleCount@# == 24 &, 1] // First  
Out[*]= {{0, 0, 0, 1, 1}, {0, 1, 1, 0, 1}, {0, 1, 1, 1, 0}, {1, 0, 1, 0, 0}, {1, 1, 0, 0, 1}}
```

```

In[ ]:= Labeled[Style[{matrixPicture[manySelfCompatibleExample], Graph[
    setGraphStyle@dispFact@hasseSymFact@manySelfCompatibleExample, ImageSize -> 550]},
    Background -> Black], Style["5x5 matrix producing 24 self-compatible vectors",
    White, Background -> Black], Top]

```

Out[]:=



```

In[ ]:= Export[FileNameJoin[{NotebookDirectory[], "many-self-compatible-example.jpg"}],
    %345, Background -> Black];

```

```

In[ ]:= matrixPicture@allSymInvertible[4][[1]]

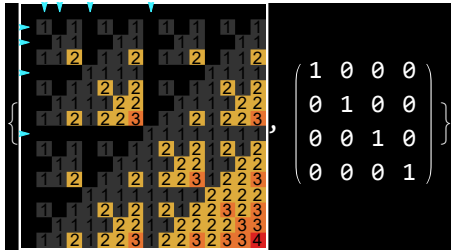
```

Out[]:=



```
In[*]:= matrixPicture@IdentityMatrix[4]
```

```
Out[*]=
```



```
In[*]:= IsomorphicSubgraphQ @@
```

```
(Graph /@ graphFullSym /@ {IdentityMatrix[4], Reverse[IdentityMatrix[4]]})
```

```
Out[*]=
```

True

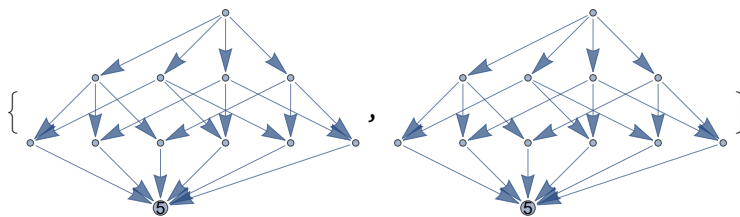
```
In[*]:= allSymPermutations[d_] :=
```

```
Select[PermutationMatrix /@ Permutations[Range@d], SymmetricMatrixQ]
```

```
In[*]:= dispFact@addGraphProps@hasseSymFact@# & /@
```

```
{IdentityMatrix[4], Reverse[IdentityMatrix[4]]}
```

```
Out[*]=
```



```
posSelfComp3 = With[{as3p = allSymPermutations[3], as3in = allSymInvertible[3]},  
(m ↦ Sort[selfCompatibleCount /@ (m.# & /@ as3p)]) /@ as3in];
```

```
possibleSelfCompCountsGrouped3 = SortBy[Tally[posSelfComp3], First]
```

```
Out[*]=
```

```
{{{4, 4, 4, 4}, 1}, {{4, 6, 7, 7}, 4}, {{5, 6, 6, 7}, 12}, {{5, 6, 7, 8}, 6}}
```

```
In[*]:= posSelfComp4 = With[{as4p = allSymPermutations[4], as4in = allSymInvertible[4]},
```

```
(m ↦ Sort[selfCompatibleCount /@ (m.# & /@ as4p)]) /@ as4in];
```

```

In[ ]:= possibleSelfCompCountsGrouped4 = SortBy[Tally[posSelfComp4], First]
Out[ ]:=
{{ {5, 5, 5, 5, 5, 5, 5, 5, 5, 5}, 1}, { {5, 5, 5, 5, 8, 8, 8, 8, 8, 8}, 4},
  { {5, 5, 6, 7, 7, 7, 7, 8, 8, 8}, 4}, { {5, 9, 9, 9, 9, 9, 11, 11, 11, 11}, 1},
  { {5, 9, 9, 9, 10, 10, 10, 12, 12, 12}, 4}, { {6, 6, 6, 8, 10, 10, 10, 12, 12, 12}, 4},
  { {6, 6, 7, 7, 8, 10, 10, 11, 11, 12}, 24}, { {6, 6, 8, 8, 8, 10, 12, 12, 12, 12}, 12},
  { {6, 10, 10, 10, 10, 12, 12, 12, 12, 12}, 12}, { {6, 10, 10, 12, 12, 13, 14, 14, 15, 16}, 24},
  { {7, 9, 9, 10, 10, 10, 11, 11, 12, 13}, 24}, { {7, 9, 9, 10, 10, 11, 11, 11, 14, 14}, 3},
  { {7, 9, 9, 11, 11, 11, 12, 12, 12, 12}, 9}, { {7, 9, 10, 12, 12, 12, 13, 13, 14, 14}, 24},
  { {7, 10, 10, 10, 10, 11, 12, 12, 14, 14}, 9}, { {7, 10, 10, 11, 12, 12, 12, 12, 12, 12}, 3},
  { {8, 9, 10, 10, 10, 11, 12, 12, 14, 14}, 24}, { {8, 10, 10, 10, 11, 12, 12, 13, 14, 14}, 24},
  { {8, 10, 10, 11, 11, 12, 14, 14, 14, 14}, 24}, { {9, 9, 9, 10, 10, 10, 11, 11, 11, 12}, 48},
  { {9, 9, 10, 10, 10, 10, 11, 11, 11, 11}, 12}, { {9, 9, 10, 10, 10, 11, 11, 12, 12, 12}, 24},
  { {9, 9, 10, 10, 10, 11, 11, 12, 12, 14}, 6}, { {9, 9, 10, 10, 11, 11, 11, 11, 12, 14}, 24},
  { {10, 10, 10, 10, 10, 10, 11, 11, 12, 12}, 12},
  { {10, 10, 10, 10, 11, 11, 12, 14, 14, 14}, 12} }

In[ ]:= posSelfComp5 = With[{as5p = allSymPermutations[5], as5in = allSymInvertible[5]},
  ParallelMap[m ↦ Sort[selfCompatibleCount /@ (m.# & /@ as5p)], as5in]];

In[ ]:= possibleSelfCompCountsGrouped5 = SortBy[Tally[posSelfComp5], First]

In[ ]:= DumpSave[
  FileNameJoin[{NotebookDirectory[], "possible-self-comp-counts-grouped-3-to-5.mx"}],
  {possibleSelfCompCountsGrouped3,
   possibleSelfCompCountsGrouped4, possibleSelfCompCountsGrouped5}];

Best definitely achievable by symmetric permutation size of self-comp set:

In[ ]:= Max[#[[All, 1, 1]] & /@ {possibleSelfCompCountsGrouped3,
  possibleSelfCompCountsGrouped4, possibleSelfCompCountsGrouped5}
Out[ ]:=
{5, 10, 16}

In[ ]:= only5selfcomp = Select[allSymInvertible[4],
  m ↦ (Union[selfCompatibleCount /@ (m.# & /@ allSymPermutations[4]]) == {5})]
Out[ ]:=
{{ {0, 1, 1, 1}, {1, 0, 1, 1}, {1, 1, 0, 1}, {1, 1, 1, 0} }}

In[ ]:= MatrixForm@First@only5selfcomp
Out[ ]//MatrixForm=

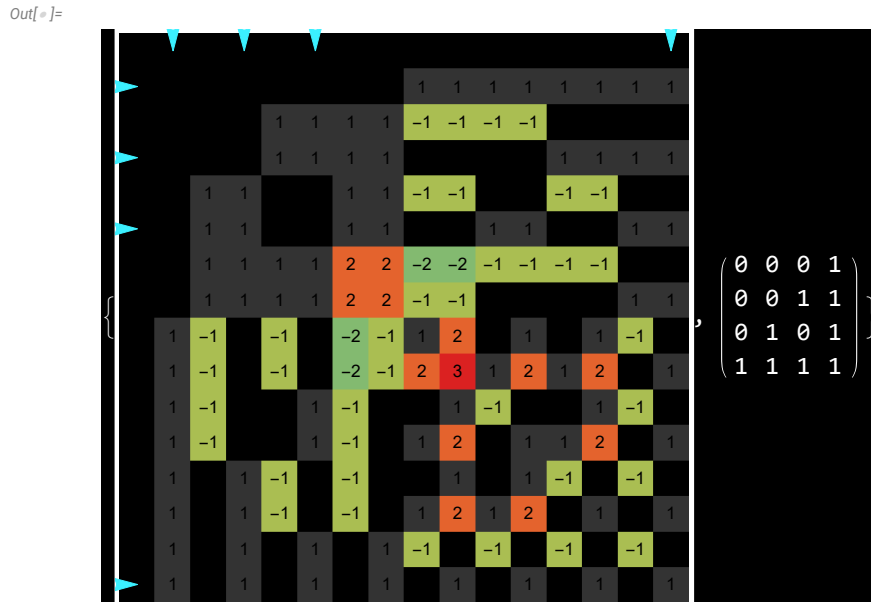
$$\begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$


In[ ]:= With[{missing5 = Table[1, {5, 5}] - IdentityMatrix[5]},
  Sort[selfCompatibleCount[missing5.#] & /@ allSymPermutations[5]]]
Out[ ]:=
{6, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12,
  14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14}

```

```
In[ ]:= unavoidablyManySelfComp4 = Select[allSymInvertible[4],
      m -> (Min[selfCompatibleCount /@ (m.# & /@allSymPermutations[4])) == 10] // First
Out[ ]:=
{{0, 0, 0, 1}, {0, 0, 1, 1}, {0, 1, 0, 1}, {1, 1, 1, 1}}
```

```
In[ ]:= matrixPicture[unavoidablyManySelfComp4]
```



Shit, product of symmetric permutations is only symmetric if they commute. The following function gives actual symmetric permutations of rows of a given (presumably symmetric) matrix:

```
In[ ]:= symPermutations[B01_] := Select[
      B01.PermutationMatrix[#] & /@Permutations[Range@Length@B01], SymmetricMatrixQ]
```

```
In[ ]:= (selfCompatibleCount /@ symPermutations[#]) & /@allSymInvertible[3]
Out[ ]:=
{{6, 6, 6, 4}, {6}, {6}, {6}, {6}, {6}, {6}, {6}, {4, 4, 4, 4}, {4, 6},
      {6}, {4, 6}, {6}, {6}, {6}, {6}, {6}, {6}, {6}, {6}, {6}, {4, 6}, {6}}
```

```
In[ ]:= (selfCompatibleCount /@ symPermutations[#]) & /@allSymInvertible[2]
Out[ ]:=
{{3, 3}, {3}, {3}}
```

```
In[ ]:= (selfCompatibleCount /@ symPermutations[#]) & /@allSymInvertible[1]
Out[ ]:=
{{2}}
```

```
In[ ]:= SortBy[Tally[Sort[selfCompatibleCount /@ symPermutations[#] & /@allSymInvertible[4]],
      #[[1, 1]] &]
Out[ ]:=
{{{5, 5, 5, 5}, 4}, {{5, 9, 9, 9}, 4}, {{5, 5, 5, 5, 5, 5, 5, 5, 5, 5}, 1},
      {{5, 9, 9, 9, 9, 9, 9, 9, 9, 9}, 1}, {{6}, 24}, {{6, 8}, 12},
      {{6, 6, 6, 8}, 8}, {{9}, 144}, {{9, 9}, 30}, {{10}, 96}, {{10, 10}, 48}}
```

```

In[*]:= correctSymPermSelfCompCount5 =
  SortBy[Tally[ParallelMap[Sort[selfCompatibleCount /@ symPermutations[#]] &,
    allSymInvertible[5]]], #[[1, 1]] &]

Out[*]:=
{{ {6, 6}, 30}, { {6, 8}, 120}, { {6, 6, 6, 6}, 20},
  { {6, 8, 8, 8, 8, 8}, 12}, { {6, 10, 10, 10, 10, 10}, 12},
  { {6, 6, 6, 6, 6, 6, 6, 6}, 10}, { {6, 6, 6, 8, 8, 8, 8, 8, 8, 10}, 10},
  { {6, 12, 12, 12, 12, 12, 12, 18, 18, 18}, 5}, { {6, 12, 12, 12, 12, 12, 12, 12,
    12, 12, 12, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14}, 1},
  { {6, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 18, 18, 18, 18, 18, 18, 18, 18,
    18, 18, 18, 18, 18, 18}, 1}, { {7, 9}, 60}, { {7, 9, 9, 9}, 60}, { {8}, 240},
  { {8, 12}, 240}, { {8, 8, 8, 10}, 20}, { {8, 8, 8, 12}, 20}, { {8, 12, 12, 12}, 30},
  { {8, 8, 8, 12, 12, 12, 12, 12}, 10}, { {9}, 120}, { {10}, 120}, { {10, 10}, 60},
  { {10, 12}, 240}, { {11}, 720}, { {12}, 1320}, { {12, 12}, 420}, { {12, 14}, 120},
  { {12, 16}, 240}, { {12, 18}, 240}, { {12, 24}, 60}, { {12, 12, 12, 12}, 80},
  { {12, 18, 18, 18}, 20}, { {12, 12, 12, 12, 14, 14, 14, 14, 14, 14}, 5}, { {13}, 840},
  { {13, 17}, 240}, { {14}, 240}, { {15}, 120}, { {15, 19}, 300}, { {15, 19, 19, 19}, 60},
  { {16}, 120}, { {16, 20}, 120}, { {16, 16, 20, 20}, 30}, { {17}, 1080}, { {17, 17}, 60},
  { {18}, 2040}, { {18, 18}, 210}, { {19}, 3120}, { {20}, 840}, { {20, 20}, 120} }

Guarantied lower bound from permutations looks (below) look line Binomial[n+1,Floor[(n+1)/2]]

In[*]:= {1 → 2, 2 → 3, 3 → 6, 4 → 10, 5 → 20}

In[*]:= Table[Binomial[n, Floor[n / 2]], {n, 1, 7}]

Out[*]:=
{1, 2, 3, 6, 10, 20, 35}

In[*]:= someAnavoidablyLargeSelfCount5 =
  Select[allSymInvertible[5], selfCompatibleCount /@ symPermutations[#] == {20, 20} &, 10]

In[*]:= Labeled[Column[Style[{matrixPicture[#] /. (ImageSize → _) → (ImageSize → 150),
  Graph[setGraphStyle@dispFact@hasseSymFact@#, ImageSize → 250]}],
  Background → Black] & /@ someAnavoidablyLargeSelfCount5],
  Style["Some 5×5 matrices unavoidably producing 20 self-compatible
  vectors (two sym perms for each)", White, Background → Black], Top]

In[*]:= Export[FileNameJoin[{NotebookDirectory[], "some-unavoidably-large-self-comp-5.jpg"}],
  %, Background → Black];

```

The last two are quite dissappointing:

```
In[ ]:= Table[Sort[selfCompatibleCount /@ symPermutations[m]],
```

$\{m, \{$

0	0	1
0	1	0
1	0	1

 $\},$

0	0	0	1
0	0	1	0
0	1	1	0
1	0	0	1

 $,$

0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	1	0	1	0
1	0	0	0	1

 $,$

0	0	0	0	0	1
0	0	0	0	1	0
0	0	0	1	0	0
0	0	1	1	0	0
0	1	0	0	1	0
1	0	0	0	0	1

 $,$

0	0	0	0	0	1
0	0	0	0	1	0
0	0	0	1	0	0
0	0	0	1	0	0
0	0	1	0	1	0
0	1	0	0	0	1
1	0	0	0	0	1

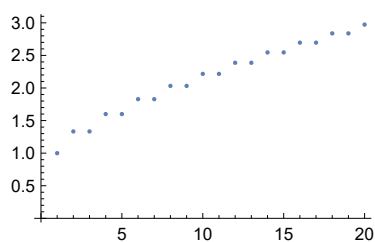
 $,$

0	0	0	0	0	0	1
0	0	0	0	0	1	0
0	0	0	0	1	0	0
0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	1	0	0	0	1	0
1	0	0	0	0	0	1

 $\}\}\}$

```
In[ ]:= ListPlot[Table[2^n / Binomial[n + 1, Floor[(n + 1) / 2]], {n, 1, 20}]]
```

```
Out[ ]:=
```



```
In[ ]:= (selfCompatibleCount /@ symPermutations[#]) & /@  
  ((manydiagMatr[#, Range[0, # - 1]] &) /@ Range[8])
```

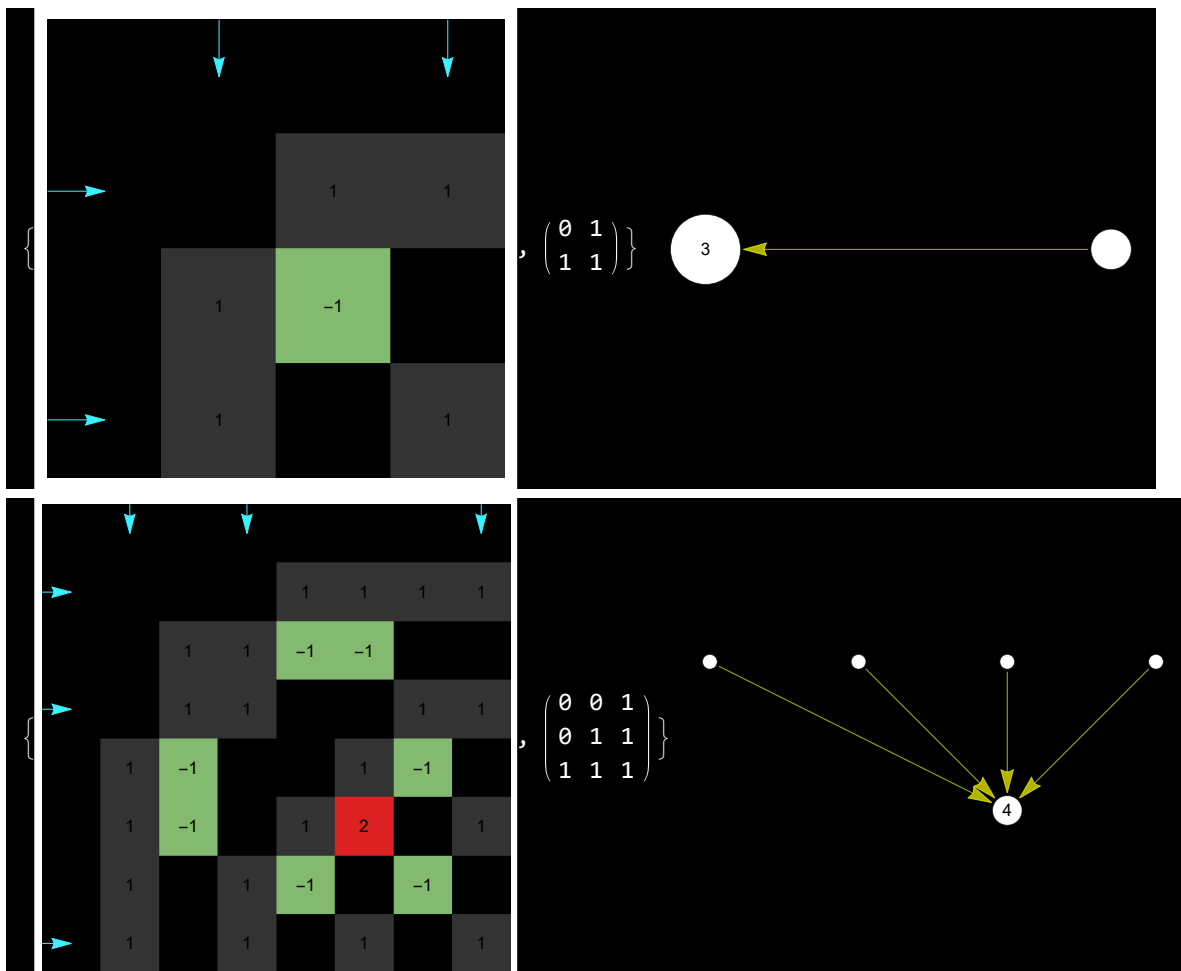
```
Out[ ]:=
```

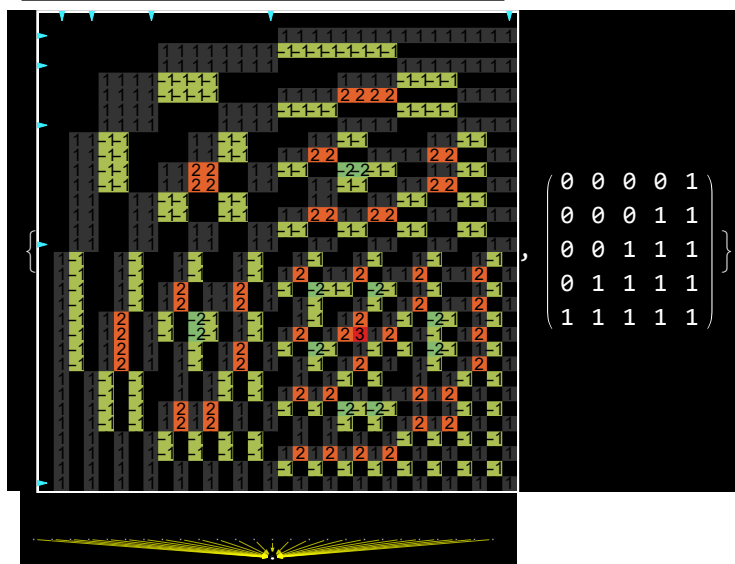
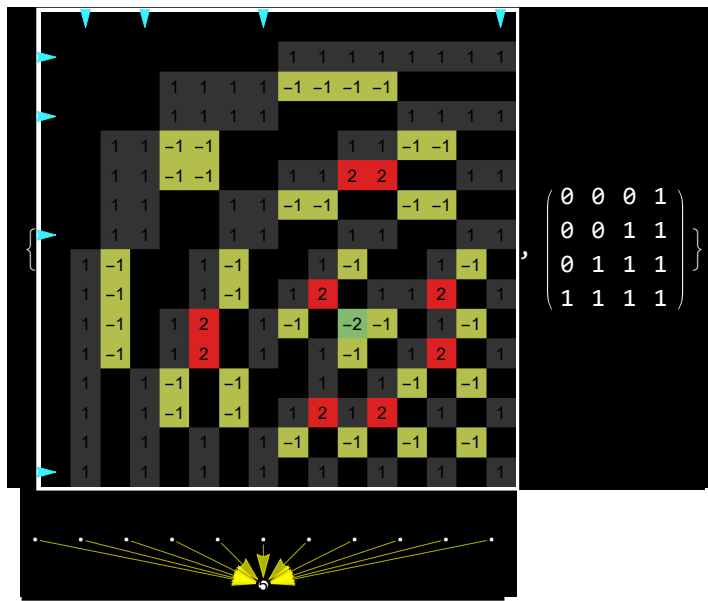
```
{{2}, {3}, {6}, {10}, {20}, {35}, {70}, {126}}
```

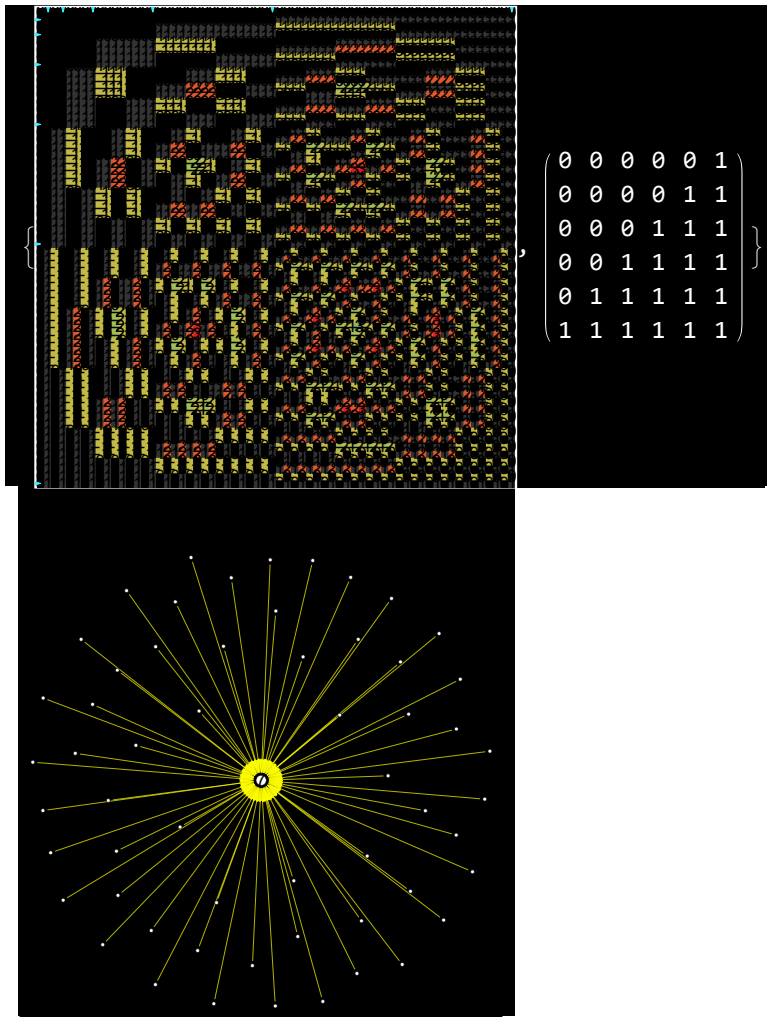
```
In[ ]:= exportBlackBack[file_, expr_] :=  
  Export[FileNameJoin[{NotebookDirectory[], file}], expr, Background → Black]
```

```
In[ ]:= With[{halfmatr = (manydiagMatr[#, Range[0, # - 1]] &) /@ Range[2, 6]},  
  Column[Style[{matrixPicture[#] /. (ImageSize → _) → (ImageSize → 250),  
    Graph[setGraphStyle@dispFact@hasseSymFact@#, ImageSize → 250]},  
    Background → Black] & /@ halfmatr]]
```

```
Out[ ]:=
```

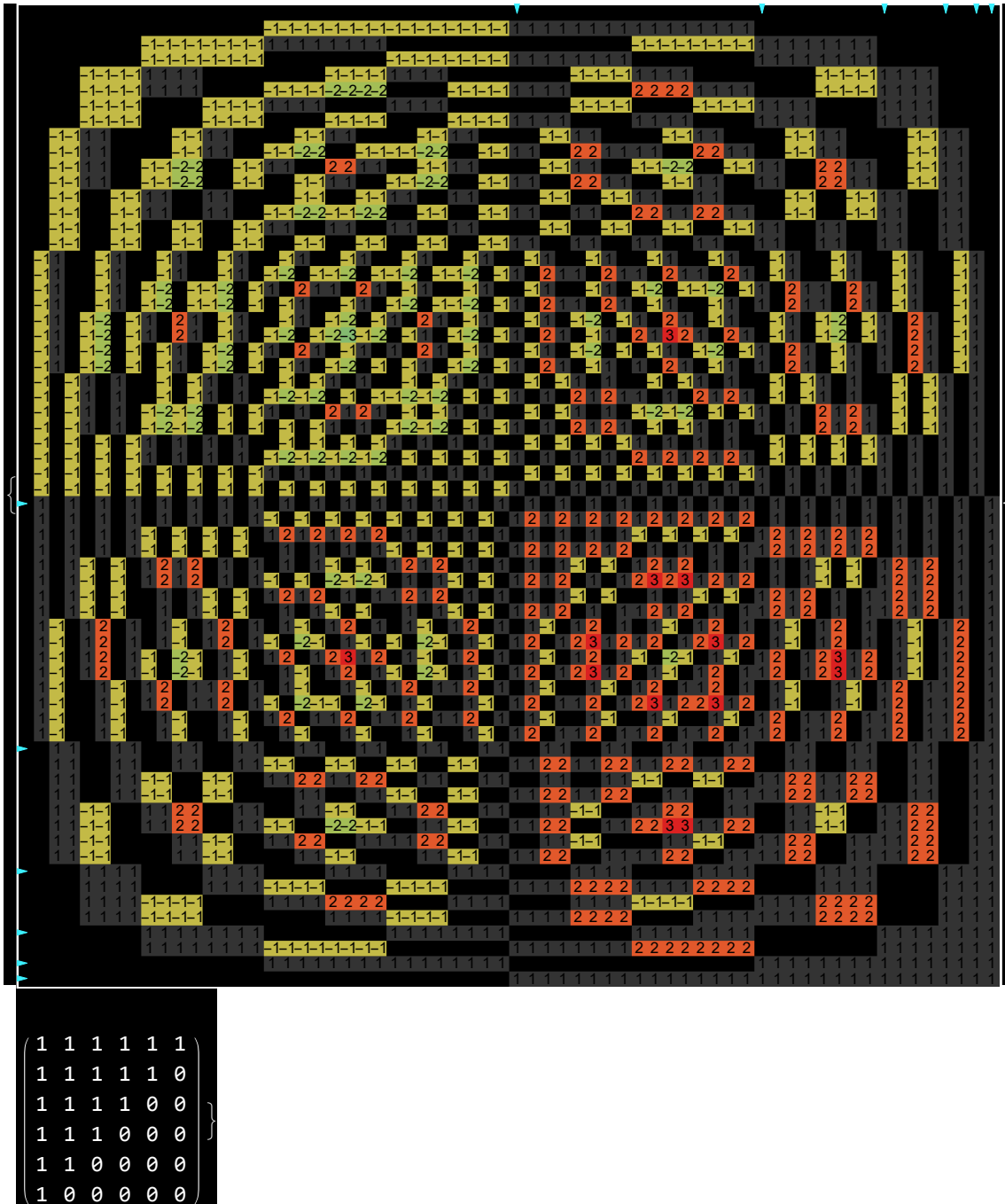






```
In[ ]:= exportBlackBack["half-matrix-demo.jpg", %];
```

```
In[ ]:= matrixPicture[manydiagMatr[6, Range[0, 5]]][[-1 ;; 1 ;; -1, -1 ;; 1 ;; -1]]
Out[ ]:=
```



```

In[*]:= SeedRandom[42];
With[{p = PermutationMatrix[RandomPermutation[5]]},
  utr = manydiagMatr[5, Range[0, 4]] [[-1 ;; 1 ;; -1, -1 ;; 1 ;; -1]], m = MatrixForm},
{p, utr // MatrixForm, p.utr // m, utr.p // m, p.utr.p // m, p.utr.(pT) // m}]

```

Out[*]=

$$\left\{ \text{PermutationMatrix} \left[\begin{array}{ccccc} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{array} \right], \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}, \right.$$

$$\left. \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix} \right\}$$

Since conjugation by a permutation doesn't effect the order and the self compatible count,

```

In[*]:= allSymInvDifCongClass[d_] := With[{prms = PermutationMatrix /@ Permutations[Range[d]]},
  DeleteDuplicatesBy[Select[(# + LowerTriangularize[#T, -1]) & /@ PadLeft /@
    (Internal`PartitionRagged[#, Range[d, 1, -1]] &) /@ Tuples[{0, 1},  $\frac{d(d+1)}{2}$ ],
    MatrixRank@# == d &], First@MinimalBy[Table[p.#.(pT), {p, prms}], Identity] &]]

allSymInvDifClassDifSort[d_] :=
  With[{prms = PermutationMatrix /@ Permutations[Range[d]]},
    DeleteDuplicatesBy[DeleteDuplicatesBy[allSymInvertible[d], Sort[#T] &],
      First@MinimalBy[Table[p.#.(pT), {p, prms}], Identity] &]]
  (*deleting duplicates by sort of transpose is likely not needed*)

```

```

In[*]:= Length[allSymInvDifCongClass[#]] & /@ Range[4]

```

Out[*]=

{1, 3, 9, 40}

```

In[*]:= Length[allSymInvertible@#] & /@ Range[5]

```

Out[*]=

{1, 3, 23, 372, 14206}

```

In[*]:= Length[allSymInvDifClassDifSort@#] & /@ Range[4]

```

Out[*]=

{1, 2, 6, 27}

```

In[*]:= Length /@ {1}, DeleteDuplicates[allSymInvertible[2],
  IsomorphicGraphQ[hasseSymFact@#1, hasseSymFact@#2] &], allThreeHasse, allFourHasse}

```

Out[*]=

{1, 1, 3, 7}

```

In[*]:= Diagonal[cube01[6].Inverse[manydiagMatr[6, Range[0, 5]] [[-1 ;; 1 ;; -1, -1 ;; 1 ;; -1]].
  (cube01[6]T)] // Tally // SortBy[First]

```

Out[*]=

{{-3, 1}, {-2, 6}, {-1, 15}, {0, 20}, {1, 15}, {2, 6}, {3, 1}}

```
In[ ]:= Diagonal[cube01[7].Inverse[
    manydiagMatr[7, Range[0, 6]] [[-1 ;; 1 ;; -1, -1 ;; 1 ;; -1]] . (cube01[7]^T)] //
    Tally // SortBy[First]
```

```
Out[ ]:=
{{-3, 1}, {-2, 7}, {-1, 21}, {0, 35}, {1, 35}, {2, 21}, {3, 7}, {4, 1}}
```

```
In[ ]:= manydiagMatr[7, Range[0, 6]] [[-1 ;; 1 ;; -1, -1 ;; 1 ;; -1]] // Inverse // MatrixForm
```

```
Out[ ]//MatrixForm=
```

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

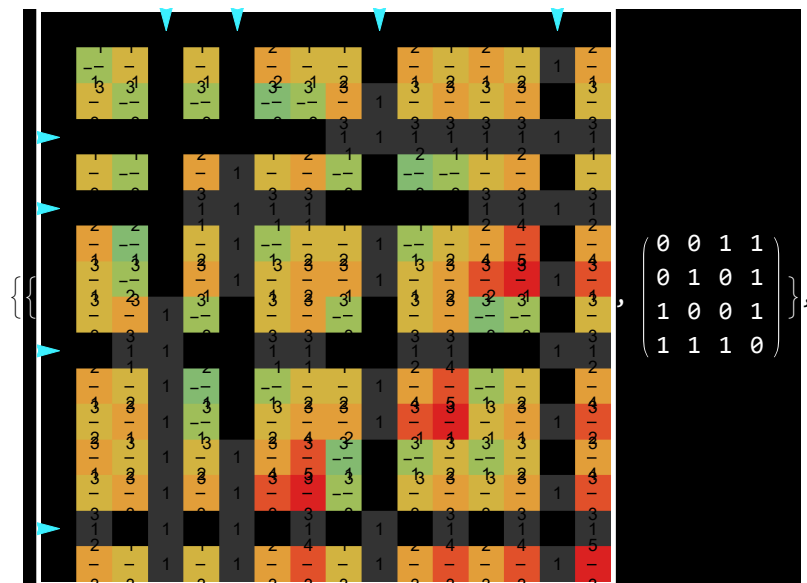
```
In[ ]:= Select[allSymInvertible[4], IsomorphicGraphQ[Graph[1 → 2], hasseSymFact@#] &]
```

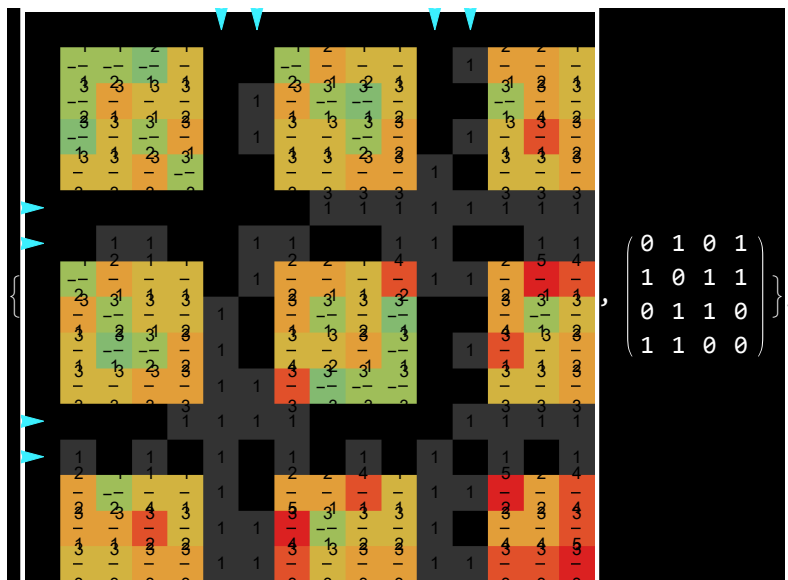
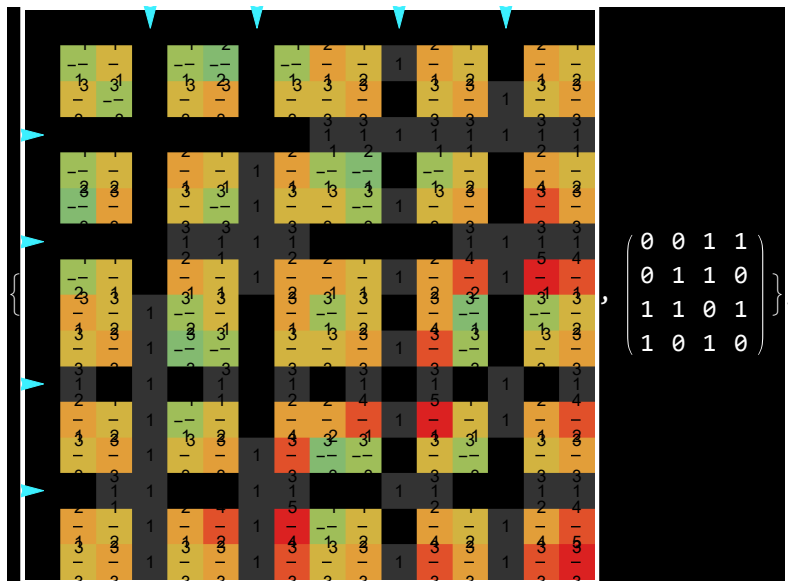
```
Out[ ]:=
```

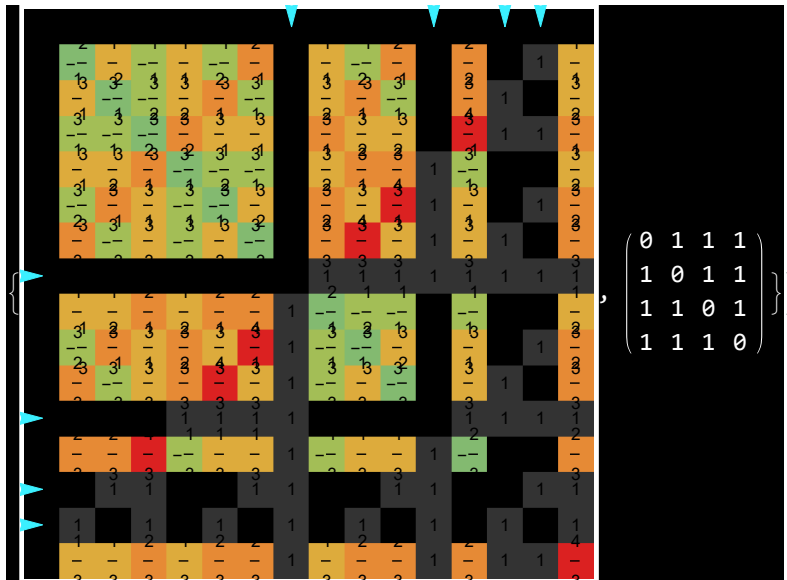
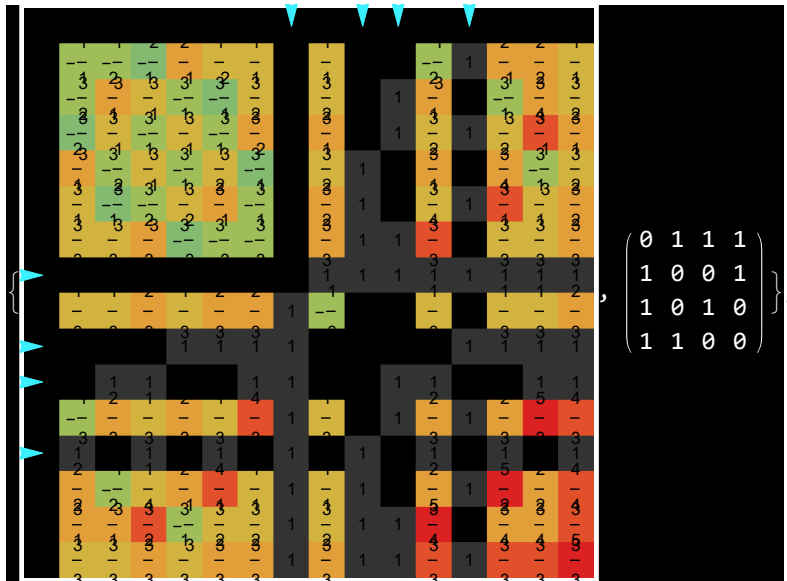
```
{{{0, 0, 1, 1}, {0, 1, 0, 1}, {1, 0, 0, 1}, {1, 1, 1, 0}},
 {{0, 0, 1, 1}, {0, 1, 1, 0}, {1, 1, 0, 1}, {1, 0, 1, 0}},
 {{0, 1, 0, 1}, {1, 0, 1, 1}, {0, 1, 1, 0}, {1, 1, 0, 0}},
 {{0, 1, 1, 1}, {1, 0, 0, 1}, {1, 0, 1, 0}, {1, 1, 0, 0}},
 {{0, 1, 1, 1}, {1, 0, 1, 1}, {1, 1, 0, 1}, {1, 1, 1, 0}}}
```

```
In[ ]:= matrixPicture /@%
```

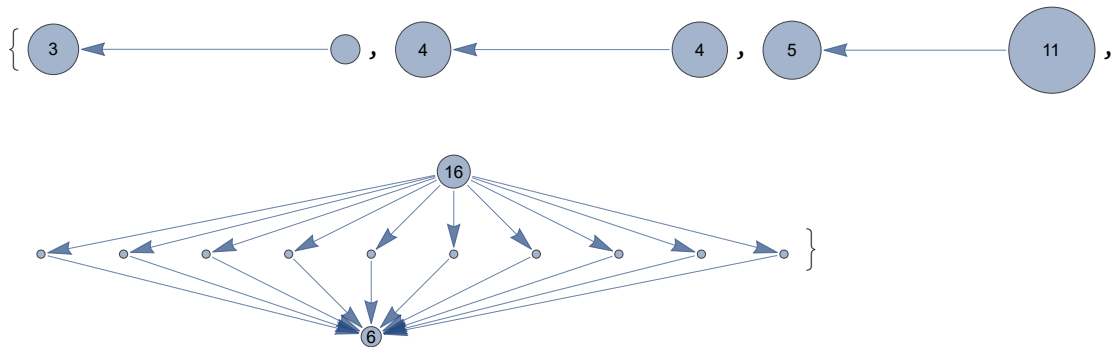
```
Out[ ]:=
```







In[*]:= `dispFact /@ (hasseSymFact /@ (Table[1, {#}, {#}] - IdentityMatrix[#] & /@ Range[2, 5]))`
 Out[*]=



```
In[ ]:= Select[allSymInvertible[5], IsomorphicGraphQ[Graph[1 → 2], hasseSymFact@#] &, 10]
```

Out[]=

$$\begin{aligned} & \{\{0, 0, 0, 1, 1\}, \{0, 0, 1, 0, 1\}, \{0, 1, 1, 1, 0\}, \{1, 0, 1, 1, 0\}, \{1, 1, 0, 0, 1\}\}, \\ & \{\{0, 0, 0, 1, 1\}, \{0, 0, 1, 1, 0\}, \{0, 1, 1, 0, 1\}, \{1, 1, 0, 1, 0\}, \{1, 0, 1, 0, 1\}\}, \\ & \{\{0, 0, 0, 1, 1\}, \{0, 1, 1, 0, 1\}, \{0, 1, 0, 1, 0\}, \{1, 0, 1, 1, 0\}, \{1, 1, 0, 0, 1\}\}, \\ & \{\{0, 0, 0, 1, 1\}, \{0, 1, 1, 1, 0\}, \{0, 1, 0, 0, 1\}, \{1, 1, 0, 1, 0\}, \{1, 0, 1, 0, 1\}\}, \\ & \{\{0, 0, 1, 0, 1\}, \{0, 0, 1, 1, 0\}, \{1, 1, 1, 0, 0\}, \{0, 1, 0, 1, 1\}, \{1, 0, 0, 1, 1\}\}, \\ & \{\{0, 0, 1, 0, 1\}, \{0, 1, 0, 1, 1\}, \{1, 0, 1, 1, 0\}, \{0, 1, 1, 0, 0\}, \{1, 1, 0, 0, 1\}\}, \\ & \{\{0, 0, 1, 0, 1\}, \{0, 1, 1, 1, 0\}, \{1, 1, 1, 0, 0\}, \{0, 1, 0, 0, 1\}, \{1, 0, 0, 1, 1\}\}, \\ & \{\{0, 0, 1, 1, 0\}, \{0, 1, 0, 1, 1\}, \{1, 0, 1, 0, 1\}, \{1, 1, 0, 1, 0\}, \{0, 1, 1, 0, 0\}\}, \\ & \{\{0, 0, 1, 1, 0\}, \{0, 1, 1, 0, 1\}, \{1, 1, 1, 0, 0\}, \{1, 0, 0, 1, 1\}, \{0, 1, 0, 1, 0\}\}, \\ & \{\{0, 0, 1, 1, 1\}, \{0, 1, 0, 0, 1\}, \{1, 0, 0, 1, 1\}, \{1, 0, 1, 0, 1\}, \{1, 1, 1, 1, 0\}\} \end{aligned}$$
$$\text{In}[\bullet] := \text{MatrixForm} / @ \%$$

Out[]=

$$\left\{ \begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 \end{pmatrix} \right\}$$

```
In[ ] := << "C:\\Users\\mitya\\Documents\\WolframPlay\\Correlating-Vectors-Research\\all-sym-  
hasse-5.mx"
```

```
In[ ] := << "C:\\Users\\mitya\\Documents\\WolframPlay\\Correlating-Vectors-Research\\all-sym-  
hasse-4.mx"
```

```
In[ ] := << "C:\\Users\\mitya\\Documents\\WolframPlay\\Correlating-Vectors-Research\\all-sym-  
hasse-3.mx"
```

```
ln[*]:= Length /@ {allThreeHasse, allFourHasse, allFiveHasse}
```

Out[]=

 $\{3, 7, 35\}$

```
ln[•]:= addGraphProps /@ hasseSymFact /@ allSymInvertible[2]
```

Out[]=

$$\{ \bullet \leftarrow \bullet, \bullet \leftarrow \bullet, \bullet \leftarrow \bullet \}$$

```
In[ ]:= allSymInvertible[5] // Length // AbsoluteTiming
```

Out[]=

 $\{1.785, 14\,206\}$

```
ln[ ]:= graphFull[B01_] := Module[{d = Length@B01, cube, matr}, cube = Tuples[{0, 1}, d];  
    matr = (cube.Inverse@B01.(cube^T)) /. {1 -> 0, x_ /; x != 0 -> 2};  
    Flatten@Table[If[i != j && And@@ NonNegative[matr[[i]] - matr[[j]]] &&  
        And@@ NonNegative[matr[[All, i]] - matr[[All, j]]],  
        cube[[i]] -> cube[[j]], Nothing], {i, 2^d}, {j, 2^d}]]
```



```

In[ ]:= graphFactorizedRaw[B01_] := Module[{d = Length@B01, cube, filtprodmatr,
  grps, filt = (# /. {1 → 0, x_ /; x ≠ 0 → 2} &)}, cube = Tuples[{0, 1}, d];
  filtprodmatr = filt[cube.Inverse[B01].(cube')];
  grps = {(p ↦ cube[[p]]) /@ #, {filtprodmatr[[#][1]], filtprodmatr[[All, #][1]]}} & /@
    GatherBy[Range[2d], {filtprodmatr[[#]], filtprodmatr[[All, #]]} &];
  Flatten@Table[If[i ≠ j && And @@ Flatten[NonNegative[grps[[i, 2]] - grps[[j, 2]]],
    grps[[i, 1]] → grps[[j, 1]], Nothing], {i, Length@grps}, {j, Length@grps}]
addGrProps[gr_] := Module[{filt = (# /. {1 → 0, x_ /; x ≠ 0 → 2} &)},
  gr // IGVertexMap[Sqrt@*Length, "SqLength" → VertexList] /* IGVertexMap[Length,
    "Length" → VertexList] /* IGVertexMap[MatrixForm[#T] &, Tooltip → VertexList]
grFact[B01_] :=
  addGrProps[Graph[graphFactorizedRaw[B01], PerformanceGoal → "Quality"]]
hasseFact[B01_] :=
  addGrProps[hasseDiagram@Graph[graphFactorizedRaw[B01], PerformanceGoal → "Speed"]]

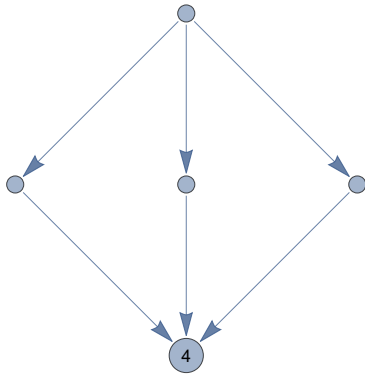
```

```

In[ ]:= dispFact[hasseFact@IdentityMatrix[3]]

```

Out[]:=



```

In[ ]:= bruteAllInv[d_] := Select[Partition[#, d] & /@ Tuples[{0, 1}, d2], MatrixRank@# == d &]

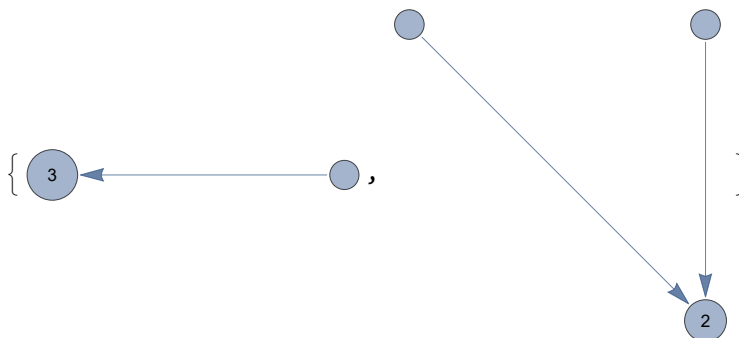
```

```

In[ ]:= DeleteDuplicates[dispFact /@ hasseFact /@ bruteAllInv[2], IsomorphicGraphQ]

```

Out[]:=



$In[*] := \text{DeleteDuplicates}[\text{dispFact} / @ \text{hasseFact} / @ \text{bruteAllInv}[3], \text{IsomorphicGraphQ}]$
 $Out[*] :=$

