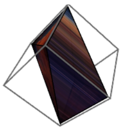
In[ ]:=
```
Quit[]
```

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]
```

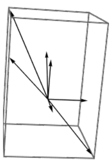
Out[ ]=
```
{{{0, 0, 0}, {1, 0, 0}, {0, 1, 0}}, {{1, 0, 1}, {0, 1, 1}, {1, 1, 1}},
 {{0, 0, 0}, {1, 0, 0}, {1, 0, 1}}, {{0, 0, 0}, {0, 1, 0}, {0, 1, 1}},
 {{1, 0, 0}, {0, 1, 0}, {1, 1, 1}}, {{0, 0, 0}, {1, 0, 1}, {0, 1, 1}},
 {{0, 1, 0}, {0, 1, 1}, {1, 1, 1}}, {{1, 0, 0}, {1, 0, 1}, {1, 1, 1}}}
```

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+basis, 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[2^d (d + 1), (Len@# + d)] &],
    bcan = Select[Subsets[Cpl[cube, B01ᵀ]], Divisible[2^d (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@# == (2^d (d + 1))/(d + 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 basises 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 * 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}}}
```

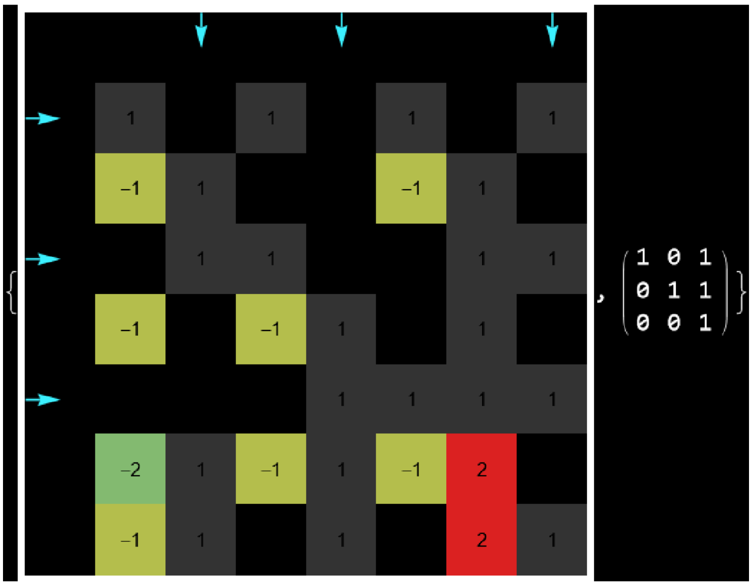
In[ ]:=
```
(achNumber /@ fourByFourSample) // AbsoluteTiming
```

Out[ ]=
```
{565.683, {2, 2, 2, 2}}
```

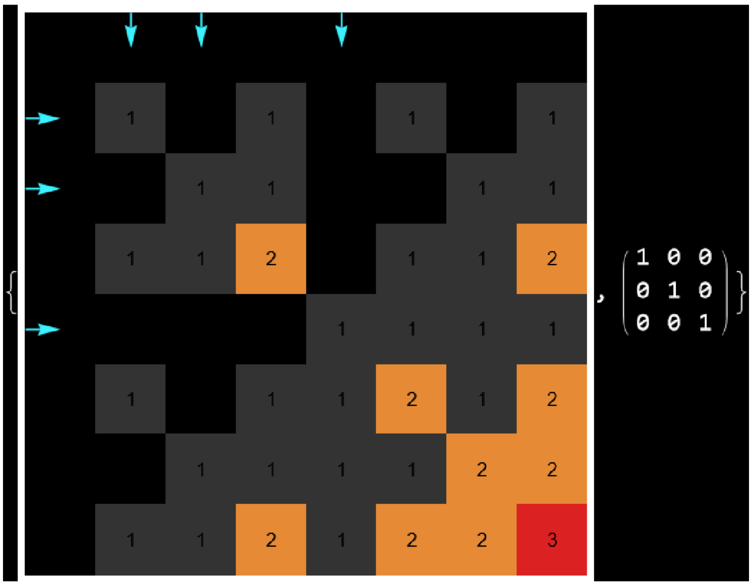
In[◦]:=
```
matrixPicture[B01_] :=
 Module[{d = Length@B01, cube, final, pos1, pos2}, cube = Tuples[{0, 1}, d];
  pos1 = Flatten[Position[cube, #] & /@ (B01ᵀ)];
  pos2 = Flatten[Position[cube, #] & /@ B01];
  final = cube.Inverse@B01.(cubeᵀ);
  Style[{ArrayPlot[final, ColorRules → {0 → ■, 1 → ■},
     ColorFunction → (ColorData["Rainbow", (# + 1)/2] &),
     Epilog → {Black, MapIndexed[Text[#1, Reverse[#2 - 0.5]] &, Reverse[final], {2}],
       ■, Arrow[{{# - 0.5, 2^d}, {# - 0.5, 2^d - 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[ ]=

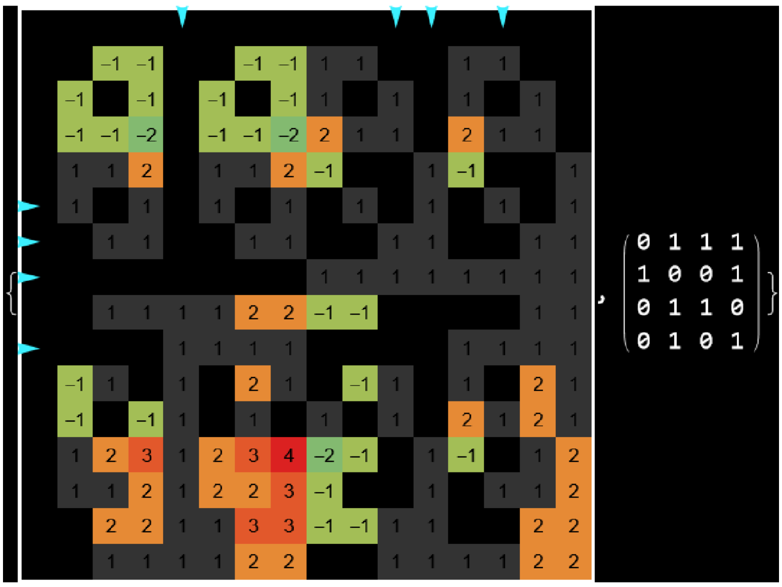


In[ ]:=
```
matrixPicture@First[
    RandomChoice[Select[Tuples[{0, 1}, {3, 3}], MatrixRank@# == 3 &], 1]] // Rasterize
imlist = Rasterize /@ matrixPicture /@ Select[Tuples[{0, 1}, {3, 3}], MatrixRank@# == 3 &];
```

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}}
```

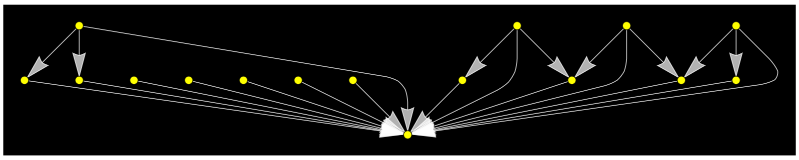
In[ ]:=
```
matrixPicture@% // Rasterize
```
Out[ ]=



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}}
```

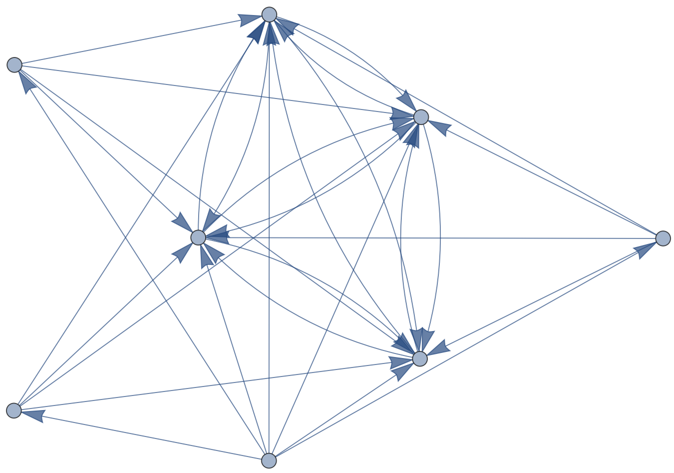
In[◦]:=
```
Graph[graphPicture@{{0, 1, 1, 1}, {1, 0, 0, 1}, {0, 1, 1, 0}, {0, 1, 0, 1}},
 Background → Black, VertexStyle → Yellow, EdgeStyle → White]
```

Out[◦]=



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

Out[◦]=



Check wether 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.(cubeᵀ) /. {1 → 0, x_ /; x ≠ 0 → 2};
  Map[Sort, final, {0, -2}] == Map[Sort, finalᵀ, {0, -2}]]
```

Check on invertible d*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
```

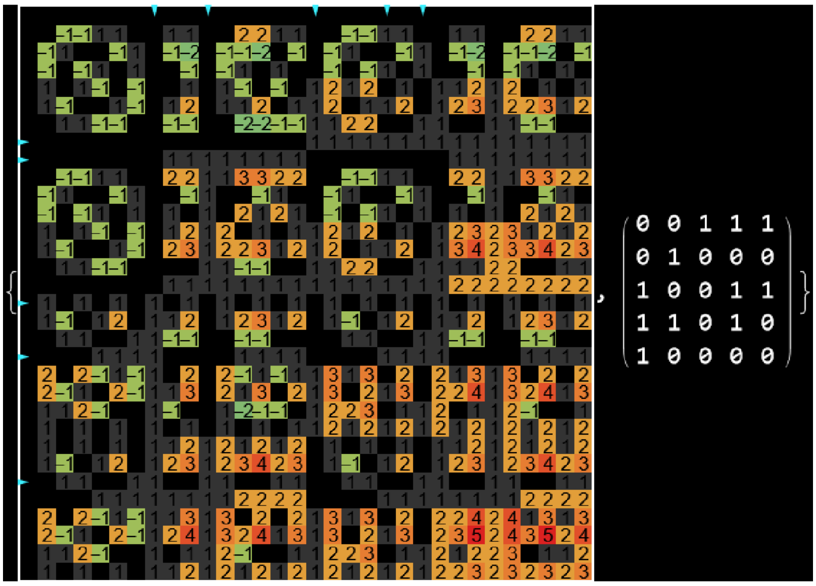
Out[∘]=

```
False
```

In[∘]:=

```
matrixPicture@counterexample5 // Rasterize
```

Out[∘]=

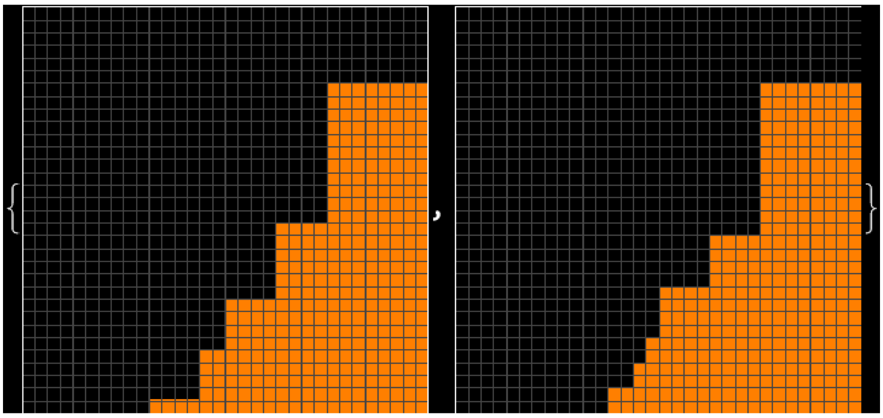


In[∘]:=

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

In[ ]:=
```
Style[With[{cube = Tuples[{0, 1}, 5]},
    ArrayPlot[Map[Sort, (cube.Inverse[#].(cubeᵀ)) /. {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, counterexample5ᵀ}, White, Background → Black] // 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}}
```

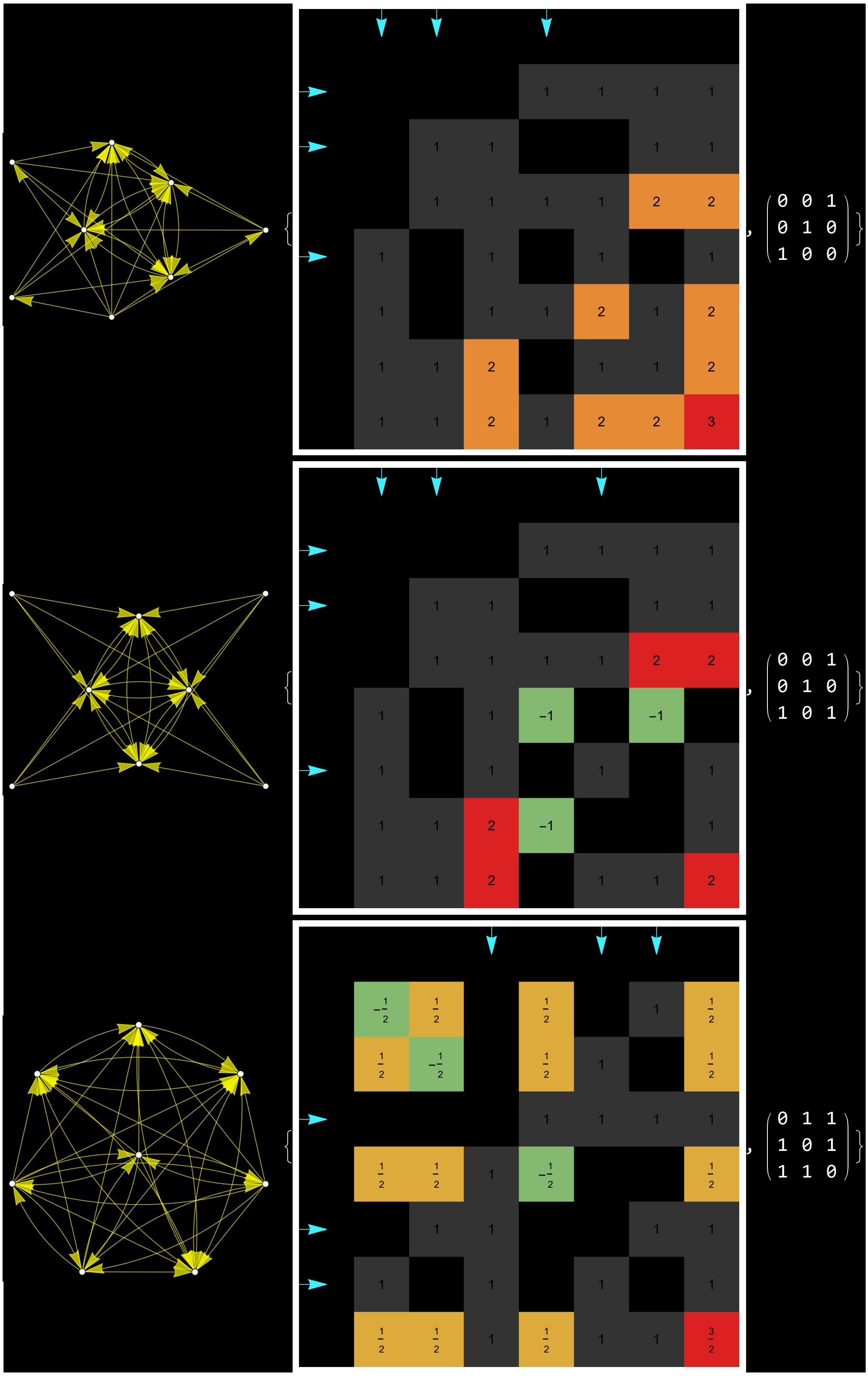
Function to produce graph of allowable relations given symmetric invertible 01 matrix:

In[ ]:=
```
graphFullSym[B01_] := Module[{d = Length@B01, cube, matr}, cube = Tuples[{0, 1}, d];
  matr = (cube.Inverse@B01.(cubeᵀ)) /. {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}]]
```

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

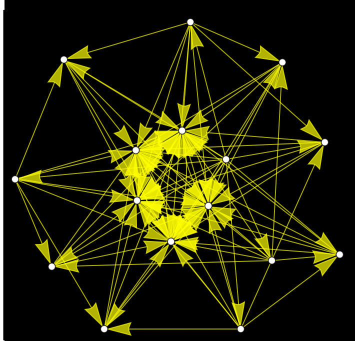
Out[ ]=

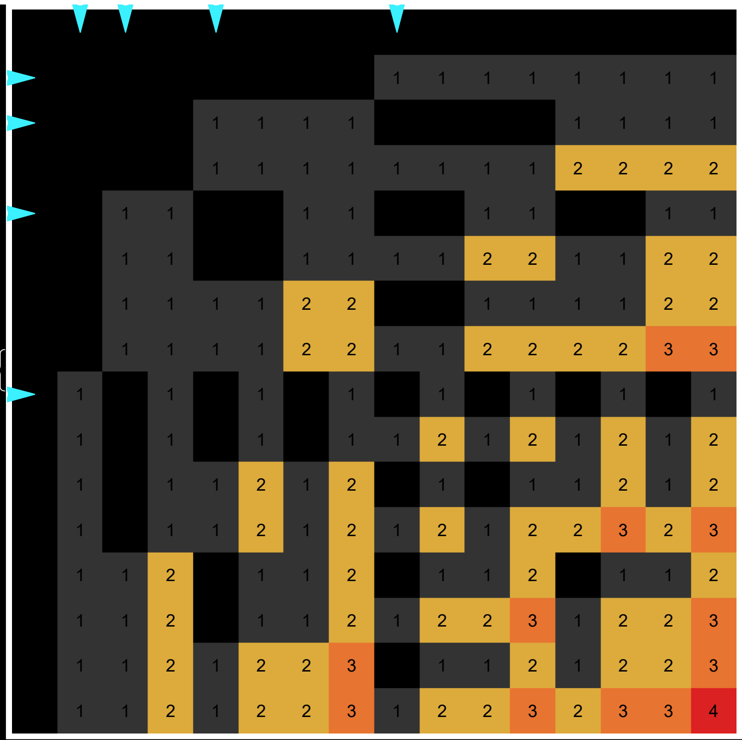
```
{{-1, 0, 1}, {0, 1, 0}, {1, 0, 0}}
```

In[ ]:=

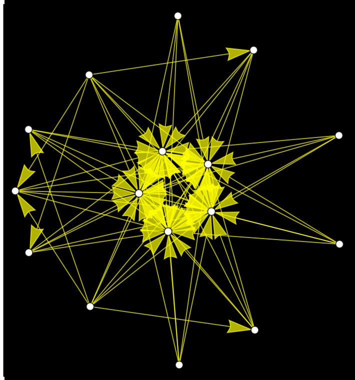
```
Style[
 Grid@DeleteDuplicates[{Graph[graphFullSym@#, Background → Black, VertexStyle → White,
       EdgeStyle → Yellow], matrixPicture@#} & /@ allSymInvertible[4],
   IsomorphicGraphQ[#1[[1]], #2[[1]]] &], Background → Black, White]
```

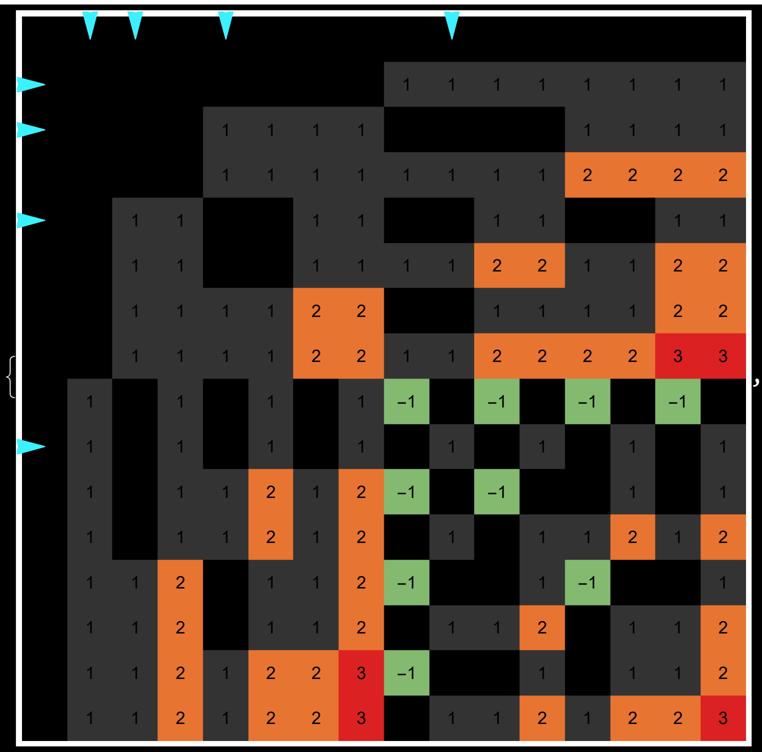
Out[ ]=



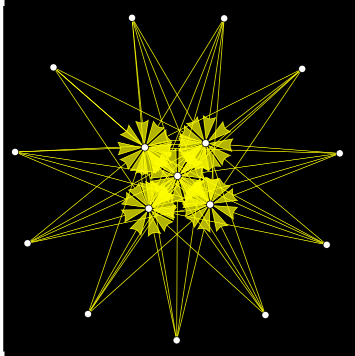


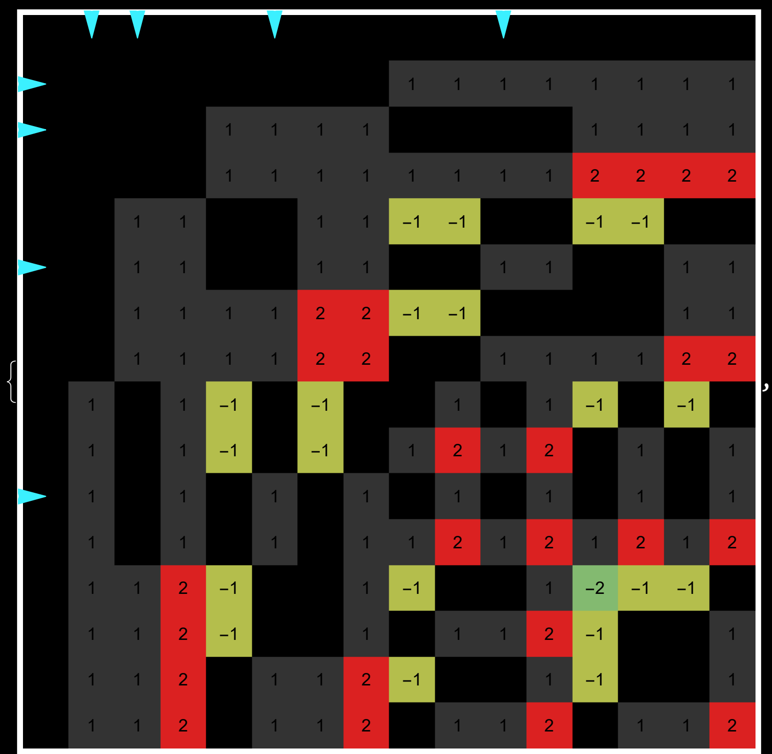
$\left\{\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}\right\}$



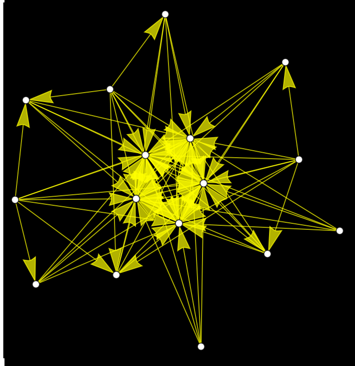


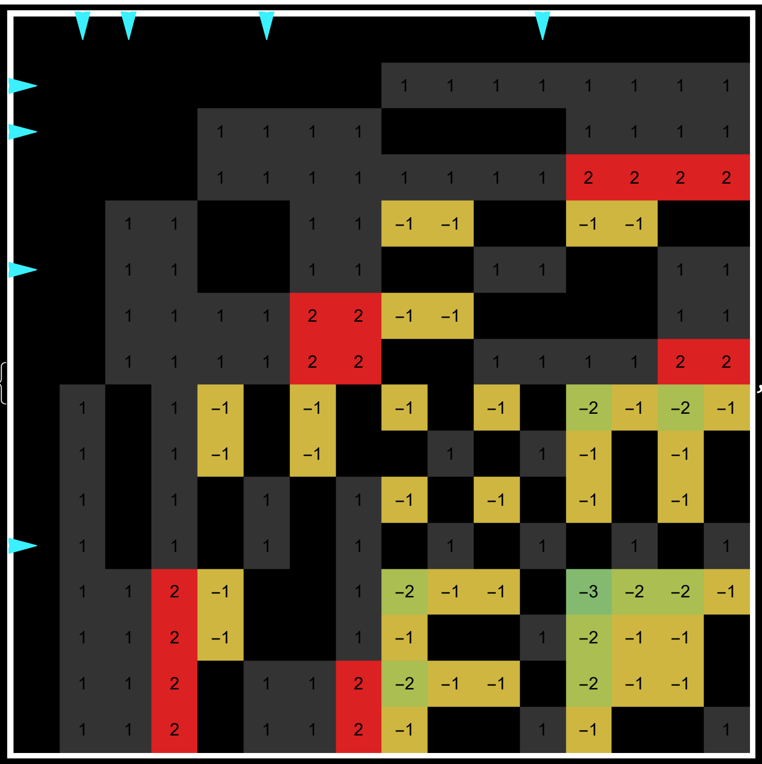
$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}\}$$



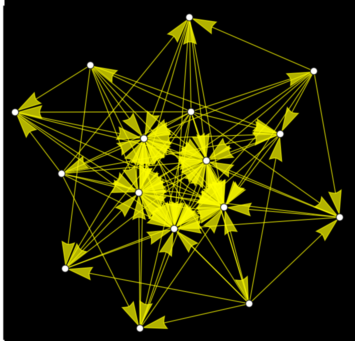


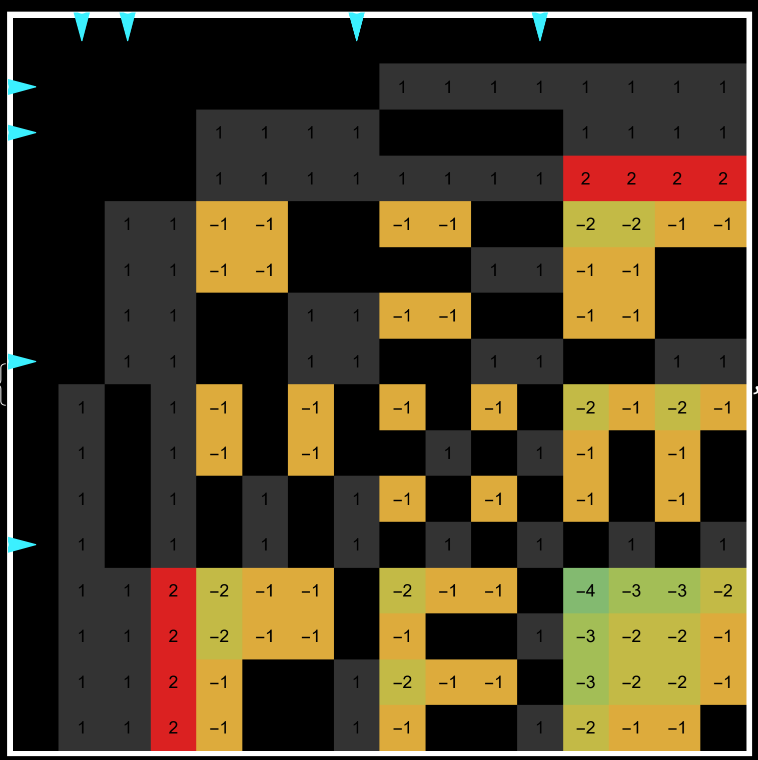
$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}\}$$



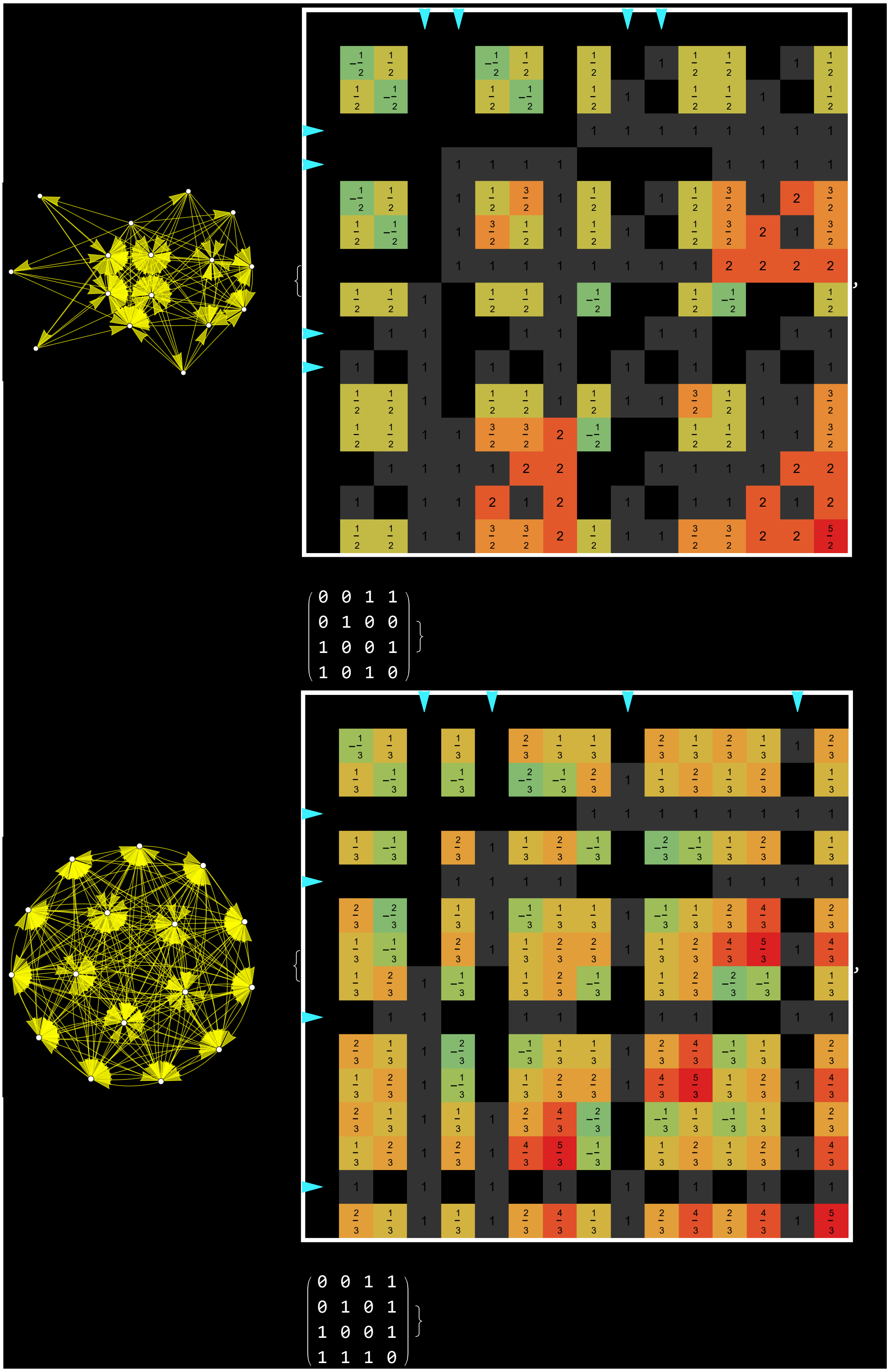


$$\left\{\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}\right\}$$

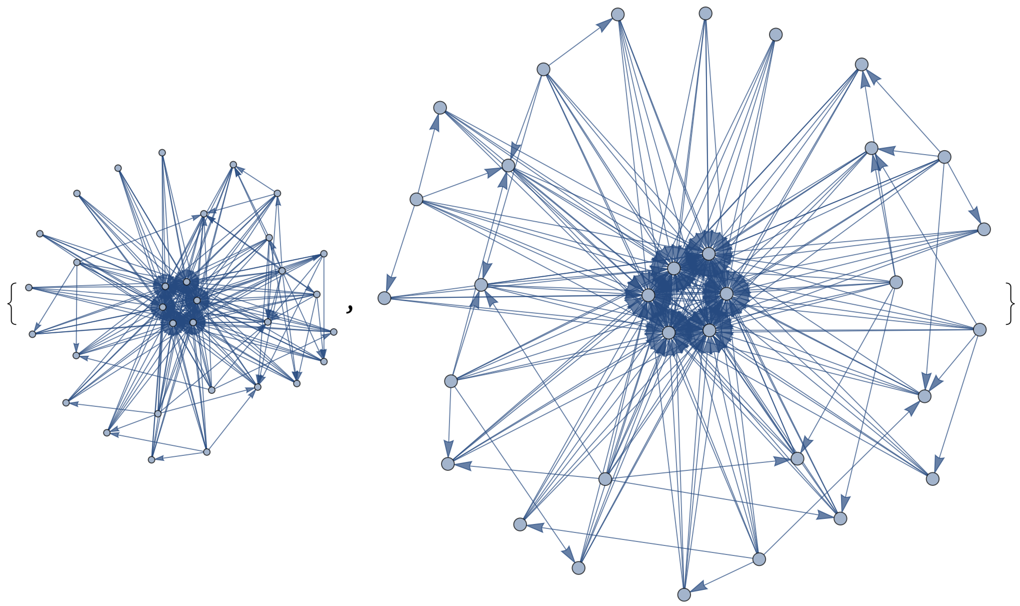




$$\left\{\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}\right\}$$

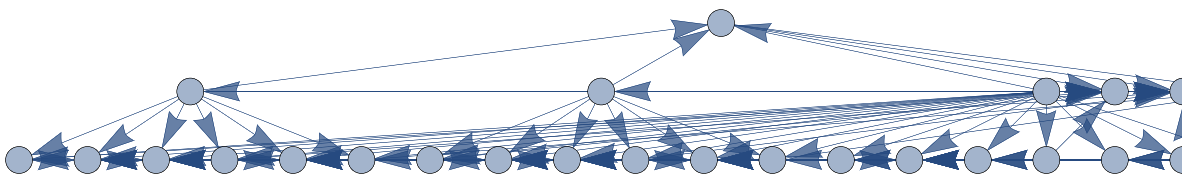

0 0 1 1
0 1 0 0
1 0 0 1
1 0 1 0
0 0 1 1
0 1 0 1
1 0 0 1
1 1 1 0

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



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, 2^d - d - 1}, {j, 2^d - d - 1}]]
```

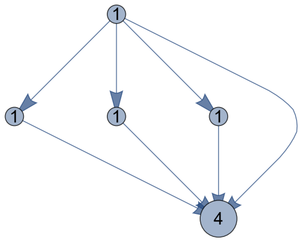
In[ ]:=
```
Graph[graphSymNoSinks@IdentityMatrix@5, GraphLayout → "LayeredEmbedding"]
```
Out[ ]=



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", 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 IGDocumentation[] to get started.

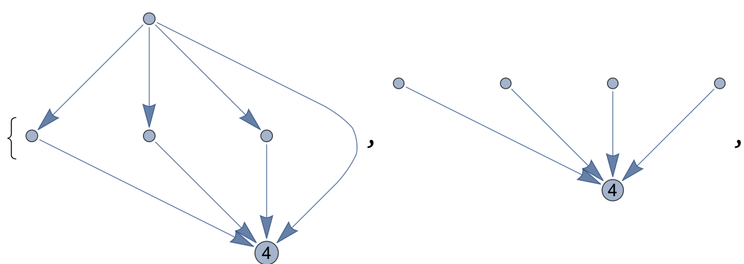
```
graphSymFactorizedRaw[B01_] :=
 Module[{d = Length@B01, cube, grps, filt = (# /. {1 → 0, x_ /; x ≠ 0 → 2} &)},
  cube = Tuples[{0, 1}, d];
  grps = {#, filt[#⟦1⟧]} & /@ GatherBy[cube.Inverse@B01.(cubeᵀ), filt];
  Flatten@Table[If[i ≠ j && And @@ NonNegative[grps⟦i, 2⟧ - grps⟦j, 2⟧],
     grps⟦i,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

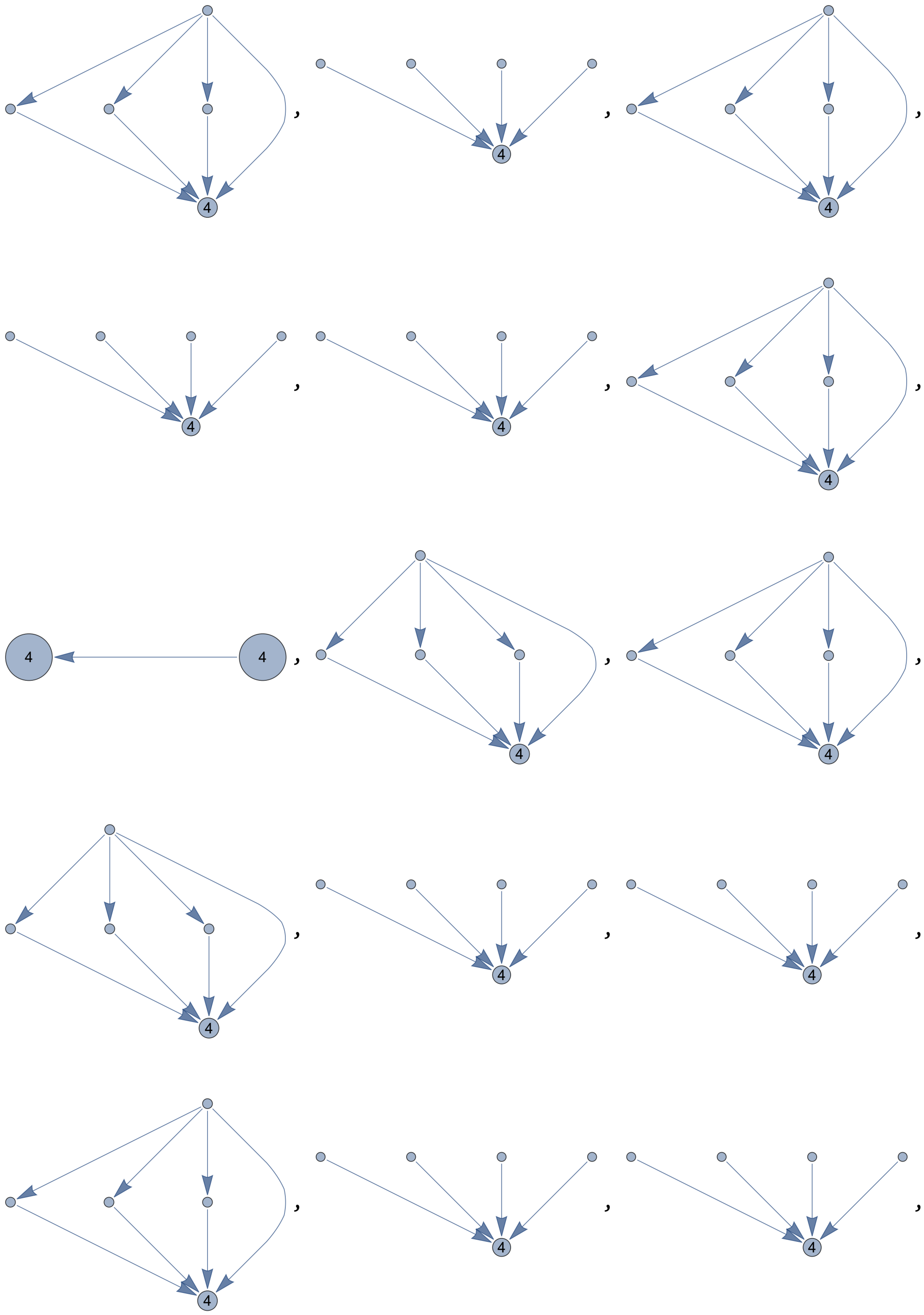
In[ ]:=
```
addGraphProps[gr_] := Module[{filt = (# /. {1 → 0, x_ /; x ≠ 0 → 2} &)},
  gr // IGVertexMap[Sqrt@*Length, "SqLength" → VertexList] /*
    IGVertexMap[Length, "Length" → VertexList] /* IGVertexMap[filt[#⟦1⟧] &,
     "Filter" → VertexList] /* IGVertexMap[MatrixForm[#ᵀ] &, Tooltip → VertexList]]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

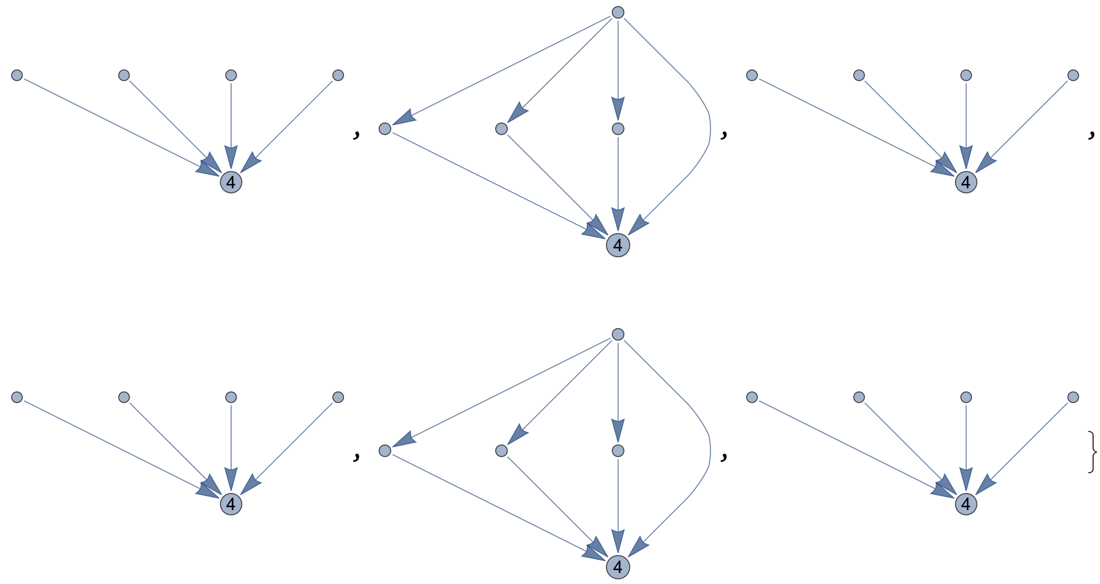
In[ ]:=
```
dispFact[gr_] :=
 gr // IGVertexMap[0.1 # &, VertexSize → IGVertexProp["SqLength"]] /* IGVertexMap[
    Placed[If[# == 1, "", #], Center] &, VertexLabels → IGVertexProp["Length"]]
```

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

Out[ ]=

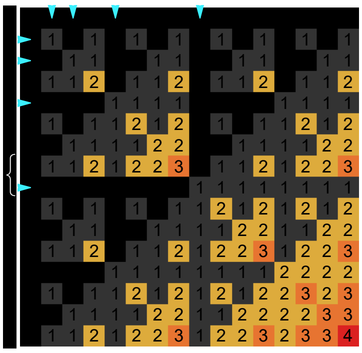







```
In[ ]:= matrixPicture@IdentityMatrix[4]
```

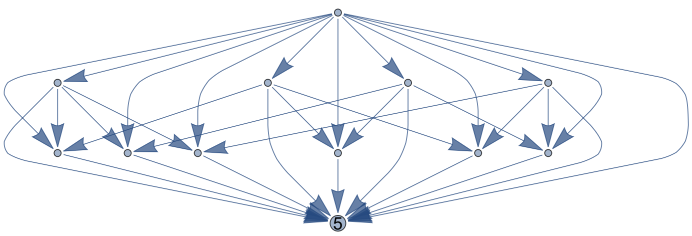
Out[ ]=



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

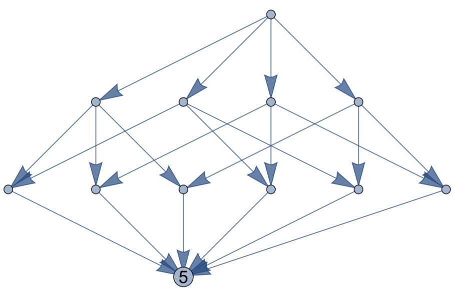
```
In[ ]:= dispFact[grSymFact[IdentityMatrix[4]]]
```

Out[ ]=




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

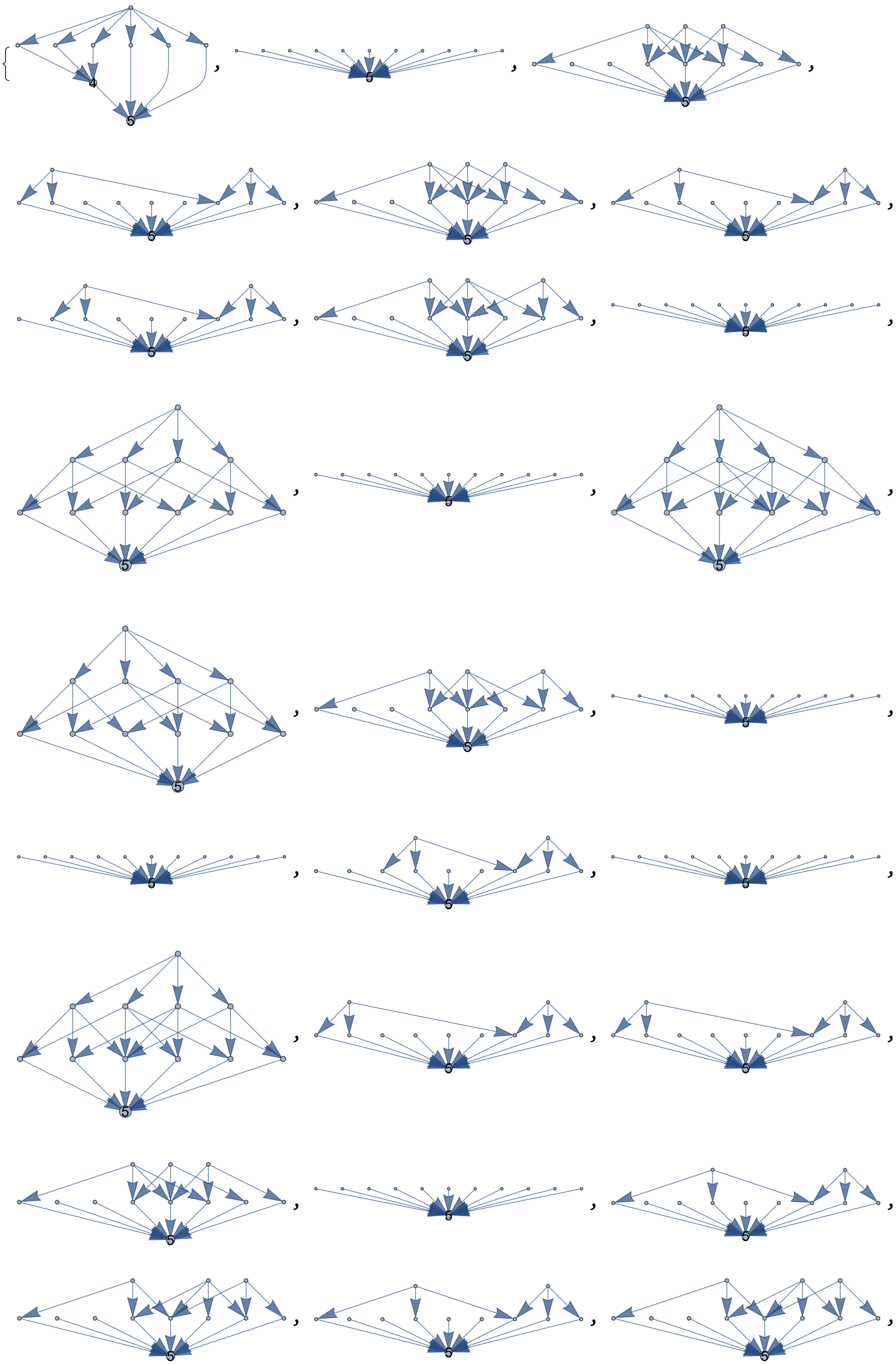
Out[ ]=

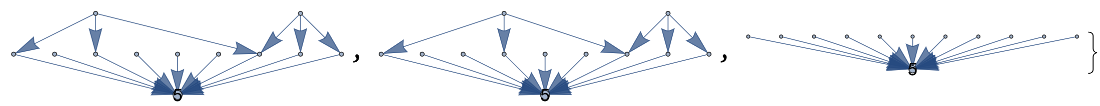


In[ ]:=

```
(dispFact@*hasseSymFact) /@ RandomSample[allSymInvertible[4], 30]
```

Out[ ]=





In[∘]:=

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

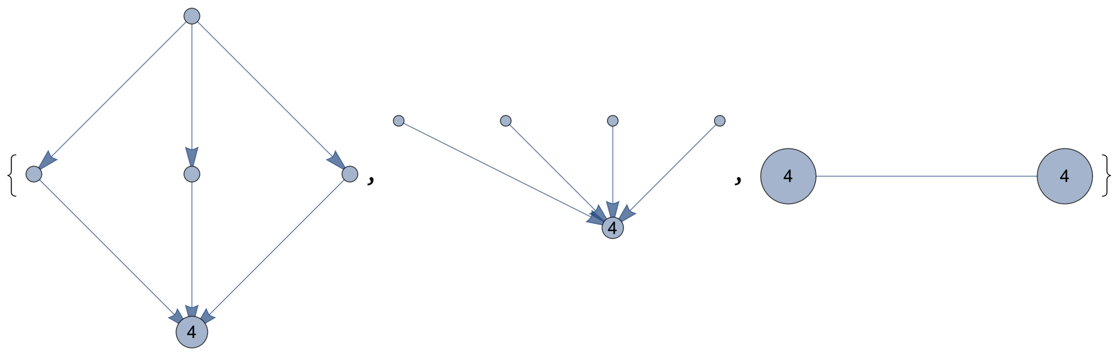
In[∘]:=

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

In[∘]:=

```
allThreeHasse = dispFact /@ hasseSymFact /@ allSymInvertibleDiffGraph[3]
```

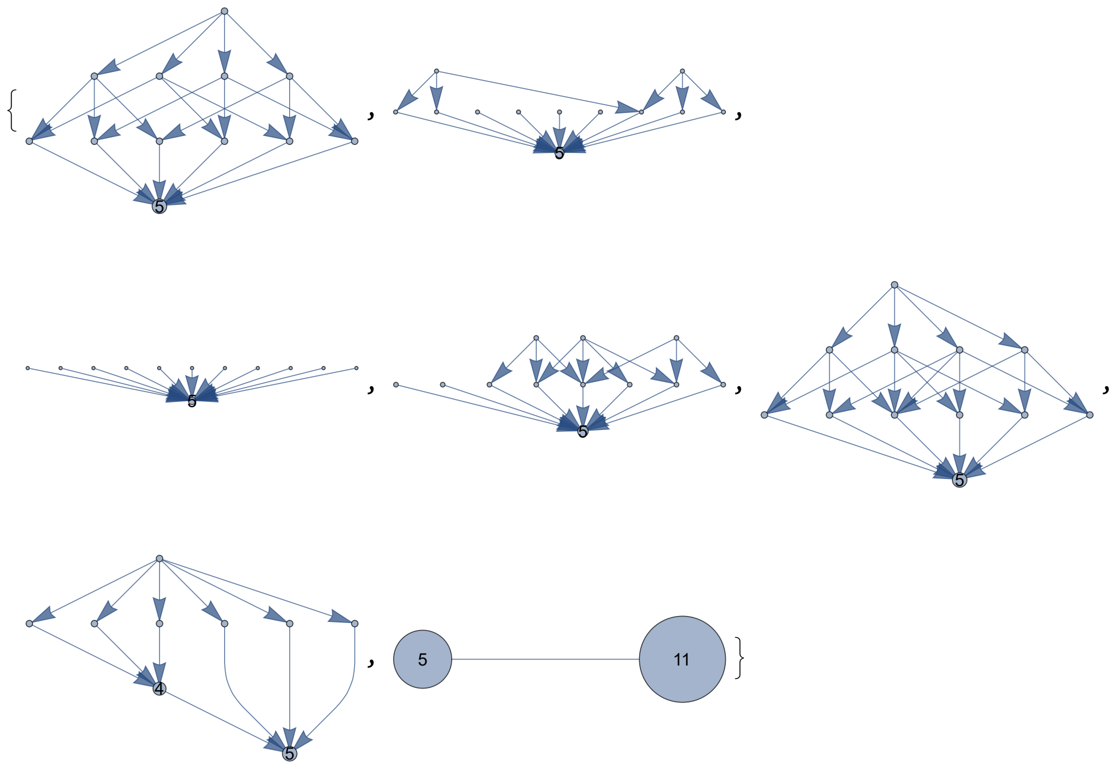
Out[∘]=



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]
```

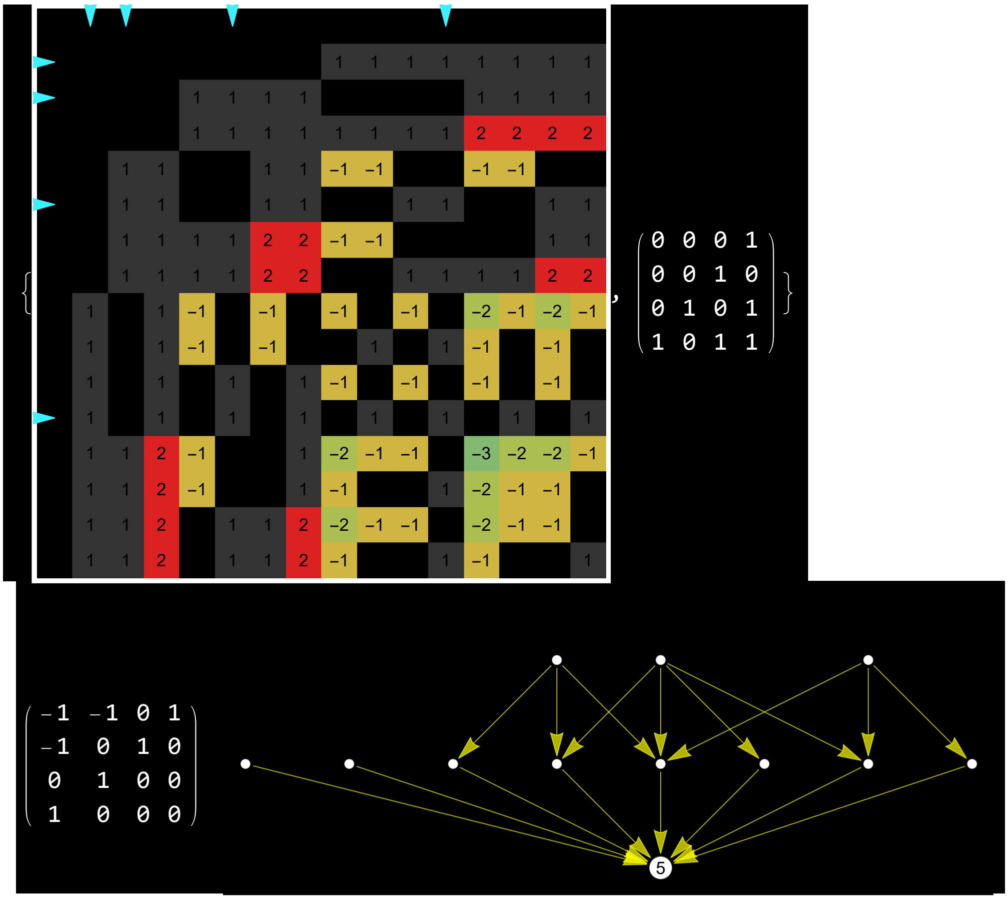
```
In[ ]:= DumpSave[FileNameJoin[{NotebookDirectory[], "all-sym-hasse-3.mx"}], allThreeHasse];
DumpSave[FileNameJoin[{NotebookDirectory[], "all-sym-hasse-4.mx"}], allFourHasse];
DumpSave[FileNameJoin[{NotebookDirectory[], "all-sym-hasse-5.mx"}], allFiveHasse];
```

```
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.(cᵀ)], MatrixForm@c, MatrixForm@ctr, MatrixForm@(cᵀ)}]
```

Unlike the simple cube case there might be more then 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 vecrors:

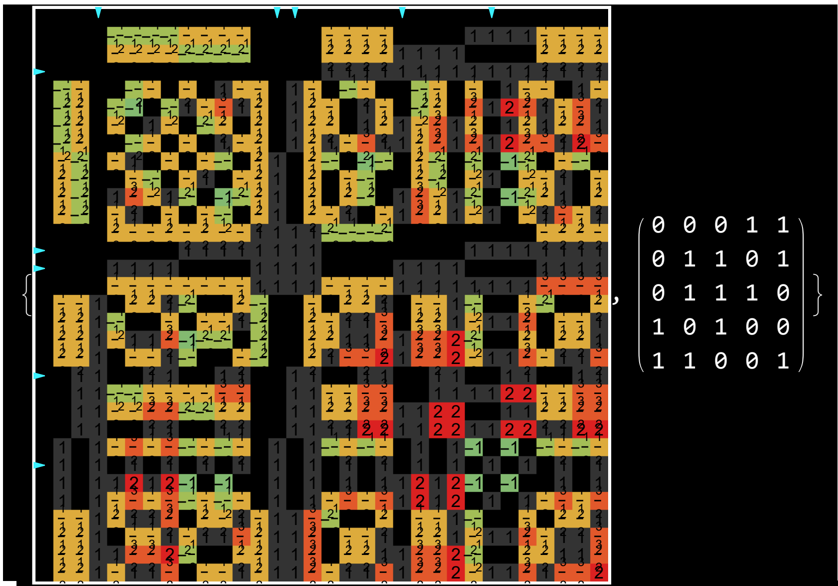
In[ ]:=
```
manySelfCompatibleExample =
 Select[allSymInvertible[5], selfCompatibleCount@# == 24 &, 1] // First
```

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}}
```

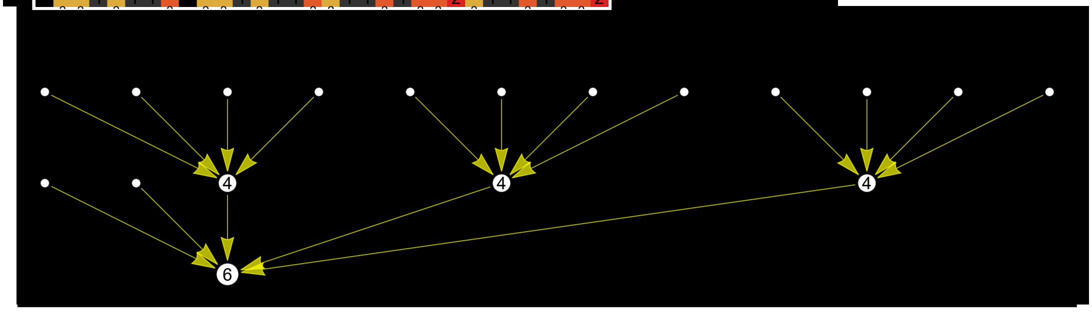
In[ ]:=
```
Labeled[Style[{matrixPicture[manySelfCompatibleExample], Graph[
    setGraphStyle@dispFact@hasseSymFact@manySelfCompatibleExample, ImageSize → 550]},
  Background → Black], Style["5×5 matrix producing 24 self-compatible vectors",
  White, Background → Black], Top]
```

Out[ ]=

5×5 matrix producing 24 self-compatible vectors



$$\left(\begin{array}{ccccc} 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{array}\right)$$



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