oeis-diagrams

Claude Heiland-Allen

2016–2018

Contents

1 2016/A134169.hs

```
-- oeis-diagrams -- unofficial diagrams of OEIS sequences
-- Copyright (C) 2016-2017 Claude Heiland-Allen
-- License CC-BY-NC <https://creativecommons.org/licenses/by-nc/3.0/>

-- https://oeis.org/A134169
-- Let P(A) be the power set of an n-element set A. Then a(n) = the number of
-- pairs of elements {x,y} of P(A) for which either (Case 0) x and y are
-- disjoint, x is not a subset of y, and y is not a subset of x; or (Case 1)
-- x and y are intersecting, but x is not a subset of y, and y is not a subset
-- of x; or (Case 2) x and y are intersecting, and either x is a proper subset
-- of y, or y is a proper subset of x; or (Case 3) x = y.

{-# LANGUAGE FlexibleContexts #-}
```

```
import Prelude hiding (null)

import Data.Bits (bit, finiteBitSize, testBit, (.&.))

import Data.Set (Set)
import qualified Data.Set as S

import Diagrams.Prelude hiding (intersection)
import Diagrams.Backend.PGF.CmdLine (B, defaultMain)

type Z = Int
type S = Int
type P = Set S

intersection :: S -> S -> S
intersection = (.&.)

isSubsetOf :: S -> S -> Bool
x ‘isSubsetOf‘ y = (x ‘intersection‘ y) == x

isProperSubsetOf :: S -> S -> Bool
x ‘isProperSubsetOf‘ y = (x ‘isSubsetOf‘ y) && x /= y

null :: S -> Bool
null x = x == 0

member :: Z -> S -> Bool
member i x = testBit x i

toList :: S -> [Z]
toList x = [ i | i <- [0 .. finiteBitSize x - 1], i ‘member‘ x ]

nset :: Z -> S
nset n = bit n - 1

npower :: Z -> P
npower n = S.fromList [0 .. bit n - 1]

data T = A | B | C | D

t :: S -> S -> Maybe T
t x y
  | y > x = Nothing
  |      null (x ‘intersection‘ y)  && not (x ‘isSubsetOf‘ y) && not (y ‘↙
      ↳ isSubsetOf‘ x) = Just A
  | not (null (x ‘intersection‘ y)) && not (x ‘isSubsetOf‘ y) && not (y ‘↙
      ↳ isSubsetOf‘ x) = Just B
  | not (null (x ‘intersection‘ y)) && ((x ‘isProperSubsetOf‘ y) || (y ‘↙
      ↳ isProperSubsetOf‘ x)) = Just C
  | x == y = Just D
  | otherwise = Nothing

label is x = [ square 2 # strokeP # lc black # fc (if i ‘member‘ x then black ↙
    ↳ else white) # pad 2 | i <- is ]
xlabel s x = vcat $ label (reverse $ toList s) x
ylabel s y = hcat $ label (          toList s) y
```

```
withEnvelope' :: Diagram B -> Diagram B -> Diagram B
withEnvelope' = withEnvelope

cell :: Maybe T -> Diagram B
cell Nothing = withEnvelope' (square 2) mempty
cell (Just A) = circle 1 # strokeP # lc red
cell (Just B) = triangle 2 # centerXY # strokeP # lc green
cell (Just C) = square 2 # strokeP # lc magenta
cell (Just D) = (p2 (-1, -1) ~~ p2 (1, 1) ‘atop‘ p2 (1, -1) ~~ p2 (-1, 1)) # lc ↙
    ↳ blue

diagram n = lwL 0.25 . vcat $
  ( hcat $ (++[withEnvelope' (ylabel s 0) mempty]) [ xlabel s x | x <- S.toList ↙
      ↳ p ] ) :
  [ hcat $ (++[ylabel s y]) [ cell (t x y) # pad 2 | x <- S.toList p ] | y <- S.↙
      ↳ toList p ]
  where
    p = npower n
    s = nset n

key a b c d = vcat
  [ cell (Just D) # pad 2 ||| d
  , cell (Just A) # pad 2 ||| a
  , cell (Just B) # pad 2 ||| b
  , cell (Just C) # pad 2 ||| c
  ] # scale 8

txt = alignedText 0 0.5

main1 :: Z -> IO ()
main1 n = defaultMain $
  let a = txt ”$ x \\cap y   =   \\emptyset \\wedge x \\not\\subseteq y \\wedge ↙
      ↳ x \\not\\supseteq y $”
      b = txt ”$ x \\cap y \\neq \\emptyset \\wedge x \\not\\subseteq y \\wedge ↙
          ↳ x \\not\\supseteq y $”
      c = txt ”$ x \\cap y \\neq \\emptyset \\wedge \\left( x \\subset y \\vee x↙
          ↳  \\supset y \\right) $”
      d = txt ”$ x = y $”
      m = 2^(n - 1) * (2^n - 1) + 1
      count = txt $ ”$ ” ++ show m ++ ” $”
      oeis = alignedText 0 0 ”\\phantom{.} OEIS / A134169”
  in  bg white . pad 1.1 . centerXY $
        alignBR (alignBL (diagram n # centerXY)
        ‘atop‘
        alignBL (key a b c d # centerXY)
        ===
        alignTL ((strutY 1.1 ||| count) # bold # scale 96))
        ‘atop‘
        alignBR (rotate (90 @@ deg) (oeis # bold # scale 8))

main :: IO ()
main = main1 6
```

2 2018/A005905.hs

```
-- oeis-diagrams -- unofficial diagrams of OEIS sequences
-- Copyright (C) 2016-2017 Claude Heiland-Allen
```

```haskell
-- License CC-BY-NC <https://creativecommons.org/licenses/by-nc/3.0/>

-- https://oeis.org/A005905
-- Number of points on surface of truncated tetrahedron: 14n^2 + 2 for n>0,
-- a(0)=1.

-- a(4)  = 226
-- a(12) = 2018

{-# LANGUAGE FlexibleContexts #-}
{-# LANGUAGE TypeFamilies #-}
import Diagrams.Prelude
import Diagrams.Backend.SVG.CmdLine (B, defaultMain)

triUp, triDown, hex :: [(Int, Int)]
triUp   = [ (0,0), (2,0), (1,1) ]
triDown = [ (0,0), (1,1), (-1,1) ]
hex     = [ (0,0), (2,0), (3,1), (2,2), (0,2), (-1,1) ]

net :: [((Int, Int), [(Int, Int)])]
net =
  [ (( 0,0), hex)
  , (( 3,1), hex)
  , (( 6,0), hex)
  , (( 9,1), hex)
  , (( 0,2), triUp)
  , (( 4,0), triDown)
  , (( 6,2), triUp)
  , ((10,0), triDown)
  ]

tabs :: [((Int, Int), (Int, Int))]
tabs =
  [ (( 0,0), ( 2,0))
  , (( 3,1), ( 4,0))
  , (( 5,1), ( 6,0))
  , (( 8,0), ( 9,1))
  , ((10,0), (11,1))
  , ((12,2), (11,3))
  , (( 9,3), ( 8,2))
  , (( 7,3), ( 6,2))
  , (( 5,3), ( 3,3))
  , (( 2,2), ( 1,3))
  , (( 0,2), (-1,1))
  ]

point :: (Int, Int) -> Point V2 Double
point (i, j) = p2 (fromIntegral i, fromIntegral j * sqrt 3)

tabVertices :: Point V2 Double -> Point V2 Double -> [Point V2 Double]
tabVertices p1 p2 = [p1,p4,p3,p2]
  where
    u = scale (1/6) (p2 .-. p1)
    p3 = translate (rotate (-2/6 @@ turn) u) p2
    p4 = translate (rotate (-1/6 @@ turn) u) p1

outline :: Path V2 Double
```

```
outline
  = fromVertices $ concat [ [point a, point b] | (a, b) <- tabs ]

drawTab :: Point V2 Double -> Point V2 Double -> Diagram B
drawTab p1 p2
  = fromVertices (tabVertices p1 p2)
  # mapLoc closeTrail
  # strokeLocTrail
  # lw thin
  # lc black
  # fc (blend 0.5 white grey)

drawShape :: (Int, Int) -> [(Int, Int)] -> Diagram B
drawShape p s
  = fromVertices (map point s)
  # closeTrail
  # (‘at‘ point p)
  # strokeLocTrail
  # lw thin
  # lc black
  # fc (blend 0.5 white (if length s > 3 then blue else red))

drawTabs :: Diagram B
drawTabs
  = mconcat [ drawTab (point a) (point b) | (a, b) <- tabs]

drawNet :: Diagram B
drawNet
  = mconcat [ drawShape p s | (p, s) <- net]

drawPattern :: Int -> Diagram B
drawPattern n
  = mconcat
    [ (circle 0.25 ‘at‘ point (i + (j ‘mod‘ 2), j))
    # translateX 0.25
    # strokeLocTrail
    # lc black
    # fc (blend 0.5 black red)
    | i <- [-n,-n + 2 .. 12 * n]
    , j <- [ 0 ..  3 * n]
    ]
  # scale (1 / fromIntegral (n - 1))
  # clipped outline

diagram :: Int -> Diagram B
diagram n
  = mconcat [ drawPattern n, drawNet, drawTabs ]
  # lineCap LineCapRound
  # lineJoin LineJoinRound
  # centerXY
  # padY (750/433)
  # pad 1.1
  # bg white

main :: IO ()
main = defaultMain (diagram 4)
```

3 2018/A056283.hs

```
-- oeis-diagrams -- unofficial diagrams of OEIS sequences
-- Copyright (C) 2016-2017 Claude Heiland-Allen
-- License CC-BY-NC <https://creativecommons.org/licenses/by-nc/3.0/>

-- https://oeis.org/A056283
-- Number of n-bead necklaces with exactly three different colored beads.

-- a(4) = 30
-- a(8) = 2018

{-# LANGUAGE FlexibleContexts #-}
{-# LANGUAGE TypeFamilies #-}
import Diagrams.Prelude hiding (size, zoom)
import Diagrams.Backend.SVG.CmdLine (B, defaultMain)

import Control.Monad (replicateM)
import Data.List (inits, nub, sort, tails)
import System.Random (newStdGen, randoms)

rotations :: [a] -> [[a]]
rotations xs = zipWith (++) (tails xs) (inits xs)

canonical :: Ord a => [a] -> [a]
canonical xs = minimum (rotations xs)

isCanonical :: Ord a => [a] -> Bool
isCanonical xs = xs == canonical xs

isFull :: Eq a => Int -> [a] -> Bool
isFull k xs = length (nub xs) == k

sequences :: Int -> Int -> [[Int]]
sequences k n = replicateM n [1..k]

necklaces :: Int -> Int -> [[Int]]
necklaces k = filter (\xs -> isCanonical xs && isFull k xs) . sequences k

a056283 :: [Int]
a056283 = map (length . necklaces 3) [1..]

chunk :: Int -> [a] -> [[a]]
chunk _ [] = []
chunk n zs = let (xs, ys) = splitAt n zs in xs : chunk n ys

colours = cycle . map (blend 0.5 black) $ [red, yellow, blue]

bead n m p
  = (circle r `at` p)
  # strokeLocTrail
  # translateX r
  # lc black
  # fc (colours !! m)
  where
    r :: Double
    r = 2 / fromIntegral n
```

```
necklace n xs =
    zipWith (bead n) xs ps # mconcat ‘atop‘
    unitCircle # lc black ‘atop‘
    strutX size ‘atop‘
    strutY (size * sqrt 3 / 2)
  where
    ps = polygon (PolygonOpts (PolyRegular n 1) OrientH origin)

size = 3.5

zoom f d = withEnvelope d (scale f d)

diagram :: [Double] -> Diagram B
diagram g
  = bg white
  . zoom 1.2
  . padX (4 / 3)
  . padY (3 / 2)
  . centerXY
  . vcat
  . zipWith translateX (cycle [0, size / 2])
  . map (hcat . map (necklace n))
  . chunk 6
  . (\xs -> zipWith (\r ys -> cycle (rotations ys) !! floor (fromIntegral n * r)) (drop (length xs) g) xs)
  . map snd . sort . zip g
  $ necklaces k n
  where
    k = 3
    n = 5

main :: IO ()
main = do
  g <- newStdGen
  print g
  defaultMain . diagram . randoms $ g
-- mapM_ print . zip [0..] . takeWhile (<= 2018) $ a056283
```

4 2018/A070211.hs

```
-- oeis-diagrams -- unofficial diagrams of OEIS sequences
-- Copyright (C) 2016-2017 Claude Heiland-Allen
-- License CC-BY-NC <https://creativecommons.org/licenses/by-nc/3.0/>

-- https://oeis.org/A070211
-- Number of compositions (ordered partitions) of n that are concave sequences.
-- Here, a finite sequence is concave if each term (other than the first or
-- last) is at least the average of the two adjacent terms. - Eric M. Schmidt,
-- Sep 29 2013

-- a(8) = 24
-- a(35) = 2018

{-# LANGUAGE FlexibleContexts #-}
import Diagrams.Prelude hiding (zoom)
import Diagrams.Backend.SVG.CmdLine (B, defaultMain)
```

```haskell
import Data.List (sort, transpose)
import System.Random (newStdGen, randoms)

compositions :: Int -> [[Int]]
compositions 0 = [[]]
compositions n =
  [ c
  | m <- [1 .. n]
  , ms <- compositions_memo !! (n - m)
  , let c = m : ms
  , concave c
  ]

compositions_memo :: [[[Int]]]
compositions_memo = map compositions [0..]

concave :: [Int] -> Bool
concave [] = True
concave [_] = True
concave [_,_] = True
concave (a:bs@(b:c:_)) = 2 * b >= a + c

sequences :: Int -> [[Int]]
sequences n = compositions_memo !! n

a070211 :: [Int]
a070211 = map length compositions_memo

chunk :: Int -> [a] -> [[a]]
chunk _ [] = []
chunk n zs = let (xs, ys) = splitAt n zs in xs : chunk n ys

baseColours = [grey, red, blue, yellow]
lightColours = map (blend 0.5 white) baseColours
darkColours = map (blend 0.5 black) baseColours
colours = drop 7 . cycle . concat . transpose $ [lightColours, darkColours]

zoom f d = withEnvelope d (scale f d)

bar s m
  = (rect 0.75 (fromIntegral m :: Double) `atop` strutX 1)
  # scaleX s
  # lc black
  # fc (colours !! m)

barchart n xs
  = map (bar (fromIntegral n / fromIntegral (length xs))) xs
  # hcat
  # centerXY
  `atop` strutX (fromIntegral n)
  `atop` strutY (fromIntegral n)

diagram :: [Double] -> Diagram B
diagram g
  = bg white
  . zoom 1.2
```

```
    . padX (4 / 3)
    . padY (3 / 2)
    . centerXY
    . vcat
    . map (hcat . map (pad 1.2 . barchart n))
    . chunk 6
    . map snd . sort . zip g
    $ sequences n
  where
    n = 8

main :: IO ()
main = do
  g <- newStdGen
  print g
  defaultMain . diagram . randoms $ g
```

5 2018/A118890.hs

```
-- oeis-diagrams -- unofficial diagrams of OEIS sequences
-- Copyright (C) 2016-2017 Claude Heiland-Allen
-- License CC-BY-NC <https://creativecommons.org/licenses/by-nc/3.0/>

-- https://oeis.org/A118890
-- Triangle read by rows: T(n,k) is the number of binary sequences of length n
-- containing k subsequences 0110 (n,k>=0).

-- a(25) = T(11,2) = 142
-- a(38) = T(14,2) = 2018

{-# LANGUAGE FlexibleContexts #-}
import Diagrams.Prelude hiding (zoom)
import Diagrams.Backend.SVG.CmdLine (B, defaultMain)

import Data.List (sort)
import System.Random (newStdGen, randoms)

binary :: Int -> [[Bool]]
binary 0 = [[]]
binary n = [ b:bs | b <- [False, True], bs <- binary_memo !! (n - 1) ]

binary_memo :: [[[Bool]]]
binary_memo = map binary [0..]

count0110 :: [Bool] -> Int
count0110 (False:True:True:rest@(False:_)) = 1 + count0110 rest
count0110 (_:rest) = count0110 rest
count0110 _ = 0

row :: Int -> Int -> [[Bool]]
row k = filter ((k ==) . count0110) . (binary_memo !!)

a118890_t :: Int -> Int -> Int
a118890_t k = length . row k

a118890_nk :: [((Int, Int), Int)]
a118890_nk = concat [ takeWhile ((/= 0) . snd) [ ((n, k), a118890_t k n) | k <- ↙
```

```
    ↳ [0..] ] | n <- [0..] ]

a118890 :: [Int]
a118890 = map snd a118890_nk

chunking :: [Int] -> [a] -> [[a]]
chunking _ [] = []
chunking (n:ns) zs = let (xs, ys) = splitAt n zs in xs : chunking ns ys

colourize :: [Bool] -> [Int]
colourize (False:True:True:False:True:True:False:rest) = [1,1,1,3,2,2,2] ++ map ↙
    ↳ (const 0) (colourize rest)
colourize (False:True:True:False:rest) = [1,1,1,1] ++ map (\x -> if x == 0 then ↙
    ↳ x else 1 + x) (colourize rest)
colourize (_:rest) = 0 : colourize rest

colours = [grey, red, blue, yellow]
colour False = (cycle (map (blend 0.5 white) colours) !!)
colour True  = (cycle (map (blend 0.5 black) colours) !!)

bit b c
  = circle (1 :: Double)
  # centerXY
  # padX 1.2
  # lw thin
  # lc black
  # fc (colour b c)

bits bs = hcat $ zipWith bit bs (colourize bs)

diagram :: [Double] -> Diagram B
diagram g
  = bg white
  . padX (780 / 750)
  . pad 1.2
  . rotate (1/4 @@ turn)
  . centerXY
  . cat' unitX (with & sep .~ sqrt 3)
  . zipWith translateY (cycle [0, 2])
  . map (cat' unitY (with & sep .~ 2) . map bits)
  . chunking [36,35,36,35]
  . map snd . sort . zip g
  $ row k n
  where
    k = 2
    n = 11

main :: IO ()
main = do
  g <- newStdGen
  print g
  defaultMain . diagram . randoms $ g
```

6 2018/A121551.hs

```
-- oeis-diagrams -- unofficial diagrams of OEIS sequences
-- Copyright (C) 2016-2017 Claude Heiland-Allen
```

```haskell
-- License CC-BY-NC <https://creativecommons.org/licenses/by-nc/3.0/>

-- https://oeis.org/A121551
-- Number of parts in all the compositions of n into Fibonacci numbers.

-- a(8)  = 457  (number of composition is 94)
-- a(10) = 2018

{-# LANGUAGE FlexibleContexts #-}
{-# LANGUAGE TypeFamilies #-}
import Diagrams.Prelude
import Diagrams.Backend.SVG.CmdLine (B, defaultMain)

import Data.List (sort, transpose)
import Data.Maybe (fromJust)
import System.Random (newStdGen, randoms)

fibs :: [Int]
fibs = 0 : 1 : zipWith (+) fibs (tail fibs)

compositions :: Int -> [[Int]]
compositions 0 = [[]]
compositions n =
  [ m : ms
  | m <- takeWhile (<= n) (drop 2 fibs)
  , ms <- compositions_memo !! (n - m)
  ]

compositions_memo :: [[[Int]]]
compositions_memo = map compositions [0..]

a121551 :: [Int]
a121551 = map (length . concat) compositions_memo

chunking :: [Int] -> [a] -> [[a]]
chunking _ [] = []
chunking (n:ns) zs = let (xs, ys) = splitAt n zs in xs : chunking ns ys

baseColours = [grey, red, blue, yellow]
lightColours = map (blend 0.5 white) baseColours
darkColours = map (blend 0.5 black) baseColours
colours = [ blend 0.5 white grey, blend 0.5 black red, blend 0.5 white blue, ↙
    ↳ blend 0.5 black blue, blend 0.5 white red]

colour n = fromJust . lookup n . zip (drop 2 fibs) $ colours

bar s n
  = rect (s * fromIntegral n) 1
  # lc black
  # fc (colour n)

bars ns = centerXY . cat' unitX (with & sep .~ d) . map (bar s) $ ns
  where
    d = 0.5
    n = sum ns
    m = length ns
    s = fromIntegral n / (fromIntegral (m - 1) * d + fromIntegral n)
```

```
diagram :: [Double] -> Diagram B
diagram g
  = bg white
  . padX (779 / 750)
  . pad 1.2
  . centerXY
  . cat' unitX (with & sep .~ sqrt 3)
  . zipWith translateY (4 : cycle [2, 0])
  . map (cat' unitY (with & sep .~ 2) . map bars)
  . chunking [15, 16, 16, 16, 16, 15]
  . map snd . sort . zip g
  $ compositions_memo !! 8

main :: IO ()
main = do
  g <- newStdGen
  print g
  defaultMain . diagram . randoms $ g
--  mapM_ print . zip [0..] . takeWhile (<= 2018) $ a121551
```

7 2018/A124255.hs

```
-- oeis-diagrams -- unofficial diagrams of OEIS sequences
-- Copyright (C) 2016-2017 Claude Heiland-Allen
-- License CC-BY-NC <https://creativecommons.org/licenses/by-nc/3.0/>

-- https://oeis.org/A124255
-- Forest-and-trees problem: square of distance to most distant visible tree.

-- a(8)  = 61   (most distant tree at (5, 6))
-- a(45) = 2018

{-# LANGUAGE FlexibleContexts #-}
import Diagrams.Prelude
import Diagrams.Backend.SVG.CmdLine (B, defaultMain)

import Data.Maybe (catMaybes)

n :: Double
n = 8

resolution :: Int
resolution = 4096

trees1 :: Diagram B
trees1 = mconcat
  [ (circle (1 / n) `at` p2 (i + 1 / n, j)) # strokeLocTrail
  | i <- [-n .. n]
  , j <- [-n .. n]
  , i /= 0 || j /= 0
  ]

trees2 :: Diagram B
trees2 = mconcat
  [ (circle (1 / n) `at` p2 (i + 1 / n, j))
  # strokeLocTrail
  # lw thin
  # if visible (p2(i, j))
    then lc black . fc (blend 0.5 black red)
    else lc (blend 0.5 black grey) . fc (blend 0.5 white grey)
  | i <- [-n .. n]
  , j <- [-n .. n]
  , i /= 0 || j /= 0
  ]

visible p = any ((< 0.5) . distance p) boundary

boundary = catMaybes
  [ rayTraceV origin (angleV $ t @@ turn) trees1
  | i <- [0 .. resolution - 1]
  , let t = fromIntegral i / fromIntegral resolution
  ] # map (‘translate‘ origin)

visibility :: Diagram B
visibility
  = boundary
  # fromVertices
  # mapLoc closeTrail
  # strokeLocTrail
  # lw veryThin
  # lc black
  # fc (blend 0.5 white blue)

observer
  = circle (1 /n)
  # lw thin
  # lc black
  # fc (blend 0.5 black blue)

diagram :: Diagram B
diagram
  = bg white
  . pad 1.2
  . padX (4 / 3)
  $ observer ‘atop‘ trees2 ‘atop‘ visibility

main :: IO ()
main = defaultMain diagram
```

8 2018/A133736.hs

```
-- oeis-diagrams -- unofficial diagrams of OEIS sequences
-- Copyright (C) 2016-2017 Claude Heiland-Allen
-- License CC-BY-NC <https://creativecommons.org/licenses/by-nc/3.0/>

-- https://oeis.org/A133736
-- Number of graphs on n unlabeled nodes that have an Eulerian cycle, i.e., a
-- cycle that goes through every edge in the graph exactly once.

-- a(6) = 15
-- a(9) = 2018

{-# LANGUAGE FlexibleContexts #-}
```

```haskell
{-# LANGUAGE TypeFamilies #-}
import Diagrams.Prelude
import Diagrams.Backend.SVG.CmdLine (B, defaultMain)

import Data.List (sort)
import System.Random (newStdGen, randoms)

chunk :: Int -> [a] -> [[a]]
chunk _ [] = []
chunk n zs = let (xs, ys) = splitAt n zs in xs : chunk n ys

cyclic ps = (ps, zip ps (drop 1 (cycle ps)))
bicyclic ps = zip ps (drop 2 (cycle ps))

graphs =
  [ case [origin] of ps -> (ps,[])
  , cyclic $ triangle (s*2)
  , cyclic $ square (s*2)
  , cyclic $ pentagon 2
  , case origin : square (s*2) of
      ps@[o,a,b,c,d] -> (ps, [(o,a),(o,b),(o,c),(o,d),(a,b),(c,d)])
  , case map p2 [(0,s),(0,-s),(-s,0),(sqrt 6 / 2,0),(sqrt 6, 0)] of
      ps@[u,d,l,m,r] -> (ps, [(u,l),(u,m),(u,r),(u,d),(d,l),(d,m),(d,r)])
  , case cyclic $ pentagon 2 of (ps, es) -> (ps, es ++ bicyclic ps)
  , cyclic $ hexagon 2
  , case square (s*2) ++ [origin, p2(s*2,0)] of
      ps@[a,b,c,d,o,r] -> (ps, [(c,d),(a,r),(b,r),(o,a),(o,b),(o,c),(o,d)])
  , case square (s*2) ++ [origin, p2(s*2,0)] of
      ps@[a,b,c,d,o,r] -> (ps, [(a,b),(b,c),(c,d),(d,a),(a,r),(b,r),(o,a),(o,b)])
  , case map p2 [(0,s),(0,-s),(-1.5*s,0),(-0.5*s,0),(0.5*s,0),(1.5*s,0)] of
      ps@[u,l,a,b,c,d] -> (ps, map ((,) u) [a,b,c,d] ++ map ((,) l) [a,b,c,d])
  , case cyclic $ hexagon 2 of
      (ps@[a,b,c,d,e,f], es) -> (ps, es ++ [(a,c),(c,e),(e,a)])
  , case hexagon 2 of
      ps@[a,r,b,c,l,d] -> (ps, [(a,b),(b,c),(c,d),(d,a),(a,c),(b,d),(a,r),(b,r),(c,l),(d,l)])
  , case cyclic $ hexagon 2 of
      (ps@[a,b,c,d,e,f], es) -> (ps, es ++ bicyclic (drop 1 ps))
  , case cyclic $ hexagon 2 of (ps, es) -> (ps, es ++ bicyclic ps)
  ]
  where
    s = sqrt 2

node es p
  = (circle r `at` p)
  # translateX r
  # strokeLocTrail
  # lc black
  # fc (blend 0.5 white ([yellow, blue, red] !! (multiplicity es p `div` 2)))
  where
    r = 0.5

multiplicity es p = length . filter (p ==) $ map fst es ++ map snd es

edge (p, q) = p ~~ q # lc black
```

```
graph :: ([P2 Double], [(P2 Double, P2 Double)]) -> Diagram B
graph (ps, es)
  = (mconcat (map (node es) ps) `atop` mconcat (map edge es))
  # centerXY `atop` strutX 6 `atop` strutY 6

diagram :: [Double] -> Diagram B
diagram g
  = bg white
  . pad 1.15
  . padY (750/600)
  . centerXY
  . vcat
  . map hcat
  . chunk 5
  . map graph
  . map snd . sort . zip g
  $ graphs

main :: IO ()
main = do
  g <- newStdGen
  print g
  defaultMain . diagram . randoms $ g
```

9 2018/A149037.hs

```
-- oeis-diagrams -- unofficial diagrams of OEIS sequences
-- Copyright (C) 2016-2017 Claude Heiland-Allen
-- License CC-BY-NC <https://creativecommons.org/licenses/by-nc/3.0/>

-- https://oeis.org/A149037
-- Number of walks within N^3 (the first octant of Z^3) starting at (0,0,0) and
-- consisting of n steps taken from {(-1, -1, 0), (-1, 0, 1), (0, 1, 1),
-- (1, -1, 1), (1, 0, -1)}.

-- a(4) = 35
-- a(7) = 2018

{-# LANGUAGE FlexibleContexts #-}
import Diagrams.Prelude hiding (project, translation)
import Diagrams.Backend.SVG.CmdLine (B, defaultMain)

import Control.Monad (guard)
import Data.List (sortOn, transpose)
import System.Random (newStdGen, randoms)

steps :: [V3 Int]
steps = [v (-1) (-1) 0, v (-1) 0 1, v 0 1 1, v 1 (-1) 1, v 1 0 (-1)]
  where
    v = mkR3

walk :: V3 Int -> [V3 Int]
walk p = do
  step <- steps
  p <- pure $ p ^+^ step
  guard (view _x p >= 0)
  guard (view _y p >= 0)
  guard (view _z p >= 0)
  pure p

extend :: [V3 Int] -> [[V3 Int]]
extend ps@(p:_) = do
  q <- walk p
  pure (q:ps)

initial :: [[V3 Int]]
initial = [[mkR3 0 0 0]]

walks :: [[[V3 Int]]]
walks = iterate (concatMap extend) initial

grid :: [V3 Int]
grid = [ mkR3 x y z | x <- [0 .. 3], y <- [0 .. 3], z <- [0 .. 3] ]

chunk :: Int -> [a] -> [[a]]
chunk _ [] = []
chunk n zs = let (xs, ys) = splitAt n zs in xs : chunk n ys

project r p = (z - 12, p2(u, v))
  where
    x = fromIntegral $ view _x p
    y = fromIntegral $ view _y p
    z = fromIntegral $ view _z p
    q = [x, y, z, 1]
    [a,b,c,d] = transformation r ‘mv‘ q
    [u,v,w] = map (/d) [a,b,c]

transformation :: Bool -> [[Double]]
transformation r
  = perspective ‘mm‘
    translation 0 0 (-10) ‘mm‘
    rotationX (-0.2) ‘mm‘
    rotationY (if r then 1 + 0.1 else 1 - 0.1) ‘mm‘
    translation (-2) (-2) (-2)

perspective :: [[Double]]
perspective
  = [ [ 1 / (ratio * tan_half_angle), 0, 0, 0 ]
    , [ 0, 1 / tan_half_angle, 0, 0 ]
    , [ 0, 0, -(far + near) / (far - near), -2 * (far * near) / (far - near) ]
    , [ 0, 0, -1, 0]
    ]
  where
    tan_half_angle = tan (angle / 2)
    near = 0.1
    far = 20
    angle = pi / 8
    ratio = 1

rotationX :: Double -> [[Double]]
rotationX t
  = [ [ 1, 0, 0, 0 ], [ 0, c, s, 0 ], [ 0, -s, c, 0 ], [ 0, 0, 0, 1 ] ]
  where
    c = cos t
```

```haskell
    s = sin t

rotationY :: Double -> [[Double]]
rotationY t
  = [ [ c, 0, s, 0 ], [ 0, 1, 0, 0 ], [ -s, 0, c, 0 ], [ 0, 0, 0, 1 ] ]
  where
    c = cos t
    s = sin t

translation :: Double -> Double -> Double -> [[Double]]
translation x y z
  = [ [ 1, 0, 0, x ]
    , [ 0, 1, 0, y ]
    , [ 0, 0, 1, z ]
    , [ 0, 0, 0, 1 ]
    ]

vv :: [Double] -> [Double] -> Double
vv a b = sum $ zipWith (*) a b

mv :: [[Double]] -> [Double] -> [Double]
mv a b = map (`vv` b) a

mm :: [[Double]] -> [[Double]] -> [[Double]]
mm a b = [ [ u `vv` v | v <- transpose b ] | u <- a ]

path :: [V3 Int] -> Diagram B
path w
  = pad 1.2
  . withEnvelope' (centerXY $ fromVertices
      (map (snd . project False) grid ++ map (snd . project True) grid))
  . centerXY
  $ fromVertices (map (snd . project False) w) # lc (blend 0.5 black red) `atop`
    fromVertices (map (snd . project True ) w) # lc (blend 0.5 black blue)

withEnvelope' :: Diagram B -> Diagram B -> Diagram B
withEnvelope' = withEnvelope

diagram :: [Double] -> Diagram B
diagram g
  = bg white
  . pad 1.1
  . padY (750/663)
  . centerXY
  . vcat
  . map (hcat . map path)
  . chunk 7
  . map snd . sortOn fst . zip g
  $ walks !! 4

main :: IO ()
main = do
  g <- newStdGen
  print g
  defaultMain . diagram . randoms $ g
```

10 2018/A213497.hs

```
-- oeis-diagrams -- unofficial diagrams of OEIS sequences
-- Copyright (C) 2016-2017 Claude Heiland-Allen
-- License CC-BY-NC <https://creativecommons.org/licenses/by-nc/3.0/>

-- https://oeis.org/A213497
-- Number of (w,x,y) with all terms in {0,...,n} and w = min(|w-x|,|x-y|).

-- a(5)  = 40
-- a(40) = 2018

{-# LANGUAGE FlexibleContexts #-}
import Diagrams.Prelude
import Diagrams.Backend.SVG.CmdLine (B, defaultMain)

import Data.List (sort)
import System.Random (newStdGen, randoms)

tuples :: Int -> [(Int, Int, Int)]
tuples n =
  [ (w, x, y)
  | w <- [0 .. n]
  , x <- [0 .. n]
  , y <- [0 .. n]
  , w == min (abs (w - x)) (abs (x - y))
  ]

chunk :: Int -> [a] -> [[a]]
chunk _ [] = []
chunk n zs = let (xs, ys) = splitAt n zs in xs : chunk n ys

colours =
  [ l grey
  , d grey
  , l blue
  , l red
  , d blue
  , d red
  ]
  where
    d = blend 0.5 white
    l = blend 0.5 black

draw :: Int -> (Int, Int, Int) -> Diagram B
draw n (w, x, y)
  = (circle (r w) # fc (colours !! w) |||
     circle (r x) # fc (colours !! x) |||
     circle (r y) # fc (colours !! y))
  # centerXY
  # atop (strutX (4 * r n + 6))
  # atop (strutY (2 * r n + 6))
  # rotate (1/7 @@ turn)
  where
    r t = 0.5 + fromIntegral t

diagram :: [Double] -> Diagram B
diagram g
  = bg white
  . pad 1.2
  . padX (777/750)
  . centerXY
  . vcat
  . map hcat
  . chunk 8
  . map (draw n)
  . map snd . sort . zip g
  $ tuples n
  where
    n = 5

main :: IO ()
main = do
  g <- newStdGen
  print g
  defaultMain . diagram . randoms $ g
```

11 2018/A229915.hs

```
-- oeis-diagrams -- unofficial diagrams of OEIS sequences
-- Copyright (C) 2016-2017 Claude Heiland-Allen
-- License CC-BY-NC <https://creativecommons.org/licenses/by-nc/3.0/>

-- https://oeis.org/A229915
-- Number of espalier polycubes of a given volume in dimension 3.

-- a(7)  = 34
-- a(20) = 2018

{-# LANGUAGE FlexibleContexts #-}
import Diagrams.Prelude hiding (cube)
import Diagrams.Backend.SVG.CmdLine (B, defaultMain)

import Control.Monad (guard)
import Data.List (nub, sort, sortOn)
import Data.Tuple (swap)
import System.Random (newStdGen, randoms)

partitions :: Int -> [[Int]]
partitions 0 = [[]]
partitions n
  = nub [ sort $ m : ms | m <- [1..n], ms <- partitions_memo !! (n - m) ]

partitions_memo :: [[[Int]]]
partitions_memo = map partitions [0..]

espaliers n = do
  a <- concat $ take (2 * n) partitions_memo
  b <- concat $ take (2 * n) partitions_memo
  guard (not (null a))
  guard (not (null b))
  guard (length a == length b)
  e <- pure $ combine (ferrers a) (ferrers b)
  guard (length e == n)
```

```haskell
  pure e

ferrers = concat . zipWith (\i j -> map ((,) i) [1..j]) [1..] . reverse

combine as bs
  = nub $ sort [ (i, j, k) | (i, j) <- as, (i', k) <- bs, i == i' ]

a229915 = map (length . nub . map sort . espaliers) [0..]

chunking :: [Int] -> [a] -> [[a]]
chunking _ [] = []
chunking (n:ns) zs = let (xs, ys) = splitAt n zs in xs : chunking ns ys

withEnvelope' :: Diagram B -> Diagram B -> Diagram B
withEnvelope' = withEnvelope

cube (i, j, k)
  = translateX x
  . translateY y
   $ rhombus grey [o,a,b,c] `atop`
     rhombus red  [o,c,d,e] `atop`
     rhombus blue [o,e,f,a]
  where
    x = fromIntegral (j - k) * sqrt 3 / 2
    y = fromIntegral i - fromIntegral (j + k) * 0.5
    [o,a,b,c,d,e,f] = origin : hexagon 1 # map (rotate (1.5 / 6 @@ turn))
    rhombus c xs
      = fromVertices xs
      # mapLoc closeTrail
      # strokeLocTrail
      # lc black
      # fc (blend 0.5 white c)

draw
  = centerXY
  . mconcat
  . map cube
  . sortOn (\(i, j, k) -> (-i, -j, -k))


diagram :: [Double] -> Diagram B
diagram g
  = bg white
  . pad 1.2
  . padY (750/739)
  . centerXY
  . lineCap LineCapRound
  . lineJoin LineJoinRound
  . vcat
  . map (centerXY . hcat)
  . chunking [7,7,6,7,7]
  . map (withEnvelope' env . draw)
  . map snd . sort . zip g
  $ espaliers n
  where
    n = 7
    env = centerXY . mconcat . map draw . espaliers $ n
```

```
main :: IO ()
main = do
  g <- newStdGen
  print g
  defaultMain . diagram . randoms $ g
--  mapM_ print . zip [0..] . takeWhile (<= 2018) $ a229915
```

12 2018/A240059.hs

```
-- oeis-diagrams -- unofficial diagrams of OEIS sequences
-- Copyright (C) 2016-2017 Claude Heiland-Allen
-- License CC-BY-NC <https://creativecommons.org/licenses/by-nc/3.0/>

-- https://oeis.org/A240059
-- Number of partitions of n such that m(1) > m(3), where m = multiplicity.

-- a(12) = 46
-- a(27) = 2018

{-# LANGUAGE FlexibleContexts #-}
{-# LANGUAGE TypeFamilies #-}
import Diagrams.Prelude
import Diagrams.Backend.SVG.CmdLine (B, defaultMain)

import Data.List (nub, sort, zipWith4)
import System.Random (newStdGen, randoms)

partitions :: Int -> [[Int]]
partitions 0 = [[]]
partitions n
  = nub [ sort $ m : ms | m <- [1..n], ms <- partitions_memo !! (n - m) ]

partitions_memo :: [[[Int]]]
partitions_memo = map partitions [0..]

multiplicity :: Int -> [Int] -> Int
multiplicity k = length . filter (k ==)

a240059_ps :: [[[Int]]]
a240059_ps
  = map (filter (\xs -> multiplicity 1 xs > multiplicity 3 xs)) partitions_memo

a240059 = map length a240059_ps

chunking :: [Int] -> [a] -> [[a]]
chunking _ [] = []
chunking (n:ns) zs = let (xs, ys) = splitAt n zs in xs : chunking ns ys

colour' b = blend 0.5 (if b then black else white)
colour 1 b = colour' b red
colour 3 b = colour' b blue
colour _ b = colour' b grey

pieChart :: Int -> [Int] -> Diagram B
pieChart n xs
```

```
  = mconcat ws
  # centerXY
  # pad padding
  where
    ys = scanl (+) 0 xs
    ts = map (\y -> fromIntegral y / fromIntegral n) ys
    ws = zipWith4 w ts (tail ts) xs (cycle [True, False])
    w lo hi m b
      = wedge 1 (rotate (lo @@ turn) xDir) ((hi - lo) @@ turn)
      # lc black
      # lineCap LineCapRound
      # lineJoin LineJoinRound
      # fc (colour m b)

padding = 1.2

diagram :: [Double] -> Diagram B
diagram g
  = bg white
  . pad padding
  . padY (750/706)
  . centerXY
  . vcat
  . zipWith translateX ([padding, 0, padding, 0, padding, 2 * padding])
  . map hcat
  . chunking [7, 8, 8, 8, 8, 7]
  . map (pieChart n)
  . map snd . sort . zip g
  $ a240059_ps !! n
  where
    n = 12

main :: IO ()
main = do
  g <- newStdGen
  print g
  defaultMain . diagram . randoms $ g
-- mapM_ print . zip [0..] . takeWhile (<= 2018) $ a240059
```

13 2018/A267255.hs

```
-- oeis-diagrams -- unofficial diagrams of OEIS sequences
-- Copyright (C) 2016-2017 Claude Heiland-Allen
-- License CC-BY-NC <https://creativecommons.org/licenses/by-nc/3.0/>

-- https://oeis.org/A267255
-- Decimal representation of the n-th iteration of the "Rule 111" elementary
-- cellular automaton starting with a single ON (black) cell.

-- a(5) = 2018

{-# LANGUAGE FlexibleContexts #-}
import Diagrams.Prelude
import Diagrams.Backend.SVG.CmdLine (B, defaultMain)

import Control.Monad (replicateM)
```

```
data Cell = O | I

rule :: [Cell] -> [Cell]
rule (I:xs@(I:I:_)) = O : rule xs
rule (I:xs@(I:O:_)) = I : rule xs
rule (I:xs@(O:I:_)) = I : rule xs
rule (I:xs@(O:O:_)) = O : rule xs
rule (O:xs@(_:_:_)) = I : rule xs
rule _ = []

initial :: (Cell, [Cell])
initial = (O, [I])

step :: (Cell, [Cell]) -> (Cell, [Cell])
step (c, xs) = (case c of I -> O ; O -> I, rule ([c,c] ++ xs ++ [c,c]))

history :: [[Cell]]
history = map snd . take 15 $ iterate step initial

cell :: Bool -> Cell -> Diagram B
cell b O = circle 1 # pad 1.2 # lc black # fc (colour b blue)
cell b I = circle 1 # pad 1.2 # lc black # fc (colour b red)

colour b = blend 0.5 (if b then black else white)

key :: [Cell] -> Diagram B
key xs@[a,b,c]
  = (centerX (cell False a ||| cell False b ||| cell False c)
     ===
     centerX (cell True (head (rule xs))))
  # centerXY # padX 1.2

legend :: Diagram B
legend = centerXY . hcat . map key . replicateM 3 $ [I,O]

diagram
  = bg white
  . pad 1.2
  . padY (750/672)
  . centerXY
  . (legend ===)
  . (strutY 6 ===)
  . centerXY
  . vcat
  . zipWith (\n -> centerX . hcat . map (cell (n == 5))) [0..]
  $ history

main = defaultMain diagram
```

14 2018/A271996.hs

```
-- oeis-diagrams -- unofficial diagrams of OEIS sequences
-- Copyright (C) 2016-2017 Claude Heiland-Allen
-- License CC-BY-NC <https://creativecommons.org/licenses/by-nc/3.0/>

-- https://oeis.org/A271996
-- The crystallogen sequence (a(n) = A018227(n)-4).
-- a(7)  = 114
-- a(21) = 2018

{-# LANGUAGE FlexibleContexts #-}
{-# LANGUAGE TypeFamilies #-}
import Diagrams.Prelude
import Diagrams.Backend.SVG.CmdLine (B, defaultMain)

shells = [2, 8, 8, 18, 18, 32, 32]
isLast = map (const False) (drop 1 shells) ++ [True]

nucleus = circle 0.25 # lc black # fc (blend 0.5 black blue)

hole p = (circle 0.25 ‘at‘ p) # translateX 0.25 # strokeLocTrail # fc (blend 0.5 white red)

electron p = (circle 0.125 ‘at‘ p) # translateX 0.125 # strokeLocTrail # fc (blend 0.5 white blue)

shell :: Int -> Int -> Bool -> Diagram B
shell i n b
  = (if odd i then rotate ((0.5 / fromIntegral n) @@ turn) else id)
  $ mconcat (zipWith ($) (if b then concat $ replicate 4 (hole : replicate ((n - 4) ‘div‘ 4) electron) else replicate n electron)
  (polygon (PolygonOpts (PolyRegular n r) NoOrient origin))) # lc black
  ‘atop‘ circle r # lc black
  where
    r = 4 * (fromIntegral (i + 2) / fromIntegral (length shells + 1))

diagram :: Diagram B
diagram
  = bg white
  . pad 1.2
  . padX (1000/750)
  . rotate (1 / 3 @@ turn)
  . centerXY
  . atop nucleus
  . mconcat
  $ zipWith3 shell [0..] shells isLast

main = defaultMain diagram
```

15 35/A000292.hs

```
-- oeis-diagrams -- unofficial diagrams of OEIS sequences
-- Copyright (C) 2016-2017 Claude Heiland-Allen
-- License CC-BY-NC <https://creativecommons.org/licenses/by-nc/3.0/>

-- https://oeis.org/A000292
-- The number of (n+2)-bit numbers which contain two runs of 1's in their binary
-- expansion. - Vladimir Shevelev, Jul 30 2010

{-# LANGUAGE FlexibleContexts #-}
import Diagrams.Prelude hiding (parts, size)
import Diagrams.Backend.SVG.CmdLine (B, defaultMain)
import Diagrams.TwoD.Arc (arcT)
import Data.List (sort, transpose)
import Data.List.Split (chunksOf)

parts n =
  [ draw
      [ replicate b True
      , replicate c False
      , replicate d True
      , replicate e False
      ]
  ‘atop‘ strutXY (size * 5)
  | a <- [0]
  , b <- [1 .. n - 1 - a]
  , c <- [1 .. n - 1 - a - b]
  , d <- [1 .. n - a - b - c]
  , let e = n - a - b - c - d
  ]

size :: Double
size = 3

grid = vcat . map hcat

cell True  = circle 1 # fcA (red ‘withOpacity‘ 0.5) # lc black ‘atop‘ strutXY size
cell False = circle 1 # fc white # lc black ‘atop‘ strutXY size

draw = centerXY . rotate (1/5 @@ turn) . centerXY . vcat . map (centerXY . (‘atop‘ strutXY size) . hcat . map cell)

strutXY x = strutX x ‘atop‘ strutY x

diagram m n
  = lw thick
  . bg white
  . centerXY
  . grid
  . chunksOf m
  $ parts (n + 2)

main = defaultMain (diagram 5 5)
```

16 69/A003269.hs

```
-- oeis-diagrams -- unofficial diagrams of OEIS sequences
-- Copyright (C) 2016-2017 Claude Heiland-Allen
-- License CC-BY-NC <https://creativecommons.org/licenses/by-nc/3.0/>

-- https://oeis.org/A003269
-- a(n+1) is the number of compositions of n into parts 1 and 4. - Joerg Arndt,
-- Jun 25 2011

{-# LANGUAGE TypeFamilies #-}

import Diagrams.Prelude hiding (parts)
import Diagrams.Backend.SVG.CmdLine (B, defaultMain)
import Data.List (sortBy)
import Data.Monoid ((<>))
import Data.Ord (comparing)

parts = [1, 4]

compositions' total
  | total < 0 = []
  | total == 0 = [[]]
  | otherwise = [ part : rest | part <- parts, rest <- compositions (total - ↙
      ↪ part) ]

compositions total = filter ((total ==) . sum) (compositions' total)

diagram1 :: [Int] -> Diagram B
diagram1 is = mconcat [strokeLocTrail $ circle (0.125) `at` v | v <- vs] # lw ↙
    ↪ thin # fc black `atop` head vs ~~ last vs `atop` strutY (1/3)
  where
    vs = map (\x -> p2 (fromIntegral x, 0)) (scanl (+) 0 is)

diagram :: Int -> Diagram B
diagram = bg white . frame 2 . centerXY . vcat . map diagram1 . compositions

main = defaultMain (diagram 15)
```

17 70/A000129.hs

```
-- oeis-diagrams -- unofficial diagrams of OEIS sequences
-- Copyright (C) 2016-2017 Claude Heiland-Allen
-- License CC-BY-NC <https://creativecommons.org/licenses/by-nc/3.0/>

-- https://oeis.org/A000129
-- a(n) is the number of compositions (ordered partitions) of n-1 into two sorts
-- of 1's and one sort of 2's.  Example: the a(3)=5 compositions of 3-1=2 are
-- 1+1, 1+1', 1'+1, 1'+1', and 2. - Bob Selcoe, Jun 21 2013

{-# LANGUAGE FlexibleContexts #-}
import Diagrams.Prelude
import Diagrams.Backend.SVG.CmdLine (B, defaultMain)

import Control.Monad (replicateM)
import Data.List (sort, transpose)
import Data.List.Split (chunksOf)

u, d, h, z :: (Int, Int)
u = (1, 1)
d = (1, -1)
h = (2, 0)
z = (0, 0)

add (a, b) (c, d) = (a + c, b + d)

v :: (Int, Int) -> V2 Double
v (a, b) = V2 (fromIntegral a) (fromIntegral b)

vs = map v . scanl add z
```

```haskell
l = fst . foldr add z

paths n =
  [ q
  | m <- [0..n]
  , q <- replicateM m [u,d,h]
  , l q == n
  ]

draw n q
  = frame 0.5
  . (`atop` centerXY (strutY (fromIntegral n)))
  . centerXY
  $ mconcat
  [ circle 0.25
  # fc white
  # translate pq
  # lw thin
  | pq <- vs q
  ] `atop` strokeT (trailFromOffsets (map v q))

grid = vcat . map hcat

diagram n m
  = bg white
  . centerXY
  . grid
  . transpose
  . chunksOf m
  . map (draw n)
  . sort
  $ paths n

main = defaultMain (diagram 5 10)
```

18 70/A000332.hs

```haskell
-- oeis-diagrams -- unofficial diagrams of OEIS sequences
-- Copyright (C) 2016-2017 Claude Heiland-Allen
-- License CC-BY-NC <https://creativecommons.org/licenses/by-nc/3.0/>

-- https://oeis.org/A000332
-- The number of equilateral triangles with vertices in an equilateral
-- triangular array of points with n rows (offset 1), with any orientation.
-- - Ignacio Larrosa Cañestro, Apr 09 2002.

{-# LANGUAGE FlexibleContexts #-}
import Diagrams.Prelude
import Diagrams.Backend.SVG.CmdLine (B, defaultMain)

import Data.List (sort, sortOn, nub, transpose)
import Data.List.Split (chunksOf)

third :: (Int, Int) -> (Int, Int) -> (Int, Int)
third (p, q) (p', q') =
  let (s, t) = (p' - p, q' - q)
  in  (p - t, q + s + t)
```

```
inTriangle :: Int -> (Int, Int) -> Bool
inTriangle n (p, q) = 0 <= p && 0 <= q && p + q < n

sizeSquared :: [(Int, Int)] -> Int
sizeSquared [(p, q), (p', q'), _] =
  let (s, t) = (p' - p, q' - q)
  in  s * s + s * t + t * t

triangles :: Int -> [[(Int, Int)]]
triangles n = sortOn sizeSquared $
  nub
  [ sort [(a, b), (c, d), (e, f)]
  | a <- [0..n]
  , b <- [0..n]
  , inTriangle n (a, b)
  , c <- [0..n]
  , d <- [0..n]
  , inTriangle n (c, d)
  , (a, b) /= (c, d)
  , (e, f) <- [ third (a, b) (c, d)
              , third (c, d) (a, b)
              ]
  , inTriangle n (e, f)
  ]

t2 :: (Int, Int) -> V2 Double
t2 (p, q)
  = V2
    (fromIntegral p + fromIntegral q / 2)
    (sqrt 3 * fromIntegral q / 2)

t2' = P . t2

draw n t@[ab,cd,ef]
  = frame 0.75
  . scale 1.25
  . rotate (15 @@ deg)
  $ mconcat
  [ circle 0.25
  # fc (if (p, q) `elem` t then grey else white)
  # translate (t2 (p, q))
  # lw thin
  | p <- [0..n]
  , q <- [0..n]
  , inTriangle n (p, q)
  ] `atop` mconcat
  [ t2' ab ~~ t2' cd
  , t2' cd ~~ t2' ef
  , t2' ef ~~ t2' ab
  ]

grid = vcat . map hcat

diagram n m
  = bg white
  . grid
  . chunksOf m
  . map (draw n)
  $ triangles n

main = defaultMain (diagram 6 7)
```

19 70/A000984.hs

```
-- oeis-diagrams -- unofficial diagrams of OEIS sequences
-- Copyright (C) 2016-2017 Claude Heiland-Allen
-- License CC-BY-NC <https://creativecommons.org/licenses/by-nc/3.0/>

-- https://oeis.org/A000984
-- The number of direct routes from my home to Granny's when Granny lives n
-- blocks south and n blocks east of my home in Grid City. To obtain a direct
-- route, from the 2n blocks, choose n blocks on which one travels south. For
-- example, a(2)=6 because there are 6 direct routes: SSEE, SESE, SEES, EESS,
-- ESES and ESSE. - Dennis P. Walsh, Oct 27 2006

{-# LANGUAGE FlexibleContexts #-}
import Diagrams.Prelude
import Diagrams.Backend.SVG.CmdLine (B, defaultMain)

import Control.Monad (replicateM)
import Data.List.Split (chunksOf)

u, d, z :: (Int, Int)
u = (1, 0)
d = (0, 1)
z = (0, 0)

add (a, b) (c, d) = (a + c, b + d)

v :: (Int, Int) -> V2 Double
v (a, b) = V2 (fromIntegral a) (fromIntegral b)

vs = map v . scanl add z
l = foldr add z

paths n =
  [ q
  | q <- replicateM (2 * n) [u,d]
  , l q == (n, n)
  ]

draw n q
  = frame 0.5
  . (`atop` centerXY (strutY (fromIntegral n)))
  . centerXY
  $ mconcat
  [ circle 0.25
  # fc white
  # translate pq
  # lw thin
  | pq <- vs q
  ] `atop` strokeT (trailFromOffsets (map v q))
```

```haskell
grid = vcat . map hcat

diagram n m
  = bg white
  . centerXY
  . grid
  . chunksOf m
  . map (draw n)
  $ paths n

main = defaultMain (diagram 4 7)
```

20 70/A001405.hs

```haskell
-- oeis-diagrams -- unofficial diagrams of OEIS sequences
-- Copyright (C) 2016-2017 Claude Heiland-Allen
-- License CC-BY-NC <https://creativecommons.org/licenses/by-nc/3.0/>

-- https://oeis.org/A001405
-- Number of meanders (walks starting at the origin and ending at any altitude
-- >= 0 that may touch but never go below the x-axis) with n steps from {-1,1}.
-- - David Nguyen, Dec 20 2016

{-# LANGUAGE FlexibleContexts #-}
import Diagrams.Prelude
import Diagrams.Backend.SVG.CmdLine (B, defaultMain)

import Control.Monad (replicateM)
import Data.List (sort, transpose)
import Data.List.Split (chunksOf)

u, d, z :: (Int, Int)
u = (1, 1)
d = (1, -1)
z = (0, 0)

add (a, b) (c, d) = (a + c, b + d)

boundedBelow = not . any ((< 0) . snd) . scanl add z

paths n =
  [ q
  | q <- replicateM n [u,d]
  , boundedBelow q
  ]

v :: (Int, Int) -> V2 Double
v (a, b) = V2 (fromIntegral a) (fromIntegral b)

vs = map v . scanl add z

draw n q
  = frame 0.5
  . (`atop` centerXY (strutY (fromIntegral n)))
  . centerXY
  $ mconcat
  [ circle 0.25
  # fc white
  # translate pq
  # lw thin
  | pq <- vs q
  ] `atop` strokeT (trailFromOffsets (map v q))

grid = vcat . map hcat

diagram n m
  = bg white
  . centerXY
  . grid
  . transpose
  . chunksOf m
  . map (draw n)
  . sort
  $ paths n

main = defaultMain (diagram 8 10)
```

21 70/A002623.hs

```
-- oeis-diagrams -- unofficial diagrams of OEIS sequences
-- Copyright (C) 2016-2017 Claude Heiland-Allen
-- License CC-BY-NC <https://creativecommons.org/licenses/by-nc/3.0/>

-- https://oeis.org/A002623
-- Number of nondegenerate triangles that can be made from rods of length
-- 1,2,3,4,...,n. - Alfred Bruckstein

{-# LANGUAGE FlexibleContexts #-}
import Diagrams.Prelude
import Diagrams.Backend.SVG.CmdLine (B, defaultMain)

import Data.List (sort, sortOn, nub, transpose)
import Data.List.Split (chunksOf)

nondegenerate :: [Int] -> Bool
nondegenerate [a,b,c] = a + b > c

corners :: [Int] -> [V2 Double]
corners [a',b',c']
  = [V2 0 0, V2 c 0, V2 x y]
  where
    a = fromIntegral a'
    b = fromIntegral b'
    c = fromIntegral c'
    x = (c^2 - a^2 + b^2) / (2 * c)
    y = sqrt $ b^2 - x^2

sizeSquared :: [Int] -> Double
sizeSquared [a',b',c']
  = s * (s - a) * (s - b) * (s - c)
  where
    a = fromIntegral a'
    b = fromIntegral b'
    c = fromIntegral c'
    s = (a + b + c) / 2

triangles :: Int -> [([Int], [V2 Double])]
triangles n
  = map (\t -> (t, corners t))
  . sortOn sizeSquared $
  [ abc
  | a <- [1..n]
  , b <- [a..n]
  , c <- [b..n]
  , let abc = [a,b,c]
  , nondegenerate abc
  ]

edge k a b
  = mconcat
  [ circle 0.25
  # fc white
  # translate p
  # lw thin
  | p <- [ lerp t a b
         | i <- [0..k]
         , let t = fromIntegral i / fromIntegral k
         ]
  ] `atop`
  (P a ~~ P b)

draw n ([a,b,c], t@[ab,cd,ef])
  = frame 0.5
  . (`atop` centerXY (strut (fromIntegral n)))
  . centerXY
  . rotate (15 @@ deg)
  $ mconcat
  [
  ] `atop` mconcat
  [ edge c ab cd
  , edge a cd ef
  , edge b ef ab
  ]

grid = vcat . map hcat

diagram n m
  = bg white
  . grid
  . chunksOf m
  . map (draw n)
  $ triangles n

main = defaultMain (diagram 8 7)
```

22 72/A002620.hs

```
-- number of multigraphs with loops on 2 nodes with 15 edges
-- https://oeis.org/A002620

graphs n =
```

```haskell
    [ (a, b, c)
    | a <- [0..n]
    , b <- [0..n]
    , c <- [0..n]
    , a + b + c == n
    , a <= c
    , a > 0 || b > 1 || c > 0
    ]

fx = 64
fy = 32

graph (u, v) (a, b, c)
  = "<g transform='translate(" ++ dist (fx * (b + 1 - (if c < 8 then 8 - c else 0) + (if odd (round c) then 0.5 else 0) + fromIntegral ((round c - 8) `div` 2))) ++ "," ++ dist (fy * (c + 1)) ++ ")'>\n"
  ++ concat [ topLoop (r + 1) | r <- [1 .. a] ]
  ++ concat [ betweenCurve x | x <- [ -b + 1, -b + 3 .. b - 1 ] ]
  ++ concat [ bottomLoop (r + 1) | r <- [1 .. c] ]
  ++ nodes
  ++ "</g>\n"

dist = coord
coord x = show (round (10 * x) :: Int)

r0 = 1
y0 = 16

nodes =  "<circle cx='0' cy='0' r='" ++ dist r0 ++ "' fill='red' />\n"
      ++ "<circle cx='0' cy='" ++ coord y0 ++ "' r='" ++ dist r0 ++ "' fill='red' />\n"
topLoop r = "<circle cx='0' cy='" ++ coord (r + y0) ++ "' r='" ++ dist r ++ "' />\n"
bottomLoop r = "<circle cx='0' cy='" ++ coord (-r) ++ "' r='" ++ dist r ++ "' />\n"
betweenCurve x = "<path d='M0,0 Q" ++ coord (2 * x) ++ "," ++ coord (y0/2) ++ " 0," ++ coord y0 ++ "' />\n"

main = putStrLn $
  "<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"yes\"?>\n" ++
  "<!DOCTYPE svg PUBLIC \"-//W3C//DTD SVG 1.1//EN\" \"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd\">\n" ++
  "<svg xmlns=\"http://www.w3.org/2000/svg\" viewBox=\"0 0 " ++ dist (fx * 16) ++ " " ++ dist (fy * 16) ++ "\">\n" ++
  "<g fill='none' stroke='black' stroke-width='1' transform='translate(5120,0) rotate(-45) translate(-5120,-2560)'>\n" ++
  concat (zipWith graph [(u, v) | v <- [1..6], u <- [1..12]] (graphs 15)) ++
  "</g></svg>"
```

23 92/A000124.hs

```haskell
-- oeis-diagrams -- unofficial diagrams of OEIS sequences
-- Copyright (C) 2016-2018 Claude Heiland-Allen
-- License CC-BY-NC <https://creativecommons.org/licenses/by-nc/3.0/>

-- https://oeis.org/A000124
-- a(n) is the maximal number of grandchildren of a binary vector of length
-- n+2. E.g., a binary vector of length 6 can produce at most 11 different
-- vectors when 2 bits are deleted.
--
-- a(13) = 92

{-# LANGUAGE FlexibleContexts #-}
import Diagrams.Prelude hiding (size)
import Diagrams.Backend.SVG.CmdLine (B, defaultMain)

import Control.Monad (replicateM)
import Data.List (sortBy, groupBy, transpose)
import Data.List.Split (chunksOf)
import Data.Ord (comparing)
import Data.Set (Set, fromList, toList, size)
import System.Random (StdGen, newStdGen, randomRs)

deletions :: [a] -> [[a]]
deletions [x] = [[]]
deletions (x:xs) = map (x:) (deletions xs) ++ [xs]

{-
n :: Int
n = 13

candidates :: [String]
candidates = replicateM (n + 2) "/\\"

equating :: Eq e => (a -> e) -> a -> a -> Bool
equating f a b = f a == f b

ancestor :: String
grandchildren :: [String]
(ancestor, grandchildren)
  = fmap toList
  . head
  . head
  . groupBy (equating (size . snd))
  . sortBy (flip $ comparing (size . snd))
  $ [ (c, s)
    | c <- candidates
    , let s = fromList
            . concatMap deletions
            . toList
            . fromList
            . deletions
            $ c
    ]
-}

ancestor :: String
grandchildren :: [String]
ancestor = "/\\/\\/\\/\\/\\/\\/\\/"
grandchildren
  = toList
  . fromList
  . concatMap deletions
  . toList
```

```haskell
  . fromList
  . deletions
  $ ancestor

up, down :: V2 Double
up = r2 (1, 1)
down = r2 (1, -1)

wiggle :: String -> Diagram B
wiggle s
  = atop (strutY 3)
  . centerXY
  . lineCap LineCapRound
  . lineJoin LineJoinRound
  . lwL 1
  . lc (colour . rampage $ s)
  . strokeP
  . fromOffsets
  . map wig
  $ s

wig :: Char -> V2 Double
wig '/'  = up
wig '\\' = down

rampage :: String -> (Int, Int, Int)
rampage s =
  ( minimum $ scanl (+) 0 $ map ramp s
  , maximum $ scanl (+) 0 $ map ramp s
  , sum (map ramp s)
  )

ramp :: Char -> Int
ramp '/' = 1
ramp '\\' = -1

colour :: (Int, Int, Int) -> Colour Double
colour ( 0, 1,  1) = black
colour (-1, 0, -1) = black
colour ( 0, 2,  1) = sRGB24 0xe7 0xd9 0x40 -- yellow
colour (-2, 0, -1) = sRGB24 0x44 0xde 0xd5 -- cyan
colour (-1, 1,  1) = sRGB24 0x40 0x45 0xce -- blue
colour (-1, 1, -1) = sRGB24 0xbf 0x6d 0xe5 -- magenta
colour (-2, 1, -1) = sRGB24 0x50 0xc3 0x36 -- green
colour ( 0, 3,  3) = sRGB24 0xcf 0x0c 0x4c -- red
colour s = error (show s)

shuffle :: StdGen -> [a] -> [a]
shuffle g
  = map snd
  . sortBy (comparing fst)
  . zip (randomRs (0, 1 :: Double) g)

diagram :: StdGen -> Diagram B
diagram g
  = bg white
  . frame 4
```

```
    . centerXY
    . vsep (1 :: Double)
    . map
        ( centerXY
        . hsep (2 :: Double)
        . map wiggle
        )
    . chunksOf 4
    . shuffle g
    $ grandchildren

main :: IO ()
main = newStdGen >>= defaultMain . diagram
```

24 CC-BY-NC.md

```
Creative Commons Legal Code
===========================

Attribution-NonCommercial 3.0 Unported
--------------------------------------

> CREATIVE COMMONS CORPORATION IS NOT A LAW FIRM AND DOES NOT PROVIDE
> LEGAL SERVICES. DISTRIBUTION OF THIS LICENSE DOES NOT CREATE AN
> ATTORNEY-CLIENT RELATIONSHIP. CREATIVE COMMONS PROVIDES THIS
> INFORMATION ON AN "AS-IS" BASIS. CREATIVE COMMONS MAKES NO WARRANTIES
> REGARDING THE INFORMATION PROVIDED, AND DISCLAIMS LIABILITY FOR
> DAMAGES RESULTING FROM ITS USE.

### *License*

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE
COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY
COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS
AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE
TO BE BOUND BY THE TERMS OF THIS LICENSE. TO THE EXTENT THIS LICENSE MAY
BE CONSIDERED TO BE A CONTRACT, THE LICENSOR GRANTS YOU THE RIGHTS
CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND
CONDITIONS.

**1. Definitions**

a.  **"Adaptation"** means a work based upon the Work, or upon the Work
    and other pre-existing works, such as a translation, adaptation,
    derivative work, arrangement of music or other alterations of a
    literary or artistic work, or phonogram or performance and includes
    cinematographic adaptations or any other form in which the Work may
    be recast, transformed, or adapted including in any form
    recognizably derived from the original, except that a work that
    constitutes a Collection will not be considered an Adaptation for
    the purpose of this License. For the avoidance of doubt, where the
    Work is a musical work, performance or phonogram, the
    synchronization of the Work in timed-relation with a moving
    image ("synching") will be considered an Adaptation for the purpose
    of this License.
```

```
b.  **"Collection"** means a collection of literary or artistic works,
    such as encyclopedias and anthologies, or performances, phonograms
    or broadcasts, or other works or subject matter other than works
    listed in Section 1(f) below, which, by reason of the selection and
    arrangement of their contents, constitute intellectual creations, in
    which the Work is included in its entirety in unmodified form along
    with one or more other contributions, each constituting separate and
    independent works in themselves, which together are assembled into a
    collective whole. A work that constitutes a Collection will not be
    considered an Adaptation (as defined above) for the purposes of
    this License.
c.  **"Distribute"** means to make available to the public the original
    and copies of the Work or Adaptation, as appropriate, through sale
    or other transfer of ownership.
d.  **"Licensor"** means the individual, individuals, entity or entities
    that offer(s) the Work under the terms of this License.
e.  **"Original Author"** means, in the case of a literary or artistic
    work, the individual, individuals, entity or entities who created
    the Work or if no individual or entity can be identified, the
    publisher; and in addition (i) in the case of a performance the
    actors, singers, musicians, dancers, and other persons who act,
    sing, deliver, declaim, play in, interpret or otherwise perform
    literary or artistic works or expressions of folklore; (ii) in the
    case of a phonogram the producer being the person or legal entity
    who first fixes the sounds of a performance or other sounds;
    and, (iii) in the case of broadcasts, the organization that
    transmits the broadcast.
f.  **"Work"** means the literary and/or artistic work offered under the
    terms of this License including without limitation any production in
    the literary, scientific and artistic domain, whatever may be the
    mode or form of its expression including digital form, such as a
    book, pamphlet and other writing; a lecture, address, sermon or
    other work of the same nature; a dramatic or dramatico-musical work;
    a choreographic work or entertainment in dumb show; a musical
    composition with or without words; a cinematographic work to which
    are assimilated works expressed by a process analogous to
    cinematography; a work of drawing, painting, architecture,
    sculpture, engraving or lithography; a photographic work to which
    are assimilated works expressed by a process analogous to
    photography; a work of applied art; an illustration, map, plan,
    sketch or three-dimensional work relative to geography, topography,
    architecture or science; a performance; a broadcast; a phonogram; a
    compilation of data to the extent it is protected as a copyrightable
    work; or a work performed by a variety or circus performer to the
    extent it is not otherwise considered a literary or artistic work.
g.  **"You"** means an individual or entity exercising rights under this
    License who has not previously violated the terms of this License
    with respect to the Work, or who has received express permission
    from the Licensor to exercise rights under this License despite a
    previous violation.
h.  **"Publicly Perform"** means to perform public recitations of the
    Work and to communicate to the public those public recitations, by
    any means or process, including by wire or wireless means or public
    digital performances; to make available to the public Works in such
    a way that members of the public may access these Works from a place
    and at a place individually chosen by them; to perform the Work to
    the public by any means or process and the communication to the
    public of the performances of the Work, including by public digital
    performance; to broadcast and rebroadcast the Work by any means
    including signs, sounds or images.
i.  **”Reproduce”** means to make copies of the Work by any means
    including without limitation by sound or visual recordings and the
    right of fixation and reproducing fixations of the Work, including
    storage of a protected performance or phonogram in digital form or
    other electronic medium.

**2. Fair Dealing Rights.** Nothing in this License is intended to
reduce, limit, or restrict any uses free from copyright or rights
arising from limitations or exceptions that are provided for in
connection with the copyright protection under copyright law or other
applicable laws.

**3. License Grant.** Subject to the terms and conditions of this
License, Licensor hereby grants You a worldwide, royalty-free,
non-exclusive, perpetual (for the duration of the applicable copyright)
license to exercise the rights in the Work as stated below:

a.  to Reproduce the Work, to incorporate the Work into one or more
    Collections, and to Reproduce the Work as incorporated in the
    Collections;
b.  to create and Reproduce Adaptations provided that any such
    Adaptation, including any translation in any medium, takes
    reasonable steps to clearly label, demarcate or otherwise identify
    that changes were made to the original Work. For example, a
    translation could be marked ”The original work was translated from
    English to Spanish,” or a modification could indicate ”The original
    work has been modified.”;
c.  to Distribute and Publicly Perform the Work including as
    incorporated in Collections; and,
d.  to Distribute and Publicly Perform Adaptations.

The above rights may be exercised in all media and formats whether now
known or hereafter devised. The above rights include the right to make
such modifications as are technically necessary to exercise the rights
in other media and formats. Subject to Section 8(f), all rights not
expressly granted by Licensor are hereby reserved, including but not
limited to the rights set forth in Section 4(d).

**4. Restrictions.** The license granted in Section 3 above is expressly
made subject to and limited by the following restrictions:

a.  You may Distribute or Publicly Perform the Work only under the terms
    of this License. You must include a copy of, or the Uniform Resource
    Identifier (URI) for, this License with every copy of the Work You
    Distribute or Publicly Perform. You may not offer or impose any
    terms on the Work that restrict the terms of this License or the
    ability of the recipient of the Work to exercise the rights granted
    to that recipient under the terms of the License. You may not
    sublicense the Work. You must keep intact all notices that refer to
    this License and to the disclaimer of warranties with every copy of
    the Work You Distribute or Publicly Perform. When You Distribute or
    Publicly Perform the Work, You may not impose any effective
    technological measures on the Work that restrict the ability of a
    recipient of the Work from You to exercise the rights granted to
    that recipient under the terms of the License. This Section 4(a)
    applies to the Work as incorporated in a Collection, but this does
    not require the Collection apart from the Work itself to be made
    subject to the terms of this License. If You create a Collection,
    upon notice from any Licensor You must, to the extent practicable,
    remove from the Collection any credit as required by Section 4(c),
    as requested. If You create an Adaptation, upon notice from any
    Licensor You must, to the extent practicable, remove from the
    Adaptation any credit as required by Section 4(c), as requested.
b.  You may not exercise any of the rights granted to You in Section 3
    above in any manner that is primarily intended for or directed
    toward commercial advantage or private monetary compensation. The
    exchange of the Work for other copyrighted works by means of digital
    file-sharing or otherwise shall not be considered to be intended for
    or directed toward commercial advantage or private monetary
    compensation, provided there is no payment of any monetary
    compensation in connection with the exchange of copyrighted works.
c.  If You Distribute, or Publicly Perform the Work or any Adaptations
    or Collections, You must, unless a request has been made pursuant to
    Section 4(a), keep intact all copyright notices for the Work and
    provide, reasonable to the medium or means You are utilizing: (i)
    the name of the Original Author (or pseudonym, if applicable) if
    supplied, and/or if the Original Author and/or Licensor designate
    another party or parties (e.g., a sponsor institute, publishing
    entity, journal) for attribution ("Attribution Parties") in
    Licensor's copyright notice, terms of service or by other reasonable
    means, the name of such party or parties; (ii) the title of the Work
    if supplied; (iii) to the extent reasonably practicable, the URI, if
    any, that Licensor specifies to be associated with the Work, unless
    such URI does not refer to the copyright notice or licensing
    information for the Work; and, (iv) consistent with Section 3(b), in
    the case of an Adaptation, a credit identifying the use of the Work
    in the Adaptation (e.g., "French translation of the Work by Original
    Author," or "Screenplay based on original Work by Original Author").
    The credit required by this Section 4(c) may be implemented in any
    reasonable manner; provided, however, that in the case of a
    Adaptation or Collection, at a minimum such credit will appear, if a
    credit for all contributing authors of the Adaptation or Collection
    appears, then as part of these credits and in a manner at least as
    prominent as the credits for the other contributing authors. For the
    avoidance of doubt, You may only use the credit required by this
    Section for the purpose of attribution in the manner set out above
    and, by exercising Your rights under this License, You may not
    implicitly or explicitly assert or imply any connection with,
    sponsorship or endorsement by the Original Author, Licensor and/or
    Attribution Parties, as appropriate, of You or Your use of the Work,
    without the separate, express prior written permission of the
    Original Author, Licensor and/or Attribution Parties.
d.  For the avoidance of doubt:

    i.  **Non-waivable Compulsory License Schemes**. In those
        jurisdictions in which the right to collect royalties through
        any statutory or compulsory licensing scheme cannot be waived,
        the Licensor reserves the exclusive right to collect such
        royalties for any exercise by You of the rights granted under
        this License;
    ii. **Waivable Compulsory License Schemes**. In those jurisdictions
```

```
        in which the right to collect royalties through any statutory or
        compulsory licensing scheme can be waived, the Licensor reserves
        the exclusive right to collect such royalties for any exercise
        by You of the rights granted under this License if Your exercise
        of such rights is for a purpose or use which is otherwise than
        noncommercial as permitted under Section 4(b) and otherwise
        waives the right to collect royalties through any statutory or
        compulsory licensing scheme; and,
    iii. **Voluntary License Schemes**. The Licensor reserves the right
        to collect royalties, whether individually or, in the event that
        the Licensor is a member of a collecting society that
        administers voluntary licensing schemes, via that society, from
        any exercise by You of the rights granted under this License
        that is for a purpose or use which is otherwise than
        noncommercial as permitted under Section 4(c).

e.  Except as otherwise agreed in writing by the Licensor or as may be
    otherwise permitted by applicable law, if You Reproduce, Distribute
    or Publicly Perform the Work either by itself or as part of any
    Adaptations or Collections, You must not distort, mutilate, modify
    or take other derogatory action in relation to the Work which would
    be prejudicial to the Original Author's honor or reputation.
    Licensor agrees that in those jurisdictions (e.g. Japan), in which
    any exercise of the right granted in Section 3(b) of this License
    (the right to make Adaptations) would be deemed to be a distortion,
    mutilation, modification or other derogatory action prejudicial to
    the Original Author's honor and reputation, the Licensor will waive
    or not assert, as appropriate, this Section, to the fullest extent
    permitted by the applicable national law, to enable You to
    reasonably exercise Your right under Section 3(b) of this License
    (right to make Adaptations) but not otherwise.

**5. Representations, Warranties and Disclaimer**

UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING, LICENSOR
OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY
KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE,
INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTIBILITY,
FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF
LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS,
WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE
EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

**6. Limitation on Liability.** EXCEPT TO THE EXTENT REQUIRED BY
APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL
THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY
DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF
LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

**7. Termination**

a.  This License and the rights granted hereunder will terminate
    automatically upon any breach by You of the terms of this License.
    Individuals or entities who have received Adaptations or Collections
    from You under this License, however, will not have their licenses
    terminated provided such individuals or entities remain in full
    compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will
    survive any termination of this License.
b.  Subject to the above terms and conditions, the license granted here
    is perpetual (for the duration of the applicable copyright in
    the Work). Notwithstanding the above, Licensor reserves the right to
    release the Work under different license terms or to stop
    distributing the Work at any time; provided, however that any such
    election will not serve to withdraw this License (or any other
    license that has been, or is required to be, granted under the terms
    of this License), and this License will continue in full force and
    effect unless terminated as stated above.

**8. Miscellaneous**

a.  Each time You Distribute or Publicly Perform the Work or a
    Collection, the Licensor offers to the recipient a license to the
    Work on the same terms and conditions as the license granted to You
    under this License.
b.  Each time You Distribute or Publicly Perform an Adaptation, Licensor
    offers to the recipient a license to the original Work on the same
    terms and conditions as the license granted to You under
    this License.
c.  If any provision of this License is invalid or unenforceable under
    applicable law, it shall not affect the validity or enforceability
    of the remainder of the terms of this License, and without further
    action by the parties to this agreement, such provision shall be
    reformed to the minimum extent necessary to make such provision
    valid and enforceable.
d.  No term or provision of this License shall be deemed waived and no
    breach consented to unless such waiver or consent shall be in
    writing and signed by the party to be charged with such waiver
    or consent.
e.  This License constitutes the entire agreement between the parties
    with respect to the Work licensed here. There are no understandings,
    agreements or representations with respect to the Work not
    specified here. Licensor shall not be bound by any additional
    provisions that may appear in any communication from You. This
    License may not be modified without the mutual written agreement of
    the Licensor and You.
f.  The rights granted under, and the subject matter referenced, in this
    License were drafted utilizing the terminology of the Berne
    Convention for the Protection of Literary and Artistic Works (as
    amended on September 28, 1979), the Rome Convention of 1961, the
    WIPO Copyright Treaty of 1996, the WIPO Performances and Phonograms
    Treaty of 1996 and the Universal Copyright Convention (as revised on
    July 24, 1971). These rights and subject matter take effect in the
    relevant jurisdiction in which the License terms are sought to be
    enforced according to the corresponding provisions of the
    implementation of those treaty provisions in the applicable national
    law. If the standard suite of rights granted under applicable
    copyright law includes additional rights not granted under this
    License, such additional rights are deemed to be included in the
    License; this License is not intended to restrict the license of any
    rights under applicable law.

> ### Creative Commons Notice
>
> Creative Commons is not a party to this License, and makes no warranty
> whatsoever in connection with the Work. Creative Commons will not be
> liable to You or any party on any legal theory for any damages
> whatsoever, including without limitation any general, special,
> incidental or consequential damages arising in connection to this
> license. Notwithstanding the foregoing two (2) sentences, if Creative
> Commons has expressly identified itself as the Licensor hereunder, it
> shall have all rights and obligations of Licensor.
>
> Except for the limited purpose of indicating to the public that the
> Work is licensed under the CCPL, Creative Commons does not authorize
> the use by either party of the trademark "Creative Commons" or any
> related trademark or logo of Creative Commons without the prior
> written consent of Creative Commons. Any permitted use will be in
> compliance with Creative Commons' then-current trademark usage
> guidelines, as may be published on its website or otherwise made
> available upon request from time to time. For the avoidance of doubt,
> this trademark restriction does not form part of the License.
>
> Creative Commons may be contacted at <https://creativecommons.org/>.
```

25 .gitignore

```
.cabal-sandbox
cabal.sandbox.config
*.aux
*.hi
*.log
*.o
*.pdf
*.png
*.svg
*.tex
```

26 README.md

```
OEIS Diagrams
=============

Some diagrams of sequences from:

> The Online Encyclopedia of Integer Sequences <https://oeis.org>

a registered trademark of The OEIS Foundation, Inc.

The OEIS is licensed under CC-BY-NC:
<https://creativecommons.org/licenses/by-nc/3.0/>

This diagramming project is neither endorsed by or affiliated with OEIS.


Running the code
----------------

In the absence of a cabal file for the project, you can do this:

    cabal sandbox init
    cabal install diagrams diagrams-cairo
```

```
# --allow-newer flag only needed on ghc-8.2.1 as of 2017-11-11
cabal install diagrams-pgf --allow-newer
# most of them use the SVG backend, only 2016 is PGF, so try this
cabal exec -- runghc dir/file.hs -w 1000 -o output.svg
# or even
for h in */*.hs
do
  cabal exec -- runghc "${h}" -w 1000 -o "${h}.svg"
done
```