IntPic

A GAP package for drawing sets of integers

Version 0.2.1

5 July 2015

Manuel Delgado

Manuel Delgado Email: mdelgado@fc.up.pt
Homepage: http://cmup.fc.up.pt/cmup/mdelgado/

Copyright

Acknowledgements

The author was partially funded by the European Regional Development Fund through the program COMPETE and by the Portuguese Government through the FCT - Fundação para a Ciência e a Tecnologia under the projects PEst-C/MAT/UI0144/2011 and PEst-C/MAT/UI0144/2013.

I want to express my gratitude to my colleagues of the Mathematics Department of the Faculty of Sciences of the University of Porto for the opportunity of taking a sabbatical year during the 2011/2012 school year. I benefited also of the sabbatical grant SFRH/BSAB/1156/2011.

For one reason or another that ranges from suggestions to encouragement, I want express my gratitude to Pedro A. García Sánchez, David Llena and James Mitchell.

CONCERNING THE MANTAINMENT:

I want to acknowledge partial support by CMUP (UID/MAT/00144/2013), which is funded by FCT (Portugal) with national (MEC) and European structural funds through the programs FEDER, under the partnership agreement PT2020.

Colophon

This manual describes the GAP package IntPic version 0.2.1 for visualizing and creating publication quality pictures of sets of integers.

Contents

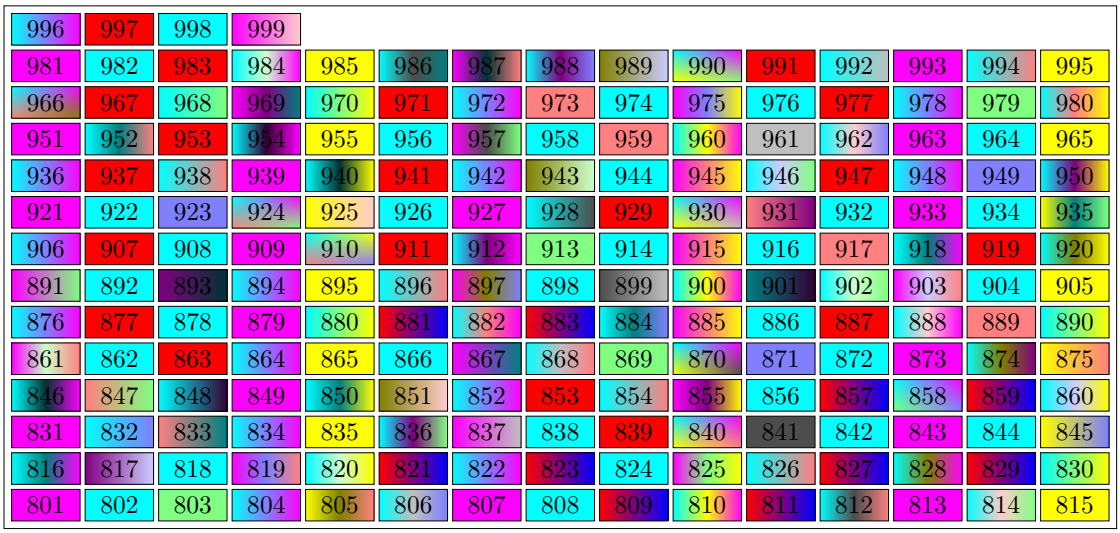
Chapter 1

The IntPic package

1.1 Overview and Introduction

The IntPic package has as its main goal producing `Tikz` code for arrays of integers to be included in a LaTeX file, which can then be processed. Some of the integers are emphasized, by using different colors for the cells.

IntPic grew up from my will to have a pictorial view of some sets of integers. I wanted, in particular, get a pictorial view of the results produced by the NumericalSgps package [DGSM15]. Effort has then been made to serve a slightly more general purpose. For instance, if the user wants to have a pictorial idea of how many primes there are between 800 and 1000, or show it to his students and, perhaps, which among these primes are twin primes, he will probably be happy by producing a picture like the following



It has clearly too much information, given through the different colors. The twin primes in the given range are in red-blue, while the remaining primes in the same range are in red.

This package contains relatively few lines of code. The heavier part is the documentation, where many examples are presented.

The design of this greatly benefits from my long experience on producing visualization tools for GAP objects. The package sgpviz [DM08] is the visible part. More recently, I got involved in a more

general project, the Viz package [DENMP12]. The experience gained there, especially through long and fruitful discussions with J. Mitchell, influenced me a lot. This package will probably be part of that more general project. For the moment it is independent, but its use in conjunction with the Viz package is recommended since in this case an immediate visualization is provided.

The package produces `tikz` code that the user may then use at his wish. In particular, he can use it in publications. But prior to obtaining results that lead to a publication, the user may benefit of viewing thousands of images. There is a (almost platform independent) function in Viz that is intended to make this task easy. It benefits from the GAP stuff on creating a temporary directory where the computations occur. The cleaning task is also left to GAP, which leaves the user free of the need of collecting the garbage. In order to produce the drawings, LaTeX, as well as some LaTeX packages, in particular `tikz` and `pgf`, must be installed and working. I will assume that this is the case. All the images in [DFGSL14] have been produced by using the IntPic package.

This package consists basically of a function with many options associated. The purpose of the manual is to illustrate the use of the options. Many examples are presented. A file, named `examples.g` contains the GAP code, including the one to save the `tikz` code, to produce the examples in the manual.

1.2 Installing IntPic

In this section we give a brief description of how to start using IntPic. If you have any problems getting IntPic working, then you could try emailing me at `mdelgado@fc.up.pt`.

It is assumed that you have a working copy of GAP with version number 4.5 or higher. The most up-to-date version of GAP and instructions on how to install it can be obtained from the main GAP web page `http://www.gap-system.org`.

If the IntPic package was obtained as a part of the GAP distribution from the "Download" section of the GAP website, you may proceed to Section 1.3. Alternatively, the IntPic package may be installed using a separate archive, for example, for an update or an installation in a non-default location (see (**Reference: GAP Root Directories**)).

Below we describe the installation procedure for the `.tar.gz` archive format, which can be obtained from `http://cmup.fc.up.pt/cmup/mdelgado/intpic/`. Installation using other archive formats or non UNIX-like systems is performed in a similar way.

To install the IntPic package, unpack the archive file, which should have a name of the form `intpic-`*`XXX`*`.tar.gz` for some version number *`XXX`*, by typing

```
gzip -dc intpic-XXX.tar.gz | tar xpv
```

It may be unpacked in one of the following locations:

- in the `pkg` directory of your GAP installation;
- or in a directory named `.gap/pkg` in your home directory (to be added to the GAP root directory unless GAP is started with `-r` option);
- or in a directory named `pkg` in another directory of your choice (e.g. in the directory `mygap` in your home directory).

In the latter case one must start GAP with the `-l` option, e.g. if your private `pkg` directory is a subdirectory of `mygap` in your home directory you might type:

```
gap -l ";myhomedir/mygap"
```

where *myhomedir* is the path to your home directory, which may be replaced by a tilde (the empty path before the semicolon is filled in by the default path of the GAP home directory).

1.3 Loading IntPic

To use the IntPic Package you have to request it explicitly. This is done by calling `LoadPackage` (**Reference: LoadPackage**):

Example

```
gap> LoadPackage("intpic");
```

The package banner, followed by `true`, will be shown, if the load has been successful.

If you want to load the IntPic package by default, you can put the `LoadPackage` command into your `gaprc` file (see Section (**Reference: The gap.ini and gaprc files**)).

Chapter 2

The IntPic package main function

This chapter consists of two sections, the first of which decribes the main function of the package. The second one can be thought just as an example to produce a table where the integers appear ordered in a non standard way.

2.1 The main function

The function `IP_TikzArrayOfIntegers` (2.1.1) is the main function of the IntPic package. It aims to produce `tikz` code for displaying arrays of integers.

2.1.1 Tikz code for arrays of integers

▷ `IP_TikzArrayOfIntegers(`*arg*`)` (function)

The arguments (at most 3) are:

1. (optional)
   - a table of integers. In this case, the length of the rows is the maximum of the lengths of the sublists in the table, *or*
   - a list of integers and, optionally, an integer which indicates the length of the rows; when the length of the rows is not indicated, a compromise between the width and the height is tried.
2. a record of options. One of the fields of this record, named `highlights`, is an array whose entries are the numbers to be highlighted: one color per sublist. See details and other options in Chapter 5.

When no list nor table is present, the smallest range containing all the integers to be highlighted is taken.

Example

```
gap> rg := [81..89];;
gap> len := 10;;
gap> arr := [Filtered(rg,IsPrime),Filtered(rg,u->(u mod 2)=0),
>         Filtered(rg,u->(u mod 3)=0)];;
gap> tkz := IP_TikzArrayOfIntegers(rg,len,rec(highlights:=arr));;
```

The aspect of the string *tkz* produced is not very appealing. We show it once, by asking it exlicitly in the next example. In the forthcomming examples we keep using two semicolons to avoid showing this kind of strings.

The tkz string

```
gap> tkz;
"%tikz\n\\begin{tikzpicture}[every node/.style={draw,scale=1pt,\nminimum width\
=20pt,inner sep=3pt,\nline width=1pt,draw=black}]\n\\matrix[row sep=2pt,column\
 sep=2pt]\n{\\node[fill=-red]{86};&\n\\node[fill=green]{87};&\n\\node[fill=-re\
d]{88};&\n\\node[fill=red]{89};\\\\\n\\node[fill=green]{81};&\n\\node[fill=-re\
d]{82};&\n\\node[fill=red]{83};&\n\\node[left color=-red,right color=green]{84\
};&\n\\node[]{85};\\\\\n};\n\\end{tikzpicture}\n"
```

This string can be used at the users wish. In particular, it can be sent to the standard output using the command `Print` (**Reference: Print**).

The tikz code

```
gap> Print(tkz);
%tikz
\begin{tikzpicture}[every node/.style={draw,scale=1pt,
minimum width=20pt,inner sep=3pt,
line width=1pt,draw=black}]
\matrix[row sep=2pt,column sep=2pt]
{\node[fill=-red]{86};&
\node[fill=green]{87};&
\node[fill=-red]{88};&
\node[fill=red]{89};\\
\node[fill=green]{81};&
\node[fill=-red]{82};&
\node[fill=red]{83};&
\node[left color=-red,right color=green]{84};&
\node[]{85};\\
};
\end{tikzpicture}
```

It can now be copied and pasted in a LaTeX document (having the appropriate packages in the preamble). See Chapter 4 for details and alternatives.

The next function uses the previous one, but is called with a simpler argument. It will hopefully be useful for simple drawings. The length of each row and the umber of columns varies. A compromise based on some experiments has been established in order to obtain not too large nor too high images.

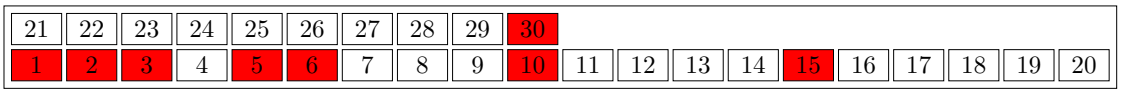
2.1.2 Tikz code for arrays, in a simplified way

▷ `IP_SimpleTikzArrayOfIntegers(`*arg*`)` (function)

The argument is either a list of integers or a matrix of integers. The integers involved are embeded in a range *rg* of minimum length and highlighted by using the list of default colors.

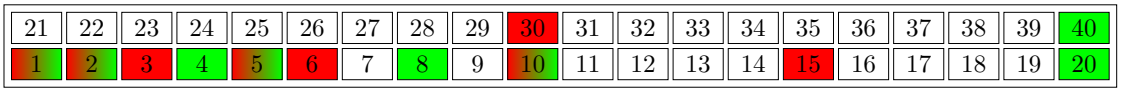
Example

```
gap> d := DivisorsInt(30);
[ 1, 2, 3, 5, 6, 10, 15, 30 ]
gap> IP_SimpleTizkArrayOfIntegers(d);;
```



Example

```
gap> d30 := DivisorsInt(30);
[ 1, 2, 3, 5, 6, 10, 15, 30 ]
gap> d40 := DivisorsInt(40);
[ 1, 2, 4, 5, 8, 10, 20, 40 ]
gap> tkz := IP_SimpleTizkArrayOfIntegers([d30,d40]);;
```



2.2 Producing tables

When the user is interested in tables of a certain kind, it may be a good idea to write some code to produce these tables. The following function (whose code is part of the file *ip_tables.gi* in the *gap* folder of this package) is convenient to deal with numerical semigroups with two generators and has been used to produce the images contained in [DFGSL14].

2.2.1 IP_TableWithModularOrder

▷ `IP_TableWithModularOrder(o, a, b, depth, height, rep, pos)` (function)

The arguments `rep` and `pos` are booleans (`true` or `false`). When `rep` is `true` there is some repetition: the last column is equal to the first, but pushed down some rows. When `pos` is `true`, no rows below 0 are considered, (contradicting `depth`, if needed).

The first five arguments arguments `o, a, b,depth` and `height` are integers. What they represent is described in what follows. There is assigned some kind of a referential on the constructed table and the fist argument, `o`, stands for the origin. A table with `b` columns ($b+1$ columns when `rep` is `true`) is constructed as follows. The row containing the origin is

- $o + [0..b-1] * a$, if `rep` is `false`, *or*
- $o + [0..b] * a$, if `rep` is `true`

The remaining rows are obtained by adding `b` (the upper ones) or subtracting `b` (the others) to these rows.

Note: when $a < b$ are co-prime, this construction provides a representation of the integers as an array.

Example

```
gap> a := 8;; b := 19;;
gap> ns := NumericalSemigroup(a,b);;
gap> c := ConductorOfNumericalSemigroup(ns);;
gap> origin := 2*c-1;
251
gap> ground := [origin..origin+b-1];;
gap>
gap> height:=2;;
```

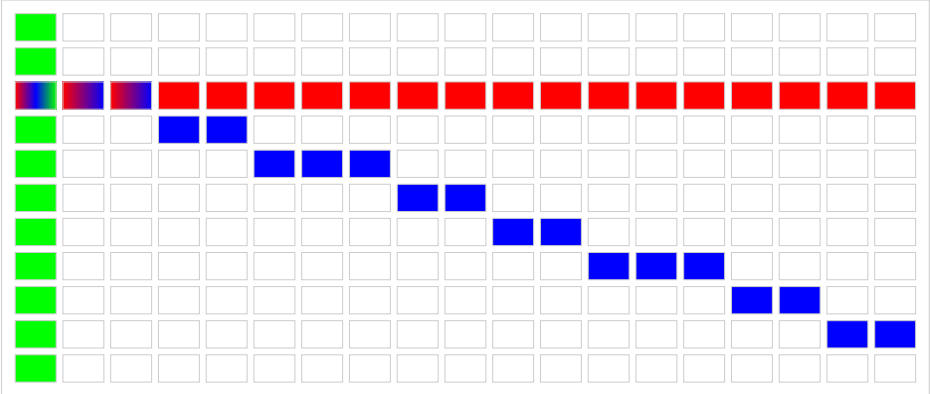
```
gap> depth:=8;;
gap>   xaxis := [origin];;
gap>   for n in [1..b-1] do
>     Add(xaxis, origin+n*a);
>   od;
gap>   yaxis := [];;
gap>   for n in [-depth..height] do
>     Add(yaxis, origin+n*b);
>   od;
gap>
gap> table := TableWithModularOrder(origin,a,b,depth,height,false,false);;
gap> arr := [xaxis,yaxis,ground];
[ [ 251, 259, 267, 275, 283, 291, 299, 307, 315, 323, 331, 339, 347, 355,
      363, 371, 379, 387, 395 ],
  [ 99, 118, 137, 156, 175, 194, 213, 232, 251, 270, 289 ], [ 251 .. 269 ] ]
gap> tkz:=IP_TikzArrayOfIntegers(table,rec(highlights:=arr));;
```

289	297	305	313	321	329	337	345	353	361	369	377	385	393	401	409	417	425	433
270	278	286	294	302	310	318	326	334	342	350	358	366	374	382	390	398	406	414
251	259	267	275	283	291	299	307	315	323	331	339	347	355	363	371	379	387	395
232	240	248	256	264	272	280	288	296	304	312	320	328	336	344	352	360	368	376
213	221	229	237	245	253	261	269	277	285	293	301	309	317	325	333	341	349	357
194	202	210	218	226	234	242	250	258	266	274	282	290	298	306	314	322	330	338
175	183	191	199	207	215	223	231	239	247	255	263	271	279	287	295	303	311	319
156	164	172	180	188	196	204	212	220	228	236	244	252	260	268	276	284	292	300
137	145	153	161	169	177	185	193	201	209	217	225	233	241	249	257	265	273	281
118	126	134	142	150	158	166	174	182	190	198	206	214	222	230	238	246	254	262
99	107	115	123	131	139	147	155	163	171	179	187	195	203	211	219	227	235	243

The next picture is obtained in the same way. The information that only the shape has interest is given by including the option `shape_only:=" "`. The variable `tkz` should be defined in a similar manner to the following one.

Example

```
gap> tkz:=IP_TikzArrayOfIntegers(table,rec(highlights:=arr,shape_only:=" ",
>              cell_width := "6",colsep:="1",rowsep:="1",inner_sep:="2",
>              line_color:="black!20"));;
```



Next, a minimum of changes, just to illustrate the effect of *rep* and *pos*.

Example

```
gap> table := TableWithModularOrder(origin,a,b,depth,50,true,true);;
gap> tkz:=IP_TikzArrayOfIntegers(table,rec(highlights:=arr));;
```

289	297	305	313	321	329	337	345	353	361	369	377	385	393	401	409	417	425	433	441
270	278	286	294	302	310	318	326	334	342	350	358	366	374	382	390	398	406	414	422
251	259	267	275	283	291	299	307	315	323	331	339	347	355	363	371	379	387	395	403
232	240	248	256	264	272	280	288	296	304	312	320	328	336	344	352	360	368	376	384
213	221	229	237	245	253	261	269	277	285	293	301	309	317	325	333	341	349	357	365
194	202	210	218	226	234	242	250	258	266	274	282	290	298	306	314	322	330	338	346
175	183	191	199	207	215	223	231	239	247	255	263	271	279	287	295	303	311	319	327
156	164	172	180	188	196	204	212	220	228	236	244	252	260	268	276	284	292	300	308
137	145	153	161	169	177	185	193	201	209	217	225	233	241	249	257	265	273	281	289
118	126	134	142	150	158	166	174	182	190	198	206	214	222	230	238	246	254	262	270
99	107	115	123	131	139	147	155	163	171	179	187	195	203	211	219	227	235	243	251
80	88	96	104	112	120	128	136	144	152	160	168	176	184	192	200	208	216	224	232
61	69	77	85	93	101	109	117	125	133	141	149	157	165	173	181	189	197	205	213
42	50	58	66	74	82	90	98	106	114	122	130	138	146	154	162	170	178	186	194
23	31	39	47	55	63	71	79	87	95	103	111	119	127	135	143	151	159	167	175
4	12	20	28	36	44	52	60	68	76	84	92	100	108	116	124	132	140	148	156
-15	-7	1	9	17	25	33	41	49	57	65	73	81	89	97	105	113	121	129	137
-34	-26	-18	-10	-2	6	14	22	30	38	46	54	62	70	78	86	94	102	110	118
-53	-45	-37	-29	-21	-13	-5	3	11	19	27	35	43	51	59	67	75	83	91	99
-72	-64	-56	-48	-40	-32	-24	-16	-8	0	8	16	24	32	40	48	56	64	72	80

Chapter 3

The colors in the IntPic package

The idea in what concerns the colors is the following: the reader is free to choose his colors (taking into account that the latex xcolor package is used), but we try to make users life reasonably easy. He is allowed to choose tones. The default colors used by IntPic are not many, although (from our experience) sufficient for most examples.

3.1 Colors by tones

The colors are divided by tones.

red

```
gap> IP_ColorsRedTones; #red
[ "red", "red!50", "red!20", "red!80!green!50", "red!80!blue!60" ]
```

green

```
gap> IP_ColorsGreenTones; #green
[ "green", "green!50", "green!20", "green!80!red!50", "green!80!blue!60" ]
```

blue

```
gap> IP_ColorsBlueTones; #blue
[ "blue", "blue!50", "blue!20", "blue!80!red!50", "blue!80!green!60" ]
```

cyan

```
gap> IP_ColorsCompRedTones; # cyan (complement of red)
[ "-red", "-red!50", "-red!20", "-red!80!green!50", "-red!80!blue!60" ]
```

magenta

```
gap> IP_ColorsCompGreenTones; # magenta (complement of green)
[ "-green", "-green!50", "-green!20", "-green!80!red!50", "-green!80!blue!60"
 ]
```

yellow

```
gap> IP_ColorsCompBlueTones; # yellow (complement of blue)
[ "-blue", "-blue!50", "-blue!20", "-blue!80!red!50", "-blue!80!green!60" ]
```

dark gray

```
gap> IP_ColorsDGrayTones; # dark gray
[ "black!80", "black!70", "black!60", "black!50", "black!40" ]
```

light gray

```
gap> IP_ColorsLGrayTones; # light gray
[ "black!30", "black!25", "black!20", "black!15", "black!10" ]
```

3.2 Lists of colors

array of colors by tones

```
gap> ListsOfIP_Colors;
[ [ "red", "red!50", "red!20", "red!80!green!50", "red!80!blue!60" ],
  [ "green", "green!50", "green!20", "green!80!red!50", "green!80!blue!60" ],
  [ "blue", "blue!50", "blue!20", "blue!80!red!50", "blue!80!green!60" ],
  [ "-red", "-red!50", "-red!20", "-red!80!green!50", "-red!80!blue!60" ],
  [ "-green", "-green!50", "-green!20", "-green!80!red!50",
      "-green!80!blue!60" ],
  [ "-blue", "-blue!50", "-blue!20", "-blue!80!red!50", "-blue!80!green!60" ],
  [ "black!80", "black!70", "black!60", "black!50", "black!40" ],
  [ "black!30", "black!25", "black!20", "black!15", "black!10" ] ]
```

list of colors by tones

```
gap> IP_Colors;
[ "red", "red!50", "red!20", "red!80!green!50", "red!80!blue!60", "green",
  "green!50", "green!20", "green!80!red!50", "green!80!blue!60", "blue",
  "blue!50", "blue!20", "blue!80!red!50", "blue!80!green!60", "-red",
  "-red!50", "-red!20", "-red!80!green!50", "-red!80!blue!60", "-green",
  "-green!50", "-green!20", "-green!80!red!50", "-green!80!blue!60", "-blue",
  "-blue!50", "-blue!20", "-blue!80!red!50", "-blue!80!green!60", "black!80",
  "black!70", "black!60", "black!50", "black!40", "black!30", "black!25",
  "black!20", "black!15", "black!10" ]
```

3.3 The IntPic default list of colors

The colors are shuffled by concatenating the transposed of the matrix `ListsOfIP_Colors`. The list obtained is taken as the default list of colors.

default list of colors

```
gap> ShuffledIP_colors;
[ "red", "green", "blue", "-red", "-green", "-blue", "black!80", "black!30",
  "red!50", "green!50", "blue!50", "-red!50", "-green!50", "-blue!50",
  "black!70", "black!25", "red!20", "green!20", "blue!20", "-red!20",
  "-green!20", "-blue!20", "black!60", "black!20", "red!80!green!50",
  "green!80!red!50", "blue!80!red!50", "-red!80!green!50", "-green!80!red!50",
  "-blue!80!red!50", "black!50", "black!15", "red!80!blue!60",
  "green!80!blue!60", "blue!80!green!60", "-red!80!blue!60",
  "-green!80!blue!60", "-blue!80!green!60", "black!40", "black!10" ]
```

These are the IntPic default colors. Although the user is free to use other colors, we warn that there is a need of compatibility with the colors used in other packages (the LaTeX xcolor, for instance). To emphasize the integers of some sets by using some of the colors in some list of colors (for instance the default colors) one may use empty lists to force the non usage of the colors whose order in the list of colors is the order of these empty lists in the array of integers to be emphasized.

Example

```
gap> m3 := Filtered([1..40],i->i mod 3=0);
[ 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39 ]
gap> m5 := Filtered([1..40],i->i mod 5=0);
[ 5, 10, 15, 20, 25, 30, 35, 40 ]
gap> m7 := Filtered([1..40],i->i mod 7=0);
[ 7, 14, 21, 28, 35 ]
gap>
gap> arr := [[],[],m3,[],m5,[],m7];;
gap> tkz:=IP_TikzArrayOfIntegers([1..40],10,rec(highlights:=arr));;
```

3.4 Functions to deal with colors

For the moment we only provide one function, which shuffles colors from lists of colors.

3.4.1 Shuffle colors from lists of colors

▷ `ShuffleIP_Colors(`*mat*`)` (function)

The argument *mat* is a list of lists of colors of the same length. The output is obtained by concatenating the transposed of *mat*.

Example

```
gap> ShuffleIP_Colors([IP_ColorsRedTones,IP_ColorsCompBlueTones]);
[ "red", "-blue", "red!50", "-blue!50", "red!20", "-blue!20",
  "red!80!green!50", "-blue!80!red!50", "red!80!blue!60", "-blue!80!green!60"
 ]
```

Chapter 4

Visualization of the pictures created

This chapter describes two easy ways to visualize the images created by using the IntPic package. Both require LaTeX and some LaTeX packages, such as Tikz and pgf, to be installed and working. One of the ways we will describe is almost completely automatic. It makes use of the function `Splash`, borrowed from the Viz package. The other is not so automatic but has the advantage of not requiring other packages, besides the LaTeX ones, and should work in any operation system.

4.1 Viewing using Viz

Producing and displaying a picture from a `tikz` string *`tkz`* may me achieved in a simple way. (Warning: extenvive tests have only been done with Linux.) One just have to type the following:

IP_Splash

```
IP_Splash(tkz);
```

A picture is popped up after this use of the function `IP_Splash`.. To see the name of the temporary directory created to perform the computations, and thus being able to copy the files involved to any other place, one should set the info level `InfoViz` to 1 or more. The following example illustrates this and the use of some options of the function `IP_Splash`. For instance, the `pdf` viewer can be changed.

infoviz: temporary directory

```
gap> SetInfoLevel(InfoViz,1);
gap> IP_Splash(tkz,rec(viewer:="okular"));
#I  The temporary directory used is: /tmp/tmJcpphI/
```

The temporary directory /tmp/tmJcpphI/ contains the file and vizpicture.tex. The file vizpicture.tex is the LaTeX document to be processed. Other files, namely the `vizpicture.pdf` are created by the `pdflatex` command that is called by the `IP_Splash` function.

`Warning:` In the case of large pictures, it may happen the LaTeX memory being exceeded. In this case, no image is produced and the user is not warned.

4.2 Viewing without using Viz

This section describes a way to visualize images without sing Viz. Besides being useful in the case of not having a working copy of Viz, it is rather convenient when the decision of where to save the pictures is made. In this case, you may start your gap session in the desired place, the working

directory. Furthermore, if your intention is, for instance, to include the images in a document, you may just decide the name for the file containing the `tikz` code and let your document input it. The glogal variables `IP_Preamble` and `Closing` can be used to pruduce a complete LaTeX document rather than only the `tizk` code for the picture. The document may then be processed by using `pdflatex` and the picture viewed by using some `pdf` viewer. The `pdf` produced can be included in a LaTeX document instead of the `tizk` code. In the later case, the code is processed each time the document is processed, which should perhaps be avoided in the case of large images.

Note the use of the `preview` package, which is used to produce the complete picture without having to pay attention to the paper size nor to crop the image. It is useful for viewing purposes and also to include the `pdf` file produced in a LaTeX document to be processed with `pdflatex`.

Preamble

```
gap> Print(IP_Preamble);
\documentclass{minimal}
\usepackage{amsmath}
\usepackage[active,tightpage]{preview}
\setlength\PreviewBorder{1pt}
\usepackage{pgf}
\usepackage{tikz}
\usepgfmodule{plot}
\usepgflibrary{plothandlers}
\usetikzlibrary{shapes.geometric}
\usetikzlibrary{shadings}
\begin{document}
\begin{preview}
```

Closing

```
gap> Print(IP_Closing);
\end{preview}
\end{document}
```

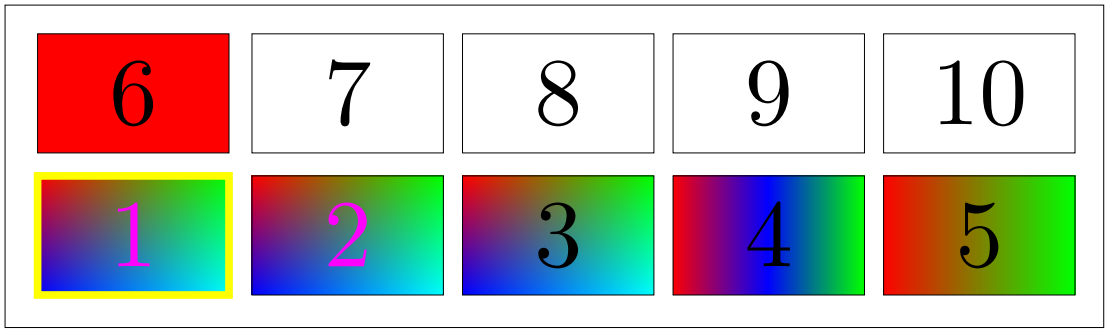
4.2.1 A complete example

Admit you want to produce a document which contains the picture corresponding to the `tikz` code obtained through the instructions

instructions to obtain some tikz code

```
arr := [[1,2,3,4,5,6],[1,2,3,4,5],[1,2,3,4],[1,2,3],[1,2],[1]];;
tkz := IP_TikzArrayOfIntegers([1..10],5,rec(highlights:=arr));;
```

The picture is:



The elements of the set $[1,2,3,4,5,6]$ are highlighted using the first color (red); those of the set $[1,2,3,4,5]$ are highlighted using the second color (green); those of the set $[1,2,3,4]$ are highlighted using the third color (blue); those of the set $[1,2,3]$ are highlighted using the fourth color (cyan); those of the set $[1,2]$ are highlighted using the fifth color (magenta); those of the set $[1]$ is highlighted using the sixth color (yellow).

Let us explain how the six colors used for the cell containing 1 are distributed: upper left corner – red; upper right corner – green; lower left corner – blue; lower right corner – cyan; the number – magenta; the border – yellow.

The colors of the cell containing 2 and 3 are distributed in a similar way.

The colors of the cell containing 4: left – red; middle – blue; right – green.

After the session listed below, the files `tikz_pic_for_complete_document.tex` and `pic_for_complete_document.tex` have been created in the current directory (that is, the one where the GAP session has started). For other directories, complete paths may have to be given.

the GAP session

```
gap> tikzfile := "tikz_pic_for_complete_document.tex";;
gap> file := "pic_for_complete_document.tex";;
gap>
gap> arr := [[1,2,3,4,5,6],[1,2,3,4,5],[1,2,3,4],[1,2,3],[1,2],[1]];;
gap> tkz := IP_TikzArrayOfIntegers([1..10],5,rec(highlights:=arr));;
gap>
gap> FileString(tikzfile,tkz);
642
gap> FileString(file,Concatenation(IP_Preamble,tkz,IP_Closing));
961
```

Executing something like

the pdf and the jpg of the picture

```
pdflatex pic_for_complete_document.tex
convert pic_for_complete_document.pdf pic_for_complete_document.jpg
```

the `pdf` and the `jpg` formats of the image have been created. The `jpg` format is usefull to be included into an html document, for instance.

Note that the tikz code has been saved into the file `tikz_pic_for_complete_document.tex`. A complete example of a LaTeX document follows.

a LaTeX document

```
\documentclass{article}
\usepackage{amsmath}
%\usepackage[active,tightpage]{preview}
%\setlength\PreviewBorder{1pt}
\usepackage{pgf}
\usepackage{tikz}
\usepgfmodule{plot}
\usepgflibrary{plothandlers}
\usetikzlibrary{shapes.geometric}
\usetikzlibrary{shadings}
\usepackage{graphicx}
\author{Author}
\title{How to include images in a \LaTeX\ document}
\date{June, 2013}
\begin{document}
```

```
%\begin{preview}
\maketitle
Using the pdf file:

\begin{center}
  \includegraphics[width=0.80\textwidth]{../images/pic_for_complete_document.pdf}
\end{center}

Using the PGF/TikZ code:

\begin{center}
\input{../images/tikz_pic_for_complete_document.tex}
\end{center}
If you want to scale this immage, please chang the ``scale'' in the file
\textt{tikz_pic_for_complete_document.tex}
%\end{preview}
\end{document}
```

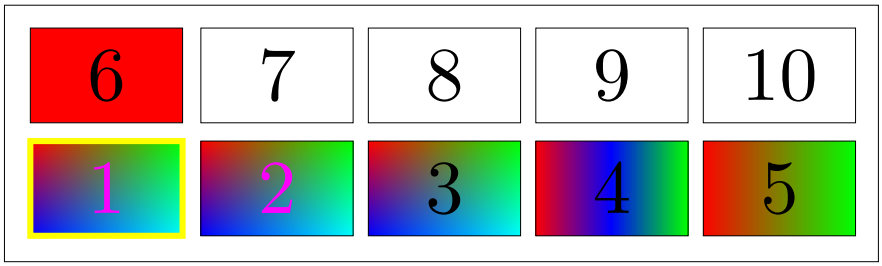
The output, after processing with `pdflatex` is as follows:

How to include images in a LaTeX document

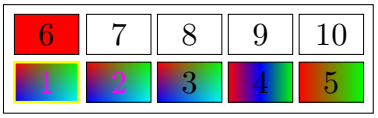
Author

June, 2013

Using the pdf file:



Using the PGF/TikZ code:



If you want to scale this image, please chang the "scale" in the file `tikz_pic_for_complete_document.tex`

4.3 Other examples of use of the IntPic package

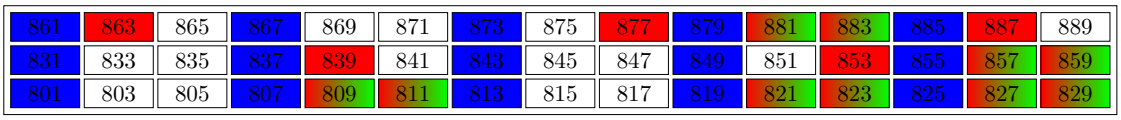
4.3.1 Varia

The following example shows how to produce `tikz` code for a picture containing the odd integers from 801 to 999. Each line (except the highest) contains 15 cells.

Example

```
gap> rg := Filtered([801..889],u->(u mod 2)<>0);;
gap> flen := 15;;
gap> twins := Filtered(Primes, p -> p + 2 in Primes);;
gap> arr := [Primes,Union(twins,twins+2),Filtered(rg,u->(u mod 3)=0)];;
gap> tkz := IP_TikzArrayOfIntegers(rg,flen,rec(highlights:=arr));;
```

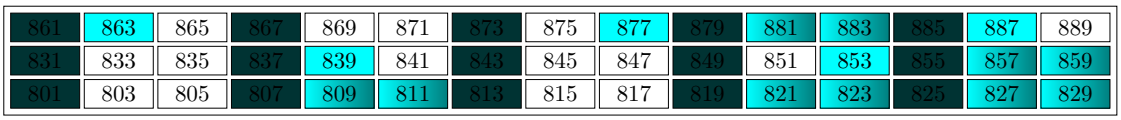
The picture obtained highlights the primes, the twin primes and the multiples of 3. As the twins are also primes, a gradient is used to highlight them. In this example the default list of colors is used.



The same computations, but defining other color lists.

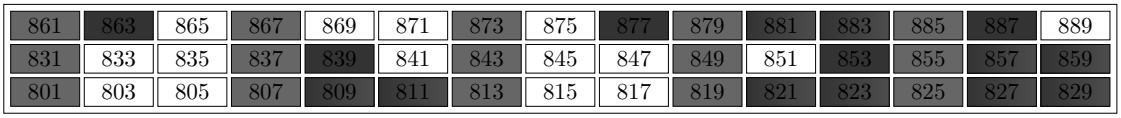
Example

```
gap> cls := IP_ColorsCompRedTones;;
gap> rg := Filtered([801..889],u->(u mod 2)<>0);;
gap> flen := 15;;
gap> twins := Filtered(Primes, p -> p + 2 in Primes);;
gap> arr := [Primes,Union(twins,twins+2),Filtered(rg,u->(u mod 3)=0)];;
gap> tkz := IP_TikzArrayOfIntegers(rg,flen,rec(colors := cls,highlights:=arr));;
```



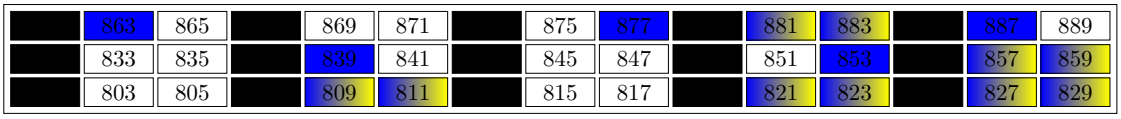
Example

```
gap> cls := IP_ColorsDGrayTones;;
gap> rg := Filtered([801..889],u->(u mod 2)<>0);;
gap> flen := 15;;
gap> twins := Filtered(Primes, p -> p + 2 in Primes);;
gap> arr := [Primes,Union(twins,twins+2),Filtered(rg,u->(u mod 3)=0)];;
gap> tkz := IP_TikzArrayOfIntegers(rg,flen,rec(colors := cls,highlights:=arr));;
```



Example

```
gap> cls := ["blue","-blue","black"];;
gap> rg := Filtered([801..889],u->(u mod 2)<>0);;
gap> flen := 15;;
gap> twins := Filtered(Primes, p -> p + 2 in Primes);;
gap> arr := [Primes,Union(twins,twins+2),Filtered(rg,u->(u mod 3)=0)];;
gap> tkz := IP_TikzArrayOfIntegers(rg,flen,rec( colors := cls,highlights:=arr));;
```

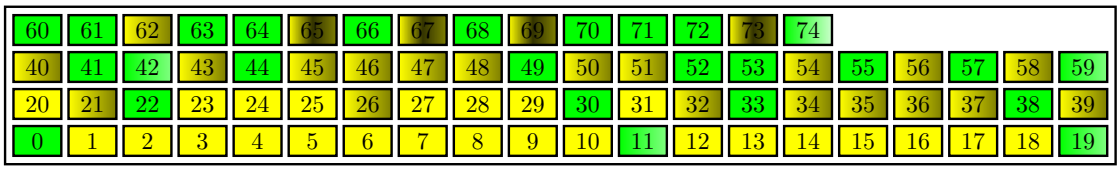


The following example uses the NumericalSgps package.

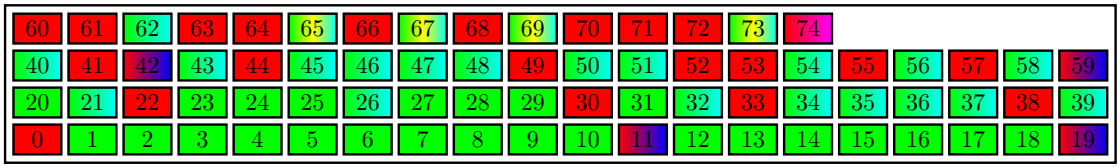
Example

```
gap> #LoadPackage("numericalsgps");
gap>
gap> ns := NumericalSemigroup(11,19,30,42,59);;
gap> cls := ShuffleIP_Colors([IP_ColorsGreenTones,IP_ColorsCompBlueTones]);;
gap> flen := 20;;
gap> #some notable elements
gap> arr := [SmallElementsOfNumericalSemigroup(ns),
```

```
>          GapsOfNumericalSemigroup(ns),
>          MinimalGeneratingSystemOfNumericalSemigroup(ns),
>          FundamentalGapsOfNumericalSemigroup(ns),
>          [ConductorOfNumericalSemigroup(ns)],
>          PseudoFrobeniusOfNumericalSemigroup(ns)];;
gap>
gap> tkz := IP_TikzArrayOfIntegers(flen,rec(colors := cls,highlights:=arr));;
```



Using the default colors



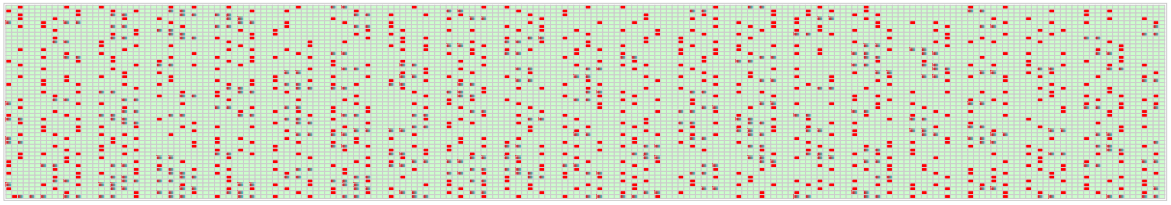
4.3.2 The banner

The code in the following example has been used to produce one possible banner for the homepage of the IntPic package. It is a nice picture that gives an idea about the primes less than 10000. Of course, other ranges could have been chosen. I warn the user that pictures involving a large amount of data may face the problem of exceeding TEX capacity...

Example

```
gap> row_length := 200;; # the legth of each row
gap> columns := 50;; # the number of colums
gap> n := row_length*columns;
10000
gap>
gap> ##compute the primes less than n
gap> # Primes is a GAP variable representing the list of primes less than 1000
gap> mp := Maximum(Primes);
997
gap> newprimes := [];;
gap> while mp < n do
>   mp := NextPrimeInt(mp);
>   Add(newprimes, mp);
> od;
gap> small_primes := Union(Primes, newprimes);;
gap> ##compute the first element of each pair of twin primes less than n
gap> twins := Filtered(small_primes, p -> IsPrime(p+2));;
gap>
gap> rg := [1..n];;
gap>
gap> arr := [Intersection(small_primes,rg),[],[],
>          Intersection(Union(twins,twins+2),rg),[],[],[],[],[],[],[],
```

```
>          [],[],[],[],[],[],Difference(rg,small_primes)];;
gap>
gap> tkz:=IP_TikzArrayOfIntegers([1..n],row_length,rec(highlights:=arr,
>              cell_width := "6",colsep:="0",rowsep:="0",inner_sep:="2",
>              shape_only:=" ",line_width:="0",line_color:="black!20" ));;
```



Chapter 5

The IntPic package options.

5.1 Available options

The list of allowed options, some of which already familiar from the examples, can be obtained as follows:

Example

```
gap> RecNames(IP_TikzDefaultOptionsForArraysOfIntegers);
[ "other", "colors", "highlights", "shape_only", "colsep", "rowsep",
  "cell_width", "allow_adjust_cell_width", "scale", "inner_sep",
  "line_width", "line_color" ]
```

Its meaning is as follows:

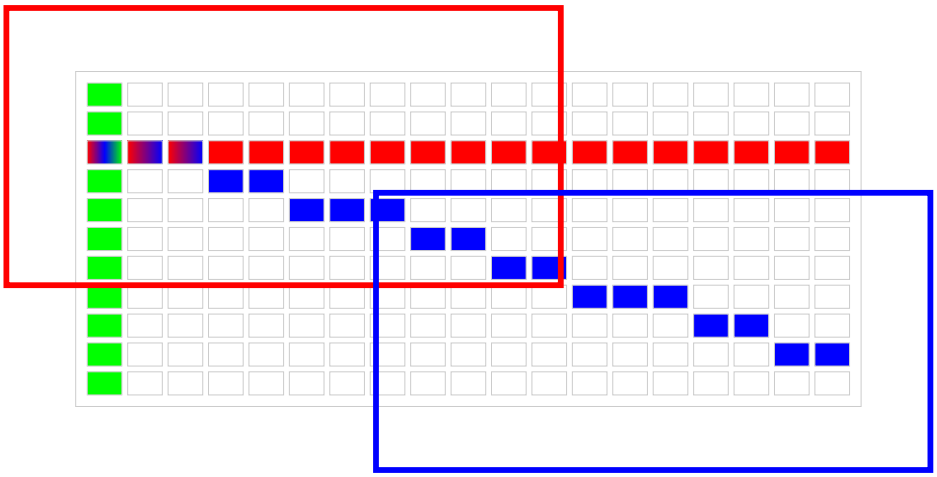
- *colors*: any list of colors (to be used with the LaTeX package xcolor)
- *highlights*: a list of lists of integers – the elements of the first are colored by using the first color, etc. In cases of elements that appear in more than one list a kind of gradient (made with 4 colors at most) is used to fill the cell; the number may be printed with a fifth color and a sixt color may be used for the border.
- *shape_only*: an option to be used when only the shape is important. When *true* or " " is used, all the nodes are empty; using a symbol, for instance a $*$, this symbol is printed in all the nodes.
- *colsep*: the tikz matrix option *column sep*
- *rowsep*: the tikz matrix option *row sep*
- *cell_width*: the tikz matrix option *minimum width*
- *scale*: the tikz matrix option *scale*
- *inner_sep*: the tikz matrix option *inner sep*
- *line_width*: the tikz matrix option *line width*
- *line_color*: the tikz matrix option *line color*: the color of the cell borders

- *allow_adjust_cell_width*: the number of points per digit (to avoid discrepancies between the width of the cells when there are numbers with different number of digits to be printed). When the user sets the option cell_width, then allow_adjust_cell_width is automatically set to *false*
- *other*: if non empty, the complete tikz code has to be written (it may be useful when several images are to be produced - otherwise, changing the tikz code would be enough)

Example

```
other := ["\\draw[postaction={draw,line width=1pt,red}] (-80pt,-8pt)
rectangle (16pt,40pt);","\\draw[postaction={draw,line width=1pt,blue}]
(-16pt,8pt) rectangle (80pt,-40pt);"]
```

Adding this option to one of the preceding examples, one obtains the following:



5.2 Default options

The defaults for the available options are as follows

- *colors*: ShuffledIP_colors
- *highlights*: [[]]
- *shape_only* : "false"
- *colsep*: "2"
- *rowsep*: "2"
- *cell_width*: "30"
- *scale*: "1"
- *inner_sep*: "3"
- *line_width*: "0"
- *line_color*: "black"

- *allow_adjust_cell_width*: "10"
- *other*: []

They may be consulted:

Example

```
gap> IP_TikzDefaultOptionsForArraysOfIntegers;
rec( allow_adjust_cell_width := "10", cell_width := "30",
  colors := [ "red", "green", "blue", "-red", "-green", "-blue", "black!80",
      "black!30", "red!50", "green!50", "blue!50", "-red!50", "-green!50",
      "-blue!50", "black!70", "black!25", "red!20", "green!20", "blue!20",
      "-red!20", "-green!20", "-blue!20", "black!60", "black!20",
      "red!80!green!50", "green!80!red!50", "blue!80!red!50",
      "-red!80!green!50", "-green!80!red!50", "-blue!80!red!50", "black!50",
      "black!15", "red!80!blue!60", "green!80!blue!60", "blue!80!green!60",
      "-red!80!blue!60", "-green!80!blue!60", "-blue!80!green!60",
      "black!40", "black!10" ], colsep := "2", highlights := [ [  ] ],
  inner_sep := "3", line_color := "black", line_width := "1", other := [  ],
  rowsep := "2", scale := "1", shape_only := "false" )
```

The user may want to change the defaults by editing the file `options.gd` in the folder `gap`. The changes are lost whenever any re-installation occurs. It is recommended that in this case a copy is made, although it is not guaranteed that it will work in the next release.

Chapter 6

Usage of IntPic in connection with the numericalsgps package

This chapter describes functions to be used in connection with the numericalsgps package. We found it particularly usefull to gain intuition that led to the obtention of the results stated in [DGSRP15].

6.1 Tikz code for drawing numerical semigroups

6.1.1 Tikz code for drawing numerical semigroups

▷ `TikzCodeForNumericalSemigroup(`*arg*`)` (function)

The arguments (at most 4) are:

1. a numerical semigroup
2. (optional) a list whose elements are either
   - lists of integers *or*
   - one of the strings "pseudo_frobenius", "small_elements", "min_generators", "frobenius_number", "conductor", "special_gaps", "fundamental_gaps" (the default: used when no list is present) *or*
3. a record whose fields are
   - func – a function name
   - (optional) argument – an argument (that may be a function name also)
4. (optional) a positive integer – if it is bigger than the conductor or biggest minimal generator, it indicates the number of cells - 1 to be drawn and these are drawn in a single line; otherwise, it indicates the maximum number of cells per line.

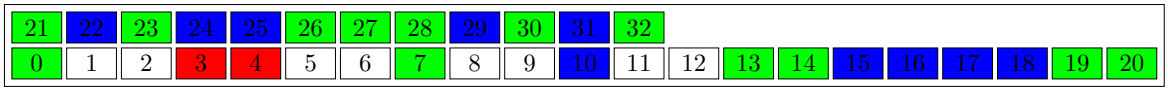
Example

```
gap> ns1 := NumericalSemigroup(3,5);;
gap> TikzCodeForNumericalSemigroup(ns1,[[3,4],"pseudo_frobenius"],20);
"%tikz\n\\begin{tikzpicture}[every node/.style={draw,scale=1pt,\nminimum width\
=20pt,inner sep=3pt,\nline width=0pt,draw=black}]\n\\matrix[row sep=2pt,column\
```

```
 sep=2pt]\n{\\node[]{0};&\n\\node[]{1};&\n\\node[]{2};&\n\\node[fill=red]{3};&\
\n\\node[fill=red]{4};&\n\\node[]{5};&\n\\node[]{6};&\n\\node[fill=green]{7};&\
\n\\node[]{8};&\n\\node[]{9};&\n\\node[]{10};&\n\\node[]{11};&\n\\node[]{12};&\
\n\\node[]{13};&\n\\node[]{14};&\n\\node[]{15};&\n\\node[]{16};&\n\\node[]{17}\
;&\n\\node[]{18};&\n\\node[]{19};&\n\\node[]{20};\\\\\n};\n\\end{tikzpicture}\
\n"
```

Example

```
gap> IP_Splash(TikzCodeForNumericalSemigroup(NumericalSemigroup(7,13,19,23),
> [[3,4],"small_elements","fundamental_gaps"],20),rec(viewer := "evince"));
```



6.2 Drawing sets of numerical semigroups

6.2.1 Set of numerical semigroups

▷ `SetOfNumericalSemigroups(arg)` (function)

This function is used to produce lists of numerical semigroups with a fixed genus or Frobenius number. They are filtered and ordered according to some criteria.

The argument is a record of options:

1. set – a record whose possible fields are genus or frobenius
2. (optional) filter – a record whose possible fields are genus, type and/or multiplicity and/or frobenius and/or embedding_dimension
3. (optional) order – ("genus", "type", "multiplicity", "frobenius", "embedding_dimension")

Example

```
gap> SetOfNumericalSemigroups(rec(set:=rec(genus:=6),filter:=rec(type:= 2),
> order:="multiplicity"));
[ <Numerical semigroup with 3 generators>,
  <Numerical semigroup with 3 generators>,
  <Numerical semigroup with 4 generators>,
  <Numerical semigroup with 5 generators>,
  <Numerical semigroup with 5 generators>,
  <Numerical semigroup with 6 generators> ]
gap> SetOfNumericalSemigroups(rec(set:=rec(genus:=6),filter:=rec(type:= 2),
> order:="frobenius"));
[ <Numerical semigroup with 3 generators>,
  <Numerical semigroup with 5 generators>,
  <Numerical semigroup with 5 generators>,
  <Numerical semigroup with 3 generators>,
  <Numerical semigroup with 4 generators>,
  <Numerical semigroup with 6 generators> ]
gap> List(last,MinimalGeneratingSystem);
[ [ 3, 10, 11 ], [ 5, 6, 7 ], [ 5, 6, 8 ], [ 3, 8, 13 ], [ 4, 7, 9 ],
  [ 6, 7, 8, 9, 11 ] ]
```

6.2.2 Draw a set of numerical semigroups

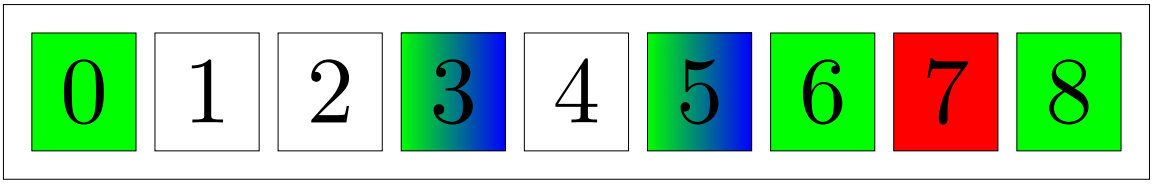
▷ `DrawSetOfNumericalSemigroups(arg)` (function)

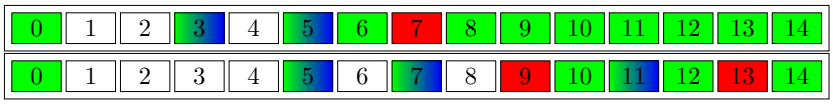
Produces a single image from the images of a set of numerical semigroups.
The arguments (at most 3) are:

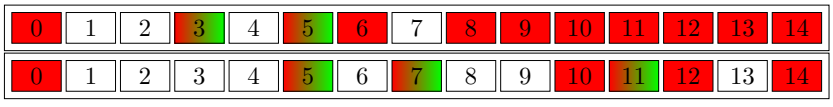
1. a list of numerical semigroups (given as a list or each given as argument)
2. (optional) an integer that (when present) determines the length of each line
3. (optional) a record whose fields are
   - (optional) splash – which (when present) consists of a record of options for the Viz Splash function
   - (optional) highlights: a list to be passed to the function that produces the tikz code for each individual semigroup (whose aim is to say which elements are to be highlighted)

Example

```
gap> ns1 := NumericalSemigroup(3,5);;
gap> ns2 := NumericalSemigroup(5,7,11);;
gap> DrawSetOfNumericalSemigroups(ns1,rec(splash:=
> rec(viewer := "evince"),highlights :=
> ["pseudo_frobenius","small_elements","min_generators"]));
gap> DrawSetOfNumericalSemigroups(ns1,ns2,rec(splash:=
> rec(viewer := "evince"),highlights :=
> ["pseudo_frobenius","small_elements","min_generators"]));
gap> DrawSetOfNumericalSemigroups([ns1,ns2],rec(splash:=
> rec(viewer := "evince"),highlights :=
> ["small_elements","min_generators"]));
```



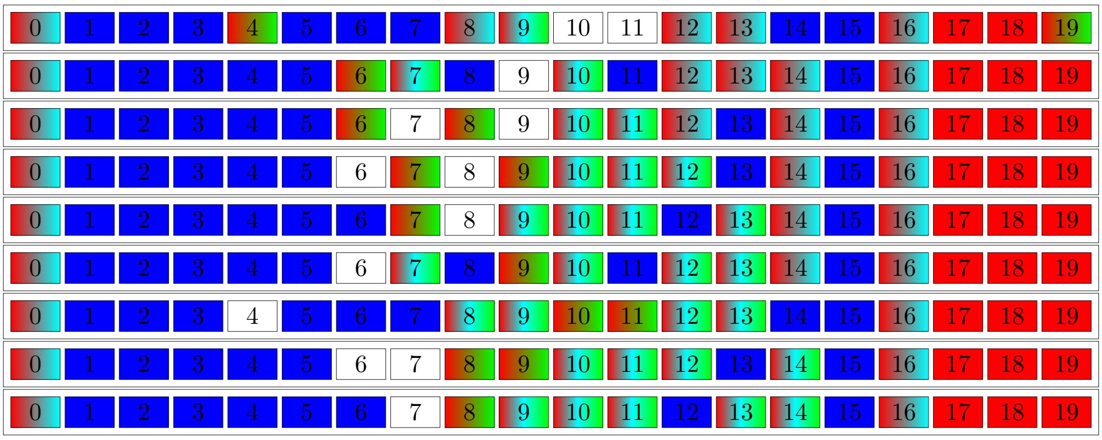




Example

```
gap> $lter := rec(type:= tipo),order := "embedding_dimension"));;
gap> $doFrobeniusOfNumericalSemigroup")]));
gap> frob := 15;;
gap> tipo := 2;;
gap> set := SetOfNumericalSemigroups(rec(set := rec(frobenius := frob),
> filter := rec(type:= tipo),order := "embedding_dimension"));;
gap> DrawSetOfNumericalSemigroups(set,rec(splash:= rec(viewer := "evince"),
```

```
> highlights := ["small_elements","min_generators",
> rec(func:= "ForcedIntegersForPseudoFrobenius",
> argument := "PseudoFrobeniusOfNumericalSemigroup")]));
```



The following example helps in the understanding of the colors present in the first line of previous figure.

Example

```
gap> ns := NumericalSemigroup(4,9,19);;
gap> SmallElements(ns);
[ 0, 4, 8, 9, 12, 13, 16 ]
gap> MinimalGeneratingSystem(ns);
[ 4, 9, 19 ]
gap> ForcedIntegersForPseudoFrobenius(PseudoFrobeniusOfNumericalSemigroup(ns));
[ [ 1, 2, 3, 5, 6, 7, 14, 15 ], [ 0, 8, 9, 12, 13, 16 ] ]
```

References

[DENMP12] M. Delgado, A. Egri-Nagy, J. D. Mitchell, and M. Pfeiffer. *Viz - a GAP package for drawing GAP objects*, 2012. Under development. 5

[DFGSL14] Manuel Delgado, José I. Farrán, Pedro A. García-Sánchez, and David Llena. On the weight hierarchy of codes coming from semigroups with two generators. *IEEE Trans. Inform. Theory*, 60(1):282–295, 2014. 5, 9

[DGSM15] M. Delgado, P. A. García-Sánchez, and J. Morais. *NumericalSgps – a GAP package for numerical semigroups*, June 2015. Version number 1.0. 4

[DGSRP15] M. Delgado, P. A. García-Sánchez, and A. M. Robles-Pérez. Numerical semigroups with a given set of pseudo-Frobenius numbers. *ArXiv e-prints*, May 2015. 27

[DM08] M. Delgado and J. Morais. *SgpViz – a user-friendly GAP package to deal with semigroups*, May 2008. Version number 0.998. 4

Index