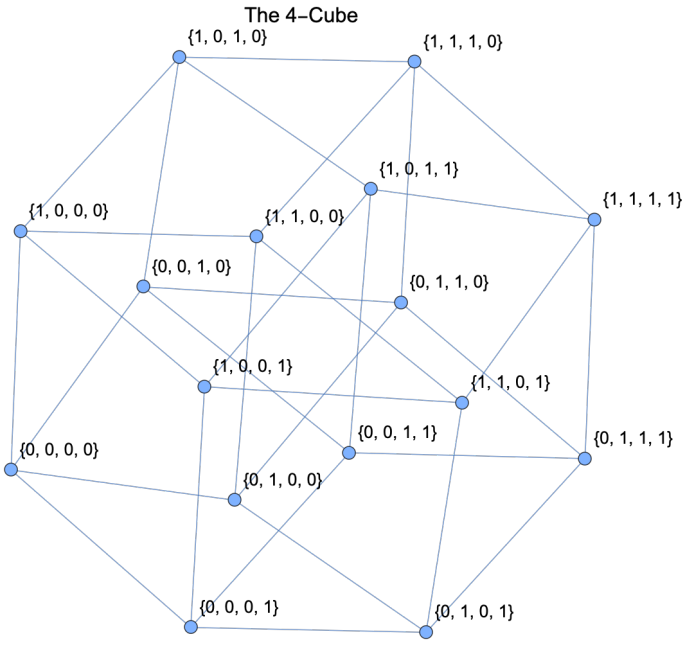
The Gray Code

2026, Kenneth Levasseur
Mathematics & Statistics
UMass Lowell
Kenneth_Levasseur@uml.edu

Out[130]=



The Gray Code is a Hamiltonian cycle through the n-Cube. It is a list of the 2^n bit strings of length n with the property that each string differs from its predecessor in exactly one position. The last string is understood to be connected to the first.

Here is the Gray Code for the 4-cube with each string converted to a base 10 integer.

In[•]:=
```
{GrayCode[4], Map[FromDigits[#, 2] &, GrayCode[4]]} // Transpose
```

Out[•]=

{0, 0, 0, 0}	0
{0, 0, 0, 1}	1
{0, 0, 1, 1}	3
{0, 0, 1, 0}	2
{0, 1, 1, 0}	6
{0, 1, 1, 1}	7
{0, 1, 0, 1}	5
{0, 1, 0, 0}	4
{1, 1, 0, 0}	12
{1, 1, 0, 1}	13
{1, 1, 1, 1}	15
{1, 1, 1, 0}	14
{1, 0, 1, 0}	10
{1, 0, 1, 1}	11
{1, 0, 0, 1}	9
{1, 0, 0, 0}	8

This isn't the usual ordering of the integers from 0 to 15 but there are reasons why it's a better ordering such as in many electronics applications. So it's of interest to know how to convert back and forth between the usual sequence of integers from 0 to $2^n - 1$ to the Gray code ordering of the same integers.

By the way, the Gray Code isn't the only Hamiltonian cycle on the 4-cube but it's one of the simplest one to construct.

If we consider the entries in the Gray Code to be integers in base 2, the numbers are as follows for the 4-cube.

In[•]:=
```
Map[FromDigits[#, 2] &, GrayCode[4]]
```

Out[•]=

{0, 1, 3, 2, 6, 7, 5, 4, 12, 13, 15, 14, 10, 11, 9, 8}

This is a permutation of the integers from 0 to 15. The next output is the "matrix representation" of this as a function on {0, 1, 2, ..., 13, 14, 15}.

In[•]:=
```
{Range[0, 15], Map[FromDigits[#, 2] &, GrayCode[4]]}
```

Out[•]=

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	3	2	6	7	5	4	12	13	15	14	10	11	9	8

There are two related questions we might ask:
1. If $0 \le k \le 2^n - 1$, what number is in position *k?*
2. If $0 \le k \le 2^n - 1$, where is *k* in the list?

Note: In answering these questions, keep in mind that positions are indexed starting at 0, not 1.

For example, the number position 7 of the Gray Code for the 4-cube is 4. We say that the conversion **to** the Gray Code of 7 is 4.

If we look for 7 in the in the Gray Code for the 4-cube, we find it in position 5. We say that the conversion **from** the Gray Code of 7 is 5

A function to answer the first question, what number is in a given position?

The function is designed to take two modes of input, but the output is always a sting of bits that we can easily convert to an integer using **FromDigits**. The code for this and other functions used in this Notebook is in the Initializations Section.

In[•]:=
```
? ToGray
```

Out[•]=

Symbol

ToGray[bitlist] returns the bit string that is in
position bitlist of the Gray Code on n bits, where n is the length of bitlist.
ToGray[k,n] returns the bitstring that is in position k in the Gray code on n bits.

In[119]:=
```
ToGray[7, 4]
```

Out[119]=

$\{0, 1, 0, 0\}$

In[•]:=
```
ToGray[{0, 1, 1, 1}]
```

Out[•]=

$\{0, 1, 0, 0\}$

In[•]:=
```
FromDigits[ToGray[{0, 1, 1, 1}], 2]
```

Out[•]=

4

A function to answer the second question, where is a given number?:

FromGray has the same basic design as **ToGray**.

In[•]:=
```
? FromGray
```

Out[•]=

Symbol

FromGray[k,n] returns the k'th entry in the Gray code on n bits. Caution: numbering starts with 0.
FromGray[bitlist] returns the location of
bitlist on the Gray Code on n bits, where n is the length of bitlist.

In[•]:=
```
FromGray[{0, 1, 1, 1}]
```

Out[•]=

$\{0, 1, 0, 1\}$

In[•]:=
```
FromGray[7, 4]
```

Out[•]=

$\{0, 1, 0, 1\}$

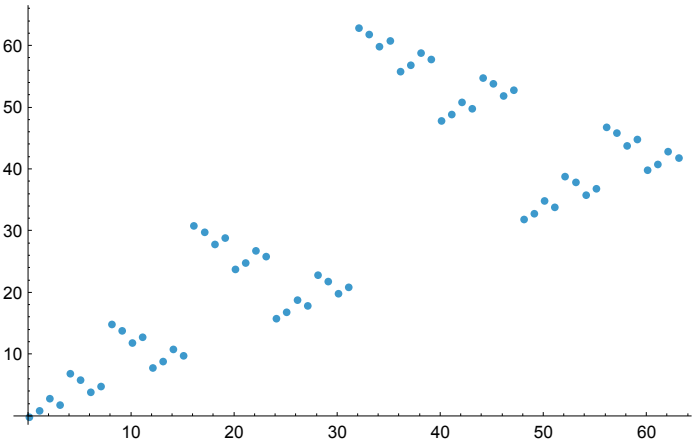
In[•]:=
```
FromDigits[FromGray[{0, 1, 1, 1}], 2]
```

Out[•]=
```
5
```

Interesting Visuals

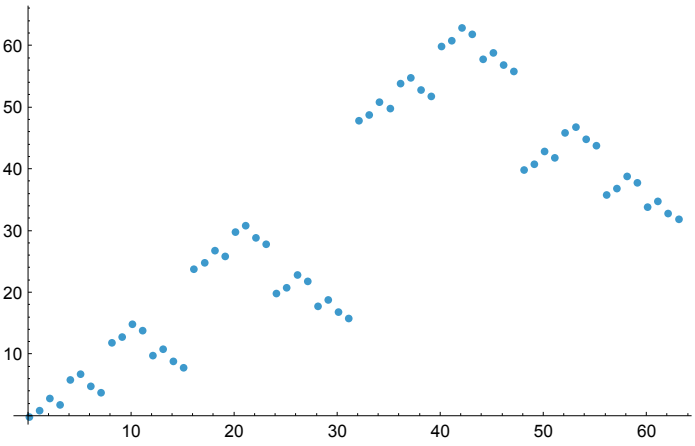
In[•]:=
```
ListPlot[Map[{#, FromDigits[FromGray[#, 6], 2]} &, Range[0, 2^6 - 1]]]
```

Out[•]=



In[•]:=
```
ListPlot[Map[{#, FromDigits[ToGray[#, 6], 2]} &, Range[0, 2^6 - 1]]]
```

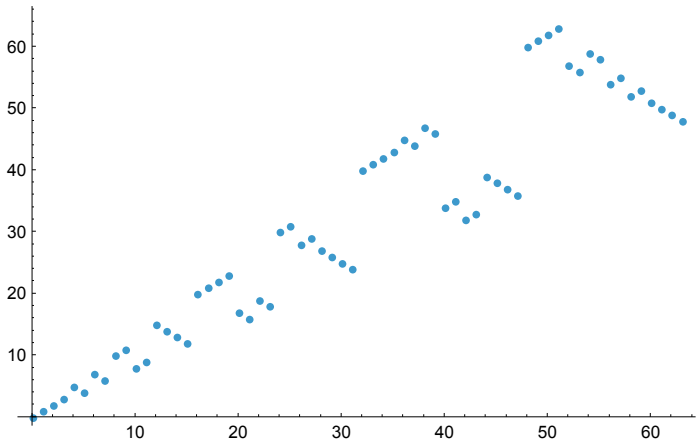
Out[•]=



Here is a composition of the ToGray function with itself.

```
Map[{#, FromDigits[ToGray[ToGray[#, 6]], 2]} &, Range[0, 63]] // ListPlot
```

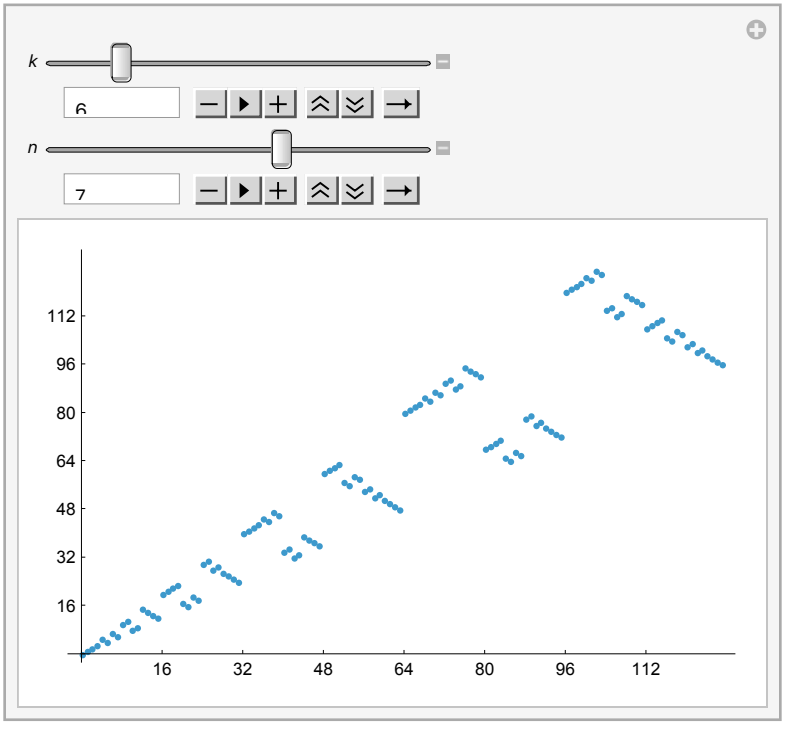
Out[ ]=



In[95]:=

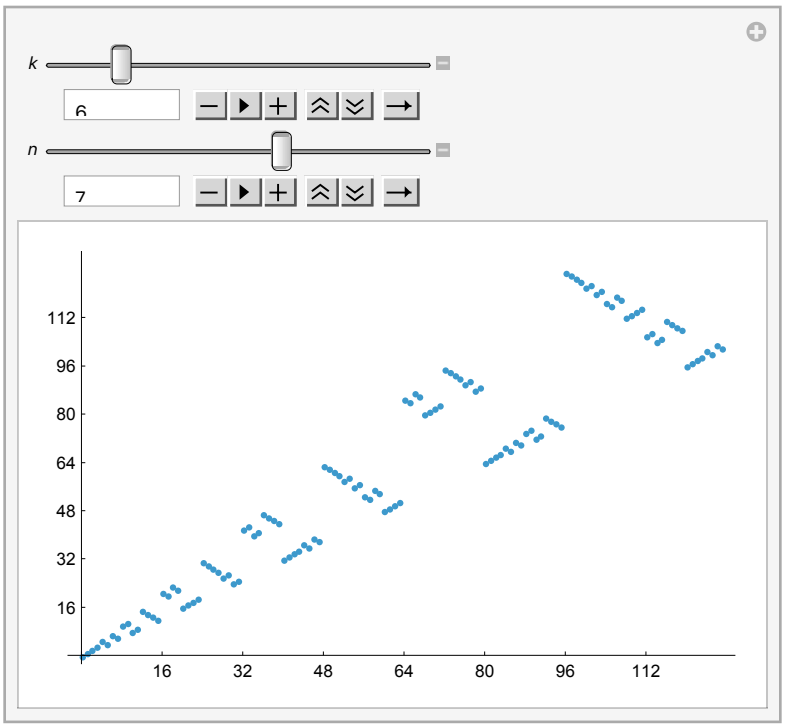
```
Manipulate[
 Map[{#, FromDigits[Nest[FromGray, FromGray[#, n], k - 1], 2]} &, Range[0, 2^n - 1]] //
  ListPlot[#, Ticks → {Range[0, 2^n - 1, Max[1, 2^(n - 3)]],
      Range[0, 2^n - 1, Max[1, 2^(n - 3)]]}] &, {k, 1, 32, 1}, {n, 2, 10, 1}]
```

Out[95]=



In[150]:=
```
Manipulate[
 Map[{#, FromDigits[Nest[ToGray, ToGray[#, n], k - 1], 2]} &, Range[0, 2^n - 1]] //
  ListPlot[#, Ticks → {Range[0, 2^n - 1, Max[1, 2^(n - 3)]],
      Range[0, 2^n - 1, Max[1, 2^(n - 3)]]}] &, {k, 1, 32, 1}, {n, 2, 10, 1}]
```

Out[150]=



Timing of the functions on the 512-Cube.

In[ ]:=
```
n = 512;
s = RandomInteger[{0, 1}, n - 1] // Prepend[#, 1] &;
k = FromDigits[s, 2];
{Timing[FromGray[k, n];], Timing[FromGray[s];]}
```

Out[ ]=

$$\begin{pmatrix} 0.000833 & \text{Null} \\ 0.000766 & \text{Null} \end{pmatrix}$$

In[ ]:=
```
n = 512;
s = RandomInteger[{0, 1}, n - 1] // Prepend[#, 1] &;
k = FromDigits[s, 2];
{Timing[ToGray[k, n];], Timing[ToGray[s];]}
```

Out[ ]=

$$\begin{pmatrix} 0.000029 & \text{Null} \\ 5.\times 10^{-6} & \text{Null} \end{pmatrix}$$

The order of the Gray Code as a permutation,

The GrayOrder function determines the order of the Gray Code function on the n-Cube.

In[120]:=
```
GrayOrder[6]
```

Out[120]=
8

Conjecture : The order of the n^{th} Gray Code is $2^{\lceil \log_2(n) \rceil}$,
which implies that the order on the 2^m – cube is 2^m.

In[•]:=
```
Map[{#, GrayOrder[#], 2^Ceiling[Log[2, #]]} &, Range[1, 17]]
```

Out[•]=

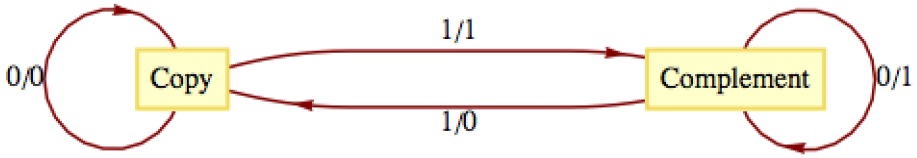
1	1	1
2	2	2
3	4	4
4	4	4
5	8	8
6	8	8
7	8	8
8	8	8
9	16	16
10	16	16
11	16	16
12	16	16
13	16	16
14	16	16
15	16	16
16	16	16
17	32	32

Gray Code Decoder

An exercise in Chapter 14 of Applied Discrete Structures introduces the Gray Code Decoder, which is the same as the function `FromGray`. Starting in Copy state, if the bits in a string are used as input from left to right, the output will be the location of that string in the Gray Code.

The state machine for the Gray Code decoder.

Out[•]=



In[151]:=
```
GrayDecoder::usage =
 "GrayDecoder[bitlist] returns the location of bitlist on the
   Gray Code on n bits, where n is the length of bitlist."
```

Out[151]=
GrayDecoder[bitlist] returns the location of bitlist on the Gray Code on n bits, where n is the length of bitlist.

In[152]:=
```
GrayDecoder[seq_List] := Module[{state}, state = 0;
  Map[If[state == 0, state = #;
     #, state = 1 - #;
     1 - #] &, seq]]
```

In[146]:=
```
GrayDecoder[{1, 0, 1, 1, 0}]
```

Out[146]=
{1, 1, 0, 1, 1}

In[147]:=
```
FromGray[{1, 0, 1, 1, 0}]
```

Out[147]=
{1, 1, 0, 1, 1}

Initializations

Tests and further experiments