Enumerative Combinatoric Algorithms

Gray code
Binary code

Standard binary code:
Ex, 3 bits:

<table>
<thead>
<tr>
<th>Binary</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>000$^b$</td>
<td>0</td>
</tr>
<tr>
<td>001$^b$</td>
<td>1</td>
</tr>
<tr>
<td>010$^b$</td>
<td>2</td>
</tr>
<tr>
<td>011$^b$</td>
<td>3</td>
</tr>
<tr>
<td>100$^b$</td>
<td>4</td>
</tr>
<tr>
<td>101$^b$</td>
<td>5</td>
</tr>
<tr>
<td>110$^b$</td>
<td>6</td>
</tr>
<tr>
<td>111$^b$</td>
<td>7</td>
</tr>
</tbody>
</table>

Problem:
In the transition from 3 to 4 every bit changes.
Thus, if timing is not absolutely perfect (it never is) then any number from 0 to 7 can be read.
e.g.: position encoder, electronic circuits, ...
Gray code

- a.k.a. reflected binary code (Frank Gray, 1947)
- binary Gray code: successive values differ in only one bit

generate a Gray code \(G(n)\) for \(n\) bits from \(G(n-1)\):

\[
\begin{align*}
G(n) & \\
0 & \vdots \\
\vdots & \vdots \\
(n-1) & \\
\downarrow & \\
0 & 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \\
1 & 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \\
\vdots & \vdots \\
0 & 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \\
1 & 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \\
\end{align*}
\]

“reflected” binary code

Observe that this way also the last and the first number differ only in one position.
Uses – applications

• initially for analog to digital conversion for compatible color television signals

• also used in telegraphy and position encoders (linear and rotary)

• error correction in digital communications

• genetic algorithms

• solving puzzles (Towers of Hanoi, Chinese rings puzzle, Spin-out, ...)

binary ↔ Gray code conversion

\[ b_{n-1} b_{n-2} \ldots b_1 b_0 \quad \leftrightarrow \quad g_{n-1} g_{n-2} \ldots g_1 g_0 \]

\[ b_{n-1} = g_{n-1} \]

\[ \forall 0 \leq i < n - 1 \]

\[ b_i = b_{i+1} \text{ XOR } g_i \quad g_i = b_{i+1} \text{ XOR } b_i \]

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>X XOR Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Ex: standard binary 1010_b \rightarrow ?

Ex: Gray code 01001_g \rightarrow ?
Puzzle: Tower of Hanoi

**Given**: A stack of \( n \) different sized disks, arranged from largest on the bottom to smallest on top, placed on a rod, \( A \). Plus two more empty rods, \( B \) and \( C \).

**Valid move**: move one disk from one rod to another one but not placing a larger disk on top of a smaller disk.

**Task**: Move the stack (tower) from rod \( A \) to rod \( B \) (using rod \( C \)) (\ldots with minimum number of moves)

**Solution (recursive)**: to move a tower, \( T(n) \), of size \( n \) from \( A \) to \( B \) using \( C \): move \( T(n-1) \) from \( A \) to \( C \) using \( B \), then the largest disk from \( A \) to \( B \), and then move \( T(n-1) \) from \( C \) to \( B \) using \( A \).
Tower of Hanoi vs. Gray code

- Label the disks from 0 (smallest) to \( n - 1 \) (largest) \( \equiv g_{n-1} g_{n-2} \ldots g_1 g_0 \ldots \) a number in Gray code.

- The changes in the Gray code (if bit \( i \) changes) define the disk (disk \( i \)) that has to be moved:
  - Disk 0 always cw (or always ccw, depending on parity).
  - Disks \( i, 0 < i < n \), unique way to move.

Ex: \( n = 4 \)

\[
\begin{array}{cccc}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & 1 & 1 \\
0 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 \\
0 & 1 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 1 & 0 & 0 \\
1 & 1 & 0 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 \\
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 \\
1 & 0 & 1 & 1 \\
1 & 0 & 1 & 0 \\
1 & 1 & 0 & 1 \\
1 & 1 & 1 & 0 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 \\
1 & 1 & 0 & 1 \\
1 & 1 & 0 & 0 \\
\end{array}
\]

Start: all 4 disks on A
Goal: all 4 disks on B

\( n \) is even \( \rightarrow \) move 0 ccw
Tower of Hanoi vs. Gray code

Ex: 
n = 4

start: all 4 disks on A
goal: all 4 disks on B

A

B

C

A

B

C

n is even → move 0 ccw

Oswin Aichholzer (slides TH): Enumerative Combinatoric Algorithms, 2017
**Tower of Hanoi vs. Gray code**

M. Gardner proved (1957,59) that the “Tower of Hanoi” problem is isomorphic to finding a Hamiltonian path on an $n$-hypercube.

$n$-hypercube has $2^n$ vertices

a path visiting them all has $2^n - 1$ edges

⇒ at least $2^n - 1$ moves

Ex: $n = 4$

\[
\begin{array}{cccc}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & 1 & 1 \\
0 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 \\
0 & 1 & 1 & 1 \\
0 & 1 & 1 & 0 \\
0 & 1 & 0 & 0
\end{array}
\]

start: all 4 disks on $A$

goal: all 4 disks on $B$

$n$ is even → move 0 ccw
Another puzzle: “Spin-out”

- very similar to “Chinese Rings puzzle”
- 7 spinners take one of two possible orientations: ↑, ←
- initially all are ↑; the goal is that all are ←
- problem: only the rightmost spinner can rotate and any spinner which is the left neighbor of the rightmost ↑
Another puzzle: “Spin-out”

- **Solution**: \( n \) spinners from left to right \( \equiv g_{n-1} \ldots g_0 \)
- Spinner \( i \) is \( \uparrow \equiv g_i = 1 \); spinner \( i \) is \( \leftarrow \equiv g_i = 0 \)
- Traverse the Gray code from \( 1 \ldots 1 \) to \( 0 \ldots 0 \) in reverse direction

**Ex:**

\[ \begin{array}{cccc}
\text{(0000)} & \text{(0001)} & \text{(0011)} & \text{(0110)} \\
\text{(0101)} & \text{(1110)} & \text{(1101)} & \text{(1000)} \\
\end{array} \]

\[ \begin{array}{c}
\uparrow \uparrow \uparrow \uparrow \\
\leftarrow \leftarrow \leftarrow \leftarrow \\
\leftarrow \leftarrow \leftarrow \leftarrow \\
\leftarrow \leftarrow \leftarrow \leftarrow \\
\text{done} \\
\end{array} \]
Generate next Gray code number

\[ G[i] \rightarrow G[i + 1] \]

- count the number \( \#1 \) of “1”s in \( G[i] \)
- if \( \#1 \) is even then “flip” \( g_0 \) of \( G[i] \) to get \( G[i + 1] \)
- else “flip” the left neighbor (bit position) of the rightmost “1”

\[
\text{func nextGray}(g): \quad \ldots g = g[n - 1] \ldots g[0] \\
\quad \text{if } (\#1(g) \text{ is even}) \text{ then flip } g[0]; \\
\quad \text{else} \\
\quad \quad r \leftarrow \text{index of rightmost “1” of } g \\
\quad \quad \text{if } (r < n - 1) \text{ then flip } g[r+1]; \text{ else flip } g[r]; \text{ fi} \\
\quad \text{fi} \\
\quad \text{return } (g) \quad \ldots \text{next Gray number } (G[0] \text{ if input was } G[2^n - 1])
\]