1 Show bijections for B_6 and B_7

Let B_k for $k \geq 2$ consist of all 0-1 strings of length k with both ends equal to 1 (there are 2^{k-2} of them). Show explicitly two bijections: between 16 strings in B_6 and their NAF representations, and 32 strings in B_7 and their NAF representations.

1.1 Solution

We show the bijections in the tables below. We also report the number of zeros in the original encoding and the number of zeros in the NAF encoding. The last column shows the ratio of the NAF zeros to the original zeros; for example, if there were 2 zeros in the original encoding and 3 zeros in the NAF encoding, then the ratio would be $3/2 = 1.50$.

Ratios r in the range $1 < r < 2$ are highlighted in yellow, and ratios $r \geq 2$ are highlighted in green. Higher ratio is better since more zeros require fewer `add` operations in the double-and-add algorithm. We see many green highlights, which means that the NAF encoding is a good, efficient encoding for the double-and-add computation.

index	original encoding	NAF encoding	original 0s	NAF 0s	NAF 0s / original 0s
0	1 0 0 0 0 1	1 0 0 0 0 1	4	4	1.00
1	1 0 0 0 1 1	1 0 0 1 0 -1	3	3	1.00
2	1 0 0 1 0 1	1 0 0 1 0 1	3	3	1.00
3	1 0 0 1 1 1	1 0 1 0 0 -1	2	3	1.50
4	1 0 1 0 0 1	1 0 1 0 0 1	3	3	1.00
5	1 0 1 0 1 1	1 0 -1 0 -1 0 -1	2	3	1.50
6	1 0 1 1 0 1	1 0 -1 0 -1 0 1	2	3	1.50
7	1 0 1 1 1 1	1 0 -1 0 0 0 -1	1	4	4.00
8	1 1 0 0 0 1	1 0 -1 0 0 0 1	3	4	1.33
9	1 1 0 0 1 1	1 0 -1 0 1 0 -1	2	3	1.50
10	1 1 0 1 0 1	1 0 -1 0 1 0 1	2	3	1.50
11	1 1 0 1 1 1	1 0 0 -1 0 0 -1	1	4	4.00
12	1 1 1 0 0 1	1 0 0 -1 0 0 1	2	4	2.00
13	1 1 1 0 1 1	1 0 0 0 -1 0 -1	1	4	4.00
14	1 1 1 1 0 1	1 0 0 0 -1 0 1	1	4	4.00
15	1 1 1 1 1 1	1 0 0 0 0 0 -1	0	5	inf

Table 1. The 16 strings in B_6.

index	original encoding	NAF encoding	original 0s	NAF 0s	NAF 0s / original 0s
0	1 0 0 0 0 0 1	1 0 0 0 0 0 1	5	5	1.00
1	1 0 0 0 0 1 1	1 0 0 0 1 0 -1	4	4	1.00
2	1 0 0 0 1 0 1	1 0 0 0 1 0 1	4	4	1.00
3	1 0 0 0 1 1 1	1 0 0 1 0 0 -1	3	4	1.33
4	1 0 0 1 0 0 1	1 0 0 1 0 0 1	4	4	1.00
5	1 0 0 1 0 1 1	1 0 1 0 -1 0 -1	3	3	1.00
6	1 0 0 1 1 0 1	1 0 1 0 -1 0 1	3	3	1.00
7	1 0 0 1 1 1 1	1 0 1 0 0 0 -1	2	4	2.00
8	1 0 1 0 0 0 1	1 0 1 0 0 0 1	4	4	1.00
9	1 0 1 0 0 1 1	1 0 1 0 1 0 -1	3	3	1.00
10	1 0 1 0 1 0 1	1 0 1 0 1 0 1	3	3	1.00
11	1 0 1 0 1 1 1	1 0 -1 0 -1 0 0 -1	2	4	2.00
12	1 0 1 1 0 0 1	1 0 -1 0 -1 0 0 1	3	4	1.33
13	1 0 1 1 0 1 1	1 0 -1 0 0 -1 0 -1	2	4	2.00
14	1 0 1 1 1 0 1	1 0 -1 0 0 -1 0 1	2	4	2.00
15	1 0 1 1 1 1 1	1 0 -1 0 0 0 0 -1	1	5	5.00
16	1 1 0 0 0 0 1	1 0 -1 0 0 0 0 1	4	5	1.25
17	1 1 0 0 0 1 1	1 0 -1 0 0 1 0 -1	3	4	1.33
18	1 1 0 0 1 0 1	1 0 -1 0 0 1 0 1	3	4	1.33
19	1 1 0 0 1 1 1	1 0 -1 0 1 0 0 -1	2	4	2.00
20	1 1 0 1 0 0 1	1 0 -1 0 1 0 0 1	3	4	1.33
21	1 1 0 1 0 1 1	1 0 0 -1 0 -1 0 -1	2	4	2.00
22	1 1 0 1 1 0 1	1 0 0 -1 0 -1 0 1	2	4	2.00
23	1 1 0 1 1 1 1	1 0 0 -1 0 0 0 -1	1	5	5.00
24	1 1 1 0 0 0 1	1 0 0 -1 0 0 0 1	3	5	1.67
25	1 1 1 0 0 1 1	1 0 0 -1 0 1 0 -1	2	4	2.00
26	1 1 1 0 1 0 1	1 0 0 -1 0 1 0 1	2	4	2.00
27	1 1 1 0 1 1 1	1 0 0 0 -1 0 0 -1	1	5	5.00
28	1 1 1 1 0 0 1	1 0 0 0 -1 0 0 1	2	5	2.50
29	1 1 1 1 0 1 1	1 0 0 0 0 -1 0 -1	1	5	5.00
30	1 1 1 1 1 0 1	1 0 0 0 0 -1 0 1	1	5	5.00
31	1 1 1 1 1 1 1	1 0 0 0 0 0 0 -1	0	6	inf

Table 2. The 32 strings in B_7.

2 Problem 2

Solve exercise 6.17 page 279 (ECIES). In (a) show the intermediate values of variables. This exercise is not in edition 4 of the textbook. Edition 4 does not include the ECIES scheme, but a similar to it cryptosystem 7.2 EC ElGamal. For this problem use the ECIES slide discussed in class (Figure 1 below).

Cryptosystem 6.2: *Simplified ECIES*

Let E be an elliptic curve defined over $\mathbb{Z}_p$ ($p > 3$ prime) such that E contains a cyclic subgroup $H = \langle P \rangle$ of prime order n in which the Discrete Logarithm problem is infeasible.
Let $\mathcal{P} = \mathbb{Z}_p{}^*$, $\mathcal{C} = (\mathbb{Z}_p \times \mathbb{Z}_2) \times \mathbb{Z}_p{}^*$, and define

$$\mathcal{K} = \{(E, P, m, Q, n) : Q = mP\}.$$

The values P, Q and n are the public key, and $m \in \mathbb{Z}_n{}^*$ is the private key.
For $K = (E, P, m, Q, n)$, for a (secret) random number $k \in \mathbb{Z}_n{}^*$, and for $x \in \mathbb{Z}_p{}^*$, define

$$e_K(x, k) = (\text{POINTCOMPRESS}(kP), x x_0 \bmod p),$$

where $kQ = (x_0, y_0)$ and $x_0 \neq 0$.
For a ciphertext $y = (y_1, y_2)$, where $y_1 \in \mathbb{Z}_p \times \mathbb{Z}_2$ and $y_2 \in \mathbb{Z}_p{}^*$, define

$$d_K(y) = y_2 (x_0)^{-1} \bmod p,$$

where

$$(x_0, y_0) = m\,\text{POINTDECOMPRESS}(y_1).$$

Figure 1. Elliptic curve integrated encryption scheme (ECIES) as discussed in-class.

Let E be the elliptic curve $y^2 = x^3 + 2x + 7$ defined over $\mathbb{Z}_{31}$. It can be shown that $\#E = 39$ and $P = (2, 9)$ is an element of order 39 in E. The simplified ECIES defined on E has $\mathbb{Z}_{31}^*$ as its plaintext space. Suppose the private key is $m = 8$.

(a) Compute $Q = mP$.

(b) Decrypt the ciphertext `((18,1),21), ((3,1),18), ((17,0),19), ((28,0),8)`.

(c) Assuming that each plaintext represents one alphabetic character, convert the plaintext into an English word. Use the correspondence $A = 1, \ldots, Z = 26$ because 0 is not allowed in a plaintext ordered pair.

2.1 Solution

Part (a) Calculations[1] shown below; we find that $Q = 8P = (8, 15)$.

operation	exponent bit value	point coordinates
initialize	1 (MSB)	(2, 9)
double	0	(10, 2)
double	0	(15, 8)
double	0 (LSB)	(8, 15)

Table 3. Detailed computation of $Q = 8P$ for the given EC.

Part (b) Recall that k is not needed for decryption. The steps are:

(a) Perform point decompression on the tuple $y_1 = (x', y')$ (e.g. the first entry in the ciphertext has $y_1 = (18, 1) = (x', y')$).

(b) Multiply the decompressed tuple by the secret key m (where $m = 8$ in this exercise); this gives the tuple (x_0, y_0).

(c) Compute x_0^{-1} from the last step (we don't need y_0).

(d) Compute $(y_2 \cdot x_0^{-1}) \bmod p$; this is the answer.

mP after decompression	x_0^{-1}	decrypted text d
(15, 8)	29	20
(2, 9)	16	9
(30, 29)	30	12
(14, 19)	20	5

Table 4. Computation details.

Part (c) This text spells TILE.

[1]Checked using `http://christelbach.com/ECCalculator.aspx`

3 Compute $87P$ with the NAF representation of 87

Solve exercise 6.18 page 279 (7.19 page 307).

(a) Determine the NAF representation of the integer 87.

(b) Using the NAF representation of 87, use Algorithm 6.5 to compute $87P$, where $P = (2, 6)$ is a point on the elliptic curve $y^2 = x^3 + x + 26$ defined over $\mathbb{Z}_{127}$. Show the partial results during each iteration of the algorithm.

3.1 Solution

Part (a)

```
87 in binary       :    1  0  1  0  1  1  1
NAF representation : 1  0 -1  0 -1  0  0 -1
```

Part (b) The final answer is $Q = 87P = (102, 88)$. Computation[2] steps are shown below.

operation	index i	NAF value	coordinates of Q
initialize	0 (MSB)	1	(2,6)
double	1	0	(118, 80)
double	2	-1	(82, 13)
also subtract	2	-1	(68, 57)
double	3	0	(85, 119)
double	4	-1	(99, 115)
also subtract	4	-1	(87, 116)
double	5	0	(91, 18)
double	6	0	(102, 39)
double	7 (LSB)	-1	(54, 119)
also subtract	7 (LSB)	-1	(102, 88)

Table 5. Computation of $87P$ for the given EC.

[2]Checked using `http://christelbach.com/ECCalculator.aspx`

4 Prove that the NAF representation is unique

(Optional) Prove that the NAF representation is unique. You need to show that two distinct NAF strings cannot encode the same integer.

Proof. The NAF encoding uses the equality (for $i > j$) that

$$2^i + 2^{i-1} + \ldots + 2^j = 2^{i+1} - 2^j. \tag{1}$$

We show that two NAF strings encode the same integer if and only if the NAF strings are identical.

Forward direction. If two NAF strings encode the same integer, then the NAF strings are identical. This is easy to see by Equation (1) since the NAF conversion algorithm transforms blocks of the form $01\ldots1$ to the form $10\ldots-1$ where the $\ldots$ are a continuous (and possibly empty) block of 1s. An integer has only one binary representation, so NAF conversion will produce the same output string.

Backward direction. If two NAF strings are identical, then the NAF strings encode the same integer. We can think of an NAF string as a set of orthogonal basis vectors $\{0, 2, 4, 8, 16, \ldots\}$ with possible coefficients $\{-1, 0, 1\}$. Since the NAF representation requires coefficients of 0 between the coefficients of -1 and 1, when we sum the basis vector components of the NAF representation, it is impossible to encode two different integers with the same NAF string. ☐

5 Source code

Listing 1. `hw06.sage`

```
# Advanced Crypto HW06 // Hannah Miller // 2020-04-07

import numpy as np
import computeEC as ec  # custom module for this homework

# ================================================================
# == Problem 1 ==

def prettyprint(a):
    '''Given array a, return a nicely formatted string for pretty
    printing, the count of zeros in a, and the count of ones in a.

    '''
    fmtstr = ''.join(['{:3d}'.format(x) for x in a])
    zeros  = sum(a == 0)
    ones   = sum(a == 1)
    return fmtstr,zeros,ones

def s2naf(a):
    '''Given an array a of 0s and 1s, convert the string into its NAF
    (non-adjacent form) and return the NAF.

    '''
    # Walk through the string right-to-left
```

```python
    i = len(a) - 1  # initialize index into the array
    while i > 0:
        j = i  # update

        while a[i] == 1:  # get a block of 1s
            i = i - 1  # decrement to "walk up" the MSBs

        if j-i > 1:  # do not update the string '...010...'
            a[i] = 1  # looks wrong but we scan right-to-left so this is correct
            a[(i+1):j] = 0
            a[j] = -1

        while a[i] == 0:  # skip over the next block of 0s
            i = i - 1  # decrement to "walk up" the MSBs

    return a  # a is now in NAF


def run_all_B(k_input):
    k = k_input - 2
    print('index  &  original encoding  &  NAF encoding  &  a zeros  &  NAF zeros
    &  NAF zeros / a zeros  \\\\')
    for num in range(2**k):
        b = bin(num)[2:].zfill(k)  # padded binary value b
        s = '01{}1'.format(b)  # string s with dummy 0 at the beginning for later
        a = np.array([int(x) for x in s])  # convert to an array a of integers
        a_str,a_zeros,a_ones = prettyprint(a[1:])  # chop off dummy 0

        naf = s2naf(a)  # NAF of the string s
        if naf[0] == 0: naf = naf[1:]  # chop off dummy 0 if needed
        naf_str,naf_zeros,naf_ones = prettyprint(naf)

        # Print the results
        print('{:2d}  &  {}  &  {} &  {}  &  {}  &  {:.2f}  \\\\'.format(
            num, a_str, naf_str, a_zeros, naf_zeros, naf_zeros/a_zeros))


#run_all_B(6)
#run_all_B(7)


# ================================================================
# == Problem 2 ==

p = 31
a =  2
b =  7
E = EllipticCurve(GF(p), [a,b])
```

```python
#print(E.cardinality())

# for pt in E.points():
#     if pt.order() == 39:
#         print(pt, pt.order())


# -- Part (a) --

def compute8x(P):
    m = 8  # private key
    m_bin = [1,0,0,0]  # m in binary

    Q = P  # initialize
    i = 0
    #print('initialize',m_bin[i],Q)
    for i in range(1,len(m_bin)):
        Q = ec.compute_x3y3(p,Q,Q,a)  # double; Q = 2Q
        #print('double',m_bin[i],Q)

        if m_bin[i] == 1:
            Q = ec.compute_x3y3(p,P,Q,a)  # add; Q = Q+P
            #print('add',m_bin[i],Q)

    #print('final answer',Q)
    return(Q)

P = (2,9)
compute8x(P)


# -- Part (b) --
def pointdecompress(y1):
    x0,y0 = y1  # extract from tuple
    p = 31
    z = (x0**3 + 2*x0 + 7) % p
    #print('z =',z)

    # Since (p+1)/4 = 8 is an integer, then sqrt(z) = z^8 mod p
    y = (z**8) % p
    #print('y =',y)

    if y == y0 % 2:
        return (x0,y)
    else:
        return (x0,p-y)

ciphertext = [((18,1),21), ((3,1),18), ((17,0),19), ((28,0),8)]
```

```
print('$mP$ after decompression  &  $x_0^{-1}$  & decrypted text $d$  \\\\')
for c in ciphertext:
    y1,y2 = c  # y1 is the tuple (x0,y0)
    P = compute8x(pointdecompress(y1))
    x0,y0 = P
    x0inv = ec.compute_inverse(p,x0)
    d = (y2*x0inv) % p  # decrypted text
    print('{}  &  {}  &  {}  \\\\'.format(P,x0inv,d))



# ================================================================
# == Problem 3 ==

# -- Part (a) --

num = 87  # the input integer
b = bin(num)[2:]  # padded binary value b
s = '0{}'.format(b)  # string s with dummy 0 for later
a = np.array([int(x) for x in s])  # convert to an array a of integers
a_str,a_zeros,a_ones = prettyprint(a)

naf = s2naf(a)  # NAF of the string s
if naf[0] == 0: naf = naf[1:]  # chop off dummy 0 if needed
naf_str,naf_zeros,naf_ones = prettyprint(naf)

# print(a_str)
# print(naf_str)


# -- Part (b) --

p = 127
a = 1

P      = (2, 6)
minusP = (2,-6)

#print('operation  &  index $i$  &  NAF value  &  coordinates of $Q$  \\\\')

Q = P  # initialize
for i in range(1,len(naf)):  # scan MSB to LSB
    Q = ec.compute_x3y3(p,Q,Q,a)  #Q <- 2Q
    #print('double  &  {}  &  {}  &  {}  \\\\'.format(i,naf[i],Q))

    if naf[i] == 1:
        Q = ec.compute_x3y3(p,Q,P,a)  # Q <- Q + P
```

```
        #print('also add & {} & {} & {} \\\\'.format(i,naf[i],Q))
    if naf[i] == -1:
        Q = ec.compute_x3y3(p,Q,minusP,a)  # Q <- Q - P
        #print('also subtract & {} & {} & {} \\\\'.format(i,naf[i],Q))

#print(Q)
```

Listing 2. `computeEC.py`

```
# These definitions are from page 258 of Stinson

import math

def compute_inverse(p,t):
    # Compute inverse of t mod  p by brute-force.  The elliptic curves
    # on the homework are small enough that this is OK.
    inv = 0  # initialize
    for i in range(p):
        x = (t*i) % p
        if x==1:
            inv = i
    return inv

def compute_slope(p,P,Q,a):
    # Step 1: Compute the slope of the line L.
    if P == Q:  # for point doubling when P=Q
        (x1,y1) = P
        inv = compute_inverse(p,2*y1)  # this is (2*y1)**(-1)
        #print(inv)
        return ( (3*x1**2 + a) * inv ) % p
    else:
        (x1,y1) = P
        (x2,y2) = Q
        if x2-x1 == 0:
            print('no inverse exists!')
            return(math.nan)
        else:
            inv = compute_inverse(p,x2-x1)
            return ( (y2-y1) * inv ) % p

def compute_x3(p,slope,P,Q):
    # Step 2: Compute x3.  The slope is used.
    (x1,y1) = P
    (x2,y2) = Q
    return (slope**2 - x1 - x2) % p

def compute_y3(p,slope,P,x3):
```

```python
    # Step 3: Compute y3.  This uses both the slope and x3.
    (x1,y1) = P
    return (slope*(x1-x3) - y1) % p

def compute_x3y3(p,P,Q,a):
    # Compute (x3,y3) = (x1,y1) + (x2,y2).
    #print(P,Q)
    (x1,y1) = P
    (x2,y2) = Q

    slope = compute_slope(p,P,Q,a)
    x3 = compute_x3(p,slope,P,Q)
    y3 = compute_y3(p,slope,P,x3)
    #print('slope, {:02d}, x3, {:02d}, y3, {:02d}'.format(slope,x3,y3))
    #return (expand(x3),expand(y3))
    return (x3,y3)

################################################################
# 2020-03-24 -- test point doubling

p = 11
a =  1

#P = (2,7)
P = (8,3)
R = P

# print(1,P)

# for i in range(12):
#     R = compute_x3y3(p,P,R,a)
#     (x,y) = R
#     print('i, {:2d}, P, ({:2d}, {:2d})'.format(i+2,x,y))
```