

Binom Bijection

Storing k -Subsets Efficiently

Volker Grabsch

September 7, 2009

Draft

This presentation is still a draft!

Overview

Motivation

Constructive Proof

Implementation

About this document

Overview

Motivation

Constructive Proof

Implementation

About this document

Storing Numbers

Store 100 numbers (a_0, \dots, a_{99}) of range $\{0, 1, 2\}$ efficiently!

- Store each number in 2 bits
 $\Rightarrow 2 \cdot 100 = \mathbf{200 \text{ bits}}$
- Store one big number (100 digits, base 3)

$$f(a_0, \dots, a_{99}) = \sum_{i=0}^{99} 3^i a_i \in \{0, \dots, 3^{100} - 1\}$$

$$\Rightarrow \lceil \log_2(3^{100}) \rceil = \mathbf{159 \text{ bits}}$$

- 3^{100} states to encode \Rightarrow impossible to do better

Storing Numbers

Store 100 numbers (a_0, \dots, a_{99}) of range $\{0, 1, 2\}$ efficiently!

- Store each number in 2 bits
 $\Rightarrow 2 \cdot 100 = \mathbf{200 \text{ bits}}$
- Store one big number (100 digits, base 3)

$$f(a_0, \dots, a_{99}) = \sum_{i=0}^{99} 3^i a_i \quad \in \{0, \dots, 3^{100} - 1\}$$

$$\Rightarrow \lceil \log_2(3^{100}) \rceil = \mathbf{159 \text{ bits}}$$

- 3^{100} states to encode \Rightarrow impossible to do better

Storing Numbers

Store 100 numbers (a_0, \dots, a_{99}) of range $\{0, 1, 2\}$ efficiently!

- Store each number in 2 bits
 $\Rightarrow 2 \cdot 100 = \mathbf{200 \text{ bits}}$
- Store one big number (100 digits, base 3)

$$f(a_0, \dots, a_{99}) = \sum_{i=0}^{99} 3^i a_i \quad \in \{0, \dots, 3^{100} - 1\}$$

$$\Rightarrow \lceil \log_2(3^{100}) \rceil = \mathbf{159 \text{ bits}}$$

- 3^{100} states to encode \Rightarrow impossible to do better

Storing Permutations

Store a permutation of 100 numbers $(0, \dots, 99)$ efficiently!

- Store each number in 7 bits
 $\Rightarrow 7 \cdot 100 = \mathbf{700 \text{ bits}}$
- Store one big number (100 digits, base 100)
 $\Rightarrow \lceil \log_2(100^{100}) \rceil = \mathbf{665 \text{ bits}}$
- Store one big multibase number
(99 digits, bases $100, 99, \dots, 2$)
 $\Rightarrow \lceil \log_2(100!) \rceil = \mathbf{525 \text{ bits}}$
- $100!$ states to encode \Rightarrow impossible to do better

Storing Permutations

Store a permutation of 100 numbers $(0, \dots, 99)$ efficiently!

- Store each number in 7 bits
 $\Rightarrow 7 \cdot 100 = \mathbf{700 \text{ bits}}$
- Store one big number (100 digits, base 100)
 $\Rightarrow \lceil \log_2(100^{100}) \rceil = \mathbf{665 \text{ bits}}$
- Store one big multibase number
(99 digits, bases $100, 99, \dots, 2$)
 $\Rightarrow \lceil \log_2(100!) \rceil = \mathbf{525 \text{ bits}}$
- 100! states to encode \Rightarrow impossible to do better

Storing Permutations

Store a permutation of 100 numbers $(0, \dots, 99)$ efficiently!

- Store each number in 7 bits
 $\Rightarrow 7 \cdot 100 = \mathbf{700 \text{ bits}}$
- Store one big number (100 digits, base 100)
 $\Rightarrow \lceil \log_2(100^{100}) \rceil = \mathbf{665 \text{ bits}}$
- Store one big multibase number
(99 digits, bases $100, 99, \dots, 2$)
 $\Rightarrow \lceil \log_2(100!) \rceil = \mathbf{525 \text{ bits}}$
- 100! states to encode \Rightarrow impossible to do better

Storing Permutations

Store a permutation of 100 numbers $(0, \dots, 99)$ efficiently!

- Store each number in 7 bits
 $\Rightarrow 7 \cdot 100 = \mathbf{700 \text{ bits}}$
- Store one big number (100 digits, base 100)
 $\Rightarrow \lceil \log_2(100^{100}) \rceil = \mathbf{665 \text{ bits}}$
- Store one big multibase number
(99 digits, bases $100, 99, \dots, 2$)
 $\Rightarrow \lceil \log_2(100!) \rceil = \mathbf{525 \text{ bits}}$
- $100!$ states to encode \Rightarrow impossible to do better

Storing k -Subsets

Store a choice of 30 out of 100 numbers $\{0, \dots, 99\}$ efficiently!

- Store each number in 7 bits
 $\Rightarrow 7 \cdot 30 = \mathbf{210 \text{ bits}}$
- Store one big number (30 digits, base 100)
 $\Rightarrow \lceil \log_2(100^{30}) \rceil = \mathbf{200 \text{ bits}}$
- Store one big number (30 digits, base 71)
 $\Rightarrow \lceil \log_2(71^{30}) \rceil = \mathbf{185 \text{ bits}}$
- Store in classic subset representation
 $\Rightarrow \mathbf{100 \text{ bits}}$ (of which 30 are set to 1)
- Theoretical minimum:
 $\lceil \log_2 \binom{100}{30} \rceil = \mathbf{85 \text{ bits}}$
- Need a bijection

$$\varphi : \binom{\{0, \dots, 99\}}{30} \xrightarrow{\sim} \left\{ 0, \dots, \binom{100}{30} - 1 \right\}$$

Storing k -Subsets

Store a choice of 30 out of 100 numbers $\{0, \dots, 99\}$ efficiently!

- Store each number in 7 bits
 $\Rightarrow 7 \cdot 30 = \mathbf{210 \text{ bits}}$
- Store one big number (30 digits, base 100)
 $\Rightarrow \lceil \log_2(100^{30}) \rceil = \mathbf{200 \text{ bits}}$
- Store one big number (30 digits, base 71)
 $\Rightarrow \lceil \log_2(71^{30}) \rceil = \mathbf{185 \text{ bits}}$
- Store in classic subset representation
 $\Rightarrow \mathbf{100 \text{ bits}}$ (of which 30 are set to 1)
- Theoretical minimum:
 $\lceil \log_2 \binom{100}{30} \rceil = \mathbf{85 \text{ bits}}$
- Need a bijection

$$\varphi : \binom{\{0, \dots, 99\}}{30} \xrightarrow{\sim} \left\{ 0, \dots, \binom{100}{30} - 1 \right\}$$

Storing k -Subsets

Store a choice of 30 out of 100 numbers $\{0, \dots, 99\}$ efficiently!

- Store each number in 7 bits
 $\Rightarrow 7 \cdot 30 = \mathbf{210 \text{ bits}}$
- Store one big number (30 digits, base 100)
 $\Rightarrow \lceil \log_2(100^{30}) \rceil = \mathbf{200 \text{ bits}}$
- Store one big number (30 digits, base 71)
 $\Rightarrow \lceil \log_2(71^{30}) \rceil = \mathbf{185 \text{ bits}}$
- Store in classic subset representation
 $\Rightarrow \mathbf{100 \text{ bits}}$ (of which 30 are set to 1)
- Theoretical minimum:
 $\lceil \log_2 \binom{100}{30} \rceil = \mathbf{85 \text{ bits}}$
- Need a bijection

$$\varphi : \binom{\{0, \dots, 99\}}{30} \xrightarrow{\sim} \left\{ 0, \dots, \binom{100}{30} - 1 \right\}$$

Storing k -Subsets

Store a choice of 30 out of 100 numbers $\{0, \dots, 99\}$ efficiently!

- Store each number in 7 bits
 $\Rightarrow 7 \cdot 30 = \mathbf{210 \text{ bits}}$
- Store one big number (30 digits, base 100)
 $\Rightarrow \lceil \log_2(100^{30}) \rceil = \mathbf{200 \text{ bits}}$
- Store one big number (30 digits, base 71)
 $\Rightarrow \lceil \log_2(71^{30}) \rceil = \mathbf{185 \text{ bits}}$
- Store in classic subset representation
 $\Rightarrow \mathbf{100 \text{ bits}}$ (of which 30 are set to 1)
- Theoretical minimum:
 $\lceil \log_2 \binom{100}{30} \rceil = \mathbf{85 \text{ bits}}$
- Need a bijection

$$\varphi : \binom{\{0, \dots, 99\}}{30} \xrightarrow{\sim} \left\{ 0, \dots, \binom{100}{30} - 1 \right\}$$

Storing k -Subsets

Store a choice of 30 out of 100 numbers $\{0, \dots, 99\}$ efficiently!

- Store each number in 7 bits
 $\Rightarrow 7 \cdot 30 = \mathbf{210 \text{ bits}}$
- Store one big number (30 digits, base 100)
 $\Rightarrow \lceil \log_2(100^{30}) \rceil = \mathbf{200 \text{ bits}}$
- Store one big number (30 digits, base 71)
 $\Rightarrow \lceil \log_2(71^{30}) \rceil = \mathbf{185 \text{ bits}}$
- Store in classic subset representation
 $\Rightarrow \mathbf{100 \text{ bits}}$ (of which 30 are set to 1)
- Theoretical minimum:
 $\lceil \log_2 \binom{100}{30} \rceil = \mathbf{85 \text{ bits}}$
- Need a bijection

$$\varphi : \binom{\{0, \dots, 99\}}{30} \xrightarrow{\sim} \left\{ 0, \dots, \binom{100}{30} - 1 \right\}$$

Storing k -Subsets

Store a choice of 30 out of 100 numbers $\{0, \dots, 99\}$ efficiently!

- Store each number in 7 bits
 $\Rightarrow 7 \cdot 30 = \mathbf{210 \text{ bits}}$
- Store one big number (30 digits, base 100)
 $\Rightarrow \lceil \log_2 (100^{30}) \rceil = \mathbf{200 \text{ bits}}$
- Store one big number (30 digits, base 71)
 $\Rightarrow \lceil \log_2 (71^{30}) \rceil = \mathbf{185 \text{ bits}}$
- Store in classic subset representation
 $\Rightarrow \mathbf{100 \text{ bits}}$ (of which 30 are set to 1)

- Theoretical minimum:
 $\lceil \log_2 \binom{100}{30} \rceil = \mathbf{85 \text{ bits}}$
- Need a bijection

$$\varphi : \binom{\{0, \dots, 99\}}{30} \xrightarrow{\sim} \left\{ 0, \dots, \binom{100}{30} - 1 \right\}$$

Binom Bijection

- Question:

$$\varphi : \binom{\{0, \dots, n-1\}}{k} \xrightarrow{\sim} \left\{ 0, \dots, \binom{n}{k} - 1 \right\} \quad (1)$$

- Solution:

$$\varphi(\{s_0 < \dots < s_{k-1}\}) = \sum_{i=0}^{k-1} \binom{s_i}{i+1} \quad (2)$$

- Need

• *Combinatorial Proof of Bijectivity*

• *Combinatorial Proof of Inverse*

Binom Bijection

- Question:

$$\varphi : \binom{\{0, \dots, n-1\}}{k} \xrightarrow{\sim} \left\{ 0, \dots, \binom{n}{k} - 1 \right\} \quad (1)$$

- Solution:

$$\varphi(\{s_0 < \dots < s_{k-1}\}) = \sum_{i=0}^{k-1} \binom{s_i}{i+1} \quad (2)$$

- Need

▶ constructive proof of bijectivity

▶ combinatorial proof

Binom Bijection

- Question:

$$\varphi : \binom{\{0, \dots, n-1\}}{k} \xrightarrow{\sim} \left\{ 0, \dots, \binom{n}{k} - 1 \right\} \quad (1)$$

- Solution:

$$\varphi (\{s_0 < \dots < s_{k-1}\}) = \sum_{i=0}^{k-1} \binom{s_i}{i+1} \quad (2)$$

- Need
 - constructive proof of bijectivity
 - implementation of φ
 - implementation of φ^{-1}

Binom Bijection

- Question:

$$\varphi : \binom{\{0, \dots, n-1\}}{k} \xrightarrow{\sim} \left\{ 0, \dots, \binom{n}{k} - 1 \right\} \quad (1)$$

- Solution:

$$\varphi (\{s_0 < \dots < s_{k-1}\}) = \sum_{i=0}^{k-1} \binom{s_i}{i+1} \quad (2)$$

- Need
 - constructive proof of bijectivity
 - implementation of φ
 - implementation of φ^{-1}

Binom Bijection

- Question:

$$\varphi : \binom{\{0, \dots, n-1\}}{k} \xrightarrow{\sim} \left\{ 0, \dots, \binom{n}{k} - 1 \right\} \quad (1)$$

- Solution:

$$\varphi(\{s_0 < \dots < s_{k-1}\}) = \sum_{i=0}^{k-1} \binom{s_i}{i+1} \quad (2)$$

- Need
 - constructive proof of bijectivity
 - implementation of φ
 - implementation of φ^{-1}

Binom Bijection

- Question:

$$\varphi : \binom{\{0, \dots, n-1\}}{k} \xrightarrow{\sim} \left\{ 0, \dots, \binom{n}{k} - 1 \right\} \quad (1)$$

- Solution:

$$\varphi (\{s_0 < \dots < s_{k-1}\}) = \sum_{i=0}^{k-1} \binom{s_i}{i+1} \quad (2)$$

- Need
 - constructive proof of bijectivity
 - implementation of φ
 - implementation of φ^{-1}

Overview

Motivation

Constructive Proof

Implementation

About this document

Proof

- Note: φ is independent of n
- ...

$$\begin{aligned}\binom{n}{k} &= \binom{k-1}{k-1} + \dots + \binom{n-1}{k-1} \\ &= \sum_{i=k-1}^{n-1} \binom{i}{k-1}\end{aligned}$$

Proof

- Note: φ is independent of n
- ...

$$\begin{aligned}\binom{n}{k} &= \binom{k-1}{k-1} + \dots + \binom{n-1}{k-1} \\ &= \sum_{i=k-1}^{n-1} \binom{i}{k-1}\end{aligned}$$

Overview

Motivation

Constructive Proof

Implementation

About this document

Implementation of φ

- Direct implementation of φ

$$\varphi(\{s_0 < \dots < s_{k-1}\}) = \sum_{i=0}^{k-1} \binom{s_i}{i+1}$$

```
def number_of_set(s):  
    return sum(binom(si, i+1)  
               for i, si in enumerate(sorted(s)))
```

- Calculate binomial coefficients $\binom{n}{k}$ in Python

```
import scipy  
  
def binom(n, k):  
    return scipy.comb(n, k, exact=1)
```

Implementation of φ

- Direct implementation of φ

$$\varphi(\{s_0 < \dots < s_{k-1}\}) = \sum_{i=0}^{k-1} \binom{s_i}{i+1}$$

```
def number_of_set(s):  
    return sum(binom(si, i+1)  
               for i, si in enumerate(sorted(s)))
```

- Calculate binomial coefficients $\binom{n}{k}$ in Python

```
import scipy  
  
def binom(n, k):  
    return scipy.comb(n, k, exact=1)
```

Naive implementation of φ^{-1}

- Naive and slow implementation of φ^{-1}

```
def set_of_number_SLOW(k, num):  
    s = set()  
    for i in xrange(k-1, -1, -1):  
        si = i  
        while num >= binom(si + 1, si - i):  
            si += 1  
        num -= binom(si, si - i - 1)  
        s.add(si)  
    return s
```

Implementation of φ^{-1}

- Use previous binomial coefficient to calculate next one

$$\binom{n}{k} = \frac{n}{k} \cdot \binom{n-1}{k-1}$$

- Fast implementation of φ^{-1}

```
def set_of_number(k, num):  
    s = set()  
    for i in xrange(k-1, -1, -1):  
        si = i  
        b = 1  
        while num >= b:  
            si += 1  
            b = (b * (si+1)) / (si-i)  
        num -= (b * (si-i)) / (si+1)  
        s.add(si)  
    return s
```

Implementation of φ^{-1}

- Use previous binomial coefficient to calculate next one

$$\binom{n}{k} = \frac{n}{k} \cdot \binom{n-1}{k-1}$$

- Fast implementation of φ^{-1}

```
def set_of_number(k, num):  
    s = set()  
    for i in xrange(k-1, -1, -1):  
        si = i  
        b = 1  
        while num >= b:  
            si += 1  
            b = (b * (si+1)) / (si-i)  
        num -= (b * (si-i)) / (si+1)  
        s.add(si)  
    return s
```

Complexity

- `number_of_set`
 - ...
- `set_of_number`

Complexity

- `number_of_set`
 - ...
- `set_of_number`
 - ...

Complexity

- `number_of_set`
 - ...
- `set_of_number`
 - ...

Complexity

- `number_of_set`
 - ...
- `set_of_number`
 - ...

Tests

- ...

```
from Binom_Bijection import number_of_set, set_of_number
```

- ...

```
assert number_of_set([]) == 0
assert set_of_number(0, 0) == set([])
```

- ...

```
for s0 in xrange(0, 400):
    assert number_of_set([s0]) == s0
    assert set_of_number(1, s0) == set([s0])
```

Tests

- ...

```
from Binom_Bijection import number_of_set, set_of_number
```

- ...

```
assert number_of_set([]) == 0
assert set_of_number(0, 0) == set([])
```

- ...

```
for s0 in xrange(0, 400):
    assert number_of_set([s0]) == s0
    assert set_of_number(1, s0) == set([s0])
```

Tests

- ...

```
from Binom_Bijection import number_of_set, set_of_number
```

- ...

```
assert number_of_set([]) == 0
assert set_of_number(0, 0) == set([])
```

- ...

```
for s0 in xrange(0, 400):
    assert number_of_set([s0]) == s0
    assert set_of_number(1, s0) == set([s0])
```

Tests

- ...

```
counter = 0
for s0 in xrange(1, 50):
    for s1 in xrange(0, s0):
        s = set([s1, s0])
        assert number_of_set(s) == counter
        assert set_of_number(2, counter) == s
        counter += 1
```

- ...

```
counter = 0
for s0 in xrange(2, 25):
    for s1 in xrange(1, s0):
        for s2 in xrange(0, s1):
            s = set([s2, s1, s0])
            assert number_of_set(s) == counter
            assert set_of_number(3, counter) == s
            counter += 1
```

Tests

- ...

```
counter = 0
for s0 in xrange(1, 50):
    for s1 in xrange(0, s0):
        s = set([s1, s0])
        assert number_of_set(s) == counter
        assert set_of_number(2, counter) == s
        counter += 1
```

- ...

```
counter = 0
for s0 in xrange(2, 25):
    for s1 in xrange(1, s0):
        for s2 in xrange(0, s1):
            s = set([s2, s1, s0])
            assert number_of_set(s) == counter
            assert set_of_number(3, counter) == s
            counter += 1
```

Tests

- ...

```
def test_k(k, max):
    def test_k_rec(s, i, max, counter):
        if i == 0:
            assert number_of_set(s) == counter[0]
            assert set_of_number(k, counter[0]) == s
            counter[0] += 1
        else:
            for si in xrange(i-1, max):
                test_k_rec(s.union([si]), i-1, si,
                           counter)
    test_k_rec(set([]), k, max, [0])
```

- ...

```
test_k(4, 18)
test_k(5, 14)
test_k(6, 13)
test_k(7, 13)
test_k(14, 17)
```

Tests

- ...

```
def test_k(k, max):
    def test_k_rec(s, i, max, counter):
        if i == 0:
            assert number_of_set(s) == counter[0]
            assert set_of_number(k, counter[0]) == s
            counter[0] += 1
        else:
            for si in xrange(i-1, max):
                test_k_rec(s.union([si]), i-1, si,
                           counter)
    test_k_rec(set([]), k, max, [0])
```

- ...

```
test_k(4, 18)
test_k(5, 14)
test_k(6, 13)
test_k(7, 13)
test_k(14, 17)
```

Tests

- ...

```
def test_consistency(s):  
    assert set_of_number(len(s), number_of_set(s)) == s
```

- ...

```
test_consistency(set(xrange(0, 400)))  
test_consistency(set(xrange(0, 400, 2)))  
test_consistency(set(xrange(0, 400, 3)))  
test_consistency(set(xrange(0, 400, 10)))  
test_consistency(set(xrange(40, 400)))  
test_consistency(set(xrange(150, 400)))  
test_consistency(set(xrange(150, 400, 2)))  
test_consistency(set(xrange(150, 400, 3)))  
test_consistency(set(xrange(150, 400, 10)))  
test_consistency(set(xrange(200, 400)))  
test_consistency(set(xrange(360, 400)))
```

Tests

- ...

```
def test_consistency(s):  
    assert set_of_number(len(s), number_of_set(s)) == s
```

- ...

```
test_consistency(set(xrange(0, 400)))  
test_consistency(set(xrange(0, 400, 2)))  
test_consistency(set(xrange(0, 400, 3)))  
test_consistency(set(xrange(0, 400, 10)))  
test_consistency(set(xrange(40, 400)))  
test_consistency(set(xrange(150, 400)))  
test_consistency(set(xrange(150, 400, 2)))  
test_consistency(set(xrange(150, 400, 3)))  
test_consistency(set(xrange(150, 400, 10)))  
test_consistency(set(xrange(200, 400)))  
test_consistency(set(xrange(360, 400)))
```

Overview

Motivation

Constructive Proof

Implementation

About this document

About this document

- Website

`http://www.profv.de/Binom_Bijection/`

- LaTeX source is also shell script

```
sh Binom_Bijection.tex
```

(Self-contained literate programming)

About this document

- Website

`http://www.profv.de/Binom_Bijection/`

- LaTeX source is also shell script

```
sh Binom_Bijection.tex
```

(Self-contained literate programming)

Build instructions (I)

- Shell options to handle errors correctly

```
set -euvx
```

- Extract Python library

```
sed '/^[\\%]/,/% Binom_Bijection.py$/ s,^,#, ' \  
< Binom_Bijection.tex > Binom_Bijection.py
```

- Extract and run tests

```
sed '/^[\\%]/,/% Tests.py$/ s,^,#, ' \  
< Binom_Bijection.tex > Tests.py
```

```
python Tests.py
```

Build instructions (I)

- Shell options to handle errors correctly

```
set -euvx
```

- Extract Python library

```
sed '/^[\\%]/,/% Binom_Bijection.py$/ s,^,#, ' \  
< Binom_Bijection.tex > Binom_Bijection.py
```

- Extract and run tests

```
sed '/^[\\%]/,/% Tests.py$/ s,^,#, ' \  
< Binom_Bijection.tex > Tests.py
```

```
python Tests.py
```

Build instructions (I)

- Shell options to handle errors correctly

```
set -euvx
```

- Extract Python library

```
sed '/^[\\%]/,/% Binom_Bijection.py$/ s,^,#, ' \  
< Binom_Bijection.tex > Binom_Bijection.py
```

- Extract and run tests

```
sed '/^[\\%]/,/% Tests.py$/ s,^,#, ' \  
< Binom_Bijection.tex > Tests.py
```

```
python Tests.py
```

Build instructions (II)

- Compile LaTeX source to PDF

```
rm -f      Binom_Bijection.{aux,log,nav,out,snm,toc,vrb}  
pdflatex  Binom_Bijection.tex  
pdflatex  Binom_Bijection.tex  
rm -f      Binom_Bijection.{aux,log,nav,out,snm,toc,vrb}
```

- Create ZIP archive

```
rm -rf     Binom_Bijection  
mkdir     Binom_Bijection  
cp        Binom_Bijection.{tex,pdf,py} Binom_Bijection/  
rm -f     Binom_Bijection.zip  
zip -r9   Binom_Bijection.zip Binom_Bijection/  
rm -rf     Binom_Bijection
```

- Remove temporary files

```
rm -f     Binom_Bijection.{py,pyc} Tests.py
```

Build instructions (II)

- Compile LaTeX source to PDF

```
rm -f      Binom_Bijection.{aux,log,nav,out,snm,toc,vrb}  
pdflatex  Binom_Bijection.tex  
pdflatex  Binom_Bijection.tex  
rm -f      Binom_Bijection.{aux,log,nav,out,snm,toc,vrb}
```

- Create ZIP archive

```
rm -rf     Binom_Bijection  
mkdir     Binom_Bijection  
cp        Binom_Bijection.{tex,pdf,py} Binom_Bijection/  
rm -f     Binom_Bijection.zip  
zip -r9   Binom_Bijection.zip Binom_Bijection/  
rm -rf     Binom_Bijection
```

- Remove temporary files

```
rm -f     Binom_Bijection.{py,pyc} Tests.py
```

Build instructions (II)

- Compile LaTeX source to PDF

```
rm -f      Binom_Bijection.{aux,log,nav,out,snm,toc,vrb}  
pdflatex Binom_Bijection.tex  
pdflatex Binom_Bijection.tex  
rm -f      Binom_Bijection.{aux,log,nav,out,snm,toc,vrb}
```

- Create ZIP archive

```
rm -rf Binom_Bijection  
mkdir Binom_Bijection  
cp Binom_Bijection.{tex,pdf,py} Binom_Bijection/  
rm -f Binom_Bijection.zip  
zip -r9 Binom_Bijection.zip Binom_Bijection/  
rm -rf Binom_Bijection
```

- Remove temporary files

```
rm -f Binom_Bijection.{py,pyc} Tests.py
```