Binomial Coefficients

Russell Impagliazzo and Miles Jones

Thanks to Janine Tiefenbruck

http://cseweb.ucsd.edu/classes/sp16/cse21-bd/

May 6, 2016

Fixed-density Binary Strings

Rosen p. 413

How many length n **binary strings contain** k **ones**?

Density is number of ones

For example, n=6 k=4

Which of these strings matches this example?

A. 101101
B. 1100011101
C. 111011
D. 1101
E. None of the above.

Fixed-density Binary Strings

Rosen p. 413

How many length n **binary strings contain** k **ones**?

Density is number of ones

For example, n=6 k=4

Product rule: How many options for the first bit? the second? the third?

$\underbrace{0\,0\,0}_{n-k}\ \underbrace{1\,1\,1\,1}_{k}$

$$\frac{n!}{(n-k)!\,k!} = \binom{n}{k}$$

Fixed-density Binary Strings

Rosen p. 413

How many length n **binary strings contain** k **ones**?

Density is number of ones

For example, n=6 k=4

Tree diagram: *gets very big & is hard to generalize*

Fixed-density Binary Strings

Rosen p. 413

How many length n **binary strings contain** k **ones**?

Density is number of ones

For example, n=6 k=4

Another approach: **use a different representation** i.e. count with categories

Objects:
Categories:
Size of each category:

categories = (# objects) / (size of each category)

Fixed-density Binary Strings

Rosen p. 413

How many length n **binary strings contain** k **ones**?

For example, n=6 k=4

Another approach: **use a different representation** i.e. count with categories

Objects: all strings made up of 0_1, 0_2, 1_1, 1_2, 1_3, 1_4
Categories: strings that agree except subscripts
Size of each category:

Subscripts so objects are distinct

categories = (# objects) / (size of each category)

Fixed-density Binary Strings

Rosen p. 413

How many length n **binary strings contain** k **ones**?

For example, n=6 k=4

Another approach: **use a different representation** i.e. count with categories

Objects: all strings made up of $0_1, 0_2, 1_1, 1_2, 1_3, 1_4$ **6!**
Categories: strings that agree except subscripts
Size of each category: **?**

categories = (# objects) / (size of each category)

Fixed-density Binary Strings

How many subscripted strings i.e. rearrangements of the symbols

$$0_1, 0_2, 1_1, 1_2, 1_3, 1_4$$

result in

$$101101$$

when the subscripts are removed?

A. 6!
B. 4!
C. 2!
D. 4!2!
E. None of the above

Fixed-density Binary Strings

Rosen p. 413

How many length n **binary strings contain** k **ones**?

For example, n=6 k=4

Another approach: **use a different representation** i.e. count with categories

Objects: all strings made up of $0_1, 0_2, 1_1, 1_2, 1_3, 1_4$ **6!**
Categories: strings that agree except subscripts
Size of each category: **4!2!**

categories = (# objects) / (size of each category)
= 6! / (4!2!)

Fixed-density Binary Strings

Rosen p. 413

How many length n **binary strings contain** k **ones**?

Another approach: **use a different representation** i.e. count with categories

Objects: all strings made up of $0_1, 0_2, \ldots, 0_{n-k}, 1_1, 1_2, \ldots, 1_k$ **n!**
Categories: strings that agree except subscripts
Size of each category: **k!(n-k)!**

 # categories = (# objects) / (size of each category)
= n!/ (k! (n-k) !)

Terminology

Universe of n objects

Rosen p. 407-413

A **permutation** of r elements from a set of n *distinct* objects is an **ordered** arrangement of them. There are

List of r distinct objects

$\frac{n!}{(n-r)!} =$ **P(n,r) = n(n-1) (n-2) …(n-r+1)**

many of these.

n n-1 n-2 ... (n-r+1)

A **combination** of r elements from a set of n *distinct* objects is an **unordered** selection of them. There are

C(n,r) = n!/ (r! (n-r) !)

(1 2 ... r)

(2 1 ... r)

r!

many of these.

Binomial coefficient
"n choose r"

$\binom{n}{r}$

Fixed-density Binary Strings

Rosen p. 413

How many length n **binary strings contain** k **ones**?

How to express this using the new terminology?

A. C(n,k)
B. C(n,n-k)
C. P(n,k)
D. P(n,n-k)
E. None of the above

$$\binom{n}{k} = \frac{n!}{k!\,(n-k)!}$$

$$\binom{n}{n-k} = \binom{n}{k}$$

Fixed-density Binary Strings

Rosen p. 413

How many length n **binary strings contain** k **ones**?

How to express this using the new terminology?

A. C(n,k) {1,2,3..n} is set of positions in string, choose k positions for 1s
B. C(n,n-k) {1,2,3..n} is set of positions in string, choose n-k positions for 0s
C. P(n,k)
D. P(n,n-k)
E. None of the above

Ice cream! redux

An ice cream parlor has n different flavors available.

How many ice cream cones are there, if we count two cones as the same if they have the same two flavors (even if they're in opposite order)?

$= P(n,2)$

Objects: cones $\quad$ **n(n-1)**
Categories: flavor pairs (regardless of order)
Size of each category: **2**

categories = (n)(n-1)/ 2

$\binom{n}{2}$

Order doesn't matter so selecting a subset of size 2 of the n possible flavors:

$C(n,2) = n!/ (2! (n-2)!) = n(n-1)/2$

What's in a name?

Rosen p. 415

Binomial: sum of two terms, say x and y.

What do powers of binomials look like?

$$
\begin{aligned}
(x+y)^4 &= (x+y)(x+y)(x+y)(x+y) \\
&= (x^2+2xy+y^2)(x^2+2xy+y^2) \\
&= x^4+4x^3y+6x^2y^2+4xy^3+y^4
\end{aligned}
$$

In general , for $(x+y)^n$

A. All terms in the expansion are (some coefficient times) $x^k y^{n-k}$ for some k, 0<=k<=n.
B. All coefficients in the expansion are integers between 1 and n.
C. There is symmetry in the coefficients in the expansion.
D. The coefficients of x^n and y^n are both 1.
E. All of the above.

Binomial Theorem

Rosen p. 416

XYXYY

$$(x+y)^n = \overbrace{(x+y)(x+y)\ldots(x+y)}^{n}$$

$$= x^n + \_\_\,x^{n-1}y + \_\_\,x^{n-2}y^2 + \ldots + \_\_\,x^k y^{n-k} + \ldots + \_\_\,x^2y^{n-2} + \_\_\,xy^{n-1} + y^n$$

Number of ways we can choose k of the n factors (to contribute to x) and hence also n-k of the factors (to contribute to y)

$$\binom{n}{k}$$

Binomial Theorem

Rosen p. 416

$(x+y)^n \quad = \quad (x+y)(x+y)\ldots(x+y)$

$= \quad x^n + \_\_ x^{n-1}y + \_\_ x^{n-2}y^2 + \ldots + \_\_ x^ky^{n-k} + \ldots + \_\_ x^2y^{n-2} + \_\_ xy^{n-1} + y^n$

Number of ways we can choose k of the n factors (to contribute to x) and hence also n-k of the factors (to contribute to y) C(n,k)

$= x^n + C(n,1)\, x^{n-1}y + \ldots + C(n,k)\, x^ky^{n-k} + \ldots + C(n,k-1)\, xy^{n-1} + y^n$

$\binom{n}{0} \quad \binom{n}{1} \quad \binom{n}{0} = \frac{n!}{0!\,n!} = 1$

Binomial Coefficient Identities

What's an **identity** ? An equation that is always true.

To prove

LHS = RHS

- Use algebraic manipulations of formulas

OR

- Interpret each side as counting some collection of strings, and then prove a statements about those sets of strings

Symmetry Identity

Rosen p. 411

Theorem: $\binom{n}{k} = \binom{n}{n-k}$

Symmetry Identity

Rosen p. 411

Theorem: $\binom{n}{k} = \binom{n}{n-k}$

Proof 1: Use formula $\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n!}{(n-k)!k!} = \binom{n}{n-k}$

Symmetry Identity

Rosen p. 411

Theorem: $$\binom{n}{k} = \binom{n}{n-k}$$

Proof 1: Use formula $$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n!}{(n-k)!k!} = \binom{n}{n-k}$$

Proof 2: Combinatorial interpretation?

LHS counts number of binary strings of length n with k ones
RHS counts number of binary strings of length n with n-k ones

Symmetry Identity

Rosen p. 411

Theorem: $$\binom{n}{k} = \binom{n}{n-k}$$

Proof 1: Use formula $$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n!}{(n-k)!k!} = \binom{n}{n-k}$$

Proof 2: Combinatorial interpretation?

LHS counts number of binary strings of length n with k ones and n-k zeros
RHS counts number of binary strings of length n with n-k ones and k zeros

Symmetry Identity

Rosen p. 411

Theorem: $$\binom{n}{k} = \binom{n}{n-k}$$

Proof 1: Use formula $$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n!}{(n-k)!k!} = \binom{n}{n-k}$$

Proof 2: Combinatorial interpretation?

LHS counts number of binary strings of length n with k ones and n-k zeros
RHS counts number of binary strings of length n with n-k ones and k zeros

Can match up these two sets by pairing each string with another where 0s, 1s are flipped. This ***bijection*** *means the two sets have the same size. So* **LHS** = **RHS**.

Pascal's Identity

if we do

if we don't

Rosen p. 418

Theorem: $$\binom{n+1}{k} = \binom{n}{k-1} + \binom{n}{k}$$

1 n n+1

Proof 1: Use formula

We either pick n+1 or not

Proof 2: Combinatorial interpretation?

If we do, we pick k-1 from n

LHS counts number of binary strings ???

RHS counts number of binary strings ???

If not, we pick k of first n

Pascal's Identity

Rosen p. 418

Theorem: $$\binom{n+1}{k} = \binom{n}{k-1} + \binom{n}{k}$$

Proof 2: Combinatorial interpretation?

LHS counts number of binary strings of length n+1 that have k ones.
RHS counts number of binary strings ???

Length n+1 binary strings with k ones

Pascal's Identity

Rosen p. 418

Theorem:
$$\binom{n+1}{k} = \binom{n}{k-1} + \binom{n}{k}$$

Proof 2: Combinatorial interpretation?

LHS counts number of binary strings of length n+1 that have k ones.
RHS counts number of binary strings ???

Start with 1 Start with 0

Pascal's Identity

Rosen p. 418

How many length n+1 strings start with 1 and have k ones in total?

A. C(n+1, k+1)
B. C(n, k)
C. C(n, k+1)
D. C(n, k-1)
E. None of the above.

Start with 1 Start with 0

Pascal's Identity

Rosen p. 418

How many length n+1 strings start with 0 and have k ones in total?

A. C(n+1, k+1)
B. C(n, k)
C. C(n, k+1)
D. C(n, k-1)
E. None of the above.

Start with 1 Start with 0

Pascal's Identity

Rosen p. 418

Theorem: $\binom{n+1}{k} = \binom{n}{k-1} + \binom{n}{k}$

1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1

Proof 2: Combinatorial interpretation?

LHS counts number of binary strings of length n+1 that have k ones.
RHS counts number of binary strings of length n+1 that have k ones, split into two.

Start with 1 Start with 0

Sum Identity

Rosen p. 417

Theorem: $\sum_{k=0}^{n} \binom{n}{k} = 2^n$

$= (1+1)^n = \sum \binom{n}{k} 1^{k} 1^{n}$

What set does the **LHS** count?

A. Binary strings of length n that have k ones.
B. Binary strings of length n that start with 1.
C. Binary strings of length n that have any number of ones.
D. None of the above.

Sum Identity

Rosen p. 417

Theorem: $$\sum_{k=0}^{n} \binom{n}{k} = 2^n$$

Proof : Combinatorial interpretation?

LHS counts number of binary strings of length n that have any number of 1s.

By sum rule, we can break up the set of binary strings of length n into disjoint sets based on how many 1s they have, then add their sizes.

RHS counts number of binary strings of length n.

This is the same set so **LHS** = **RHS**.

Review: Terminology

Rosen p. 407-413

A **permutation** of r elements from a set of n *distinct* objects is an **ordered** arrangement of them. There are

$$P(n,r) = n(n-1)\,(n-2)\,\ldots(n-r+1)$$

many of these.

A **combination** of r elements from a set of n distinct objects is an **unordered** slection of them. There are

$$C(n,r) = n!/\,(\,r!\,(n-r)\,!\,)$$

many of these.

Binomial coefficient
"n choose r" $\binom{n}{r}$

Fixed-density Binary Strings

Rosen p. 413

How many length n **binary strings contain** k **ones**?

Density is number of ones

Objects: all strings made up of 0_1, 0_2, 1_1, 1_2, 1_3, 1_4 **n!**
Categories: strings that agree except subscripts
Size of each category: **k!(n-k)!**

 # categories = (# objects) / (size of each category)
= n!/ (k! (n-k) !) = C(n,k) = $\binom{n}{k}$

Encoding Fixed-density Binary Strings

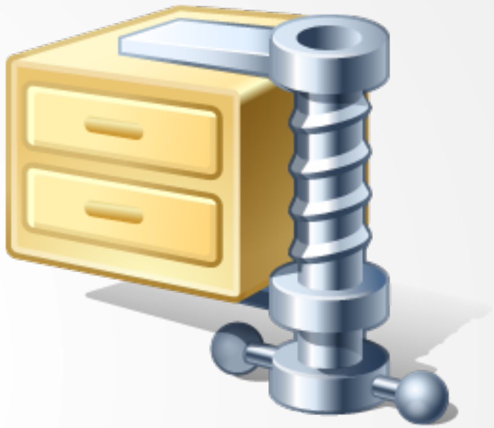
Rosen p. 413

What's the **smallest** number of **bits** that we need to specify a binary string if we know it **has k ones and n-k zeros**?

A. n
B. k
C. $\log_2(C(n,k))$
D. ??

Data Compression

Store / transmit information in as little space as possible



Data Compression: Video

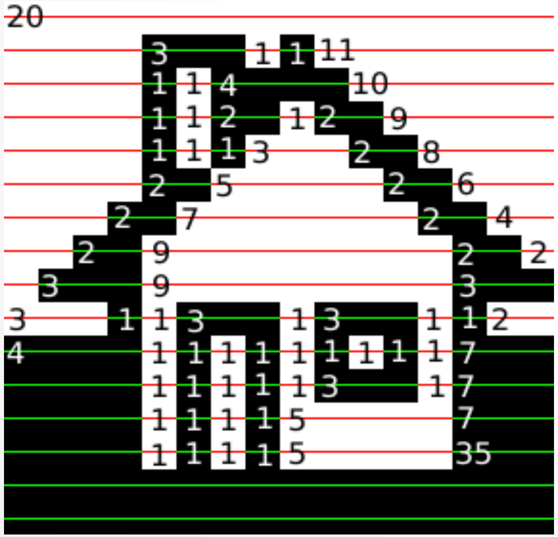
Video: stored as sequence of still frames.

Idea: instead of storing each frame fully, record change from previous frame.

Data Compression: Run-Length Encoding

Image: described as grid of pixels, each with RED, GREEN, BLUE values.

Idea: instead of storing RGB value of each pixel, store **run-length** of run of same color.

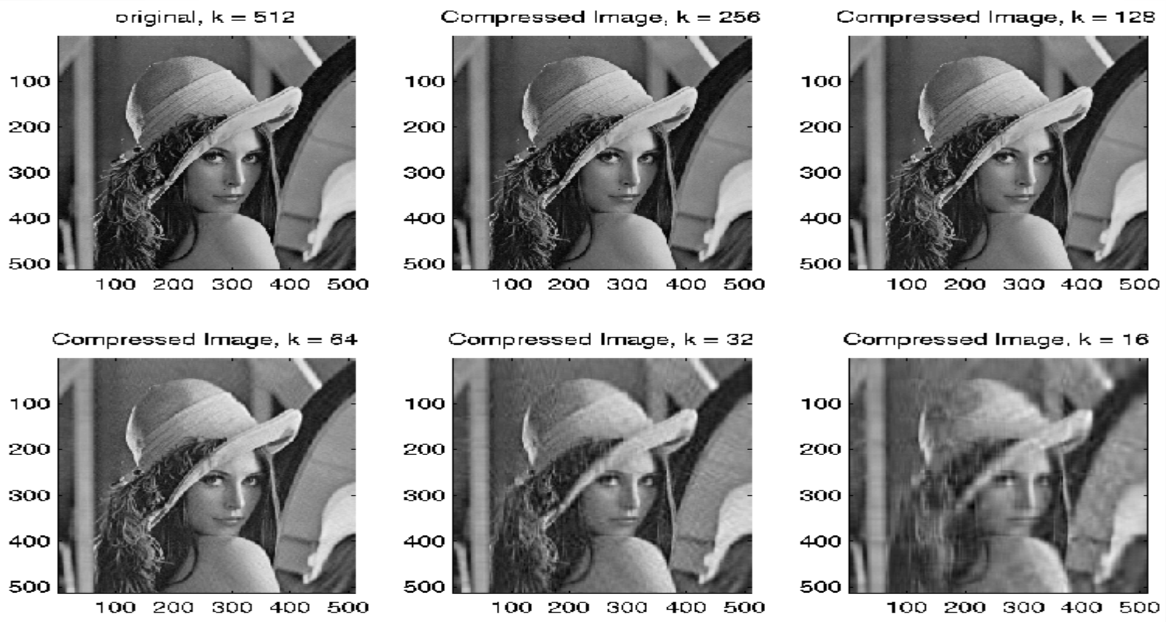


When is this a good coding mechanism?
Will there be any loss in this compression?

Lossy Compression: Singular Value Decomposition

Image: described as grid of pixels, each with RED, GREEN, BLUE values.

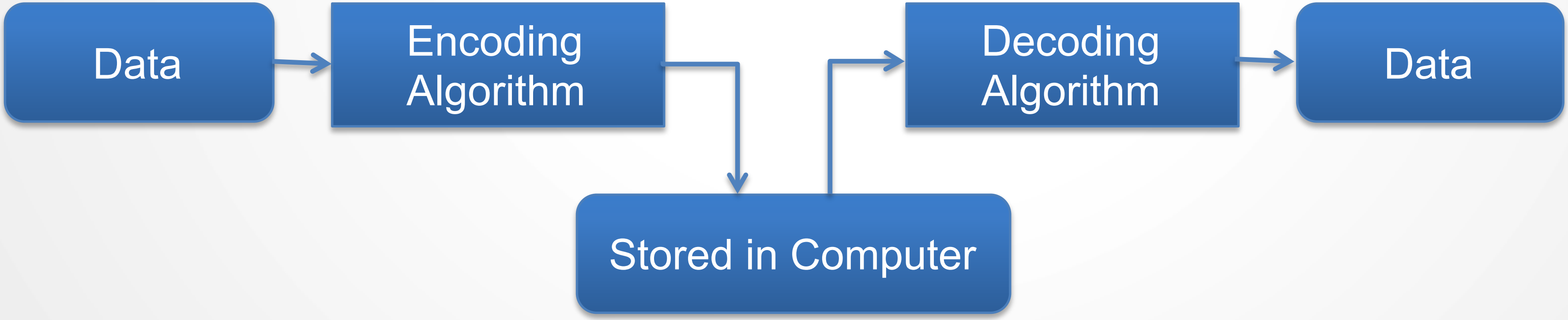
Idea: use Linear Algebra to compress data to a fraction of its size, with minimal loss.



Data Compression: Trade-off

Complicated compression scheme

… save storage space
… may take a long time to encode / decode



Encoding: Binary Palindromes

Palindrome: string that reads the same forward and backward.

Which of these are binary palindromes?

A. The empty string.
B. 0101.
C. 0110.
D. 101.
E. All but one of the above.

Encoding: Binary Palindromes

Palindrome: string that reads the same forward and backward.

How many length n binary palindromes are there?

A. 2^n
B. n
C. $n/2$
D. $\log_2 n$
E. None of the above

Encoding: Binary Palindromes

Palindrome: string that reads the same forward and backward.

How many bits are (optimally) required to encode length n binary palindromes?

A. n
B. n-1
C. n/2
D. $\log_2 n$
E. None of the above.

Is there an algorithm that achieves this?

Encoding: Fixed Density Strings

Goal: encode a length n binary string that we know has k ones (and n-k zeros).

How would you represent such a string with n-1 ***bits?***

Encoding: Fixed Density Strings

Goal: encode a length n binary string that we know has k ones with k<<n.

How would you represent such a string with n-1 ***bits?***

Can we do better?

Encoding: Fixed Density Strings

Goal: encode a length n binary string that we know has k ones (and n-k zeros).

How would you represent such a string with n-1 ***bits?***

Can we do better?

Idea: give positions of 1s in the string within some smaller window.

- Fix window size.
- If there is a 1 in the current "window" in the string, record its position and move the window over.
- Otherwise, record a 0 and move the window over.

Encoding: Fixed Density Strings

Idea: give positions of 1s in the string within some smaller window.

- Fix window size.
- If there is a 1 in the current "window" in the string, record its position and move the window over.
- Otherwise, record a 0 and move the window over.

Example $n=12$, $k=3$, window size $n/k = 4$.

How do we encode s = 011000000010 ?

Encoding: Fixed Density Strings

Idea: give positions of 1s in the string within some smaller window.

- Fix window size.
- If there is a 1 in the current "window" in the string, record its position and move the window over.
- Otherwise, record a 0 and move the window over.

Example n=12, k=3, window size n/k = 4.

How do we encode s = 011000000010 ?   Output:

There's a 1! What's its position?

Encoding: Fixed Density Strings

Idea: give positions of 1s in the string within some smaller window.

- Fix window size.
- If there is a 1 in the current "window" in the string, record its position and move the window over.
- Otherwise, record a 0 and move the window over.

Example n=12, k=3, window size n/k = 4.

How do we encode s = 011000000010 ? Output: 01

There's a 1! What's its position?

Encoding: Fixed Density Strings

Idea: give positions of 1s in the string within some smaller window.

- Fix window size.
- If there is a 1 in the current "window" in the string, record its position and move the window over.
- Otherwise, record a 0 and move the window over.

Example n=12, k=3, window size n/k = 4.

How do we encode s = 01<u>1000</u>000010 ? Output: 01

There's a 1! What's its position?

Encoding: Fixed Density Strings

Idea: give positions of 1s in the string within some smaller window.

- Fix window size.
- If there is a 1 in the current "window" in the string, record its position and move the window over.
- Otherwise, record a 0 and move the window over.

Example n=12, k=3, window size n/k = 4.

How do we encode s = 011000000010 ?

Output: 0100

There's a 1! What's its position?

Encoding: Fixed Density Strings

Idea: give positions of 1s in the string within some smaller window.

- Fix window size.
- If there is a 1 in the current "window" in the string, record its position and move the window over.
- Otherwise, record a 0 and move the window over.

Example n=12, k=3, window size n/k = 4.

How do we encode s = 011<u>0000</u>00010 ? Output: 0100

No 1s in this window.

Encoding: Fixed Density Strings

Idea: give positions of 1s in the string within some smaller window.

- Fix window size.
- If there is a 1 in the current "window" in the string, record its position and move the window over.
- Otherwise, record a 0 and move the window over.

Example n=12, k=3, window size n/k = 4.

How do we encode s = 011<u>0000</u>00010 ? Output: 01000

No 1s in this window.

Encoding: Fixed Density Strings

Idea: give positions of 1s in the string within some smaller window.
- Fix window size.
- If there is a 1 in the current "window" in the string, record its position and move the window over.
- Otherwise, record a 0 and move the window over.

Example n=12, k=3, window size n/k = 4.

How do we encode s = 011000000010 ? Output: 01000

There's a 1! What's its position?

Encoding: Fixed Density Strings

Idea: give positions of 1s in the string within some smaller window.

- Fix window size.
- If there is a 1 in the current "window" in the string, record its position and move the window over.
- Otherwise, record a 0 and move the window over.

Example n=12, k=3, window size n/k = 4.

How do we encode s = 011000000010 ?   Output: 0100011

There's a 1! What's its position?

Encoding: Fixed Density Strings

Idea: give positions of 1s in the string within some smaller window.

- Fix window size.
- If there is a 1 in the current "window" in the string, record its position and move the window over.
- Otherwise, record a 0 and move the window over.

Example n=12, k=3, window size n/k = 4.

How do we encode s = 01100000001<u>0</u> ?

Output: 0100011

No 1s in this window.

Encoding: Fixed Density Strings

Idea: give positions of 1s in the string within some smaller window.

- Fix window size.
- If there is a 1 in the current "window" in the string, record its position and move the window over.
- Otherwise, record a 0 and move the window over.

Example n=12, k=3, window size n/k = 4.

How do we encode s = 01100000001<u>0</u> ?

Output: 01000110.

No 1s in this window.

Encoding: Fixed Density Strings

Idea: give positions of 1s in the string within some smaller window.

- Fix window size.
- If there is a 1 in the current "window" in the string, record its position and move the window over.
- Otherwise, record a 0 and move the window over.

Example n=12, k=3, window size n/k = 4.

How do we encode s = 01100000001<u>0</u> ? Output: 01000110.

Compressed to 8 bits!

But can we recover the original string? Decoding …

Encoding: Fixed Density Strings

With n=12, k=3, window size n/k = 4. **Output**: 01000110

Can be parsed as the (intended) input: s = 011000000010 ?
But also:

01: one in position 1
0: no ones
00: one in position 0
11: one in position 3
0: no ones

s' = 010000100010

Problem: two different inputs with same output. Can't uniquely decode.

Compression Algorithm

A **valid compression algorithm** must:

- Have outputs of shorter (or same) length as input.
- Be uniquely decodable.

Encoding: Fixed Density Strings

Can we modify this algorithm to get unique decodability?

Idea: use marker bit to indicate when to interpret output as a position.

- Fix window size.
- If there is a 1 in the current "window" in the string,
  record a 1 to interpret next bits as position,
  then record its position and move the window over.
- Otherwise, record a 0 and move the window over.

Encoding: Fixed Density Strings

Idea: use marker bit to indicate when to interpret output as a position.

- Fix window size.
- If there is a 1 in the current "window" in the string,
  record a 1 to interpret next bits as position,
  then record its position and move the window over.
- Otherwise, record a 0 and move the window over.

Example n=12, k=3, window size n/k = 4.

How do we encode s = 011000000010 ? Output:

Encoding: Fixed Density Strings

Idea: use marker bit to indicate when to interpret output as a position.

- Fix window size.
- If there is a 1 in the current "window" in the string, record a 1 to interpret next bits as position, then record its position and move the window over.
- Otherwise, record a 0 and move the window over.

Example n=12, k=3, window size n/k = 4.

How do we encode s = <u>0110</u>00000010 ? Output:

Encoding: Fixed Density Strings

Idea: use marker bit to indicate when to interpret output as a position.

- Fix window size.
- If there is a 1 in the current "window" in the string, record a 1 to interpret next bits as position, then record its position and move the window over.
- Otherwise, record a 0 and move the window over.

Example n=12, k=3, window size n/k = 4.

How do we encode s = <u>0110</u>00000010 ? Output:

What output corresponds to these first few bits?

A. 0

B. 1

C. 01

D. 101

E. None of the above.

Encoding: Fixed Density Strings

Idea: use marker bit to indicate when to interpret output as a position.

- Fix window size.
- If there is a 1 in the current "window" in the string,
  record a 1 to interpret next bits as position,
  then record its position and move the window over.
- Otherwise, record a 0 and move the window over.

Example n=12, k=3, window size n/k = 4.

How do we encode s = 011000000010 ? Output: 101

Interpret next bits as position of 1; this position is 01

Encoding: Fixed Density Strings

Idea: use marker bit to indicate when to interpret output as a position.

- Fix window size.
- If there is a 1 in the current "window" in the string, record a 1 to interpret next bits as position, then record its position and move the window over.
- Otherwise, record a 0 and move the window over.

Example n=12, k=3, window size n/k = 4.

How do we encode s = 011000000010 ?   Output: 101

Encoding: Fixed Density Strings

Idea: use marker bit to indicate when to interpret output as a position.

- Fix window size.
- If there is a 1 in the current "window" in the string,
  record a 1 to interpret next bits as position,
  then record its position and move the window over.
- Otherwise, record a 0 and move the window over.

Example n=12, k=3, window size n/k = 4.

How do we encode s = 011000000010 ? Output: 101100

Interpret next bits as position of 1; this position is 00

Encoding: Fixed Density Strings

Idea: use marker bit to indicate when to interpret output as a position.

- Fix window size.
- If there is a 1 in the current "window" in the string,
  record a 1 to interpret next bits as position,
  then record its position and move the window over.
- Otherwise, record a 0 and move the window over.

Example n=12, k=3, window size n/k = 4.

How do we encode s = 011<u>0000</u>00010 ?   Output: 101100

Encoding: Fixed Density Strings

Idea: use marker bit to indicate when to interpret output as a position.

- Fix window size.
- If there is a 1 in the current "window" in the string,
  record a 1 to interpret next bits as position,
  then record its position and move the window over.
- Otherwise, record a 0 and move the window over.

Example n=12, k=3, window size n/k = 4.

How do we encode s = 011<u>0000</u>00010 ?

Output: 1011000

No 1s in this window.

Encoding: Fixed Density Strings

Idea: use marker bit to indicate when to interpret output as a position.

- Fix window size.
- If there is a 1 in the current "window" in the string,
  record a 1 to interpret next bits as position,
  then record its position and move the window over.
- Otherwise, record a 0 and move the window over.

Example n=12, k=3, window size n/k = 4.

How do we encode s = 01100000<u>0010</u> ? Output: 1011000

Encoding: Fixed Density Strings

Idea: use marker bit to indicate when to interpret output as a position.

- Fix window size.
- If there is a 1 in the current "window" in the string,
  record a 1 to interpret next bits as position,
  then record its position and move the window over.
- Otherwise, record a 0 and move the window over.

Example n=12, k=3, window size n/k = 4.

How do we encode s = 011000000010 ? Output: 1011000111

Interpret next bits as position of 1; this position is 11

Encoding: Fixed Density Strings

Idea: use marker bit to indicate when to interpret output as a position.

- Fix window size.
- If there is a 1 in the current "window" in the string,
  record a 1 to interpret next bits as position,
  then record its position and move the window over.
- Otherwise, record a 0 and move the window over.

Example n=12, k=3, window size n/k = 4.

How do we encode s = 011000000010 ? Output: 1011000111

Encoding: Fixed Density Strings

Idea: use marker bit to indicate when to interpret output as a position.

- Fix window size.
- If there is a 1 in the current "window" in the string,
  record a 1 to interpret next bits as position,
  then record its position and move the window over.
- Otherwise, record a 0 and move the window over.

Example n=12, k=3, window size n/k = 4.

How do we encode s = 01100000001<u>0</u> ?

Output: 10110001110

No 1s in this window.

Encoding: Fixed Density Strings

Idea: use marker bit to indicate when to interpret output as a position.

- Fix window size.
- If there is a 1 in the current "window" in the string, record a 1 to interpret next bits as position, then record its position and move the window over.
- Otherwise, record a 0 and move the window over.

Example n=12, k=3, window size n/k = 4.

How do we encode s = 01100000001<u>0</u> ?

Output: 10110001110

Compare to previous output: 01000110

Output uses more bits than last time. Any redundancies?

Encoding: Fixed Density Strings

Idea: use marker bit to indicate when to interpret output as a position.

- Fix window size.
- If there is a 1 in the current "window" in the string, record a 1 to interpret next bits as position, then record its position and move the window over.
- Otherwise, record a 0 and move the window over.

Example n=12, k=3, window size n/k = 4.

How do we encode s = 01100000001<u>0</u> ? Output: 10110001110

Compare to previous output: 01000110

* After see the last 1, don't need to add 0s to indicate empty windows. *

Encoding: Fixed Density Strings

procedure WindowEncode (input: $b_1b_2 \ldots b_n$, with exactly k ones and n-k zeros)

```
1.  w := floor (n/k)
2.  count := 0
3.  location := 1
4.  While count < k:
5.   If there is a 1 in the window starting at current location
6.       Output 1 as a marker, then output position of first 1 in window.
7.       Increment count.
8.       Update location to immediately after first 1 in this window.
9.   Else
10.      Output 0.
11.      Update location to next index after current window.
```

Uniquely decodable?

Decoding: Fixed Density Strings

```
procedure WindowDecode (input: x_1x_2…x_m, target is exactly k ones and n-k zeros)
    w := floor ( n/k )
    b := floor ( log_2(w))
    s := empty string
    i := 0
    While i < m
        If x_i = 0
            s += 0…0 (j times)
            i += 1
        Else
            p := decimal value of the bits x_{i+1}…x_{i+b}
            s += 0…0 (p times)
            s += 1
            i := i+b+1
    If length(s) < n
        s += 0…0 ( n-length(s) times )
    Output s.
```

Encoding/Decoding: Fixed Density Strings

Correctness?

E(s) = result of encoding string s of length n with k 1s, using `WindowEncode`.

D(t) = result of decoding string t to create a string of length n with k 1s, using `WindowDecode`.

Well-defined functions?
Inverses?

Goal: For each s, D(E(s)) = s.
Strong Induction!

Encoding/Decoding: Fixed Density Strings

Output size?

Assume n/k is a power of two. Consider s a binary string of length n with k 1s.

How long is E(s)?

A. n-1
B. $\log_2(n/k)$
C. Depends on where 1s are located in s

Encoding/Decoding: Fixed Density Strings

Output size?

Assume n/k is a power of two. Consider s a binary string of length n with k 1s.

For which strings is E(s) shortest?

A. More 1s toward the beginning.
B. More 1s toward the end.
C. 1s spread evenly throughout.

Encoding/Decoding: Fixed Density Strings

Output size?

Assume n/k is a power of two. Consider s a binary string of length n with k 1s.

Best case : 1s toward the beginning of the string. $E(s)$ has

- One bit for each 1 in s to indicate that next bits denote positions in window.
- $\log_2(n/k)$ bits for each 1 in s to specify position of that 1 in a window.
- k such ones.
- No bits representing 0s because all 0s are "caught" in windows with 1s or after the last 1.

Total $|E(s)| = k \log_2(n/k) + k$

Encoding/Decoding: Fixed Density Strings

Output size?

Assume n/k is a power of two. Consider s a binary string of length n with k 1s.

Worst case : 1s toward the end of the string. E(s) has

- Some bits representing 0s since there are no 1s in first several windows.
- One bit for each 1 in s to indicate that next bits denote positions in window.
- $\log_2(n/k)$ bits for each 1 in s to specify position of that 1 in a window.
- k such ones.

What's an upper bound on the number of these bits?

A. n
B. n-k
C. k
D. 1
E. None of the above.

Encoding/Decoding: Fixed Density Strings

Output size?

Assume n/k is a power of two. Consider s a binary string of length n with k 1s.

Worst case : 1s toward the end of the string. E(s) has

- At most k bits representing 0s since there are no 1s in first several windows.
- One bit for each 1 in s to indicate that next bits denote positions in window.
- $\log_2(n/k)$ bits for each 1 in s to specify position of that 1 in a window.
- k such ones.

Total $|E(s)| \le k \log_2(n/k) + 2k$

Encoding/Decoding: Fixed Density Strings

Output size?

Assume n/k is a power of two. Consider s a binary string of length n with k 1s.

$$k \log_2(n/k) + k \quad <= \quad |E(s)| \quad <= \quad k \log_2(n/k) + 2k$$

Using this inequality, there are at most _____ length n strings with k 1s.

A. 2^n
B. n
C. $(n/k)^2$
D. $(n/k)^k$
E. None of the above.

Encoding/Decoding: Fixed Density Strings

Output size?

Assume n/k is a power of two. Consider s a binary string of length n with k 1s. Given **| E(s) | <= k $\log_2(n/k)$ + 2k,** we need at most k $\log_2(n/k)$ + 2k bits to represent all length n binary strings with k 1s. Hence, there are at most $2^{\cdots}$ many such strings.

Encoding/Decoding: Fixed Density Strings

Output size?

Assume n/k is a power of two. Consider s a binary string of length n with k 1s. Given **| E(s) | <= k $\log_2$(n/k) + 2k,** we need at most k $\log_2$(n/k) + 2k bits to represent all length n binary strings with k 1s. Hence, there are at most $2^{\cdots}$ many such strings.

$$
\begin{aligned}
2^{(k\log(n/k)+2k)} &= 2^{(k\log(n/k))}\cdot 2^{(2k)} \\
&= \left(2^{(\log(n/k))}\right)^{k}\cdot 2^{(2k)} \\
&= (n/k)^k \cdot 4^k = (4n/k)^k
\end{aligned}
$$

Encoding/Decoding: Fixed Density Strings

Output size?

Assume n/k is a power of two. Consider s a binary string of length n with k 1s. Given **| E(s) | <= k $\log_2(n/k)$ + 2k,** we need at most k $\log_2(n/k)$ + 2k bits to represent all length n binary strings with k 1s. Hence, there are at most $2^{\cdots}$ many such strings.

$$
\begin{aligned}
2^{(k\log(n/k)+2k)} &= 2^{(k\log(n/k))}\cdot 2^{(2k)} \\
&= \left(2^{(\log(n/k))}\right)^k \cdot 2^{(2k)} \\
&= (n/k)^k \cdot 4^k = (4n/k)^k
\end{aligned}
$$

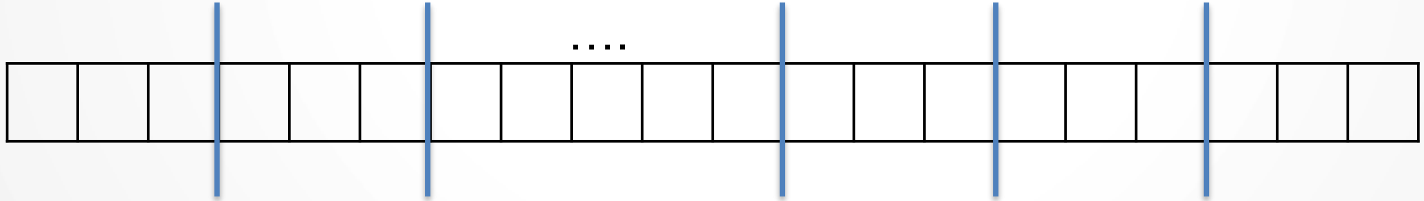
C(n,k) = **# Length n binary strings with k 1s** <= $(4n/k)^k$

Bounds for Binomial Coefficients

Using `windowEncode()`: $\binom{n}{k} \le (4n/k)^k$

Lower bound?

Idea: find a way to count a subset of the fixed density binary strings.



Some fixed density binary strings have one 1 in each of k chunks of size n/k.

How many such strings are there?

A. n^n B. $k!$ C. $(n/k)^k$ D. $C(n,k)^k$ E. None of the above.

Bounds for Binomial Coefficients

Using `windowEncode()`: $$\binom{n}{k} \leq (4n/k)^k$$

Using evenly spread strings:

$$(n/k)^k \leq \binom{n}{k}$$

Counting helps us analyze our **compression algorithm.**

Compression algorithms help us **count.**