COMP S264F Unit 6: Combinatorics

Dr. Keith Lee

School of Science and Technology

The Open University of Hong Kong

Overview

Combinatorics is a mathematics area primarily concerned with counting, both as a means and an end in obtaining results and generating the arrangements of objects.

- Permutations
- Permutations with repetition
- Combinations
- Combinations with repetition

- Puzzle: Locks and Keys for a Safe
- Lower bound

- Combinatorial proofs

Permutations

Let S be a finite set of *n* distinct objects.

- A permutation of S is an *ordered arrangement* of all the objects in S (i.e., an *n*-tuple).

  E.g., S = {John, Mary, Peter, Tom}.

  (Peter, Tom, Mary, John) is a permutation of S,

  and (Tom, Mary, John, Peter) is another.

- For any integer ***r*** $\leq$ |S|, an ***r***-permutation of S is an ordered arrangement of ***r*** elements of S.

  E.g., (Peter, John, Mary) is a 3-permutation of S.

Generating permutations in Python

- `permutations(iterable, r=None)` function in the `itertools` package generates all *r*-permutations from iterable (e.g., a list).

```
from itertools import permutations

# Generate all 3-permutations
perm = permutations(['John', 'Mary', 'Peter', 'Tom'], 3)

for i in perm:
    print(i)
```

Output:
('John', 'Mary', 'Peter')
('John', 'Mary', 'Tom')
('John', 'Peter', 'Mary')
…

Number of Permutations

Let $P(n, r)$ denote the number of r-permutations of a set with n elements.

$$P(n, r) = n(n-1)(n-2)\ldots(n-r+1)$$
$$= \frac{n(n-1)(n-2)\ldots(n-r+1)(n-r)(n-r-1)\ldots 2\cdot 1}{(n-r)(n-r-1)\ldots 2\cdot 1} = \frac{n!}{(n-r)!}\ .$$

Proof: To form an r-permutations, we have

- n ways to choose the 1st element, followed by
- $n-1$ ways to choose the 2nd element, followed by
- $n-2$ ways to choose the 3rd element, followed by
- …
- $n-r+1$ ways to choose the r-th element.

By the product rule, $P(n, r) = n(n-1)(n-2)\ldots(n-r+1)$.

NB. $n! = n(n-1)(n-2)\ldots1$; $0! = 1$; $P(n, 0) = 1$

Example

- Suppose that there are eight runners in a race.
- How many different ways are there to award the gold, silver, and bronze medals, if all possible outcomes can occur?
- Answer: $P(8, 3) = 8 \times 7 \times 6 = 336$.

Number of Permutations in Python

- P(n, r) can also be denoted by ${}_nP_r$ or P^n_r.

- We can compute P(n,r) as follows:

```
from math import factorial

def nPr(n, r):
    return factorial(n) // factorial(n-r)
    # Use // for integer division (floor division)

print(nPr(8,3))
```

Output:
336

Permutations with repetition

- A **multiset** (a.k.a. **bag**) is like a set but allows for *multiple instances* of each of its elements.
  E.g., multiset {**a**, **a**, b, b, b}

- If a bag contains m_1 object 1, m_2 object 2, m_3 object 3, …, m_k object k, then the number of its permutations is

$$\frac{(m_1 + m_2 + \ldots + m_k)!}{m_1!\, m_2! \ldots m_k!}$$

because the m_i! different permutations on the m_i object i would be treated as the same permutation.

E.g., (**a**, b, b, **a**, b) and (**a**, b, b, **a**, b) are the same.

Combinations

- With respect to a set S of n elements,

  an ***r***-combination is an *unordered* selection

  of ***r*** elements of S (i.e., a subset of size ***r***).

S = {1, 2, 3, 4}

- {4, 3, 1} is a 3-combination of S;
- {1, 3, 4} refers to the same 3-combination.

Generating combinations in Python

- `combinations(`*`iterable, r`*`)` function in the `itertools` package generates all *r*-combinations from iterable (e.g., a list).

```
from itertools import combinations

# Generate all 2-combinations
comb = combinations([1, 2, 3], 2)
for i in comb:
    print(i)
```

Output:
(1, 2)
(1, 3)
(2, 3)

Number of Combinations

Let **C(*n*, *r*)** denote the number of ***r***-combinations of a set with *n* elements.

$$\mathbf{C}(n, r) = \frac{n!}{r!\,(n-r)!}$$

Proof:

All the ***r***-permutations of n elements can be generated as follows:

- enumerate every possible ***r***-combinations; and
- for each chosen ***r***-combination, generate all possible *ordered* arrangements of the elements in this ***r***-combination.

Thus, $P(n, r) = \mathbf{C}(n, r) \times \mathbf{P}(r, r) = \mathbf{C}(n, r) \times r\,!$.

It follows that $\mathbf{C}(n, r) = \dfrac{P(n, r)}{r\,!} = \dfrac{n!}{r\,!\,(n-r)!}$.

Example

S = {A, B, C, D, E}

3-combination:

{A, B, D}

All possible ordered arrangements:

(A, B, D), (A, D, B), (B, D, A), (B, A, D), (D, A, B), (D, B, A)

NB. C(n, 0) = 1

Number of Combinations in Python

- C(n, r) can also be denoted by ${}_nC_r$, C_r^n or $\binom{n}{r}$.
- `comb(n, r, exact=True)` function in `scipy.special` package computes C(n, r).
- E.g., Computing C(10, 3):

```
from scipy.special import comb

print(comb(10, 3, exact=True))
```

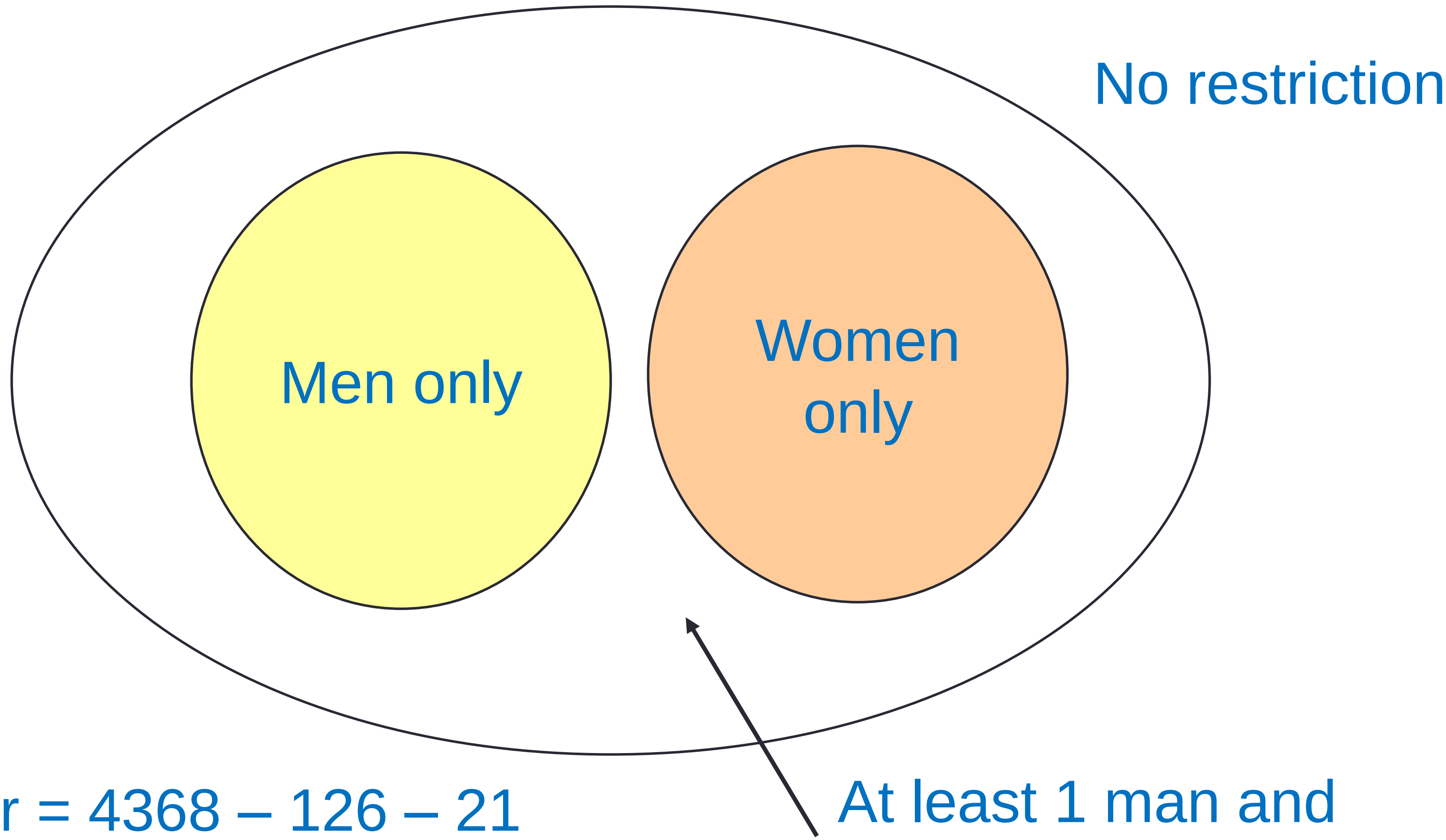
Output:
120

Example

- The teaching staff of School XXX comprises 7 women and 9 men.

- How many ways are there to select a committee of five members if at least one woman and at least one man must be on the committee?

- # of 5-member committees <u>without any restriction</u> = C(16, 5) = 4368.
- # of 5-member committees containing <u>men only</u> = C(9, 5) = 126.
- # of 5-member committees containing <u>women only</u> = C(7, 5) = 21.

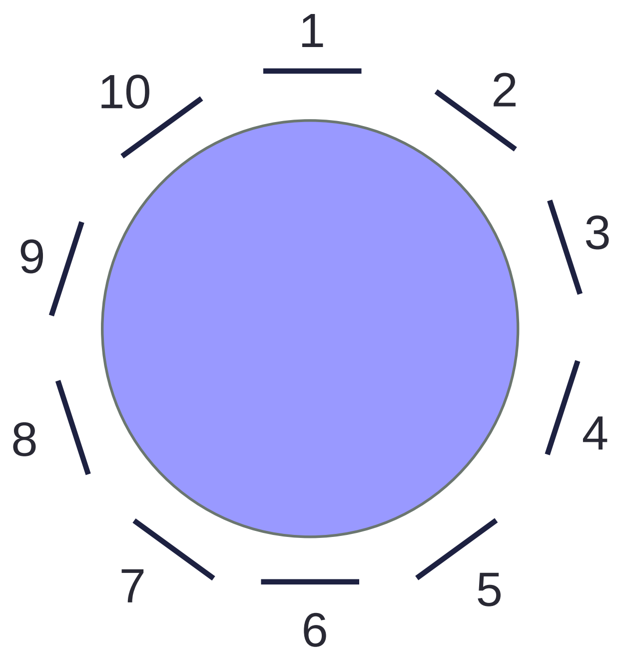


Answer = 4368 – 126 – 21
= 4221

Seating Plan

- How many ways are there to seat 10 people around a circular table, where two seating plans are considered to be the same if they can be obtained from each other by rotating the table?

- Answer:



Combinations with repetition

- How many ways to select 4 bills from a cash box containing $10 bills, $20 bills, $50 bills, $100 bills?
- E.g., one of each kind

  x | x | x | x
- $10: 2; $50; 2

  x x | | x x |
- $10: 1; $20: 2; $100: 1

  x | x x | | x
- $100: 4

  | | | x x x x

Fact: Any ordering of 4 x's and 3 |'s defines a unique way to select 4 bills.

How many ways to order 4 x's and 3 |'s ?

Combinations with repetition (cont')

- If repetition is allowed,

  the number of ***r***-combinations of a set of **n** items is C(n+r-1, r).

Combinations with repetition in Python

- `combinations_with_replacement(iterable, r)` function in the `itertools` package generates all *r*-combinations with repeated elements from iterable (e.g., a list).

```
from itertools import combinations_with_replacement

# Generate all 2-combinations with repeated elements
comb = combinations_with_replacement([1, 2, 3], 2)
for i in comb:
    print(i)
```

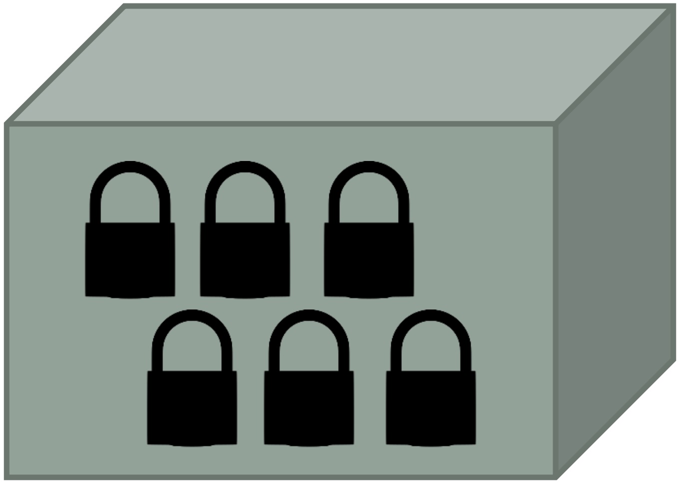
Output:
(1, 1)
(1, 2)
(1, 3)
(2, 2)
(2, 3)
(3, 3)

- `comb(n, r, exact=True, repetition=True)` function in `scipy.special` package computes the number of *r*-combinations with repeated items from *n* elements.

```
from scipy.special import comb
print(comb(4+4-1, 4, exact=True))
print(comb(4, 4, exact=True, repetition=True))
```

Output:
35
35

Puzzle: Locks and Keys for a Safe



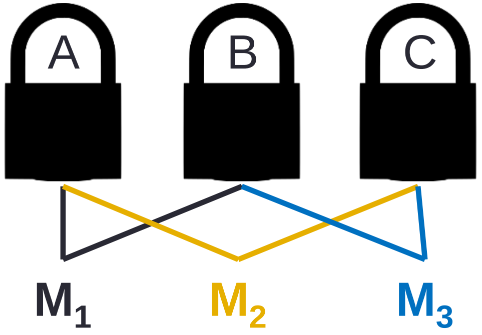
- There are *n* (say, 3) committee members who are holding keys to these locks.
- Design a system (How many locks? How are keys distributed?) so that the safe can only be opened by any *m* (say, 2) members.

Puzzle: Examples

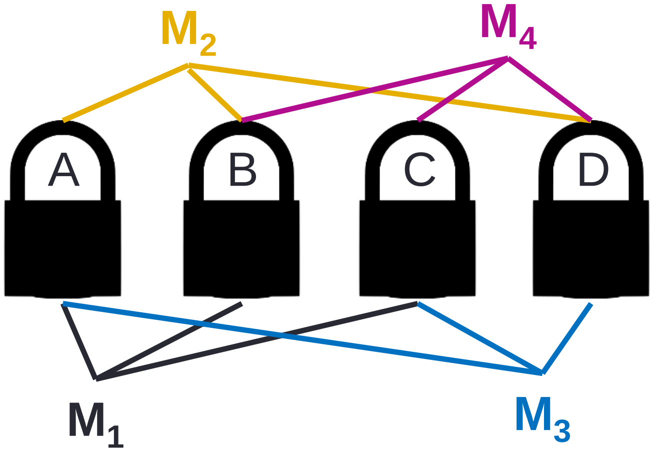
- Design a safe with **multiple** locks that can only be opened by any *m* of *n* members.

Example:

- **m = 1:** 1 lock, 1 key/member
- **m = n:** *n* locks, 1 key/member



- **m = 2, n = 3:**
  3 locks, 2 keys/member

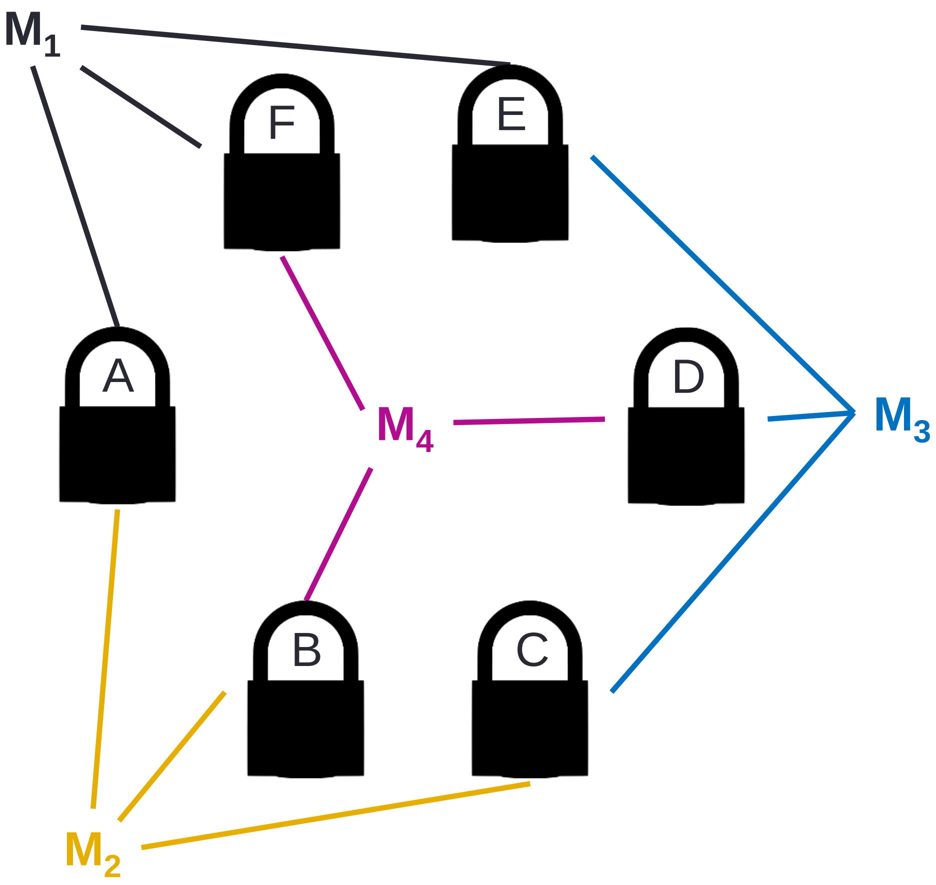


- **m = 2, n = 4:**
  4 locks, 3 keys/member

Puzzle: Examples (cont')

- How many locks and keys are needed for *m = 3, n = 4*?

- 6 locks and
  3 keys/member



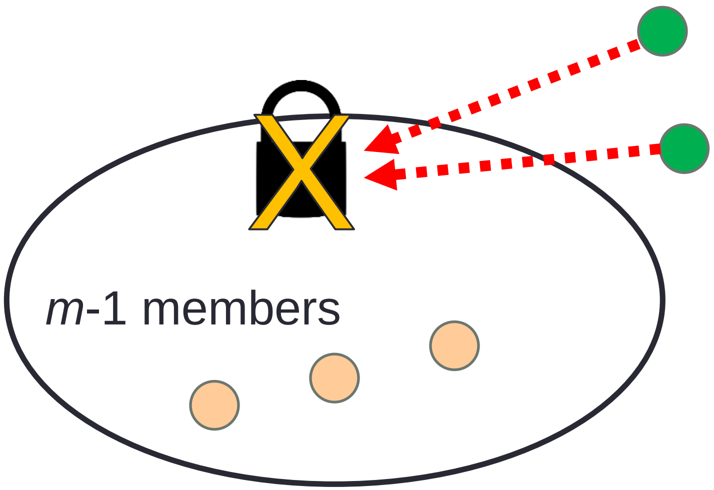
Lower Bounds (minimum, at least …)

Can we use less locks and keys for the above cases?

- What is the minimum number of locks?
- What is the minimum number of keys per member?

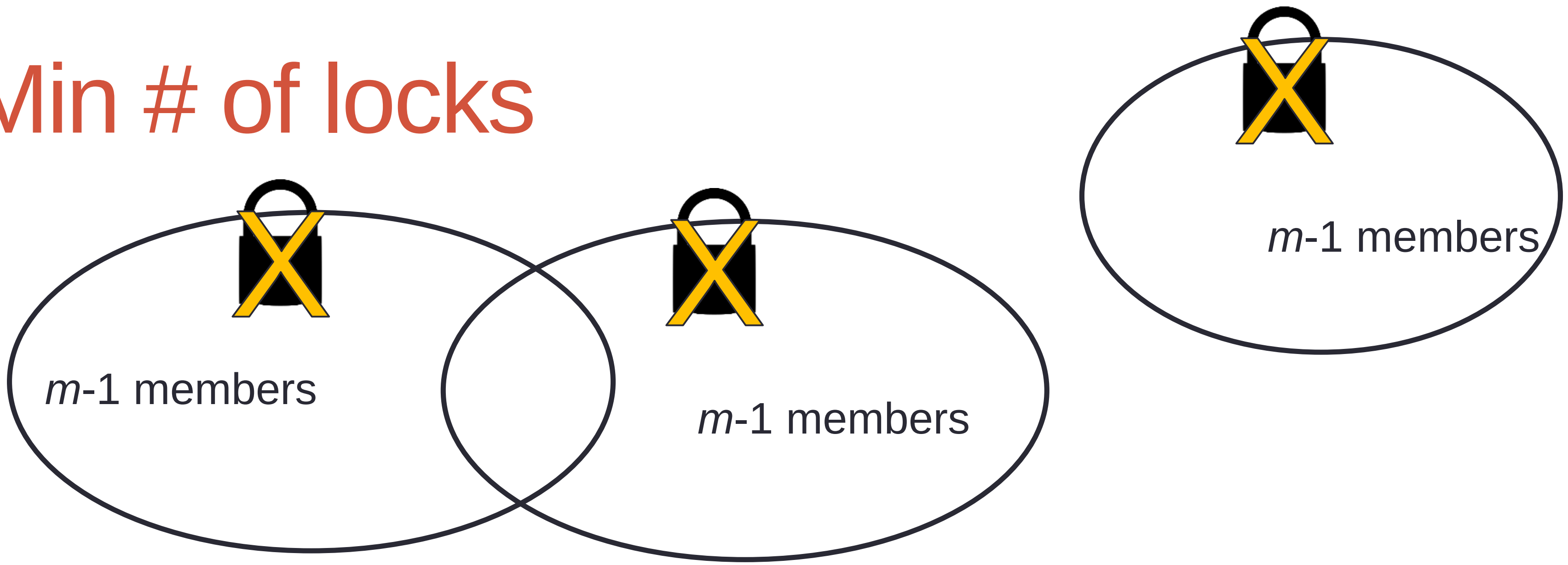
- Minimum number of locks required = $C(n, m-1)$
- Minimum number of keys per member required = $C(n-1, m-1)$

Groups & Locks



- **Consider a group A of *m*-1 members.**
  - They cannot open at least one lock, say, X.
  - X can be opened by any member <u>outside</u> group A; otherwise, there exists a group of *m* members who cannot open the safe.
  - X can be opened by <u>any other group</u> of *m*-1 members.

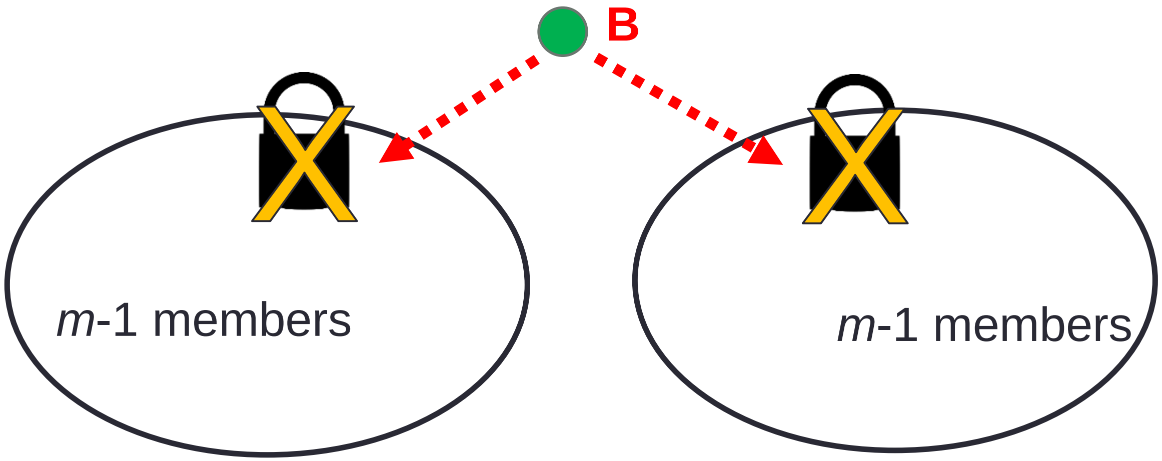
Min # of locks



- Each group of (m-1) members is characterized by at least one distinct lock (which only they can't open).
- There are C(n, m-1) different groups of (m-1) members.

  Thus, there are at least C(n, m-1) distinct locks.
- Minimum number of locks = C(n, m-1).

Min # of keys per member



Each member **B** can open the lock that can't be opened by any group of (m-1) members not including **B**.

- There are C(n-1, m-1) such groups and at least C(n-1, m-1) such locks.
- **B** must have at least C(n-1, m-1) keys.
- Minimum number of keys per member = C(n-1, m-1).

Number of locks and keys

Check:

- **m = 2, n = 3:**

  C(3,1) = 3 locks,   C(2,1) = 2 keys/member

- **m = 2, n = 4:**

  C(4,1) = 4 locks,   C(3,1) = 3 keys/member

- **m = 3, n = 4:**

  C(4,2) = 6 locks,   C(3,2) = 3 keys/member

Interesting Identities

For any non-negative integers m, n, k, where $k \leq m$ & $k \leq n$,

1. $C(n, k) = C(n, n-k)$

2. $\Sigma_{0 \leq i \leq n} C(n, i) = 2^n$

3. $C(n, 0) - C(n, 1) + C(n, 2) - C(n, 3) + \ldots = 0$

4. $C(n+1, k) = C(n, k-1) + C(n, k)$

5. $C(m+n, k) = \Sigma_{0 \leq i \leq k} C(m, k-i) \times C(n, i)$

$\Sigma_{0 \leq i \leq n} C(n, i) = 2^n$

Combinatorial proof of an identity is a proof that uses counting arguments to prove that both sides of the identity count the same objects but in different ways.

Proof:

- The number of different ways of forming a subset of a set of *n* elements is 2^n.

- $C(n, i)$ is the number of ways of choosing a subset of size i from a set of *n* elements.
- By sum rule, the number of different ways of forming a subset of a set of *n* elements = $\Sigma_{0 \leq i \leq n} C(n, i)$.

Therefore, $\Sigma_{0 \leq i \leq n} C(n, i) = 2^n$.

$C(n, 0) – C(n, 1) + C(n, 2) – C(n, 3) + \ldots = 0$

Binomial Theorem:

$$(x+y)^n = C(n, 0)\ x^n + C(n, 1)\ x^{n-1}y + C(n, 2)\ x^{n-2}y^2 + C(n,3)\ x^{n-3}y^3 + \ldots + C(n, n)\ y^n$$

Let x = 1 and y = -1. Then,

$$0 = C(n, 0) + C(n, 1)\ (-1) + C(n, 2)\ (-1)^2 + C(n,3)\ (-1)^3 + \ldots + C(n, n)\ (-1)^n$$

$\Rightarrow \quad C(n, 0) – C(n, 1) + C(n, 2) – C(n, 3) + \ldots = 0.$

Combinatorial proof: C(n+1, k) = C(n, k-1) + C(n, k)

- Let A be any set of n+1 elements.
  Let x be one of the elements in A.

- C(n+1, k) is equal to the number of k-combinations of A; each k-combination may or may not contain x.
  - # of k-combinations containing x = C(n, k-1)
  - # of k-combinations not containing x = C(n, k)

- Therefore, C(n+1, k) = C(n, k-1) + C(n, k).

Pascal Triangle

$$\binom{0}{0}$$

$$\binom{1}{0} \quad \binom{1}{1}$$

$$\binom{2}{0} \quad \binom{2}{1} \quad \binom{2}{2}$$

$$\binom{3}{0} \quad \binom{3}{1} \quad \binom{3}{2} \quad \binom{3}{3}$$

$$\binom{4}{0} \quad \binom{4}{1} \quad \binom{4}{2} \quad \binom{4}{3} \quad \binom{4}{4}$$

1

1 1

1 2 1

1 3 3 1

1 4 6 4 1

1 5 10 10 5 1

1 6 15 20 15 6 1

$C(m+n, k) = \Sigma_{0 \le i \le k} C(m, k-i) \times C(n, i)$

Let A be a set containing *n* boys and *m* girls.

A k-combination of A contains either

- 0 boy and k girls, or
- 1 boy and k-1 girls, or
- 2 boys and k-2 girls, or
- …
- k boys and 0 girl.

For $0 \le i \le k$,

the number of k-combinations that contain *i* boys and k-*i* girls

$= C(n, i) \times C(m, k-i)$

C(m+n, k)

= the number of k-combinations of A

$= \Sigma_{0 \le i \le k} C(m, k-i) \times C(n, i)$.