南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

CS201 DISCRETE MATHEMATICS FOR COMPUTER SCIENCE

Dr. Qi Wang

Department of Computer Science and Engineering
Office: Room903, Nanshan iPark A7 Building
Email: wangqi@sustech.edu.cn

Counting

- Assume we have a set of objects with certain properties

Counting

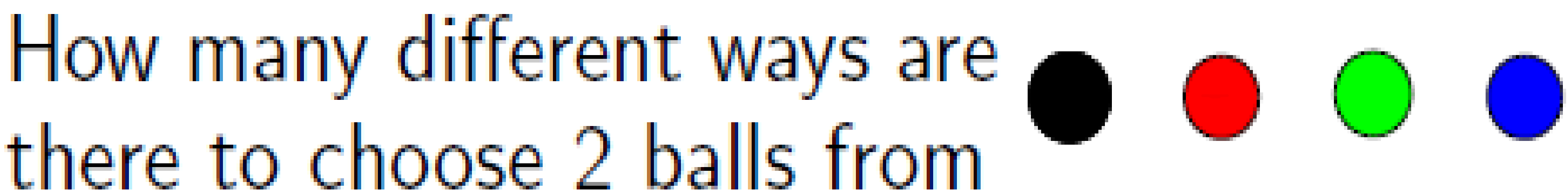
- Assume we have a set of objects with certain properties

  Counting is used to determine the number of these objects.

Counting

- Assume we have a set of objects with certain properties

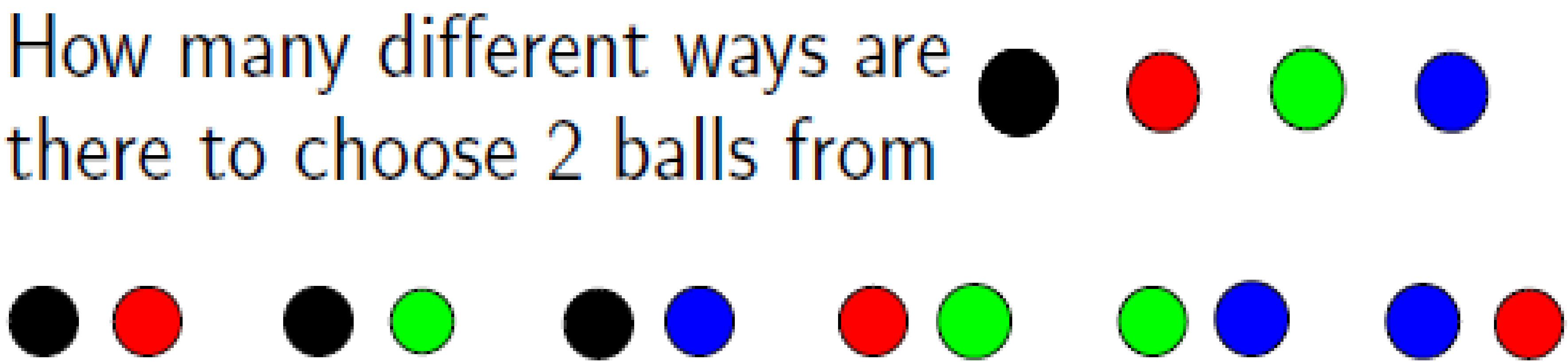
  Counting is used to determine the number of these objects.



Counting

- Assume we have a set of objects with certain properties

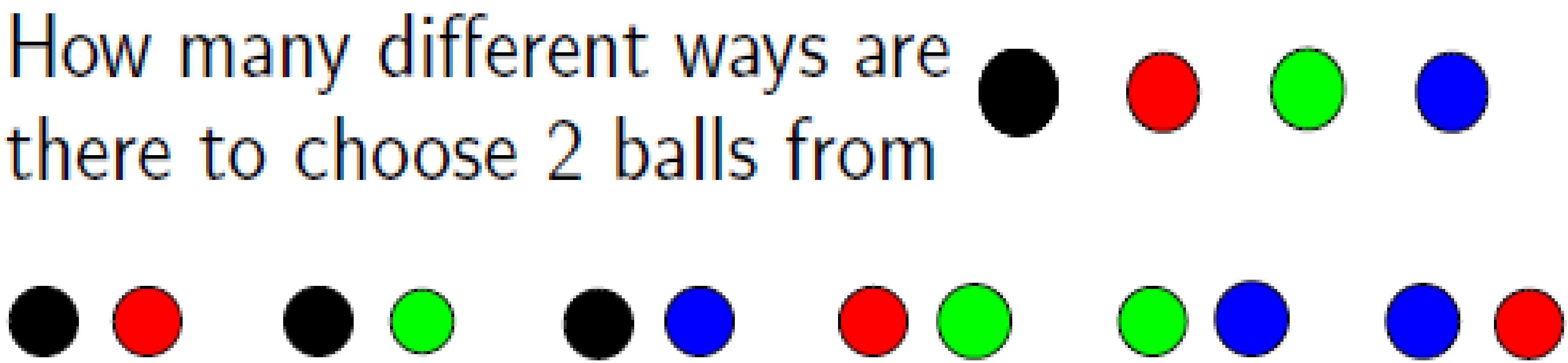
  Counting is used to determine the number of these objects.



Counting

- Assume we have a set of objects with certain properties

  Counting is used to determine the number of these objects.

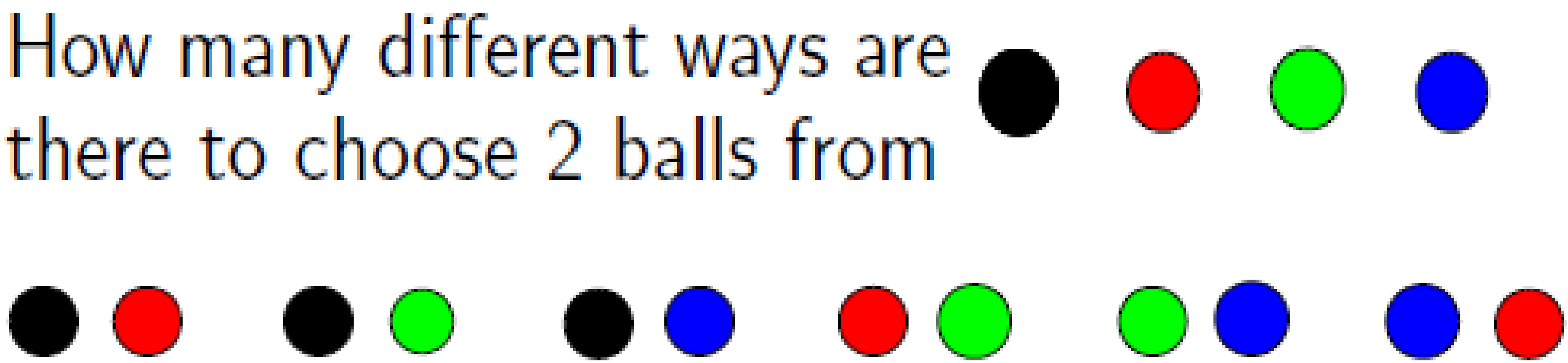


What about when order counts?

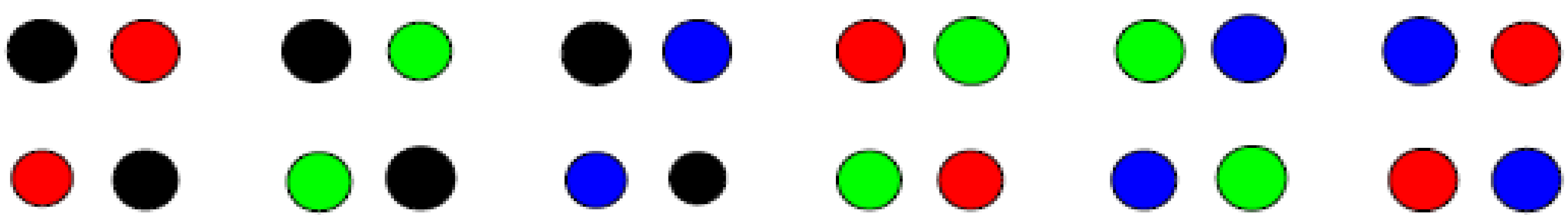
Counting

- Assume we have a set of objects with certain properties

  Counting is used to determine the number of these objects.



What about when order counts?



Counting

- Assume we have a set of objects with certain properties

  Counting is used to determine the number of these objects.

Counting

- Assume we have a set of objects with certain properties

  Counting is used to determine the number of these objects.

  Examples

  ◇ the number of steps in a computer program

  ◇ the number of passwords between $6 - 10$ characters

  ◇ the number of telephone numbers with 8 digits

Counting

- Assume we have a set of objects with certain properties

  Counting is used to determine the number of these objects.

  Examples

  - ◇ the number of steps in a computer program
  - ◇ the number of passwords between $6 - 10$ characters
  - ◇ the number of telephone numbers with 8 digits

  Counting may be very hard, not trivial.

Counting

- Assume we have a set of objects with certain properties

  Counting is used to determine the number of these objects.

  Examples

  ◇ the number of steps in a computer program

  ◇ the number of passwords between $6 - 10$ characters

  ◇ the number of telephone numbers with 8 digits

  Counting may be very hard, not trivial.

  – simplify the solution by decomposing the problem

Basic Counting Rules

- *the Product Rule*

- *the Sum Rule*

Basic Counting Rules

- *the Product Rule*
  - ◇ A count decomposes into a sequence of dependent counts (each element in the first count is associated with all elements of the second count)
- *the Sum Rule*
  - ◇ A count decomposes into a set of independent counts (elements of counts are alternatives)

The Product Rule

- A count decomposes into a sequence of dependent counts (each element in the first count is associated with all elements of the second count)

The Product Rule

- A count decomposes into a sequence of dependent counts (each element in the first count is associated with all elements of the second count)

Example

In an auditorium, the seats are labeled by a letter and numbers in between 1 to 50 (e.g., $A23$). What is the total number of seats?

The Product Rule

- A count decomposes into a sequence of dependent counts (each element in the first count is associated with all elements of the second count)

Example

In an auditorium, the seats are labeled by a letter and numbers in between 1 to 50 (e.g., $A23$). What is the total number of seats?

We may either list all or use the product rule.

$$26 \times 50 = 1300$$

The Product Rule

- **Product Rule**: If a count of elements can be broken down into a sequence of dependent counts where the first count yields n_1 elements, the second n_2 elements, and kth count n_k elements, then the total number of elements is

$$n = n_1 \cdot n_2 \cdot \cdots \cdot n_k$$

The Product Rule

- **Product Rule**: If a count of elements can be broken down into a sequence of dependent counts where the first count yields n_1 elements, the second n_2 elements, and kth count n_k elements, then the total number of elements is

$$n = n_1 \cdot n_2 \cdot \cdots \cdot n_k$$

Example

How many different bit strings of length 7 are there?

The Product Rule

- **Product Rule**: If a count of elements can be broken down into a sequence of dependent counts where the first count yields n_1 elements, the second n_2 elements, and kth count n_k elements, then the total number of elements is

$$n = n_1 \cdot n_2 \cdot \cdots \cdot n_k$$

Example

How many different bit strings of length 7 are there?

How many different functions are there from a set with m elements to a set with n elements?

The Product Rule

- **Product Rule**: If a count of elements can be broken down into a sequence of dependent counts where the first count yields n_1 elements, the second n_2 elements, and kth count n_k elements, then the total number of elements is

$$n = n_1 \cdot n_2 \cdot \cdots \cdot n_k$$

Example

How many different bit strings of length 7 are there?

How many different functions are there from a set with m elements to a set with n elements?

How many one-to-one functions are there from a set with m elements to a set with n elements?

The Product Rule

- **Product Rule**: If a count of elements can be broken down into a sequence of dependent counts where the first count yields n_1 elements, the second n_2 elements, and kth count n_k elements, then the total number of elements is

$$n = n_1 \cdot n_2 \cdot \cdots \cdot n_k$$

Example

How many different bit strings of length 7 are there?

How many different functions are there from a set with m elements to a set with n elements?

How many one-to-one functions are there from a set with m elements to a set with n elements?

How many onto functions?

The Product Rule

- The following loop is a part of program computing the prodcut of two matrices.

```
(1) for i = 1 to r
(2)   for j = 1 to m
(3)     S = 0
(4)     for k = 1 to n
(5)       S = S + A[i,k] * B[k,j]
(6)     C[i,j] = S
```

The Product Rule

- The following loop is a part of program computing the prodcut of two matrices.

```
(1) for i = 1 to r
(2)   for j = 1 to m
(3)     S = 0
(4)     for k = 1 to n
(5)       S = S + A[i,k] * B[k,j]
(6)     C[i,j] = S
```

How many multiplications (in terms of r, m, n) does this program carry out in total among all iterations of line 5?

The Sum Rule

- A count decomposes into a set of independent counts (elements of counts are alternatives)

The Sum Rule

- A count decomposes into a set of independent counts (elements of counts are alternatives)

Example

You need to travel from city A to B. You may either fly, take a train, or a bus. There are 12 different flights, 5 different trains and 10 buses. How many options do you have to get from A to B?

The Sum Rule

- A count decomposes into a set of independent counts (elements of counts are alternatives)

Example

You need to travel from city A to B. You may either fly, take a train, or a bus. There are 12 different flights, 5 different trains and 10 buses. How many options do you have to get from A to B?

We may use the sum rule.

$$12 + 5 + 10$$

The Sum Rule

- **Sum Rule**: If a count of elements can be broken down into a set of independent counts where the first count yields n_1 elements, the second n_2 elements, and kth count n_k elements, then the total number of elements is

$$n = n_1 + n_2 + \cdots + n_k$$

The Sum Rule

- The following loop is from selection sort.

```
(1) for i = 1 to n-1
(2)     for j = i+1 to n
(3)         if (A[i] > A[j])
(4)             exchange A[i] and A[j]
```

The Sum Rule

- The following loop is from selection sort.

```
(1) for i = 1 to n-1
(2)    for j = i+1 to n
(3)       if (A[i] > A[j])
(4)          exchange A[i] and A[j]
```

How many comparisons (in terms of n) does this program carry out in total among all iterations of line 3?

More Complex Counting

- Typically requies a combination of the sum and product rules.

More Complex Counting

- Typically requies a combination of the sum and product rules.

Example

Each password is 6 to 8 characters long, where each character is an lowercase letter or a digit. Each password must contain at least one digit. How many possible passwords are there?

More Complex Counting

- Typically requies a combination of the sum and product rules.

Example

Each password is 6 to 8 characters long, where each character is an lowercase letter or a digit. Each password must contain at least one digit. How many possible passwords are there?

$$P = P_6 + P_7 + P_8$$

$$P_6 = 36^6 - 26^6$$

Tree Diagrams

- A *tree* is a structure that consists of a root, branches and leaves.

Tree Diagrams

- A *tree* is a structure that consists of a root, branches and leaves.

  Can be useful to represent a counting problem and record the choices we made for alternatives. The count appears on the leaves.

Tree Diagrams

- A *tree* is a structure that consists of a root, branches and leaves.

  Can be useful to represent a counting problem and record the choices we made for alternatives. The count appears on the leaves.

  Example

  What is the number of bit strings of length 4 that do not have two consecutive 1's?

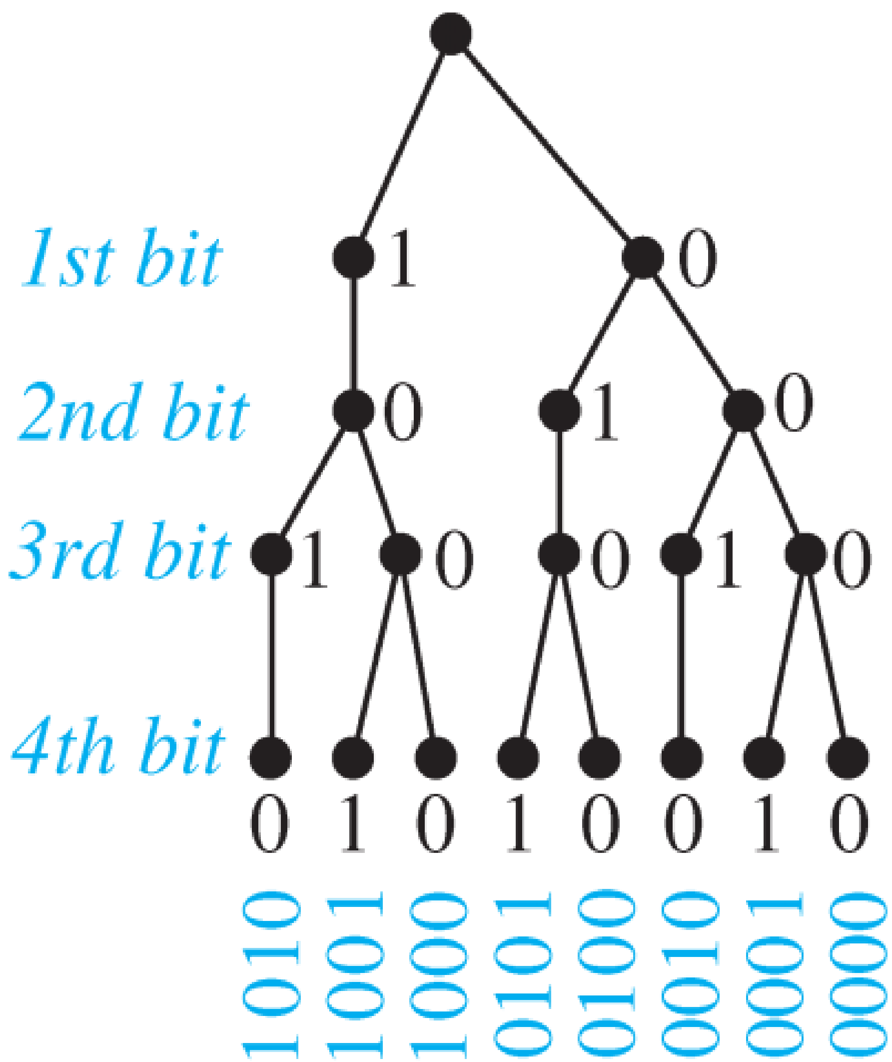
Tree Diagrams

- A *tree* is a structure that consists of a root, branches and leaves.

Can be usefu ɔlem and record the choices w count appears on the leaves.

Example

What is the ı h 4 that do not have two con

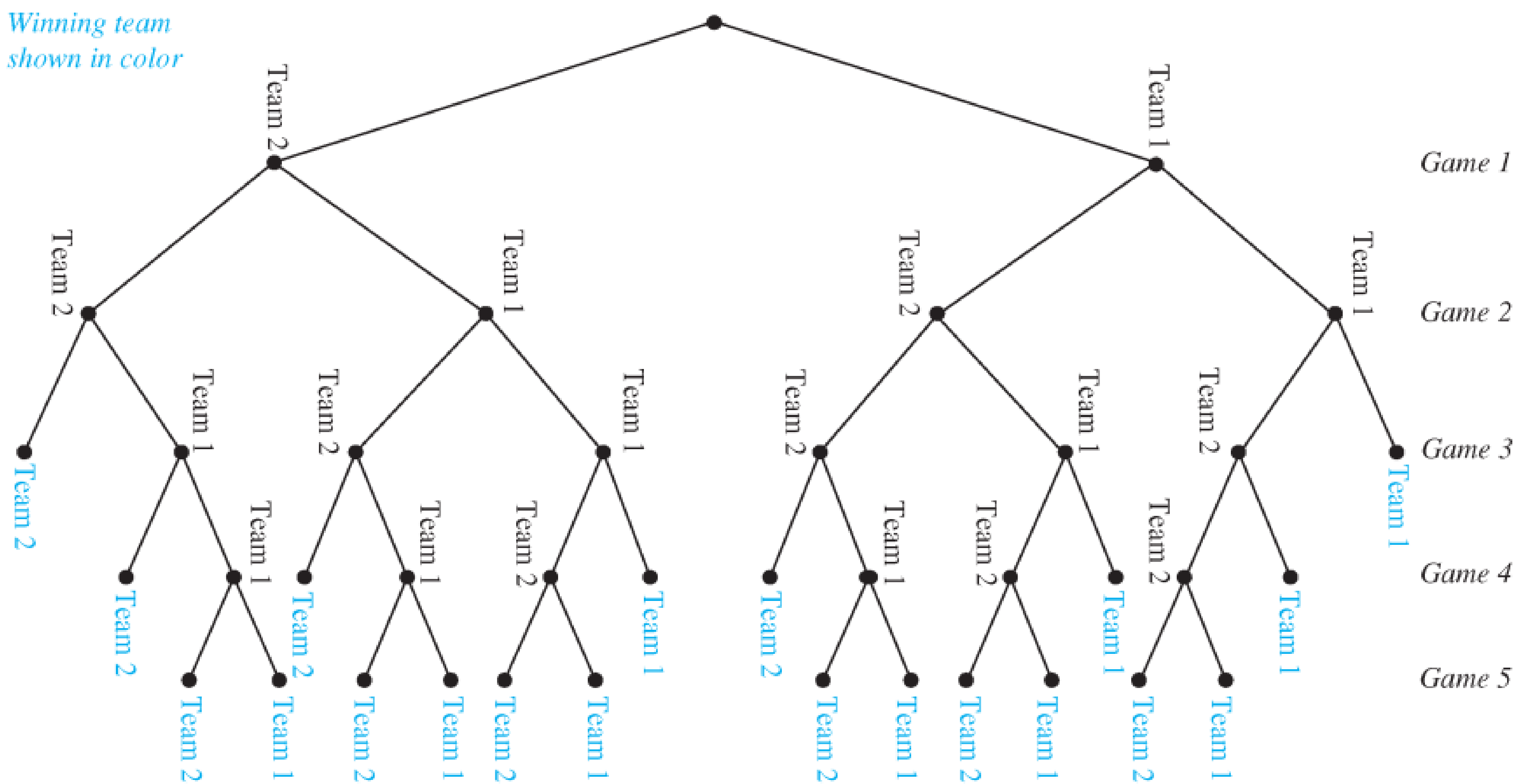


Tree Diagram

- How many different ways can a "best 3 of 5" playoff occur?

Tree Diagram

- How many different ways can a "best 3 of 5" playoff occur?



Pigeonhole Principle

- Assume that there are a set of objects and a set of bins to store them.

Pigeonhole Principle

- Assume that there are a set of objects and a set of bins to store them.

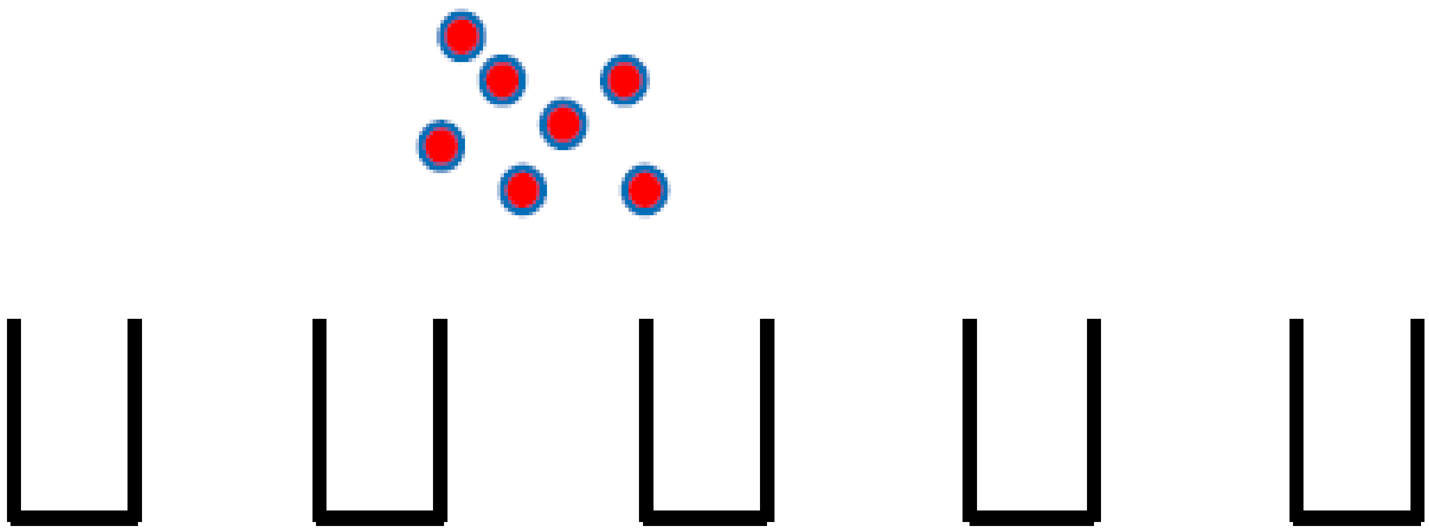
  The pigeonhole principle states that if there are more objects than bins then there is at least one bin with more than one object.

Pigeonhole Principle

- Assume that there are a set of objects and a set of bins to store them.

  The pigeonhole principle states that if there are more objects than bins then there is at least one bin with more than one object.

  Example: 7 balls and 5 bins to store them

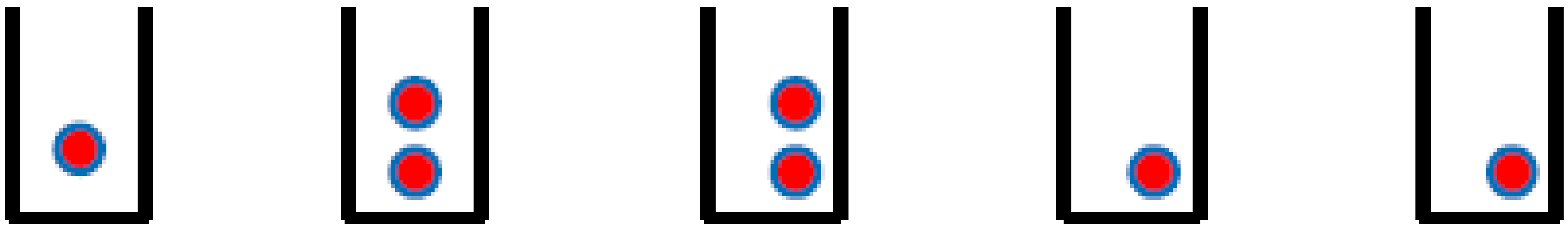


Pigeonhole Principle

- Assume that there are a set of objects and a set of bins to store them.

  The pigeonhole principle states that if there are more objects than bins then there is at least one bin with more than one object.

  Example: 7 balls and 5 bins to store them



Pigeonhole Principle

- **Theorem** If there are $k+1$ objects and k bins, then there is at least one bin with two or more objects.

Pigeonhole Principle

- **Theorem** If there are $k + 1$ objects and k bins, then there is at least one bin with two or more objects.

  Proof by contradiction

Pigeonhole Principle

- **Theorem** If there are $k+1$ objects and k bins, then there is at least one bin with two or more objects.

  Proof by contradiction

  Example

  Assume that there are 367 students. Are there any two people who have the same birthday?

  There are 5 bins and 12 objects. Then there must be a bin with at least 3 objects. Why?

Generalized Pigeonhole Principle

- If N objects are placed into k bins, then there is at least one bin containing at least $\lceil N/k \rceil$ objects.

Generalized Pigeonhole Principle

- If N objects are placed into k bins, then there is at least one bin containing at least $\lceil N/k \rceil$ objects.

Example

Assume there are 100 students. How many of them were born in the same month?

Bijections and Permutations

- A function that is both one-to-one and onto is called a *bijection*, or a *one-to-one correspondence*.

Bijections and Permutations

- A function that is both one-to-one and onto is called a *bijection*, or a *one-to-one correspondence*.

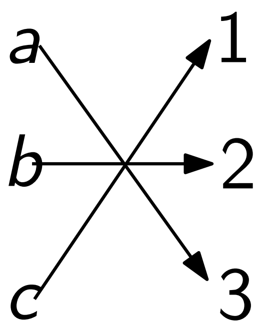
  How many bijections are there?

Bijections and Permutations

- A function that is both one-to-one and onto is called a *bijection*, or a *one-to-one correspondence*.

  How many bijections are there?

$\boxed{\text{$f : \{a, b, c\} \to \{1, 2, 3\}$ defined by $f(a) = 3, f(b) = 2, f(c) = 1$ is a bijection.}}$

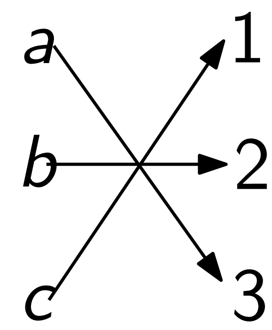


Bijections and Permutations

- A function that is both one-to-one and onto is called a *bijection*, or a *one-to-one correspondence*.

  How many bijections are there?

  $\boxed{f : \{a, b, c\} \to \{1, 2, 3\} \text{ defined by } f(a) = 3, f(b) = 2, f(c) = 1 \text{ is a bijection.}}$

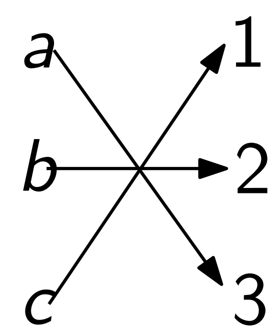
  A bijection from a set onto itself is called a *permutation*.

Bijections and Permutations

- A function that is both one-to-one and onto is called a *bijection*, or a *one-to-one correspondence*.

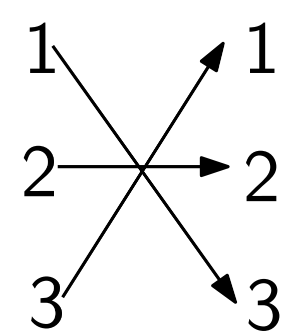
  How many bijections are there?

  $\boxed{f:\{a,b,c\}\to\{1,2,3\} \text{ defined by } f(a)=3, f(b)=2, f(c)=1 \text{ is a bijection.}}$

  A bijection from a set onto itself is called a *permutation*.

  $\boxed{f:\{1,2,3\}\to\{1,2,3\} \text{ defined by } f(1)=3, f(2)=2, f(3)=1 \text{ is a bijection.}}$

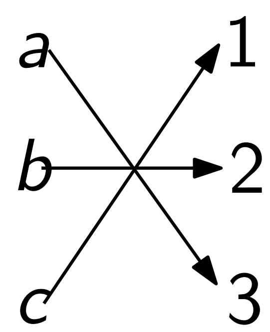
  

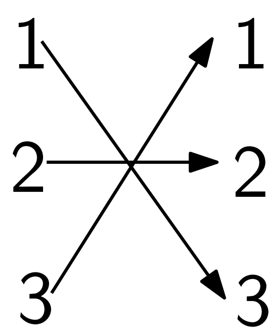
Bijections and Permutations

- A function that is both one-to-one and onto is called a *bijection*, or a *one-to-one correspondence*.

  A bijection from a set onto itself is called a *permutation*.

  In a bijection,
  exactly one arrow leaves each item on the left and exactly one arrow arrives at each item on the right.





Bijections and Permutations

- A function that is both one-to-one and onto is called a *bijection*, or a *one-to-one correspondence*.

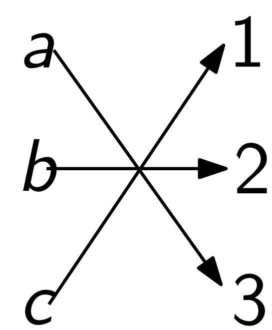
  A bijection from a set onto itself is called a *permutation*.

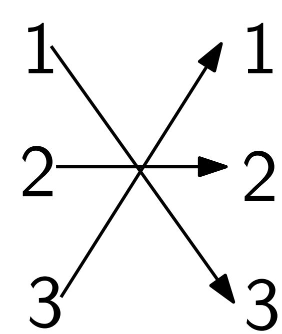
  In a bijection,

  exactly one arrow leaves each item on the left and exactly one arrow arrives at each item on the right.

  Thus,

  the left and right sides must have the same size.





The Bijection Principle

- The following loop is a part of program to determine the number of triangles formed by n points in the plane.

```
(1) trianglecount = 0
(2)   for i = 1 to n
(3)     for j = i+1 to n
(4)       for k = j+1 to n
(5)         if points i, j, k are not collinear
(6)           trianglecount = trianglecount + 1
```

The Bijection Principle

- The following loop is a part of program to determine the number of triangles formed by n points in the plane.

```
(1) trianglecount = 0
(2)   for i = 1 to n
(3)     for j = i+1 to n
(4)       for k = j+1 to n
(5)         if points i, j, k are not collinear
(6)           trianglecount = trianglecount + 1
```

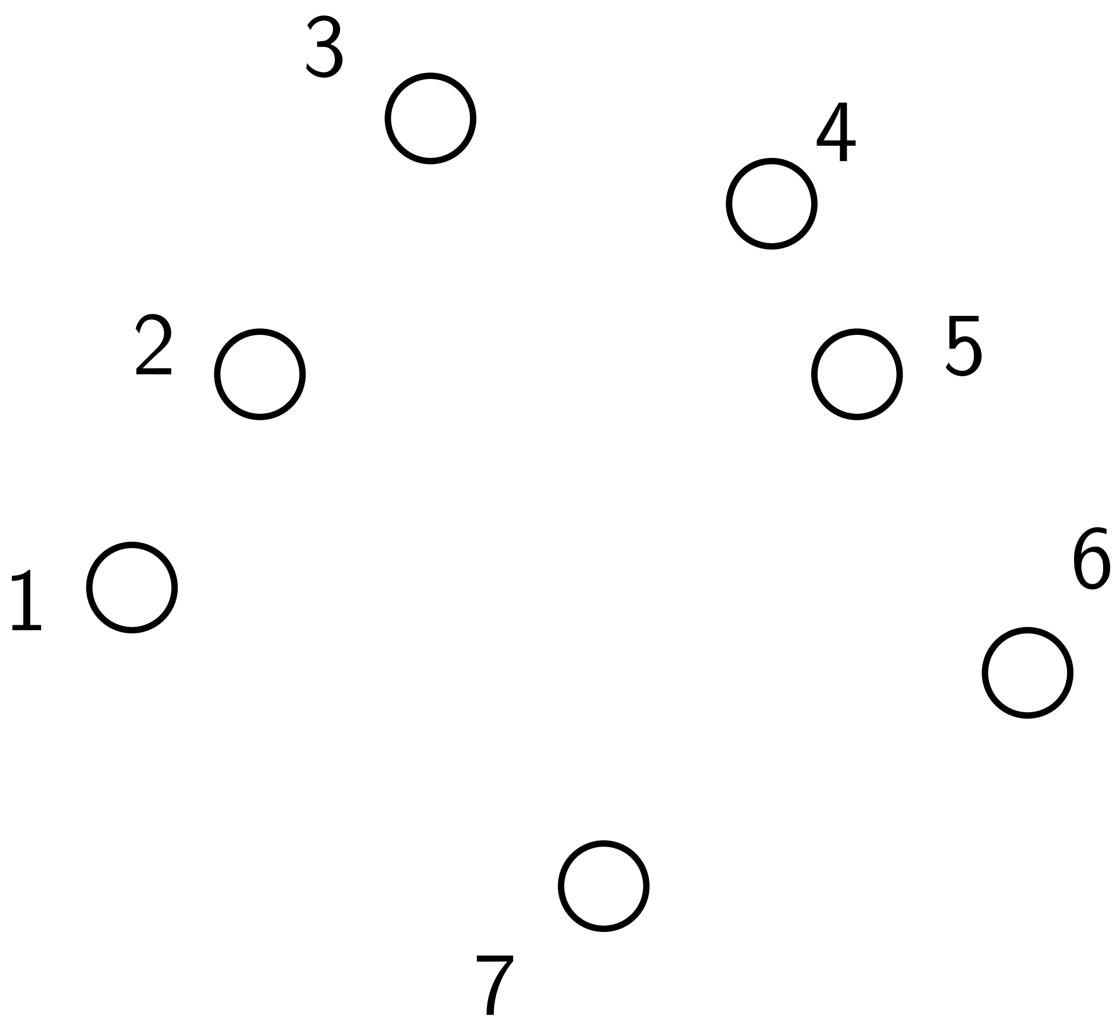
Among all iterations of line 5, what is the total number of times this line checks three points to see if they are collinear?

Counting Triangles

- 3 points form a triangle if and only if they are non collinear

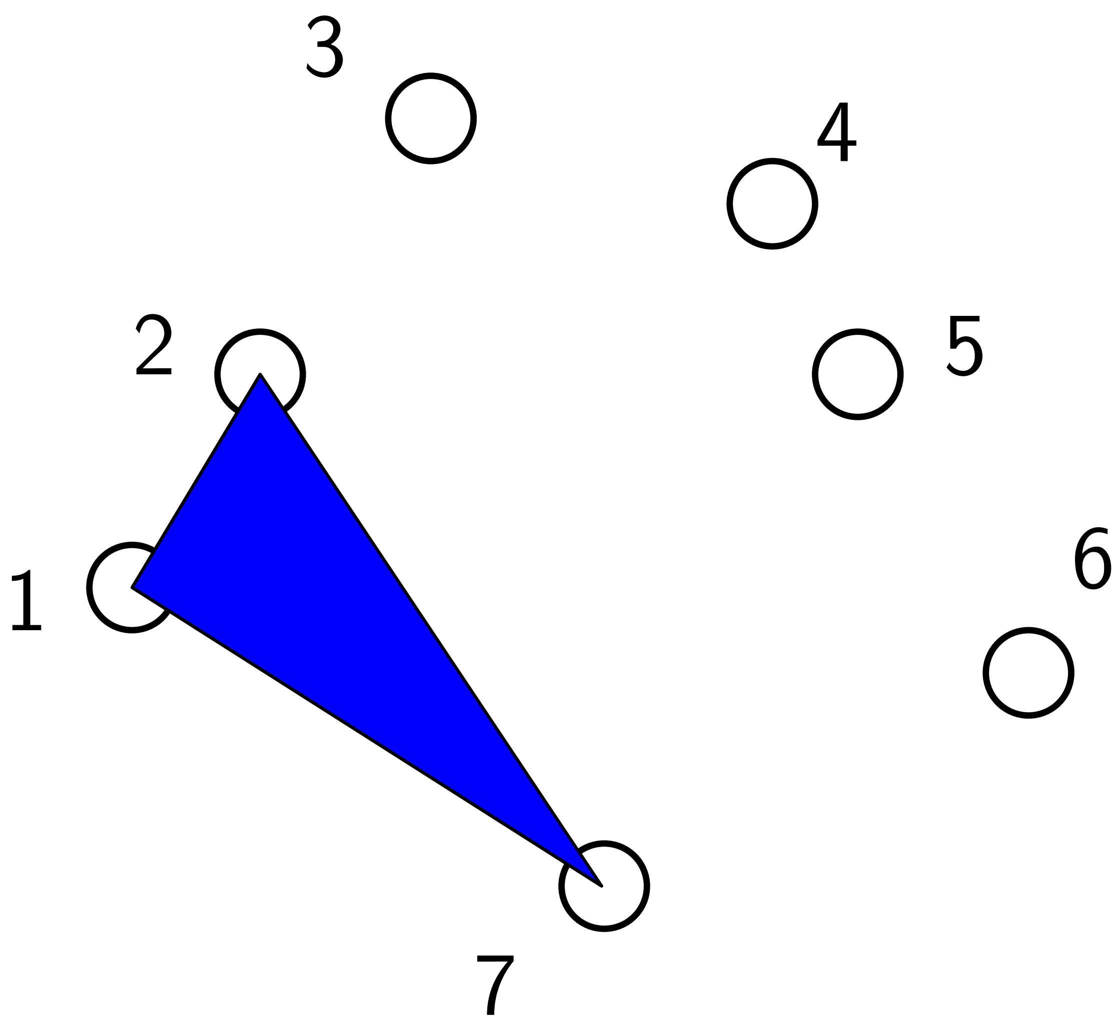
Counting Triangles

- 3 points form a triangle if and only if they are non collinear



Counting Triangles

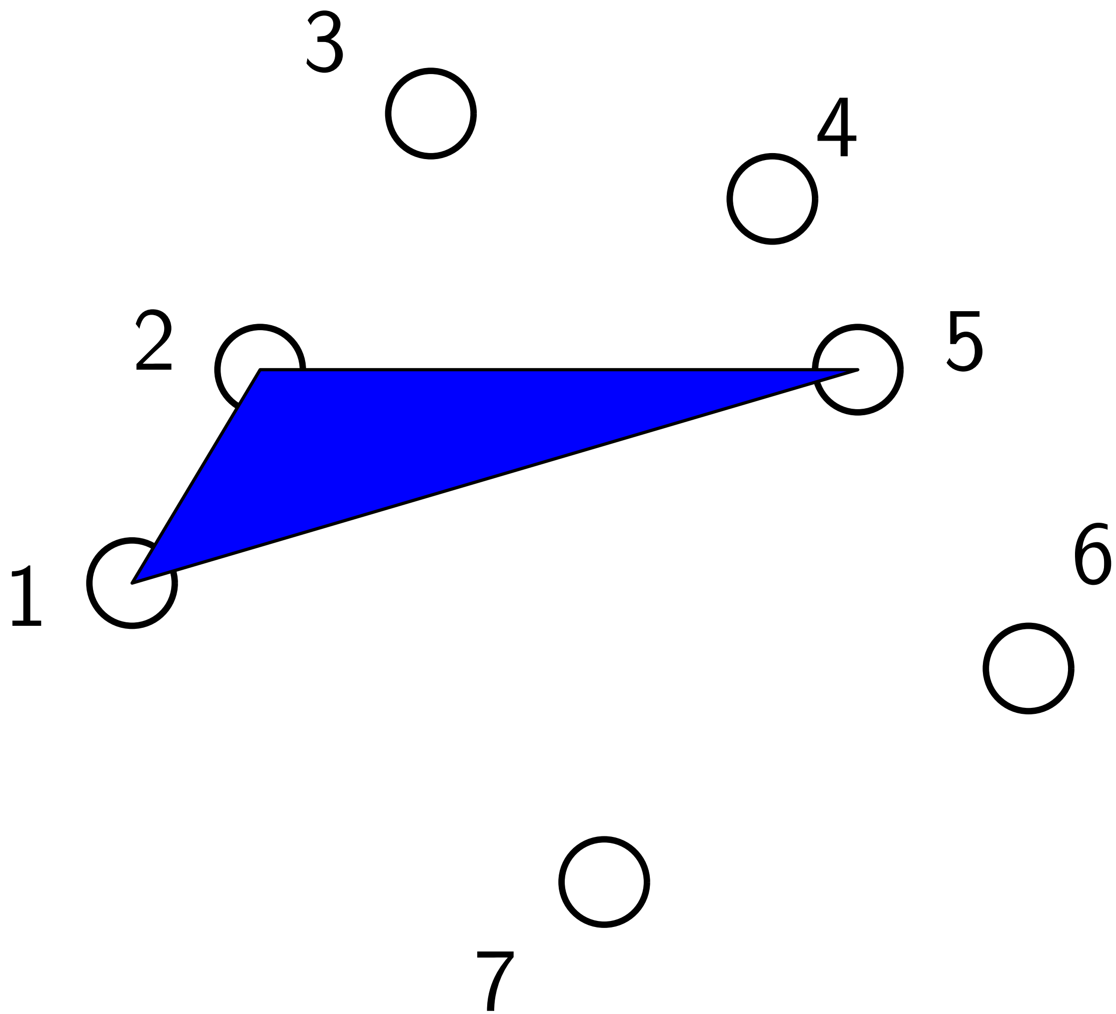
- 3 points form a triangle if and only if they are non collinear



1 – 2 – 7: yes

Counting Triangles

- 3 points form a triangle if and only if they are non collinear

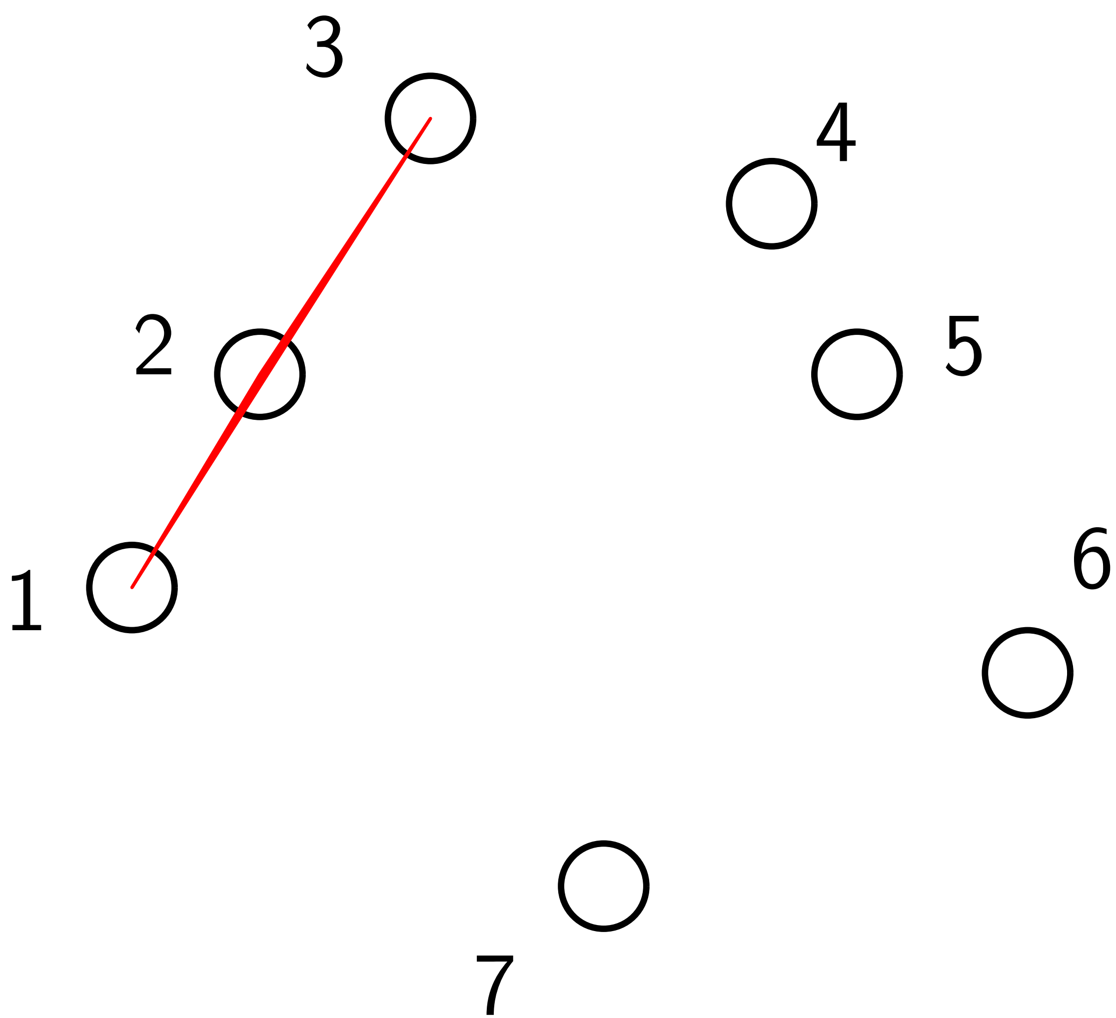


$1 - 2 - 7$: yes

$1 - 2 - 5$: yes

Counting Triangles

- 3 points form a triangle if and only if they are non collinear



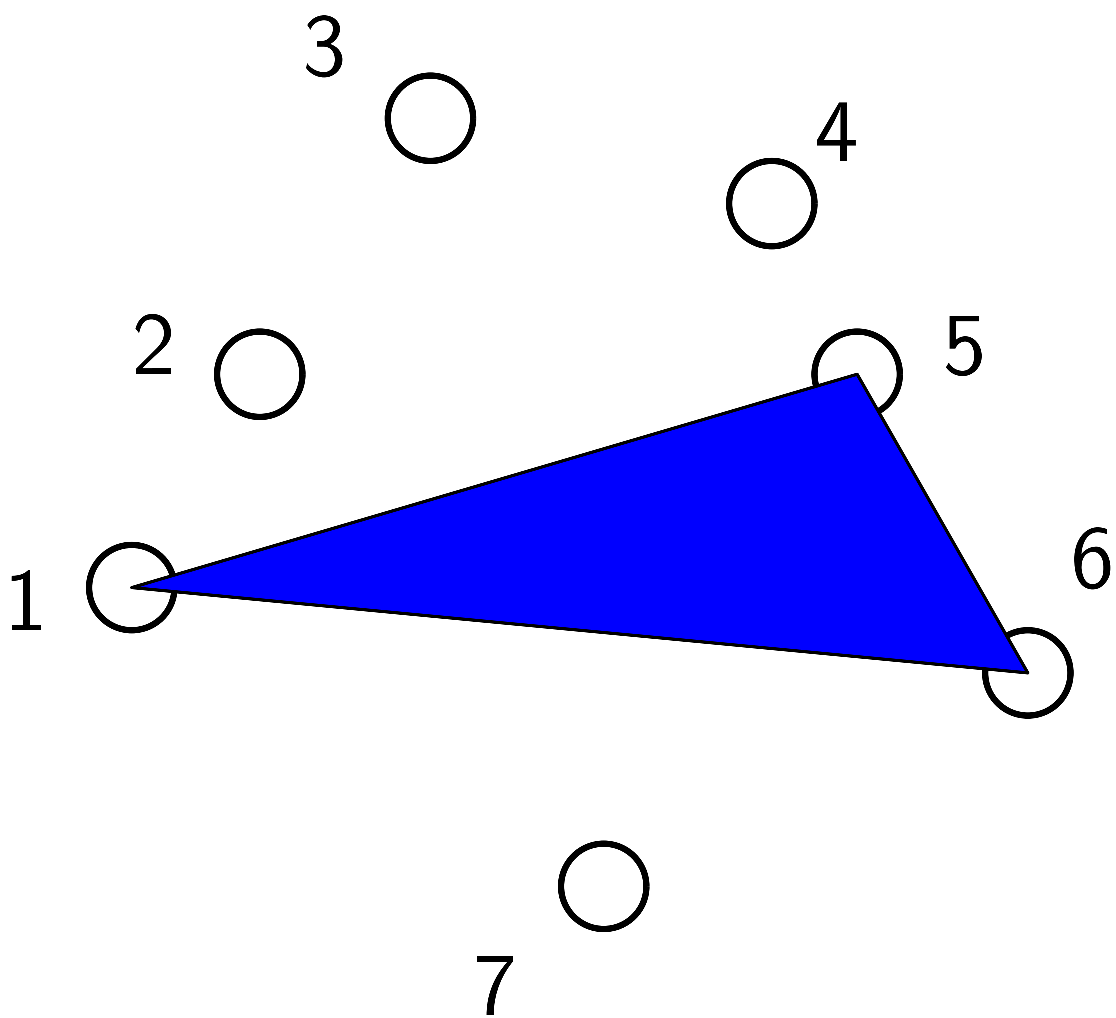
$1 - 2 - 7$: yes

$1 - 2 - 5$: yes

$1 - 2 - 3$: no

Counting Triangles

- 3 points form a triangle if and only if they are non collinear



1 – 2 – 7: yes

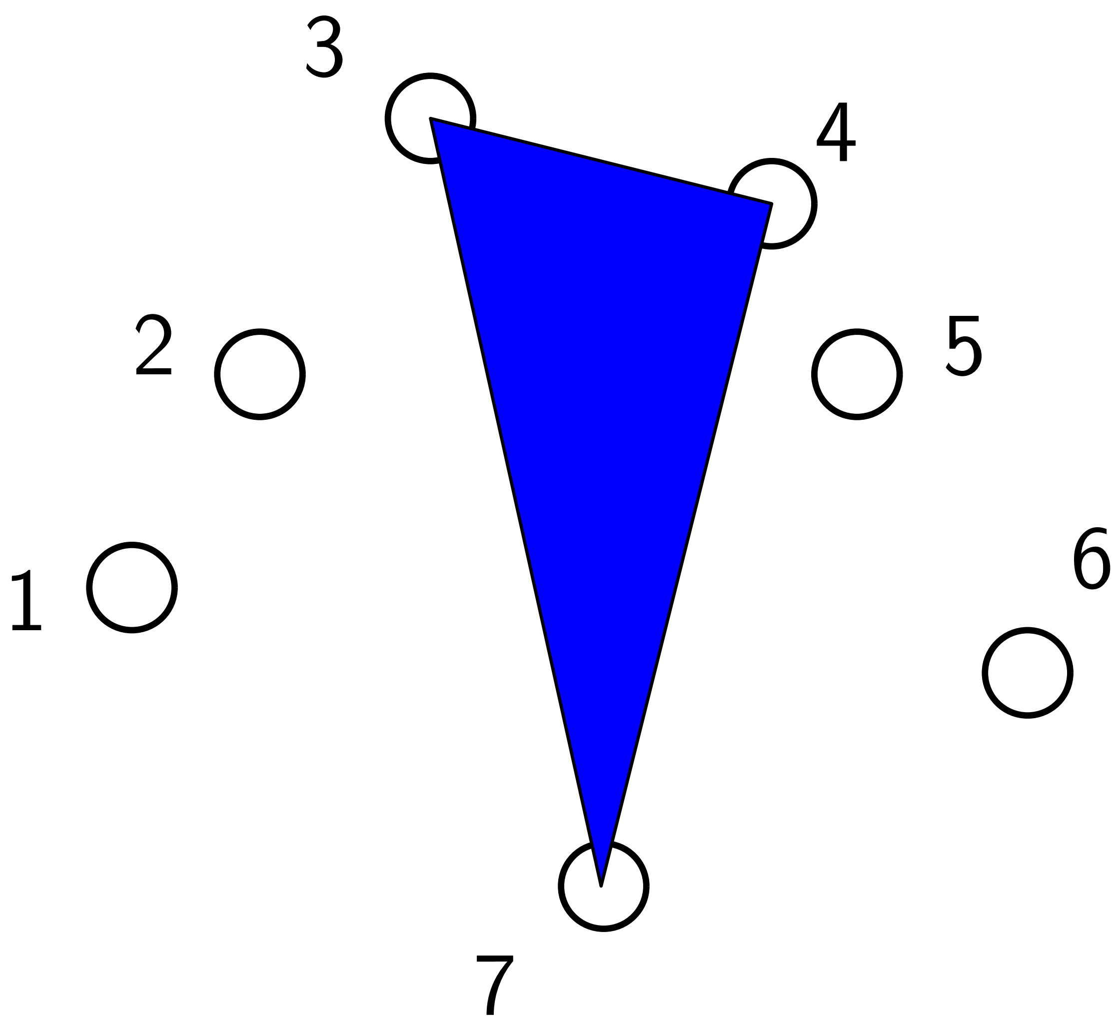
1 – 2 – 5: yes

1 – 2 – 3: no

1 – 5 – 6: yes

Counting Triangles

- 3 points form a triangle if and only if they are non collinear



$1 - 2 - 7$: yes

$1 - 2 - 5$: yes

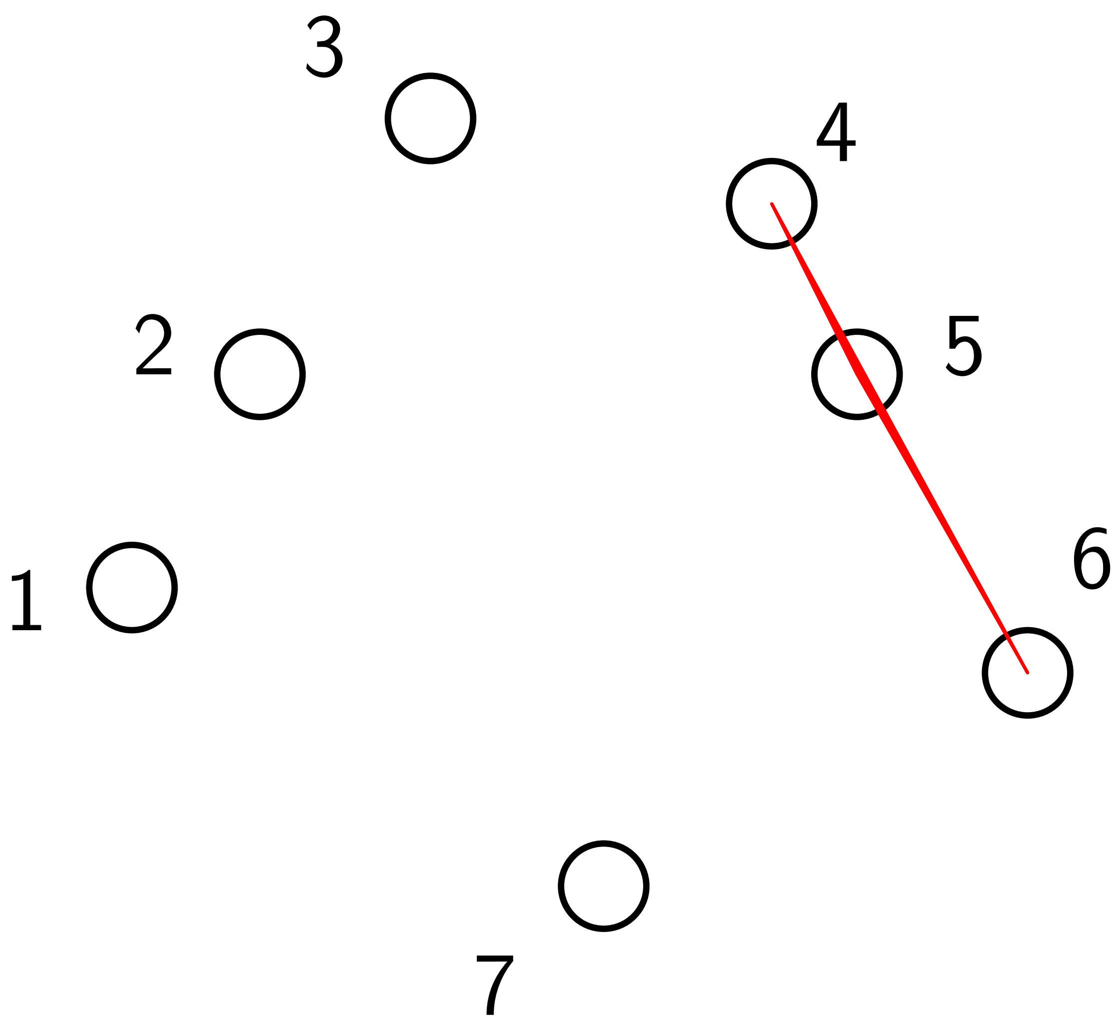
$1 - 2 - 3$: no

$1 - 5 - 6$: yes

$3 - 4 - 7$: yes

Counting Triangles

- 3 points form a triangle if and only if they are non collinear



1 – 2 – 7: yes

1 – 2 – 5: yes

1 – 2 – 3: no

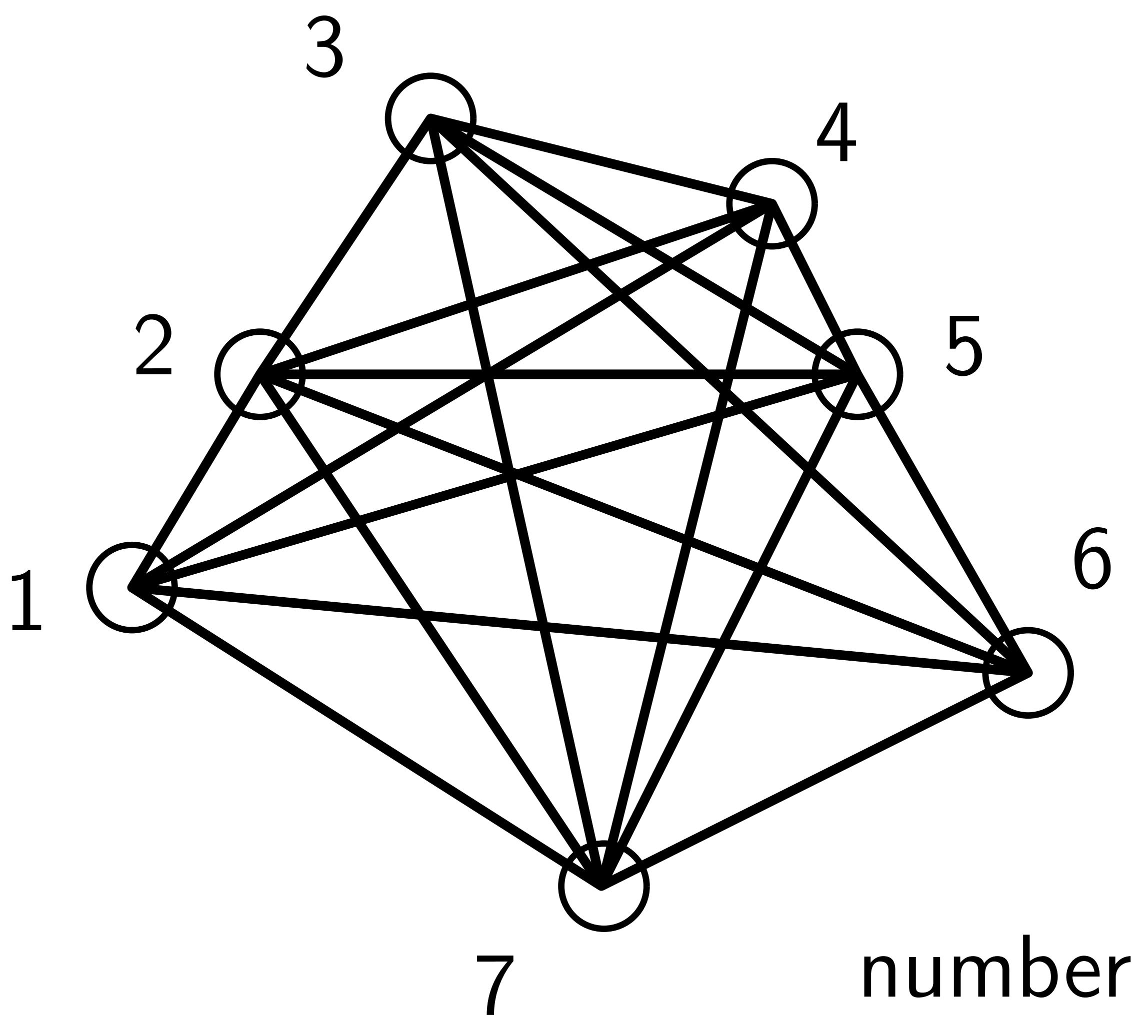
1 – 5 – 6: yes

3 – 4 – 7: yes

4 – 5 – 6: no

Counting Triangles

- 3 points form a triangle if and only if they are non collinear



1 – 2 – 7: yes

1 – 2 – 5: yes

1 – 2 – 3: no

1 – 5 – 6: yes

3 – 4 – 7: yes

4 – 5 – 6: no

number of triangles: 33

Counting Triangles

```
triaglecount = 0
  for i = 1 to n
    for j = i+1 to n
      for k = j+1 to n
        if points i, j, k are not collinear
          trianglecount = trianglecount + 1
```

Counting Triangles

```
(1) trianglecount = 0
(2)   for i = 1 to n
(3)     for j = i+1 to n
(4)       for k = j+1 to n
(5)         if points i, j, k are not collinear
(6)           trianglecount = trianglecount + 1
```

A loop

Counting Triangles

```
trianglecount = 0
  for i = 1 to n
    for j = i+1 to n
      for k = j+1 to n
        if points i, j, k are not collinear
          trianglecount = trianglecount + 1
```

A loop embedded in a loop

Counting Triangles

```
triangleCount = 0
for i = 1 to n
  for j = i+1 to n
    for k = j+1 to n
      if points i, j, k are not collinear
        triangleCount = triangleCount + 1
```

A loop embedded in a loop embedded in another loop.

Counting Triangles

```
(1) trianglecount = 0
(2)    for i = 1 to n
(3)       for j = i+1 to n
(4)          for k = j+1 to n
(5)             if points i, j, k are not collinear
(6)                trianglecount = trianglecount + 1
```

A loop embedded in a loop embedded in another loop.

Second loop begins with $j = i + 1$ and j increases up to n.
Third loop begins with $k = j + 1$ and k increases up to n.

Counting Triangles

```
triangle count = 0
  for i = 1 to n
    for j = i+1 to n
      for k = j+1 to n
        if points i, j, k are not collinear
          trianglecount = trianglecount + 1
```

A loop embedded in a loop embedded in another loop.

Second loop begins with $j = i + 1$ and j increases up to n.
Third loop begins with $k = j + 1$ and k increases up to n.

Thus each triple i, j, k with $i < j < k$ is examined ~~exactly once~~.

Counting Triangles

```
triangleconut = 0
```

```
triangleconut = 0
  for i = 1 to n
    for j = i+1 to n
      for k = j+1 to n
        if points i, j, k are not collinear
          trianglecount = trianglecount + 1
```

A loop embedded in a loop embedded in another loop.

Second loop begins with $j = i + 1$ and j increases up to n.
Third loop begins with $k = j + 1$ and k increases up to n.

Thus each triple i, j, k with $i < j < k$ is examined exactly once.

For example, if $n = 4$, then triples (i, j, k) used by algorithm are (1,2,3), (1,2,4), (1,3,4), and (2,3,4).

Counting Triangles

- Want to compute the number of
  increasing triples (i, j, k) with $1 \le i < j < k \le n$.

Counting Triangles

- Want to compute the number of
  increasing triples (i, j, k) with $1 \leq i < j < k \leq n$.

  Claim: Number of increasing triples is exactly the same as number of 3-element subsets from $\{1, 2, \ldots, n\}$

Counting Triangles

- Want to compute the number of *increasing triples* (i, j, k) with $1 \le i < j < k \le n$.

  Claim: Number of increasing triples is exactly the same as number of 3-element subsets from $\{1, 2, \ldots, n\}$

  Why? Let $X =$ set of increasing triples and $Y =$ set of 3-element subsets from $\{1, 2, \ldots, n\}$

Counting Triangles

- Want to compute the number of
  increasing triples (i, j, k) with $1 \le i < j < k \le n$.

  Claim: Number of increasing triples is exactly the same as number of 3-element subsets from $\{1, 2, \ldots, n\}$

  Why? Let $X =$ set of increasing triples and
  $Y =$ set of 3-element subsets from $\{1, 2, \ldots, n\}$

  Define: $f : X \to Y$ by $f((i, j, k)) = \{i, j, k\}$
  Claim: f is a **bijection** (why) so $|X| = |Y|$

Counting Triangles

- Want to compute the number of *increasing triples* (i, j, k) with $1 \le i < j < k \le n$.

Claim: Number of increasing triples is exactly the same as number of 3-element subsets from $\{1, 2, \ldots, n\}$

Why? Let X = set of increasing triples and Y = set of 3-element subsets from $\{1, 2, \ldots, n\}$

Define: $f : X \to Y$ by $f((i, j, k)) = \{i, j, k\}$
Claim: f is a **bijection** (why) so $|X| = |Y|$

f is a bijection because
f is one-to-one
if $(i, j, k) \neq (i', j', k') \Rightarrow f(\,(i, j, k)\,) \neq f(\,(i', j', k'))$
f is onto
if γ is a 3-element subset then it can be written as $\gamma = \{i, j, k\}$ where $i < j < k$ so $f(\,(i, j, k)\,) = \gamma$.

Counting Pairs

- The number of

  increasing pairs (i, j) with $1 \leq i < j \leq n$

  is the same as the number of

  2-sets from $\{1, 2, \ldots, n\}$

Counting Pairs

- The number of
  increasing pairs (i, j) with $1 \leq i < j \leq n$
  is the same as the number of
  2-sets from $\{1, 2, \ldots, n\}$

  Define $f : X \to Y$ by $f(\,(i, j)\,) = \{i, j\}$
  Claim: f is a **bijection** so $|X| = |Y|$

Counting Pairs

- The number of

  increasing pairs (i, j) with $1 \leq i < j \leq n$

  is the same as the number of

  2-sets from $\{1, 2, \ldots, n\}$

  Define $f : X \to Y$ by $f(\,(i, j)\,) = \{i, j\}$

  Claim: f is a **bijection** so $|X| = |Y|$

  We actually already saw that $|X| = |Y| = \binom{n}{2}$

The Bijection Principle

- Two sets have the same size if and only if there is a one-to-one function from one set onto the other.

The Bijection Principle

- Two sets have the same size if and only if there is a one-to-one function from one set onto the other.

  A standard first step in counting the size of a set is to use a bijection to show that it has the same size as a 2nd set, and then count the 2nd set instead.

The Bijection Principle

- Two sets have the same size if and only if there is a one-to-one function from one set onto the other.

  A standard first step in counting the size of a set is to use a bijection to show that it has the same size as a 2nd set, and then count the 2nd set instead.

  In practice, in real problems we often only *implicitly* use the bijection and don't *explicitly* describe it

The Bijection Principle

- Two sets have the same size if and only if there is a one-to-one function from one set onto the other.

  A standard first step in counting the size of a set is to use a bijection to show that it has the same size as a 2nd set, and then count the 2nd set instead.

  In practice, in real problems we often only *implicitly* use the bijection and don't *explicitly* describe it

  Currently, we started with the problem of counting the $\#$ of increasing triples and changed it to the problem of counting the $\#$ of 3-element sets from $\{1, 2, \ldots, n\}$

Inclusion-Exclusion Principle

- Used in counts where the decomposition yields two independent counting tasks with overlapping elements

Inclusion-Exclusion Principle

- Used in counts where the decomposition yields two independent counting tasks with overlapping elements

  If we use the sum rule, some elements would be counted twice.

Inclusion-Exclusion Principle

- Used in counts where the decomposition yields two independent counting tasks with overlapping elements

  If we use the sum rule, some elements would be counted twice.

Inclusion-Exclusion Principle: uses a sum rule and then corrects for the overlapping elements.

Inclusion-Exclusion Principle

- Used in counts where the decomposition yields two independent counting tasks with overlapping elements

  If we use the sum rule, some elements would be counted twice.

  Inclusion-Exclusion Principle: uses a sum rule and then corrects for the overlapping elements.

$$|A \cup B| = |A| + |B| - |A \cap B|$$

- **Example**

  How many bit strings of length 8 either start with a '1' bit or end with the two bits '00'?

Inclusion-Exclusion Principle

- **Example**

  How many bit strings of length 8 either start with a '1' bit or end with the two bits '00'?

  - ◇ it is easy to count bit strings starting with '1':

- **Example**

  How many bit strings of length 8 either start with a '1' bit or end with the two bits '00'?

  - ◇ it is easy to count bit strings starting with '1': 2^7

Inclusion-Exclusion Principle

- **Example**

  How many bit strings of length 8 either start with a '1' bit or end with the two bits '00'?

  ◇ it is easy to count bit strings starting with '1': 2^7

  ◇ it is easy to count bit strings ending with '00':

Inclusion-Exclusion Principle

- **Example**

  How many bit strings of length 8 either start with a '1' bit or end with the two bits '00'?

  - ◇ it is easy to count bit strings starting with '1': 2^7
  - ◇ it is easy to count bit strings ending with '00': 2^6

Inclusion-Exclusion Principle

- **Example**

  How many bit strings of length 8 either start with a ‘1’ bit or end with the two bits ‘00’?

  ◇ it is easy to count bit strings starting with ‘1’: 2^7

  ◇ it is easy to count bit strings ending with ‘00’: 2^6

  Overcounting!!!

Inclusion-Exclusion Principle

- **Example**

  How many bit strings of length 8 either start with a '1' bit or end with the two bits '00'?

  ◇ it is easy to count bit strings starting with '1': 2^7

  ◇ it is easy to count bit strings ending with '00': 2^6

  Overcounting!!!

  ◇ deduct the number of strings starting with '1' and ending with "00":

Inclusion-Exclusion Principle

- **Example**

  How many bit strings of length 8 either start with a '1' bit or end with the two bits '00'?

  ◇ it is easy to count bit strings starting with '1': 2^7

  ◇ it is easy to count bit strings ending with '00': 2^6

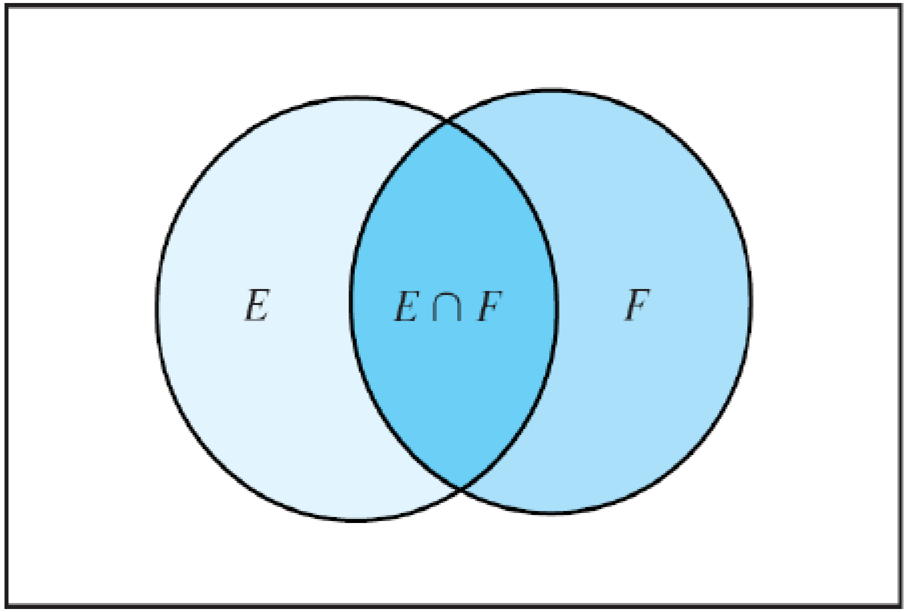
  Overcounting!!!

  ◇ deduct the number of strings starting with '1' and ending with "00": 2^5

Inclusion-Exclusion Principle

- Two sets

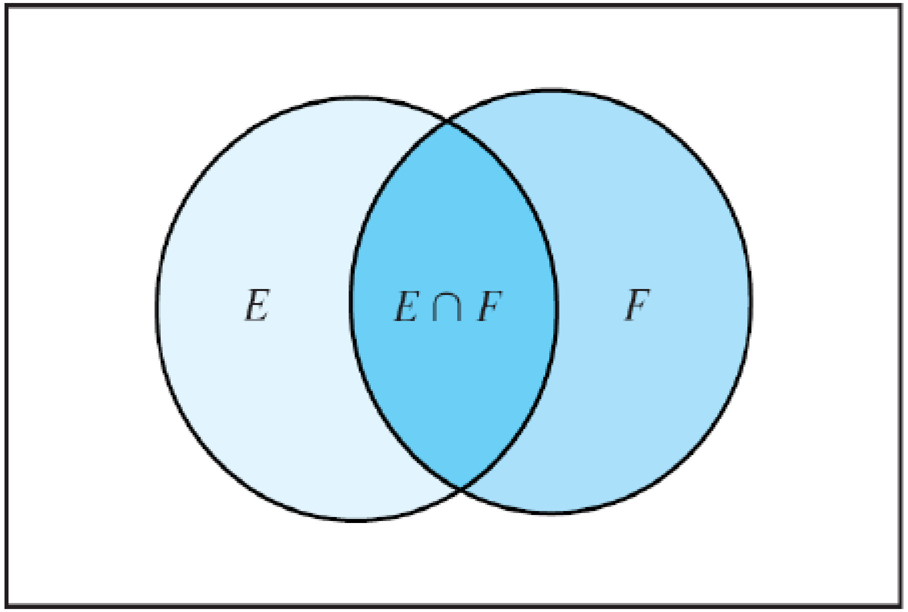
$$|E \cup F| = |E| + |F| - |E \cap F|$$



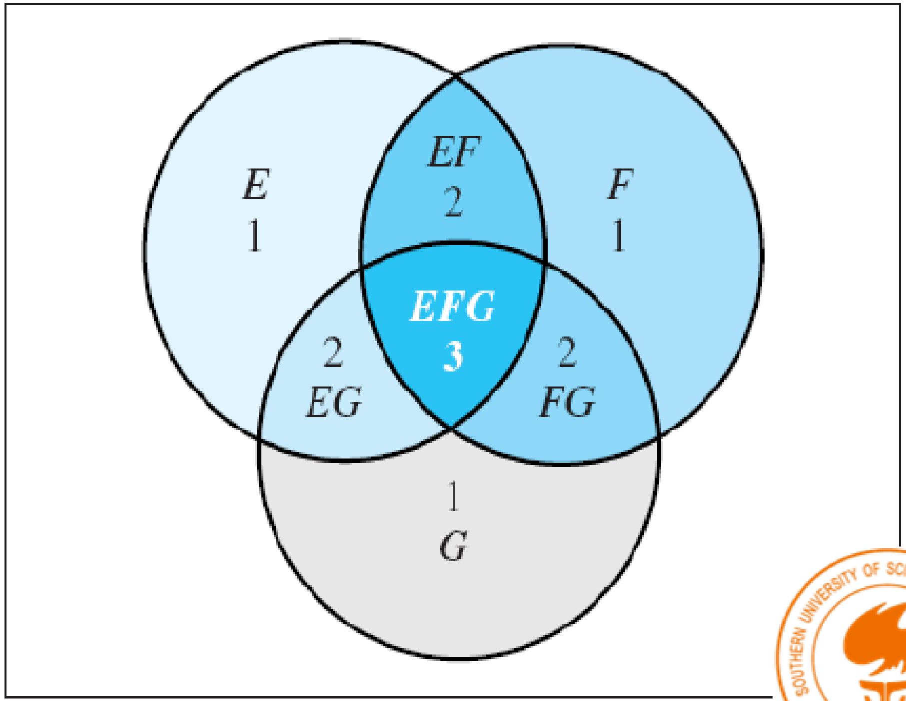
Inclusion-Exclusion Principle

- Two sets

$$|E \cup F| = |E| + |F| - |E \cap F|$$



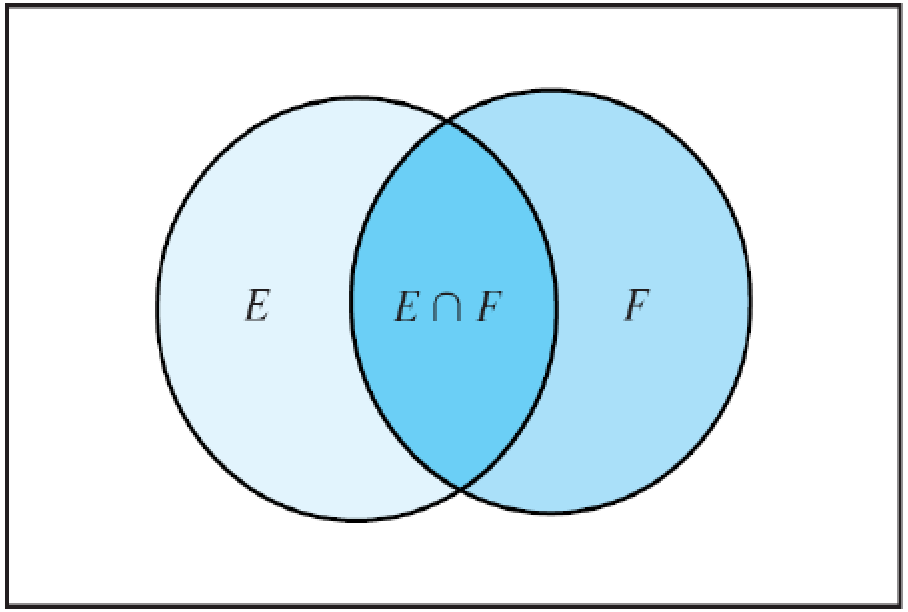
Three sets



Inclusion-Exclusion Principle

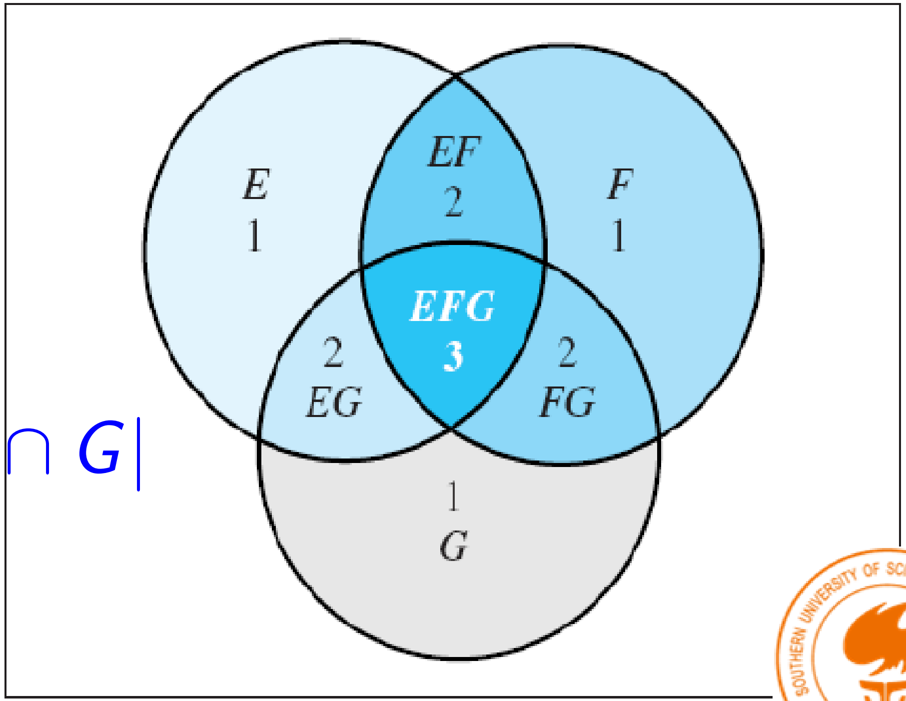
- Two sets

$$|E \cup F| = |E| + |F| - |E \cap F|$$



Three sets

$$\begin{aligned}&|E \cup F \cup G|\\ &= |E| + |F| + |G|\\ &\quad -|E \cap F| - |E \cap G| - |F \cap G|\\ &\quad +|E \cap F \cap G|\end{aligned}$$



Inclusion-Exclusion Principle

- $$|\cup_{i=1}^{n} E_i| = \sum_{k=1}^{n} (-1)^{k+1} \sum_{1 \leq i_1 < i_2 < \cdots < i_k \leq n} |E_{i_1} \cap E_{i_2} \cap \cdots \cap E_{i_k}|$$

Inclusion-Exclusion Principle

- $$|\cup_{i=1}^{n} E_i| = \sum_{k=1}^{n} (-1)^{k+1} \sum_{1 \le i_1 < i_2 < \cdots < i_k \le n} |E_{i_1} \cap E_{i_2} \cap \cdots \cap E_{i_k}|$$

Proof by induction

Inclusion-Exclusion Principle

- $$|\cup_{i=1}^{n} E_i| = \sum_{k=1}^{n} (-1)^{k+1} \sum_{1 \le i_1 < i_2 < \cdots < i_k \le n} |E_{i_1} \cap E_{i_2} \cap \cdots \cap E_{i_k}|$$

Proof by induction

Base case $(n = 2)$

$$|E \cup F| = |E| + |F| - |E \cap F|$$

Inclusion-Exclusion Principle

- $$|\cup_{i=1}^{n} E_i| = \sum_{k=1}^{n} (-1)^{k+1} \sum_{1 \le i_1 < i_2 < \cdots < i_k \le n} |E_{i_1} \cap E_{i_2} \cap \cdots \cap E_{i_k}|$$

Proof by induction

Base case $(n = 2)$

$$|E \cup F| = |E| + |F| - |E \cap F|$$

Inductive Hypothesis

$$\left|\cup_{i=1}^{n-1} E_i\right| = \sum_{k=1}^{n-1} (-1)^{k+1} \sum_{1 \le i_1 < i_2 < \cdots < i_k \le n-1} |E_{i_1} \cap E_{i_2} \cap \cdots \cap E_{i_k}|$$

Inclusion-Exclusion Principle

- Inductive step

  Set $E = E_1 \cup \cdots \cup E_{n-1}$, and $F = E_n$.

Inclusion-Exclusion Principle

- Inductive step

  Set $E = E_1 \cup \cdots \cup E_{n-1}$, and $F = E_n$.

  By $|E \cup F| = |E| + |F| - |E \cap F|$

Inclusion-Exclusion Principle

- Inductive step

  Set $E = E_1 \cup \cdots \cup E_{n-1}$, and $F = E_n$.

  By $|E \cup F| = |E| + |F| - |E \cap F|$

$$|\cup_{i=1}^{n} E_i| = \left|\cup_{i=1}^{n-1} E_i\right| + |E_n| - \left|\left(\cup_{i=1}^{n-1} E_i\right) \cap E_n\right|$$

Inclusion-Exclusion Principle

- Inductive step

  Set $E = E_1 \cup \cdots \cup E_{n-1}$, and $F = E_n$.

  By $|E \cup F| = |E| + |F| - |E \cap F|$

$$|\cup_{i=1}^{n} E_i| = \boxed{|\cup_{i=1}^{n-1} E_i|} + |E_n| - \left|\left(\cup_{i=1}^{n-1} E_i\right) \cap E_n\right|$$

  The first term is given by i.h.

Inclusion-Exclusion Principle

- Inductive step

  Set $E = E_1 \cup \cdots \cup E_{n-1}$, and $F = E_n$.

  By $|E \cup F| = |E| + |F| - |E \cap F|$

  $$|\cup_{i=1}^{n} E_i| = \boxed{|\cup_{i=1}^{n-1} E_i|} + |E_n| - \boxed{\left|(\cup_{i=1}^{n-1} E_i) \cap E_n\right|}$$

  The first term is given by i.h.

  For the third term, by distributive law,

  $$\left|(\cup_{i=1}^{n-1} E_i) \cap E_n\right| = \left|\cup_{i=1}^{n-1} (E_i \cap E_n)\right| = \left|\cup_{i=1}^{n-1} G_i\right|$$

  where $G_i = E_i \cap E_n$.

Inclusion-Exclusion Principle

- So far

$$\left|\cup_{i=1}^{n} E_i\right| = \left|\cup_{i=1}^{n-1} E_i\right| + |E_n| - \left|\cup_{i=1}^{n-1} G_i\right|$$

where $G_i = E_i \cap E_n$.

Inclusion-Exclusion Principle

- So far

$$\left|\cup_{i=1}^{n} E_i\right| = \left|\cup_{i=1}^{n-1} E_i\right| + |E_n| - \left|\cup_{i=1}^{n-1} G_i\right|$$

where $G_i = E_i \cap E_n$.

Note that (why?)

$$-(-1)^{k+1}|G_{i_1} \cap G_{i_2} \cap \cdots \cap G_{i_k}|$$
$$= (-1)^{k+2}|E_{i_1} \cap E_{i_2} \cap \cdots \cap E_{i_k} \cap E_n|$$

Inclusion-Exclusion Principle

- So far
$$\left|\cup_{i=1}^{n} E_i\right| = \left|\cup_{i=1}^{n-1} E_i\right| + |E_n| - \left|\cup_{i=1}^{n-1} G_i\right|$$
where $G_i = E_i \cap E_n$.

Note that (why?)
$$-(-1)^{k+1}|G_{i_1} \cap G_{i_2} \cap \cdots \cap G_{i_k}|$$
$$= (-1)^{k+2}|E_{i_1} \cap E_{i_2} \cap \cdots \cap E_{i_k} \cap E_n|$$

Some discussion:
first summation sums $(-1)^{k+1}|E_{i_1} \cap E_{i_2} \cap \cdots \cap E_{i_k}|$ over all lists $i_1, i_2, \ldots, i_k$ that do not contain n
$|E_n|$ and second summation together sum $(-1)^{k+1}|E_{i_1} \cap E_{i_2} \cap \cdots \cap E_{i_k}|$ over all lists $i_1, i_2, \ldots, i_k$ that do contain n

Inclusion-Exclusion Principle

- $$|\cup_{i=1}^{n} E_i| = \sum_{k=1}^{n} (-1)^{k+1} \sum_{1 \le i_1 < i_2 < \cdots < i_k \le n} |E_{i_1} \cap E_{i_2} \cap \cdots \cap E_{i_k}|$$

Inclusion-Exclusion Principle

- $$|\cup_{i=1}^{n} E_i| = \sum_{k=1}^{n} (-1)^{k+1} \sum_{1 \leq i_1 < i_2 < \cdots < i_k \leq n} |E_{i_1} \cap E_{i_2} \cap \cdots \cap E_{i_k}|$$

  This can be used to determine the number of onto functions

Inclusion-Exclusion Principle

- $$|\cup_{i=1}^{n} E_i| = \sum_{k=1}^{n} (-1)^{k+1} \sum_{1 \le i_1 < i_2 < \cdots < i_k \le n} |E_{i_1} \cap E_{i_2} \cap \cdots \cap E_{i_k}|$$

  This can be used to determine the number of onto functions

  A, B are two sets with $|A| = m$ and $|B| = n$.

Inclusion-Exclusion Principle

- $$|\cup_{i=1}^{n} E_i| = \sum_{k=1}^{n} (-1)^{k+1} \sum_{1 \le i_1 < i_2 < \cdots < i_k \le n} |E_{i_1} \cap E_{i_2} \cap \cdots \cap E_{i_k}|$$

  This can be used to determine the number of onto functions

  A, B are two sets with $|A| = m$ and $|B| = n$.

  (a) How many onto functions are there from A to B?

  (b) How many functions are there from A to B that map nothing to at least one element of B?

Inclusion-Exclusion Principle

- $$|\cup_{i=1}^{n} E_i| = \sum_{k=1}^{n}(-1)^{k+1} \sum_{1 \le i_1 < i_2 < \cdots < i_k \le n} |E_{i_1} \cap E_{i_2} \cap \cdots \cap E_{i_k}|$$

This can be used to determine the number of onto functions

A, B are two sets with $|A| = m$ and $|B| = n$.

(a) How many onto functions are there from A to B?

(b) How many functions are there from A to B that map nothing to at least one element of B?

$$\#(a) + \#(b) = n^m$$

Inclusion-Exclusion Principle

- This can be used to determine the number of onto functions

  A, B are two sets with $|A| = m$ and $|B| = n$.

  (a) How many onto functions are there from A to B?

  (b) How many functions are there from A to B that map nothing to at least one element of B?

$$\#(a) + \#(b) = n^m$$

Inclusion-Exclusion Principle

- This can be used to determine the number of onto functions

  A, B are two sets with $|A| = m$ and $|B| = n$.

  (a) How many onto functions are there from A to B?

  (b) How many functions are there from A to B that map nothing to at least one element of B?

  $$\#(a) + \#(b) = n^m$$

  Set E_i – set of functions that map nothing to element i of B

- This can be used to determine the number of onto functions

  A, B are two sets with $|A| = m$ and $|B| = n$.

  (a) How many onto functions are there from A to B?

  (b) How many functions are there from A to B that map nothing to at least one element of B?

  $$\#(a) + \#(b) = n^m$$

  Set E_i – set of functions that map nothing to element i of B

  $\#(b) = |\cup_{i=1}^{n} E_i|$

Inclusion-Exclusion Principle

- This can be used to determine the number of onto functions

  A, B are two sets with $|A| = m$ and $|B| = n$.

  (a) How many onto functions are there from A to B?

  (b) How many functions are there from A to B that map nothing to at least one element of B?

$$\#(a) + \#(b) = n^m$$

  Set E_i – set of functions that map nothing to element i of B

  $\#(b) = |\cup_{i=1}^{n} E_i|$

  $= \sum_{k=1}^{n}(-1)^{k+1} \sum_{1 \le i_1 < i_2 < \cdots < i_k \le n} |E_{i_1} \cap E_{i_2} \cap \cdots \cap E_{i_k}|$

Inclusion-Exclusion Principle

- This can be used to determine the number of onto functions

  A, B are two sets with $|A| = m$ and $|B| = n$.

  (a) How many onto functions are there from A to B?

  (b) How many functions are there from A to B that map nothing to at least one element of B?

$$\#(a) + \#(b) = n^m$$

Set E_i – set of functions that map nothing to element i of B

$\#(b) = |\cup_{i=1}^{n} E_i|$

$= \sum_{k=1}^{n} (-1)^{k+1} \sum_{1 \le i_1 < i_2 < \cdots < i_k \le n} |E_{i_1} \cap E_{i_2} \cap \cdots \cap E_{i_k}|$

$= \sum_{k=1}^{n} (-1)^{k+1} \binom{n}{k} (n-k)^m$

Next Lecture

- counting II ...