Advanced Counting Techniques

CS1200, CSE IIT Madras

Meghana Nasre

April 6, 2020

- Principle of Inclusion-Exclusion
- Recurrences and its applications
- Solving Recurrences

Principle of Inclusion Exclusion

Principle of Inclusion Exclusion gives a formula to find the size of union of **finite** sets.

It is a generalization of the familiar formula below.

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

Ex: Write down the formula for $|A \cup B \cup C|$.

Principle of Inclusion Exclusion: Let $A_1, A_2, \ldots, A_n$ be n **finite** sets. Then,

$$\begin{aligned} |A_1 \cup A_2 \cup \ldots \cup A_n| = \quad & \sum_{i=1}^{n} |A_i| - \sum_{1 \le i < j \le n} |A_i \cap A_j| \\ + \quad & \sum_{1 \le i < j < k \le n} |A_i \cap A_j \cap A_k| + \ldots \\ + \quad & (-1)^{n+1} |A_1 \cap A_2 \cap \ldots \cap A_n| \end{aligned}$$

Example: Number of Derangements

We are given n objects, say $1, 2, 3, \ldots, n$.

We are given n objects, say $1, 2, 3, \ldots, n$.

Derangement: A permutation such that no object is in its original position.

We are given n objects, say $1, 2, 3, \ldots, n$.

Derangement: A permutation such that no object is in its original position.

Input: $1, 2, 3, 4, 5$

$3, 1, 2, 5, 4$

We are given n objects, say $1, 2, 3, \ldots, n$.

Derangement: A permutation such that no object is in its original position.

Input: $1, 2, 3, 4, 5$

$3, 1, 2, 5, 4$ ✓

We are given n objects, say $1, 2, 3, \ldots, n$.

Derangement: A permutation such that no object is in its original position.

Input: $1, 2, 3, 4, 5$

$3, 1, 2, 5, 4$ ✓

$4, 1, 3, 5, 2$

We are given n objects, say $1, 2, 3, \ldots, n$.

Derangement: A permutation such that no object is in its original position.

Input: $1, 2, 3, 4, 5$

$3, 1, 2, 5, 4$	✓	
$4, 1, 3, 5, 2$	×	since 3 is at its original position

Example: Number of Derangements

We are given n objects, say $1, 2, 3, \ldots, n$.

Derangement: A permutation such that no object is in its original position.

Input: $1, 2, 3, 4, 5$

$3, 1, 2, 5, 4$	✓	
$4, 1, 3, 5, 2$	×	since 3 is at its original position

Goal: Count the number of derangements of n objects, denote it by D_n.

We are given n objects, say $1, 2, 3, \ldots, n$.

Derangement: A permutation such that no object is in its original position.

Input: $1, 2, 3, 4, 5$

$3, 1, 2, 5, 4$ ✓

$4, 1, 3, 5, 2$ × since 3 is at its original position

Goal: Count the number of derangements of n objects, denote it by D_n.

Ex:

- Find out D_2 and D_3.
- Cast D_n as properties that we wish to avoid.

Example: Number of Derangements

Derangement: A permutation of n objects s.t. no object is in its original position.

P_i: A permutation that has i at its original position.

Derangement: A permutation of n objects s.t. no object is in its original position.

P_i: A permutation that has i at its original position.

$$2, 3, 4, 1, 5 \text{ satisfies } P_5.$$
$$1, 2, 3, 4, 5 \text{ satisfies } P_5.$$

Example: Number of Derangements

Derangement: A permutation of n objects s.t. no object is in its original position.

P_i: A permutation that has i at its original position.

$$2, 3, 4, 1, 5 \text{ satisfies } P_5.$$
$$1, 2, 3, 4, 5 \text{ satisfies } P_5.$$

A_i: Set of permutations that have i at its original position.

Example: Number of Derangements

Derangement: A permutation of n objects s.t. no object is in its original position.

P_i: A permutation that has i at its original position.

$$2, 3, 4, 1, 5 \text{ satisfies } P_5.$$
$$1, 2, 3, 4, 5 \text{ satisfies } P_5.$$

A_i: Set of permutations that have i at its original position.

$A_1 \cup A_2 \cup \ldots \cup A_n$: Set of permutations that have at least one object at its original position.

Example: Number of Derangements

Derangement: A permutation of n objects s.t. no object is in its original position.

P_i: A permutation that has i at its original position.

$$2, 3, 4, 1, 5 \text{ satisfies } P_5.$$
$$1, 2, 3, 4, 5 \text{ satisfies } P_5.$$

A_i: Set of permutations that have i at its original position.

$A_1 \cup A_2 \cup \ldots \cup A_n$: Set of permutations that have at least one object at its original position.

Modified Goal: Compute the size of $A_1 \cup A_2 \cup \ldots \cup A_n$.

Number of Derangements:

$$D_n = n! - |A_1 \cup A_2 \cup \ldots \cup A_n|$$

Ex: Complete the above to show that the number of derangements is

$$D_n = n! \left[1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \ldots + (-1)^n \frac{1}{n!}\right]$$

Number of Derangements: A recursive specification

Now we give a recursive definition of D_n.

$$
\begin{aligned}
D_1 &= 0; \qquad D_2 = 1 \\
D_n &= (n-1)(D_{n-1} + D_{n-2})
\end{aligned}
$$

Number of Derangements: A recursive specification

Now we give a recursive definition of D_n.

$$\begin{aligned} D_1 &= 0; \qquad D_2 = 1 \\ D_n &= (n-1)(D_{n-1} + D_{n-2}) \end{aligned}$$

We give a combinatorial proof of the above. Verify base cases.

Number of Derangements: A recursive specification

Now we give a recursive definition of D_n.

$$\begin{aligned} D_1 &= 0; \qquad D_2 = 1 \\ D_n &= (n-1)(D_{n-1} + D_{n-2}) \end{aligned}$$

We give a combinatorial proof of the above. Verify base cases.

Think of a derangement as n objects to be placed in n positions with each position having exactly one forbidden object.

Number of Derangements: A recursive specification

Now we give a recursive definition of D_n.

$$\begin{aligned} D_1 &= 0; \qquad D_2 = 1 \\ D_n &= (n-1)(D_{n-1} + D_{n-2}) \end{aligned}$$

We give a combinatorial proof of the above. Verify base cases.

Think of a derangement as n objects to be placed in n positions with each position having exactly one forbidden object.

Number of choices for first position $= (n-1)$.

Number of Derangements: A recursive specification

Now we give a recursive definition of D_n.

$$\begin{aligned} D_1 &= 0; \qquad D_2 = 1 \\ D_n &= (n-1)(D_{n-1} + D_{n-2}) \end{aligned}$$

We give a combinatorial proof of the above. Verify base cases.

Think of a derangement as n objects to be placed in n positions with each position having exactly one forbidden object.

Number of choices for first position $= (n-1)$.

Every derangement starting at k where $2 \leq k \leq n-1$ can be categorized as:

Number of Derangements: A recursive specification

Now we give a recursive definition of D_n.

$$\begin{aligned} D_1 &= 0; \qquad D_2 = 1 \\ D_n &= (n-1)(D_{n-1} + D_{n-2}) \end{aligned}$$

We give a combinatorial proof of the above. Verify base cases.

Think of a derangement as n objects to be placed in n positions with each position having exactly one forbidden object.

Number of choices for first position $= (n-1)$.

Every derangement starting at k where $2 \leq k \leq n-1$ can be categorized as:

- Object 1 is at position k. In this case, we can omit the two positions 1 and k and the two objects and consider derangements on $n-2$ positions with each position having one forbidden object. The number of such derangements is D_{n-2}.

Number of Derangements: A recursive specification

Now we give a recursive definition of D_n.

$$
\begin{aligned}
D_1 &= 0; \qquad D_2 = 1 \\
D_n &= (n-1)(D_{n-1} + D_{n-2})
\end{aligned}
$$

We give a combinatorial proof of the above. Verify base cases.

Think of a derangement as n objects to be placed in n positions with each position having exactly one forbidden object.

Number of choices for first position $= (n-1)$.

Every derangement starting at k where $2 \leq k \leq n-1$ can be categorized as:

- Object 1 is at position k. In this case, we can omit the two positions 1 and k and the two objects and consider derangements on $n-2$ positions with each position having one forbidden object. The number of such derangements is D_{n-2}.
- Object 1 is not at position k. In this case, we have $n-1$ objects and $n-1$ positions. Each position has exactly one forbidden object. Note that position k cannot have object 1 now. Thus number of such derangements is D_{n-1}.

Number of Derangements: A recursive specification

Now we give a recursive definition of D_n.

$$\begin{aligned} D_1 &= 0; \qquad D_2 = 1 \\ D_n &= (n-1)(D_{n-1} + D_{n-2}) \end{aligned}$$

We give a combinatorial proof of the above. Verify base cases.

Think of a derangement as n objects to be placed in n positions with each position having exactly one forbidden object.

Number of choices for first position $= (n-1)$.

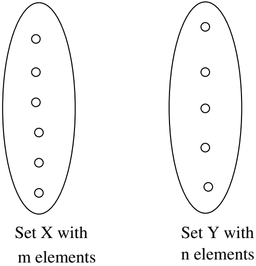
Every derangement starting at k where $2 \leq k \leq n-1$ can be categorized as:

- Object 1 is at position k. In this case, we can omit the two positions 1 and k and the two objects and consider derangements on $n-2$ positions with each position having one forbidden object. The number of such derangements is D_{n-2}.
- Object 1 is not at position k. In this case, we have $n-1$ objects and $n-1$ positions. Each position has exactly one forbidden object. Note that position k cannot have object 1 now. Thus number of such derangements is D_{n-1}.

This completes the proof.

Example: Number of Functions

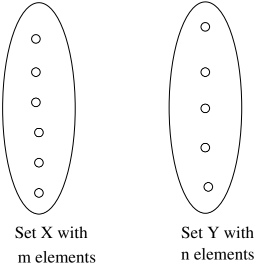
Number of functions from a set with m elements to a set with n elements.



- Total number of functions from X to Y :

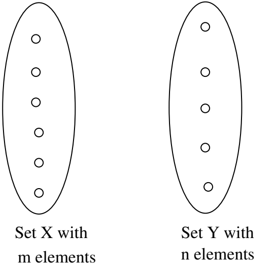
Example: Number of Functions

Number of functions from a set with m elements to a set with n elements.



- Total number of functions from X to Y : n^m

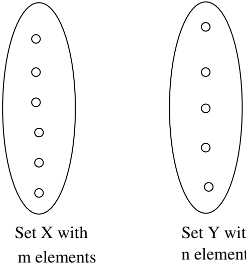
Number of functions from a set with m elements to a set with n elements.



- Total number of functions from X to Y : n^m
- Number of one-to-one functions from X to Y :

Example: Number of Functions

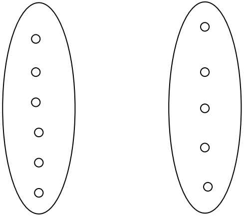
Number of functions from a set with m elements to a set with n elements.



- Total number of functions from X to Y : n^m
- Number of one-to-one functions from X to Y : $n \cdot (n-1) \cdot (n-2) \cdots (n-m+1)$

Example: Number of Functions

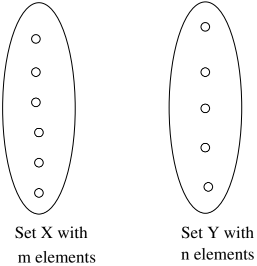
Number of functions from a set with m elements to a set with n elements.



- Total number of functions from X to Y : n^m
- Number of one-to-one functions from X to Y : $n \cdot (n-1) \cdot (n-2) \cdots (n-m+1)$ (requires $m \leq n$)

Example: Number of Functions

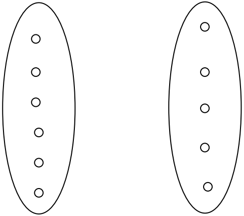
Number of functions from a set with m elements to a set with n elements.



- Total number of functions from X to Y : n^m
- Number of one-to-one functions from X to Y : $n \cdot (n-1) \cdot (n-2) \cdots (n-m+1)$ (requires $m \leq n$)
- Goal: Number of onto functions from X to Y.

Example: Number of Functions

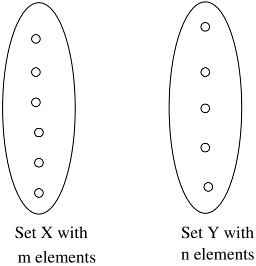
Number of functions from a set with m elements to a set with n elements.



- Total number of functions from X to Y : n^m
- Number of one-to-one functions from X to Y :
  $n \cdot (n-1) \cdot (n-2) \cdots (n-m+1)$
  (requires $m \leq n$)
- Goal: Number of onto functions from X to Y.
  (requires $m \geq n$)

Example: Number of Functions

Number of functions from a set with m elements to a set with n elements.

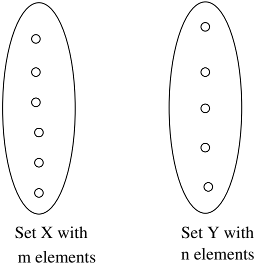


- Total number of functions from X to Y : n^m
- Number of one-to-one functions from X to Y : $n \cdot (n-1) \cdot (n-2) \cdots (n-m+1)$ (requires $m \leq n$)
- Goal: Number of onto functions from X to Y. (requires $m \geq n$)

Try counting without the principle of inclusion exclusion.

Example: Number of Functions

Number of functions from a set with m elements to a set with n elements.

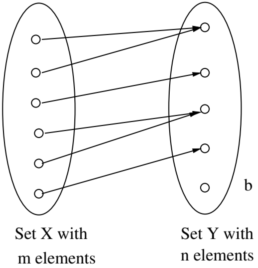


- Total number of functions from X to Y : n^m
- Number of one-to-one functions from X to Y : $n \cdot (n-1) \cdot (n-2) \cdots (n-m+1)$ (requires $m \leq n$)
- Goal: Number of onto functions from X to Y. (requires $m \geq n$)

Try counting without the principle of inclusion exclusion.

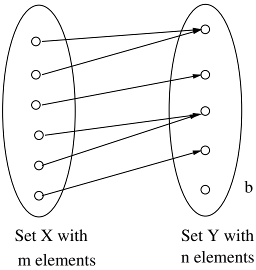
Can we cast it as certain properties that we wish to avoid?

Number of onto functions from a set with m elements to a set with n elements.



- Not an onto function. b_n is not in the range of the function.

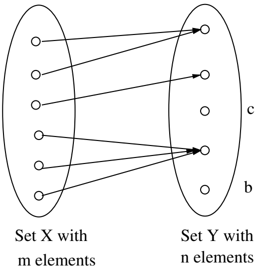
Number of onto functions from a set with m elements to a set with n elements.



- Not an onto function. b_n is not in the range of the function.
- This is precisely something that we would like to avoid.

Example: Number of Onto Functions

Number of onto functions from a set with m elements to a set with n elements.



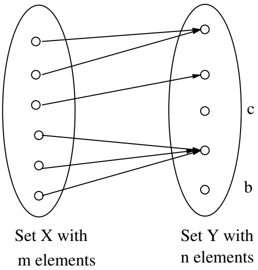
- Not an onto function. b_n is not in the range of the function.
- This is precisely something that we would like to avoid.

Lets label the elements in Y as $b_1, b_2, \ldots, b_n$.

- A_i: Set of functions from X to Y such that b_i is not in the range. note some other b's may not be in the range as well.

Example: Number of Onto Functions

Number of onto functions from a set with m elements to a set with n elements.



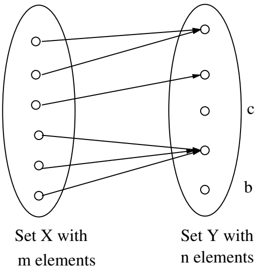
- Not an onto function. b_n is not in the range of the function.
- This is precisely something that we would like to avoid.

Lets label the elements in Y as $b_1, b_2, \ldots, b_n$.

- A_i: Set of functions from X to Y such that b_i is not in the range. note some other b's may not be in the range as well.
- $A_i \cap A_j \cap A_k$: Set of functions from X to Y that do have b_i, b_j and b_k in the range.

Example: Number of Onto Functions

Number of onto functions from a set with m elements to a set with n elements.



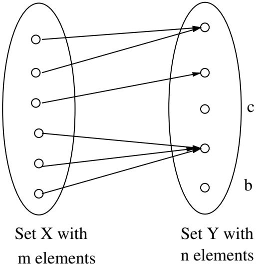
- Not an onto function. b_n is not in the range of the function.
- This is precisely something that we would like to avoid.

Lets label the elements in Y as $b_1, b_2, \ldots, b_n$.

- A_i: Set of functions from X to Y such that b_i is not in the range. note some other b's may not be in the range as well.
- $A_i \cap A_j \cap A_k$: Set of functions from X to Y that do have b_i, b_j and b_k in the range.
- $A_1 \cup A_2 \cup \ldots \cup A_n$: Set of functions from X to Y that leave at least one $b \in Y$ is not in the range.

Example: Number of Onto Functions

Number of onto functions from a set with m elements to a set with n elements.

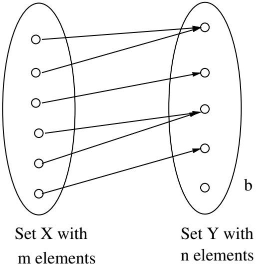


- Not an onto function. b_n is not in the range of the function.
- This is precisely something that we would like to avoid.

Lets label the elements in Y as $b_1, b_2, \ldots, b_n$.

- A_i: Set of functions from X to Y such that b_i is not in the range. note some other b's may not be in the range as well.
- $A_i \cap A_j \cap A_k$: Set of functions from X to Y that do have b_i, b_j and b_k in the range.
- $A_1 \cup A_2 \cup \ldots \cup A_n$: Set of functions from X to Y that leave at least one $b \in Y$ is not in the range. these are the functions we want to avoid!

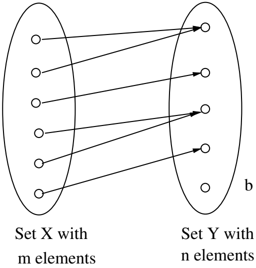
Number of onto functions from a set with m elements to a set with n elements.



- Not an onto function. b_n is not in the range of the function.

Example: Number of Onto Functions

Number of onto functions from a set with m elements to a set with n elements.



- Not an onto function. b_n is not in the range of the function.

Modified Goal: Compute the size of $A_1 \cup A_2 \cup \ldots \cup A_n$.

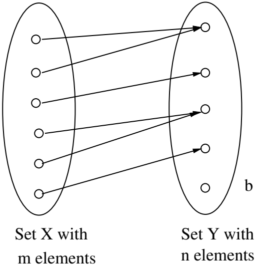
Number of Onto Functions:

$$n^m - |A_1 \cup A_2 \cup \ldots \cup A_n|$$

Example: Number of Onto Functions

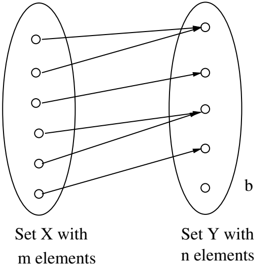
Number of onto functions from a set with m elements to a set with n elements.

Modified Goal: Compute $|A_1 \cup A_2 \cup \ldots \cup A_n|$.



Number of onto functions from a set with m elements to a set with n elements.

Modified Goal: Compute $|A_1 \cup A_2 \cup \ldots \cup A_n|$.

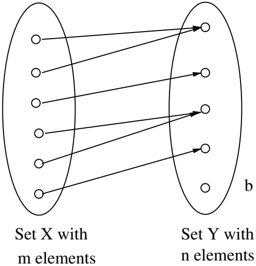


- $|A_i| = (n-1)^m$

Example: Number of Onto Functions

Number of onto functions from a set with m elements to a set with n elements.

Modified Goal: Compute $|A_1 \cup A_2 \cup \ldots \cup A_n|$.

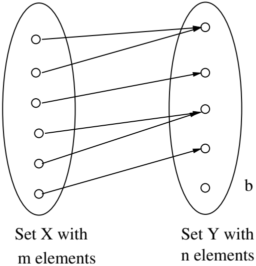


- $|A_i| = (n-1)^m$
- $|A_i \cap A_j| = (n-2)^m$

Example: Number of Onto Functions

Number of onto functions from a set with m elements to a set with n elements.

Modified Goal: Compute $|A_1 \cup A_2 \cup \ldots \cup A_n|$.

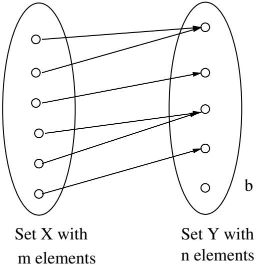


- $|A_i| = (n-1)^m$
- $|A_i \cap A_j| = (n-2)^m$
- Number of functions that do not have k many b's in the range $= (n-k)^m$

Example: Number of Onto Functions

Number of onto functions from a set with m elements to a set with n elements.

<u>Modified Goal:</u> Compute $|A_1 \cup A_2 \cup \ldots \cup A_n|$.

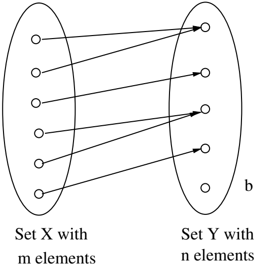


- $|A_i| = (n-1)^m$
- $|A_i \cap A_j| = (n-2)^m$
- Number of functions that do not have k many b's in the range $= (n-k)^m$
- Number of functions that do not have n many b's in the range $=$

Example: Number of Onto Functions

Number of onto functions from a set with m elements to a set with n elements.

Modified Goal: Compute $|A_1 \cup A_2 \cup \ldots \cup A_n|$.

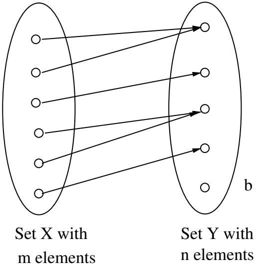


- $|A_i| = (n-1)^m$
- $|A_i \cap A_j| = (n-2)^m$
- Number of functions that do not have k many b's in the range $= (n-k)^m$
- Number of functions that do not have n many b's in the range $= 0$.

Example: Number of Onto Functions

Number of onto functions from a set with m elements to a set with n elements.

Modified Goal: Compute $|A_1 \cup A_2 \cup \ldots \cup A_n|$.

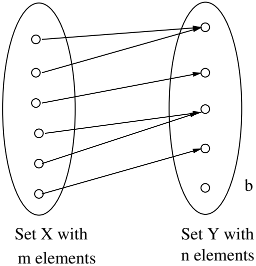


- $|A_i| = (n-1)^m$
- $|A_i \cap A_j| = (n-2)^m$
- Number of functions that do not have k many b's in the range $= (n-k)^m$
- Number of functions that do not have n many b's in the range $= 0$. There cannot be any such function!

Example: Number of Onto Functions

Number of onto functions from a set with m elements to a set with n elements.

Modified Goal: Compute $|A_1 \cup A_2 \cup \ldots \cup A_n|$.



- $|A_i| = (n-1)^m$
- $|A_i \cap A_j| = (n-2)^m$
- Number of functions that do not have k many b's in the range $= (n-k)^m$
- Number of functions that do not have n many b's in the range $= 0$. There cannot be any such function!

The number of onto functions from an m element set to an n element set where $m \geq n$ is:

$$n^m - \binom{n}{1}(n-1)^m + \binom{n}{2}(n-2)^m - \ldots + (-1)^{n-1}\binom{n}{n-1}1^m$$

Number of Onto Functions: a recursive specification

We now present a recursive definition of number of onto functions.

Number of Onto Functions: a recursive specification

We now present a recursive definition of number of onto functions.

$S(m, n)$: Number of onto functions from a set X with m elements to a set Y with n elements where $m \geq n$.

$$\begin{aligned} S(m, n) &= 1 && n = 1 \\ &= n^m - \sum_{k=1}^{n-1} \binom{n}{k} S(m, k) && \textit{otherwise} \end{aligned}$$

We now present a recursive definition of number of onto functions.

$S(m, n)$: Number of onto functions from a set X with m elements to a set Y with n elements where $m \geq n$.

$$
\begin{aligned}
S(m, n) &= 1 && n = 1 \\
&= n^m - \sum_{k=1}^{n-1} \binom{n}{k} S(m, k) && \textit{otherwise}
\end{aligned}
$$

Proof: The base case is readily verified. For the recursive step we observe:

- A function from X to Y is not onto if it has $1 \leq k \leq n-1$ many elements of Y in the range.

Number of Onto Functions: a recursive specification

We now present a recursive definition of number of onto functions.

$S(m, n)$: Number of onto functions from a set X with m elements to a set Y with n elements where $m \geq n$.

$$
\begin{aligned}
S(m,n) &= 1 && n = 1 \\
&= n^m - \sum_{k=1}^{n-1} \binom{n}{k} S(m,k) && \textit{otherwise}
\end{aligned}
$$

Proof: The base case is readily verified. For the recursive step we observe:

- A function from X to Y is not onto if it has $1 \leq k \leq n-1$ many elements of Y in the range.
- There are $\binom{n}{k}$ ways of selecting k elements from the set Y. Once we select these k elements as Y_k then $S(m, k)$ denotes the number of onto functions from X to Y_k.

We now present a recursive definition of number of onto functions.

$S(m, n)$: Number of onto functions from a set X with m elements to a set Y with n elements where $m \geq n$.

$$
\begin{aligned}
S(m, n) &= 1 && n = 1 \\
&= n^m - \sum_{k=1}^{n-1} \binom{n}{k} S(m, k) && \textit{otherwise}
\end{aligned}
$$

Proof: The base case is readily verified. For the recursive step we observe:

- A function from X to Y is not onto if it has $1 \leq k \leq n - 1$ many elements of Y in the range.
- There are $\binom{n}{k}$ ways of selecting k elements from the set Y. Once we select these k elements as Y_k then $S(m, k)$ denotes the number of onto functions from X to Y_k.
- Thus, $\binom{n}{k} S(m, k)$ denotes the number of functions from X to Y having exactly k elements in the range. Hence $\sum_{k=1}^{n-1} \binom{n}{k} S(m, k)$ is the number of functions from X to Y that are not onto.

We now present a recursive definition of number of onto functions.

$S(m,n)$: Number of onto functions from a set X with m elements to a set Y with n elements where $m \geq n$.

$$
\begin{aligned}
S(m,n) &= 1 & n = 1 \\
&= n^m - \sum_{k=1}^{n-1} \binom{n}{k} S(m,k) & \textit{otherwise}
\end{aligned}
$$

Proof: The base case is readily verified. For the recursive step we observe:

- A function from X to Y is not onto if it has $1 \leq k \leq n-1$ many elements of Y in the range.
- There are $\binom{n}{k}$ ways of selecting k elements from the set Y. Once we select these k elements as Y_k then $S(m,k)$ denotes the number of onto functions from X to Y_k.
- Thus, $\binom{n}{k} S(m,k)$ denotes the number of functions from X to Y having exactly k elements in the range. Hence $\sum_{k=1}^{n-1} \binom{n}{k} S(m,k)$ is the number of functions from X to Y that are not onto.
- Subtracting this from the total number of functions n^m gives the desired result.

Recurrences and its applications

We have already seen recursive sequences.

Next, we will see how to formulate real-world problems as recursive sequences and eventually get closed form expressions.

Qn: A code-word is made up of digits from 0 to 9 and a code word is valid if it contains odd number of 0s. Write a recursive formula for a_n which gives the number of code words of length n.

Qn: A code-word is made up of digits from 0 to 9 and a code word is valid if it contains odd number of 0s. Write a recursive formula for a_n which gives the number of code words of length n.

Examples:

- 0 is the only one length valid code-word. Thus $a_1 = 1$.
- 120078201 is valid but 120078200 is invalid.

Qn: A code-word is made up of digits from 0 to 9 and a code word is valid if it contains odd number of 0s. Write a recursive formula for a_n which gives the number of code words of length n.

Examples:

- 0 is the only one length valid code-word. Thus $a_1 = 1$.
- 120078201 is valid but 120078200 is invalid.

A recursive formula:

- To create a n length code-word, we can take an $n-1$ length code-word and add an appropriate digit.

Qn: A code-word is made up of digits from 0 to 9 and a code word is valid if it contains odd number of 0s. Write a recursive formula for a_n which gives the number of code words of length n.

Examples:

- 0 is the only one length valid code-word. Thus $a_1 = 1$.
- 120078201 is valid but 120078200 is invalid.

A recursive formula:

- To create a n length code-word, we can take an $n-1$ length code-word and add an appropriate digit.
  - If $n-1$ length code-word is valid, we can extend it in 9 different ways

Recurrence relations : a simple example

Qn: A code-word is made up of digits from 0 to 9 and a code word is valid if it contains odd number of 0s. Write a recursive formula for a_n which gives the number of code words of length n.

Examples:

- 0 is the only one length valid code-word. Thus $a_1 = 1$.
- 120078201 is valid but 120078200 is invalid.

A recursive formula:

- To create a n length code-word, we can take an $n-1$ length code-word and add an appropriate digit.
  - If $n-1$ length code-word is valid, we can extend it in 9 different ways(why?)

Recurrence relations : a simple example

Qn: A code-word is made up of digits from 0 to 9 and a code word is valid if it contains odd number of 0s. Write a recursive formula for a_n which gives the number of code words of length n.

Examples:

- 0 is the only one length valid code-word. Thus $a_1 = 1$.
- 120078201 is valid but 120078200 is invalid.

A recursive formula:

- To create a n length code-word, we can take an $n-1$ length code-word and add an appropriate digit.
  - If $n-1$ length code-word is valid, we can extend it in 9 different ways(why?) The number of $n-1$ length valid code-words is a_{n-1}.

Recurrence relations : a simple example

Qn: A code-word is made up of digits from 0 to 9 and a code word is valid if it contains odd number of 0s. Write a recursive formula for a_n which gives the number of code words of length n.

Examples:

- 0 is the only one length valid code-word. Thus $a_1 = 1$.
- 120078201 is valid but 120078200 is invalid.

A recursive formula:

- To create a n length code-word, we can take an $n-1$ length code-word and add an appropriate digit.
  - If $n-1$ length code-word is valid, we can extend it in 9 different ways(why?) The number of $n-1$ length valid code-words is a_{n-1}.
  - If $n-1$ length code-word is invalid, we can extend it in 1 way, by adding a 0.

Recurrence relations : a simple example

Qn: A code-word is made up of digits from 0 to 9 and a code word is valid if it contains odd number of 0s. Write a recursive formula for a_n which gives the number of code words of length n.

Examples:

- 0 is the only one length valid code-word. Thus $a_1 = 1$.
- 120078201 is valid but 120078200 is invalid.

A recursive formula:

- To create a n length code-word, we can take an $n-1$ length code-word and add an appropriate digit.
  - If $n-1$ length code-word is valid, we can extend it in 9 different ways(why?) The number of $n-1$ length valid code-words is a_{n-1}.
  - If $n-1$ length code-word is invalid, we can extend it in 1 way, by adding a 0. The number of invalid code-words of length $n-1$ is $10^{n-1} - a_{n-1}$.

Recurrence relations : a simple example

Qn: A code-word is made up of digits from 0 to 9 and a code word is valid if it contains odd number of 0s. Write a recursive formula for a_n which gives the number of code words of length n.

Examples:

- 0 is the only one length valid code-word. Thus $a_1 = 1$.
- 120078201 is valid but 120078200 is invalid.

A recursive formula:

- To create a n length code-word, we can take an $n-1$ length code-word and add an appropriate digit.
  - If $n-1$ length code-word is valid, we can extend it in 9 different ways(why?) The number of $n-1$ length valid code-words is a_{n-1}.
  - If $n-1$ length code-word is invalid, we can extend it in 1 way, by adding a 0. The number of invalid code-words of length $n-1$ is $10^{n-1} - a_{n-1}$.

$$
\begin{aligned}
a_n &= 9a_{n-1} + 10^{n-1} - a_{n-1} \\
&= 8a_{n-1} + 10^{n-1}
\end{aligned}
$$

Recurrence relations : a simple example

Qn: A code-word is made up of digits from 0 to 9 and a code word is valid if it contains odd number of 0s. Write a recursive formula for a_n which gives the number of code words of length n.

Examples:

- 0 is the only one length valid code-word. Thus $a_1 = 1$.
- 120078201 is valid but 120078200 is invalid.

A recursive formula:

- To create a n length code-word, we can take an $n-1$ length code-word and add an appropriate digit.
  - If $n-1$ length code-word is valid, we can extend it in 9 different ways(why?) The number of $n-1$ length valid code-words is a_{n-1}.
  - If $n-1$ length code-word is invalid, we can extend it in 1 way, by adding a 0. The number of invalid code-words of length $n-1$ is $10^{n-1} - a_{n-1}$.

$$\begin{aligned} a_n &= 9a_{n-1} + 10^{n-1} - a_{n-1} \\ &= 8a_{n-1} + 10^{n-1} \end{aligned}$$

Ex: What if the valid code-words need to contain even number of 0s. How does the formula change?

Summary

- Two important applications of principle of Inclusion Exclusion.
- Recursive definitions of the same.
- Recursive formula for a simple example.
- References Section 8.6 and 8.1 [KR]