Summer Bootcamp 2021

Introduction to Computer Science

Lecture 4

The Basics of Combinatorics

Artem Burmyakov

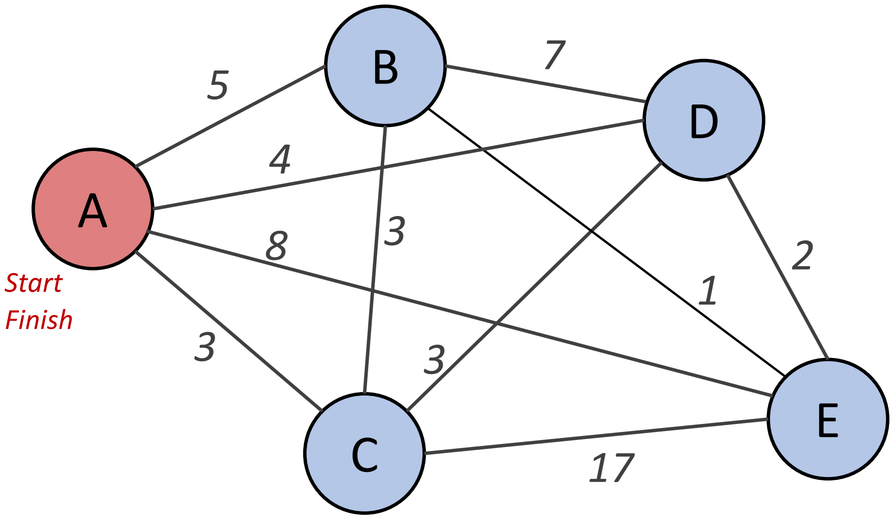
August 05, 2021

INNOPOLIS UNIVERSITY

Introduction: the Traveling Salesman Problem (TSP)

Find the shortest route through all cities, starting and ending at city A,

such that no city is visited twice.



Graph representing the distances (or travel times) between cities

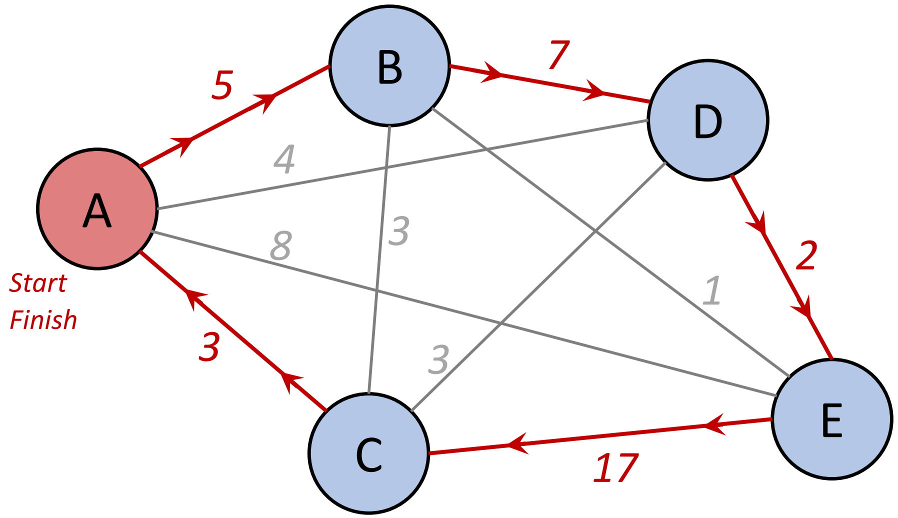
Assumptions:

- Any two cities are directly connected;
- Any city except A is visited once

Introduction: the Traveling Salesman Problem (TSP)

Find the shortest route through all cities, starting and ending at city A,

such that no city is visited twice.



Route length: 5+7+2+17+3 = 34

One possible route (selected in red)

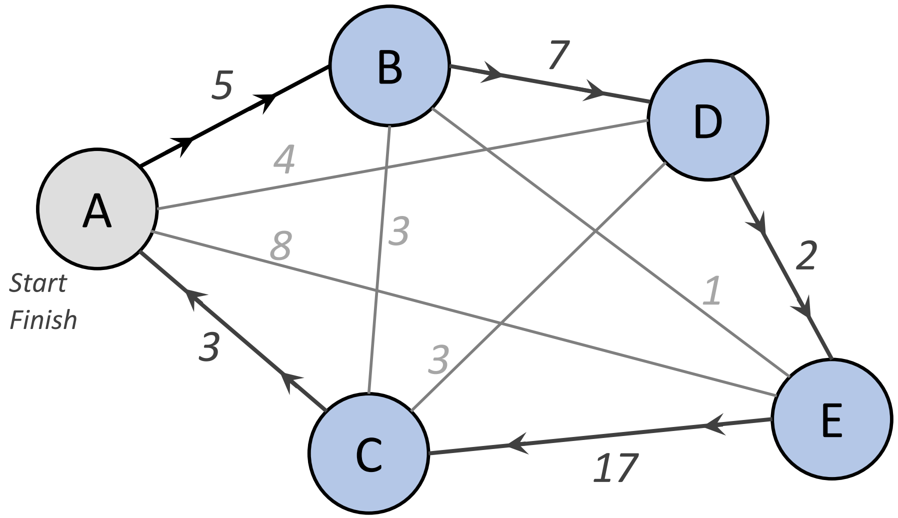
$A \longrightarrow B \longrightarrow$D $\longrightarrow$E $\longrightarrow$C $\longrightarrow$A

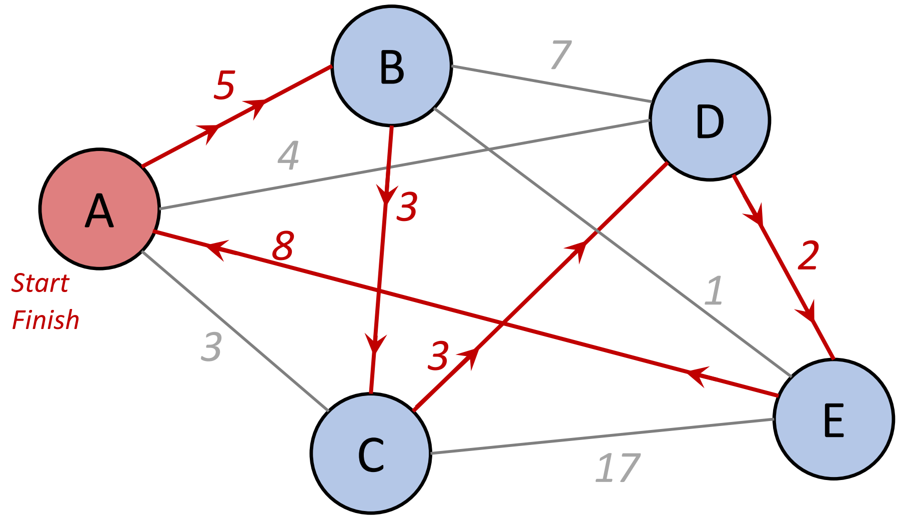
has length 5+7+2+17+3 = 34

Introduction: the Traveling Salesman Problem (TSP)

Find the shortest route through all cities, starting and ending at city A, such that no city is visited twice.

A different shorter route:





Route length: 5+7+2+17+3 = 34

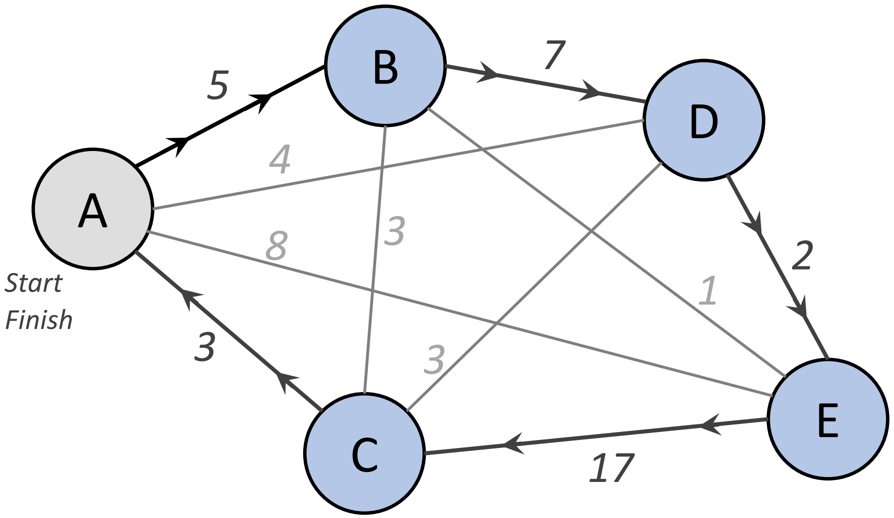
Route length: 5+3+3+2+8 = 21

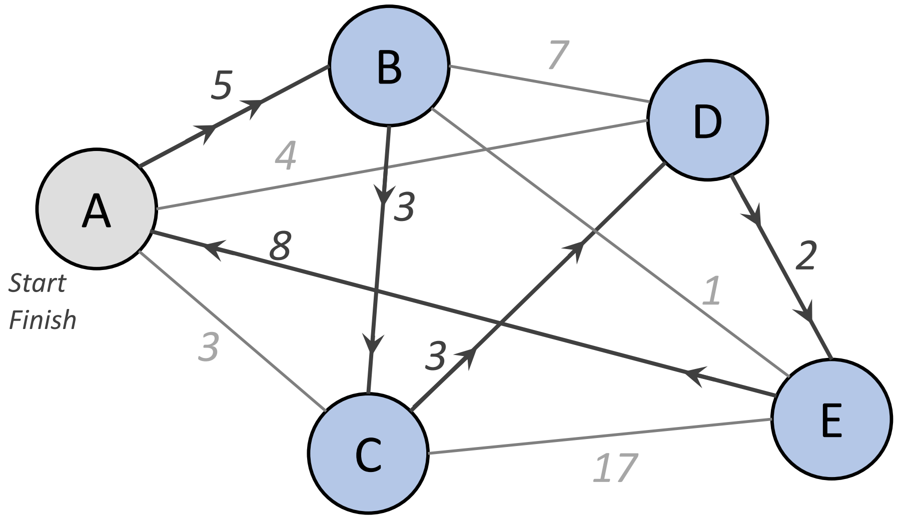
(But is it the shortest (optimal) one?!)

Introduction: the Traveling Salesman Problem (TSP)

Find the shortest route through all cities, starting and ending at city A,

such that no city is visited twice.



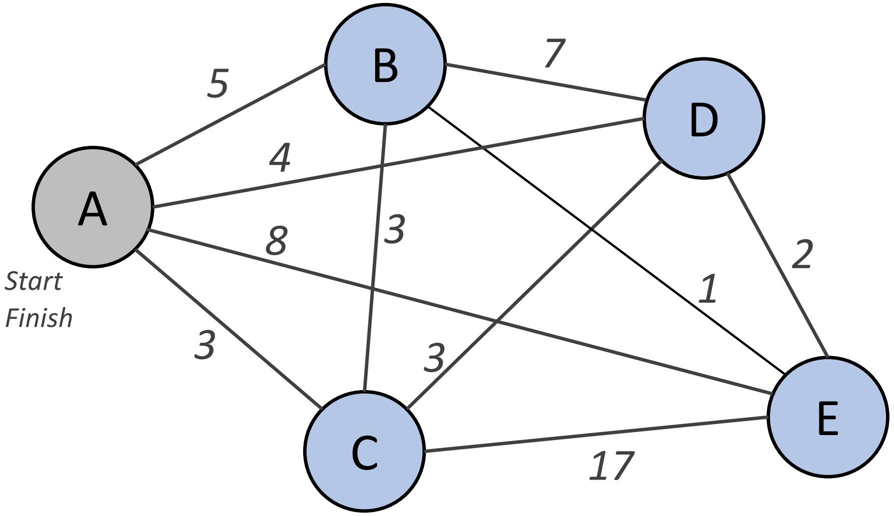


Route length: 5+7+2+17+3 = 34

Route length: 5+3+3+2+8 = 21

For the case of 4 cities (plus A), 24 routes exist in total

Combinatorics: How many feasible routes are?



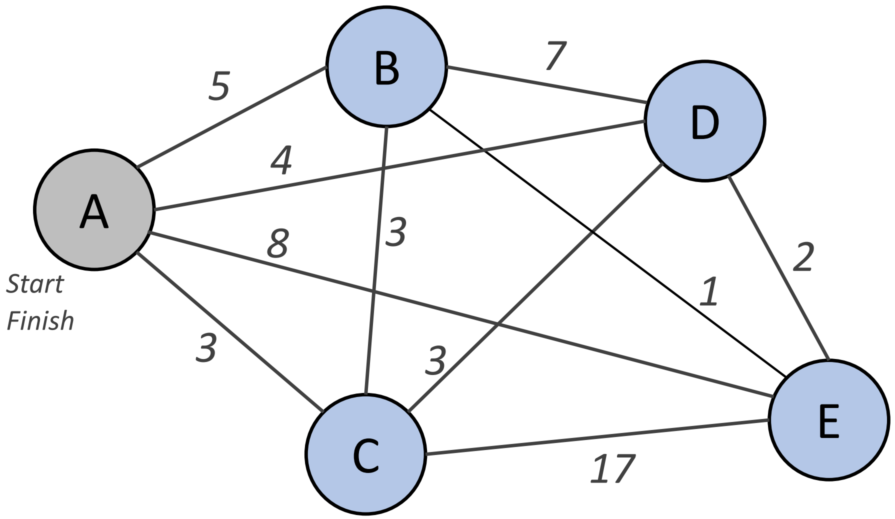
Route passes through all cities, starting and ending at A:

A	?	?	?	?	A

Assumptions:

- Any two cities are directly connected;
- Any city except A is visited once

Combinatorics: How many feasible routes are?



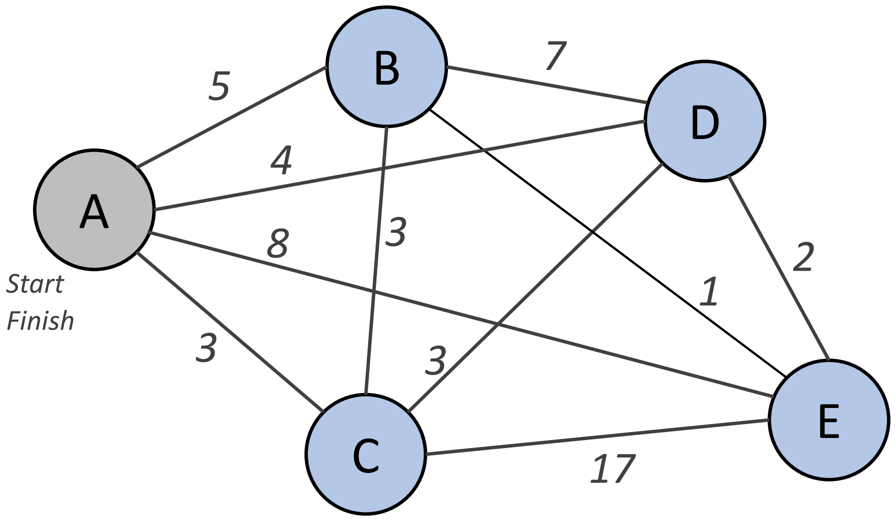
A	?	?	?	?	A

4 possible cases:
B, C, D, or E

Assumptions:

- Any two cities are directly connected;
- Any city except A is visited once

Combinatorics: How many feasible routes are?



A	?	?	?	?	A

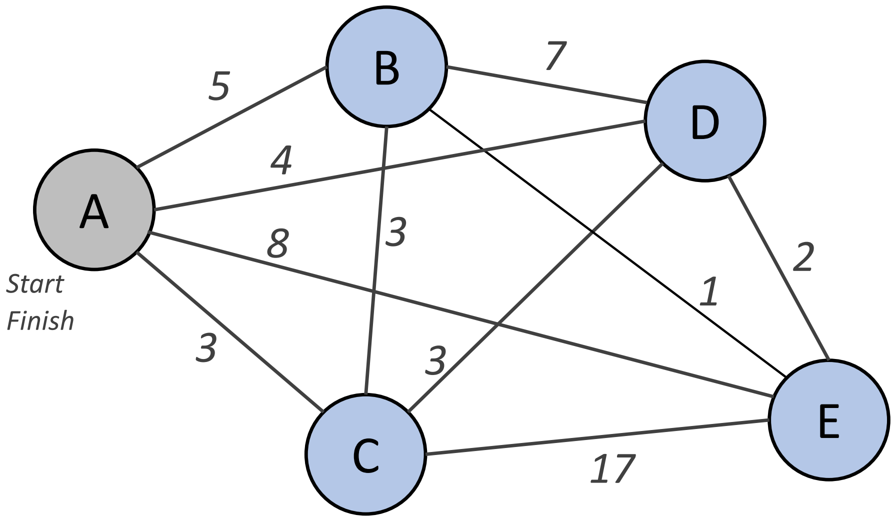
4 cases
(B, C, D, or E)

3 cases:
all cities (B, C, D, or E),
except the one visited before

Assumptions:

- Any two cities are directly connected;
- Any city except A is visited once

Combinatorics: How many feasible routes are?



A	?	?	?	?	A

4 cases
(B, C, D, or E)

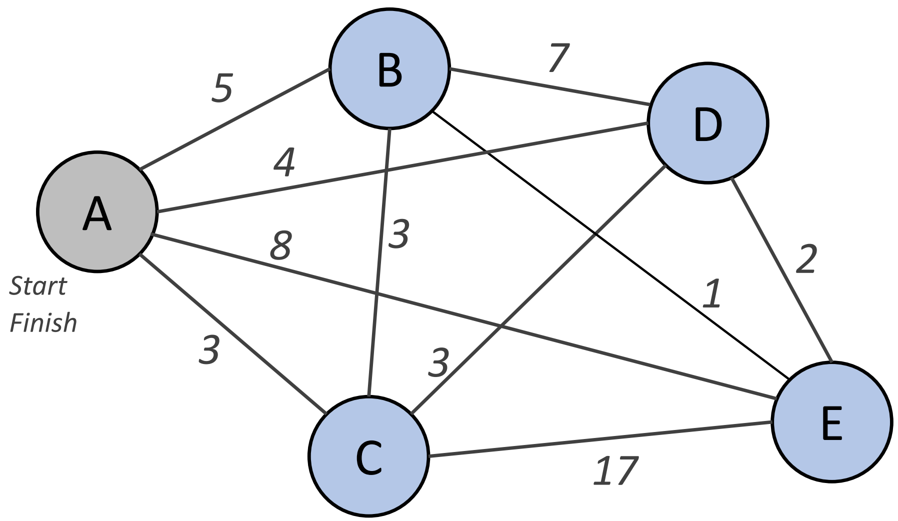
3 cases

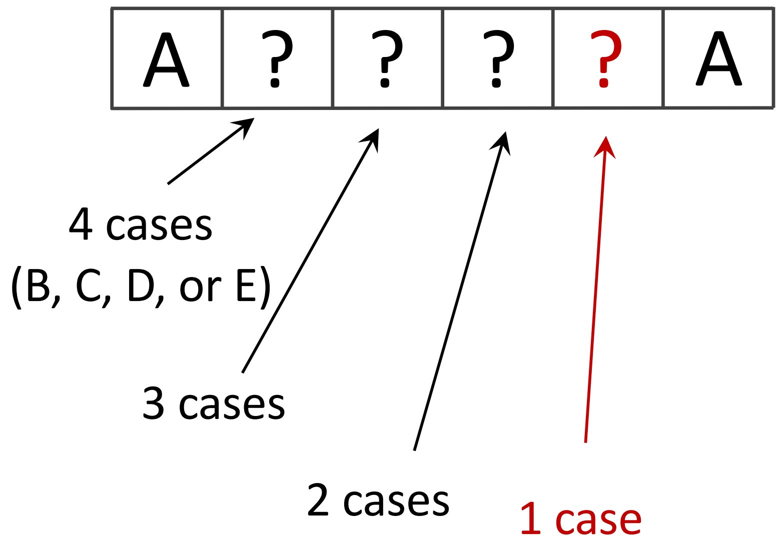
2 cases
(2 cities out of 4 have been already visited)

Assumptions:

- Any two cities are directly connected;
- Any city except A is visited once

Combinatorics: How many feasible routes are?





Assumptions:

- Any two cities are directly connected;
- Any city except A is visited once

Combinatorics:
How many feasible routes are there in the graph?

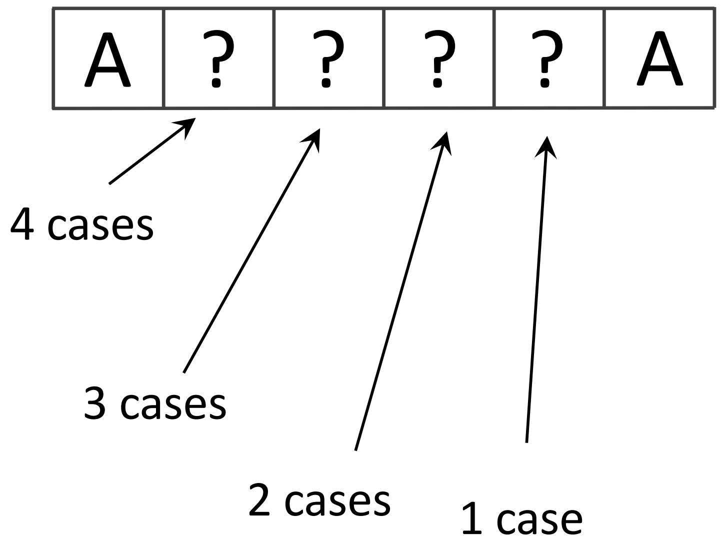
There are 4×3×2 ×1 = 24 feasible routes
to be examined for our graph, which are:

$A \longrightarrow B \longrightarrow C \longrightarrow D \longrightarrow E \longrightarrow A$
$A \longrightarrow B \longrightarrow C \longrightarrow E \longrightarrow D \longrightarrow A$
$A \longrightarrow B \longrightarrow D \longrightarrow C \longrightarrow E \longrightarrow A$
$A \longrightarrow B \longrightarrow D \longrightarrow E \longrightarrow C \longrightarrow A$
$A \longrightarrow B \longrightarrow E \longrightarrow C \longrightarrow D \longrightarrow A$
$A \longrightarrow B \longrightarrow E \longrightarrow D \longrightarrow C \longrightarrow A$

$A \longrightarrow C \longrightarrow B \longrightarrow D \longrightarrow E \longrightarrow A$
$A \longrightarrow C \longrightarrow B \longrightarrow E \longrightarrow D \longrightarrow A$
$A \longrightarrow C \longrightarrow D \longrightarrow B \longrightarrow E \longrightarrow A$
$A \longrightarrow C \longrightarrow D \longrightarrow E \longrightarrow B \longrightarrow A$
$A \longrightarrow C \longrightarrow E \longrightarrow B \longrightarrow D \longrightarrow A$
$A \longrightarrow C \longrightarrow E \longrightarrow D \longrightarrow B \longrightarrow A$

$A \longrightarrow D \longrightarrow B \longrightarrow C \longrightarrow E \longrightarrow A$
$A \longrightarrow D \longrightarrow B \longrightarrow E \longrightarrow C \longrightarrow A$
$A \longrightarrow D \longrightarrow C \longrightarrow B \longrightarrow E \longrightarrow A$
$A \longrightarrow D \longrightarrow C \longrightarrow E \longrightarrow B \longrightarrow A$
$A \longrightarrow D \longrightarrow E \longrightarrow B \longrightarrow C \longrightarrow A$
$A \longrightarrow D \longrightarrow E \longrightarrow C \longrightarrow B \longrightarrow A$

$A \longrightarrow E \longrightarrow B \longrightarrow C \longrightarrow D \longrightarrow A$
$A \longrightarrow E \longrightarrow B \longrightarrow D \longrightarrow C \longrightarrow A$
$A \longrightarrow E \longrightarrow C \longrightarrow B \longrightarrow D \longrightarrow A$
$A \longrightarrow E \longrightarrow C \longrightarrow D \longrightarrow B \longrightarrow A$
$A \longrightarrow E \longrightarrow D \longrightarrow B \longrightarrow C \longrightarrow A$
$A \longrightarrow E \longrightarrow D \longrightarrow C \longrightarrow B \longrightarrow A$



4×3×2 ×1 = 24 cases in total

Combinatorics studies the following questions:

How many possible routes are?

How many feasible combinations or permutations of objects exist?

Etc.

Combinatorics studies the following questions:

How many possible routes are?

How many feasible combinations or permutations of objects exist?

Etc.

Various subfields of combinatorics are identified, including:

- Enumerative combinatorics;
- Graph theory;
- Probabilistic combinatorics;
- Combinatorial game theory.

Combinatorics studies the following questions:

How many possible routes are?

How many feasible combinations or permutations of objects exist?

Etc.

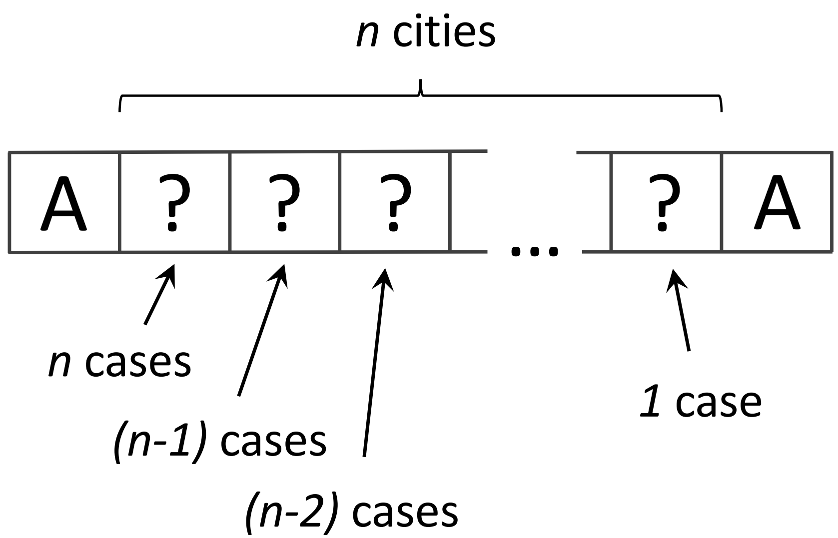
Various subfields of combinatorics are identified, including:

- Enumerative combinatorics;
- Graph theory;
- Probabilistic combinatorics;
- Combinatorial game theory.

Combinatorics is the foundation for computer science and probability theory (of a discrete variable) in particular

Combinatorics:
How many feasible routes are there in the graph?

General case for *n* cities:



The number of feasible routes to be examined equals to:

$$\boxed{n! = n\times(n-1)\times(n-2)\times\cdots\times3\times2\times1}$$

("n factorial")

Factorial: $n! \stackrel{\text{def}}{=} 1\times 2\times 3\times\cdots\times(n-2)\times(n-1)\times n$

n	$n!$
0	1
1	1
2	2
3	6
4	24
5	120
6	720
7	5040
8	40320
9	362880
10	3628800
11	39916800
12	479001600
13	6227020800
14	87178291200
15	1307674368000
16	20922789888000
17	355687428096000
18	6402373705728000
19	121645100408832000
20	2432902008176640000

Solution Strategies for Computationally Intensive Problems	

Solution Strategies for Computationally Intensive Problems	
Exact (Optimal)	
• Finds an exact optimal solution;	

Solution Strategies for Computationally Intensive Problems	
Exact (Optimal)	
• Finds an exact optimal solution; • Enumerates all feasible solutions, until there is 100% guarantee for a solution optimality	

<table>
<tr><th colspan="2">Solution Strategies for Computationally Intensive Problems</th></tr>
<tr><td>Exact (Optimal)</td><td>Approximate (Suboptimal)</td></tr>
<tr><td>Finds an exact optimal solution;
Enumerates all feasible solutions, until there is 100% guarantee for a solution optimality</td><td>Aims at finding a “good” (not necessarily the best) solution in a reasonable time;</td></tr>
</table>

Solution Strategies for Computationally Intensive Problems	
Exact (Optimal)	Approximate (Suboptimal)
• Finds an exact optimal solution; • Enumerates all feasible solutions, until there is 100% guarantee for a solution optimality	• Aims at finding a “good” (not necessarily the best) solution in a reasonable time; • Checks a subset of feasible solutions, and chooses the best among them;

Solution Strategies for Computationally Intensive Problems	
Exact (Optimal)	Approximate (Suboptimal)
• Finds an exact optimal solution; • Enumerates all feasible solutions, until there is 100% guarantee for a solution optimality	• Aims at finding a “good” (not necessarily the best) solution in a reasonable time; • Checks a subset of feasible solutions, and chooses the best among them; • No guarantee on solution optimality is provided

<table>
<tr><th colspan="2">Solution Strategies for Computationally Intensive Problems</th></tr>
<tr><td>Exact (Optimal)</td><td>Approximate (Suboptimal)</td></tr>
<tr><td>Finds an exact optimal solution;
Enumerates all feasible solutions, until there is 100% guarantee for a solution optimality</td><td>Aims at finding a “good” (not necessarily the best) solution in a reasonable time;
Checks a subset of feasible solutions, and chooses the best among them;
No guarantee on solution optimality is provided</td></tr>
<tr><td>Takes a lot of time
(might never complete, be intractable)</td><td></td></tr>
</table>

Solution Strategies for Computationally Intensive Problems	
Exact (Optimal)	Approximate (Suboptimal)
• Finds an exact optimal solution; • Enumerates all feasible solutions, until there is 100% guarantee for a solution optimality	• Aims at finding a “good” (not necessarily the best) solution in a reasonable time; • Checks a subset of feasible solutions, and chooses the best among them; • No guarantee on solution optimality is provided
Takes a lot of time (might never complete, be intractable)	**Significantly faster**

Solution Strategies for Computationally Intensive Problems	
Exact (Optimal)	Approximate (Suboptimal)
• Finds an exact optimal solution; • Enumerates all feasible solutions, until there is 100% guarantee for a solution optimality	• Aims at finding a “good” (not necessarily the best) solution in a reasonable time; • Checks a subset of feasible solutions, and chooses the best among them; • No guarantee on solution optimality is provided
Takes a lot of time (might never complete, be intractable)	**Significantly faster**
Consumes a lot of memory	**Lower memory consumption**

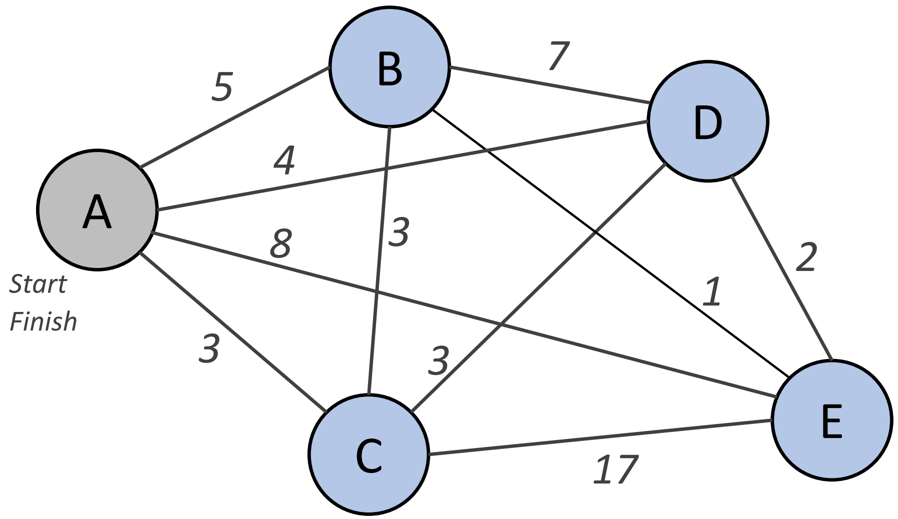
Solution Strategies for Computationally Intensive Problems	
Exact (Optimal)	**Approximate (Suboptimal)**
• Finds an exact optimal solution; • Enumerates all feasible solutions, until there is 100% guarantee for a solution optimality	• Aims at finding a "good" (not necessarily the best) solution in a reasonable time; • Checks a subset of feasible solutions, and chooses the best among them; • No guarantee on solution optimality is provided
Takes a lot of time (might never complete, be intractable)	**Significantly faster**
Consumes a lot of memory	**Lower memory consumption**

Solution Strategies for Computationally Intensive Problems	
Exact (Optimal)	Approximate (Suboptimal)
• Finds an exact optimal solution; • Enumerates all feasible solutions, until there is 100% guarantee for a solution optimality	• Aims at finding a “good” (not necessarily the best) solution in a reasonable time; • Checks a subset of feasible solutions, and chooses the best among them; • No guarantee on solution optimality is provided
Takes a lot of time (might never complete, be intractable)	Significantly faster
Consumes a lot of memory	Lower memory consumption

Examples: Brute-force enumeration (or search) The Nearest Neighbour Heuristic

Generalization of the Traveling Salesman Problem (TSP)

Find the shortest route through all cities, starting and ending at city A



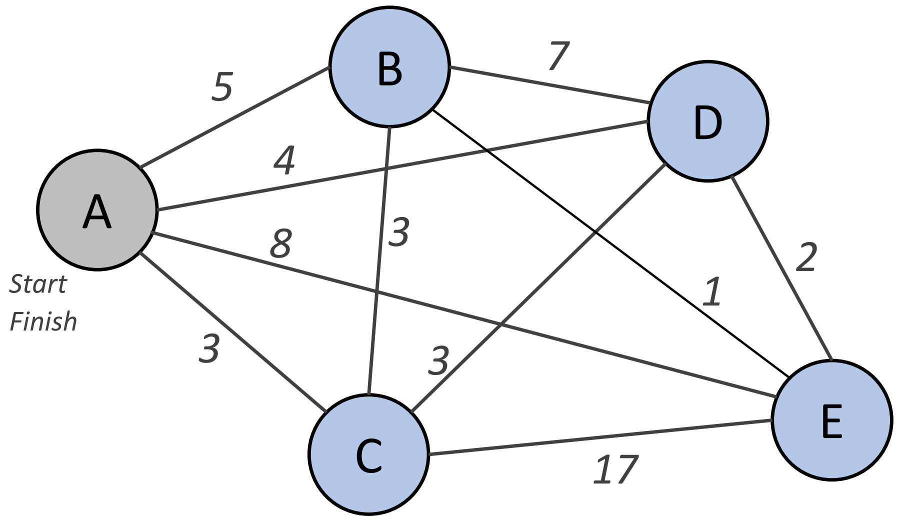
Graph representing the distances (or travel times) between cities

Possible generalizations:

- Cities can be visited multiple times;

Generalization of the Traveling Salesman Problem (TSP)

Find the shortest route through all cities, starting and ending at city A

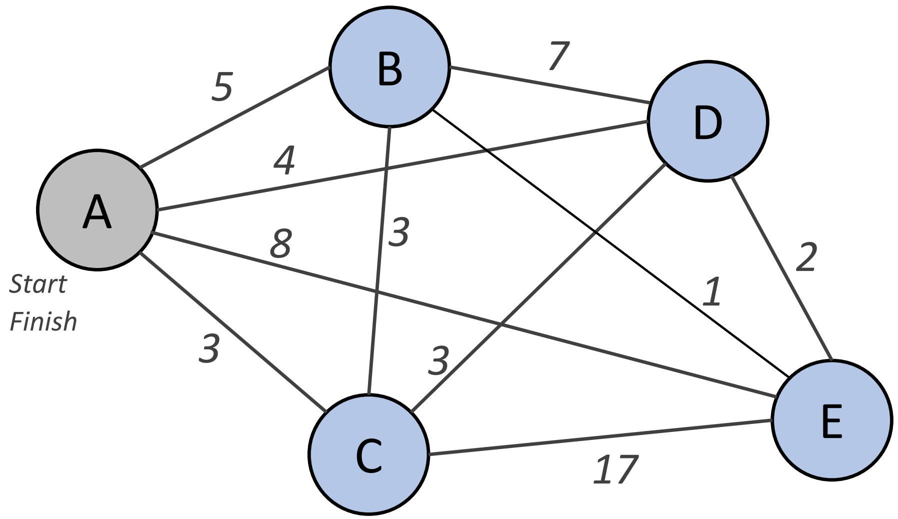


Possible generalizations:

- Cities can be visited multiple times;
- Some cities are not connected directly;

Generalization of the Traveling Salesman Problem (TSP)

Find the shortest route through all cities, starting and ending at city A

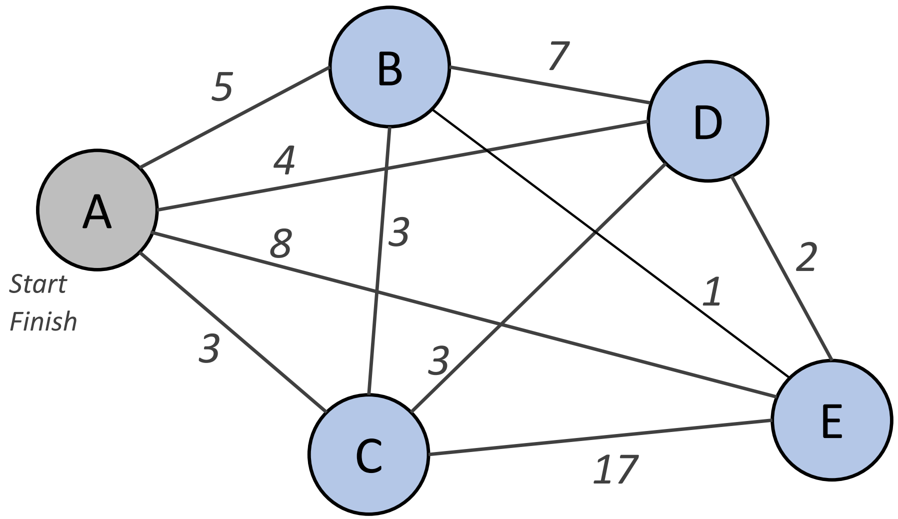


Possible generalizations:

- Cities can be visited multiple times;
- Some cities are not connected directly;
- Not all cities must be visited;
- Etc.

Summary of the Traveling Salesman Problem (TSP)

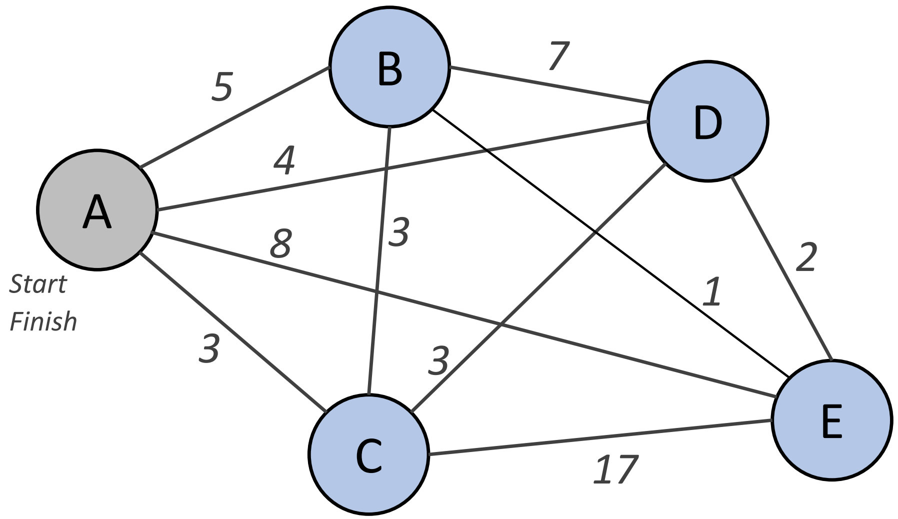
Find the shortest route through all cities, starting and ending at city A



Traveling Salesman Problem is an example of an ***NP*-hard** computational problem.

Summary of the Traveling Salesman Problem (TSP)

Find the shortest route through all cities, starting and ending at city A

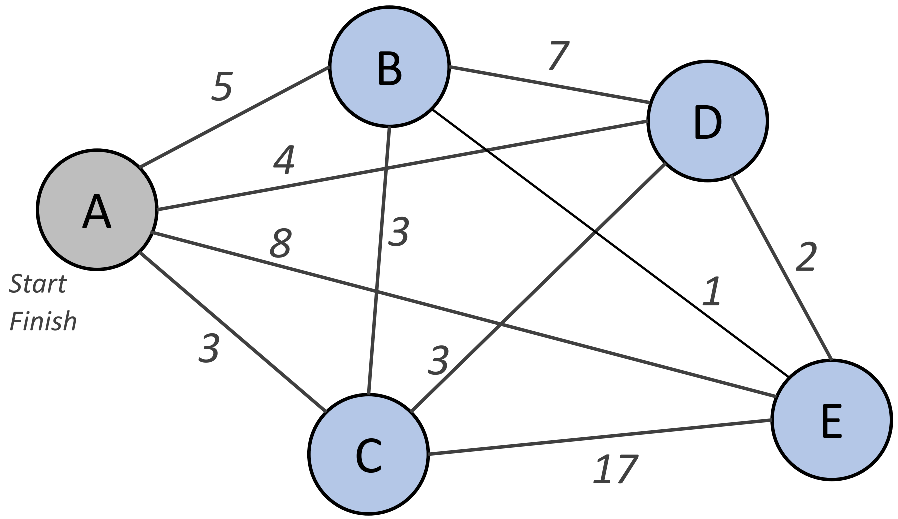


Traveling Salesman Problem is an example of an ***NP*-hard** computational problem.

No efficient fast exact solving algorithm has been derived so far, only approximate solutions are available.

Summary of the Traveling Salesman Problem (TSP)

Find the shortest route through all cities, starting and ending at city A



Traveling Salesman Problem is an example of an ***NP*-hard** computational problem.

No efficient fast exact solving algorithm has been derived so far, only approximate solutions are available.

The problem is fundamental for Computer Science, e.g. an optimal routing of data packets in a computer network.

Combinatorics: Key Terms

Permutation of objects

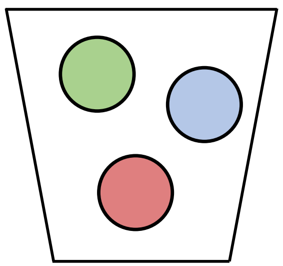
Combination of objects *with* repetition

Combination of objects *without* repetition

Permutation – an ordered set of objects (the order matters!)

Example:

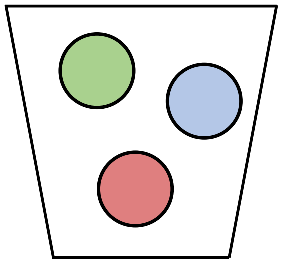
There are 3 balls of different colours in a basket



Permutation – an ordered set of objects (the order matters!)

Example:

There are 3 balls of different colours in a basket

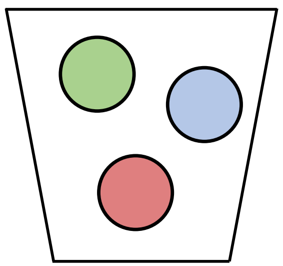


We remove all balls from the basket, one by one

Permutation – an ordered set of objects (the order matters!)

Example:

There are 3 balls of different colours in a basket



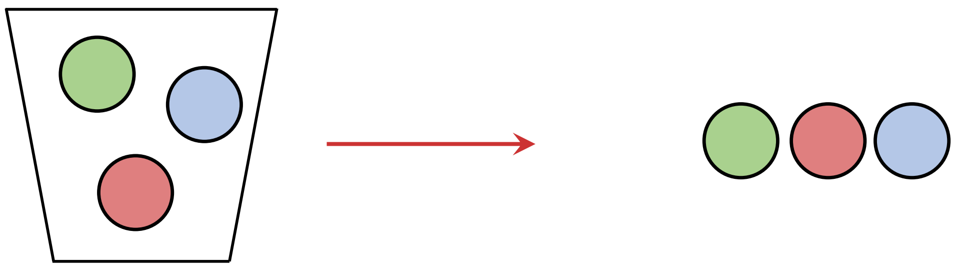
We remove all balls from the basket, one by one

How many different outcomes are feasible?

Permutation – an ordered set of objects (the order matters!)

Example:

There are 3 balls of different colours in a basket



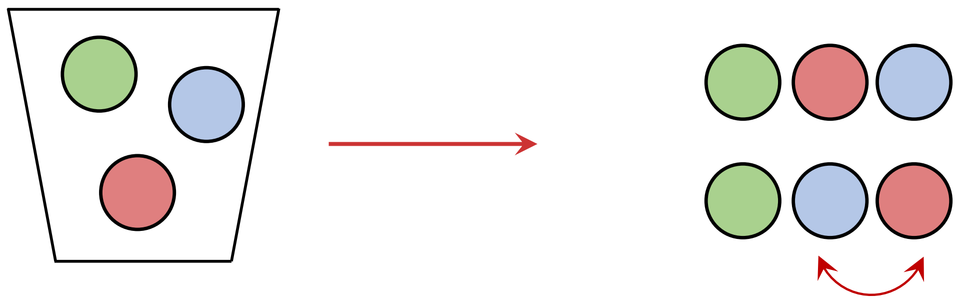
We remove all balls from the basket, one by one

How many different outcomes are feasible?

Permutation – an ordered set of objects (the order matters!)

Example:

There are 3 balls of different colours in a basket



We remove all balls from the basket, one by one

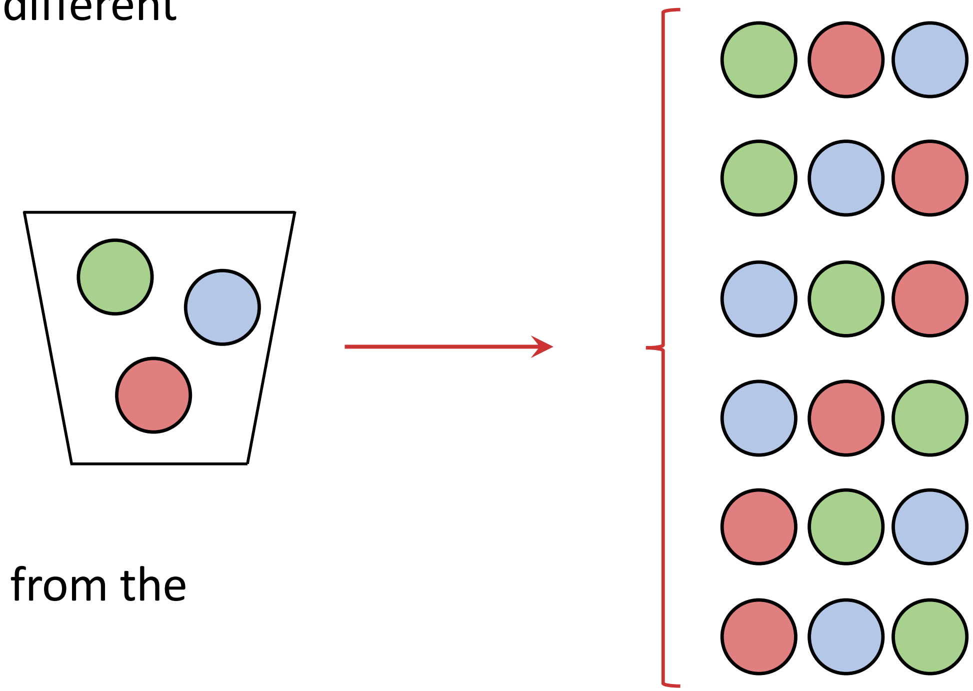
Or instead we take out a blue ball before red one

How many different outcomes are feasible?

Permutation – an ordered set of objects (the order matters!)

Example:

There are 3 balls of different colours in a basket



6 feasible permutations

We remove all balls from the basket, one by one

How many different outcomes are feasible?

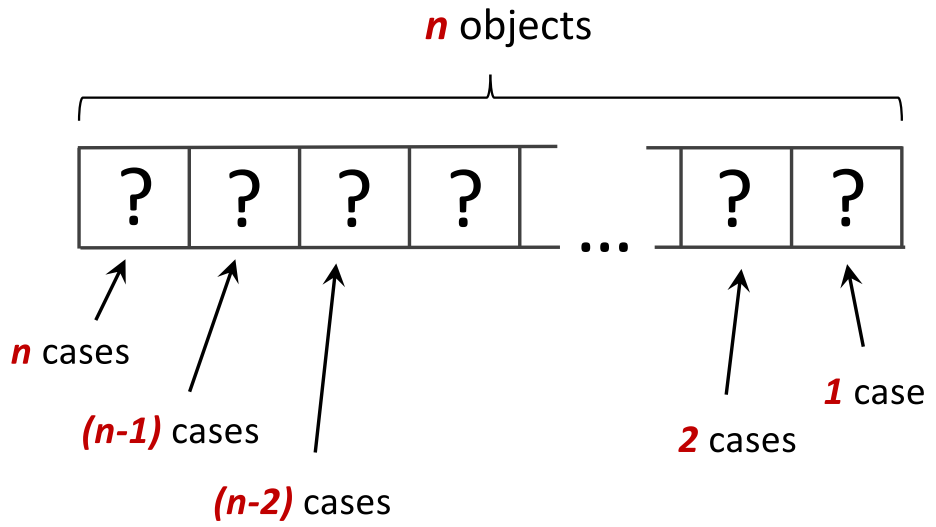
Permutation – an ordered set of objects (the order matters!)

P_n – the number of feasible permutations of n distinct objects

Permutation – an ordered set of objects (the order matters!)

P_n – the number of feasible permutations of n distinct objects

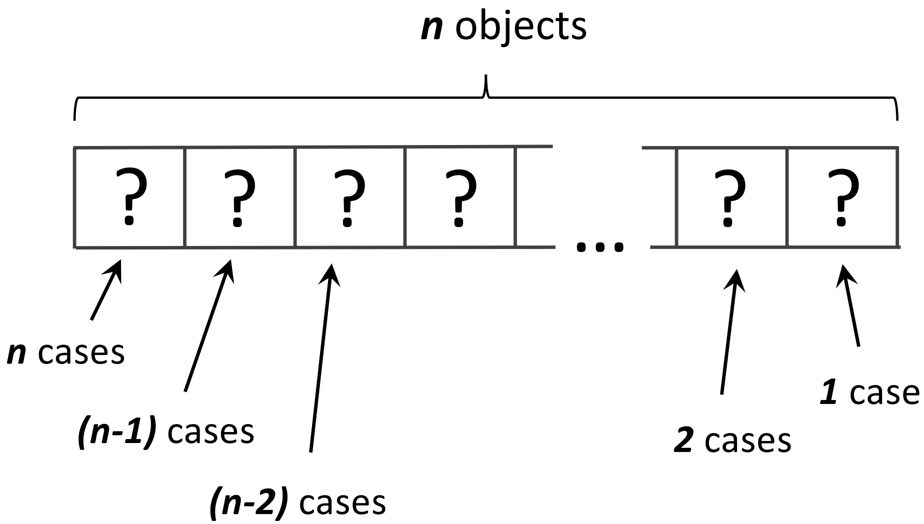
General case: There are $n!$ feasible permutations of *n* distinct objects:



Permutation – an ordered set of objects (the order matters!)

P_n – the number of feasible permutations of n distinct objects

General case: There are $n!$ feasible permutations of *n* distinct objects:



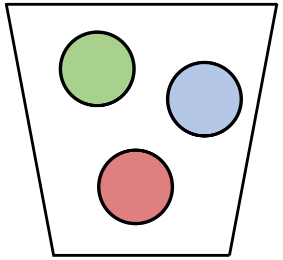
The number of permutations $P(n)$:

$$P_n = n\times(n-1)\times(n-2)\times\cdots\times3\times2\times1 = n!$$

Partial Permutation

Example:

There are 3 balls of different colours in a basket

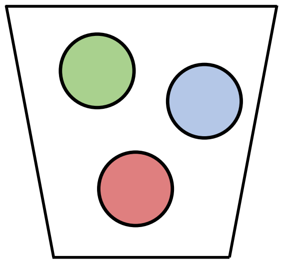


We now take out only 2 balls from the basket, one by one

Partial Permutation

Example:

There are 3 balls of different colours in a basket



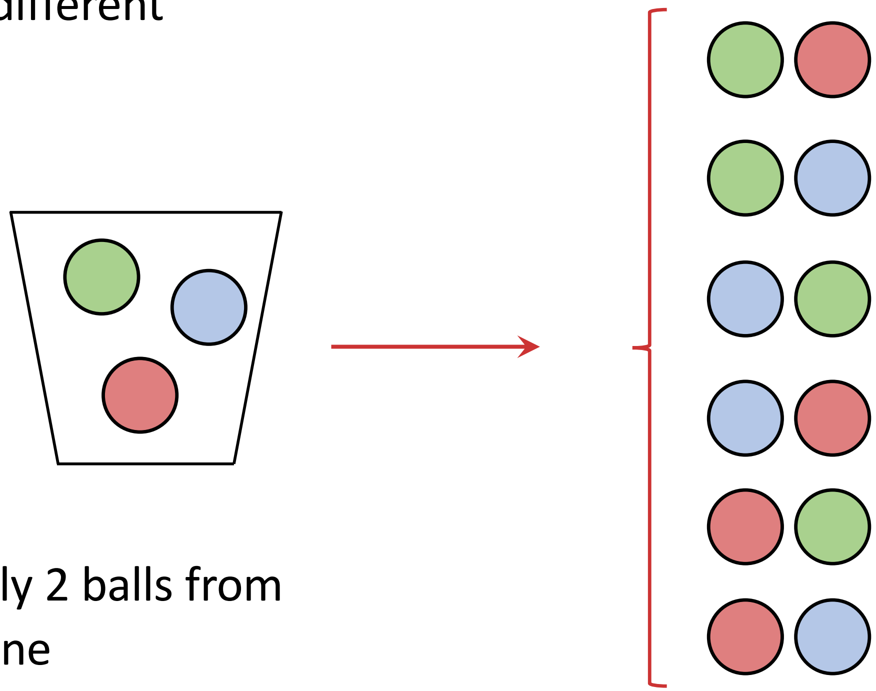
We now take out only 2 balls from the basket, one by one

How many different outcomes are feasible?

Partial Permutation

Example:

There are 3 balls of different colours in a basket



6 feasible partial permutations as well

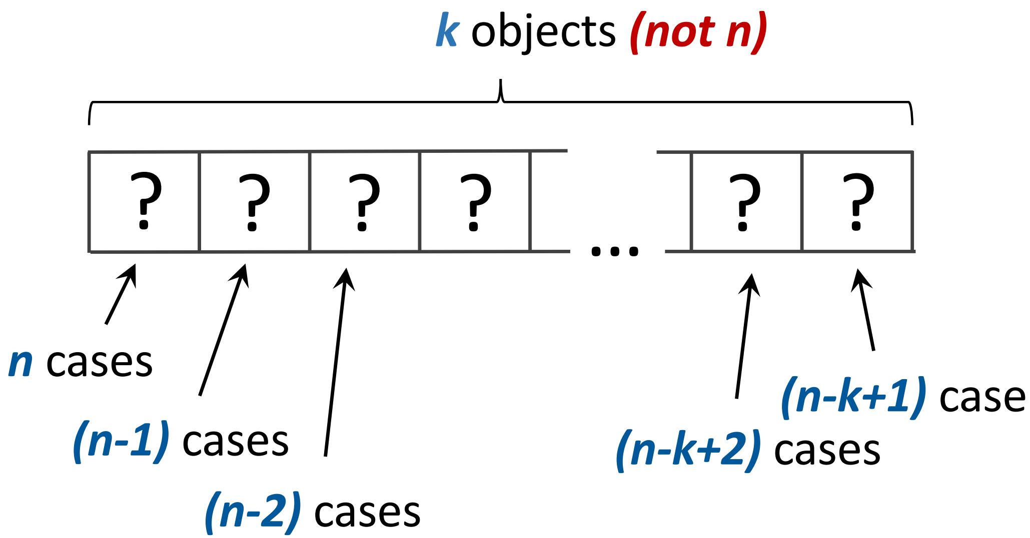
(coincidence)

We now take out only 2 balls from the basket, one by one

How many different outcomes are feasible?

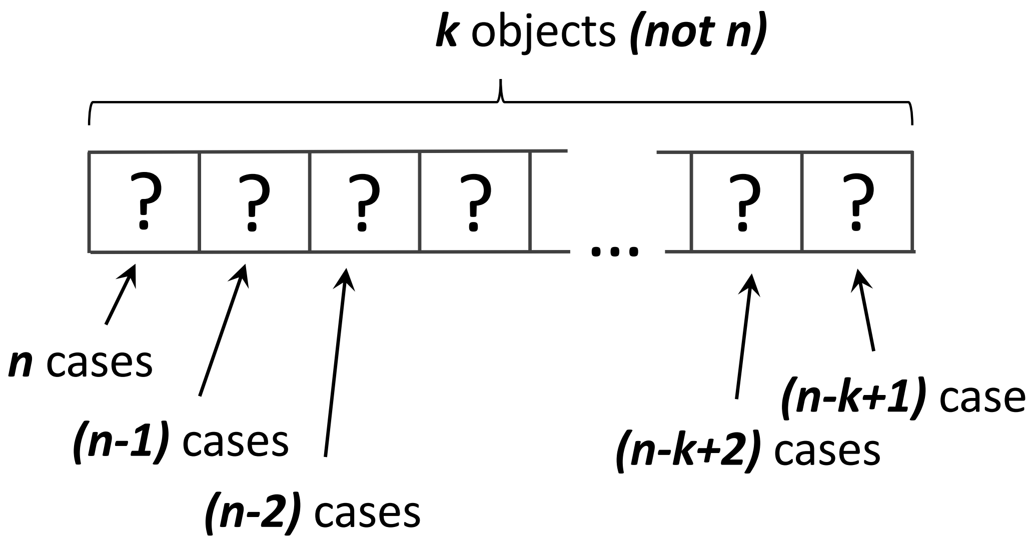
Partial Permutation

P_n^k – the number of feasible permutations of n distinct objects



Partial Permutation

P_n^k – the number of feasible permutations of n distinct objects

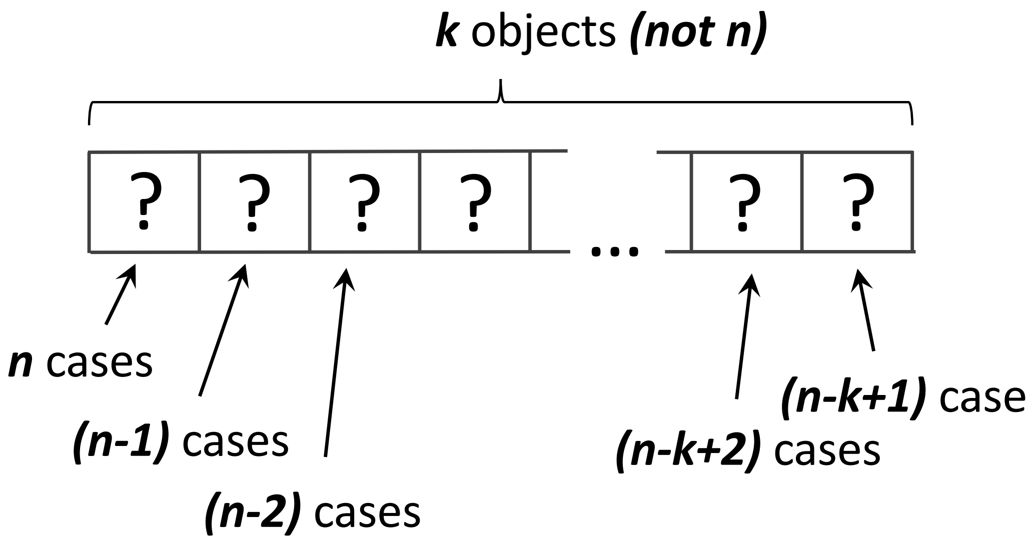


The number of permutations $P(n, k)$:

$$P_n^k = n \times (n-1) \times (n-2) \times \cdots \times (n-k+2) \times (n-k+1)$$

Partial Permutation

P_n^k – the number of feasible permutations of n distinct objects

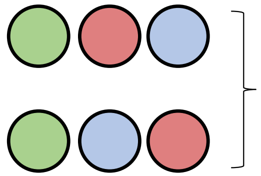


The number of permutations $P(n, k)$:

$$P_n^k = n \times (n-1) \times (n-2) \times \cdots \times (n-k+2) \times (n-k+1) = {n!}/{(n-k)!}$$

Permutations vs. Combinations

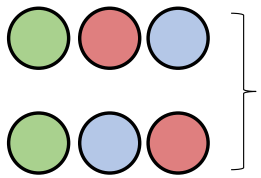
Permutation – the order matters:



Two different permutations

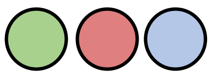
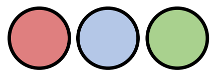
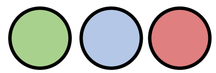
Permutations vs. Combinations

Permutation – the order matters:



Two different permutations

Combination – the order does not matter

Combination  = *Combination*  = *Combination* 

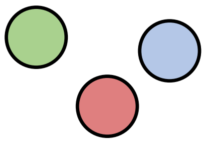
Combination of Objects (the order does not matter!)

Combination – a selection of *k* objects out of *n* types, with $k \leq n$, either with or without repetition

Combination of Objects (the order does not matter!)

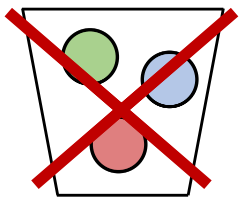
Combination – a selection of k objects out of n types, with $k \leq n$, either with or without repetition

Example: Possible combinations of 2 balls out of 3 types of balls:



Types of balls available (3 in total)

An infinite number of balls is available for each type

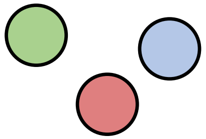


No basket with a limited number of objects

Combination of Objects (the order does not matter!)

Combination – a selection of *k* objects out of *n* types, with $k \leq n$, either with or without repetition

Example: Possible combinations of 2 balls out of 3 types of balls:

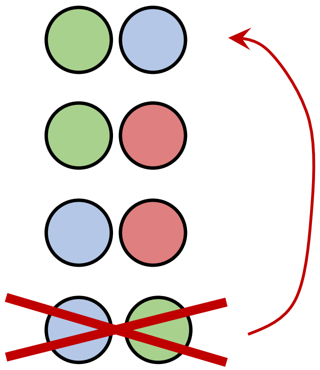


Types of balls available (3 in total)

An infinite number of balls is available for each type

without repetition of types:

3 cases

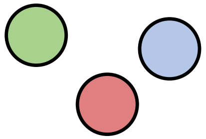


This combination is already present
(recall that the order does not matter)

Combination of Objects (the order does not matter!)

Combination – a selection of k objects out of n types, with $k \leq n$, either with or without repetition

Example: Possible combinations of 2 balls out of 3 types of balls:

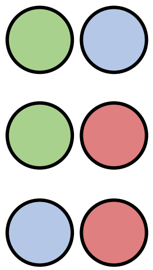


Types of balls available (3 in total)

An infinite number of balls is available for each type

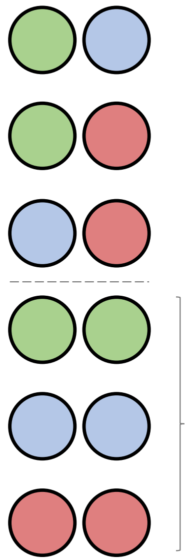
without repetition of types:

3 cases



with repetitions of types:

6 cases



repetitions

Combination of Objects Without Repetitions

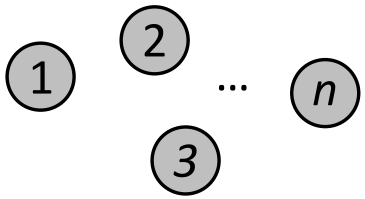
$C_n^k,\ C(n,k), \binom{n}{k}$ – the number of feasible *k*-combinations of *n* objects (without repetitions)

Combination of Objects Without Repetitions

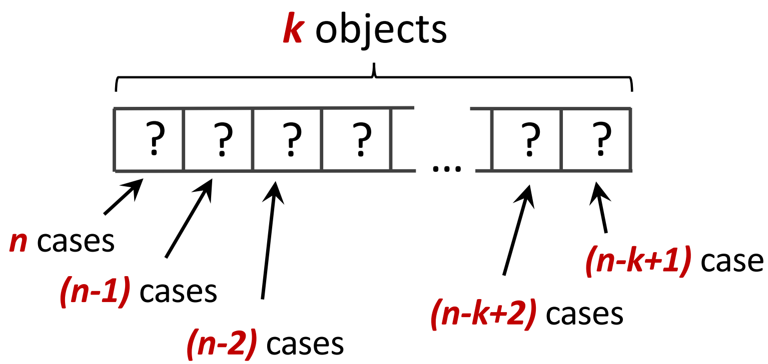
$C_n^k,\ C(n,k),\ \binom{n}{k}$ – the number of feasible *k*-combinations of *n* objects (without repetitions)

Types of balls available

(*n* in total):



An infinite number of balls is available for each type



Number of combinations without repetitions:

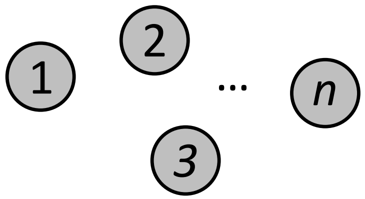
$$C_n^k = \frac{n!}{(n-k)! * k!}$$

Combination of Objects Without Repetitions

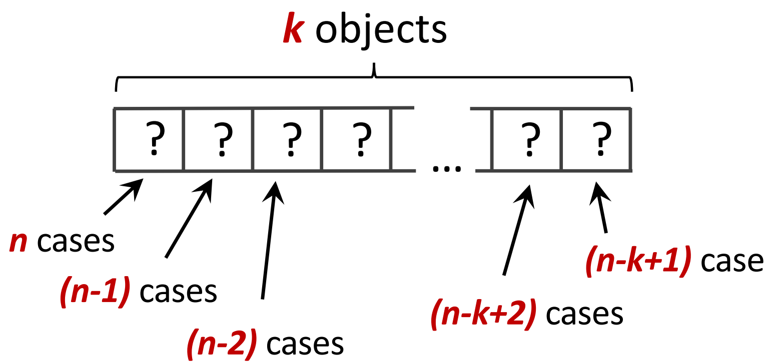
C_n^k, $C(n,k)$, $\binom{n}{k}$ – the number of feasible *k*-combinations of *n* objects (without repetitions)

Types of balls available

(*n* in total):



An infinite number of balls is available for each type



Number of combinations without repetitions:

$$C_n^k = \frac{n!}{(n-k)! * k!}$$

Number of *k*-permutations out of *n* objects (see prev. slides)

To exclude combinations of the same objects, but of different orders (their number in total is *k*!)

Combination of Objects With Repetitions

Number of feasible combinations of k objects out of n types, with repetitions, is computed by:

$$C_{n+k-1}^{k} = \frac{(n+k-1)!}{(n-1)! * k!}$$

(The computation is outside the scope of this presentation)

Binomial Theorem

The number of feasible *k*-combinations of *n* objects

$$C_n^k = \frac{n!}{(n-k)! * k!}$$

is also known as the ***binomial coefficient***

$$(x+y)^n = C_n^0 x^0 y^n + C_n^1 x^1 y^{n-1} + C_n^2 x^2 y^{n-2} + \cdots + C_n^{n-1} x^{n-1} y^1 + C_n^n x^n y^0$$

Binomial coefficients

Binomial Theorem

The number of feasible *k*-combinations of *n* objects

$$C_n^k = \frac{n!}{(n-k)! * k!}$$

is also known as the ***binomial coefficient***

$$(x+y)^n = C_n^0 x^0 y^n + C_n^1 x^1 y^{n-1} + C_n^2 x^2 y^{n-2} + \cdots + C_n^{n-1} x^{n-1} y^1 + C_n^n x^n y^0$$

$$(x+y)^n = \sum_{k=0}^{n} C_n^k x^k y^{n-k}$$

Pascal's Triangle – an efficient way to compute binomial coefficients $C_n^k = \frac{n!}{(n-k)! * k!}$

Pascal’s Triangle – an efficient way to compute binomial coefficients $C_n^k = \frac{n!}{(n-k)! * k!}$

						1						
					1		1					
				1				1				
			1						1			
		1								1		
	1										1	
1												1

Pascal’s Triangle – an efficient way to compute binomial coefficients $C_n^k = \frac{n!}{(n-k)!*k!}$

						1						
					1		1					
				1		+ **2**		1				
			1						1			
		1								1		
	1										1	
1												1

Pascal’s Triangle – an efficient way to compute binomial coefficients $C_n^k = \frac{n!}{(n-k)!*k!}$

						1						
					1		1					
				1		2		1				
			1		**3**				1			
		1								1		
	1										1	
1												1

Pascal's Triangle – an efficient way to compute binomial coefficients $C_n^k = \frac{n!}{(n-k)! * k!}$

						1						
					1		1					
				1		2		1				
			1		**3**		**3**		1			
		1								1		
	1										1	
1												1

Pascal's Triangle – an efficient way to compute binomial coefficients $C_n^k = \frac{n!}{(n-k)!*k!}$

						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

Pascal's Triangle – an efficient way to compute binomial coefficients $C_n^k = \frac{n!}{(n-k)!*k!}$

						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

Pascal’s Triangle – an efficient way to compute binomial coefficients

Interpretation:

$$C_n^k = \frac{n!}{(n-k)! * k!}$$

						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

Pascal's Triangle – an efficient way to compute binomial coefficients

Interpretation:

$$C_{n}^{k} = \frac{n!}{(n-k)! * k!}$$

Row number

0							1						
1						1		1					
2					1		2		1				
3				1		3		3		1			
4			1		4		6		4		1		
5		1		5		10		10		5		1	
6	1		6		15		20		15		6		1

Pascal’s Triangle – an efficient way to compute binomial coefficients

Interpretation:

Position in that row

$$C_{n}^{k} = \frac{n!}{(n-k)! * k!}$$

Row number

0							1						
1						1		1					
2					1		2		1				
3				1		3		3		1			
4			1		4		6		4		1		
5		1		5		10		10		5		1	
6	1		6		15		20		15		6		1

Pascal’s Triangle – an efficient way to compute binomial coefficients

Interpretation:

Position in that row

C_6^4

Row number

0							1						
1						1		1					
2					1		2		1				
3				1		3		3		1			
4			1		4		6		4		1		
5		1		5		10		10		5		1	
6	1		6		15		20		$\boxed{15}$		6		1
	0		*1*		*2*		*3*		*4*				

Note: Rows and columns are numbered from 0, not 1

Summary: Basic Formulas of Combinatorics

The number of

Permutations of *n* objects	$P(n) = n!$
k-Permutations of *n* objects	$P(n) = {}^{n!}/_{(n-k)!}$
k-combinations of *n* objects, without repetition	$C(n,k) = \dfrac{n!}{(n-k)!\,k!}$
k-combination of *n* objects, with repetition	$C(n+k-1,k) = \dfrac{(n+k-1)!}{(n-1)!\,k!}$

Combinatorics is the foundation for probability theory that we study next time

Terminology

Combinatorics

Traveling Salesman Problem (TSP)

An optimal route

Brute-force search

Exhaustive enumeration

General case

Particular case

Factorial

Feasible scenarios

An optimal solution

Problem complexity

Computational problem

Computational complexity of an algorithm

Exact solution

Approximate solution

Permutation

Combination

Repetition

Distinct objects

Binomial theorem

Binomial coefficient

Pascal’s triangle