UNIVERSITY OF AMSTERDAM

Finding the optimal route in a road traffic network

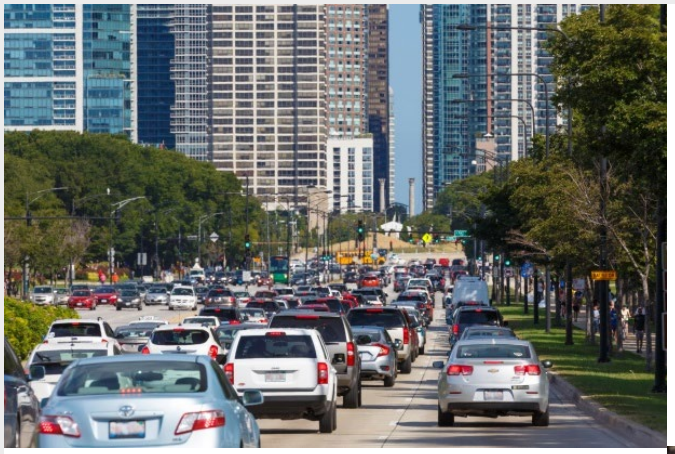
NETWORKS goes to school
April 5th, 2023

Nikki Levering
Nicos Starreveld
Mehmet Akif Yildiz

NWO

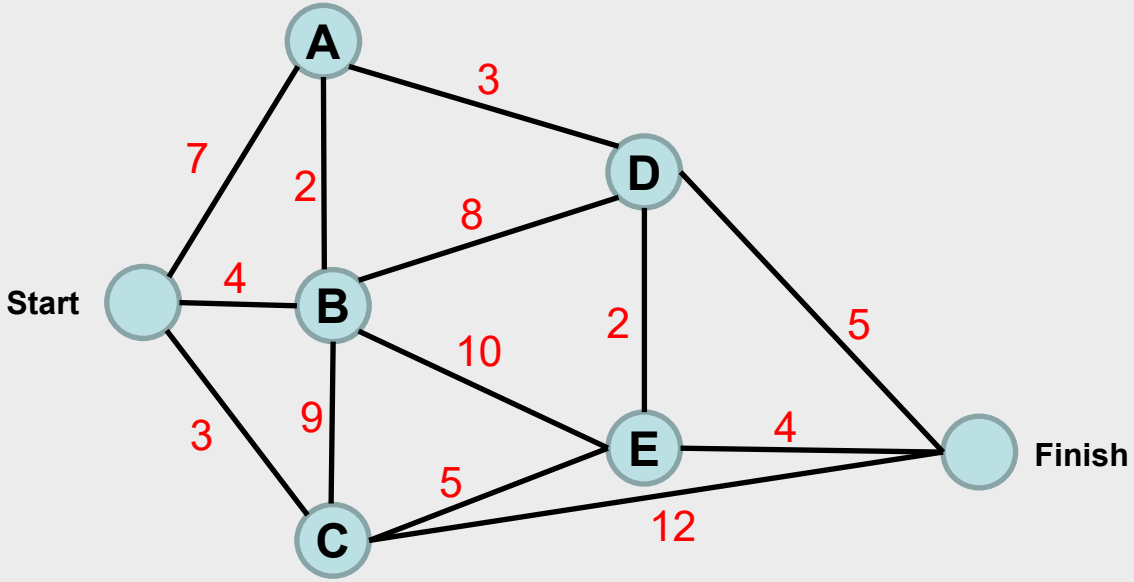
NET WORKS
THENETWORKCENTER.NL

Why?

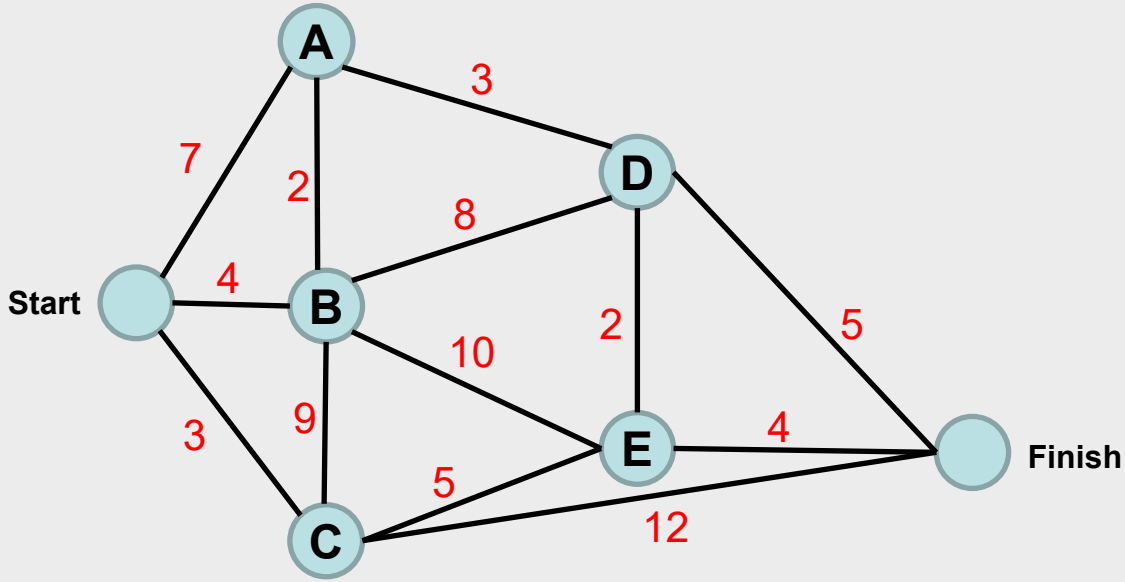


postnl

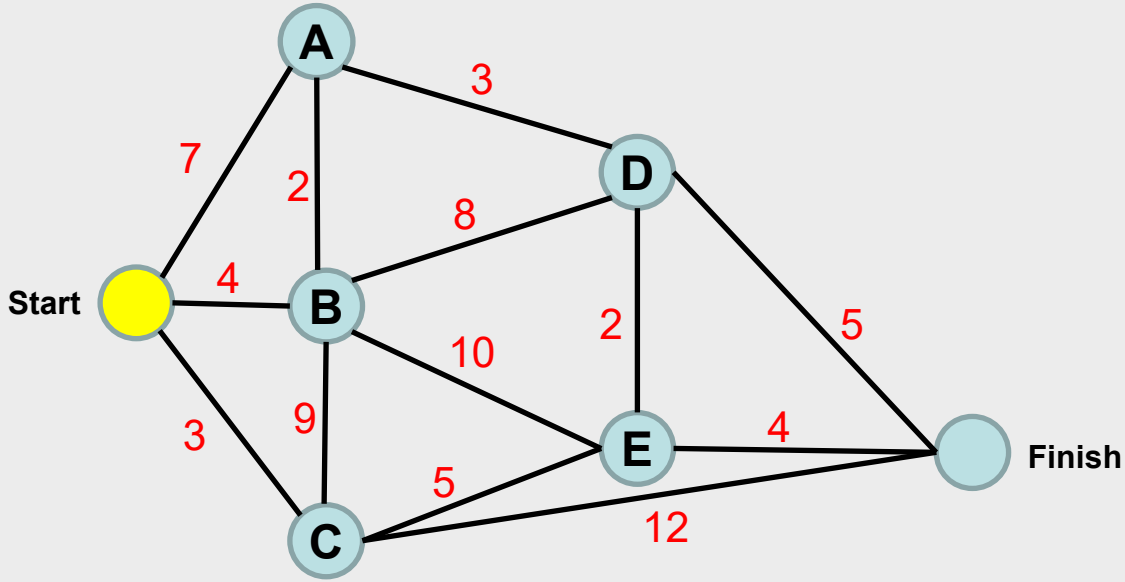
Thuisbezorgd.nl



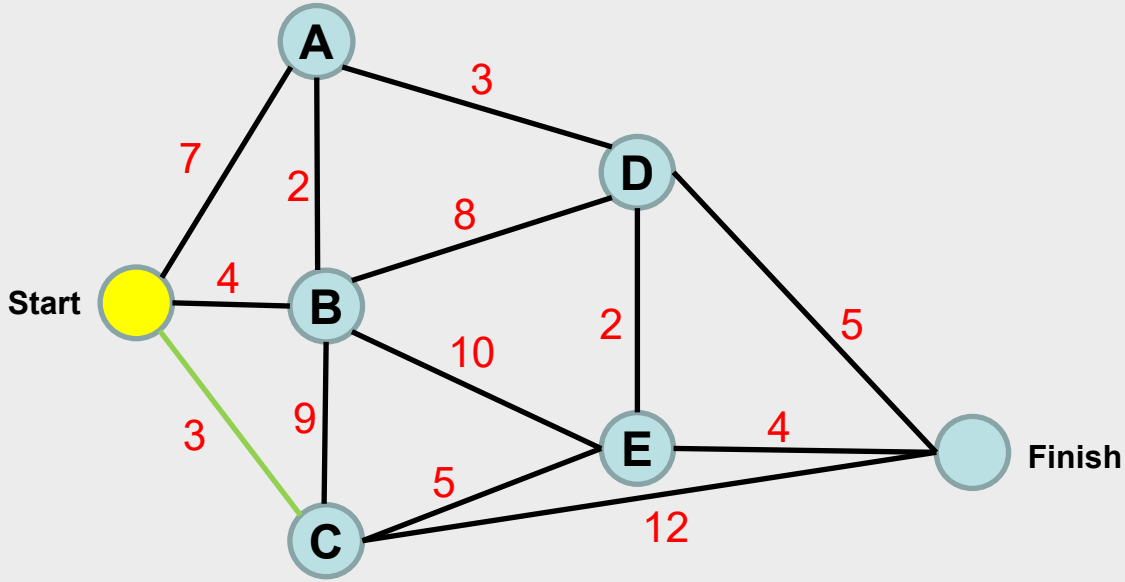
- What is the shortest route from start to finish?
- Dijkstra's algorithm!



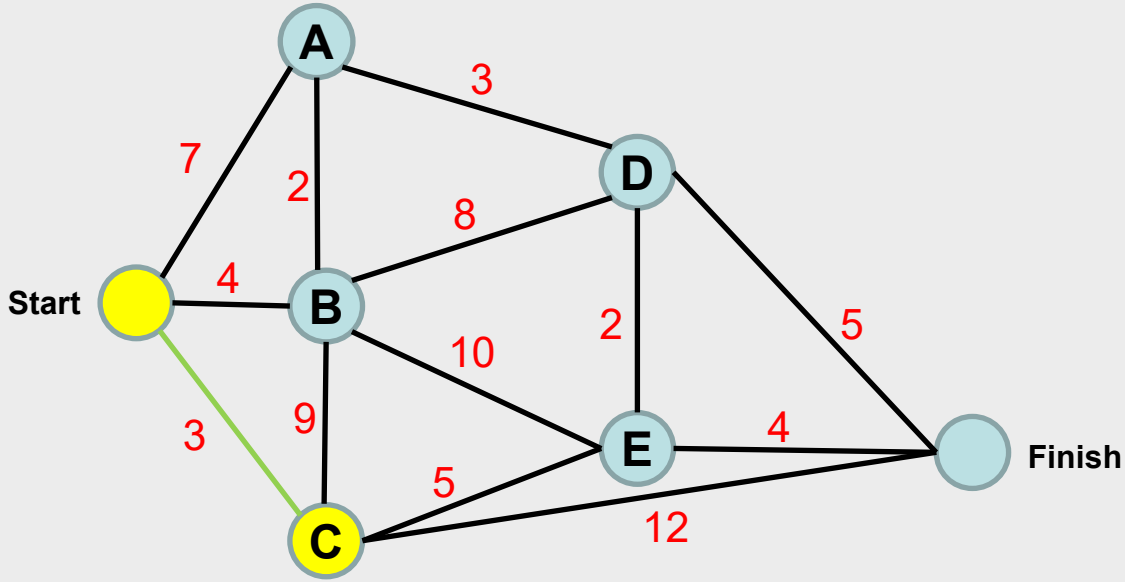
A	B	C	D	E	Finish



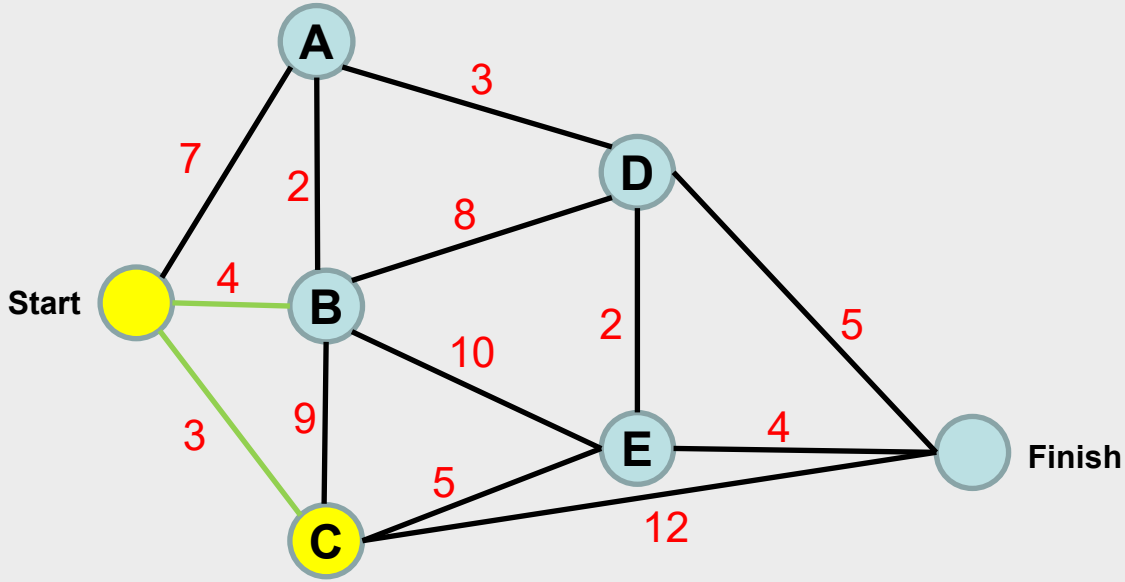
A	B	C	D	E	Finish
7, start	4, start	3, start			



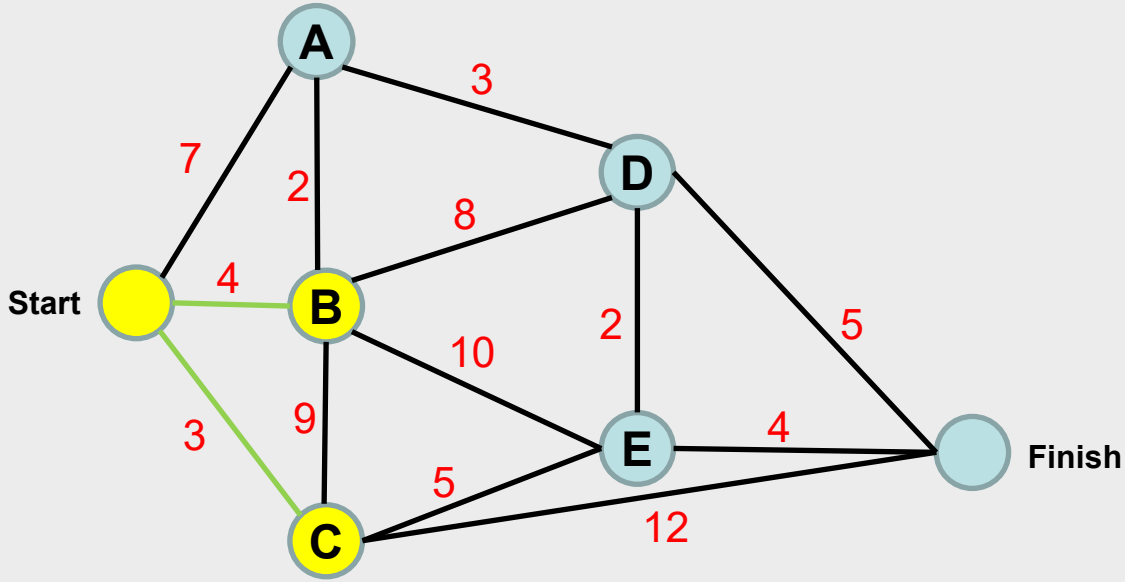
A	B	C	D	E	Finish
7, start	4, start	**3, start**			



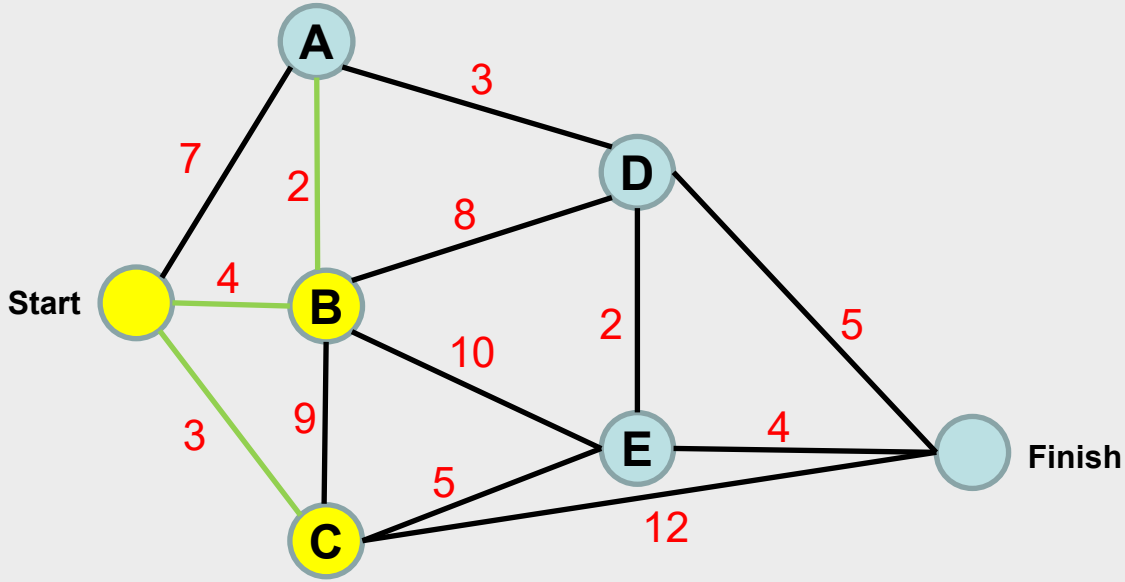
A	B	C	D	E	Finish
7, start	4, start	**3, start**			
7, start	4, start	*		8, C	15, C
		*			
		*			
		*			
		*			



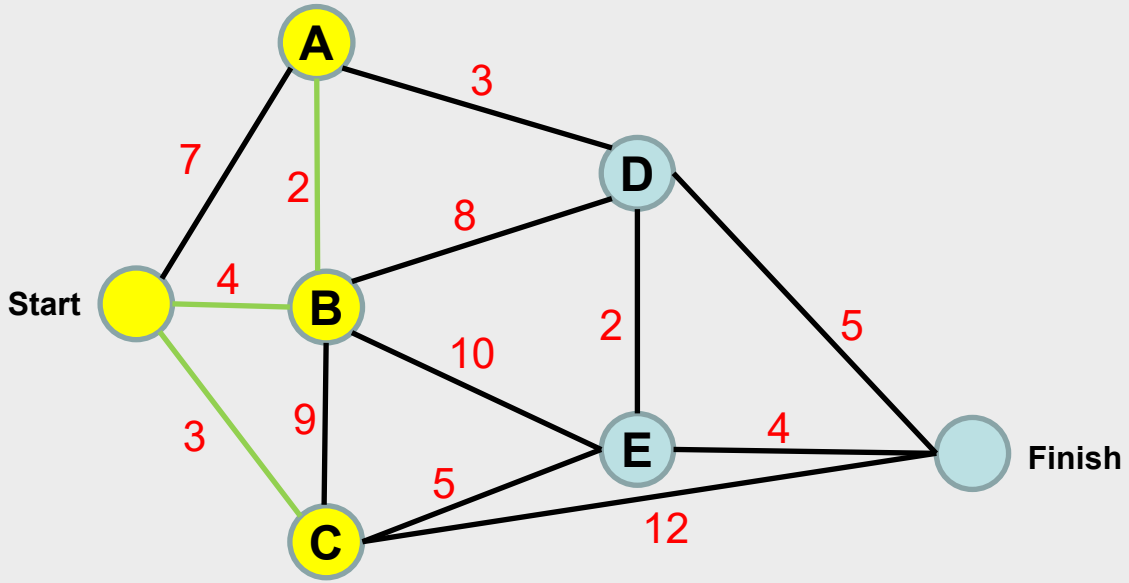
A	B	C	D	E	Finish
7, start	4, start	**3, start**			
7, start	**4, start**	*		8, C	15, C
		*			
		*			
		*			
		*			



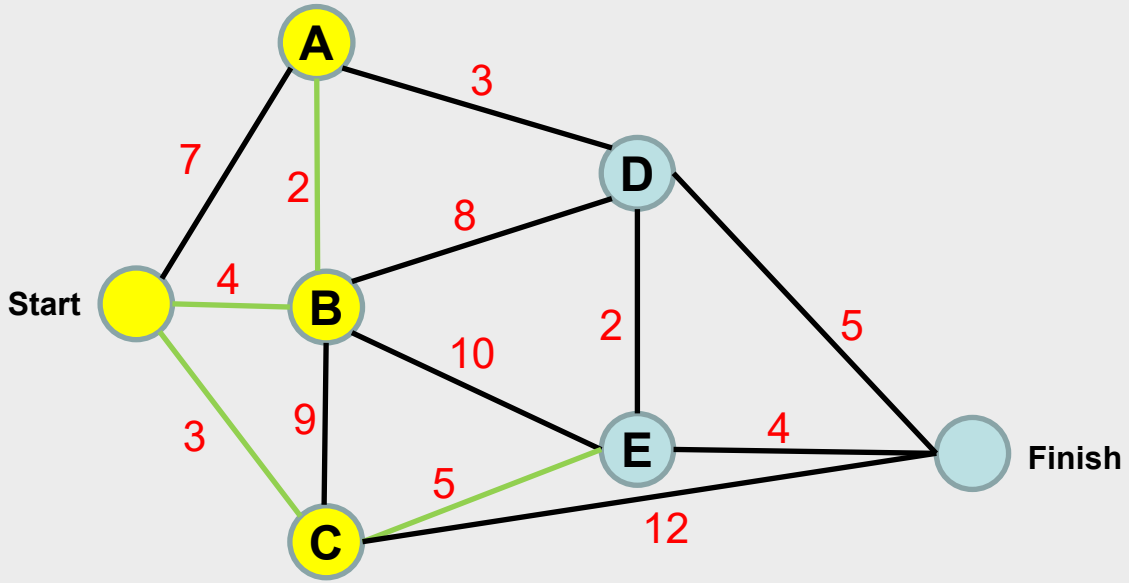
A	B	C	D	E	Finish
7, start	4, start	**3, start**			
7, start	**4, start**	*		8, C	15, C
6, B	*	*	12, B	8, C	15, C
	*	*			
	*	*			
	*	*			



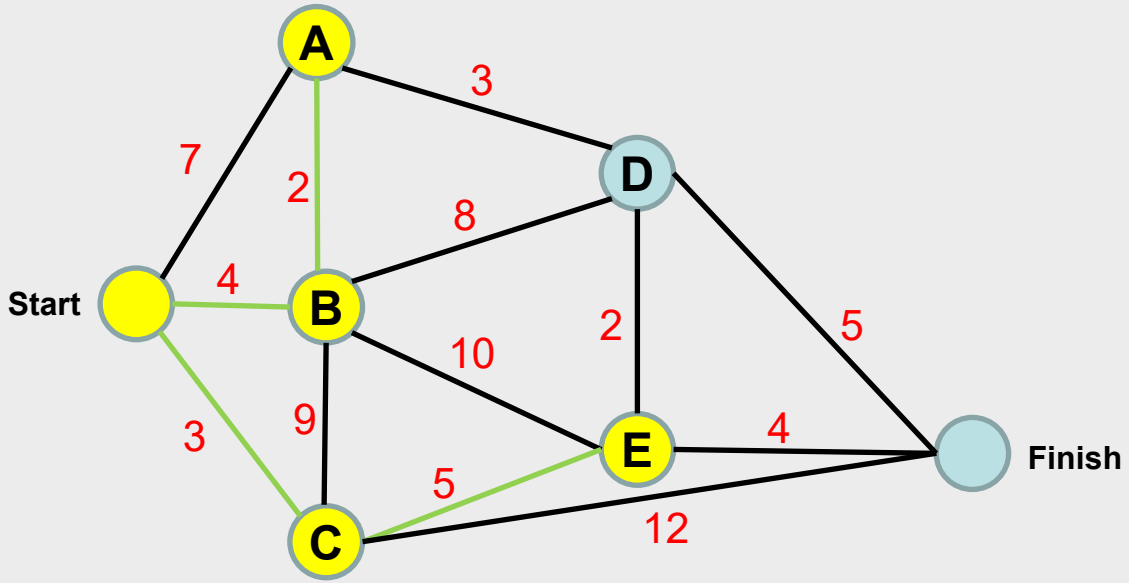
A	B	C	D	E	Finish
7, start	4, start	**3, start**			
7, start	**4, start**	*		8, C	15, C
6, B	*	*	12, B	8, C	15, C
	*	*			
	*	*			
	*	*			



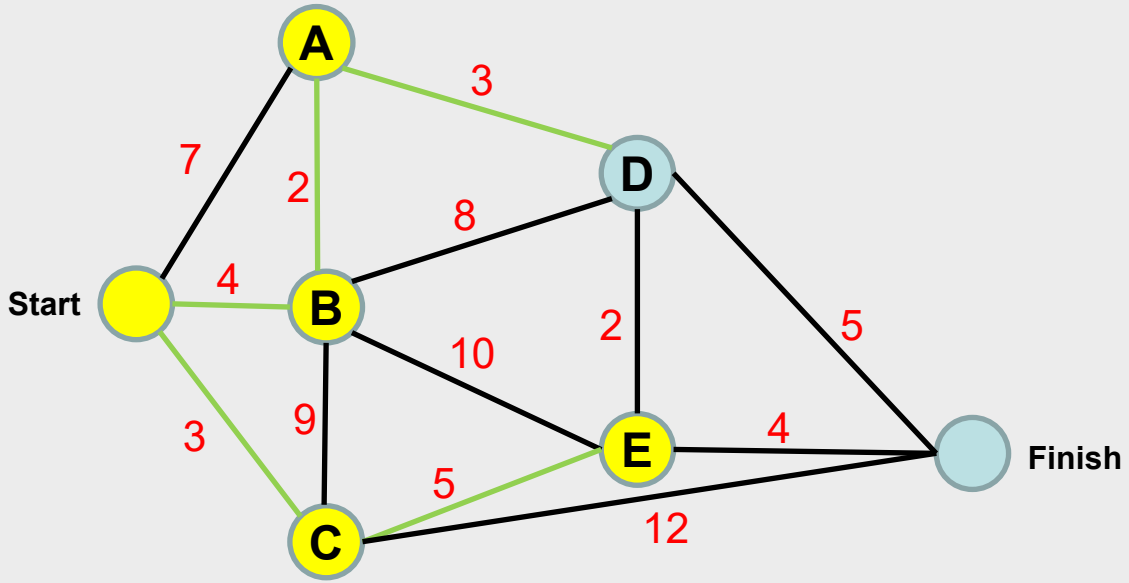
A	B	C	D	E	Finish
7, start	4, start	**3, start**			
7, start	**4, start**	*		8, C	15, C
6, B	*	*	12, B	8, C	15, C
*	*	*	9, A	8, C	15, C
*	*	*			
*	*	*			



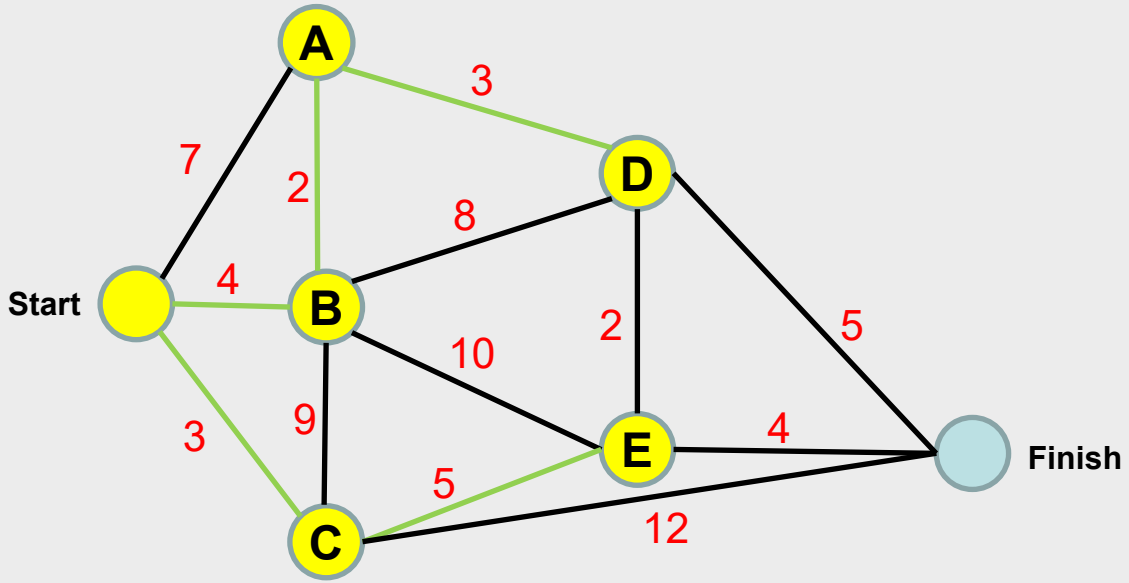
A	B	C	D	E	Finish
7, start	4, start	**3, start**			
7, start	**4, start**	*		8, C	15, C
6, B	*	*	12, B	8, C	15, C
*	*	*	9, A	**8, C**	15, C
*	*	*			
*	*	*			



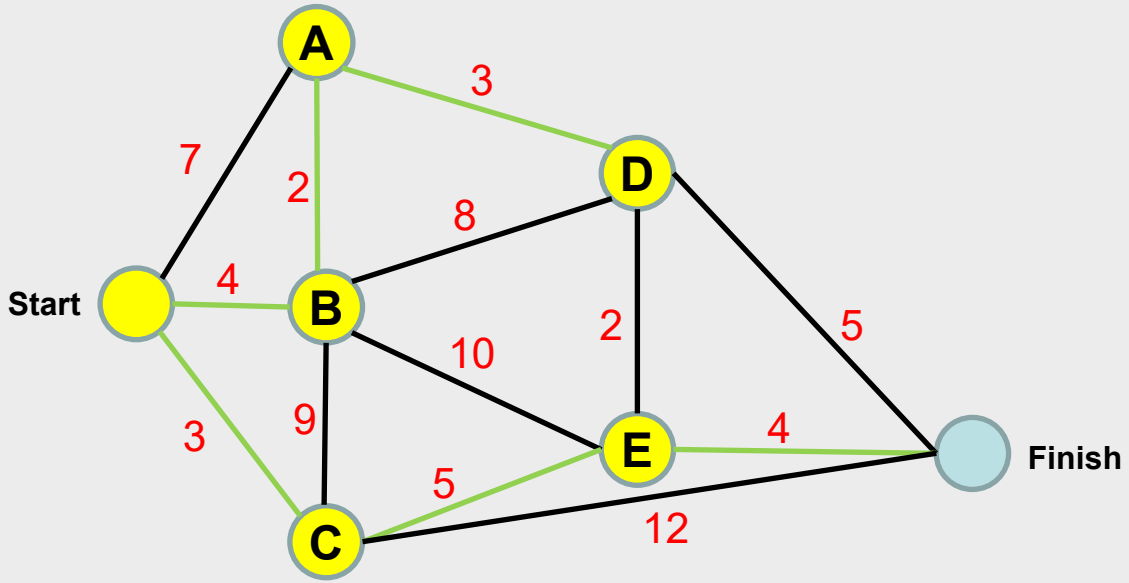
A	B	C	D	E	Finish
7, start	4, start	**3, start**			
7, start	**4, start**	*		8, C	15, C
6, B	*	*	12, B	8, C	15, C
*	*	*	9, A	**8, C**	15, C
*	*	*	9, A	*	12, E
*	*	*		*	



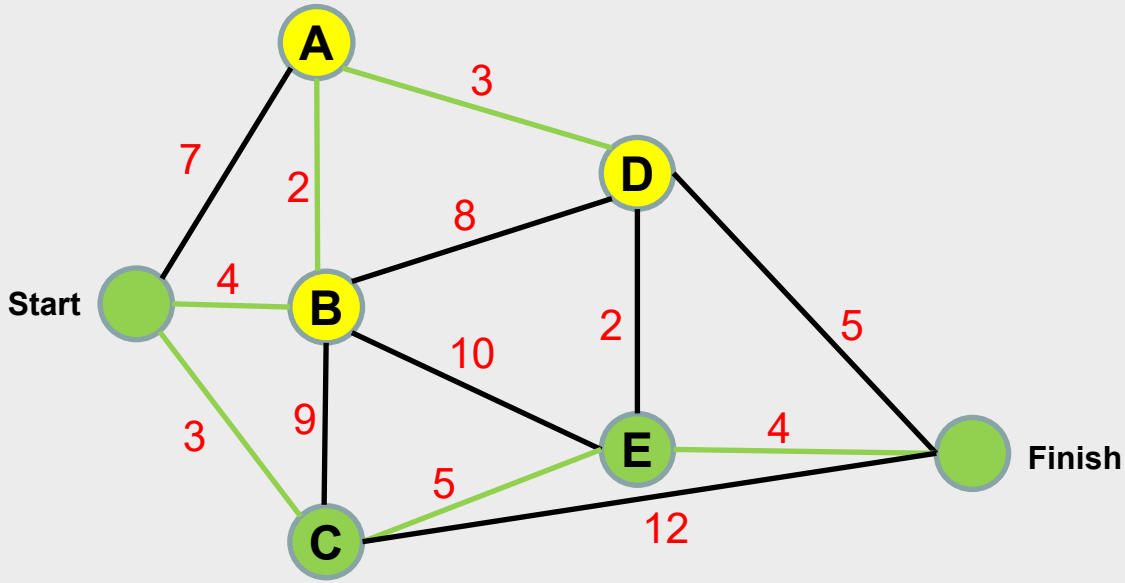
A	B	C	D	E	Finish
7, start	4, start	**3, start**			
7, start	**4, start**	*		8, C	15, C
6, B	*	*	12, B	8, C	15, C
*	*	*	9, A	**8, C**	15, C
*	*	*	**9, A**	*	12, E
*	*	*		*	



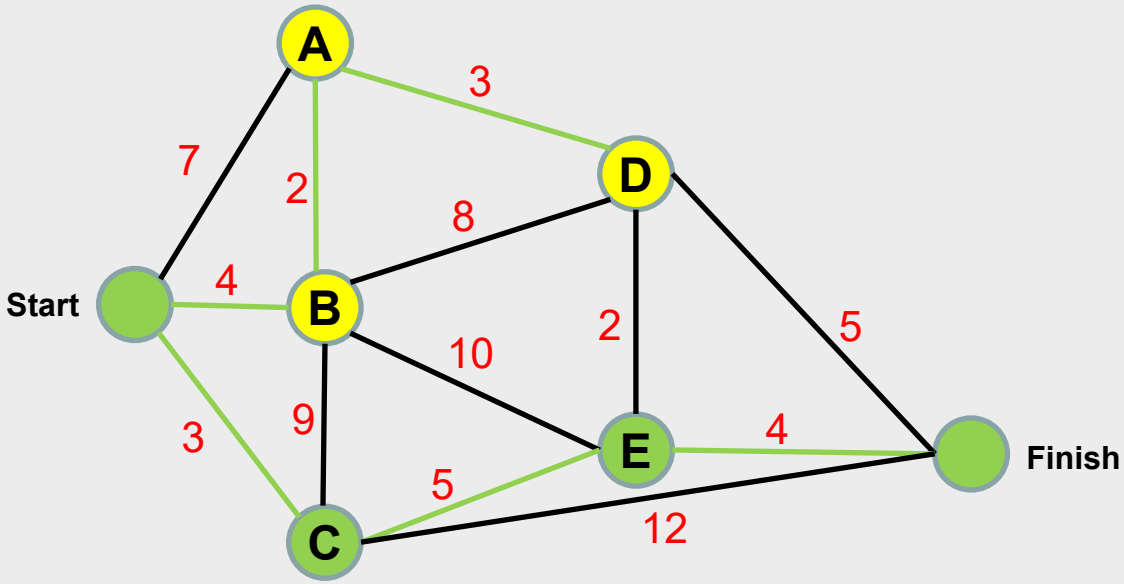
A	B	C	D	E	Finish
7, start	4, start	**3, start**			
7, start	**4, start**	*		8, C	15, C
6, B	*	*	12, B	8, C	15, C
*	*	*	9, A	**8, C**	15, C
*	*	*	**9, A**	*	12, E
*	*	*	*	*	12, E



A	B	C	D	E	Finish
7, start	4, start	**3, start**			
7, start	**4, start**	*		8, C	15, C
6, B	*	*	12, B	8, C	15, C
*	*	*	9, A	**8, C**	15, C
*	*	*	**9, A**	*	12, E
*	*	*	*	*	**12, E**



A	B	C	D	E	Finish
7, start	4, start	**3, start**			
7, start	**4, start**	*		8, C	15, C
6, B	*	*	12, B	8, C	15, C
*	*	*	9, A	**8, C**	15, C
*	*	*	**9, A**	*	12, E
*	*	*	*	*	**12, E**



- The shortest route is given by: Start→C→E→Finish

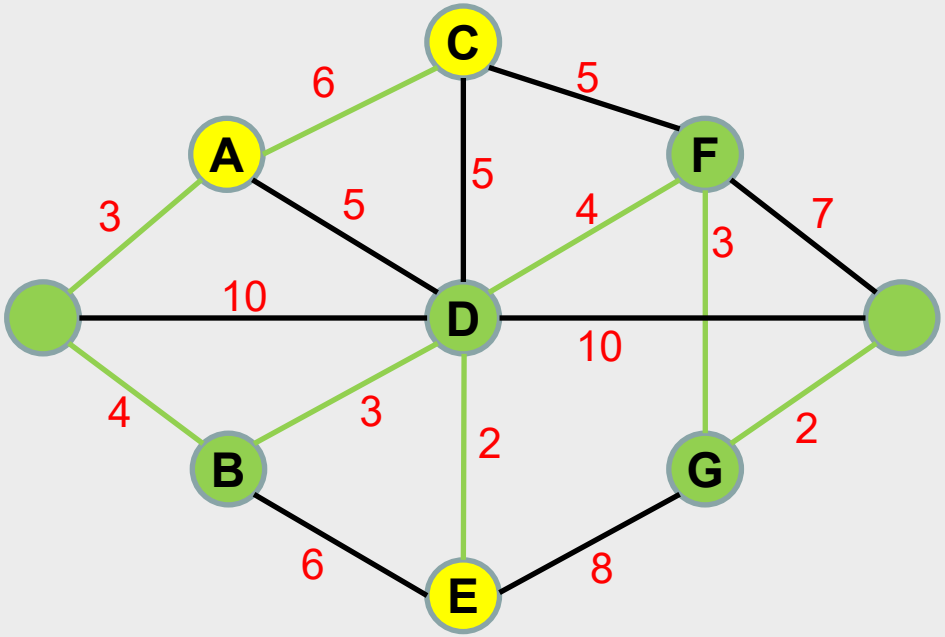
Do

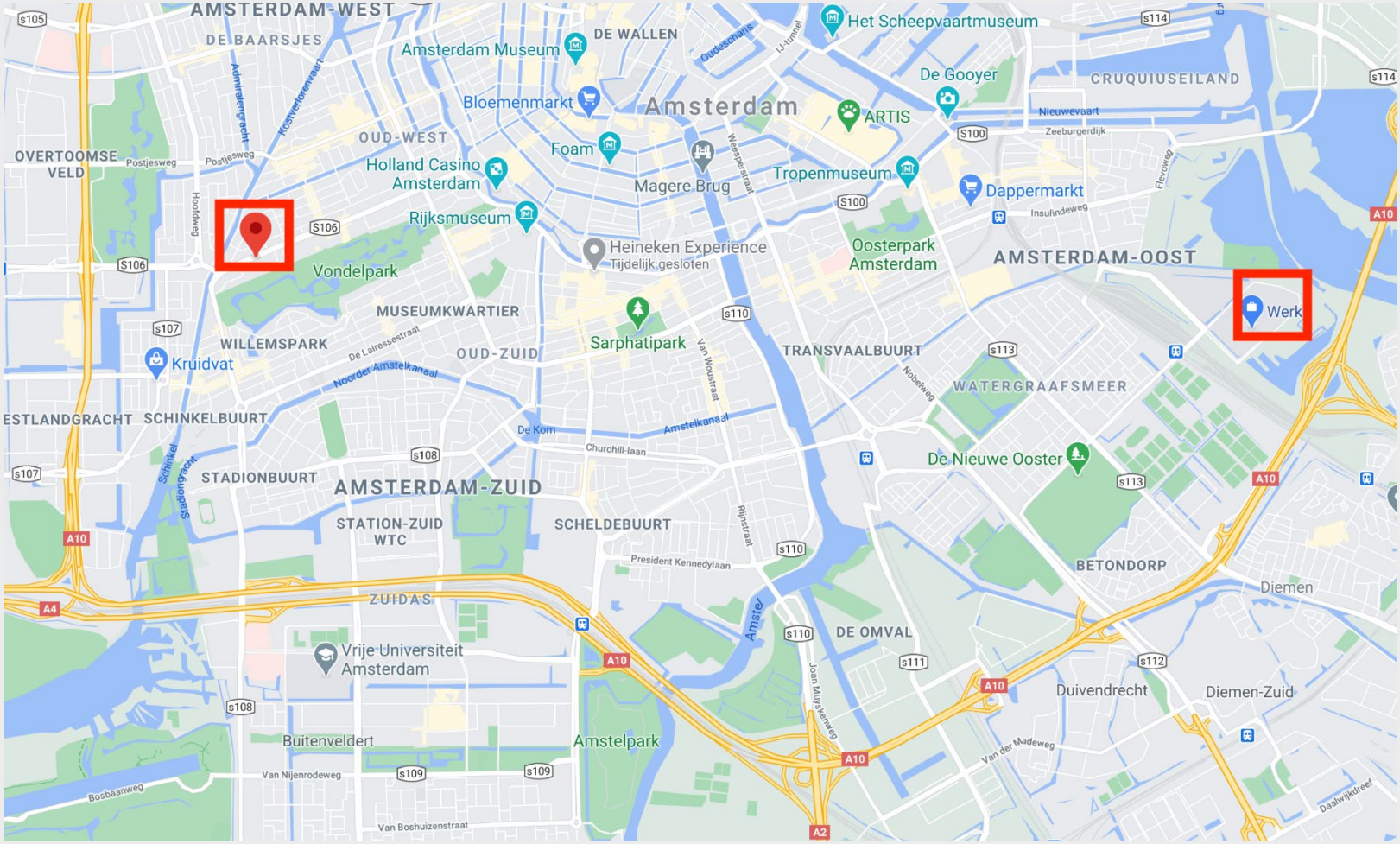
use Dijkstra's algorithm?

Not in its current form…

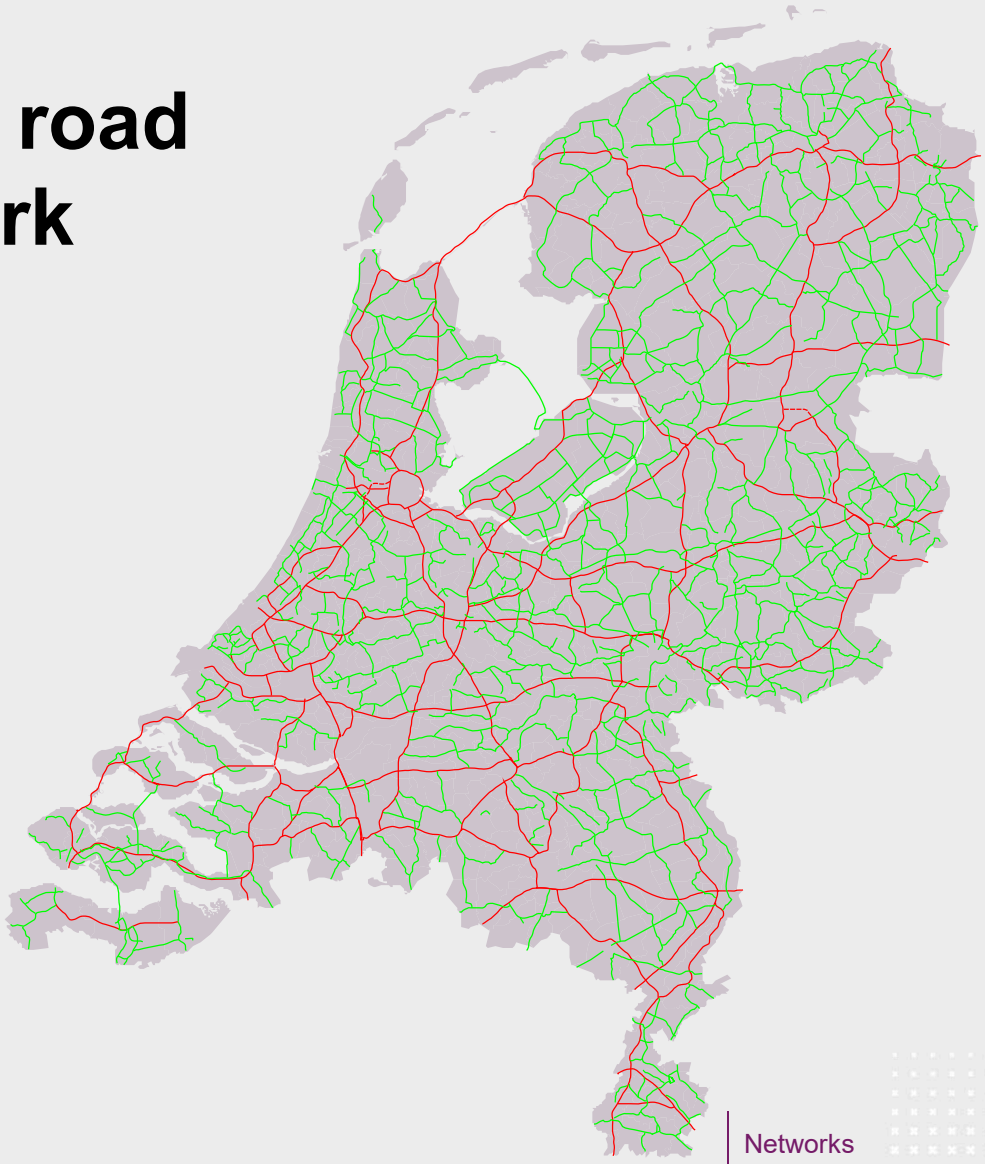
Why?

1. Too slow


C
A
F
D
B
G
E
6
5
3
5
5
4
3
7
10
10
4
3
2
2
6
8


AMSTERDAM-WEST
DE BAARSJES
DE WALLEN
Amsterdam Museum
Het Scheepvaartmuseum
De Gooyer
CRUQUIUSEILAND
Bloemenmarkt
Amsterdam
ARTIS
OUD-WEST
Foam
OVERTOOMSE VELD
Holland Casino Amsterdam
Magere Brug
Tropenmuseum
Dappermarkt
Rijksmuseum
Heineken Experience
Tijdelijk gesloten
Oosterpark Amsterdam
AMSTERDAM-OOST
Vondelpark
Werk
MUSEUMKWARTIER
Sarphatipark
WILLEMSPARK
OUD-ZUID
TRANSVAALBUURT
Kruidvat
WATERGRAAFSMEER
SCHINKELBUURT
De Nieuwe Ooster
STADIONBUURT
AMSTERDAM-ZUID
STATION-ZUID WTC
SCHELDEBUURT
BETONDORP
Diemen
ZUIDAS
DE OMVAL
Vrije Universiteit Amsterdam
Duivendrecht
Diemen-Zuid
Buitenveldert
Amstelpark

Dutch road network



Not in its current form…

Why?

1. Too slow
2. Limited to shortest distances

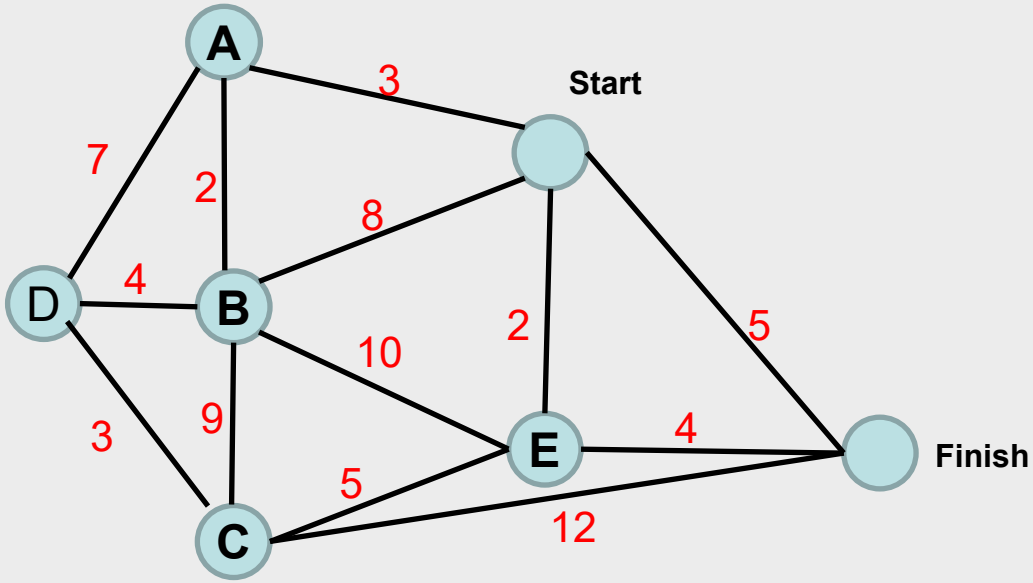
Drivers may have different wishes:

- What is the fastest route to reach my destination?
- What route is least likely to suffer from traffic jams?
- What route should I take to arrive at my destination *in time*?

Today's plan:

- Speeding-up Dijkstra's algorithm
- Extending Dijkstra's algorithm beyond the simple shortest path (i.e. in distance) setting.

Weaknesses of Dijkstra's algorithm



DEMO

How can we improve Dijkstra's algorithm (in terms of speed)?

Take the location of the destination into account!

For example:

- Bi-directional Dijkstra
- A-star algorithm

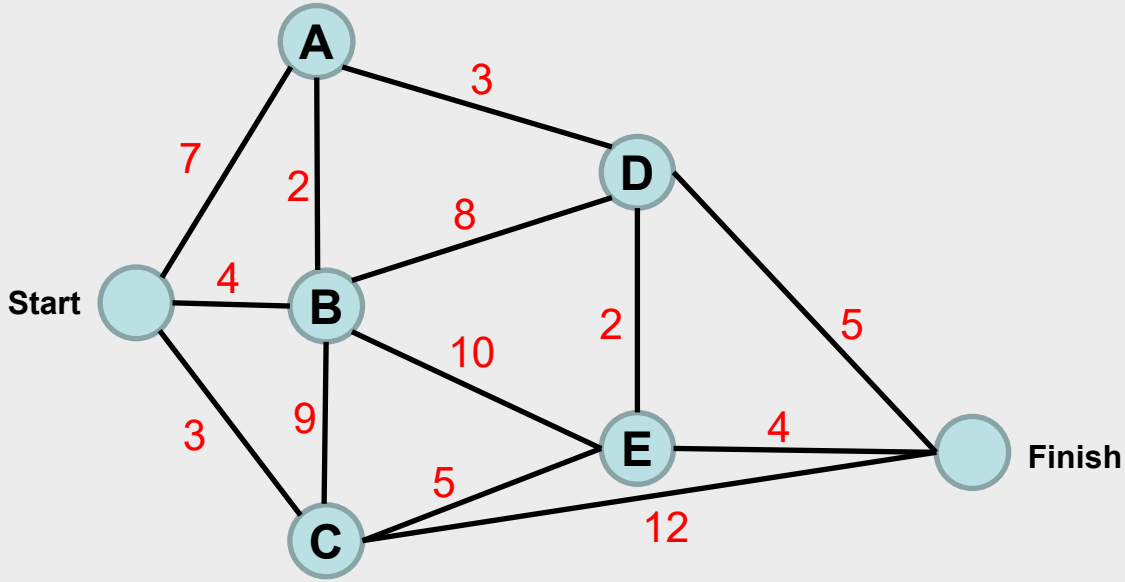
Bi-directional Dijkstra (“=” two-sided)

- Idea: run Dijkstra’s algorithm twice, with

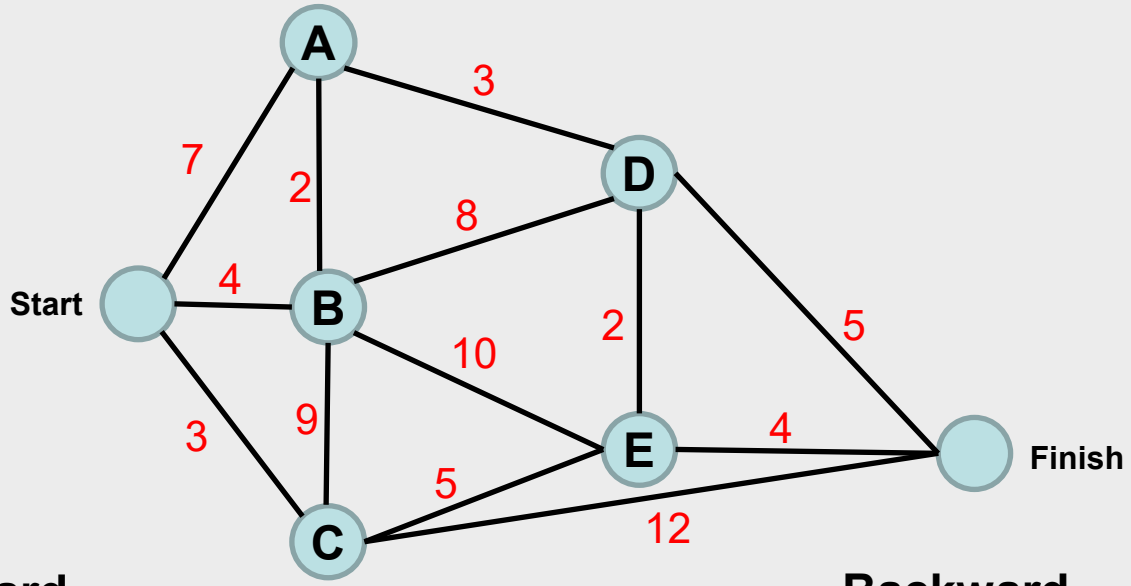
  - origin as starting node
  - destination as starting node

  Stop when they ‘meet’.

DEMO



A	B	C	D	E	Finish

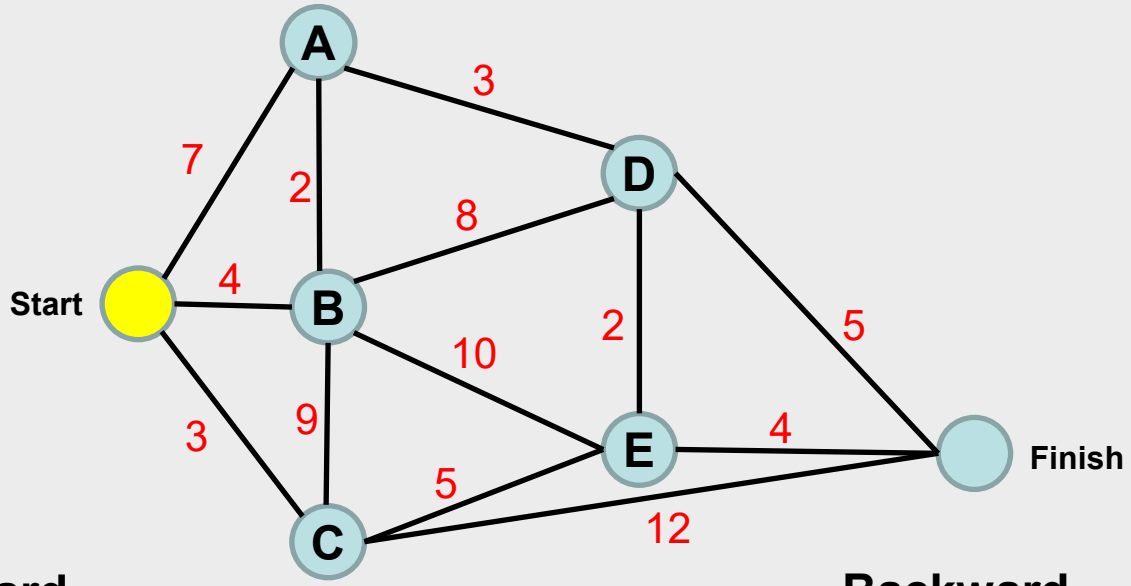


Forward

A	B	C	D	E	Finish

Backward

A	B	C	D	E	Start

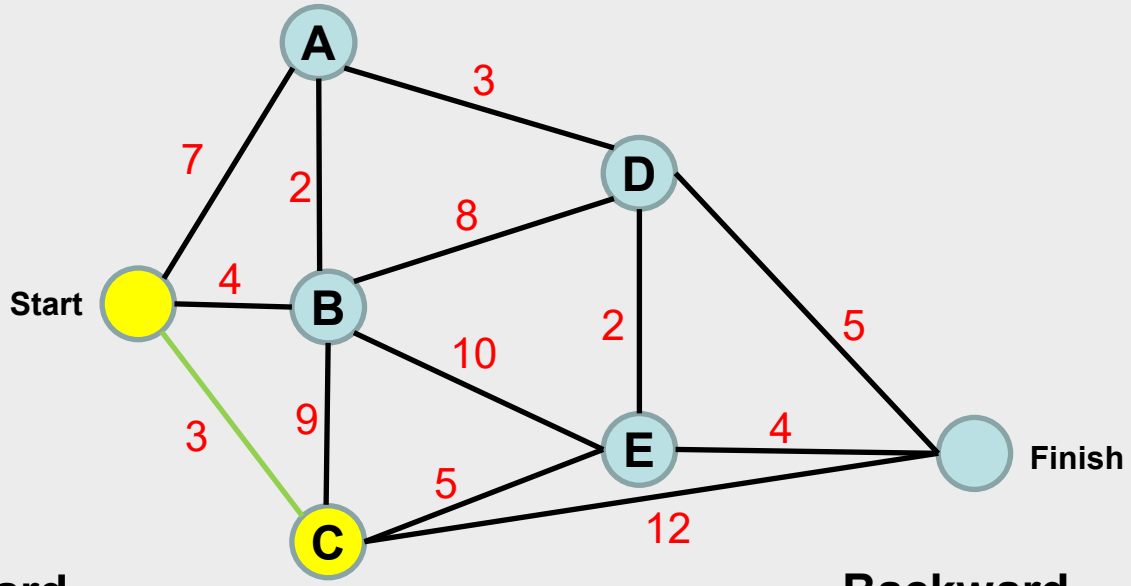


Forward

A	B	C	D	E	Finish
7,s	4,s	3,s			

Backward

A	B	C	D	E	Start

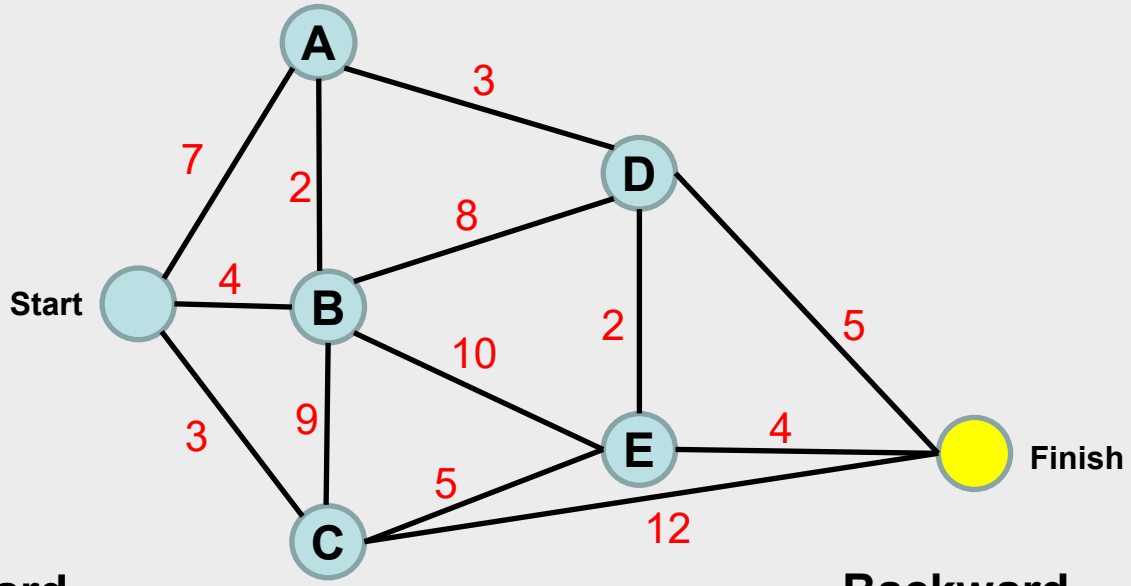


Forward

A	B	C	D	E	Finish
7,s	4,s	**3,s**			

Backward

A	B	C	D	E	Start

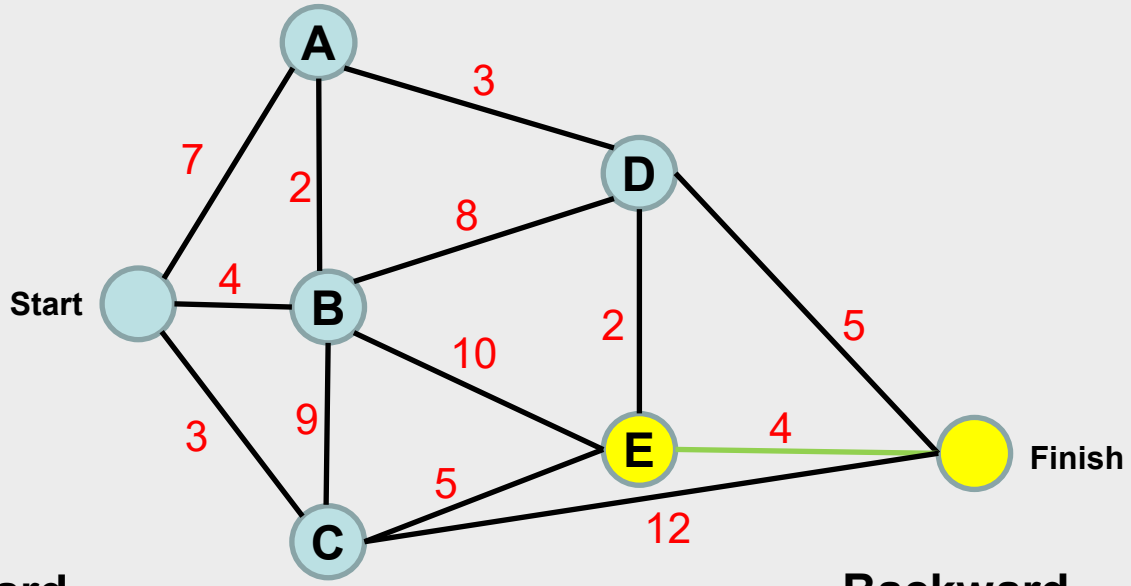


Forward

A	B	C	D	E	Finish
7,s	4,s	**3,s**			

Backward

A	B	C	D	E	Start
		12,f	5,f	4,f	

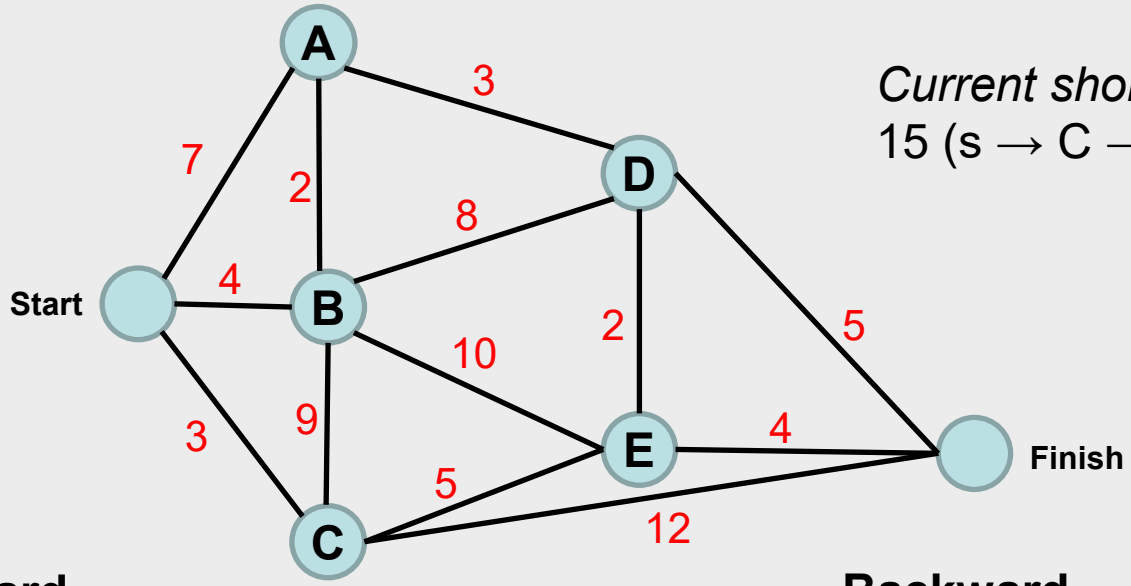


Forward

A	B	C	D	E	Finish
7,s	4,s	**3,s**			

Backward

A	B	C	D	E	Start
		12,f	5,f	**4,f**	



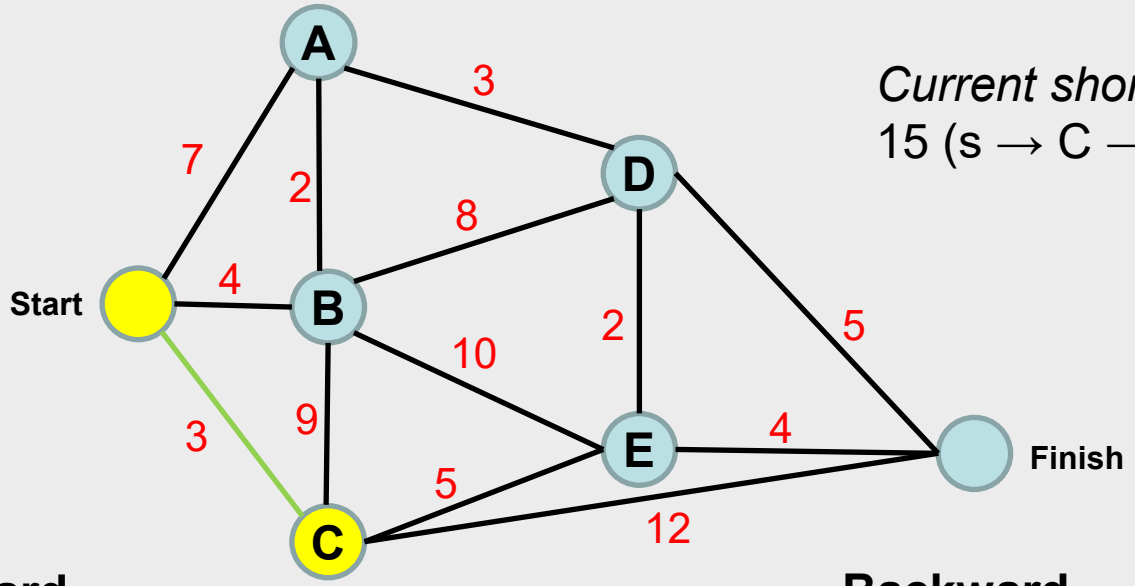
Current shortest path:
15 (s → C → f)

Forward

A	B	C	D	E	Finish
7,s	4,s	**3,s**			

Backward

A	B	C	D	E	Start
		12,f	5,f	**4,f**	



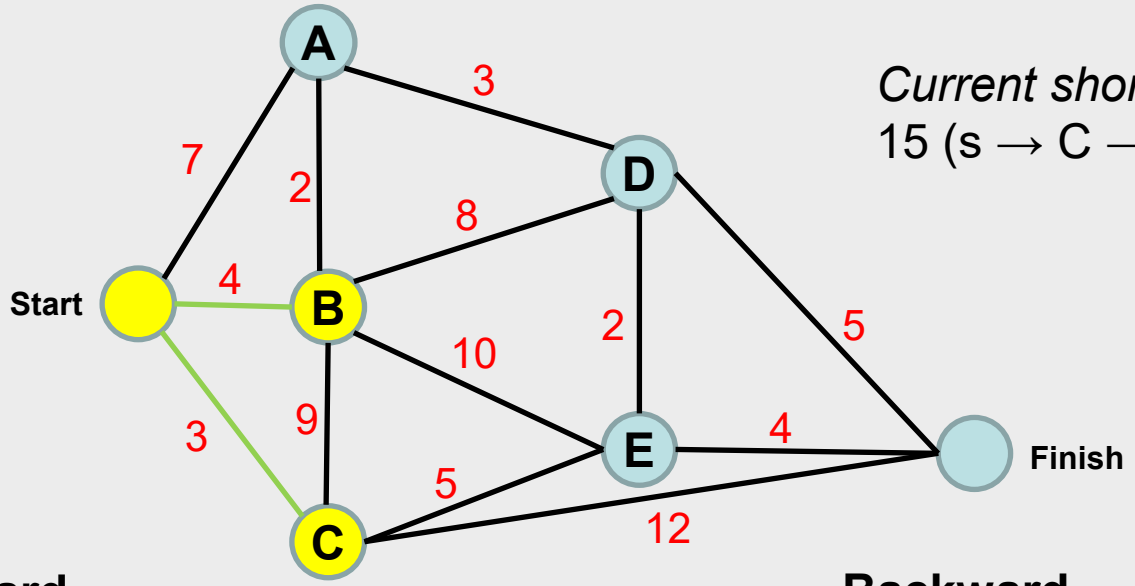
Current shortest path:
15 (s → C → f)

Forward

A	B	C	D	E	Finish
7,s	4,s	**3,s**			
7,s	4,s	*		8,C	15,C
		*			
		*			
		*			
		*			

Backward

A	B	C	D	E	Start
		12,f	5,f	**4,f**	



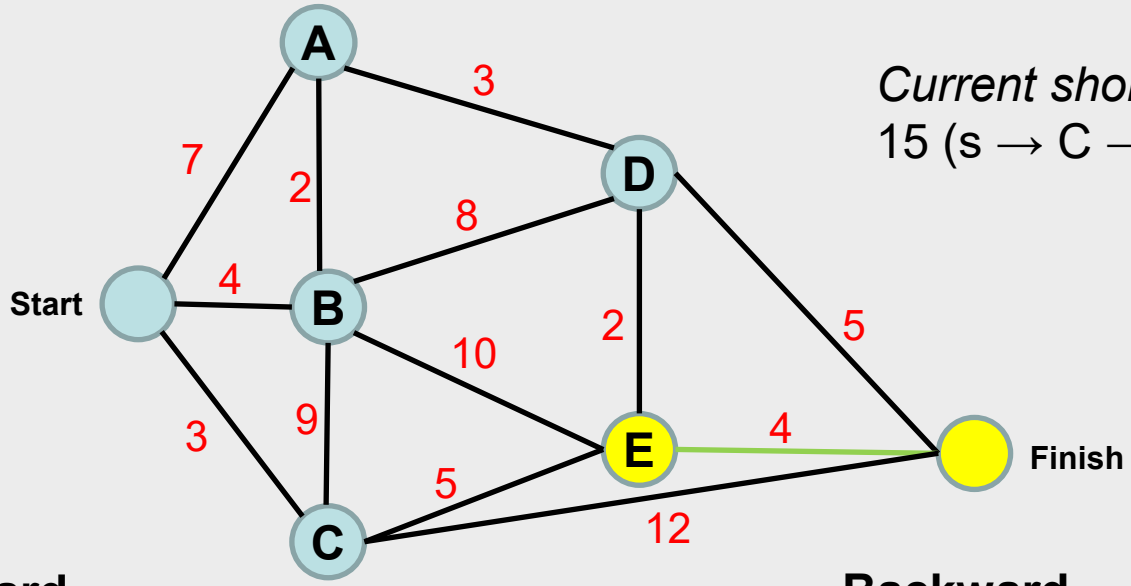
Current shortest path:
15 (s → C → f)

Forward

A	B	C	D	E	Finish
7,s	4,s	**3,s**			
7,s	**4,s**	*		8,C	15,C
		*			
		*			
		*			
		*			

Backward

A	B	C	D	E	Start
		12,f	5,f	**4,f**	



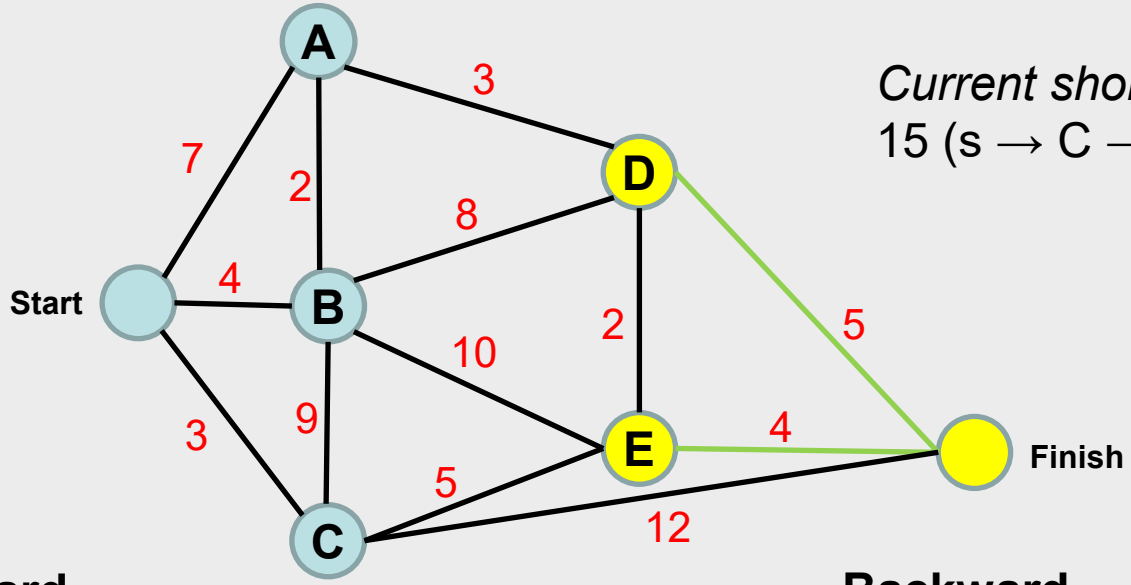
Current shortest path:
15 (s → C → f)

Forward

A	B	C	D	E	Finish
7,s	4,s	**3,s**			
7,s	**4,s**	*		8,C	15,C
		*			
		*			
		*			
		*			

Backward

A	B	C	D	E	Start
		12,f	5,f	**4,f**	
	14,E	9,E	5,f	*	
				*	
				*	
				*	
				*	



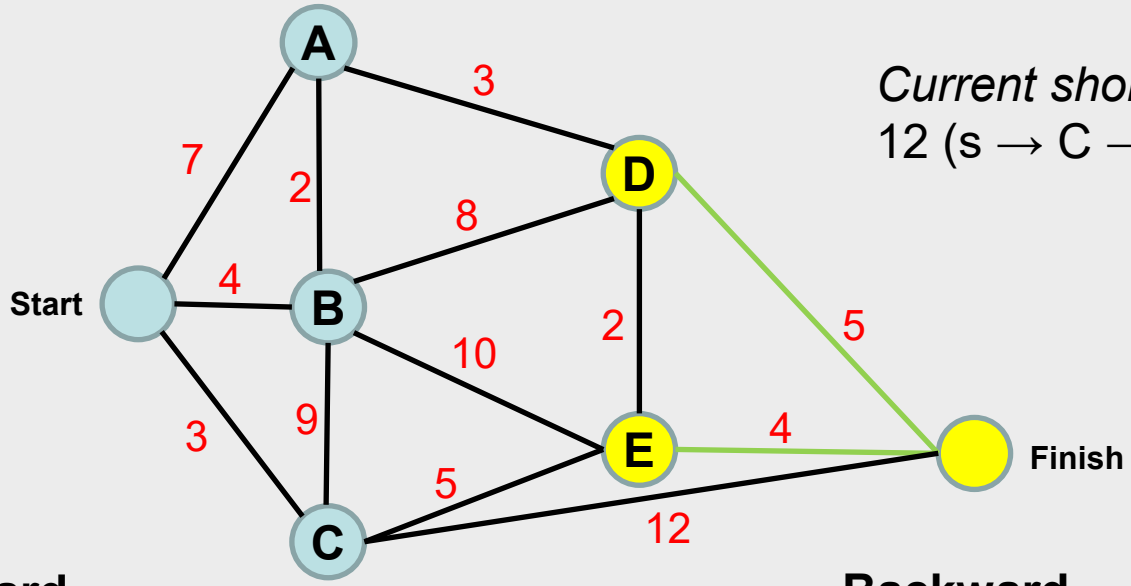
Current shortest path:
15 (s → C → f)

Forward

A	B	C	D	E	Finish
7,s	4,s	**3,s**			
7,s	**4,s**	*		8,C	15,C
		*			
		*			
		*			
		*			

Backward

A	B	C	D	E	Start
		12,f	5,f	**4,f**	
	14,E	9,E	**5,f**	*	
				*	
				*	
				*	
				*	



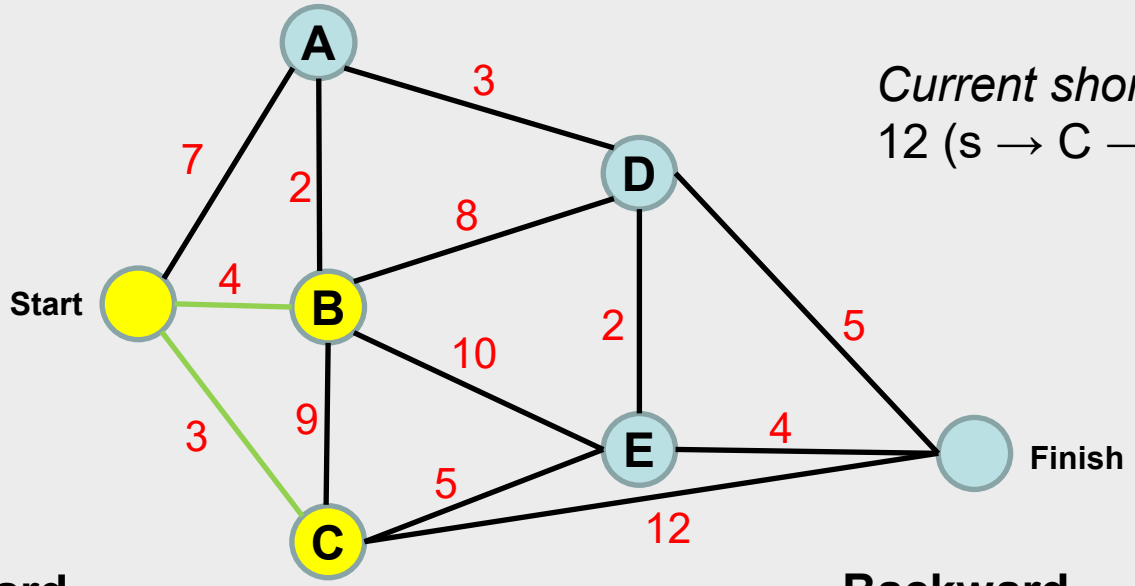
Current shortest path:
12 (s → C → E → f)

Forward

A	B	C	D	E	Finish
7,s	4,s	**3,s**			
7,s	**4,s**	*		8,C	15,C
		*			
		*			
		*			
		*			

Backward

A	B	C	D	E	Start
		12,f	5,f	**4,f**	
	14,E	9,E	**5,f**	*	
				*	
				*	
				*	
				*	



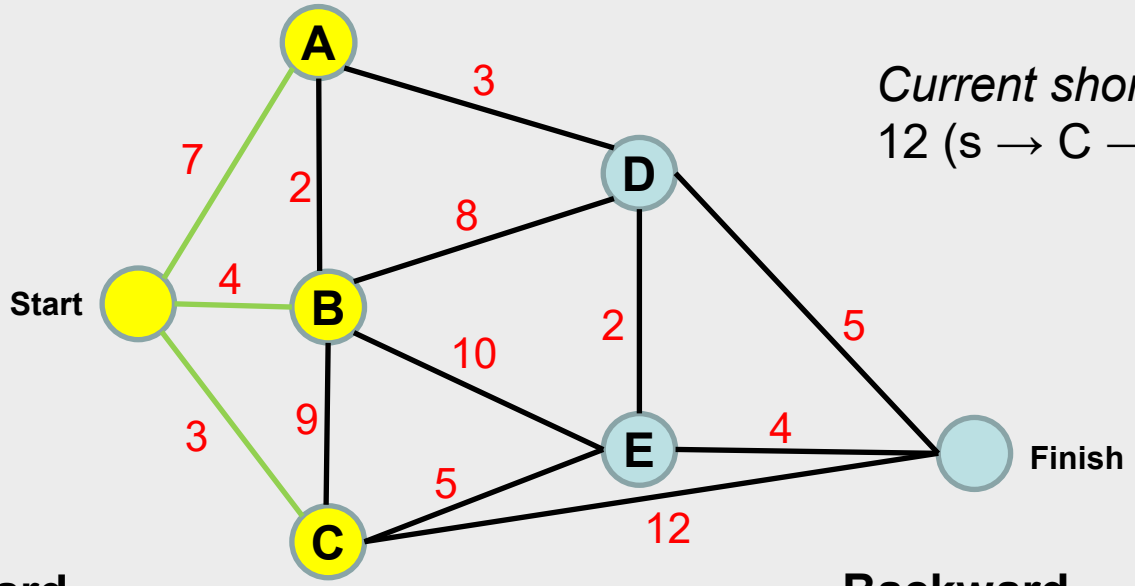
Current shortest path:
12 (s → C → E → f)

Forward

A	B	C	D	E	Finish
7,s	4,s	**3,s**			
7,s	**4,s**	*		8,C	15,C
6,B	*	*	12,B	8,C	15,C
	*	*			
	*	*			
	*	*			

Backward

A	B	C	D	E	Start
		12,f	5,f	**4,f**	
	14,E	9,E	**5,f**	*	
				*	
				*	
				*	
				*	



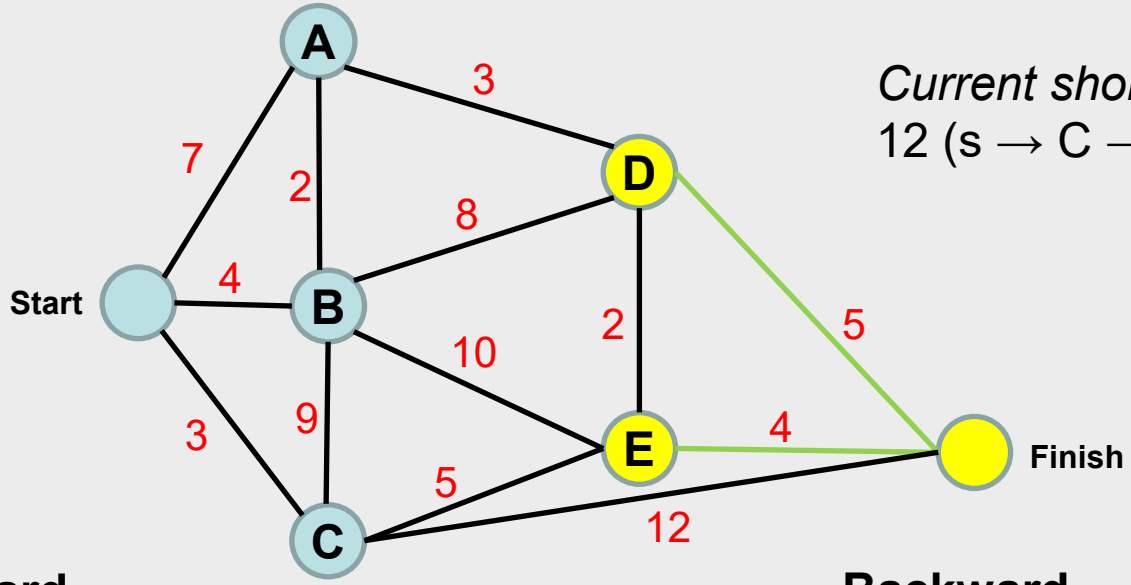
Current shortest path:
12 (s → C → E → f)

Forward

A	B	C	D	E	Finish
7,s	4,s	**3,s**			
7,s	**4,s**	*		8,C	15,C
6,B	*	*	12,B	8,C	15,C
	*	*			
	*	*			
	*	*			

Backward

A	B	C	D	E	Start
		12,f	5,f	**4,f**	
	14,E	9,E	**5,f**	*	
				*	
				*	
				*	
				*	



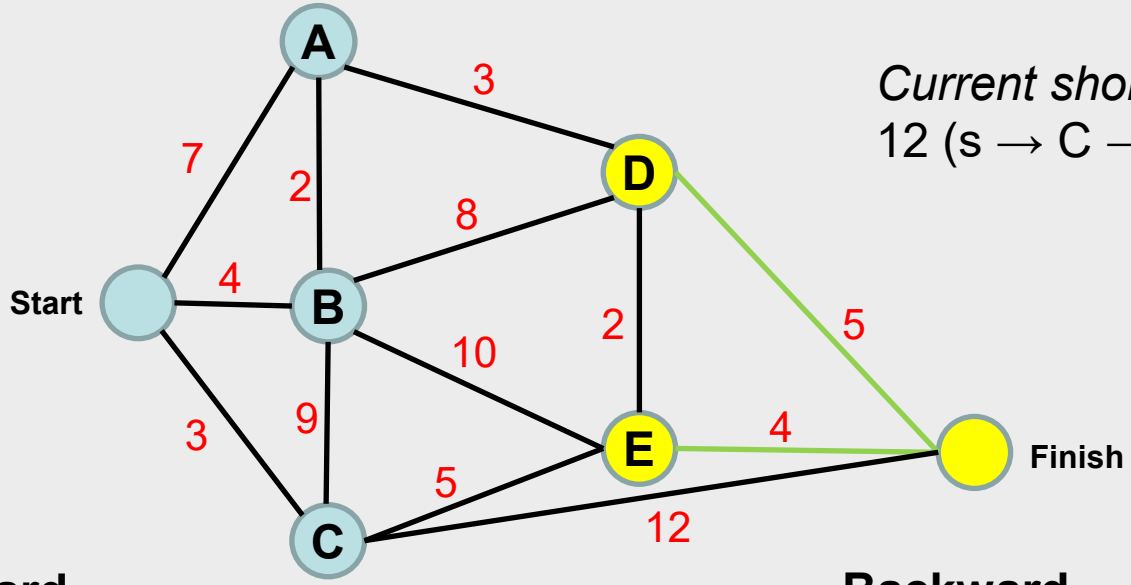
Current shortest path:
12 (s → C → E → f)

Forward

A	B	C	D	E	Finish
7,s	4,s	**3,s**			
7,s	**4,s**	*		8,C	15,C
6,B	*	*	12,B	8,C	15,C
	*	*			
	*	*			
	*	*			

Backward

A	B	C	D	E	Start
		12,f	5,f	**4,f**	
	14,E	9,E	**5,f**	*	
8,D	13, D	9,E	*	*	
			*	*	
			*	*	
			*	*	



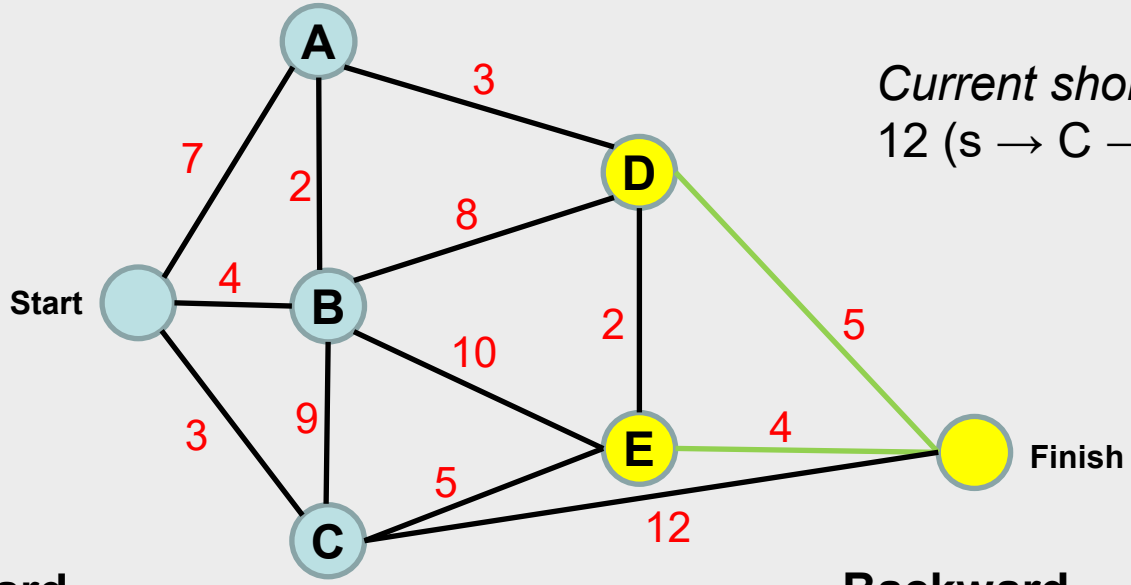
Current shortest path:
12 (s → C → E → f)

Forward

A	B	C	D	E	Finish
7,s	4,s	**3,s**			
7,s	**4,s**	*		8,C	15,C
6,B	*	*	12,B	8,C	15,C
	*	*			
	*	*			
	*	*			

Backward

A	B	C	D	E	Start
		12,f	5,f	**4,f**	
	14,E	9,E	**5,f**	*	
8,D	13, D	9,E	*	*	
			*	*	
			*	*	
			*	*	



Current shortest path:
12 (s → C → E → f)

Forward

A	B	C	D	E	Finish
7,s	4,s	**3,s**			
7,s	**4,s**	*		8,C	15,C
6,B	*	*	12,B	8,C	15,C
	*	*			
	*	*			
	*	*			

Backward

A	B	C	D	E	Start
		12,f	5,f	**4,f**	
	14,E	9,E	**5,f**	*	
8,D	13, D	9,E	*	*	
			*	*	
			*	*	
			*	*	

Claim: we have found the optimal path when the sum of the distances of the **new permanent nodes** is **longer** than the shortest path we have already found.

***Proof by contradiction*:** assume not true, and then show that this will give an error.

Idea: We look at the forward and the backward table. From now on every new permanent node will either have a dinstance longer than 6 from Finish or londer than 8 from Start.

Forward

A	B	C	D	E	Finish
7,s	4,s	**3,s**			
7,s	**4,s**	*		8,C	15,C
6,B	*	*	12,B	8,C	15,C
	*	*			
	*	*			
	*	*			

Backward

A	B	C	D	E	Start
		12,f	5,f	**4,f**	
	14,E	9,E	**5,f**	*	
8,D	13, D	9,E	*	*	
			*	*	
			*	*	
			*	*	

Proof by contradiction

Assume there is another path from start to finish that is shorter (than the path with length 14!).

Then one of the following two paths should exist:

- A path from Start to some point that has length shorter than 6.
- A path from Finish to some point that has length shorter than 8.

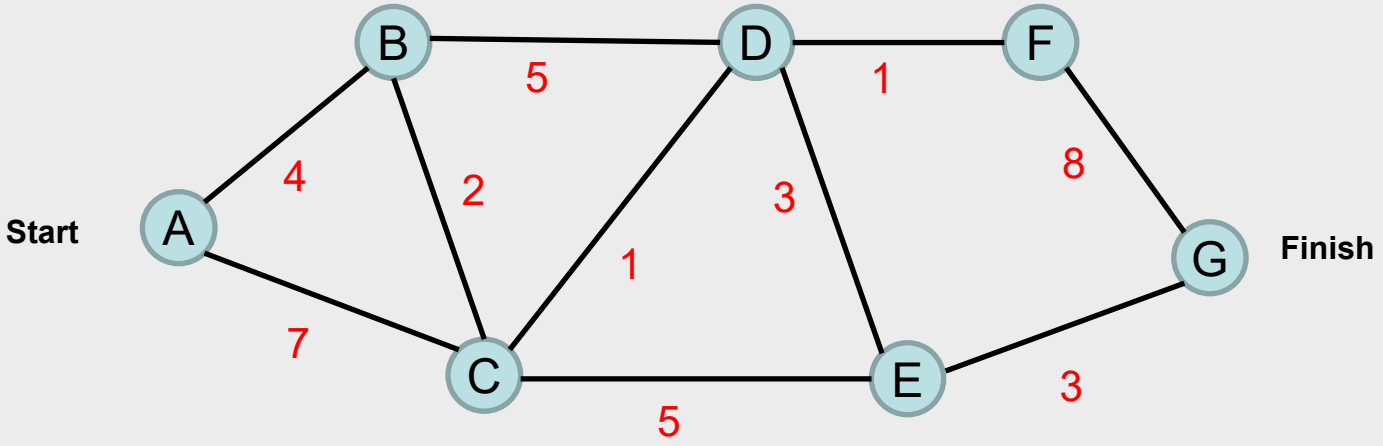
Bi-directional Dijkstra ("=" two-sided)

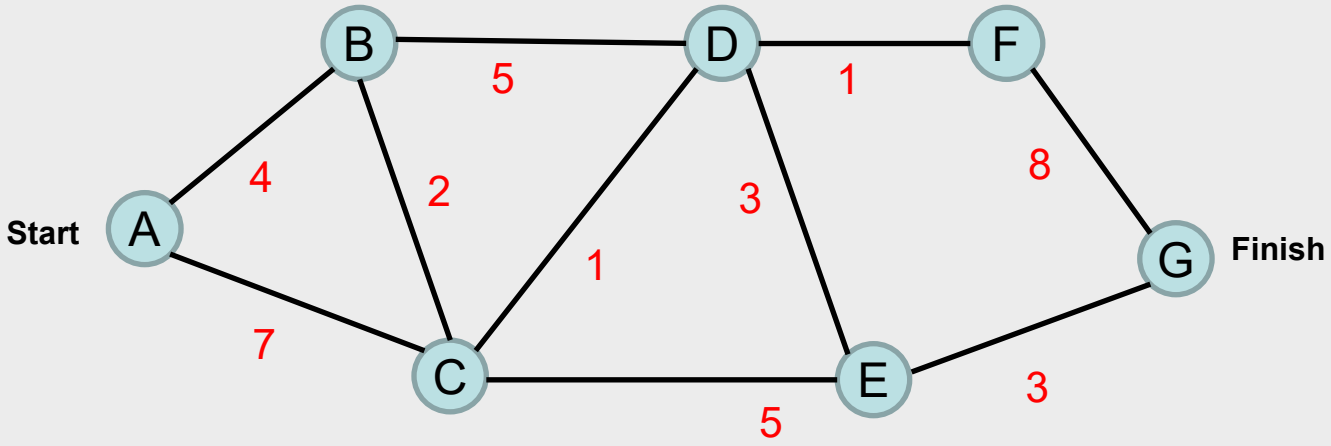
- Idea: run Dijkstra's algorithm twice, with
  - origin as starting node
  - destination as starting node

  Stop when they 'meet'.
- Speed-up because:
  - less steps
  - rows of tables computed simultaneously

Exercise!

Use bi-directional Dijkstra to find the shortest path in the following road network:



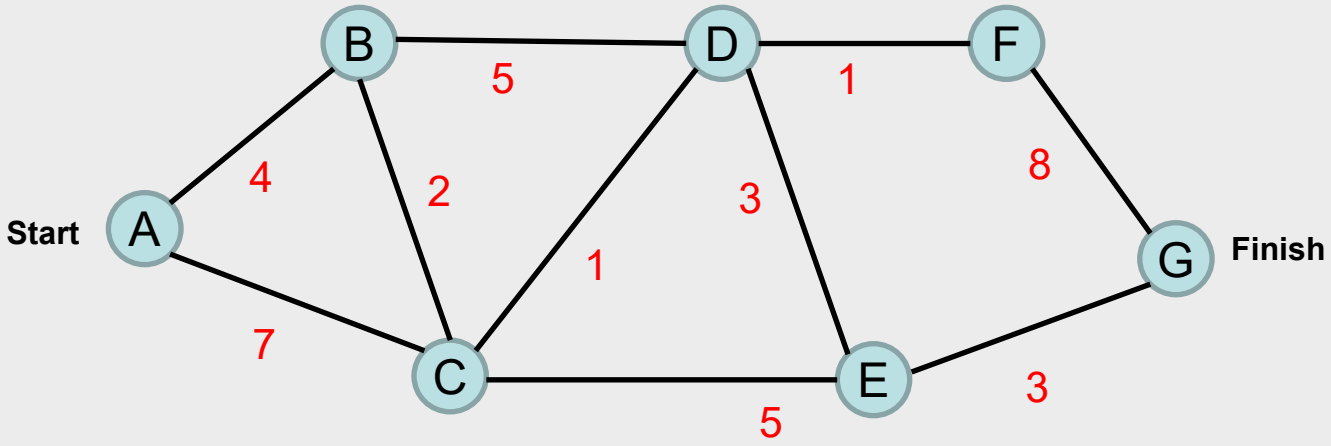


Forward

B	C	D	E	F	Finish

Backward

A	B	C	D	E	F

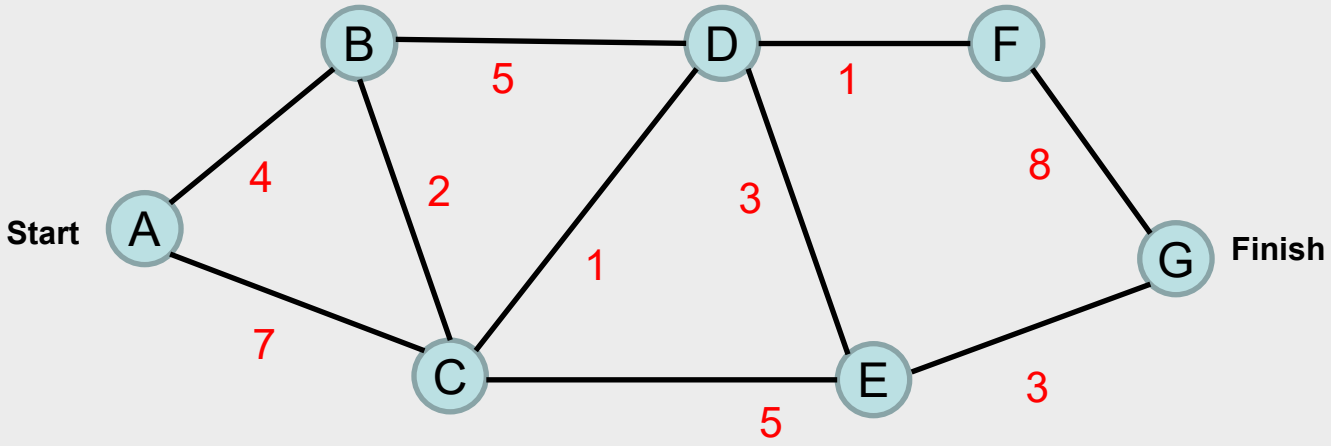


Forward

B	C	D	E	F	Finish
4,A	7,A				

Backward

A	B	C	D	E	F

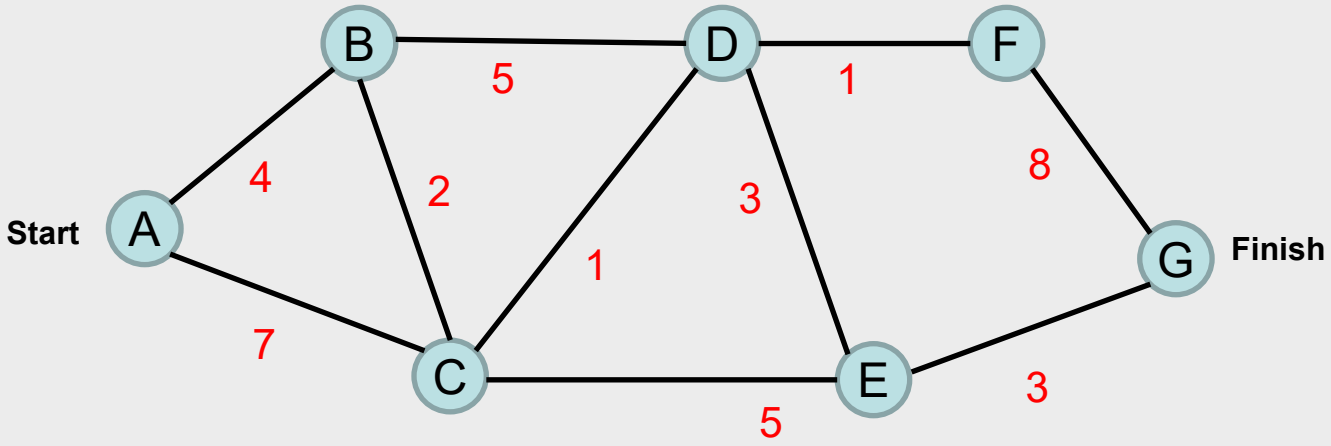


Forward

B	C	D	E	F	Finish
4,A	7,A				

Backward

A	B	C	D	E	F
				3,G	8,G

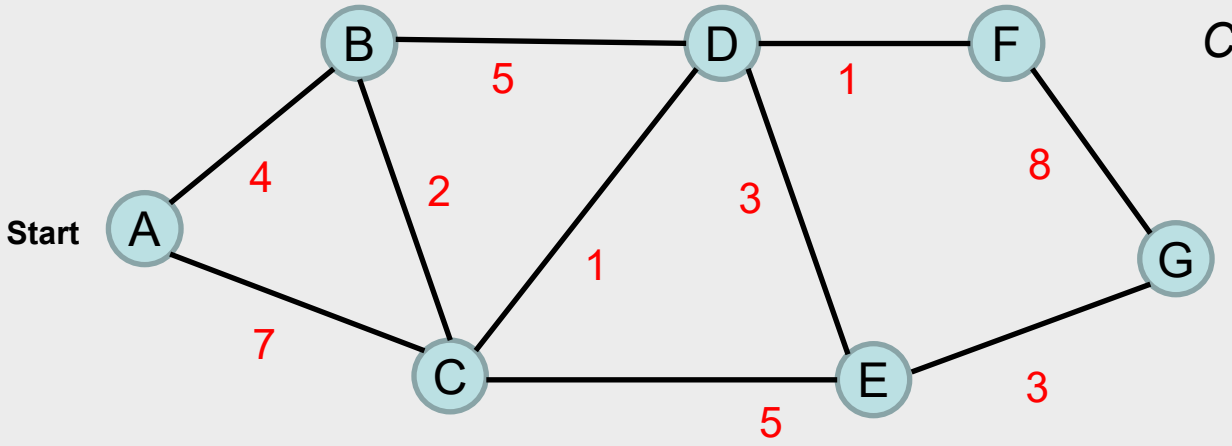


Forward

B	C	D	E	F	Finish
4,A	7,A				
*	**6,B**	9,B			
*					
*					
*					
*					

Backward

A	B	C	D	E	F
				3,G	8,G
		8,E	**6,E**	*	8,G
				*	
				*	
				*	
				*	



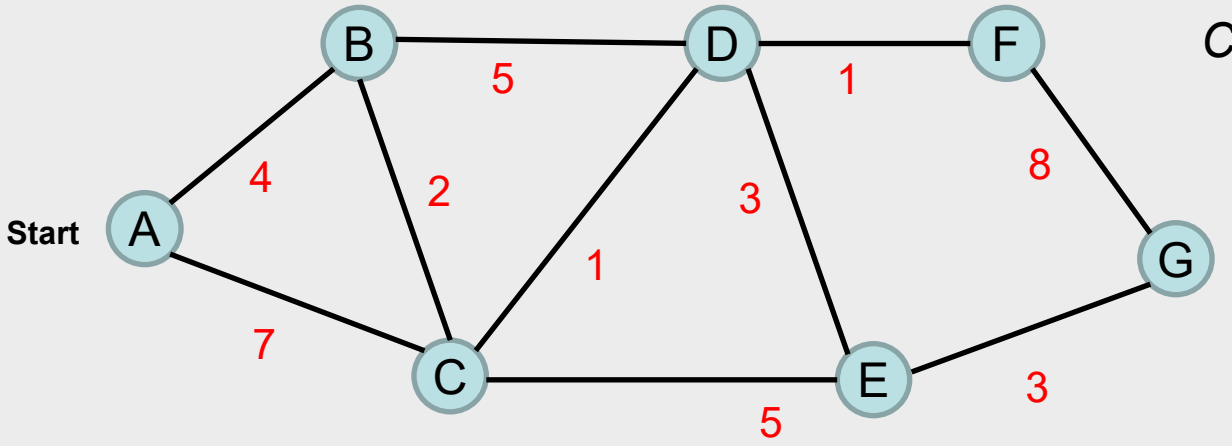
Current shortest path:
(A→B→C→E→G)
14

Forward

B	C	D	E	F	Finish
4,A	7,A				
*	**6,B**	9,B			
*					
*					
*					
*					

Backward

A	B	C	D	E	F
				3,G	8,G
		8,E	**6,E**	*	8,G
				*	
				*	
				*	
				*	



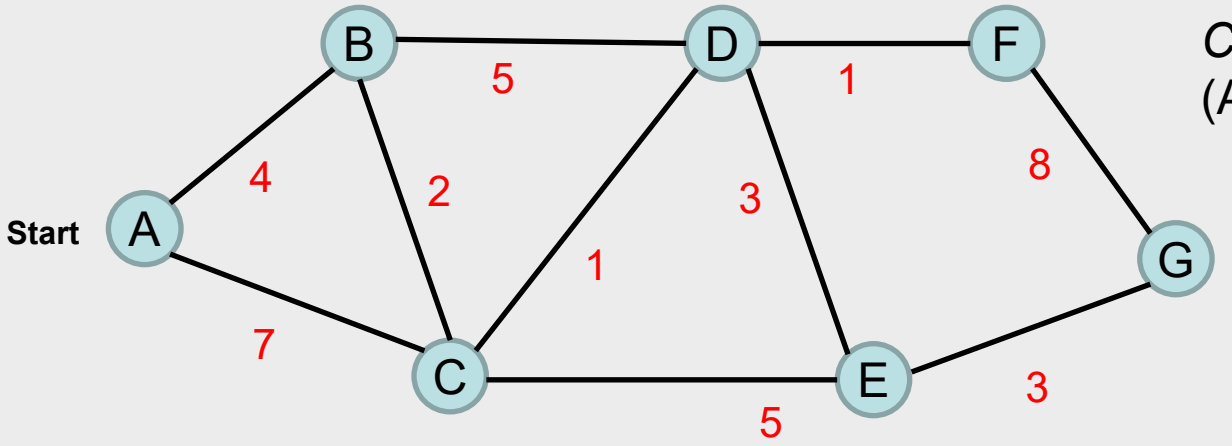
Current shortest path:
(A→B→C→E→G)
14

Forward

B	C	D	E	F	Finish
4,A	7,A				
*	**6,B**	9,B			
*	*	**7,C**	12,C		
*	*				
*	*				
*	*				

Backward

A	B	C	D	E	F
				3,G	8,G
		8,E	**6,E**	*	8,G
	11,B	7,D	*	*	**7,F**
			*	*	
			*	*	
			*	*	



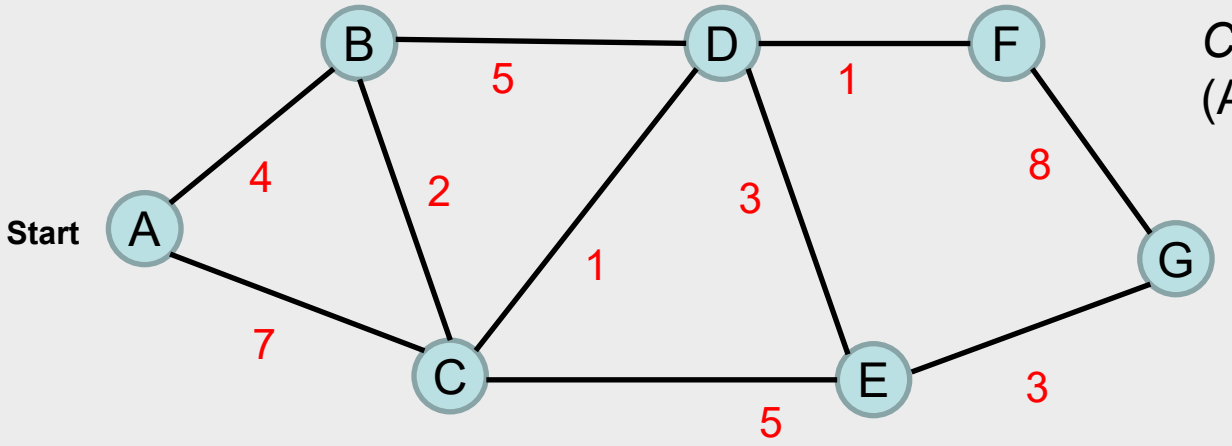
Current shortest path:
(A→B→C→D→E→G)
13

Forward

B	C	D	E	F	Finish
4,A	7,A				
*	**6,B**	9,B			
*	*	**7,C**	12,C		
*	*				
*	*				
*	*				

Backward

A	B	C	D	E	F
				3,G	8,G
		8,E	**6,E**	*	8,G
	11,B	7,D	*	*	**7,F**
			*	*	
			*	*	
			*	*	



Current shortest path:
(A→B→C→D→E→G)
13

Forward

B	C	D	E	F	Finish
4,A	7,A				
*	**6,B**	9,B			
*	*	**7,C**	12,C		
*	*				
*	*				
*	*				

Backward

A	B	C	D	E	F
				3,G	8,G
		8,E	**6,E**	*	8,G
	11,B	7,D	*	*	**7,F**
			*	*	
			*	*	
			*	*	

How can we improve Dijkstra's algorithm (in terms of speed)?

Take the location of the destination into account!

For example:

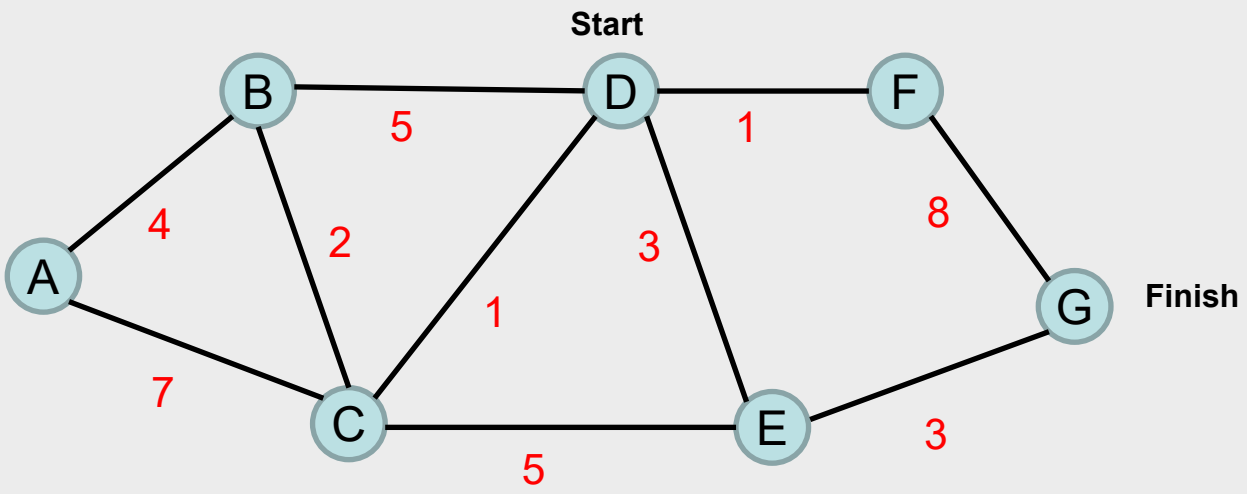
- Bi-directional Dijkstra 
- A-star algorithm

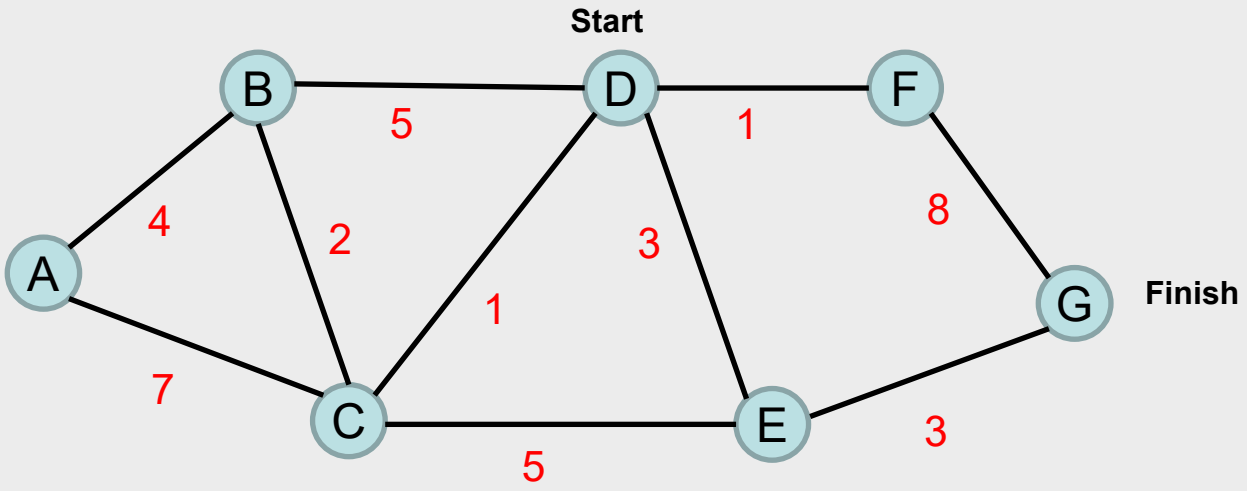
A-star algorithm

- Idea: only update the nodes located between origin and destination.

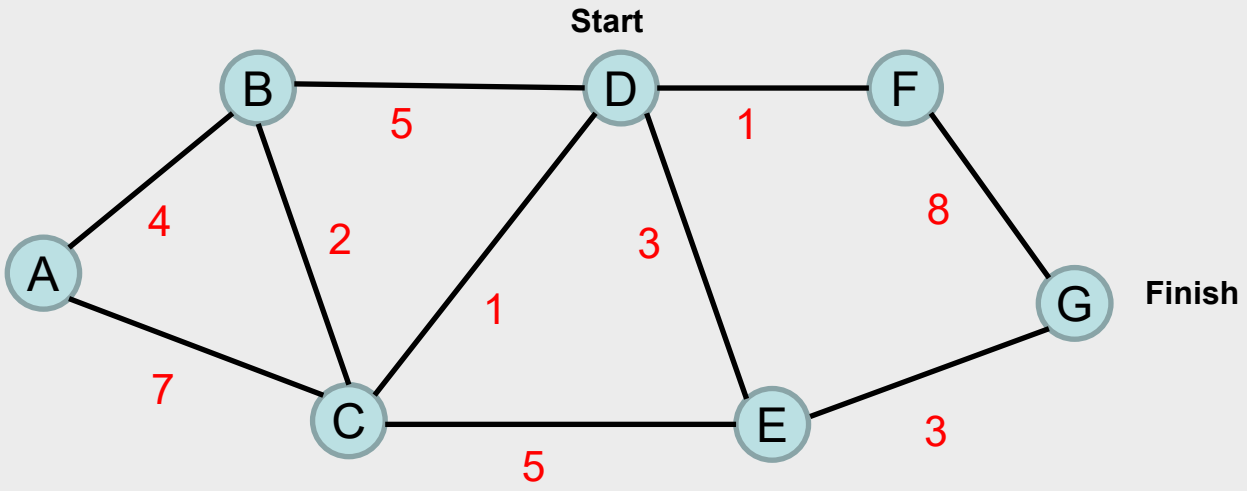
- How? By including the estimated future costs

DEMO

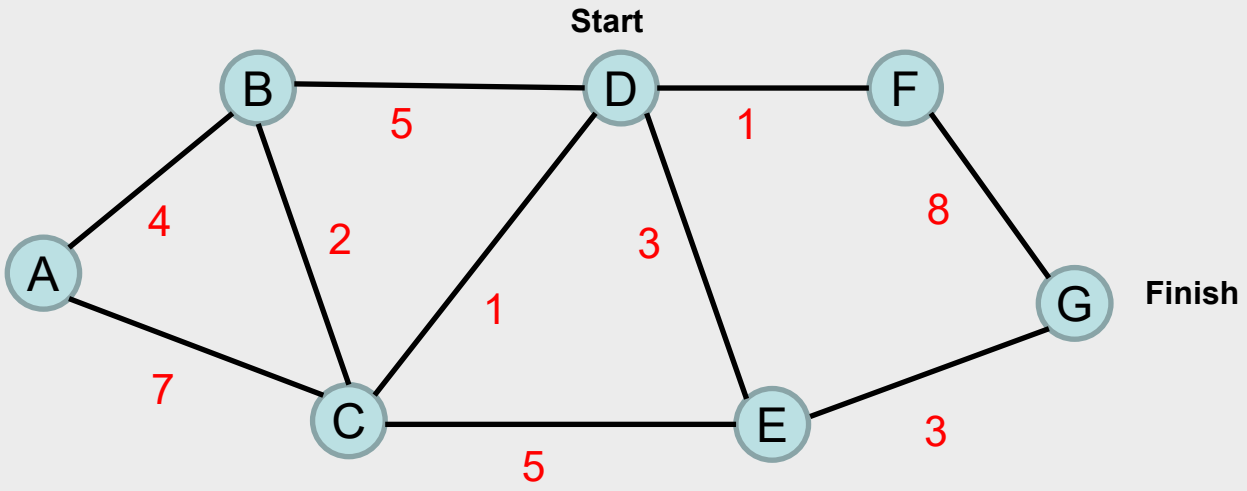

Start
B
D
F
5
1
4
2
3
8
A
1
G
Finish
7
C
E
3
5



A	B	C	D	E	F	G

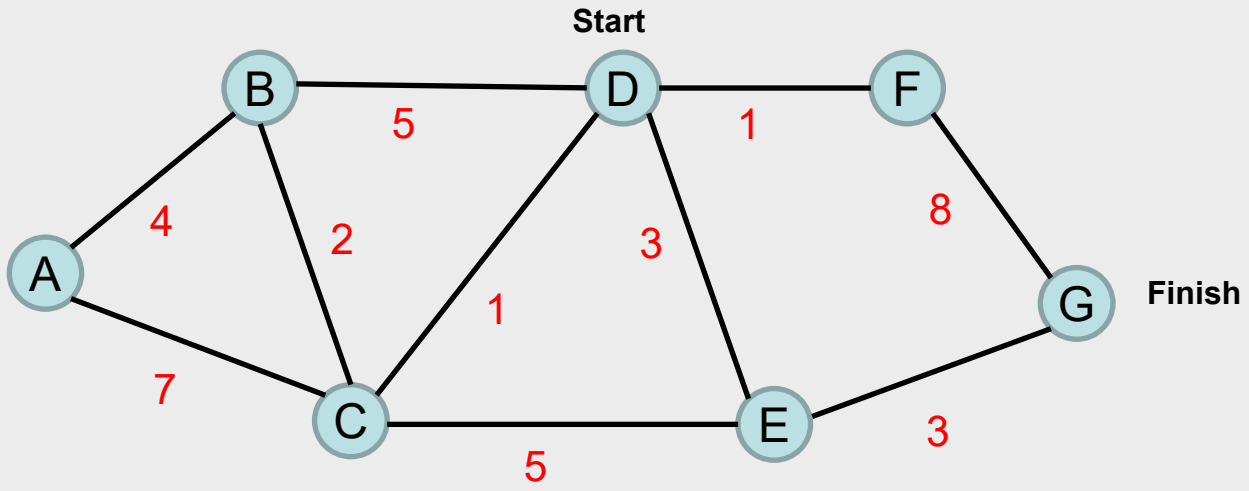


A	B	C	D	E	F	G
	5,D	**1,D**		3,D	1,D	
8,C	3,C	*		3,D	**1,D**	
8,C	3,C	*		**3,D**	*	9,F
8,C	**3,C**	*		*	*	6,E
7,B	*	*		*	*	**6,E**



A	B	C	D	E	F	G
	5,D	**1,D**		3,D	1,D	
8,C	3,C	*		3,D	**1,D**	
8,C	3,C	*		**3,D**	*	9,F
8,C	**3,C**	*		*	*	6,E
7,B	*	*		*	*	**6,E**

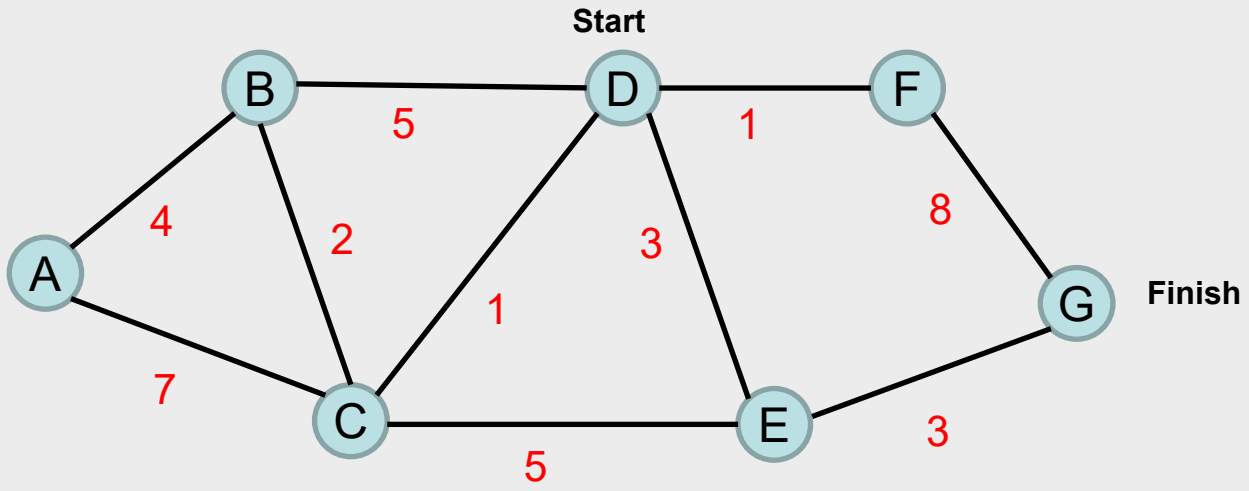
D → E → G



geo. length

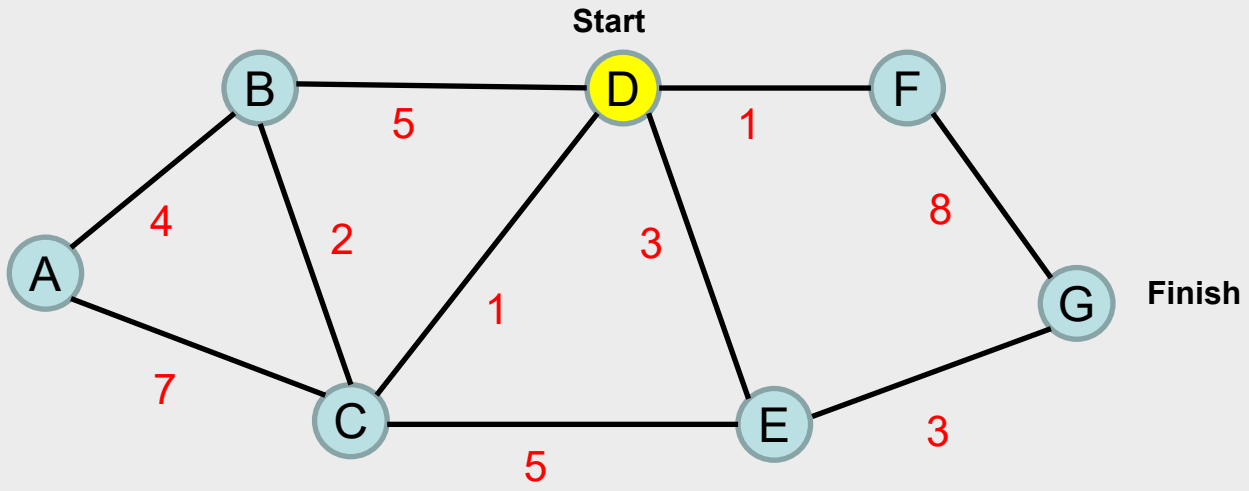
A	B	C	D	E	F	G
9	8	7	6	4	2	0


Groningen
Fryslân (Friesland)
Drenthe
erland (The Netherlands)
Gelderland
Utrecht
Noord-Brabant
(North Brabant)
Nordrhein-
Westfalen
(North Rhine-
Westphalia)
Limburg



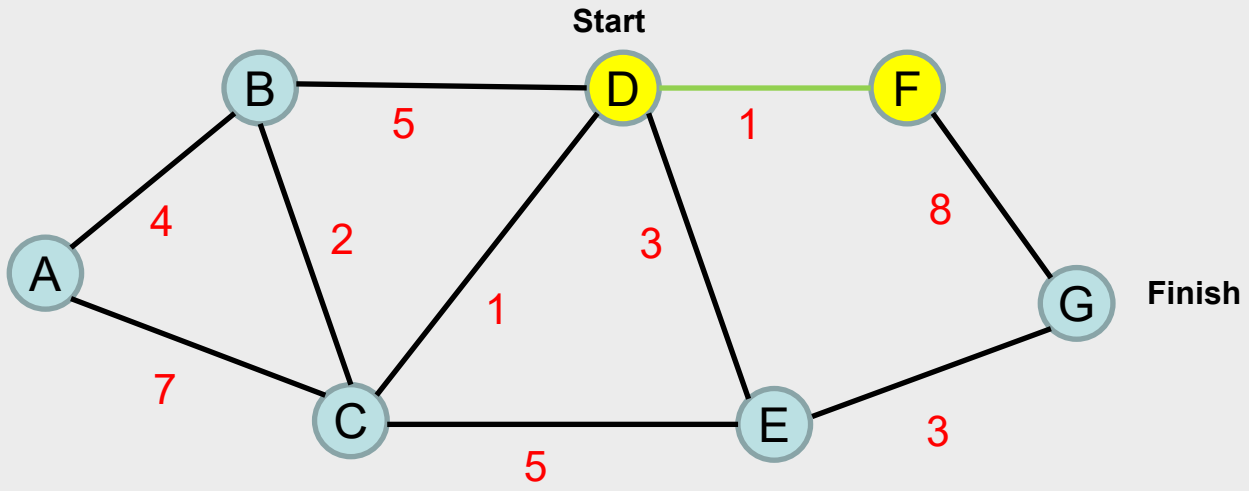
geo. length

A	B	C	D	E	F	G
9	8	7	6	4	2	0



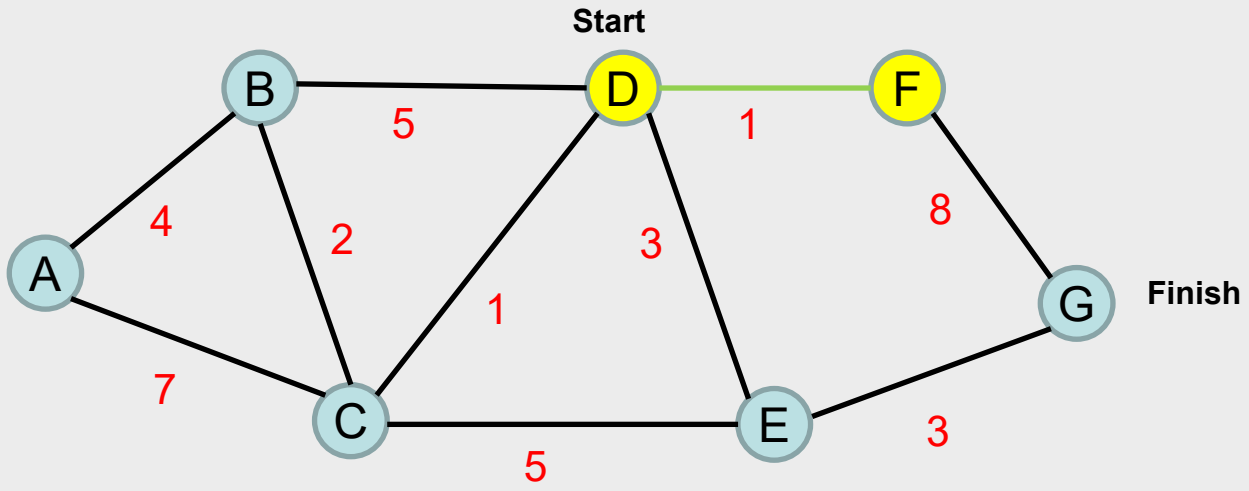
geo. length

A	B	C	D	E	F	G
9	8	7	6	4	2	0
	13,5,D	8,1,D		7,3,D	3,1,D	



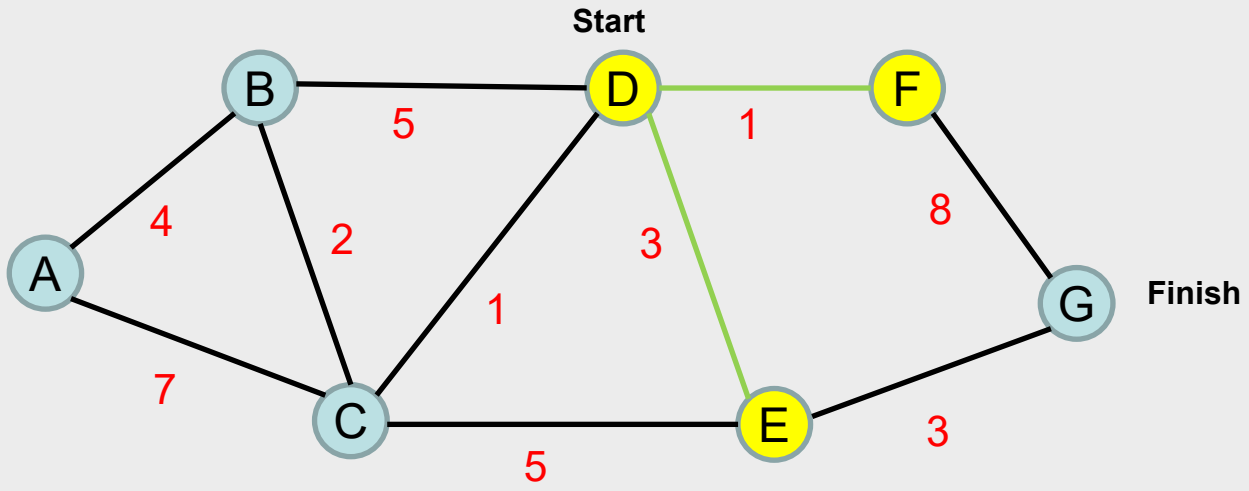
geo. length

A	B	C	D	E	F	G
9	8	7	6	4	2	0
	13,5,D	8,1,D		7,3,D	**3,1,D**	



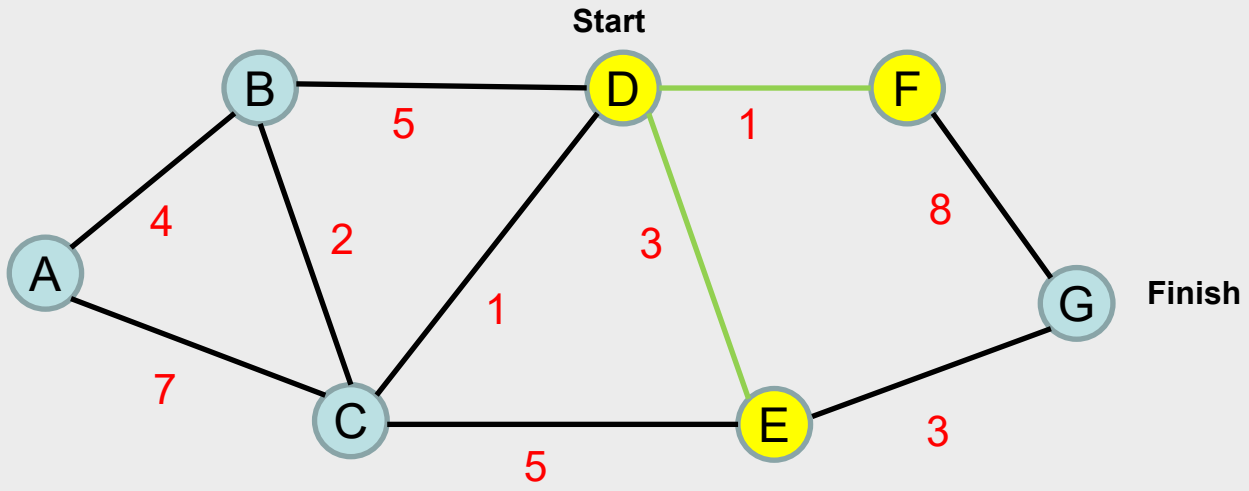
geo. length

A	B	C	D	E	F	G
9	8	7	6	4	2	0
	13,5,D	8,1,D		7,3,D	**3,1,D**	
	13,5,D	8,1,D		7,3,D	*	9,9,F
					*	
					*	



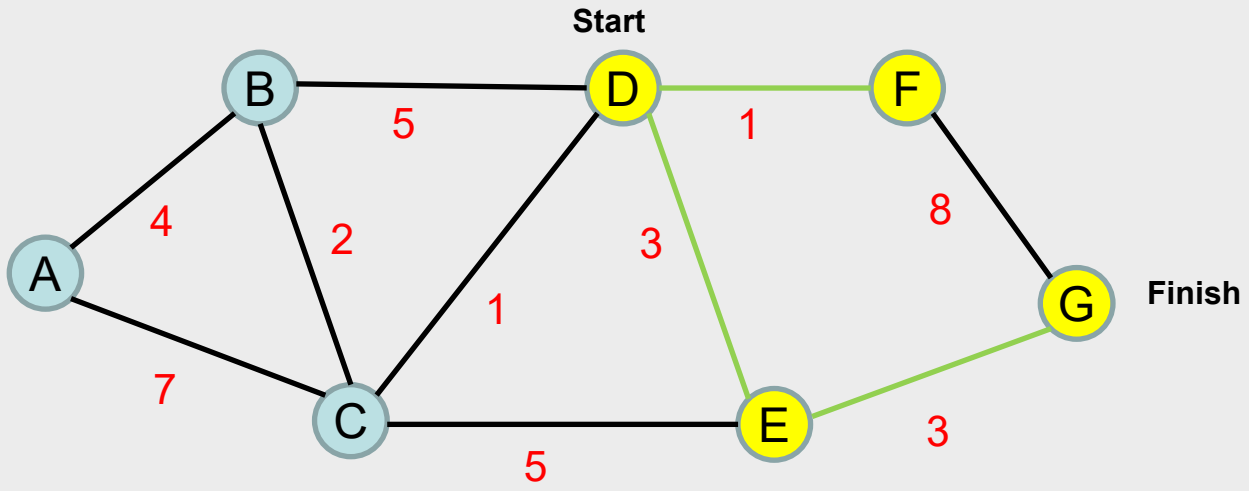
geo. length

A	B	C	D	E	F	G
9	8	7	6	4	2	0
	13,5,D	8,1,D		7,3,D	**3,1,D**	
	13,5,D	8,1,D		**7,3,D**	*	9,9,F
					*	
					*	



geo. length

A	B	C	D	E	F	G
9	8	7	6	4	2	0
	13,5,D	8,1,D		7,3,D	**3,1,D**	
	13,5,D	8,1,D		**7,3,D**	*	9,9,F
	13,5,D	8,1,D		*	*	6,6,E
				*	*	



	A	B	C	D	E	F	G
geo. length	9	8	7	6	4	2	0
		13,5,D	8,1,D		7,3,D	**3,1,D**	
		13,5,D	8,1,D		**7,3,D**	*	9,9,F
		13,5,D	8,1,D		*	*	**6,6,E**
					*	*	

A-star algorithm

- Idea: only update the nodes located between origin and destination.

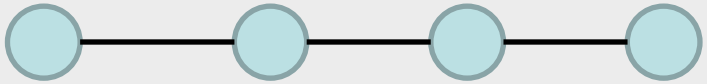
- How? By including the estimated future costs

Quiz time!

- Is A-star always faster than Dijkstra's algorithm?
- Draw a network in which A-star will be faster than the bi-directional Dijkstra algorithm.
- Draw a network in which the bi-directional Dijkstra algorithm will be faster than A-star.

Quiz time!

- Is A-star always faster than Dijkstra's algorithm?

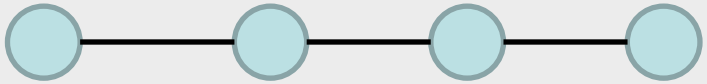


- Draw a network in which A-star will be faster than the bi-directional Dijkstra algorithm.

- Draw a network in which the bi-directional Dijkstra algorithm will be faster than A-star.

Quiz time!

- Is A-star always faster than Dijkstra's algorithm?

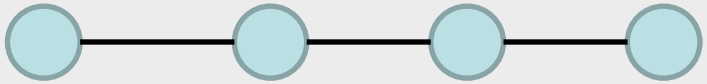


- Draw a network in which A-star will be faster than the bi-directional Dijkstra algorithm.

- Draw a network in which the bi-directional Dijkstra algorithm will be faster than A-star.

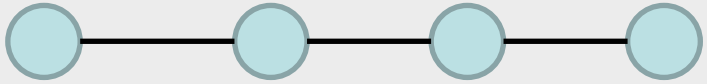
Quiz time!

- Is A-star always faster than Dijkstra's algorithm?



- Draw a network in which A-star will be faster than the bi-directional Dijkstra algorithm.

- Draw a network in which the bi-directional Dijkstra algorithm will be faster than A-star.



Today's plan:

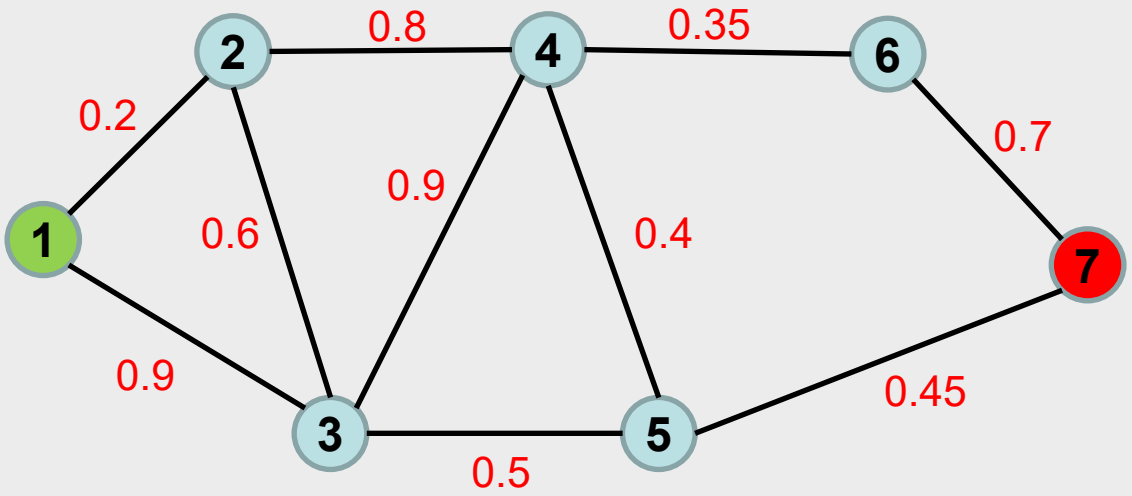
- Speeding-up Dijkstra's algorithm 
- Extending Dijkstra's algorithm beyond the simple shortest path (i.e. in distance) setting.

Introducing uncertainty

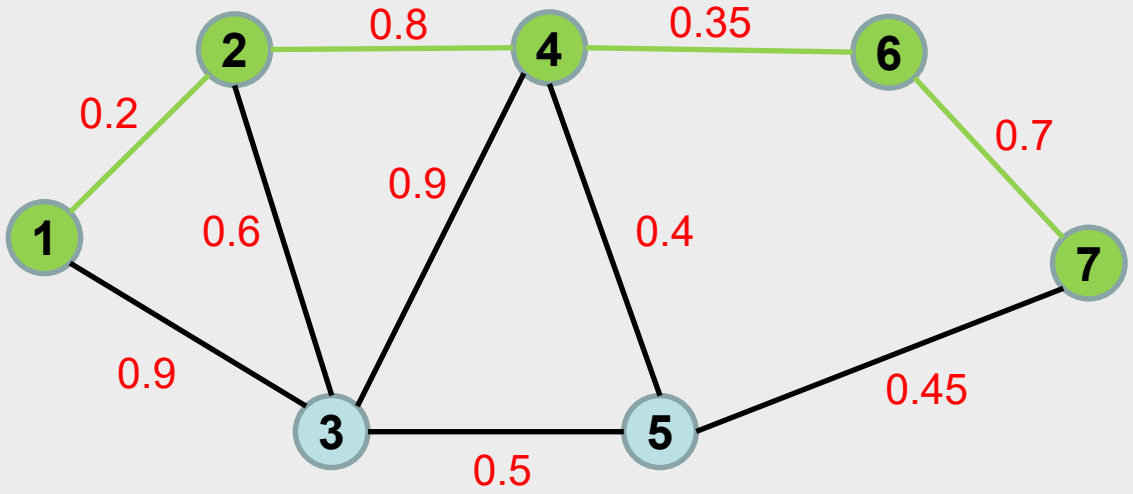
- So far, we have only been interested in finding the shortest route
- Perhaps there are routes other than the shortest route that are ‘better’
- The shortest route would not be a very desireable route if it is likely to suffer from traffic jams…

Introducing uncertainty

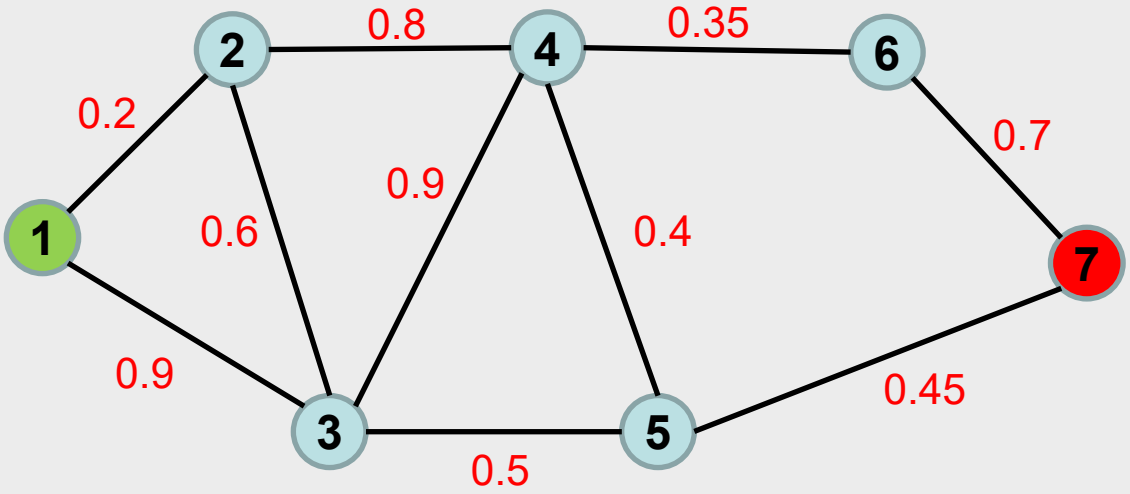
- Finding the route from 1 to 7 that has the lowest probability of getting stuck in a traffic jam



- The numbers denote the probability of not getting stuck in a traffic jam



- Consider the route $1 \rightarrow 2 \rightarrow 4 \rightarrow 6 \rightarrow 7$
- The probability of not getting stuck in a traffic jam is
  $p_{17} = p_{12} \times p_{24} \times p_{46} \times p_{67}$
- So, $p_{17} = 0.2 \times 0.8 \times 0.35 \times 0.7 \approx 0.04$
- Can we directly apply Dijkstra's algorithm to find the 'most-reliable' route?



- Note that we want to **maximise** a **product**

- We need to find a way to rewrite our problem such that is becomes a problem of mimimising a sum

Applying Dijkstra in a non-standard setting

- Recall that $\log(xy) = \log(x) + \log(y)$
- Therefore,
$$\log(p_{17}) = \log(p_{12}) + \log(p_{24}) + \log(p_{46}) + \log(p_{67})$$
- So, we can find the route that maximises p_{17}, by finding the route that maximises the log of p_{17}

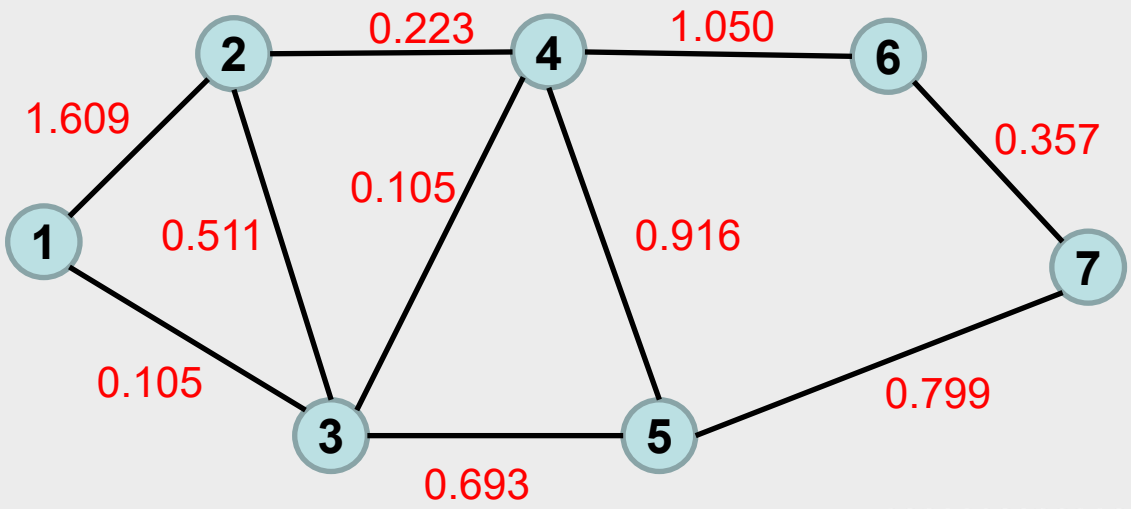
Applying Dijkstra in a non-standard setting

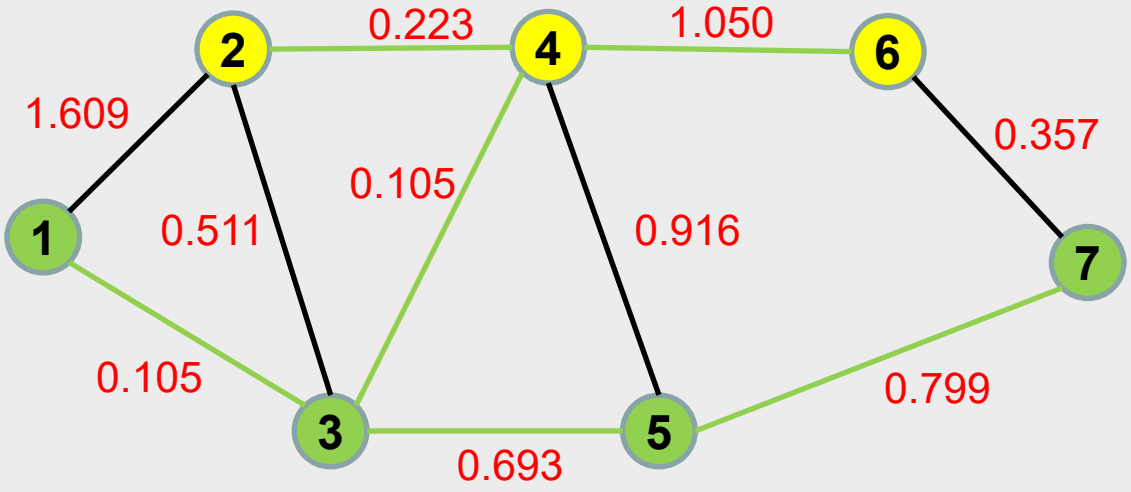
- Recall that Dijkstra's algorithm minimises a sum, but we want to maximise a sum
- We can make use of the following trick: The route that maximises $\log(p_{17})$ will minimise $-\log(p_{17})$
- Conclusion: We can use Dijkstra to find the route that minimises $-\log(p_{17})$ and this is the route we are interested in finding

Exercise

- Find the route that minimises the probability of getting stuck in a traffic jam

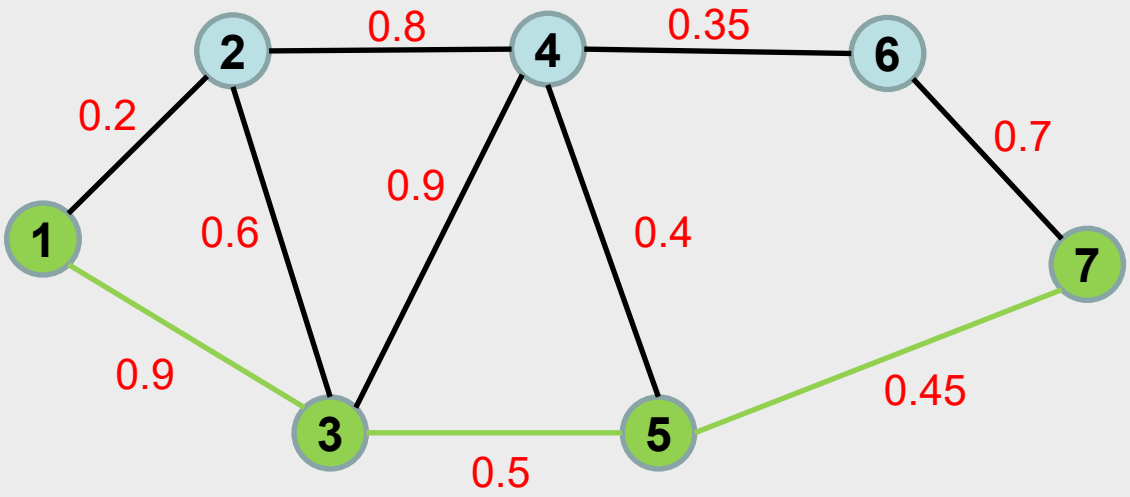
- We have already applied the –log transformation for you:





nodes	2	3	4	5	6	7
Step 1	$(1.609, 1)$	$(0.105, 1)^*$	No edge	No edge	No edge	No edge
Step 2	$(0.616, 3)$	-	$(0.210, 3)^*$	$(0.798, 3)$	No edge	No edge
Step 3	$(0.433, 4)^*$	-	-	$(0.798, 3)$	$(1.260, 4)$	No edge
Step 4	-	-	-	$(0.798, 3)^*$	$(1.260, 4)$	No edge
Step 5	-	-	-	-	$(1.260, 4)^*$	$(1.597, 5)$
Step 6	-	-	-	-	-	$(1.597, 5)^*$

- The path that minimises $-\log(p_{17})$, maximises p_{17}



- We conclude that we found a route such that the probability of not getting stuck in a traffic jam is

$$p_{17} = 0.9 \times 0.5 \times 0.45 \approx 0.20$$

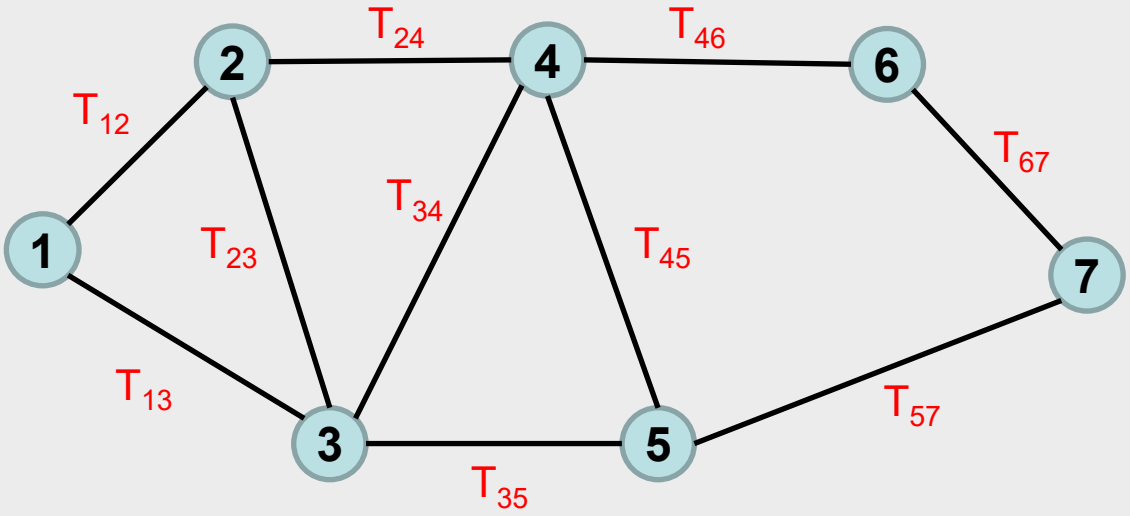
Refinements

- Up to now, we have considered two extremes
  - The shortest route, not taking into account the reliability of the route
  - The most reliable route, not taking into account the length of the route
- We would like to find a route that is both quick and reliable…

Travel times as random variables

- Instead of distances, we are now going to work with travel times
- Travel times can be uncertain for many reasons
- We model travel times as random variables

Stochastic travel times



Travel times as random variables

- Instead of distances, we are now going to work with travel times
- Travel times can be uncertain for many reasons
- We model travel times as random variables
- We will assume the travel times are normally distributed

The density of a normal distribution

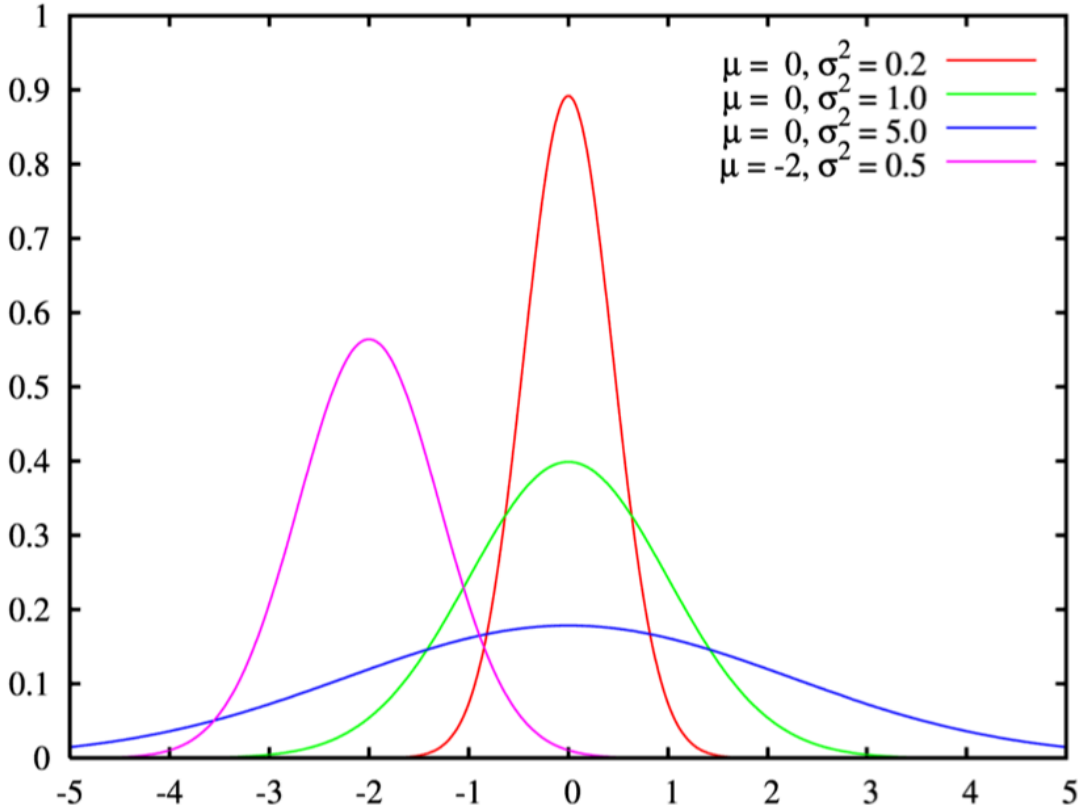
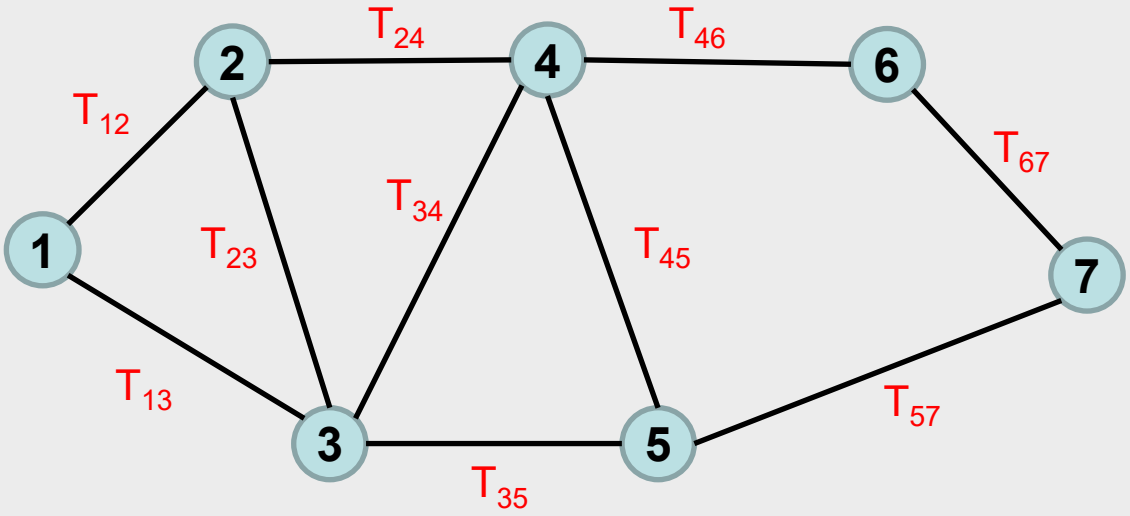


Figure 1.2.1. The density function of a normally distributed random variable with parameters μ and σ^2.

Stochastic travel times



- The travel times T_{ij} have a mean and a variance
- Suppose that T_{13} has mean 15 and variance 4
- Suppose that T_{12} has mean 12 and variance 16

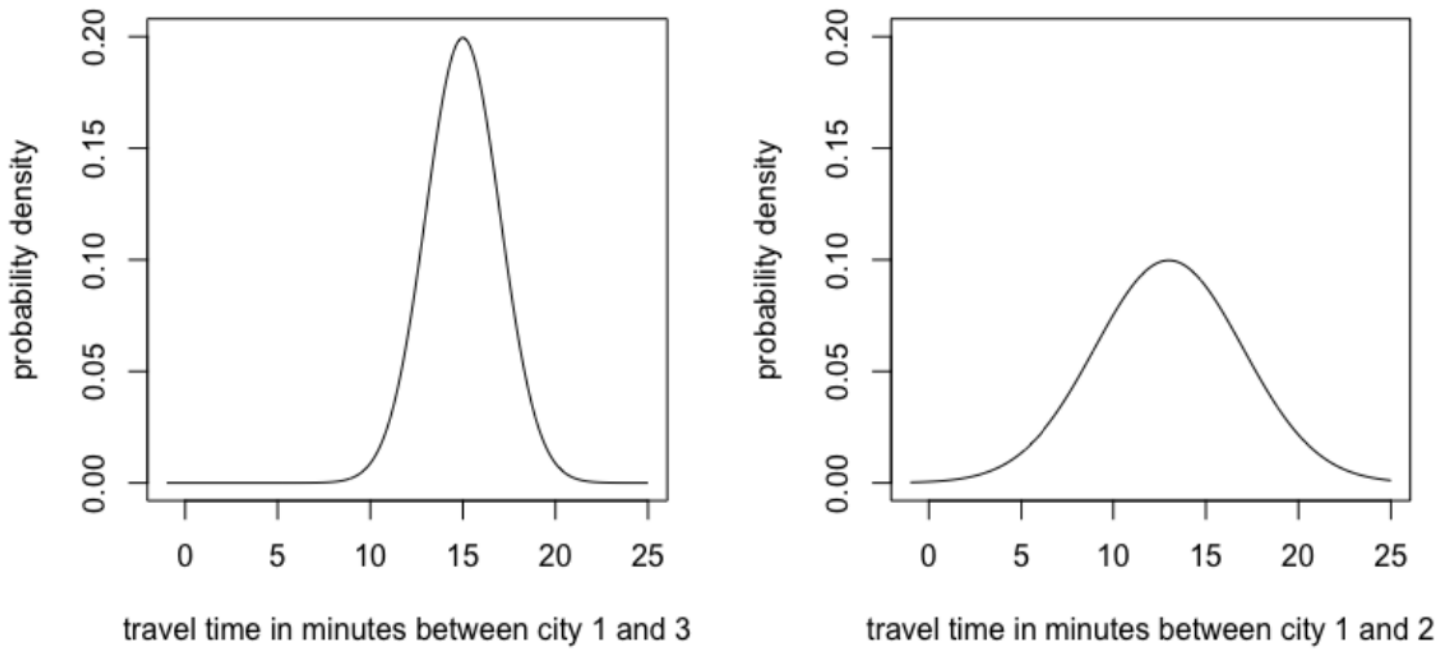
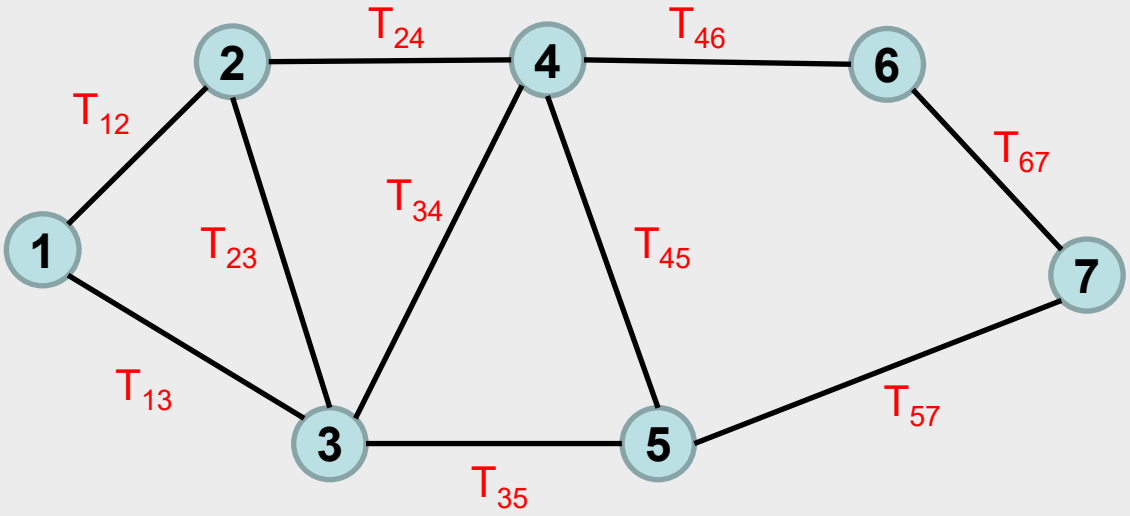


Figure 2.2.4. The density of $T_{13} \sim \mathcal{N}(15, 4)$ (left) and $T_{12} \sim \mathcal{N}(12, 16)$ (right).

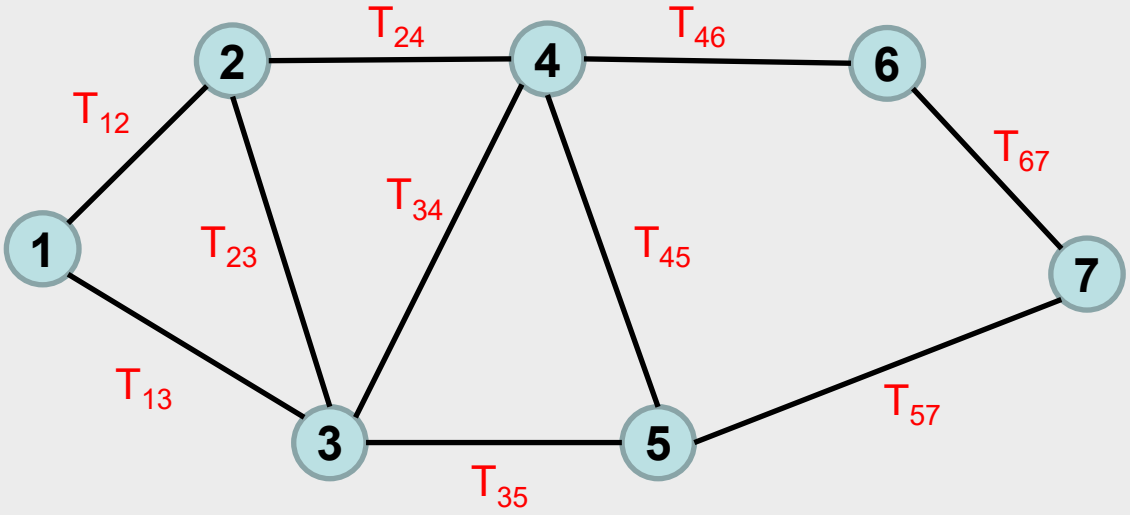
Stochastic travel times



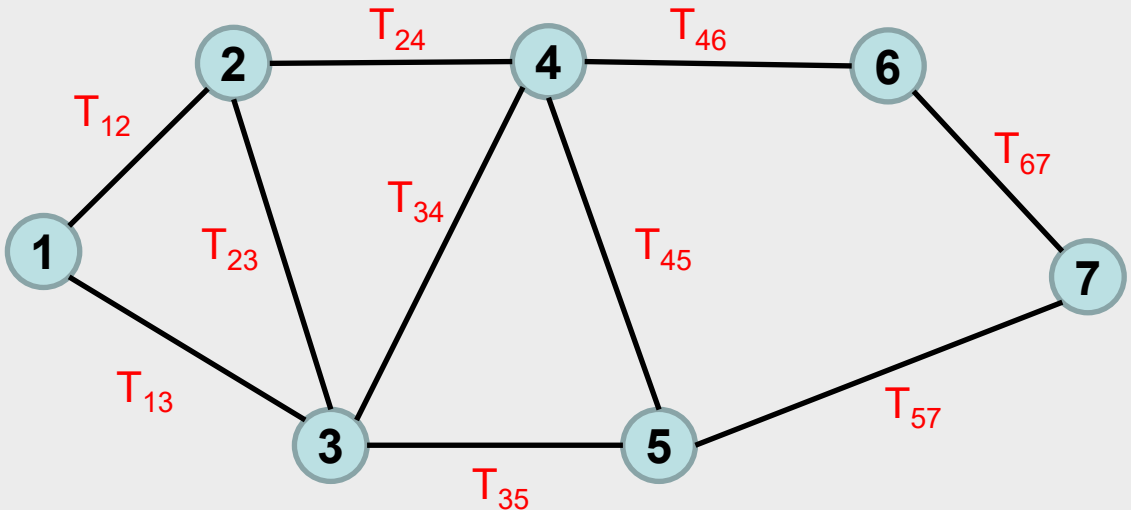
- The travel times T_{ij} have a mean and a variance
- Suppose that T_{13} has mean 15 and variance 4
- Suppose that T_{12} has mean 12 and variance 16

Stochastic travel times

What is the 'optimal' route?



- The travel times T_{ij} have a mean and a variance
- Suppose that T_{13} has mean 15 and variance 4
- Suppose that T_{12} has mean 12 and variance 16

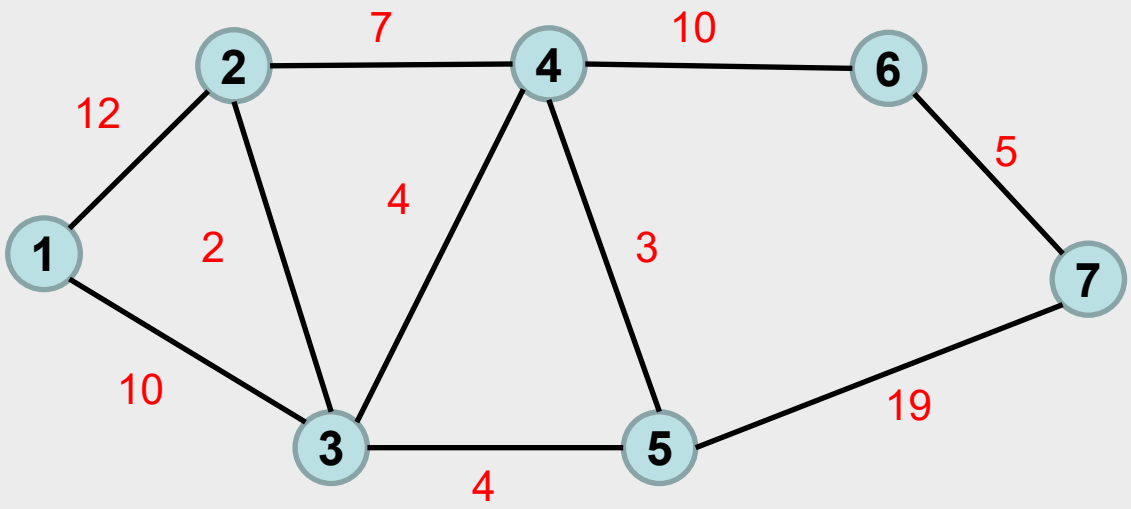


	T_{12}	T_{13}	T_{23}	T_{24}	T_{34}	T_{35}	T_{45}	T_{46}	T_{57}	T_{67}
μ	12	10	2	7	4	4	3	10	19	5
σ^2	1	9	1	9	4	1	1	16	1	4

Table 3.3.1. T_{ij} denotes the travel time between city i and j in minutes and is normally distributed with parameters μ and σ^2.

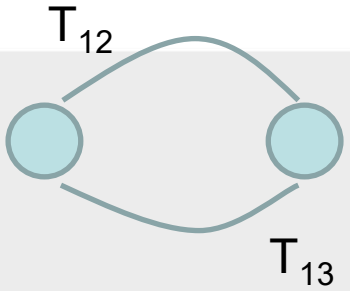
- We can find the route with the lowest expected travel time by applying Dijkstra (Exercise!)

Simply replace each random variable T_{ij} by its mean:



Arriving in time

- What if you want to find the route that maximises the probability of arriving at your destination in time?
- This could be a different route!
- …and, unfortunately, this route cannot be found by Dijkstra...



Why different routes?

- If interested in expectated travel time:

  T_{12} (mean 12) preferred over (mean 15) T_{13}

- However, if interested in the probability of arriving within 20 minutes:

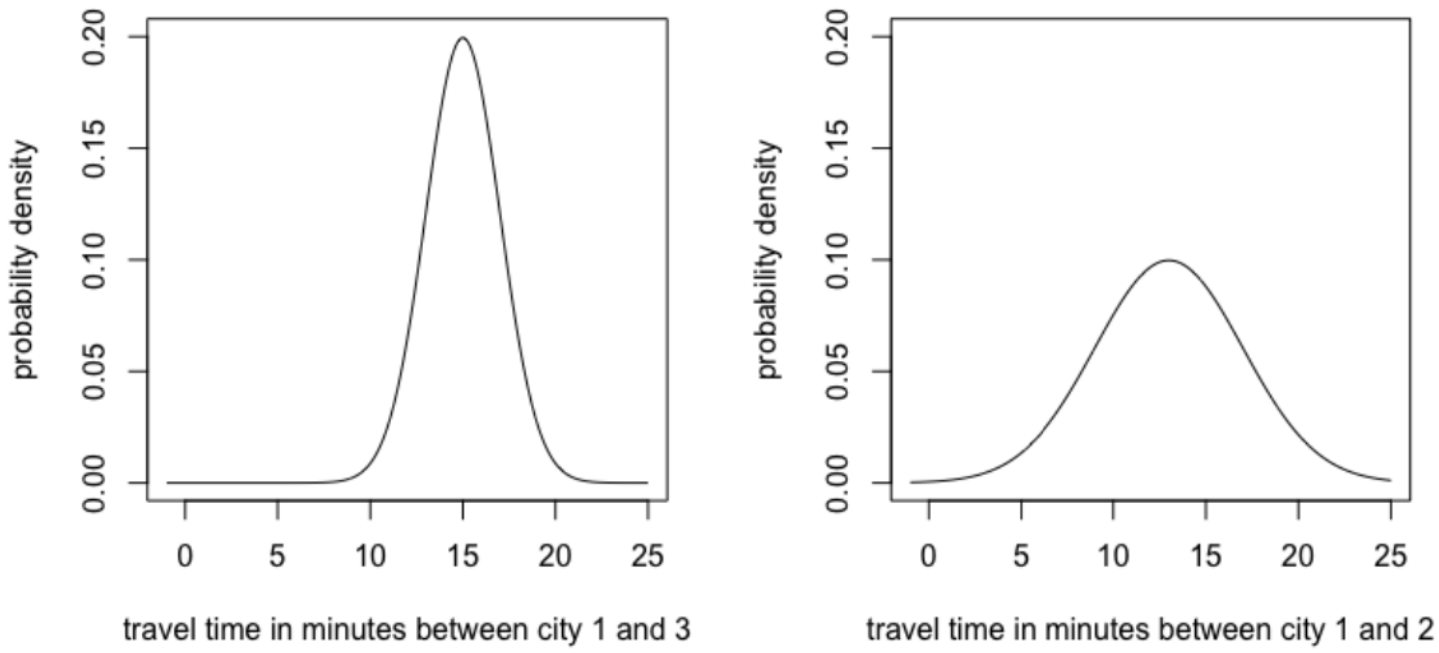


Figure 2.2.4. The density of $T_{13} \sim \mathcal{N}(15, 4)$ (left) and $T_{12} \sim \mathcal{N}(12, 16)$ (right).

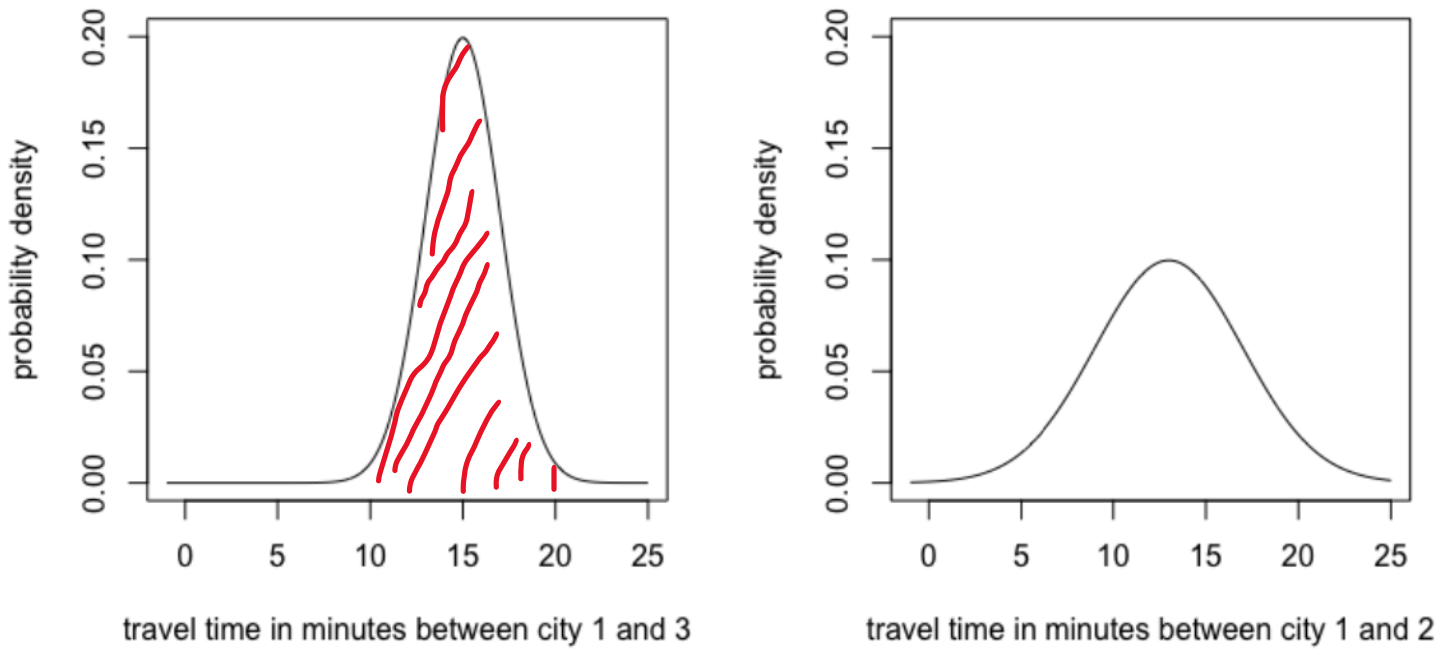


Figure 2.2.4. The density of $T_{13} \sim \mathcal{N}(15, 4)$ (left) and $T_{12} \sim \mathcal{N}(12, 16)$ (right).

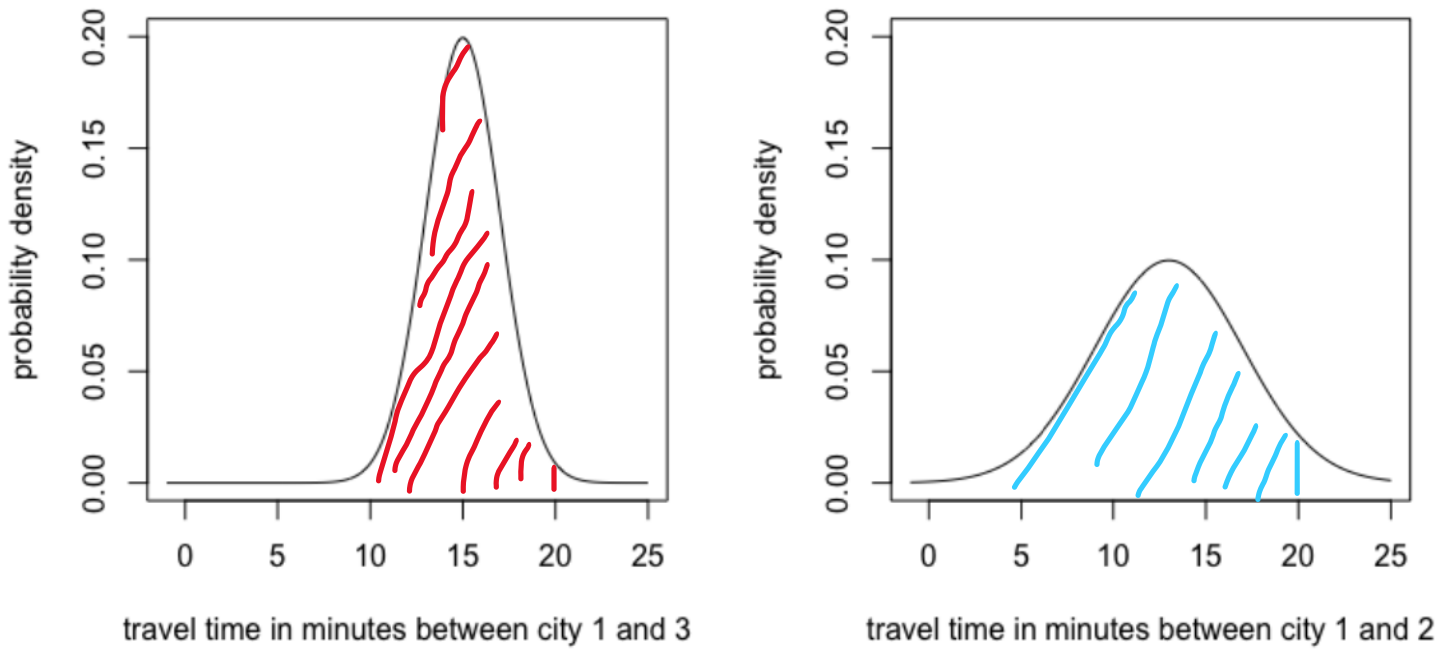
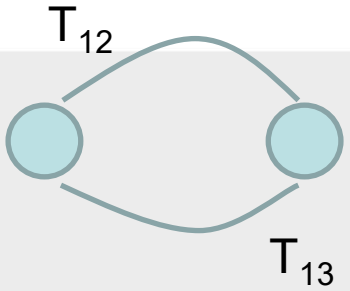


Figure 2.2.4. The density of $T_{13} \sim \mathcal{N}(15, 4)$ (left) and $T_{12} \sim \mathcal{N}(12, 16)$ (right).



Why different routes?

- If interested in expectated travel time:

  T_{12} (mean 12) preferred over (mean 15) T_{13}

- However, if interested in the probability of arriving within 20 minutes (exercise!):

  $P(T_{12} < 20) \approx 0.9772$

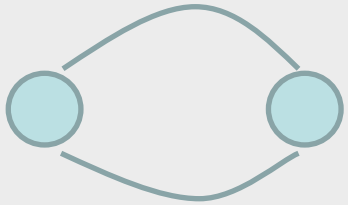
  $P(T_{13} < 20) \approx 0.9938$

- Thus, in that sense, T_{12} not preferred over T_{13}

From road pieces to routes

- This idea can be pushed further

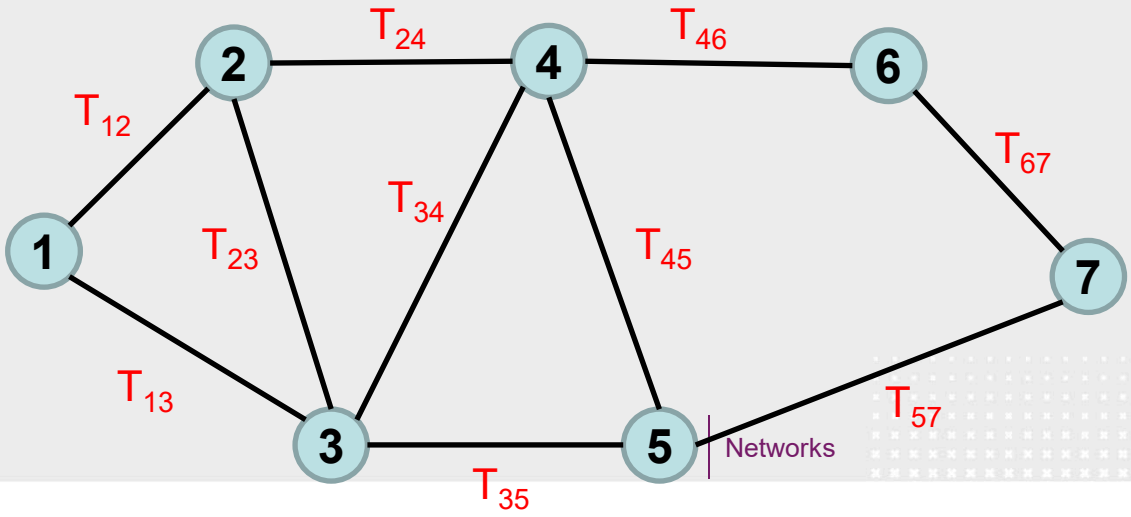
- Instead of single road pieces, we may compare the probability of arriving on time for different routes (exercise!)



From road pieces to routes

- This idea can be pushed further

- Instead of single road pieces, we may compare the probability of arriving on time for different routes (exercise!)



From road pieces to routes

- This idea can be pushed further
- Instead of single road pieces, we may compare the probability of arriving on time for different routes (exercise!)
- …but, optimal route can't be found with Dijkstra

Why not Dijkstra?

Why not Dijkstra?

$$P(T_1 + T_2 < 20) \neq P(T_1 < 20) + P(T_2 < 20)$$
$$P(T_1 + T_2 < 20) \neq P(T_1 < 20) * P(T_2 < 20)$$

Thus, can only find the optimal route in a very inefficient way…

Today's plan:

- Speeding-up Dijkstra's algorithm 
- Extending Dijkstra's algorithm beyond the simple shortest path (i.e. in distance) setting.

Concluding remarks

- Dijkstra's algorithm is a very useful tool to finding shortest paths (in distance) on a network
- Speed-up versions are available and necessary
- In reality: travel times are uncertain
- There is not one notion of 'optimal path': it depends on the preference of the driver
- For some preferences, Dijkstra is still useful (expected travel times, most reliable route), for other preferences, Dijkstra cannot directly be used (on-time arrival)

Thank you!