

Algorithms for network discovery, exploration and path calculations

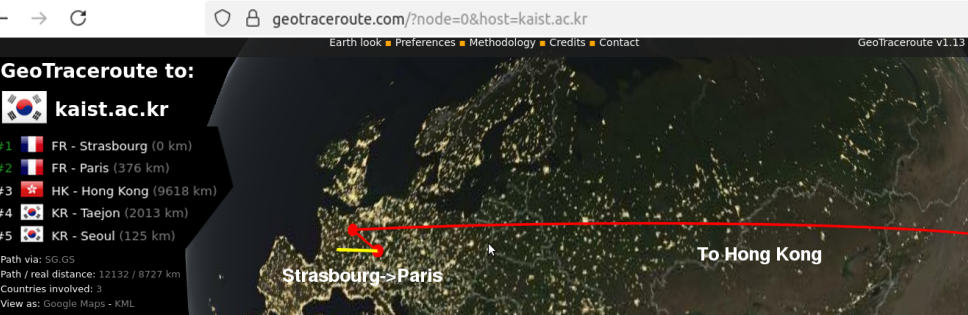
International Master CNAM

Daniel Porumbel (daniel.porumbel@cnam.fr)

Geo traceroot to kaist.ac.kr

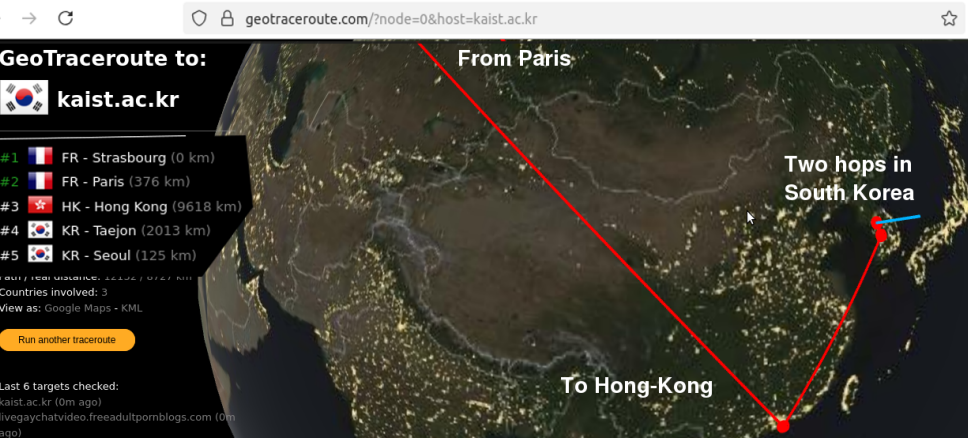
Starting point : Strasbourg, France

This is how the package travelled through Europe :

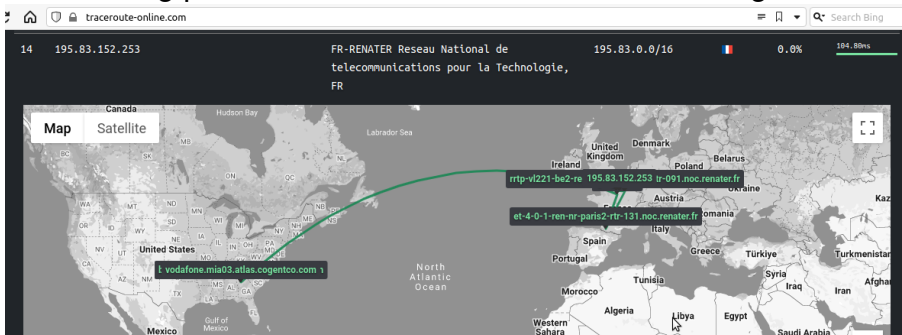


Geo traceroot to kaist.ac.kr

This is how the package traveled through Asia



The starting point for this on-line tool is Atlanta, Georgia, USA



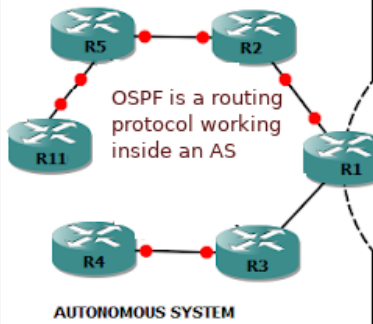
Zoom on the path to www.picardie.fr



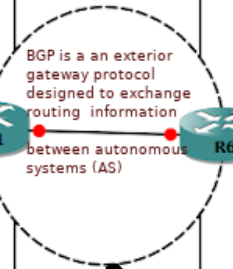
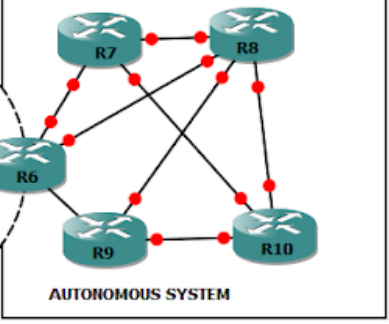
Zoom on the path to www.picardie.fr



Interior gateway protocols



Interior gateway protocols



Exterior Gateway Protocols (EGPs)

Routing inside an Autonomous System (AS)

An AS is typically an IP network owned by an ISP, where two types of protocols can be used :

- ① **link-state routing protocols** : every node constructs a map of the connectivity to the network, in the form of a graph, showing which nodes are connected to which other nodes.
 - Open Shortest Path First (OSPF)
 - Intermediate System to Intermediate System

Routing inside an Autonomous System (AS)

An AS is typically an IP network owned by an ISP, where two types of protocols can be used :

- ① **link-state routing protocols** : every node constructs a map of the connectivity to the network, in the form of a graph, showing which nodes are connected to which other nodes.
 - Open Shortest Path First (OSPF)
 - Intermediate System to Intermediate System
- ② **distance-vector routing protocols** : The term distance vector refers to the fact that the protocol manipulates vectors (arrays) of distances to other nodes in the network.
 - Routing Information Protocol (RIP) \oplus RIPv2, IGRP.

AS Routing : data exchanged

- 1 **link-state routing protocols** broadcast all information of its local connectivity (the local neighbors).
- 2 **distance-vector-routing** broadcast full routing tables, each route associated to a metric
 - You can see such information on a Linux system using :
`ip route show` or `route -n`

AS Routing : data exchanged

- ① **link-state routing protocols** broadcast all information of its local connectivity (the local neighbors).
- ② **distance-vector-routing** broadcast full routing tables, each route associated to a metric
 - You can see such information on a Linux system using :
`ip route show` or `route -n`
- ③ In a typical Linux, the routing table is populated using information received by DHCP (or NDP on IPv6)
 - The following commands change your default gateway :
`route add default gw 10.0.0.1 eth0`
`ip route add default via 10.0.0.1 dev eth0`
 - The following commands adds a new route
`ip route add 10.0.3.0/24 via 10.0.3.1`

AS Routing : data exchanged

- ① **link-state routing protocols** broadcast all information of its local connectivity (the local neighbors).
- ② **distance-vector-routing** broadcast full routing tables, each route associated to a metric
 - You can see such information on a Linux system using :
`ip route show` or `route -n`
- ③ In a typical Linux, the routing table is populated using information received by DHCP (or NDP on IPv6)
 - The following commands change your default gateway :
`route add default gw 10.0.0.1 eth0`
`ip route add default via 10.0.0.1 dev eth0`
 - The following commands adds a new route
`ip route add 10.0.3.0/24 via 10.0.3.1`
 - write 1 in file `/proc/sys/net/ipv4/ip_forward` to become a router, but a manual one (no OSPF or RIP by default)

AS Routing : routing tables in practice

A **routing table** gives information like :

destination 1 : go via neighbor A with a metric/cost of *a*

destination 2 : go via neighbor B with a metric/cost of *b*

destination 3 : go via neighbor C with a metric/cost of *c*

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
0.0.0.0	163.173.112.2	0.0.0.0	UG	600	0	0	wlo1
162.254.0.0	169.254.0.1	255.255.255.0	UG	0	0	0	wlo1
163.173.112.0	0.0.0.0	255.255.252.0	U	600	0	0	wlo1
163.254.0.0	169.254.0.1	255.255.255.0	UG	0	0	0	wlo1
169.254.0.0	0.0.0.0	255.255.0.0	U	1000	0	0	wlo1

```
daniel@daniel-i7:~$ sudo ip route add 160.254.0.0/24 via 169.254.0.1
```

```
daniel@daniel-i7:~$ route -n
```

Kernel IP routing table

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
0.0.0.0	163.173.112.2	0.0.0.0	UG	600	0	0	wlo1
160.254.0.0	169.254.0.1	255.255.255.0	UG	0	0	0	wlo1
162.254.0.0	169.254.0.1	255.255.255.0	UG	0	0	0	wlo1
163.173.112.0	0.0.0.0	255.255.252.0	U	600	0	0	wlo1
163.254.0.0	169.254.0.1	255.255.255.0	UG	0	0	0	wlo1
169.254.0.0	0.0.0.0	255.255.0.0	U	1000	0	0	wlo1

A bit of history

- The original ARPANET routing was using a distance vector algorithm. Most such algorithms use the [Bellman–Ford](#) algorithm to calculate the best route.

A bit of history

- The original ARPANET routing was using a distance vector algorithm. Most such algorithms use the [Bellman–Ford](#) algorithm to calculate the best route.
- [Open Shortest Path First \(OSPF\)](#) was developed by the interior gateway protocol (IGP) working group of the Internet Engineering Task Force (IETF). The working group was formed in 1988 to design an IGP based on shortest path algorithms, because in the mid-1980s, RIP was increasingly unable to serve larger networks.

A bit of history

- The original ARPANET routing was using a distance vector algorithm. Most such algorithms use the [Bellman–Ford](#) algorithm to calculate the best route.
- [Open Shortest Path First \(OSPF\)](#) was developed by the interior gateway protocol (IGP) working group of the Internet Engineering Task Force (IETF). The working group was formed in 1988 to design an IGP based on shortest path algorithms, because in the mid-1980s, RIP was increasingly unable to serve larger networks.
 - advantage :OSPF has full network knowledge and may enable one to calculate routes that satisfy other criteria
 - get the shortest distance in terms of hops instead of metric
 - constrained shortest path

OSPF : Open Shortest Path First

- each router “broadcasts” to the other routers all information of its local connectivity (the set of its neighbors), called LSA (Link State Advertisements)
 - classless : the updates include the subnet mask of each route it knows about, enabling variable-length subnet masks
- Goal : record the same info about the network in each node

- Each node/router knows the entire network structure



- it can independently compute the path to reach the destination by calling Dijkstra's algorithm

The main steps of executing OSPF

The information recorded by each router about its neighbors is called LSDB (Link State Database)

- 1 Become OSPF neighbors : two connected routers on the same link create a neighbor relationship.

The main steps of executing OSPF

The information recorded by each router about its neighbors is called LSDB (Link State Database)

- ➊ Become OSPF neighbors : two connected routers on the same link create a neighbor relationship.
- ➋ Exchange database topology information : after becoming the neighbors, the two routers exchange the LSDB information with each other.

The main steps of executing OSPF

The information recorded by each router about its neighbors is called LSDB (Link State Database)

- ➊ Become OSPF neighbors : two connected routers on the same link create a neighbor relationship.
- ➋ Exchange database topology information : after becoming the neighbors, the two routers exchange the LSDB information with each other.
- ➌ Choose the best route : once the LSDB information has been exchanged with each other, call Dijkstra to determine the best route to be inserted into a routing table

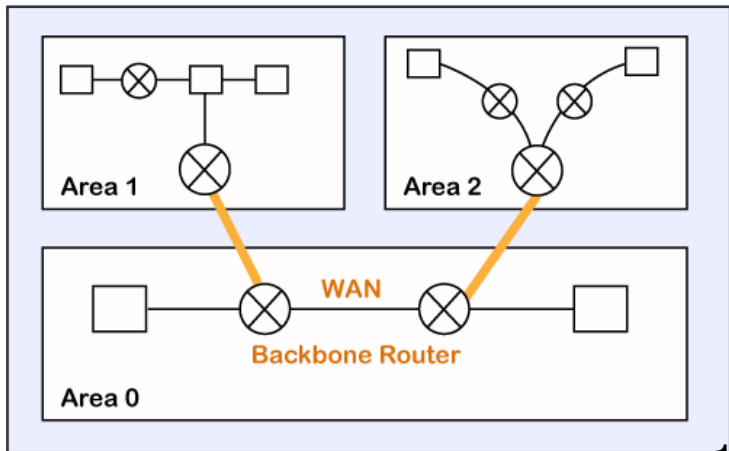
OSPF Areas

- In a **big and rapidly-evolving network**, the frequency of the topology updates and the time needed to run Dijkstra may be too large. This is why OSPF is only used inside an AS.

OSPF Areas

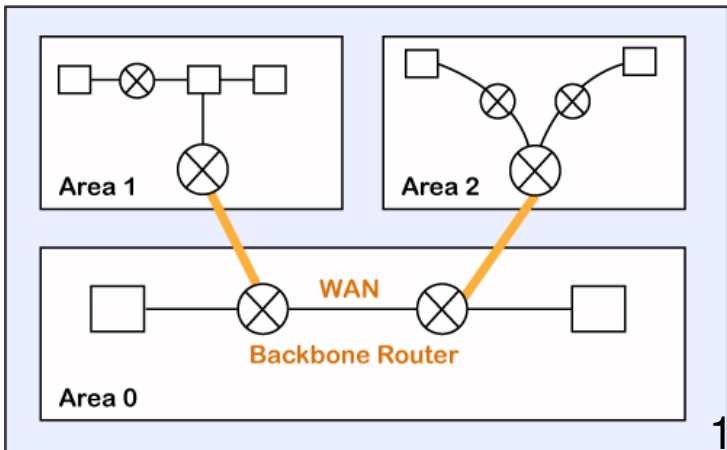
- In a **big and rapidly-evolving network**, the frequency of the topology updates and the time needed to run Dijkstra may be too large. This is why OSPF is only used inside an AS.

Still, OSPF can use areas :



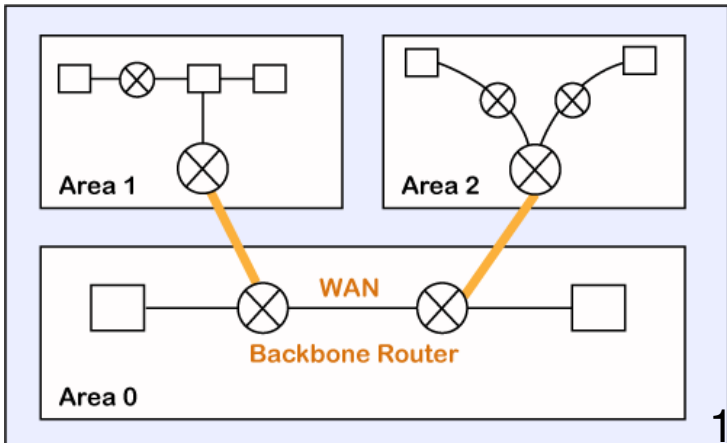
OSPF Areas

- Routers inside an area **flood only the area** with routing info
- Special routers at the border of an area are called Area Border Routers (big routers below).
 - Such a router summarizes the information about an area and shares the information with other Area Border Routers



OSPF Areas

- Routers inside an area **flood only the area** with routing info
- Special routers at the border of an area are called Area Border Routers (big routers below).
 - All communication between areas has to go through a primary area (0) that contain Backbone routers



- 1 Breadth first search in a routing network
- 2 Recalling Breadth-First-Search : a new pseudo-code
- 3 Dijkstra's algorithm is a generalization of Breadth First Search
- 4 Bellman-Ford shortest path algorithm in the *Routing Information Protocol*

Breadth-first-Search for network mapping

- Network mapping means to systematically discover, document, and visually represent the devices, connections, and topology of a network.

Breadth-first-Search for network mapping

- Network mapping means to systematically discover, document, and visually represent the devices, connections, and topology of a network.
 - This helps understanding how the routers are interconnected and how data flows through the network.

Breadth-first-Search for network mapping

- Network mapping means to systematically discover, document, and visually represent the devices, connections, and topology of a network.
 - This helps understanding how the routers are interconnected and how data flows through the network.
- Breadth First Search (BFS) is used for this purpose, in network exploration, topology discovery, and shortest path calculations in graphs.

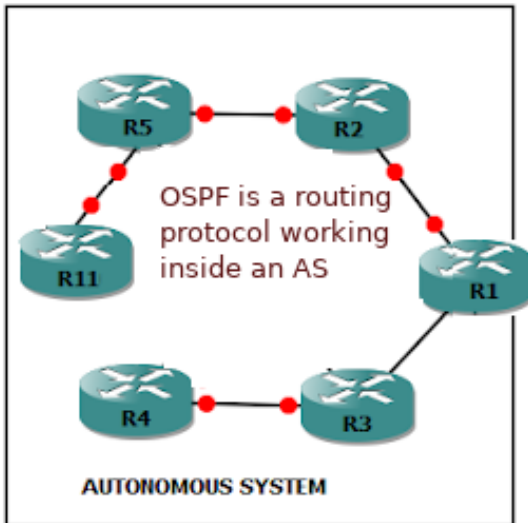
Breadth-first-Search for network mapping

- Network mapping means to systematically discover, document, and visually represent the devices, connections, and topology of a network.
 - This helps understanding how the routers are interconnected and how data flows through the network.
- Breadth First Search (BFS) is used for this purpose, in network exploration, topology discovery, and shortest path calculations in graphs.
- BFS explores nodes layer by layer from a starting point, which makes it effective for traversing a network to discover all reachable nodes and their connections.

- Router R1 may need to explore or discover the other routers and their connections

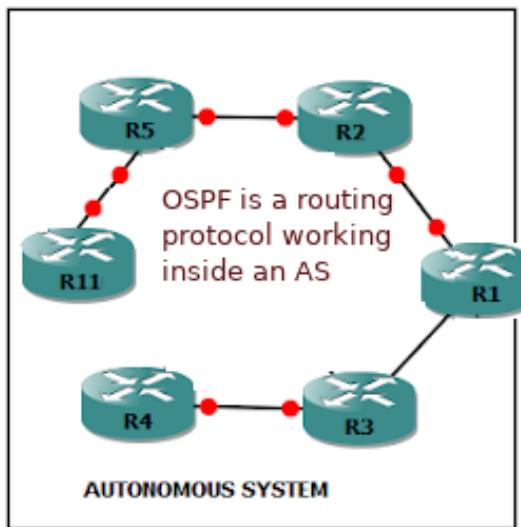


Find a visit order or rank



Depth first search

- 1 Take any neighbor of R1
- 2 Take a neighbor of this neighbor
- 3 At each step, go deeper if possible : follow each branch to the end



Depth first search

- 1 Take any neighbor of R1
- 2 Take a neighbor of this neighbor
- 3 At each step, go deeper if possible : follow each branch to the end

-  *Queen Elizabeth II (1926–2022)*

 **King Charles III**

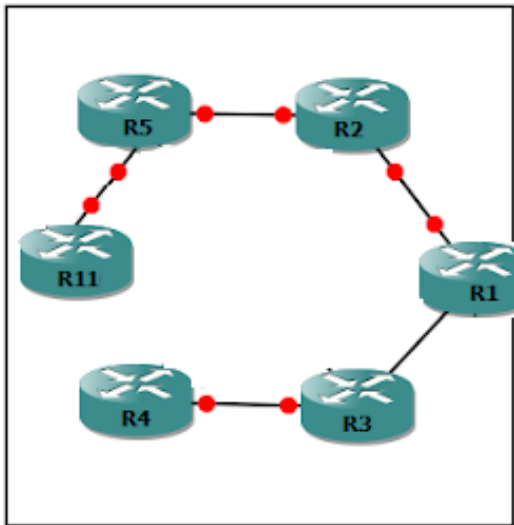
- (1) William, Prince of Wales
 - (2) Prince George of Wales
 - (3) Princess Charlotte of Wales
 - (4) Prince Louis of Wales
- (5) Prince Harry, Duke of Sussex
 - (6) Prince Archie of Sussex
 - (7) Princess Lilibet of Sussex
- (8) Prince Andrew, Duke of York
 - (9) Princess Beatrice

Depth first search

- 1 Take any neighbor of R1
- 2 Take a neighbor of this neighbor
- 3 At each step, go deeper if possible : follow each branch to the end

Breadth first search

- 1 Neighbors of R1 (1 hop)

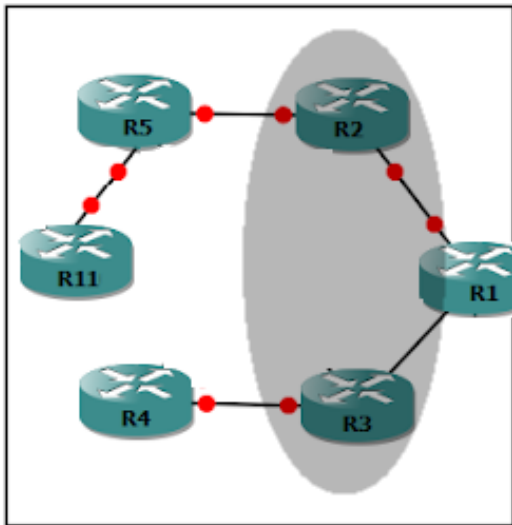


Depth first search

- 1 Take any neighbor of R1
- 2 Take a neighbor of this neighbor
- 3 At each step, go deeper if possible : follow each branch to the end

Breadth first search

- 1 Neighbors of R1 (1 hop)

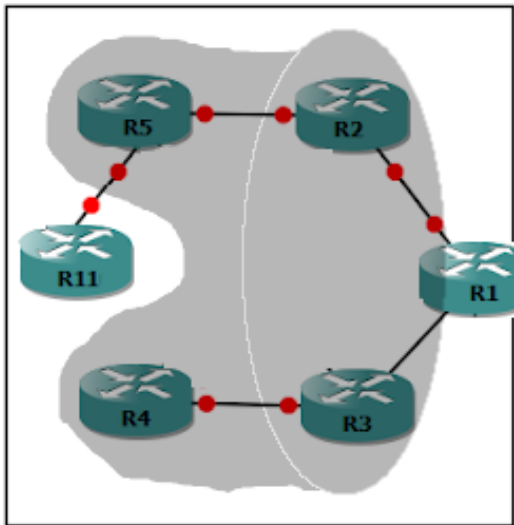


Depth first search

- 1 Take any neighbor of R1
- 2 Take a neighbor of this neighbor
- 3 At each step, go deeper if possible : follow each branch to the end

Breadth first search

- 2 Routers at 2 hops distance, or topological distance 2

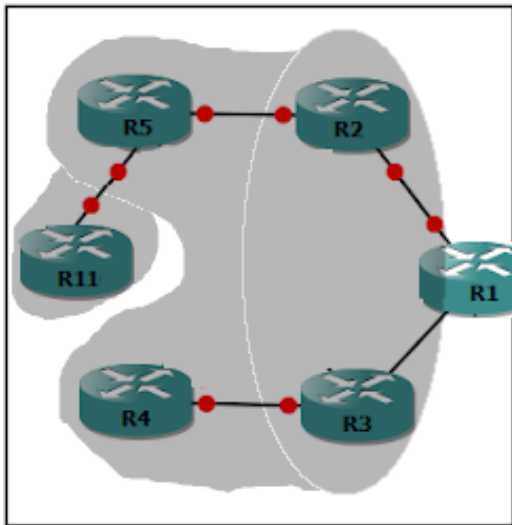


Depth first search

- 1 Take any neighbor of R1
- 2 Take a neighbor of this neighbor
- 3 At each step, go deeper if possible : follow each branch to the end

Breadth first search

- 3 Routers at 3 hops dist.



- BFS gives a network vision in layers
 - 1 first the root's neighbors
 - 2 the neighbors of the neighbors
 - 3 the neighbors of the neighbors of the neighbors
 -

- BFS gives a network vision in layers
 - ① first the root's neighbors
 - ② the neighbors of the neighbors
 - ③ the neighbors of the neighbors of the neighbors
 -
- This also gives the shortest path in hops to each router

- BFS gives a network vision in layers
 - ① first the root's neighbors
 - ② the neighbors of the neighbors
 - ③ the neighbors of the neighbors of the neighbors
 -
- This also gives the shortest path in hops to each router
- You may see topological sorting (with Mrs Kedad)

- BFS gives a network vision in layers
 - ① first the root's neighbors
 - ② the neighbors of the neighbors
 - ③ the neighbors of the neighbors of the neighbors
 -
- This also gives the shortest path in hops to each router
- You may see topological sorting (with Mrs Kedad)
- Find the closest exit point from an AS or local network !
 - it is like searching for the closes exit point from a labyrinth
 - Why study something not used by OSPF ?

- BFS gives a network vision in layers
 - ① first the root's neighbors
 - ② the neighbors of the neighbors
 - ③ the neighbors of the neighbors of the neighbors
 -
- This also gives the shortest path in hops to each router
- You may see topological sorting (with Mrs Kedad)
- Find the closest exit point from an AS or local network !
 - it is like searching for the closes exit point from a labyrinth
 - Why study something not used by OSPF ? We do not want to be “experts” that are lost as soon as you go a bit outside a fixed comfort zone !

- 1 Breadth first search in a routing network
- 2 **Recalling Breadth-First-Search : a new pseudo-code**
- 3 Dijkstra's algorithm is a generalization of Breadth First Search
- 4 Bellman-Ford shortest path algorithm in the *Routing Information Protocol*

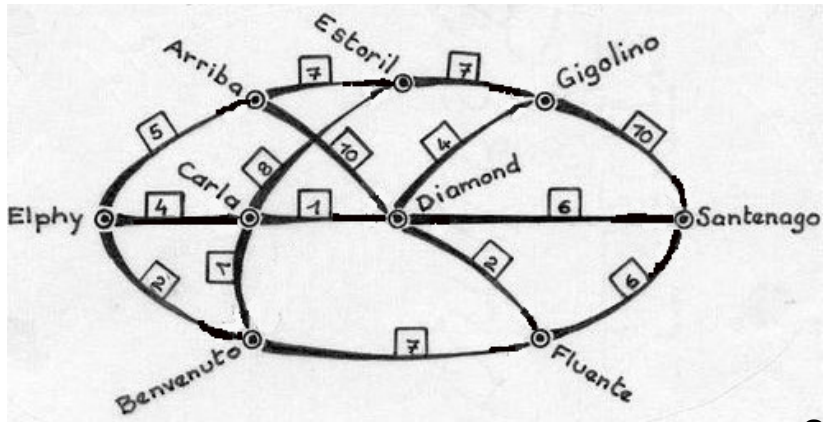
Breadth first search : the pseudo-code

Input: graph $G = (V, E)$ and a vertex $s \in V$.

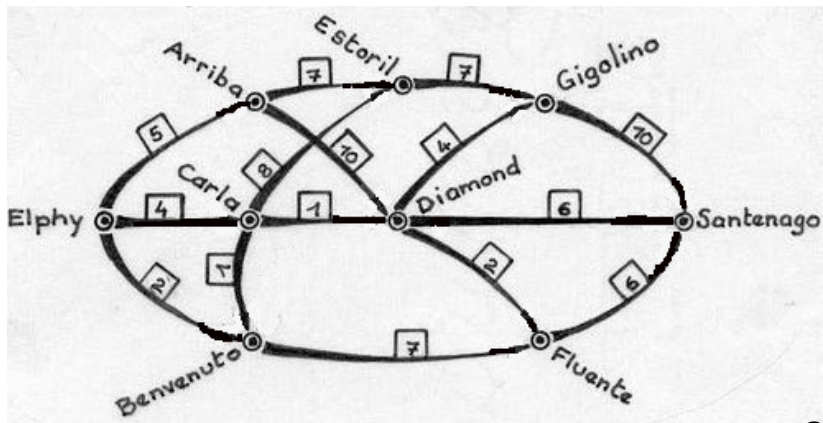
```
1 mark  $s$  as explored, all other vertices as unexplored
2  $Q :=$  a queue data structure, initialized with  $s$ 
3 while  $Q$  is not empty do
4     remove the vertex from the front of  $Q$ , call it  $v$ 
5     for each edge  $(v, w)$  in  $v$ 's adjacency list do
6         if  $w$  is unexplored then
7             mark  $w$  as explored
8             add  $w$  to the end of  $Q$ 
```

- 1 Breadth first search in a routing network
- 2 Recalling Breadth-First-Search : a new pseudo-code
- 3 Dijkstra's algorithm is a generalization of Breadth First Search
- 4 Bellman-Ford shortest path algorithm in the *Routing Information Protocol*

- The queue contains the yet-unvisited nodes
 - These nodes in the queue have to be kept sorted according to their distance towards the root, we have a **priority queue**



- The queue contains the yet-unvisited nodes
 - These nodes in the queue have to be kept sorted according to their distance towards the root, we have a **priority queue**
- Each time we visit a node, we update the distances towards yet-unvisited nodes \Rightarrow the priority queue may change the order



One step from BFS to Dijkstra

Input: graph $G = (V, E)$ and a vertex $s \in V$.

- 1 mark s as explored, all other vertices as unexplored
- 2 $Q :=$ a queue data structure, initialized with s
- 3 **while** Q is not empty **do**
- 4 remove the front vertex v of Q , mark it explored
- 5 **for** each edge (v, w) in v 's adjacency list **do**
- 6 **if** w is unexplored **then**
- 7 add w to Q ,but use it as a priority queue
- 8

Standard BFS \rightarrow Dijkstra

- 1 `qpush(root, distance=0)`
- 2 `visited \leftarrow [False]*n`
- 3 **while**(!qEmpty())
 - 1 `v,dist \leftarrow qHeadPop()`
 - 2 `print("Optimal distance to", v, "found:", dist)`
 - 3 `visited[v] \leftarrow true`
 - 4 **for each** non-visited neighbor *w* of *v* :
 - `qPush(w,dist+MetricVal[v][w]);` (*metric or edge weight*)
 - `qRemoveDuplicates()`

Standard BFS \rightarrow Dijkstra

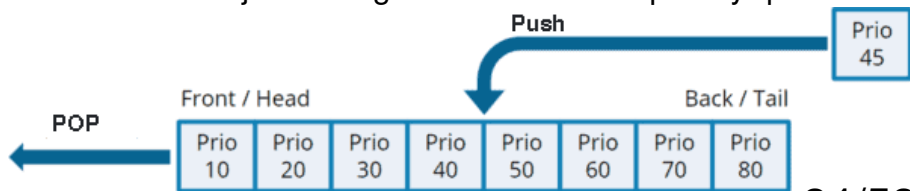
- 1 `qpush(root, distance=0)`
- 2 `visited \leftarrow [False]*n`
- 3 **while**(!qEmpty())
 - 1 `v, dist \leftarrow qHeadPop()`
 - 2 `print("Optimal distance to", v, "found:", dist)`
 - 3 `visited[v] \leftarrow true`
 - 4 **for each** non-visited neighbor *w* of *v* :
 - `qPush(w, dist+MetricVal[v][w]);` (*metric or edge weight*)
 - `qRemoveDuplicates()`

This executes Dijkstra's algorithm if we use a priority queue

Standard BFS \rightarrow Dijkstra

- 1 `qpush(root, distance=0)`
- 2 `visited \leftarrow [False]*n`
- 3 **while**(!qEmpty())
 - 1 `v, dist \leftarrow qHeadPop()`
 - 2 `print("Optimal distance to", v, "found:", dist)`
 - 3 `visited[v] \leftarrow true`
 - 4 **for each** non-visited neighbor w of v :
 - `qPush(w, dist+MetricVal[v][w]);` (*metric or edge weight*)
 - `qRemoveDuplicates()`

This executes Dijkstra's algorithm if we use a priority queue



This is the output of Dijkstra's algorithm

```
PRIQ QUEUE: Elphy/0
```

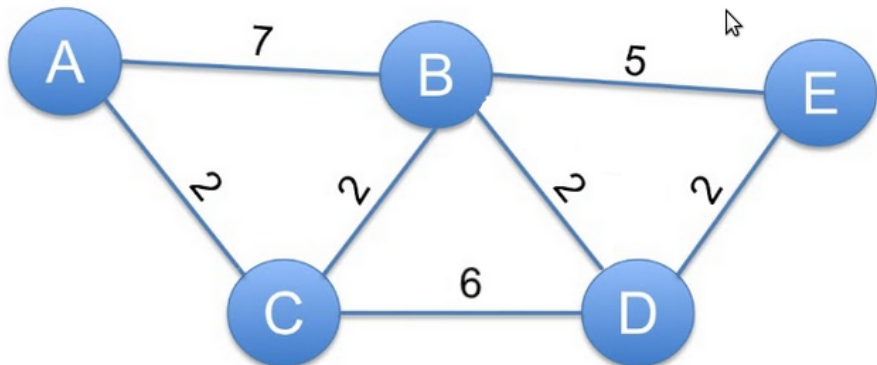
```
-----  
PRIQ QUEUE: Elphy/0 Benvenuto/2 Carla/4 Arriba/5  
VISITED   : Elphy/0
```

```
-----  
PRIQ QUEUE: Carla/4-->3 Arriba/5 Fluente/9  
VISTED    : Elphy/0 Benvenuto/2
```

```
-----  
PRIQ QUEUE: Diamond/4 Arriba/5 Fluente/9  
            Estoril/11  
VISITED    : Elphy/0 Benvenuto/2 Carla/3
```

```
-----  
PRIQ QUEUE: Arriba/5 Fluente/9-->6 Gigolino/8  
            Santenago/10 Estoril/11  
VISITED    : Elphy/0 Benvenuto/2 Carla/3 Diamond/4
```

Exercise : the shortest-path from *A* to *E* !



This is how the output may start

```
A/0  visited A: opt dist = 0    QUEUE:A/0/null
C/2  visited C: opt dist = 2    QUEUE C/2/A:I get to C coming from A
                                   B/7/A:I get to B coming from A
                                   QUEUE B/4/C:I get to B coming from C
                                   D/8/C:I get to D coming from C
```

- 1 Breadth first search in a routing network
- 2 Recalling Breadth-First-Search : a new pseudo-code
- 3 Dijkstra's algorithm is a generalization of Breadth First Search
- 4 Bellman-Ford shortest path algorithm in the *Routing Information Protocol*

Shortest path algorithms overview

- positive edge weights \implies Dijkstra is the best, especially because OSPF knows the complete network

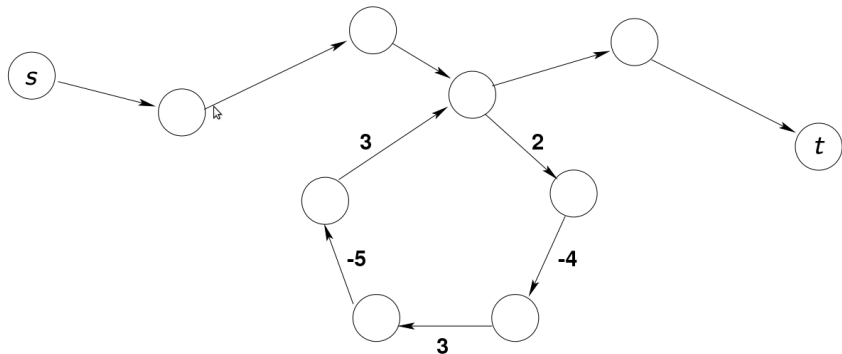
Shortest path algorithms overview

- positive edge weights \implies Dijkstra is the best, especially because OSPF knows the complete network
- Arbitrary weights
 - Bellman-Ford
 - Useful in RIP routing : each node can independently compute the shortest path only by accessing its neighbors – no need to know the full network
 - RIP (Routing Information Protocol) : a distance-vector routing protocol
 - Roy-Floyd-Warshall

There are NO negative weights in routing

Yet, what would happen if any negative weight circuit existed ?

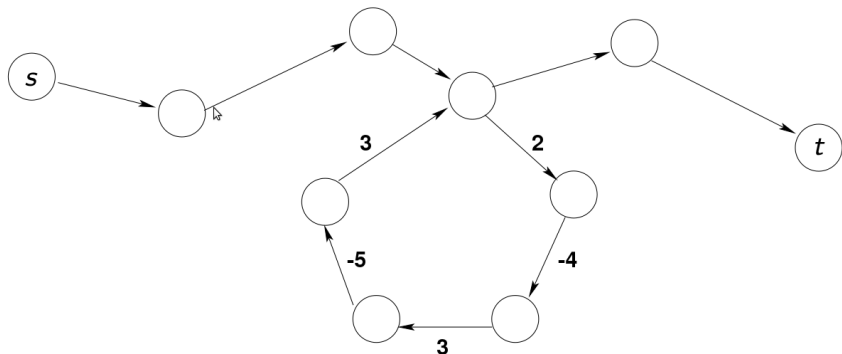
- We do not want to become “experts” that are **lost as soon as we go a bit outside** the comfort zone !



There are NO negative weights in routing

Yet, what would happen if any negative weight circuit existed?

- We do not want to become “experts” that are **lost as soon as we go a bit outside** the comfort zone!



Is there any shortest path from source s to target t ?

Bellman Ford algorithm

Notations :

Input A weighed graph $G(V, E, w)$ with n vertices V and a source s

Output

- 1 The shortest path $SP(v)$ from s to any vertex v , and also

Bellman Ford algorithm

Notations :

Input A weighed graph $G(V, E, w)$ with n vertices V and a source s

Output

- 1 The shortest path $SP(v)$ from s to any vertex v , and also
- 2 $SP(v, i)$ the shortest path to v using at most i edges, equivalent to i hops excluding s including v

Bellman Ford algorithm

Notations :

Input A weighed graph $G(V, E, w)$ with n vertices V and a source s

Output

- ① The shortest path $SP(v)$ from s to any vertex v , and also
- ② $SP(v, i)$ the shortest path to v using at most i edges, equivalent to i hops excluding s including v
 - it answers this : what is the shortest path to s with a hop count of i ?
 - Note : RIP has a maximum hop count of 15 : no more than 15 hops to get anywhere

Bellman Ford algorithm

Notations :

Input A weighed graph $G(V, E, w)$ with n vertices V and a source s

Output

- 1 The shortest path $SP(v)$ from s to any vertex v , and also
- 2 $SP(v, i)$ the shortest path to v using at most i edges, equivalent to i hops excluding s including v
 - it answers this : what is the shortest path to s with a hop count of i ?
 - Note : RIP has a maximum hop count of 15 : no more than 15 hops to get anywhere

We have no negative weight circuit

The shortest path won't visit the same vertex twice \implies

Bellman Ford algorithm

Notations :

Input A weighed graph $G(V, E, w)$ with n vertices V and a source s

Output

- 1 The shortest path $SP(v)$ from s to any vertex v , and also
- 2 $SP(v, i)$ the shortest path to v using at most i edges, equivalent to i hops excluding s including v
 - it answers this : what is the shortest path to s with a hop count of i ?
 - Note : RIP has a maximum hop count of 15 : no more than 15 hops to get anywhere

We have no negative weight circuit

The shortest path won't visit the same vertex twice \implies the shortest path has at maximum $n - 1$ edges so

$$SP(v) = SP(v, n - 1)$$

Starting the algorithm

We initialize :

- $SP(s, 0) = SP(s, 1) = \dots SP(s, n - 1) = 0$
 - there is no cost to go from s to s
- For all neighbors v of s , set $SP(v, 1) = w(s, v)$, where recall w is the metric/distance/weight of edge $\{s, v\}$.

Starting the algorithm

We initialize :

- $SP(s, 0) = SP(s, 1) = \dots SP(s, n - 1) = 0$
 - there is no cost to go from s to s
- For all neighbors v of s , set $SP(v, 1) = w(s, v)$, where recall w is the metric/distance/weight of edge $\{s, v\}$.

An intuition of how to compute $SP(v, 2)$?

- Case 1 : if v has only one neighbor u such that $w(u, v) = 7$ and $SP(u, 1) = 10$?

Starting the algorithm

We initialize :

- $SP(s, 0) = SP(s, 1) = \dots SP(s, n - 1) = 0$
 - there is no cost to go from s to s
- For all neighbors v of s , set $SP(v, 1) = w(s, v)$, where recall w is the metric/distance/weight of edge $\{s, v\}$.

An intuition of how to compute $SP(v, 2)$?

- Case 1 : if v has only one neighbor u such that $w(u, v) = 7$ and $SP(u, 1) = 10$?
- Case 2 : if v has a second neighbor u' with $w(u', v) = 6$ and $SP(u', 1) = 9$?

Next layer : move from $i - 1$ to i

We have to compute $SP(v, i)$ and we see two cases :

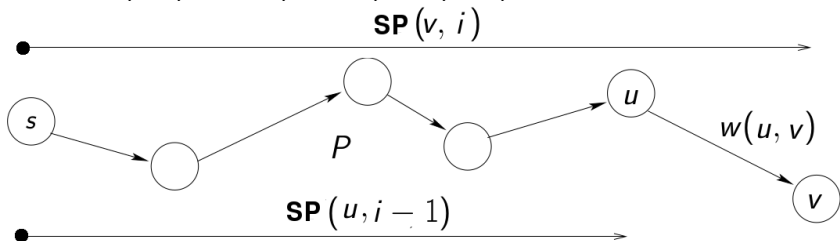
- The shortest path with at most i edges contains $i - 1$ edges
 $\implies SP(v, i) = SP(v, i - 1)$.

Next layer : move from $i - 1$ to i

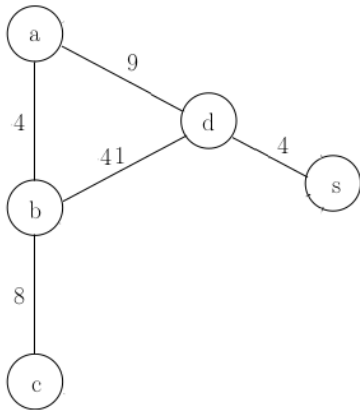
We have to compute $SP(v, i)$ and we see two cases :

- The shortest path with at most i edges contains $i - 1$ edges
 $\implies SP(v, i) = SP(v, i - 1)$.
- The shortest path with at most i edges contains indeed i edges

$\implies SP(v, i) = SP(u, i - 1) + w(u, v)$ for some intermediary u

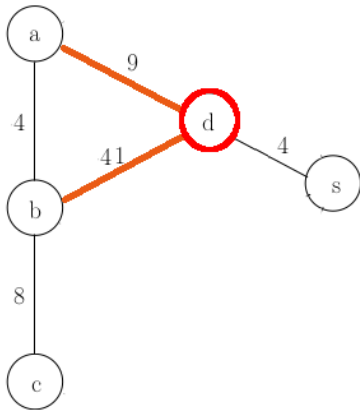


Example : we start from s



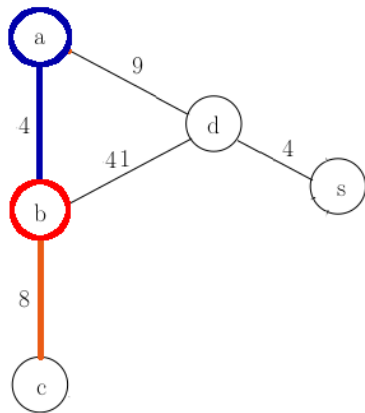
	i=1	i=2	i=3	i=4
a	∞			
b	∞			
c	∞			
d	4			

Example : we start from s



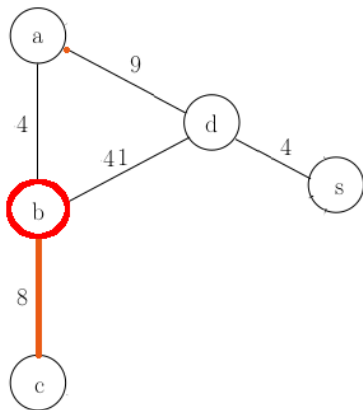
	i=1	i=2	i=3	i=4
a	∞	13		
b	∞	45		
c	∞	∞		
d	4	4		

Example : we start from s



	i=1	i=2	i=3	i=4
a	∞	13	13	
b	∞	45	17	
c	∞	∞	53	
d	4	4	4	

Example : we start from s



	i=1	i=2	i=3	i=4
a	∞	13	13	13
b	∞	45	17	17
c	∞	∞	53	25
d	4	4	4	4

The algorithm will iterate this update

SP=Shortest Path

for each $i = 2, 3, 4, \dots, n - 1$

$$SP(v, i) = \min(SP(v, i - 1), \\ \min_{\{u, v\} \in E} SP(u, i - 1) + w(u, v) \\)$$

The first pseudo-code

- 1 For each vertex v in the graph
 - if v is a neighbor of s , $SP(v, 1) = w(s, v)$
 - else, $SP(v, 1) = \infty$
- 2 For $i = 2$ to $n - 1$
 - For each node $v \in V$
$$SP(v, i) = \min(SP(v, i - 1),$$
$$\min_{\{u, v\} \in E} SP(u, i - 1) + w(u, v)$$
$$)$$

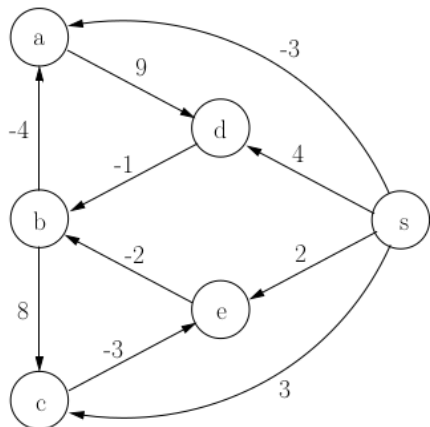
Another pseudo-code formulation

- 1 For each vertex $v \in V$
 - if v is a neighbor of s , $SP(v, 1) = w(v, s)$
 - else, $SP(v, 1) = \infty$
- 2 For $i = 2$ to $n - 1$
 - For each node $v \in V$
 - $SP(v, i) = SP(v, i - 1)$;
 - For each edge $\{u, v\}$
 - $cost_via_u = SP(u, i - 1) + w(u, v)$
 - if $SP(v, i) > cost_via_u$
 $SP(v, i) = cost_via_u$

Pseudo-code *Bellman-Ford* with directed arcs

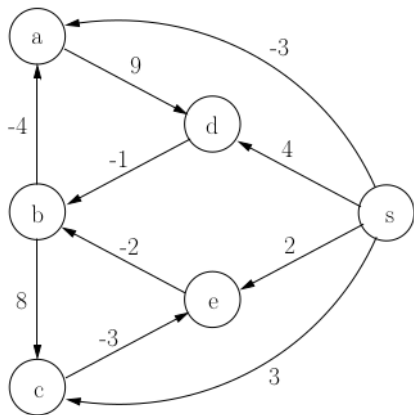
- 1 For each $v \in V$
 - if v is a neighbor of s , $SP(v, 1) = w(v, s)$
 - else, $SP(v, 1) = \infty$
- 2 For $i = 2$ to $n - 1$
 - For each node $v \in V$
 - $SP(v, i) = SP(v, i - 1)$;
 - For each arc (u, v) , i.e., for each arc $u \rightarrow v$
 - $cost_via_u = SP(u, i - 1) + w(u, v)$
 - if $SP(v, i) > cost_via_u$
 $SP(v, i) = cost_via_u$

Example with uni-directional arcs



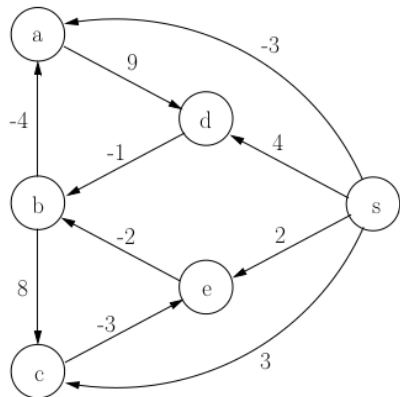
	1	2	3	4	5
<i>s</i>	0				
<i>a</i>	-3				
<i>b</i>	∞				
<i>c</i>	3				
<i>d</i>	4				
<i>e</i>	2				

Example with uni-directional arcs



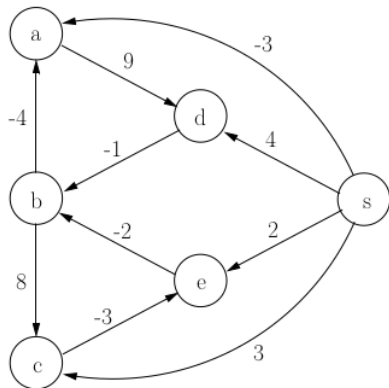
	1	2	3	4	5
s	0	0			
a	-3	-3			
b	∞	0			
c	3	3			
d	4	4			
e	2	0			

Example with uni-directional arcs



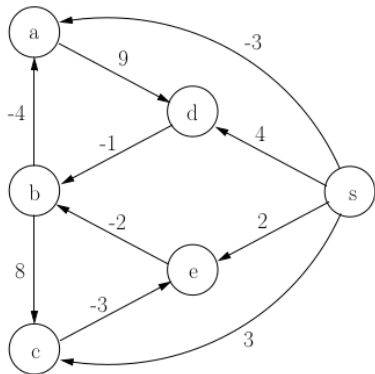
	1	2	3	4	5
s	0	0	0		
a	-3	-3	-4		
b	∞	0	-2		
c	3	3	3		
d	4	4	4		
e	2	0	0		

Example with uni-directional arcs



	1	2	3	4	5
s	0	0	0	0	
a	-3	-3	-4	-6	
b	∞	0	-2	-2	
c	3	3	3	3	
d	4	4	4	4	
e	2	0	0	0	

Example with uni-directional arcs



	1	2	3	4	5
s	0	0	0	0	0
a	-3	-3	-4	-6	-6
b	∞	0	-2	-2	-2
c	3	3	3	3	3
d	4	4	4	4	3
e	2	0	0	0	0

Applications in routing [\[edit \]](#)

A distributed variant of the Bellman–Ford algorithm is used in [distance-vector routing protocols](#), for example the [Routing Information Protocol](#) (RIP). The algorithm is distributed because it involves a number of nodes (routers) within an [Autonomous system \(AS\)](#), a collection of IP networks typically owned by an ISP. It consists of the following steps:

Applications in routing [\[edit \]](#)

A distributed variant of the Bellman–Ford algorithm is used in [distance-vector routing protocols](#), for example the [Routing Information Protocol](#) (RIP). The algorithm is distributed because it involves a number of nodes (routers) within an [Autonomous system \(AS\)](#), a collection of IP networks typically owned by an ISP. It consists of the following steps:

1. Each node calculates the distances between itself and all other nodes in the AS and stores this information as a table.
2. Each node sends its table to all neighboring nodes.
3. When a node receives distance tables from its neighbors, it calculates the shortest routes to all other nodes and updates its own table to reflect any changes.

Applications in routing [\[edit \]](#)

A distributed variant of the Bellman-Ford algorithm is used in [distance-vector routing protocols](#), for example the [Routing Information Protocol](#) (RIP). The algorithm is distributed because it involves a number of nodes (routers) within an [Autonomous system \(AS\)](#), a collection of IP networks typically owned by an ISP. It consists of the following steps:

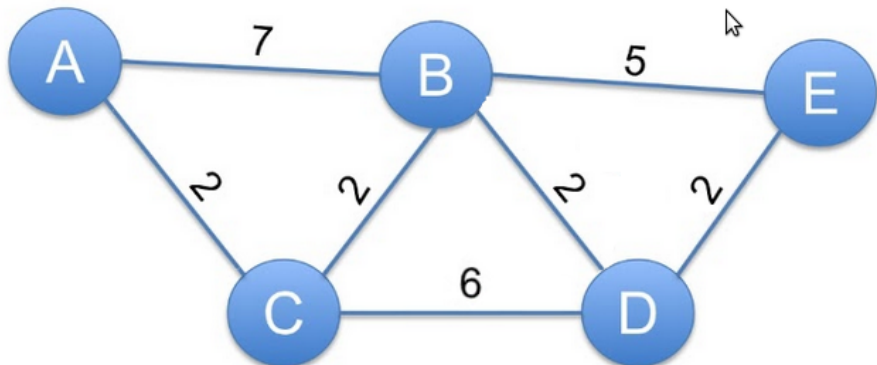
1. Each node calculates the distances between itself and all other nodes in the AS and stores this information as a table.
2. Each node sends its table to all neighboring nodes.
3. When a node receives distance tables from its neighbors, it calculates the shortest routes to all other nodes and updates its own table to reflect any changes.

The main disadvantage of the Bellman-Ford algorithm :

1 / 1

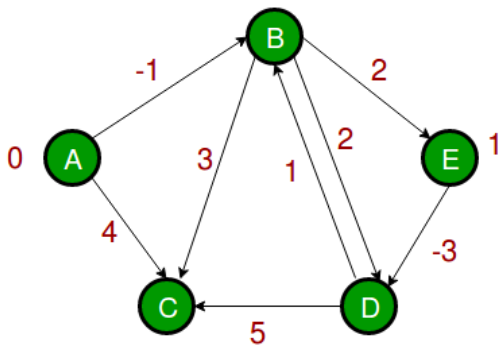
- Changes in [network topology](#) are not reflected quickly since updates are spread node-by-node.

Exercise : the shortest-path from *A* to *E* !



Exercise with some negative weights

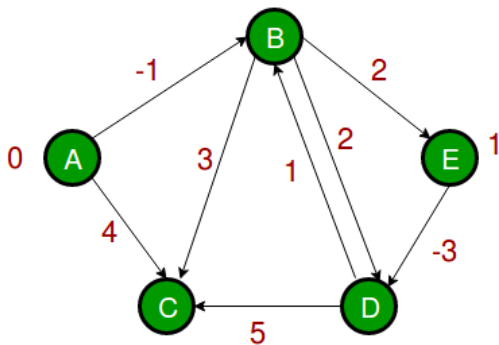
A B C D E



What is the first step or initialization?

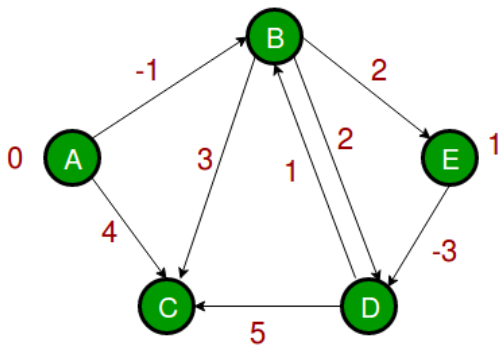
Exercise with some negative weights

A	B	C	D	E
0	-1	4	∞	∞



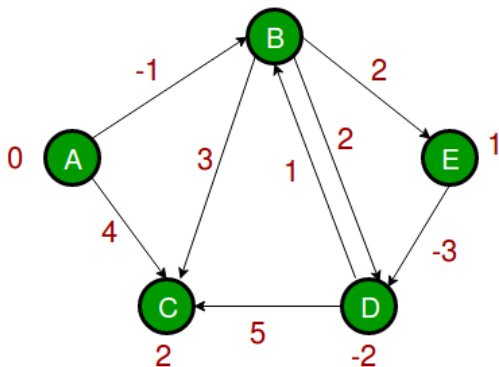
Exercise with some negative weights

A	B	C	D	E
0	-1	4	∞	∞
0	-1	2	1	1

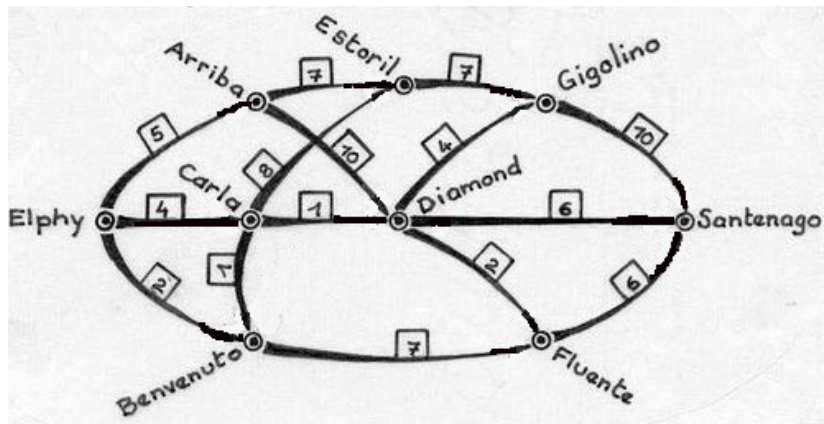


Exercise with some negative weights

A	B	C	D	E
0	-1	4	∞	∞
0	-1	2	1	1
0	-1	2	-2	1



Exercise to compare against Dijkstra



Usage overview : Diskstra and Bellman-Ford

Remember :

- Dijkstra is useful in link-state routing when each node/router has full knowledge of the map

Usage overview : Diskstra and Bellman-Ford

Remember :

- Dijkstra is useful in link-state routing when each node/router has full knowledge of the map
- Bellman-Ford is useful in distance-vector routing, where each node/router only maintains a **routing table** of the form
 - destination 1** : go via neighbor A with a metric/cost of a
 - destination 2** : go via neighbor B with a metric/cost of b
 - destination 3** : go via neighbor C with a metric/cost of c
- Recall : RIP has a maximum hop count of 15 (no more than 15 hops to get anywhere)

A third algorithm : Roy-Floyd-Warshall

Initialize each distance d_{ij}^1 with the metric/cost of edge $\{i, j\}$.


ROY-FLOYD-WARSHALL

1. **for** $k \leftarrow 1$ **to** n
2. **do for** $i \leftarrow 1$ **to** n
3. **do for** $j \leftarrow 1$ **to** n
4. $d_{ij}^k \leftarrow \min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1})$

Roy-Floyd-Warshall Version 2

Initialize each distance d_{ij}^1 with the metric/cost of edge $\{i, j\}$.


ROY-FLOYD-WARSHALL

1. **for** $k \leftarrow 1$ **to** n
 2. **do for** $i \leftarrow 1$ **to** n
 3. **do for** $j \leftarrow 1$ **to** n
 4. $d_{ij}^k \leftarrow \min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1})$
- 

Roy-Floyd-Warshall Version 2

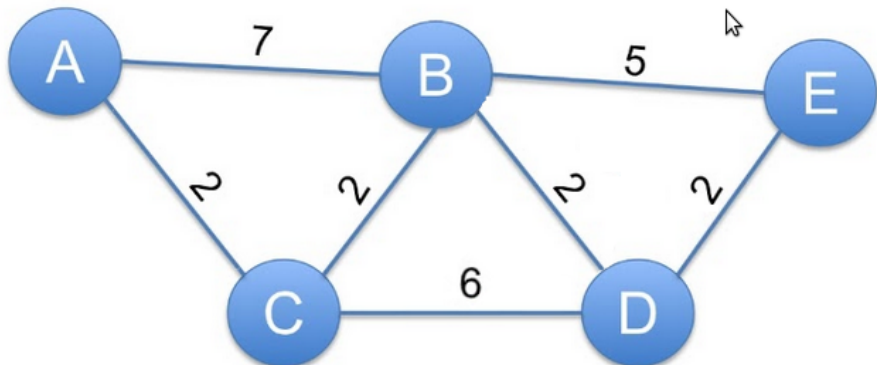
Initialize each distance d_{ij}^1 with the metric/cost of edge $\{i, j\}$.

ROY-FLOYD-WARSHALL

1. **for** $k \leftarrow 1$ **to** n
 2. **do for** $i \leftarrow 1$ **to** n
 3. **do for** $j \leftarrow 1$ **to** n
 4. $d_{ij}^k \leftarrow \min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1})$
- 

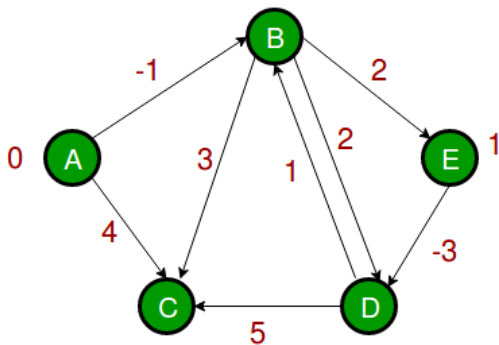
Question : does k really needs to go up to n ?

Exercise : the shortest-path from *A* to *E* !



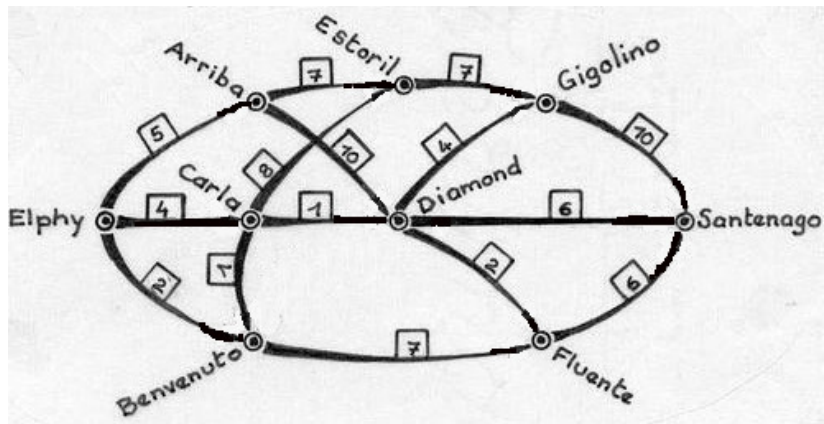
Exercise with some negative weights

A B C D E



What is the first step or initialization?

Exercise to compare against Dijkstra



Routing inside an Autonomous System (AS)

An AS is typically an IP network owned by an ISP, where two types of protocols can be used :

- ① **link-state routing protocols** : every node constructs a map of the connectivity to the network, in the form of a graph, showing which nodes are connected to which other nodes.
 - Open Shortest Path First (OSPF)
 - Intermediate System to Intermediate System

Routing inside an Autonomous System (AS)

An AS is typically an IP network owned by an ISP, where two types of protocols can be used :

- 1 **link-state routing protocols** : every node constructs a map of the connectivity to the network, in the form of a graph, showing which nodes are connected to which other nodes.
 - Open Shortest Path First (OSPF)
 - Intermediate System to Intermediate System
- 2 **distance-vector routing protocols** : The term distance vector refers to the fact that the protocol manipulates vectors (arrays) of distances to other nodes in the network.
 - Routing Information Protocol (RIP) \oplus RIPv2, IGRP.