Part II: Search for Shortest Path

Introduction

- One of the core challenges in graph theory is determining the route with the smallest distance between two nodes, commonly known as the **shortest path problem**.
- This concept underpins a wide range of real-world applications, including **network routing**, route planning, traffic control, and pathfinding in games and transportation systems.
- In this chapter, we explore the problem in detail, review the most commonly used algorithms to solve it, and emphasize their practical application in **routing protocols**.

Problem & définitions

- The shortest path problem (SPP) involves finding the **least-cost** (distance) path between two vertices in a weighted graph, whether directed or undirected.
- The **cost of a path** in SPP is the sum of all costs of the edges that making it up.
  - Ex: λ (A,X) = d(A,B)+d(B,C)+d(C,X)
- SPP make no sense in case of **absorbing circuits** (of negative cost) sinse the least-cost is - ∞, so the absence of such a circuit is required in SPP algorithms.
  - Ex: λ (A,X) = d(A,B)+d(B,C)+d(C,X) = 2-5+1 = -2

Type of shortest path problems

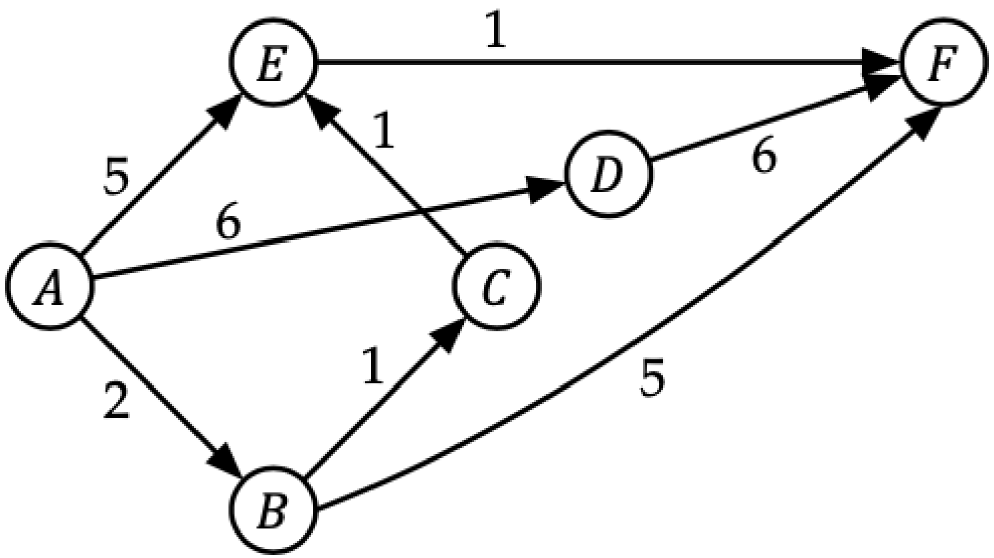
- Shortest path between two specified vertices.
- Shortest paths between all pairs of vertices.
- Shortest paths from a specified vertex to all others.
- Shortest path between specified vertices that passes through specified vertices.
- The second, third, and so on, shortest paths.

Dijkstra’s Algorithm

$S = \{x_0\}$
$\lambda x_0 = 0$
for each successor x_i of x_0
    $\lambda x_i = d(x_0, x_i)$
for each non-successor x_j of x_0
    $\lambda x_j = \infty$
while $S \neq X$
    Choose $x_k \notin S$ such that $\lambda x_k = \min_{x_l \notin S} \lambda x_l$
    $S = S \cup \{x_k\}$
    for each $x_m \in \Gamma^+(x_k) - S$
        $\lambda x_m = \min(\lambda x_k + d(x_k, x_m), \lambda x_m)$

1. **Initialization:** Set distance to source = 0, all others = ∞; mark all nodes as unvisited.
2. **Selection:** Choose the unvisited node with the smallest distance.
3. **Relaxation:** Update the distances for all neighbors of the current node.
4. **Iteration:** Mark the current node as visited and repeat until all nodes are processed.

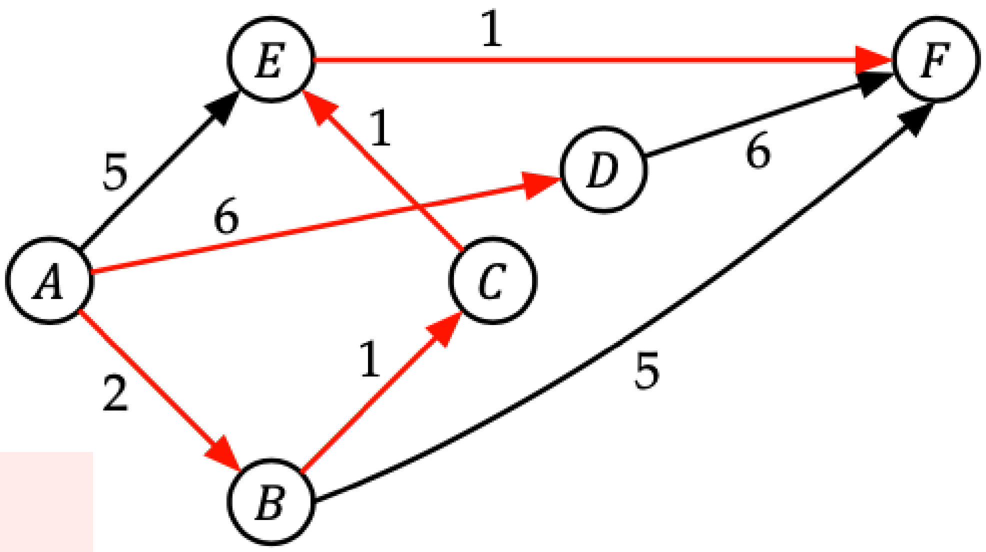
Dijkstra's Algorithm (Example)



S	λA	λB	λC	λD	λE	λF
A	0	**2**	∞	6	5	∞
A, B	0	2	**3**	6	5	9
A, B, C	0	2	3	6	**4**	9
A, B, C, E	0	2	3	6	4	**5**
A, B, C, E, F	0	2	3	**6**	4	5
A, B, C, E, F, D	0	2	3	6	4	5

Dijkstra’s Algorithm (SP Arborescence/tree)

- Dijkstra algorithm consider only graphs with positive weights



for each edge $(x_i, x_j) \in E$
    if $\lambda x_i - \lambda x_j = d(x_i, x_j)$
        The arc (x_i, x_j) belongs **to** the arborescence

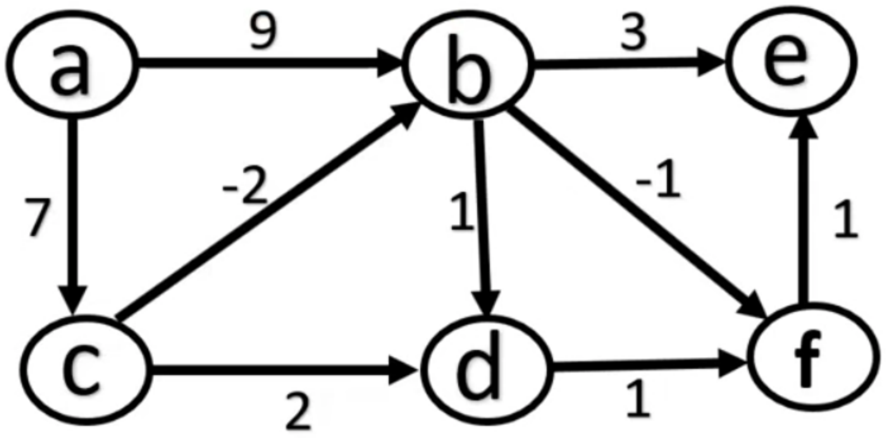
Bellman-Ford's Algorithm

```
d(v[1]) ← 0
for j = 2,..,n do
    d(v[j]) ← ∞
for i = 1,..,(|V|-1) do
    for all (u,v) in E do
        d(v) ← min(d(v), d(u) + l(u,v))
for all (u,v) in E do
    if d(v) > d(u) + l(u,v) do
        Message: "Negative Cycle"
```

1. **Initialization:** Set the distance for the source to 0 and all others to ∞.
2. **Relaxation:** For |V|-1 iterations, update the distance to each vertex by considering each edge.
3. **Cycle Check:** Verify that no negative weight cycles exist.

Bellman-Ford's Algorithm (Example)

(ab), (ac), (bd), (cd), (be), (df), (fe), (bf), (cb)



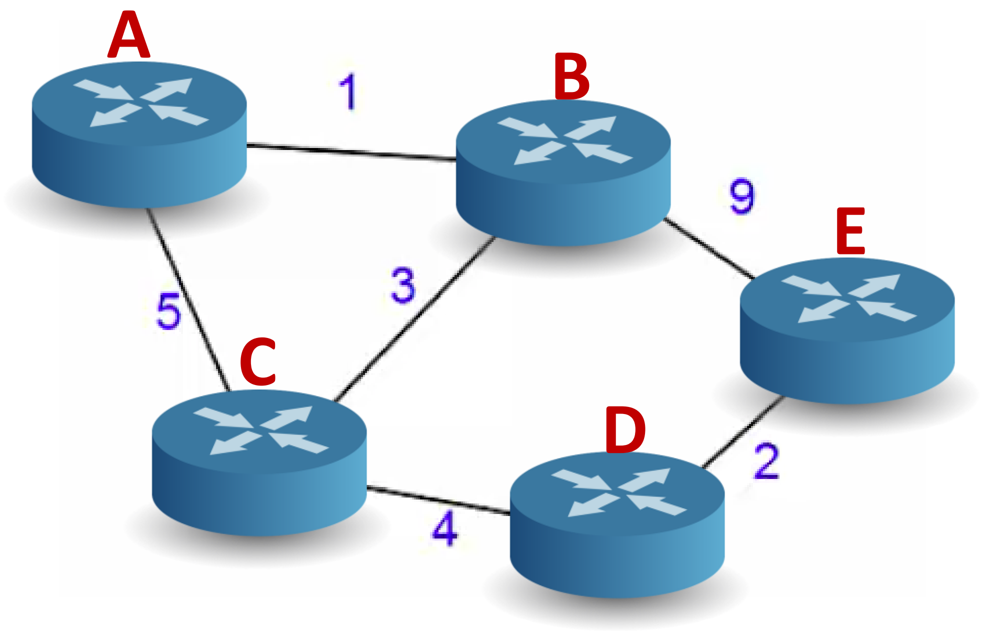
	a	b	c	d	e	f
initially	0	∞	∞	∞	∞	∞
1	0	5	7	9	11	8
2	0	5	7	6	8	4
3	0	5	7	6	5	4
4	0	5	7	6	5	4
5						

RIP (Routing Information Protocol)

- **RIP** determines the best path based on hop count.
- Routers share their **routing tables** (distance vectors) with immediate neighbors periodically.
- Each router updates its own table using the **Bellman-Ford algorithm**.
- In smaller networks, RIP offers a simple solution for routing. Routers **exchange information** to **update routes** based on **hop counts**.

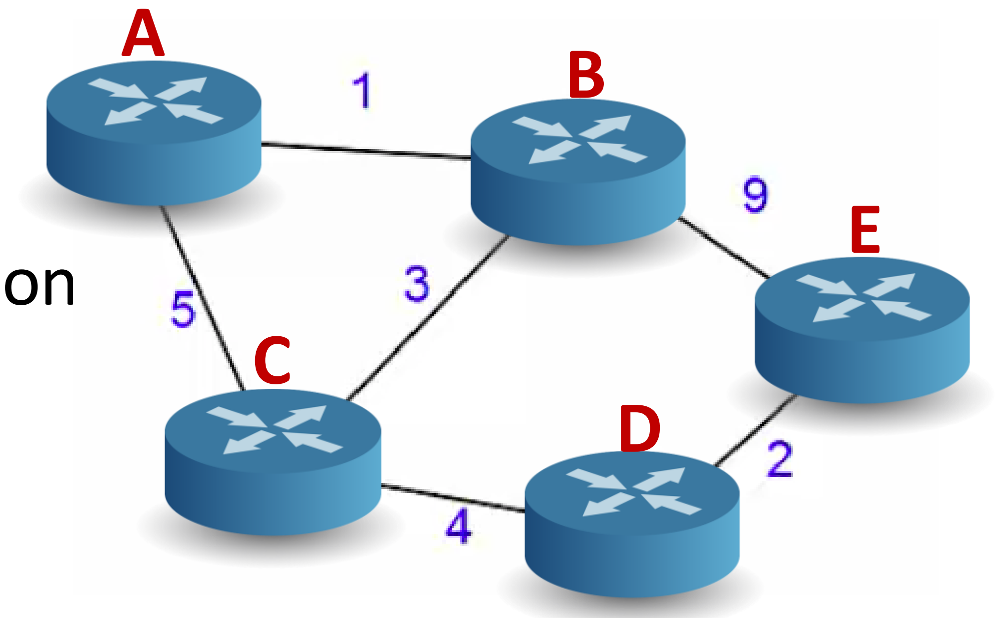
RIP - Protocol framework

- **Initial state**: all neighbors costs are known
- **Final state**: all nodes costs are known with the next hop
- Need to handle:
  - What information to exchange ?
  - How to act on a message ?
  - When to send a message ?



RIP - Protocol framework

- Each node maintains a routing table (distance victor)
- Table information: Destination, Destination cost, next hop to reach destination
- Intial state: cost to neighbors
- Updating table: use Bellman rule
- Final state: cost to all nodes



Initial Routing table at B

Dest	Cost	Next Hop
A	1	A
C	3	C
E	9	E

Dest	Cost	Next Hop
A	1	A
C	3	C
D	7	C
E	9	E

Final Routing table at B

RIP - Example

Initial state

D	C	H
A	5	A
B	3	B
D	4	D

To	A
A	0
B	1
C	5

→

D	C	H
A	5	A
B	3	B
D	4	D

D	C	H
A	5	A
B	3	B
D	4	D

To	B
A	1
B	0
C	3
E	9

→

D	C	H
A	4	B
B	3	B
D	4	D
E	12	B

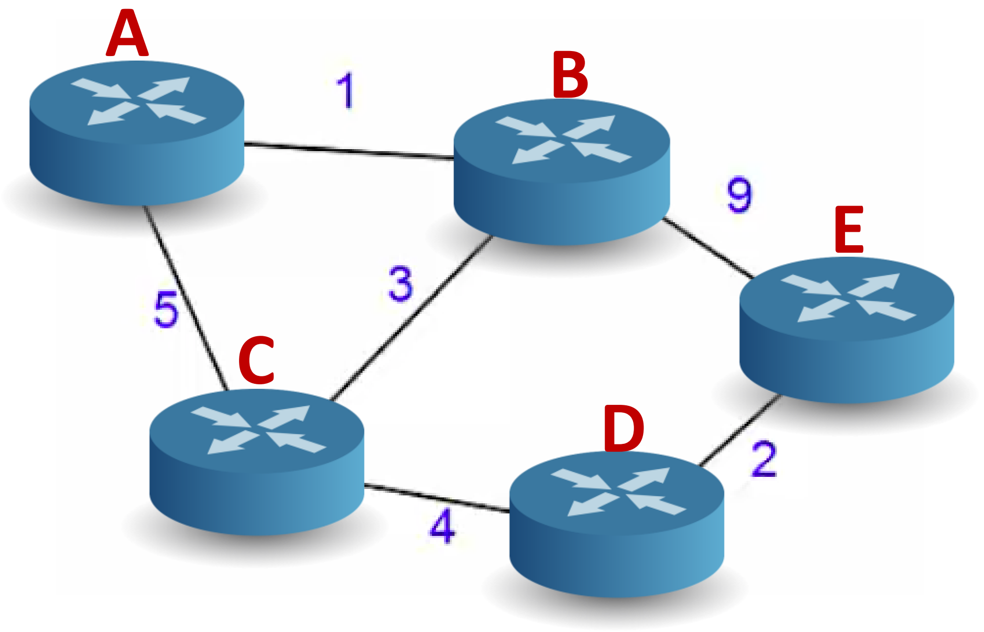
D	C	H
A	4	B
B	3	B
D	4	D
E	12	B

To	D
C	4
D	0
E	2

→

D	C	H
A	4	B
B	3	B
D	4	D
E	6	D

Final state



Distributed Algorithm that works with local knowledge (Bellman-Ford)

OSPF (Open Shortest Path First)

- **OSPF** is a link-state routing protocol that considers link cost instead of hop count, making it more efficient for large-scale networks.
- It calculates the shortest path using **Dijkstra algorithm**.
- It organizes networks into **areas** and exchanges topology information using **Link-State Advertisements** (LSAs).
- OSPF converges faster than RIP but it requires more processing power.

OSPF - Steps

1) Enable the local routing process and choose **RID** (Router Id)
2) Establish neighbor adjacencies (establish **Neighbor tables**)
3) Exchange **LSAs** and build the **Topology table** (**LSDB**):
    a) Sending DBD to neighbors
    b) Exchanges LSAs to Create/Update LSDB throw **LSA-flooding** process
        - LSAs contain each directly connected link's cost and IP settings
        - All routers in the same **area** have the same LSDB.
4) Execute the SPF (Dijkstra’s) algorithm against LSDB:
    a) Best paths are calculated based on the advertised **cost** of each link
    b) Creates the **SPF tree** from the point of view of the local router.
5) Update the **Routing table** with the best paths of the SPF tree

OSPF - Data

LSA		
IP	Interface	Cost
13.0.0.1	s1/0	10 mb/s
172.16.0.1	f0/0	10 kb/s
14.0.0.2	s0/0	1 mb/s

LSA R1

LSA R2

LSA R3

IP	Interface	Cost

…

LSA Rn		
IP	Interface	Cost
13.0.0.1	s1/0	10 mb/s
172.16.0.1	f0/0	10 kb/s
14.0.0.2	s0/0	1 mb/s

LSDB

Neighbors table		
ID	IP	state
1.1.1.1	10.1.1.1	Init
2.2.2.2	10.1.1.2	full

$$\text{Cost} = \frac{\text{Reference Bandwidth}}{\text{Link Bandwidth}}$$

Dijkstra's Algo

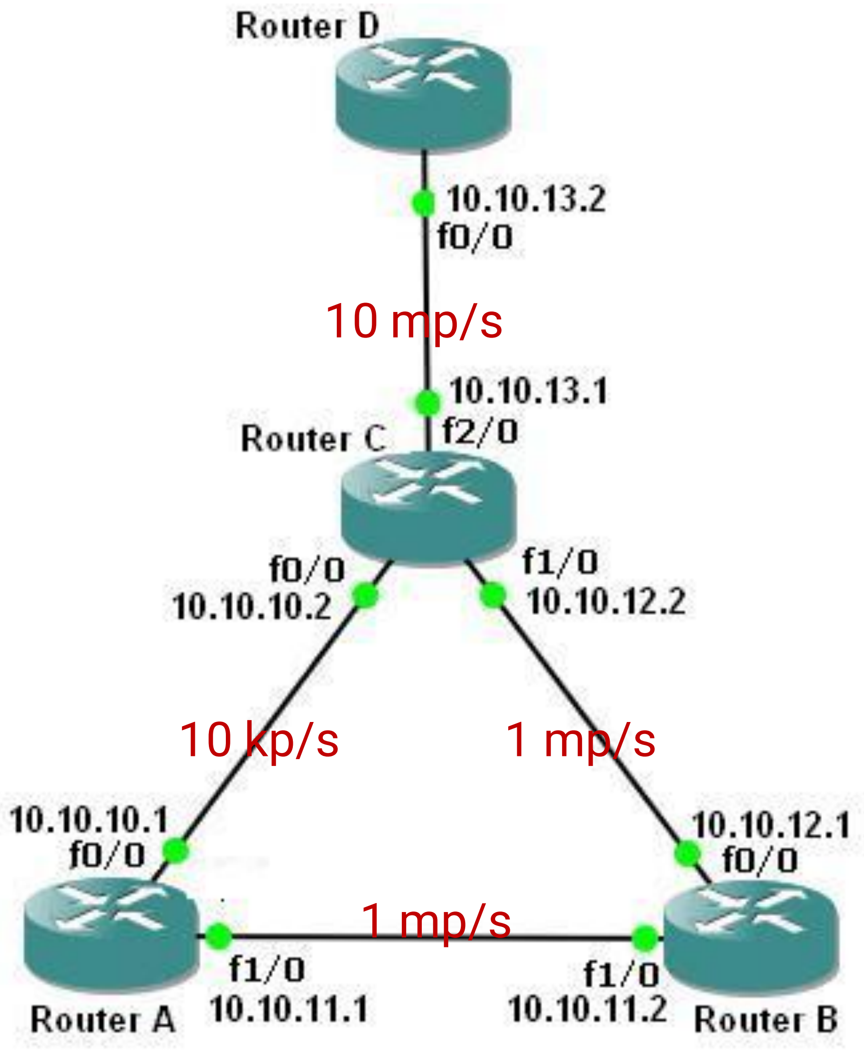
Routing table			
Dest IP	Out interf	Next-hop IP	Cost
13.0.0.1	s1/0	10.1.1.1	10 mb/s
14.16.0.1	f0/0	10.1.1.2	10 kb/s
20.1.1.2	s0/0	10.1.1.2	1 mb/s

OSPF - Dynamic

- Through the exchange of messages, OSPF routers update their routing tables.
- A new router can trigger an update by sending a **Hello** message.
- Periodic messages are also sent to update tables in response to topology changes (link failures, network disruptions, ...)

OSPF - Example

- LSA-A, LSA-B, LSA-C, LSA-D (LSDB) ?
- Costs if Reference Bandwidth = 10^8 ?
- Routing table of Router D ?



TD1 - SP Problem in RIP & OSPF Protocols

Given the following network of routers:

1. Construct the shortest path tree from A and C using the Bellman-Ford algorithm.
2. Build the RIP routing tables for A and C.
3. Compare the execution steps (time/space complexity) in the two previous cases (1 and 2). Do they yield the same results?
4. Construct the Link-State Database (LSDB) of the network.
5. How does the LSDB change after the failure of link BF?
6. Construct the shortest path tree from D and B using Dijkstra's algorithm.
7. Build the OSPF routing tables for D and B, and compare them with the results of (6).