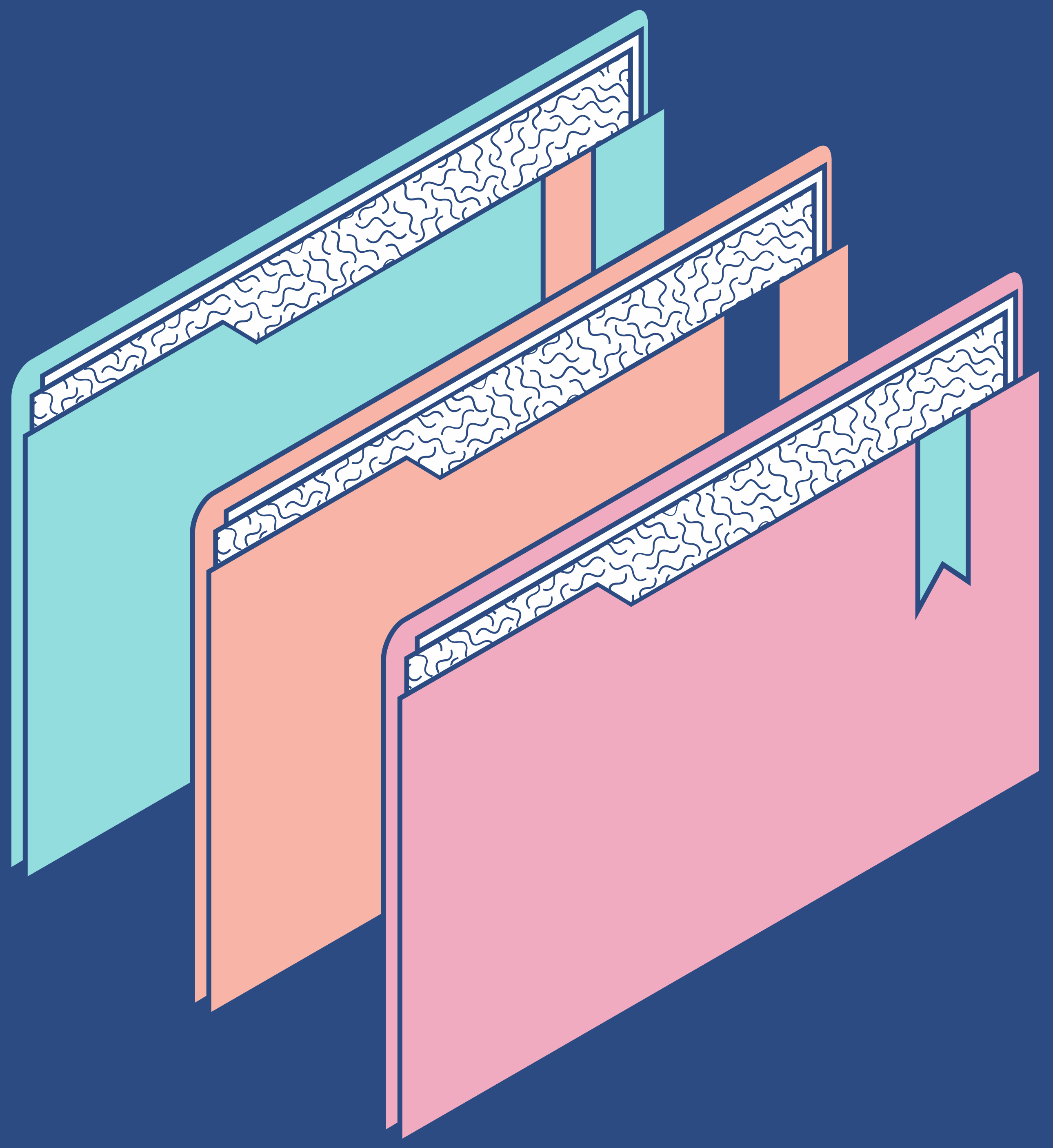
Bidirectional Dijkstra: Optimizing Shortest Paths

ADA Final Project, Spring 2025
Qurba Mushtaque, Hiba Shahid

Paper authored by:Bernhard Haeupler et. al



Agenda

KEY TOPICS DISCUSSED IN
THIS PRESENTATION

- Problem Statement
- Preexisting Algorithms
- Best Option: Bidirectional Dijkstra
- Instance Optimality in light of Correctness
- The algorithm in action
- Demo of code

Problem Statement

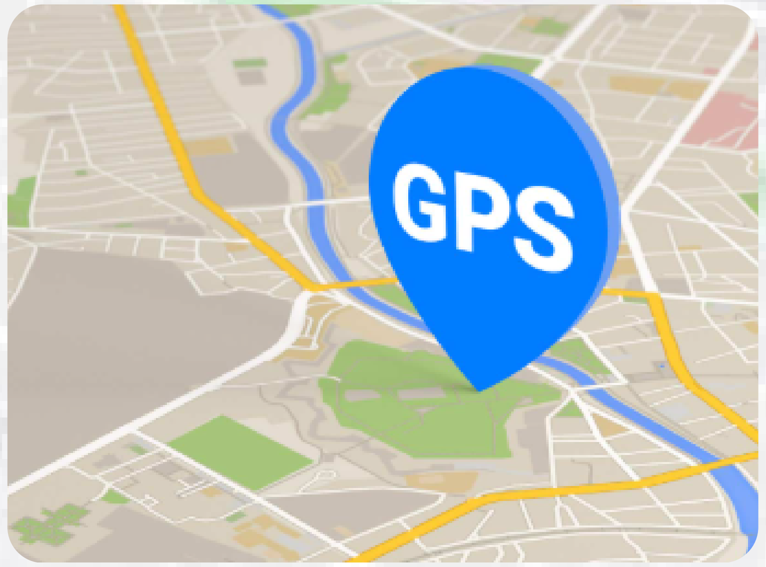
Race Against Time: It's 7:45 AM. Your ADA class starts at 8:30. You overslept, and Karachi's traffic is a nightmare. You need the shortest, fastest route to make it on time. How do apps like Google Maps save you?

Problem Statement: Shortest path in weighted graphs powers navigation, logistics, and gaming, especially in complex cities like Karachi.

Challenge: Standard algorithms explore too many nodes, slowing down on graphs with 1000s of intersections when seconds matter.

Objective: Minimize distance and nodes traversed for the fastest route.

Applications:



Navigation


Logistics


Gaming

Pre-existing Algorithms

Uni-directional Dijkstra (Edsger Dijkstra, 1959):

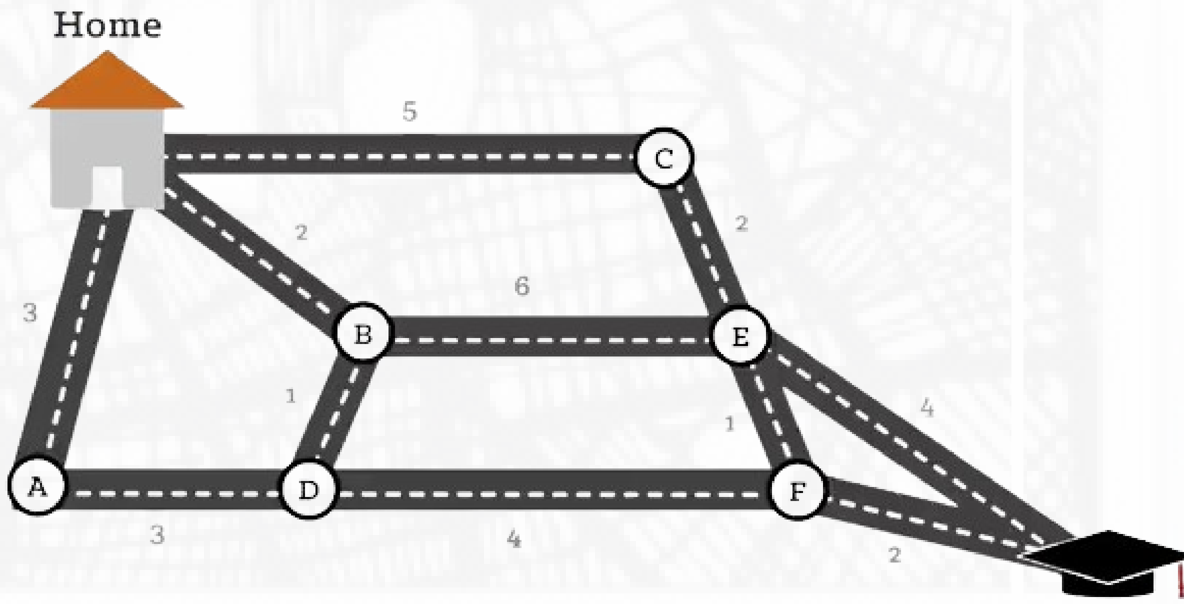
- **Method:** Explores all nodes from source(A) with a priority queue, picking the closest unvisited node.
- **Pros:** Reliable, works for positive weights
- **Cons:** Slow, checking even jammed roads.
- **Complexity:** O((V+E)logV).

A (Hart et al., 1968)*:

- **Method:** Uses heuristics (e.g., straight-line distance to destination) to prioritize routes.
- **Pros:** Slightly faster than uni-directional dijkstra, focuses on likely paths.
- **Cons:** Heuristics falter in Karachi’s unpredictable traffic (roadblocks, jams).
- **Complexity:** O((V+E)logV) with good heuristics.

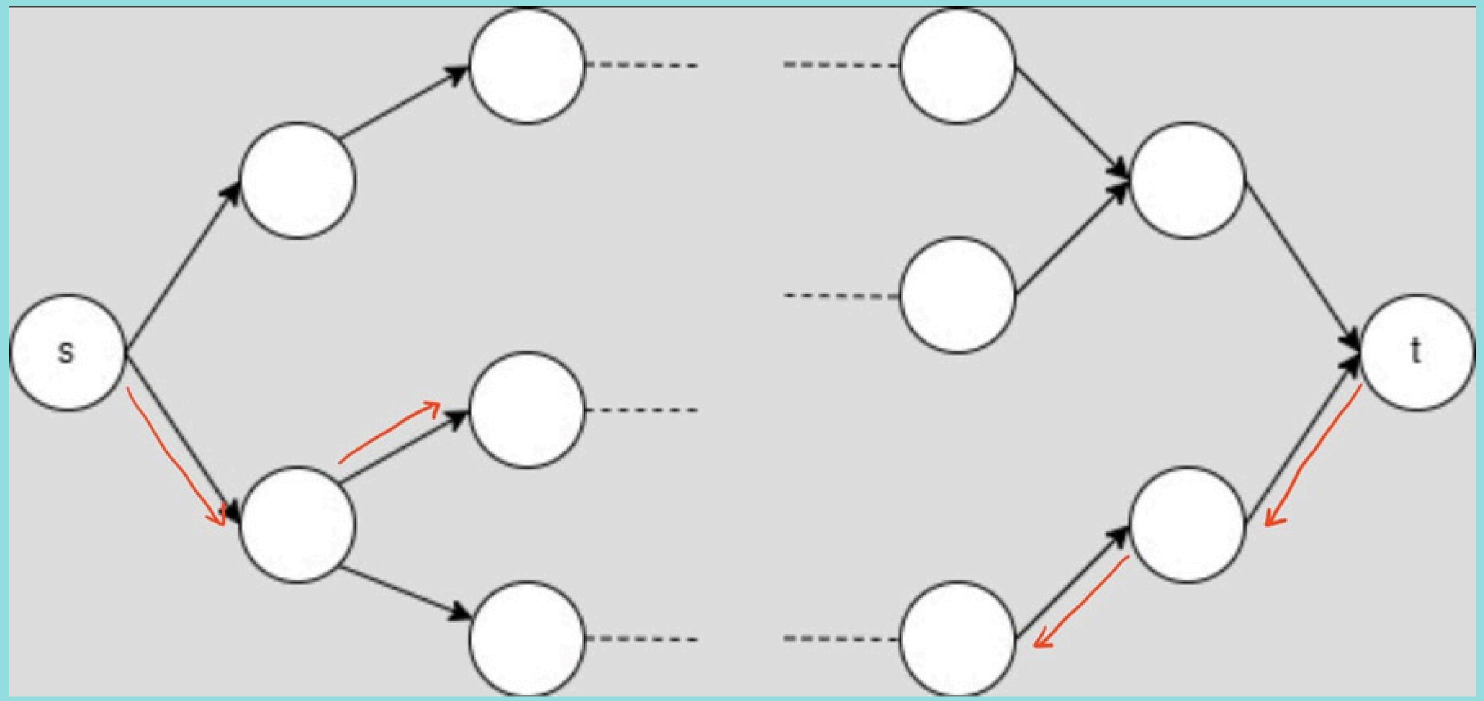
Bellman-Ford (1950s):

- **Method**: Iterates all edges repeatedly to find the shortest path.
- **Pros:** Handles negative weights (rare in routing).
- **Cons:** Very slow , O(VE), like manually checking every Karachi street.



Best option: bidirectional Dijkstra

- Runs two Dijkstra's at the same time
- Works on all types of graphs
- Directed and Undirected graphs with *positive* weights: O((V+E) log V)
- Unweighted graphs: O(Δ) - where Δ = max degree of G
- Explores roughly √n nodes in ideal setting
- Instance-optimal



What makes it different?

- Traverses lesser nodes in general
- One execution from *s* and one from *t*
- Stops when the two searches meet
- How do they stop?
- Multiple stopping conditions proposed
- Best method: $\hat{d}(s, u_s) + \hat{d}(u_t, t) > \mu$

where, $$\mu = \min_{v \in S_f \cap S_b} (d_f(v) + d_b(v))$$

Instance Optimality in light of Correctness

CORRECTNESS

- How do we ensure correctness?
- Use of Dijkstra - an algorithm that already exists
- *Valid* path found from *s* to *t*
- Does not stop too early, and miss a shorter path

Proof:

- Forward search from *s* to all reachable nodes
- Backward search from *t* does the same
- Each direction relaxes edges: recall Dijkstra’s

Instance Optimality in light of Correctness

OPTIMALITY

- Instance-optimality: for any given input, no other correct algorithm can use fewer edge queries by more than a constant factor
- Sublinear query model: access graph through basic operations - getting a node's neighbor

Proof:

- Let A be some algorithm, explore fewer edges than Bidirectional Dijkstra
- *G'* and *G* two graphs, differing only in few edges that A does **not** look at
- Same answer on both - contradiction

Limitations

1- Positive Weights Requirement:

- Only works with positive weights (e.g., road distances: A→B = 2, C→E = 4).
- Fails with negative weights (e.g., profit-loss models like fuel savings).

2- Memory Usage for Large Graphs:

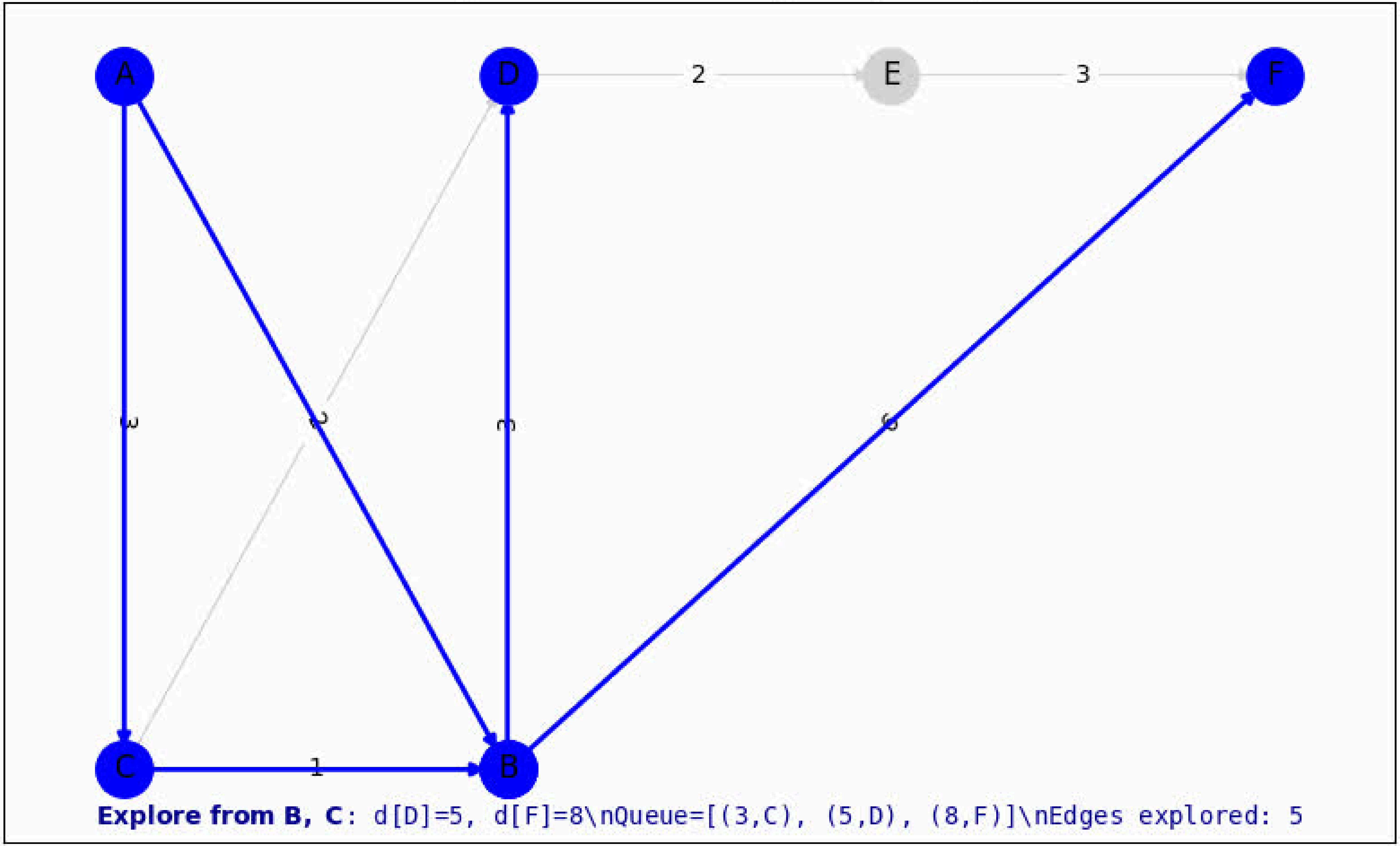
- Two searches (A forward, F backward) = 2 priority queues.
- Doubles memory vs. unidirectional Dijkstra.

3- Path Reconstruction Complexity:

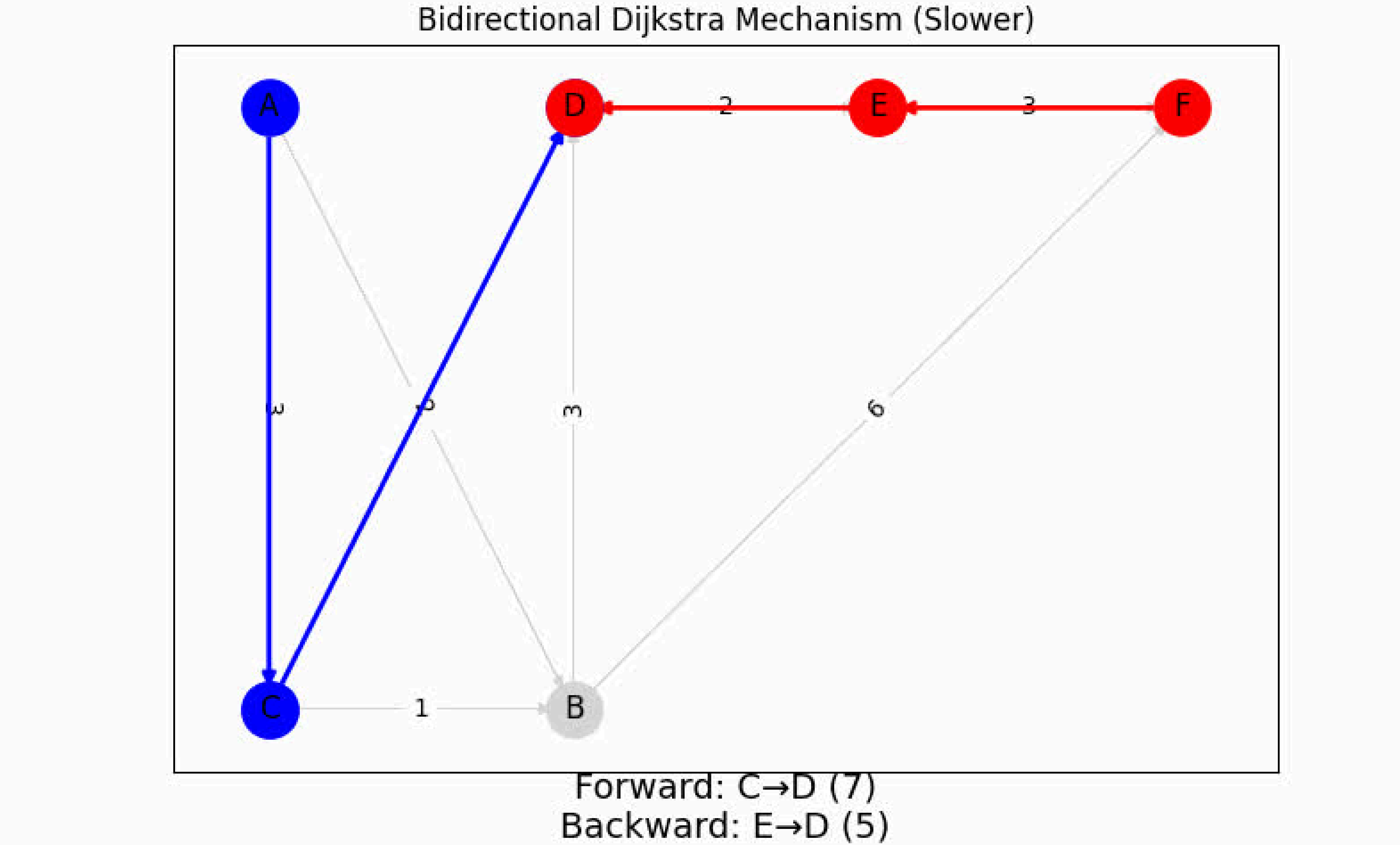
- Merging forward and backward paths is complex.
- Process: Trace forward path to source, reverse it, trace backward path to target, merge at meeting node.
- Challenge: Picking the best meeting node and merging without duplication; harder with multiple meeting points

Working of Algorithm (Uni-directional dijkstra)

Step 2: Unidirectional Dijkstra (A to F)



Working of Algorithm
Bi-directional Dijkstra


Bidirectional Dijkstra Mechanism (Slower)
A
D
E
F
C
B
2
3
3
3
6
1
Forward: C→D (7)
Backward: E→D (5)

THANK YOU!

Do you have any questions?