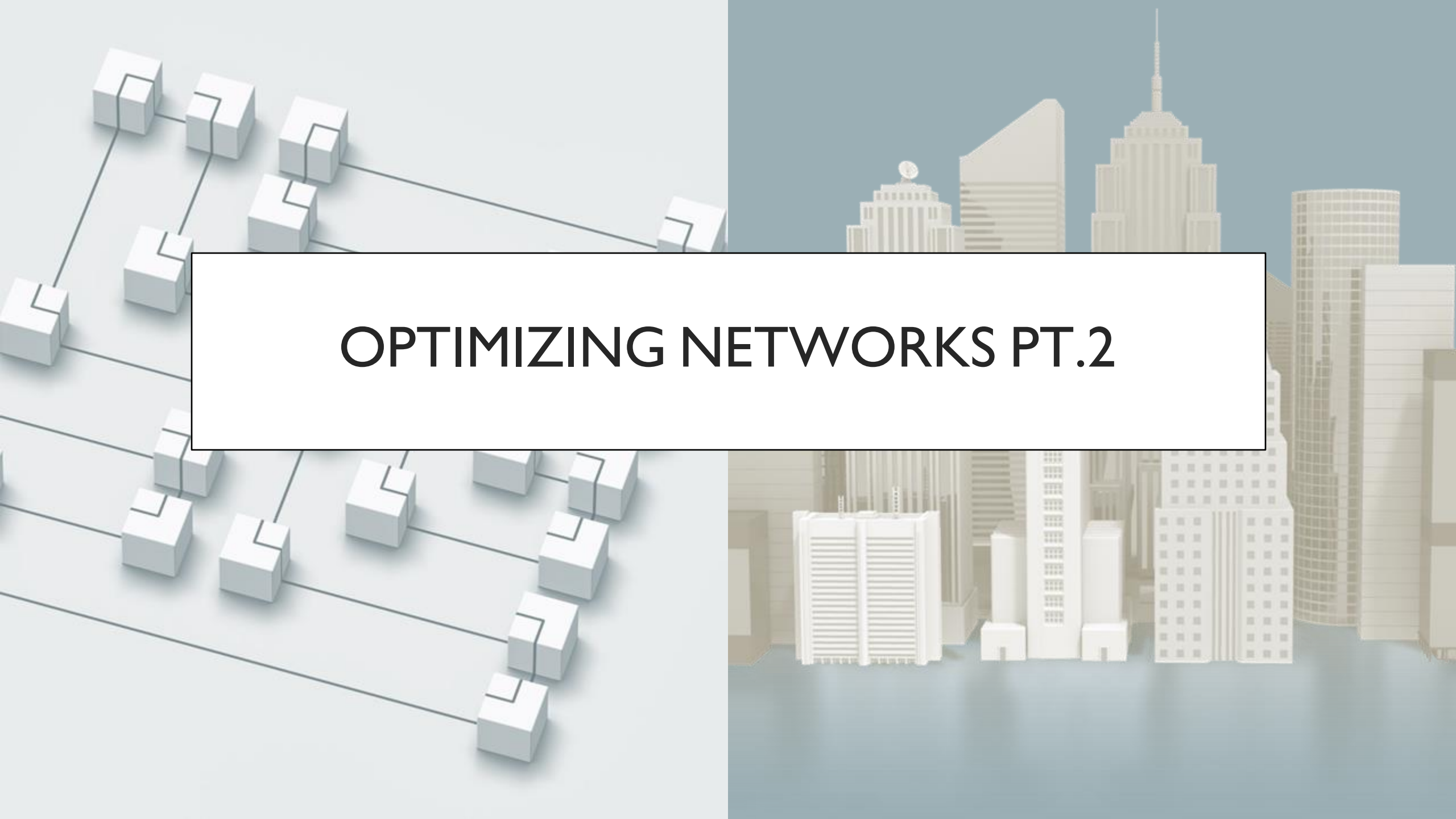


OPTIMIZING NETWORKS PT.2

RECAP OF FIRST PRESENTATION

Networks and graphs are fundamental mathematical structures used to model relationships and connections between entities.

Basic Concepts:

• Nodes (vertices) and edges: Nodes represent entities, and edges represent relationships or connections between them.
• Types of graphs: Simple, weighted/unweighted, directed/undirected, cyclic/acyclic, connected/disconnected.

• Common Greedy algorithms: Shortest Path algorithms (e.g., Dijkstra's algorithm), Minimum Spanning Trees (e.g., Prim's and Kruskal's algorithms),
• Common Non-Greedy algorithms: Graph Traversal (Depth-First and Breath-First search).
• Minimum Spanning Trees: Subset of edges that connect all vertices together with minimum total edge weight.

Network Applications in Nature:

• Ant Colony Optimization and Physarum Polycephalum slime mold

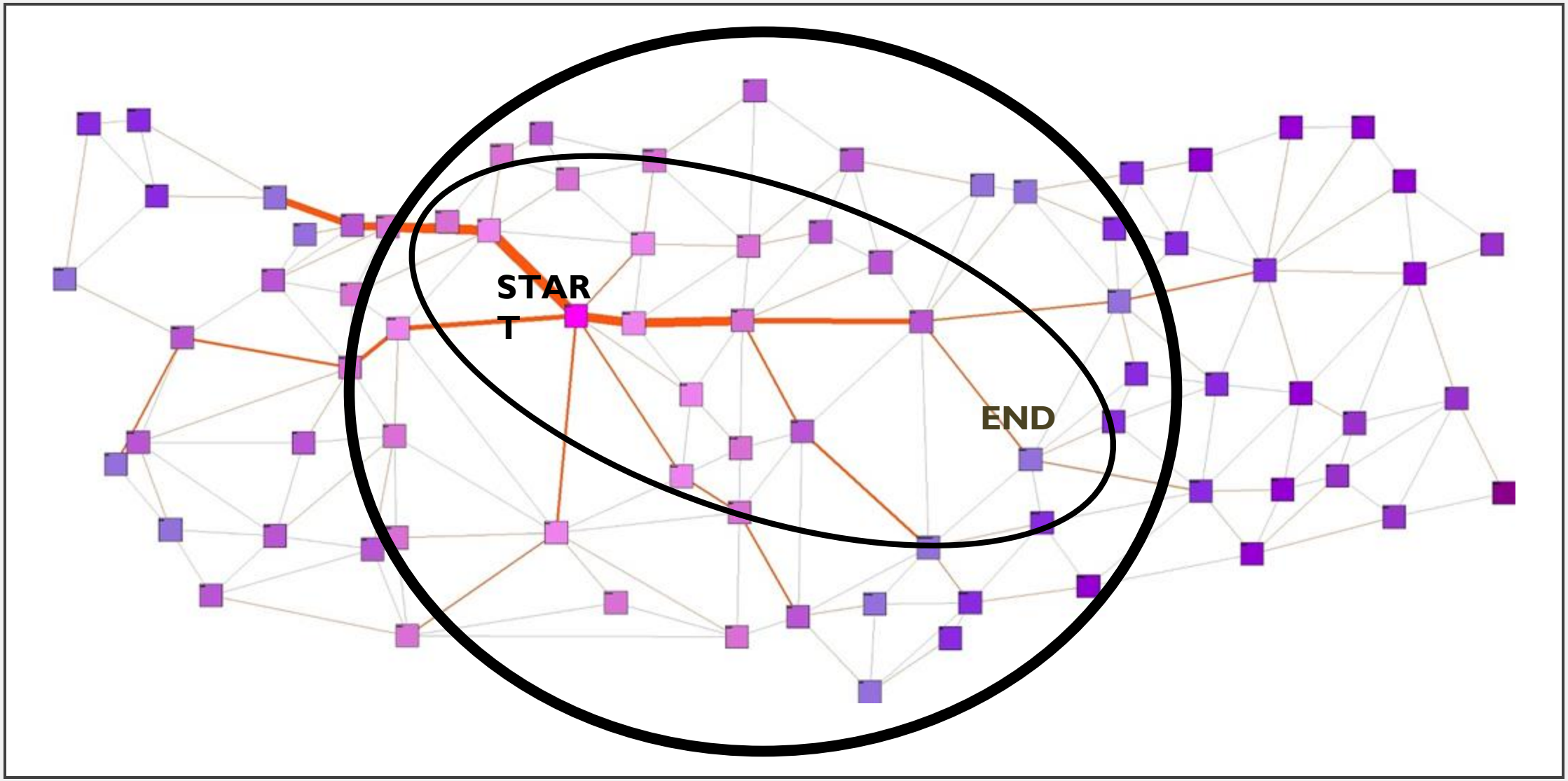
Future Advancements/Challenges:
• Integration with smart city technologies.
• Advancements in machine learning and AI.
• Consideration of environmental factors and multi-modal transportation systems.

DIJKSTRA'S ALGORITHM

Dijkstra Steps:

1) Mark all nodes as unvisited
2) Assume all nodes have a tentative value of infinity before visiting
3) Determine the current node
4) At the current node, compare the distance to any unvisited neighbor nodes
5) Update the shortest distance if applicable
6) Mark current node as visited
7) Choose the next current node as unvisited node with shortest distance

DIJKSTRA'S ALGORITHM A* ALGORITHM



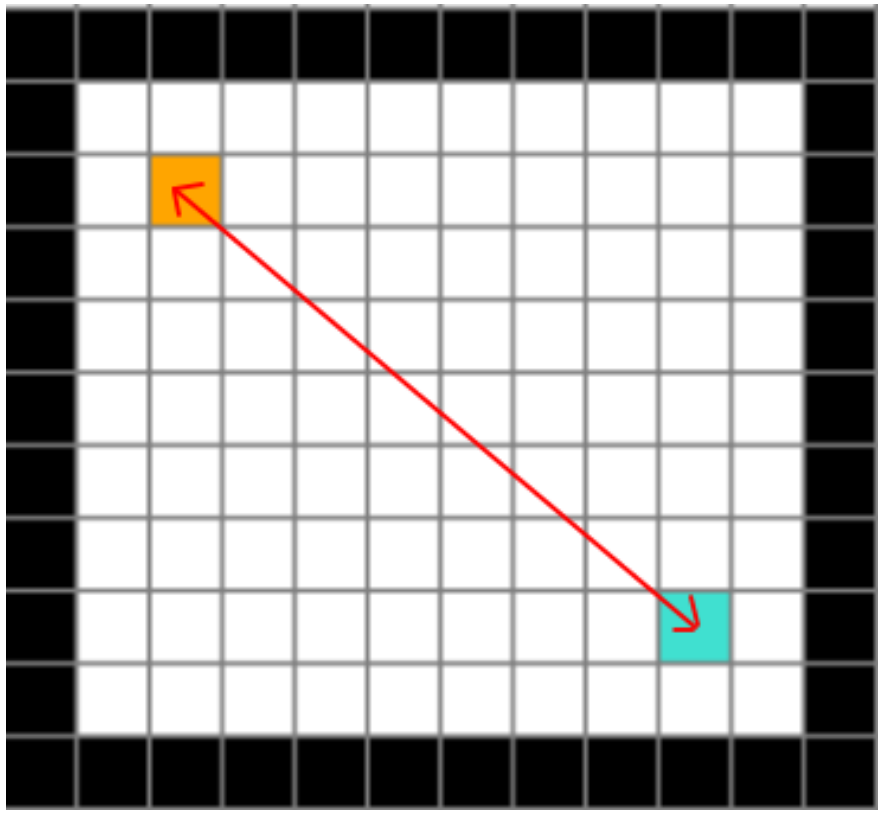
A* ALGORITHM

A* score = Cost of Path + Heuristic

A heuristic is a method used to solve a problem more quickly.

A* score [f(n)] = g(n) + h(n)
Where f(n) is final cost, g(n) is distance between nodes, and h(n) being a heuristic estimate for a nodes value.

We can calculate the heuristic exactly or make an approximation:



$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Approximating Heuristic Techniques

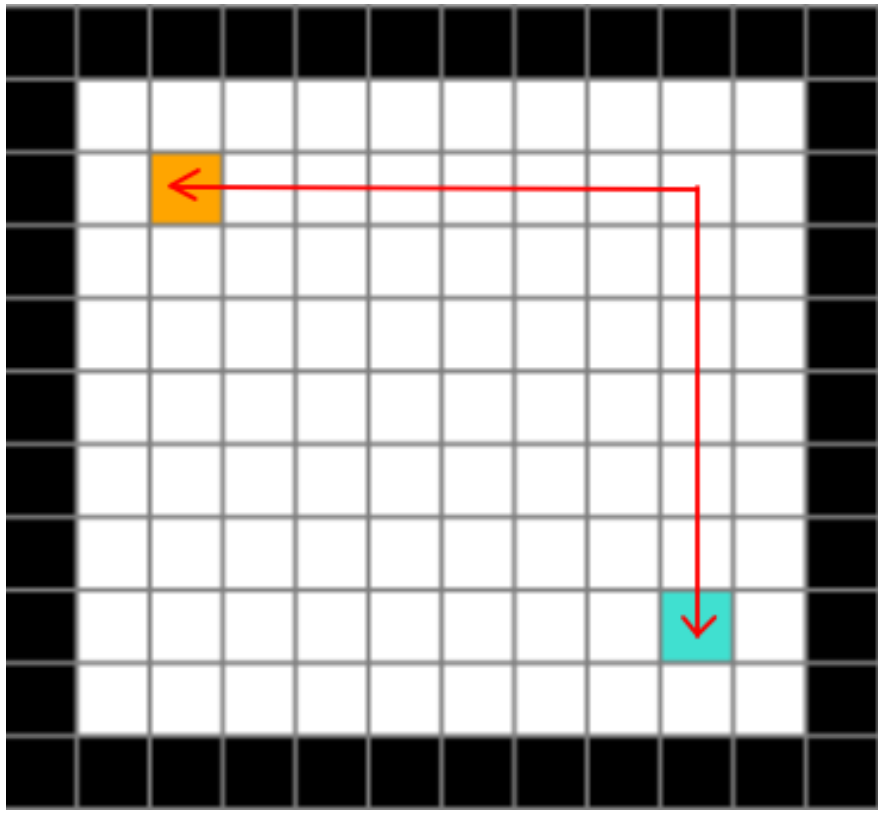
Euclidean Distance:

-Calculates Distance between current node and goal node as a straight line

-This method can be used when trying to move in any direction.

This method can be utilized assuming every node has some (x,y) coordinate. An advantage of this path is that it will never overestimate the optimal route. Essentially, the travel distance to your destination will never be shorter than the given line.

APPROXIMATING HEURISTIC TECHNIQUES



$$Mdist = |x2 - x1| + |y2 - y1|$$

Manhattan Distance:

-This method calculates the sum of absolute value differences between the current node's (x,y) coordinates and the goal nodes (x,y) coordinates.

-This method can be utilized when moving in one of the 4 cardinal directions (up, down, left, right)

A* STEPS

1. **Initialization**:

- Mark all nodes as unvisited.
- Assign a tentative distance value to each node, typically set to infinity.
- Set the initial node as the current node and its tentative distance value to 0.

2. Explore Neighbors:

-For each unvisited neighbor of the current node:

- Calculate the tentative distance from the start node to the neighbor node.
- Update the tentative distance if it's lower than the previous value.
- Calculate the heuristic estimate (typically the Euclidean distance) from the neighbor node to the goal node.
- Calculate the total estimated distance (tentative distance + heuristic) for the neighbor node.
- Update the neighbor node's tentative distance and total estimated distance.

3. Select Next Node:

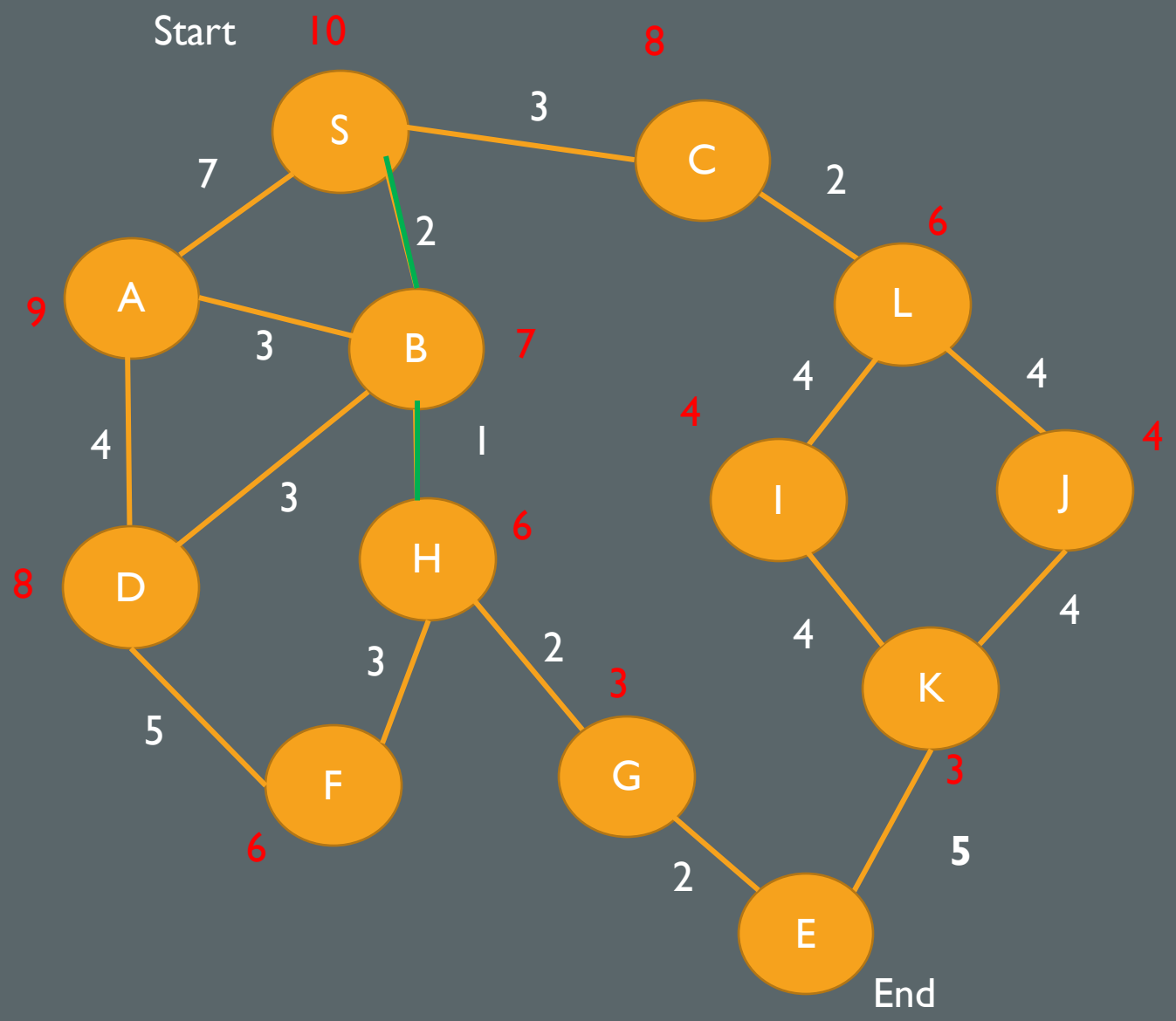
- Mark the current node as visited.
- Choose the unvisited node with the lowest total estimated distance as the next current node.

4. Termination:

- Continue until the goal node is reached or there are no more unvisited nodes.

5. Path Reconstruction:

- If the goal node is reached, reconstruct the shortest path from the start node to the goal node using the recorded tentative distances.



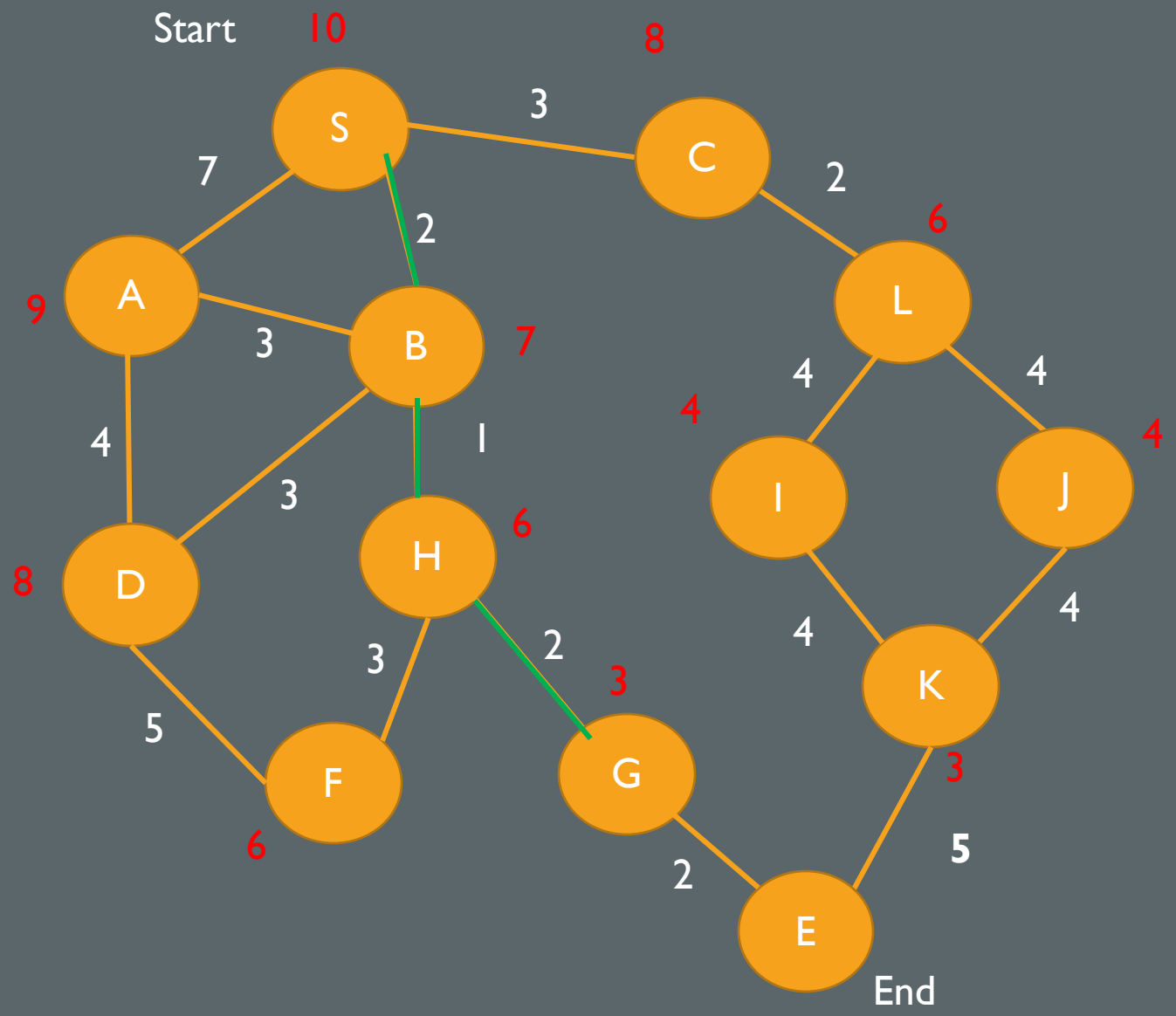
S → B →

A* ALGORITHM

S 0 10 -

Node	Path Cost	Heuristic Path	Prior Node
A B	7 2	9 6	S S
G H	3 3	1 8	E S
A A	5 7	1 3	B B
A D	5 5	1 3	B B
A	5	14	B

A* ALGORITHM

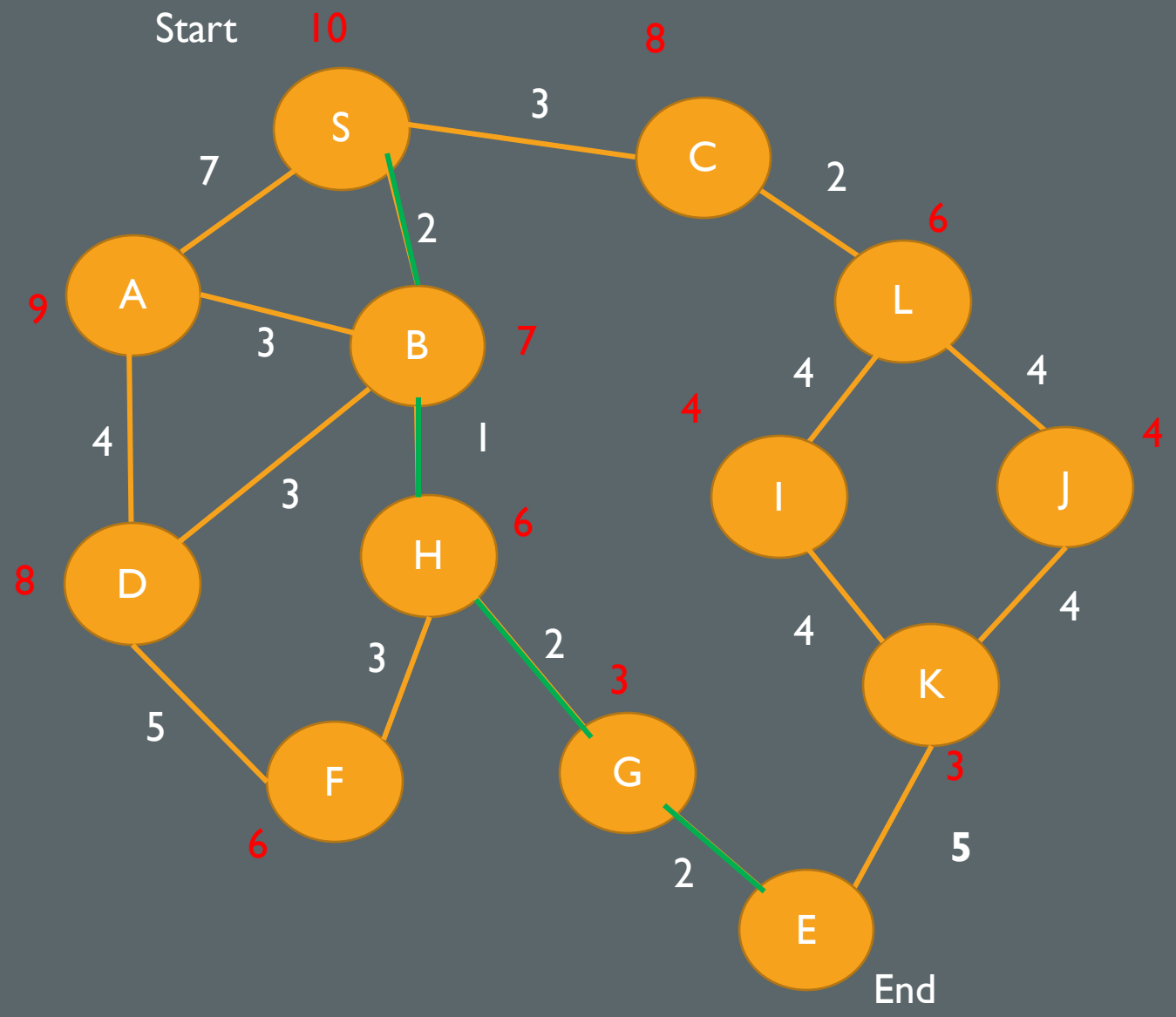


S → B → H

S 0 10 -

Node	Path Cost	Heuristic Path	Prior Node
H	3	9	B
G	5	8	HS
D	6	12	H
A	6	12	B
D	5	13	B
A	5	14	B

A* ALGORITHM



S → B → H → G → E

S 0 10 -

Node	Path Cost	Heuristic Path	Prior Node
G	5	8	H
E	7	7	G
F	6	12	H
D	6	12	H
A	5	13	B
A	5	4	B

A* ALGORITHM BEHAVIOR

How different heuristic estimates affect A*:

If h(n)=0,
- A* score, f(n),= g(n).
-The algorithm will essentially only follow Dijkstra's Algorithm.

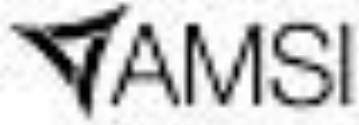
If h(n) < the cost from the current node to the goal node:
-A* is guaranteed to find a shortest route.

If h(n) is exactly equal to the cost from the current node to the goal node:
-A* only follows the best route, never expanding, hence very fast.

If h(n) > the cost from the current node to the goal node, A* is not guaranteed to find a shortest path, but it can still run faster

MATHS DELIVERS

AIRLINE SCHEDULING



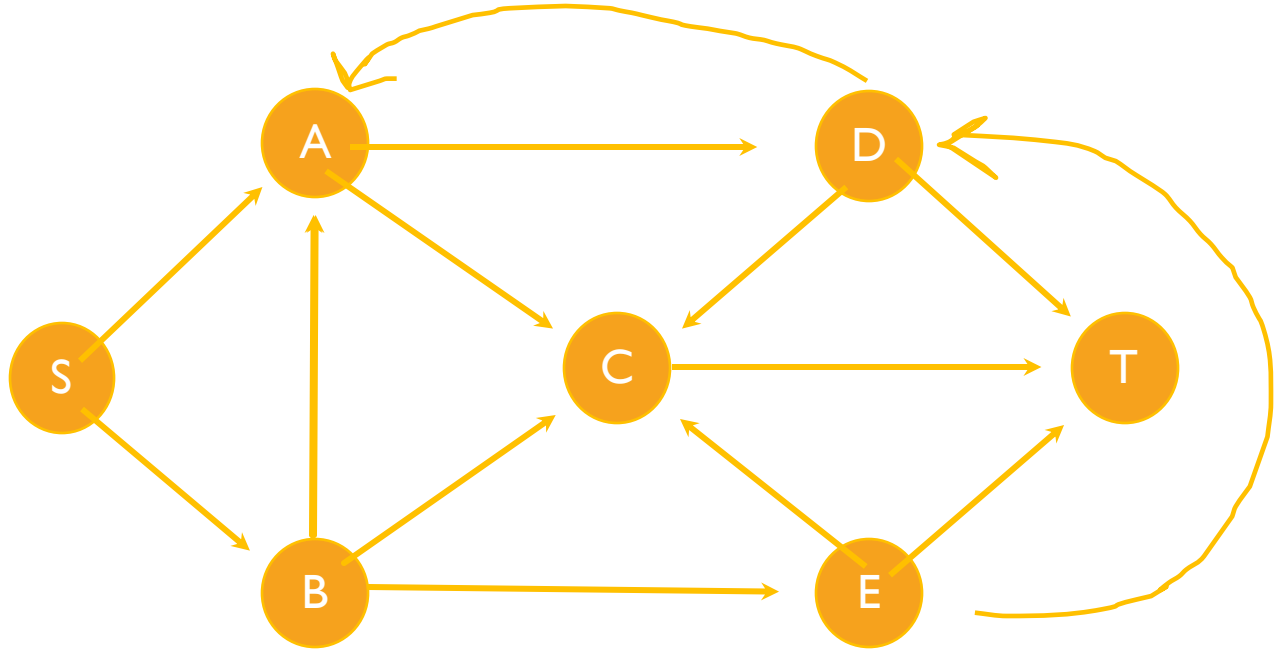
Operations Research

Operations Research (OR) is a discipline that deals with the application of advanced analytical methods to help make better decisions. It employs mathematical modeling, statistical analysis, and optimization techniques to solve complex problems in various domains, including logistics, supply chain management, finance, healthcare, and transportation, among others.

1. **Transportation Networks**: Network algorithms, such as shortest path algorithms (like Dijkstra's algorithm) and minimum spanning tree algorithms, are used to find the most efficient routes, minimizing costs or travel time.
2. **Supply Chain Networks**: Network optimization techniques help in determining the optimal locations for warehouses and distribution centers, as well as the most efficient routes for transporting goods between them.
3. **Facility Location and Network Design**: Operations Research helps in determining the optimal locations for facilities, such as factories, hospitals, or schools, considering factors like demand, transportation costs, and facility capacities. Network algorithms aid in designing the layout of the network to minimize overall costs or maximize service coverage.

Maximum Flow

(Some basics)



Let g(v) denote edges for vertices, v.

Let g^{+}(v) denote outgoing edges that start at vertices, v, and end elsewhere.

Let g^{-}(v) denote incoming edges that start elsewhere and end at vertices, v.

EX: g^{+}(C)={(C,T)}
g^{-}(C)={(A,C),(B,C), (D ,C), (E ,C)}

Maximum Flow

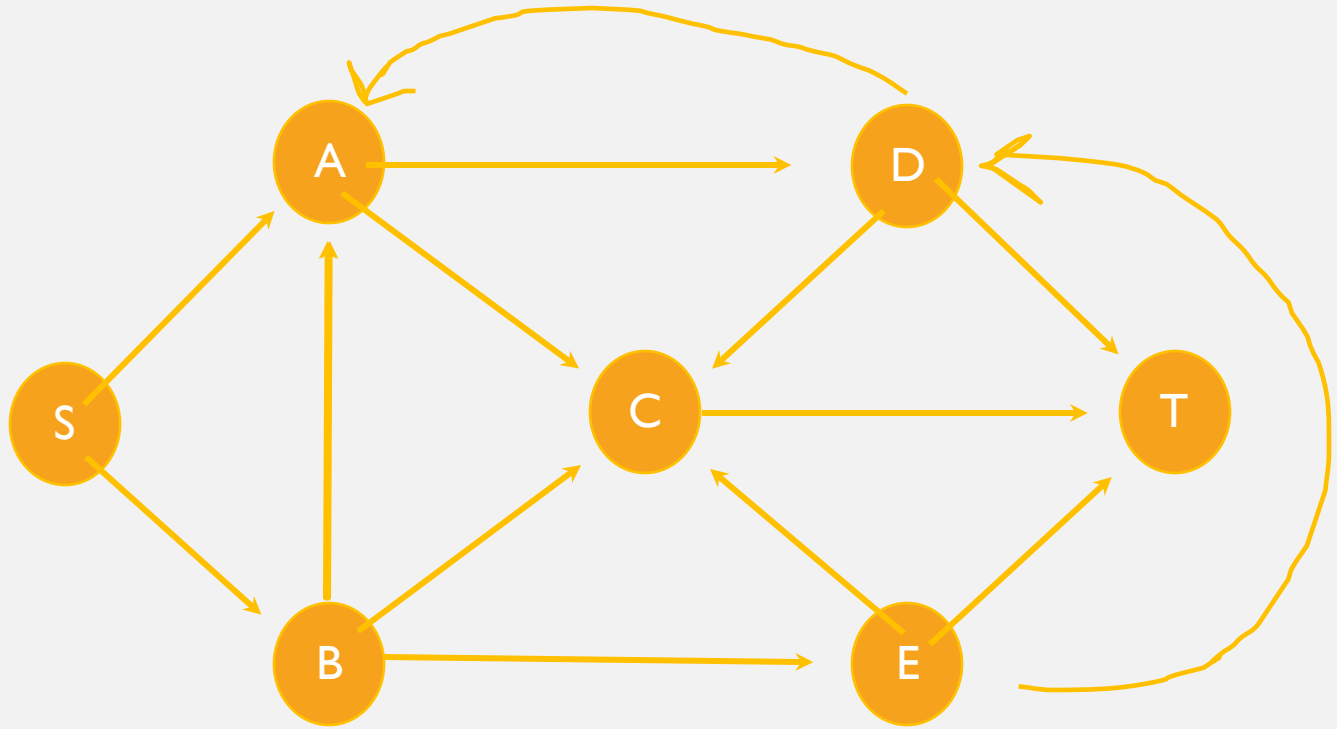
Flow on a directed graph must satisfy the following requirements:

1) An edges flow can not be greater than its capacity

$$f(e) \leq c(e) \text{ for all } e \in E$$

2) The amount of flow entering a node must equal the amount of flow exiting that node (barring the source and sink)

$$\sum_{e \in g^{-}(v)} f(e) = \sum_{e \in g^{+}(v)} f(e)$$



The value of the flow, val (f), is the net flow leaving the source node or entering the sink node.

$$\sum_{e \in g^{+}(S)} f(e) - \sum_{e \in g^{-}(S)} f(e) = \sum_{e \in g^{-}(T)} f(e) - \sum_{e \in g^{+}(T)} f(e)$$

Maximum Flow

An augmented path , P, is a path from the source to the sink node on an undirected version of the graph such that:

1) Every edge traversed in the forward direction by P has a flow less than that edges capacity.

$$f(e) < c(e)$$

2) Every edge traversed in the backward direction has a flow greater than zero:

$$f(e) > 0$$

Ford-Fulkerson algorithm for Maximum-Flow:

Given a graph, G = (V, E) with edge capacities $c : E \to \mathbb{Z}+$ and node $s,t \in V$

1.Set flows f(e) = 0 for all edges e ∈ E.
2.While there exists some f-augmenting path P,

a) Assign Γ = min {c(e) - f(e) if P traverses e forwards
f(e) if P traverses e backward}

b) For each edge e ∈ P,
f(e) <- { f (e) + Γ if P traverses e forwards
f(e) - Γ if P traverses e backwards}

- A flow f can only be considered a maximum if there is no augmenting path remaining in the network.

Maximum Flow

An augmented path , P, is a path from the source to the sink node on an undirected version of the graph such that:

1) Every edge traversed in the forward direction by P has a flow less than that edges capacity.

$$f(e) < c(e)$$

2) Every edge traversed in the backward direction has a flow greater than zero:

$$f(e) > 0$$

Edmonds-Karp algorithm for Maximum-Flow.
**Just one modification

Given a graph G = (V, E) with edge capacities
c :E -> $\mathbb{Z}+$ and node s,t $\in$ V

1. Set flows f(e) = 0 for all edges e $\in$ E.
2. While there exists a shortest f-augmenting path P,

a) Assign Γ = min {c(e) - f(e) if P travels e forwards
f(e) if P travels e backward}

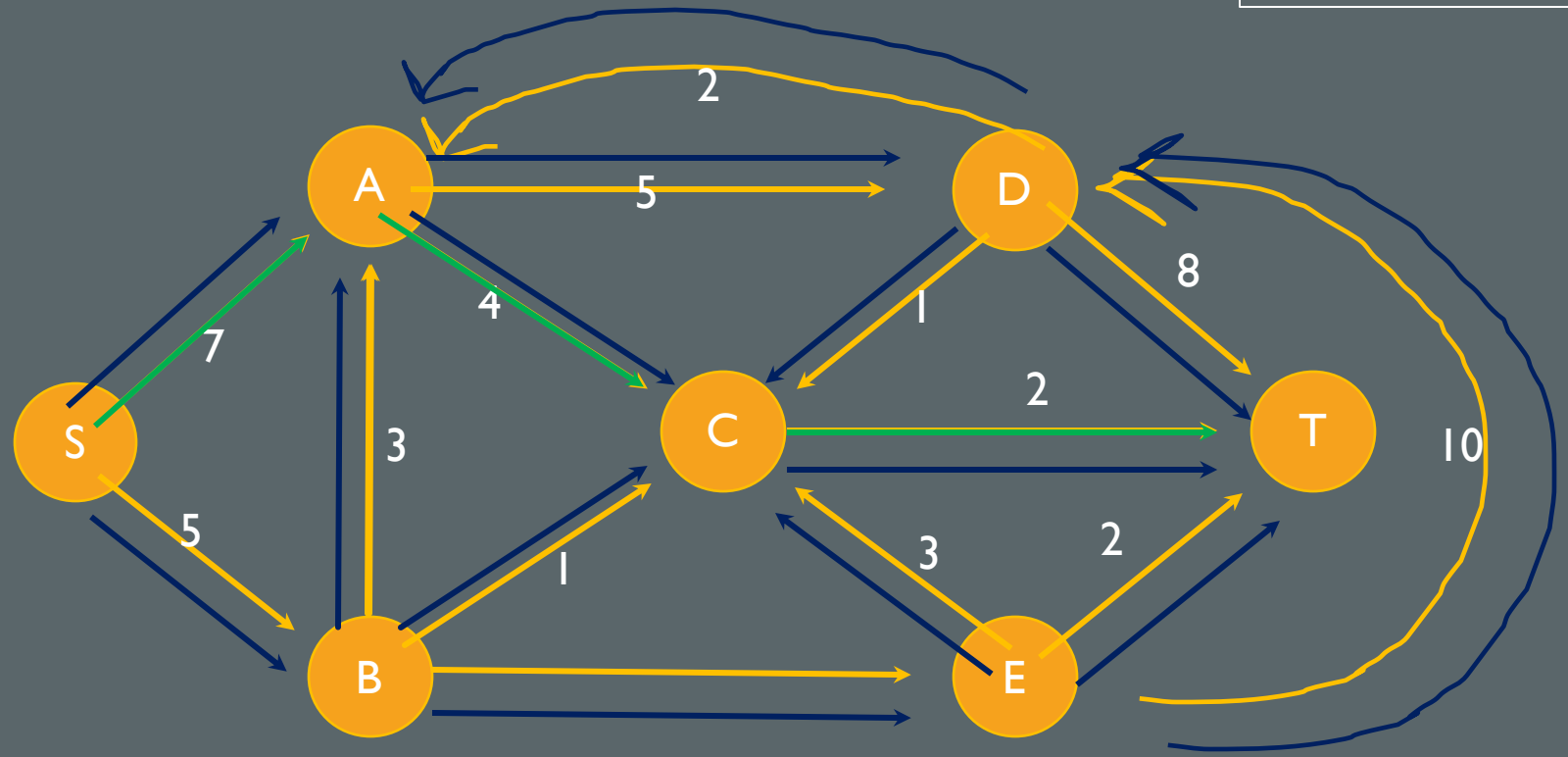
b) For each edge e $\in$ P,
f(e) <- { f (e) + Γ if P travels e forwards
f(e) - Γ if P travels e backwards}

The Edmonds-Karp algorithm will terminate at most (|V|*|E|)/2 augmentations (O(|V|*|E|^{2}) time).

EDMONDS-KARP ALGORITHM

We will consider the following example.
Initially, f(e) = 0 for all edges e ∈ E.

The highlighted edges are those that can be used by an f-augmenting path.
We use breadth-first search to find a shortest f-augmenting path.

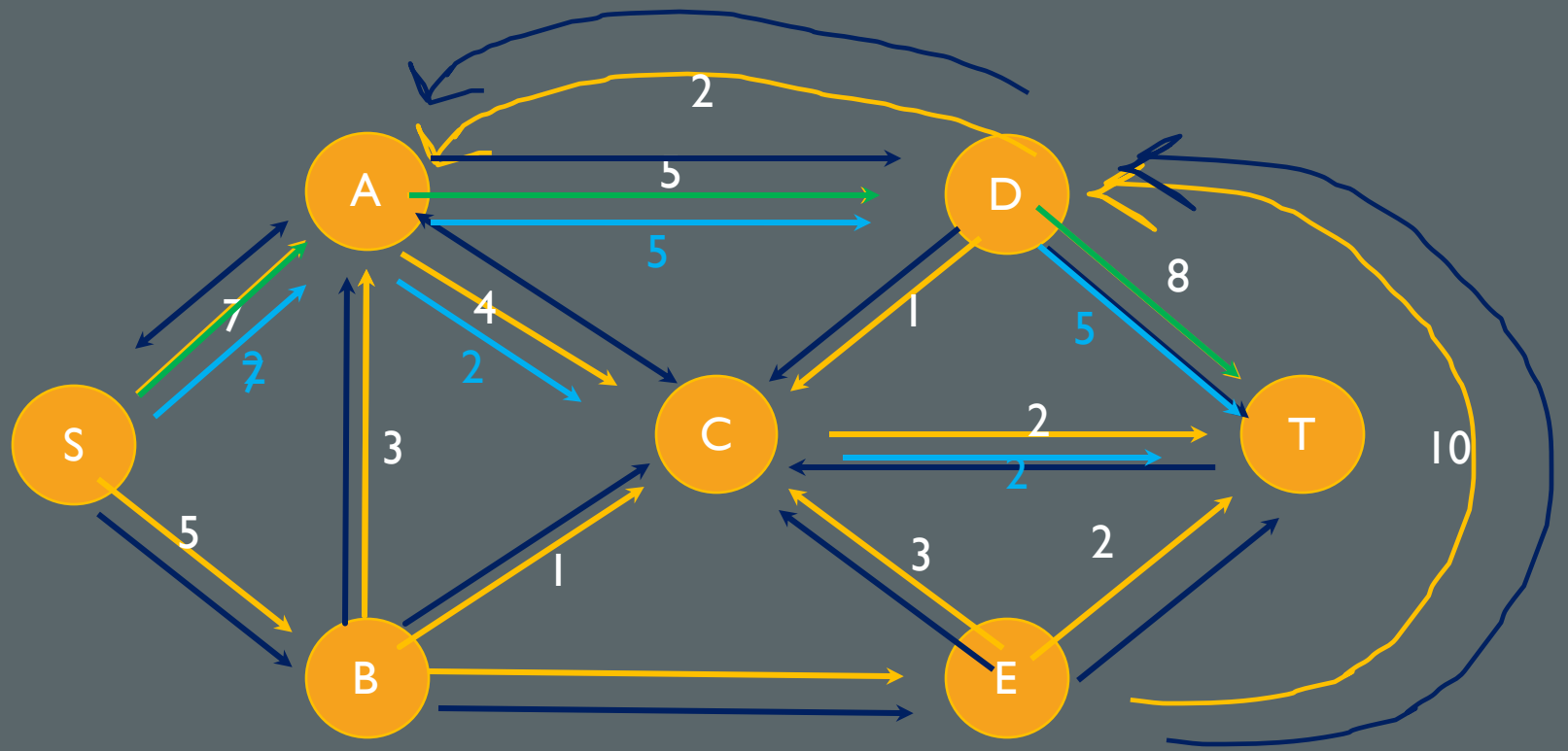


The path has the following augmented path:

Γ = min c(e) - f(e)
Γ = c((C,t)) – f((C,t))
=2 units of flow

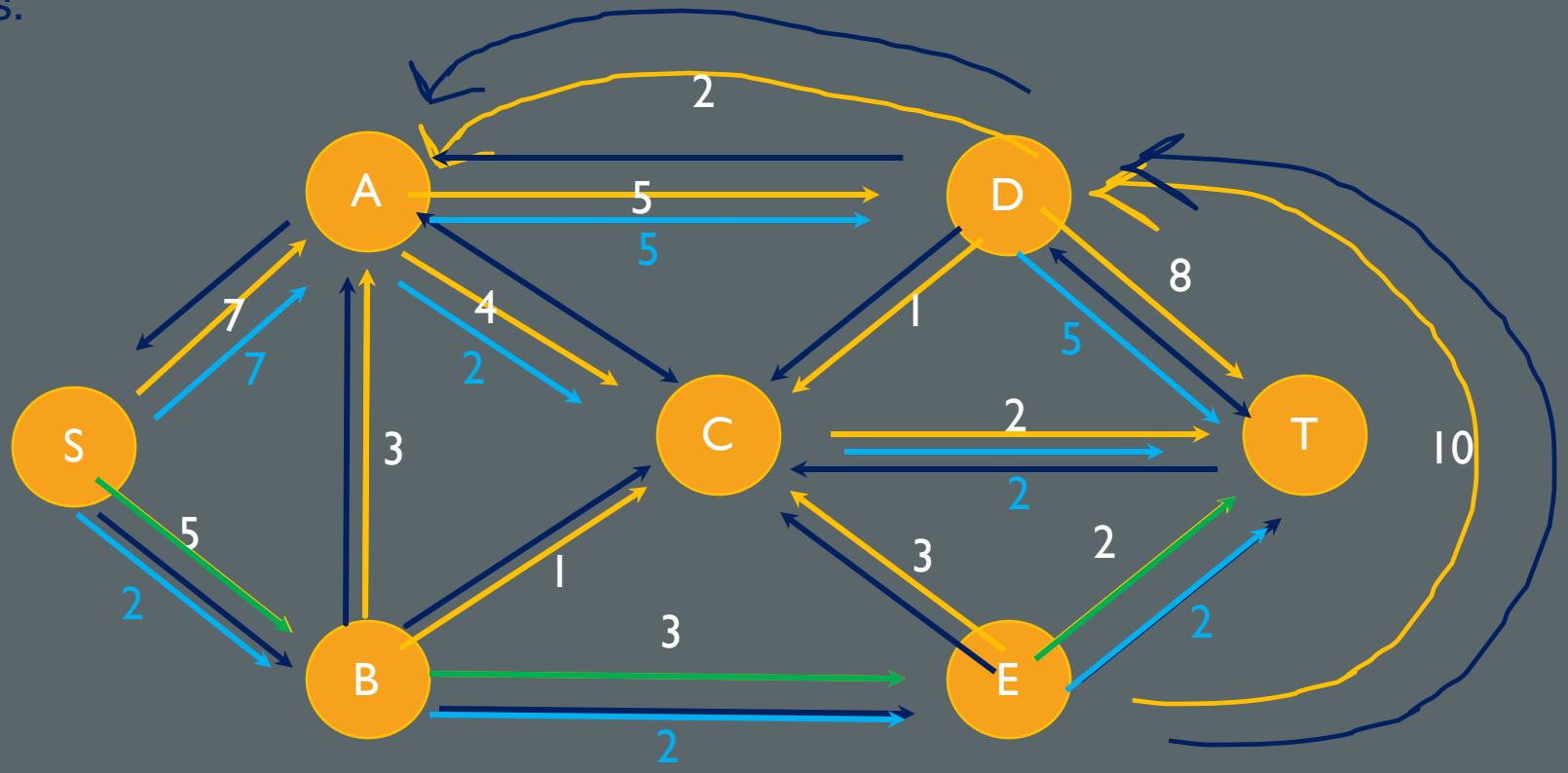
EDMONDS-KARP ALGORITHM

We again search for a shortest f-augmenting path among the highlighted edges.



EDMONDS-KARP ALGORITHM

We again search for a shortest f-augmenting path among the highlighted edges.

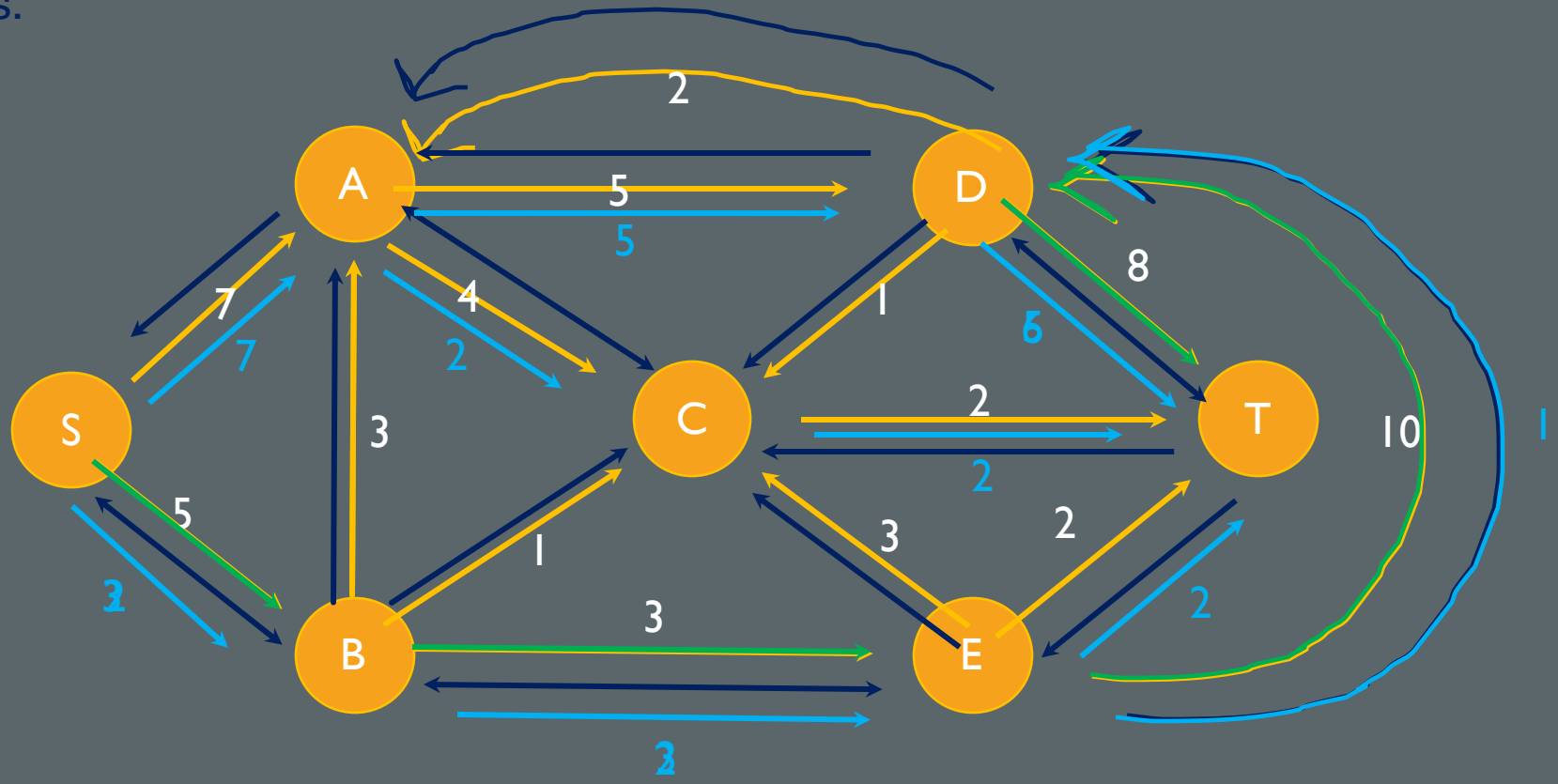


The path has the following augmented path:

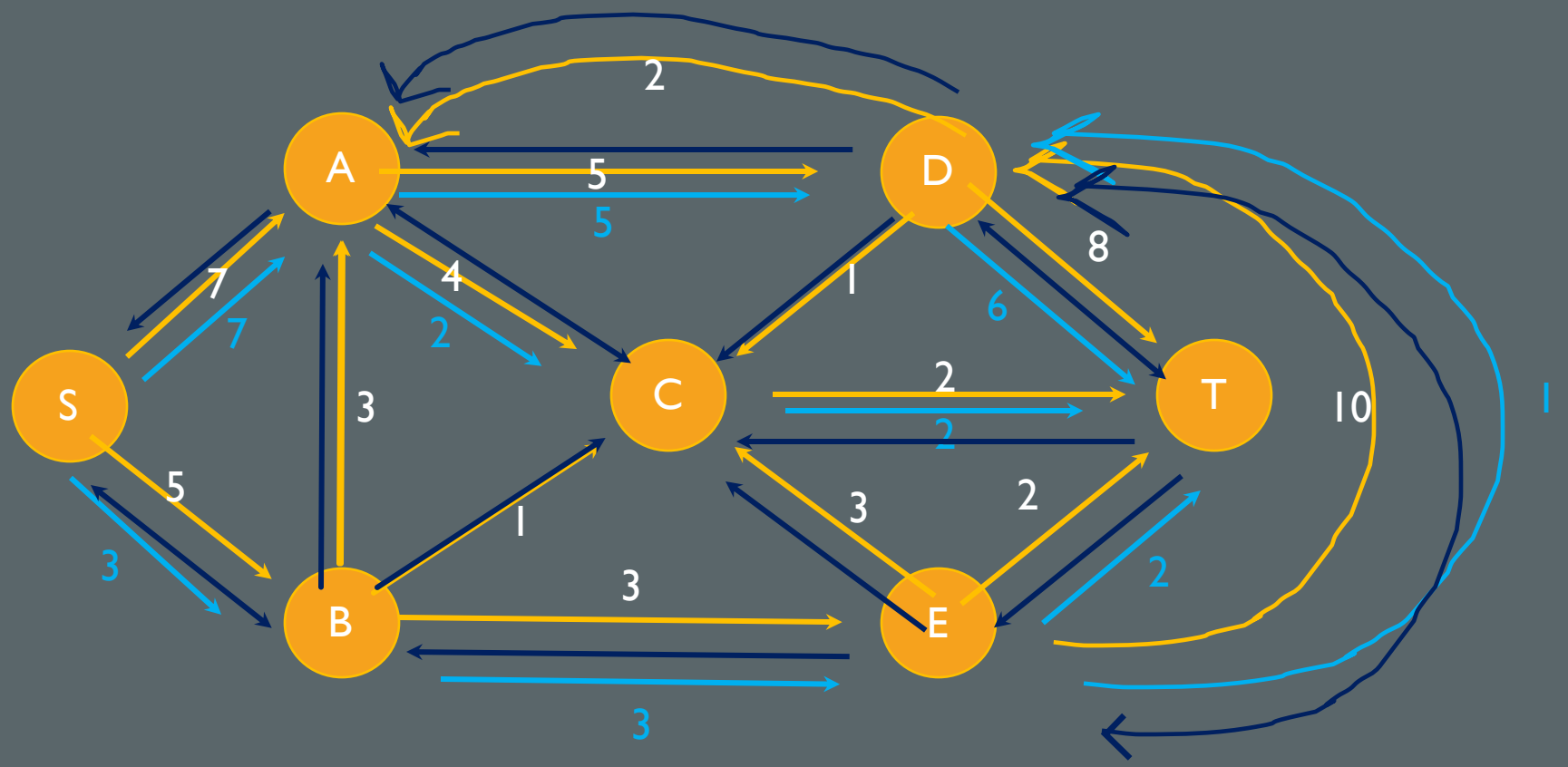
Γ = min c(e) - f(e)
Γ = c((E,t)) – f((E,t))
=2 units of flow

EDMONDS-KARP ALGORITHM

We again search for a shortest f-augmenting path among the highlighted edges.



EDMONDS-KARP ALGORITHM



We again search for a shortest f-augmenting path among the highlighted edges.

But there is no other augmentable path from the source to the sink, so the algorithm terminates

The value of the flow, val (f), is the net flow leaving the source node or entering the sink node.

$$\sum_{e \in g^{+}(S)} f(e) - \sum_{e \in g^{-}(S)} f(e) = \sum_{e \in g^{-}(T)} f(e) - \sum_{e \in g^{+}(T)} f(e)$$

The maximum (s, t)-flow, f, has value 10.

Edmonds-Karp Algorithm

Characteristics

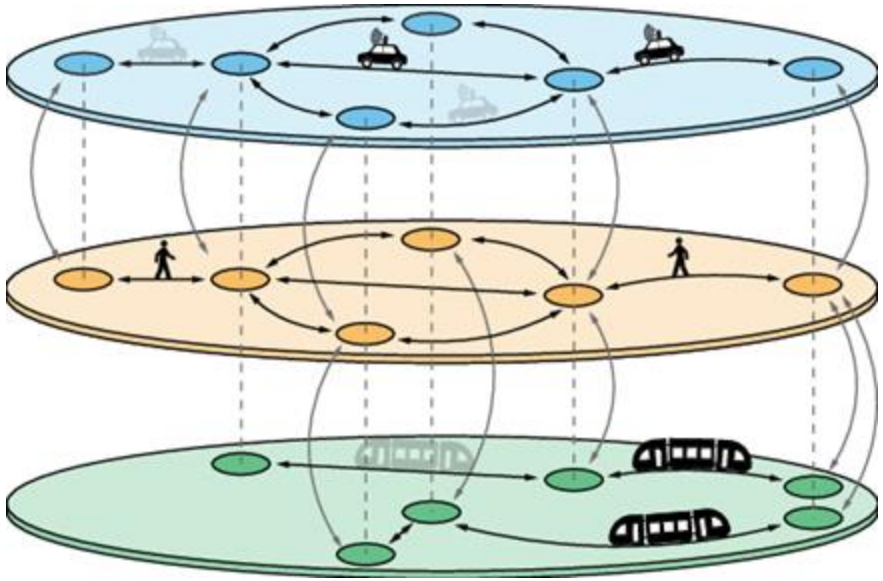
1. **Efficient:** Guaranteed time complexity of O(VE^2), where V is the number of vertices and E is the number of edges. Ford Fulkerson depends on how augmented paths are chosen. BFS always finds a shortest path
2. **Easy implementation:** The algorithm is relatively easy to understand and implement.
3. **Not optimal for certain edge configurations:** Only works with integer values.

EDMONDS-KARP ALGORITHM APPLICATIONS

- **Airline Scheduling**

We can utilize this algorithm considering the network of possible flight routes as a directed graph. Then each flight can be viewed as a directed edge where each edge's capacity is associated with the number of available seats per flight. Source and sink nodes will represent initial starting point and ultimate endpoint of the aircraft.

Algorithm is used to maximize number of passengers reaching the desired destination. Maximum flow algorithms are also used in conjunction with minimum cost algorithms to reduce costs for airlines.



EDMONDS-KARP ALGORITHM APPLICATIONS

Optimizing Transportation Networks

We can view transportation networks as a directed graph, where each vertices indicates a location, and each edge is associated with a road, railway, or some kind of path between two locations. An edges capacity is then representative of the maximum amount of goods/traffic that flow through this network per unit of time.

Other examples:

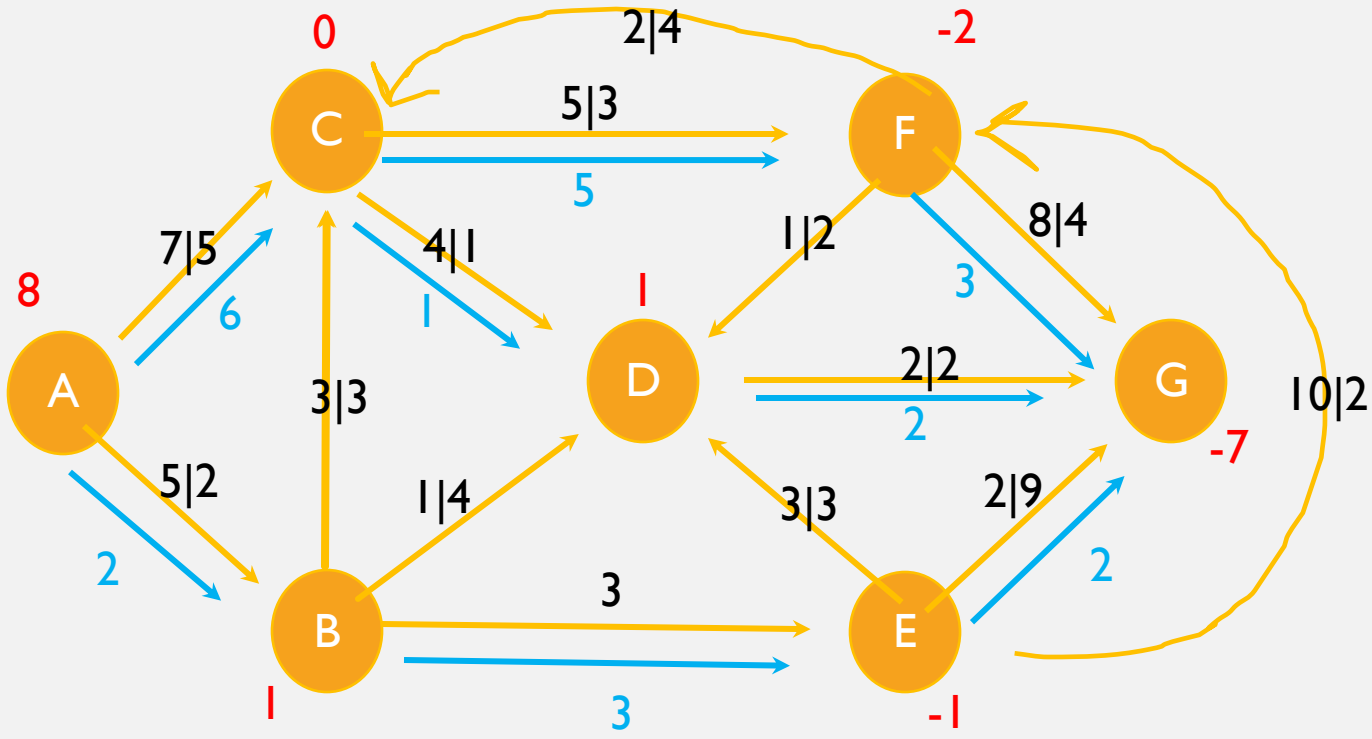
- **Energy Networks**
- **Supply Chain Management**

Minimum Cost Flow

The Minimum-Cost-Flow problem takes into account edge capacities as well as edge costs (may be shortest path) at the same time.
A b-flow on a directed graph G = (V,E) with edge capacities $c \in \mathbb{R}+$, and node balances b: V⟶ R with $\sum_{v \in V} b(v) = 0$ is a function f: E⟶ $\mathbb{R}_{v \geq 0}$ satisfying:

1) Capacity: An edge cannot carry more flow than its capacity.

$$f(e) \leq c(e) \quad \forall\ e \in E$$

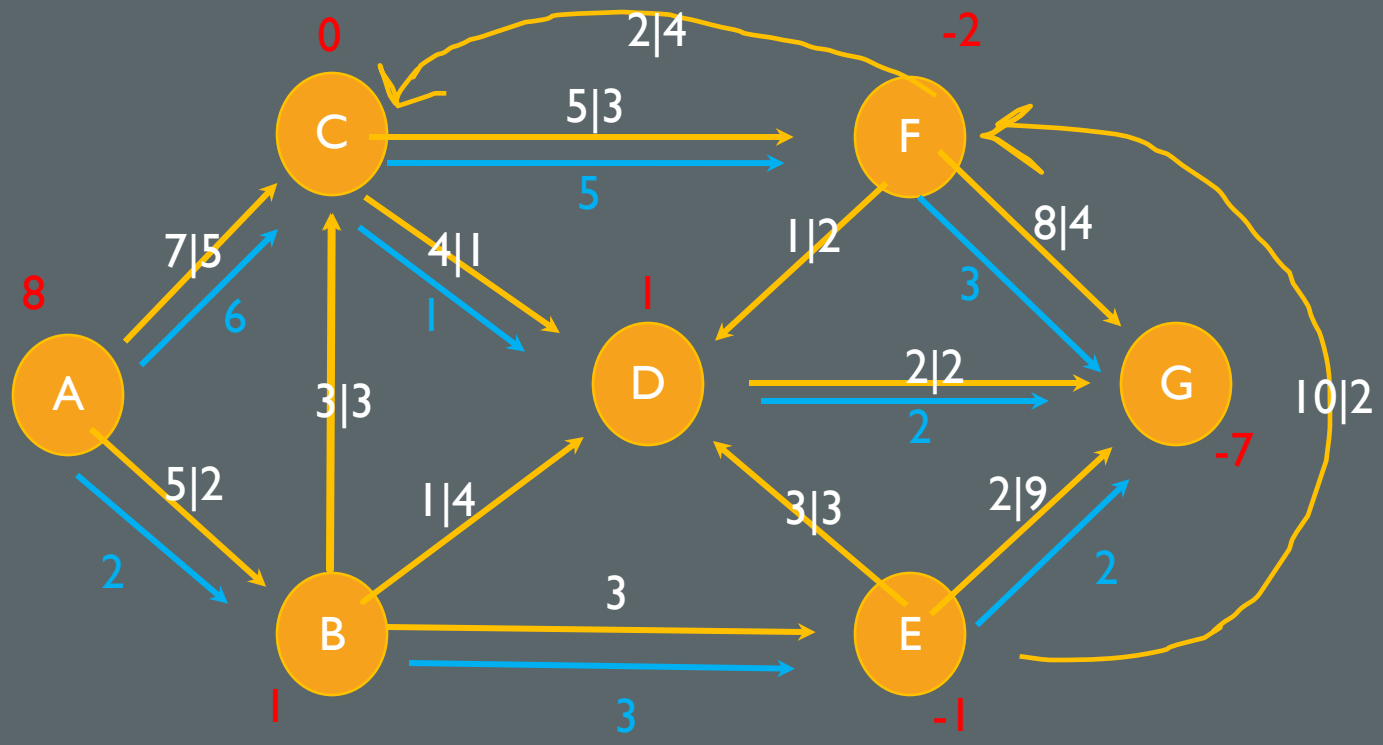


2) Balance: the difference in flow entering and leaving a node v must equal b(v).

$$\sum_{e \in g^{+}(V)} f(e) \quad - \quad \sum_{e \in g^{-}(V)} f(e) \quad = \quad b(v) \quad \forall\ v \in V$$

MINIMUM COST FLOW PROBLEM

Assume we're dealing with a directed graph G = (V,E) with edge capacities c: E→ $\mathbb{R}_+$, edge weights w: E→ $\mathbb{R}$, and node balances b: V→ $\mathbb{R}$ with $\sum_{v \in V} b(v) = 0$, find a b-flow f that minimizes $\sum_{e \in E} w(e)f(e)$.



Minimum Cost Flow

An augmented cycle, C, is a cycle on the undirected version of a graph, G such that:

1) Every edge that's crossed in the forward direction by C has a flow less than that edges capacity.

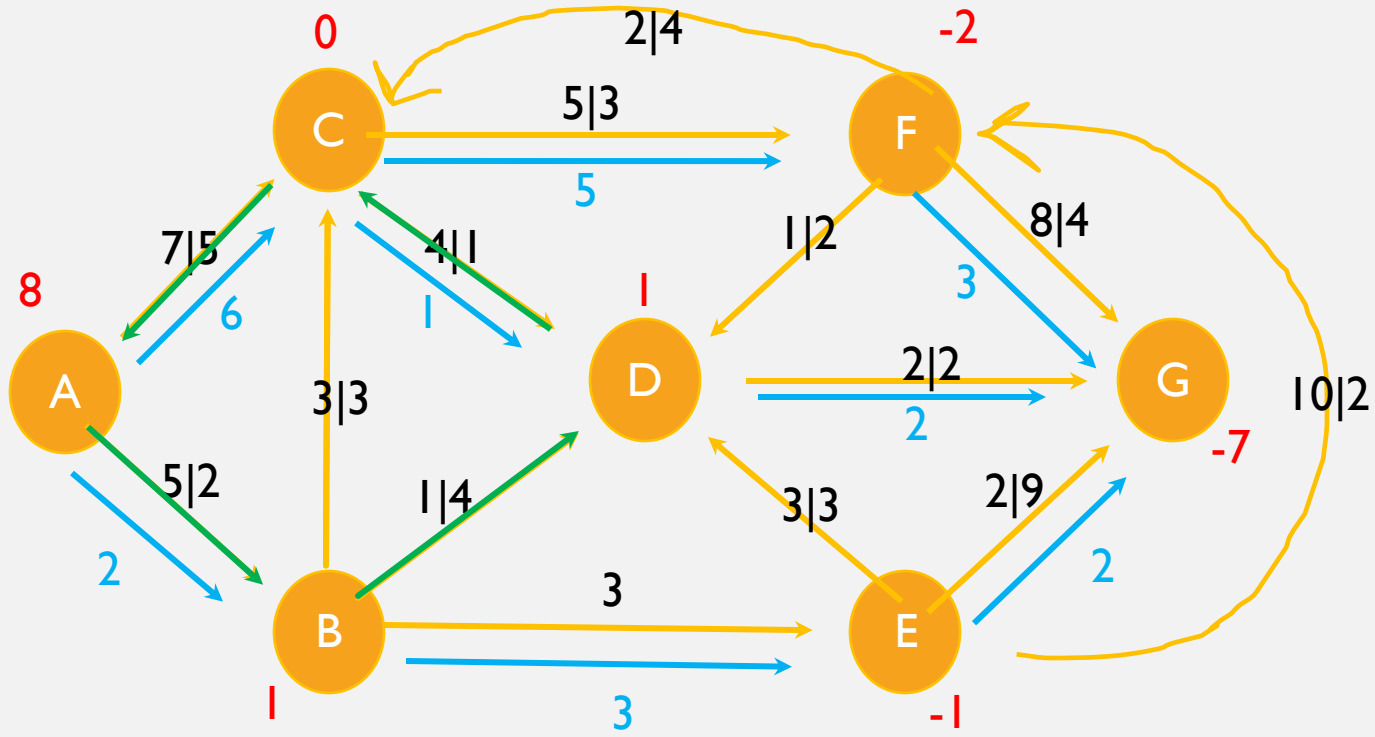
$$f(e) < c(e)$$

2) Every edge that's crossed in the backward direction by C has a positive flow greater than zero:

$$f(e) > 0$$

Assume we are given a directed graph G = (V,E) with edge capacities c: E→ $\mathbb{R}_+$, edge weights w: E→ $\mathbb{R}$, and node balances b: V→ $\mathbb{R}$ with $\sum_{v \in V}$ b(v) = 0 and a b-flow, f, on G.

-Traversing an edge in the opposite direction can be denoted as –w(e)



Minimum Cost Flow

A b-flow obtains a minimum cost when there is no f-augmenting cycle with negative edge weight.

Minimum Mean Cycle Canceling Algorithm

Assume we are dealing with a directed graph G = (V,E) with edge capacities c: E$\rightarrow \mathbb{R}_+$, edge weights w: E$\rightarrow \mathbb{R}$, and node balances b: V$\rightarrow \mathbb{R}$ with $\sum_{v \in V} b(v) = 0$ and a b-flow, f, on G.

1. Find some b-flow using a Maximum Flow Algorithm (ex: Edmonds-Karp)
2. While we can still obtain am f-augmenting cycle C with negative total weight and minimum mean weight,

a) Assign Γ = min {c(e) - f(e) if C traverses e forwards
f(e) if C traverses e backward}

b) For each edge e $\in$ C,
f(e) <- { f(e) + Γ if C traverses e forwards
f(e) - Γ if C traverses e backwards}

Conditional Networks

Time-Dependent Networks: Algorithms that consider time-dependent factors in network optimization, such as varying traffic conditions, time-dependent costs, and scheduling constraints. Applications in transportation planning and real-time routing.

Dynamic Graphs: Algorithms and data structures for handling dynamic changes in networks, such as edge insertions, deletions, and updates. Concepts include incremental algorithms and dynamic connectivity.

1.SCATS (Sydney Coordinated Adaptive Traffic System):

1. Operates on a centralized control system.
2. Two leveled hierarchical structure
3. Coordinates traffic signals centrally based on real-time data. Utilizes algorithms for dynamic adaptation.
4. Offers extensive reporting and monitoring capabilities.


scats
Move Smarter
How SCATS Works?

Global Optimization Systems

- System addresses unique transportation challenges with features tailored to optimize traffic flow, reduce congestion, and promote sustainable mobility.

2. SCOOT (Split Cycle Offset Optimization Technique):
-A centralized adaptive real time traffic control system. Uses flow data from traffic sensors, but other forms of detecting are increasingly being used. Adjusts cycle lengths and offsets dynamically. Known for quick adaptation to changing traffic conditions.

Seattle | English

Description

We've installed a new system along Mercer St between 3rd Ave W and I-5 that coordinates traffic signals to help keep people driving moving. The system, called "SCOOT" (**Split Cycle Offset Optimization Technique**), is the first in Seattle. In total, 32 signalized intersections use SCOOT.

Map

References

Melih. (1970, January 1). *Graphminator*. Ekim 2015. https://melihsozdinler.blogspot.com/2015/10/

Heuristics. (n.d.). https://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html

Prasanna. (n.d.). *A* algorithm (graph traversal and path search algorithm)*. enjoyalgorithms. https://www.enjoyalgorithms.com/blog/a-star-search-algorithm

YouTube. (2017, February 15). *A* (a star) search algorithm - computerphile*. YouTube. https://www.youtube.com/watch?v=ySN5Wnu88nE

YouTube. (2017b, April 20). *Airline scheduling – maths delivers*. YouTube. https://www.youtube.com/watch?v=-hDiXYoKJlw

Home. The OR Society. (n.d.). https://www.theorsociety.com/about-or/history-of-or/

Schrijver, A. (n.d.). On the history of the transportation and maximum flow . https://homepages.cwi.nl/~lex/files/histtrpclean.pdf

Shen, C. (2016, May 27). Max Flow, Min Cut. https://web.stanford.edu/class/archive/cs/cs161/cs161.1166/lectures/lecture16.pdf

Network flow II. (n.d.). https://www.cs.cmu.edu/~avrim/451f11/lectures/lect1027.pdf

Maximum flow - ford-fulkerson and Edmonds-Karp¶. Maximum flow - Ford-Fulkerson and Edmonds-Karp - Algorithms for Competitive Programming. (2023, September 17). https://cp-algorithms.com/graph/edmonds_karp.html

Daymude, J. (2022, October 20). *CSE 550 (2022, fall): 3.5 algorithms for maximum-flow*. YouTube. https://www.youtube.com/watch?v=H8kGp1cQO9w&t=793s

Friggstad, Z. (n.d.). Lecture 9 (Sept 26): The mean cycle canceling algorithm. https://webdocs.cs.ualberta.ca/~zacharyf/courses/combopt_2016/notes/lec9.pdf

Network flows. (n.d.-b). https://ac.informatik.uni-freiburg.de/lak_teaching/ws11_12/combopt/notes/network_flows.pdf

Scats. Home. (n.d.). https://www.scats.nsw.gov.au/home

Mercer Scoot. Mercer SCOOT - Transportation. (n.d.). https://www.seattle.gov/transportation/projects-and-programs/programs/technology-program/mercer-scoot

THANK YOU

QUESTIONS?