fit@hcmus

DATA STRUCTURES & ALGORITHMS

Lecture 7: GRAPHS – part 2

Lecturer: Dr. Nguyen Hai Minh

OUTLINE

- Other applications of Graphs
  - Shortest Path
  - Circuits
  - Difficult Problems

SHORTEST PATH

Dijkstra's Algorithm

Shortest Paths

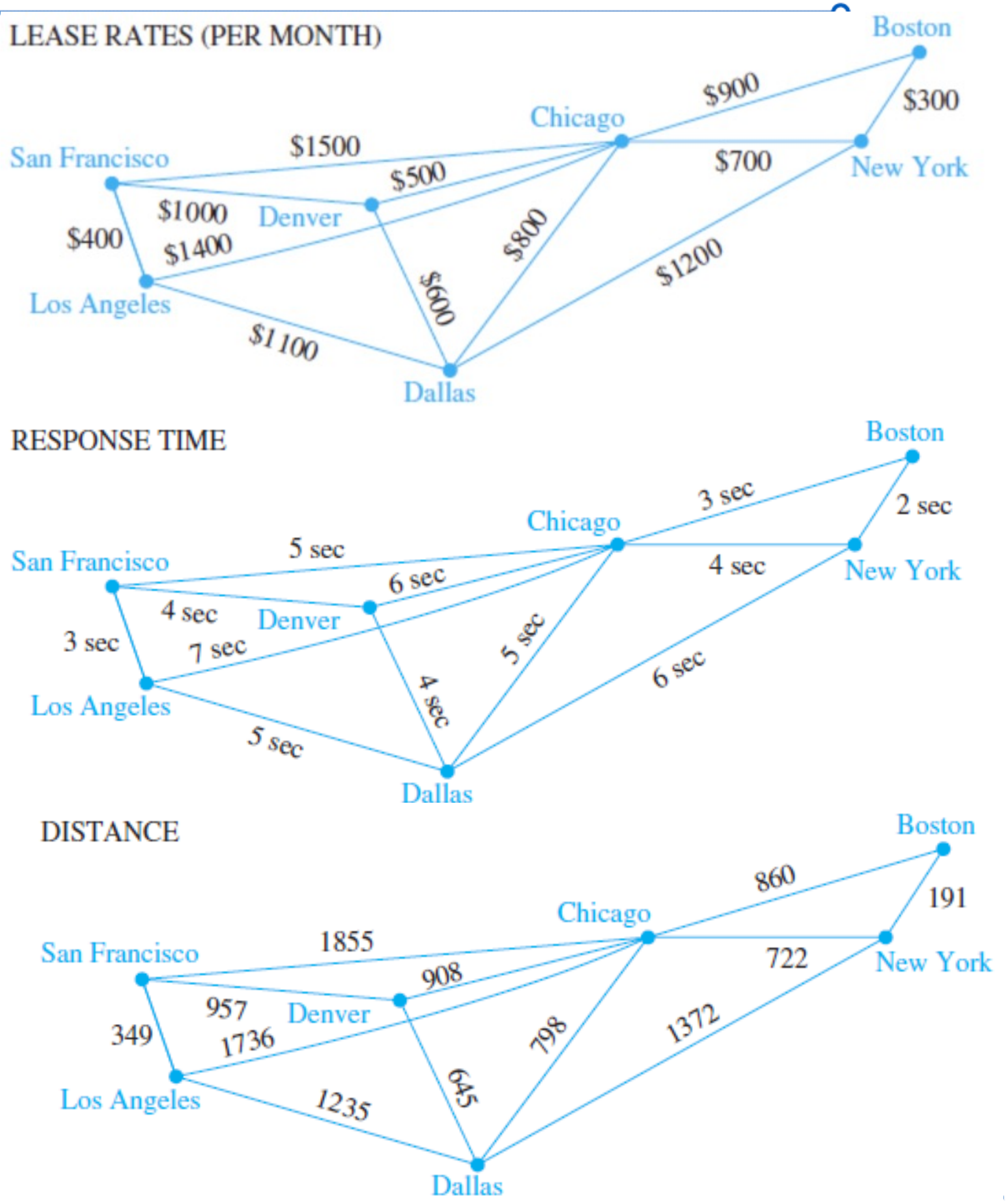
- Many problems can be modeled using graphs with weights assigned to their edges:
  - Airline flight problems
  - Computer networks problems
  - GPS navigation systems

Airline Flight Problems

- Vertices: cities
- Edges: flights
- Weights of edges depend on the problem.
  - Distance between cities
  - Flight fare
  - Flight time
  - …

Computer Networks Problems

- Vertices: computers
- Edges: lines between computers
- Weights:
  - Communication costs (e.g., monthly cost of leasing a telephone line)
  - Response times of the computers over these lines
  - Distances between computers



Shortest Paths

- The input to the shortest-paths problem is a weighted, directed graph $G = (V, E)$, with a weight function $w: E \rightarrow \mathbb{R}$ mapping each edges to real-valued weights.
- We define the *shortest-path weight* $\boldsymbol{\delta(u, v)}$ from u to v by

$$\boldsymbol{\delta(u,v)} = \begin{cases} \mathbf{min}\left\{\boldsymbol{w(p)}: \boldsymbol{u} \overset{\boldsymbol{p}}{\rightarrow} \boldsymbol{v}\right\} & \textbf{if there is a path from } \boldsymbol{u} \textbf{ to } \boldsymbol{v} \\ \infty & \textbf{otherwise} \end{cases}$$

where

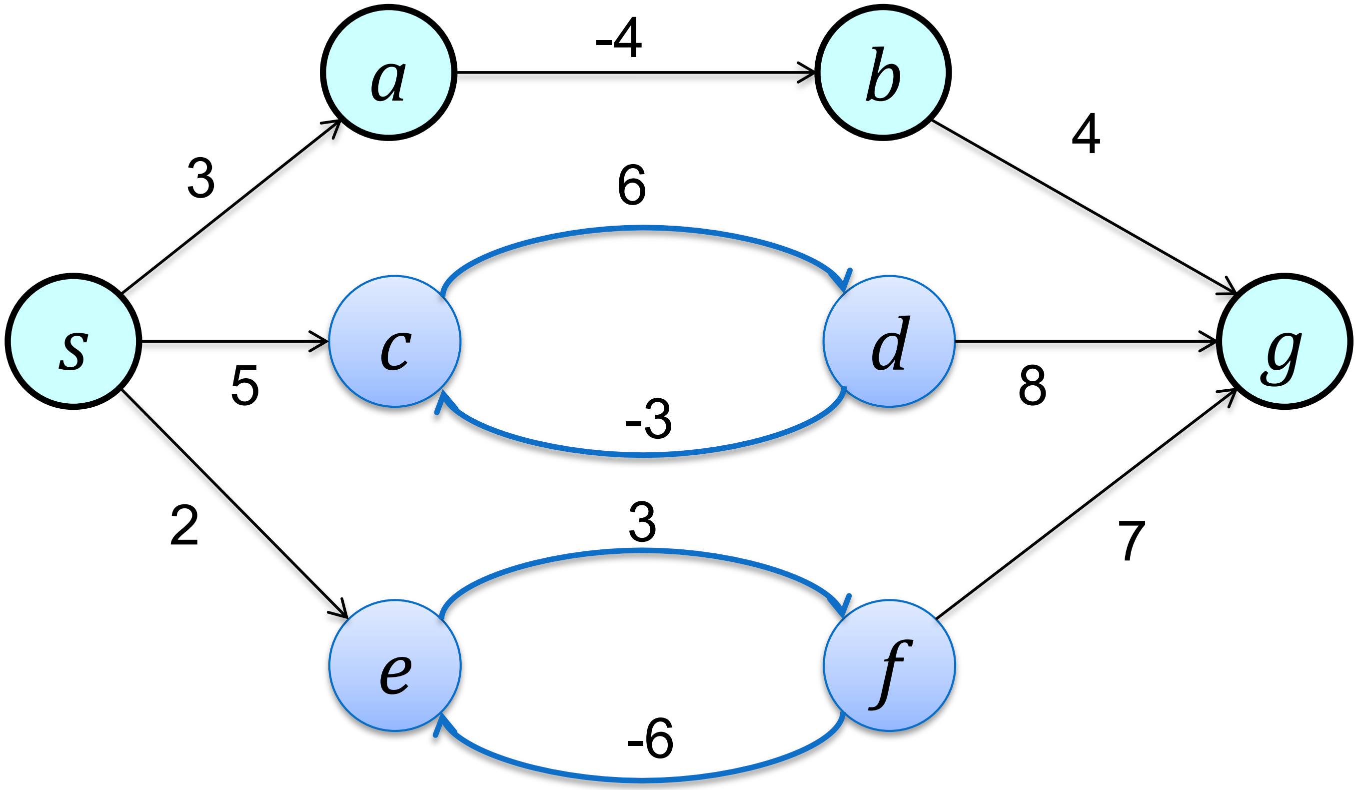
$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$ is the weight of path $p = \langle v_0, v_1, \dots, v_k \rangle$

Shortest Paths – Variants

- Single-source shortest paths problem
- Single-destination shortest paths problem
- Single-pair shortest paths problem
- All-pairs shortest paths problem

Shortest Paths – Negative-weight

- Negative-weight edges



Shortest path	cost
$s \to a$	3
$s \to b$	-1
$s \to c$	5
$s \to d$	11
$s \to e$	$-\infty$
$s \to f$	$-\infty$
$s \to g$	$-\infty$

Negative cycles

Shortest Paths

- Cycles:
  - A shortest path cannot contain a cycle
- Representing shortest paths: each vertex v in the graph G has:
  - A predecessor $\boldsymbol{v.\pi}$ that is another vertex or NIL
  - A shortest-path estimate $\boldsymbol{v.d}$ which is an upper bound on the weight of a shortest path from source *s* to *v*

Shortest Paths

- Dijkstra's shortest-path algorithm
  - Determine the shortest path between a given vertex and all other vertices (Single-source shortest paths)
  - Assume that weight of edges ≥ 0
  - The directed graph G is stored in the adjacency-list representation.

Dijkstra's Algorithm Idea

- We maintain a set of vertices *S* whose final shortest path lengths have already been determined
  - In each time we consider not yet discovered vertices in the graph, and all edges going from a discovered vertex (*u*) to an undiscovered vertex (*v*).
  - We choose an undiscovered vertex with an edge from *u* to *v*, that gives the shortest path length.
  - The length from *u* to *v* for each vertex *v*, is given by the length of *u*, plus the weight between *u* and *v*.
- In the initialization step:
  - We include source node *S* in the set of discovered nodes and set its length to 0.
  - All other lengths are initially infinity.
- Then we keep expanding set *S* of discovered nodes in a greedy manner.

Dijkstra's Algorithm

INITIALIZE-SINGLE-SOURCE(G, s)

for each vertex $v \in G.V$
    $v.d = \infty$
    $v.\pi = \text{NIL}$
$s.d = 0$

DIJKSTRA(G, w, s)

INITIALIZE-SINGLE-SOURCE(G, s)
$S = \emptyset$
$Q = \emptyset$
for each vertex $u \in G.V$
    INSERT(Q, u)
while $Q \neq \emptyset$
    $u =$ EXTRACT-MIN(Q)
    $S = S \cup \{u\}$
    for each vertex v in $G.Adj[u]$
        RELAX(u, v, w)
        if the call of RELAX decreased $v.d$
            DECREASE-KEY$(Q, v, v.d)$

RELAX(u, v, w)

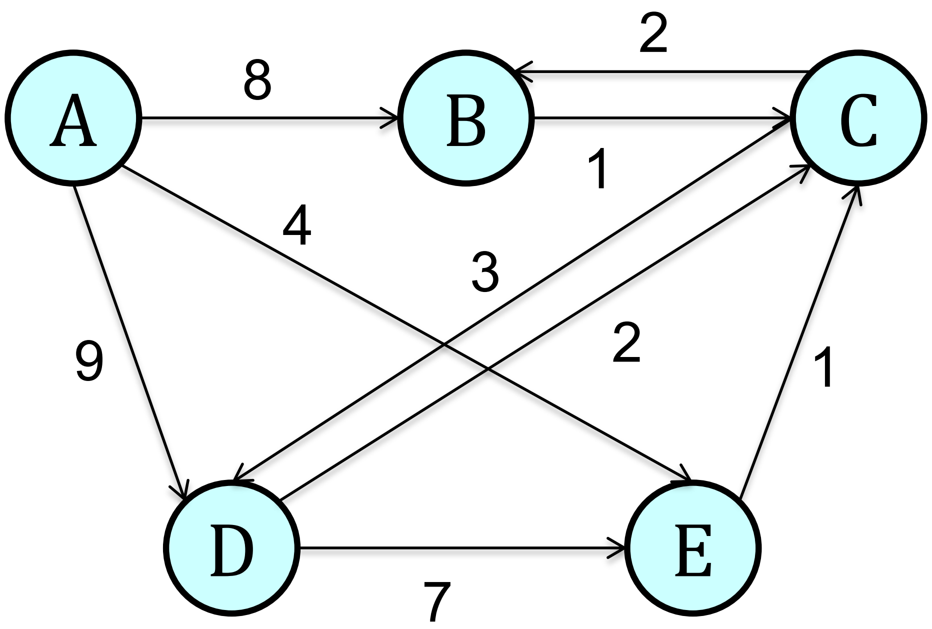
if $v.d > u.d + w(u, v)$
    $v.d = u.d + w(u, v)$
    $v.\pi = u$

Update the min-priority queue

EXTRACT-MIN(H) deletes the element from heap H whose key is minimum, returning a pointer to the element.

Dijkstra's Algorithm – Example

- A weighted directed graph and its adjacency list



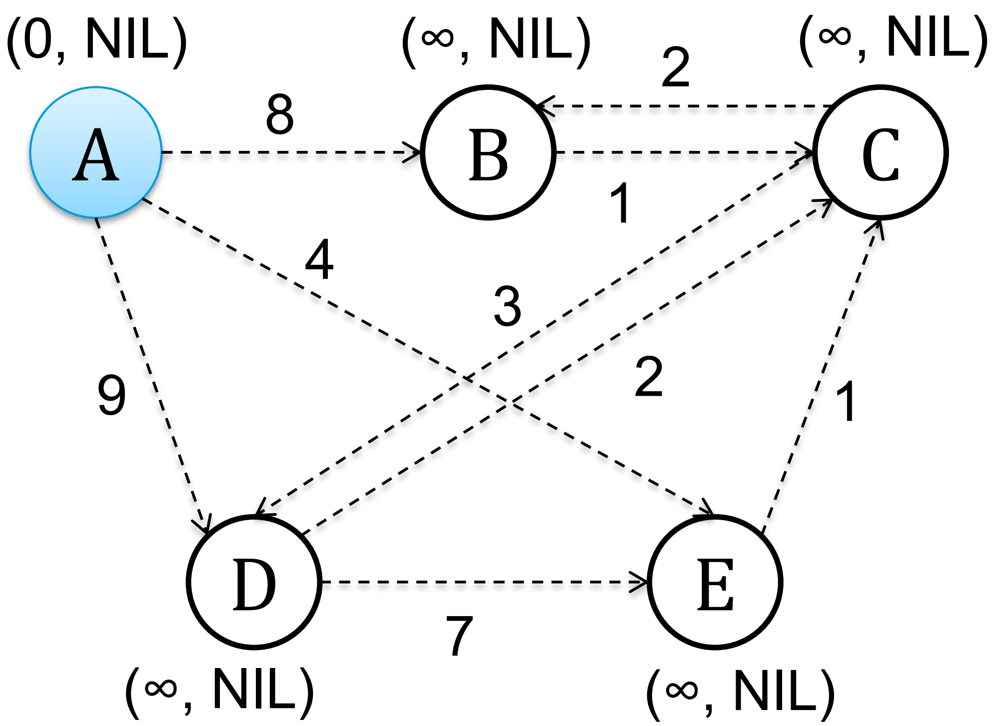
A	B_8 D_9 E_4
B	C_1
C	B_2 D_3
D	C_2 E_7
E	C_1

Dijkstra's Algorithm – Example

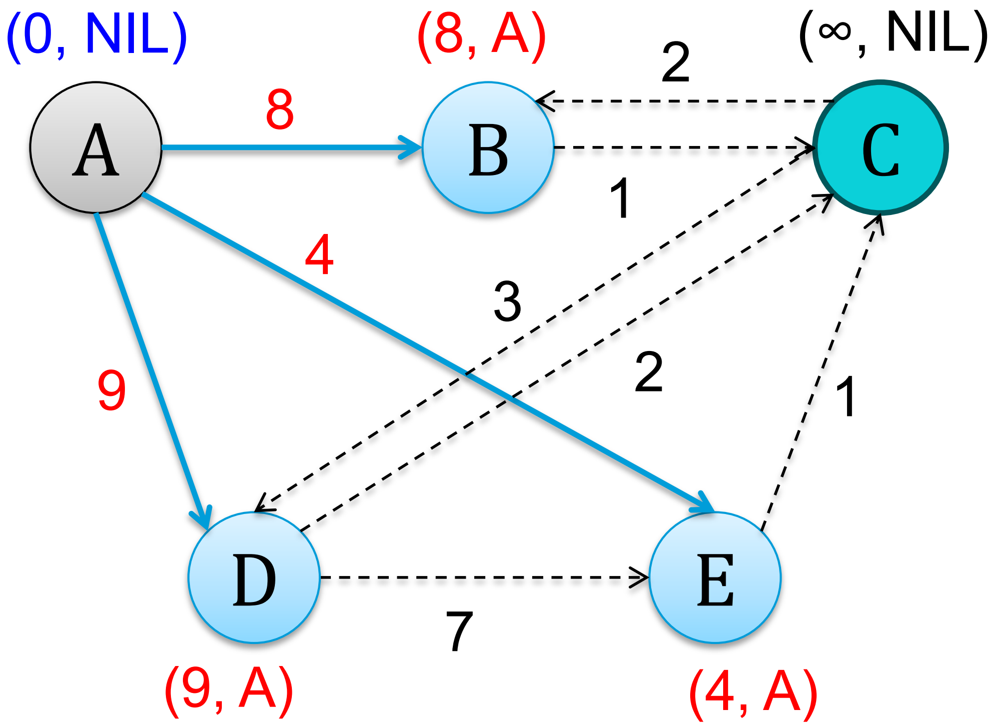
Step	Q	S	$(v.d, v.\pi)$				
			A	B	C	D	E
Init	A, B, C, D, E	$\emptyset$	$(0, NIL)$	(∞, NIL)	(∞, NIL)	(∞, NIL)	(∞, NIL)
1	B, C, D, E	A	$(0, NIL)$	$(8, A)$	(∞, NIL)	$(9, A)$	$(4, A)$
2	B, C, D	A, E	$(0, NIL)$	$(8, A)$	$(5, E)$	$(9, A)$	$(4, A)$
3	B, D	A, E, C	$(0, NIL)$	$(7, C)$	$(5, E)$	$(8, C)$	$(4, A)$
4	D	A, E, C, B	$(0, NIL)$	$(7, C)$	$(5, E)$	$(8, C)$	$(4, A)$
5	$\emptyset$	A, E, C, B, D	$(0, NIL)$	$(7, C)$	$(5, E)$	$(8, C)$	$(4, A)$

Dijkstra's Algorithm – Example

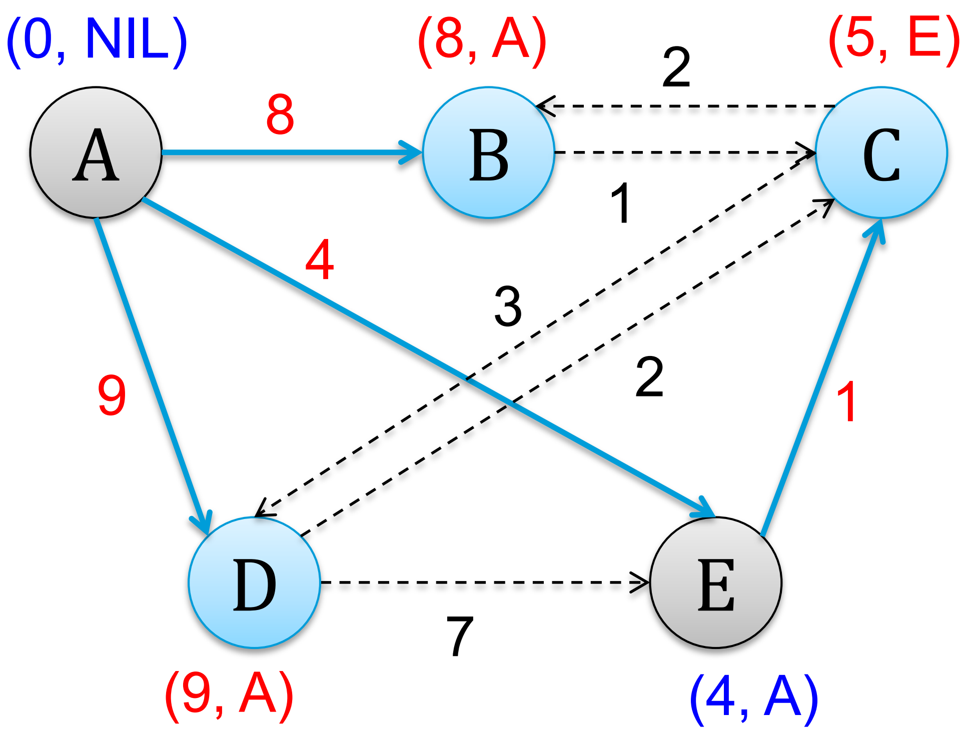
- Init the distances and predecessors from A to all v in G.



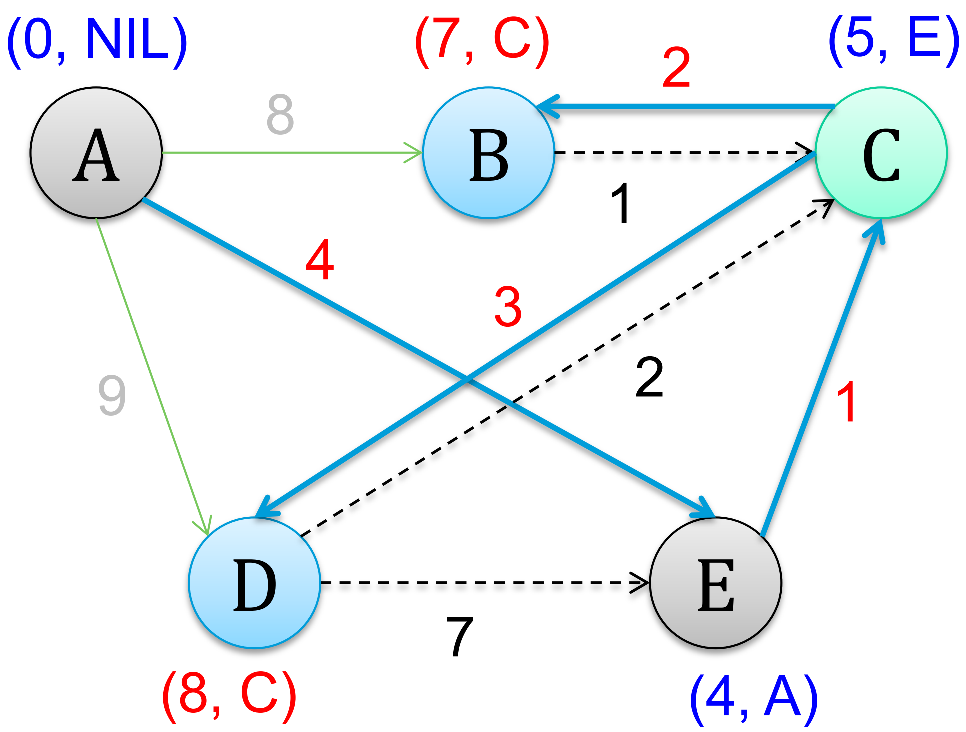
Dijkstra's Algorithm – Example



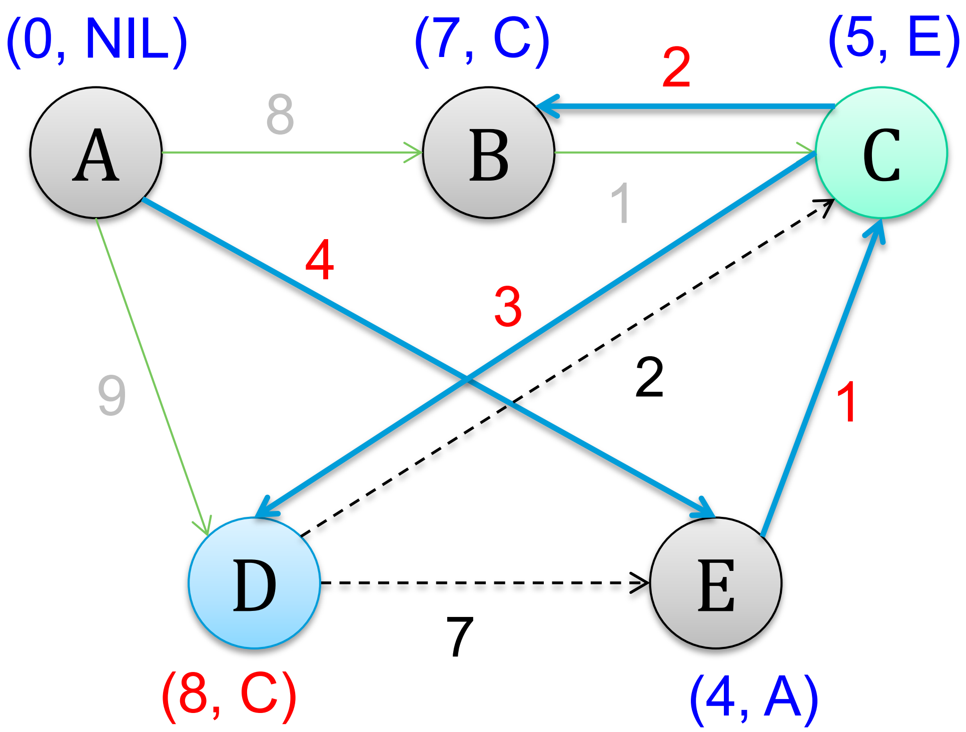
Dijkstra's Algorithm – Example



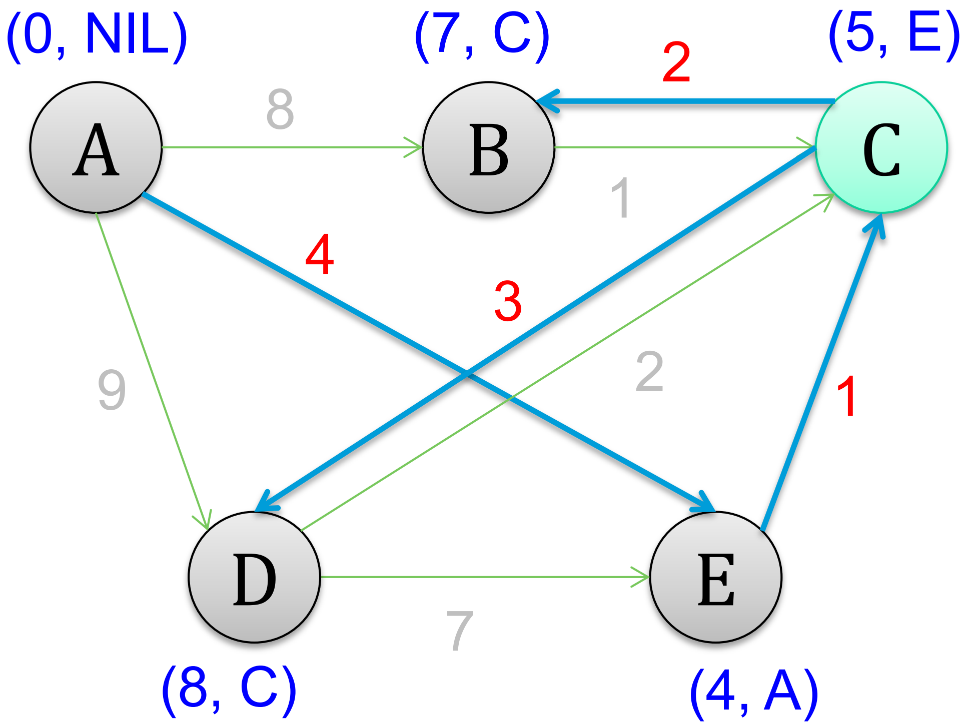
Dijkstra's Algorithm – Example



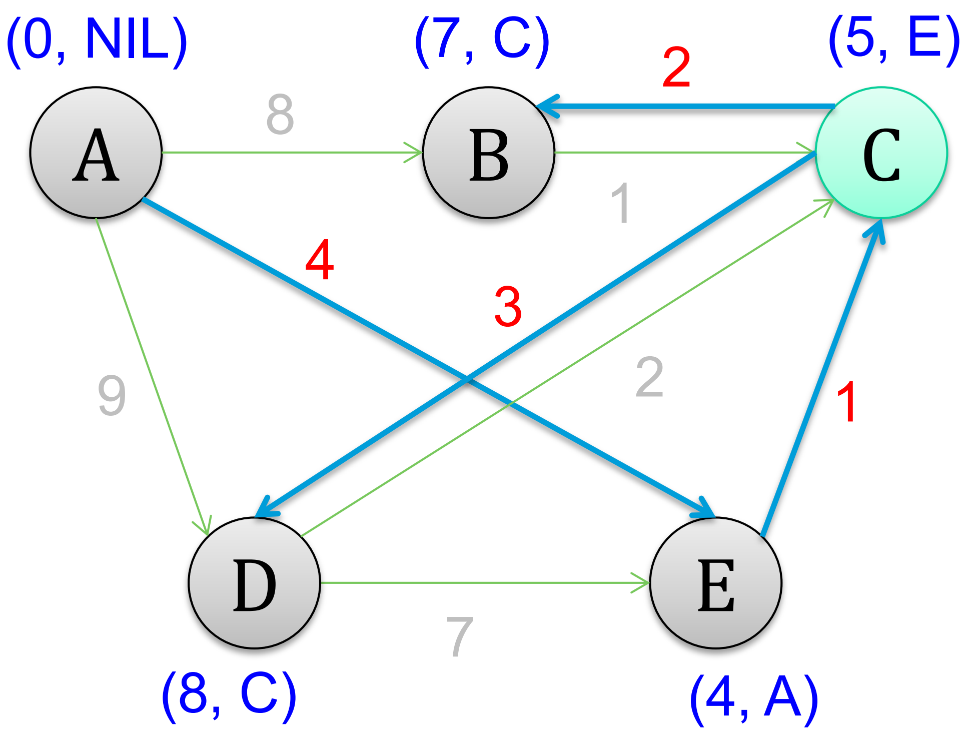
Dijkstra’s Algorithm – Example



Dijkstra's Algorithm – Example



Dijkstra's Algorithm – Example



Dijkstra's Algorithm – Analysis

- We run through each node once.
- For each node we look into its adjacency list.

→ $(|V| + |E|)$ number of operations.

- However, each operation takes time since we need to find the minimum among all possible edges.

→ Use a priority queue with each minimum taking $O(\log_2 |V|)$ time.

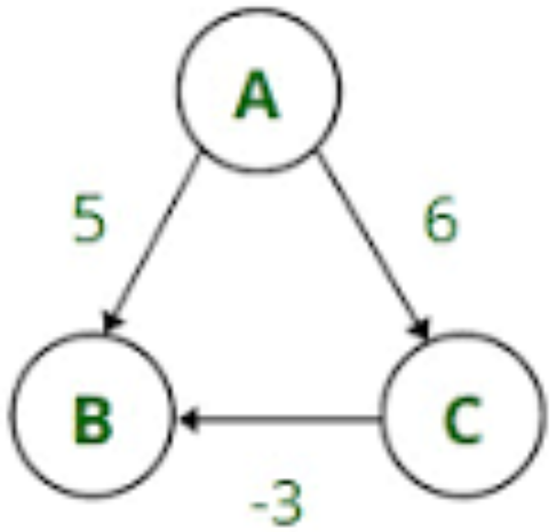
- As a consequence, we have $O((|V| + |E|) \log_2 |V|)$ complexity.

Dijkstra's Algorithm – Analysis

- We can improve this complexity to $O(|E| + |V| \log_2 |V|)$ just like in Prim's algorithm; i.e., implement the min-priority queue using Fibonacci Heap.

Dijkstra's Algorithm – Negative Edges

- Dijkstra's algorithm fails when the graph has negative weight.



CIRCUITS

Eurler Circuit

Hamilton Circuit

CIRCUITS

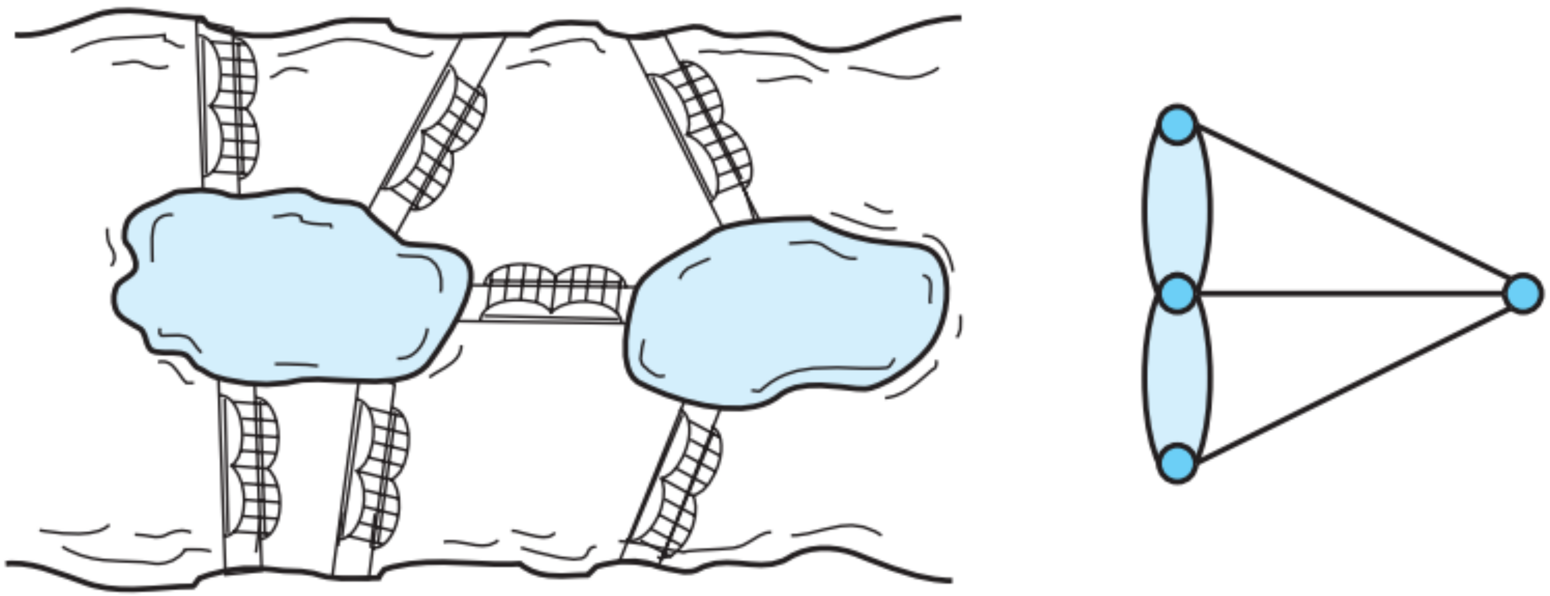
- A **circuit** is simply another name for a type of cycle that is common in some problems.
  - Recall: a cycle in a graph is a path that begins and ends at the same vertex.
- Typical circuits either visit every vertex once or visit every edge once.

CIRCUITS – The Bridge Problem

- The first application of graphs (early 1700s) by Euler:
  - Two islands in a river are joined to each other and to the river banks by several bridges
  - The bridges → *edges in multigraph*
  - The land masses → *vertices*
  - The problem asked whether you can begin at a vertex *v*, pass through every edge exactly **once**, and terminate at *v*.

CIRCUITS – The Bridge Problem

- No solution exists for this particular configuration of edges and vertices.

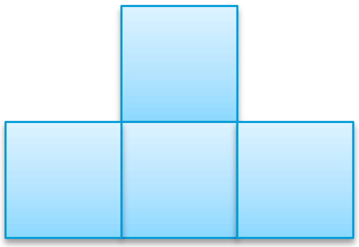


Euler Circuits

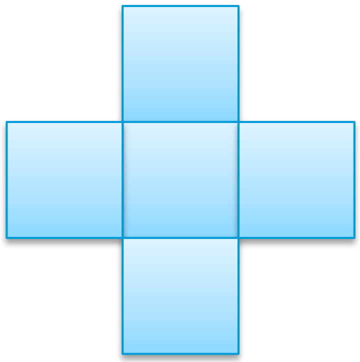
- **Euler circuit:** path that begins at a vertex *v*, passes through every edge in the graph exactly once, and terminates at *v*.
  - Consider a simple undirected graph rather than a multigraph.
  - Euler circuit exists if and only if each vertex touches an even number of edges (or has an even degree).

Euler Circuits – Example

- Pencil and Paper drawings:
  - Drawing without lifting your pencil or redrawing a line, ending at your starting point.



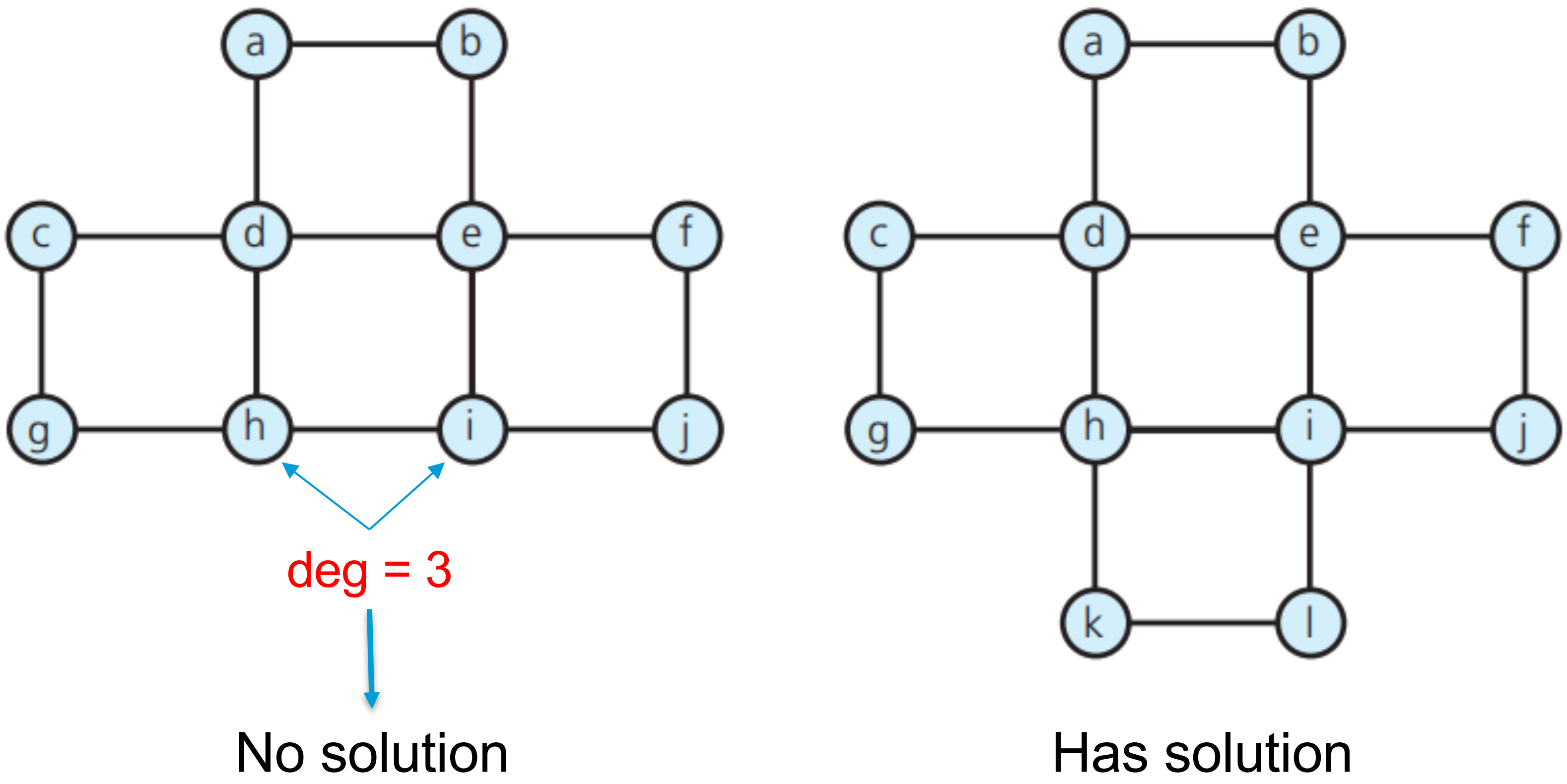
No solution



Has solution

Euler Circuits – Example

- Graphs based on previous example



Euler Circuits – Algorithm

- In case the Euler Circuit exists, we can find it by a strategy using DFS that marks edges instead of vertices as they are traversed.
  - You will find a cycle.
  - Then, find the first vertex along the cycle that touches an unvisited edge.
  - Loop until there is no unvisited edge.

SOME DIFFICULT PROBLEMS

- The travelling salesperson problem
- The three utilities problem
- The four-color problem

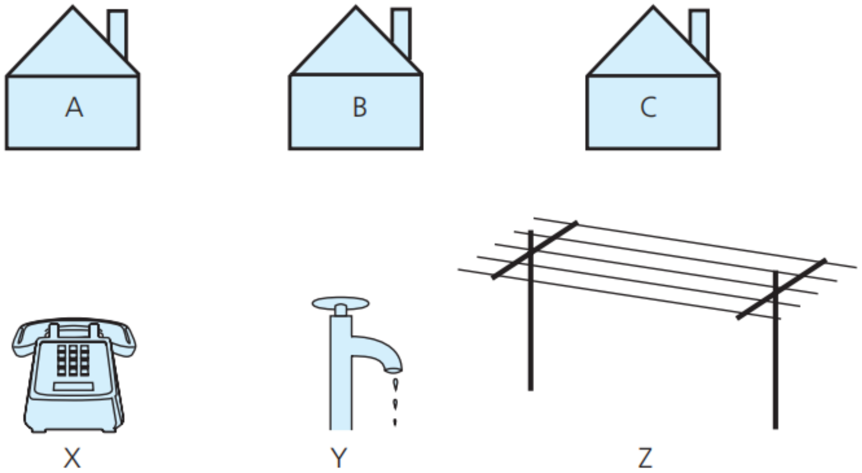
The travelling salesperson problem

- A **Hamilton circuit** is a path that begins at a vertex v, passes through every vertex in the graph exactly once, and terminates at v.
  - → Determine whether a graph contains a Hamilton circuit is difficult!
- The TSP is a variation of this problem:
  - Involves a weighted graph that represent a roadmap.
  - Each edge has a cost.
  - → The salesperson must visit every city exactly once and return to the original city with the least cost.

→ *Solving this problem is not easy!*

The three utilities problem

- Is it possible to connect each house to each utility with edges that do not cross one another?



NO!
Because it is not a **planar graph!**

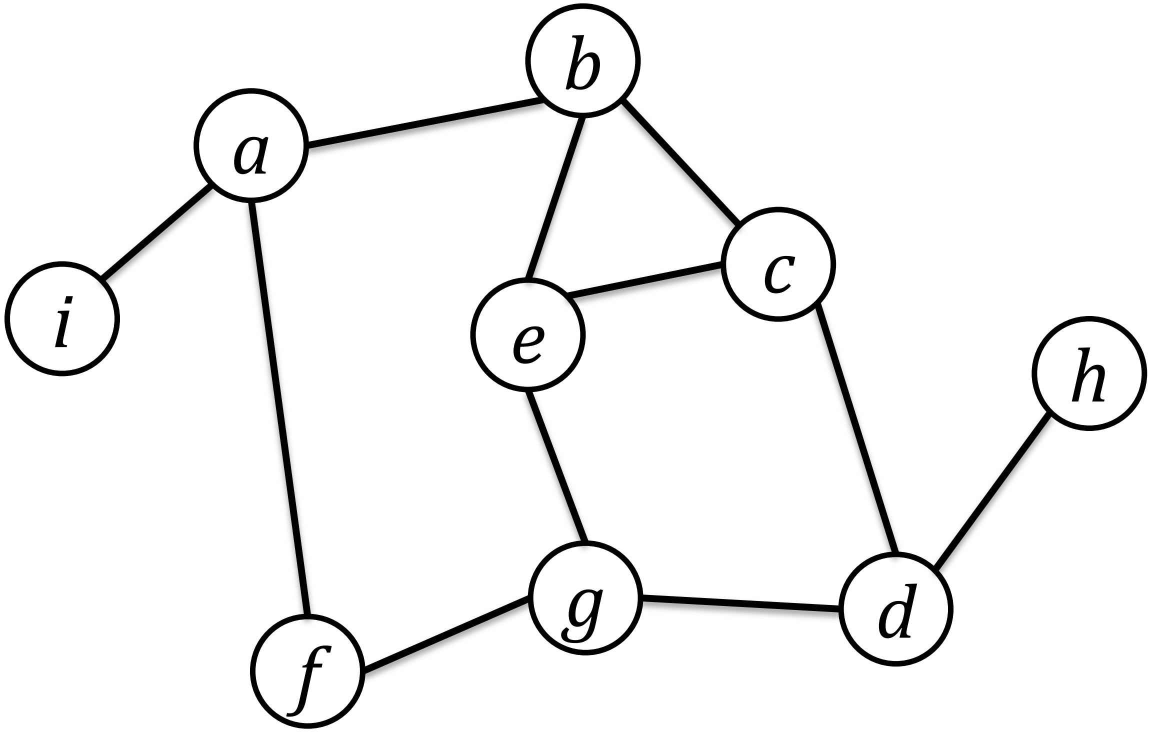
- Generation: Determine whether a given graph is planar?
  - Design an electronic circuit so that the connections do not cross

The four-color problem

- Given a planar graph, can you color the vertices so that no adjacent vertices have the same color, if you use at most four colors?
  - → The answer is **YES**, but it is difficult to prove.
  - → In fact, this problem was posed more than a century before it was solved in the 1970s with the use of a computer.

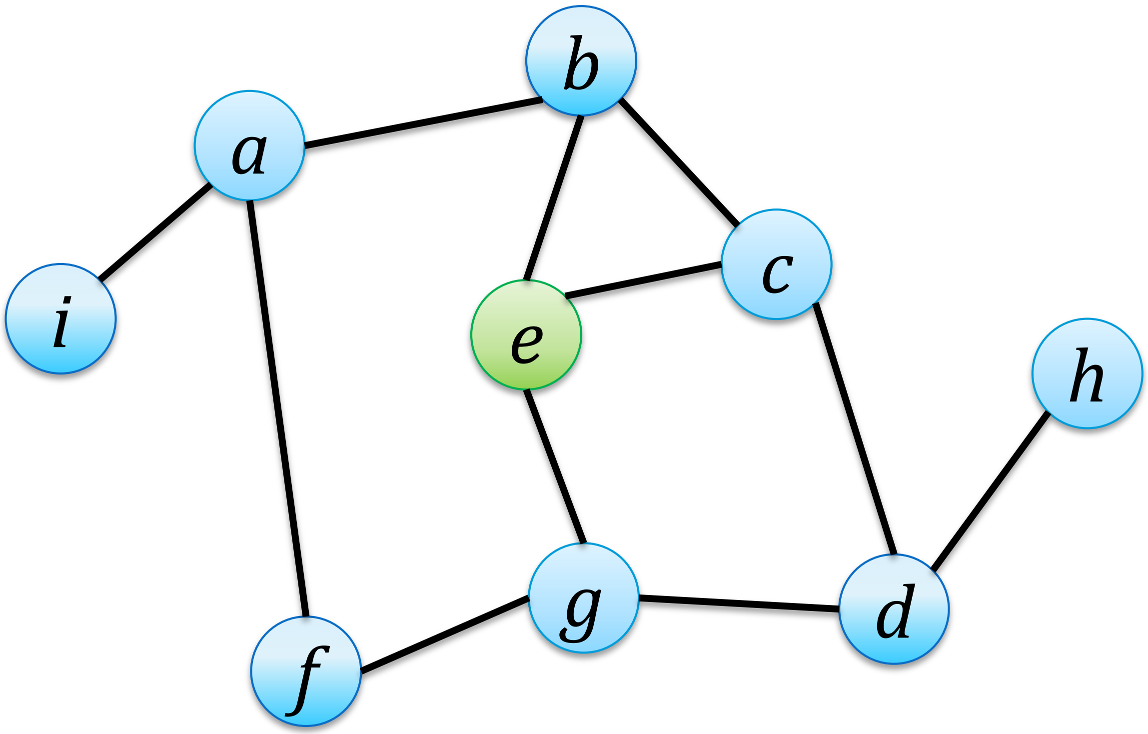
The four-color problem – Example

- Use 3 colors to color the following map



The four-color problem – Example

- Use 3 colors to color the following map



What's Next?

- After today:
  - Read Textbook 1: Chapter 20, 21, 22
  - Read Textbook 2: Chapter 20

Q & A