# Algorithms on Graphs

Umang Bhaskar & Juhi Chaudhary

Day 1, session 1: Basic graph algorithms


STCS Vigyan Vidushi 2024

# Hello!

Umang Bhaskar

born and brought up in Allahabad, UP

spent time in:

MNNIT Allahabad → TCS → IIT Bombay → Dartmouth College → Caltech → U of Waterloo

faculty in STCS, TIFR since 2015

enjoy traveling, hiking, reading (fantasy / sci fi / biographies / …) , chatting, …

dislike bullet points

# Hello!

Juhi Chaudhary

spent time in:

Miranda House (Maths (H)) → IITD → Ben-Gurion University → TIFR (since Jan ’24)

work in graph theory, game theory, parametrized complexity, …

will also be joining you for the mentorship sessions

# This course

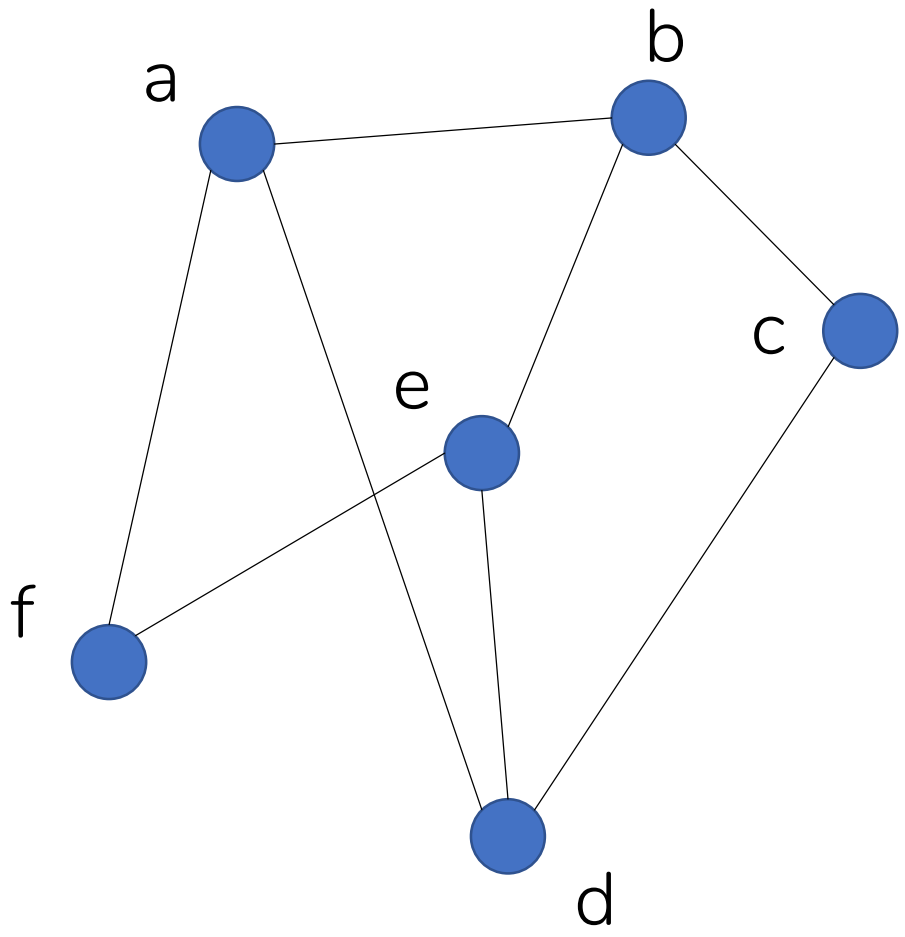
Graphs are v v fundamental

Focus is going to be on understanding

All material will be uploaded to the webpage
www.tcs.tifr.res.in/~umang/vv24.html

To start with, please put your name and where you're from
on the sheet of paper being passed around.

# Nomenclature




A graph (or network)

consists of vertices (or nodes)

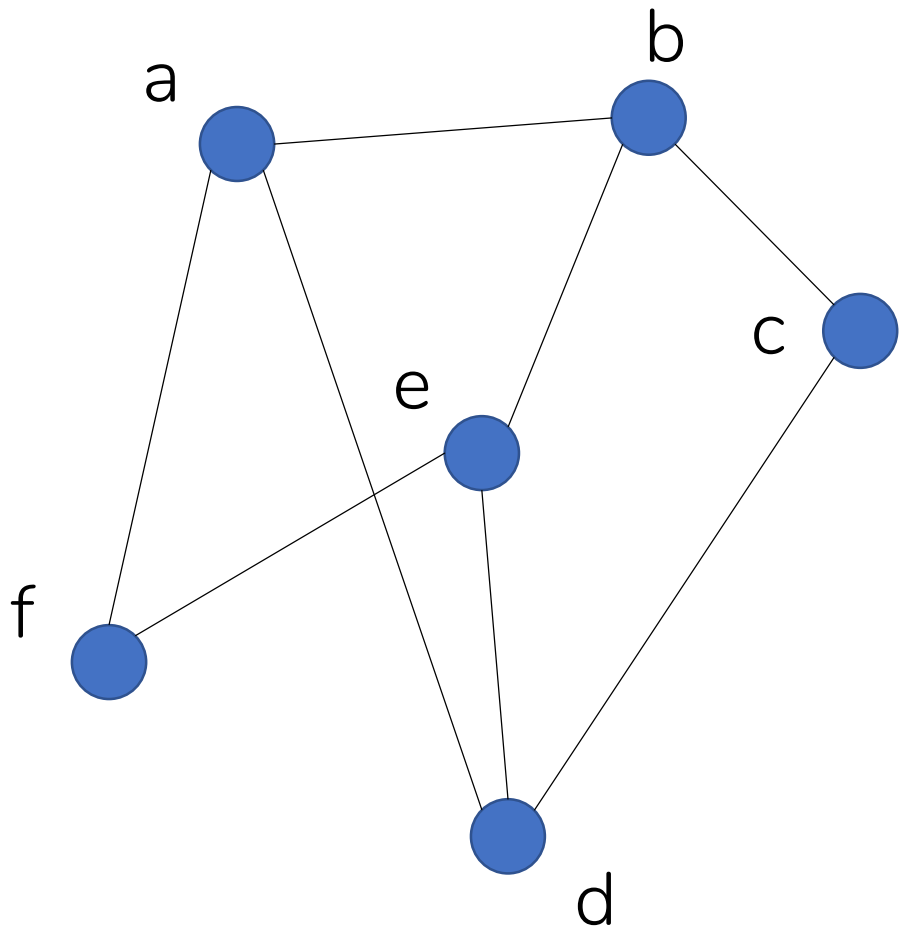
connected by edges (or links or arcs)

$$G = (V, E)$$

where $V = \{a, b, c, d, e, f\}$

and $E = \{\{a, b\}, \{a, f\}, \{a, d\}, \{b, e\}, \{b, c\}, \{c, d\}, \{e, d\}, \{e, f\}\}$

# Nomenclature

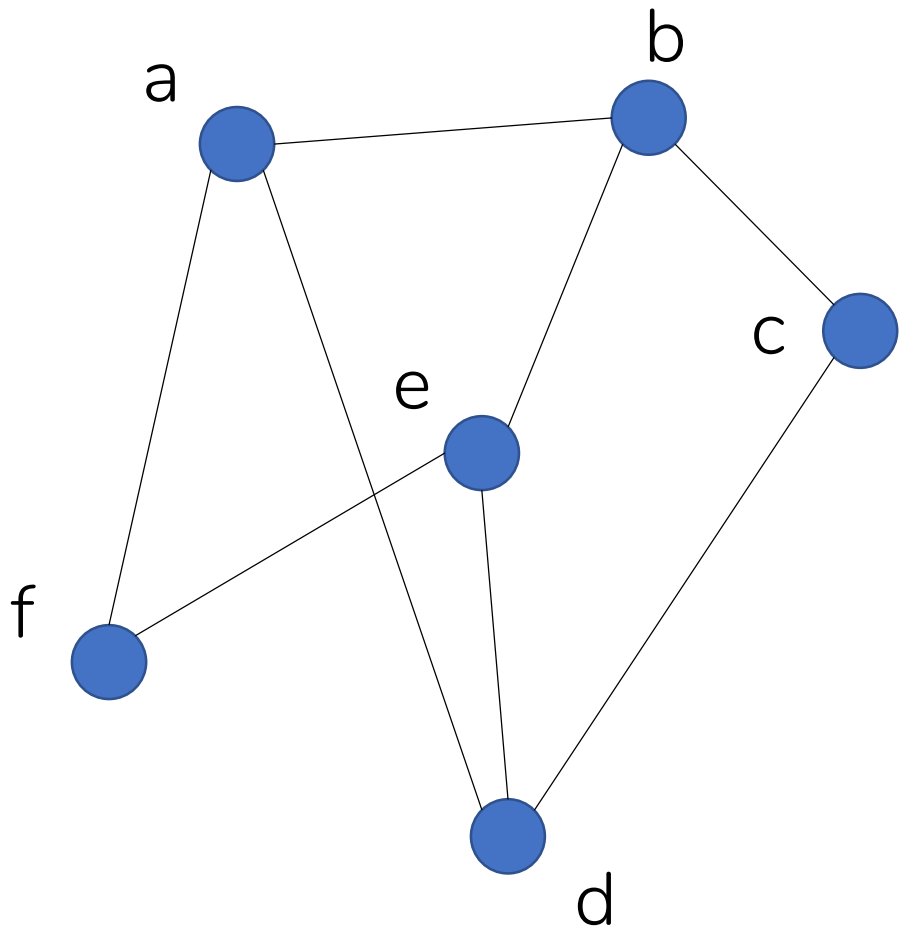



Neighbourhood $N(v)$: set of vertices adjacent to $v$

E.g., $N(a) = \{b, d, f\}$

Degree $deg(v) \coloneqq |N(v)|$

# Nomenclature




Neighbourhood $N(v)$: set of vertices adjacent to $v$

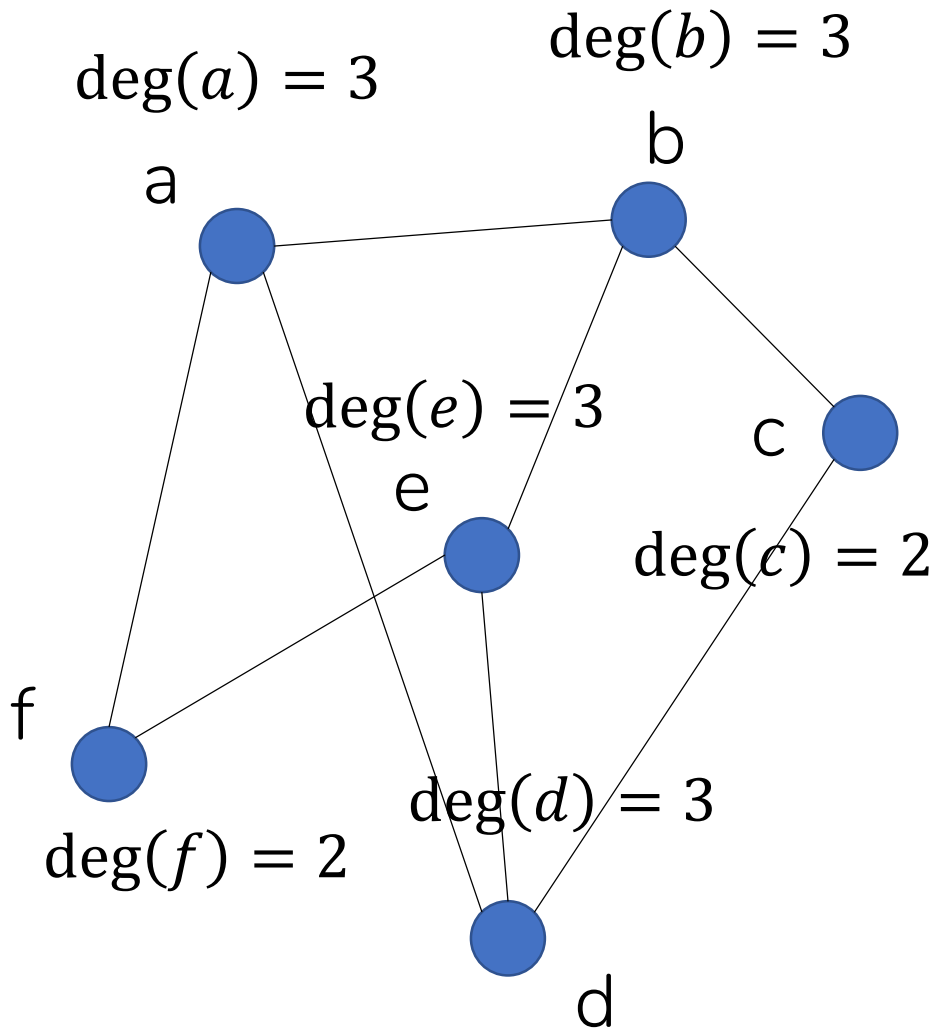
E.g., $N(a) = \{b, e, f\}$

Degree $deg(v) := |N(v)|$

Q: Given a graph G = (V,E),

$\sum_v \deg(v)$ ?? $|E|$

# Nomenclature




Neighbourhood $N(v)$: set of vertices adjacent to $v$

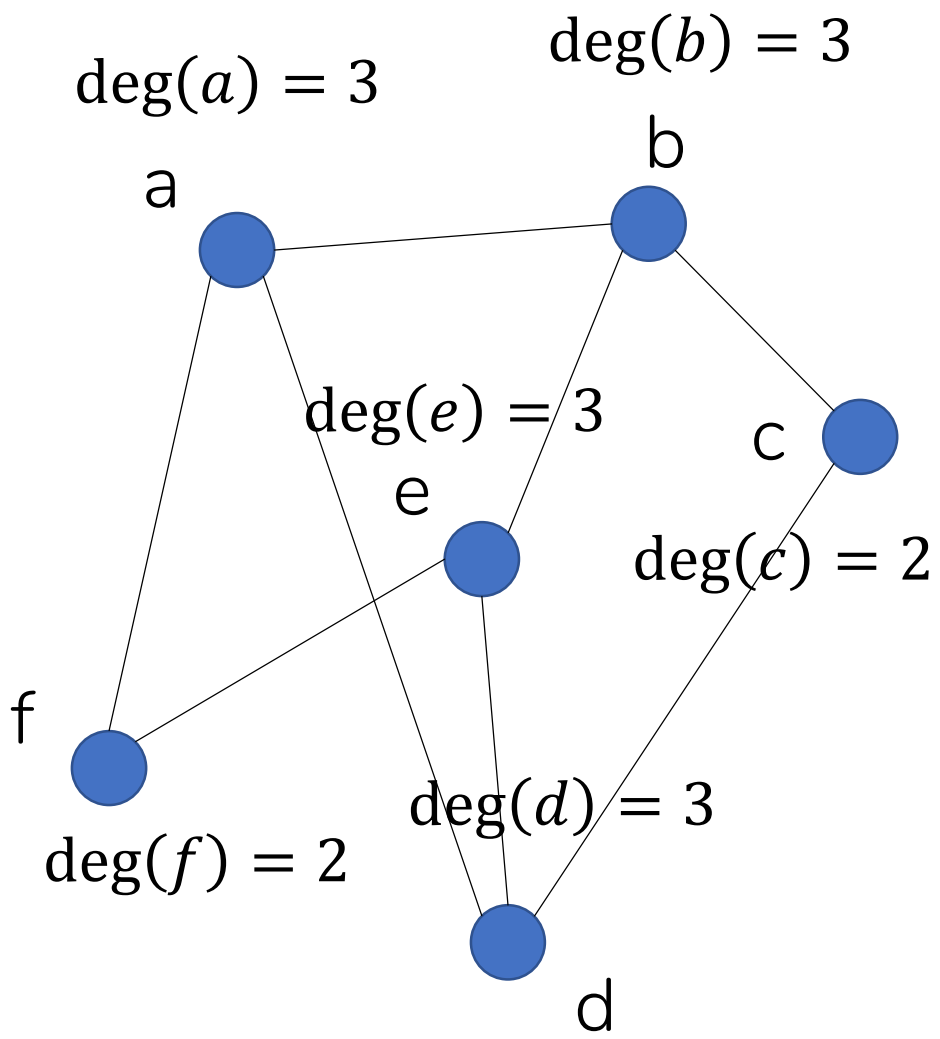
E.g., $N(a) = \{b, e, f\}$

Degree $deg(v) := |N(v)|$

Q: Given a graph G = (V,E),

$\sum_v \deg(v)$ ?? $|E|$

# Nomenclature




$\sum_v \deg(v) = 16, \quad |E| = 8$

Neighbourhood $N(v)$: set of vertices adjacent to $v$

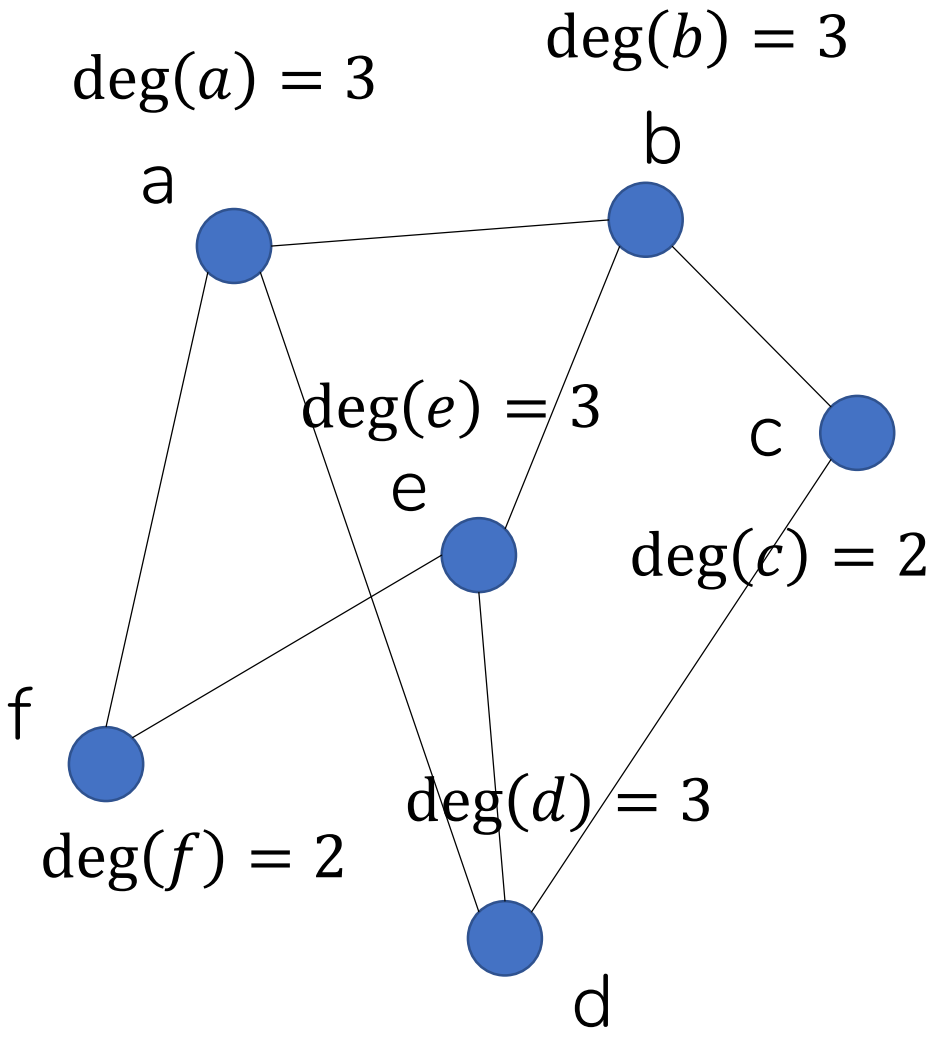
E.g., $N(a) = \{b, e, f\}$

Degree $deg(v) := |N(v)|$

Q: Given a graph G = (V,E),

$\sum_v \deg(v)$ ?? $|E|$

# Handshaking Lemma




$\sum_v \deg(v) = 16, \quad |E| = 8$

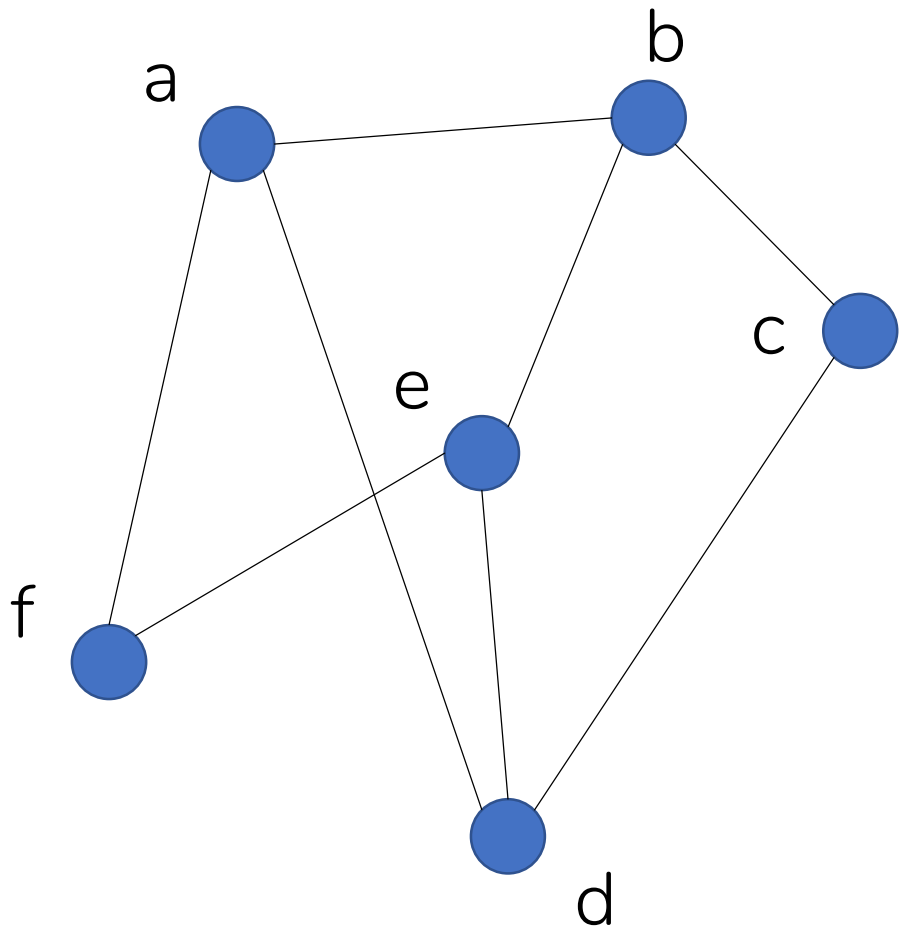
Given a graph $G = (V, E)$,

$$\sum_v \deg(v) = 2|E|$$

Hence,
# vertices with odd degree is even

(this is called the handshaking *lemma*)

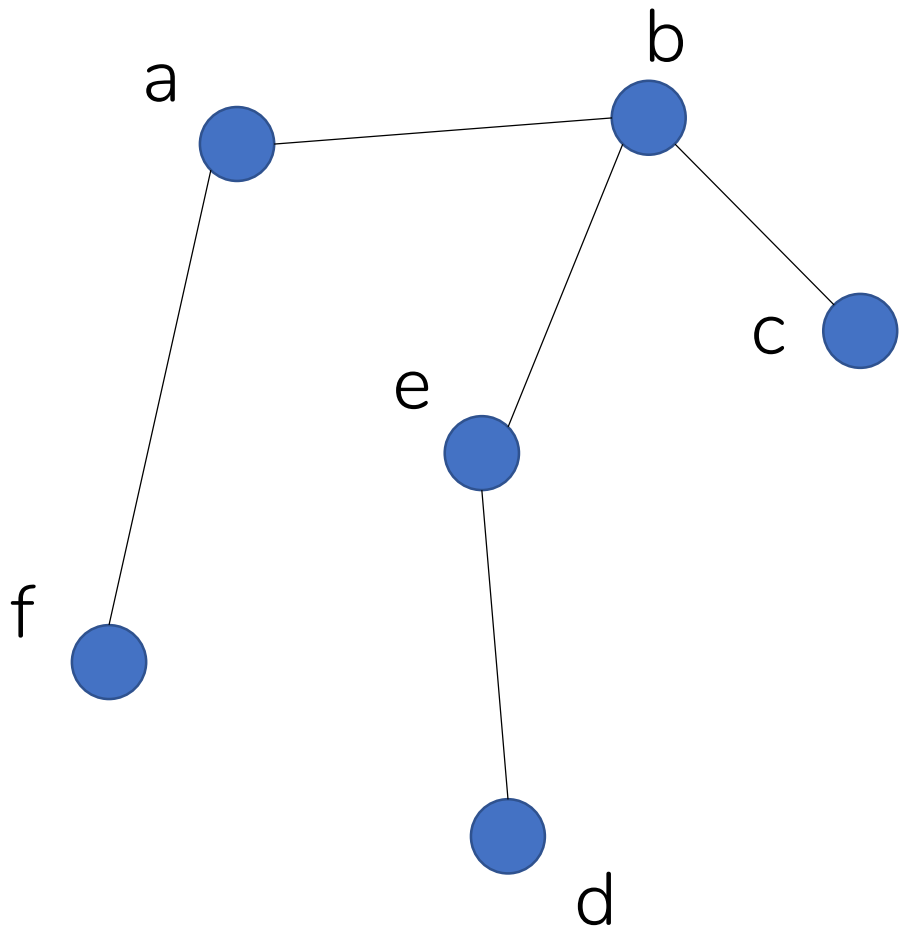
# Connected graph




For graph $G = (V, E)$ to be connected,

$$|E| \geq |V| - 1$$

# Connected graph

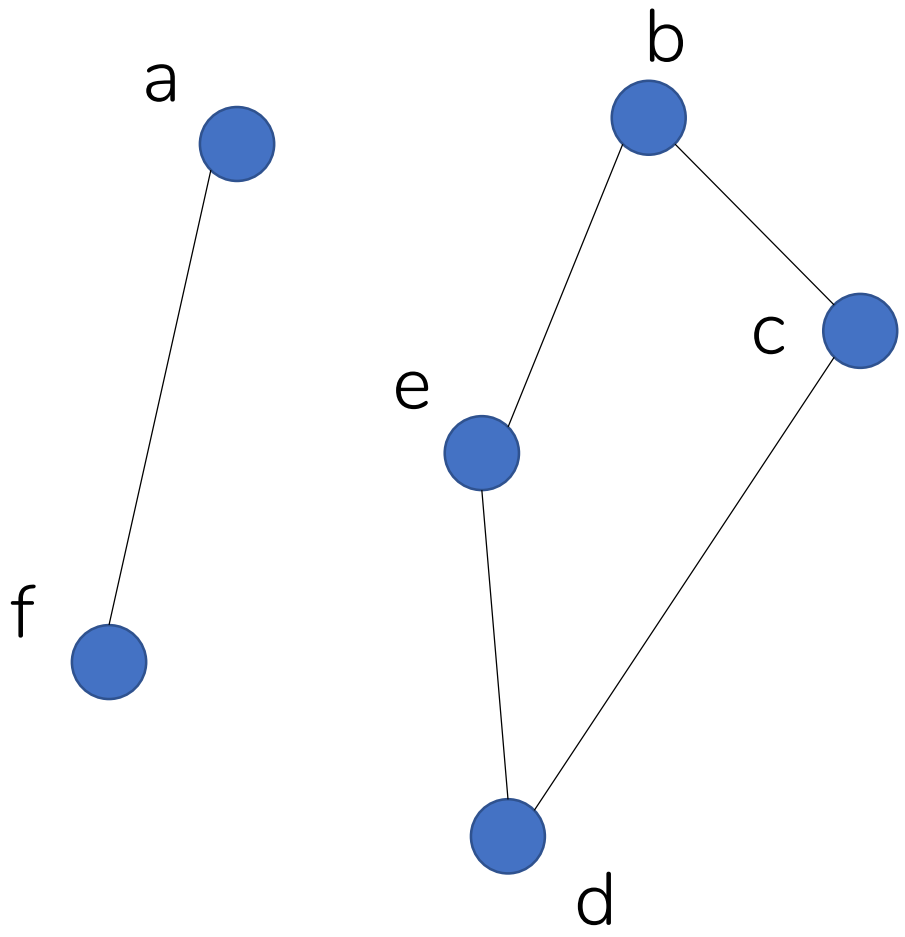



For graph $G = (V, E)$ to be connected,

$$|E| \geq |V| - 1$$

Q: Prove this

# Connected graph

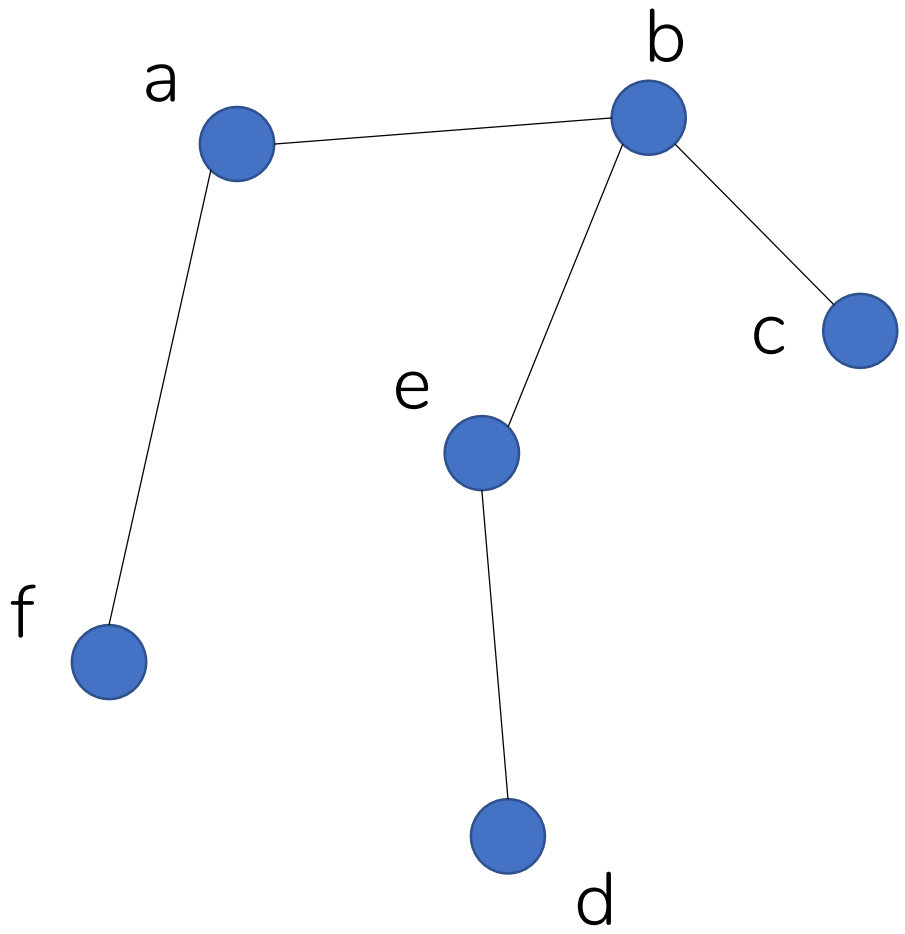



For graph $G = (V, E)$ to be connected,

$$|E| \geq |V| - 1$$

(necessary but not sufficient)

# Connected graph

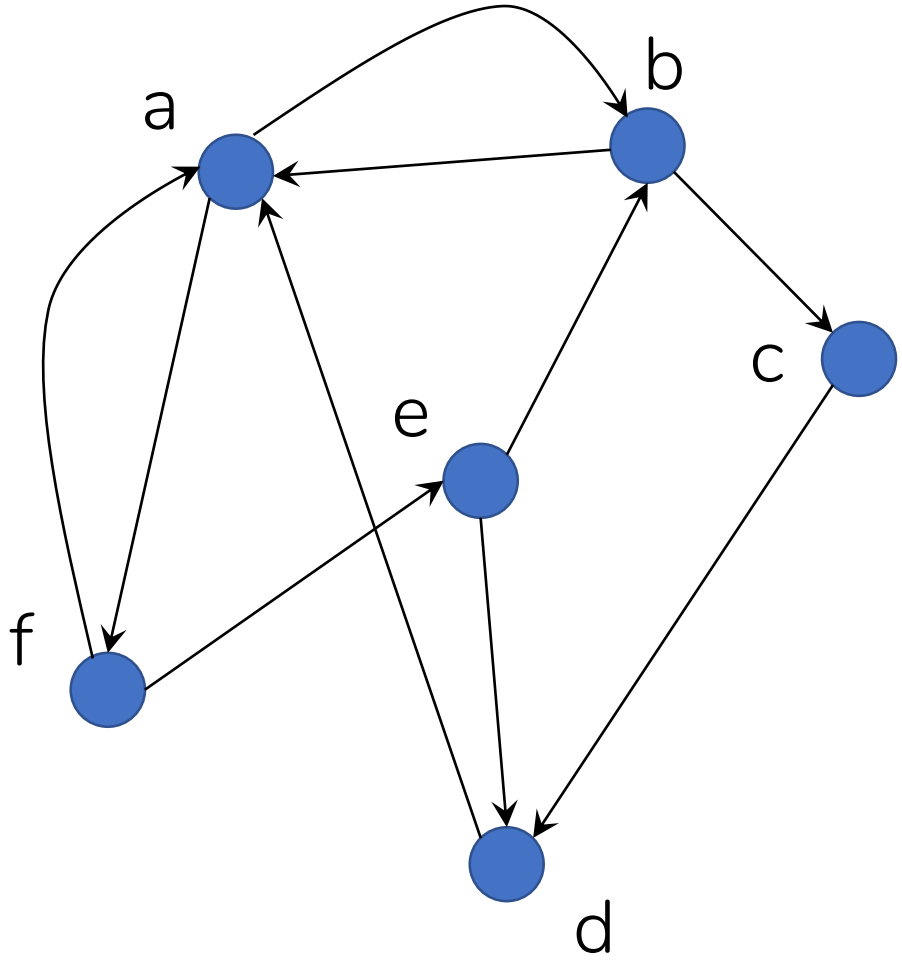



For graph $G = (V, E)$ to be connected,

$$|E| \geq |V| - 1$$

If $|E| = |V| - 1$ and graph is connected, edges form a spanning tree

# Directed Graphs




Directed graph

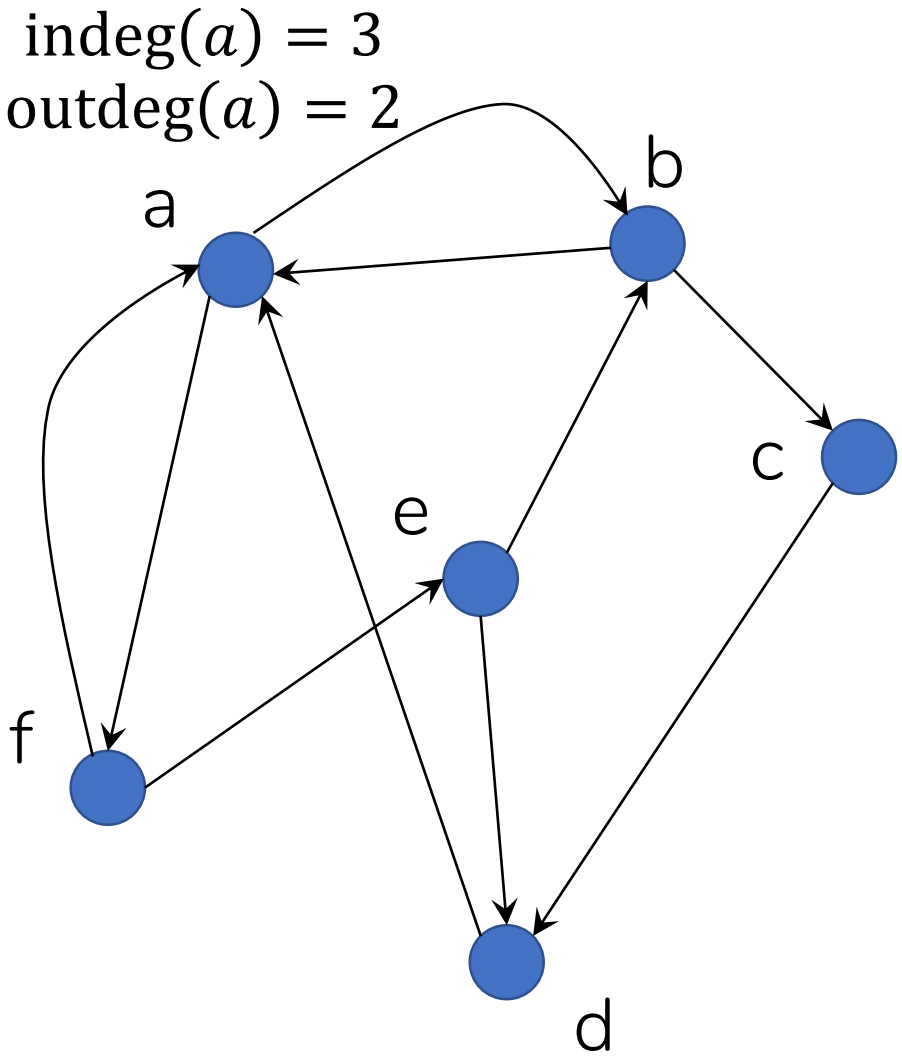
$G = (V, E)$

where $V = \{a, b, c, d, e, f\}$

and $E = \{(a, b), (b, a), (b, c), (d, a), \ldots\}$

(2 degrees: in-degree & out-degree)

# Directed Graphs




Directed graph

$G = (V, E)$

where $V = \{a, b, c, d, e, f\}$

and $E = \{(a, b), (b, a), (b, c), (d, a), \ldots\}$

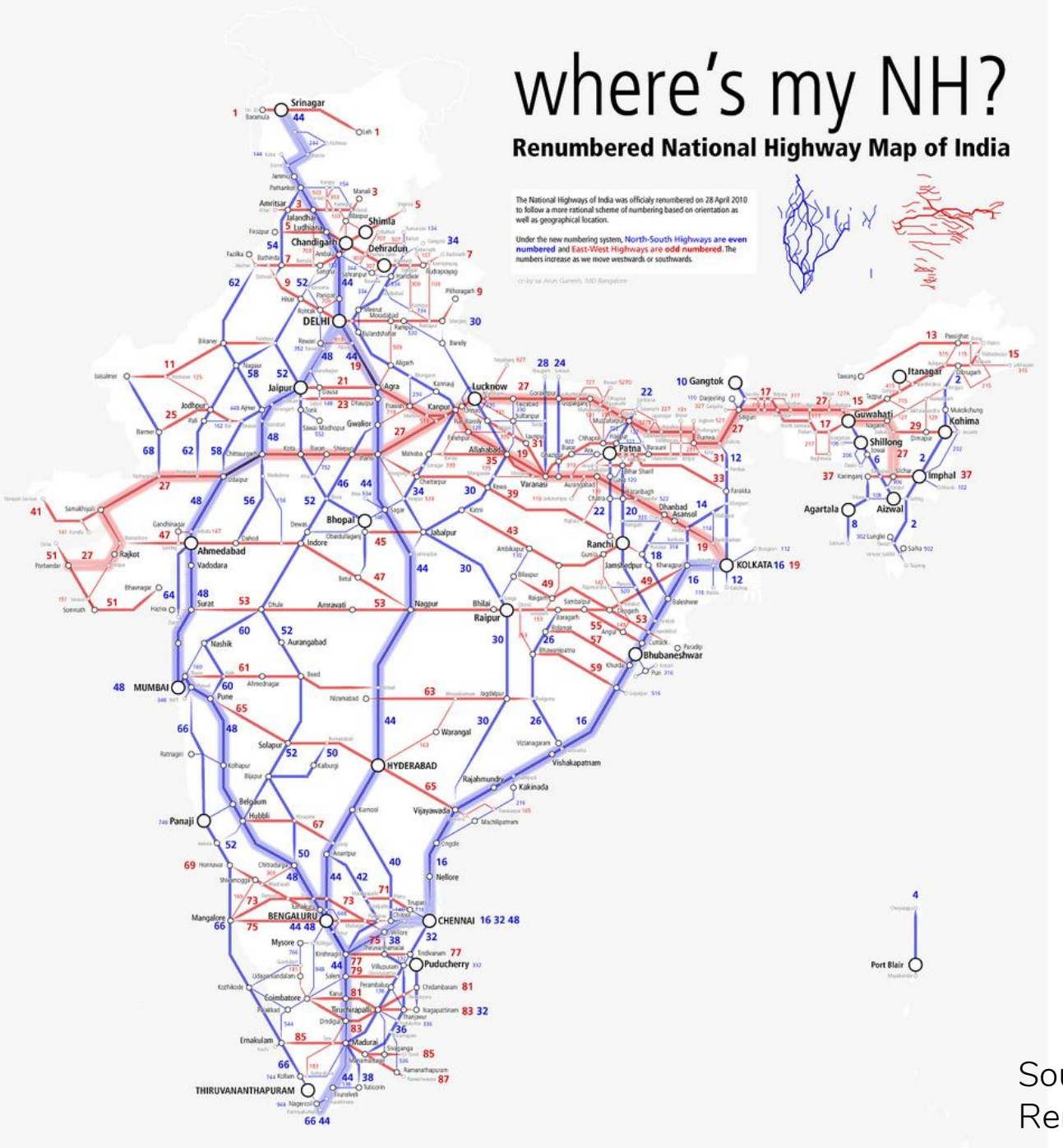
(2 degrees: in-degree & out-degree)

# Why Graphs?

Talks on graphs so far:

- Bipartite matching (scaling algorithm)
- Stable and popular matchings
- Colouring planar graphs

# Graphs are Everywhere

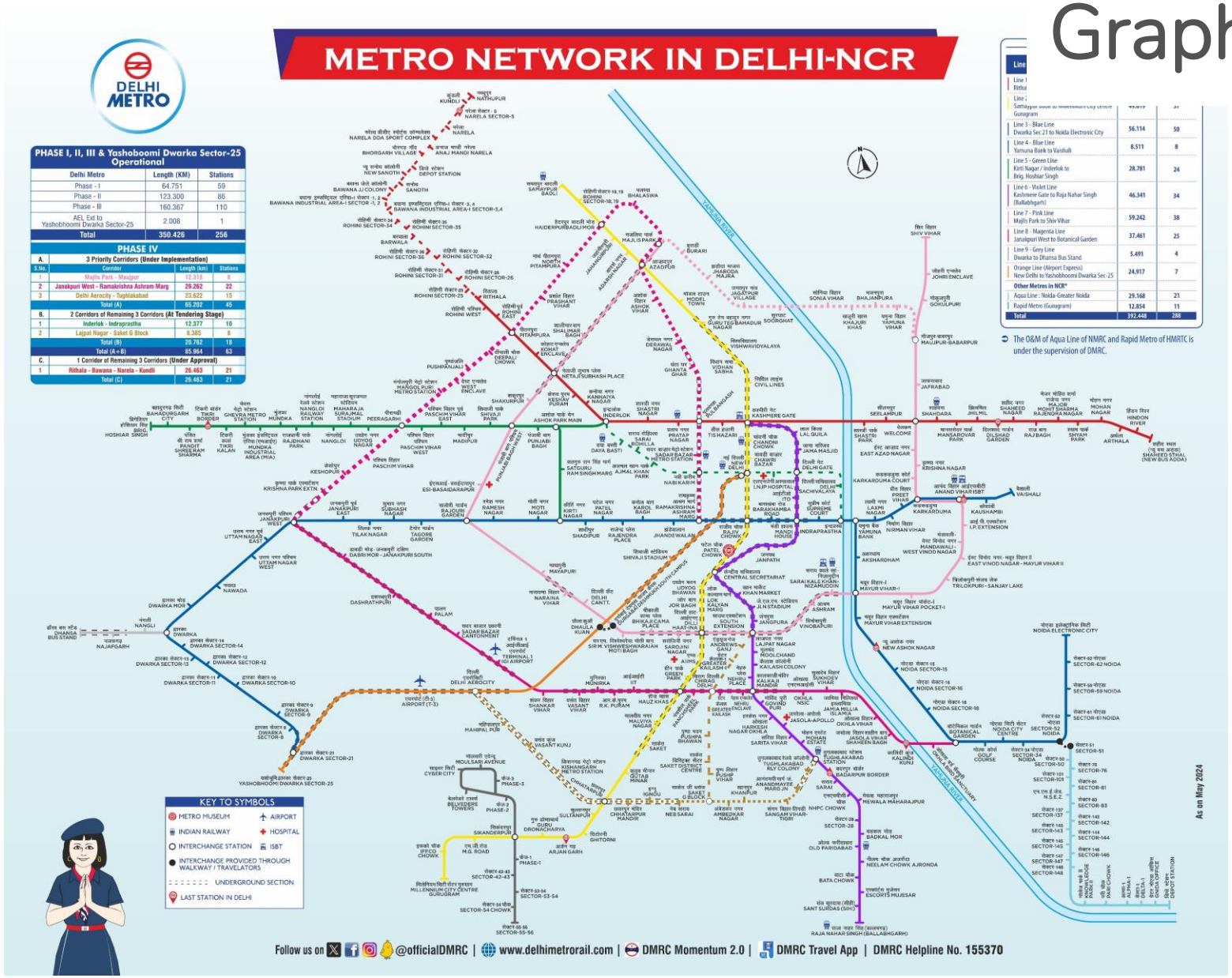



Roads

Source: Arun Ganesh, Flickr and Wikimedia,
Renumbered National Highways map of India (Schematic).jpg

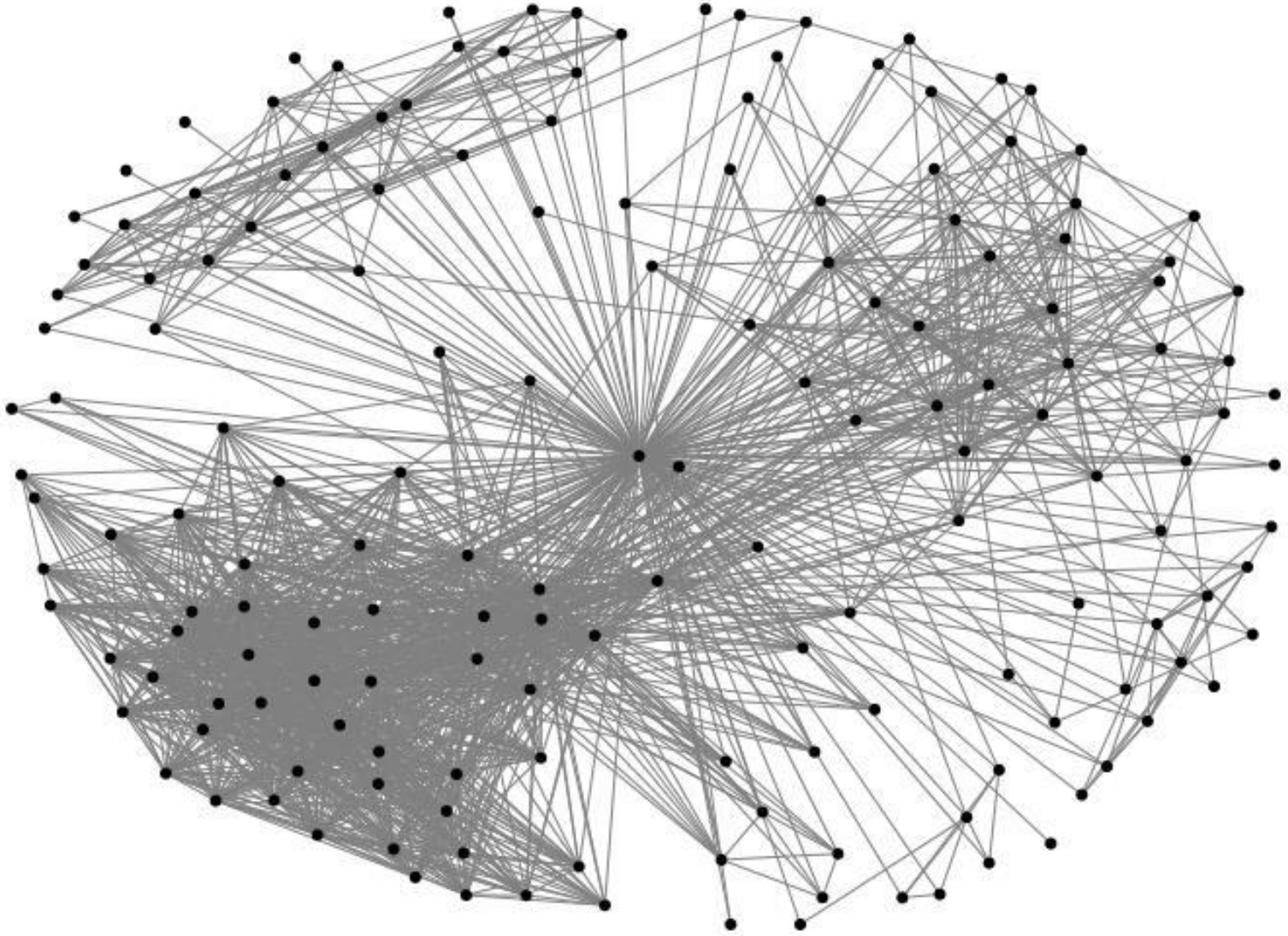
# Graphs are Everywhere

## Trains




Source: https://www.delhimetrorail.com/network_map

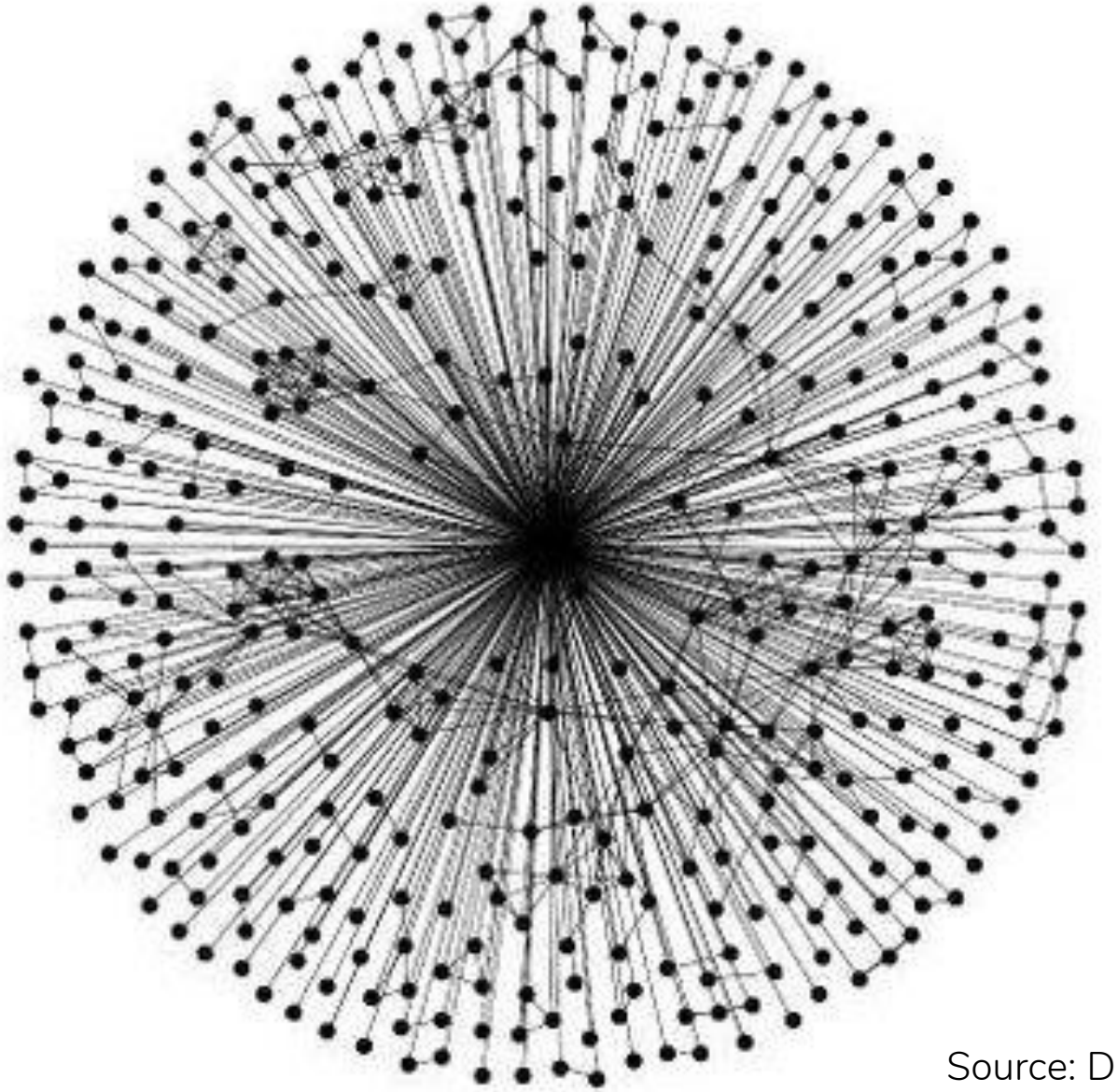
# Graphs are Everywhere



Social networks

Source: Audrey Austin, https://inbound.business.wayne.edu/blog/bid/277012/Facebook-Analysis-Using-Social-Network-Analysis

# Graphs are Everywhere



Scientific collaboration

Source: Drozdz et al., “Hierarchical organization of H. Eugene Stanley scientific collaboration community in weighted network representation,” Journal of Informetrics

# Graphs are Everywhere


Sports tournaments

Source: https://conceptdraw.com/a1997c3/preview , based on FIFA 2014 elimination rounds

# Graphs are Everywhere

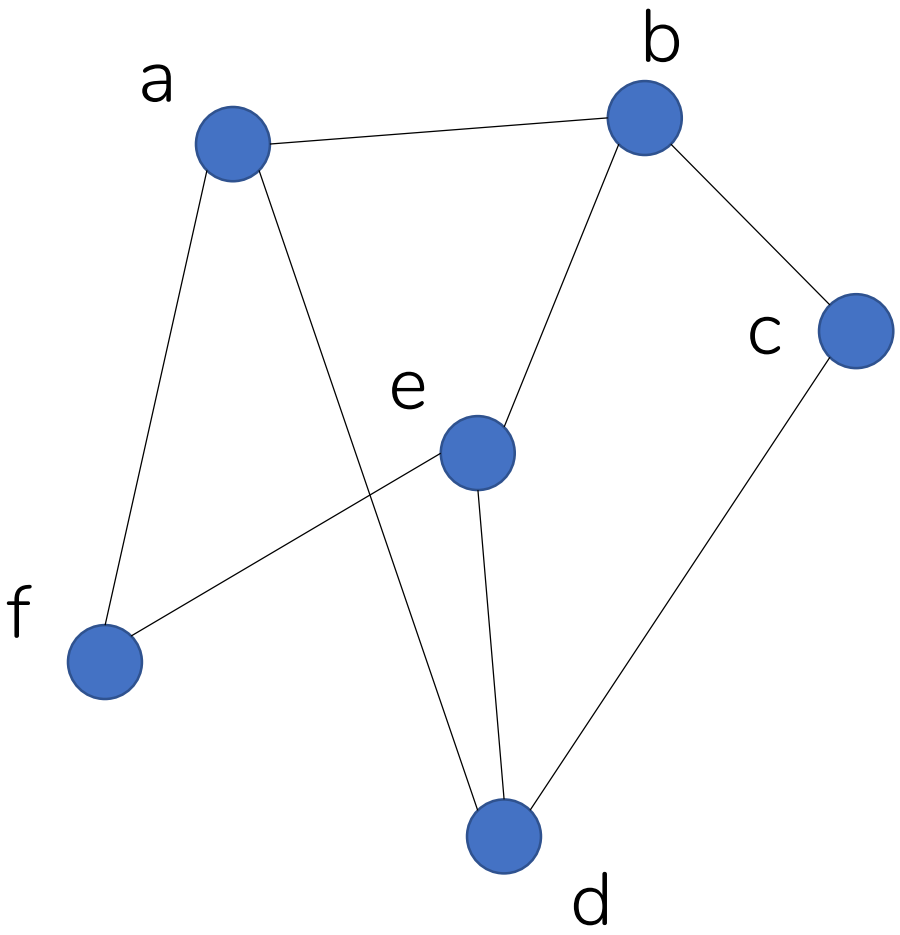
- electrical networks
- data networks
- holdings of mutual funds in stocks
- assigning students to colleges
- assigning time slots to courses, and courses to lecture halls
- ?

# Basic Problems in Graphs

- assignment: students to colleges; doctors to hospitals; etc.
- is a graph connected?
- find a path between two vertices
- find a shortest path between two vertices
- add edges to connect a set of vertices
- remove edges to disconnect a set of vertices
- ?

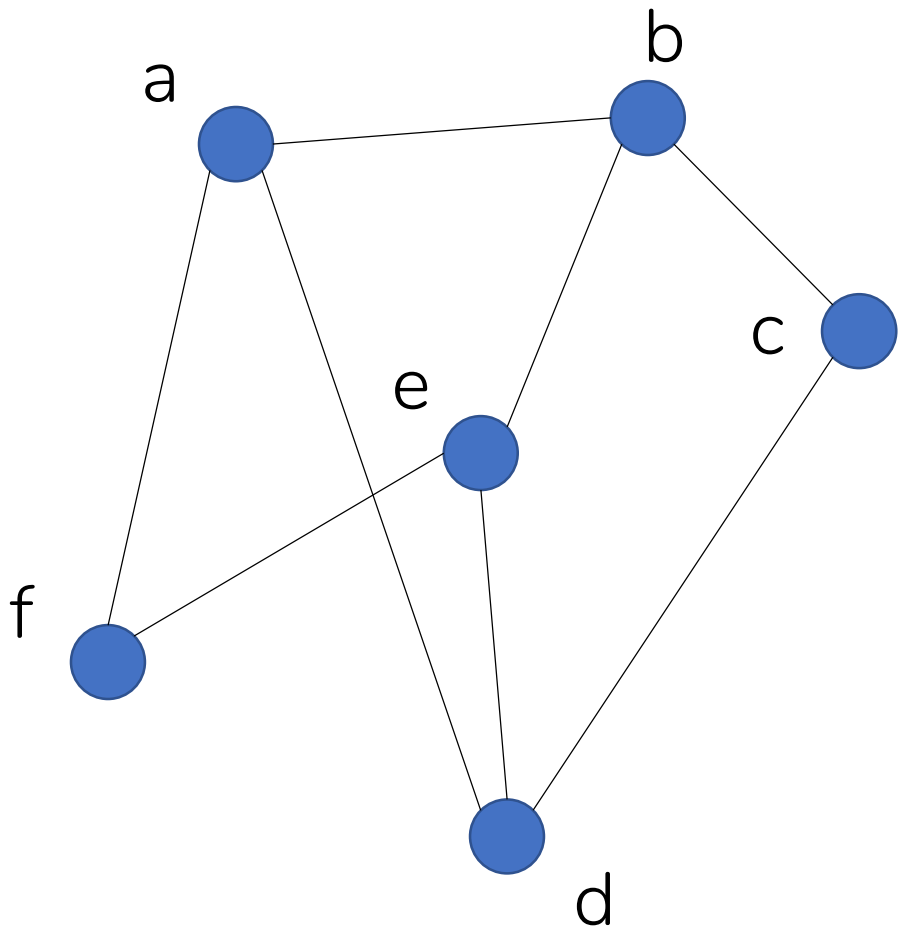
# Basic Algorithms

0. storing a graph:



# Basic Algorithms

0. storing a graph:



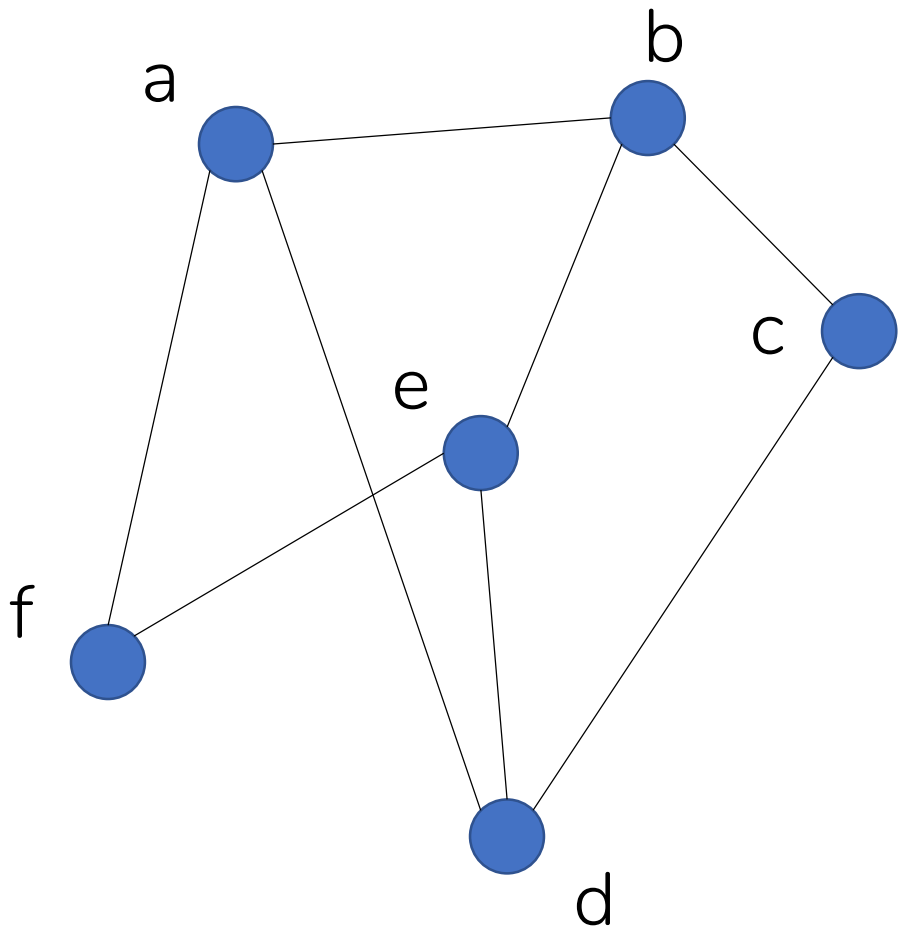

adjacency matrix:

| | a | b | c | d | e | f |
|---|---|---|---|---|---|---|
| a | | 1 | | 1 | | 1 |
| b | 1 | | 1 | | 1 | |
| c | | 1 | | 1 | | |
| d | 1 | | 1 | | 1 | |
| e | | 1 | | 1 | | 1 |
| f | 1 | | | | 1 | |

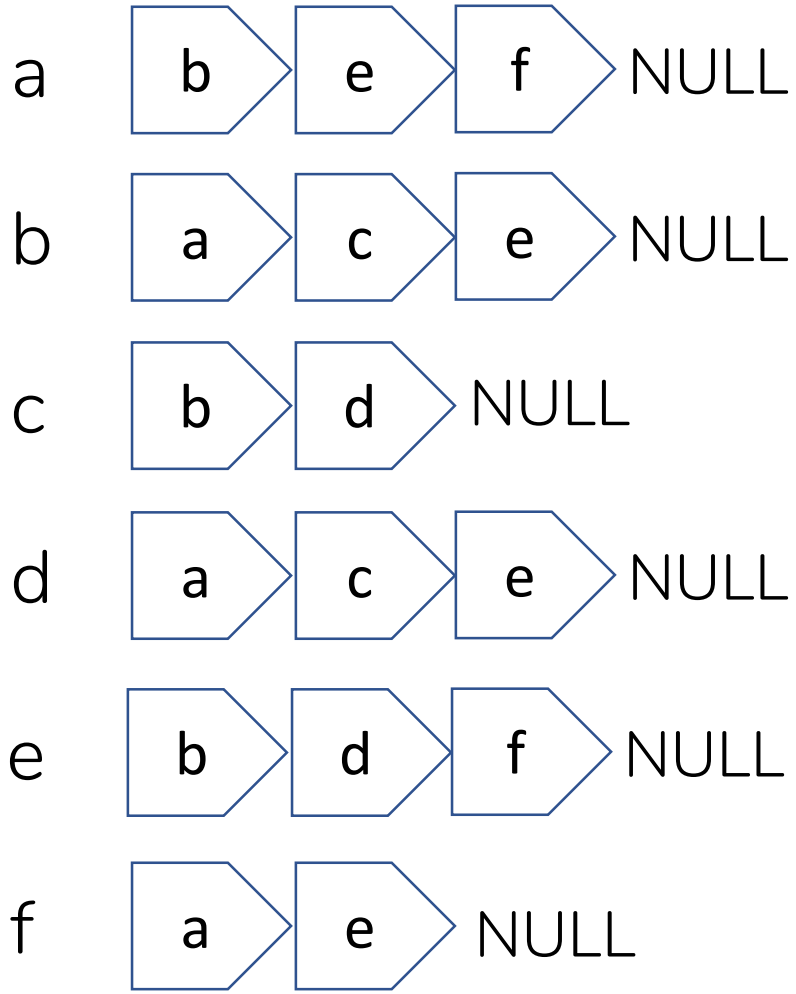
Size: $O(|V|^2)$

# Basic Algorithms

0. storing a graph:

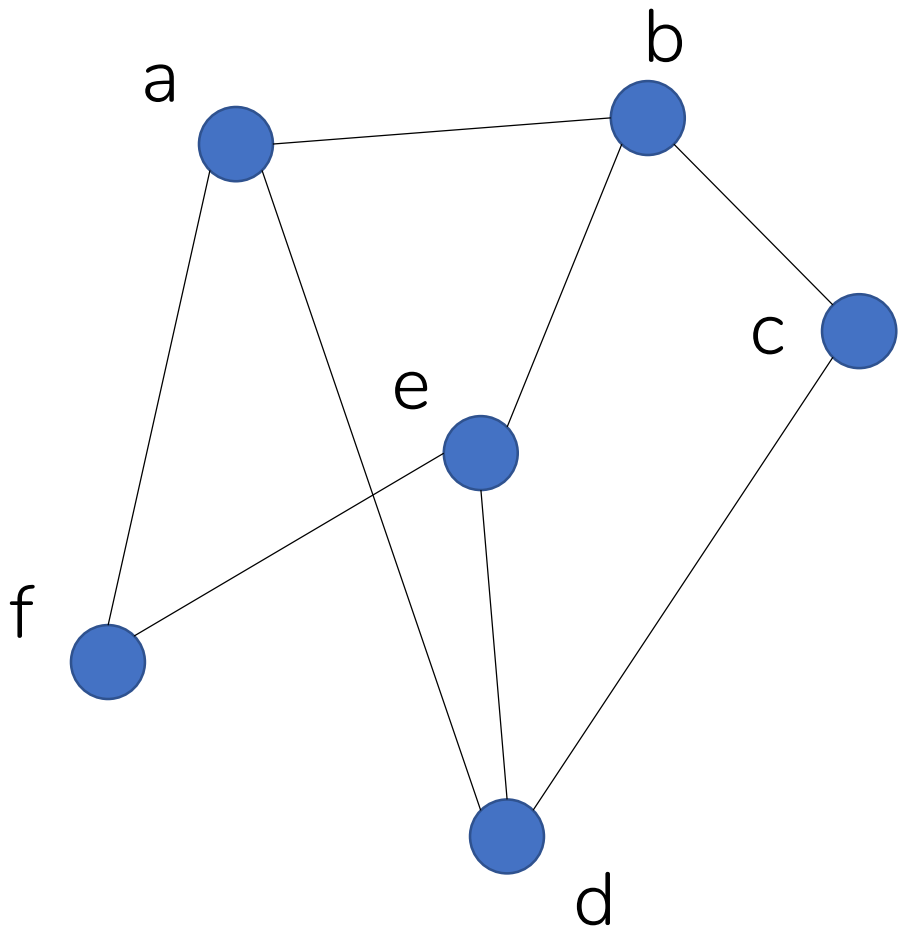



adjacency list:

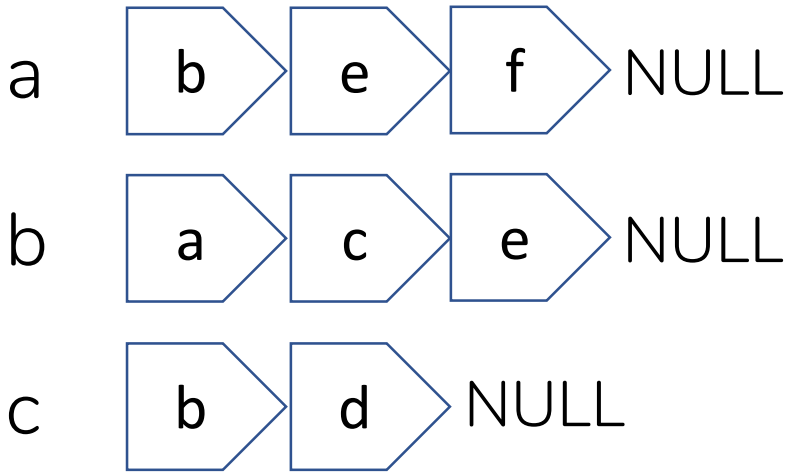


# Basic Algorithms

0. storing a graph:




adjacency list:

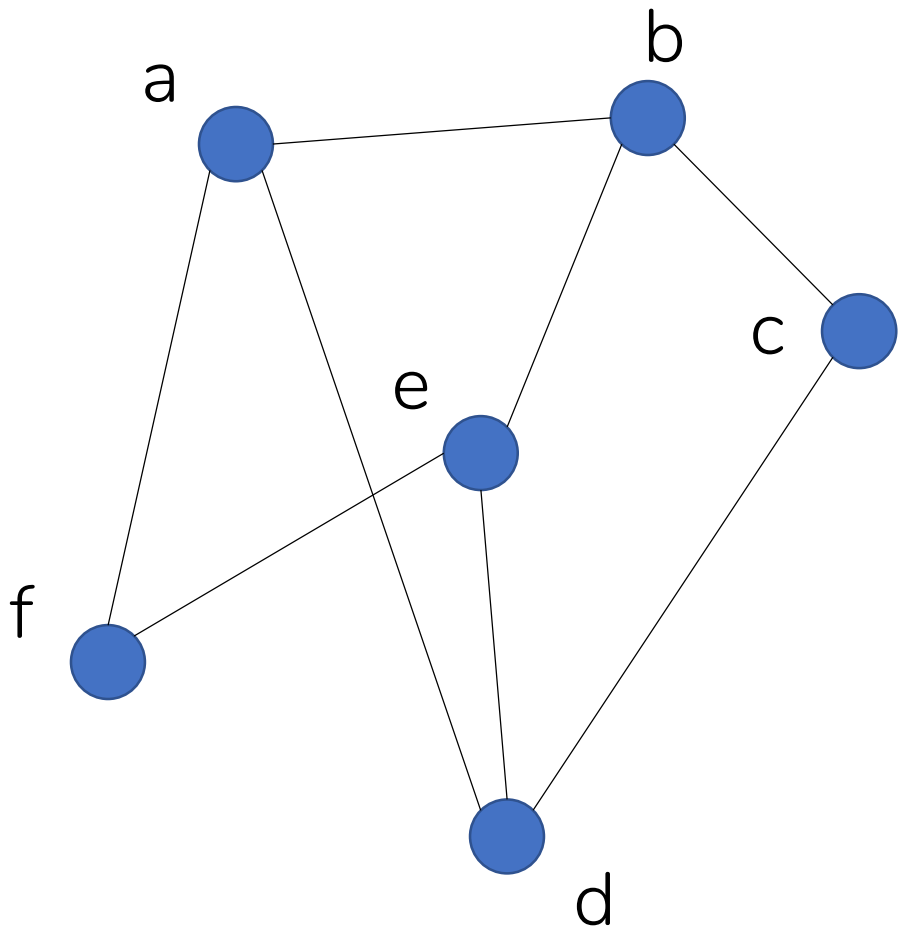



...

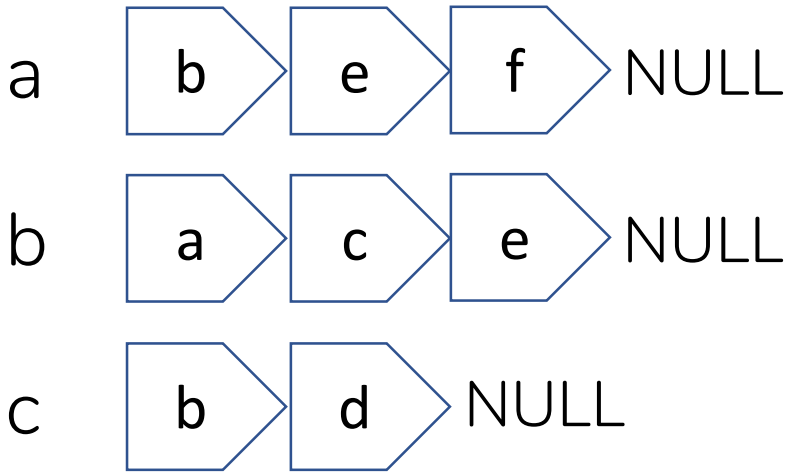
Size: $O(|V| + |E|)$

# Basic Algorithms

0. storing a graph:




adjacency list:

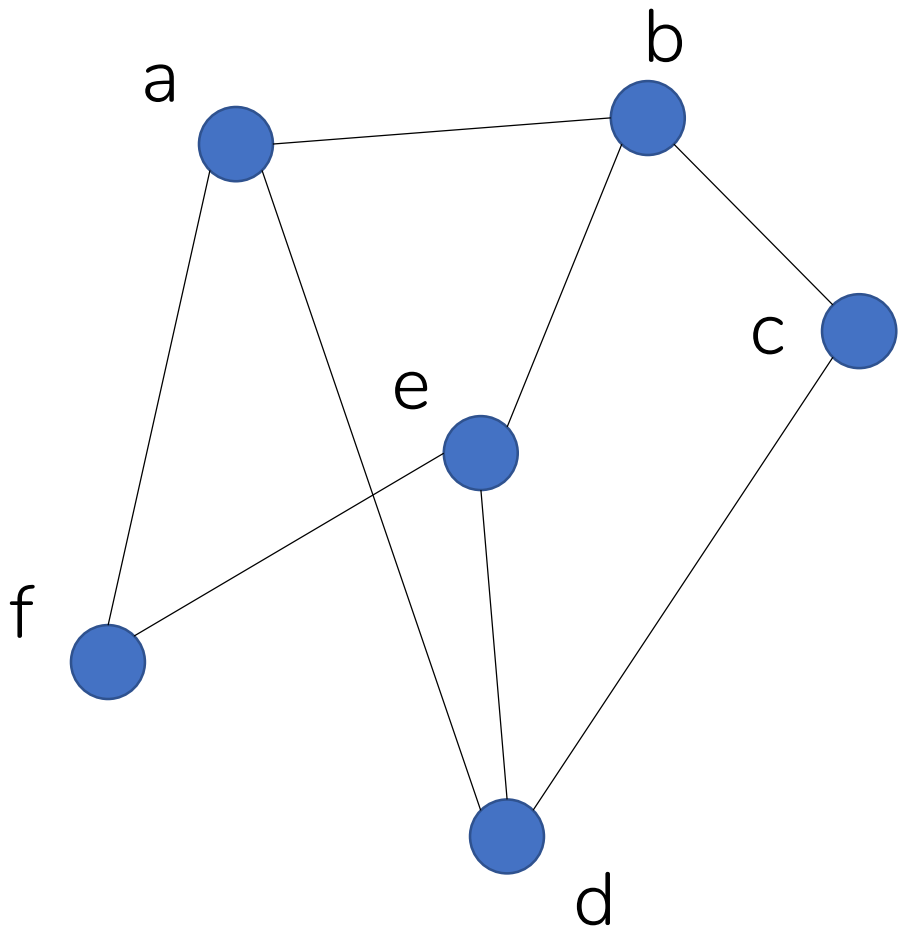



...

Size: $O(|V| + |E|)$

# Basic Algorithms

0. storing a graph:



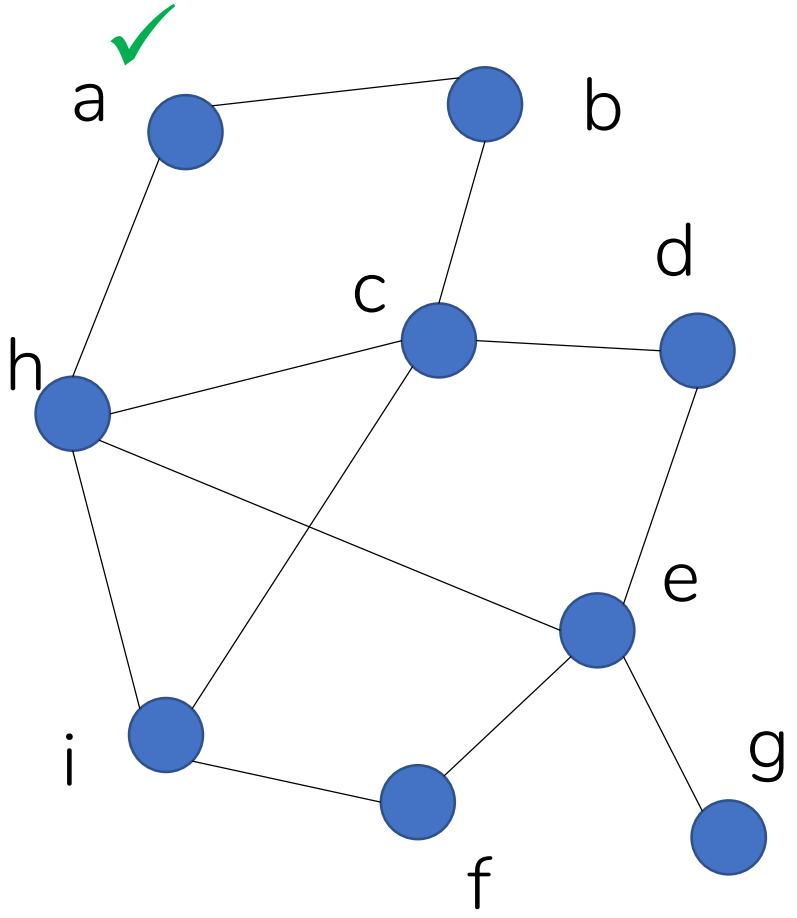

1. adjacency matrix

2. adjacency list

Questions?

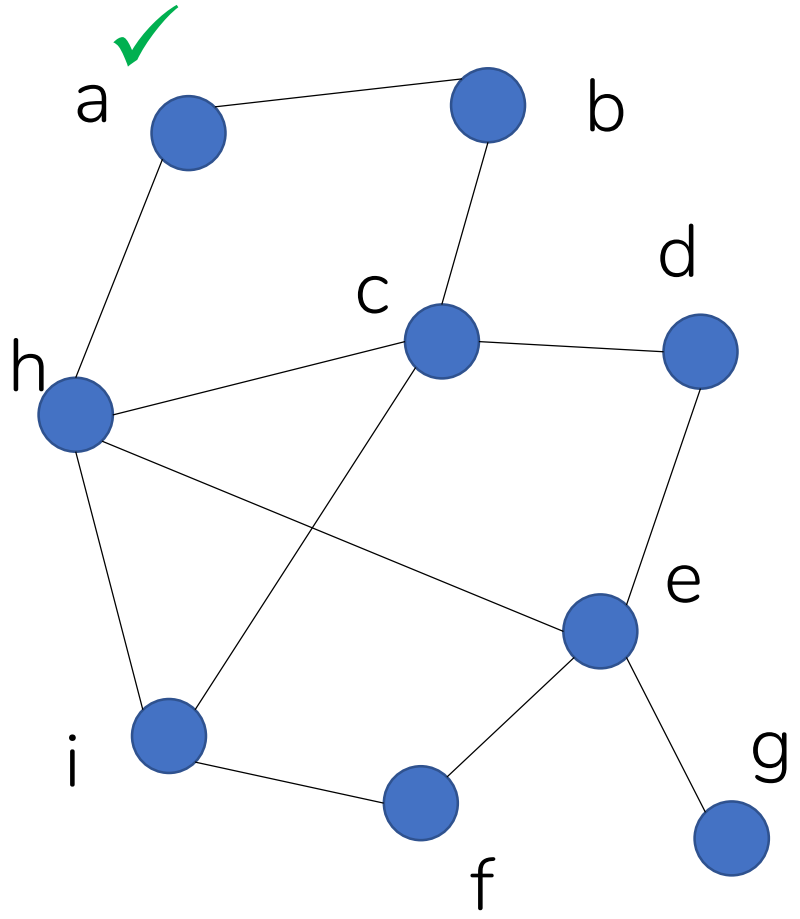
# Basic Algorithms

1. traversing a graph: list all nodes connected to a given node



# Basic Algorithms

1. traversing a graph: list all nodes connected to a given node

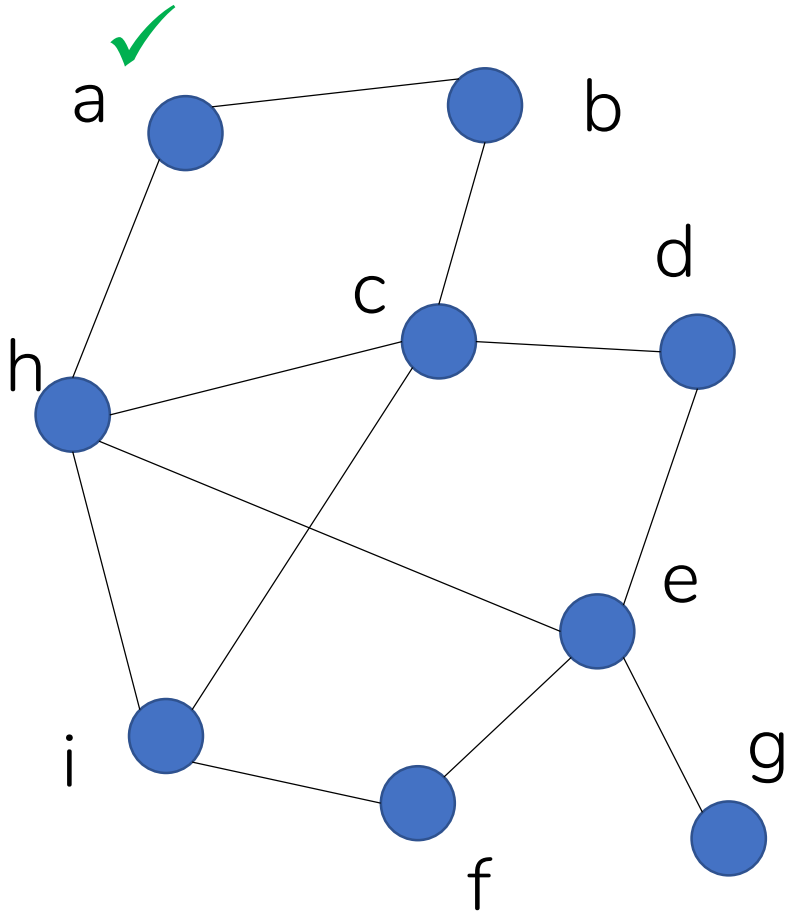



Breadth-First-Search ( $G = (V, E)$ , a)

Q:

# Basic Algorithms

1. traversing a graph: list all nodes connected to a given node

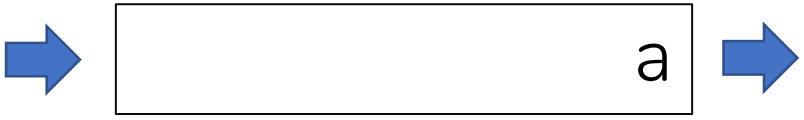



Breadth-First-Search ( $G = (V, E)$ , a)

---

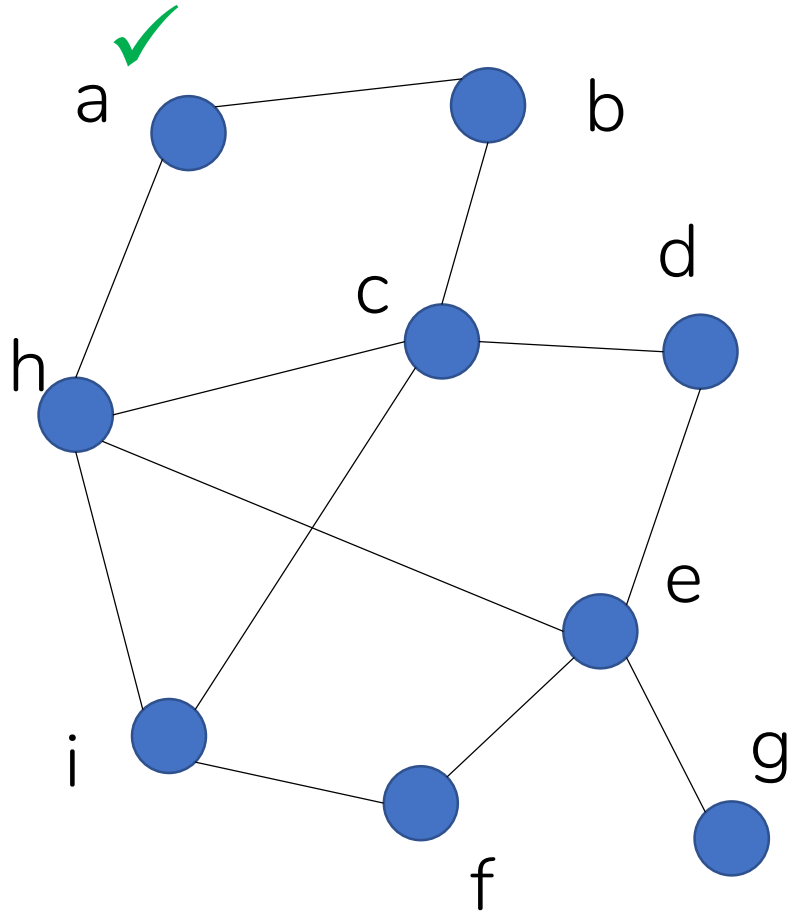
"visit" a; add a to Q

---

Q: 

# Basic Algorithms

1. traversing a graph: list all nodes connected to a given node




Breadth-First-Search ( $G = (V, E)$ , a)

---

"visit" a; add a to Q

while (Q is not empty)

pick a vertex v from Q

"visit" all unvisited neighbours of v

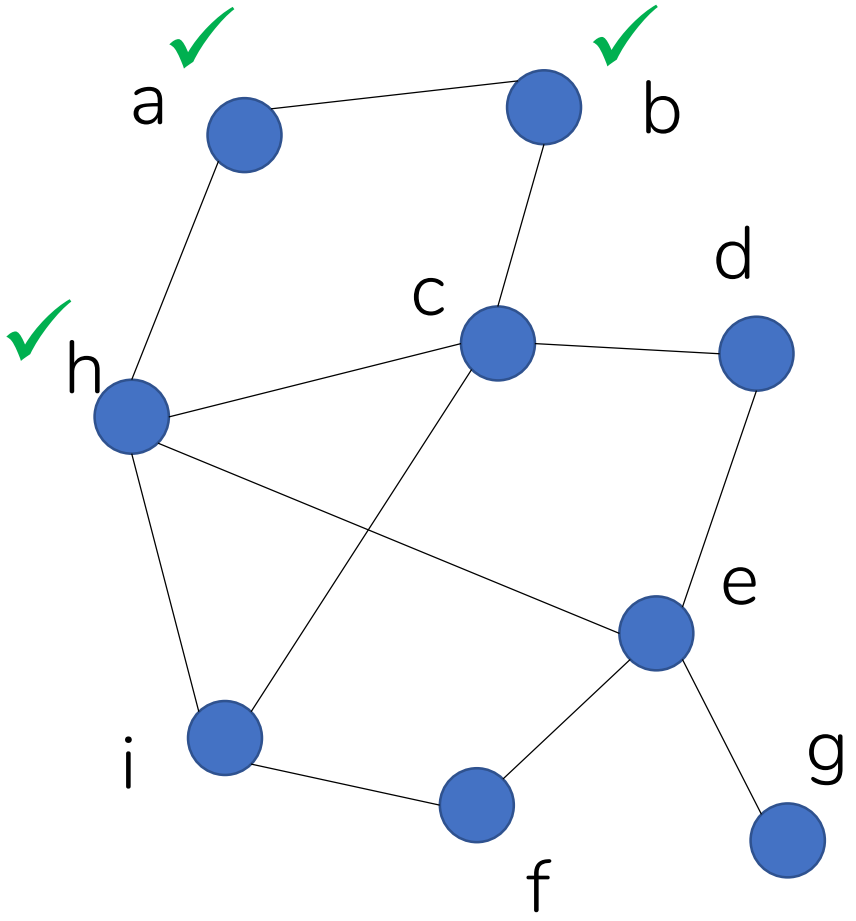
add these to Q

---

Q: → [ a ] →

# Basic Algorithms

1. traversing a graph: list all nodes connected to a given node




Breadth-First-Search ( $G = (V, E)$ , a)

---

"visit" a; add a to Q

while (Q is not empty)

pick a vertex v from Q

"visit" all unvisited neighbours of v

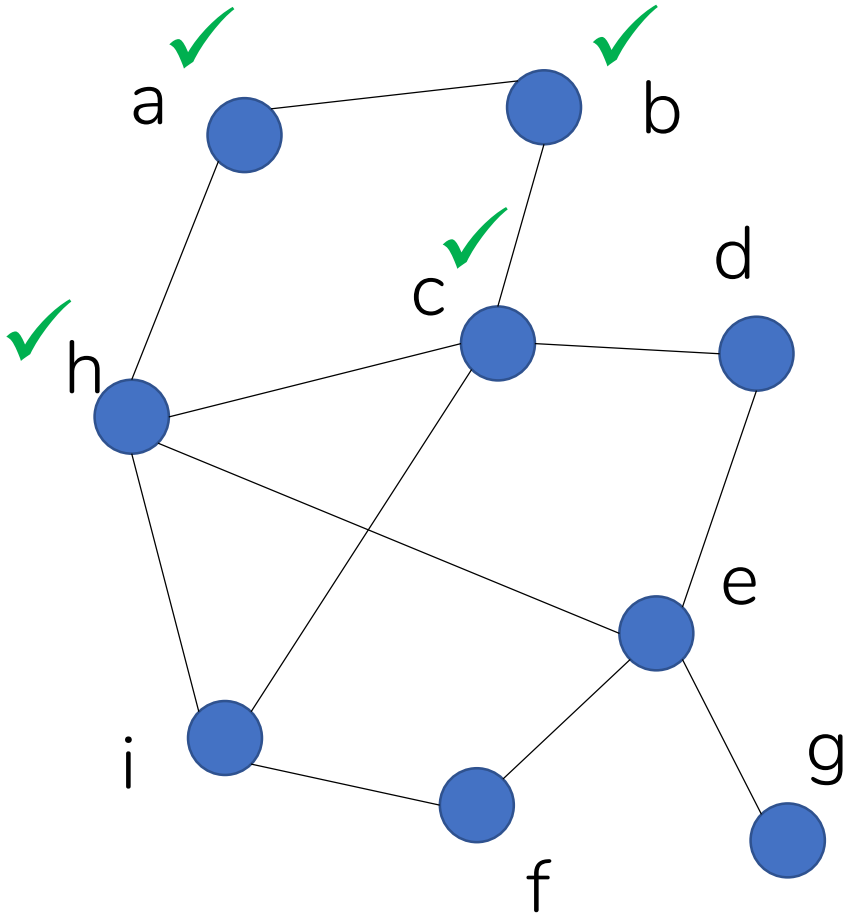
add these to Q

---

Q: ➡ h b ~~a~~ ➡

# Basic Algorithms

1. traversing a graph: list all nodes connected to a given node




Breadth-First-Search ( $G = (V, E)$ , a)

---

"visit" a; add a to Q

while (Q is not empty)

pick a vertex v from Q

"visit" all unvisited neighbours of v

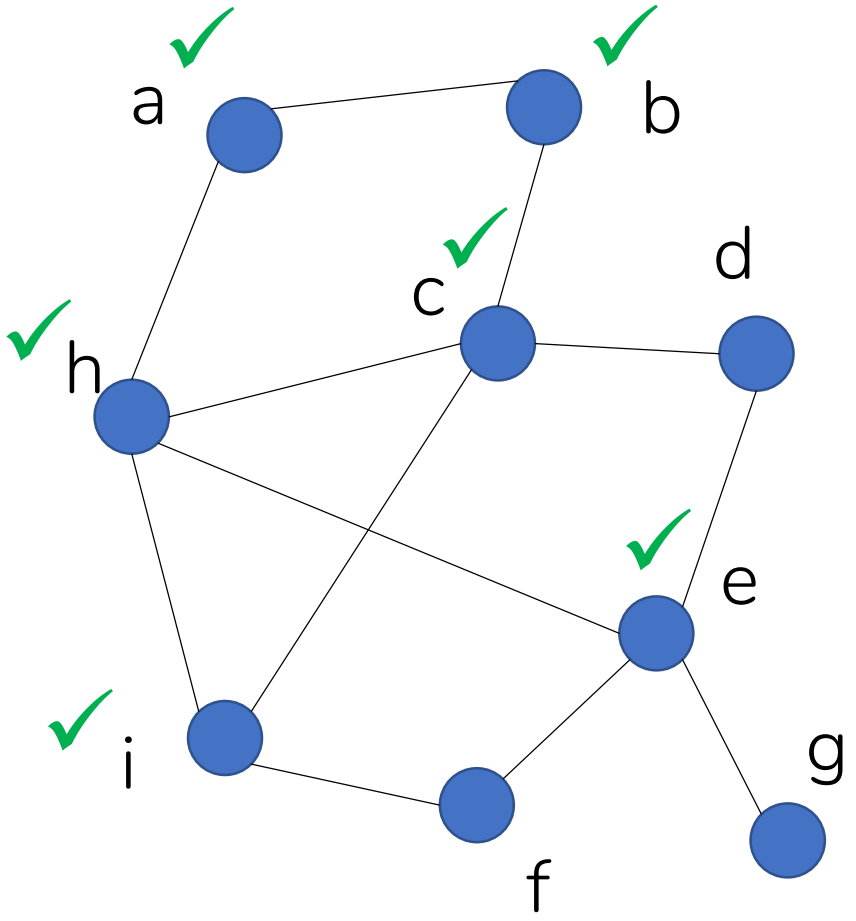
add these to Q

---

Q: ➡ c h ~~b a~~ ➡

# Basic Algorithms

1. traversing a graph: list all nodes connected to a given node




Breadth-First-Search ( $G = (V, E)$ , a)

---

"visit" a; add a to Q

while (Q is not empty)

pick a vertex v from Q

"visit" all unvisited neighbours of v

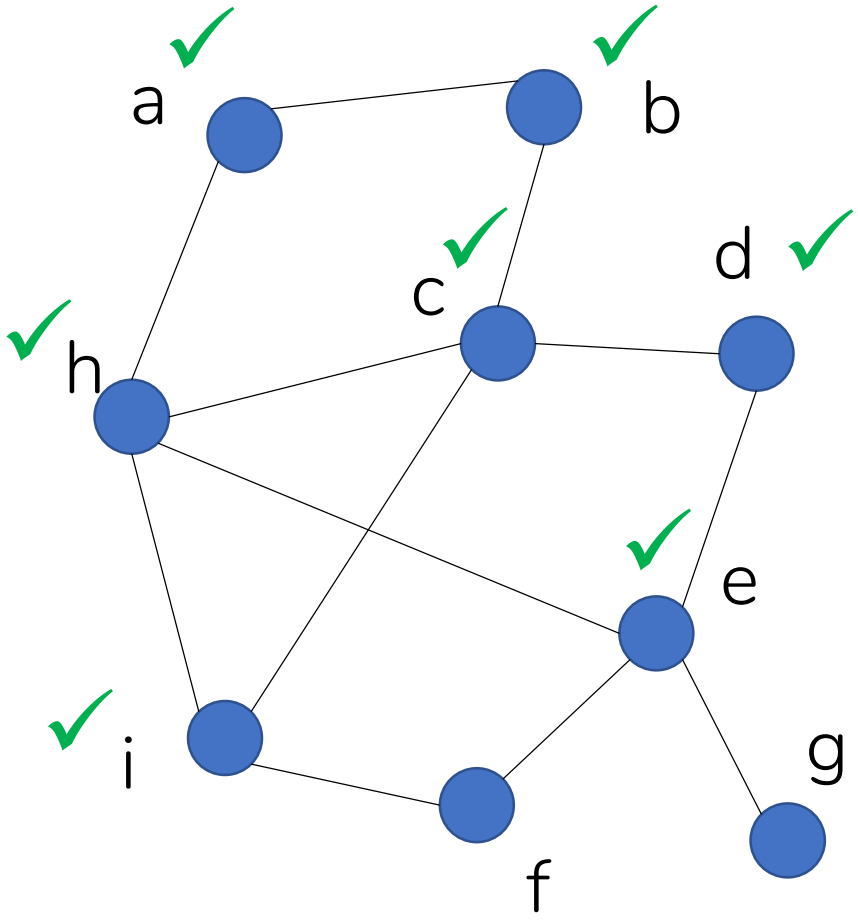
add these to Q

---

Q: e i c ~~h b a~~

# Basic Algorithms

1. traversing a graph: list all nodes connected to a given node




Breadth-First-Search ( $G = (V, E)$ , a)

---

"visit" a; add a to Q

while (Q is not empty)

pick a vertex v from Q

"visit" all unvisited neighbours of v

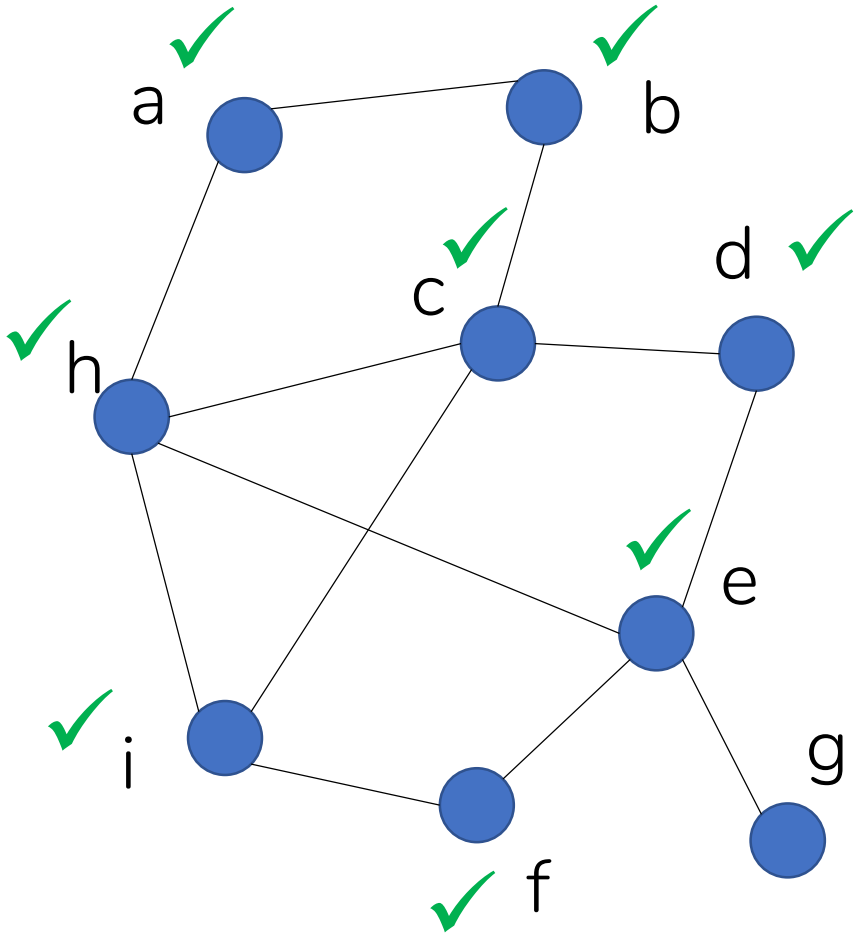
add these to Q

---

Q: ➡ d e i ~~c h b a~~ ➡

# Basic Algorithms

1. traversing a graph: list all nodes connected to a given node




Breadth-First-Search ( $G = (V, E)$ , a)

---

"visit" a; add a to Q

while (Q is not empty)

pick a vertex v from Q

"visit" all unvisited neighbours of v

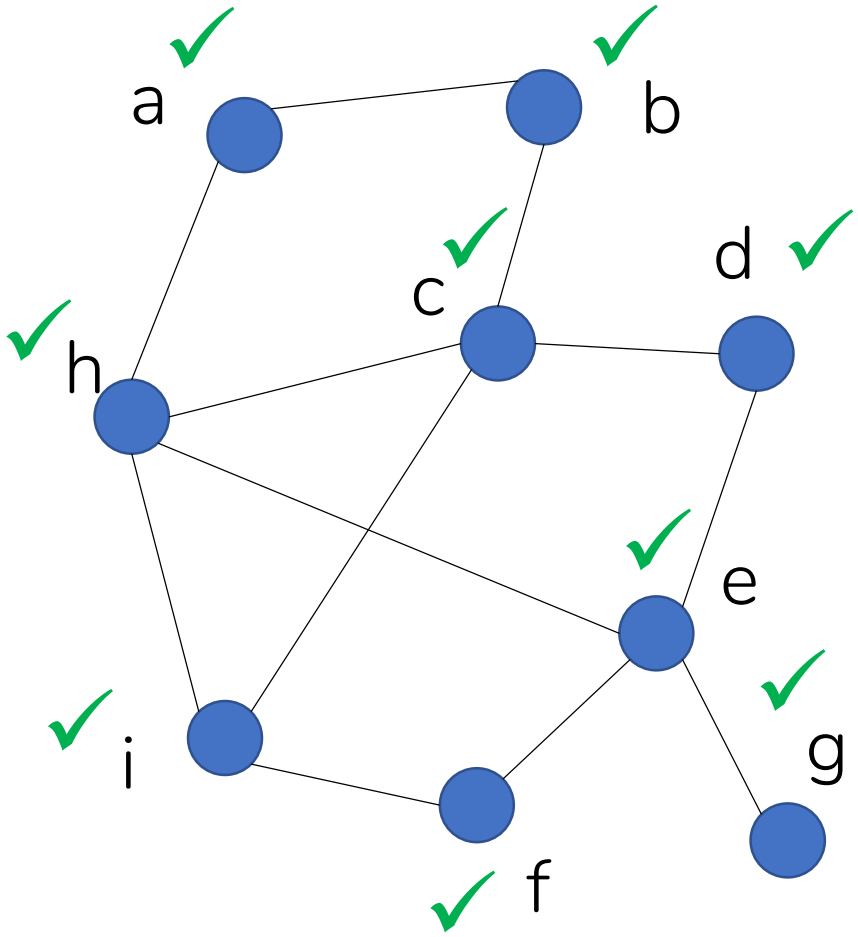
add these to Q

---

Q: ➡ f d e ~~i c h b a~~ ➡

# Basic Algorithms

1. traversing a graph: list all nodes connected to a given node




Breadth-First-Search ( $G = (V, E)$ , a)

---

"visit" a; add a to Q

while (Q is not empty)

pick a vertex v from Q

"visit" all unvisited neighbours of v

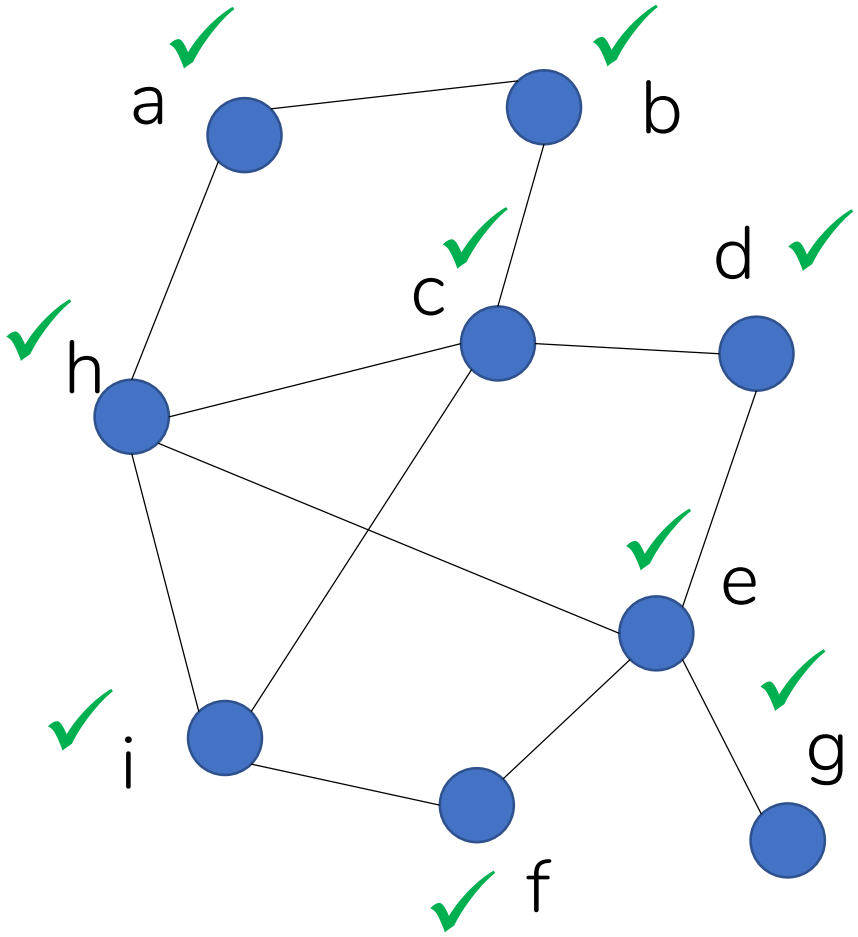
add these to Q

---

Q: ➡ g f d ~~e i c h b a~~ ➡

# Basic Algorithms

1. traversing a graph: list all nodes connected to a given node




Breadth-First-Search ( $G = (V, E)$ , a)

---

"visit" a; add a to Q

while (Q is not empty)

pick a vertex v from Q

"visit" all unvisited neighbours of v

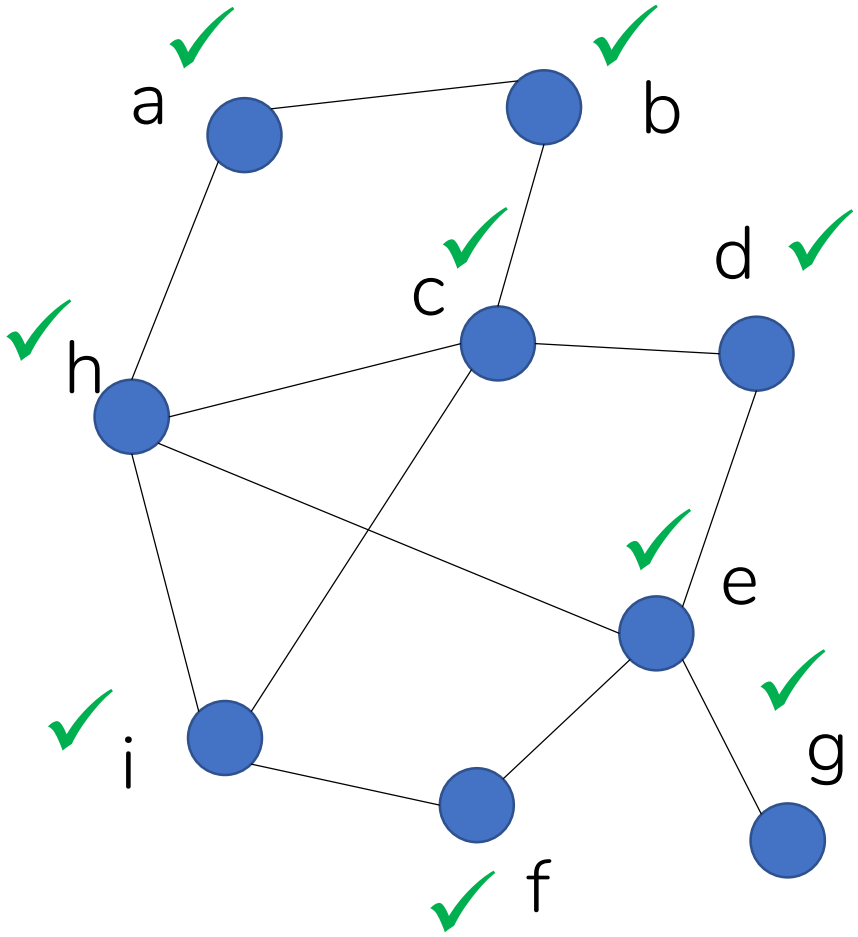
add these to Q

---

Q: ➡ ~~g f d e i c h b a~~ ➡

# Basic Algorithms

1. traversing a graph: list all nodes connected to a given node




Breadth-First-Search ( $G = (V, E)$ , a)

---

"visit" a; add a to Q

while (Q is not empty)

pick a vertex v from Q

"visit" all unvisited neighbours of v

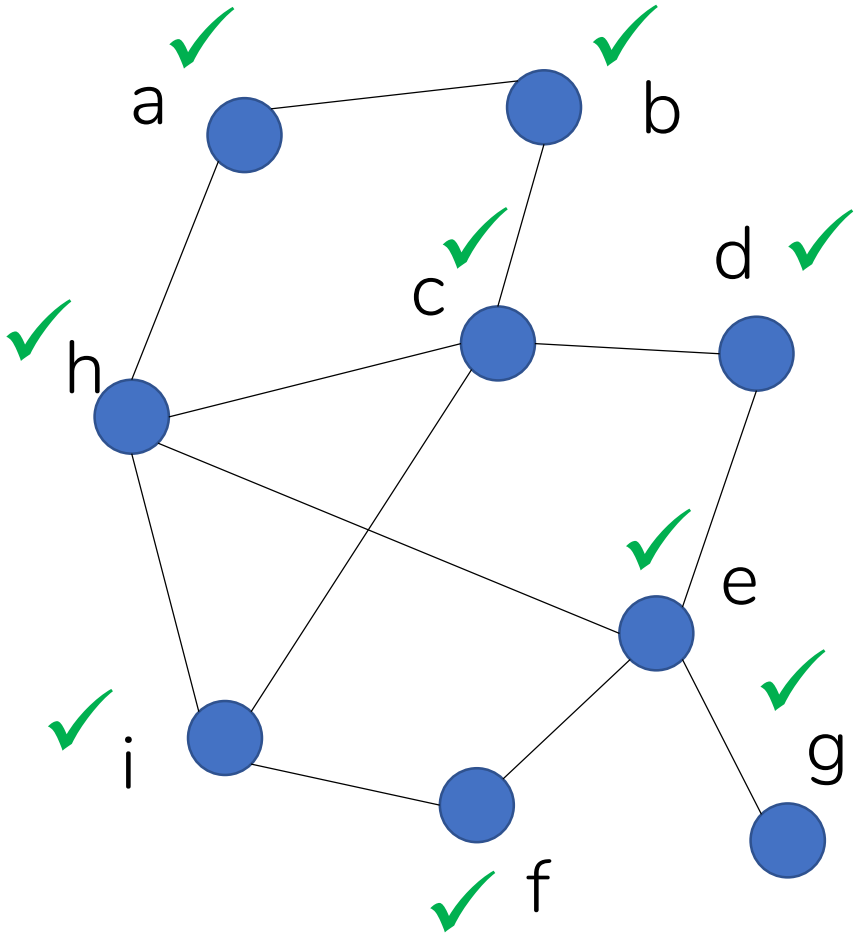
add these to Q

---

Q: Time taken?

# Basic Algorithms

1. traversing a graph: list all nodes connected to a given node




Breadth-First-Search ( $G = (V, E)$ , a)

---

"visit" a; add a to Q

while (Q is not empty)

pick a vertex v from Q

"visit" all unvisited neighbours of v

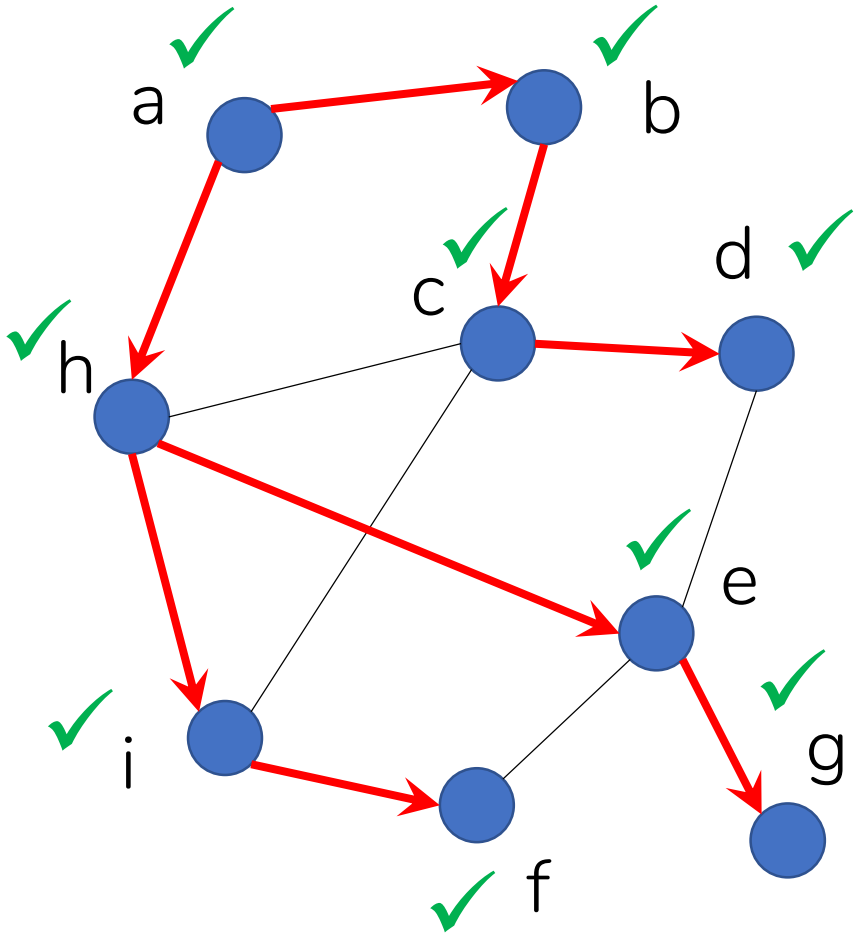
add these to Q (and add edge from v)

---

Q: ➡ ~~g f d e i c h b a~~ ➡

# Basic Algorithms

1. traversing a graph: list all nodes connected to a given node




Breadth-First-Search ( $G = (V, E)$ , a)

---

"visit" a; add a to Q

while (Q is not empty)

pick a vertex v from Q

"visit" all unvisited neighbours of v

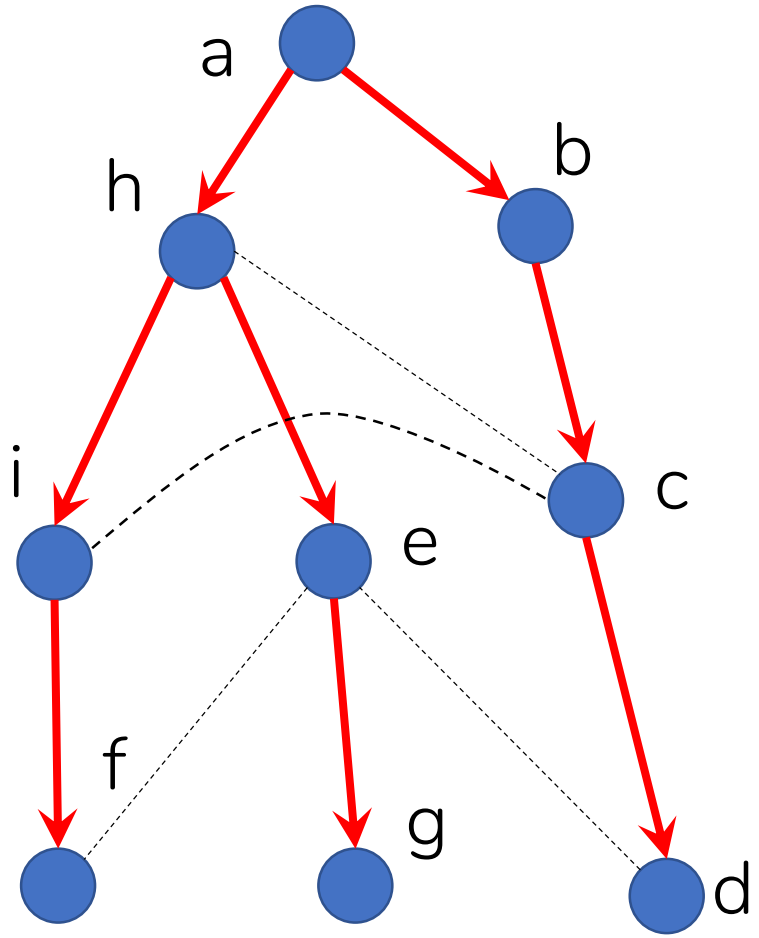
add these to Q (and add edge from v)

---

Q: ➡ ~~g f d e i c h b a~~ ➡

# Basic Algorithms

1. traversing a graph: list all nodes connected to a given node




Breadth-First-Search ( $G = (V, E)$ , a)

---

"visit" a; add a to Q

while (Q is not empty)

pick a vertex v from Q

"visit" all unvisited neighbours of v

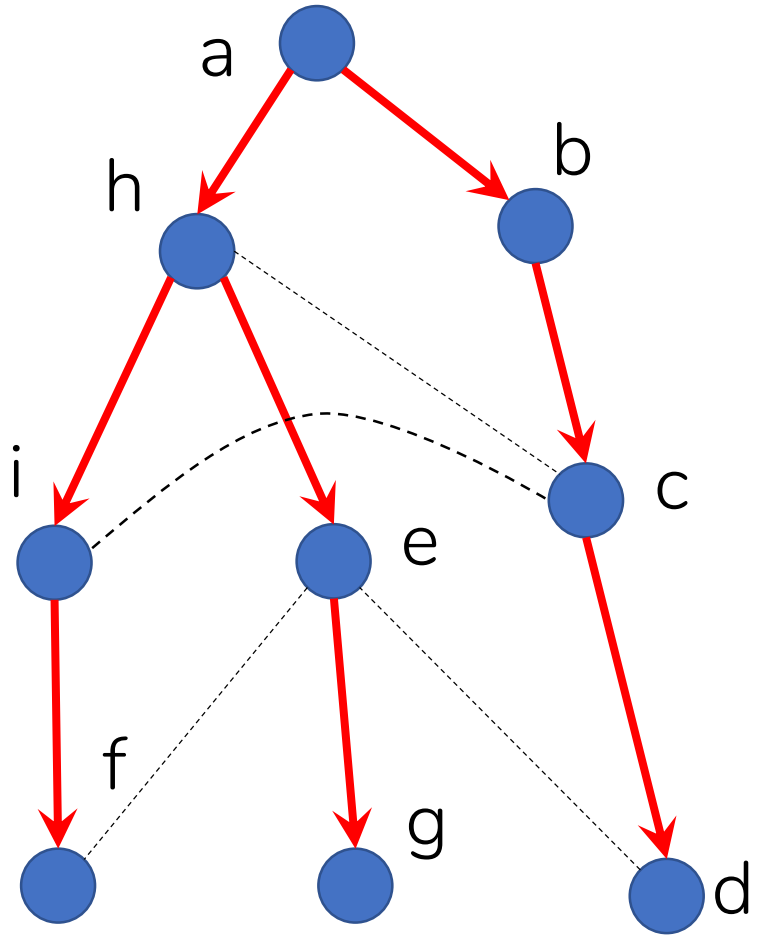
add these to Q (and add edge from v)

---

Q: ➡ ~~g f d e i c h b a~~ ➡

# Basic Algorithms

1. traversing a graph: list all nodes connected to a given node

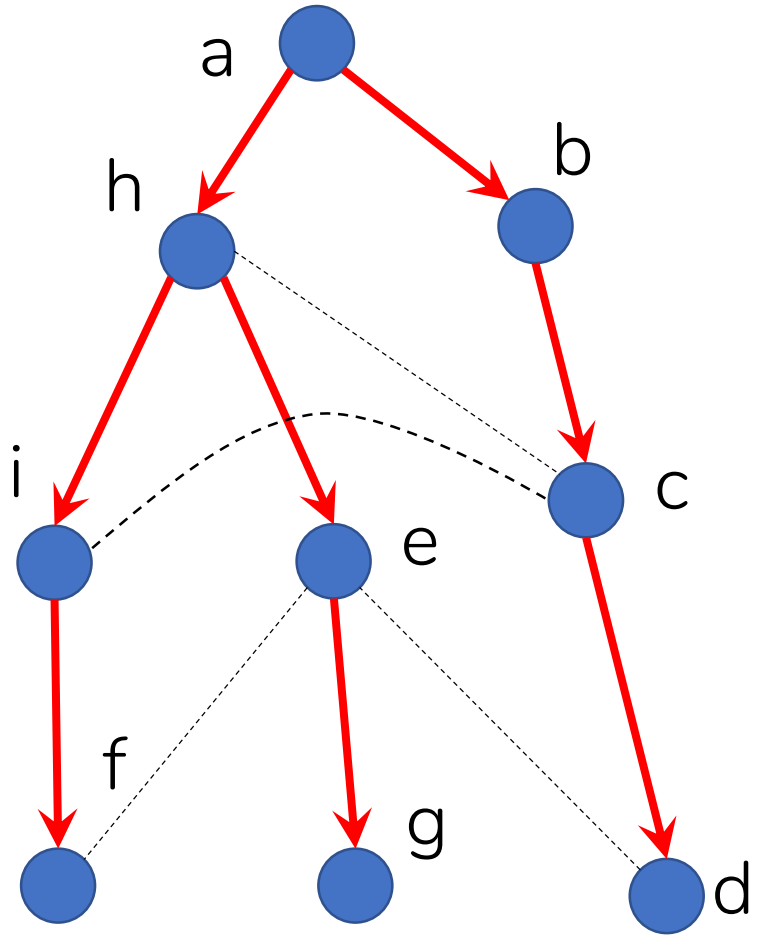



Breadth-First-Search ( $G = (V, E)$ , a)

- set of (undirected) edges used to "visit" nodes gives a tree
- all non-tree edges are between vertices:
  - on the same level, or
  - at a difference of one level

# Basic Algorithms

1. traversing a graph: list all nodes connected to a given node



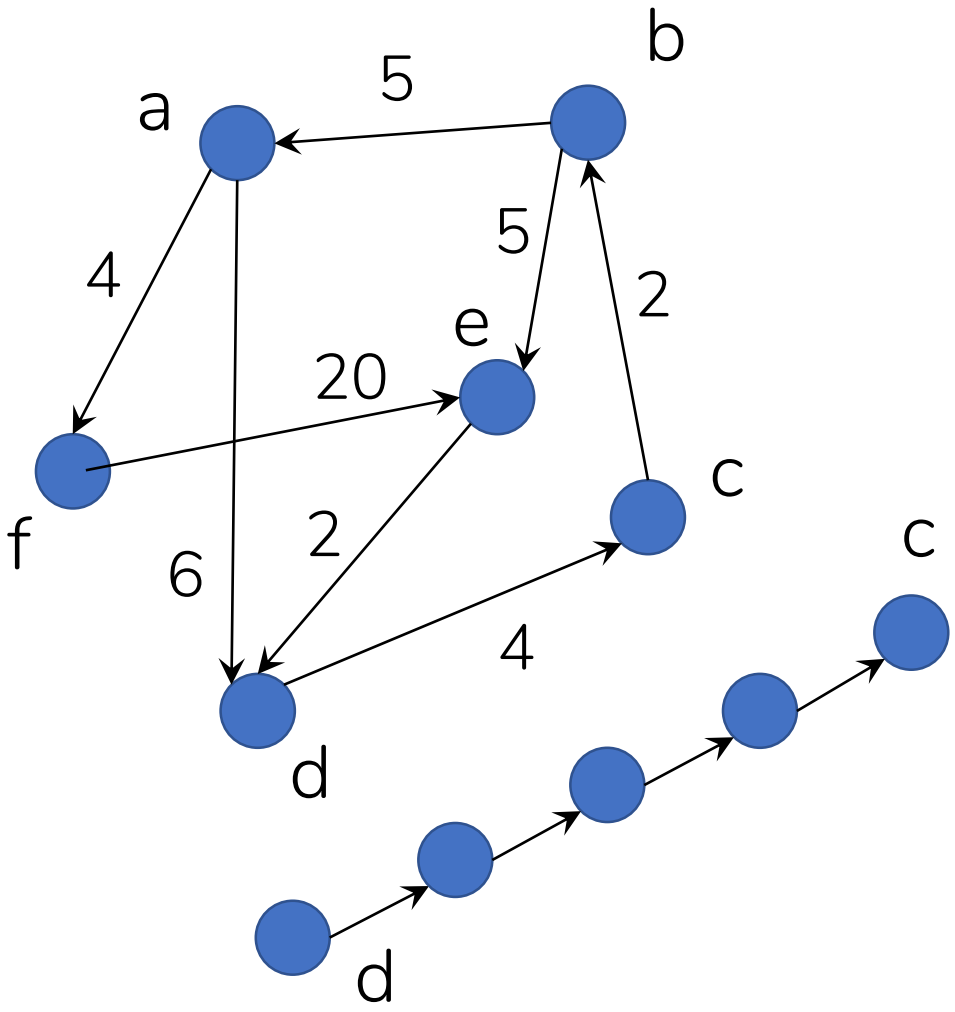

Breadth-First-Search ( $G = (V, E)$ , a)

- set of (undirected) edges used to "visit" nodes gives a tree
- all non-tree edges are between vertices:
  - on the same level, or
  - at a difference of one level

Thm: For any vtx v,
the path from a along tree edges
is a shortest path in $G$

# Basic Algorithms

1. traversing a graph: list all nodes connected to a given node

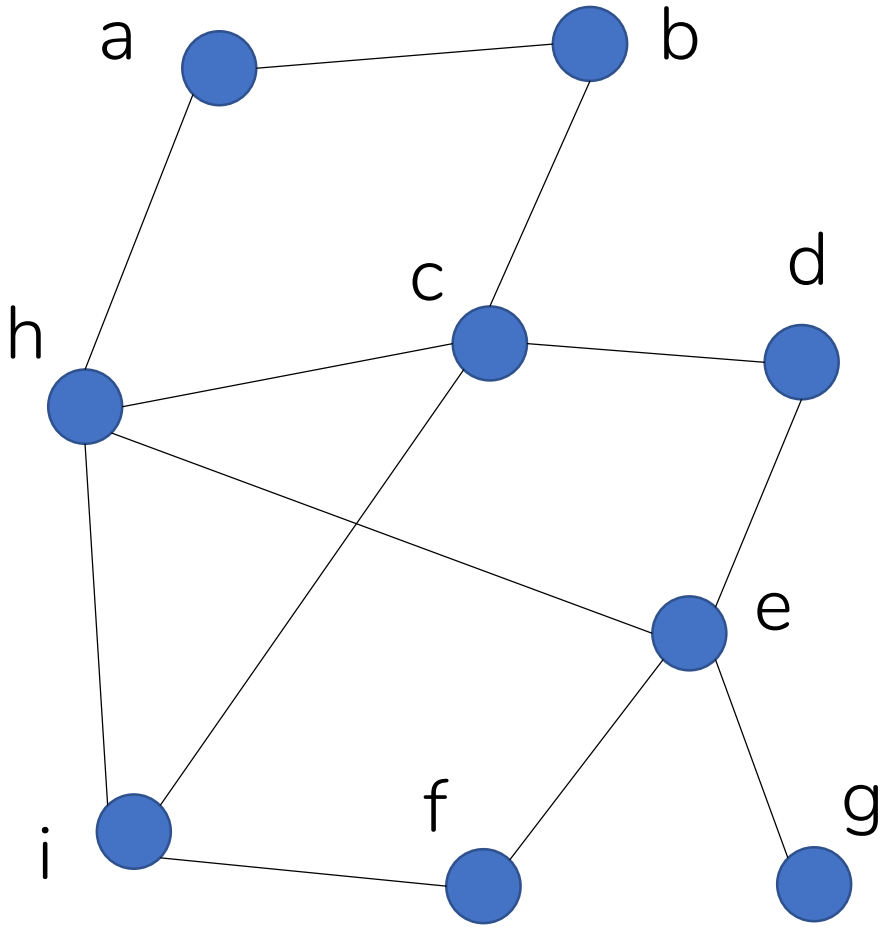



BFS gives shortest-paths from a node to <u>all</u> other nodes.

- works in directed graphs also
- works when edges have lengths also (but may take a long time)

# Basic Algorithms

1. traversing a graph: list all nodes connected to a given node




Depth-First-Search ( $G = (V, E)$ , a)

---

"visit" a; add a to Stack

while (Stack is not empty)

pick a vertex v from Stack

"visit" all unvisited neighbours of v

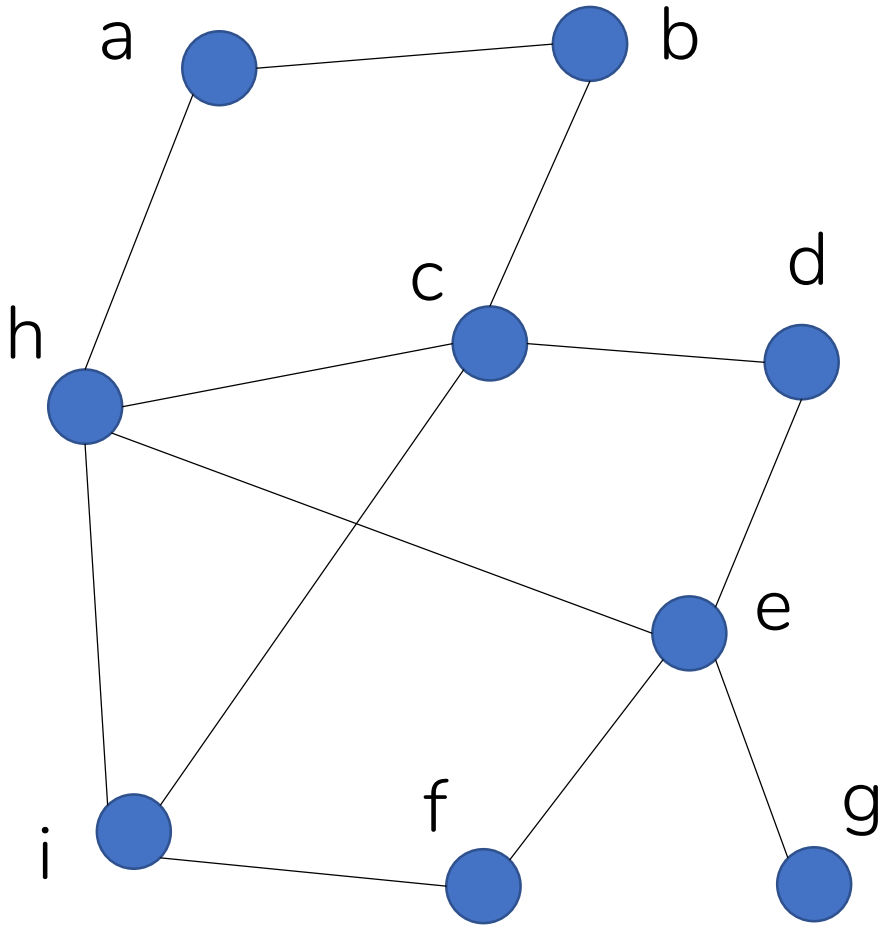
add these to Stack

---

Stack: a

# Basic Algorithms

1. traversing a graph: list all nodes connected to a given node

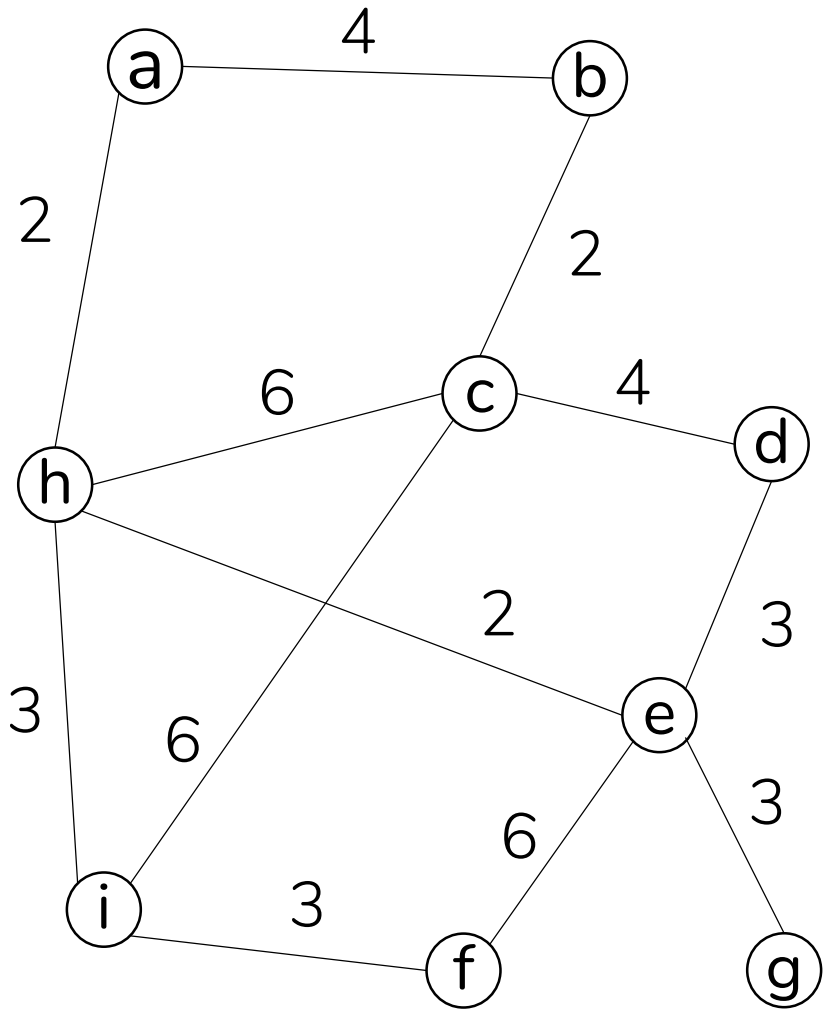



1. Breadth First Search
2. Depth First Search

Questions?

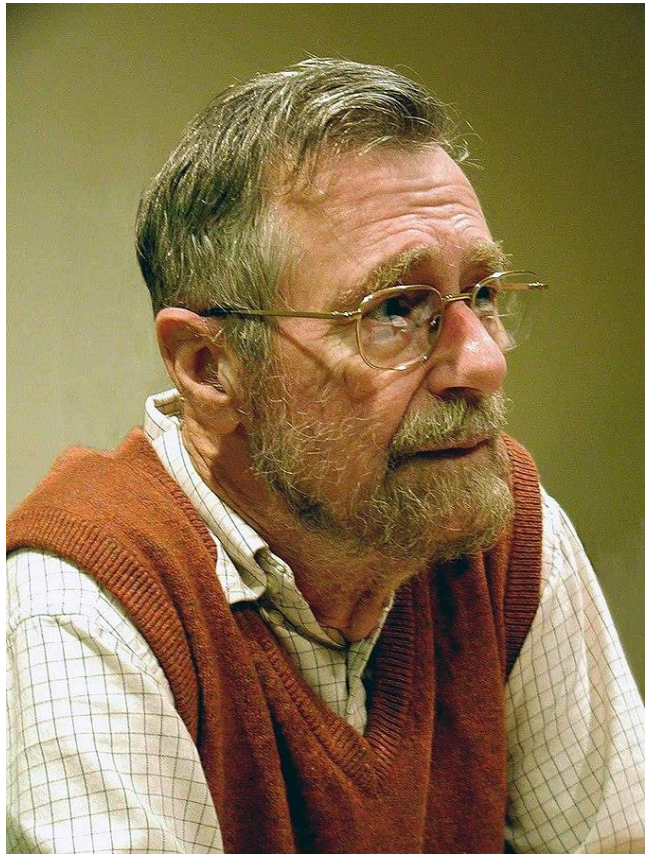
# Basic Algorithms

2. find shortest-paths in a weighted graph




Dijkstra ( $G = (V, E, l)$ , a)

# Basic Algorithms



Edsger Wybe Dijkstra

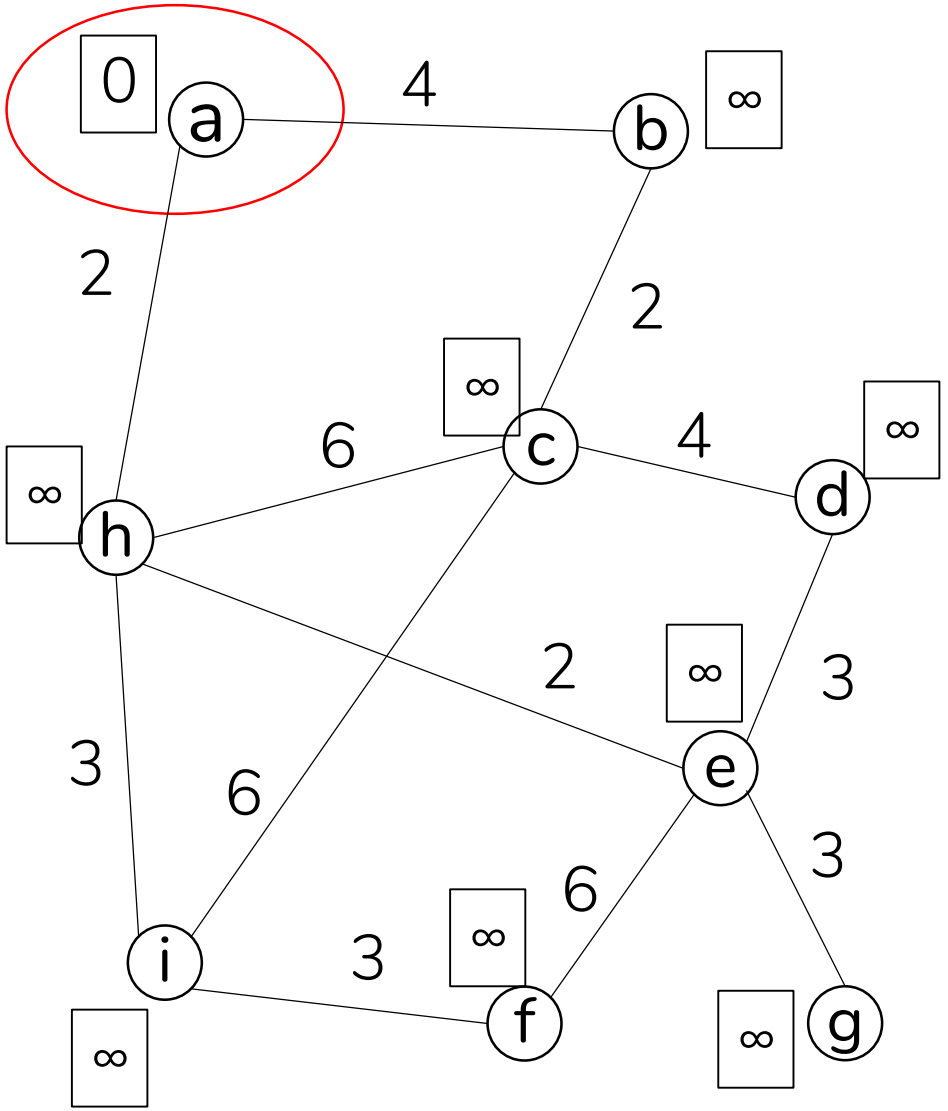
Dutch mathematician, theoretical physicist, computer programmer

Solved the shortest path problem in 1956

Received Turing award, ACM PODC influential paper award (renamed the Dijkstra prize) for contributions to programming languages and distributed computing

Wiki link

2. find shortest-paths in a weighted graph



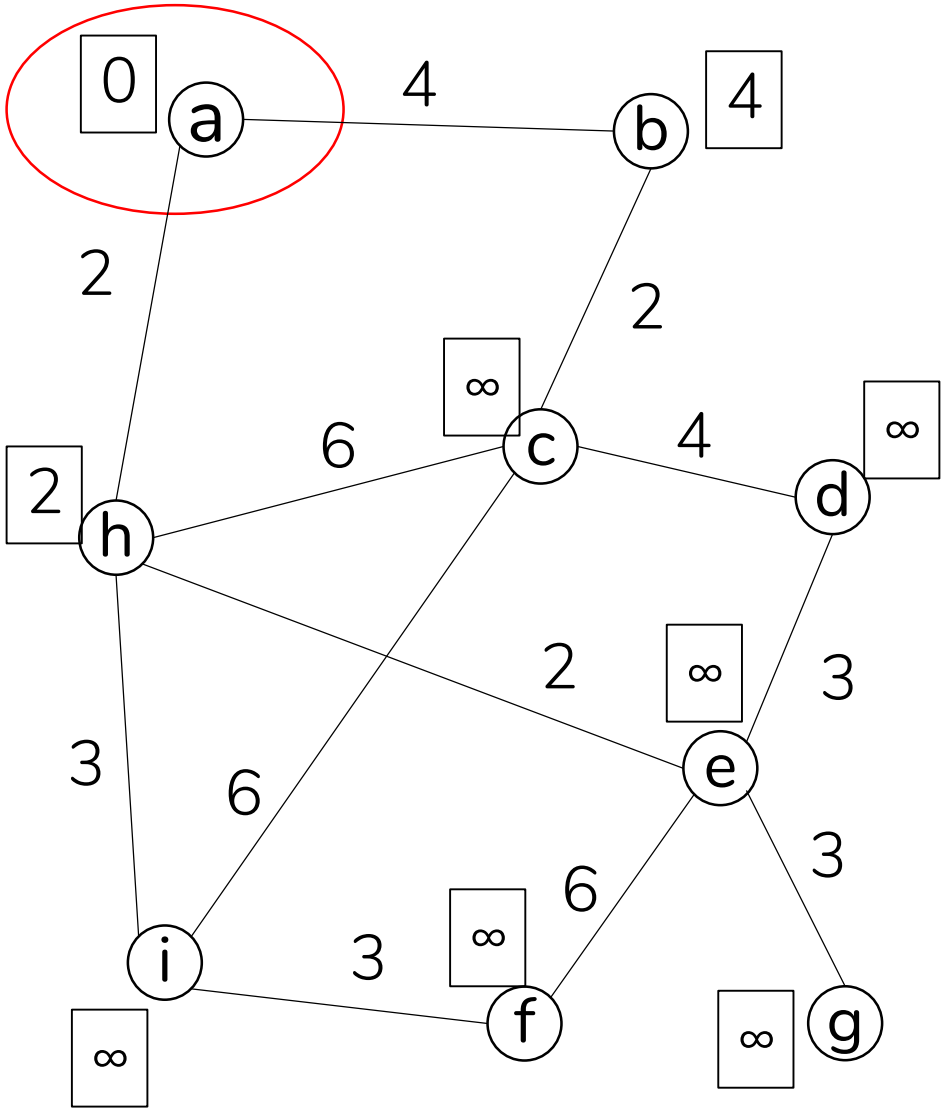

Dijkstra ( $G = (V, E, l)$ , a)

---

dist(a) = 0, dist(v) = ∞ for all others

S = {a} \\ set of shortest-distance vertices

2. find shortest-paths in a weighted graph




Dijkstra ( $G = (V, E)$ , a)

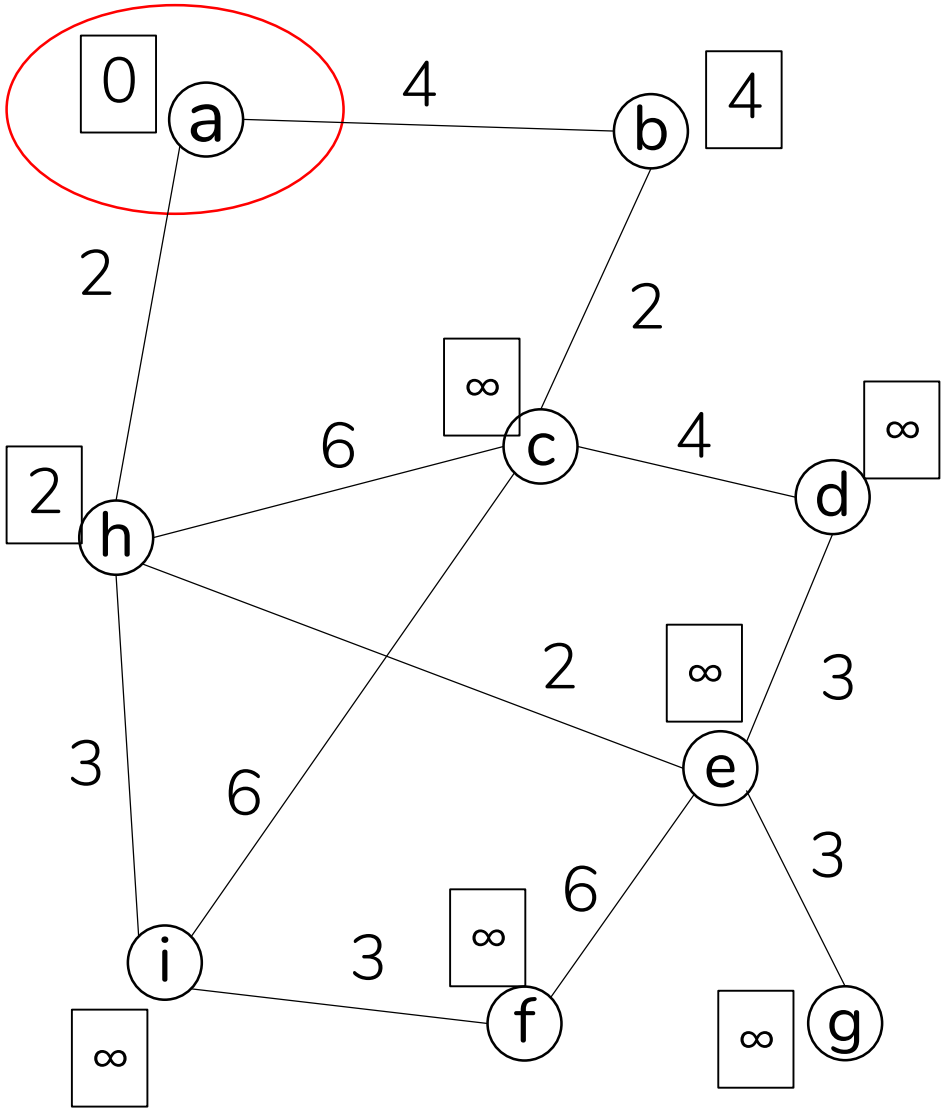
---

dist(a) = 0, dist(v) = ∞ for all others

S = {a} \\ set of shortest-distance vertices

update distance for vtxs adjacent to a

2. find shortest-paths in a weighted graph




Dijkstra ( $G = (V, E)$ , a)

---

dist(a) = 0, dist(v) = ∞  for all others

S = {a}  \\ set of shortest-distance vertices

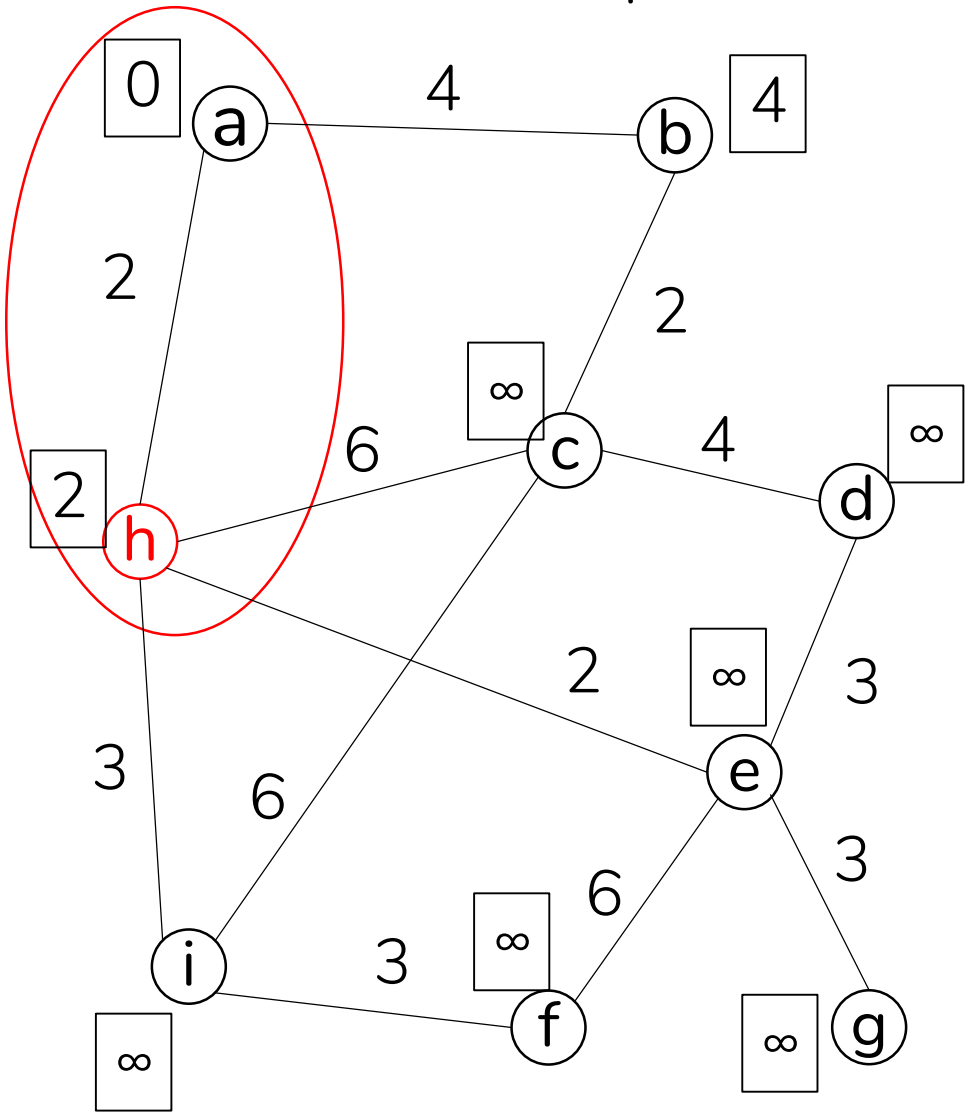
update distance for vtxs adjacent to a

while S ≠ V

  add vtx v with smallest dist to S

  update distance for vtxs w adjacent to v

2. find shortest-paths in a weighted graph




Dijkstra ( $G = (V, E)$ , a)

---

dist(a) = 0, dist(v) = ∞  for all others

S = {a}  \\ set of shortest-distance vertices

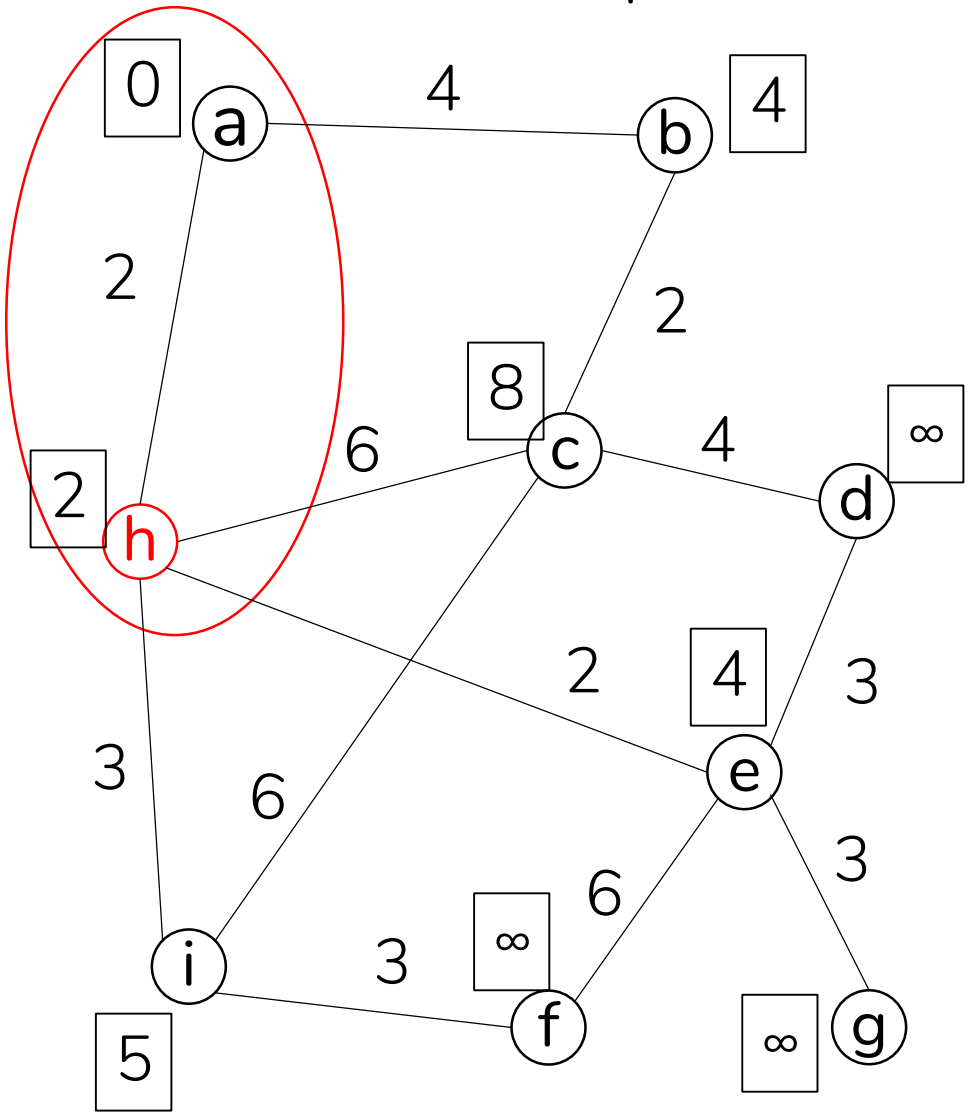
update distance for vtxs adjacent to a

while S ≠ V

  add vtx v with smallest dist to S

  update distance for vtxs w adjacent to v

2. find shortest-paths in a weighted graph




Dijkstra ( $G = (V, E)$ , a)

---

dist(a) = 0, dist(v) = ∞  for all others

S = {a}  \\ set of shortest-distance vertices

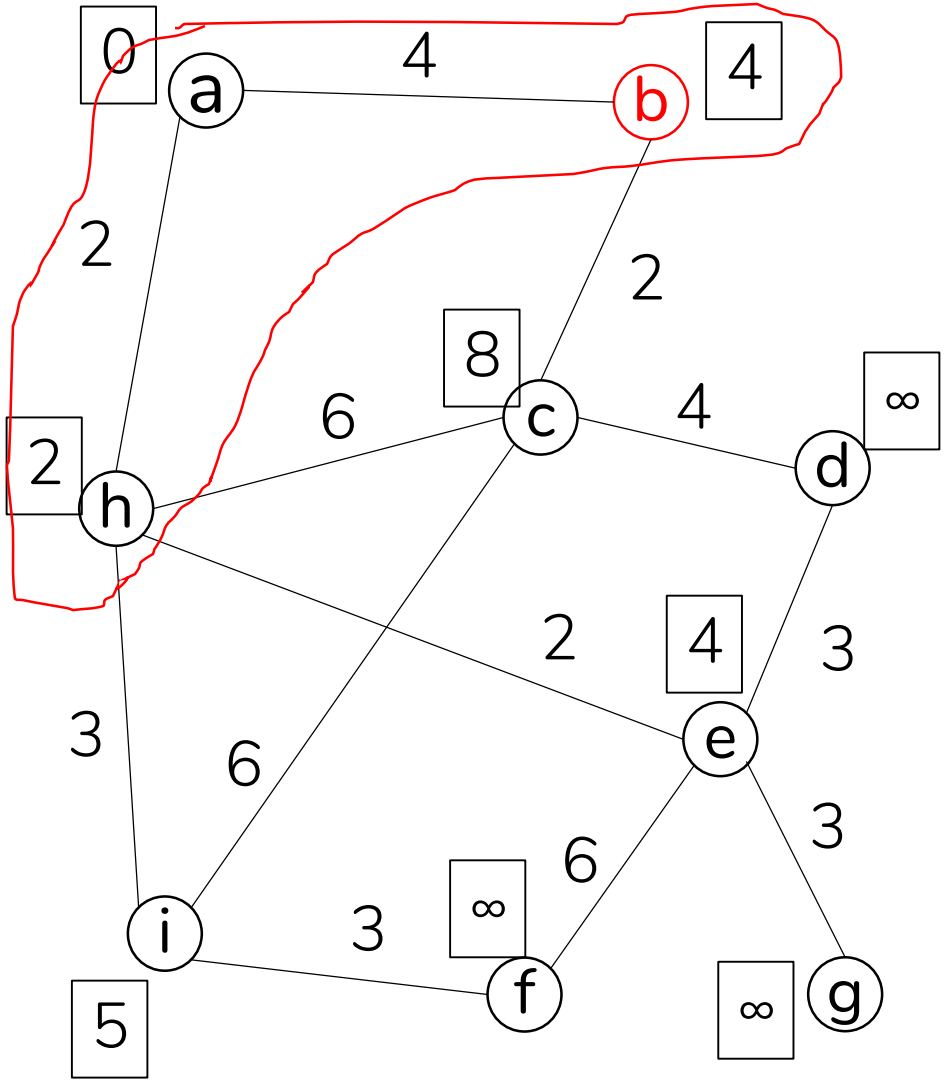
update distance for vtxs adjacent to a

while S ≠ V

  add vtx v with smallest dist to S

  update distance for vtxs w adjacent to v

2. find shortest-paths in a weighted graph




Dijkstra ( $G = (V, E)$ , a)

dist(a) = 0, dist(v) = ∞ for all others

S = {a} \\ set of shortest-distance vertices

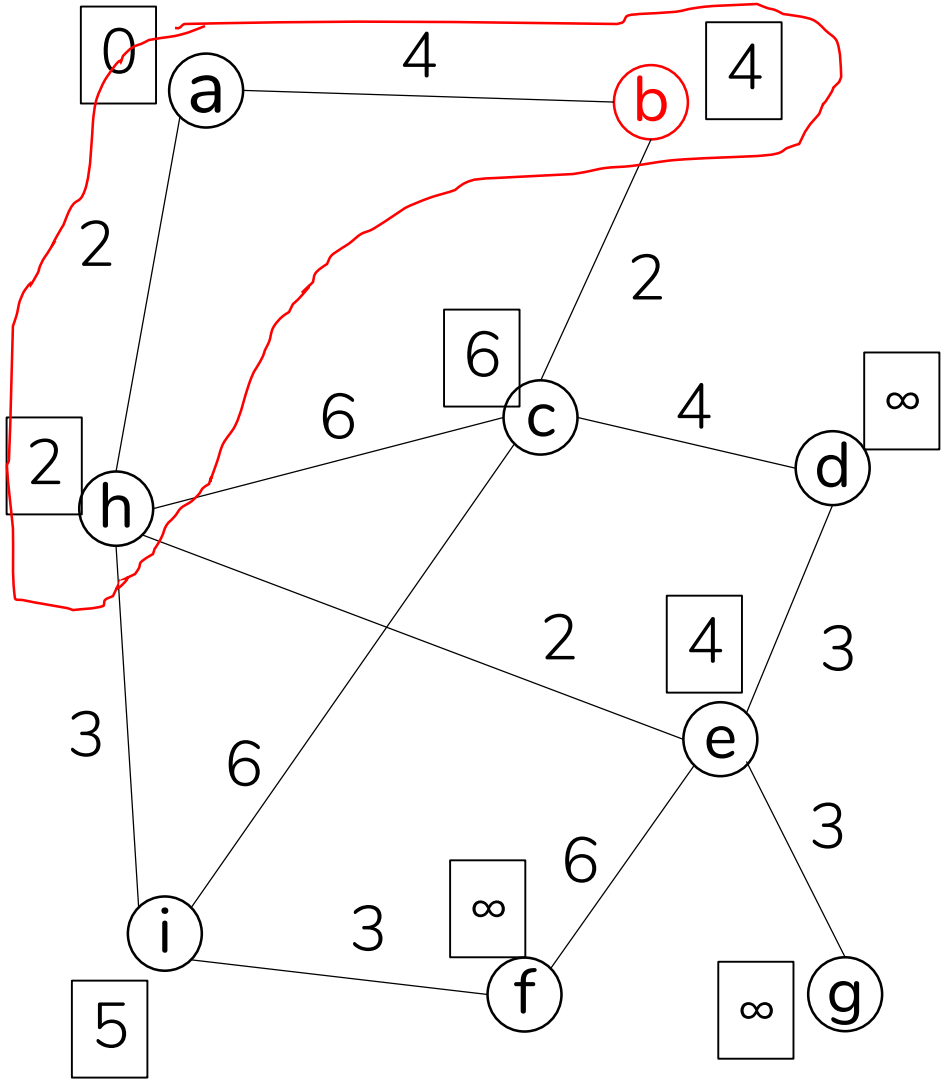
update distance for vtxs adjacent to a

while S ≠ V

add vtx v with smallest dist to S

update distance for vtxs w adjacent to v

2. find shortest-paths in a weighted graph




Dijkstra ( $G = (V, E)$ , a)

---

dist(a) = 0, dist(v) = ∞  for all others

S = {a}  \\ set of shortest-distance vertices

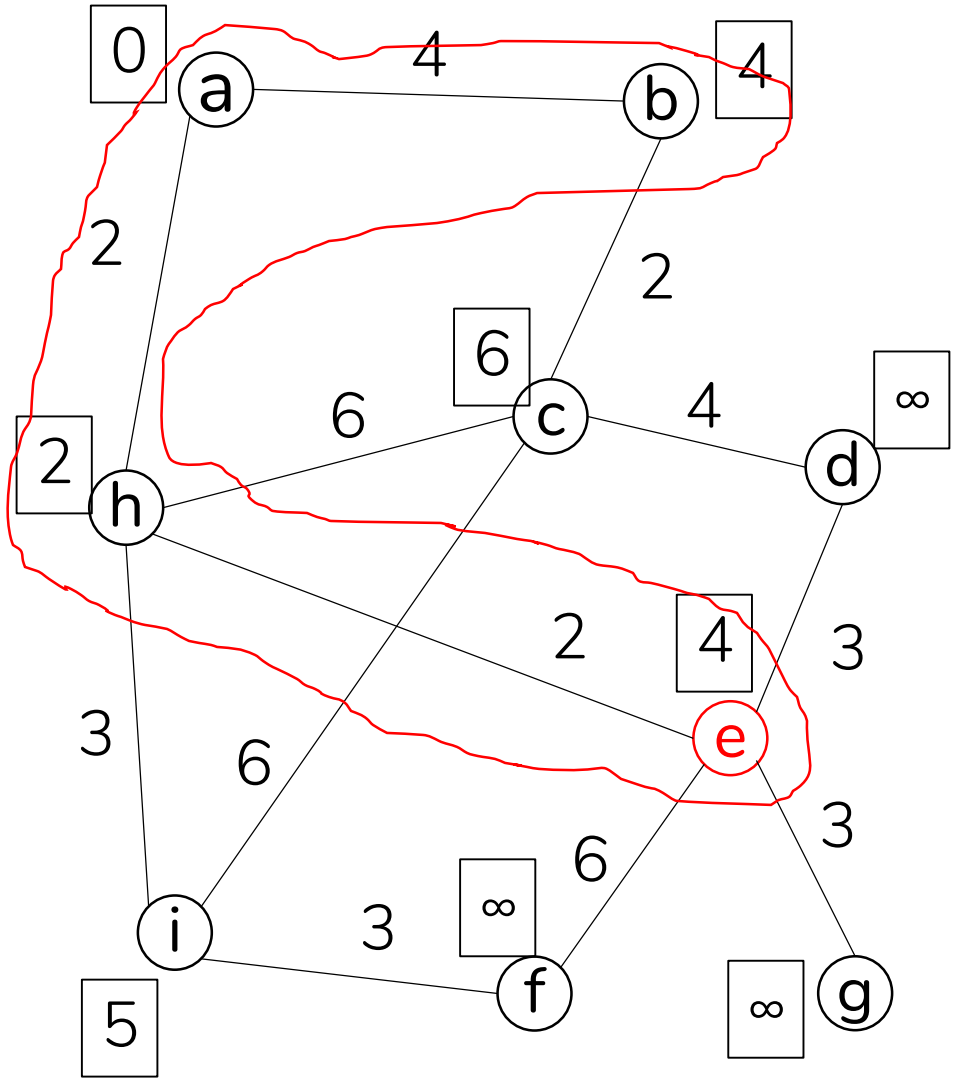
update distance for vtxs adjacent to a

while S ≠ V

  add vtx v with smallest dist to S

  update distance for vtxs w adjacent to v

2. find shortest-paths in a weighted graph




Dijkstra ( $G = (V, E)$ , a)

---

dist(a) = 0, dist(v) = ∞  for all others

S = {a}  \\ set of shortest-distance vertices

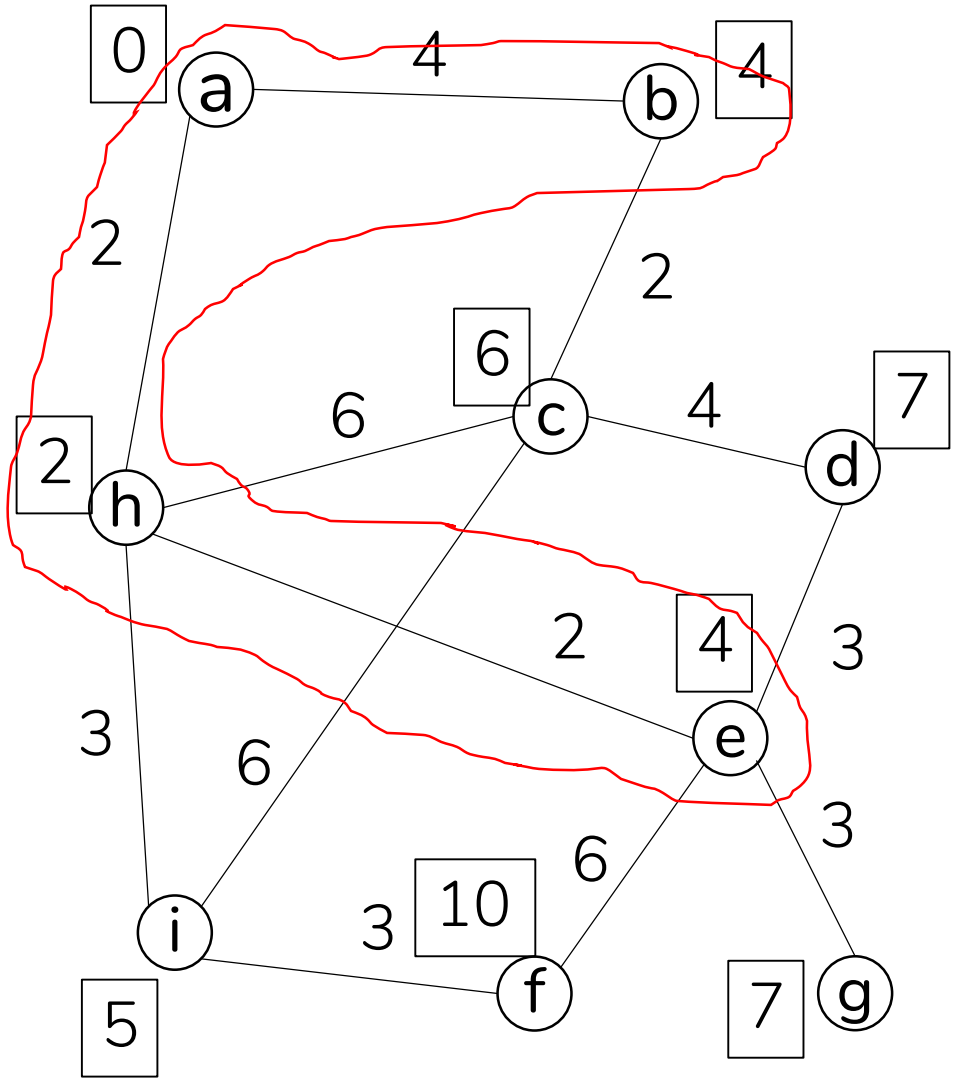
update distance for vtxs adjacent to a

while S ≠ V

  add vtx v with smallest dist to S

  update distance for vtxs w adjacent to v

2. find shortest-paths in a weighted graph




Dijkstra ( $G = (V, E)$ , a)

---

dist(a) = 0, dist(v) = ∞  for all others

S = {a}  \\ set of shortest-distance vertices

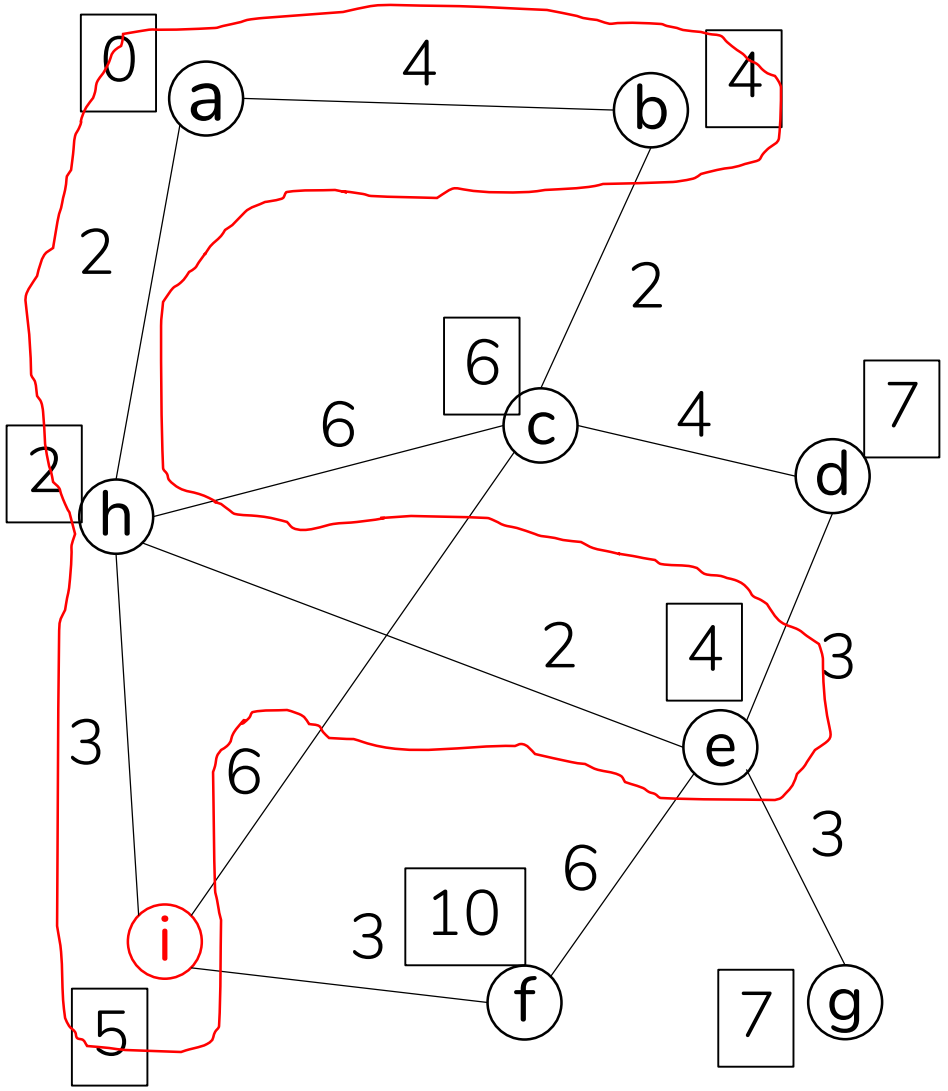
update distance for vtxs adjacent to a

while S ≠ V

  add vtx v with smallest dist to S

  update distance for vtxs w adjacent to v

2. find shortest-paths in a weighted graph




Dijkstra ( $G = (V, E)$ , a)

---

dist(a) = 0, dist(v) = ∞  for all others

S = {a}  \\ set of shortest-distance vertices

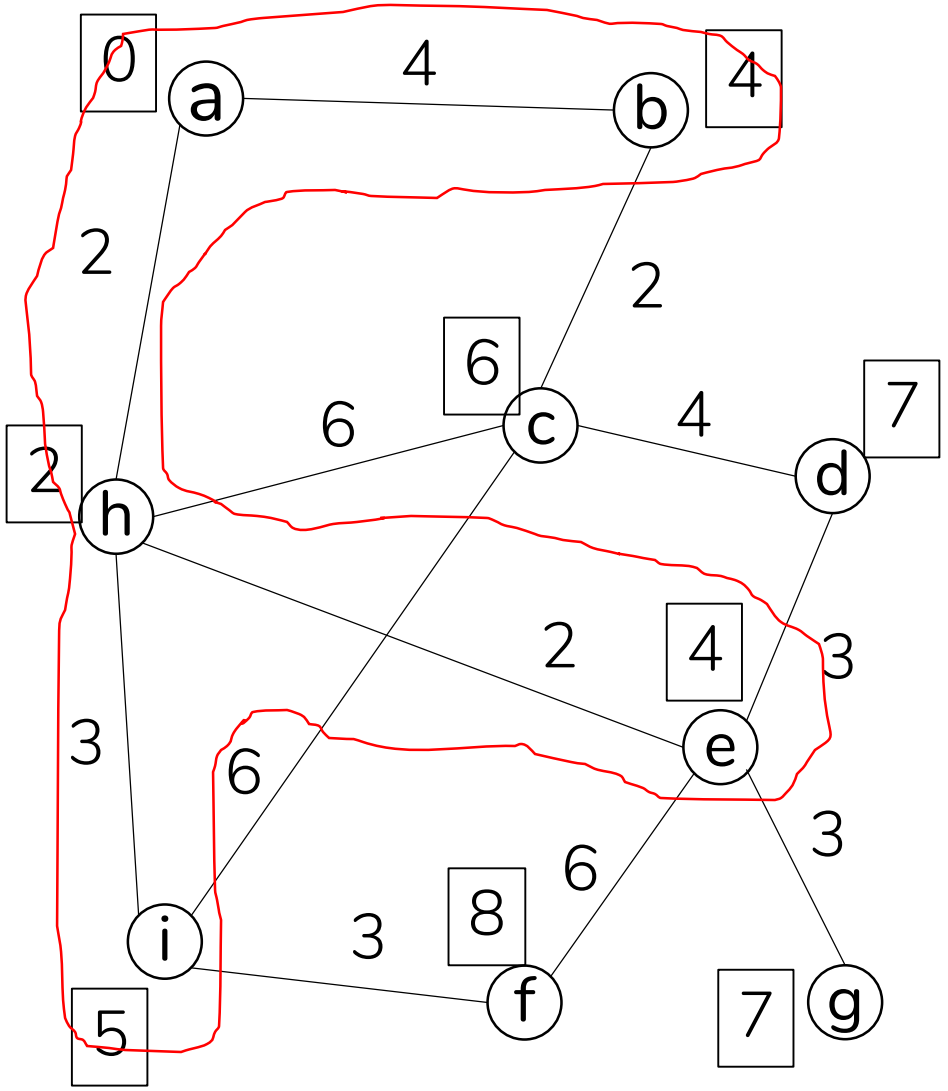
update distance for vtxs adjacent to a

while S ≠ V

  add vtx v with smallest dist to S

  update distance for vtxs w adjacent to v

2. find shortest-paths in a weighted graph




Dijkstra ( $G = (V, E)$ , a)

---

dist(a) = 0, dist(v) = ∞  for all others

S = {a}  \\ set of shortest-distance vertices

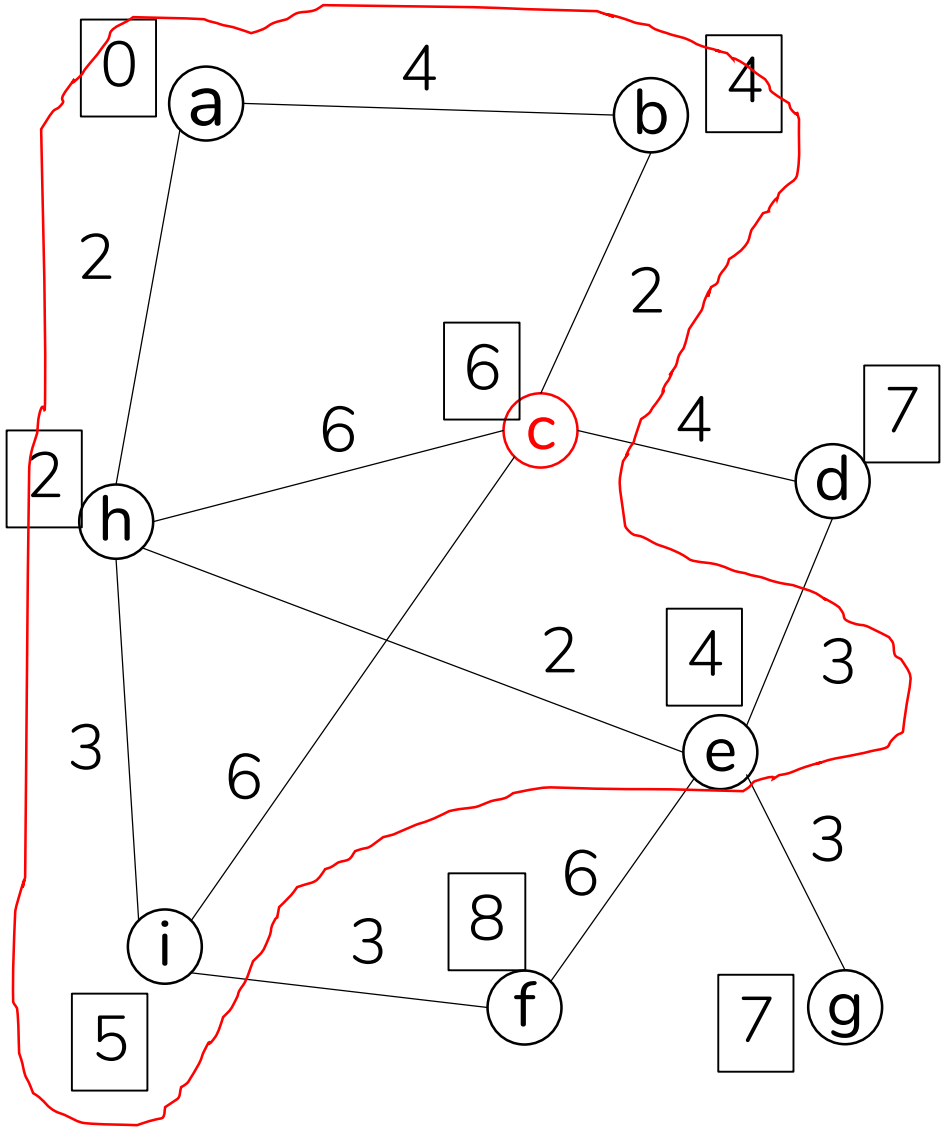
update distance for vtxs adjacent to a

while S ≠ V

  add vtx v with smallest dist to S

  update distance for vtxs w adjacent to v

2. find shortest-paths in a weighted graph




Dijkstra ( $G = (V, E)$ , a)

---

dist(a) = 0, dist(v) = ∞  for all others

S = {a}  \\ set of shortest-distance vertices

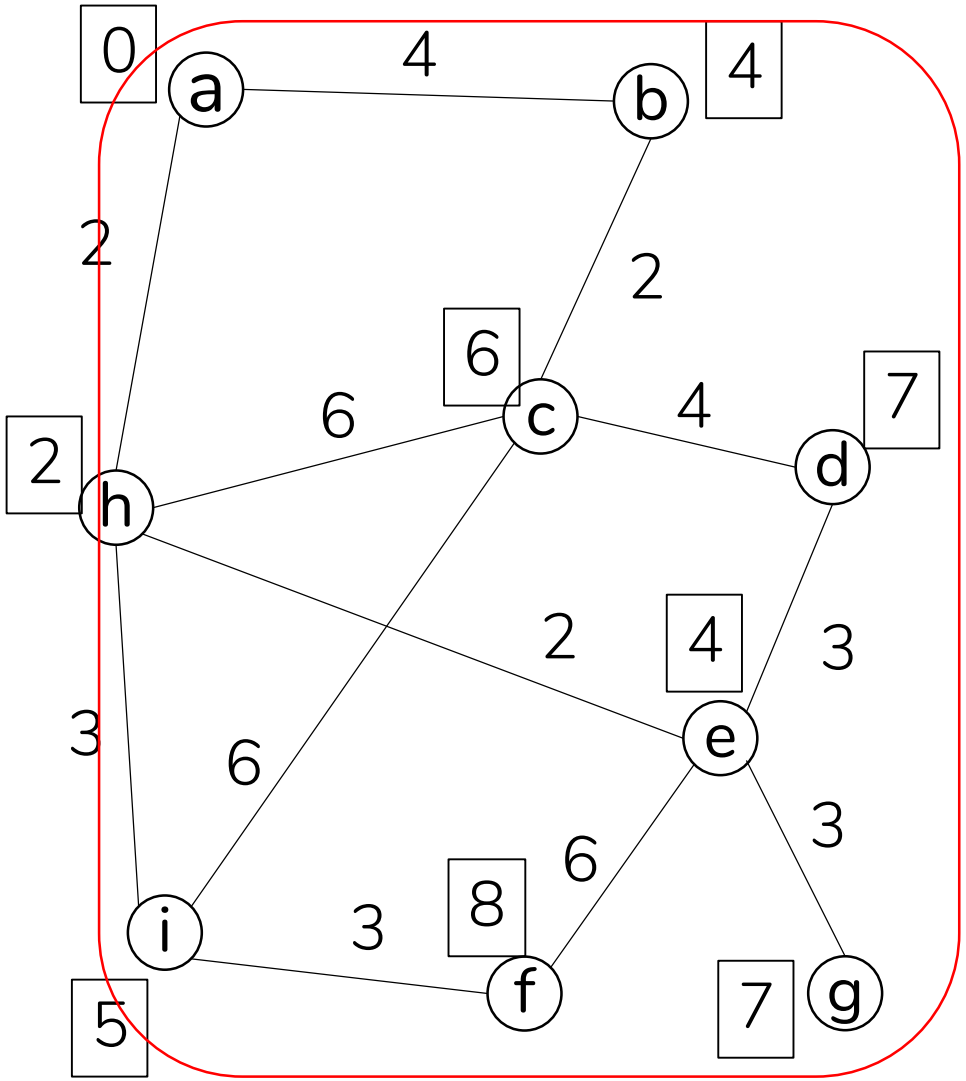
update distance for vtxs adjacent to a

while S ≠ V

  add vtx v with smallest dist to S

  update distance for vtxs w adjacent to v

2. find shortest-paths in a weighted graph




Dijkstra ( $G = (V, E)$ , a)

dist(a) = 0, dist(v) = ∞  for all others

S = {a}  \\ set of shortest-distance vertices

update distance for vtxs adjacent to a

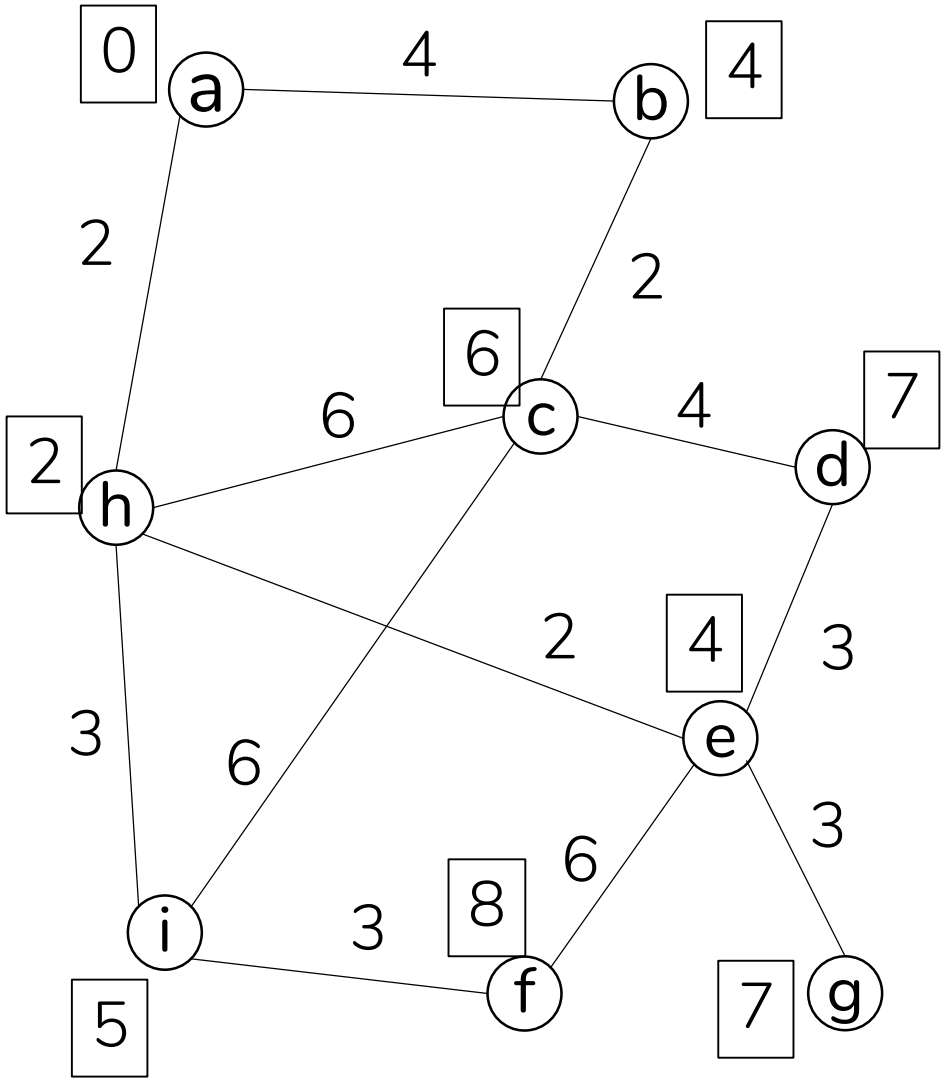
while S ≠ V

  add vtx v with smallest dist to S

  update distance for vtxs w adjacent to v

# Basic Algorithms

2. find shortest-paths in a weighted graph




Dijkstra ( $G = (V, E)$ , a)

dist(a) = 0, dist(v) = ∞  for all others

S = {a}  \\ set of shortest-distance vertices

update distance for vtxs adjacent to a

while S ≠ V

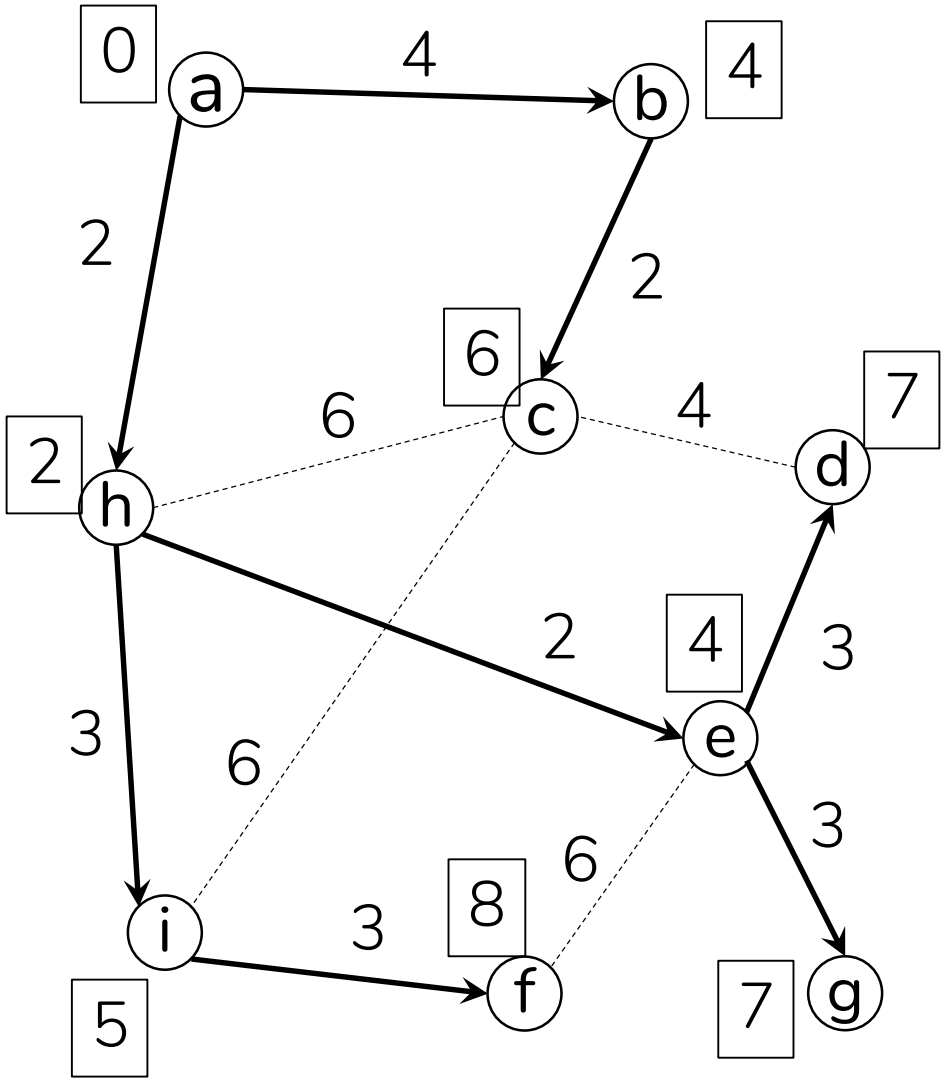
  add vtx v with smallest dist to S

  update distance for vtxs w adjacent to v

  add directed edge (v,w)
  (and remove other edges into w)

# Basic Algorithms

2. find shortest-paths in a weighted graph




Dijkstra ( $G = (V, E)$ , a)

---

dist(a) = 0, dist(v) = ∞  for all others

S = {a}  \\ set of shortest-distance vertices

update distance for vtxs adjacent to a

while S ≠ V

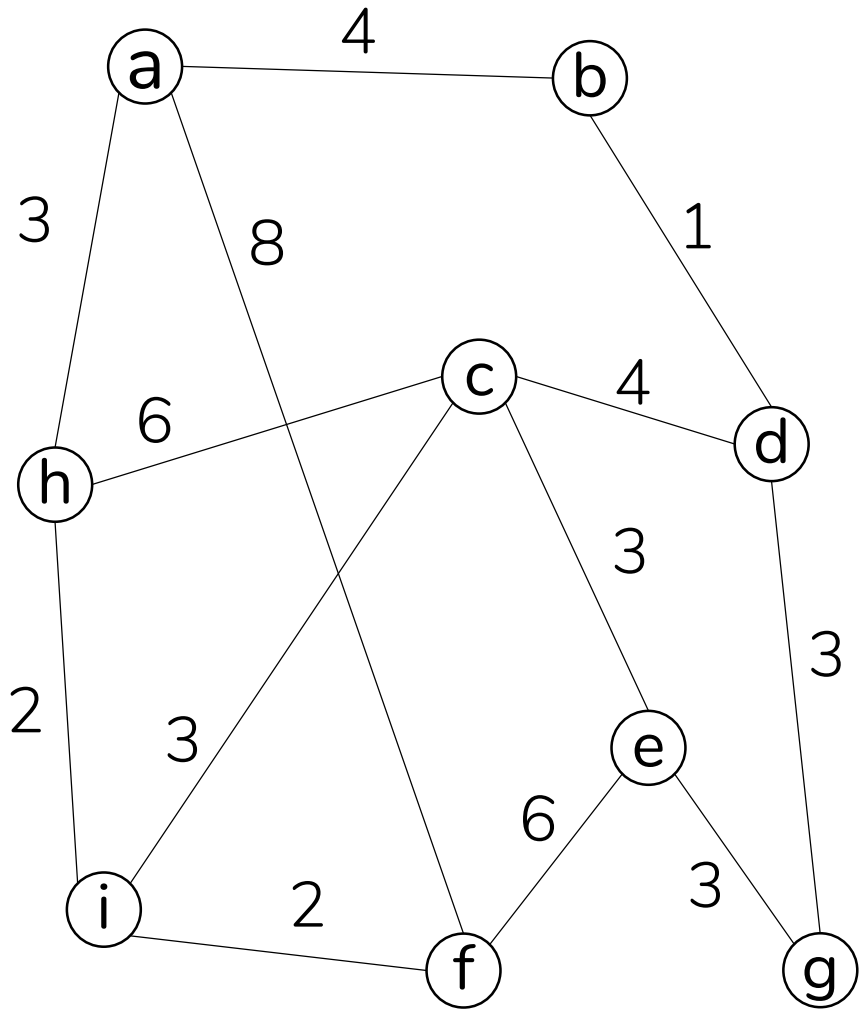
  add vtx v with smallest dist to S

  update distance for vtxs w adjacent to v

  add directed edge (v,w)
  (and remove other edges into w)

# Basic Algorithms

2. find shortest-paths in a weighted graph




Dijkstra ( $G = (V, E)$ , a)

---

dist(a) = 0, dist(v) = ∞  for all others

S = {a}  \\ set of shortest-distance vertices

update distance for vtxs adjacent to a

while S ≠ V

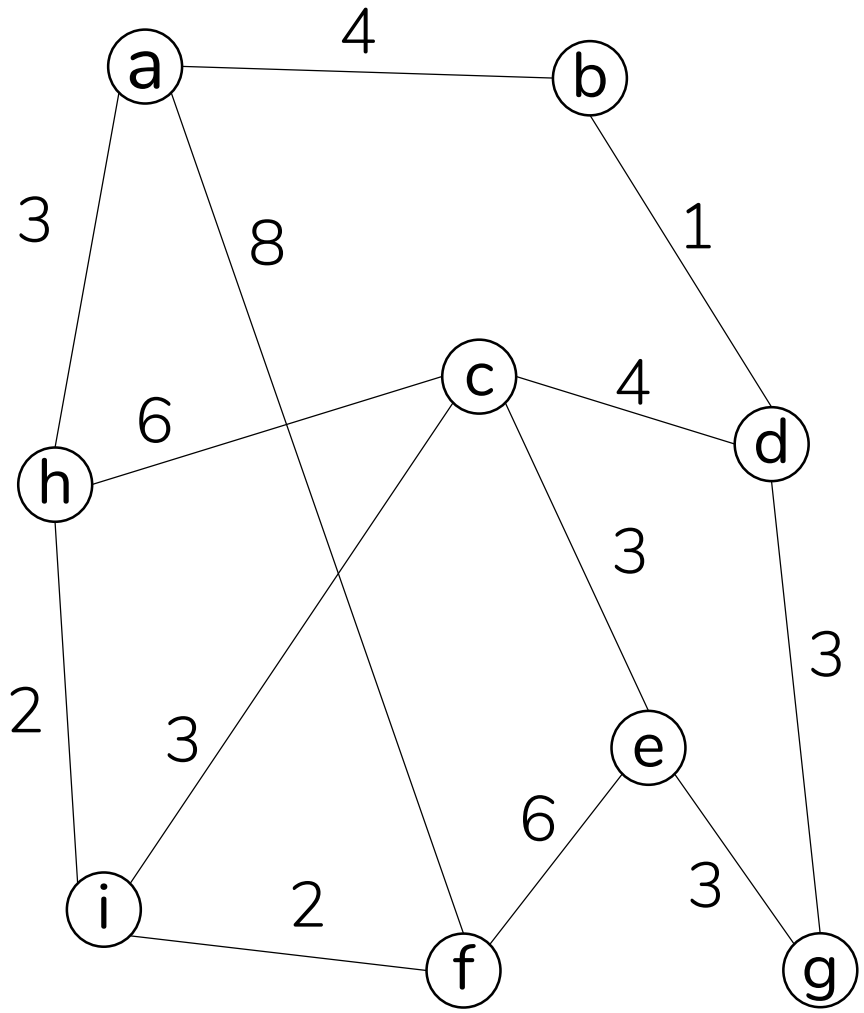
  add vtx v with smallest dist to S

  update distance for vtxs w adjacent to v

  add directed edge (v,w)
  (and remove other edges into w)

# Basic Algorithms

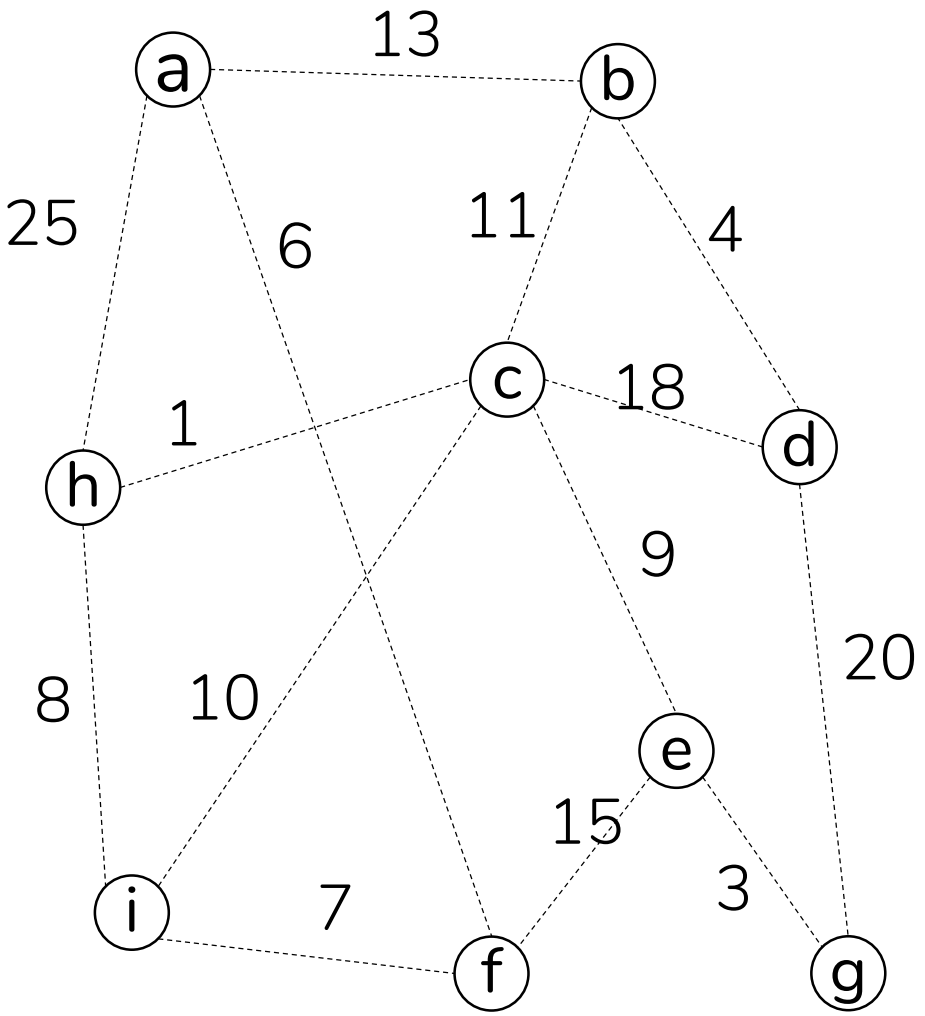
2. find shortest-paths in a weighted graph




Dijkstra's algorithm

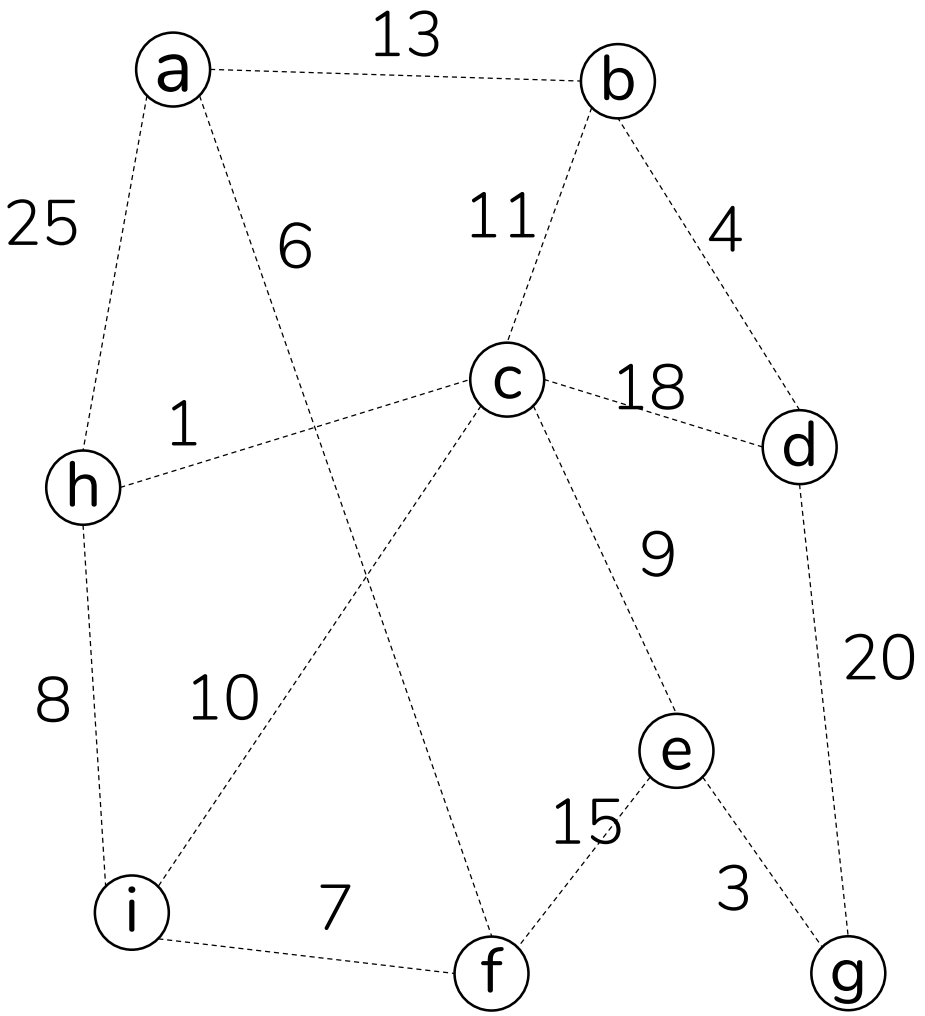
Questions?

3. find min-cost graph that connects all vertices



# Basic Algorithms

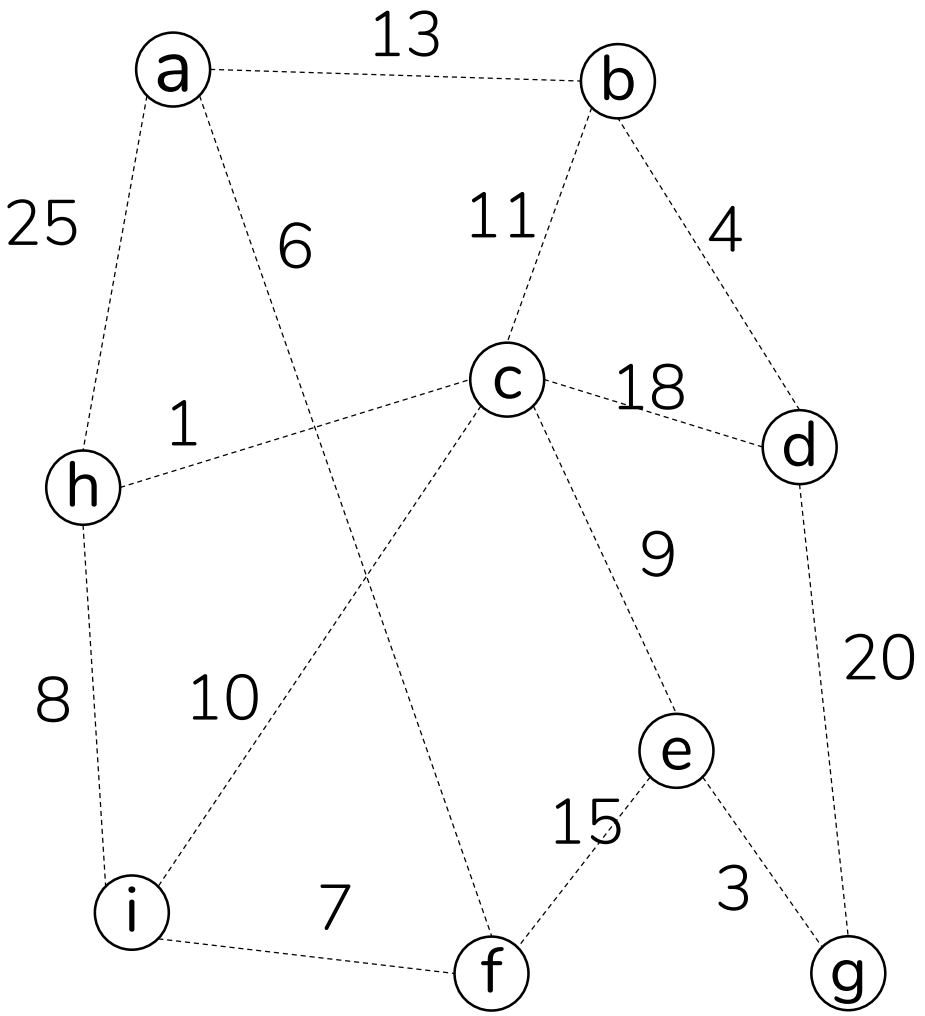
3. find min-cost graph that connects all vertices




can this graph have cycles?

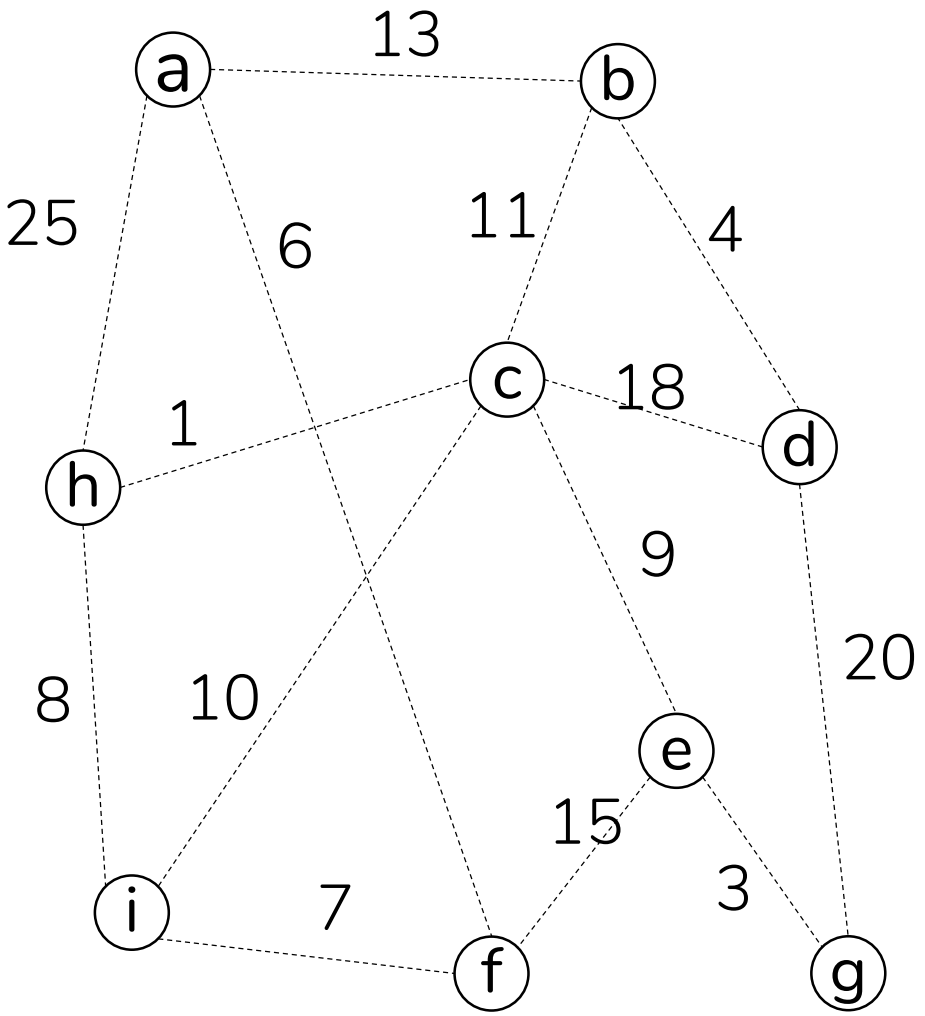
how many edges does this graph have?

3. find min-cost graph that connects all vertices




Kruskal ( $G = (V, E, c)$)

3. find min-cost graph that connects all vertices

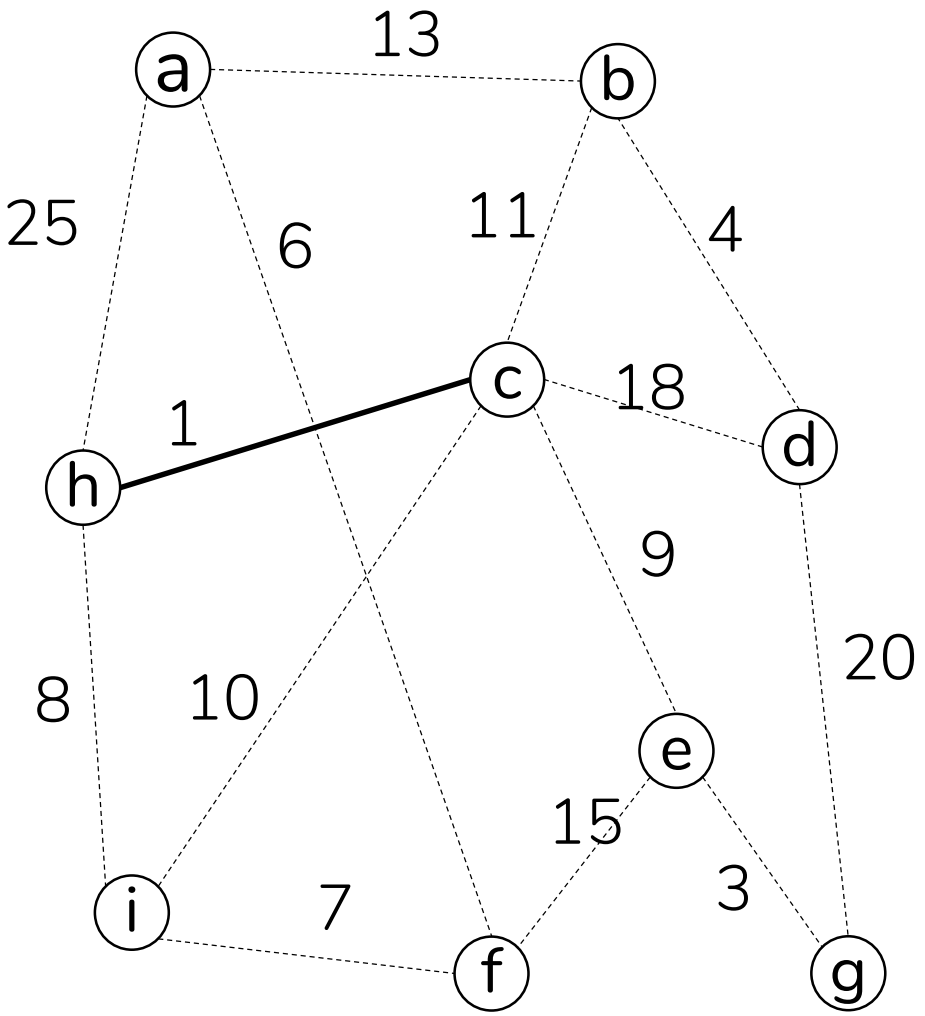



Kruskal ( $G = (V, E, c)$)

---

Keep picking min-cost edges, as long as you don't get a cycle

3. find min-cost graph that connects all vertices

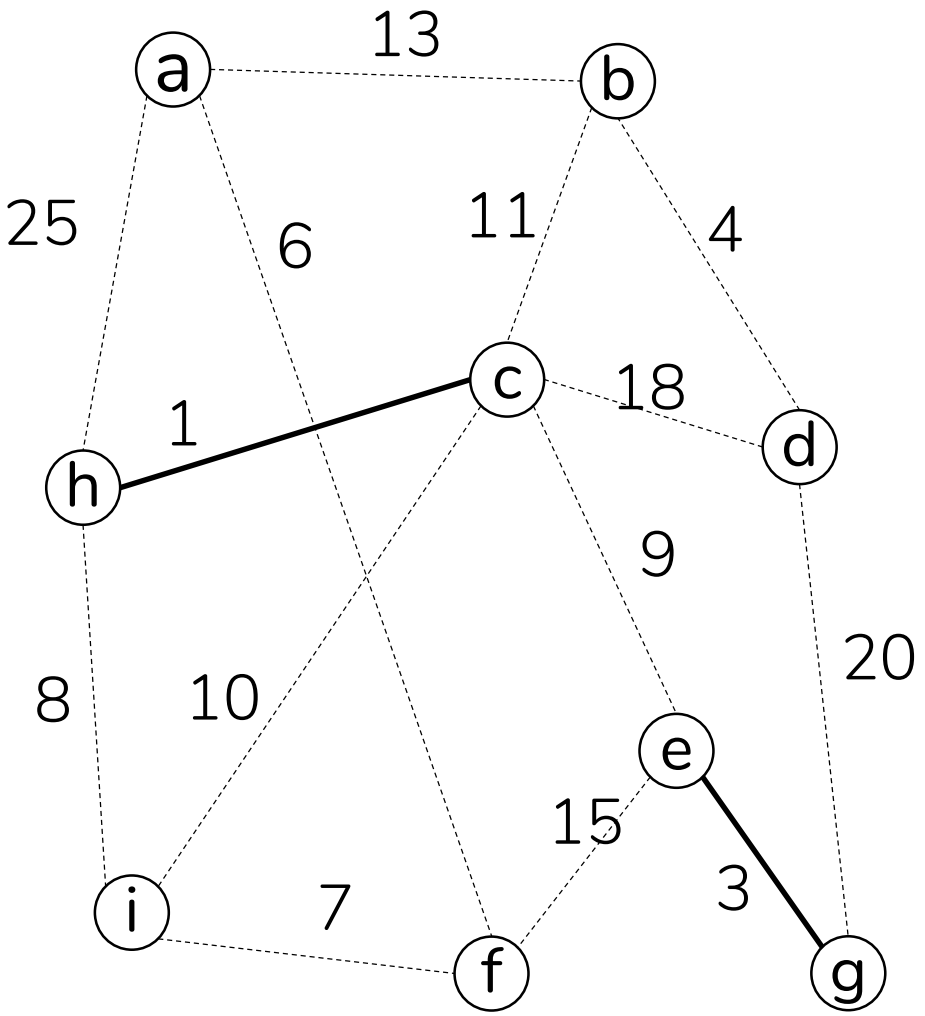



Kruskal ( $G = (V, E, c)$)

---

Keep picking min-cost edges, as long as you don't get a cycle

3. find min-cost graph that connects all vertices

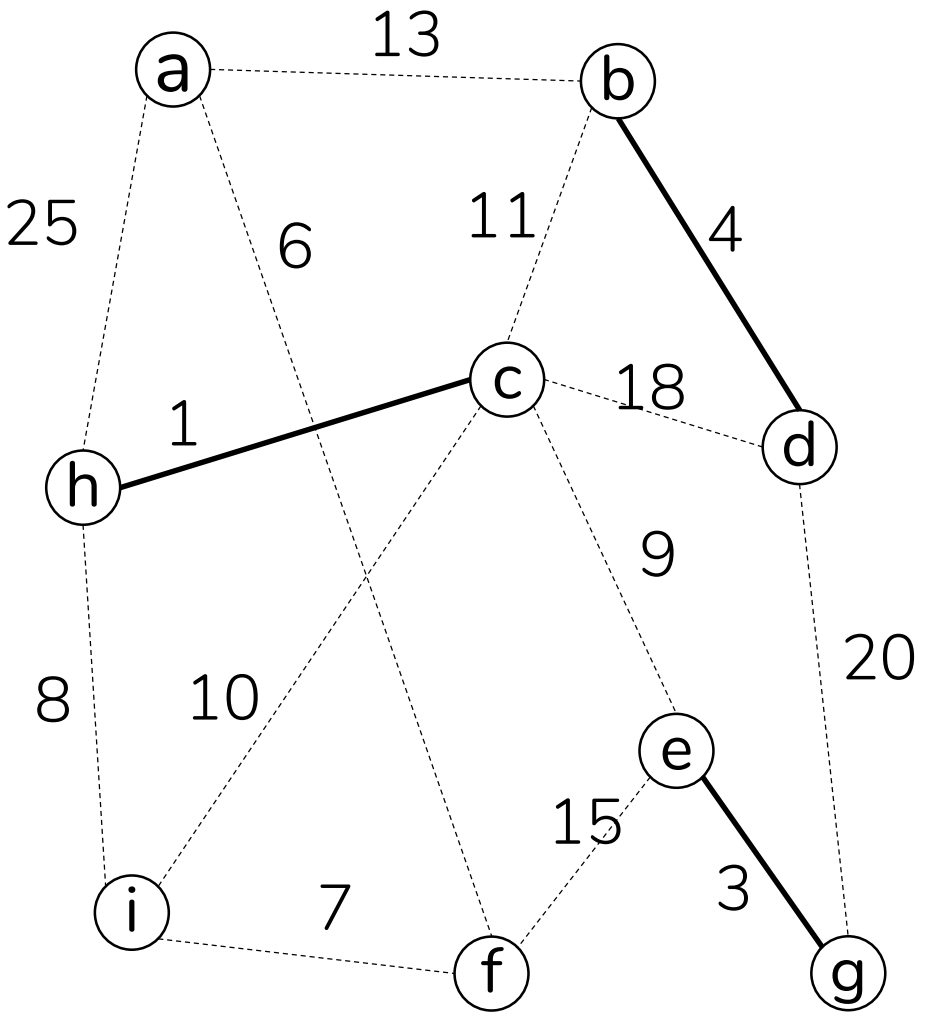



Kruskal ( $G = (V, E, c)$)

---

Keep picking min-cost edges, as long as you don't get a cycle

3. find min-cost graph that connects all vertices

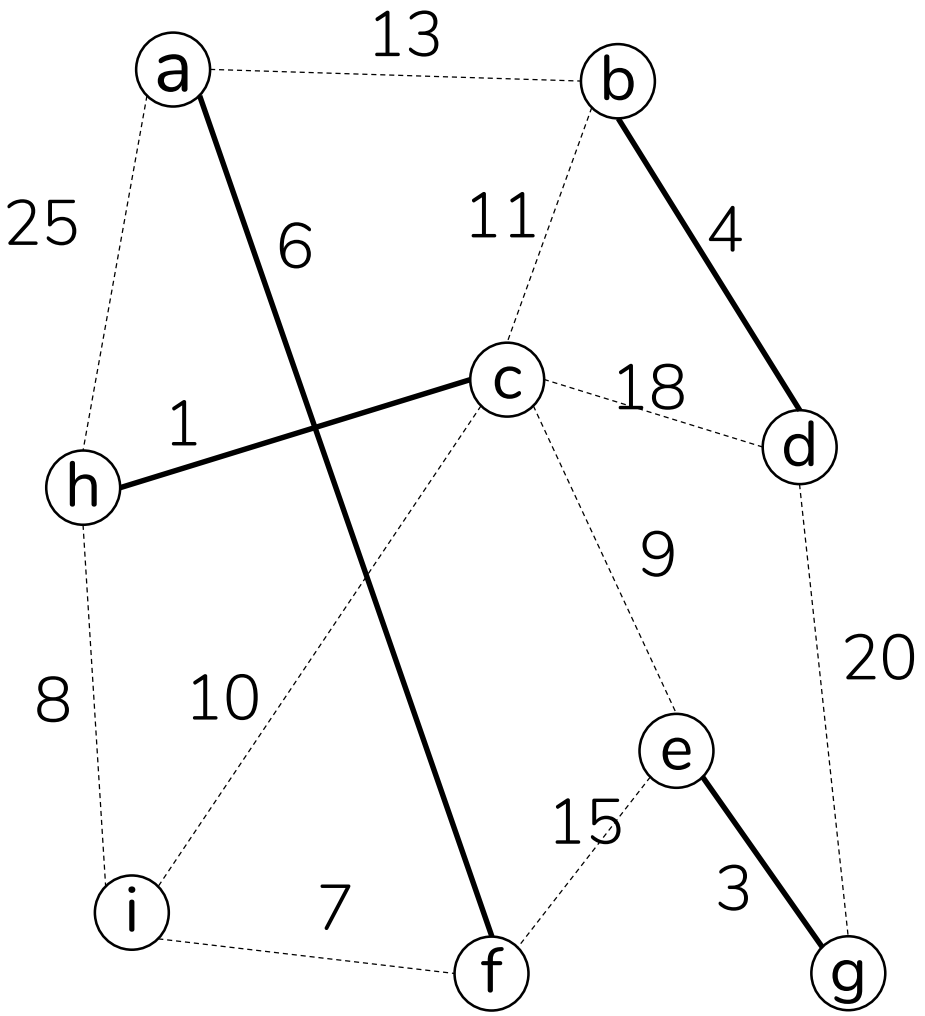



Kruskal ( $G = (V, E, c)$)

---

Keep picking min-cost edges, as long as you don't get a cycle

3. find min-cost graph that connects all vertices

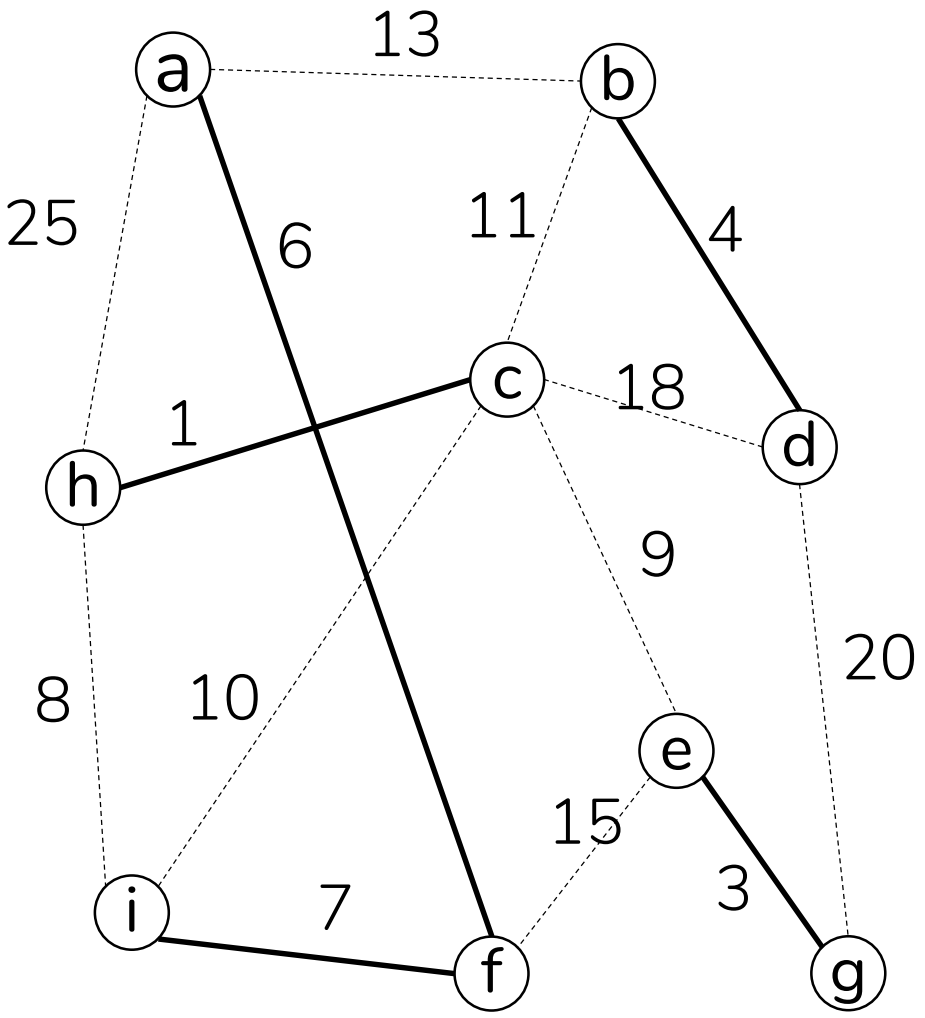



Kruskal ( $G = (V, E, c)$)

---

Keep picking min-cost edges, as long as you don't get a cycle

3. find min-cost graph that connects all vertices

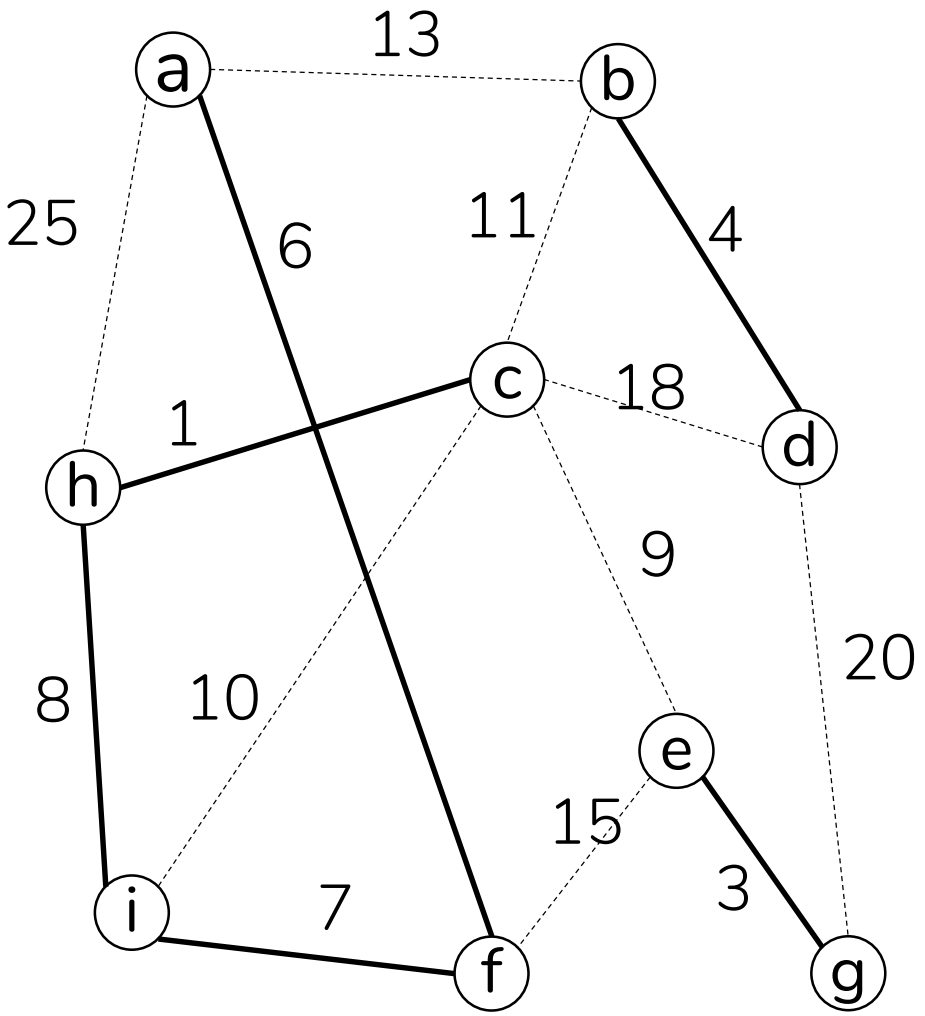



Kruskal ( $G = (V, E, c)$)

---

Keep picking min-cost edges, as long as you don't get a cycle

3. find min-cost graph that connects all vertices

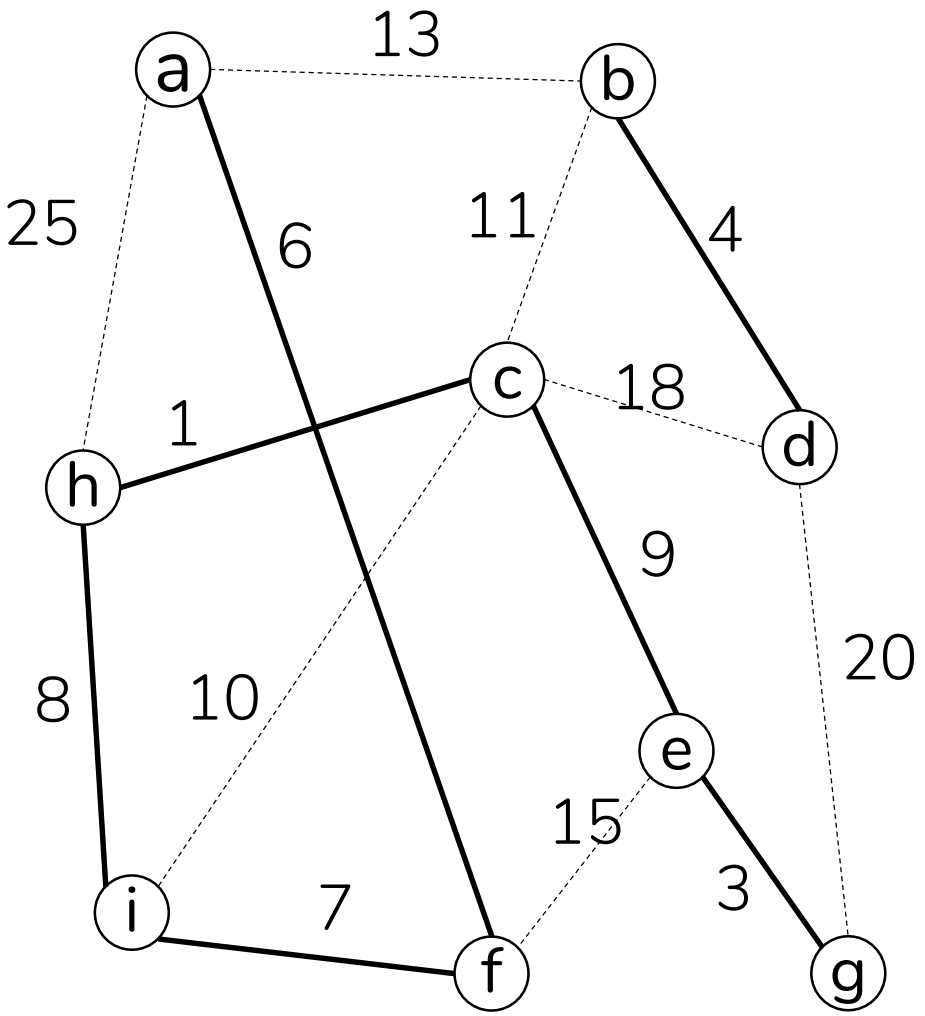



Kruskal ( $G = (V, E, c)$)

---

Keep picking min-cost edges, as long as you don't get a cycle

3. find min-cost graph that connects all vertices

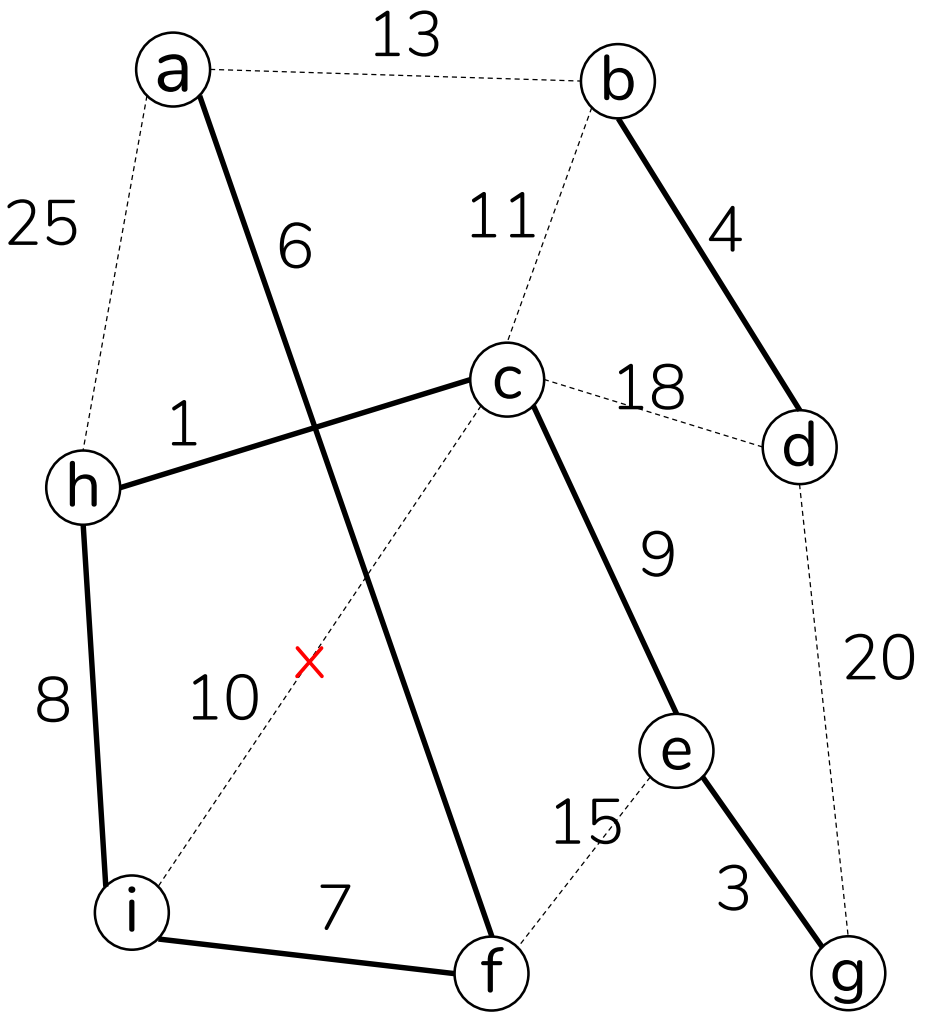



Kruskal ( $G = (V, E, c)$)

---

Keep picking min-cost edges, as long as you don't get a cycle

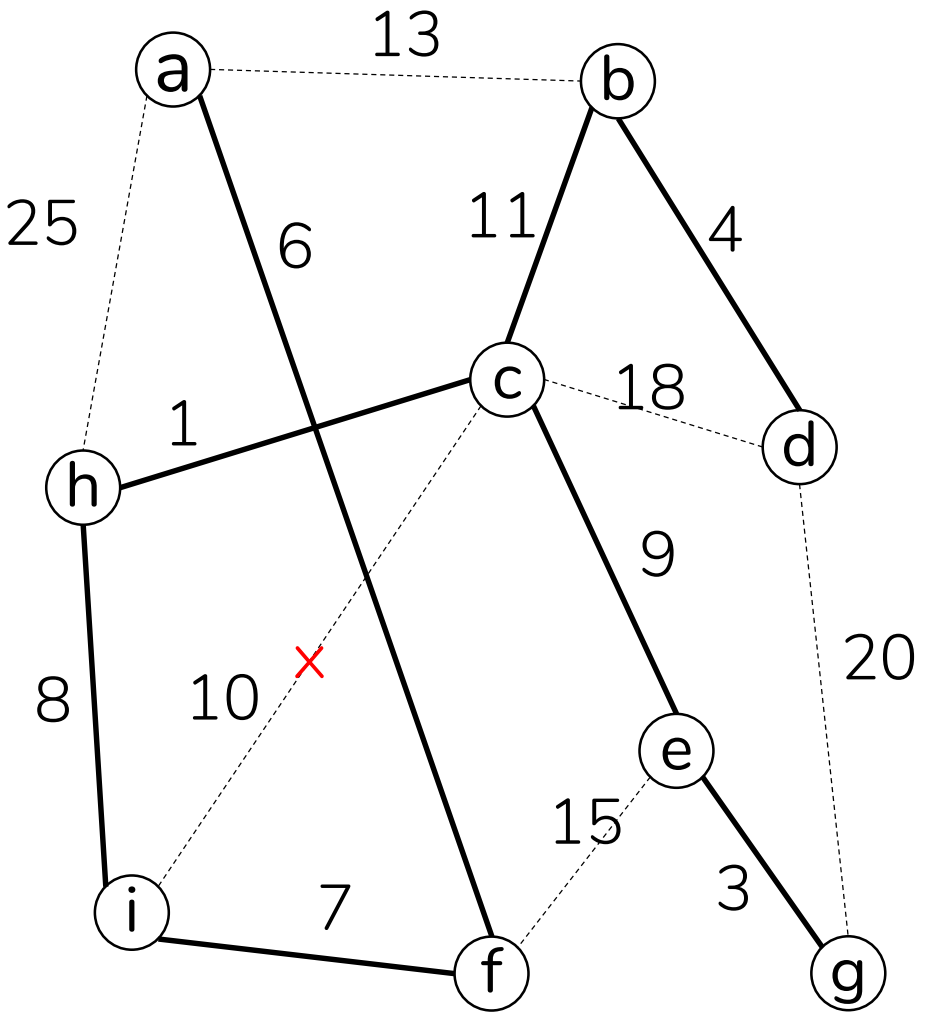
3. find min-cost graph that connects all vertices




Kruskal ( $G = (V, E, c)$)

Keep picking min-cost edges, as long as you don't get a cycle

3. find min-cost graph that connects all vertices

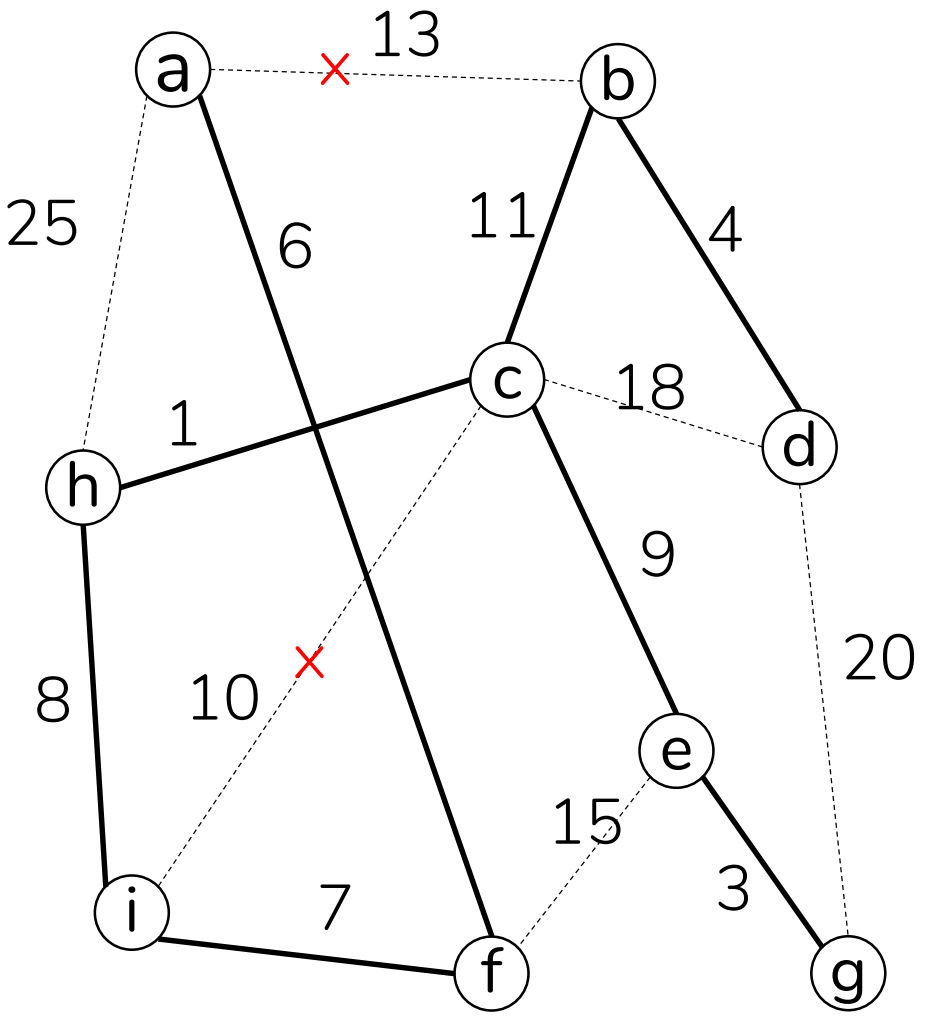



Kruskal ( $G = (V, E, c)$)

---

Keep picking min-cost edges, as long as you don't get a cycle

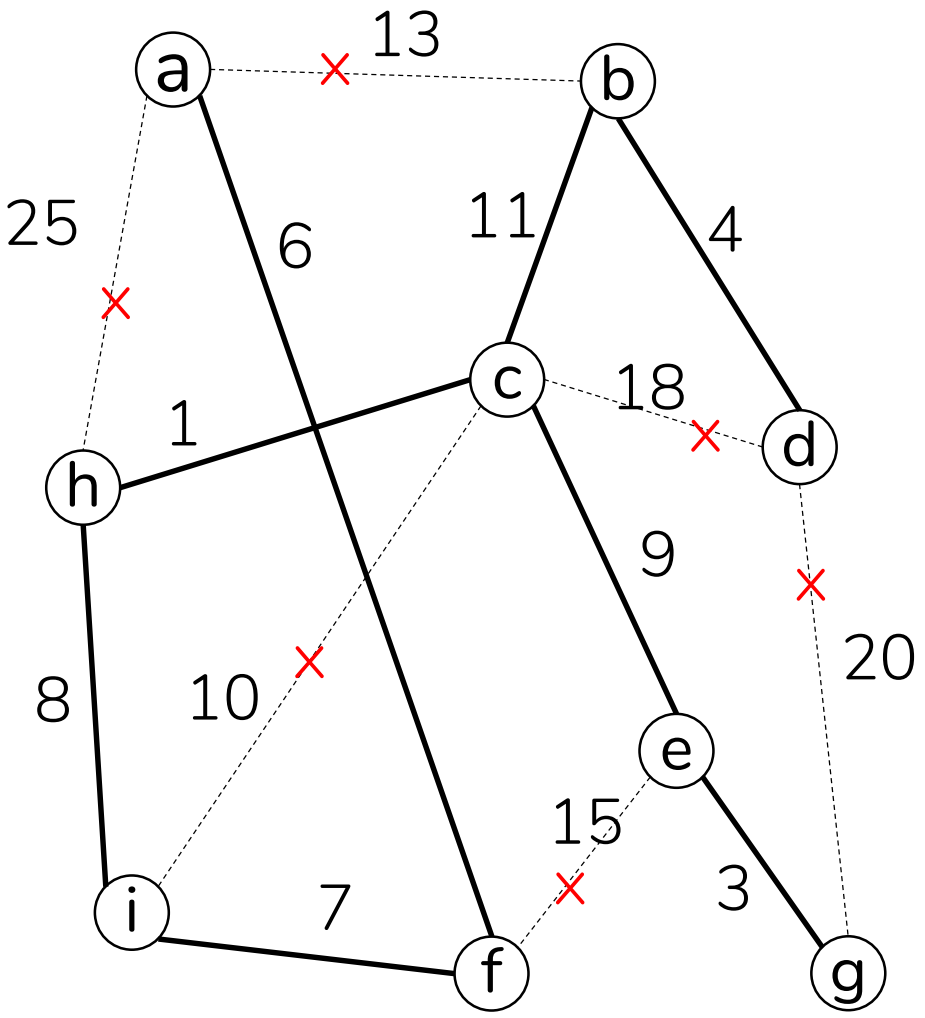
3. find min-cost graph that connects all vertices




Kruskal ( $G = (V, E, c)$)

Keep picking min-cost edges, as long as you don't get a cycle

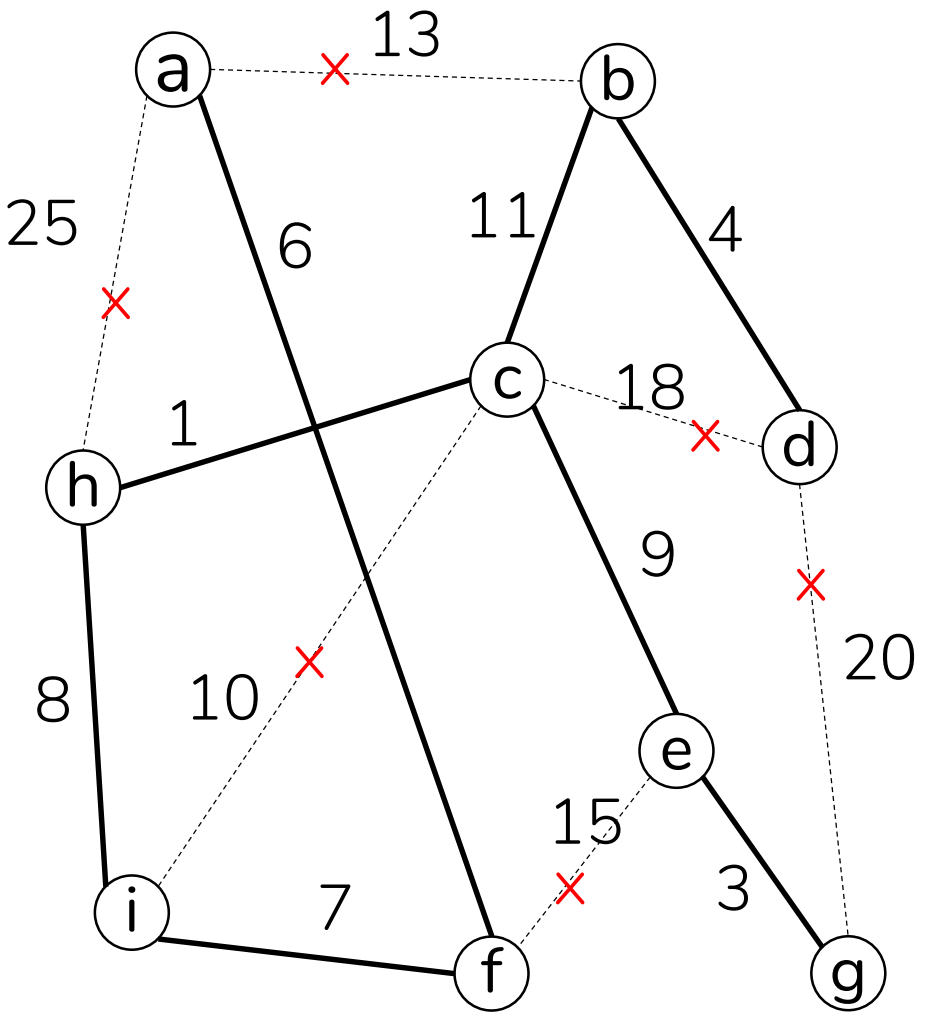
3. find min-cost graph that connects all vertices




Kruskal ( $G = (V, E, c)$)

Keep picking min-cost edges, as long as you don't get a cycle

3. find min-cost graph that connects all vertices




Kruskal's algorithm (gives an MST)

Questions?

# Summary

Basic graph algorithms

0. representing graphs – adjacency matrix, adjacency list
1. traversing graphs – breadth first search

   (gives shortest paths)
2. quickly finding shortest paths – Dijkstra's algorithm
3. cheaply connecting all vertices – Kruskal's algorithm

# Summary

Basic graph algorithms

0. representing graphs – adjacency matrix, adjacency list
1. traversing graphs – breadth first search

   (gives shortest paths)
2. quickly finding shortest paths – Dijkstra's algorithm
3. cheaply connecting all vertices – Kruskal's algorithm

See you after the break!