CSE332: Data Abstractions

Section 7

HyeIn Kim

Winter 2013

Section Agenda

- **Graph Review**
  - Graph Terminologies
  - Graph Representations: matrix & list
  - Topological sort
  - Graph traversal: BFS, DFS
  - Shortest Path: Dijkstra’s Algorithm
- **Project 3 Introduction**
  - Analyzing US census data

Graphs

Graph terminology

Graphs

- **G = (V, E)**

  Contains set of vertices and set of edges

- | V | = number of vertices
- | E | = number of edges

Max | E | for undirected graph

$$|V| + (|V| - 1) + (|V| - 2) + \dots + 1 = |V|(|V| + 1) / 2$$

Max | E | for directed graph

$$|V| + |V| + |V| + \dots + |V| = |V| * |V| = |V|^2$$

Graph Terms

- **Path**

List of vertices $[v_0, v_1, ..., v_n]$, such that $(v_i, v_{i+1}) \in E$ for all $0 \leq i < n$

- Path length = number of edges on path
- Path cost = sum of all edge weights on path

- **Cycle**

A path that begins and ends at the same node

Undirected Graph

- **Edges have no directions**

- **Connected**

  If there is a path between all pairs of vertices

- **Fully Connected**

  If there is an edge between all pairs of vertices

Directed Graph

- **Edges have direction**
- **Weakly Connected**

If there is an undirected path between all pairs of vertices

- **Strongly Connected**

If there is a directed path between all pairs of vertices

- **Fully Connected**

If there is edge (both way) between all pairs of vertices

Graph Representation

Adjacency matrix & Adjacency list

Graph Representation

- **The ‘Best one’ depends on:**

- Graph density

- Common Queries

Insert an edge

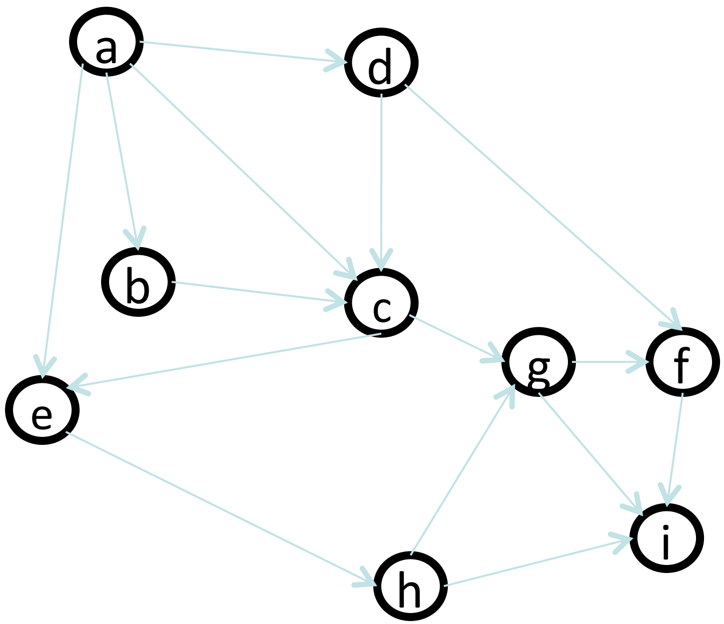
Delete an edge

Find an edge

Compute indegree of a vertex

Compute outdegree of a vertex

Adjacency Matrix



f\t	a	b	c	d	e	f	g	h	i
a		1	1	1	1				
b			1						
c					1		1		
d			1			1			
e								1	
f									1
g						1			1
h							1		1
i									

- **Space Requirement:** $|V|^2$

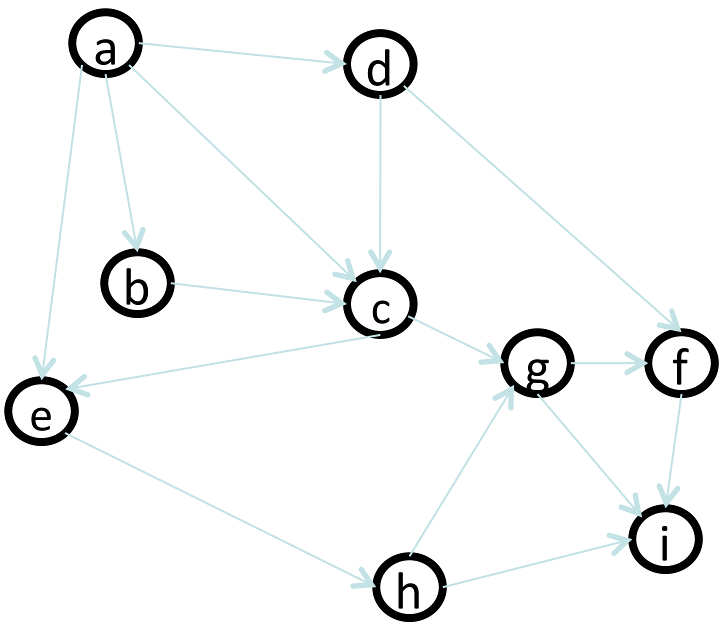
Adjacency Matrix

- **Get indegree:** **O(|V|)**
- **Get outdegree:** **O(|V|)**
- **Find an edge:** **O(1)**
- **Insert an edge:** **O(1)**
- **Delete an edge:** **O(1)**

f\t	a	b	c	d	e	f	g	h	i
a		1	1	1	1				
b			1						
c					1		1		
d			1			1			
e								1	
f									1
g						1			1
h							1		1
i									

- Dense graph |E| >>> |V|, so good for dense graph

Adjacency List



a	b	c	d	e
b	c			
c	e	g		
d	c	f		
e	h			
f	i			
g	f	i		
h	g	i		
i				

- **Space Requirement:** **$O(|V| + |E|)$**

Adjacency List

- Let d = outdegree

- **Get indegree:** O(|V|+|E|)
- **Get outdegree:** O(d), O(1) possible
- **Find an edge:** O(d)
- **Insert an edge:** O(1)
- **Delete an edge:** O(d)

a	b	c	d	e
b	c			
c	e	g		
d	c	f		
e	h			
f	i			
g	f	i		
h	g	i		
i				

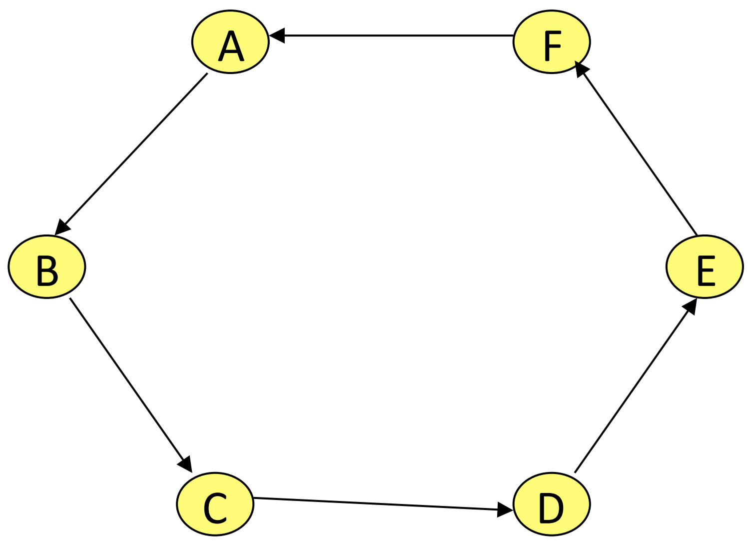
- Sparse graph |V| >>> d, so good for sparse graph

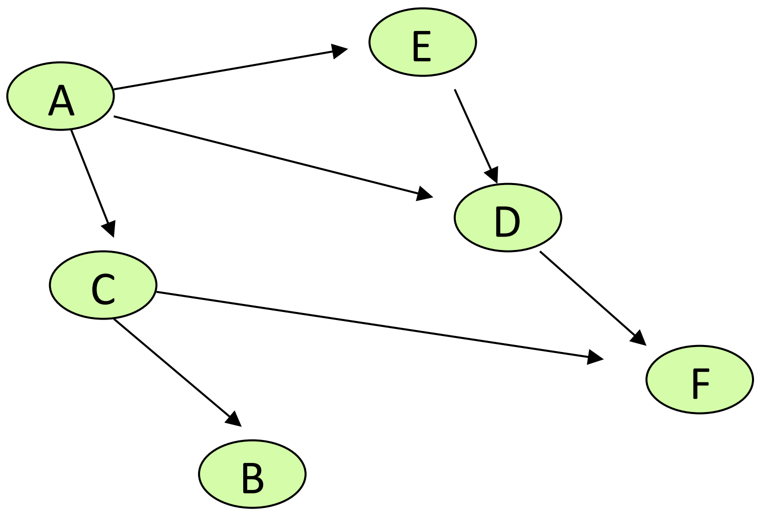
Topological Sort

`Get linear order of tasks with dependencies`

Topological Sort

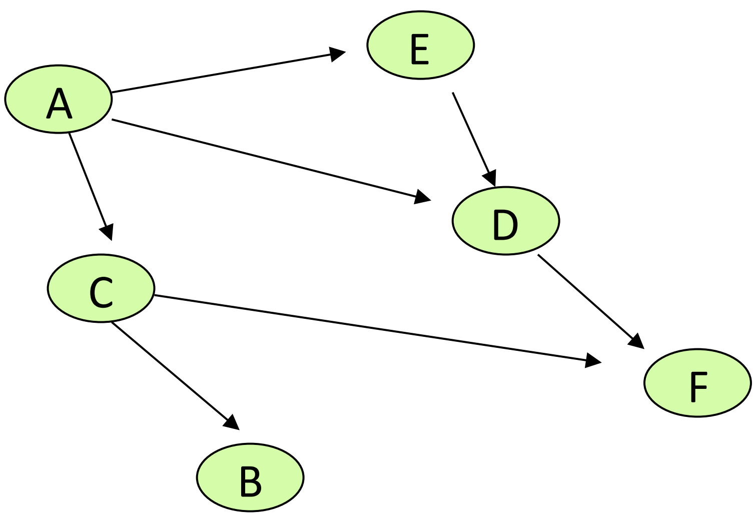
- Given a set of tasks with precedence constraints, find a linear order of the tasks
  - No topological ordering in graph with cycle
  - Possible to have many topological ordering

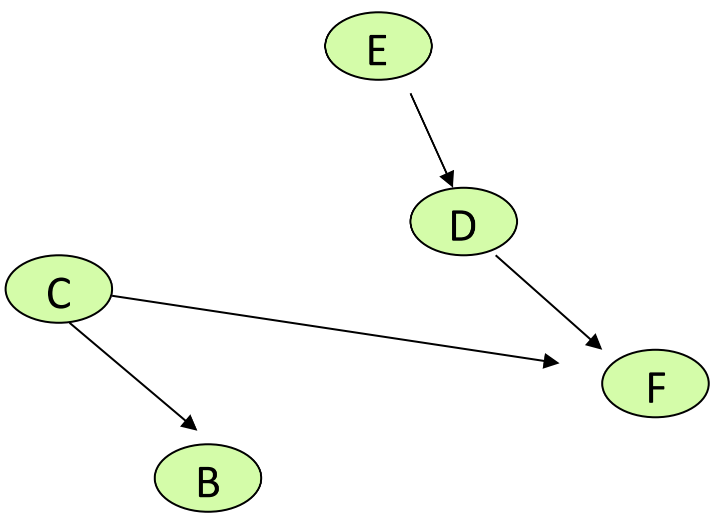




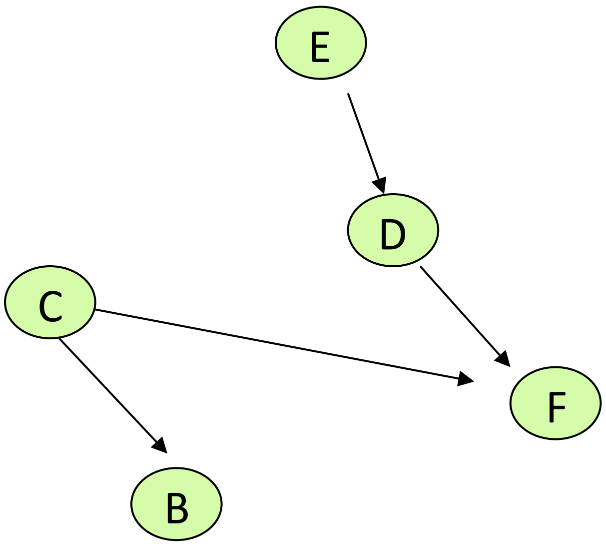
Topological Sort

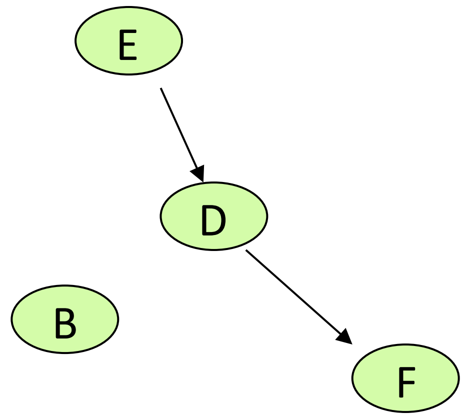
- **Topological sort algorithm**
  - Choose a vertex v with indegree 0
  - Output v & Remove v and all of its edges
  - Repeat until no more vertices left

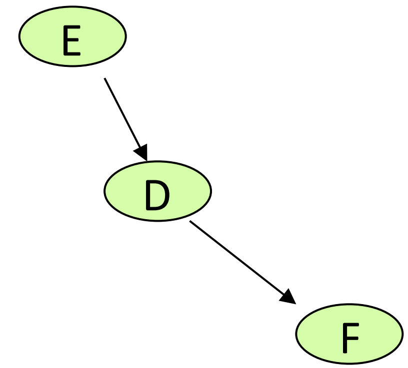


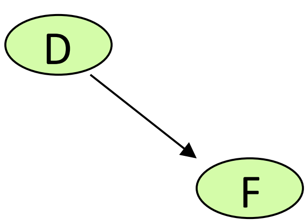


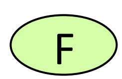
Topological Sort











A C B E D F

Topological Sort

- **Topological sort Runtime**

  - Choose a vertex v with indegree 0

    Single step (No Q / Q): $O(|V|)$ $O(1)$

    Total (No Q / Q): $O(|V|^2)$ $O(|V|)$

  - Output v & Remove v Total: $O(|V|)$

  - Remove all of v's edges Total: $O(|E|)$

- **Total Runtime:** $O(|V|^2+|E|)$ No Queue

  $O(|V|+|E|)$ Queue

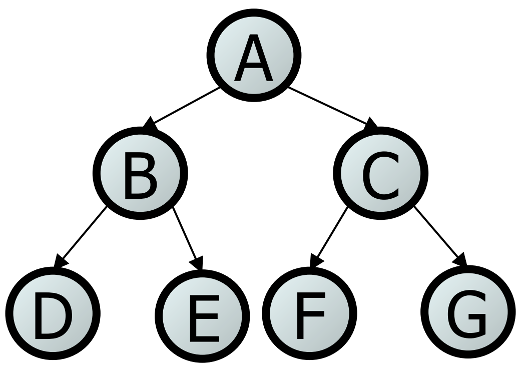
Graph Traversal

BFS & DFS

Breadth First Search

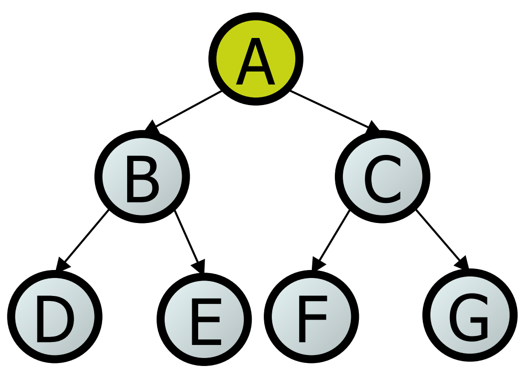
- **Pick the shallowest unmarked node**

  - Use queue, new node go to the end



Start with the root in the queue

Queue	A



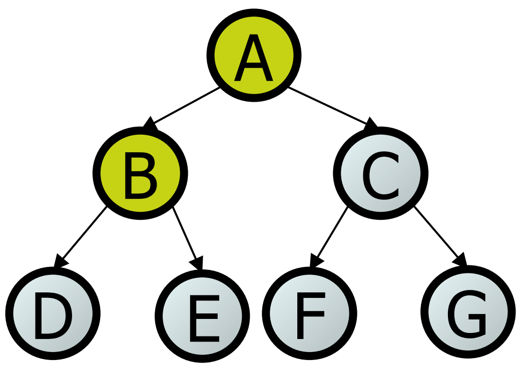
Pop one out, mark it,
put its child into the queue

Queue	B	C

Breadth First Search

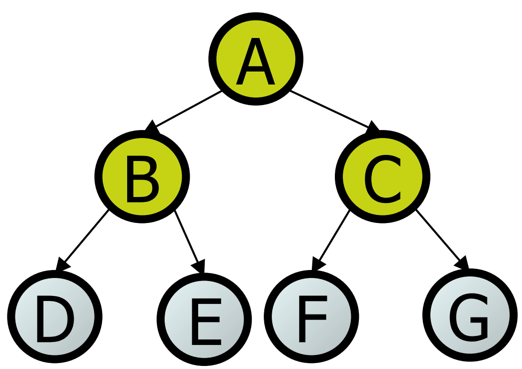
- **Pick the shallowest unmarked node**

  - Use queue, new node go to the end



Pop one out, mark it,
put its child into the queue

Queue	C	D	E



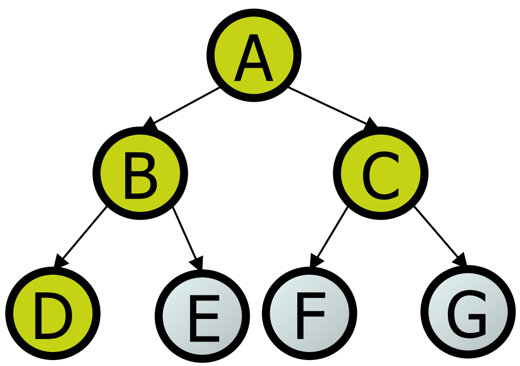
Pop one out, mark it,
put its child into the queue

Queue	D	E	F	G

Breadth First Search

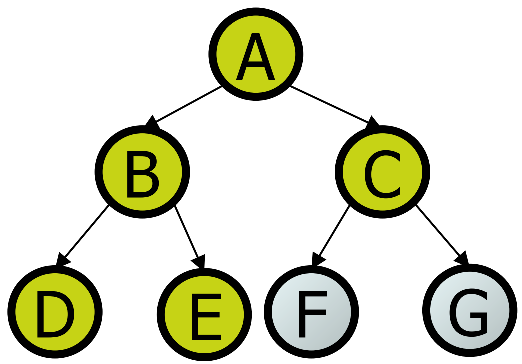
- **Pick the shallowest unmarked node**

  - Use queue, new node go to the end



Pop one out, mark it,
put its child into the queue

Queue	E	F	G



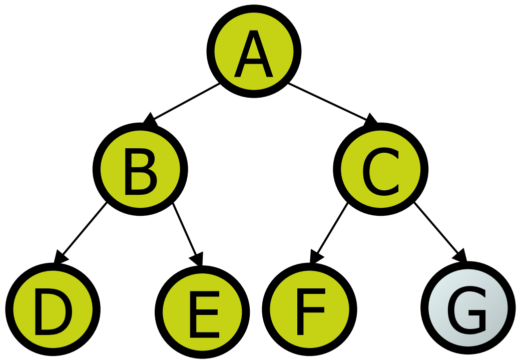
Pop one out, mark it,
put its child into the queue

Queue	F	G

Breadth First Search

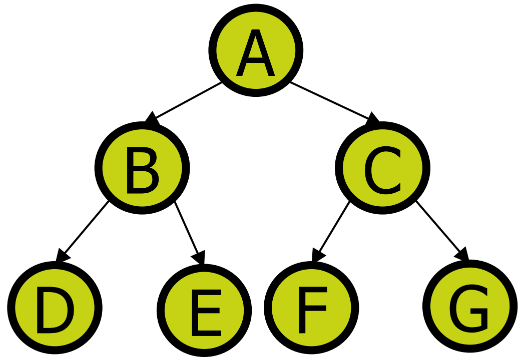
- **Pick the shallowest unmarked node**

  - Use queue, new node go to the end



Pop one out, mark it,
put its child into the queue

Queue	G



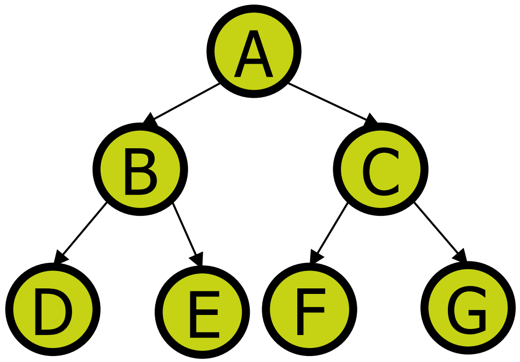
Pop one out, mark it,
put its child into the queue

Queue

Breadth First Search

- **Pick the shallowest unmarked node**

  - Use queue, new node go to the end



The queue is empty, Done!

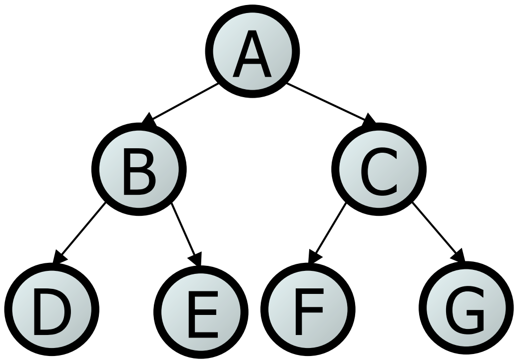
- The order of traversal: A B C D E F G

- Let b = branching factor, h = height
  Space requirement: $O(b^h)$

Depth First Search

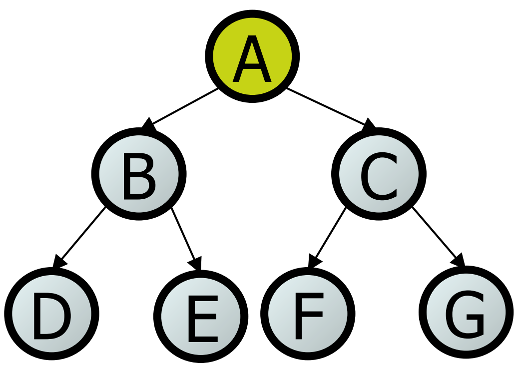
- **Pick the deepest unmarked node**

  - Use stack, new node go to the top



Start with the root in the stack

Stack	A



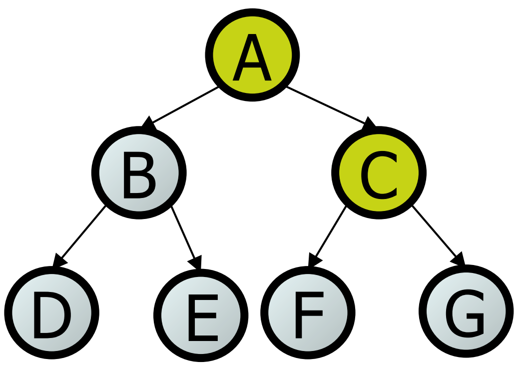
Pop one out, mark it,
put its child into the stack

Stack	B	C

Depth First Search

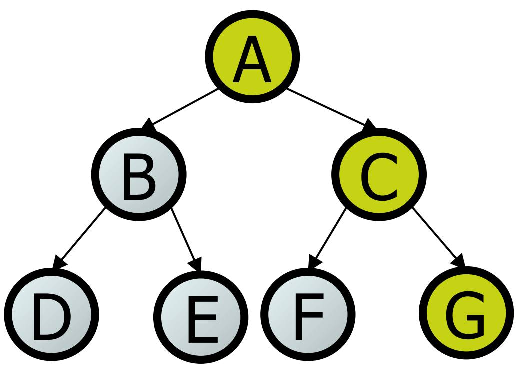
- **Pick the deepest unmarked node**

  - Use stack, new node go to the top



Pop one out, mark it,
put its child into the stack

Stack	B	F	G



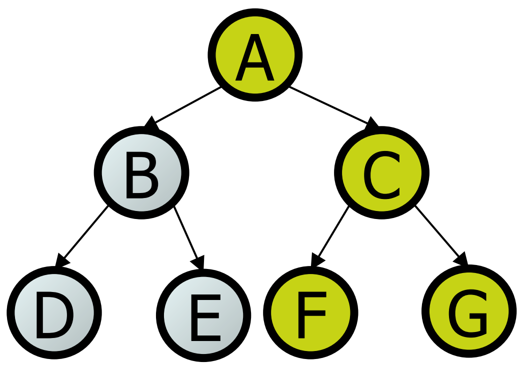
Pop one out, mark it,
put its child into the stack

Stack	B	F

Depth First Search

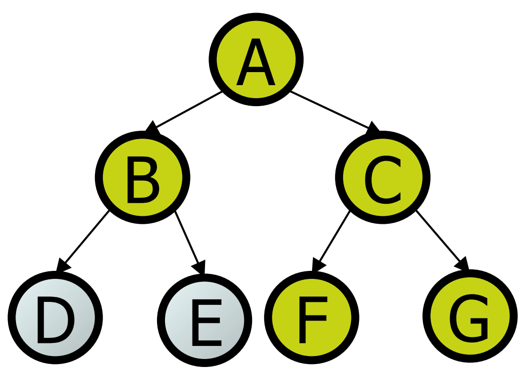
- **Pick the deepest unmarked node**

  - Use stack, new node go to the top



Pop one out, mark it,
put its child into the stack

Stack	B



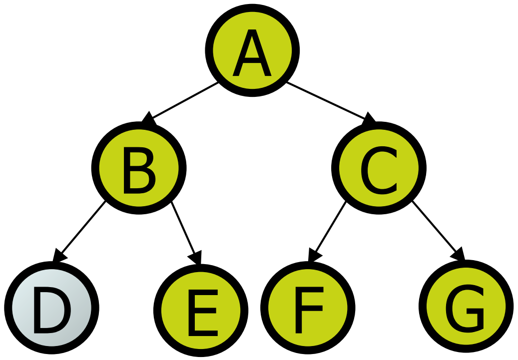
Pop one out, mark it,
put its child into the stack

Stack	D	E

Depth First Search

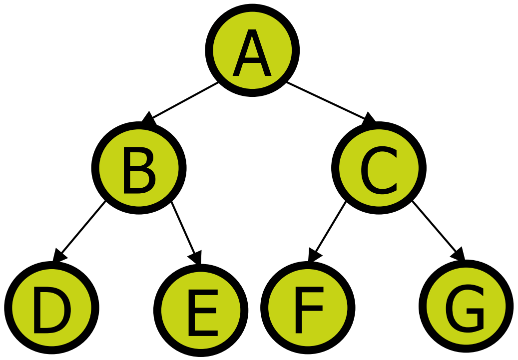
- **Pick the deepest unmarked node**

  - Use stack, new node go to the top



Pop one out, mark it,
put its child into the stack

Stack	D



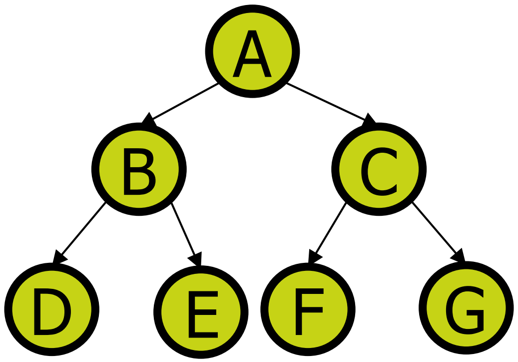
Pop one out, mark it,
put its child into the stack

Stack

Depth First Search

- **Pick the deepest unmarked node**

  - Use stack, new node go to the top



The stack is empty, Done!

- The order of traversal: A C G F B E D

- Let b = branching factor, h = height
  Space requirement: **O(b*h)**

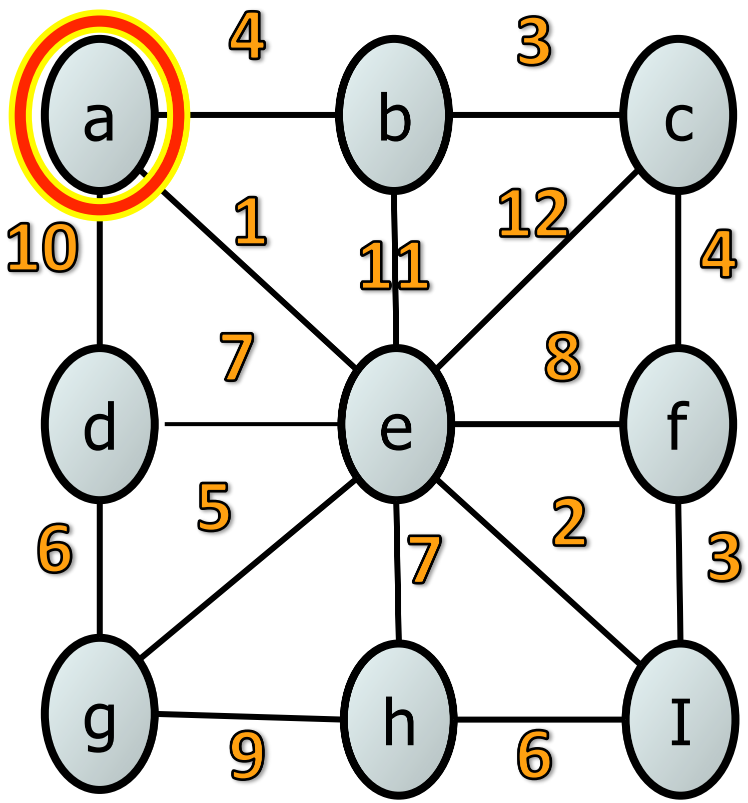
Find Shortest Path

Dijkstra's Algorithm

Dijkstra's Algorithm

Source Node: A

Pick one with shortest distance from source: **A**

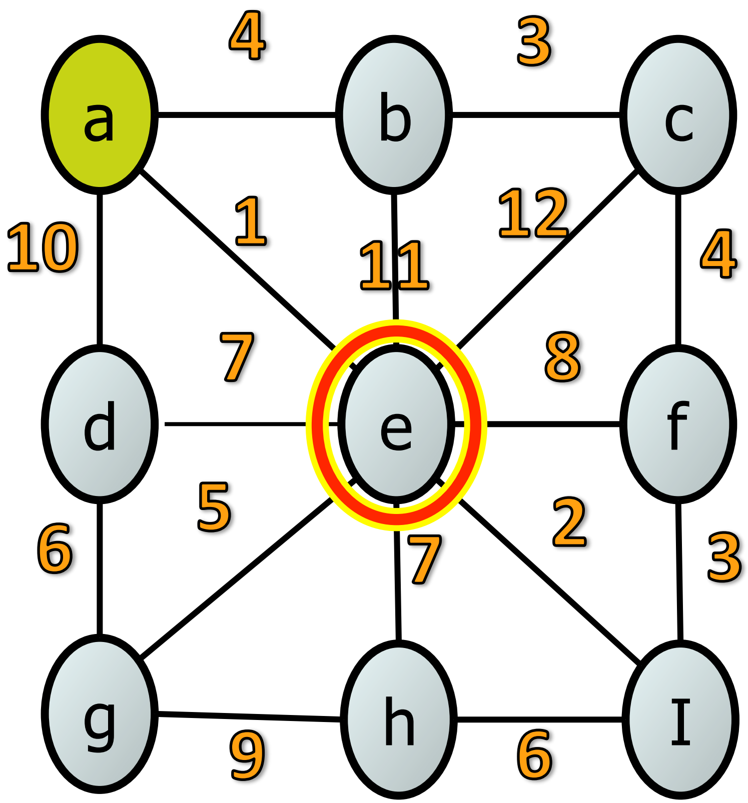


Node	Mark	Dist	Path	Mark	Dist	Path
A		0		1	0	-
B		∞			4	A
C		∞			∞	
D		∞			10	A
E		∞			1	A
F		∞			∞	
G		∞			∞	
H		∞			∞	
I		∞			∞	

Dijkstra's Algorithm

Source Node: A

Pick one with shortest distance from source: **E**

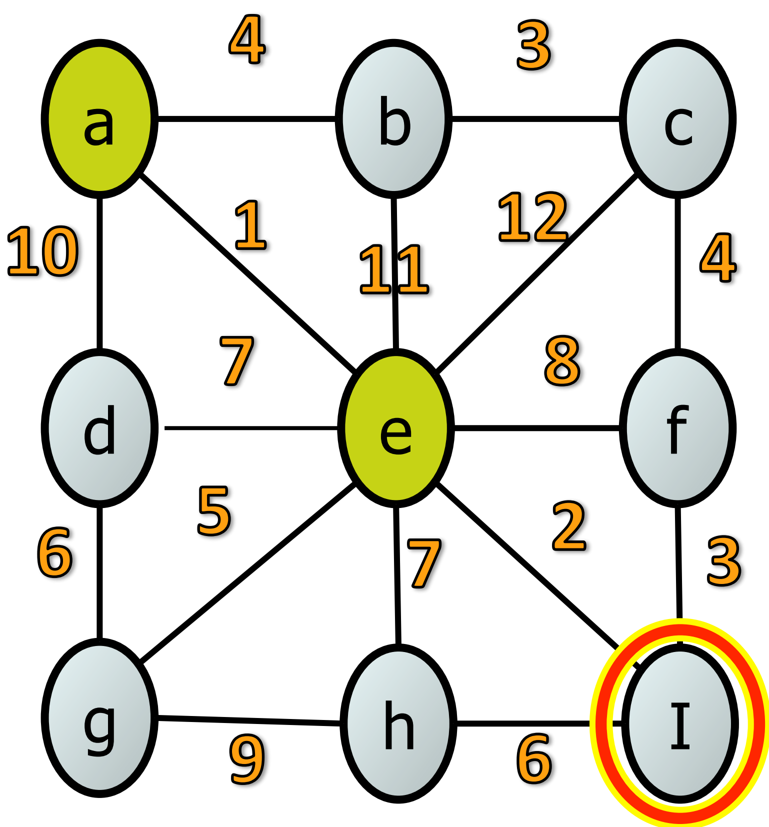


Node	Mark	Dist	Path	Mark	Dist	Path
A	1	0	-	1	0	-
B		4	A		4	A
C		∞			13	E
D		10	A		8	E
E		1	A	1	1	A
F		∞			9	E
G		∞			6	E
H		∞			8	E
I		∞			3	E

Dijkstra's Algorithm

Source Node: A

Pick one with shortest distance from source: **I**

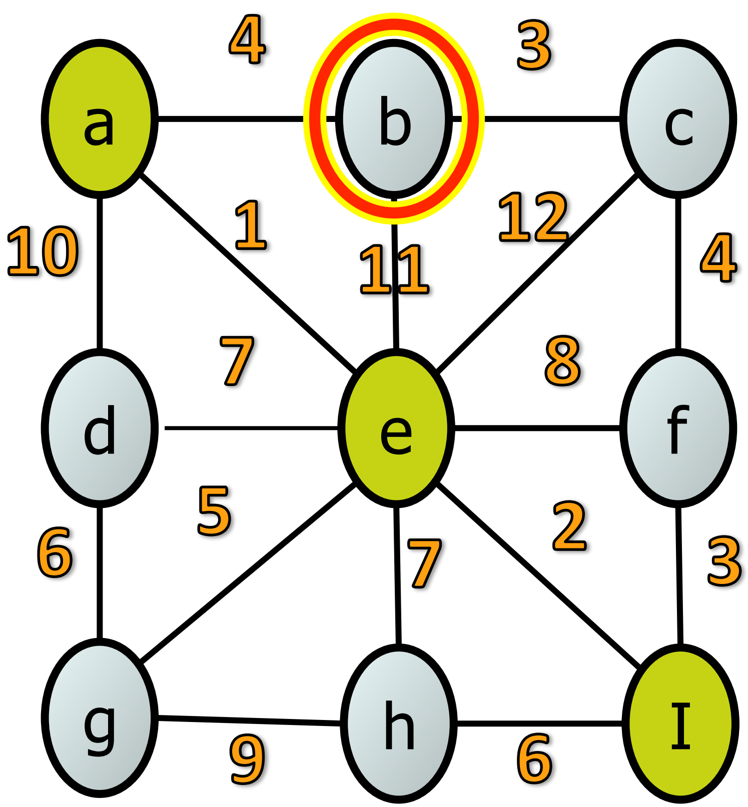


Node	Mark	Dist	Path	Mark	Dist	Path
A	1	0	-	1	0	-
B		4	A		4	A
C		13	E		13	E
D		8	E		8	E
E	1	1	A	1	1	A
F		9	E		6	I
G		6	E		6	E
H		8	E		8	E
I		3	E	1	3	E

Dijkstra's Algorithm

Source Node: A

Pick one with shortest distance from source: **B**

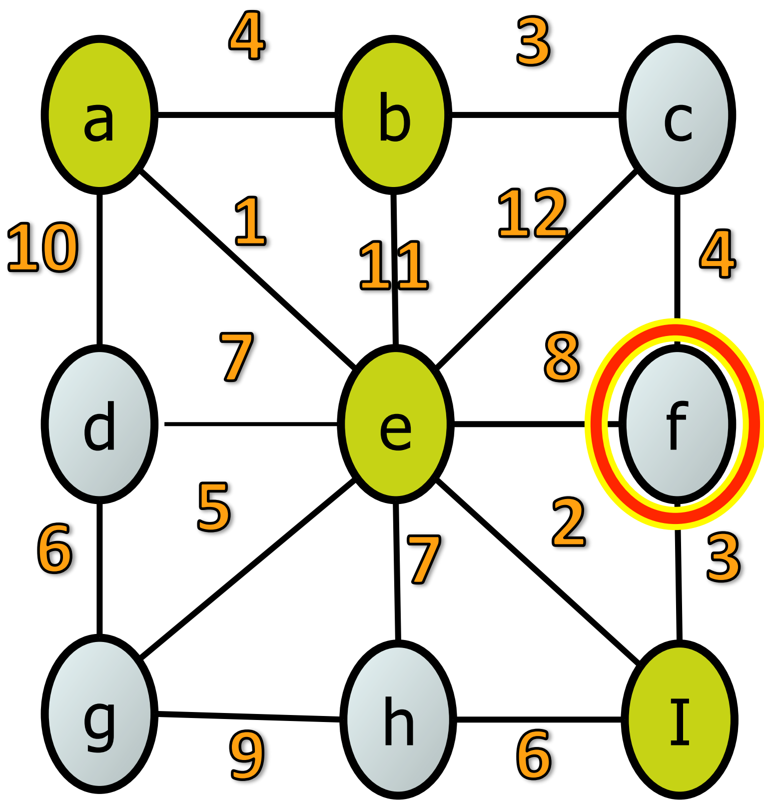


Node	Mark	Dist	Path	Mark	Dist	Path
A	**1**	**0**	**-**	**1**	**0**	**-**
B		4	A	**1**	**4**	**A**
C		13	E		7	B
D		8	E		8	E
E	**1**	**1**	**A**	**1**	**1**	**A**
F		6	I		6	I
G		6	E		6	E
H		8	E		8	E
I	**1**	**3**	**E**	**1**	**3**	**E**

Dijkstra's Algorithm

Source Node: A

Pick one with shortest distance from source: **F**

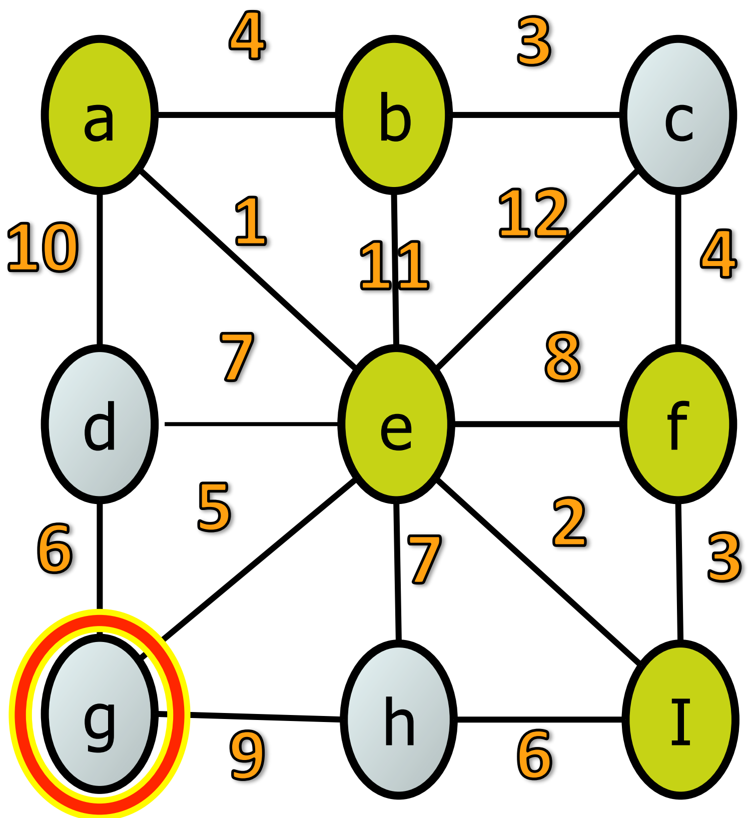


Node	Mark	Dist	Path	Mark	Dist	Path
A	1	0	-	1	0	-
B	1	4	A	1	4	A
C		7	B		7	B
D		8	E		8	E
E	1	1	A	1	1	A
F		6	I	1	6	I
G		6	E		6	E
H		8	E		8	E
I	1	3	E	1	3	E

Dijkstra's Algorithm

Source Node: A

Pick one with shortest distance from source: **G**

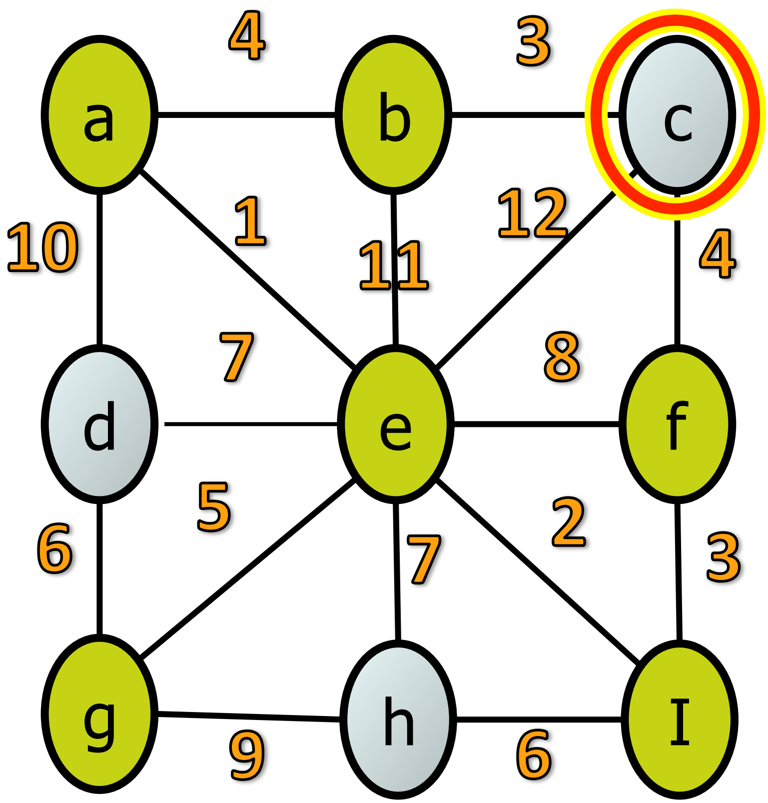


Node	Mark	Dist	Path	Mark	Dist	Path
A	1	0	-	1	0	-
B	1	4	A	1	4	A
C		7	B		7	B
D		8	E		8	E
E	1	1	A	1	1	A
F	1	6	I	1	6	I
G		6	E	1	6	E
H		8	E		8	E
I	1	3	E	1	3	E

Dijkstra's Algorithm

Source Node: A

Pick one with shortest distance from source: **C**

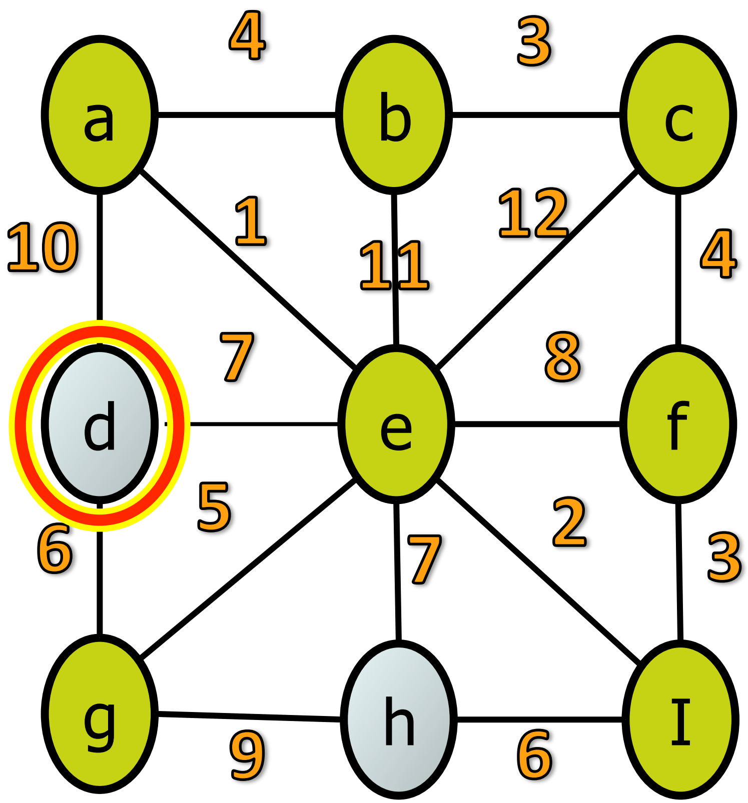


Node	Mark	Dist	Path	Mark	Dist	Path
A	1	0	-	1	0	-
B	1	4	A	1	4	A
C		7	B	1	7	B
D		8	E		8	E
E	1	1	A	1	1	A
F	1	6	I	1	6	I
G	1	6	E	1	6	E
H		8	E		8	E
I	1	3	E	1	3	E

Dijkstra's Algorithm

Source Node: A

Pick one with shortest distance from source: **D**

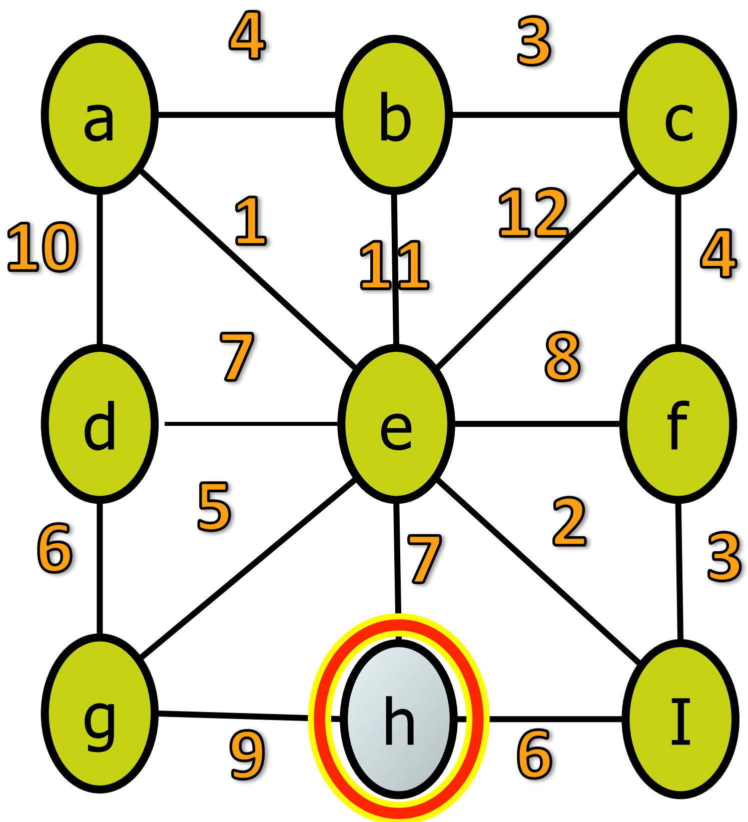


Node	Mark	Dist	Path	Mark	Dist	Path
A	1	0	-	1	0	-
B	1	4	A	1	4	A
C	1	7	B	1	7	B
D		8	E	1	8	E
E	1	1	A	1	1	A
F	1	6	I	1	6	I
G	1	6	E	1	6	E
H		8	E		8	E
I	1	3	E	1	3	E

Dijkstra’s Algorithm

Source Node: A

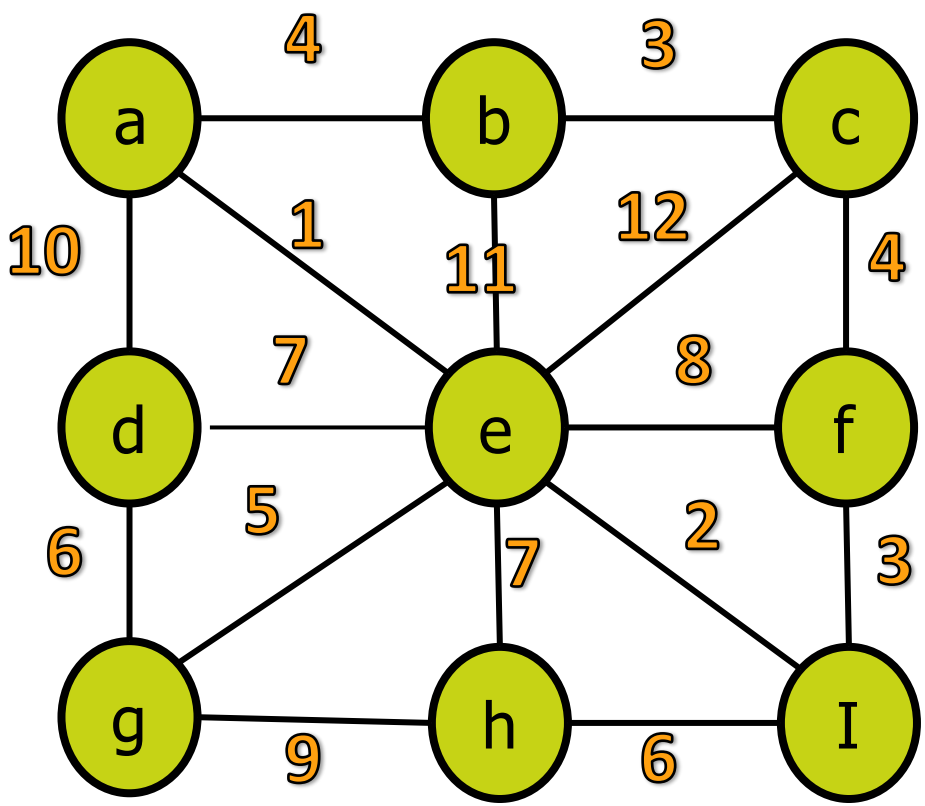
Pick one with shortest distance from source: **H**



Node	Mark	Dist	Path	Mark	Dist	Path
A	1	0	-	1	0	-
B	1	4	A	1	4	A
C	1	7	B	1	7	B
D	1	8	E	1	8	E
E	1	1	A	1	1	A
F	1	6	I	1	6	I
G	1	6	E	1	6	E
H		8	E	1	8	E
I	1	3	E	1	3	E

Dijkstra's Algorithm

Source Node: A



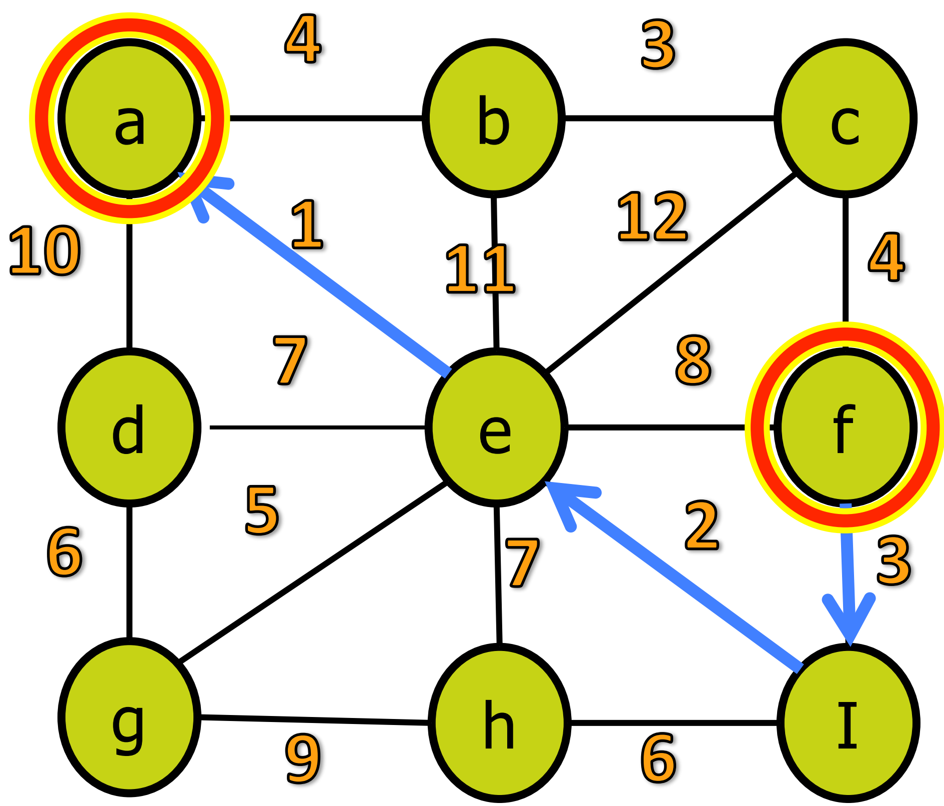
Done!

Node	Mark	Dist	Path
A	1	0	-
B	1	4	A
C	1	7	B
D	1	8	E
E	1	1	A
F	1	6	I
G	1	6	E
H	1	8	E
I	1	3	E

Dijkstra's Algorithm

Source Node: A

Find shortest path from F to A



Node	Mark	Dist	Path
A	1	0	-
B	1	4	A
C	1	7	B
D	1	8	E
E	1	1	A
F	1	6	I
G	1	6	E
H	1	8	E
I	1	3	E

Dijkstra's Algorithm

- **Dijkstra's Algorithm Runtime**

  - Initializing each node $O(|V|)$

  - Pick smallest v & Mark v

    Single step (No PQ / PQ): $O(|V|)$ $O(\log |V|)$

    Total (No Q / Q): $O(|V|^2)$ $O(|V|*\log|V|)$

  - Update cost of all neighbors of v

    Total (No PQ): $O(|E|)$

    Total (PQ): $O(|E|*\log|V|)$

- **Total Runtime:** $O(|V|^2+|E|)$ No Priority Queue

  $O((|V|+|E|)*\log|v|)$ Priority Queue

Dijkstra's Algorithm

- **Total Runtime:** $O(|V|^2+|E|)$ No Priority Queue
  $O((|V|+|E|)*\log|v|)$ Priority Queue

  - Sparse graph: $|V| >>> |E|$, $O(|V|*\log|V|)$
    Better with Priority Queue

  - Dense graph: $|E| >>> |V|$, $O(|E|*\log|V|)$
    => $O(|V|^2*\log|V|)$
    Better without Priority Queue

Project 3

`Where are the people?`

Project 3

- **US Census Bureau**

state	county	population	latitude	longitude
01	001	1188	+32.48103	-86.486781
01	001	733	+32.465409	-86.486868
01	001	857	+32.479465	-86.473928
31	055	772	+41.262168	-95.982255
31	055	978	+41.265888	-95.963541
31	055	743	+41.261542	-95.968029
31	055	756	+41.265632	-95.970912
72	153	1827	+18.048574	-66.88596
72	153	3456	+18.058711	-66.875287
72	153	1444	+18.061441	-66.8652
72	153	2097	+18.04479	-66.865791
72	153	2677	+18.038417	-66.865411

Treat towns as points

Project 3

- **US Census Bureau**

1. Divide US with X by Y Grid
2. Compute Population in selected Rectangle

Project 3

- **US Census Bureau**

Population: 26360678

Percentage of total US: 9.24%

Project 3

- **Five different implementation**

1. Simple & Sequential
2. Simple & Parallel

Phase A

3. Smarter & Sequential
4. Smarter & Parallel
5. Smarter & Lock-Based

Phase B

- **Experiments & Write up** **Phase C**

- Compare 5 versions with different queries