5CCS2FC2: Foundations of Computing II

# The Cook-Levin Theorem & Introduction to Graph Algorithms

## Week 4

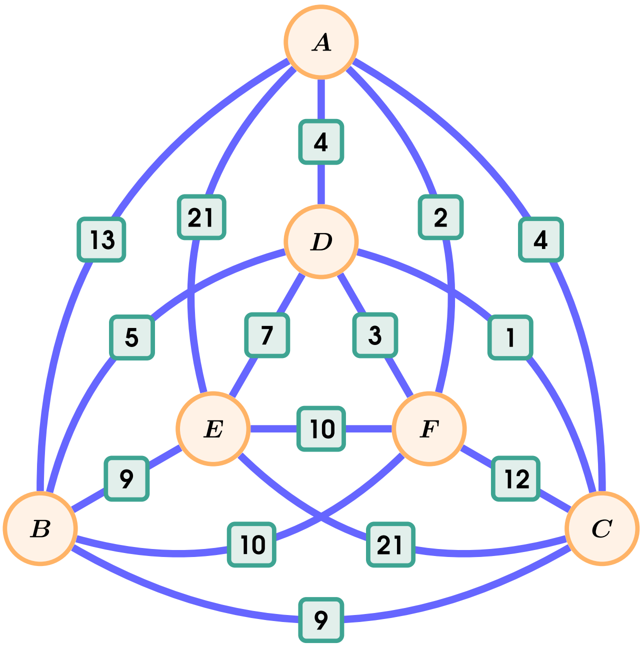
**Dr Christopher Hampson**

*Department of Informatics*

*King's College London*

## Warm-up : Find the shortest path

- Find the shortest path though the following graph that visits each vertex **exactly once** and returns to the starting point.




$$
\begin{array}{c}
\\
A \\ B \\ C \\ D \\ E \\ F
\end{array}
\begin{array}{c}
\begin{array}{cccccc} A & B & C & D & E & F \end{array} \\
\begin{bmatrix}
0 & 13 & 4 & 4 & 7 & 2 \\
13 & 0 & 9 & 13 & 9 & 10 \\
4 & 9 & 0 & 1 & 21 & 12 \\
4 & 13 & 1 & 0 & 7 & 3 \\
7 & 9 & 21 & 7 & 0 & 10 \\
2 & 10 & 12 & 3 & 10 & 0
\end{bmatrix}
\end{array}
$$

# List of NP-complete Problems

- **(Incomplete) List of NP-complete Problems**
  - The Boolean Satisfiability Problem ✓
  - The Graph Colouring Problem ✓
  - The Clique problem ✓
  - The Hamiltonian Cycle Problem
  - The Travelling Salesman Problem (TSP)
  - The Knapsack Problem

`https://en.wikipedia.org/wiki/List_of_NP-complete_problems`

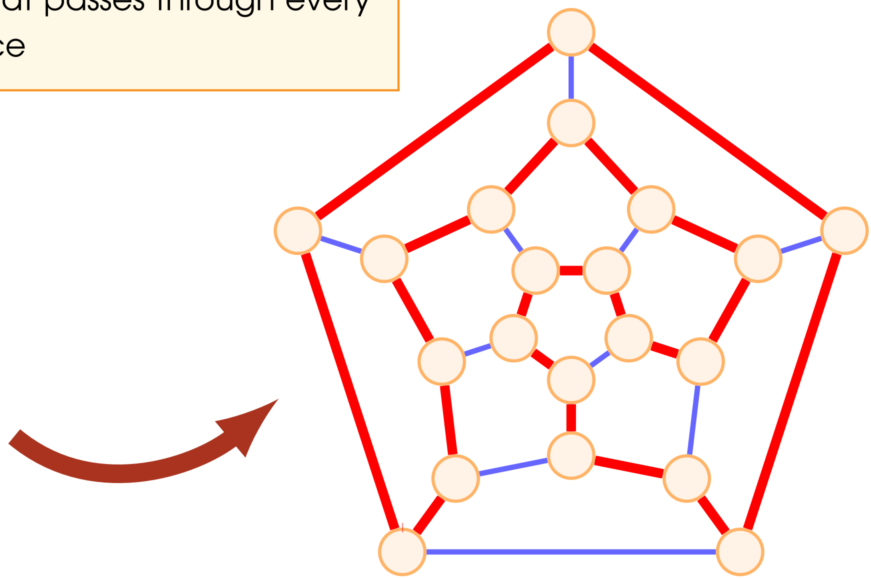
# More NP-complete Problems

## The Hamiltonian Cycle Problem

**Input)** An undirected graph $G = (V, E)$

**Output)** **True** if and only if there is a *closed path* through $G$ that passes through every vertex exactly once

**Has a Hamiltonian Cycle**



# The Travelling Salesman Problem

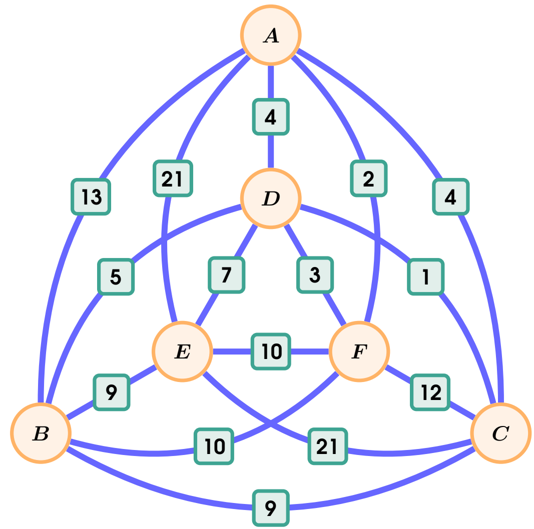
**Input)** A complete weighted graph $G = (V, d)$, where $d : V \times V \to \mathbb{R}$, and target distance $D > 0$.

**Output)** **True** if and only if there is a *closed path* through $G$ of length $< D$ that passes through every vertex exactly once

$D = 75?$ **True**

$D = 20?$ **False**

$D = 50?$ **True**



## The Knapsack Problem

**Input)** A collection of items $I = \{1, 2, \ldots, n\}$ each with weight $w_i$ and value $v_i$, a target value $V$ and a maximum capacity $W$.

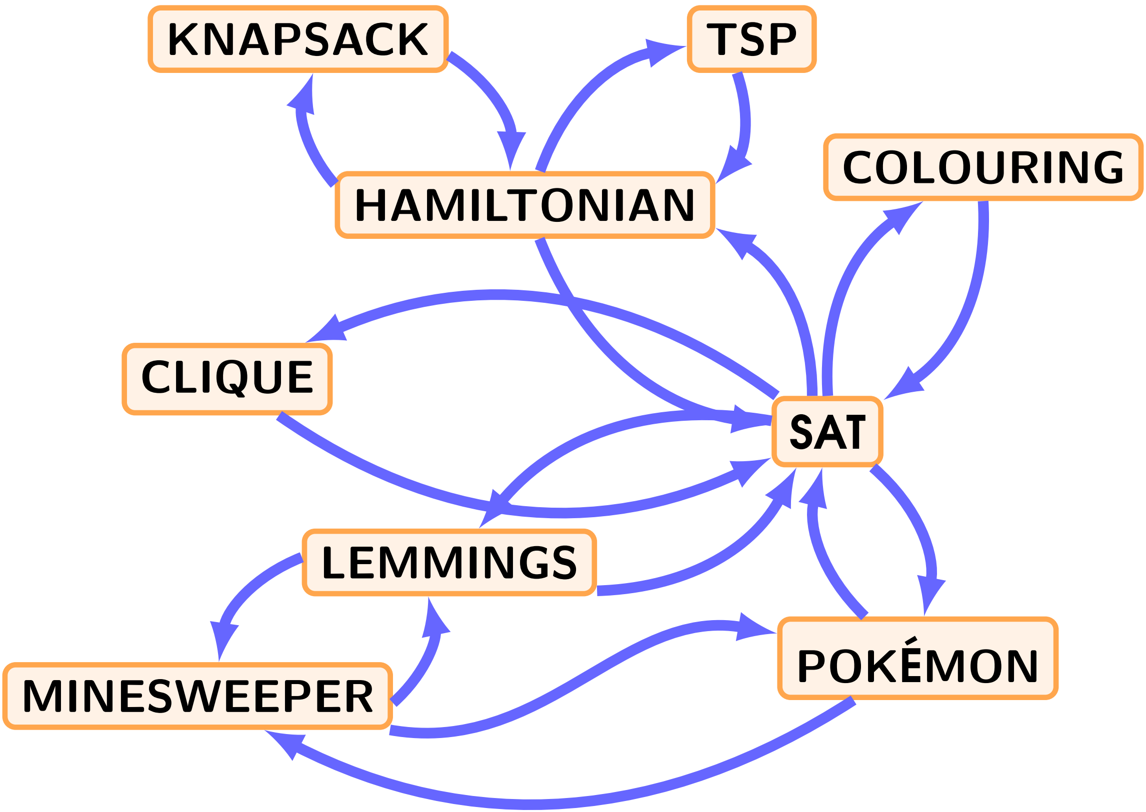
**Output)** **True** if and only if there is subset of items $S \subseteq I$ such that $\sum_{i \in S} v_i > V$ and $\sum_{i \in S} w_i < W$.

| Item 1 | Item 2 | Item 3 |
|---|---|---|
| Weight : 10 | Weight : 20 | Weight : 30 |
| Value : £ 60 | Value : £ 100 | Value : £ 120 |

$V = £\,270$, $W = 70$ — **True**

$V = £\,200$, $W = 40$ — **False**

## Relationships between NP-complete Problems



# The Cook-Levin Theorem

## (The Existence of NP-Complete Problems)

## The Cook-Levin Theorem

> **Theorem (The Cook-Levin Theorem)** The Boolean Satisfiability problem **SAT** is **NP**-complete.

(*i.e.* SAT is among the hardest problems in NP)

**Proof (sketch):** **(you do NOT need to memorise this proof)**

We want to show that **every** problem $X \in$ **NP** can be reduced to **SAT**.

**Step 1)** Let $X \in$ **NP** be **any problem** belonging to the class **NP**.

By **definition** there is some NDTM $\mathcal{M}$ such that

> $\mathcal{M}$ has a polynomial-length computation that accepts $w$ $\iff$ $w \in X$

(for all input words $w \in \Sigma^*$)

# The Cook-Levin Theorem

**Step 2)** For each **state** $q \in Q$, **symbol** $a \in \Sigma$ and **integers** $i, t \leq T(n)$

| | | |
|---|---|---|
| $\textbf{cell}_{i,t}(a)$ | $=$ | Cell $i$ of the $\mathcal{M}$'s tape contains symbol $a$ at time $t$ |
| $\textbf{head}_{i,t}$ | $=$ | The tape head is in position $i$ at time $t$ |
| $\textbf{state}_{q,t}$ | $=$ | The machine is in state $q$ at time $t$ |

**(where $T(n) \in O(n^k)$ is the worst-case termination-time of $\mathcal{M}$)**

**Step 3)** With these **propositional variables**, we make the following claim:

**Claim:** We can write down a propositional formula that describes the behaviour of a NDTM on a given input

$$F_{\mathcal{M},w} \quad = \quad \mathcal{M} \text{ has a computation that accepts } w$$

## The Cook-Levin Theorem

**Step 4)** We then have that

$$\boxed{w \in X \qquad \Longleftrightarrow \qquad F_{\mathcal{M},w} \text{ if satisfiable}}$$

which is to say that $X \leq_p$ **SAT**.

**Step 5)** Since we have assumed **nothing special** about $X$ other than it can be solved by a NDTM in polynomial time (and space), we must have that

$$\boxed{X \leq_p \textbf{SAT} \qquad \text{for all } X \in \textbf{NP}}$$

which is to say that **SAT** is **NP-complete**

**Q.E.D.**

# Graph Algorithms

## Data Structures and Representaions

- **Data Structures for Graphs**
  - The type of **data structure** used to store a graph can affect the effeciency of your algorithms, and memory requirments

### List of Edges

| Edges | Edges |
|---|---|
| $(1,5)$ | $(2,1)$ |
| $(3,4)$ | $(5,2)$ |
| $(1,2)$ | $(5,1)$ |
| $(2,4)$ | $(3,2)$ |
| $(2,3)$ | $(4,2)$ |
| $(2,5)$ | $(4,3)$ |

### Adjacency List

| Vertex | Successors |
|---|---|
| 1 | $2,5$ |
| 2 | $1,3,4,5$ |
| 3 | $2,4$ |
| 4 | $2,3,5$ |
| 5 | $1,2,4$ |

### Adjacency Matrix

$$\begin{array}{c c} & \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

# Breadth-First-Search

## Breadth-First-Search (BFS)

**Step 1)** Select a root node $r \in V$ and set $d(r) = 0$.

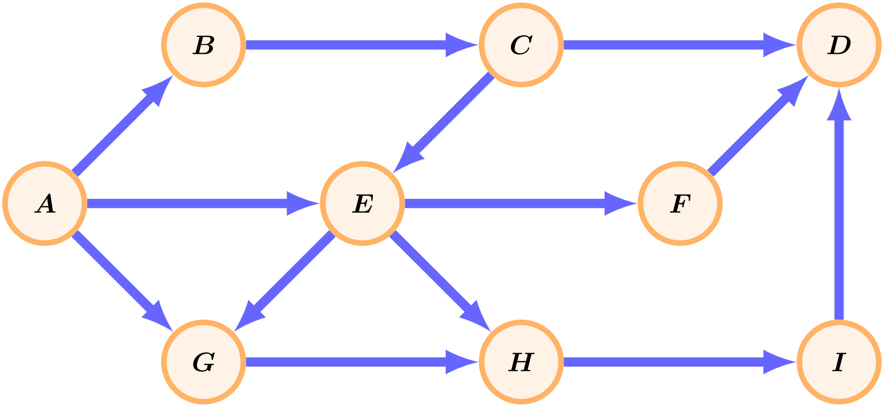
**Step 2)** Add $v$ to a queue.

**Step 3)** De-queue the first element $u$ from the queue.

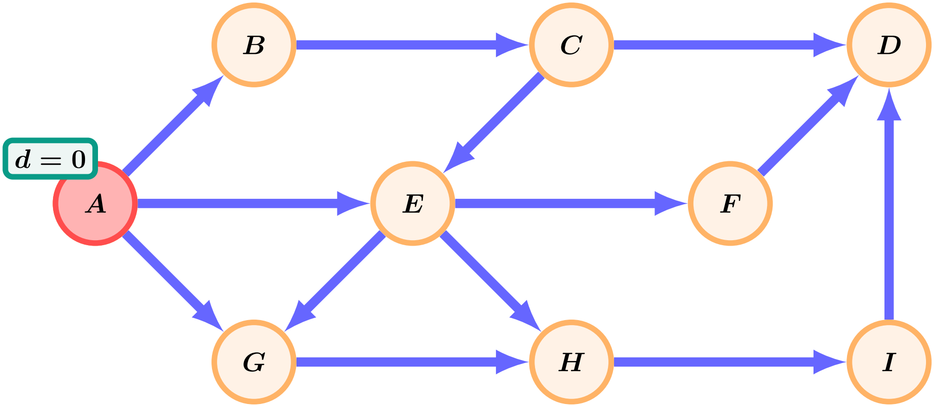
**Step 4)** For each vertex $v$ adjacent to $u$, if $d(v)$ is not yet defined, then set $d(v) = d(u)+1$ and add $v$ to the queue.

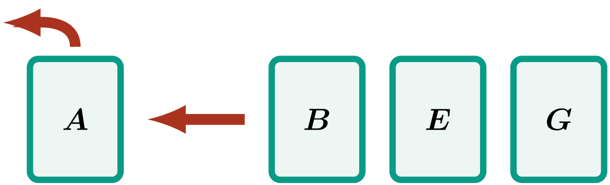
**Step 5)** Repeat from Step 3 until the queue is empty.

# Breadth-First-Search

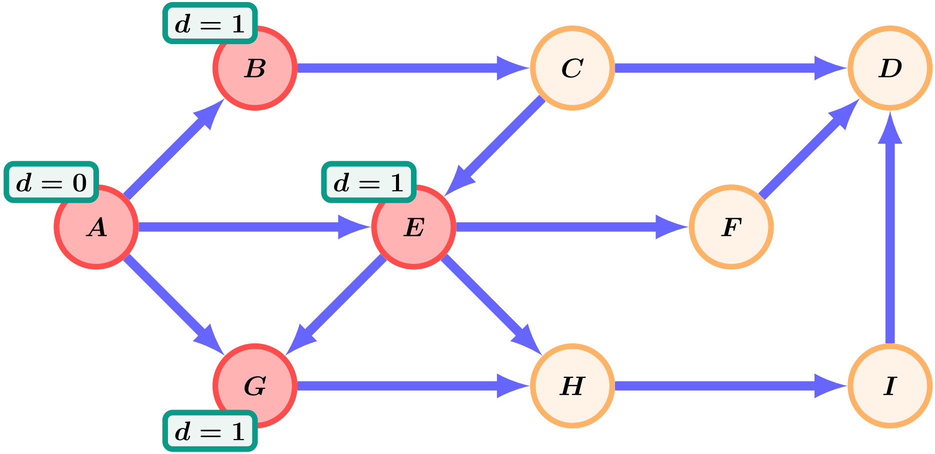


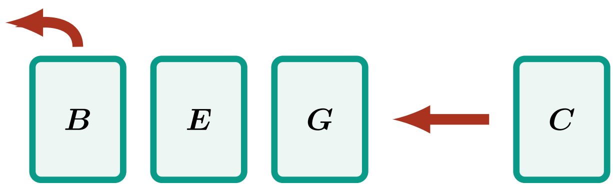
# Breadth-First-Search



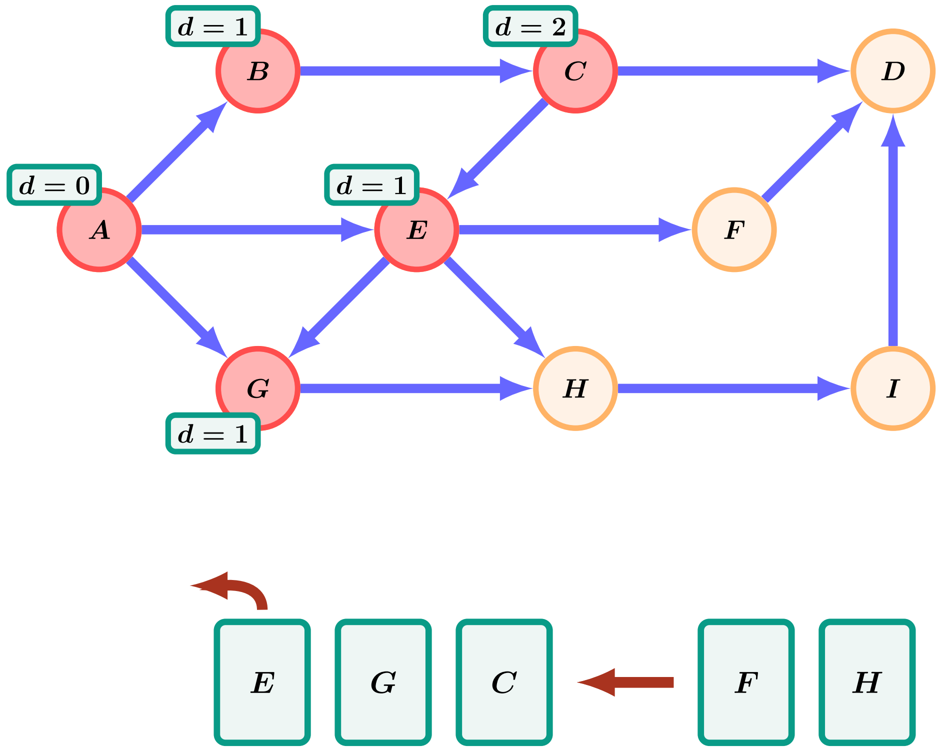



# Breadth-First-Search

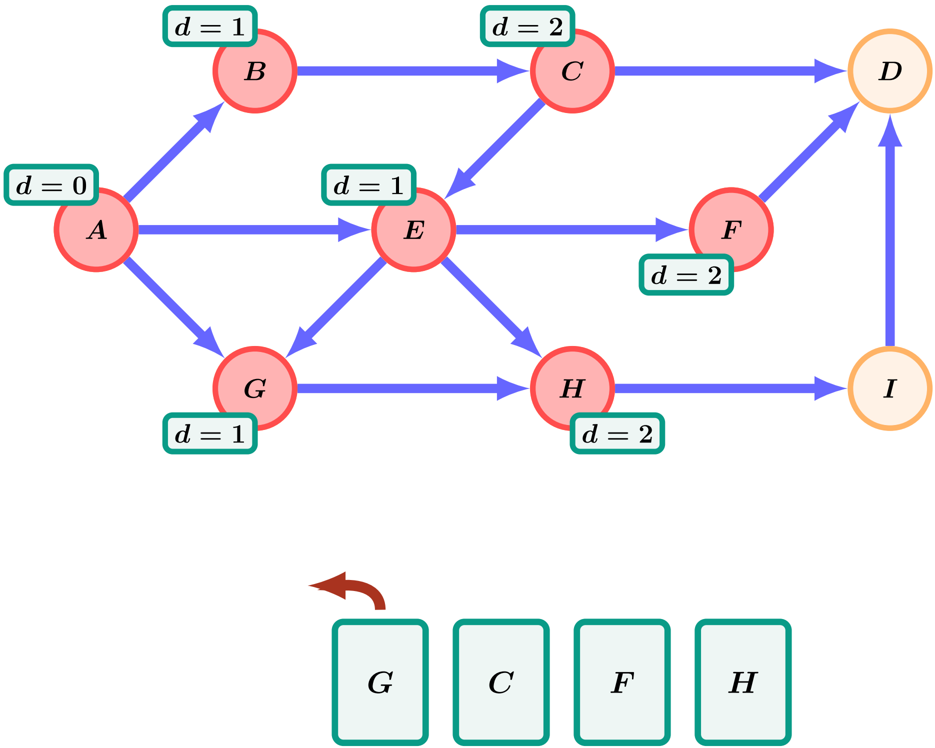


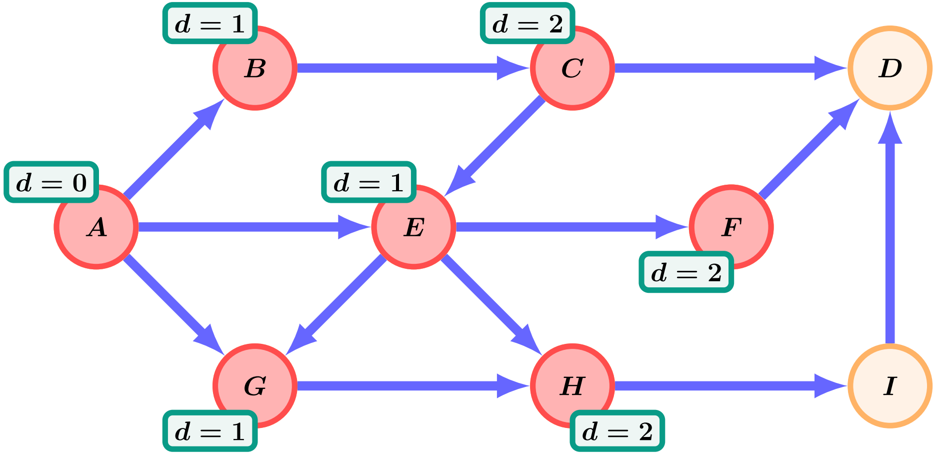
# Breadth-First-Search

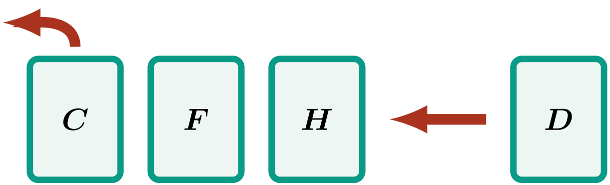


# Breadth-First-Search

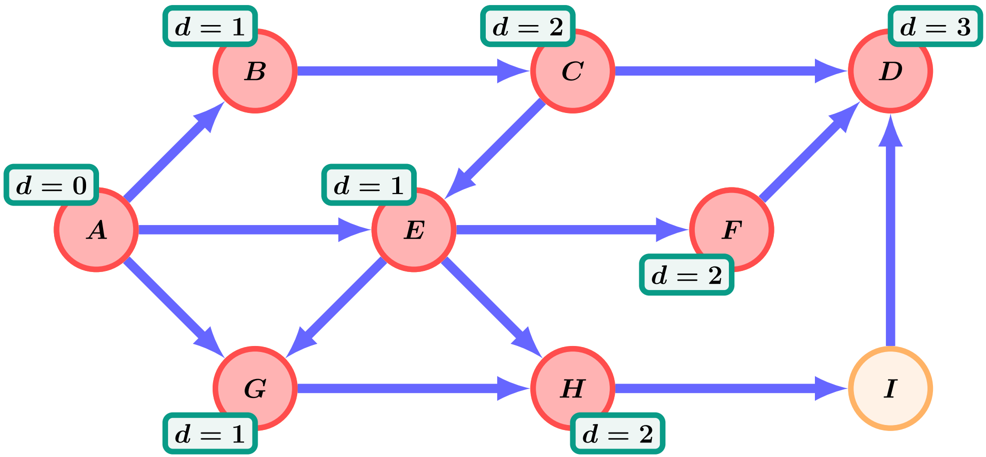


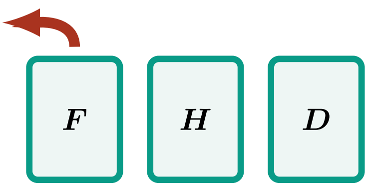
## Breadth-First-Search



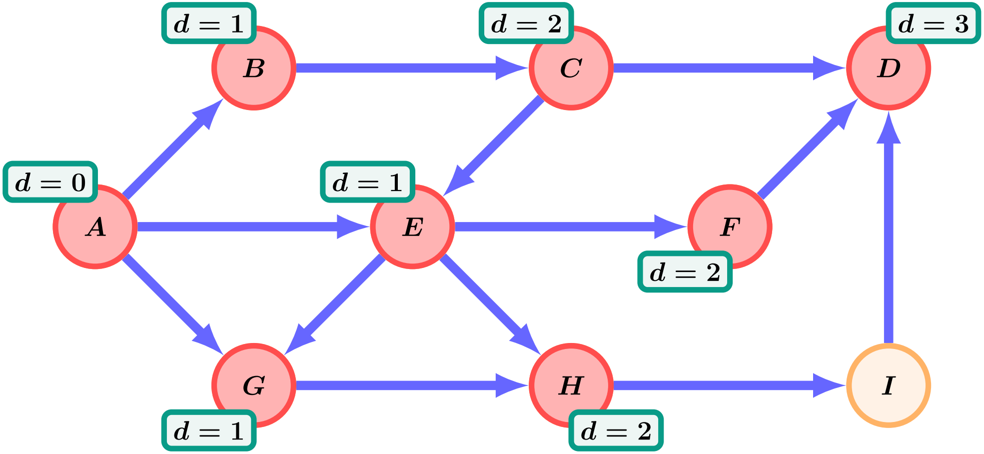



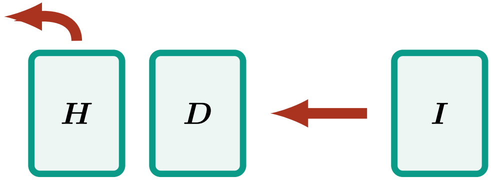
# Breadth-First-Search



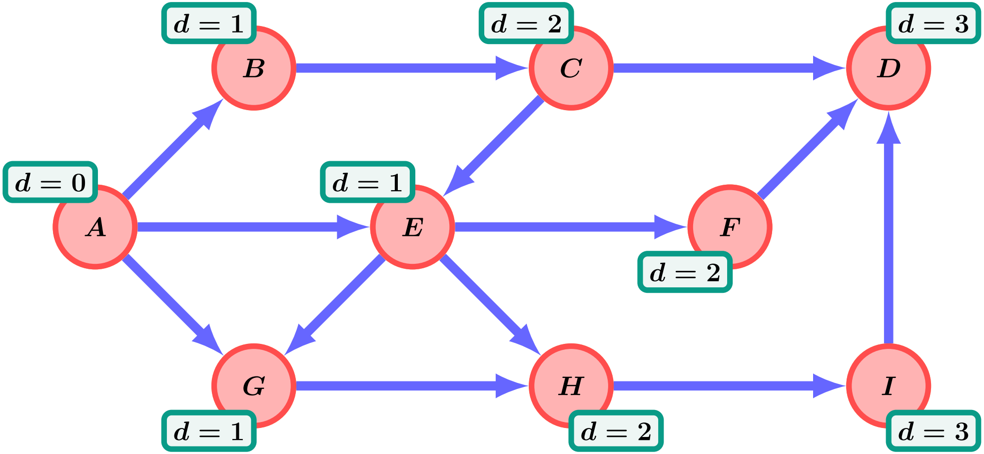



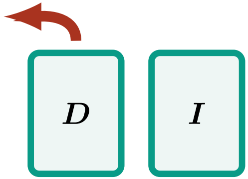
# Breadth-First-Search



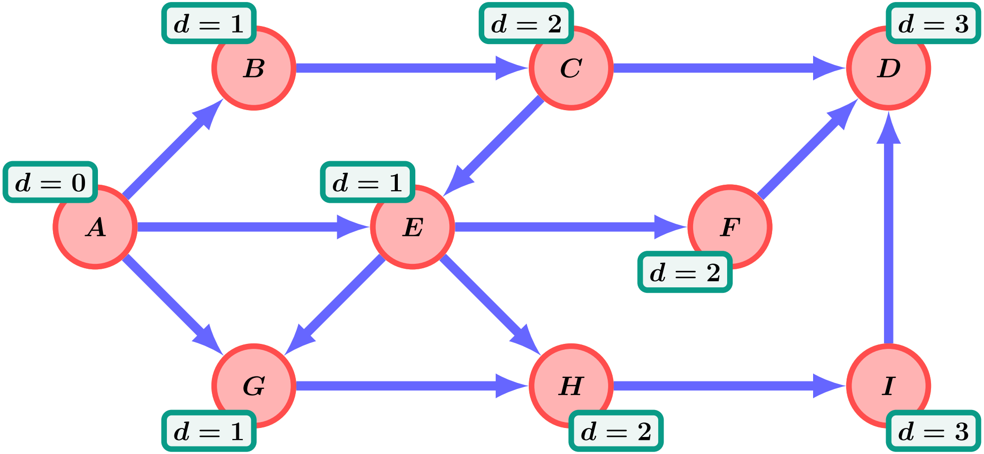



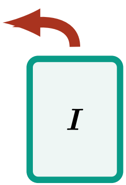
## Breadth-First-Search



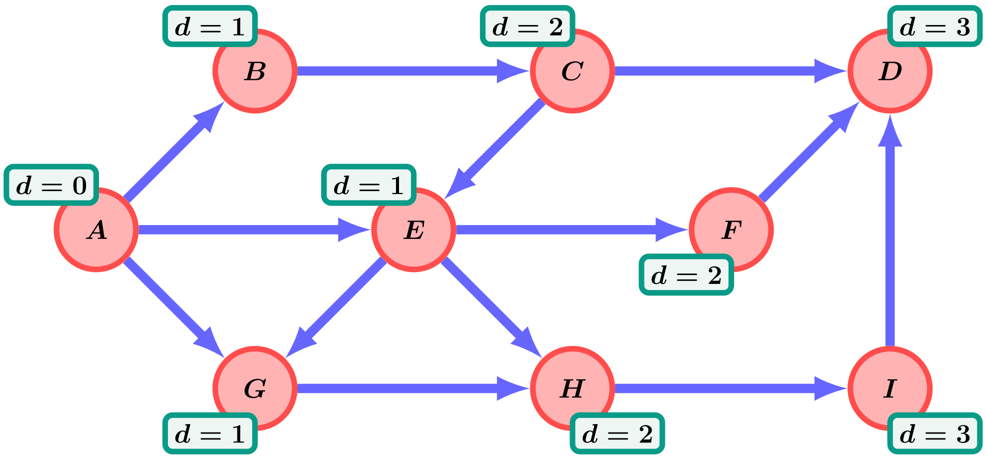



# Breadth-First-Search

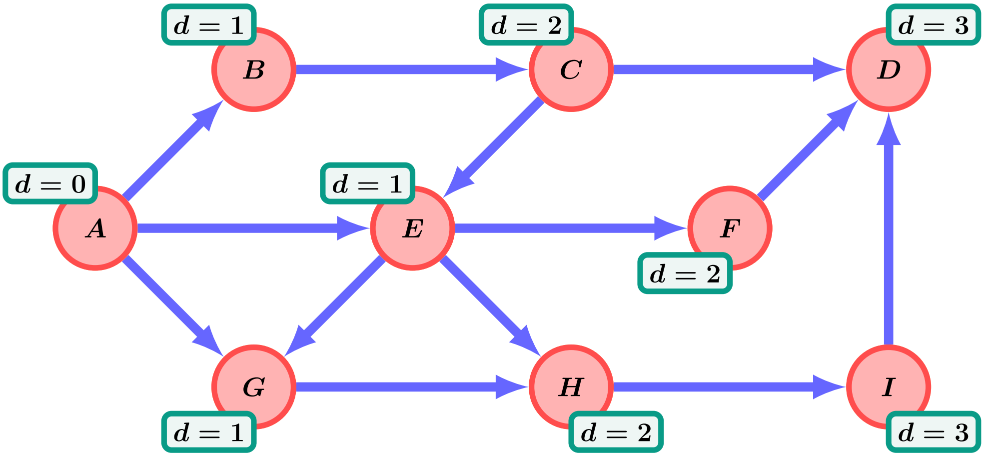


# Breadth-First-Search



# Breadth-First-Search




## Queue Empty so Terminate!

## Breadth-First-Search

**Theorem** The worst-case termination time for Breadth-First-Search is $O(|V| + |E|)$.

**Proof:**

- Each vertex is enqueued at most once and so the number of times that **Step 3** and **Step 5** are repeated is at most $O(|V|)$-times.
- Furthermore, for each vertex $u$ we repeat **Step 4** at most $|\mathbf{Adj}(u)|$-times **(where Adj$(u) \subseteq E$ is the set of adjacent vertices)** In total, we repeat **Step 4** at most $O(|E|)$-times, since

$$\boxed{|\mathbf{Adj}(u_1)| + |\mathbf{Adj}(u_2)| + \cdots + |\mathbf{Adj}(u_n)| = |E|}$$

**Q.E.D.**

# Depth-First-Search

**Depth-First-Search (DFS)**

**Step 1)** Select a root node $r \in V$ and set $d(r) = 0$.

**Step 2)** Add $v$ to a stack.

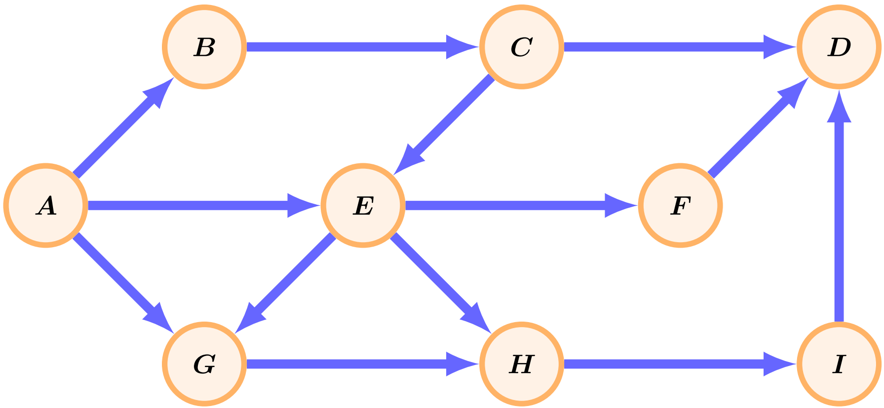
**Step 3)** Pop the first element $u$ from the stack.

**Step 4)** For each vertex $v$ adjacent to $u$, if $d(v)$ is not yet defined, then set $d(v) = d(u) + 1$ and push $v$ to the stack.

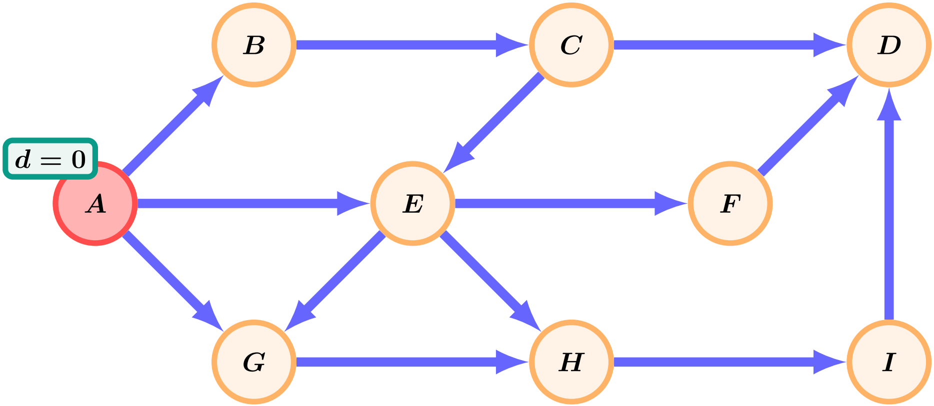
**Step 5)** Repeat from Step 3 until the stack is empty.

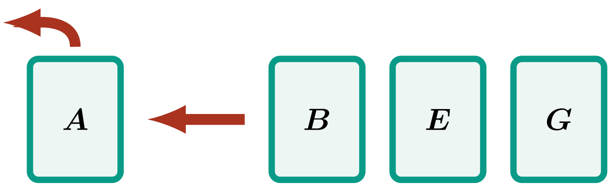
**(the implementation appears differently in Cormen et.al, but the principle is the same)**

# Depth-First-Search

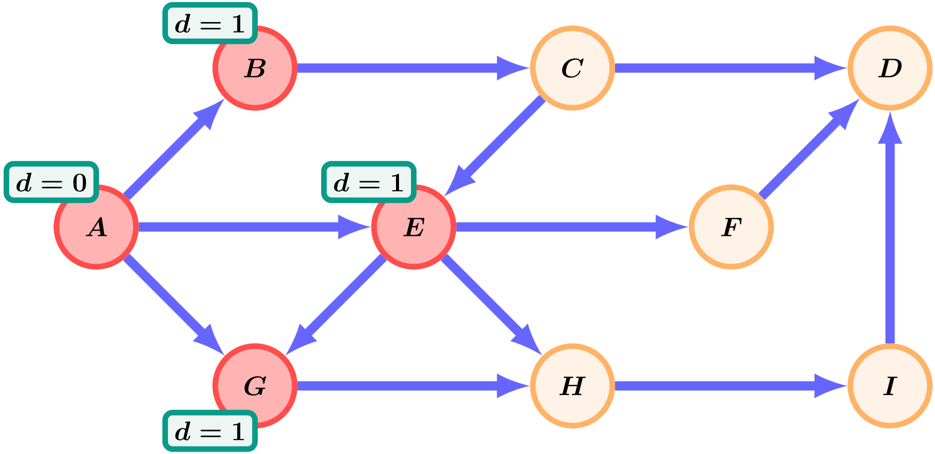


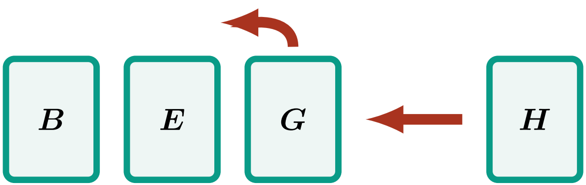
# Depth-First-Search



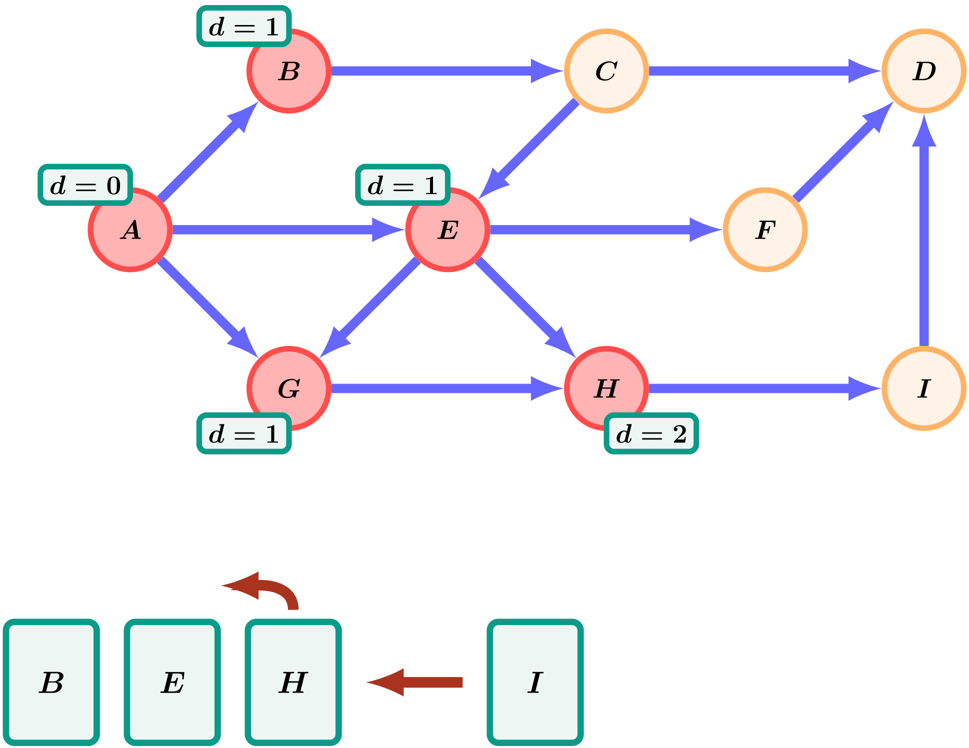



# Depth-First-Search

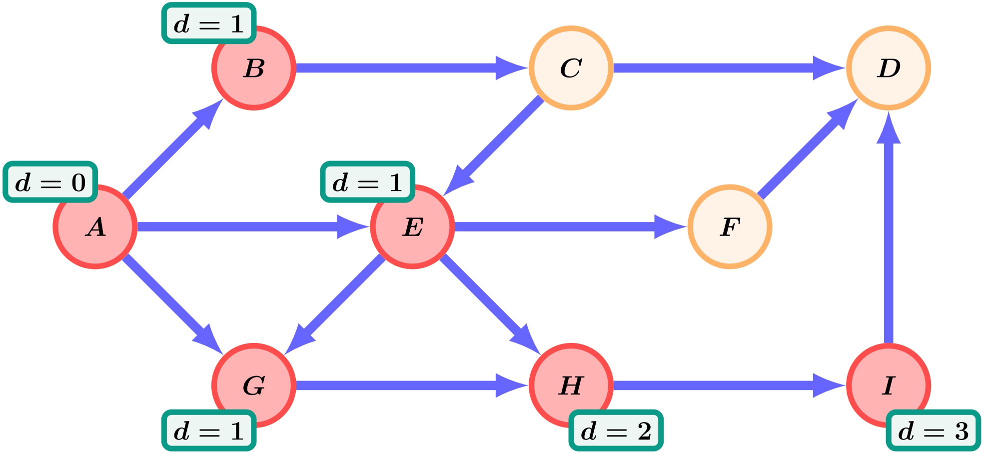


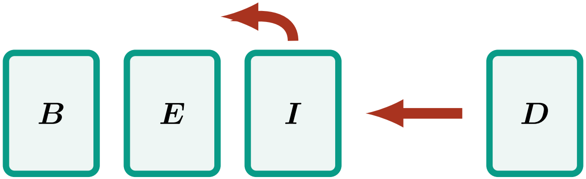



## Depth-First-Search

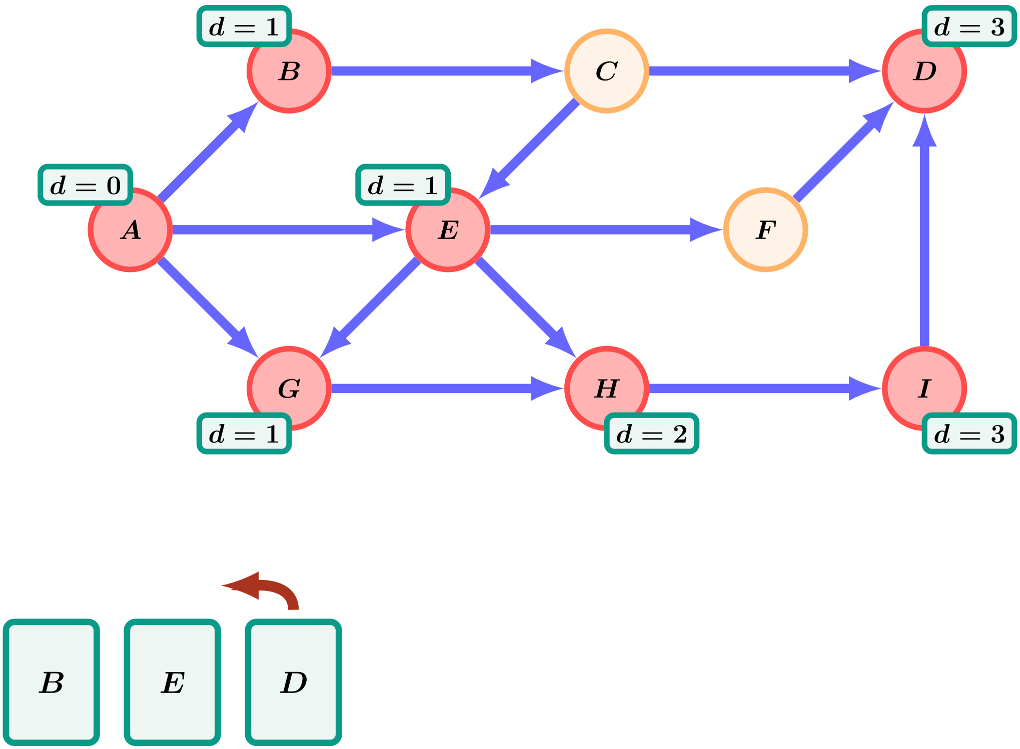


# Depth-First-Search

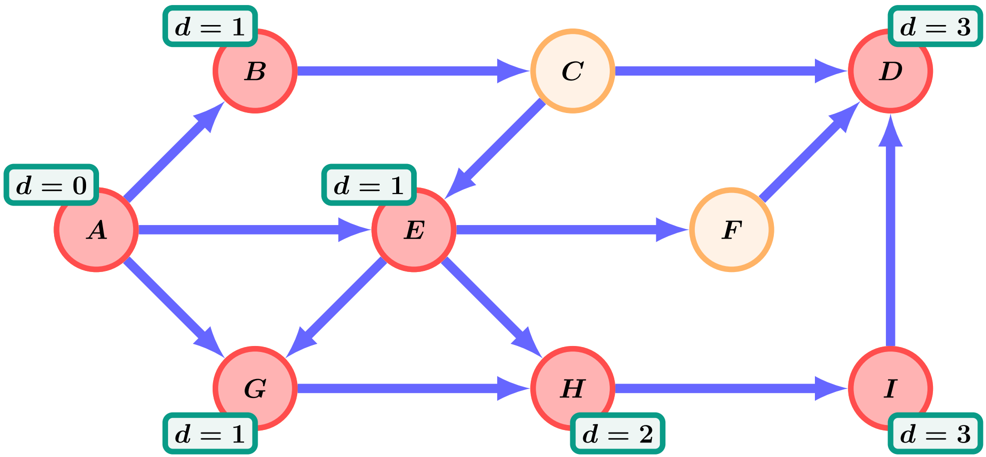


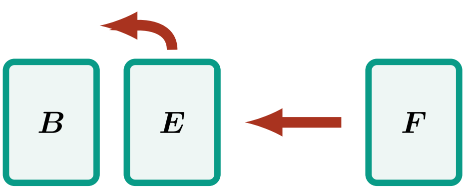



## Depth-First-Search

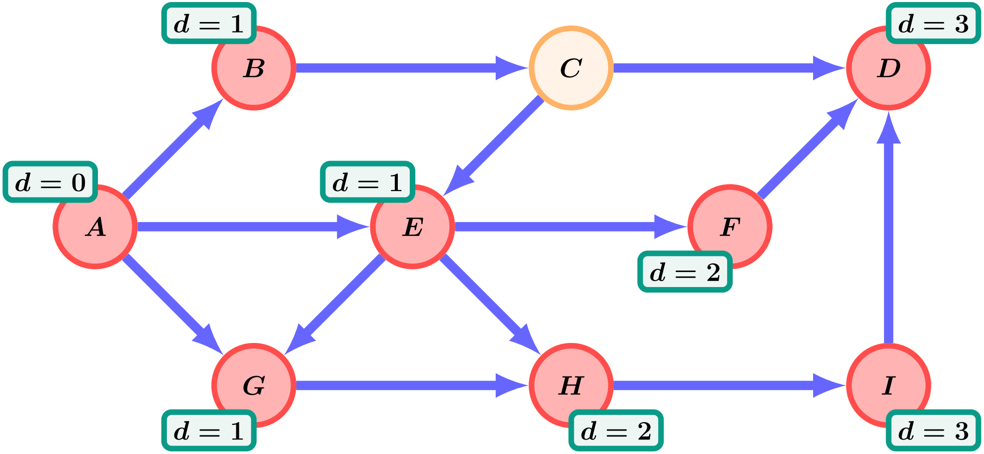


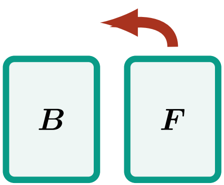
## Depth-First-Search



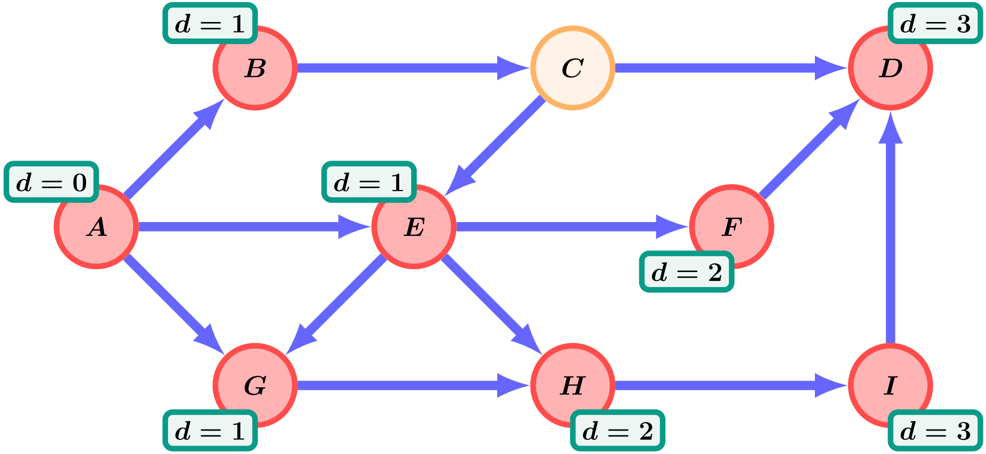



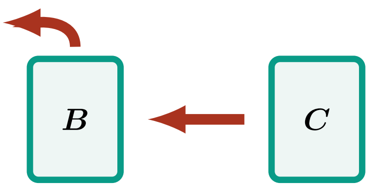
## Depth-First-Search



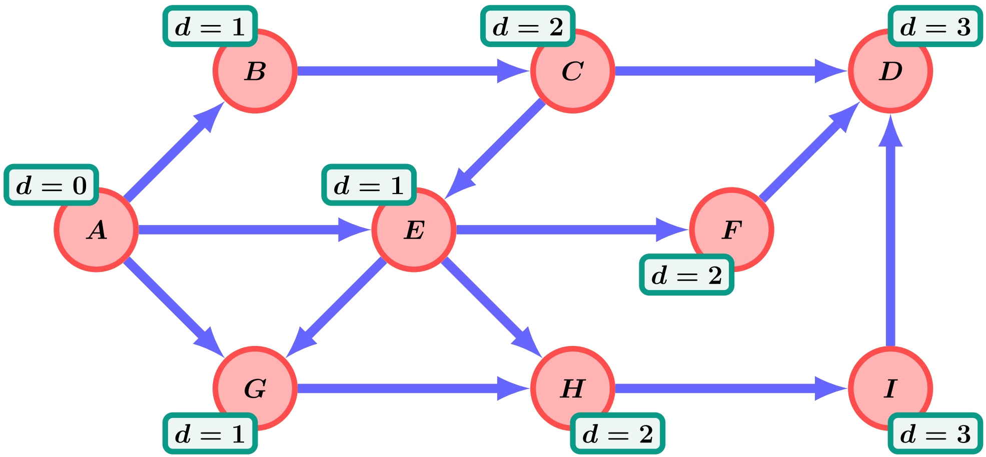



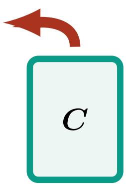
# Depth-First-Search



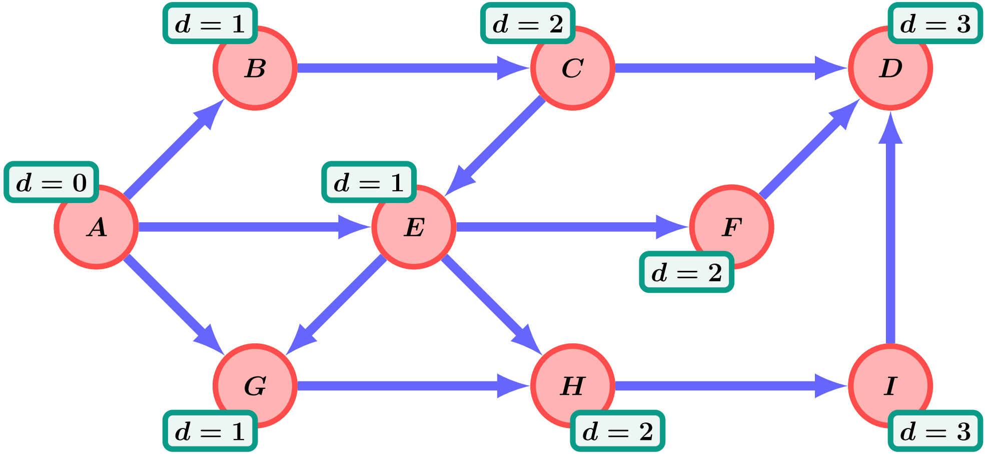



# Depth-First-Search

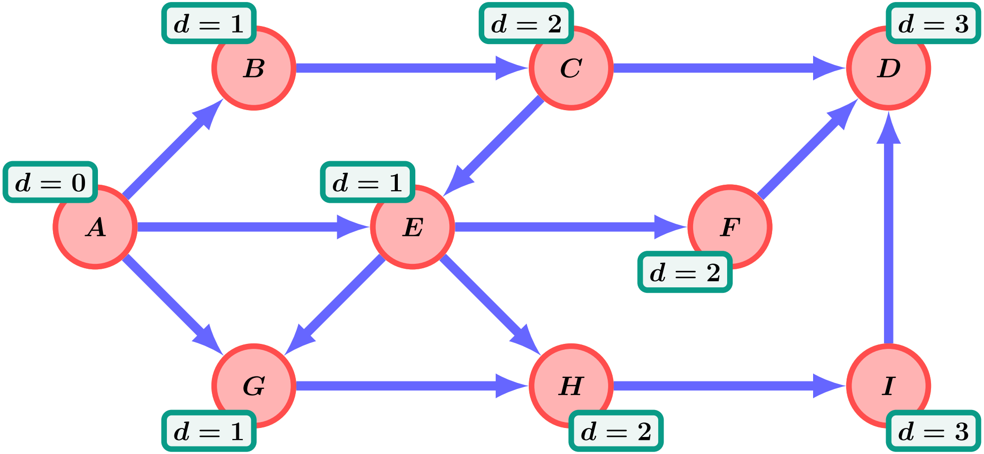


# Depth-First-Search



## Depth-First-Search




# Stack Empty so Terminate!

## Depth-First-Search

**Theorem** The worst-case termination time for Depth-First-Search is $O(|V| + |E|)$.

**Proof:**

- The proof is almost identical to that of the **Breadth-First-Search**, replacing the **queue** for a **stack**.

**Q.E.D.**

# End of Slides!

🎉

## Feedback

- **Let me know how you found today's lecture!**

https://goo.gl/forms/Q7NWfrgQiKOfKSJd2