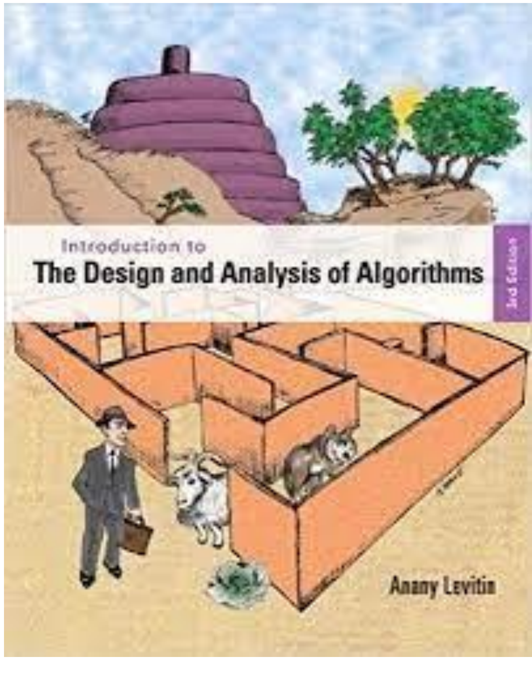
3-Brute Force and Exhaustive Search

A. Levitin "Introduction to the Design & Analysis of Algorithms," 3rd ed., Ch. 1 ©2012 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved

BLM202 Veri Yapıları Lecture Notes



Brute Force

A straightforward approach, usually based directly on the problem’s statement and definitions of the concepts involved

Examples:

1. Computing a^n ($a > 0$, n a nonnegative integer)

2. Computing $n!$

3. Multiplying two matrices

4. Searching for a key of a given value in a list

Brute-Force Sorting Algorithm

Selection Sort   Scan the array to find its smallest element and swap it with the first element. Then, starting with the second element, scan the elements to the right of it to find the smallest among them and swap it with the second elements. Generally, on pass i ($0 \leq i \leq n$-2), find the smallest element in $A[i..n\text{-}1]$ and swap it with $A[i]$:

$$A[0] \leq \ . \ . \ . \ \leq A[i\text{-}1] \ | \ A[i], \ . \ . \ . \ . \ , A[min], . \ . \ ., A[n\text{-}1]$$

in their final positions

Example: 7   3   2   5

Selection Sort

Select 1 (the smallest) and swap it with 2 (the first) in the list.	swap 2 9 5 4 8 1 6	
The number 1 is now in the correct position and thus no longer needs to be considered.	swap 1 9 5 4 8 2 6	Select 2 (the smallest) and swap it with 9 (the first) in the remaining list.
The number 2 is now in the correct position and thus no longer needs to be considered.	swap 1 2 5 4 8 9 6	Select 4 (the smallest) and swap it with 5 (the first) in the remaining list.
The number 4 is now in the correct position and thus no longer needs to be considered.	1 2 4 5 8 9 6	5 is the smallest and in the right position. No swap is necessary.
The number 5 is now in the correct position and thus no longer needs to be considered.	swap 1 2 4 5 8 9 6	Select 6 (the smallest) and swap it with 8 (the first) in the remaining list.
The number 6 is now in the correct position and thus no longer needs to be considered.	swap 1 2 4 5 6 9 8	Select 8 (the smallest) and swap it with 9 (the first) in the remaining list.
The number 8 is now in the correct position and thus no longer needs to be considered.	1 2 4 5 6 8 9	Since there is only one element remaining in the list, the sort is completed.

Analysis of Selection Sort

ALGORITHM $\mathit{SelectionSort}(A[0..n-1])$

//Sorts a given array by selection sort
//Input: An array $A[0..n-1]$ of orderable elements
//Output: Array $A[0..n-1]$ sorted in nondecreasing order
for $i \leftarrow 0$ **to** $n-2$ **do**
    $min \leftarrow i$
    for $j \leftarrow i+1$ **to** $n-1$ **do**
        if $A[j] < A[min]$ $min \leftarrow j$
    swap $A[i]$ and $A[min]$

Time efficiency:

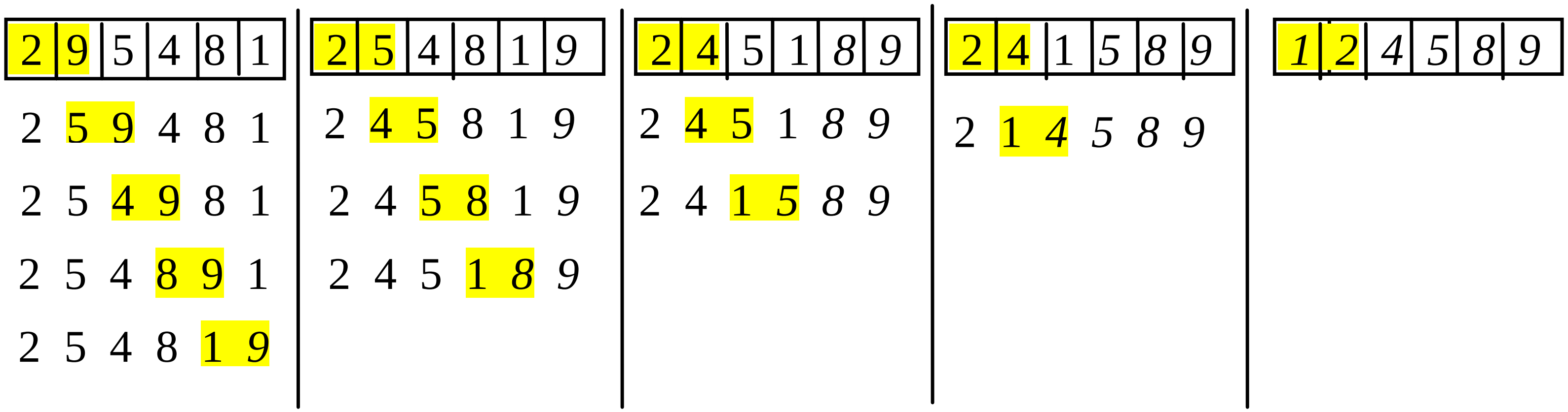
Space efficiency:

Stability:

Bubble Sort

- Pass through the array of elements
- Exchange adjacent elements, if necessary
- When no exchanges are required, then array is sorted.
- Make as many passes as the number of elements of the array

Bubble Sort



(a) 1st pass (b) 2nd pass (c) 3rd pass (d) 4th pass (e) 5th pass

Bubble Sort

ALGORITHM $BubbleSort(A[0..n-1])$

//Sorts a given array by bubble sort
//Input: An array $A[0..n-1]$ of orderable elements
//Output: Array $A[0..n-1]$ sorted in nondecreasing order
for $i \leftarrow 0$ **to** $n-2$ **do**
  for $j \leftarrow 0$ **to** $n-2-i$ **do**
    if $A[j+1] < A[j]$ swap $A[j]$ and $A[j+1]$

Time efficiency:

Brute-Force String Matching

- *pattern*: a string of *m* characters to search for
- *text*: a (longer) string of *n* characters to search in
- problem: find a substring in the text that matches the pattern

$$
\begin{array}{ccccccccccc}
t_0 & \ldots & t_i & \ldots & t_{i+j} & \ldots & t_{i+m-1} & & \ldots & t_{n-1} & \text{text } T \\
 & & \updownarrow & & \updownarrow & & \updownarrow & & & & \\
 & & p_0 & \ldots & p_j & \ldots & p_{m-1} & & \text{pattern } P & &
\end{array}
$$

Brute-Force String Matching

<u>Brute-force algorithm</u>

Step 1 Align pattern at beginning of text

Step 2 Moving from left to right, compare each character of pattern to the corresponding character in text until

- all characters are found to match (successful search); or
- a mismatch is detected

Step 3 While pattern is not found and the text is not yet exhausted, realign pattern one position to the right and repeat Step 2

Brute-Force String Matching

Brute-force algorithm

Step 1 Align pattern at beginning of text

Step 2 Moving from left to right, compare each character of pattern to the corresponding character in text until

- all characters are found to match (successful search); or
- a mismatch is detected

Step 3 While pattern is not found and the text is not yet exhausted, realign pattern one position to the right and repeat Step 2

Examples of Brute-Force String Matching

Pattern: 001011

Text: 10010101101001100101111010

Pattern: happy

Text: It is never too late to have a happy childhood

Pseudocode and Efficiency

ALGORITHM $BruteForceStringMatch(T[0..n-1], P[0..m-1])$

//Implements brute-force string matching
//Input: An array $T[0..n-1]$ of n characters representing a text and
//   an array $P[0..m-1]$ of m characters representing a pattern
//Output: The index of the first character in the text that starts a
//   matching substring or -1 if the search is unsuccessful
for $i \leftarrow 0$ **to** $n - m$ **do**
    $j \leftarrow 0$
    while $j < m$ **and** $P[j] = T[i + j]$ **do**
        $j \leftarrow j + 1$
    if $j = m$ **return** i
return -1

Time efficiency:

Brute-Force Polynomial Evaluation

Problem: Find the value of polynomial

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x^1 + a_0$$

at a point $x = x_0$

Brute-force algorithm

```
p ← 0.0
for i ← n downto 0 do
    power ← 1
        for  j ← 1 to i do  //compute x^i
            power ← power * x
        p ← p + a[i] * power
return p
```

Efficiency:

Polynomial Evaluation: Improvement

- We can do better by evaluating from right to left:

<u>Better brute-force algorithm</u>

$p \leftarrow a[0]$
$power \leftarrow 1$
for $i \leftarrow 1$ **to** n **do**
    $power \leftarrow power * x$
    $p \leftarrow p + a[i] * power$
return p

Efficiency:

Closest- Pair Problem

Find the two closest points in a set of *n* points (in the two-dimensional Cartesian plane).

<u>Brute-force algorithm</u>

Compute the distance between every pair of distinct points and return the indexes of the points for which the distance is the smallest.

Closest- Pair Problem

ALGORITHM *BruteForceClosestPoints*(P)

```
//Input: A list P of n (n ≥ 2) points P_1 = (x_1, y_1), ..., P_n = (x_n, y_n)
//Output: Indices index1 and index2 of the closest pair of points
dmin ← ∞
for i ← 1 to n − 1 do
    for j ← i + 1 to n do
        d ← sqrt((x_i − x_j)^2 + (y_i − y_j)^2) //sqrt is the square root function
        if d < dmin
            dmin ← d; index1 ← i; index2 ← j
return index1, index2
```

Efficiency:

How to make it faster?

Brute-Force Strengths and Weaknesses

- <u>Strengths</u>
  - wide applicability
  - simplicity
  - yields reasonable algorithms for some important problems (e.g., matrix multiplication, sorting, searching, string matching)

- <u>Weaknesses</u>
  - rarely yields efficient algorithms
  - some brute-force algorithms are unacceptably slow
  - not as constructive as some other design techniques

Exhaustive Search

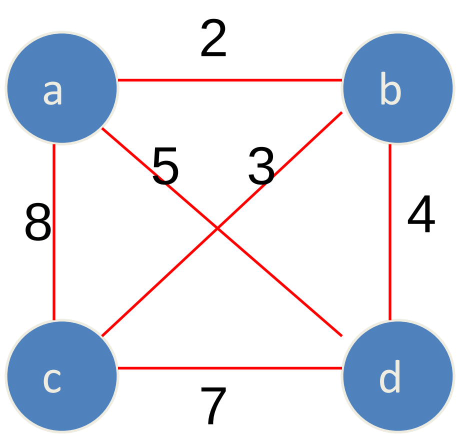
A brute force solution to a problem involving search for an element with a special property, usually among combinatorial objects such as permutations, combinations, or subsets of a set.

Method:

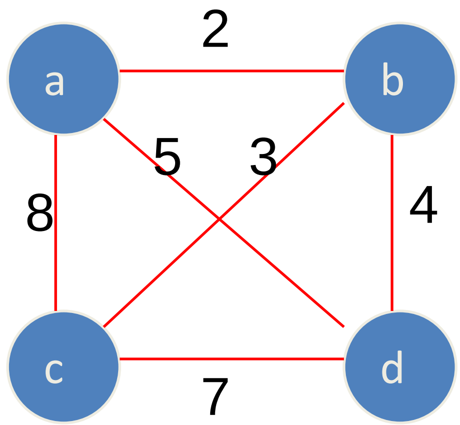
- generate a list of all potential solutions to the problem in a systematic manner
- evaluate potential solutions one by one, disqualifying infeasible ones and, for an optimization problem, keeping track of the best one found so far
- when search ends, announce the solution(s) found

Example 1: Traveling Salesman Problem (TSP)

- Given n cities with known distances between each pair, find the shortest tour that passes through all the cities exactly once before returning to the starting city
- Alternatively: Find shortest Hamiltonian circuit in a weighted connected graph
- Example:



TSP by Exhaustive Search



Tour	Cost
a→b→c→d→a	2+3+7+5 = 17
a→b→d→c→a	2+4+7+8 = 21
a→c→b→d→a	8+3+4+5 = 20
a→c→d→b→a	8+7+4+2 = 21
a→d→b→c→a	5+4+3+8 = 20
a→d→c→b→a	5+7+3+2 = 17

More tours?

Less tours?

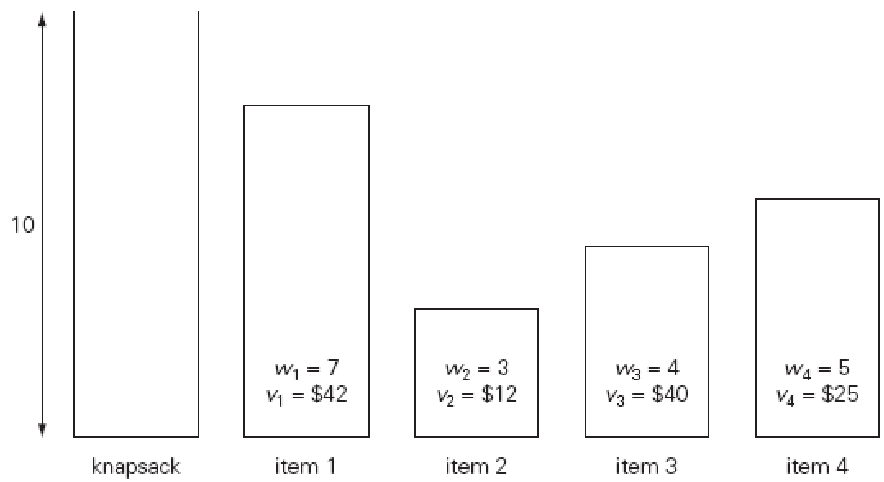
Efficiency:

Example 2: Knapsack Problem

Given n items:

- weights: $w_1 \quad w_2 \quad \dots \quad w_n$
- values: $v_1 \quad v_2 \quad \dots \quad v_n$
- a knapsack of capacity W

Find most valuable subset of the items that fit into the knapsack



(a)

0-1 Knapsack problem

The problem is called a “0-1” problem, because each item must be entirely accepted or rejected.

Problem, in other words, is to find

$$max \sum_{i \in T} v_i$$

subject to

$$\sum_{i \in T} w_i \leq W$$

Example 2: Knapsack Problem

Example: Knapsack capacity $W = 16$

item	weight	value
1	2	\$20
2	5	\$30
3	10	\$50
4	5	\$10

We go through all combinations (subsets) and find the one with maximum value and with total weight less or equal to W

Knapsack Problem by Exhaustive Search

Subset	Total weight	Total value
{1}	2	$20
{2}	5	$30
{3}	10	$50
{4}	5	$10
{1,2}	7	$50
{1,3}	12	$70
{1,4}	7	$30
{2,3}	15	$80
{2,4}	10	$40
{3,4}	15	$60
{1,2,3}	17	not feasible
{1,2,4}	12	$60
{1,3,4}	17	not feasible
{2,3,4}	20	not feasible
{1,2,3,4}	22	not feasible

Efficiency:

Example 3: The Assignment Problem

There are n people who need to be assigned to n jobs, one person per job. The cost of assigning person i to job j is $C[i, j]$. Find an assignment that minimizes the total cost.

	Job 0	Job 1	Job 2	Job 3
Person 0	9	2	7	8
Person 1	6	4	3	7
Person 2	5	8	1	8
Person 3	7	6	9	4

Algorithmic Plan: Generate all legitimate assignments, compute their costs, and select the cheapest one.

How many assignments are there?

Assignment Problem by Exhaustive Search

Pose the problem as the one about a cost matrix:

$$C = \begin{matrix} 9 & 2 & 7 & 8 \\ 6 & 4 & 3 & 7 \\ 5 & 8 & 1 & 8 \\ 7 & 6 & 9 & 4 \end{matrix}$$

<u>Assignment</u> (col.#s)	<u>Total Cost</u>
1, 2, 3, 4	9+4+1+4=18
1, 2, 4, 3	9+4+8+9=30
1, 3, 2, 4	9+3+8+4=24
1, 3, 4, 2	9+3+8+6=26
1, 4, 2, 3	9+7+8+9=33
1, 4, 3, 2	9+7+1+6=23
	etc.

(For this particular instance, the optimal assignment can be found by exploiting the specific features of the number given. It is:)

Final Comments on Exhaustive Search

- Exhaustive-search algorithms run in a realistic amount of time only on very small instances

- In some cases, there are much better alternatives!
  - Euler circuits
  - shortest paths
  - minimum spanning tree
  - assignment problem

- In many cases, exhaustive search or its variation is the only known way to get exact solution

Graph Traversal Algorithms

Many problems require processing all graph vertices (and edges) in systematic fashion

Graph traversal algorithms:

- Depth-first search (DFS)
- Breadth-first search (BFS)

Depth-first search (DFS)

- Visits graph's vertices by always moving away from last visited vertex to unvisited one, backtracks if no adjacent unvisited vertex is available.
- Uses a stack
  - a vertex is pushed onto the stack when it's reached for the first time
  - a vertex is popped off the stack when it becomes a dead end, i.e., when there is no adjacent unvisited vertex
- "Redraws" graph in tree-like fashion (with tree edges and back edges for undirected graph)
- https://www.cs.usfca.edu/~galles/visualization/DFS.html

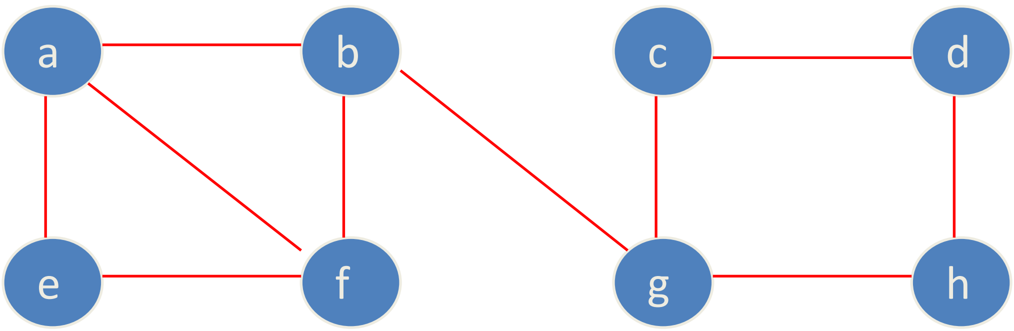
Pseudocode of DFS

ALGORITHM $DFS(G)$

```
//Implements a depth-first search traversal of a given graph
//Input: Graph G = ⟨V, E⟩
//Output: Graph G with its vertices marked with consecutive integers
//          in the order they are first encountered by the DFS traversal
mark each vertex in V with 0 as a mark of being "unvisited"
count ← 0
for each vertex v in V do
    if v is marked with 0
        dfs(v)

dfs(v)
//visits recursively all the unvisited vertices connected to vertex v
//by a path and numbers them in the order they are encountered
//via global variable count
count ← count + 1;   mark v with count
for each vertex w in V adjacent to v do
    if w is marked with 0
        dfs(w)
```

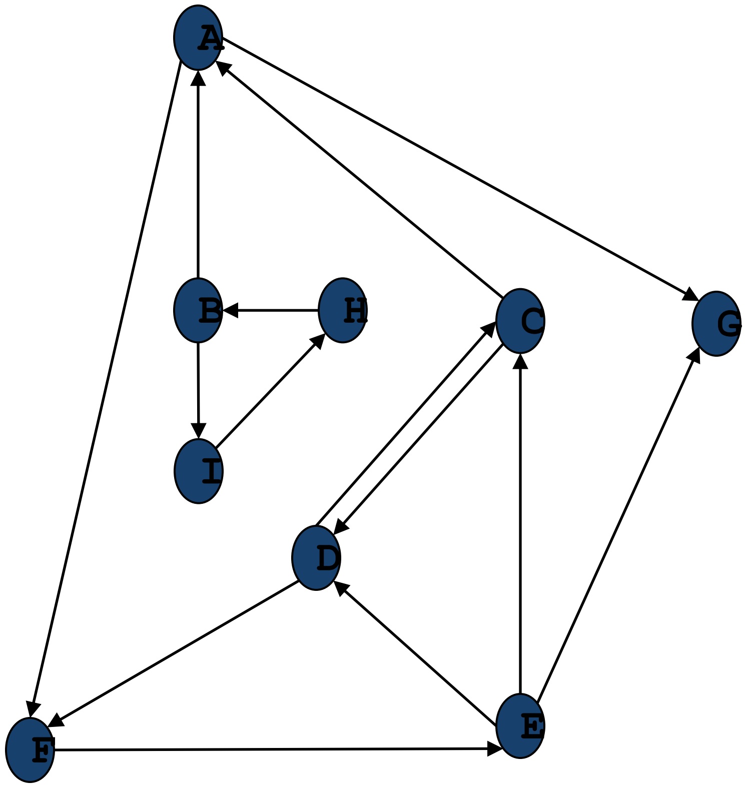
Example: DFS traversal of undirected graph



DFS traversal stack:

DFS tree:

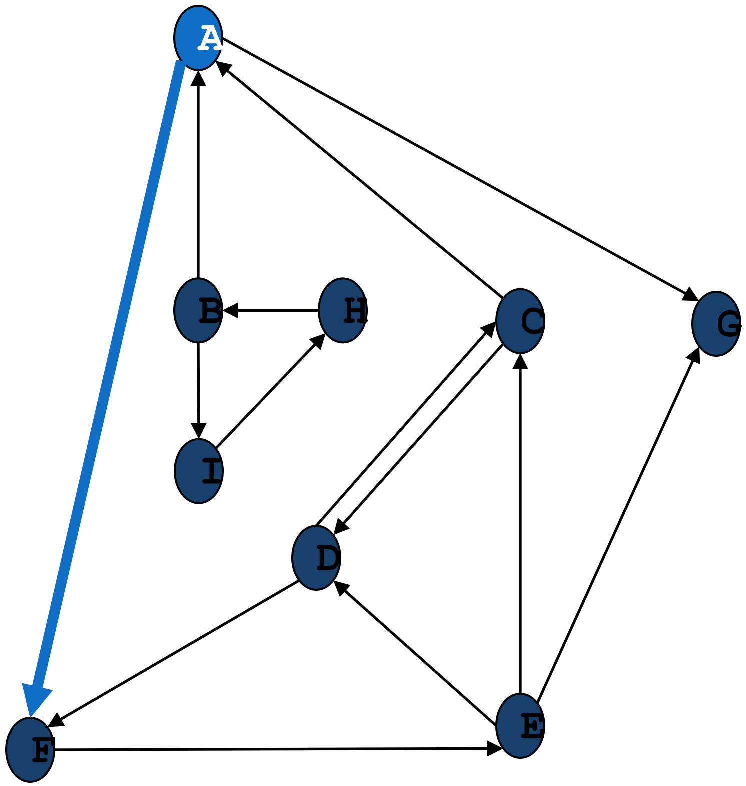
Depth-First Search

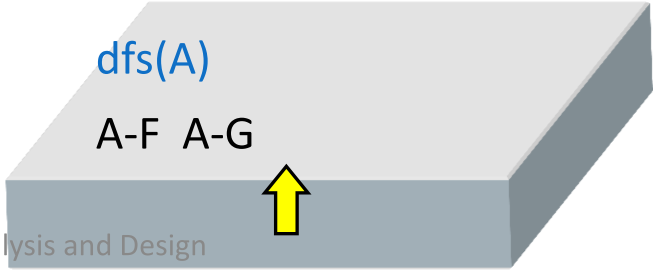


Adjacency Lists

A: F G
B: A H
C: A D
D: C F
E: C D G
F: E
G:
H: B
I: H

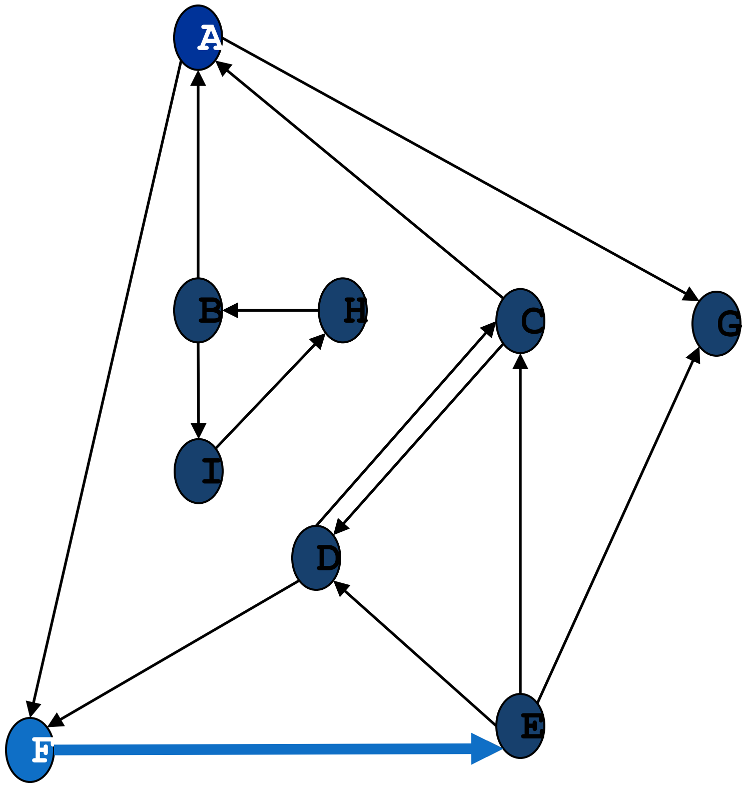
Depth-First Search

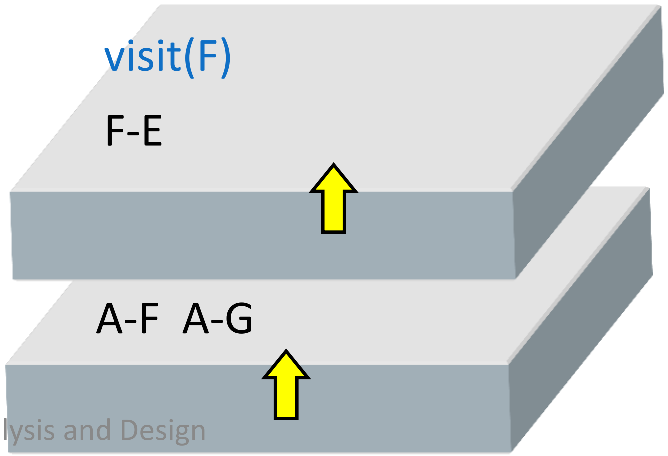




Function call stack:

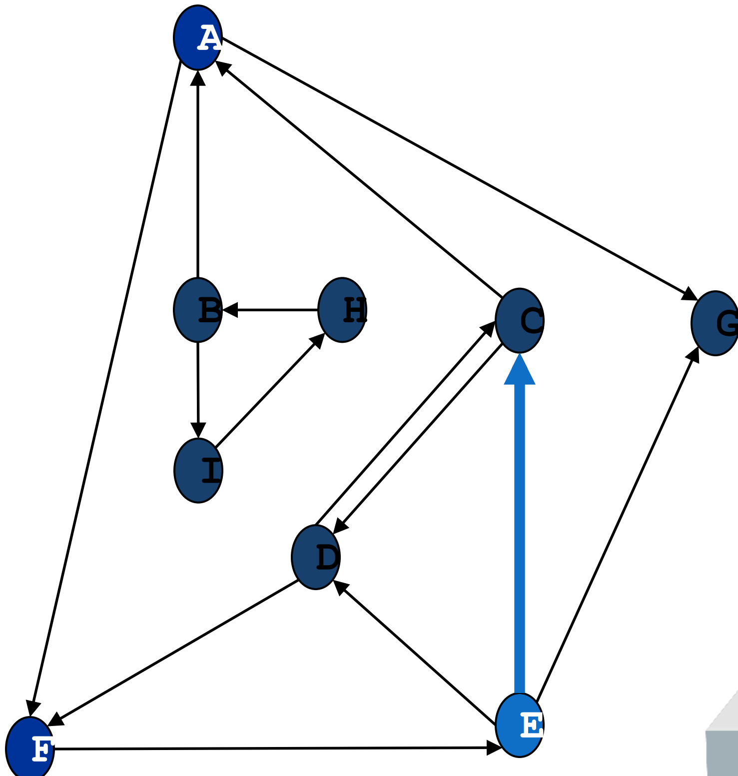
Depth-First Search

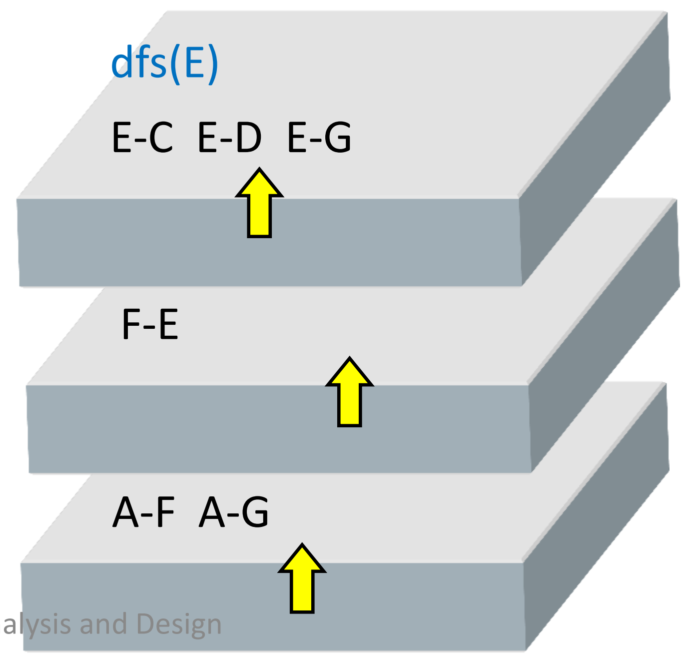




Function call stack:

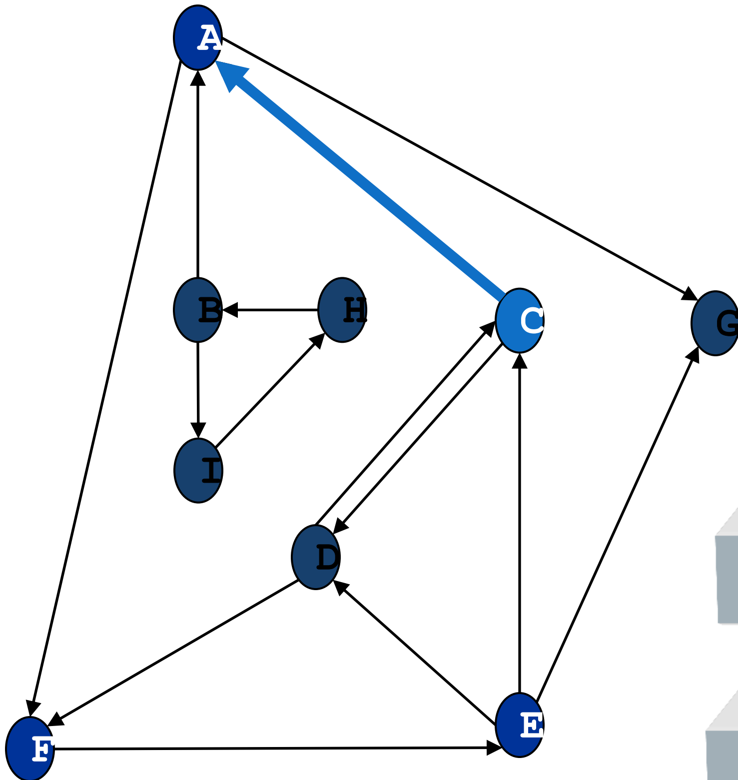
Depth-First Search

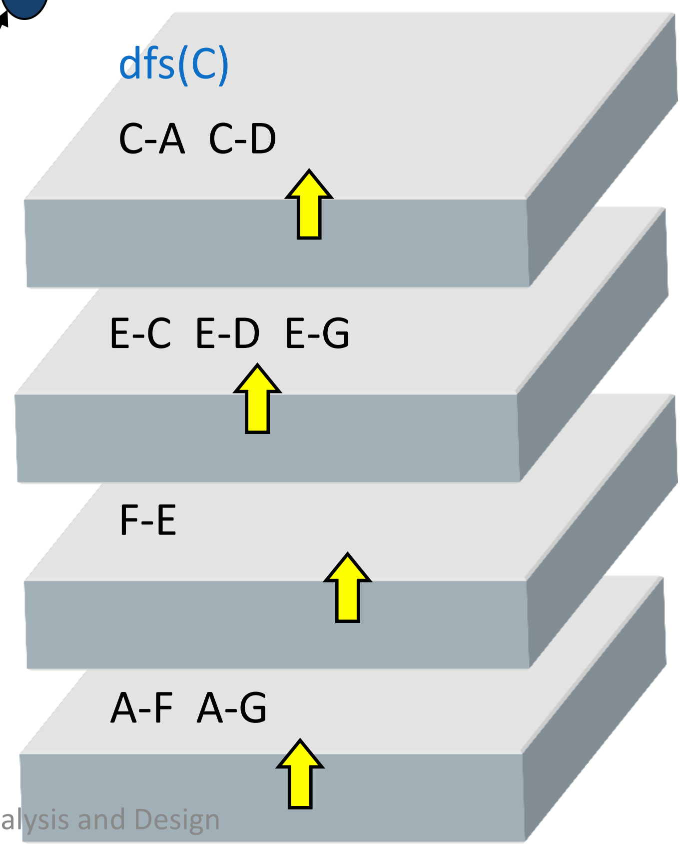




Function call stack:

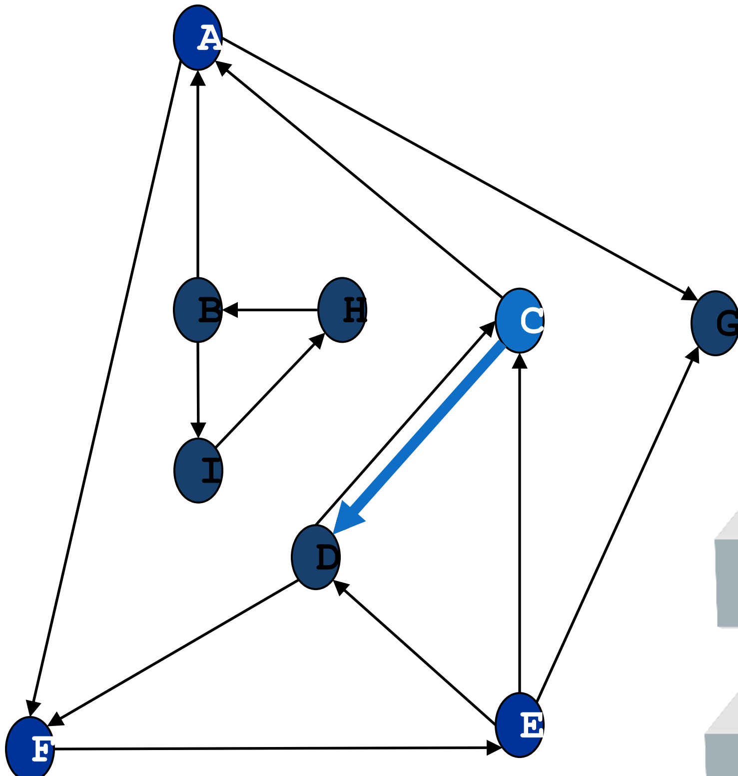
Depth-First Search

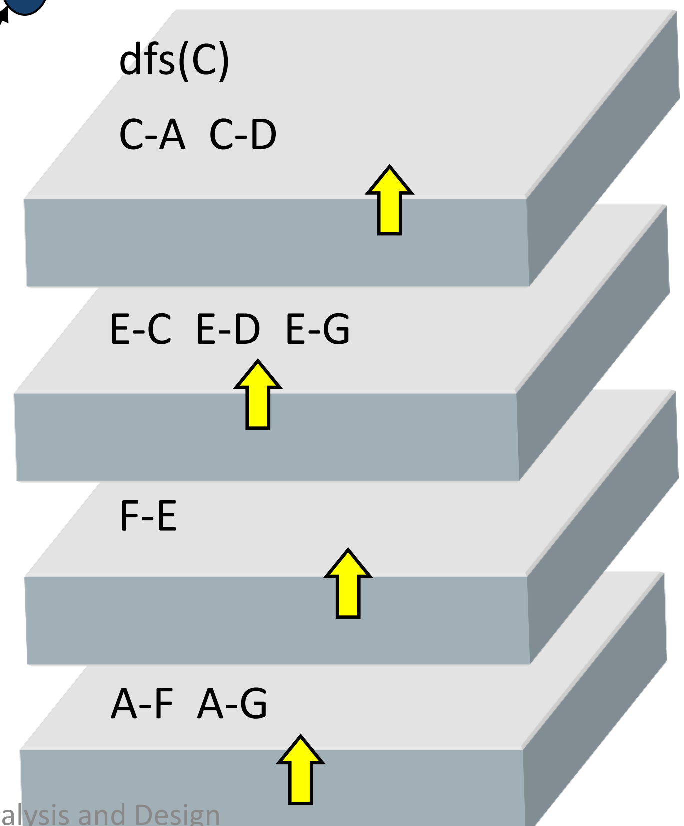




Function call stack:

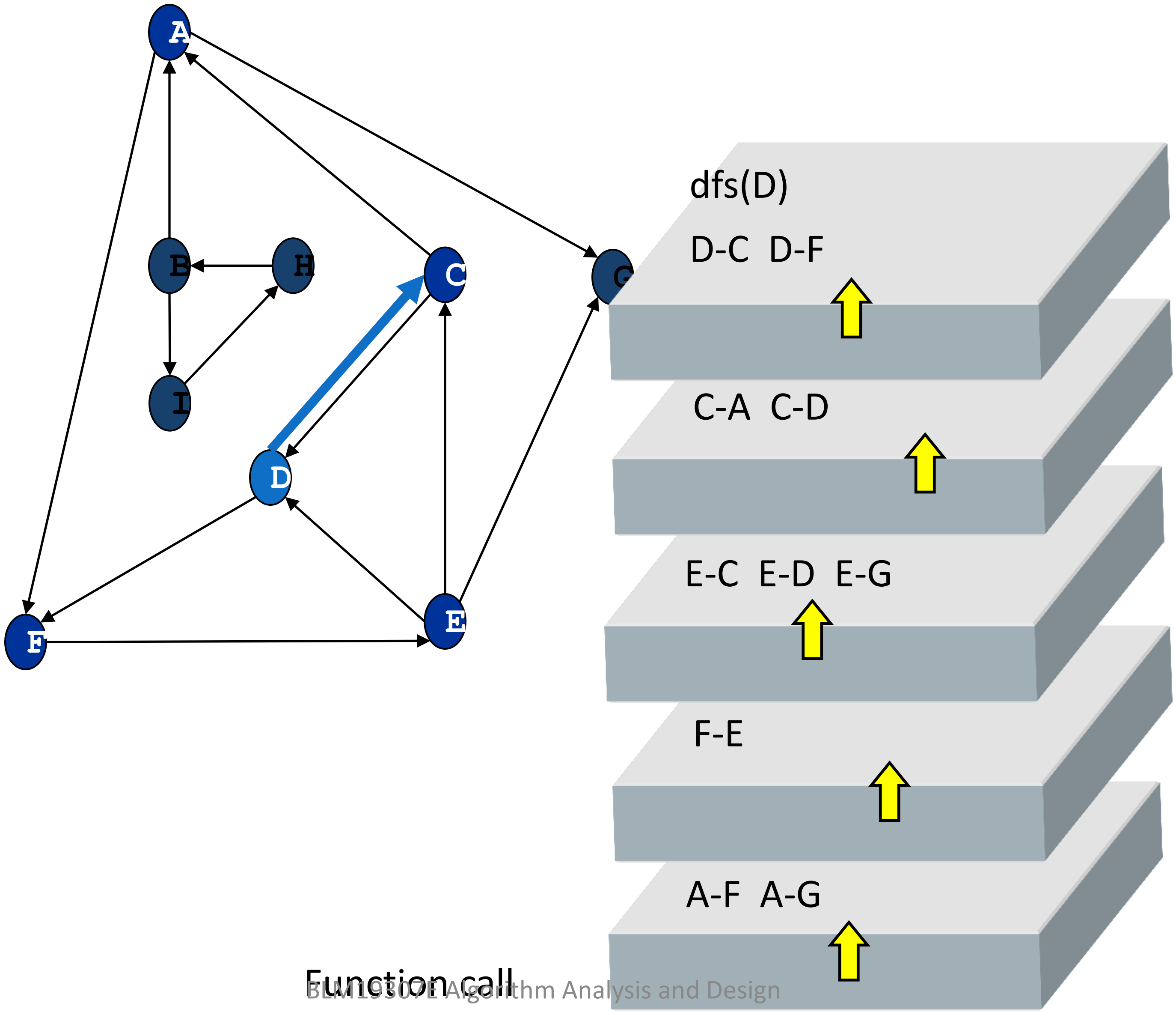
Depth-First Search





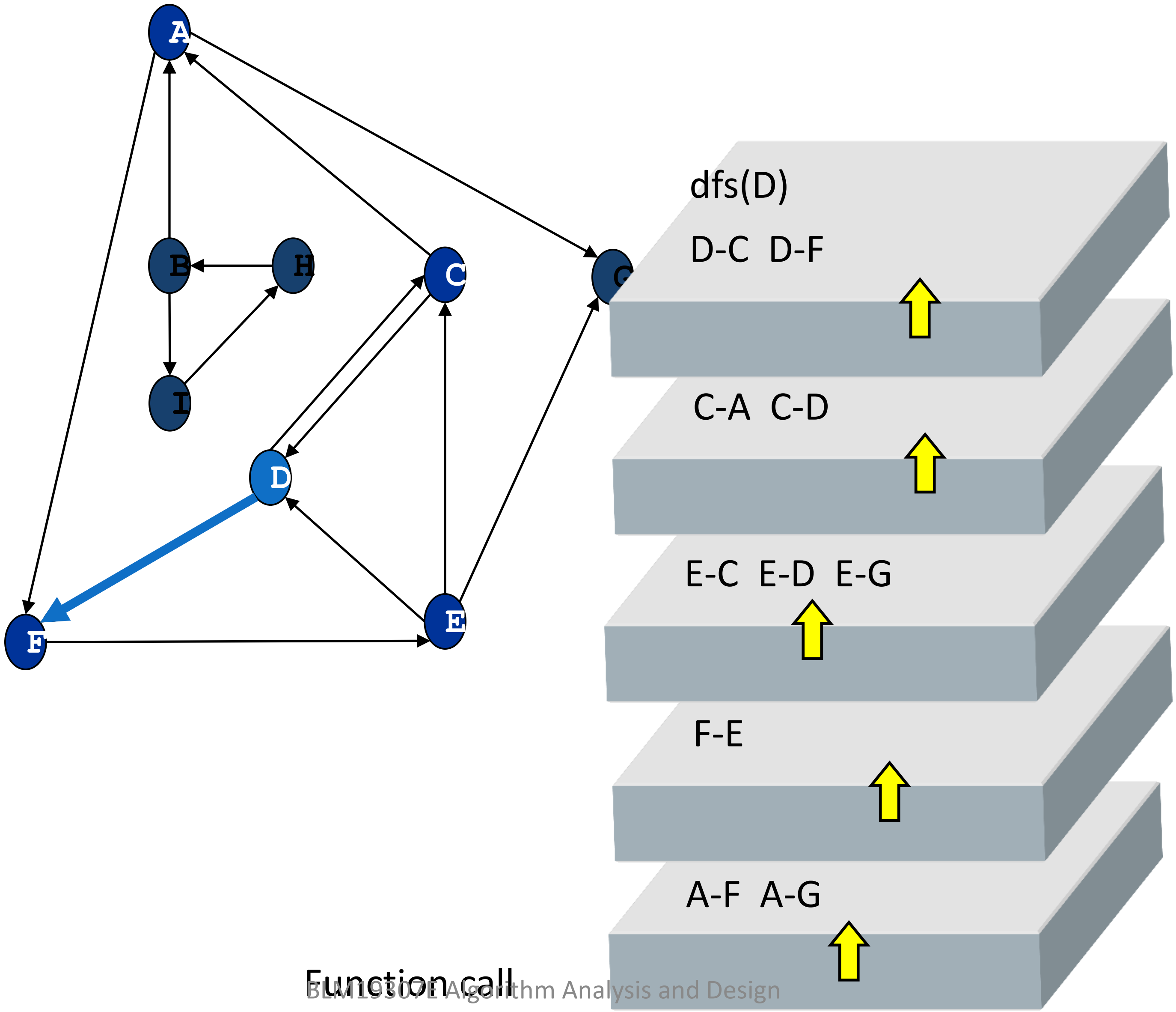
Function call stack:

Depth-First Search



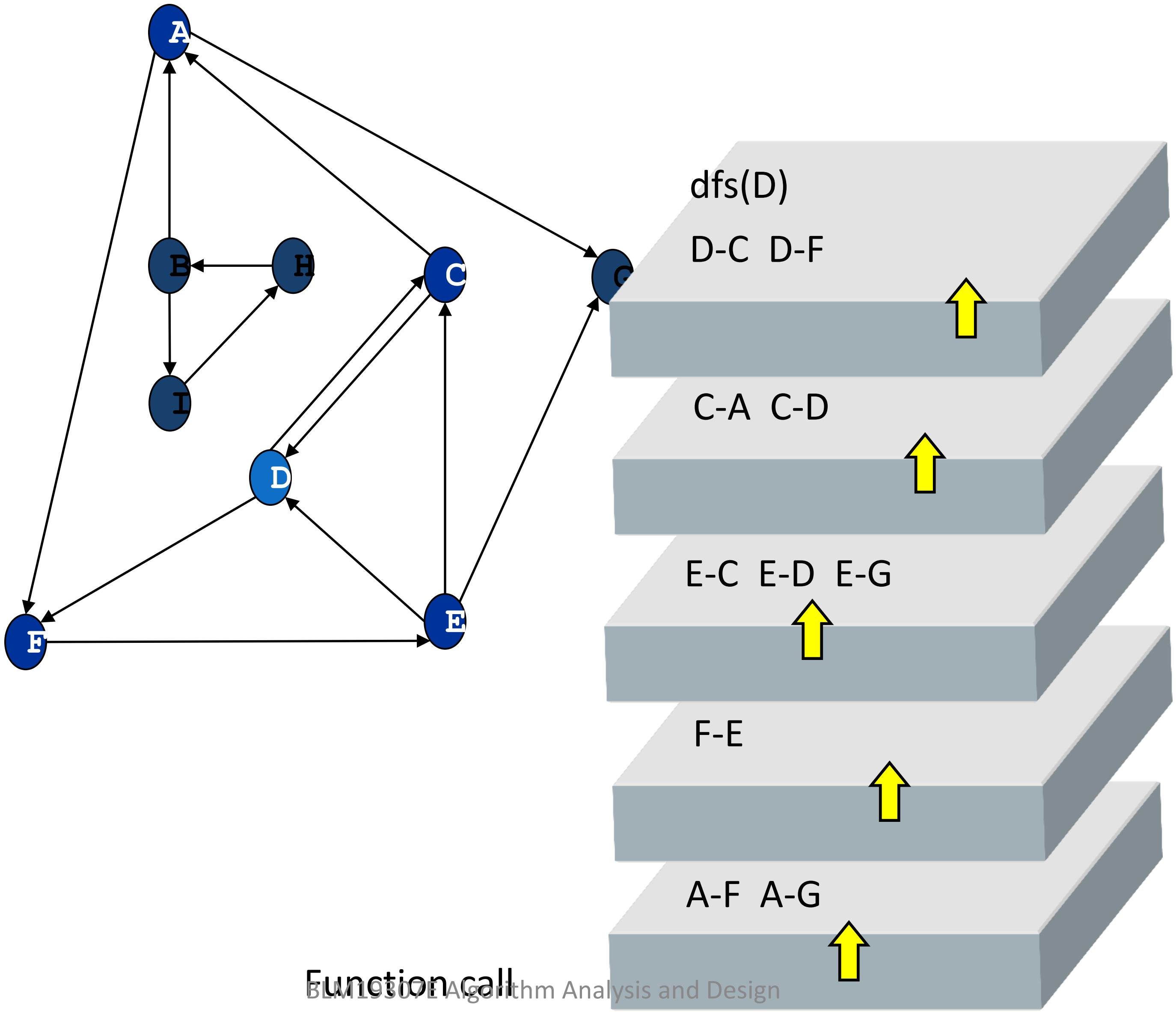
Function call stack:

Depth-First Search



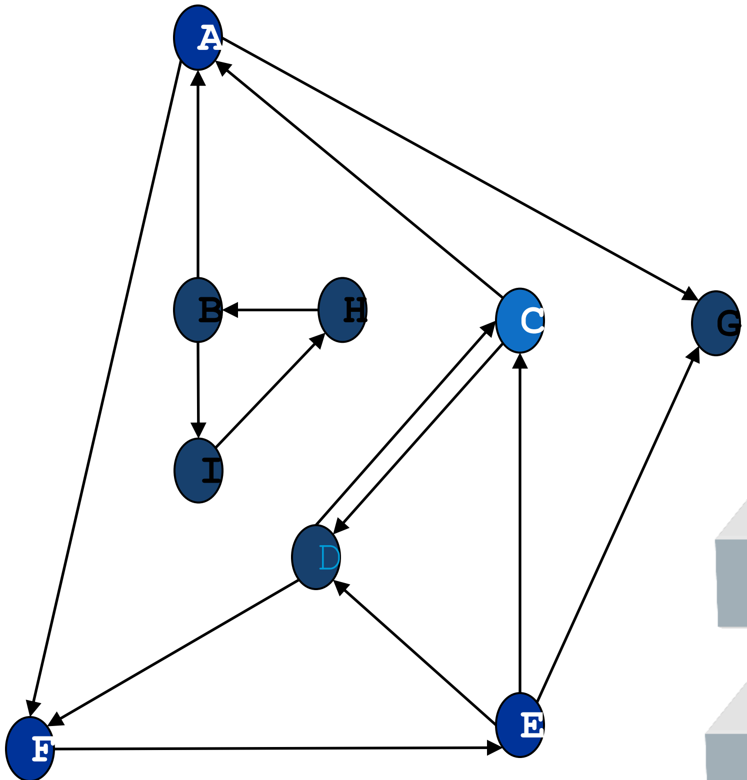
Function call stack:

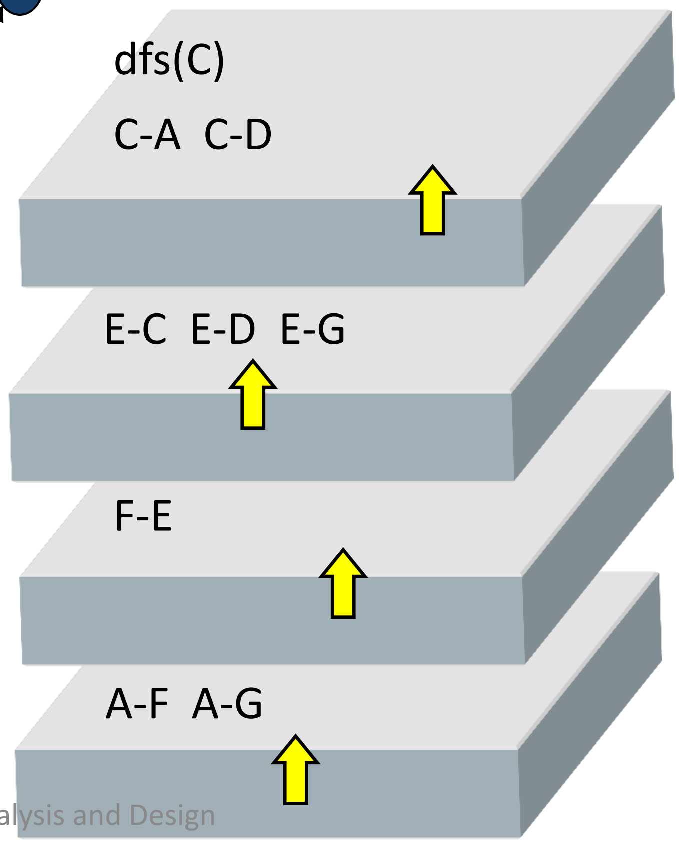
Depth-First Search



Function call stack:

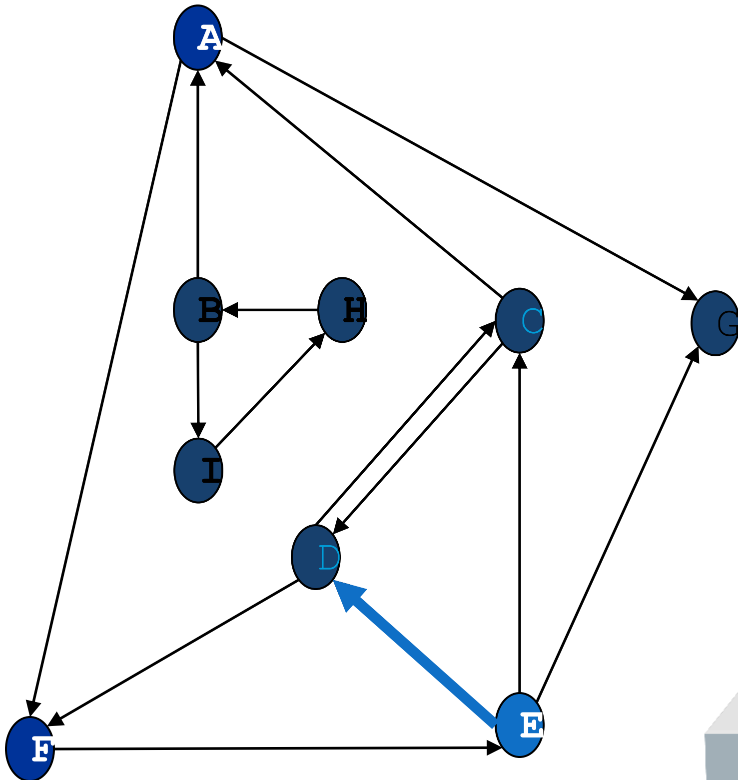
Depth-First Search

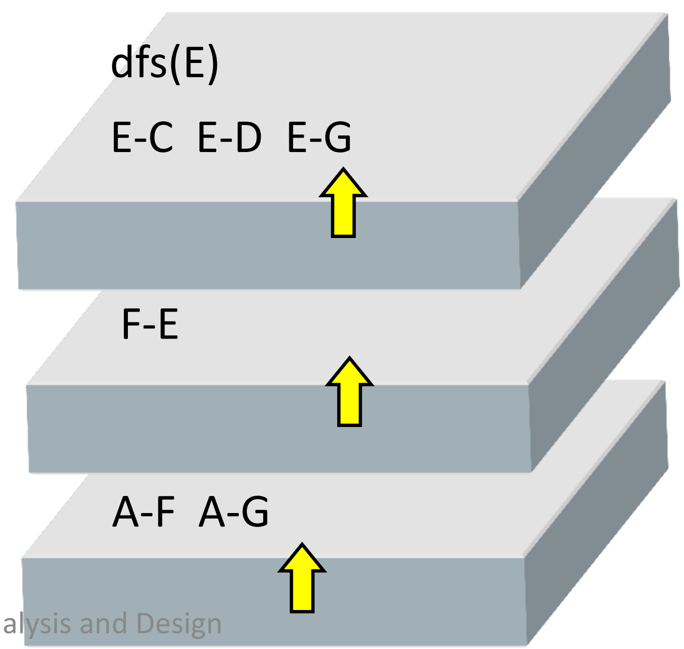




Function call stack:

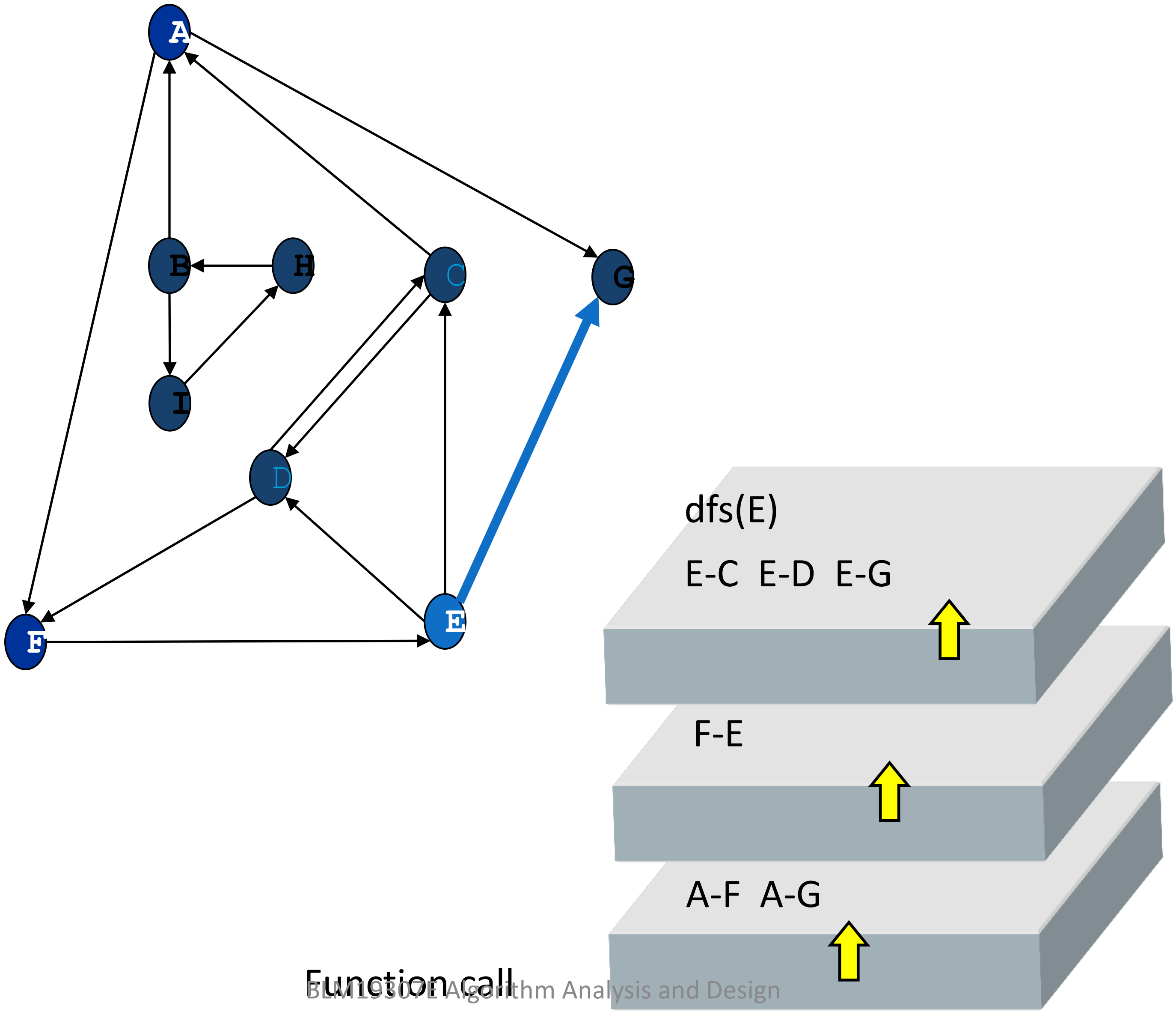
Depth-First Search





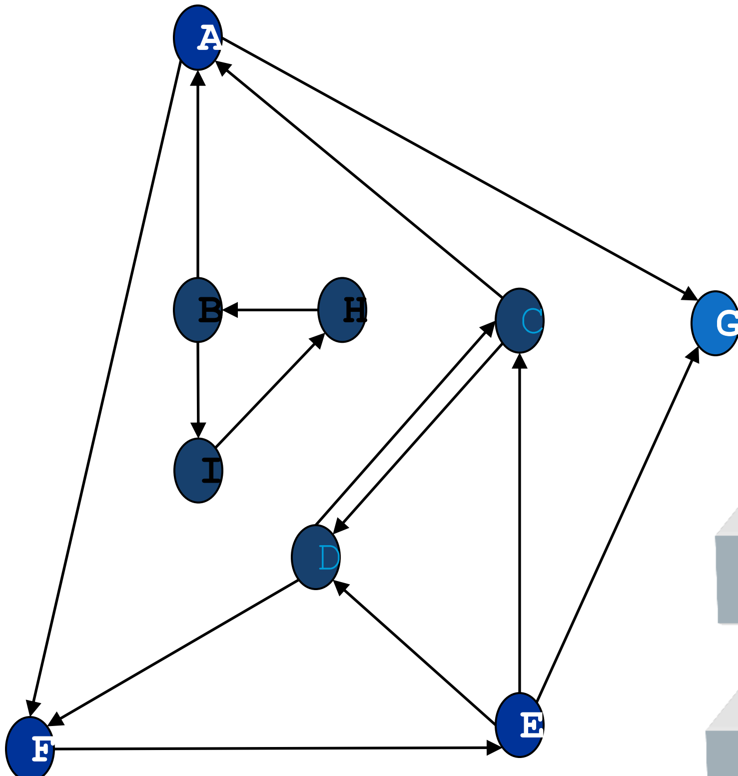
Function call stack:

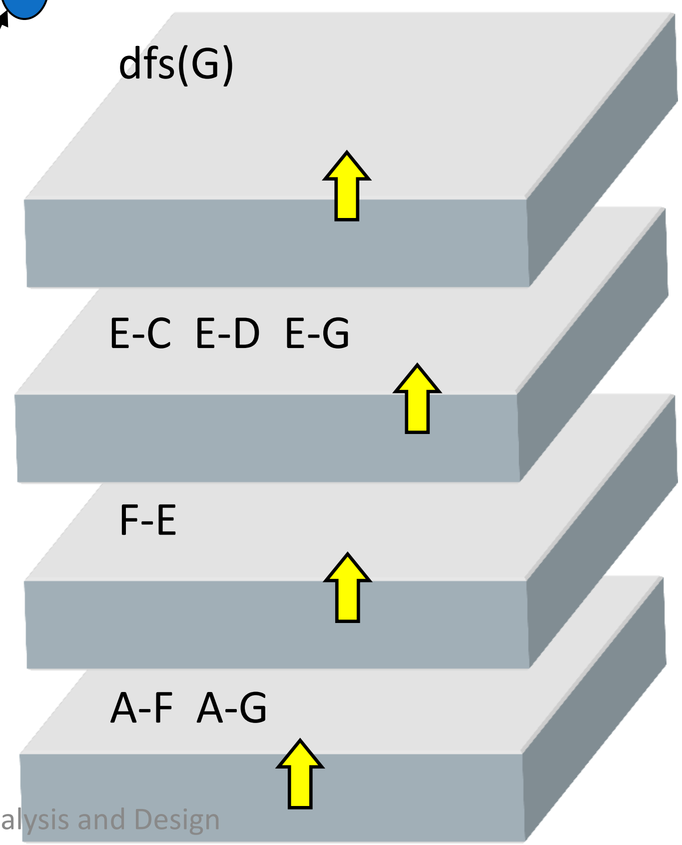
Depth-First Search



Function call stack:

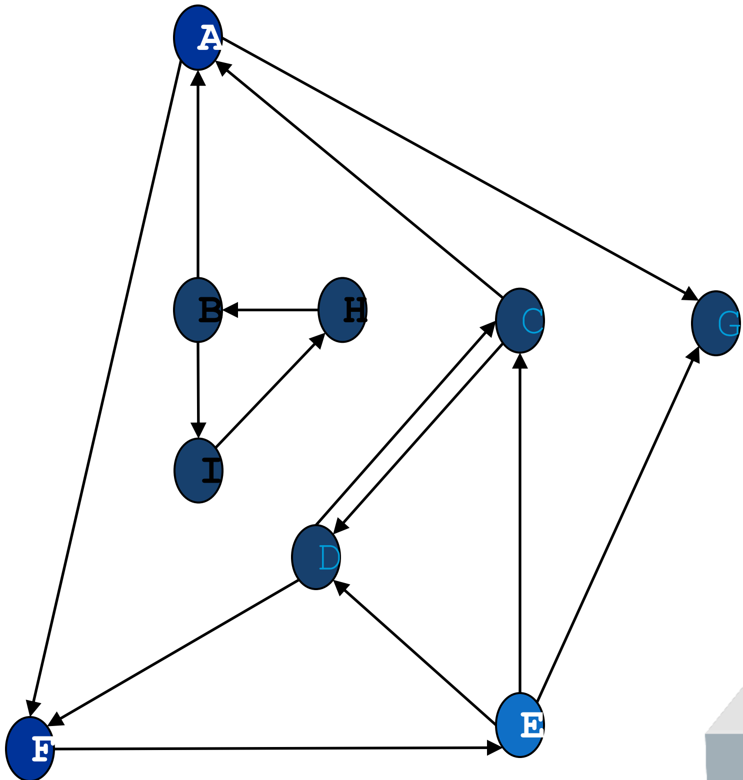
Depth-First Search

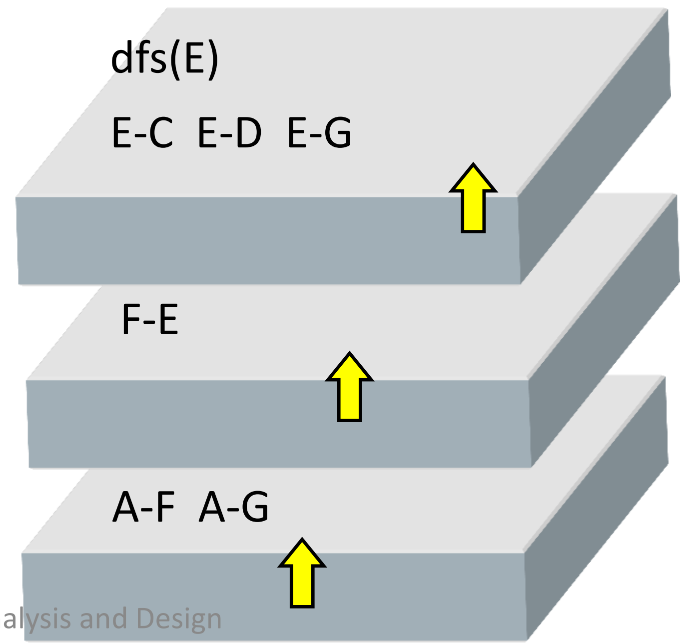




Function call stack:

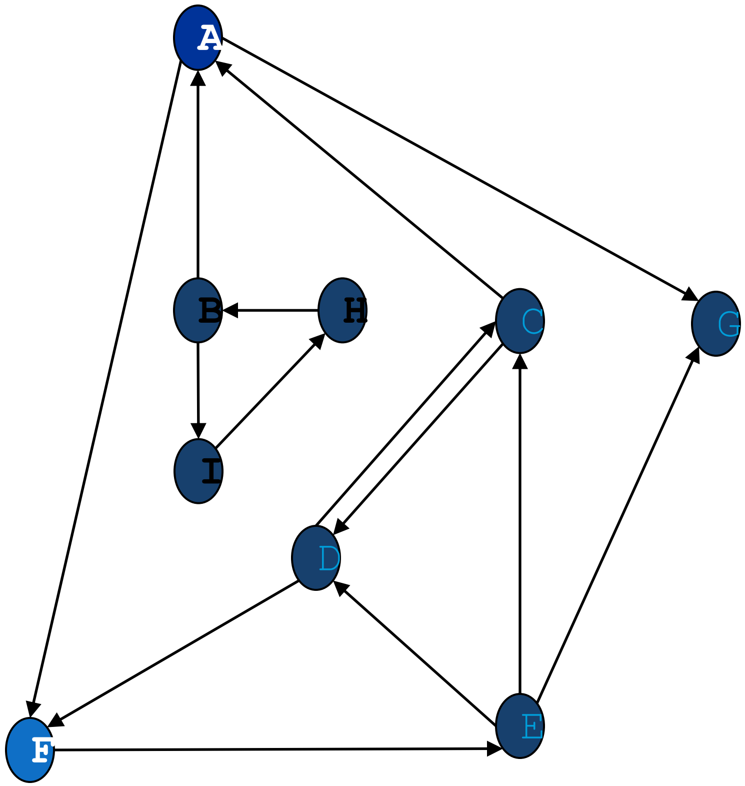
Depth-First Search

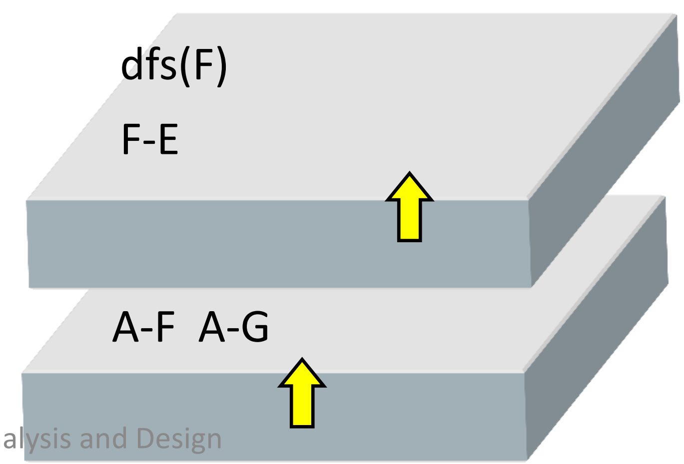




Function call stack:

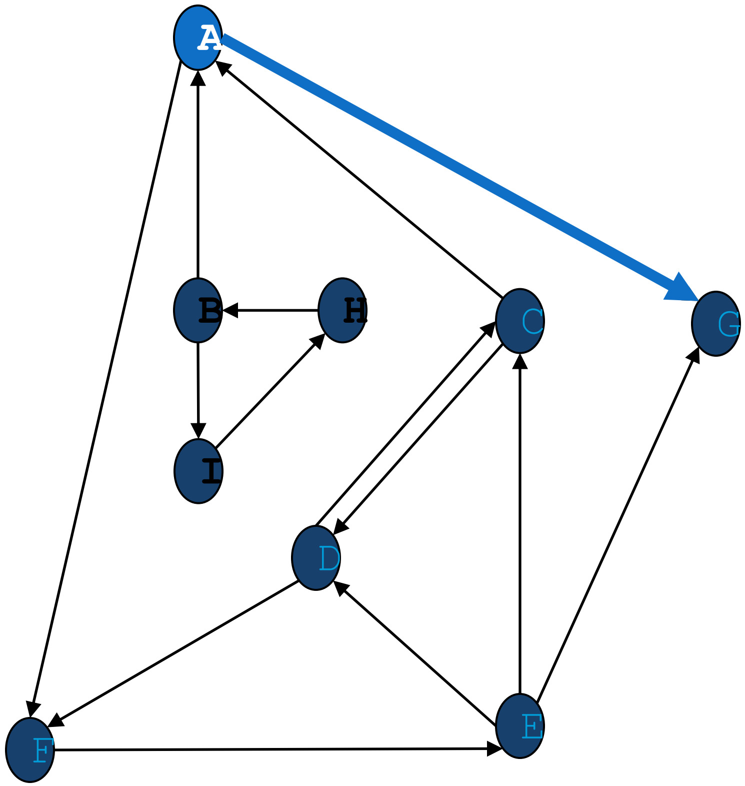
Depth-First Search

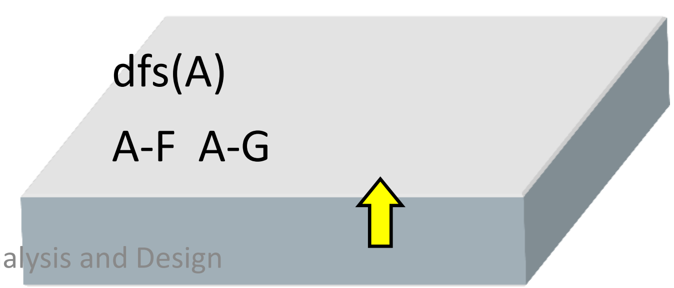




Function call stack:

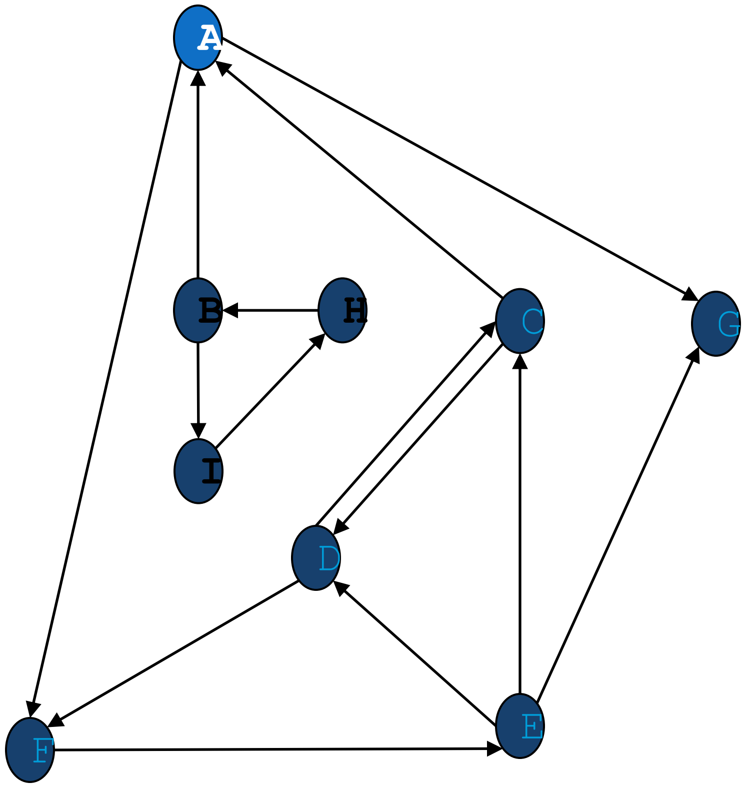
Depth-First Search

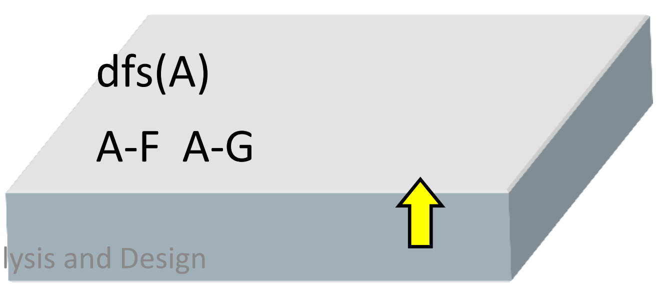




Function call stack:

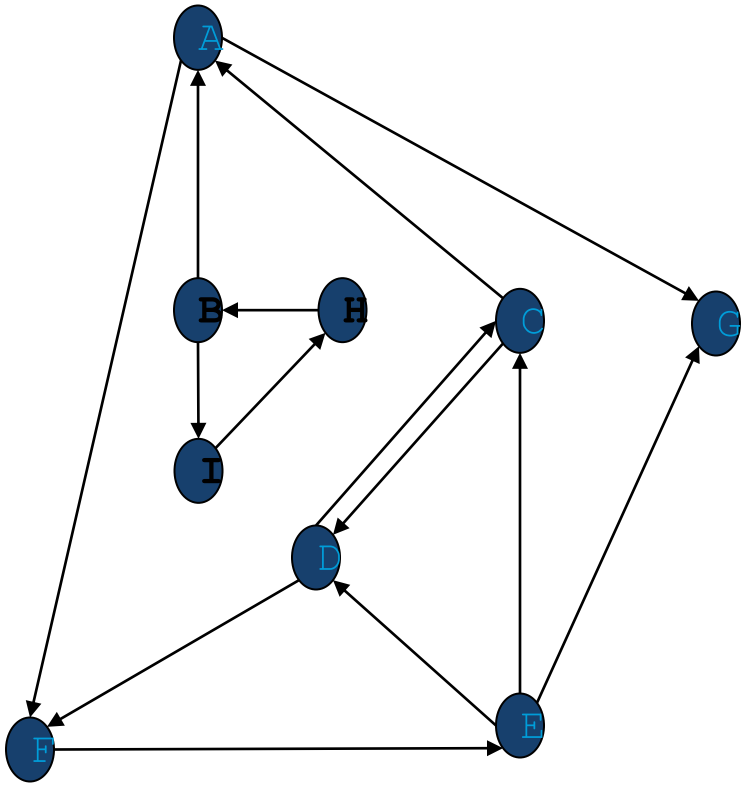
Depth-First Search





Function call stack:

Depth-First Search



Nodes reachable from A: A, C, D, E, F, G

Notes on DFS

- DFS can be implemented with graphs represented as:
  - adjacency matrices: $\Theta(V^2)$
  - adjacency lists: $\Theta(|V| + |E|)$

- Yields two distinct ordering of vertices:
  - order in which vertices are first encountered (pushed onto stack)
  - order in which vertices become dead-ends (popped off stack)

Applications of the DFS

- Detecting whether a graph is connected. Search the graph starting from any vertex. If the number of vertices searched is the same as the number of vertices in the graph, the graph is connected. Otherwise, the graph is not connected.
- Detecting whether there is a path between two vertices.
- Finding a path between two vertices.
- Finding all connected components. A connected component is a maximal connected subgraph in which every pair of vertices are connected by a path.
- Detecting whether there is a cycle in the graph.
- Finding a cycle in the graph.

Breadth-first search (BFS)

- Visits graph vertices by moving across to all the neighbors of last visited vertex
- Instead of a stack, BFS uses a queue
- Similar to level-by-level tree traversal
- “Redraws” graph in tree-like fashion (with tree edges and cross edges for undirected graph)
- https://www.cs.usfca.edu/~galles/visualization/BFS.html

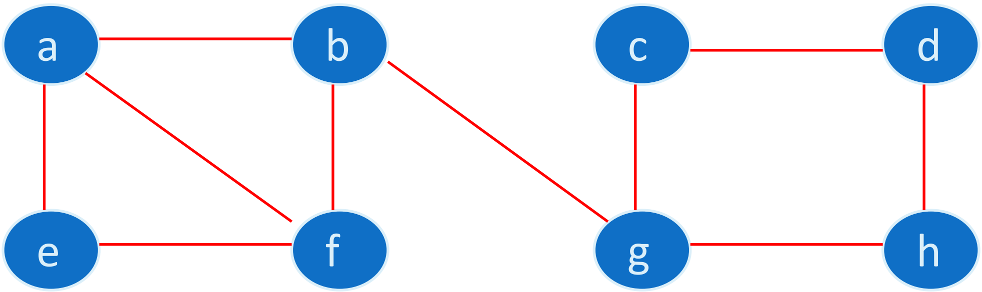
Pseudocode of BFS

ALGORITHM $BFS(G)$

```
//Implements a breadth-first search traversal of a given graph
//Input: Graph G = ⟨V, E⟩
//Output: Graph G with its vertices marked with consecutive integers
//            in the order they are visited by the BFS traversal
mark each vertex in V with 0 as a mark of being "unvisited"
count ← 0
for each vertex v in V do
    if v is marked with 0
        bfs(v)

bfs(v)
//visits all the unvisited vertices connected to vertex v
//by a path and numbers them in the order they are visited
//via global variable count
count ← count + 1;   mark v with count and initialize a queue with v
while the queue is not empty do
    for each vertex w in V adjacent to the front vertex do
        if w is marked with 0
            count ← count + 1;   mark w with count
            add w to the queue
    remove the front vertex from the queue
```

Example of BFS traversal of undirected graph



BFS traversal queue:

BFS tree:

Notes on BFS

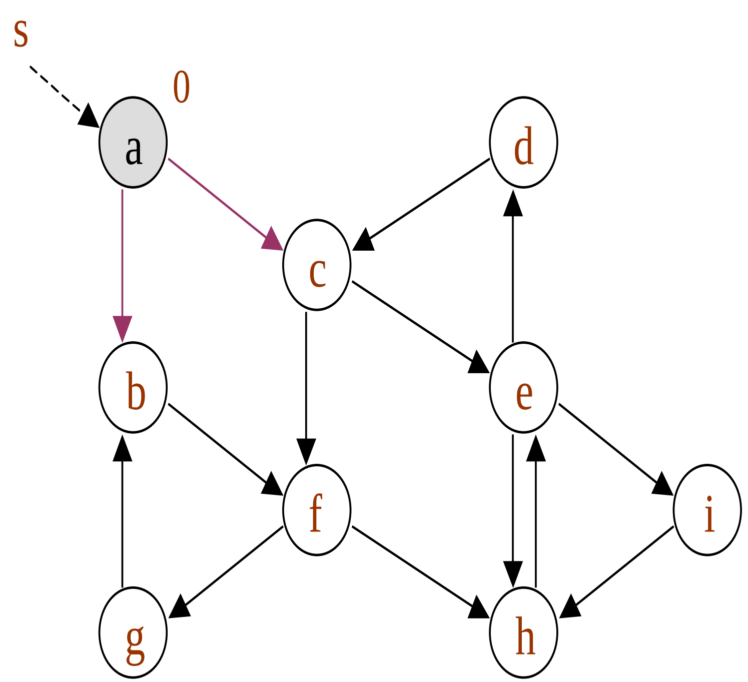
- BFS has same efficiency as DFS and can be implemented with graphs represented as:
  - adjacency matrices: $\Theta(V^2)$
  - adjacency lists: $\Theta(|V| + |E|)$
- Yields single ordering of vertices (order added/deleted from queue is the same)

Applications of the BFS

- Detecting whether a graph is connected. A graph is connected if there is a path between any two vertices in the graph.
- Detecting whether there is a path between two vertices.
- Finding a shortest path between two vertices. You can prove that the path between the root and any node in the BFS tree is the shortest path between the root and the node.
- Finding all connected components. A connected component is a maximal connected subgraph in which every pair of vertices are connected by a path.
- Detecting whether there is a cycle in the graph.
- Finding a cycle in the graph.
- Testing whether a graph is bipartite. A graph is bipartite if the vertices of the graph can be divided into two disjoint sets such that no edges exist between vertices in the same set

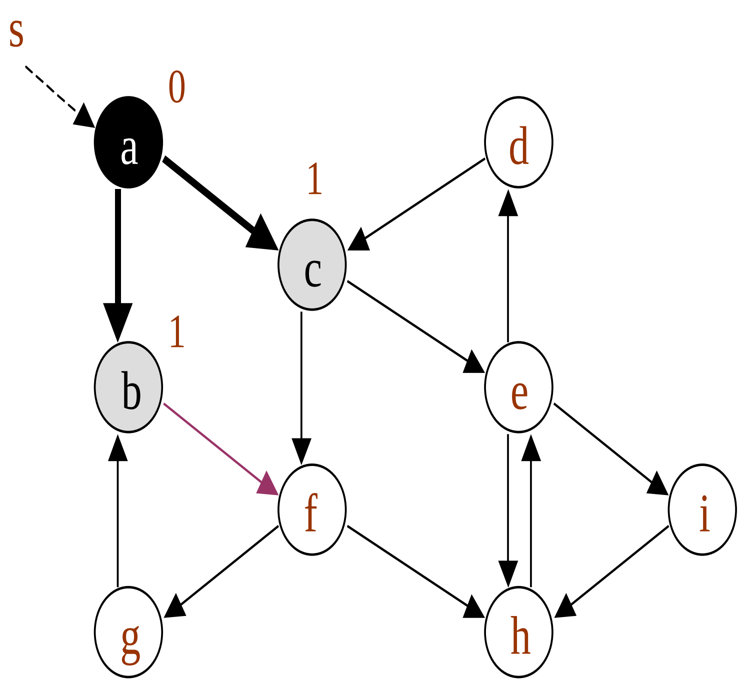
Breadth-First Search

Sample Graph:



FIFO queue Q	just after processing vertex
$\langle a \rangle$	-

Breadth-First Search

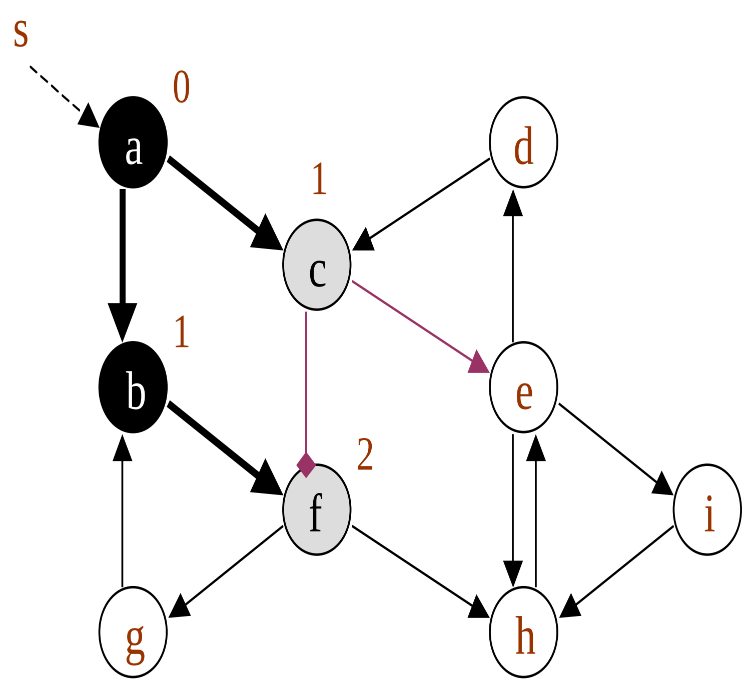


FIFO just after

queue Q processing vertex

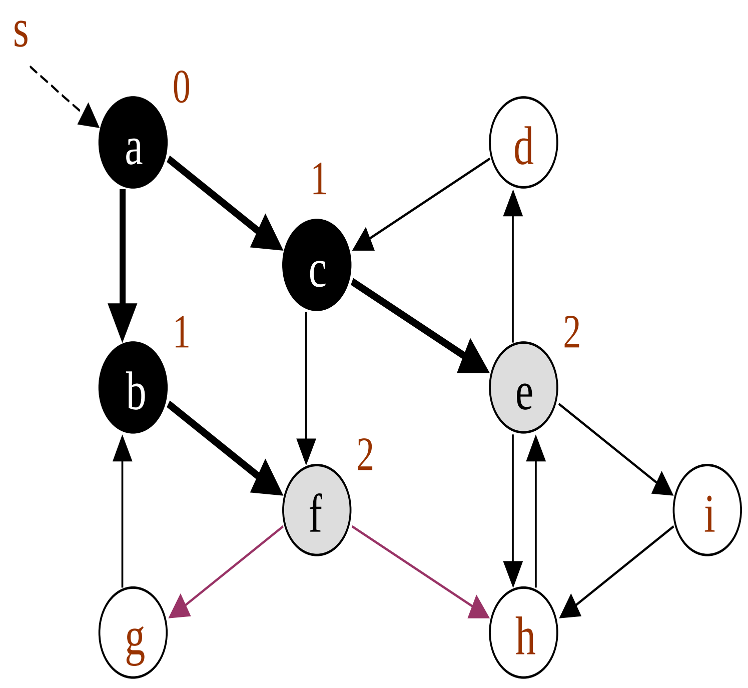
⟨a⟩	-
⟨a,b,c⟩	a

Breadth-First Search



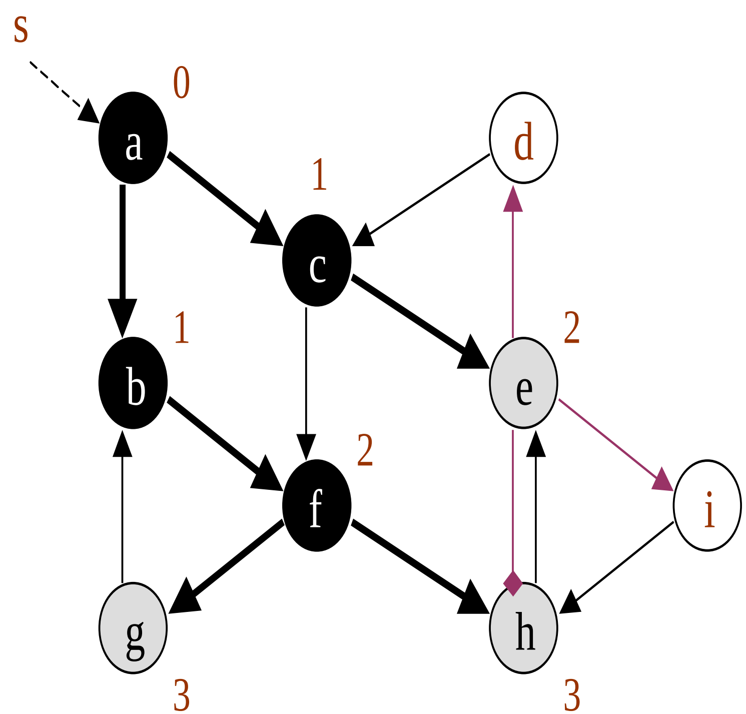
FIFO queue Q just after processing vertex	
⟨a⟩	-
⟨a,b,c⟩	a
⟨a,b,c,f⟩	b

Breadth-First Search



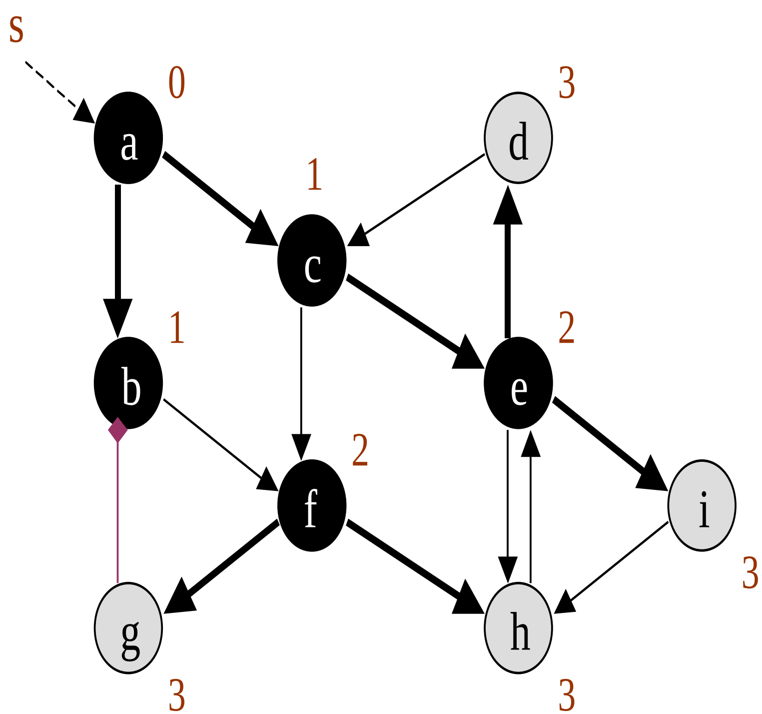
FIFO queue Q	just after processing vertex
⟨a⟩	-
⟨a,b,c⟩	a
⟨a,b,c,f⟩	b
⟨a,b,c,f,e⟩	c

Breadth-First Search



FIFO queue Q	just after processing vertex
⟨a⟩	-
⟨a,b,c⟩	a
⟨a,b,c,f⟩	b
⟨a,b,c,f,e⟩	c
⟨a,b,c,f,e,g,h⟩	f

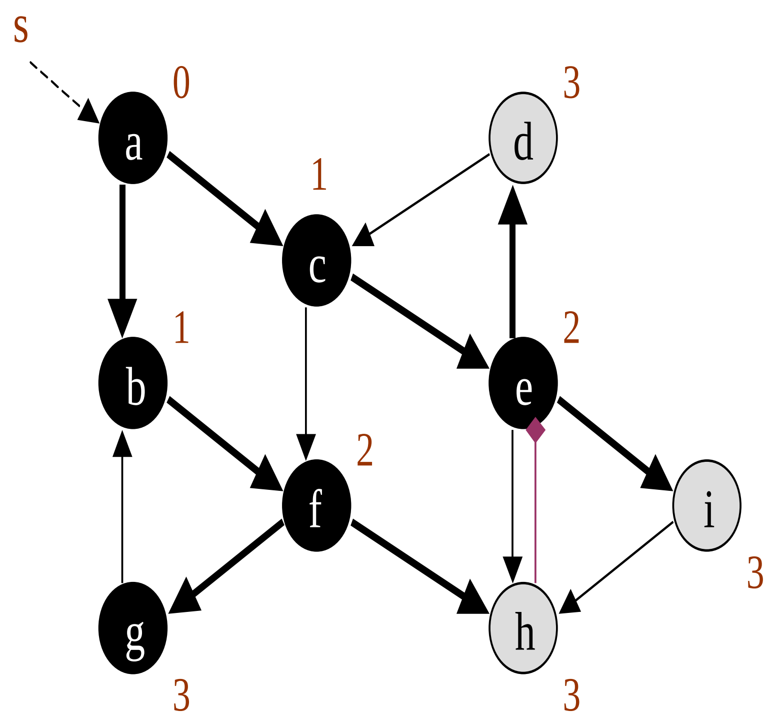
Breadth-First Search



FIFO queue Q	just after processing vertex
⟨a⟩	-
⟨a,b,c⟩	a
⟨a,b,c,f⟩	b
⟨a,b,c,f,e⟩	c
⟨a,b,c,f,e,g,h⟩	f
⟨a,b,c,f,e,g,h,d,i⟩	e

all distances are filled in after processing e

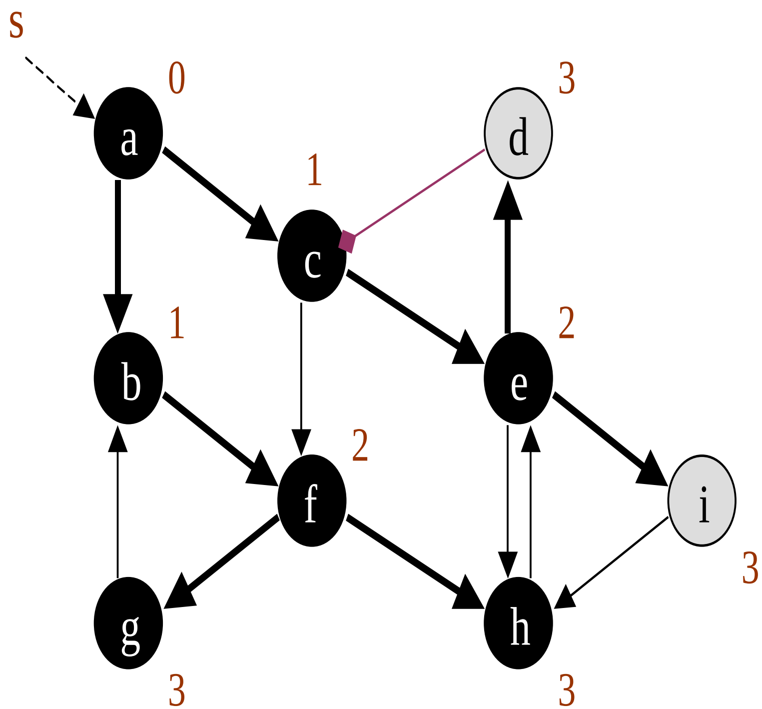
Breadth-First Search



FIFO queue Q just after processing vertex

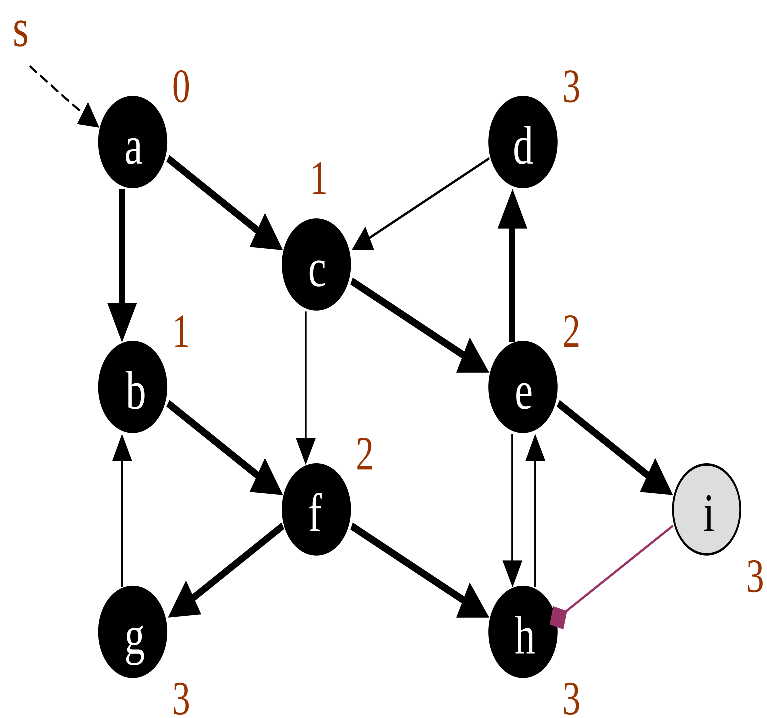
queue Q	vertex
⟨a⟩	-
⟨a,b,c⟩	a
⟨a,b,c,f⟩	b
⟨a,b,c,f,e⟩	c
⟨a,b,c,f,e,g,h⟩	f
⟨a,b,c,f,e,g,h,d,i⟩	g

Breadth-First Search



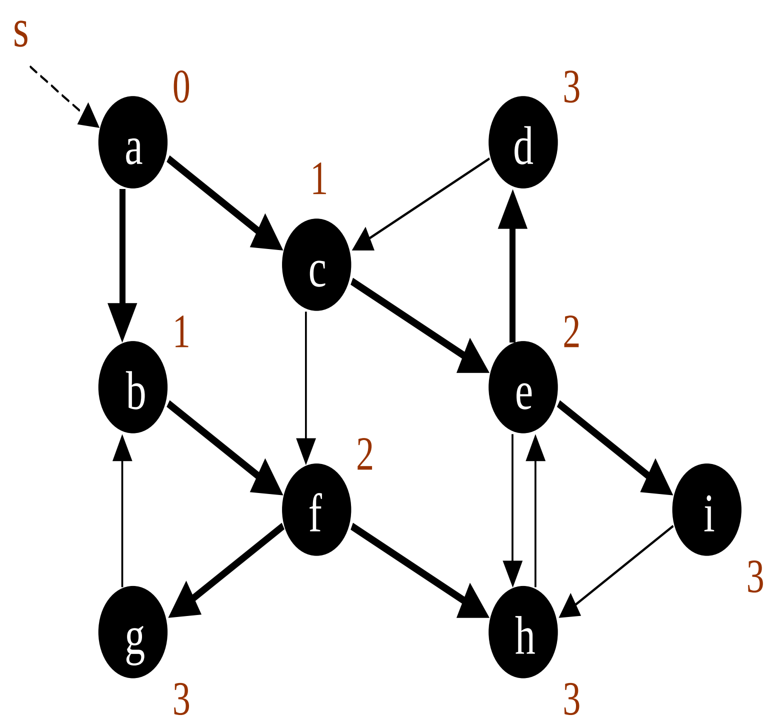
FIFO queue *Q*	just after processing vertex
⟨a⟩	-
⟨a,b,c⟩	a
⟨a,b,c,f⟩	b
⟨a,b,c,f,e⟩	c
⟨a,b,c,f,e,g,h⟩	f
⟨a,b,c,f,e,g,h,d,i⟩	h

Breadth-First Search



FIFO queue Q	just after processing vertex
⟨a⟩	-
⟨a,b,c⟩	a
⟨a,b,c,f⟩	b
⟨a,b,c,f,e⟩	c
⟨a,b,c,f,e,g,h⟩	f
⟨a,b,c,f,e,g,h,d,i⟩	d

Breadth-First Search



FIFO queue Q	just after processing vertex
⟨a⟩	-
⟨a,b,c⟩	a
⟨a,b,c,f⟩	b
⟨a,b,c,f,e⟩	c
⟨a,b,c,f,e,g,h⟩	f
⟨a,b,c,f,e,g,h,d,i⟩	i

algorithm terminates: all vertices are processed