CS340 - Analysis of Algorithms

Review: Discrete Math, Complexity, and Graphs

Logistics:

Office hours:

- Alternating Tuesday and Thursdays 7--9pm
- Cecilia will post each week on Piazza

Join the piazza and gradescope if you have not already

Make sure you have a goldengate account

- Email David Diaz ddiaz1@brynmawr.edu if you encounter issues

Upcoming deadlines:

Lab0 due tomorrow

HW1 due Monday (9/15)

Lab1 tomorrow

Quiz next class

Agenda

1. **Stable matching algorithm review**
2. Some math review
3. Complexity
4. Data structures
5. Graphs

Proof of Correctness:

For this algorithm to be correct, it must produce a perfect and stable matching.

A matching is perfect if every element of X and Y occurs in some pair.

Proof by contradiction: suppose there is some employer e who was not matched.

Function findStableMatching(E, A)
  All $e \in E$ and $a \in A$ are unpaired
  while *there is an e unpaired that hasn't made an offer to every a* **do**
    choose such an e
    let a be the highest-ranked applicant in e's preference list who e has not made an offer to yet
    if *a is unpaired* **then**
      pair a and e
    end
    else
      a is currently paired with e'
      if *a prefers e' to e* **then**
        e remains unpaired
      end
      else
        a is paired with e and e' becomes unpaired
      end
    end
  end
  return the set of pairs

Proof of Correctness:

Proof of stability by contradiction: Suppose there exists some e and some a’ such that e prefers a’ to its match a and a’ prefers e to its match e.

Either: (1) e never offered a job to a’ or (2) a’ rejected the offer

Function findStableMatching(E, A)
  All $e \in E$ and $a \in A$ are unpaired
  while *there is an e unpaired that hasn't made an offer to every a* **do**
    choose such an e
    let a be the highest-ranked applicant in e's preference list who e has not made an offer to yet
    if *a is unpaired* **then**
      pair a and e
    end
    else
      a is currently paired with e'
      if *a prefers e' to e* **then**
        e remains unpaired
      end
      else
        a is paired with e and e' becomes unpaired
      end
    end
  end
  return the set of pairs

Agenda

1. Stable matching algorithm review
2. **Some math review**
3. Complexity
4. Data structures
5. Graphs

Simplify the following equations:

1. $(3x^4y^2)(2x^3y^5)$

2. $(6x^7y^5) / (2x^3y^2)$

3. $((x^2y^3) / (z^4))^3$

Exponent Rules For $a \neq 0, b \neq 0$	
Product Rule	$a^x \times a^y = a^{x+y}$
Quotient Rule	$a^x \div a^y = a^{x-y}$
Power Rule	$\left(a^x\right)^y = a^{xy}$
Power of a Product Rule	$(ab)^x = a^x b^x$
Power of a Fraction Rule	$\left(\frac{a}{b}\right)^x = \frac{a^x}{b^x}$
Zero Exponent	$a^0 = 1$
Negative Exponent	$a^{-x} = \frac{1}{a^x}$
Fractional Exponent	$a^{\frac{x}{y}} = \sqrt[y]{a^x}$

Logarithms

1. log(5x) + log(2y)

2. log(100) - log(4)

3. 3·log(x) – ½·log(y)

Logarithmic Properties	
Product Rule	$\log_a(xy) = \log_a x + \log_a y$
Quotient Rule	$\log_a\left(\frac{x}{y}\right) = \log_a x - \log_a y$
Power Rule	$\log_a x^p = p \log_a x$
Change of Base Rule	$\log_a x = \frac{\log_b x}{\log_b a}$
Equality Rule	If $\log_a x = \log_a y$ then $x = y$

Polynomials

$f(n) = n^c$

- C is a constant
- Adding polynomials: combine like terms and simplify coefficients
- Example 1: $(3x^2 + 5x + 2) + (4x^2 - 3x + 7)$
  - $= (3x^2 + 4x^2) + (5x - 3x) + (2 + 7)$
  - $= 7x^2 + 2x + 9$
- Example 2: $(2x^2y + 3xy^2 + y) + (5x^2y - 2xy^2 + 4y)$
  - $7x^2y + xy^2 + 5y$

Exponentials

$f(n) = r^n$

- Constant base r
- Grows *much faster* than polynomials (constant factor, variable base)

Summations

- Constant series

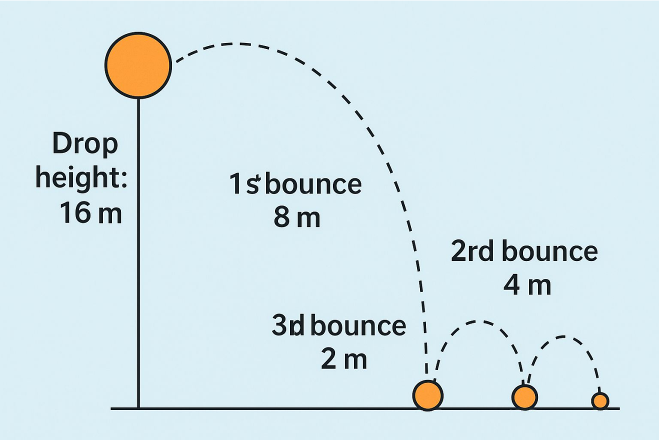
$$\sum_{i=a}^{b} 1 = \max(b - a + 1, 0)$$

- Arithmetic series

$$\sum_{i=1}^{n} i = 1 + 2 + \cdots + n = \frac{n(n+1)}{2} \qquad O(n^2)$$

- Geometric series

$$\sum_{i=0}^{n} c^i = 1 + c + c^2 \ldots + c^n = \frac{c^{n+1} - 1}{c - 1} \in \begin{cases} O(1), & c < 1 \\ O(c^n), & c > 1 \end{cases}$$



Summations

- Quadratic series

$$\sum_{i=1}^{n} i^2 = 1^2 + 2^2 + \cdots + n^2 = \frac{2n^3 + 3n^2 + n}{6}$$

O(n^3)

- Linear-geometric series

$$\sum_{i=1}^{n} ic^i = c + 2c^2 + \cdots + nc^n$$

$$= \frac{(n-1)c^{n+1} - nc^n + c}{(c-1)^2}$$

O(nc^n)

Summations

- Harmonic series

$$H_n = \sum_{i=1}^{n} \frac{1}{i} = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} = \ln n + O(1)$$

Agenda

1. Stable matching algorithm review
2. Some math review
3. **Complexity**
4. Data structures
5. Graphs

Now let’s relate this to algorithms...

Basics of Algorithm Analysis

- Last class we discussed trade-offs between two algorithms
  - “How does algorithm 1’s time and space usage grow with increasing input size?”

Table 2.1 The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds 10^{25} years, we simply record the algorithm as taking a very long time.

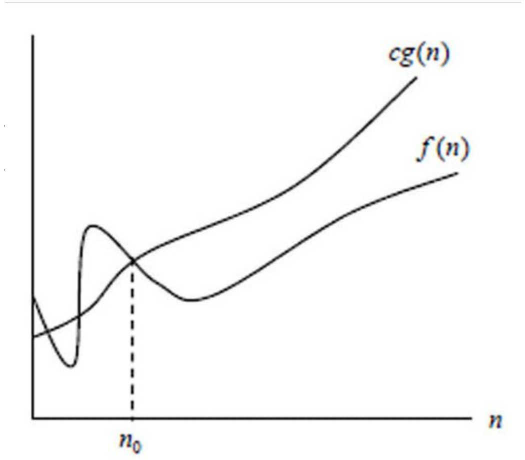
	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1,000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10,000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100,000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1,000,000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

Big O

Big O notation expresses **Asymptotic Upper Bounds**

"If f(n) doesn't grow faster than g(n), up to a constant factor, for large enough n."

If $\exists\ n_0 \geq 0, c > 0 : f(n) \leq c \cdot g(n)\ \forall\ n \geq n_0$ then $O(f(n)) = g(n$



Basics of Algorithm Analysis

- Q: Why do we simplify to only keep the dominant term in Big-O notation?
  - Worst case run time of $2n$ expressed as $O(n)$
  - Worst case run time of $n^2 + 3n$ expressed as $O(n^2)$
- A:
  - We'd like to classify run times at a coarse level of granularity so we can see similarities among different algorithms
  - Number of steps is arbitrary (language and machine dependant)

What is the Asymptotic Upper Bound of each of these functions? Write in Big-O notation

1. $f_2(x) = 15x^2 + 7x \log^3 x$
   $= O(x^2)$

2. $f_1(x) = 43x^2 \log^4 x + 12x^3 \log x + 52x \log x$
   $= O(x^3 \log x)$

3. $f_3(x) = 3x + 4 \log_5 x + 91x^2$
   $= O(x^2)$

4. $f_4(x) = 13 \cdot 3\text{^}(2x+9) + 4x^9$
   $= O(9\text{^}x)$

5. $f_5(x) = \sum$ (from $x=0$ to ∞) $1/2\text{^}x$
   $= O(1)$

Basics of Algorithm Analysis

- Sublinear time: O(logn) - algorithms where we can throw away a constant fraction of the input with each step of the algorithm
    - Binary search
    - Queries in a *balanced* BST
- Linear Time Algorithms: O(n) - run time is at most a constant factor times the input size
    - Process the input in a single pass spending constant time on each item
    - min/max algorithm, linear search
- O(nlogn): Sorting algorithms
- Quadratic time: $O(n^2)$ - nested loops
    - Processing all pairs in a list (every employer and every applicant
- Polynomial time: $O(n^k)$
    - Consider all subsets of size k
- Exponential time: $O(2^n)$
    - All subsets of a set of size n
- Factorial time: O(n!)
    - Number of ways to match n items with eachother

Agenda

1. Stable matching algorithm review
2. Some math review
3. Complexity
4. **Data structures**
5. Graphs

Asymptotic Upper Bound of Common Operations

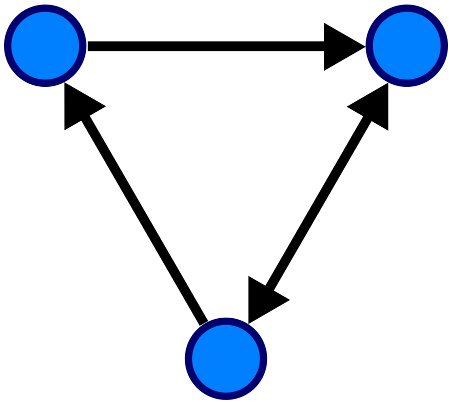
	Access	Search	Insertion	Deletion
Array				
Linked List				
Stack				
Queue				
BST				
AVL Tree				
Hash Map				

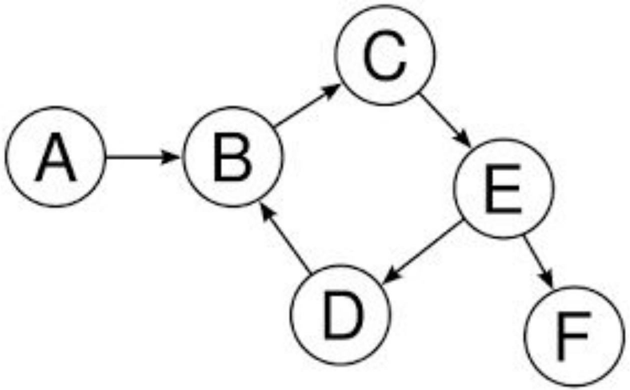
Agenda

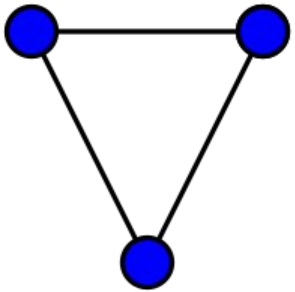
1. Stable matching algorithm review
2. Some math review
3. Complexity
4. Data structures
5. **Graphs**

Graphs

- A way of representing relationships between pairs of objects
- Consist of ***Vertices (V)*** with pairwise connections between them ***Edges (E)***
- A ***Graph G*** is a set of vertices and edges **(V, E)**

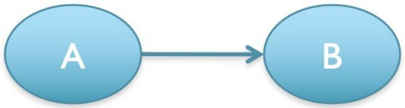






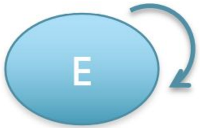
Edges

- An edge (u, v) connects vertices u and v
- Edges can be ***directed*** or ***undirected***
- An edge is said to be ***incident*** to a vertex if the vertex is one of the endpoints
- The ***degree*** of a vertex **deg(v)** is
  - the number of incident edges on v if in an undirected graph
  - the number of outgoing edges in a directed graph



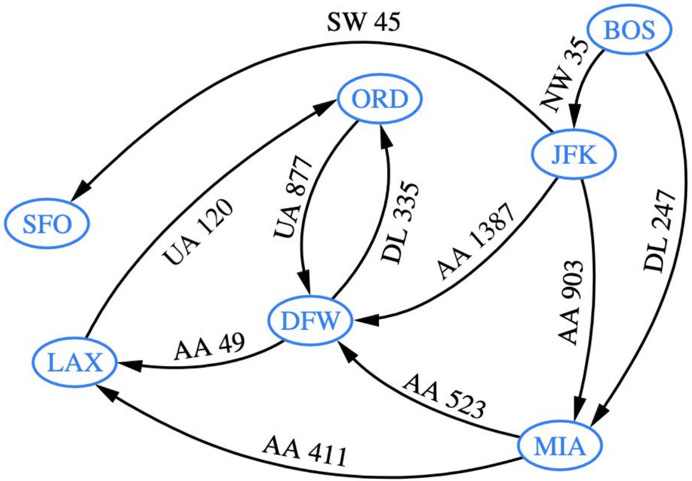
Directed Edge

Undirected Edge



Self Edge
(Unusual but usually allowed)

Directed vs Undirected Graphs



Example of a directed graph representing a flight network.

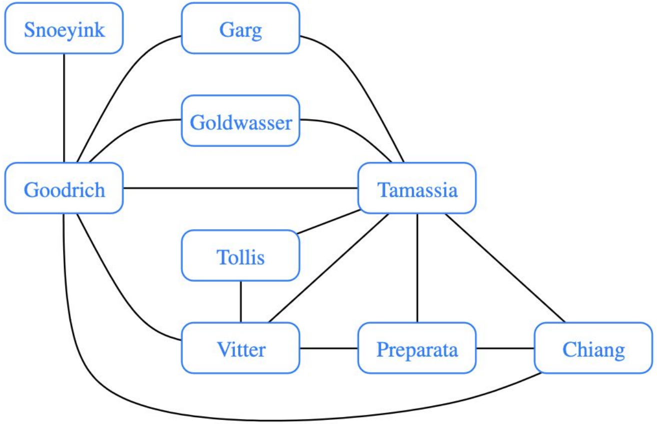
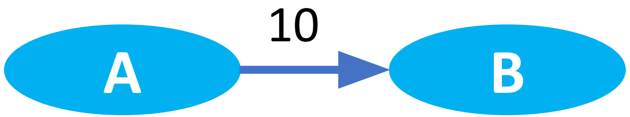


Figure 14.1: Graph of coauthorship among some authors.

Weighted Graphs

Edges have weights/costs



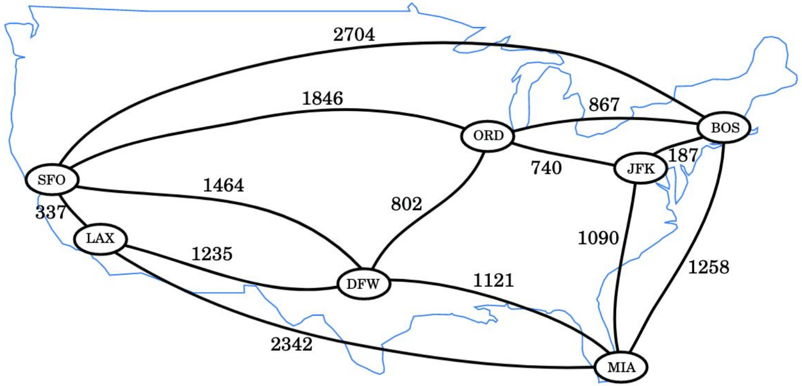
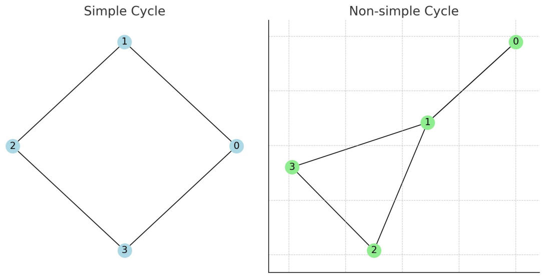


Figure 14.14: A weighted graph whose vertices represent major U.S. airports and whose edge weights represent distances in miles. This graph has a path from JFK to LAX of total weight 2,777 (going through ORD and DFW). This is the minimum-weight path in the graph from JFK to LAX.

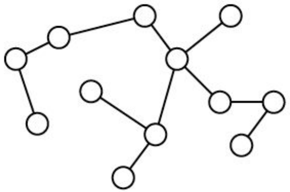
More terminology



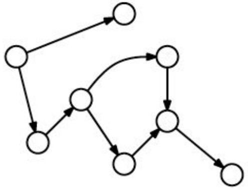
- A *path* in a graph is a sequence of vertices $\langle v_0, \ldots, v_k \rangle$ such that (v_{i-1}, v_i) is an edge for $i = 1, \ldots, k$
- The *length* of a path is the number of edges, k.
- A *cycle* is a path containing at least one edge and for which $v_0 = v_k$
- A cycle is simple if its edges and vertices (except for v_0 and v_k) are distinct.

More terminology

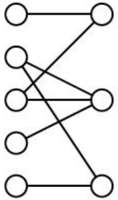
- An *acyclic* graph contains no simple cycles
- An acyclic connected graph is a *tree*
- The vertices of a *bipartite* graph can be partitioned into two disjoint subsets, V_1 and V_2 such that all edges have one endpoint in V_1 and the other one in V_2



(Free) Tree



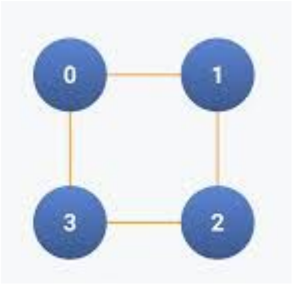
DAG

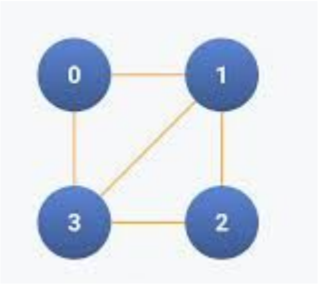


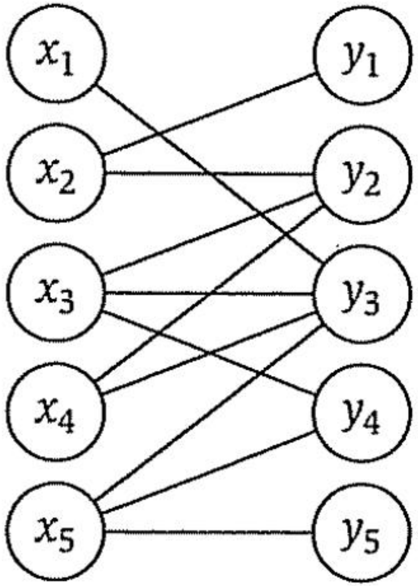
Bipartite graph

Bipartite Graphs

Is this graph bipartite?







Basic Graph Combinatorics

For an undirected graph:

- $|E| = 0 \leq m \leq \binom{n}{2}$
- $= \frac{n(n-1)}{2} \in O(n^2)$
- $\sum_{v \in V} \deg(v) = 2m$

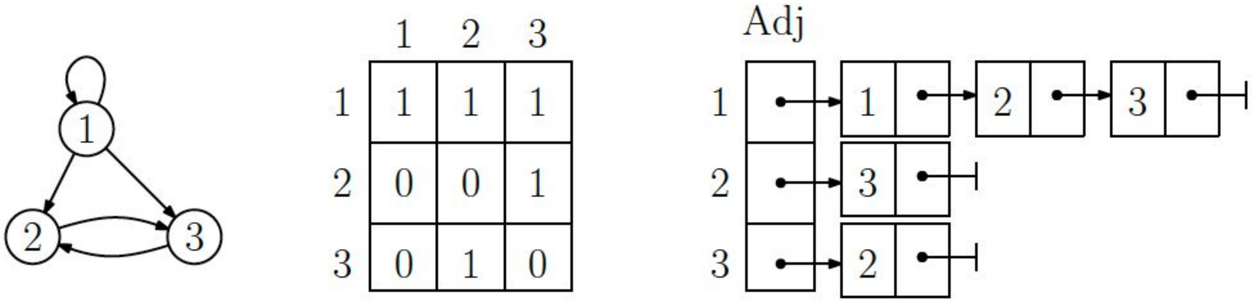
Basic Graph Combinatorics

For a directed graph:

- $|E| = 0 \leq m \leq n^2 - n$
- $\in O(n^2)$
- $\sum_{v \in V} \text{indeg}(v) = m$
- $\sum_{v \in V} \text{outdeg}(v) = m$

Representing a graph

- Adjacency matrix: An $n \times n$ matrix defined for $1 \leq$ (u, v) $\leq n$: A [u , v] $= 1$ if $(u, v) \in E$
- Adjacency list: An array of pointers where for $1 \leq v \leq n$, $Adj[v]$ points to a list containing the vertices that are adjacent to v



	1	2	3
1	1	1	1
2	0	0	1
3	0	1	0

Adjacency matrix

Adjacency list

Representing a graph - Adjacency List

Runtime Complexity: (In terms of V and E

- addVertex:
  - O(V)
- addEdge:
  - O(E)
- removeVertex:
  - O(V*E)
- removeEdge:
  - O(E)

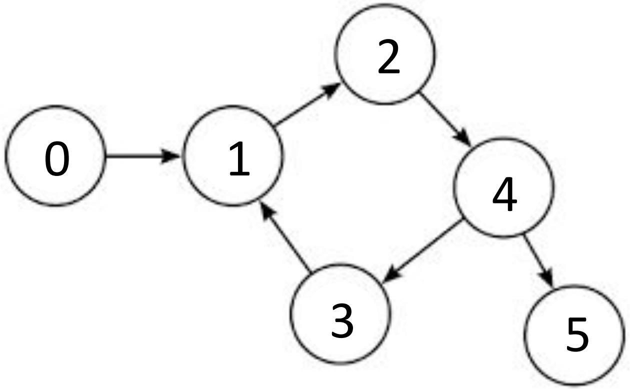
Representing a graph

Adjacency Matrix -

each index in the array is another array

Maintains an VxV matrix

where each slot (i,j) represents an outgoing edge from i to j



	1				
		1			
				1	
	1				
			1		1

Representing a graph - Adjacency Matrix

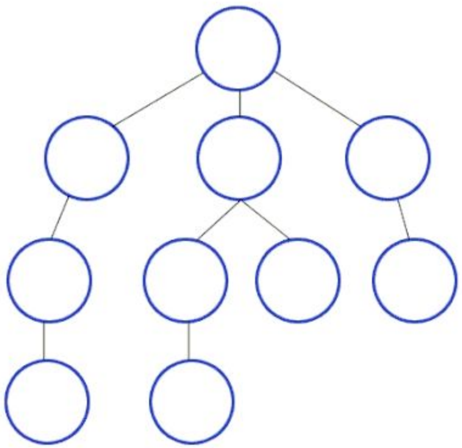
Runtime Complexity: (In terms of V and E)

- addVertex:
  - O(V^2) if we need to expand
- addEdge:
  - O(1)
- removeVertex:
  - O(V^2)
- removeEdge:
  - O(1)

Graph Traversals

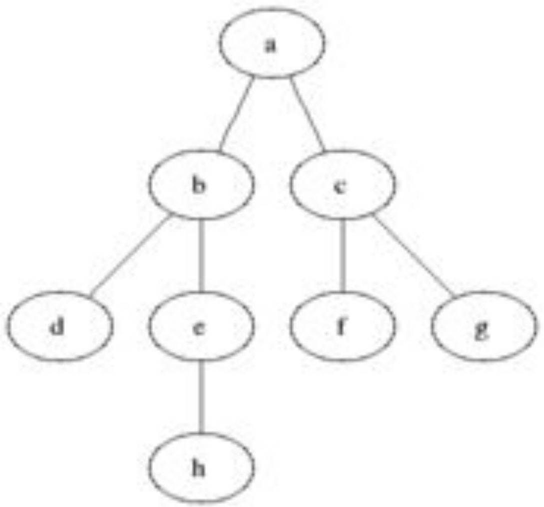
Depth First Search (DFS)

- start at root node and explore as far as possible along each branch
- Pre-order, in-order, and post-order traversals are DFS



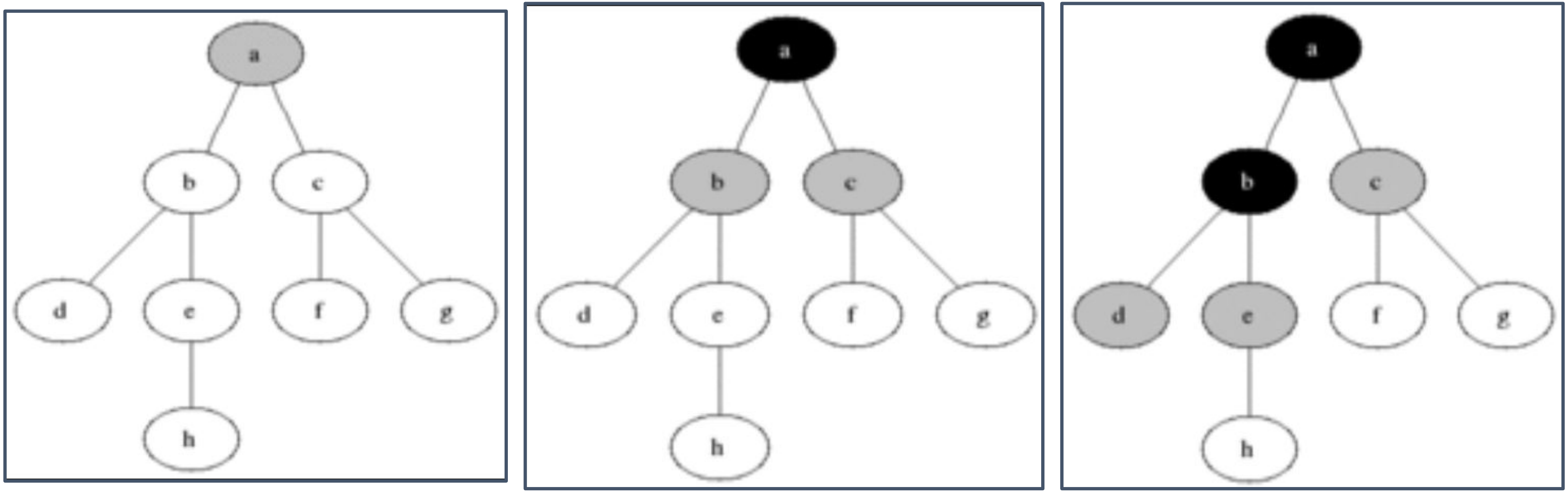
Breadth First Search (BFS)

- Starts at the root and explores all nodes at the present “depth” before moving to nodes on the next level
- Extra memory is usually required to keep track of the nodes that have not yet been explored

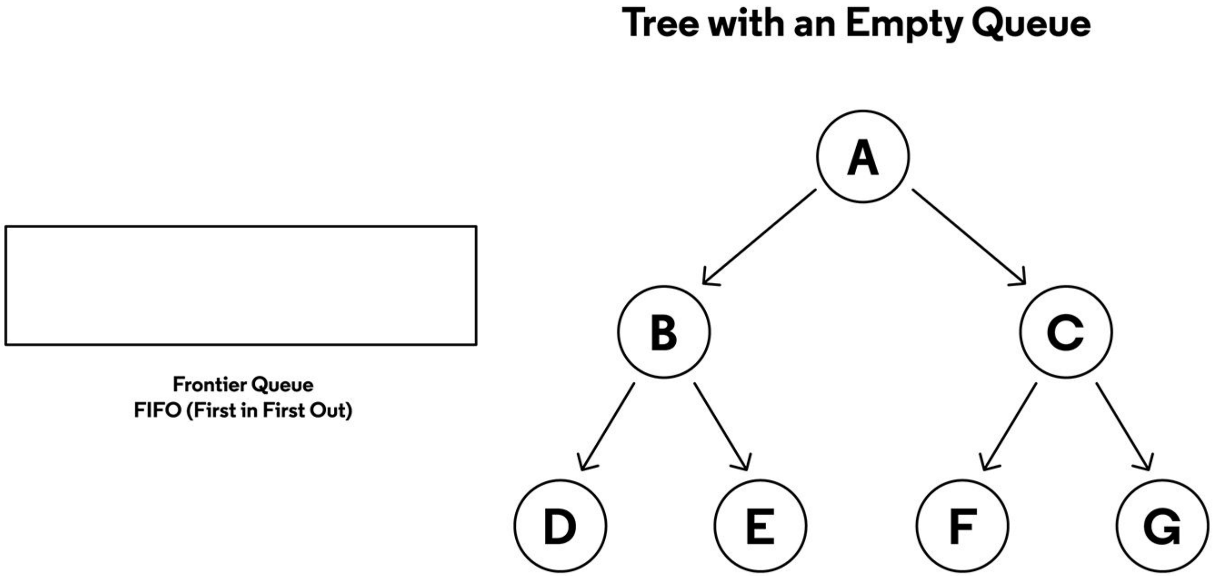


Breadth-First Traversal

pseudo-code?



Breadth First Search (BFS)



https://www.codecademy.com/article/tree-traversal

Runtime Analysis of BFS

$|V| = n, |E| = m$

1. Initialization:
   a. $O(n)$

2. Traversal
   a. While-loop: we visit each vertex once
   b. Nested for-loop: we visit each child of that vertex
      i. Number of iterations depends on the *degree*

$$
\begin{aligned}
T(n) &= n + \Sigma_{u \in V} (\deg(u) + 1) \\
&= n + (\Sigma_{u \in V} \deg(u)) + n = 2n + \Sigma_{u \in V} \deg(u) \\
&= 2n + 2m \\
&= O(n + m)
\end{aligned}
$$

Summary

1. Know the following:
    a. Exponent, logarithm, and summation rules and how to simplify equations
    b. Growth rate of classes of algorithms (logn grows slower than n which grows slower than n^2...)
    c. Asymptotic Upper Bound of functions (given a function write in Big-O notation)
    d. Data structures - asymptotic upper bound of basic operations
    e. Graphs - know terminology, basic rules, how to represent them, and traversal algorithms

2. Upcoming deadlines:
    a. Lab 0 due tomorrow
    b. HW1 due Sep 15th (next Monday)
    c. Continue reading textbook

3. Next class: more basics of algorithmic analysis
    a. **Quiz** on basic math rules, complexity, and DS review