CSE332: Data Abstractions

Section 6

HyeIn Kim

Winter 2014

Section Agenda

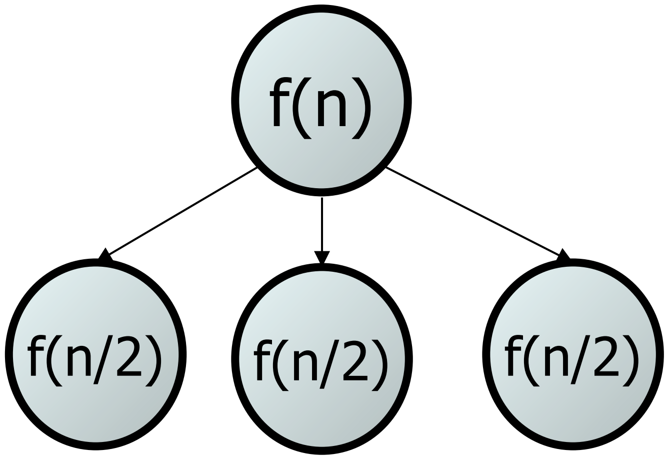
- **Another View of Recurrence**
- **Graph Review**
  - Graph Terminologies
  - Graph Representations: matrix & list
  - Topological sort
  - Graph traversal: BFS, DFS
- **Project 3 Introduction**
  - Analyzing US census data

Recurrence Relations

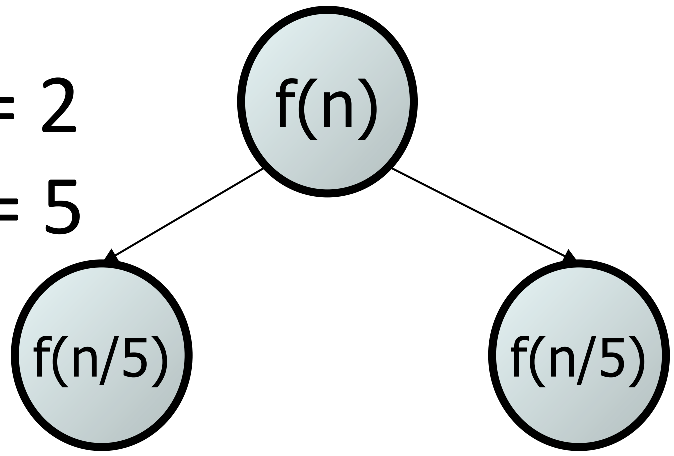
- **T(n) = a * T(n / b) + f(n)**

  - a: Branching factor
  - b: Work reduction
  - f(n): Work

a = 3
b = 2



a = 2
b = 5



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 $[v_0, v_1, \ldots, v_n]$, 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

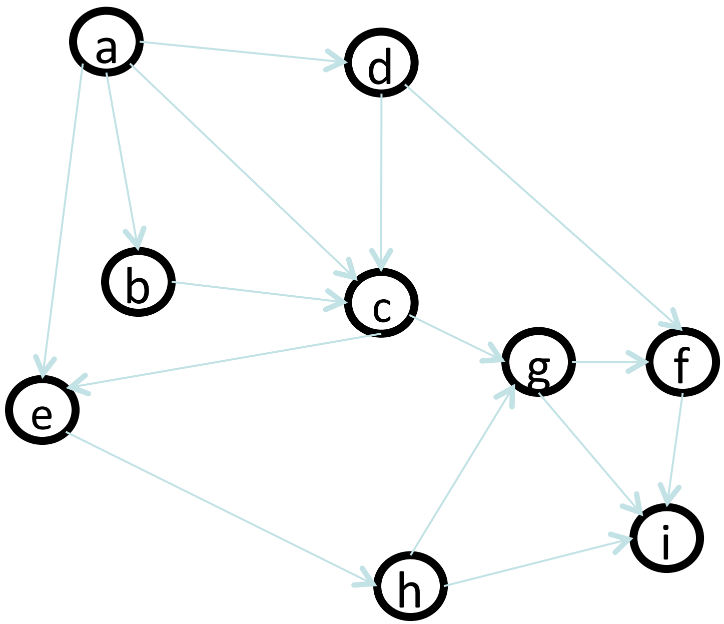
Delete an edge

Find an edge

Compute in-degree of a vertex

Compute out-degree 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:** $|V|^2$

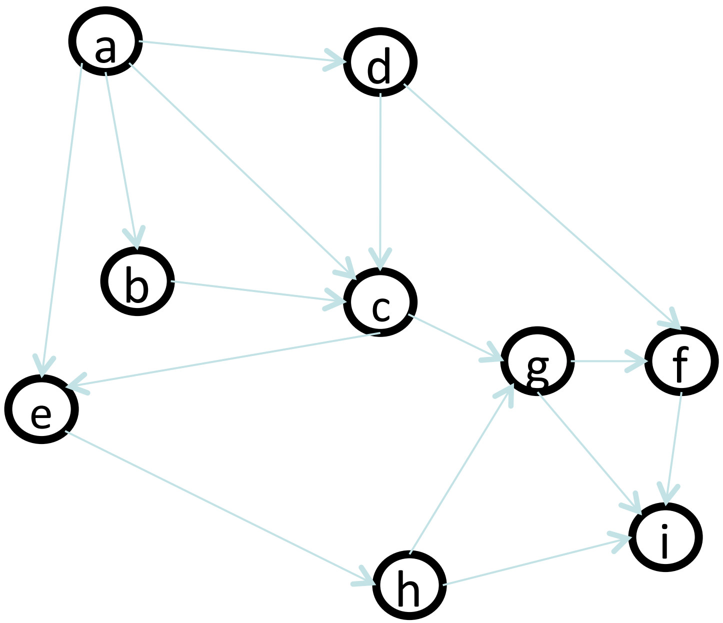
Adjacency Matrix

- **Get in-degree:** O(|V|)
- **Get out-degree:** 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 = ave out-degree

- **Get in-degree:** O(|V|+|E|)
- **Get out-degree:** O(d), O(1) possible
- **Find an edge:** O(d)
- **Insert an edge:** O(d)
- **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				

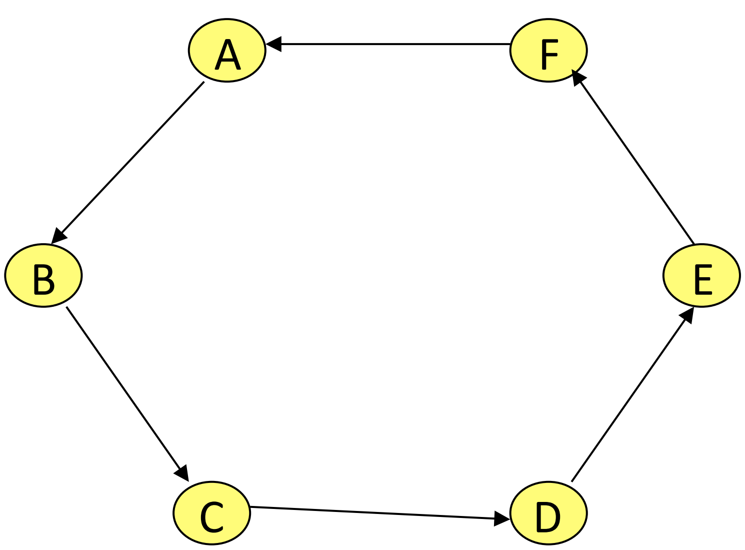
- Sparse graph |V| >>> d, so good for sparse graph

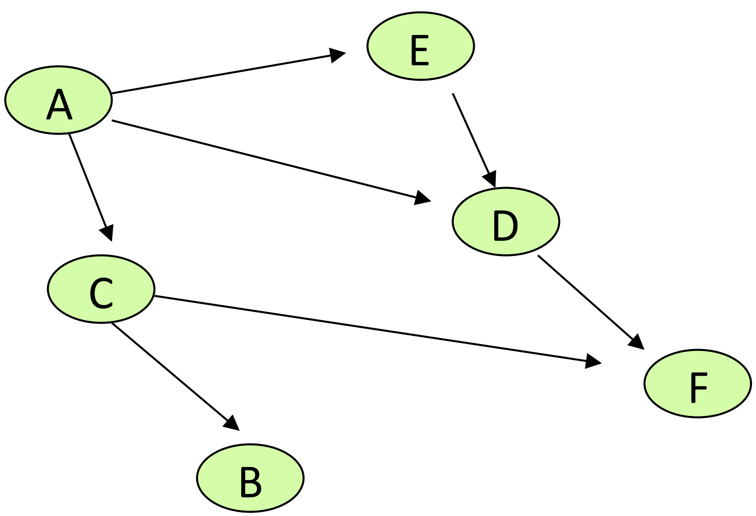
Topological Sort

Get linear order of tasks
with dependencies

Topological Sort

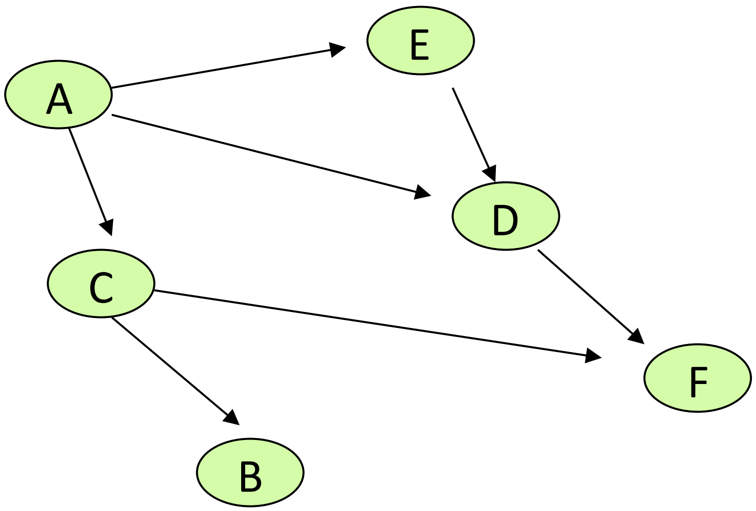
- Given a set of tasks with precedence constraints, find a linear order of the tasks
  - No topological ordering in graph with cycle
  - Possible to have many topological ordering

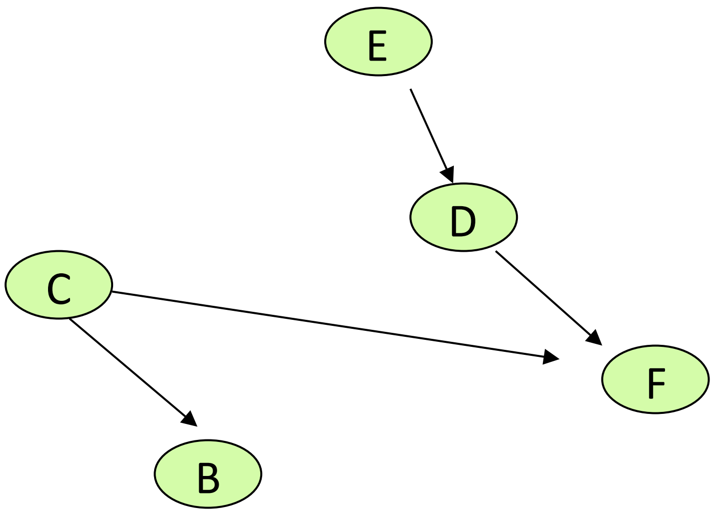




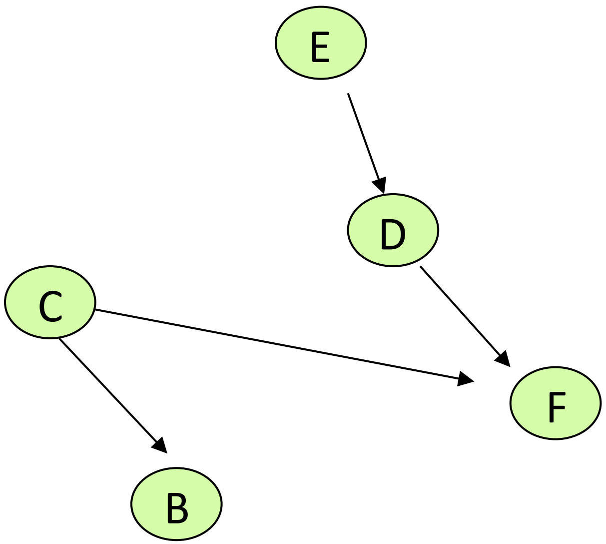
Topological Sort

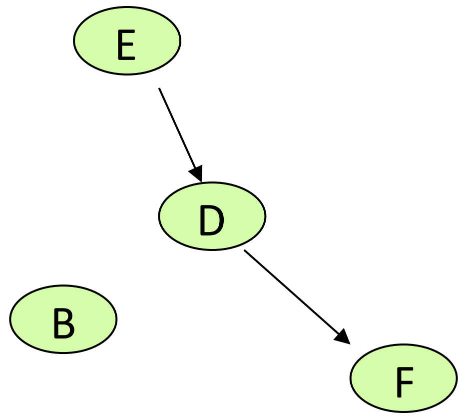
- **Topological sort algorithm**
  - Choose a vertex v with in-degree 0
  - Output v & Remove v and all of its edges
  - Repeat until no more vertices left

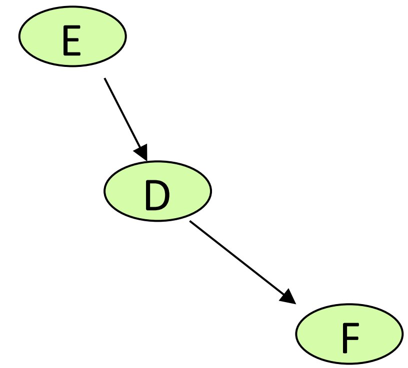


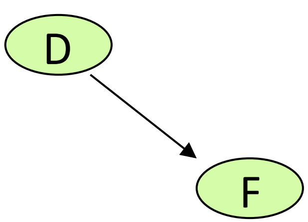


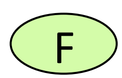
Topological Sort











A C B E D F

Topological Sort

- **Topological sort Runtime**

  - Choose a vertex v with in-degree 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:** $O(|V|^2+|E|) \sim O(|V|^2)$ No Queue

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

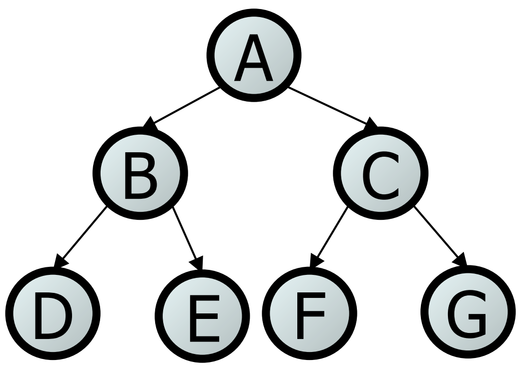
Graph Traversal

BFS & DFS

Breadth First Search

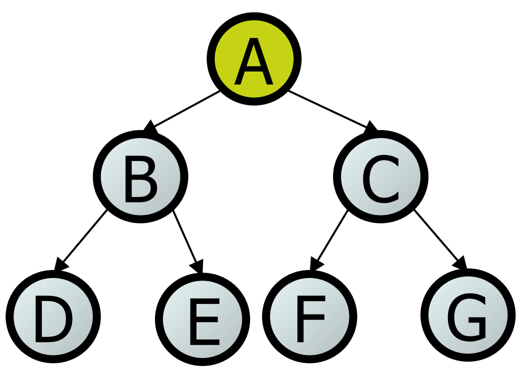
- **Pick the shallowest unmarked node**

  - Use queue, new node go to the end



Start with the root in the queue

Queue	A



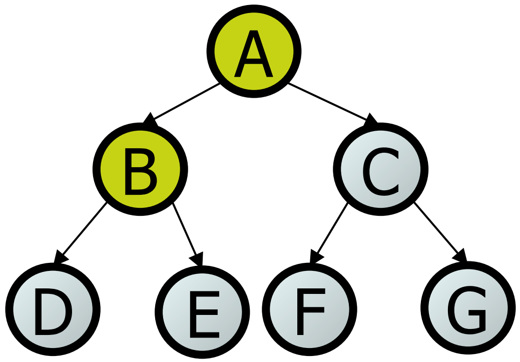
Pop one out, mark it,
put its child into the queue

Queue	B	C

Breadth First Search

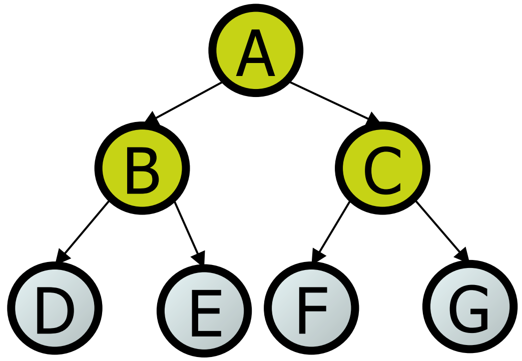
- **Pick the shallowest unmarked node**

  - Use queue, new node go to the end



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

Queue	C	D	E



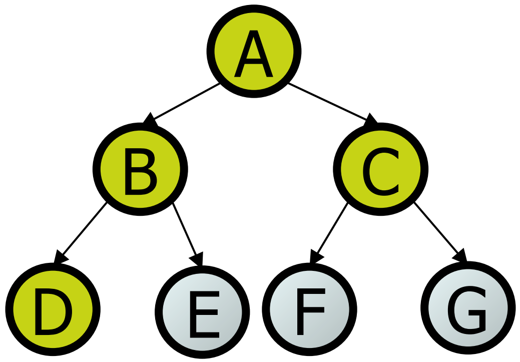
Pop one out, mark it,
put its child into the queue

Queue	D	E	F	G

Breadth First Search

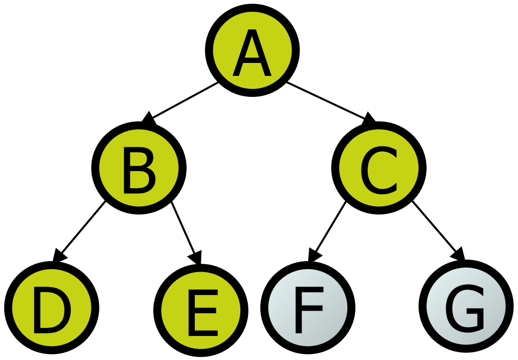
- **Pick the shallowest unmarked node**

  - Use queue, new node go to the end



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

Queue	E	F	G



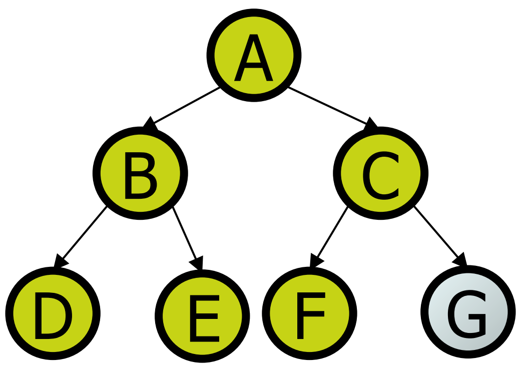
Pop one out, mark it,
put its child into the queue

Queue	F	G

Breadth First Search

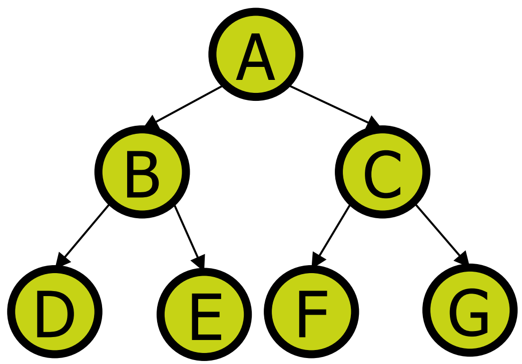
- **Pick the shallowest unmarked node**

  - Use queue, new node go to the end



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

Queue G



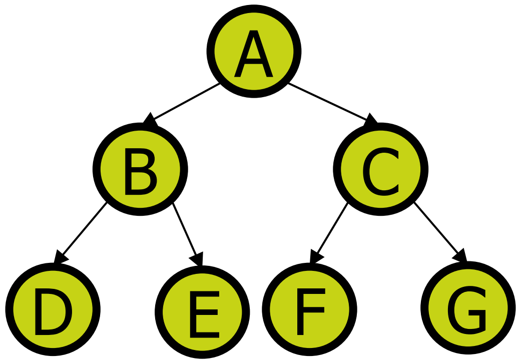
Pop one out, mark it,
put its child into the queue

Queue

Breadth First Search

- **Pick the shallowest unmarked node**

  - Use queue, new node go to the end



The queue is empty, Done!

Queue

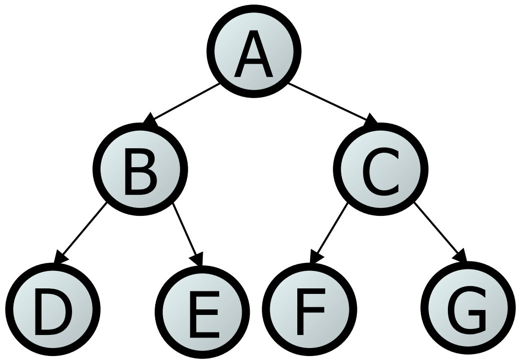
- The order of traversal: A B C D E F G

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

Depth First Search

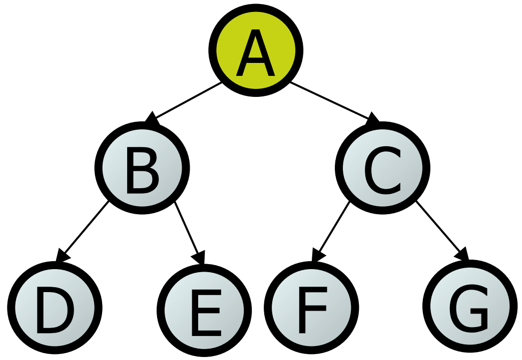
- **Pick the deepest unmarked node**

  - Use stack, new node go to the top



Start with the root in the stack

Stack	A



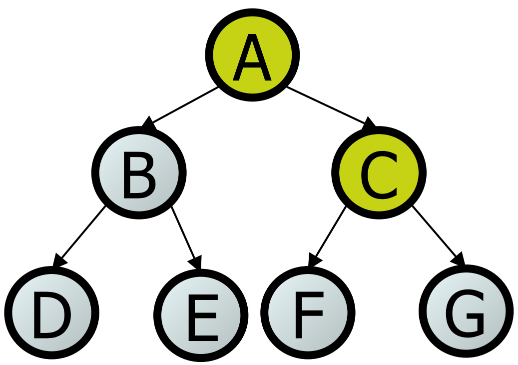
Pop one out, mark it,
put its child into the stack

Stack	B	C

Depth First Search

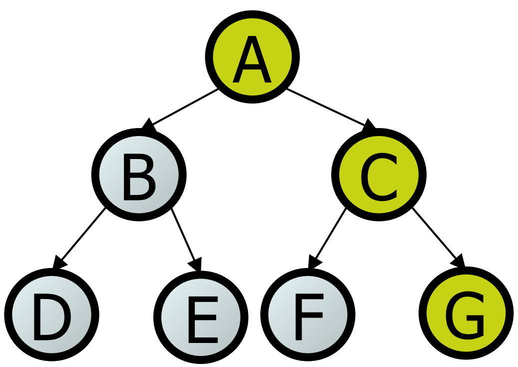
- **Pick the deepest unmarked node**

  - Use stack, new node go to the top



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

Stack	B	F	G



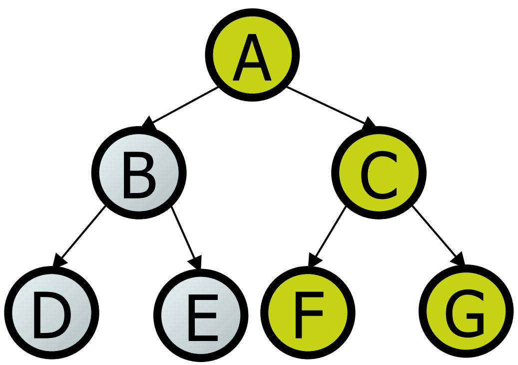
Pop one out, mark it,
put its child into the stack

Stack	B	F

Depth First Search

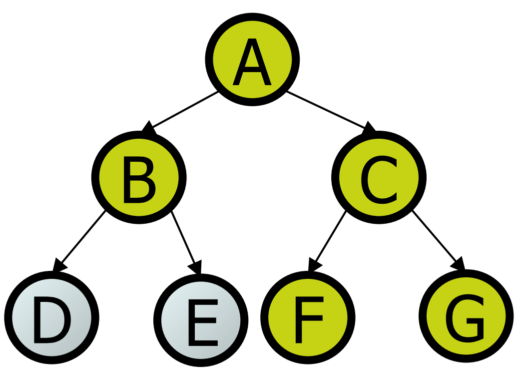
- **Pick the deepest unmarked node**

  - Use stack, new node go to the top



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

Stack	B



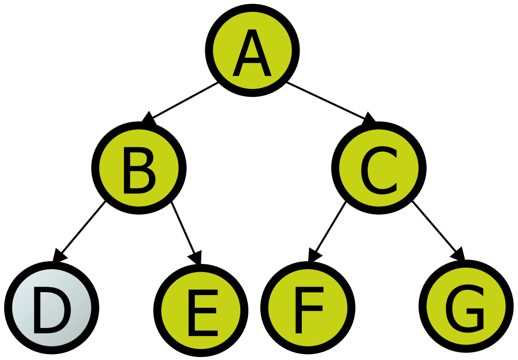
Pop one out, mark it,
put its child into the stack

Stack	D	E

Depth First Search

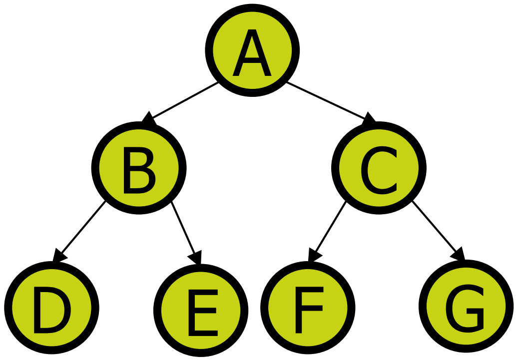
- **Pick the deepest unmarked node**

  - Use stack, new node go to the top



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

Stack	D



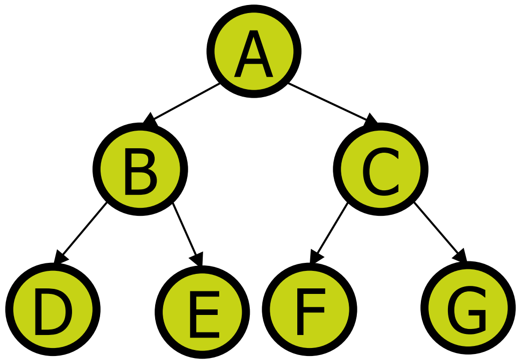
Pop one out, mark it,
put its child into the stack

Stack

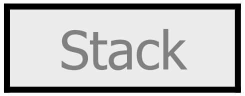
Depth First Search

- **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)**

Project 3

`Where are the people?`

Project 3 – Find Partner

- **Form 2 person team**

- Use discussion board to find partner

https://catalyst.uw.edu/gopost/area/swansond/127023

- Complete catalyst survey to form team
by Friday February 21st
(Only one survey per team!)

https://catalyst.uw.edu/webq/survey/kainby87/225338

- You still need complete survey even if you keep
your partner from project 2

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

Phase A (1–2)

Phase B (3–5)

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

  - Compare 5 versions with different queries