CS61B Lecture #33

gs: Graph Structures: *DSIJ*, Chapter 12

Why Graphs?

ng non-hierarchically related items

: pipelines, roads, assignment problems
ting processes: flow charts, Markov models
ting partial orderings: PERT charts, makefiles
seen, in representing connected structures as used in

Some Terminology

sists of

odes (aka *vertices*)

dges: pairs of nodes.

h an edge between are *adjacent*.

on problem, nodes or edges may have *labels* (or *weights*)

node set $V = \{v_0, \ldots\}$, and edge set E.

have an order (first, second), they are *directed edges,* a *directed graph (digraph),* otherwise an *undirected*

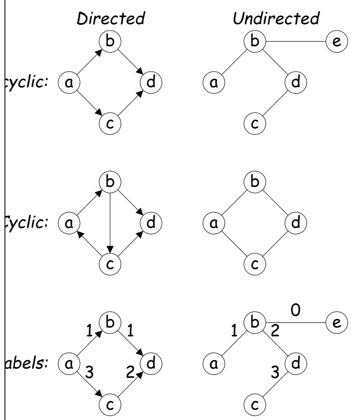
cident to their nodes.

es *exit* one node and *enter* the next.

path without repeated edges leading from a node back lowing arrows if directed).

clic if it has a cycle, else *acyclic*. Abbreviation: Directed h—*DAG*.

Some Pictures



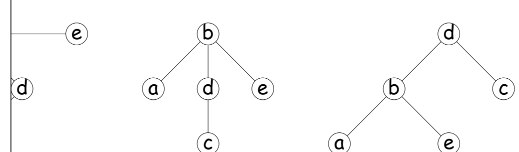
Trees are Graphs

onnected if there is a (possibly directed) path between nodes.

e node of the pair is *reachable* from the other.

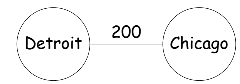
rooted) tree iff connected, and every node but the root one parent.

, acyclic, undirected graph is also called a *free tree.* ree to pick the root; e.g., all the following are the same

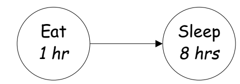


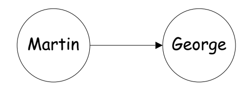
Examples of Use

cting road, with length.



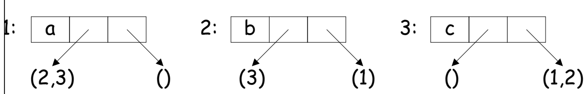
be completed before; Node label = time to complete.





Representation

- l to number the nodes, and use the numbers in edges.
- *presentation:* each node contains some kind of list (e.g., array) of its successors (and possibly predecessors).



- ollection of all edges. For graph above:

$$\{(1,2),(1,3),(2,3)\}$$

- *atrix:* Represent connection with matrix entry:

$$\begin{array}{cc} & \begin{array}{ccc} \mathit{1} & \mathit{2} & \mathit{3} \end{array} \\ \begin{array}{c} \mathit{1} \\ \mathit{2} \\ \mathit{3} \end{array} & \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \end{array}$$

ive Depth-First Traversal of a Graph

- ng and combinatorial problems using the "bread-crumb" in earlier lectures for a maze.
- nodes as we traverse them and don't traverse previously s.
- to talk about *preorder* and *postorder*, as for trees.

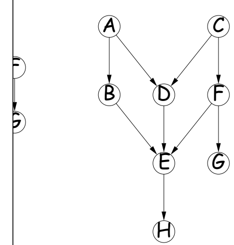
```
Traverse(Graph G, Node v)

nmarked) {


e(v, w) ∈ G)
se(G, w);
```

```
void postorderTraverse(Graph G, Node v)
{
  if (v is unmarked) {
    mark(v);
    for (Edge(v, w) ∈ G)
      traverse(G, w);
    visit v;
  }
}
```


Topological Sorting

- n a DAG, find a linear order of nodes consistent with
- r the nodes $v_0,\ v_1,\ \ldots$ such that v_k is never reachable $> k$.
- this. Also PERT charts.

aph (two views)

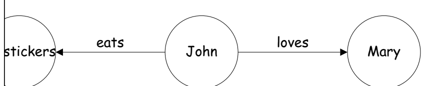


Possible Orderings

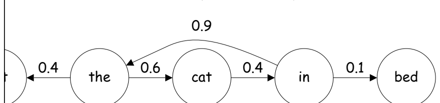
A	C	C
C	A	F
B	F	G
D	D	A
F	B	B
E	G	D
G	E	E
H	H	H

More Examples

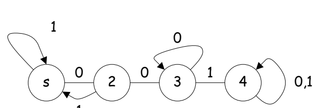
- relationship



- state might be (with probability)

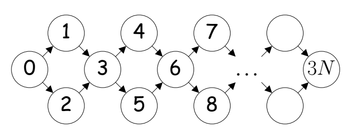


- state in state machine, label is triggering input. (Start state 4 means "there is a substring '001' somewhere in



Traversing a Graph

- hms on graphs depend on traversing all or some nodes.
- se recursion because of cycles.
- ic graphs, can get combinatorial explosions:



- he root and do recursive traversal down the two edges node: $\Theta(2^N)$ operations!
- try to visit each node constant # of times (e.g., once).

Depth-First Traversal of a Graph (II)

- n interested in traversing *all* nodes of a graph, not just able from one node.
- epeat the procedure as long as there are unmarked

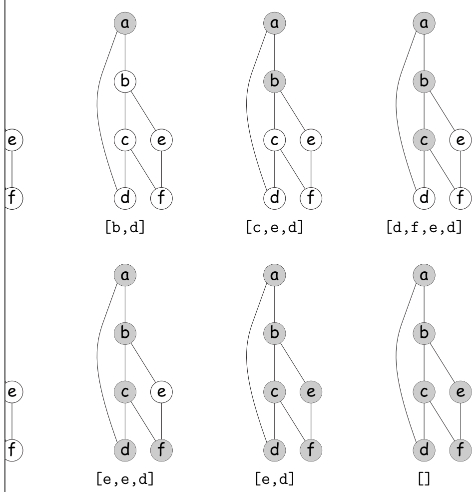
```
orderTraverse(Graph G) {
ll marks;
 ∈ nodes of G) {
eorderTraverse(G, v);


torderTraverse(Graph G) {
ll marks;
 ∈ nodes of G) {
storderTraverse(G, v);
```

eneral Graph Traversal Algorithm

```
OF_VERTICES fringe;

IAL_COLLECTION;
.isEmpty()) {
ringe.REMOVE_HIGHEST_PRIORITY_ITEM();

D(v)) {


dge(v,w) {
DS_PROCESSING(w))
 to fringe;
```

TION_OF_VERTICES, INITIAL_COLLECTION, etc. with xpressions, or methods to different graph algorithms.

epth-First Traversal Illustrated



ortest Paths: Dijkstra's Algorithm

a graph (directed or undirected) with non-negative ompute shortest paths from given source node, s, to

sum of weights along path is smallest.

le, keep estimated distance from s, ...

eceding node in shortest path from s.

```
Vertex> fringe;
v { v.dist() = ∞; v.back() = null; }

ty queue ordered by smallest .dist();
to fringe;
.isEmpty()) {
ringe.removeFirst();

e(v,w) {
() + weight(v,w) < w.dist())
() = v.dist() + weight(v,w); w.back() = v; }
```

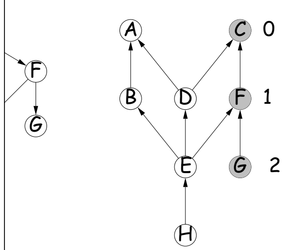

Sorting and Depth First Search

: Suppose we *reverse the links* on our graph.

ecursive DFS on the reverse graph, starting from node le, we will find all nodes that must come *before* H.

earch reaches a node in the reversed graph and there ssors, we know that it is safe to put that node first.

postorder traversal of the *reversed* graph visits nodes l predecessors have been visited.



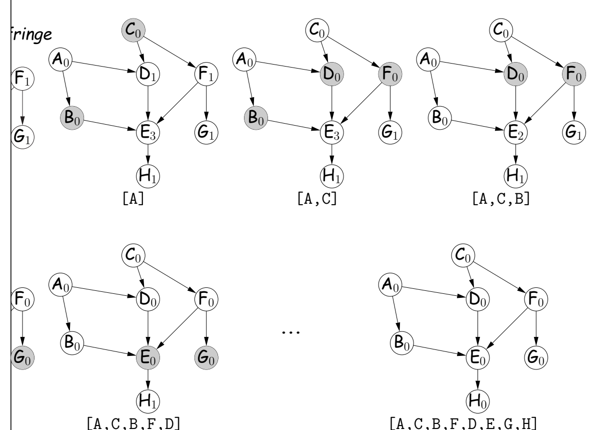
Numbers show post-order traversal order starting from G: everything that must come before G.

Example: Depth-First Traversal

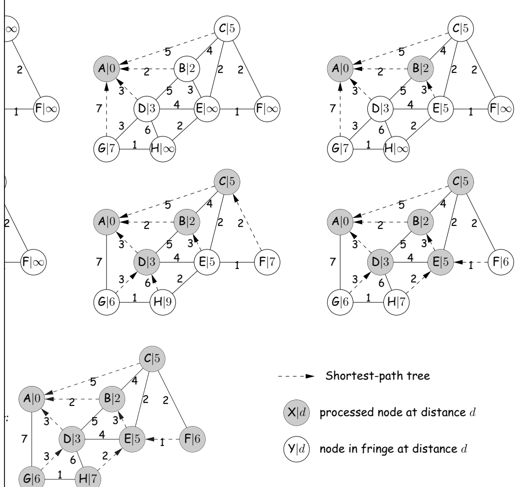
every node reachable from v once, visiting nodes further r.

```
ions are specializations of general algorithm
x> fringe;

ack containing {v};
nge.isEmpty()) {
= fringe.pop();

ed(v)) {
;
v);
n edge(v,w) {
marked(w))
nge.push(w);
```


Topological Sort in Action



Example


Shortest-path tree
processed node at distance d
node in fringe at distance d