CS211, LECTURE 23
INTRODUCTION TO GRAPHS

ANNOUNCEMENTS:

- A6 posted
- Prelim 2 tonight (OH 155: A-S; OH 165 T-Z)
- A7 TBA…
- office hours this week

OVERVIEW:

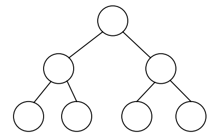
- motivation
- terminology
- DFS
- BFS
- algorithm for solution process

1. Motivation

- What happens if data structure links "cross over?
- think loops and circular linked lists
- informally, we have a ***graph***

1.1 The Gist

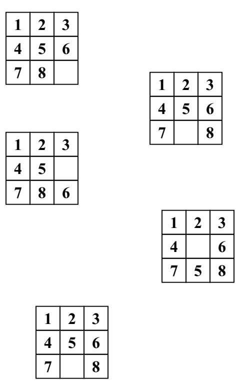
connect the nodes:



1.2 Applications

- "traveling salesman" and maps
- circuits
- structural models
- finite state machines
- and many more!

1.3 N-Puzzle Application



2. Graphs

2.1 Nodes

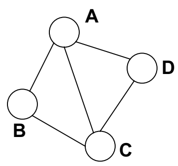
- data, items, points…
- the things/states/info that you want to connect
- also called ***verticies*** (set V)

2.2 Edges

- the lines between the points (set E)
- shows how and which points are connected
- can apply weights and direction

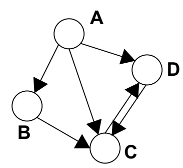
2.3 Graph

- set of edges, set of verticies
- $|V|$ = size of V, $|E|$ = size of E
- generalization of many other data structures!
- example:



2.4 Directed Graphs

- also called ***digraphs***
- G = (V,E)
- edges have 1 direction
- write edge as ordered pair (s,d) (source, destination) or s→d
- an edge may have node connect to itself (s==d)
- for 2-way direction, use another edge
- example:



Directed Graph G = (*V*,*E*)

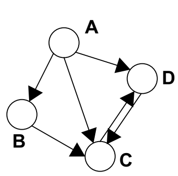
Vertices = *V* = {A,B,C,D}

Edges = *E* =
{(A,B),(B,C),(A,D),(A,C),(C,D),(D,C)}

Example: Edges (D,C) and (C,D) are different!

2.5 More Directed Graphs Terms

- ***adjacency***: for (a,b), b is adjacent to a because there is an edge connecting b to a (reverse is not true, because of directed graph)
- ***out-edges*** of node n: set of edges whose source is n
- ***out-degree*** of node n: number of out-edges of n
- ***in-edges*** of node n: set of edges whose destination is n
- ***in-degree*** of node n: number of in-edges of n



adjacency

out-edge

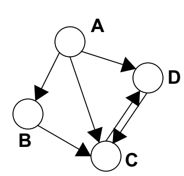
out-degree

in-edge

in-degree

2.6 Continuing Directed Graph Terms

- ***path***: sequence of edges in which destination node of an edge is source node of next edge in sequence; also, set of vertices that satisfy the same property
  ex) edge def: (A,B),(B,C),(C,D)
  ex) node def: A,B,C,D
- ***length of path***: number of edges in path or sum of weights on path (see Weight
- ***source of path***: source of first edge on path
- ***destination of path***: destination of last edge on path
- ***reachability***: nodes n is reachable from node m is there is a path from m to n (might have many paths between nodes)
- ***simple path***: a path in which every node is the source and destination of at most two edges on the path (*path does not cross vertex more than once*)



2.7 Cycles

- ***cycle***: a simple path whose source and destination nodes are the same
- length of cycle: length of path (depends on choice of nodes or edges for description)
- loop: path (a,b),(b,a) (edges) or (a,a) (nodes)

3. More Graph Types/Qualities

3.1 Undirected Graphs

- edges have no arrows, so use set for edges: {a,b}
- can go any direction on edge
- nodes cannot form loops ({a,a} becomes just {a})

3.2 Directed Acyclic Graphs

- also called DAGs
- digraph with no cycles
- note: trees are DAGs (but not vice versa)

3.3 Connected Graphs

- a graph with path between every pair of distinct verticies
- disconnected graph includes "lone wolf" nodes (no edges)

3.4 Complete Graphs

- edge between every pair of distinct vertex

3.5 Labeled Graphs

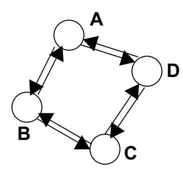
- attach additional info to nodes and/or edges
- ***weights***/***costs***: values on edges (best/worst edges)
  - edge ex) choosing shortest/quickest/best roads to take to get between towns
  - node ex) importance of reaching certain towns ("fun quotient")
- also called ***weighted graphs***

3.6 Trees?

- yes, directed acyclic graphs
- see Tree notes for pretty much the same definitions of vertex and edge

3.7 Sparse and Dense Graphs

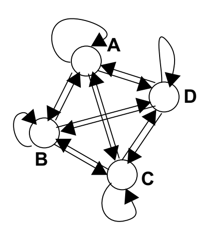
- sparse: not many edges
  - $|E| = O(|V|)$
  - ex) graph with same number of edges emanating from nodes has $|E| = k|V|$, so $|E| = O(|V|)$



$$|V| = 4$$

$$|E| = \left(2\frac{edges}{node}\right)(4nodes) = 8edges$$

- dense: many edges
  - $|E|$ essentially on the order of $|V|^2$
  - see pg. 546 DS&A (Def 16.6) for more precision



$$|V| = 4$$
$$|E| = 16$$

4. Representations

4.1 Implicit

- rules/model creates a network of nodes/edges
- ex) puzzle moves
  - each move makes a new puzzle
  - treat each state as a node
  - so, rules implicit define a graph
- common for games!

4.2 Explicit

- define all nodes V and edges E ahead of time
- want system to represent edges
- why? it's the "biggest problem":
  - $G = (V,E)$ and each edge e in E is a pair (v1,v2)
  - most edges possible? $|V|$^2
    (form pairs from all nodes)
  - most sets of edges possible? 2^($|V|$^2)
- so, use container to represent edges
  - adjacency matrix
  - adjacency list

4.3 Adjacency Matrix

- ***adjacency matrix***

$$A_{ij} = \begin{cases} w_{ij} & \{v_i, v_j\} \in E \\ 0 & otherwise \end{cases}$$

- terms

  v_i : node i; v_j node j

  $\{v_i, v_j\} \in E$: edge between nodes i (v_i) and j (v_j)
  belongs to set of edges E

  w_{ij} : weight of edge between nodes i and j

- A_{ij} : the matrix (rectangular 2x2 array) as rows (i) and cols (j); coords correspond to nodes i and j

4.4 Adjacency List

- adjacency list: linked list of nodes adjacent to a node
- need $|V|$ lists

4.5 graph types to develop:

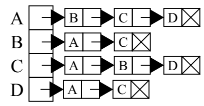
- undirected
- directed
- weighted

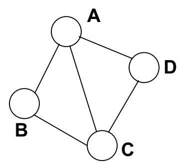
4.6 Undirected

$$A_{ij} = \begin{cases} 1 & \{v_i, v_j\} \in E \\ 0 & otherwise \end{cases}$$

Use array A of lists:
A_i stores a linked list of nodes
no edge implied by order *in* list
nodes must be adjacent to A_i

i \ j	A	B	C	D
A		1	1	1
B	1		1	
C	1	1		1
D	1		1	



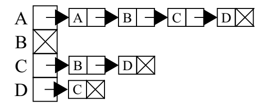


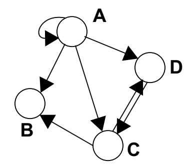
4.7 Directed

$$A_{ij} = \begin{cases} 1 & (v_i, v_j) \in E \\ 0 & otherwise \end{cases}$$

Use array A of lists:
A_i stores a linked list of nodes
no edge implied by order *in* list
nodes must be adjacent to A_i

i \ j	A	B	C	D
A	1	1	1	1
B				
C		1		1
D			1	





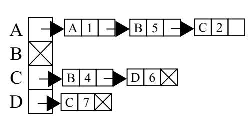
4.8 Weighted

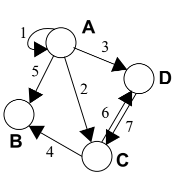
- assuming also weighted
- w_{ij} : cost or weight of edge from node i to node j

$$A_{ij} = \begin{cases} w_{ij} & (v_i, v_j) \in E \\ 0 & otherwise \end{cases}$$

Use array A of lists: include weights
List for i contains j,w for edge (i,j)

i \ j	A	B	C	D
A	1	5	2	3
B				
C		4		6
D			7	





4.9 Choice of AM or AL?

- Adjacency Matrix
  - uses $O(|V^2|)$ space
  - can answer "is there an edge from i to j?" in $O(1)$ time
  - enumerating all nodes adjacent to i: $O(|V|)$ (find all nodes j in row for i)
  - could be sparse because of wasted space (0s)
  - better for dense graphs (lots of edges)!
- Adjacency List
  - uses O(|V|+|E|) space (|V| for i nodes, |E| for j nodes emanating from each i node)
  - can answer question "is there an edge from i to j?" in $O(|E|)$ time
  - enumerating all nodes adjacent to i: $O(1)$ per adjacent node in linked list
  - better for sparse graphs (few edges)!

5. Interesting Problems

5.1 Paths

- find ways to reach/find/collect/organize information from network of nodes
- focus of a lot of research!

5.2 Reachability

- is there a path from a given node to another node?
- ex) find the solved state of N-Puzzle from scrambled state

5.3 Minimal Path

- find the shortest path from a node to another
- find the shortest path from every node to another
- use weights to find min/max distances

5.4 Cycles

- ex) Traveling Salesman problem
- find the smallest length cucle that passes through all nodes
- no one knows if there is an efficient algorithm for this (NP/NP-complete problems)

6. Exercises

- Show all edges and verticies of a 2x2 N-Puzzle.
- Demonstrate a scenario/game/model that forms an implicit graph.
- Demonstrate why we use edges for explicit representations of graphs. (Section 4.2)