CS1800
11/21 - Fri :)

Admin

- HW6 graded!
- HW7 due 12/2 9pm
- optional lecture 11/25
  - ↳ poll!
- second chance HW
  - ↳ due 12/5 9pm
- Quiz 4 12/5 in class
  - ↳ 30-min + buffer
- Final exam
  - ↳ 12/12 8-10AM } Blackman
    12/13 1-3pm }

Agenda

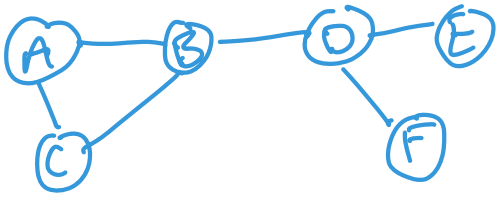
1. Graphs overview
2. Graph traversal

1. Graph Overview

↳ Discrete structures: sets, sequences, graphs

- graphs rep complex ideas, relationships
- graphs often represent maps
  - every map can be a graph
  - not every graph is a map

ex



$G = (V, E)$

- vertices (set): A, B, C, D, E, F
- edges (set): defined by endpoints

  $(A, B), (D, E), (F, D) \ldots$

ex as a map:

- vertices: intersections
- edges: streets

Properties of this graph:

- simple
- undirected
- unweighted
- cyclic
- degree(A) = 2
- degree(G) = 12
- # vertices = 6
- # edges = 6
- connected

- graph can have a vertex without edges

- can't have an edge without vertices

Simple

- no self loops

- NO double edges

Undirected

- edge (A, B) means we can go from A to B and B to A

- A has B as a neighbor
- B has A as a neighbor

Directed

- edge (A, B) exists
- edge (B, A) does not exist
- A has B as a neighbor
- B does not have A as a neighbor

Unweighted

- no values assoc. with edges
- edges exist, or not

Weighted

- edges have values

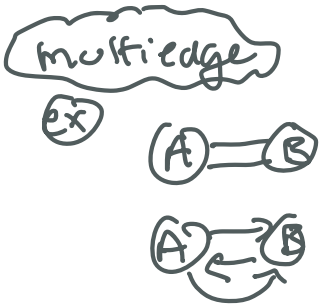
- rep time, cost, distance, etc.

Cyclic

- path: sequence of vertices connected by edges

- cycle: path that starts/ends at same vertex

multiedge



DAG → directed, acyclic graph

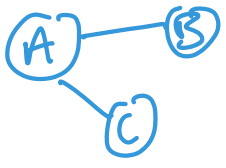
↳ known problems can be solved :)

Connected

- path exists between any 2 vertices

degrees

- deg(v) # edges incident on vertex v



deg(A) = 2
deg(B) = 1
deg(C) = 1

- total degree of a graph is $2 \cdot |E|$

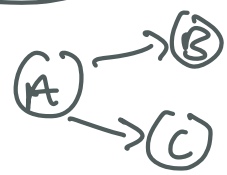
Strongly connected

- same idea, but graph is directed

- useful to isolate strongly connected components

degree of directed graph

- in degree
- out degree



2. Graph Traversal

↳ visit every vertex

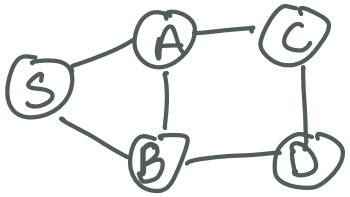
math: $G = (V, E)$ sets

prog. implementation:

- adjacency list *
- adjacency matrix

Adjacency list:

- set of vertices
- each vertex: linked list of neighbors
- neighbors: connected by one edge
- undirected graph: edges are repeated!



S: A, B ✓

A: S, B, C ✓

B: S, D, A

C: A, D

D: B, C

Breadth First Search (BFS)

- finds every vertex reachable from a starting vertex (S)
- visit S's neighbors (distance = 1)
- visit neighbors' neighbors (distance = 2)
- . . .
- track distance from S (v.d)
- track path from S (v.π)
- break ties in alpha order

	S	A	B	C	D
v.d	0	1	1	2	2
v.π	/	S	S	A	B

- A, B in one step — S, A; S, B
- C, D in two steps — S, A, C; S, B, D

BFS pros :)

- run-time is linear
- if unweighted, finds shortest path to every reachable vertex

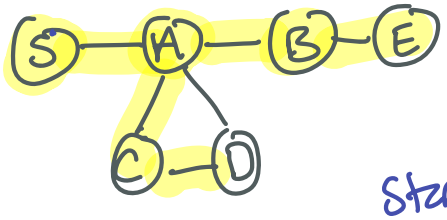
optimization !

Traversal #2 Depth First Search (DFS)

- find all vertices reachable from starting vertex (S)
- not ⇒ shortest path

DFS Approach

- adj list
- start at S, visit new vertices as deep as possible
- backtrack when nothing new to visit
- track start, finish times



	S	A	B	C	D	E
start	0	1	2	6	7	3
finish	11	10	5	9	8	4

What do we get from DFS?

- order by finish time: topological sort
- detect a cycle
- DFS tree (simplified graph)
- detect strongly connected components