11.1 graphs

Tuesday, November 15, 2022

TODAY:

- hash table review
- graphs

HASH TABLE REVIEW

- use large array so that accessing an item is $O(1)$
- waste space to gain time
- hash function maps each key to an array index

factor $= \frac{\text{size}}{\text{capacity}}$ measures how full the hash table is

REMEMBER to convert one of these to a float when dividing (or this will cause a bug!)

Q: List the 4 implementations we've seen for the Dictionary ADT:

fastest

hash tables	$O(1)$	* amortized, assuming the hash function is good
AVL trees	$O(\log_2 n)$	
BST tree	$O(\text{height})$	
linear dictionaries	$O(n)$	

slowest

Q: What are some reasons NOT to use a hash table?

- You know that you need to save memory
- If you need to frequently call getItems() or getKeys()
- If your use case is simple or you know beforehand that the data set is small
- If you don't have a good hash function for the key type used
- hash tables don't order the keys so if we want to for example find the smallest or largest key, a BST or AVL tree is much easier

GRAPHS

A way to represent relationships between entities

$G = (V, E)$ where:

V = set of vertices

$G = (V, E)$ where:

V = set of vertices

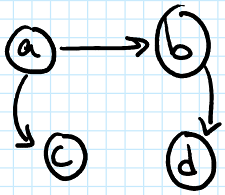
E = set of edges

Graphs can be directed or undirected.

Edges can be weighted or unweighted.

Edges can also have labels.

example



directed

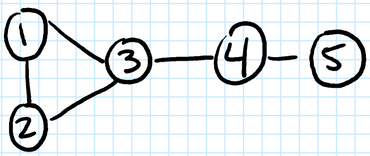
unweighted

$V = \{a, b, c, d\}$

$E = \{ (a, c), (a, b), (b, d) \}$

source   destination

example



undirected

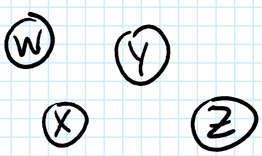
unweighted

$V = \{1, 2, 3, 4, 5\}$ $\qquad |V| = n = 5$

$E = \{ (1,2), (1,3), (2,3), (3,4), (4,5) \}$

$|E| = m = 5$

example



This is a valid graph.

$V = \{w, x, y, z\}$ $\qquad |V| = n = 4$

$E = \{ \}$ $\qquad |E| = m = 0$

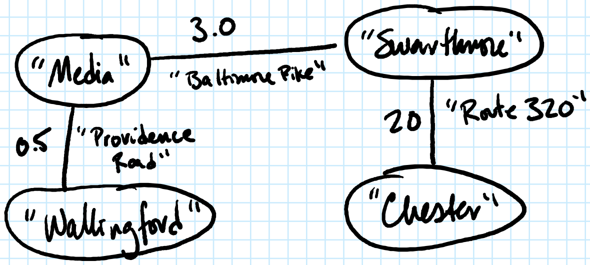
When measuring running time of an algorithm with a graph as input

$n = |V|$ = number of vertices

$m = |E|$ = number of edges

APPLICATIONS OF GRAPHS

example maps



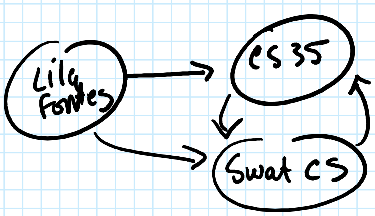
vertices: locations/towns (V is strings)
edges: roads
weights: distances (weights are floats)
labels: road names (E is strings)

example social networks (like Instagram, Tiktok, Facebook)

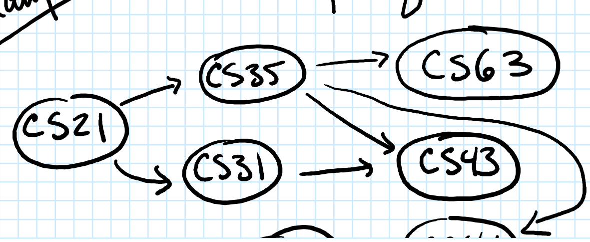
vertices: profiles/users
edges: represents "following" or "is friends with"
weights? time duration of following or some count of mutual followers or # of times checked per day (creepy!)
directed / undirected? directed because X can follow Y but Y might not follow X

example the world wide web

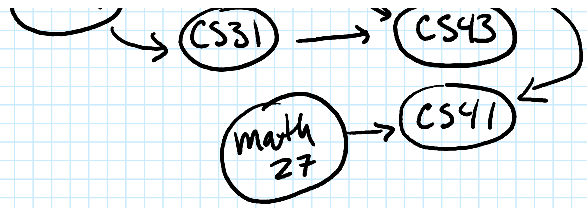
vertices: web pages/URLs
edges: hyperlinks
directed/undirected?



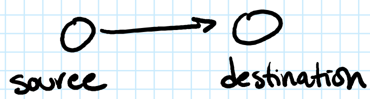
example CS class prerequisites



vertices: CS courses
edges: represent the "prerequisite" relationship
directed/undirected? directed

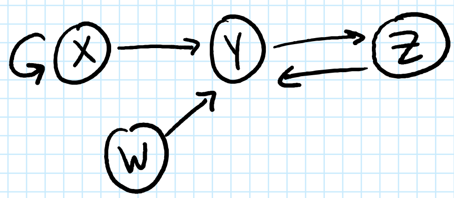


GRAPH VOCABULARY



Two vertices are NEIGHBORS if they are directly connected by an edge.

in-degree of vertex v: number of edges with v as destination
out-degree of vertex v: number of edges with v as source
degree of vertex v : the number of neighbors of v.



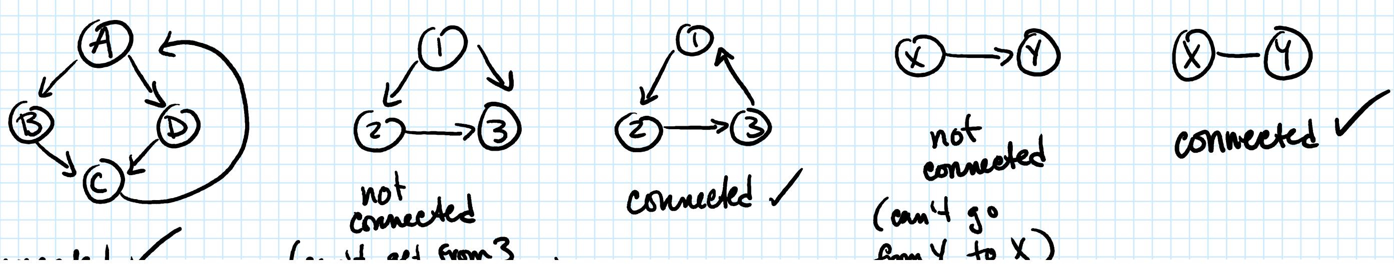
in-degree (W) = 0
out-degree (W) = 1
in-degree (X) = 1
out-degree (X) = 2
in-degree (Y) = 3
out-degree (Y) = 1
in-degree (Z) = 1
out-degree (Z) = 1

Note: degree (Z) = 1 as Z has only one neighbor vertex

A PATH is a sequence of edges where the destination of each edge is the source of the next one.

example path: $X \to Y \to Z$ has edges (X,Y), (Y,Z)

Two vertices are CONNECTED if there is a path between them.
A graph is CONNECTED if every pair of vertices is connected.



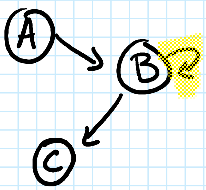
connected ✓

not connected (can't get from 3 to any other vertex)

connected ✓

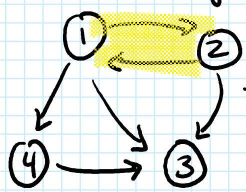
(can't go from Y to X)

A graph is SIMPLE if it has no self loops and no duplicate edges.



simple?

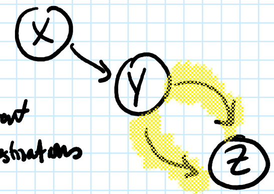
no, B has a loop to itself



these edges are not duplicates, they have different sources + destinations

simple?

yes



simple?

no, duplicate edge

Q: Is a tree a graph? Yes, a tree is a special type of graph.

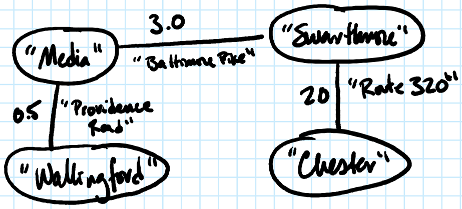
GRAPH ADT

templated on 3 things:
- V vertex label type (must be unique)
- E edge label type (can have duplicates)
- W edge weight type (usually numerical)

methods:

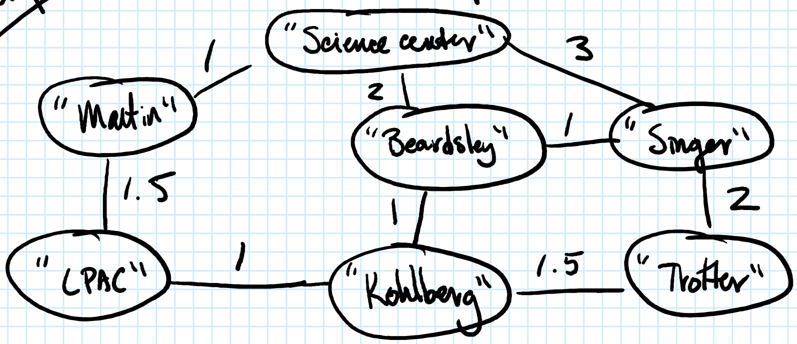
```
void insertVertex(V vertex)

void removeVertex(V vertex)

void insertEdge(V source, V destination, E label, W weight)

void removeEdge(V source, V destination)

vector<Edge<V,E,W>> getEdges()

vector<V> getVertices()

bool containsVertex(V vertex)

bool containsEdge(V source, V destination)

Edge<V,E,W> getEdge(V source, V destination)

vector<Edge<V,E,W>> getOutgoing(V vertex)

vector<Edge<V,E,W>> getIncoming(V vertex)

vector<V> getNeighbors(V vertex)
```

previous example:



V is type string
E is type string
W is type float

example: Swarthmore north campus



Vertices = { "Science center", "Martin", "Beardsley", "Singer", "LPAC", "Kohlberg", "Trotter" }

Edges = { ("Martin", "LPAC", 1.5), ("Science center", "Beardsley", 2), ... }

V type: strings
E type: (not used in this example)
W type: float

Q: What is the in-degree("Beardsley")?
Q: What is the out-degree("LPAC")?
Q: What is a path from "Martin" to "Trotter"?

APPLICATIONS OF GRAPHS

We will use graphs to answer many sorts of questions:

- What's the shortest path between two vertices?
- What's the least expensive path between two vertices?
- Is it possible to get from one vertex to another specific vertex?
- Is it possible to reach every other vertex from a specific starting vertex?

- Is it possible to reach every other vertex from a specific starting vertex?
- Is it possible to reach every vertex from every other vertex?