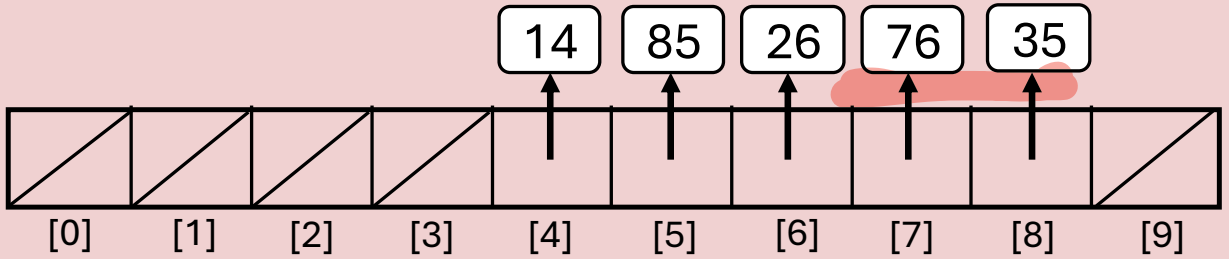
Poll Everywhere

PollEv.com/2110fa25 text 2110fa25 to 22333

Suppose we add 14, 26, 35, 76, 85 (in some order) to a probing hash table (where their hash codes are their values). The state of the hash table is:



Performance of probing hash tables depends on insertion order.

Which could *not* be the order these elements were added?

14, 85, 26, 76, 35	(A)	85, 26, 76, 35, 14	(C)
14, 26, 85, 35, 76	(B)	26, 14, 76, 85, 35	(D)

Exam Reminders

Prelim 2 is Tonight!

Early Exam: 5:30-7:00, Statler Hall 265
Main Exam: 7:30-9:00, Statler Hall 185

Bring your **Cornell ID Card** and a couple **writing utensils** (pencils, erasers, pens)
Exam is closed-book (with provided reference sheet)

More information and review materials linked on website / Ed

No OHs tomorrow because of exam grading.

You've learned a lot since Prelim 1! Time to show it off!

Lecture 21: Graphs

CS 2110

November 6, 2025

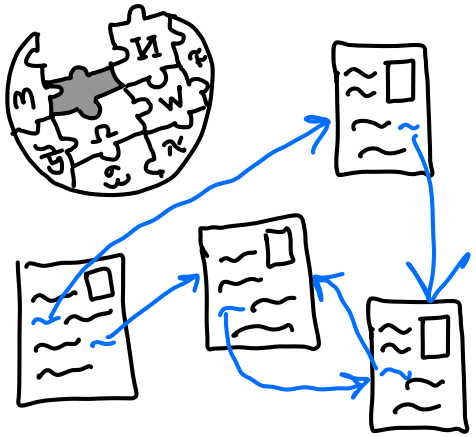
Today's Learning Outcomes

88. Translate between a formal description (list of vertices, edges) and an illustration of a (weighted) graph.

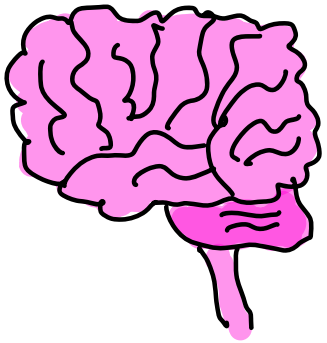
89. Identify substructures in graphs such as neighborhoods, paths, and cycles both visually and programmatically.

90. Describe adjacency list and matrix representations of a graph and compare the space/time complexities of operations on each of these representations.

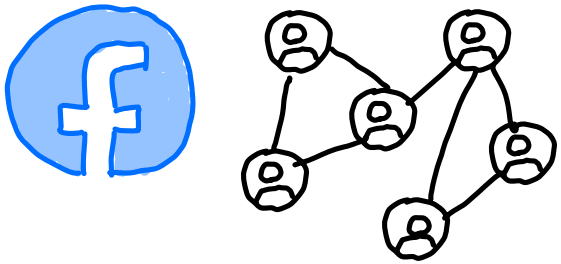
Modeling Structure in Real-World Settings



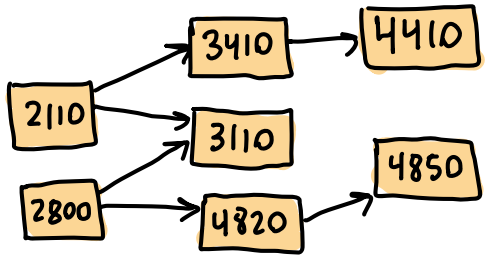
Wikipedia Links



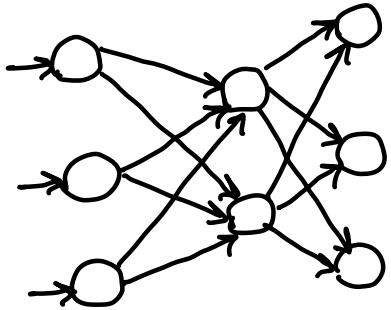
Neural Networks

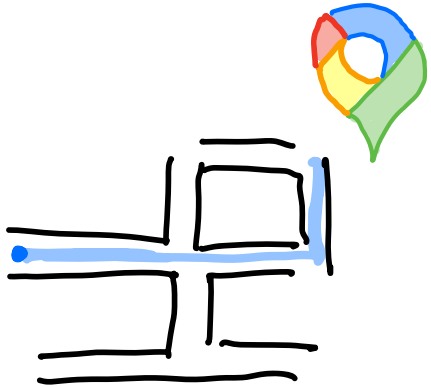


Social Networks



Course Planning





Route Planning

Directed Graphs

A directed graph is a structure described by two sets.

Vertices (V): units we're connecting

(people, addresses, cities, webpages, ...)

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

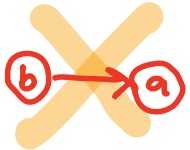
$E = \{(a,b), (a,d), (b,c), (b,e), (c,b), (d,c), (d,e), (e,a)\}$

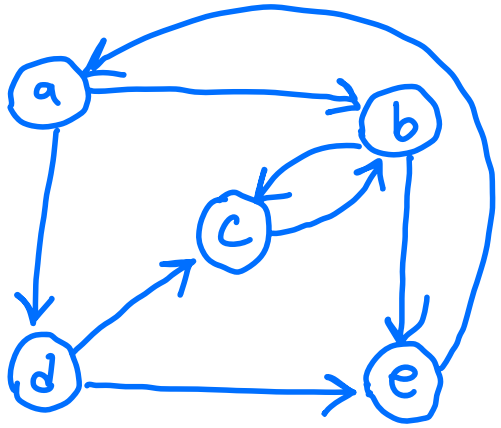
Edges (E): directed connections between pairs of vertices

(friendships, roads, flights, prereqs)

written as ordered pairs

(a,b) = 

not 



Other Graph Varieties

Undirected graphs: All edges connect in both directions

Self-loops: Edges that start/end at same vertex

Multigraphs: Have multiple parallel edges between the same pair of vertices (in same order)

Simple graphs don't have self-loops or parallel edges.

* We'll focus on simple directed graphs in CS 2110.

Poll Everywhere

PollEv.com/2110fa25 text 2110fa25 to 22333

What is the maximum number of edges that a *simple* directed graph with 6 vertices can have?

6 vertices can be tail * 5 vertices can be head = 30

More generally, what is the maximum possible number of edges that a *simple* directed graph with $|V|$ vertices can have?

$|V| \cdot (|V|-1) = O(|V|^2)$

Key Idea: $|E|$ falls between

$|V| \leq |E| \leq |V|^2$

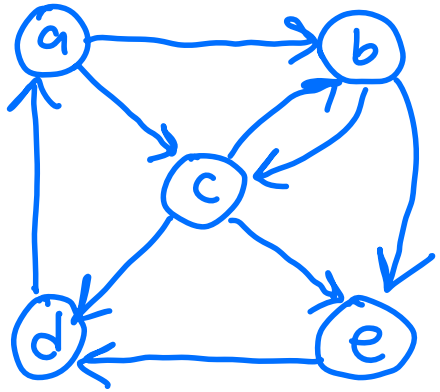
↖ if graph connected

Neighborhoods in Graphs

If $(u,v) \in E$, we say v is a (n out-) neighbor of u.

u's neighborhood includes all of u's neighbors. $N_G(u)$

u's degree is the size of its neighborhood. $d_G(u)$



$N_G(c) = \{b, d, e\}$

$d_G(c) = 3$

Paths and Cycles

A path is a sequence of distinct contiguous edges in a graph.

$p = ((v_0, v_1), (v_1, v_2), \ldots, (v_{\ell-1}, v_\ell))$ with $(v_0, v_1) \in E$, $(v_1, v_2) \in E$, $(v_{\ell-1}, v_\ell) \in E$

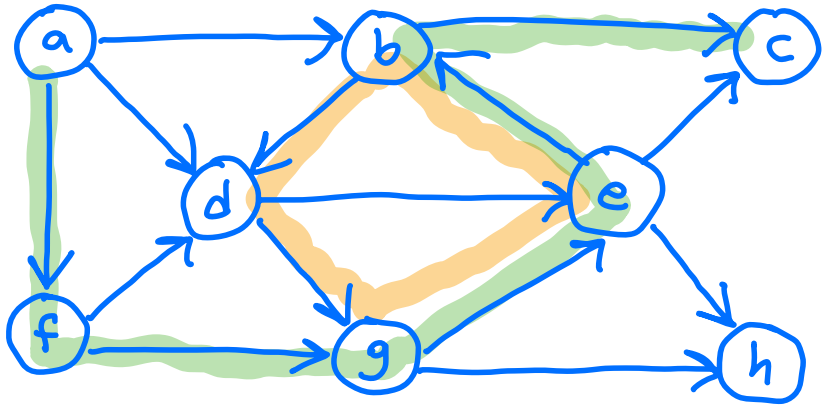
source; same vertex; destination

length of path = # of edges (ℓ)

shorthand: $v_0 \to v_1 \to v_2 \to \ldots \to v_\ell$

a $v_0 \rightsquigarrow v_\ell$ path

A cycle is a path whose source = its destination



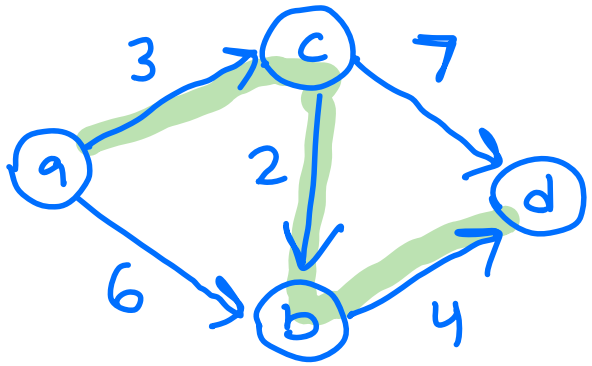
Path $a \to f \to g \to e \to b \to c$ has length 5

Cycle $b \to d \to g \to e$

Labeled Graphs

Sometimes, we label vertices/edges with extra info

Common: Label edges with number
- cost, weight, length



shortest $a \rightsquigarrow d$ path

$a \to c \to b \to d$ has length 9

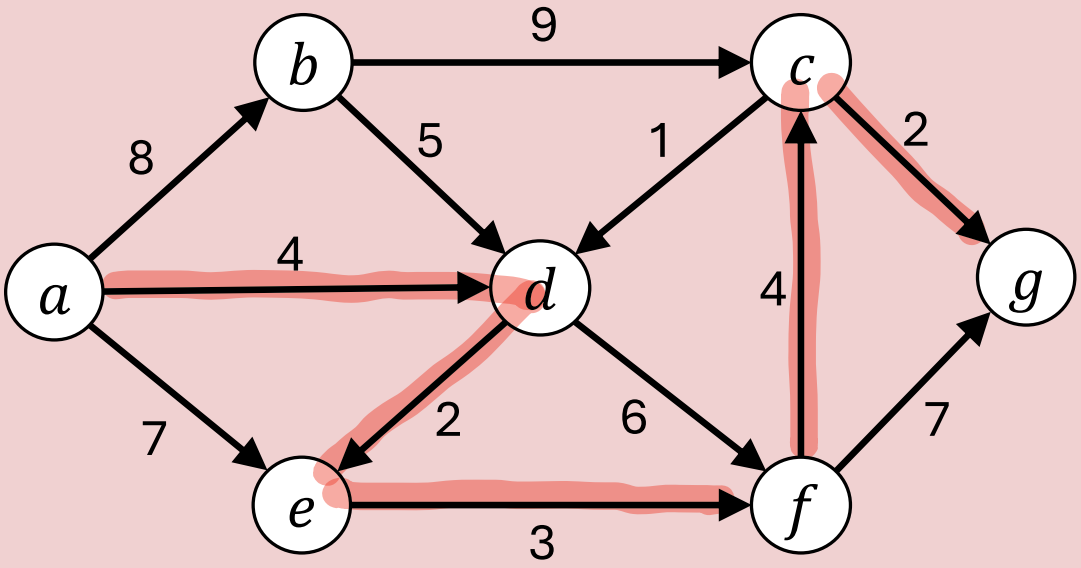
Length of paths overloaded term that refers to sum of edge labels along path

Shortest path has minimum label sum, not necessarily fewest edges.

Poll Everywhere

PollEv.com/2110fa25 text 2110fa25 to 22333

What is the length of the shortest $a \rightsquigarrow g$ path in this graph?



15	(A)
16	(B)
17	(C)
18	(D)

A Graph ADT

Brainstorm: What operations should a Graph support?

Query: Access info

- check for vertex
- check for edge
- get # vertices
- get # edges
- get edge weight
- get degree
- check for path/cycle

Mutate: Change

- add vertex
- add edge
- remove vertex
- remove edge
- update weight

Iterate

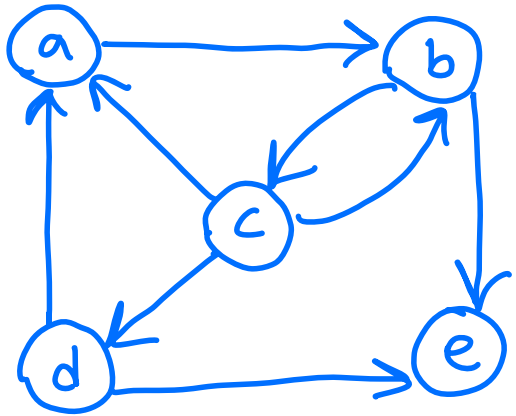
- over all vertices
- over all edges
- over neighbors of one vertex
- over edges in a path

No built-in Graph ADT...
we'll need to build ourselves

Coding Demo: Graph ADTs

Adjacency Matrix Representation

Keep track of edges using 2D array/list structure



tail

head

	a	b	c	d	e
a		✓			
b			✓		✓
c	✓	✓		✓	
d	✓				✓
e					

can be booleans to indicate presence/absence or Edge variables with references/null

Coding Demo: AdjMatrixGraph

Adjacency Matrix Operation Complexities

```
class AdjMatrixGraph implements Graph<...> {
   record AdjMatrixEdge(...) implements Edge<...> { }

   class AdjMatrixVertex implements Vertex<...> {
      private String label;
      private int index;
   }

   private HashMap<String, AdjMatrixVertex> vertices;
   private ArrayList<ArrayList<AdjMatrixEdge>> edges;

   // methods
}
```

Iterate over neighbors: $O(|V|)$

Count vertices: $O(1)$ - vertices.size()

Count edges: $O(|V|^2)$

Check for an edge: $O(1)$ using vertices and index

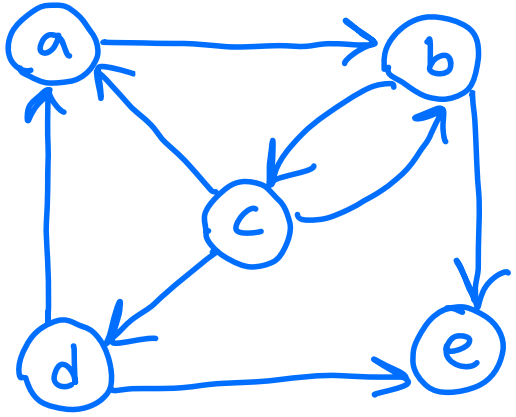
Add vertex: $O(|V|)$ amortized

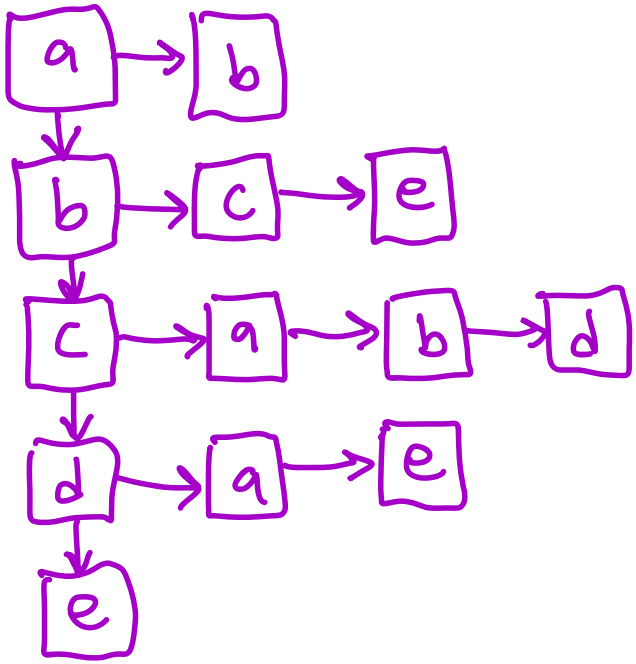
Add edge: $O(1)$

Iterate over all edges: $O(|V|^2)$

Adjacency List Representation (Classical)

Maintain a list of all vertices, and have each vertex maintain a list of all its neighbors





Adjacency List Representation (Improved)

Iterating over nested list structure is slow

- linear scan over vertices to find tail's neighbors list
- linear scan over neighbor list to search for edge

To improve efficiency: Replace both layers of lists with hash maps keyed on vertex labels

O(|V|) scan → expected O(1) lookup

Coding Demo: `AdjListGraph`

Adjacency List Operation Complexities

```
class AdjListGraph implements Graph<...> {
   record AdjListEdge(...) implements Edge<...> { }

   static class AdjListVertex implements Vertex<...> {
      String label;
      HashMap<String, AdjListEdge> outEdges;
   }

   private HashMap<String, AdjListVertex> vertices;

   // methods
}
```

Iterate over neighbors: $O(\text{degree}) = O(|V|)$

Count vertices: $O(1)$ using `vertices.size()`

Count edges: $O(|V|)$ using `outEdges.size()`

Check for an edge: $O(1)$ using map operations

Add vertex: $O(1)$

Add edge: $O(1)$

Iterate over all edges: $O(|V| + |E|)$

Comparing Representations

Representation	Memory Usage	Time to Check for an Edge	Time to Iterate over Neighborhood	Time to Iterate over all Edges
Adj Matrix	$O(\|V\|^2)$	$O(1)$	$O(\|V\|)$	$O(\|V\|^2)$
Adj List (Linked)	$O(\|V\| + \|E\|)$	$O(\|V\|)$	$O(\|V\|)$	$O(\|V\| + \|E\|)$
Adj List (Map)	$O(\|V\| + \|E\|)$	$O(1)$	$O(\|V\|)$	$O(\|V\| + \|E\|)$

dense graphs have $|E| \approx O(|V|^2)$: AM + AL have similar performance

memory layout likely makes AM ops faster

sparse graphs have $|E| \approx O(|V|)$: AL has much better performance

↖ most real-world graphs