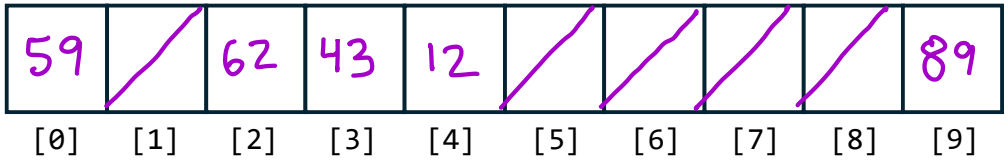
contains() in Probing Hash Tables



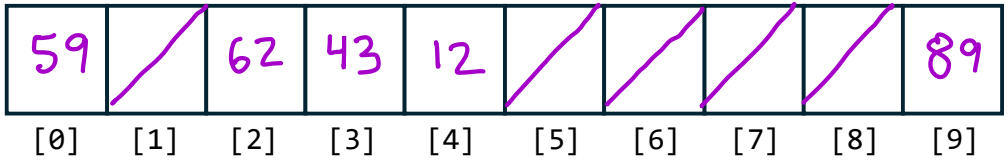
Always start at nominal index:

Best case: Nominal bucket contains element contains(43) == true

Nominal bucket is empty contains(16) == false

If nominal bucket contains different element,

remove() in Probing Hash Tables



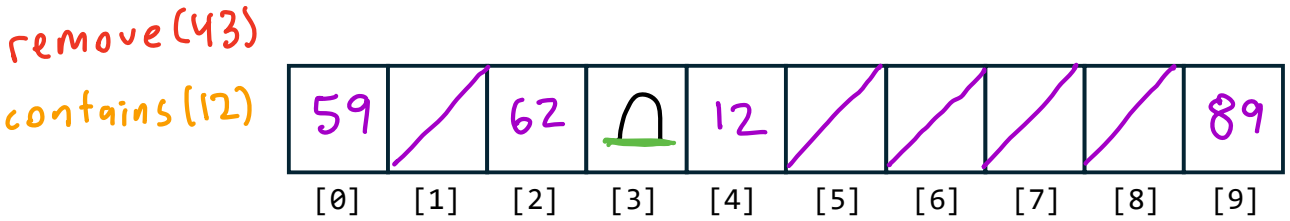
Just like other methods, start at nominal index and probe for element.

Incorporating Tombstones

When we add tombstones to our hash table in the `remove()` method, we'll need to account for them in our `add()` and `contains()` logic.

`add()`: Probe until we reach empty bucket or a tombstone (which we overwrite). Remove tombstones during resize

`contains()`: Probe past tombstones, since only empty buckets signal the end of a probing sequence.



Probing Hash Table Performance

The expected runtime of all operations in a probing hash table is also upper bounded by a function of λ, so it also amortizes to $O(1)$ with proper resizing.

Comparing Chaining vs. Probing:

- Probing has worse dependence on λ since nearby hash values can affect performance, not just exact collisions.
- Probing hash tables have smaller, centralized storage, so they can perform better due to hardware considerations like *cache locality* that fall outside of our scope.



Lecture 21: Graphs

CS 2110

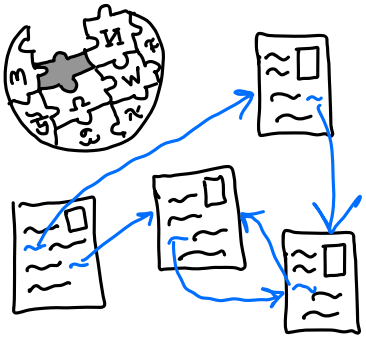
November 5, 2026

Today's Learning Outcomes

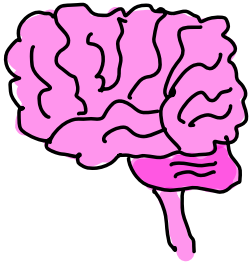
80. Identify substructures in graphs such as neighborhoods, paths, and cycles both visually and programmatically.

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

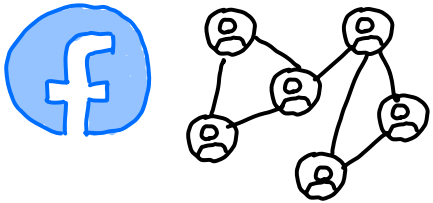
Modeling Structure in Real-World Settings



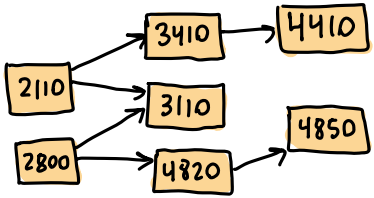
Wikipedia Links



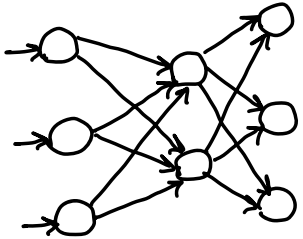
Neural Networks



Social Networks



Course Planning





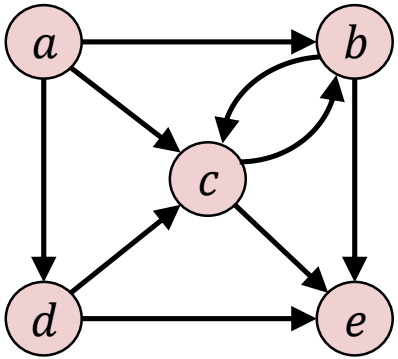
Route Planning

Directed Graphs

A directed graph is a mathematical structure described by two sets.

Vertices (V)**:**

Edges (E)**:**



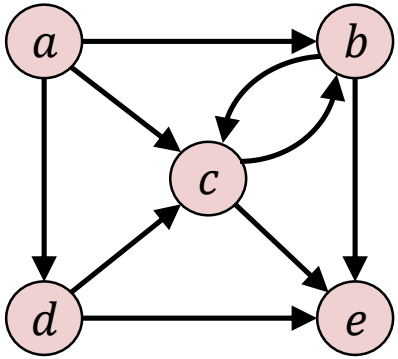
Other Graph Varieties

Undirected graphs:

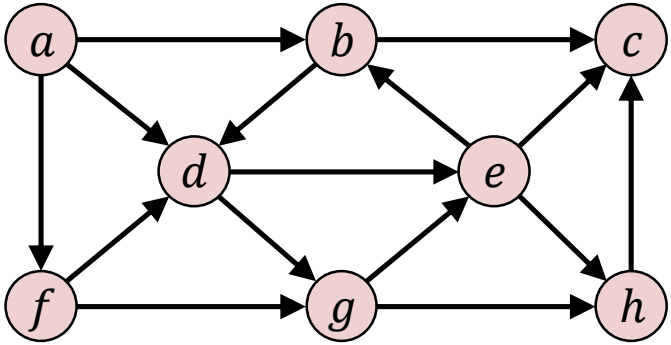
Graphs with self-loops:

Multigraphs:

Neighborhoods in Graphs

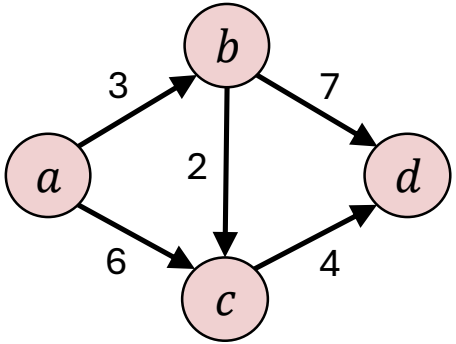


Paths and Cycles



Labeled/Weighted Graphs

Sometimes, we *label* the vertices or edges in a graph with extra info.



A Graph ADT

Brainstorm: What operations should a Graph support?

Query: Access info

Mutate: Change

Iterate

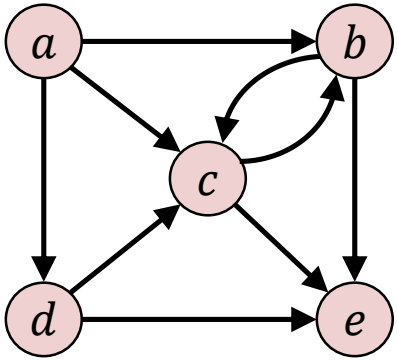


Coding Demo: Graph ADTs



Adjacency Matrix Representation

Use a 2D array/list structure to keep track of which edges are present



AdjMatrixGraph State

```
public class AdjMatrixGraph implements Graph<AMVertex, Edge<AMVertex>> {
  /** A vertex in this graph */
  public class AMVertex implements Vertex<Edge<AMVertex>> {
    /** The label of this vertex */
    private final String label;

    /** The index of this vertex in the edges 2D ArrayList */
    private final int index;
  }

  /** Map associating labels with AMVertex objects. */
  private final HashMap<String, AMVertex> vertices;

  /** edges.get(i).get(j) = reference to Edge (i,j), or null if
   *  this edge is not present in the graph. */
  private final ArrayList<ArrayList<Edge<AMVertex>>> edges;
}
```

Adjacency Matrix Operation Complexities

```
public class AdjMatrixGraph implements Graph<...> {
  public class AMVertex implements Vertex<...> {
    private String label;
    private int index;

    // methods
  }

  private HashMap<String, AMVertex> vertices;
  private ArrayList<ArrayList<Edge>>> edges;

  // methods
}
```

Count/Iterate vertices:

Count/Iterate neighbors:

Check for an edge:

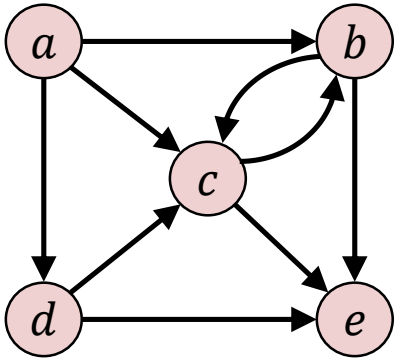
Count/Iterate edges:

Add edge:

Add vertex:

Adjacency List Representation (Classical)

- Maintain a list of all vertices.
- Have each vertex maintain a list of all its neighbors.



Adjacency List Representation (Improved)

AdjListGraph State

```
public class AdjListGraph implements Graph<ALVertex, Edge<ALVertex>> {
  /** A vertex in this graph responsible for tracking its neighbors. */
  public static class ALVertex implements Vertex<Edge<ALVertex>> {
    /** The label of this vertex */
    private final String label;

    /** Map associating neighbor's labels with Edges connecting to them. */
    private final HashMap<String, Edge<ALVertex>> outEdges;
  }

  /** Map associating labels with ALVertex objects. */
  private final HashMap<String, ALVertex> vertices;
}
```

Adjacency List Operation Complexities

```
class AdjListGraph implements Graph<...> {
  static class ALVertex implements Vertex<...> {
    private String label;
    private HashMap<String, Edge<...>> outEdges;

    // methods
  }

  private HashMap<String, ALVertex> vertices;

  // methods
}
```

Count vertices/neighbors:

Iterate over neighbors:

Check for an edge:

Count edges:

Iterate over edges:

Add edge/vertex:

Comparing Representations

Representation	Memory	Time to Query Edge	Time to Iterate Neighbors	Time to Iterate all Edges
Adj Matrix				
Adj List (Linked)				
Adj List (Map)				

Dense Graphs:

Sparse Graphs: