CSE 326: Data Structures
Lecture #17
Trees and DAGs and Graphs, Oh MY!

Bart Niswonger

Summer Quarter 2001

Today's Outline

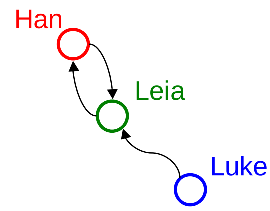
- Project IV
- Stuff Bart didn't get to Monday
- Graphs (what are they?)
- Topological Sort
- Graph Data Structures
- Graph Properties

Graph… ADT?

Graphs - a formalism for representing relationships

a graph **G** is represented as **G = (V, E)**

- **V** is a set of vertices: $\{v_1, v_2, \ldots, v_n\}$
- **E** is a set of edges: $\{e_1, e_2, \ldots, e_m\}$ where each e_i connects two vertices (v_{i1}, v_{i2})

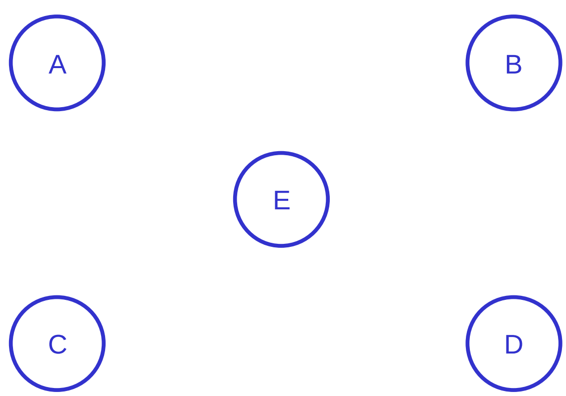



```
V = {Han, Leia, Luke}
E = {(Luke, Leia),
     (Han, Leia),
     (Leia, Han)}
```

operations include:

- iterating over vertices
- iterating over edges
- iterating over vertices adjacent to a specific vertex
- asking whether an edge exists connected two vertices

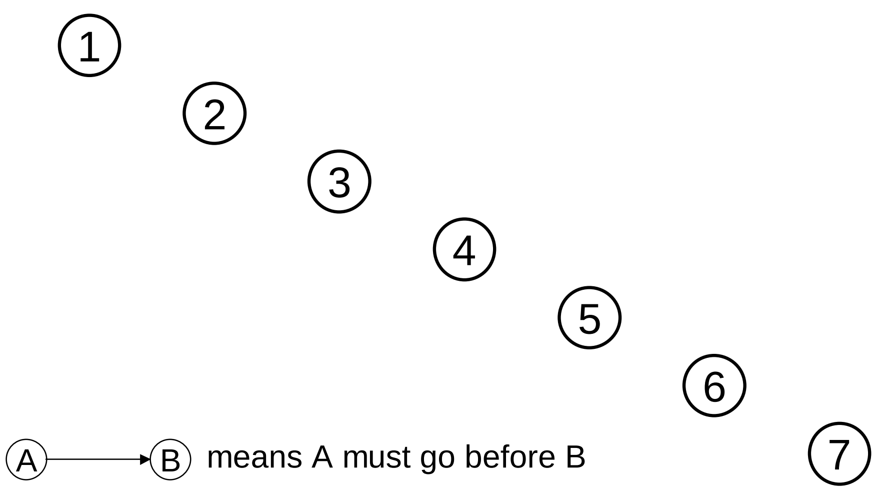
How Many Edges?



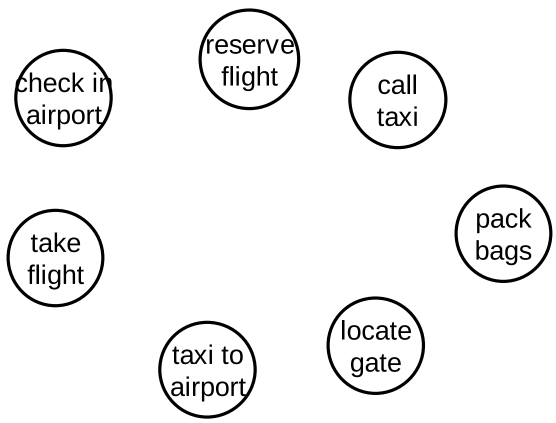
Graph Applications

- Storing things that are graphs by nature
  - distance between cities
  - airline flights, travel options
  - relationships between people, things
  - distances between rooms in Clue
- Compilers
  - *callgraph* - which functions call which others
  - *dependence graphs* - which variables are defined and used at which statements

Total Order

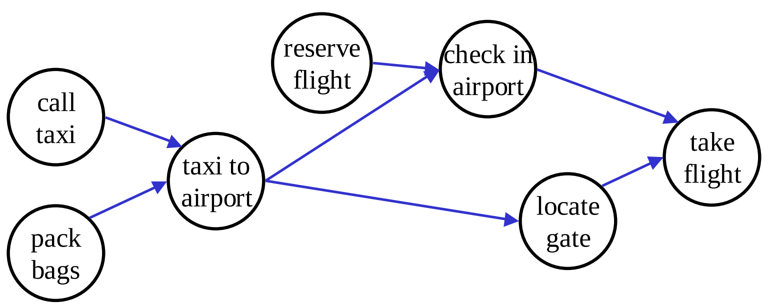


Partial Order: Planning a Trip



Topological Sort

Given a graph, **`G = (V, E)`**, output all the vertices in **`V`** such that no vertex is output before any other vertex with an edge to it.



Topo-Sort Take One

Label each vertex's *in-degree* (# of inbound edges)

While there are vertices remaining

1. Pick a vertex with in-degree of zero and output it
2. Reduce the in-degree of all vertices adjacent to it
3. Remove it from the list of vertices

runtime:

Topo-Sort Take Two

Label each vertex's in-degree

Initialize a queue to contain all in-degree zero vertices

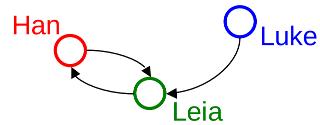
While there are vertices remaining in the queue

1. Pick a vertex *v* with in-degree of zero and output it
2. Reduce the in-degree of all vertices adjacent to *v*
3. Put any of these with new in-degree zero in the queue
4. Remove *v* from the queue

runtime:

Graph Representations

- List of vertices + list of edges

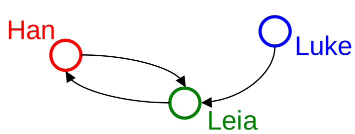


- 2-D matrix of vertices (marking edges in the cells)
  "adjacency matrix"

- List of vertices each with a list of adjacent vertices
  "adjacency list"

Adjacency Matrix

A **|V| x |V|** array in which an element **(u, v)** is true if and only if there is an edge from **u** to **v**



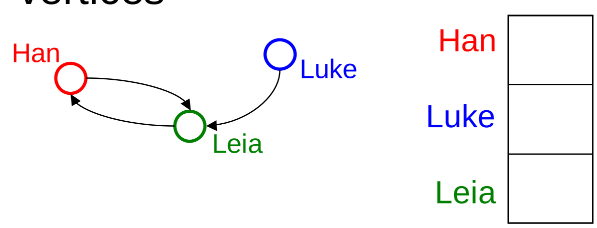
	Han	Luke	Leia
Han			
Luke			
Leia			

runtime:

space requirements:

Adjacency List

A **|V|**-ary list (array) in which each entry stores a list (linked list) of all adjacent vertices

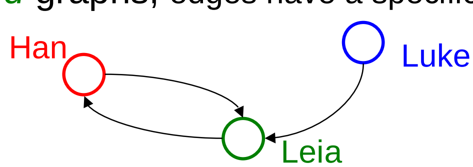


runtime: space requirements:

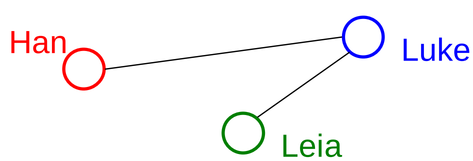
Directed vs. Undirected Graphs

- In *directed* graphs, edges have a specific direction:

aka: di-graphs



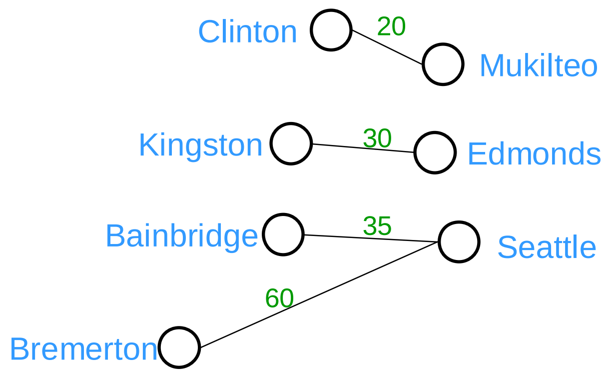
- In *undirected* graphs, they don't (edges are two-way):



- Vertices **u** and **v** are *adjacent* if $(u, v) \in E$

Weighted Graphs

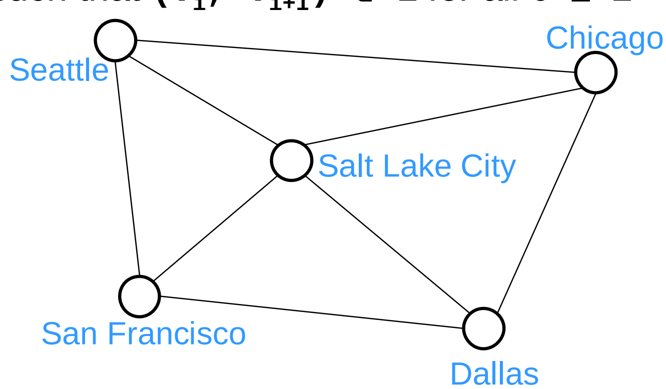
Each edge has an associated weight or cost.



There may be more information in the graph as well.

Paths

A *path* is a list of vertices $\{v_1, v_2, \ldots, v_n\}$ such that $(v_i, v_{i+1}) \in E$ for all $0 \leq i < n$.

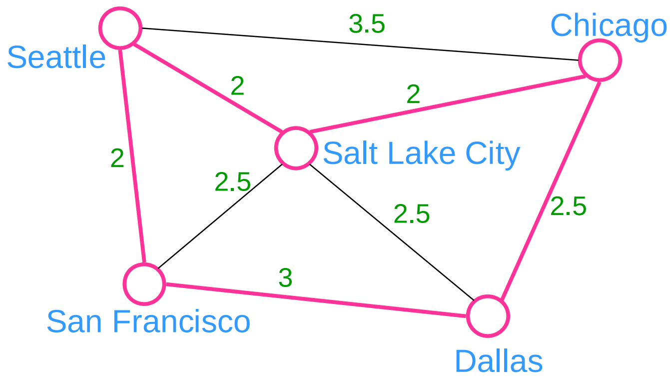


p = {Seattle, Salt Lake City, Chicago, Dallas, San Francisco, Seattle}

Path Length and Cost

Path length: the number of edges in the path

Path cost: the sum of the costs of each edge



length(p) = 5 cost(p) = 11.5

Simple Paths and Cycles

A *simple path* repeats no vertices (except that the first can be the last):

- p = {Salt Lake City, San Francisco, Dallas}
- p = {Seattle, Salt Lake City, Dallas, San Francisco, Seattle}

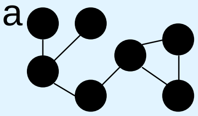
A *cycle* is a path that starts and ends at the same node:

- p = {Seattle, Salt Lake City, Dallas, San Francisco, Seattle}

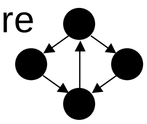
A *simple cycle* is a cycle that repeats no vertices except that the first vertex is also the last (in undirected graphs, no edge can be repeated)

Connectivity

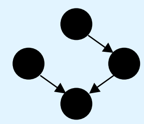
Undirected graphs are *connected* if there is a path between any two vertices



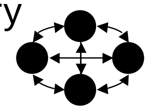
Directed graphs are *strongly connected* if there is a path from any one vertex to any other



Di-graphs are *weakly connected* if there is a path between any two vertices, *ignoring direction*



A *complete* graph has an edge between every pair of vertices



Graph Density

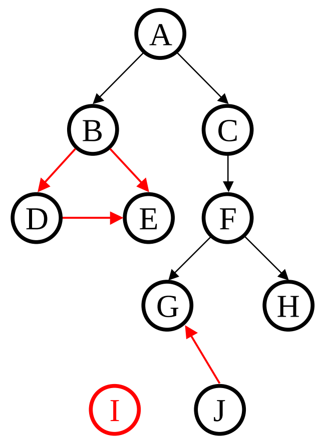
A *sparse* graph has O(|V|) edges

A *dense* graph has $\Theta(|V|^2)$ edges

Anything in between is either *sparsish* or *densy* depending on the context.

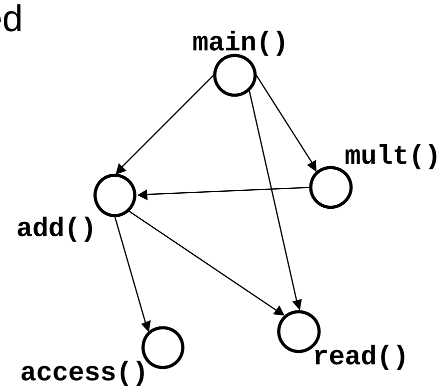
Trees as Graphs

- Every tree is a graph with some restrictions:
  - the tree is *directed*
  - there are *no cycles* (directed or undirected)
  - there is a *directed path from the root to every node*



Directed Acyclic Graphs (DAGs)

DAGs are directed graphs with no cycles.



Trees ⊂ DAGs ⊂ Graphs

To Do

- Finish Project III (due Today!)
- Read chapter 9 (see Calendar)
- Read Project IV writeup

Coming Up

- **Graph Algorithms!**
- Quiz (tomorrow)
- Project IV code