[TAP:CWJXL] Balanced Trees

- Which of the following are not guaranteed to be "balanced"?
  A. AVL Tree
  B. Red-black Tree
  [✓ C. Splay Tree]
  D. They are all balanced
  E. Whatever

Today's Outline

- Graphs
  - Undirected Graph
  - Directed Graph
  - Implementation

Graphs Describe the World

- map of cities

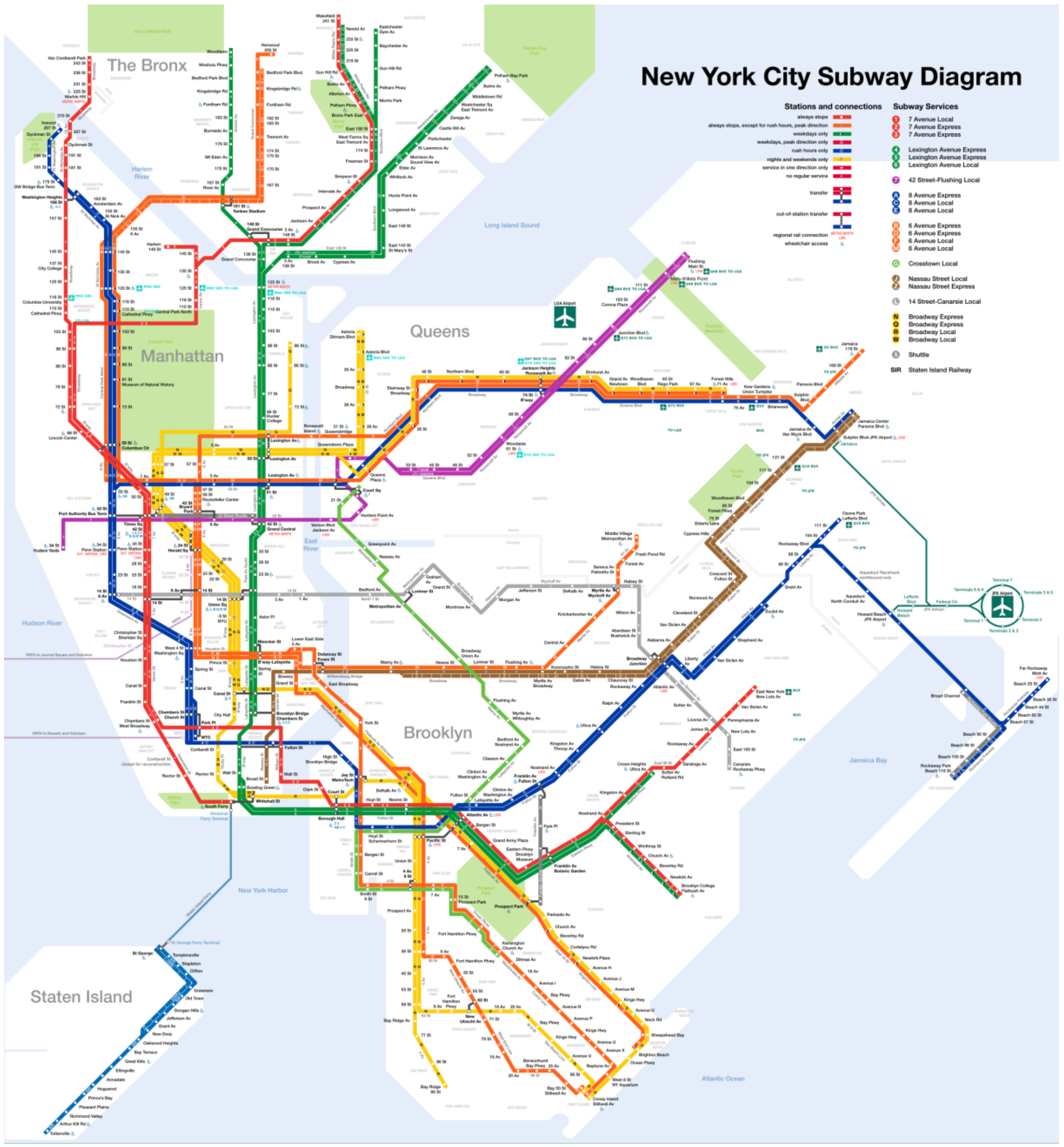
  cities , roads

  cities , rivers

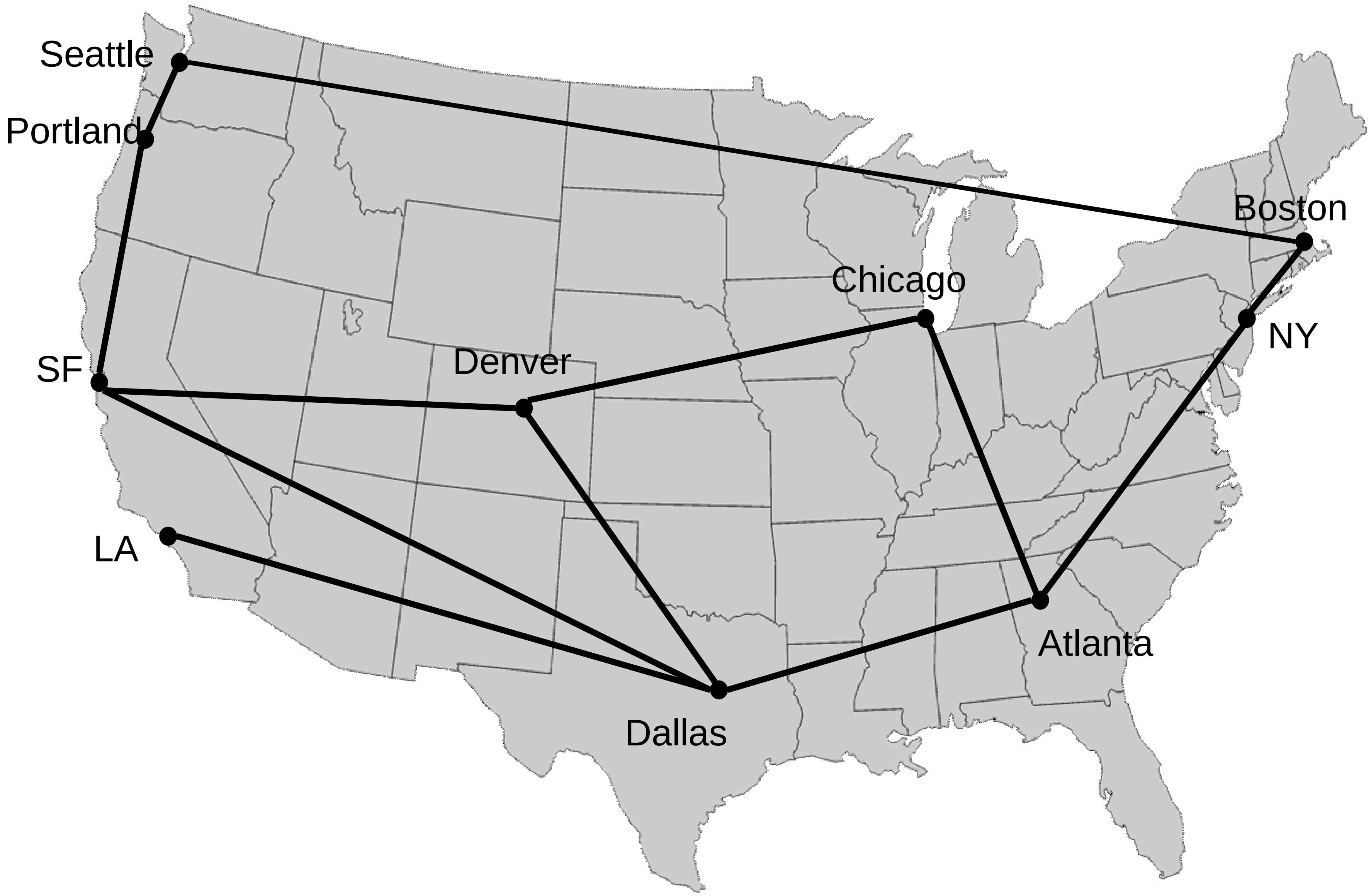
- Social network

  users , following (twitter)

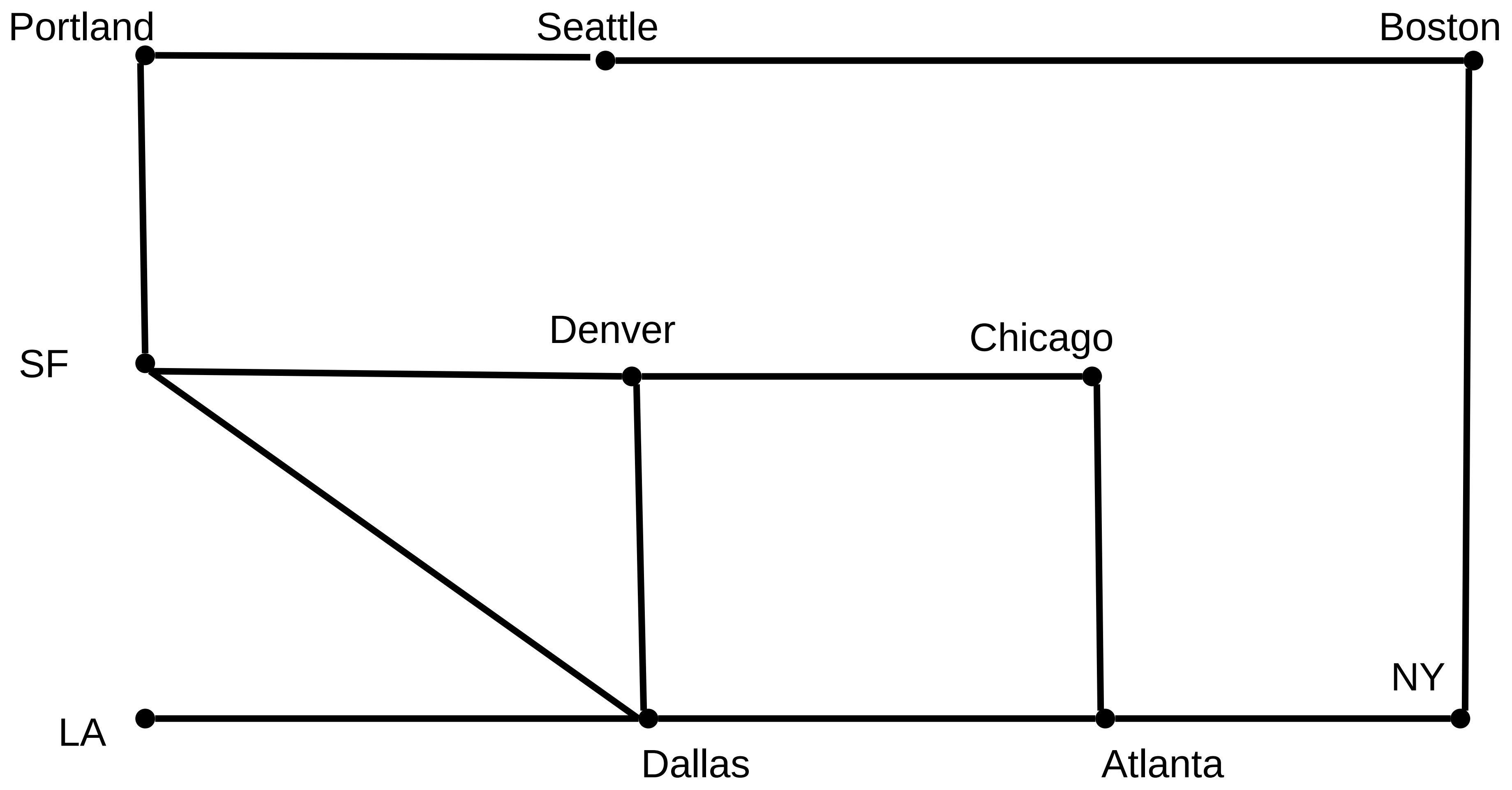
  users , friend (facebook)



Nodes = subway stops; Edges = track between stops

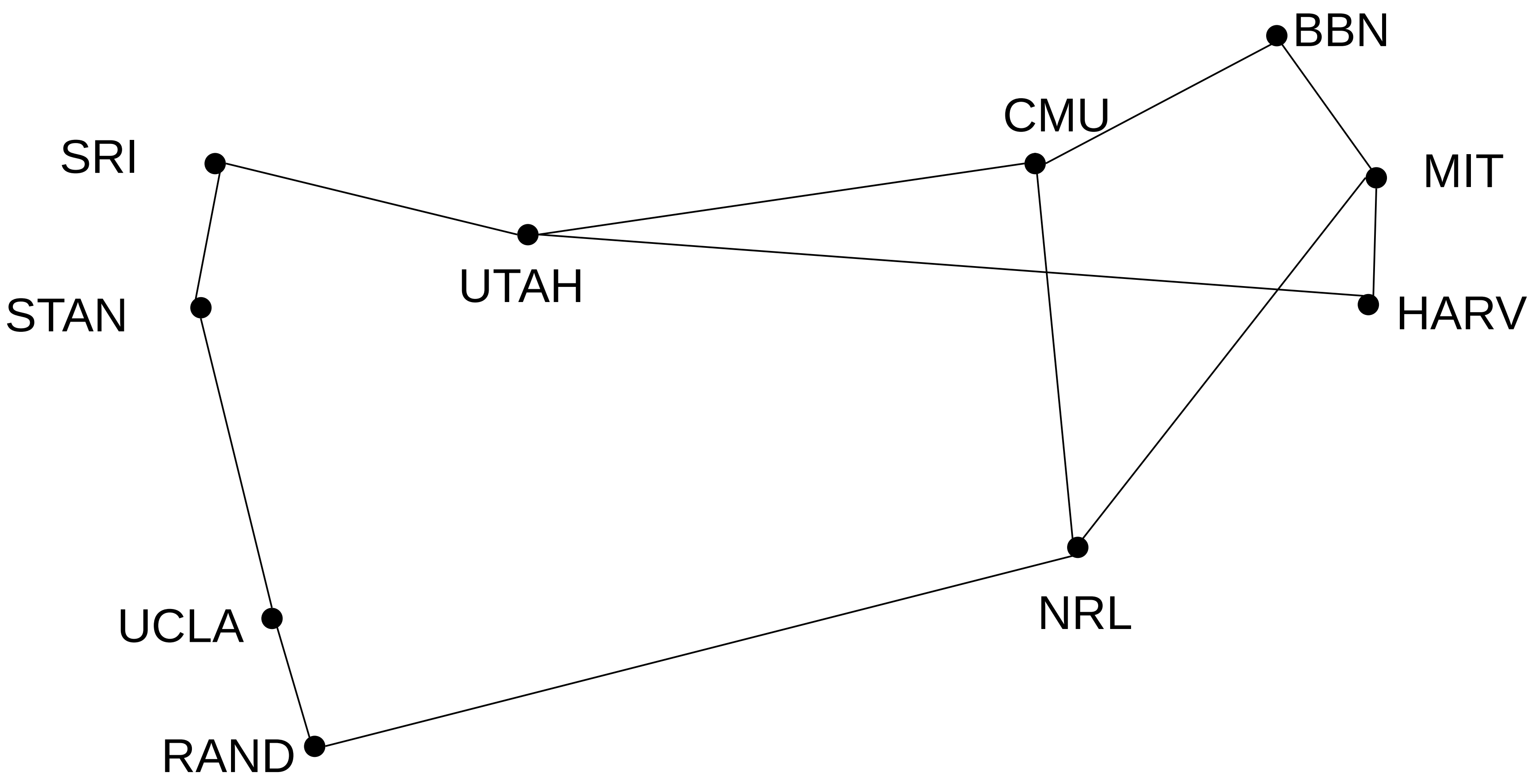


Nodes = cities; Edges = rail lines connecting cities



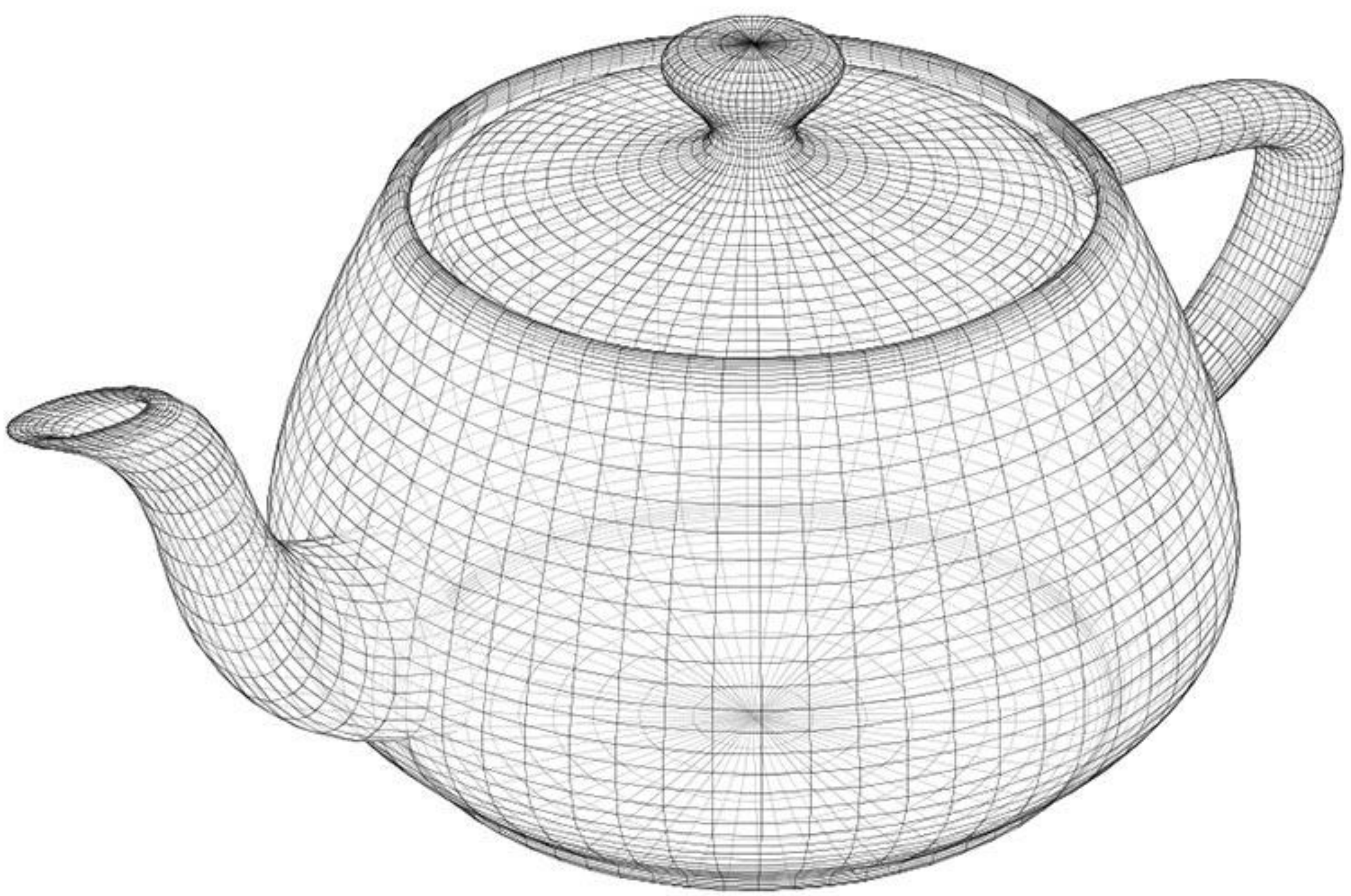
Connections in graph matter, not precise locations of nodes

Internet (~1972)



Facebook social network graph

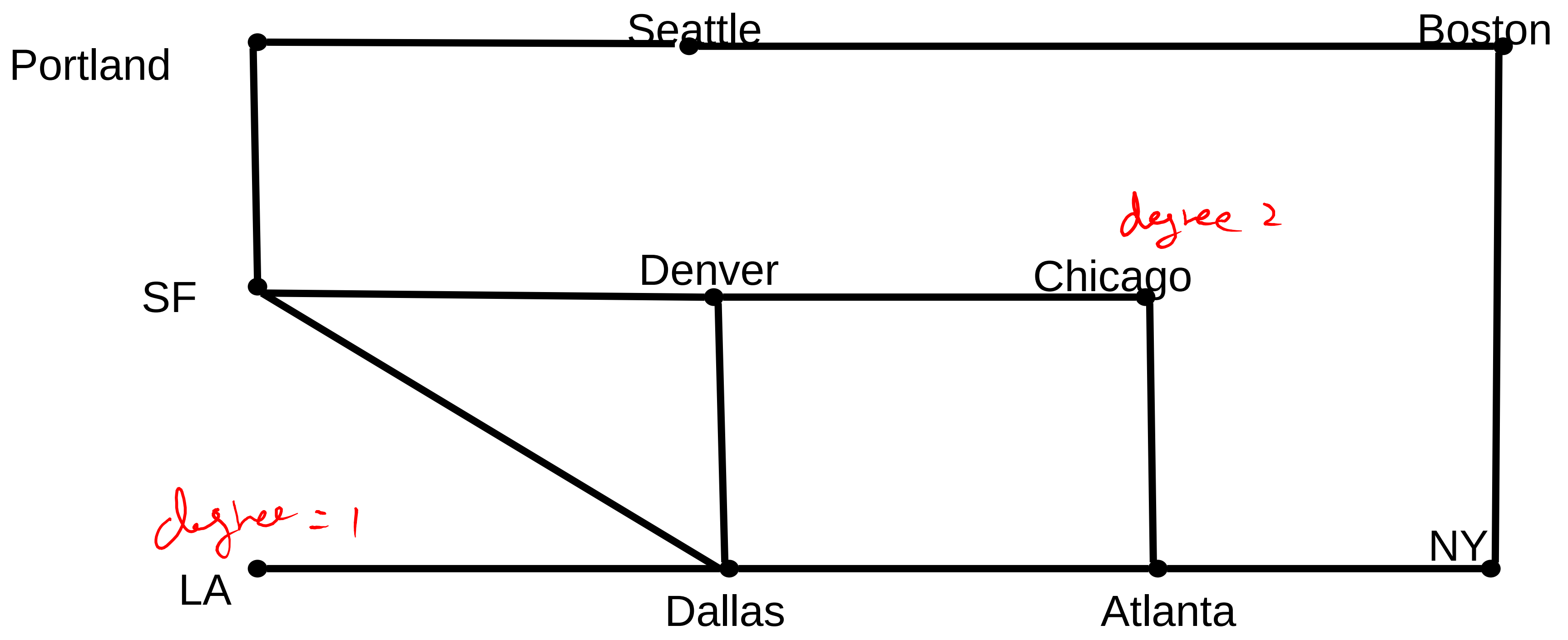
Wire-Frame Models



Today's Outline

- Graphs
  - Undirected Graph
  - Directed Graph
  - Implementation

Undirected Graph



An *undirected graph* is denoted as G = (V,E), where

- V: set of vertices
- E: set of edges, and each edge is an unordered pair of vertices (we write e = {u,v})

u + v are adjacent if there is an edge connecting u and v.

degree (v) = # of edges that are incident to v

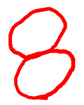
Walking Along a Graph

if u = v

- A *walk from u to v* in a graph G = (V,E) is an *alternating* sequence of vertices and edges (often, we just write the vertices) closed walk

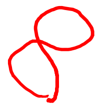
$$u = v_0, e_1, v_1, e_2, v_2, \dots , v_{k-1}, e_k, v_k = v$$

such that each $e_i = \{v_i , v_{i+1}\}$ for i = 1, ... , k



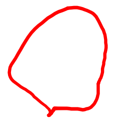
- A *path (trail)* is a walk where no edge appears more than once

circuit



- A *simple path (path)* is a walk where no vertex appears more than once

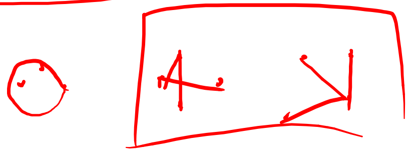
cycle



Reachability and Connectedness

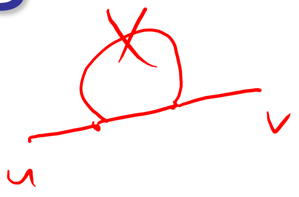
- A vertex v in G is *reachable* from a vertex u in G if there is a path from u to v
  - Note, v is $\overset{\text{or walk}}{\text{reachable}}$ from u *iff* u is reachable from v
- An undirected graph G is *connected* if for every pair of vertices u, v in G, v is reachable from u (and vice versa)
- The set of all vertices reachable from v, along with all edges of G connecting any two of them, is called the *connected component of v*

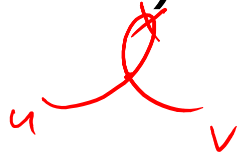
← G is not connected but has 2 CC.



Little Tiny Theorems

- If there is a walk from u to v, then
  - there is a walk from v to u.
  - there is a path from u to v (and from v to u)
- If there is a path from u to v, then
  - there is a simple path from u to v (and v to u)





[TAP] Sum of degrees

- Let *deg(v) be* the degree of a vertex v. Is the following statement true?
- For any graph *G = (V,E)*

$$\sum_{v \in V} \deg(v) = 2|E|$$

where |*E*| is the number of edges in *G*

T

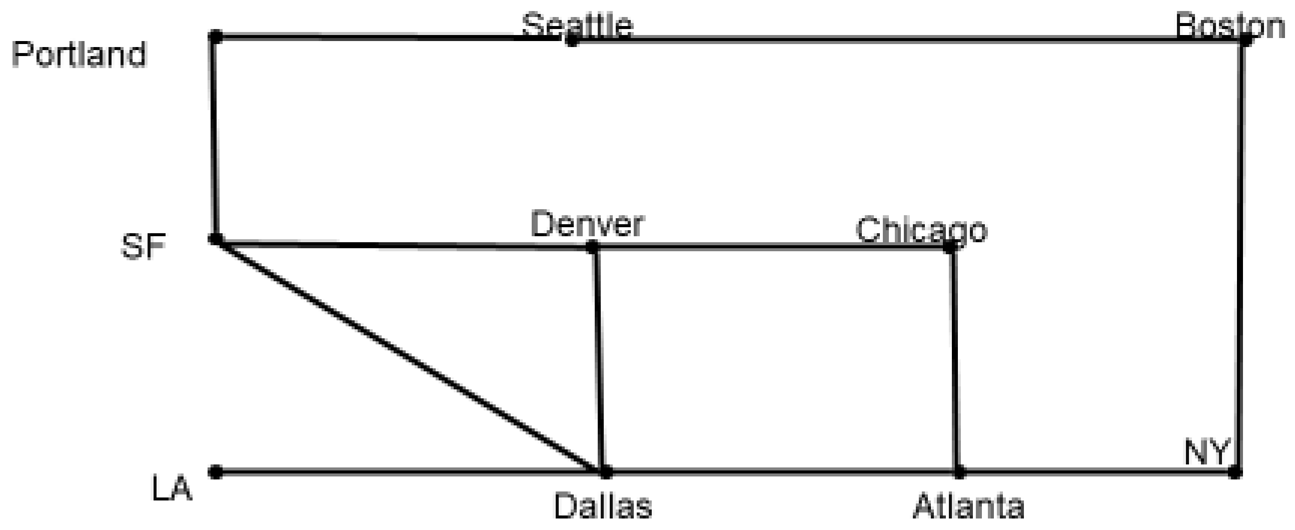
Q

Today’s Outline

- Graphs
  - Undirected Graph
  - Directed Graph
  - Implementation

Directed Graph

Undirected Graph

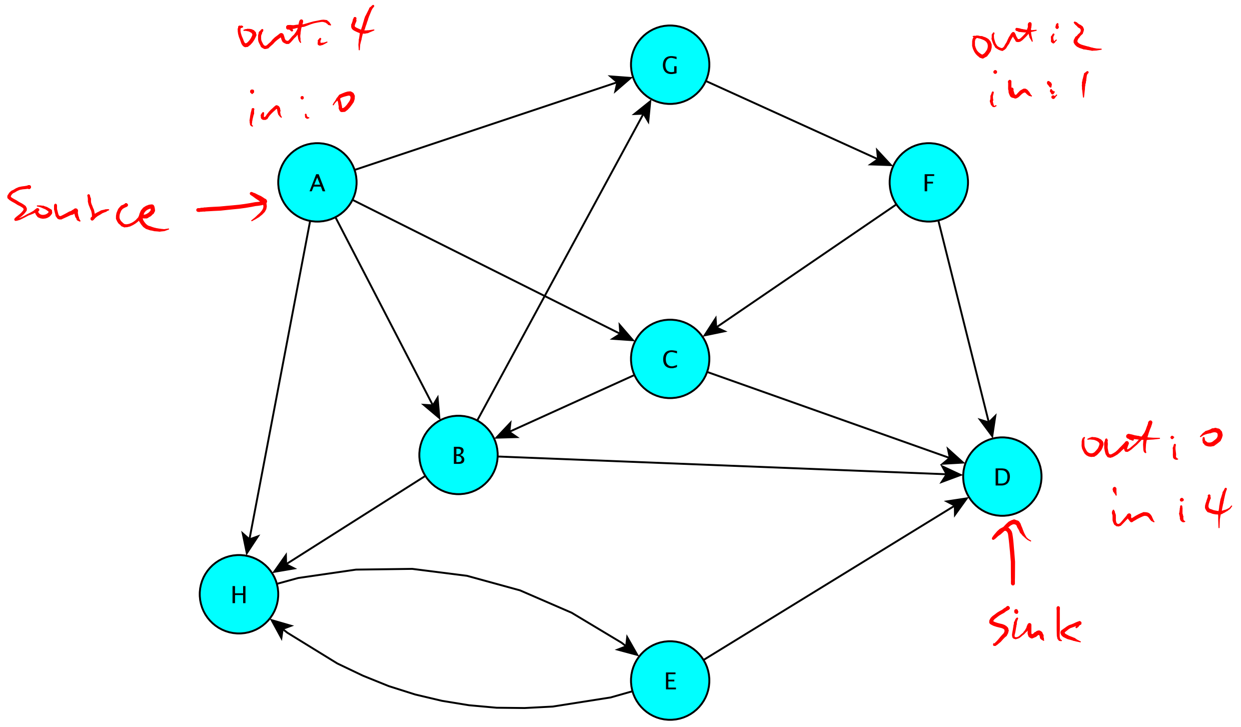


An *undirected graph* is denoted as G = (V,E), where

- V: set of vertices
- E: set of edges, and each edge is an ~~unordered~~ ordered pair of vertices (we write $e = \{\underset{\text{source}}{u},\underset{\text{target/destination}}{v}\}$)

13

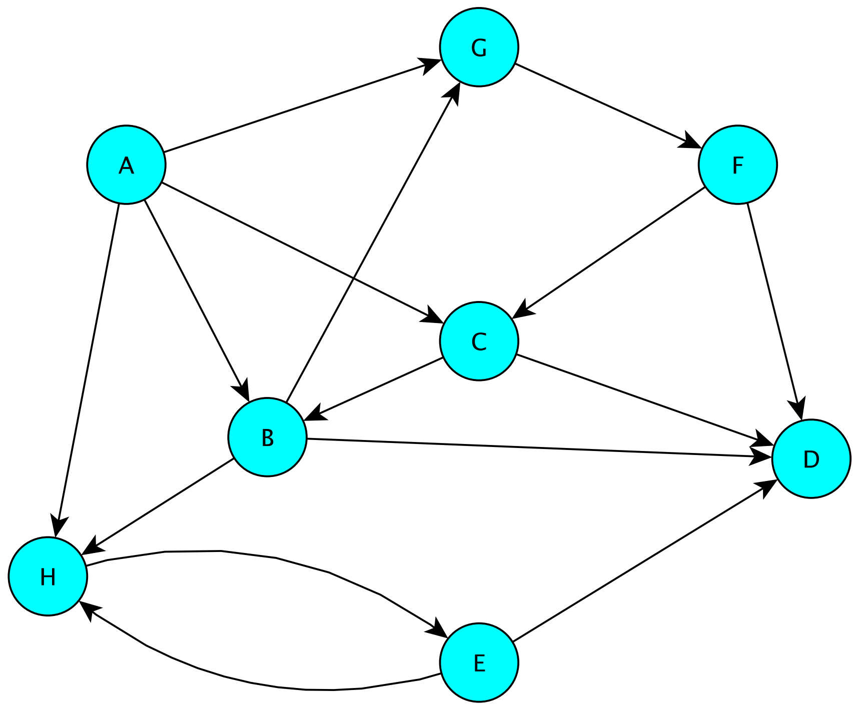
Degrees



out-degree = # of outgoing edges

in-degree = # of incoming edges

Walking Along a Graph



The concept of a walk and path is still the same, but you can only walk along the direction of the edges.

Reachability and Connectedness

- A vertex v in G is *reachable* from a vertex u in G if there is a path from u to v
  - Note, v is reachable from u *iff* u is reachable from v (not necessarily. It's true if u and v are mutually reachable)

connected

not strongly connected

- An undirected graph G is strongly *connected* if for every pair of vertices u, v in G, v is reachable from u (and vice versa)
- The set of all vertices reachable from v, along with all edges of G connecting any two of them, is called the strongly *connected component of v*

15

Today's Outline

- Graphs
    - Undirected Graph
    - Directed Graph
    - Implementation

Implementing Graphs

- Involves a number of implementation decisions, depending on intended uses
  - What kinds of graphs will be supported?
    - Undirected, directed, mixed
  - What underlying data structures will be used?
  - What functionality will be provided
  - What aspects will be public/protected/private

Graphs in structure5

- Interface Graph<V,E> extends Structure<V>
  - Type V holds a *label* for a vertex
  - Type E holds a *label* for an edge

"information"

e.g. city name,

distance

Desired Functionality

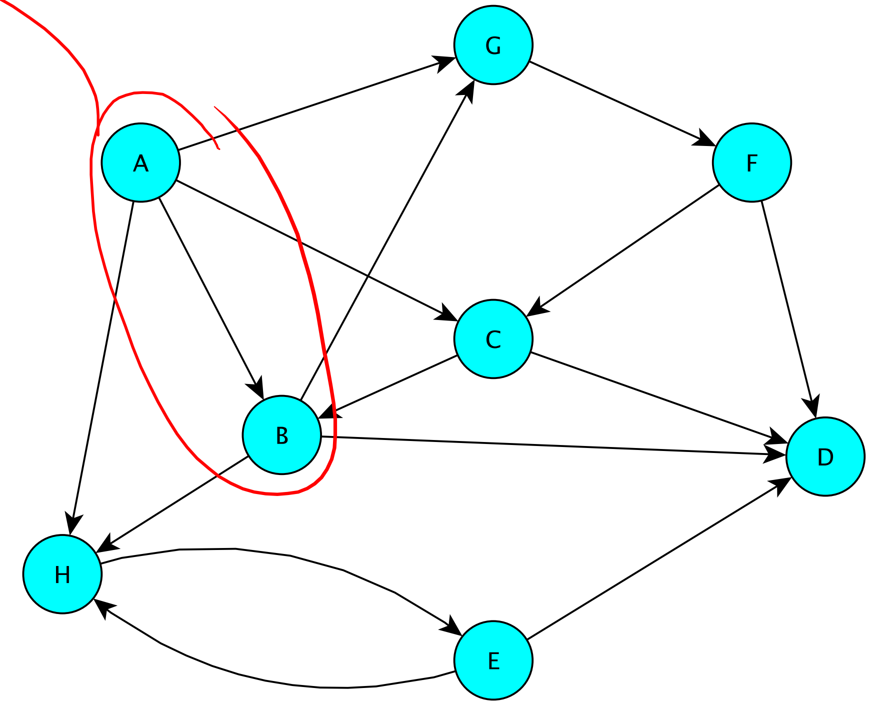
- What are the basic operations we need to describe algorithms on graphs?
  - Given vertices u and v: are they adjacent?
  - Given vertex v and edge e, are they incident?
  - Given an edge e, get its incident vertices (*ends*)
  - How many vertices are adjacent to v? (*degree* of v)
    - The vertices adjacent to v are called its *neighbors*
  - Get a list of the neighbors of v (or the edges incident with v)

Representing Graphs

- Two standard approaches
  - Option 1: Array-based (directed and undirected)
  - Option 2: List-based (directed and undirected)

Adjacency Array: Directed Graph

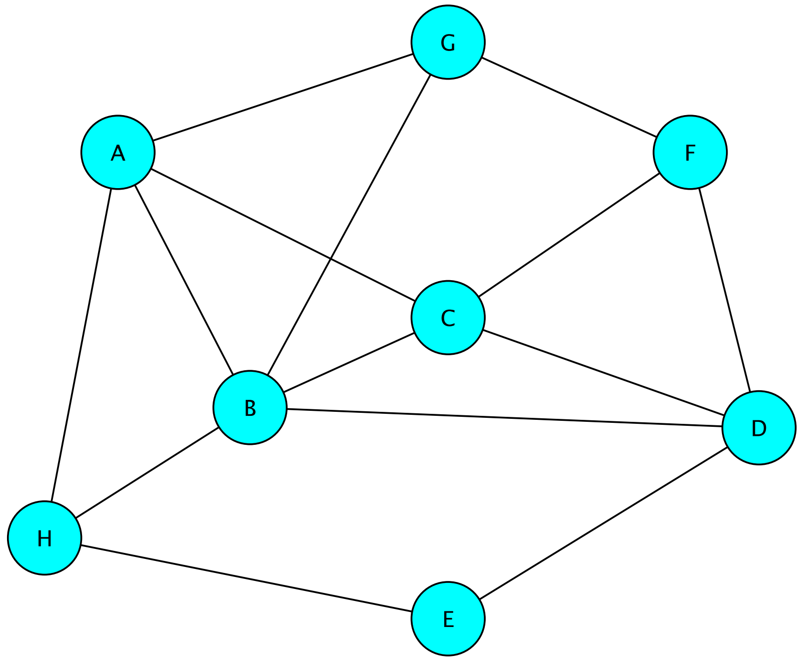
	A	B	C	D	E	F	G	H
A	0	1	1	0	0	0	1	1
B	0	0	0	1	0	0	1	1
C	0	1	0	1	0	0	0	0
D	0	0	0	0	0	0	0	0
E	0	0	0	1	0	0	0	1
F	0	0	1	1	0	0	0	0
G	0	0	0	0	0	1	0	0
H	0	0	0	0	1	0	0	0



Entry (i,j) stores 1 if there is an edge from i to j; 0 otherwise
E.G.: edges(B,C) = 1 but edges(C,B) = 0

Adjacency Array: Undirected Graph

	A	B	C	D	E	F	G	H
A	0	1	1	0	0	0	1	1
B	1	0	1	1	0	0	1	1
C	1	1	0	1	0	1	0	0
D	0	1	1	0	1	1	0	0
E	0	0	0	1	0	0	0	1
F	0	0	1	1	0	0	1	0
G	1	1	0	0	0	1	0	0
H	1	1	0	0	1	0	0	0



Entry (i,j) store 1 if there is an edge between i and j; else 0
E.G.: edges(B,C) = 1 = edges(C,B)

Adjacency Array: Undirected Graph

Halving the Space (not in structure5)

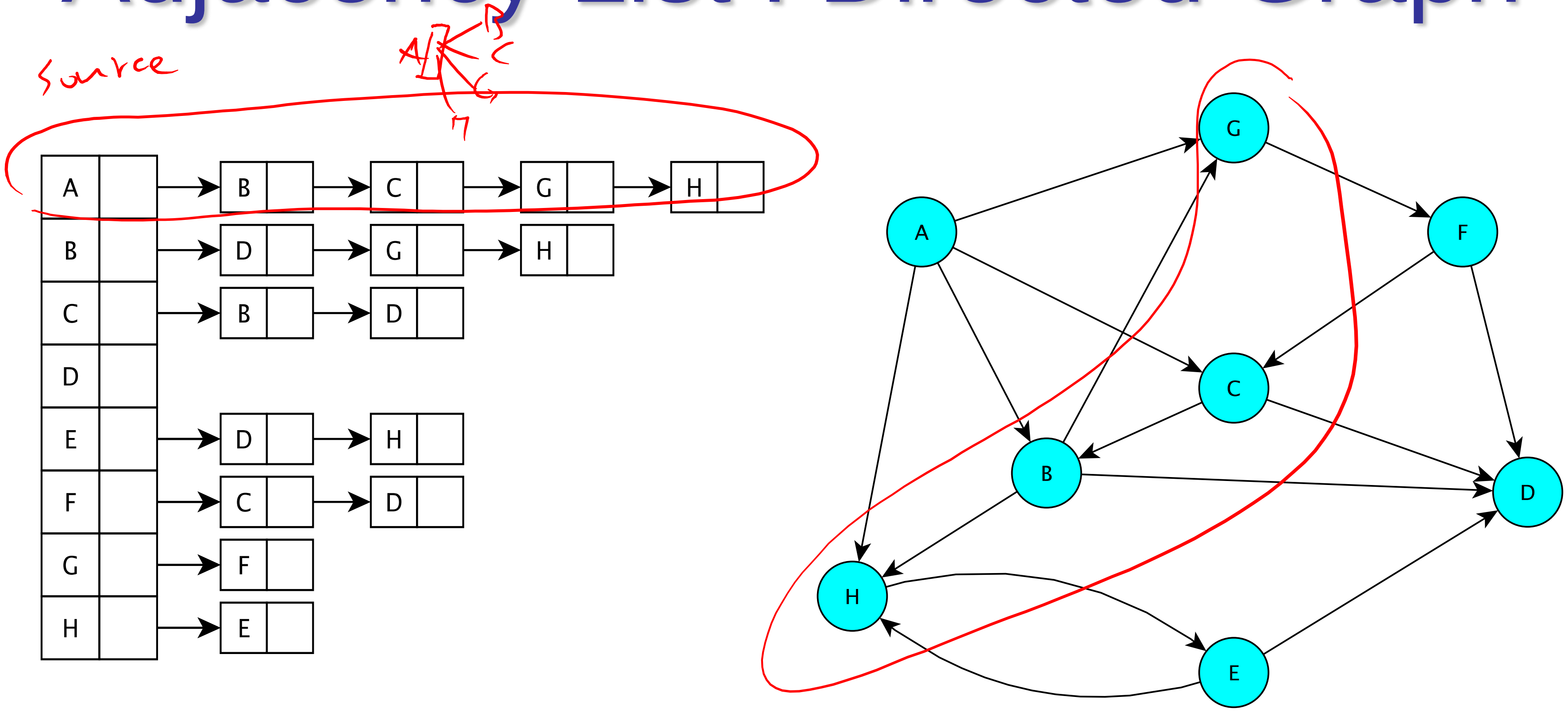
	0	1	2	3	4	5	6
0	0	1	1	0	0	0	1
1	1	0	1	1	0	0	1
2	1	1	0	1	0	1	0
3	0	1	1	0	1	1	0
4	0	0	0	1	0	0	0
5	0	0	1	1	0	0	1
6	1	1	0	0	0	1	0

	0	1	2	3	4	5	6
0	0	1	1	0	0	0	1
1		0	1	1	0	0	1
2			0	1	0	1	0
3				0	1	1	0
4					0	0	0
5						0	1
6							0

0	1	2	3	4	5	6	7	8	9	…																	
0	1	1	0	0	0	1	0	1	1	0	0	1	0	1	0	1	0	0	1	1	0	0	0	0	0	1	0

(i,j) maps to i*7+j

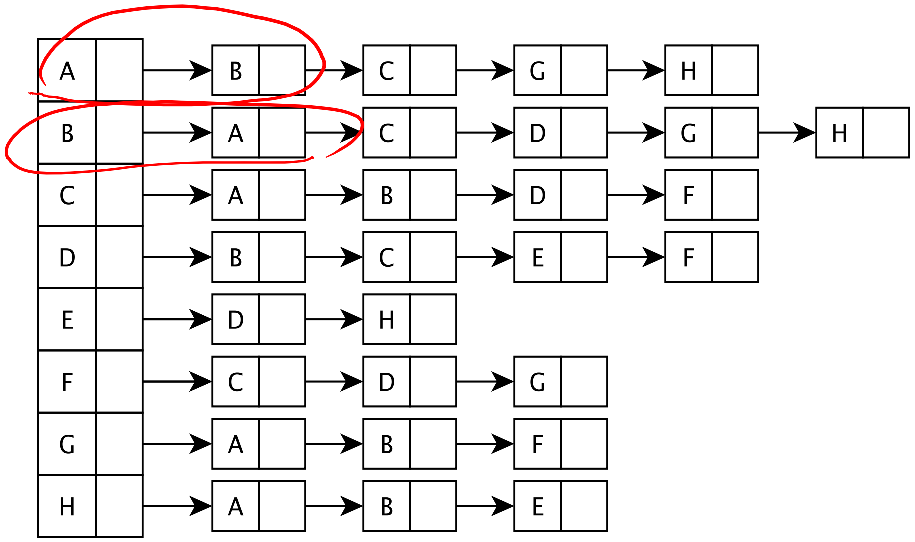
Adjacency List : Directed Graph

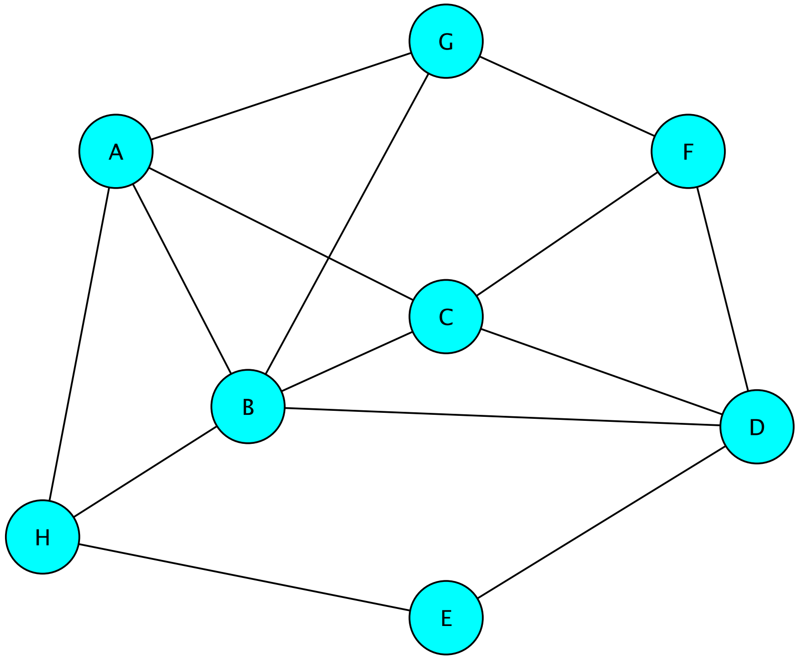


The vertices are stored in an array V[]

V[] contains a linked list of edges having a given source

Adjacency List : Undirected Graph





The vertices are stored in an array V[]
V[] contains a linked list of edges incident to a given vertex