Lecture 19
Graphs and Graph Algorithms

EECS 281: Data Structures & Algorithms

Formal Definition: Graph

- Definition: A **graph** $G = (V, E)$ is a set of **vertices** $V = \{v_1, v_2, \ldots\}$ together with a set of **edges** $E = \{e_1, e_2, \ldots\}$ that connect pairs of vertices.
- Edges are often represented as ordered pairs, such as $e_m = (v_s, v_t)$.

Graph: More Detail

- In general
  - Parallel edges are allowed
  - Self-loops are allowed
- However, graphs <u>without</u> parallel edges and <u>without</u> self-loops are called *simple* graphs
- In general, assume a graph is *simple* unless explicitly specified

Graphs: Definitions

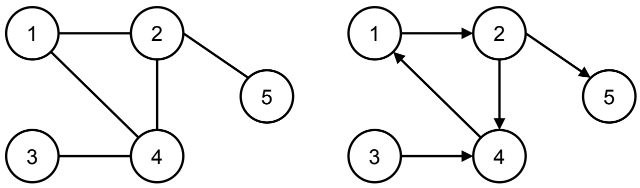
- **Simple Path**: sequence of edges leading from one vertex to another with no vertex appearing twice
- **Connected Graph**: a simple path exists between any pair of vertices
- **Cycle**: simple path, except that first and final nodes are the same

Graphs: Directed vs. Undirected

- Directed Graph or "digraph"
  - Edges have direction (one-way)
  - Nodes on edges form ordered pairs
    - Order of vertices in edge is important
    - $e_n = (u, v)$ means there is an edge <u>from</u> u <u>to</u> v
- Undirected Graph
  - Nodes on edges form unordered pairs
    - Order of vertices in edge is not important
    - $e_n = (u, v)$ means there is an edge <u>between</u> u and v

Directed vs. Undirected

Undirected Graph Directed Graph

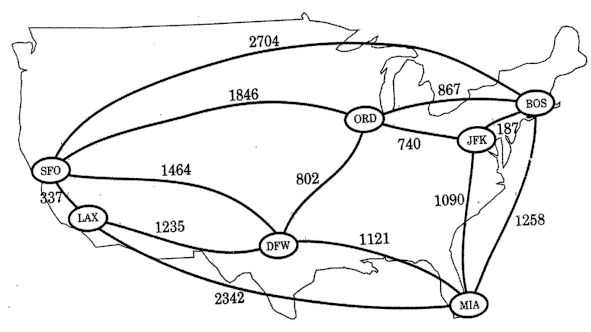


Graphs: Weighted Graphs

Edges may be "weighted"

- Think of weight as the "distance between nodes" or "cost to traverse the edge"
- In undirected graphs, weights may be different for sets of parallel edges
- Algorithms often search a graph for *a path* (unweighted), or *least cost path* (weighted)

Airline Routes

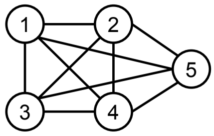


Graphs: Data Structures

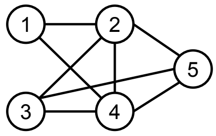
- Complete Graph
  - All possible edges ($|E| = |V| * |V - 1| / 2 \approx |V|^2$)
- Dense Graph
  - Many edges ($|E| \approx |V|^2$)
  - Represent as adjacency matrix (adjmat)
- Sparse Graph
  - Few edges ($|E| << |V|^2$) or ($|E| \approx |V|$)
  - Represent as adjacency list (adjlist)

Complete vs. Dense vs. Sparse

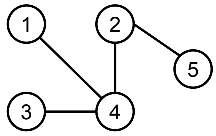
Complete Graph



Dense Graph

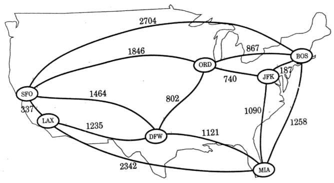


Sparse Graph



Adjacency Matrix

	SFO	LAX	DFW	ORD	MIA	JFK	BOS
SFO	0	1	1	1	0	0	1
LAX	1	0	1	0	1	0	0
DFW	1	1	0	1	1	0	0
ORD	1	0	1	0	0	1	1
MIA	0	1	1	0	0	1	1
JFK	0	0	0	1	1	0	1
BOS	1	0	0	1	1	1	0



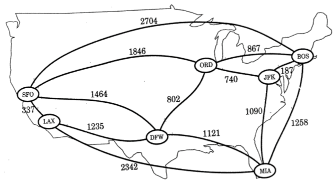
Adjacency List

SFO → LAX → DFW → ORD → BOS

LAX → SFO → DFW → MIA

DFW → SFO → LAX → ORD → MIA

⋮



Note: Adjacency list nodes can contain distances

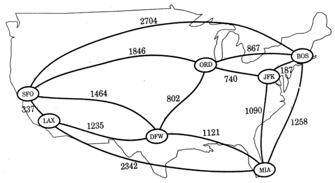
Graphs: Data Structures

Adjacency Matrix Implementation

- $|V|$ x $|V|$ matrix representing graph
- Directed vs. undirected
  - Directed adjmat has to/from
  - Undirected adjmat only needs $\sim V^2/2$ space
- Unweighted vs. weighted
  - Unweighted: 0 = no edge, 1 = edge
  - Weighted: ∞ = no edge, value = edge

Undirected Distance Matrix

	SFO	LAX	DFW	ORD	MIA	JFK	BOS
SFO	∞	337	1464	1846	∞	∞	2704
LAX	337	∞	1235	∞	2342	∞	∞
DFW	1464	1235	∞	802	1121	∞	∞
ORD	1846	∞	802	∞	∞	740	867
MIA	∞	2342	1121	∞	∞	1090	1258
JFK	∞	∞	∞	740	1090	∞	187
BOS	2704	∞	∞	867	1258	187	∞



Representing Infinity in C++

- `#include <limits>`
- Use `numeric_limits<double>::infinity()`
  - Adding, subtracting, multiplying and dividing finite values often does nothing to it
  - Multiplying by 0 results in 0 (Visual Studio) or nan (not a number) in g++
  - Dividing infinity by itself results in 1 (Visual Studio) or nan (g++)
  - Subtracting infinity from itself results in 0 (Visual Studio) or nan (g++)
- Don't use `numeric_limits<double>::max()`

Graphs: Data Structures

Adjacency List

- Assume random distribution of edges
- $\sim E/V$ edges on each vertex list
- Access vertex list: $O(1)$
- Find edge on vertex list: $O(E/V)$
- Average cost for individual vertex is $O(1 + E/V)$
- Cost for *all* vertices is $V \times O(1 + E/V) = O(V + E)$

Graphs: Data Structures

Adjacency List

- Directed vs. undirected
  - Directed adjlist contains each edge once in edge set
  - Undirected adjlist contains each edge twice in edge set
- Unweighted vs. weighted
  - Unweighted: nothing = no edge, <list_item> = edge
  - Weighted: nothing = no edge, <list_item_with_val> = edge

$O(1)$ to access a vertex list?

SFO → LAX → DFW → ORD → BOS
LAX → SFO → DFW → MIA
DFW → SFO → LAX → ORD → MIA
ORD → SFO → DFW → JFK → BOS
MIA → LAX → DFW → JFK → BOS
JFK → ORD → MIA → BOS
BOS → SFO → ORD → MIA → JFK

Graphs: Complexity

Complexity of graph algorithms is typically defined in terms of:

- Number of edges $|E|$, or
- Number of vertices $|V|$, or
- Both

Graph Algorithm Questions

- Task: Determine whether non-stop flight from X to Y exists.
- Worst/best/average complexity for both representations?

Adjacency Matrix

	SFO	LAX	DFW	ORD	MIA	JFK	BOS
SFO	∞	337	1464	1846	∞	∞	2704
LAX	337	∞	1235	∞	2342	∞	∞
DFW	1464	1235	∞	802	1121	∞	∞
ORD	1846	∞	802	∞	∞	740	867
MIA	∞	2342	1121	∞	∞	1090	1258
JFK	∞	∞	∞	740	1090	∞	187
BOS	2704	∞	∞	867	1258	187	∞

Worst: $O(1)$
Best: $O(1)$
Avg: $O(1)$

Adjacency List

SFO → LAX → DFW → ORD → BOS
LAX → SFO → DFW → MIA
DFW → SFO → LAX → ORD → MIA
ORD → SFO → DFW → JFK → BOS

Worst: $O(V)$
Best: $O(1)$
Avg: $O(1 + E/V)$

Graph Algorithm Questions

- Task: What is the closest other airport starting at X?
- Worst/best/average complexity for both representations?

Adjacency Matrix

	SFO	LAX	DFW	ORD	MIA	JFK	BOS
SFO	∞	337	1464	1846	∞	∞	2704
LAX	337	∞	1235	∞	2342	∞	∞
DFW	1464	1235	∞	802	1121	∞	∞
ORD	1846	∞	802	∞	∞	740	867
MIA	∞	2342	1121	∞	∞	1090	1258
JFK	∞	∞	∞	740	1090	∞	187
BOS	2704	∞	∞	867	1258	187	∞

Worst: $O(V)$
Best: $O(V)$
Avg: $O(V)$

Adjacency List

SFO → LAX → DFW → ORD → BOS
LAX → SFO → DFW → MIA
DFW → SFO → LAX → ORD → MIA
ORD → SFO → DFW → JFK → BOS

Worst: $O(V)$
Best: $O(1)$
Avg: $O(1 + E/V)$

Graph Algorithm Questions

- Task: Determine if ANY flights depart from airport X.
- Worst/best/average complexity for both representations?

Adjacency Matrix

	SFO	LAX	DFW	ORD	MIA	JFK	BOS
SFO	∞	337	1464	1846	∞	∞	2704
LAX	337	∞	1235	∞	2342	∞	∞
DFW	1464	1235	∞	802	1121	∞	∞
ORD	1846	∞	802	∞	∞	740	867
MIA	∞	2342	1121	∞	∞	1090	1258
JFK	∞	∞	∞	740	1090	∞	187
BOS	2704	∞	∞	867	1258	187	∞

Worst: $O(V)$
Best: $O(1)$
Avg: $O(V)$

Adjacency List

SFO → LAX → DFW → ORD → BOS
LAX → SFO → DFW → MIA
DFW → SFO → LAX → ORD → MIA
ORD → SFO → DFW → JFK → BOS

Worst: $O(1)$
Best: $O(1)$
Avg: $O(1)$

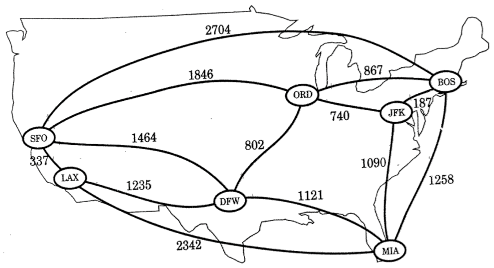
Graph Algorithm Questions

- Associate a distance with each edge
- Associate a cost with each edge
- Describe an algorithm to determine greatest distance for least cost (ratio) that can be flown from JFK on a non-stop flight
- Give complexity

JFK → ORD 740 \$200 → MIA 1090 \$600 → BOS 187 \$400

Route Planning

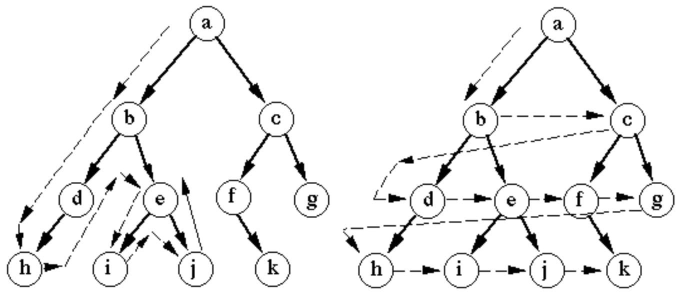
Task: determine the most efficient route (i.e. lowest cost) flown from X to Y on any trip, non-stop or connecting



Single-Source Shortest Path

- Find the **shortest** path to get to any vertex from some given starting point
- Depth First Search (DFS)
  Only works on trees; may find the wrong answer due to multiple paths in graphs
- Breadth First Search (BFS)
  Works for unweighted edges or where all weights are considered the same
- Dijkstra's Algorithm
  Works for weighted edges

Depth-First vs. Breath-First



Depth-First Search   Breadth-First Search

Depth-First Search

Given a graph $G = (V, E)$, explore the edges of G to discover if ***any*** path exists from the source s to the goal g

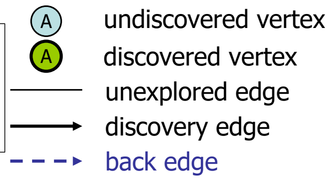
- Use a stack
- Algorithm works on graphs and digraphs
- Path discovered may be a shortest path, but it is not guaranteed to be the shortest

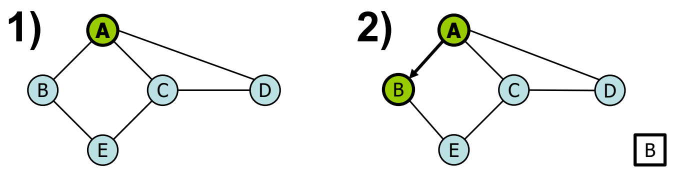
Depth-First Search

```
Algorithm GraphDFS
   Mark source as visited
   Push source to Stack
   While Stack is not empty
      Get/Pop candidate from top of Stack
      For each child of candidate
         If child is unvisited
            Mark child visited
            Push child to top of Stack
            If child is goal
               Return success
   Return failure
```

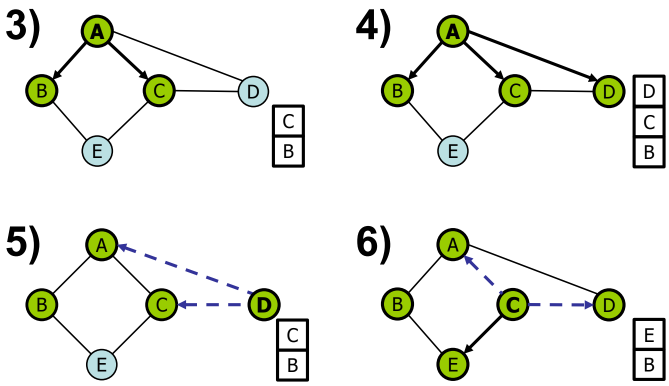

Example

Use Depth-First Search to discover if a path exists from A to E.





Example (cont.)



DFS: Analysis of Adjacency List

- DFS:
  - Called for each vertex at most once - $O(V)$
  - adjlist for each vertex is visited at most once and set of edges is distributed over set of vertices - $O(1 + E/V)$
- $O(V + E)$: linear with number of vertices and edges

DFS: Analysis of Adjacency Matrix

- DFS:
  - Visits each vertex at most once - $O(V)$
  - adjmat row for each vertex is visited at most once - $O(V)$
- $O(V^2)$: quadratic with number of vertices

Breadth-First Search

Given an unweighted graph $G = (V, E)$, explore the edges of G to discover a ***shortest*** path from source s to goal g, if any exists

- Use a queue
- Algorithm works on graphs and digraphs
- Discovers a ***shortest*** path only on unweighted graphs or where all edges have equal weight

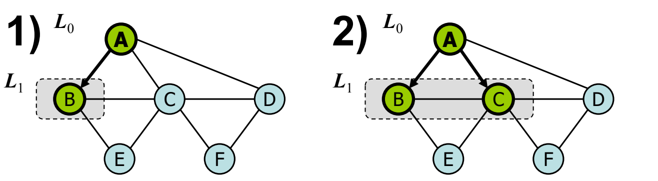
Breadth-First Search

```
Algorithm GraphBFS
   Mark source as visited
   Push source to back of Queue
   While Queue is not empty
      Get/Pop candidate from front of Queue
      For each child of candidate
         If child is unvisited
            Mark child visited
            Push child to back of Queue
            If child is goal
               Return success
   Return failure
```

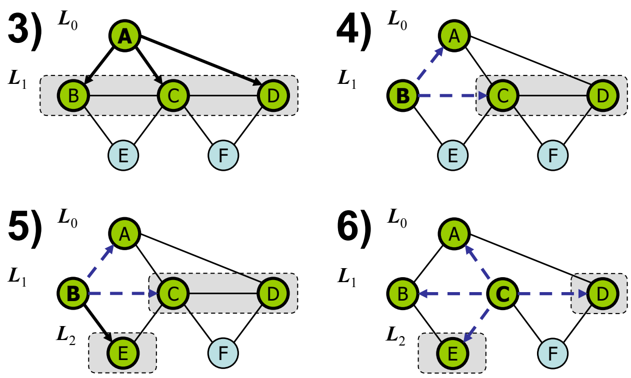

Example

Use Breadth-First Search to discover if a path exists from A to F.

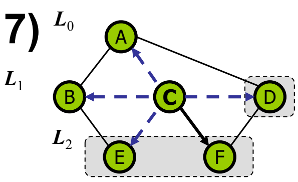
undiscovered vertex
discovered vertex
unexplored edge
discovery edge
back edge



Example (cont.)



Example (cont.)



BFS: Analysis of Adjacency List

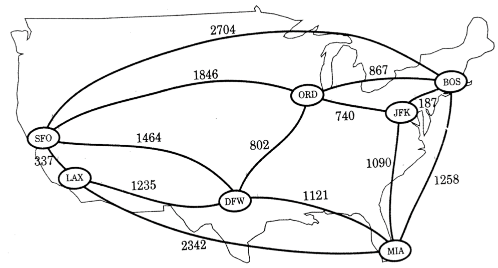
- BFS:
  - Visits each vertex at most once - $O(V)$
  - adjlist for each vertex is visited at most once and set of edges is distributed over set of vertices - $O(1 + E/V)$
- $O(V + E)$: linear with number of vertices and edges

BFS: Analysis of Adjacency Matrix

- BFS:
  - Called for each vertex at most once - $O(V)$
  - Adjmat row for each vertex is visited at most once - $O(V)$
- $O(V^2)$: quadratic with number of vertices

Dijkstra's Algorithm

Given a **weighted** graph $G = (V, E)$, explore the edges of G to discover a **shortest** path from source s to goal g



Graph Search Algorithms Summary

- Background and Definitions
- Implementation
  - As adjacency matrix
  - As adjacency list
- Depth-First Search
  - Implement with stack
- Breadth-First Search
  - Implement with queue
  - Optimal in unweighted graph
- Dijkstra's Algorithm
  - Optimal for weighted graphs (in an upcoming lecture)