Lecture 18
Graphs and Graph Algorithms

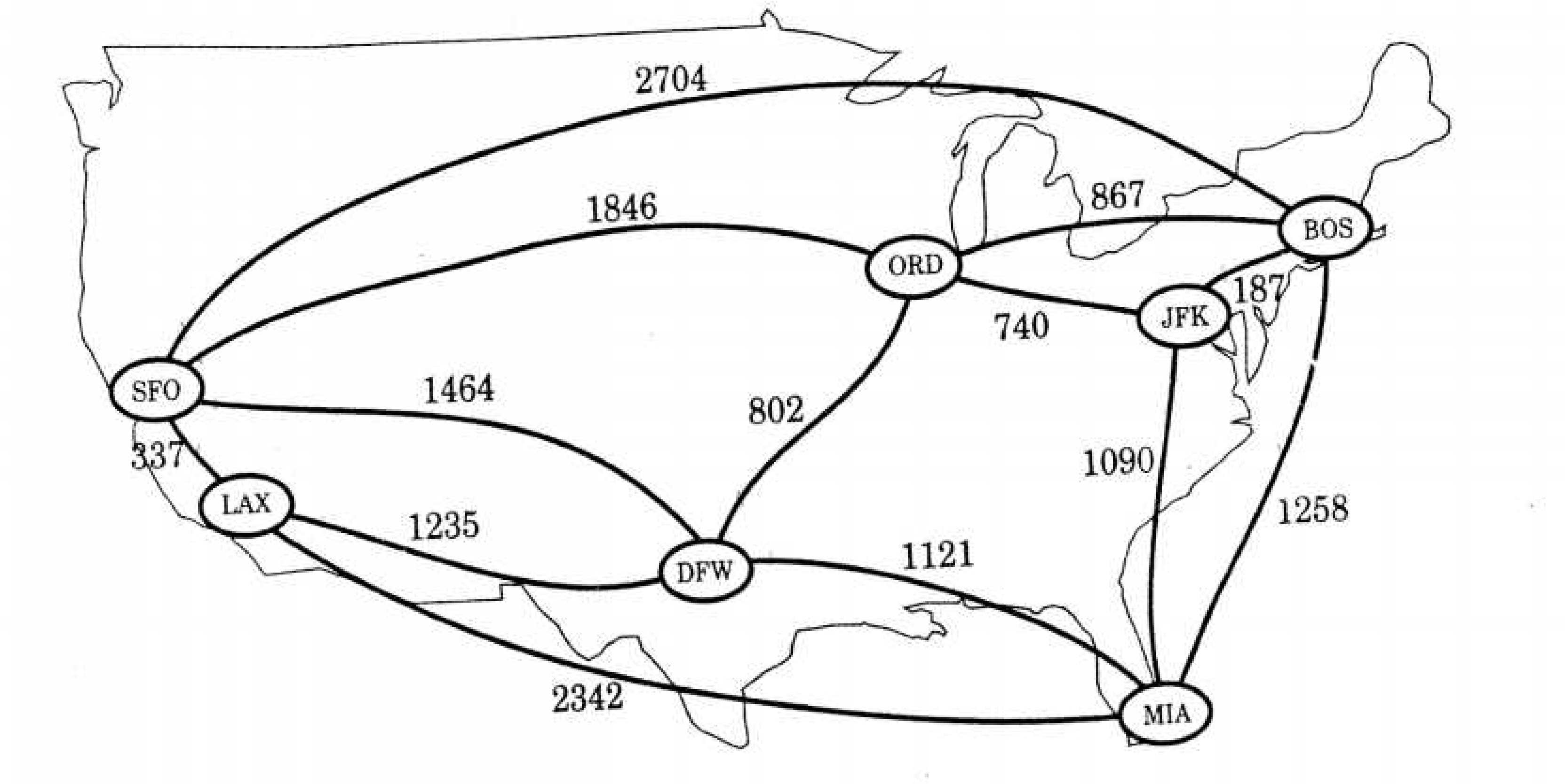
EECS 281: Data Structures & Algorithms

A Map and Some Crayons

A Board Game

http://boardgames.about.com/od/tickettoride/ig/Ticket-to-Ride/USA-Map.htm

Airline Routes



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 can be thought of as tuples of vertices. That is $e_m = (v_s, v_t)$

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 graph is *simple* unless otherwise specified

Graphs: Complexity

- Complexity of graph algorithms is typically defined in terms of:
  - Number of edges $|E|$, or
  - Number of vertices $|V|$, or
  - Both

Graphs: Data Structures

- Sparse Graph
  - few edges ($|E| << |V^2|$) or ($|E| \approx |V|$)
  - represent as adjacency list
- Dense Graph
  - many edges ($|E| \approx |V^2|$)
  - represent as adjacency matrix

Adjacency Matrix

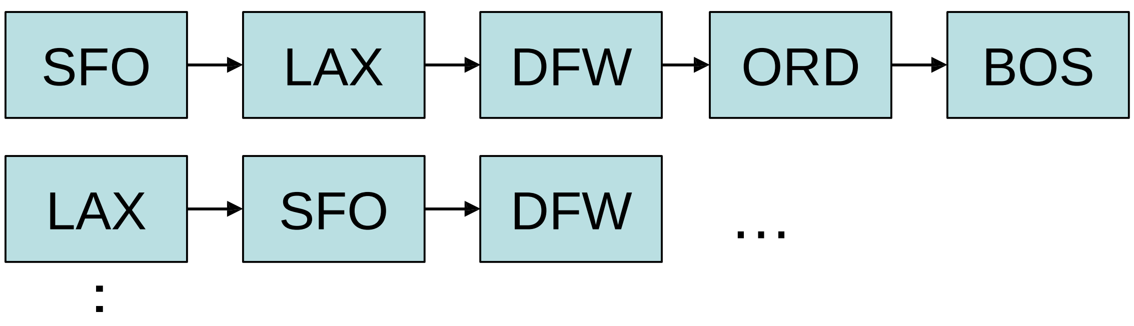
	SFO	LAX	DFW	ORD	MIA	JFK	BOS
SFO	0	1	1	1	0	0	1
LAX	1	0	1	0	1	0	0
DFW	1	1	0	1	1	0	0
ORD	1	0	1	0	0	1	1
MIA	0	1	1	0	0	1	1
JFK	0	0	0	1	1	0	1
BOS	1	0	0	1	1	1	0

Distance Matrix

	SFO	LAX	DFW	ORD	MIA	JFK	BOS
SFO	0	337	1464	1846	0	0	2704
LAX	337	0	1235	0	2342	0	0
DFW	1464	1235	0	802	1121	0	0
ORD	1846	0	802	0	0	740	867
MIA	0	2342	1121	0	0	1090	1258
JFK	0	0	0	740	1090	0	187
BOS	2704	0	0	867	1258	187	0

Adjacency List

	SFO	LAX	DFW	ORD	MIA	JFK	BOS
SFO	0	337	1464	1846	0	0	2704
LAX	337	0	1235	0	2342	0	0
DFW	1464	1235	0	802	1121	0	0
ORD	1846	0	802	0	0	740	867
MIA	0	2342	1121	0	0	1090	1258
JFK	0	0	0	740	1090	0	187
BOS	2704	0	0	867	1258	187	0



Note: Adjacency list nodes can contain distances

Graphs: Directed vs Undirected

- Directed Graph (aka digraph)
  - Edges have direction
  - Nodes on edges form ordered pairs
    - Order of vertices in edge is important
    - $e_n = (u, v)$ means there is an edge from u to 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 between u and v

Graphs: Weighted Graphs

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

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: Data Structures

Adjacency Matrix Implementation

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

Graphs: Data Structures

Adjacency List Implementation

- Complexity determined as follows:
  - Edges are distributed on vertices (E/V)
  - Costs 1 to access a vertex list
  - Average cost for individual vertex is $O(1 + E/V)$
  - Cost for all vertices is $O(V) \times O(1 + E/V) = O(V + E)$
- 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: NULL = no edge, <list_item> = edge
  - Weighted: NULL = no edge, <list_item_with_val> = edge

Graph Algorithm Questions

- Describe algorithm to determine all non-stop flights from JFK
- Give complexity for adjmat and adjlist

	SFO	LAX	DFW	ORD	MIA	JFK	BOS
SFO	0	337	1464	1846	0	0	2704
LAX	337	0	1235	0	2342	0	0
DFW	1464	1235	0	802	1121	0	0
ORD	1846	0	802	0	0	740	867
MIA	0	2342	1121	0	0	1090	1258
JFK	0	0	0	740	1090	0	187
BOS	2704	0	0	867	1258	187	0

Graph Algorithm Questions

- Describe algorithm to determine if any non-stop flights from JFK exist
- Give complexity

- Describe algorithm to determine greatest distance that can be flown from JFK on a non-stop flight
- Give complexity

Graph Algorithm Questions

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

Graph Algorithm Questions

- Describe an algorithm to determine the greatest distance for the least cost (ratio) that can be flown from JFK on any flight (non-stop or connecting)

- Give complexity

Depth-First Search

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

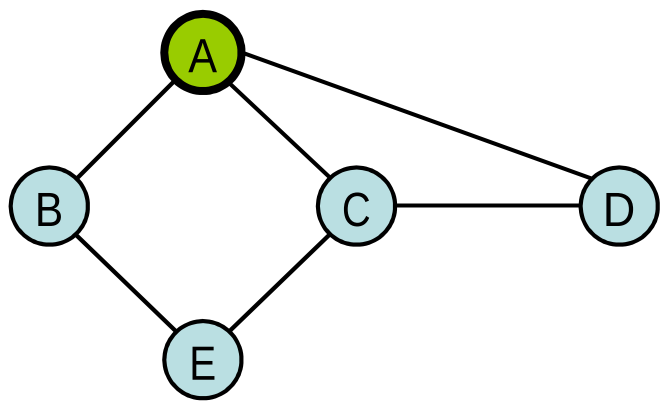
- Use a stack
- Algorithm works on graphs and digraphs
- Discovers ***a*** path from source s to goal g, if one exists

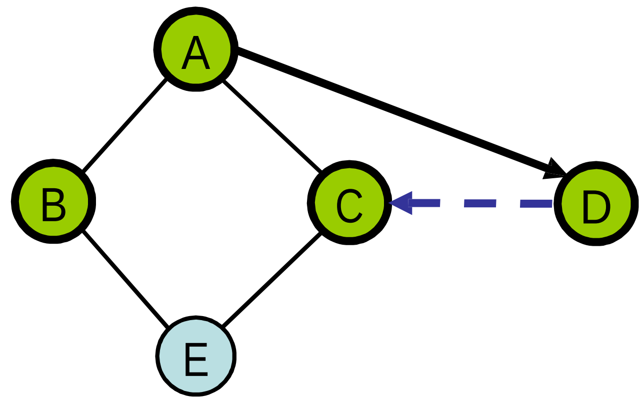
Depth-First Search

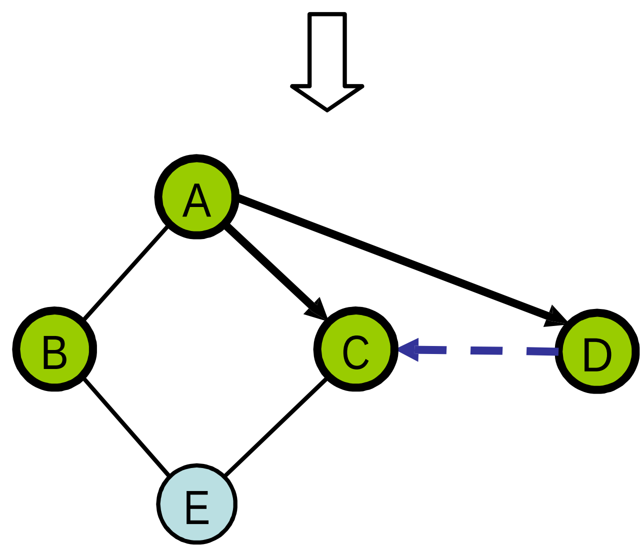
```
Algorithm GraphDFS
    Mark source as visited
    Push source to Stack
    While Stack is not empty
        Pop candidate from Stack
        If candidate is goal
            Return success
        Else
            For child of candidate
                If child is unvisited
                    Mark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

Example

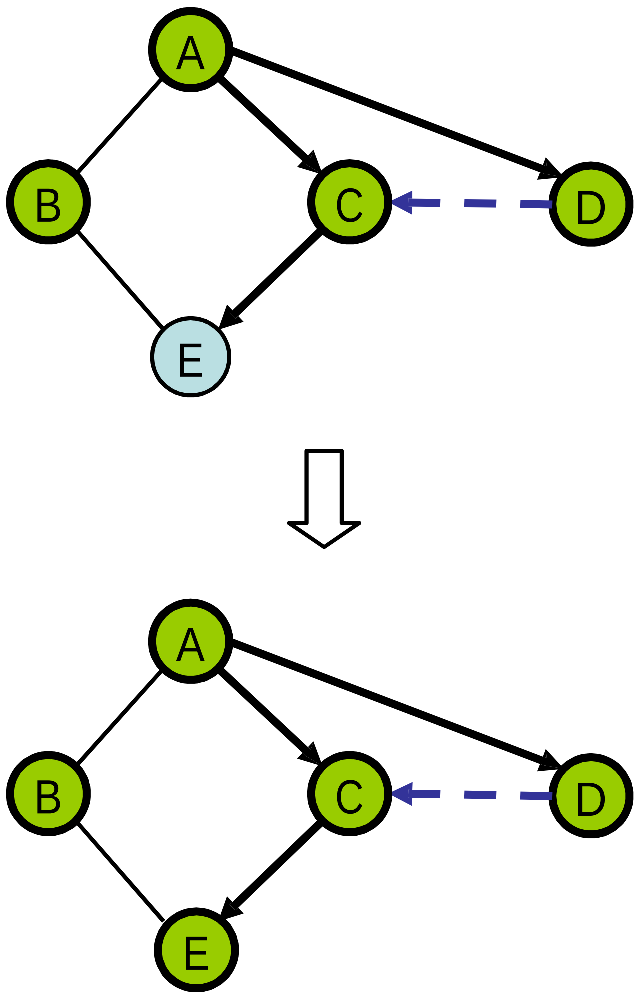
- unexplored vertex
- visited vertex
- unexplored edge
- discovery edge
- back edge







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:
  - 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

Breadth-First Search

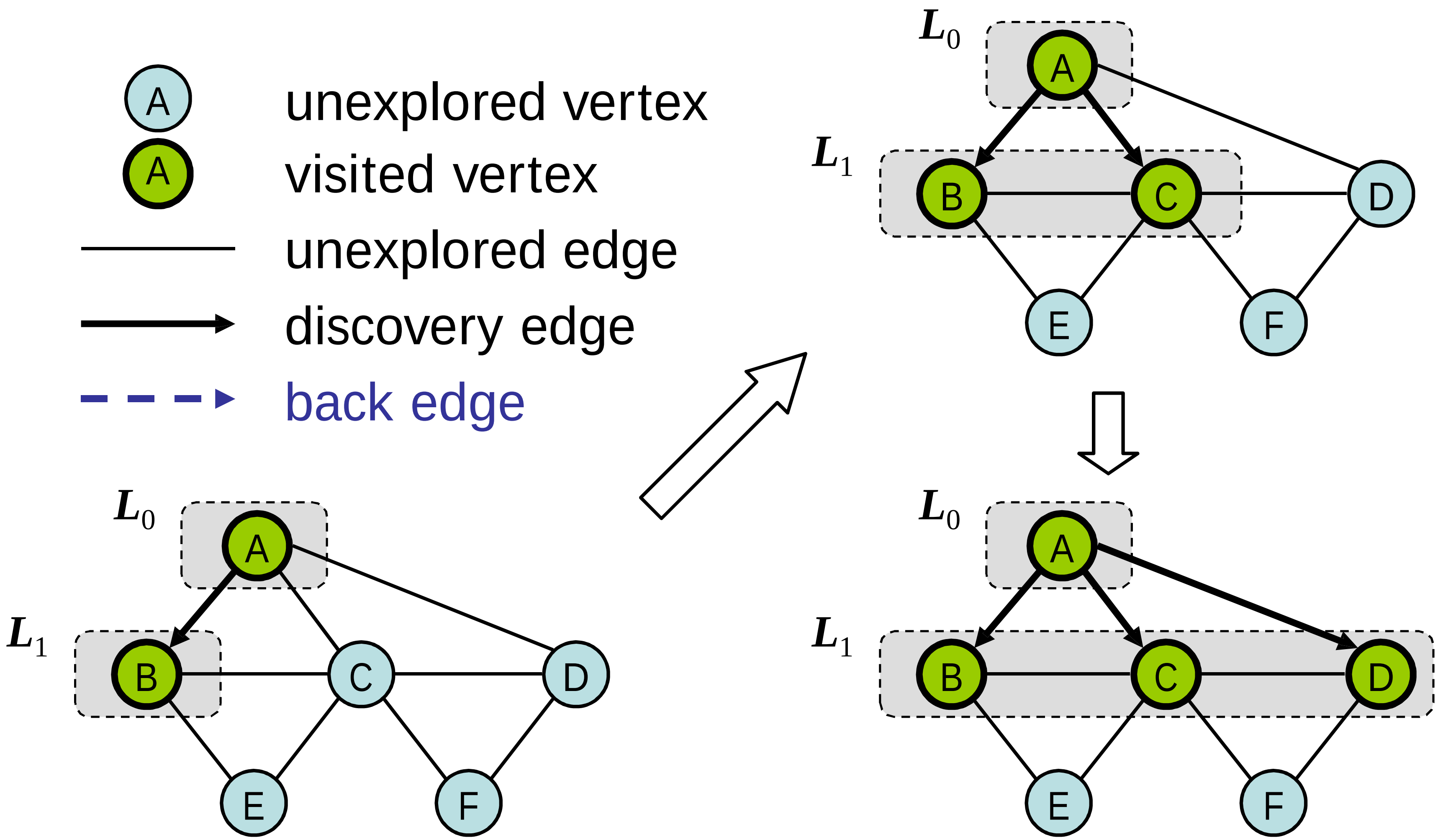
Given a graph $G = (V, E)$, systematically explore the edges of G to discover ***a*** shortest path from the source s to the goal g

- Use a queue
- Algorithm works on graphs and digraphs
- Discovers a ***shortest*** path from source s to goal g, if one exists
  - *Only works if all costs are equal*

Breadth-First Search

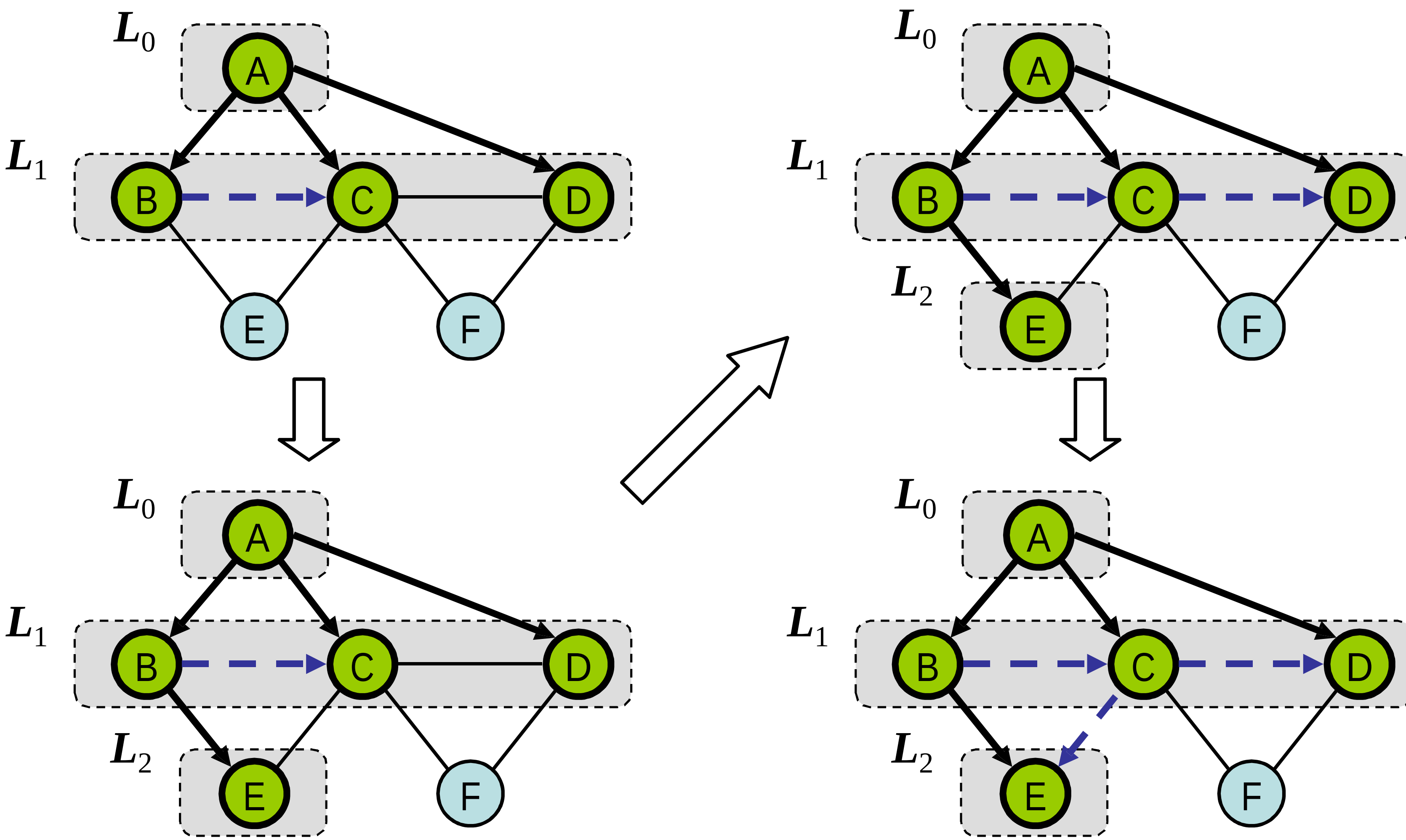
```
Algorithm GraphBFS
    Mark source as visited
    Add source to back of Queue
    While Queue is not empty
        Remove candidate from front of Queue
        If candidate is goal
            Return success
        Else
            For child of candidate
                If child is unvisited
                    Mark child visited
                    Add child to back of Queue
    Return failure
```

Example

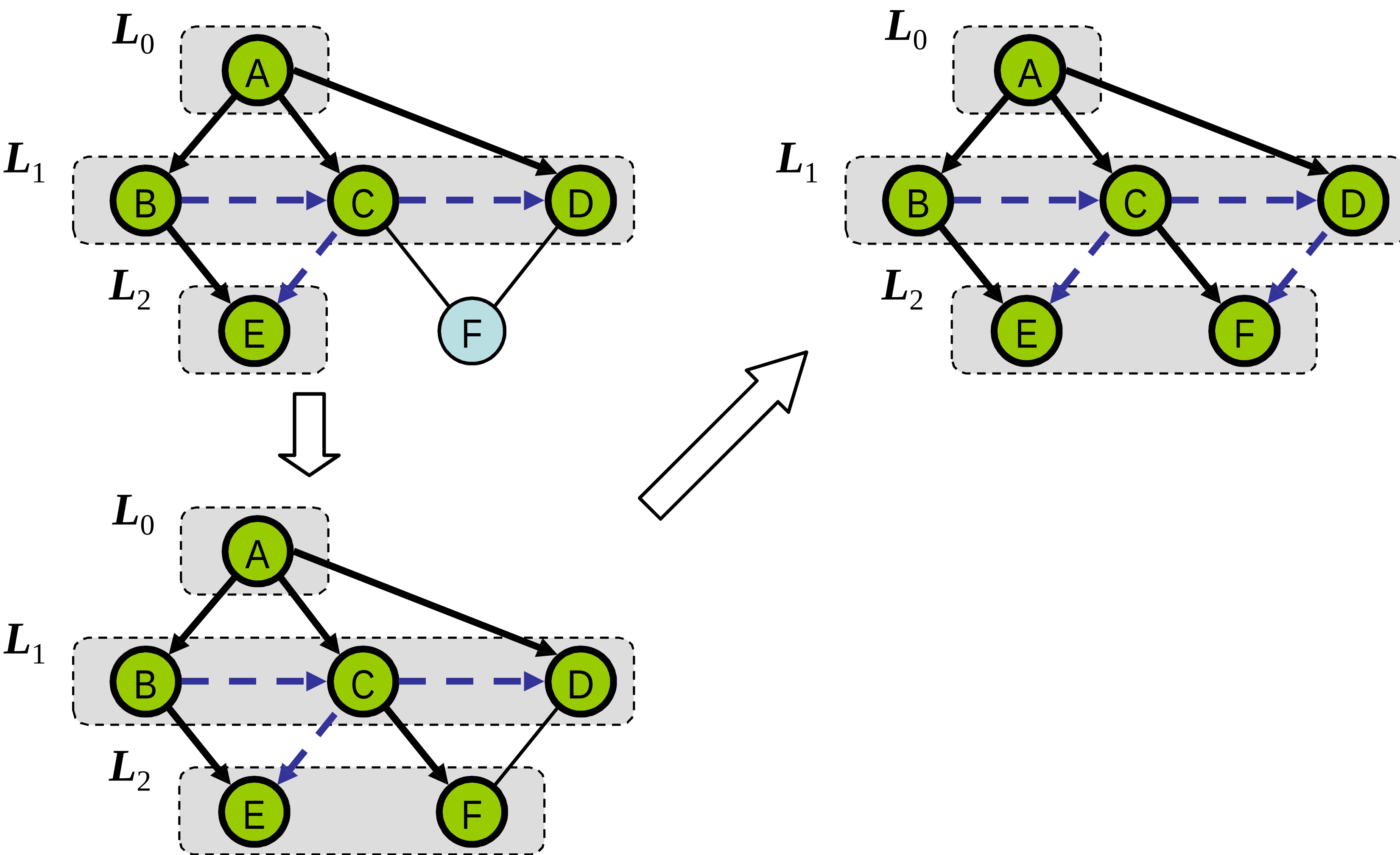


A
unexplored vertex
A
visited vertex
unexplored edge
discovery edge
back edge
L_0
L_1
A
B
C
D
E
F

Example (cont.)



Example (cont.)



BFS: Analysis of Adjacency List

- BFS:
  - 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

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

Graph Algorithm Questions

- Describe algorithm to determine if a path from JFK to SFO exists
- Give complexity

- Describe algorithm to determine minimal cost from JFK to SFO
- Give complexity

Graph Algorithm Questions

- Suppose you are planning a family reunion. Your family is spread out all over the US. You don't know where to have the reunion, but you want to minimize total travel cost. Describe the algorithm
- Give complexity

Graph Algorithm Questions

- Suppose numbers are cost of building high-speed rail (in $1000). Describe algorithm to determine least cost construction, such that any city can be reached from any other city
- Give complexity

Graph Summary

- Background and Definitions
- Implementation
  - As adjacency matrix
  - As adjacency list
- Depth-First Search
  - Implementation with stack
- Breadth-First Search
  - Implementation with queue