School of
Information Systems

SMU
SINGAPORE MANAGEMENT UNIVERSITY

COR-IS1702
COMPUTATIONAL THINKING
WEEK 9: GRAPHS

2020/21 Term 1

(08) Graphs Part 1a: Traversals

Video (30 mins): https://youtu.be/QMo7VhaCnsk

Road Map

Algorithm Design and Analysis

(Weeks 1 - 5)

Fundamental Data Structures

- Week 6: Linear data structures (stack, queue)
- Week 7: Hierarchical data structure (binary tree)
- This week → Week 9: Networked data structure (graph)

Computational Intractability and Heuristic Reasoning

(Weeks 10 – 13)

It's A Small World After All

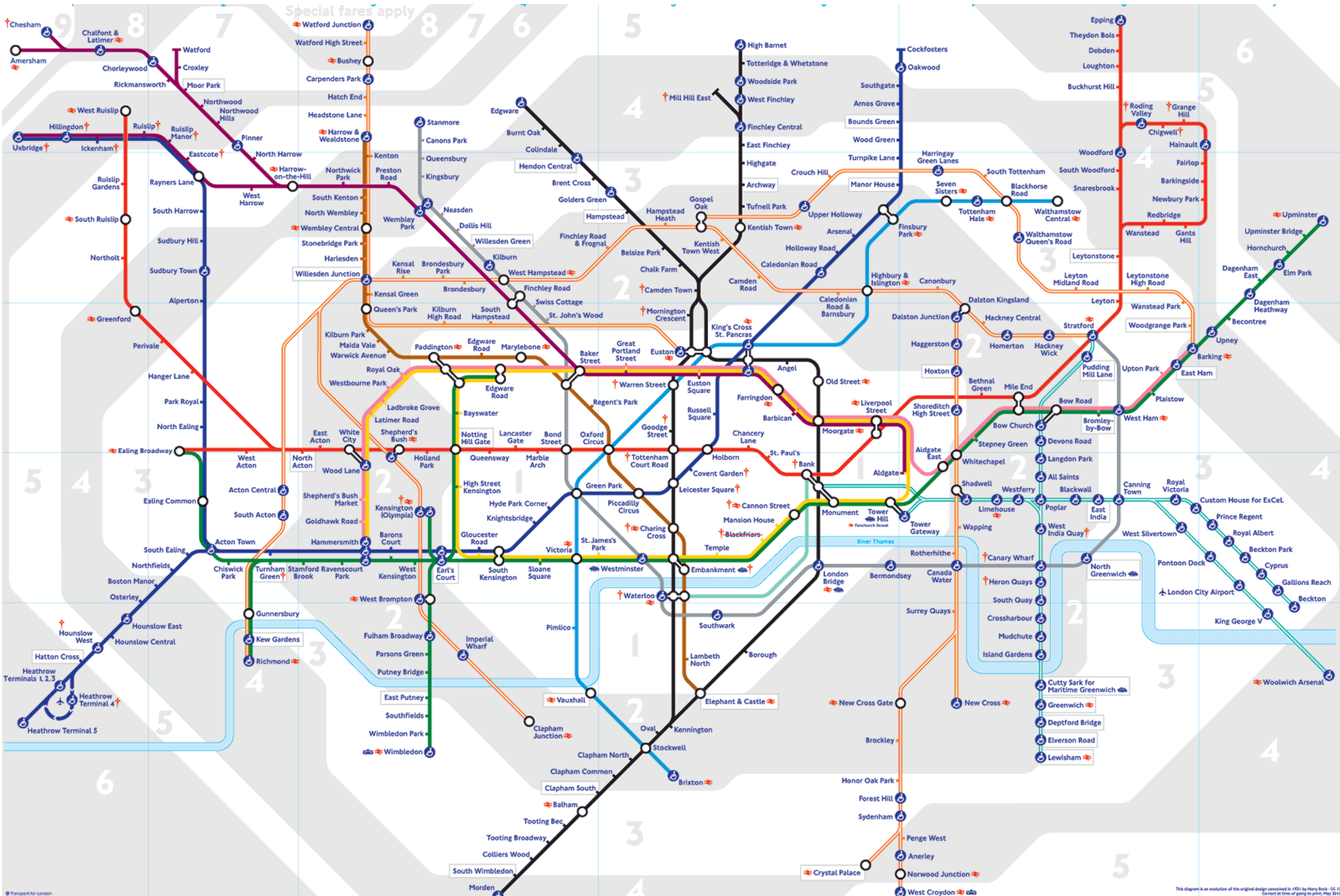
Modeling networks with graphs

- Graph
- Traversal
- Algorithms

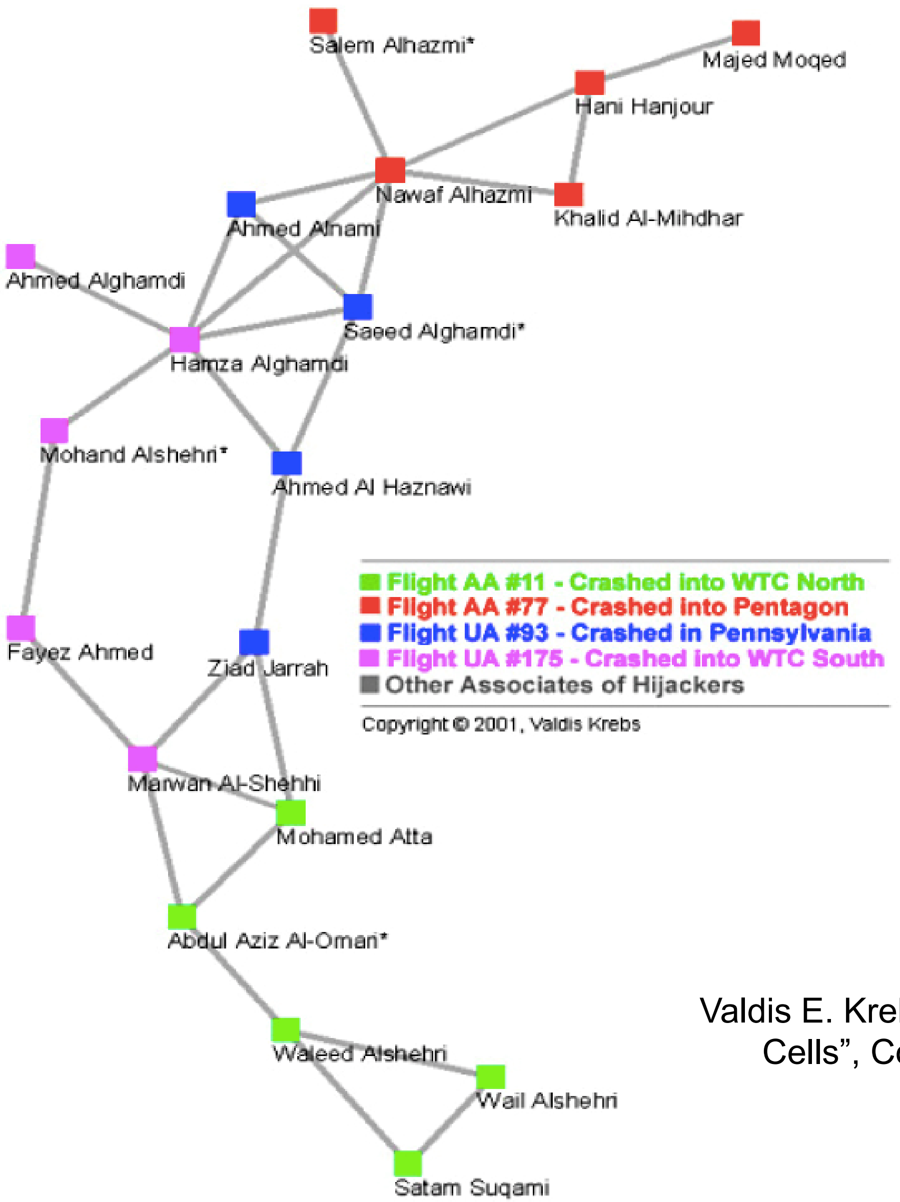
Reference

- Handout on Fundamental Data Structures Chapter 3
  - PDF copy downloadable from eLearn

Transportation Network (Land)

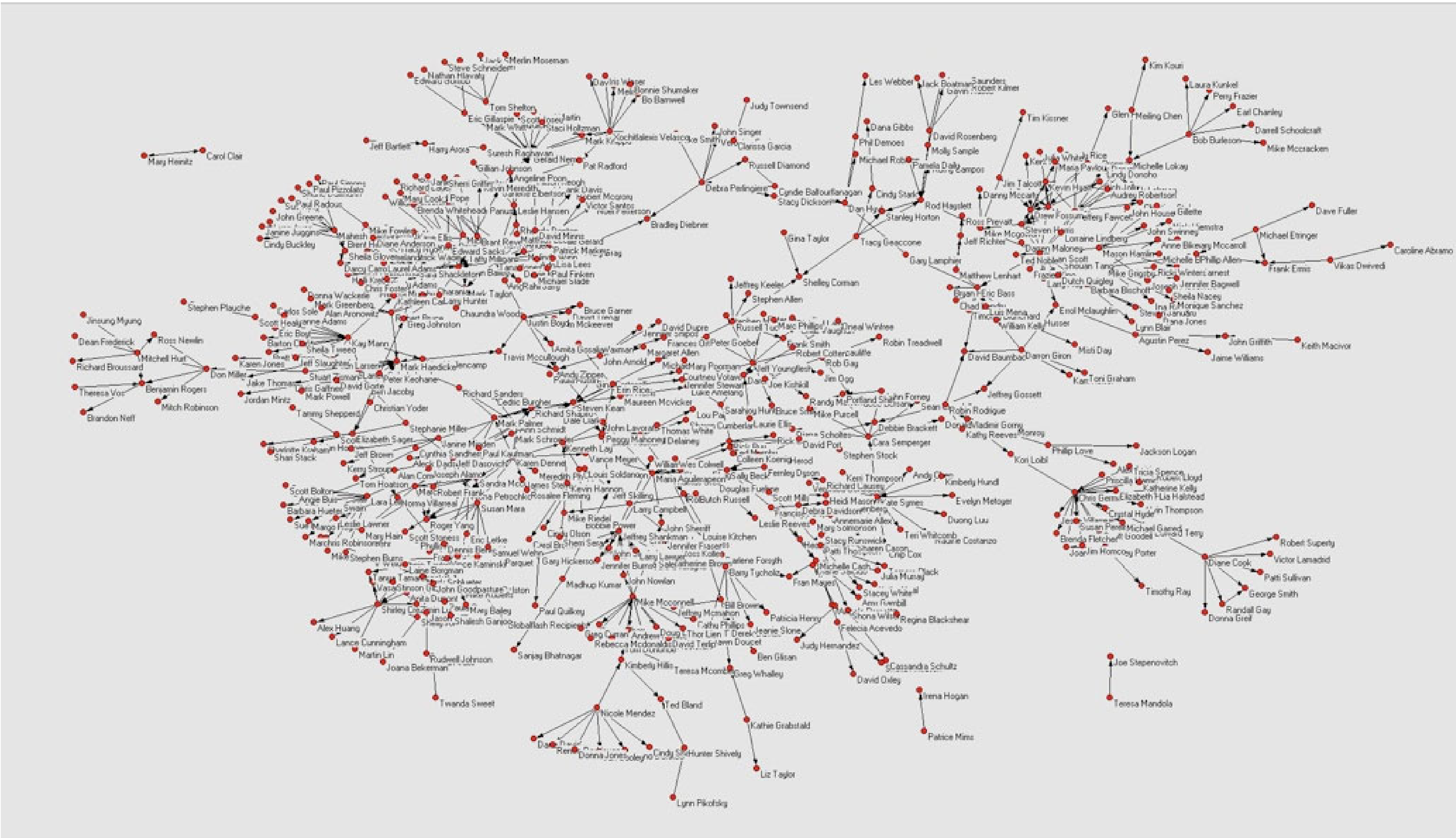


Social Network (9/11 Terrorists)



Valdis E. Krebs, “Mapping Networks of Terrorist Cells”, Connections 24(3): 43-52, 2002.

Social Network (Enron Emails)



http://www1.cs.columbia.edu/~sara/cs6998/enron11-29-00m10.jpg

Social Network (Hollywood Collaborations)

http://chestofbooks.com/computers/search/55-Ways-to-Have-Fun-With-Google/19-Kevin-Bacon-and-the-Google-Network.html

Milgram's Small World Experiment

- An experiment conducted by Dr. Stanley Milgram, an American psychologist
- The objective of the experiment is to assess the connectivity of social network in the United States

Start: Omaha, Nebraska & Wichita, Kansas

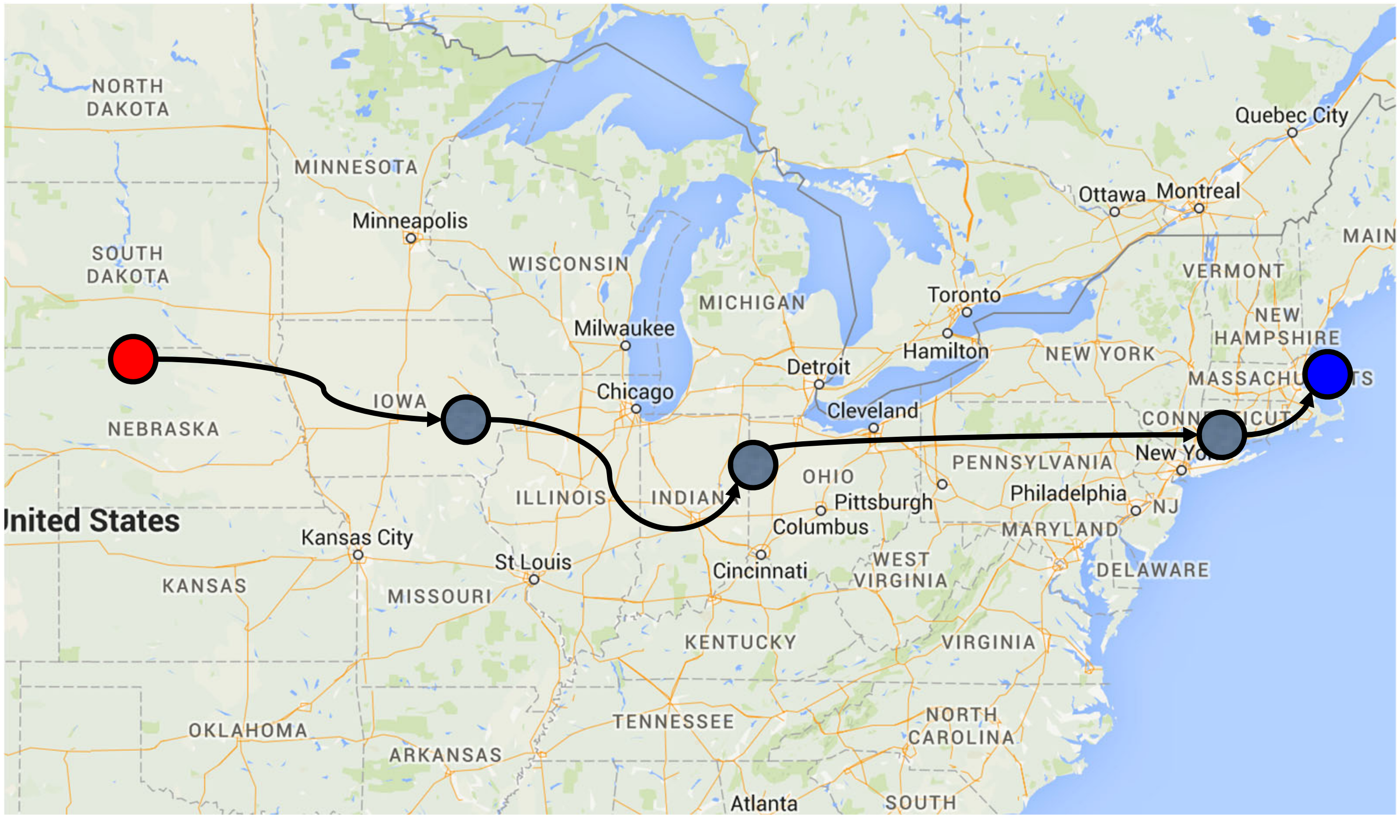
End: Boston, Massachusetts

Each recipient might:

- Send it directly to the target person if known personally
- Forward it to the next person deemed most likely to know the target person
- Do nothing

http://en.wikipedia.org/wiki/Stanley_milgram

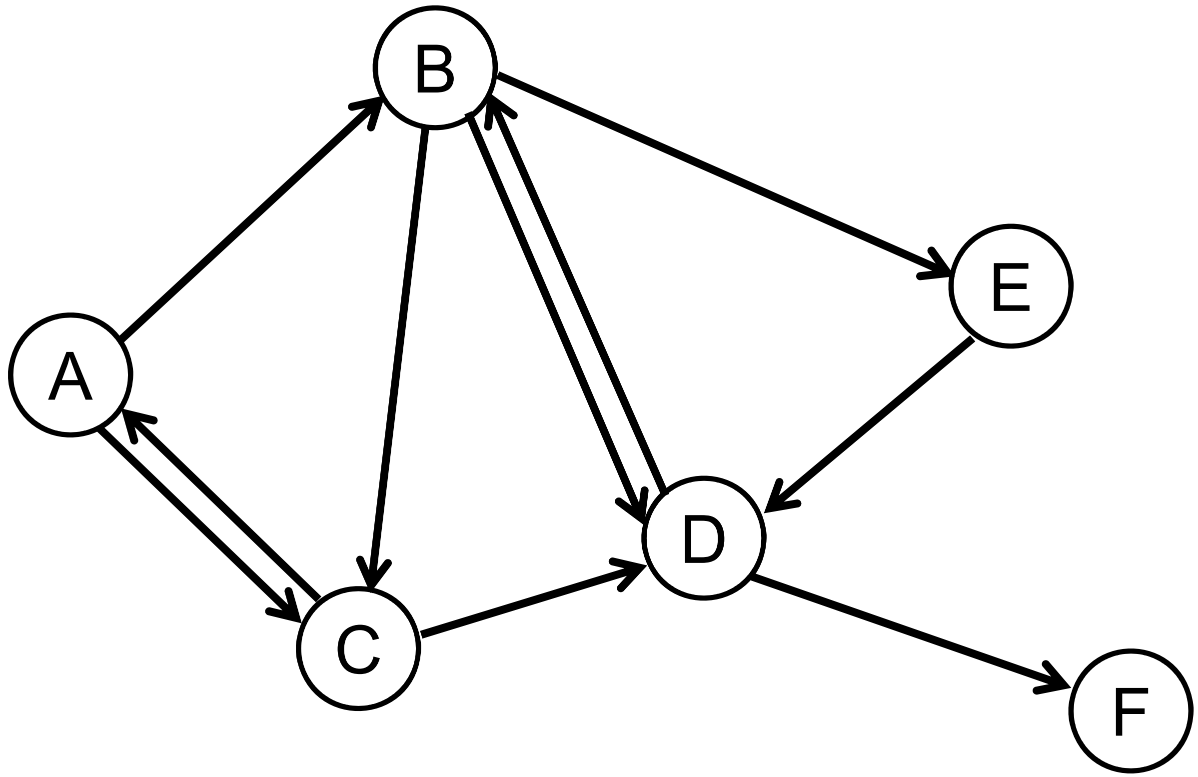
Milgram's Small World Experiment



- 64 / 296 succeeded, with average path length ~ 5 to 6
- ➔ "Six degrees of separation"

Stanley Milgram, "The Small-World Problem," Psychology Today, vol. 1, May 1967, pp 61-67.

Directed Graph



A graph `G` is associated with a set of vertices `V`, and a set of edges `E`.

Representation

Adjacency List

A	B	C	
B	C	D	E
C	D	A	
D	B	F	
E	D		
F			

Adjacency Matrix

to

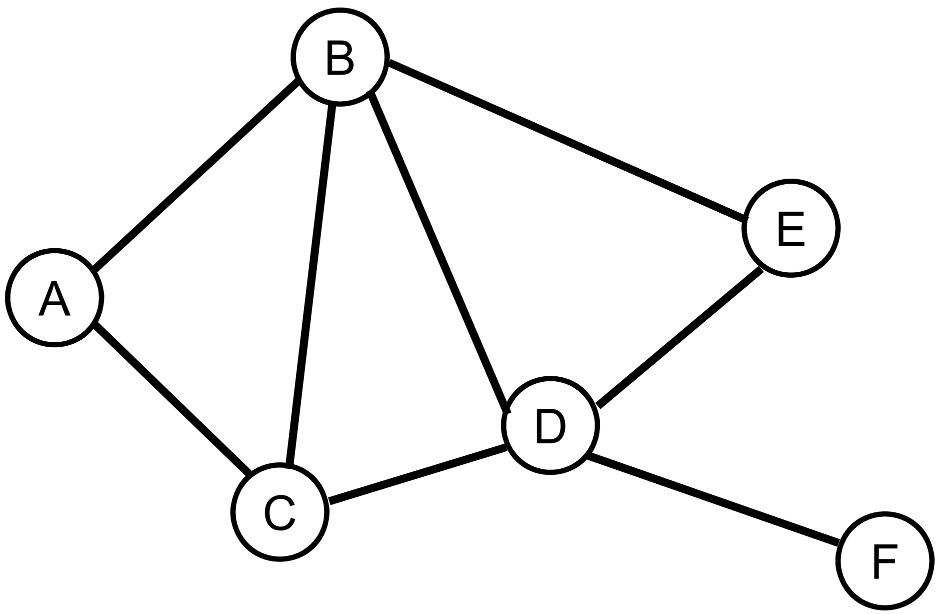
from

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

Connectivity

- In-degree of a vertex
  - number of incoming edges
- Out-degree of a vertex
  - number of outgoing edges
- Path
  - a sequence of vertices connected by edges
- Simple path
  - a path with a unique set of vertices
- Cycle
  - a path that begins and ends at the same vertex

Undirected Graph



Adjacency List

A	B	C		
B	A	C	D	E
C	D	A	B	
D	B	E	F	C
E	D	B		
F	D			

Adjacency Matrix

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

Operations

- `Graph` object
  - `Graph()`
  - `vertices`
  - `addVertex(v, x*, y*)`
    - `(x,y) coordinates are optional for visualization`
  - `deleteVertex(v)`
  - `addEdge(v1, v2, weight*)`
    - `weight (optional) is by default 1`
  - `deleteEdge(v1, v2)`
- `Vertex` object
  - `Vertex(label)`
  - `adjList` # array that contains the list of adjacent vertices
  - `visited` # True/False, whether the vertex has been visited

Operations (cont.)

- `UndirectedGraph` object
  - `UndirectedGraph()`

- `view_graph(graph, show_weights*)`

Traversals of a Graph

- "Walk through" the graph by visiting each vertex that can be reached from the starting vertex.
- To find a sequence of actions leading to a goal state.

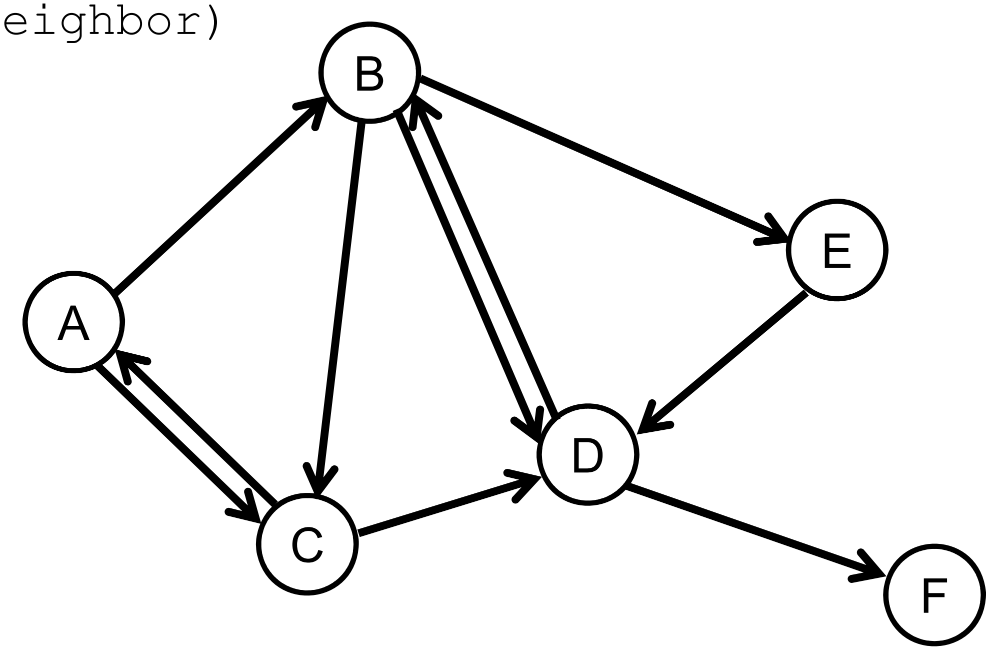
Two strategies:

- Depth-first search (DFS)
  - Prioritizes edges along a path.
- Breadth-first search (BFS)
  - Prioritizes edges closest to the starting vertex.

DFS (Recursive version)

```
def dfs_traversal(vertex):
  visit(vertex)
  for i in range(len(vertex.adjList)):
    neighbor = vertex.adjList[i]
    if not neighbor.isVisited():
      dfs_traversal(neighbor)
```

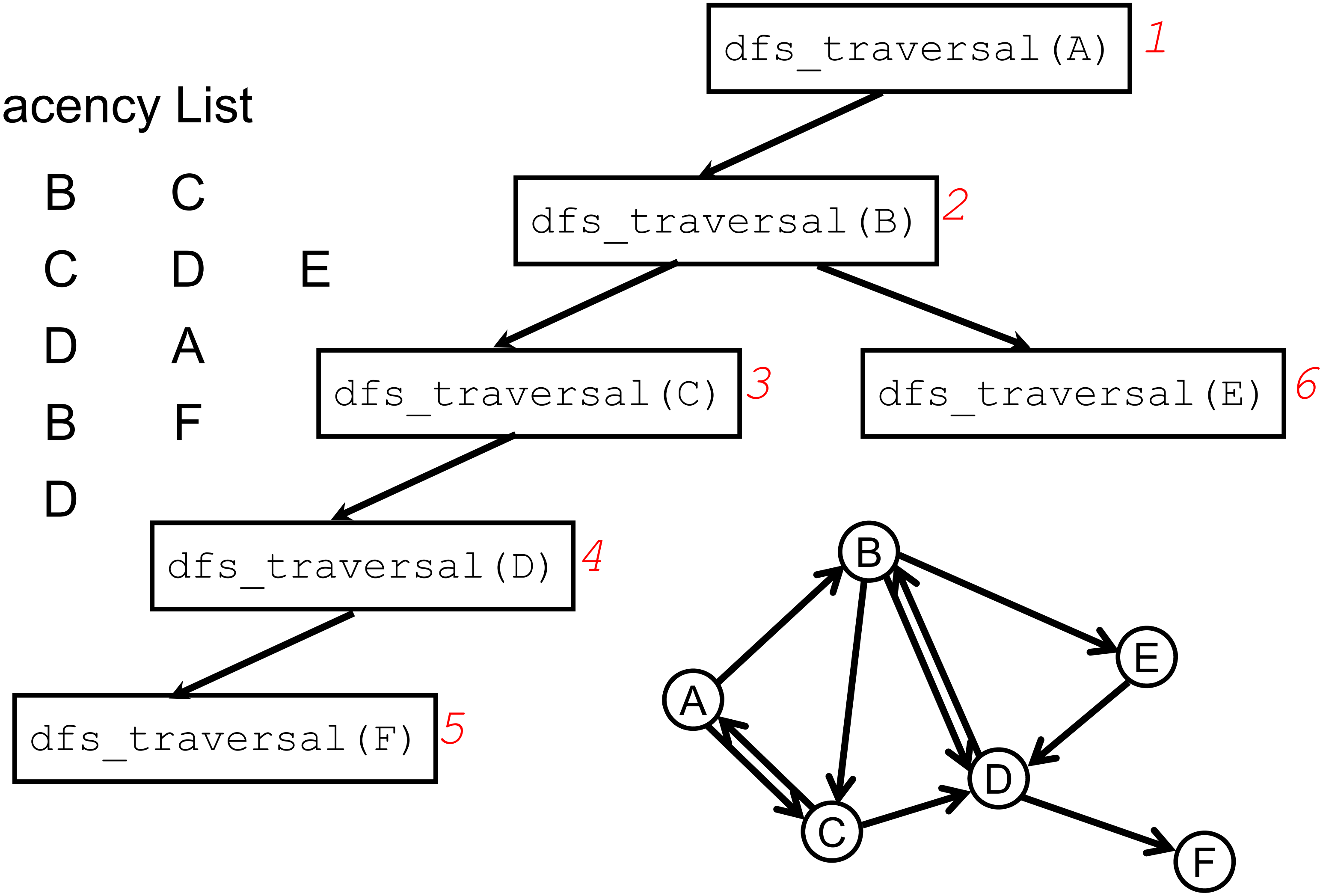
```
def visit(vertex):
  vertex.visited = True
  print(vertex)
```



DFS (Recursive version)

Adjacency List

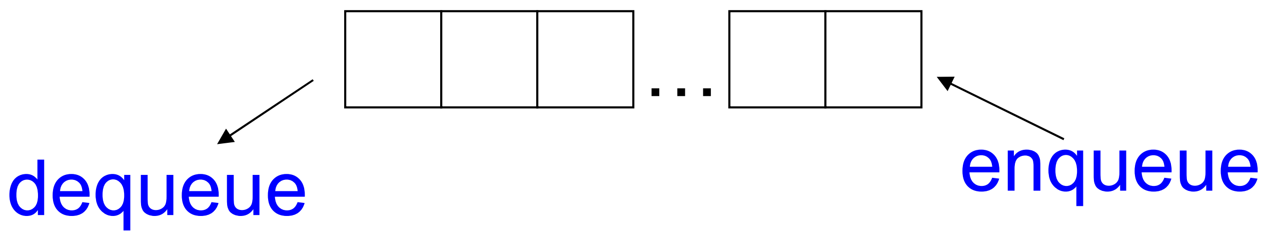
A	B	C	
B	C	D	E
C	D	A	
D	B	F	
E	D		
F			



Animation: http://student.seas.gwu.edu/~idsv/idsv.html

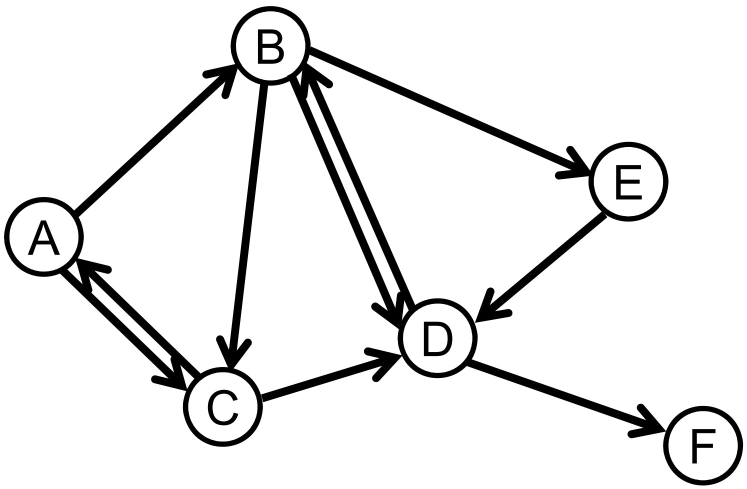
Breadth-First Search (BFS)

- List of candidate nodes stored in a **queue**: first-in-first-out (FIFO).
  - Get operation returns **oldest** item.



- Always expands **closest** node.

Breadth-First Search (BFS)



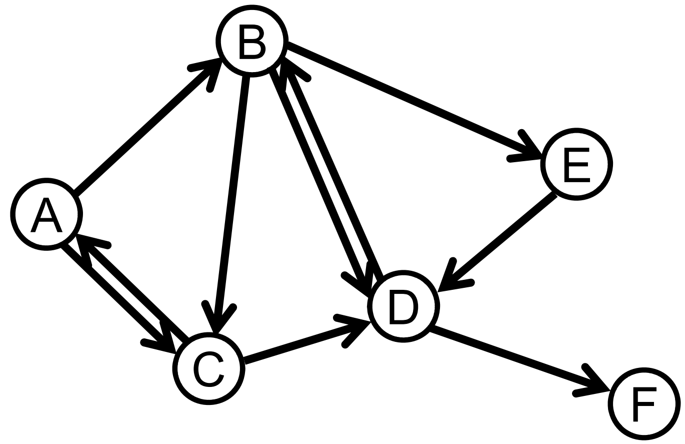

```
def bfs_traversal(vertex):
  q = Queue()
  visit(vertex)
  q.enqueue(vertex)
  while !q.isEmpty():
    v = q.dequeue()

      for i in range(len(v.adjList))
        neighbor = v.adjList[i]

        if not neighbor.isVisited():
          visit(neighbor)
          q.enqueue(neighbor)
```

for each unvisited neighbour of current vertex:
visit and enqueue it

Example: BFS



Adjacency List

A	B	C	
B	C	D	E
C	D	A	
D	B	F	
E	D		
F			

step	dequeued	visited/enqueued	queue: [H..T]
0		A	[A]
1	A	B, C	[B, C]
2	B	D, E	[C, D, E]
3	C		[D, E]
4	D	F	[E, F]
5	E		[F]
6	F		[ ]