Introduction to Graphs

Dr. Robert Amelard
(adapted from Dr. Igor Ivkovic)

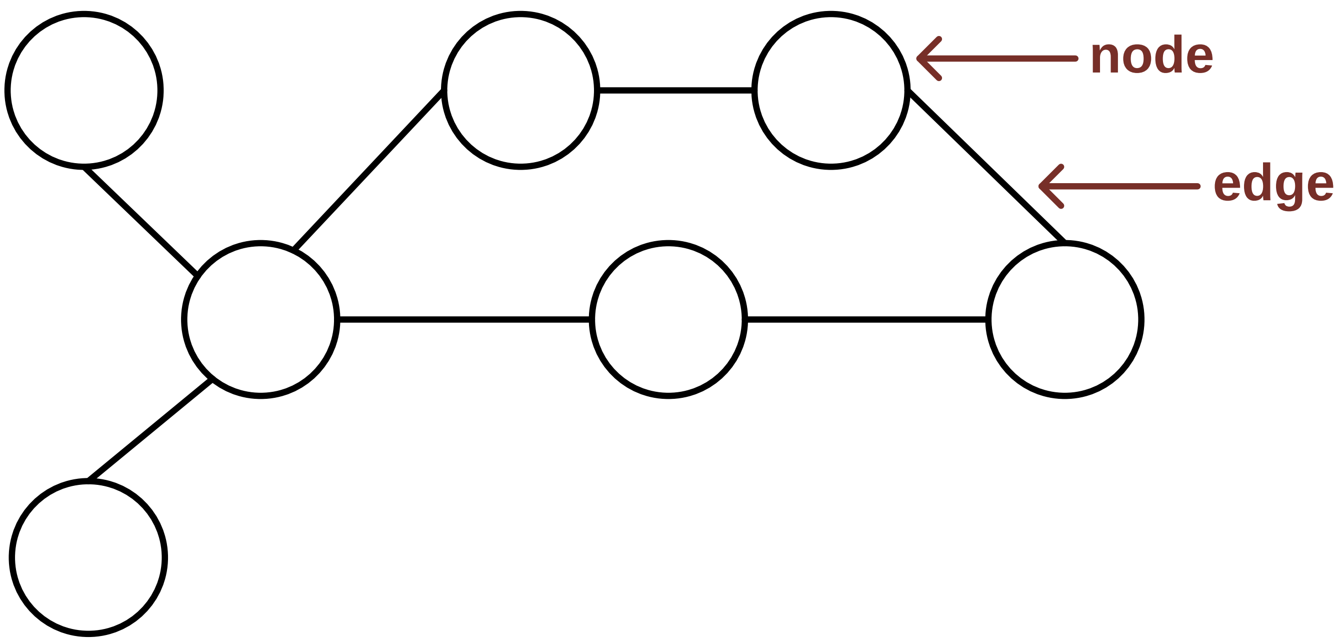
ramelard@uwaterloo.ca

Objectives

- Graph Structure
- Graph Cycle
- Degree of Node
- Graph Representation
- Graph Traversal
- Path Finding in Graphs

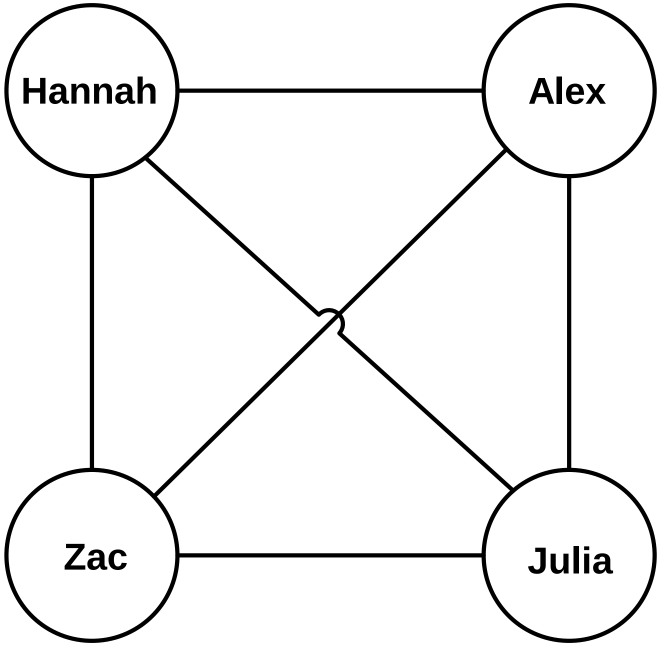
Introduction to Graphs /1

- **Graph:**
  - A collection of **nodes** (vertices) connected by **edges**
  - A node represents an entity that is being modeled within the graph
  - An edge represents a pair-wise relationship between two nodes in a graph



Introduction to Graphs /2

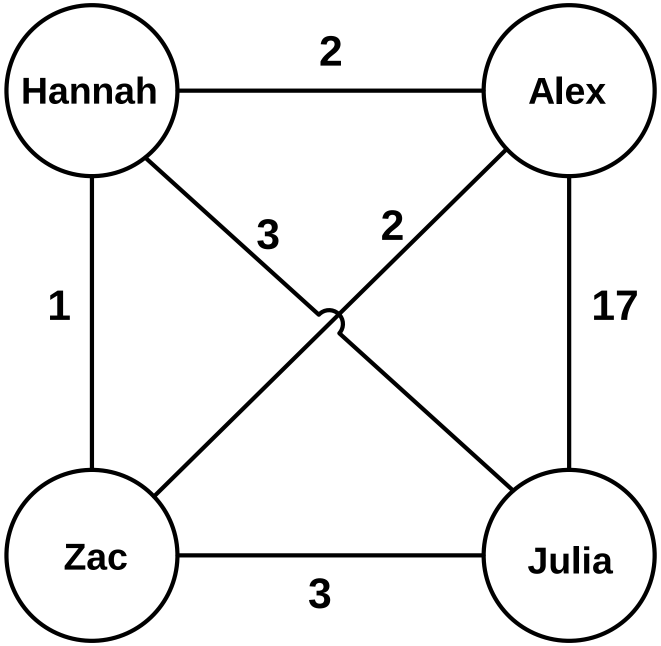
- **Graph Use Example:**
  - Arranging communication among four team members
  - Each team member represents a node
  - Communication between any two members is an edge



Introduction to Graphs /3

- **Edges can have inherent weights**
  - Example: resistance/difficulty of getting from A to B

Hannah & Zac are best friends



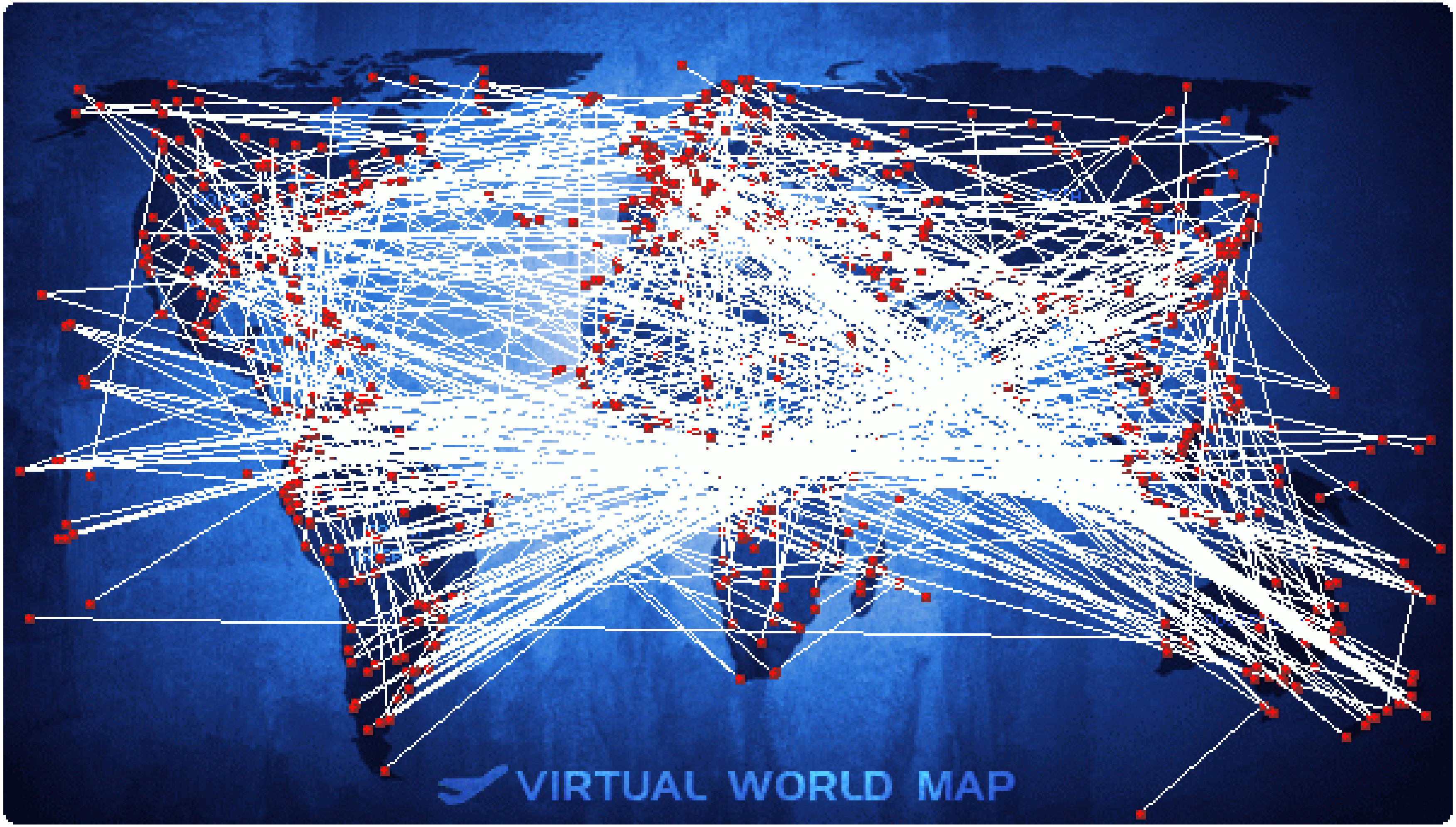
Alex and Julia got in a fight...

Introduction to Graphs /4

- **Graphs can be used to model complex real-world systems, such as the systems used in**
  - Google Maps
  - Robotic navigation
  - Network design
  - Social networks

- **For example, graphs are used to model flight connections among the airports around the world**
  - The nodes represent individual airports
  - The edges represent the valid flight connections between the airports (weight: duration, price, etc.)
  - The graph structures are used to plan optimal routes between destinations (e.g., Toronto to Sydney, Australia)

Introduction to Graphs /5

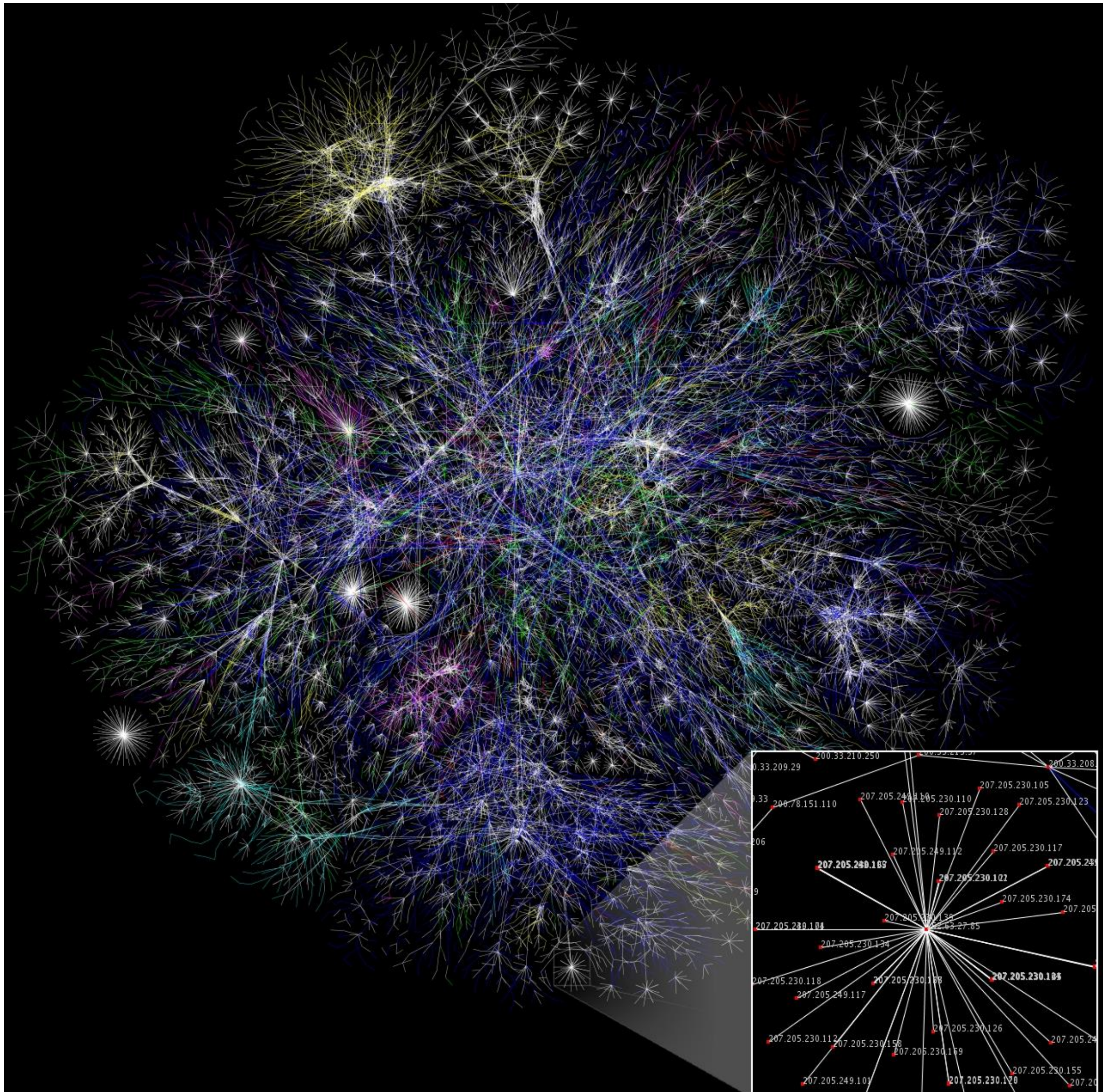


Flight Destinations, Source: http://www.airliners.net

Introduction to Graphs /6

- **Another example of graph use is the Internet, and modeling of the underlying infrastructure**
  - The nodes are different routers, servers, and other networked devices
  - The edges represent valid network connections between the devices (**weight: latency**)
  - The graph structures are used to discover optimal routes between destinations, and to maintain network performance

Introduction to Graphs /7

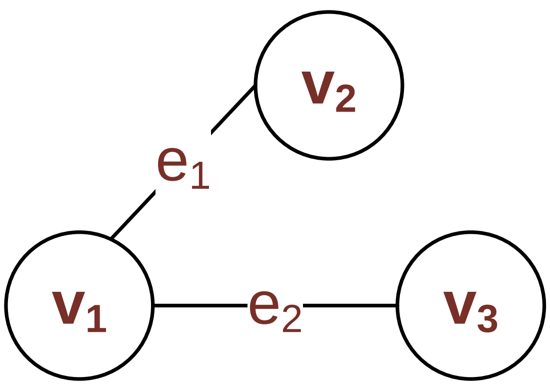


The Internet Backbone, Source: WikiMedia Commons

Introduction to Graphs /8

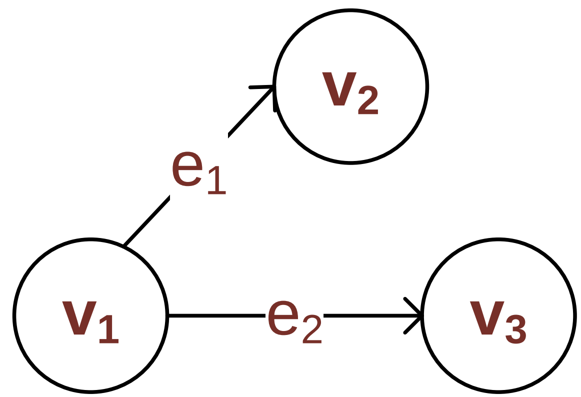
- **Formal Definition of a Graph G:**
  - A graph G can be defined as: $G = (V, E)$, where $v \in V$ is the set of nodes (vertices), and $e \in E$ is the set of edges
  - An edge $e(v_i, v_j)$ represents an edge from v_i to v_j

- **Undirected Graph:**
  No direction on edges
  - $e(v_1, v_2) = e(v_2, v_1)$



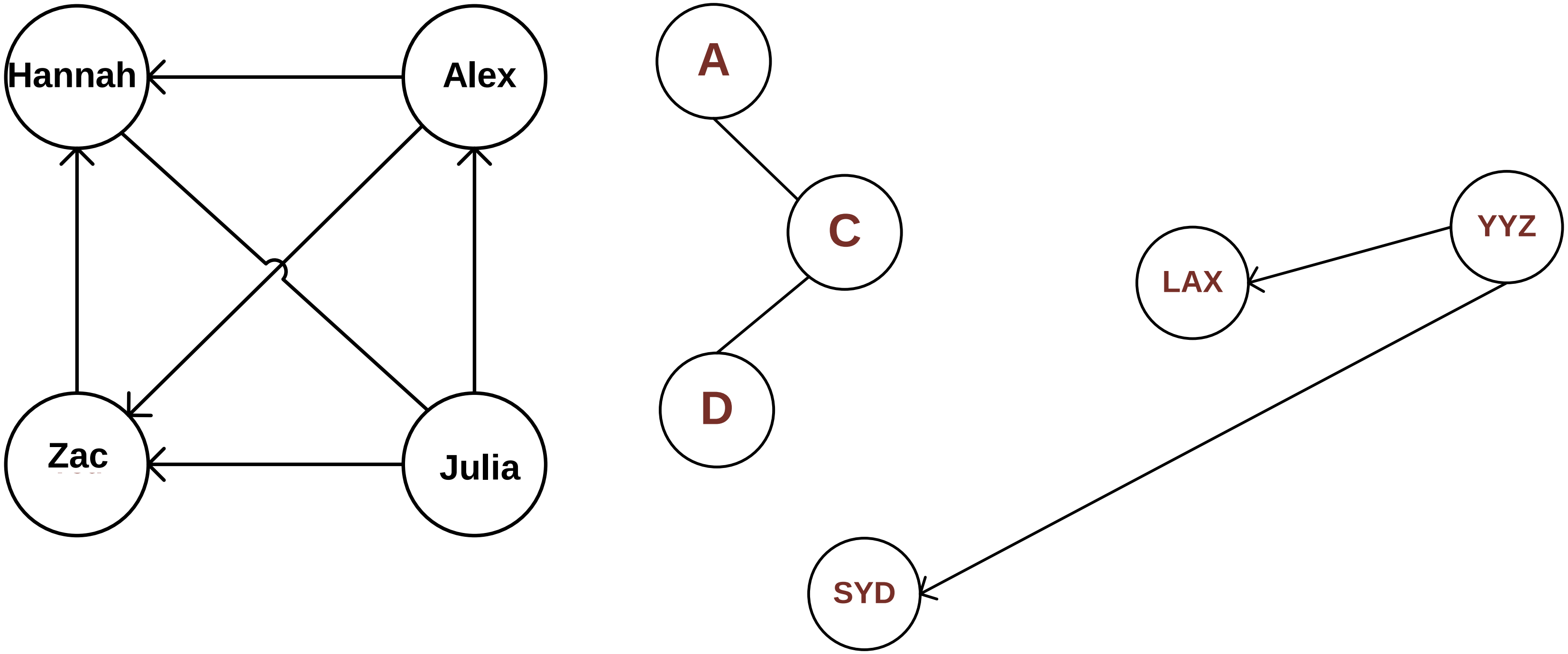
Directed Graph:
Edges are directed

$e(v_1, v_2) \neq e(v_2, v_1)$



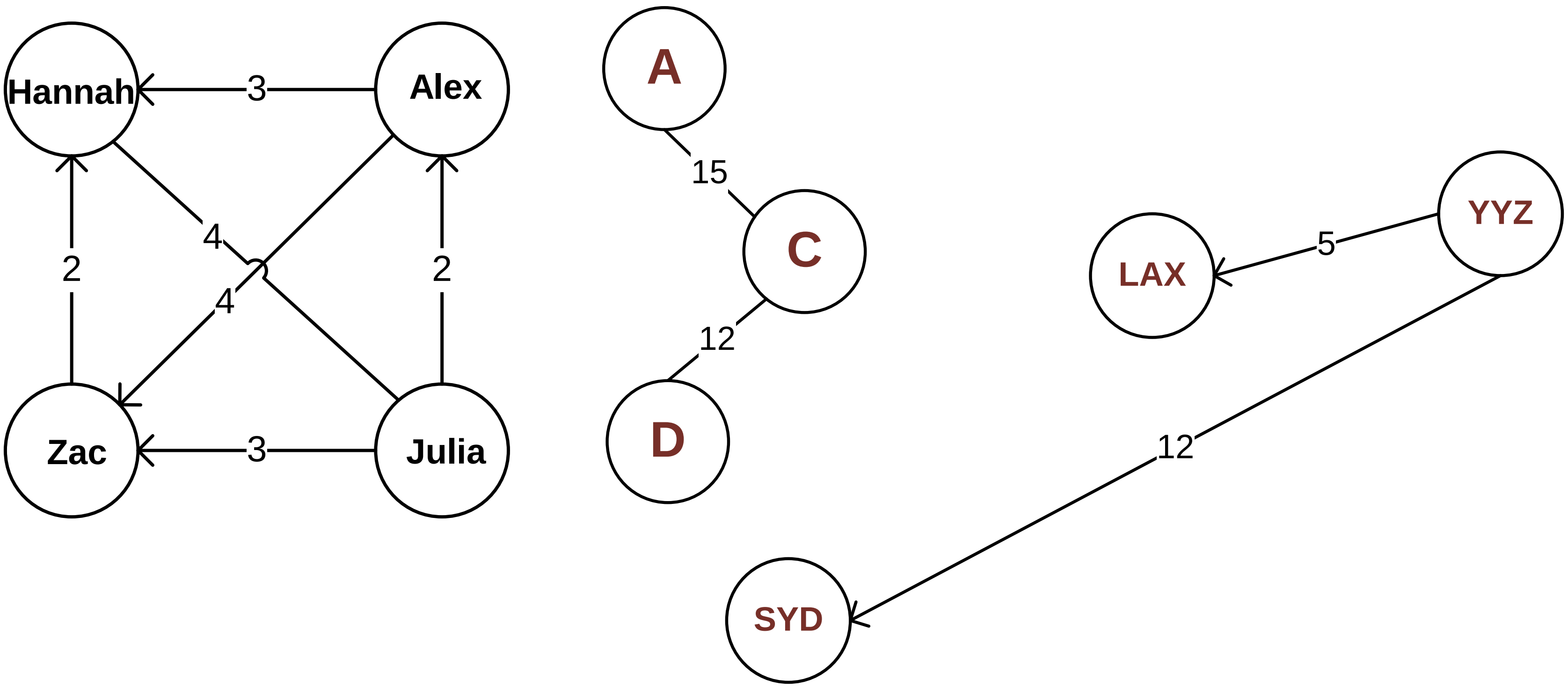
Introduction to Graphs /9

- **Labeled Graph:**
  - A graph where each node has a unique symbolic label associated with it
  - A labeled graph can be directed or undirected



Introduction to Graphs /10

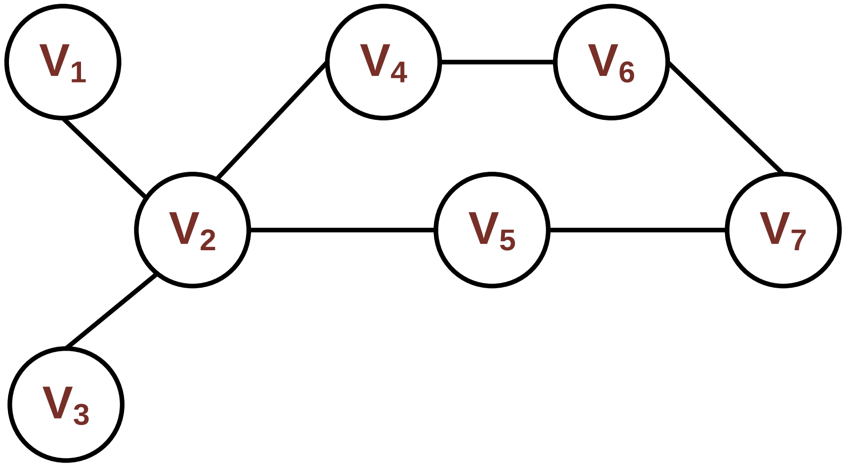
- **Weighted Graph:**
  - A graph where each edge has a numerical value associated with it
  - Formally, $w(v_i, v_j)$ is the weight for an edge $e(v_i, v_j)$



Introduction to Graphs /11

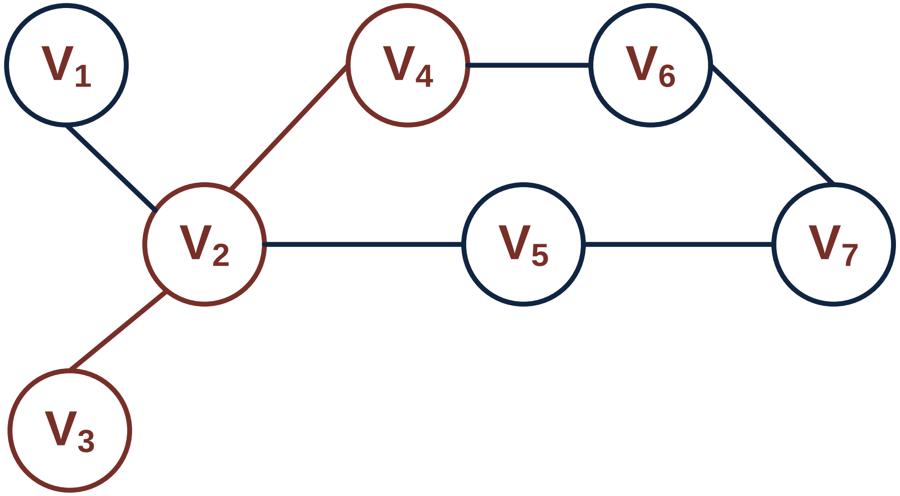
- **Graph Connectivity:**
  - A path is a sequence of edges that connects a sequence of graph nodes
  - The length of the path between two nodes is measured as the number of edges that constitute the path
  - A subset of nodes of a graph is considered connected if there is a path between every two nodes of the subset

- **Connected Graph Example:**



Introduction to Graphs /12

- **Another Connected Graph Example:**
  - The length of the path from v_2 to v_7 via v_5 is 2

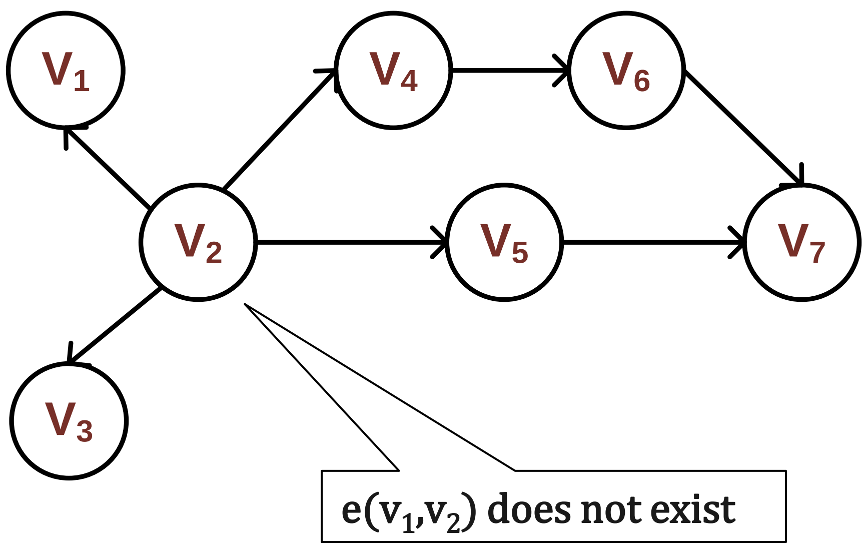


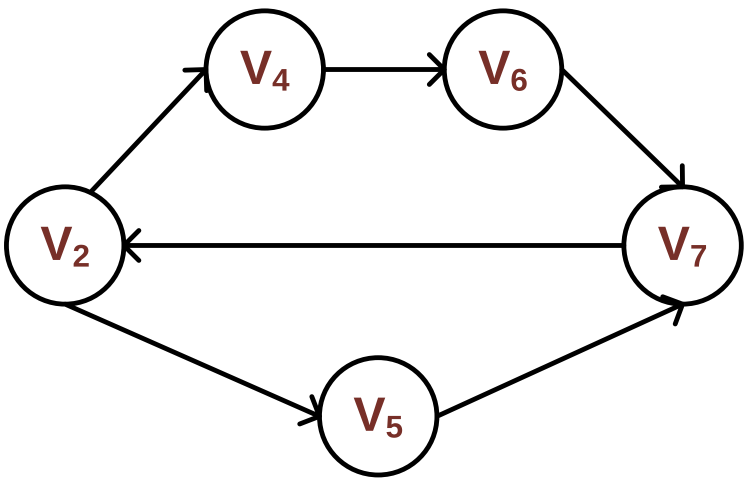
  - Subgraph $G_1 = (\{v_2, v_3, v_4\}, \{e(v_2, v_3), e(v_2, v_4)\})$ is also connected
  - Subgraph $G_2 = (\{v_5, v_6\}, \{\})$ is not connected

Introduction to Graphs /13

- **Strong vs. Weak Connectivity:**
  - A directed graph is strongly connected if each pair of nodes in the graph is connected (directly or indirectly) **“reachable”**
  - A directed graph is weakly connected if there exists a pair of nodes in the graph that is not connected

- **Weakly Connected:** **Strongly Connected:**

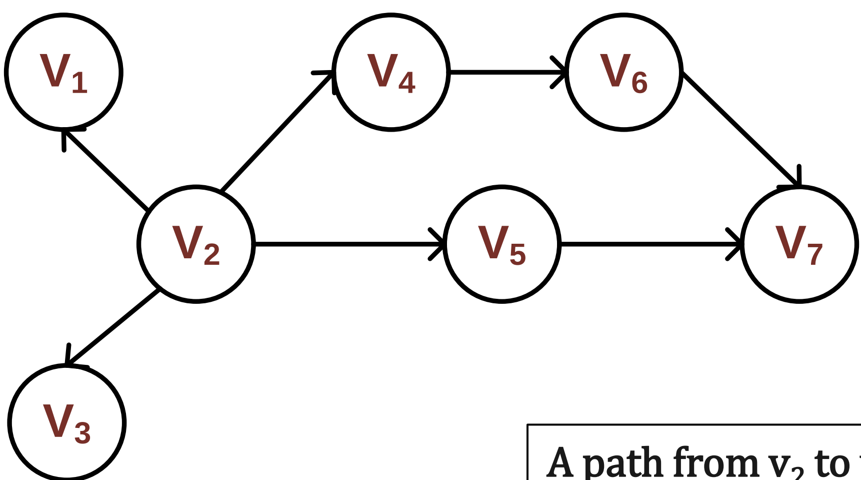




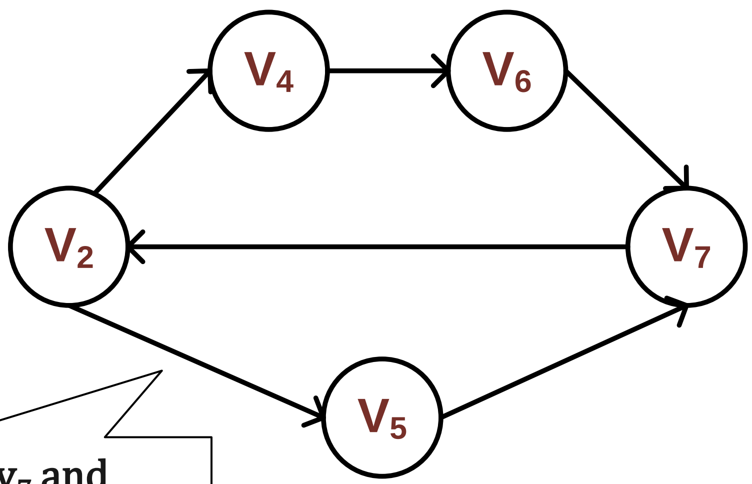
Introduction to Graphs /14

- **Graph Cycle:**
  - A cycle is a path in a graph where one node is visited more than once
  - In a simple cycle, only the starting node is visited more than once
  - A graph with no cycles is referred to as acyclical

- **Acyclical Graph:**



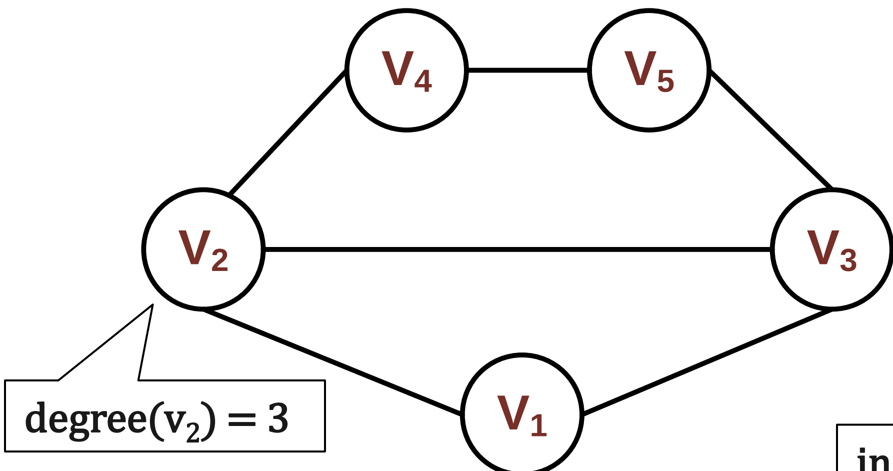
Cyclical Graph:

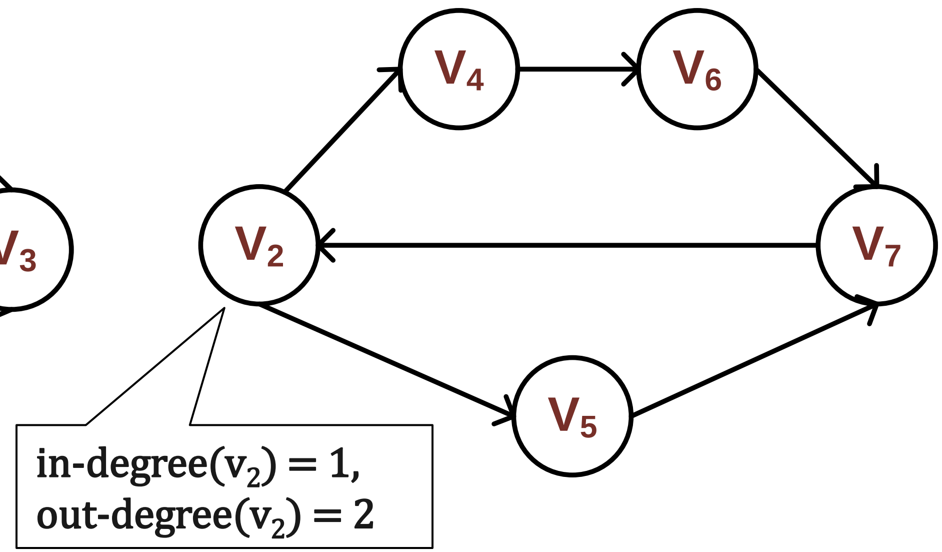


A path from v_2 to v_5 to v_7 and then back to v_2 is a simple cycle

Introduction to Graphs /15

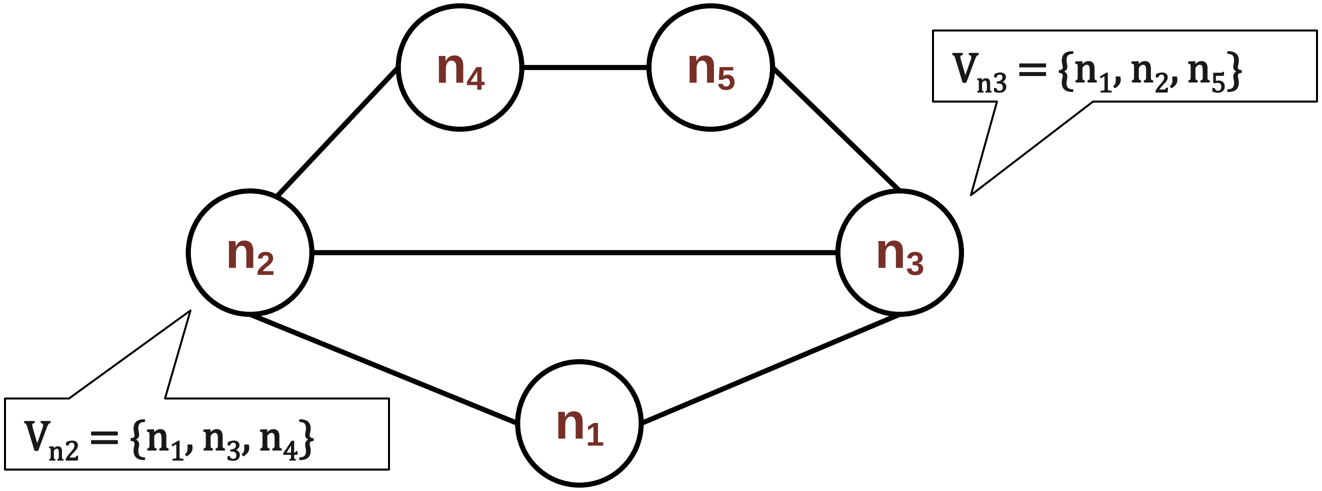
- **Degree of a Node:**
  - In an undirected graph, **degree** of a node is the number of distinct edges where the node is an end point
  - In a directed graph, **in-degree** of a node is the number of distinct edges where the node is the end point
  - In a directed graph, **out-degree** of a node is the number of distinct edges where the node is the starting point





Introduction to Graphs /16

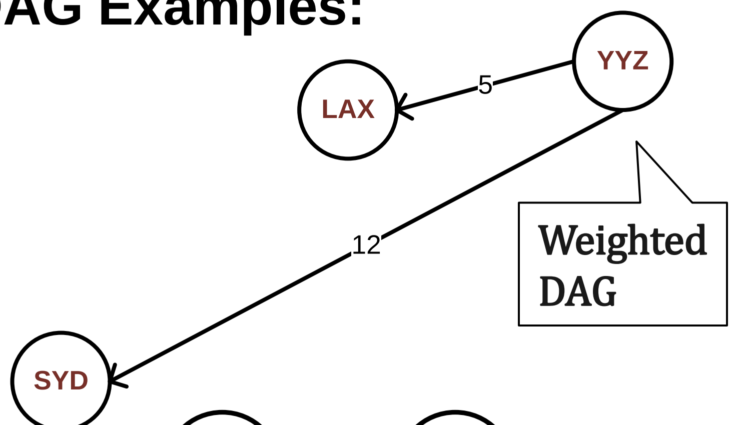
- **Adjacency of Nodes:**
  - Two nodes are adjacent if there exists an edge that connects them
  - The adjacency set of a node x, which is denoted as V_X, is the set of all nodes that are adjacent to x
  - Formally, $V_X = \{y \mid e(x, y) \in E\}$

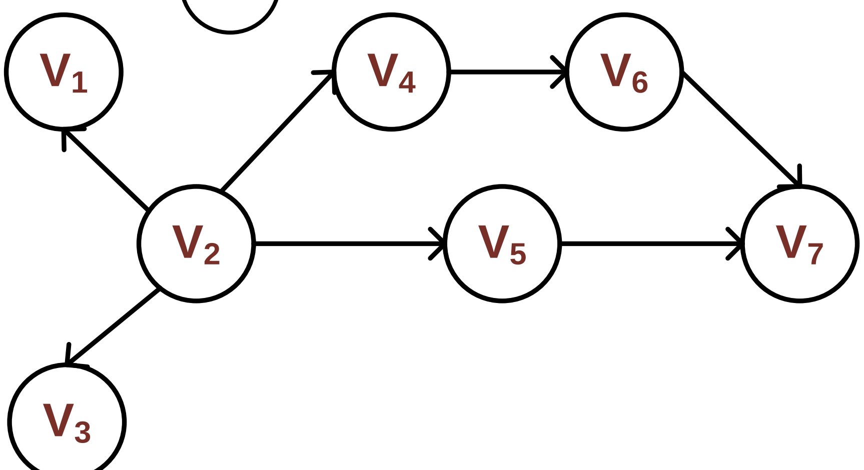


Directed Acyclic Graph (DAG)

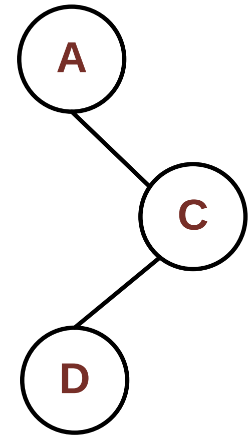
- **Directed Acyclic Graph (DAG):**
  - A directed graph that includes no cycles

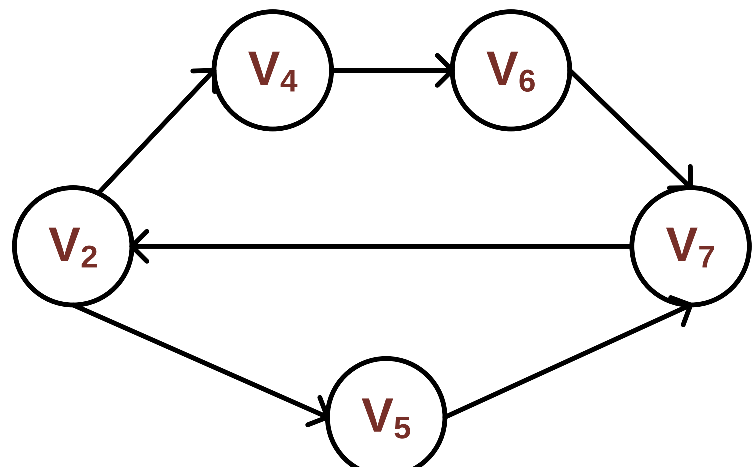
- **DAG Examples:**





Not DAGs:





Graph Representation /1

- **How to represent a graph as a data structure?**
  - Using sequential representation:
    as a matrix
  - Using linked representation:
    as a sequence of linked lists

Graph Representation /2

- **Matrix representation of a graph:**
  - For a graph with $\mathbf{n}$ nodes, create a matrix of $\mathbf{n \times n}$ size

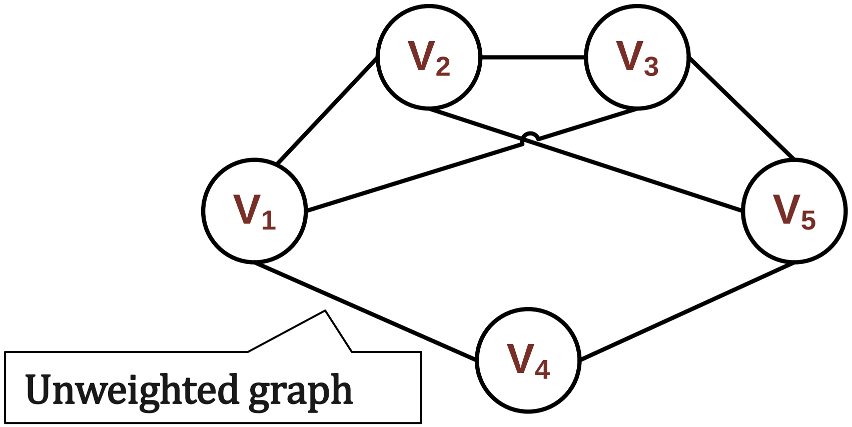
- **For undirected and unweighted graph:**
  - If there exists an edge from i to j, store 1 as the value for $\mathrm{M(i,j)}$ and $\mathrm{M(j,i)}$; otherwise, store 0 for $\mathrm{M(i,j)}$ and $\mathrm{M(j,i)}$

- **For directed and unweighted graph:**
  - If there exists an edge from i to j, store 1 as the value for $\mathrm{M(i,j)}$; otherwise, store 0 for $\mathrm{M(i,j)}$

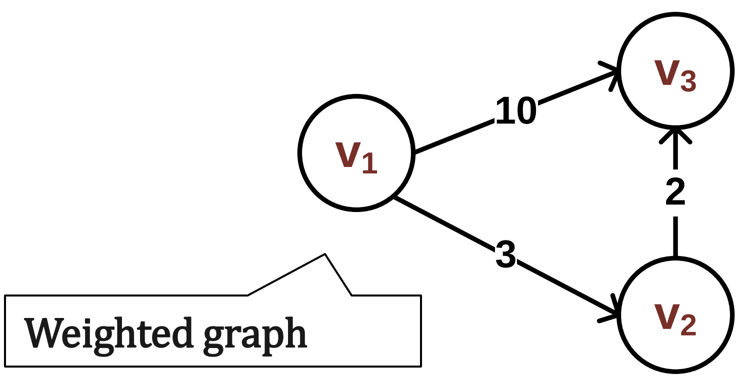
- **For directed and weighted graph:**
  - If there exists an edge from i to j, store $\mathrm{w(i,j)}$ as the value for $\mathrm{M(i,j)}$; otherwise, store ∞ for $\mathrm{M(i,j)}$

Graph Representation /3

- **Matrix representation example:**



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



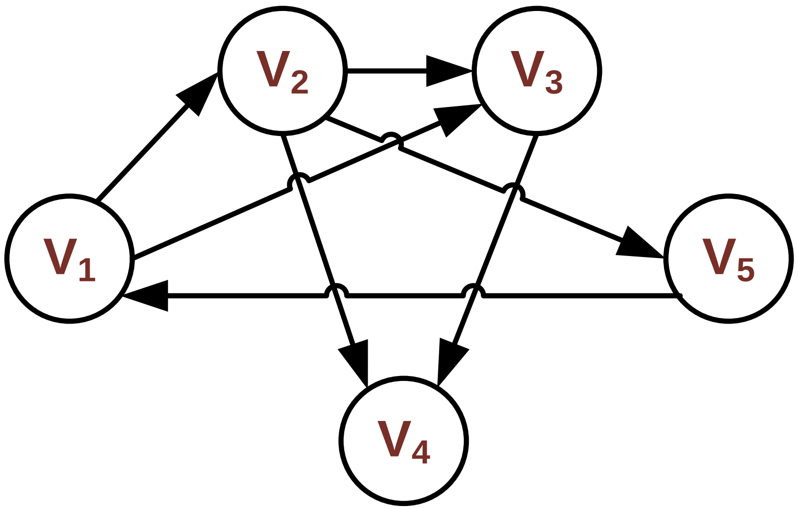
	1	2	3
1	∞	3	10
2	∞	∞	2
3	∞	∞	∞

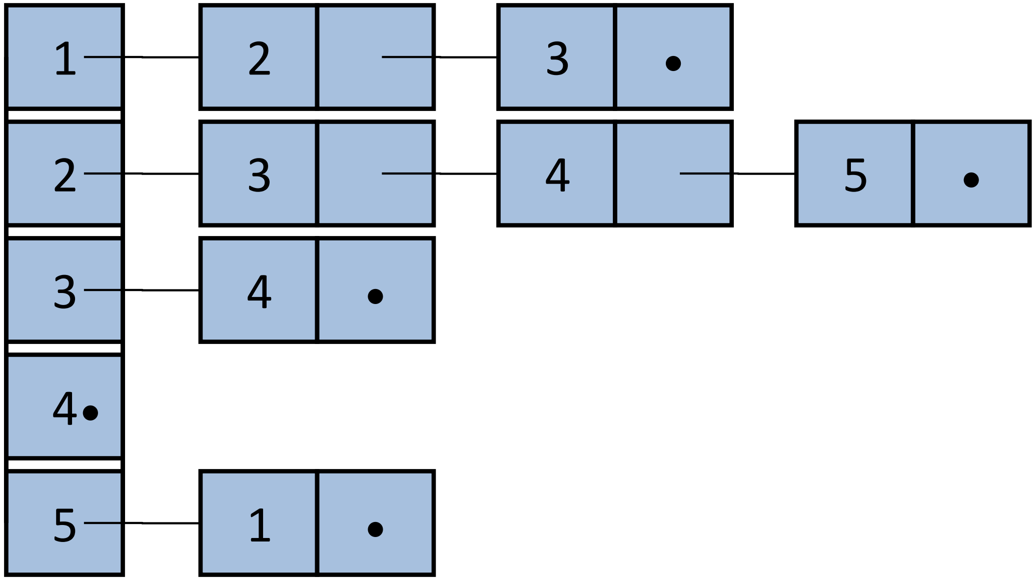
Graph Representation /4

- **Linked representation of a graph:**
  - For a graph with $\mathbf{n}$ nodes, create a list $\mathbf{L}$ of size $\mathbf{n}$, with one list element for each graph node
  - Each element of the list is itself a list that specifies all edges that originate from the corresponding graph node
  - The list element $L(i)$ points to the i-th linked list, where each node j in that linked list represents an edge going from node i to node j

Graph Representation /5

- **Linked representation example:**

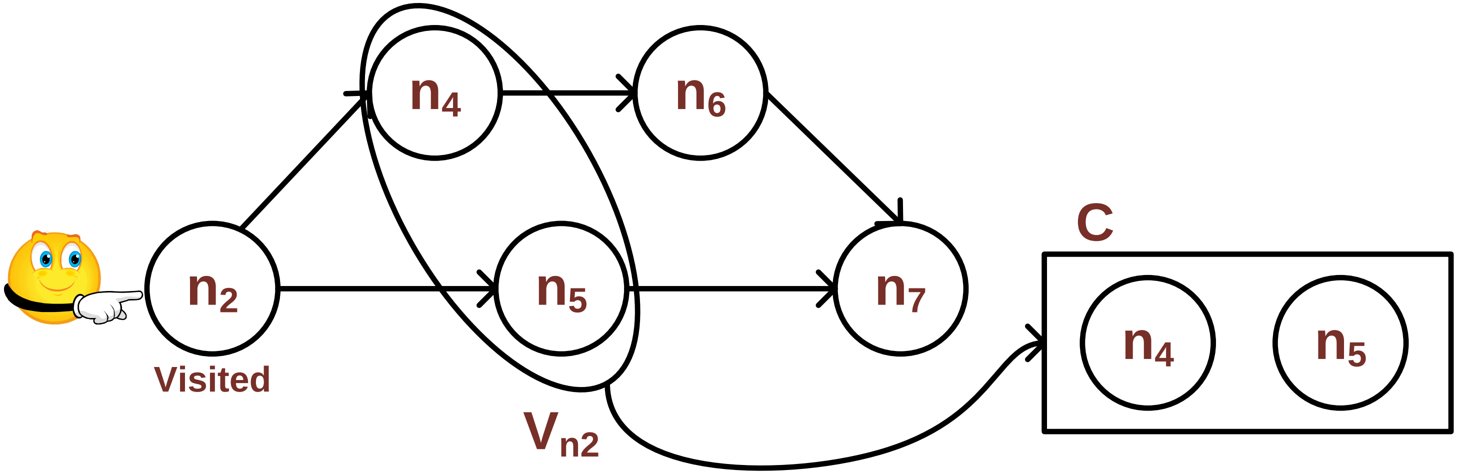




Graph Traversal /1

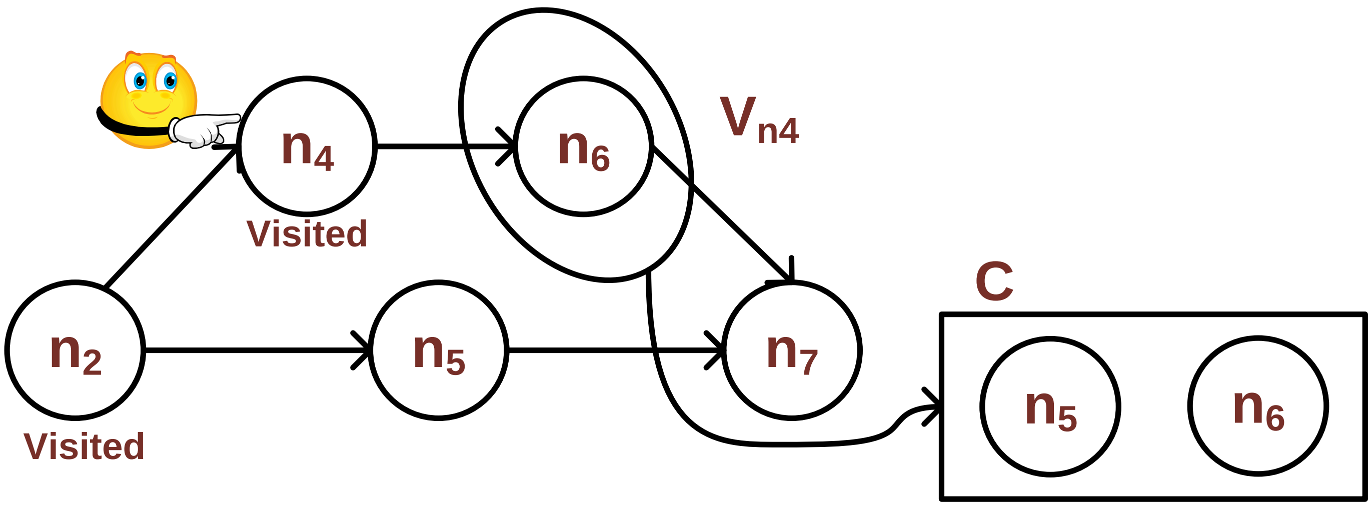
- **Graph Traversal:**
  - A common graph operation is **graph search**, where the goal is to traverse through a graph based on its edges, and visit all nodes in the graph in a particular order

- **Common Steps:**
  - Step1. At a node v_x, access its adjacency set V_x and insert all unvisited nodes in V_x into some container C



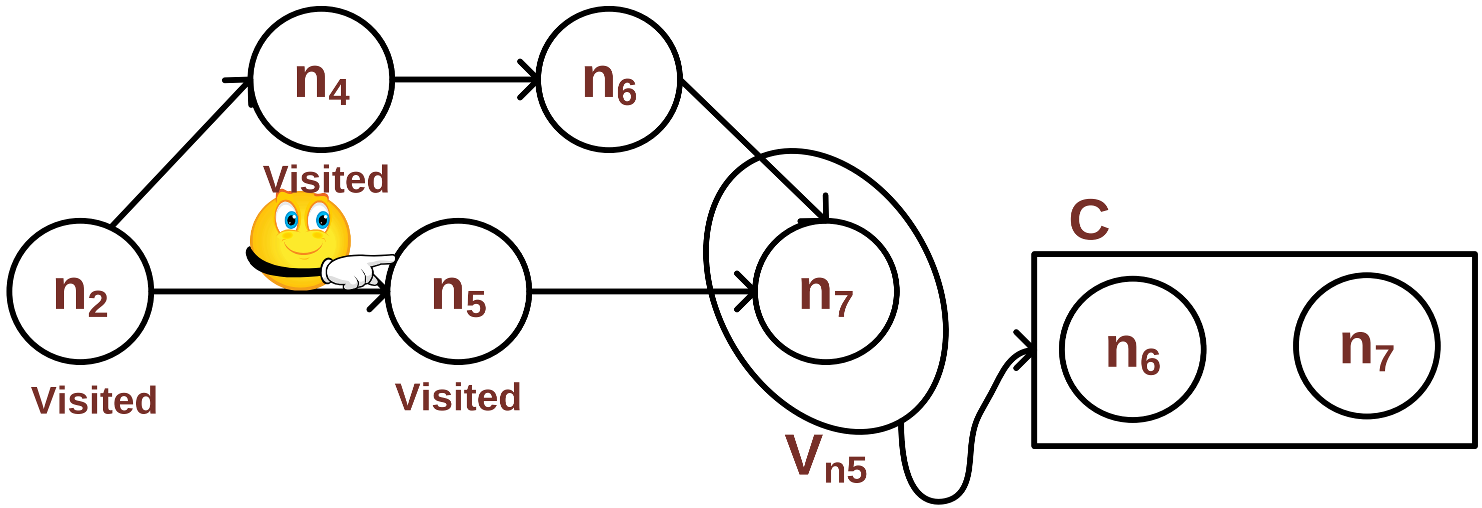
Graph Traversal /2

- **Common Steps Continued:**
  - Step2. Remove a node v_y from C, mark it as "visited", and insert all unvisited nodes in the adjacency set V_y into C



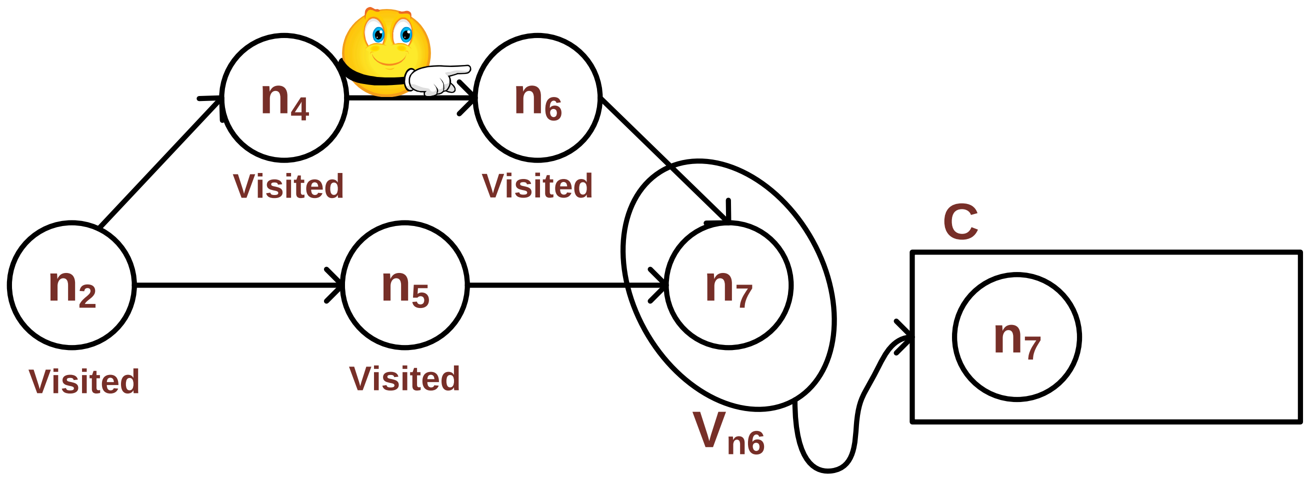
Graph Traversal /3

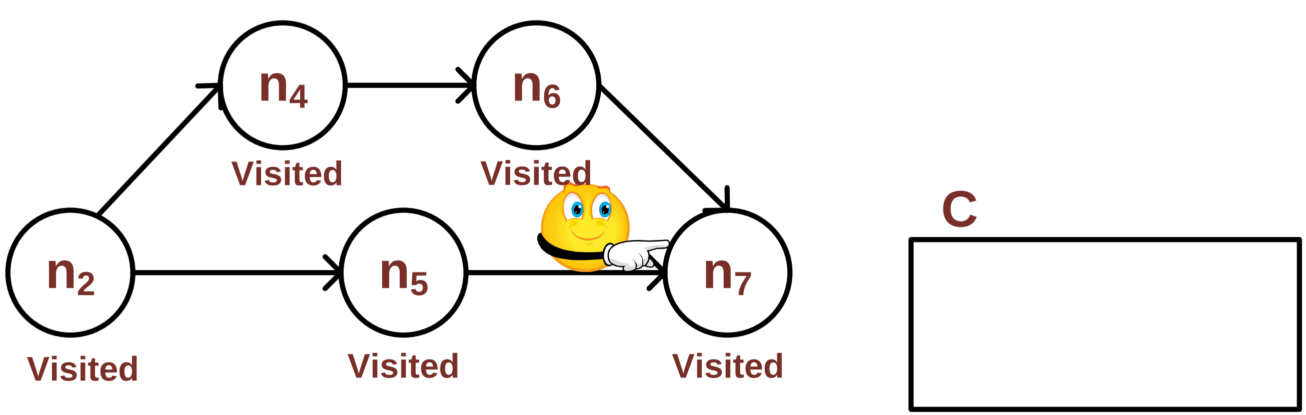
- **Common Steps Continued:**
  - Step3. Repeat until the container is empty



Graph Traversal /4

- **Common Steps Continued:**
  - Step3. Repeat until the container is empty

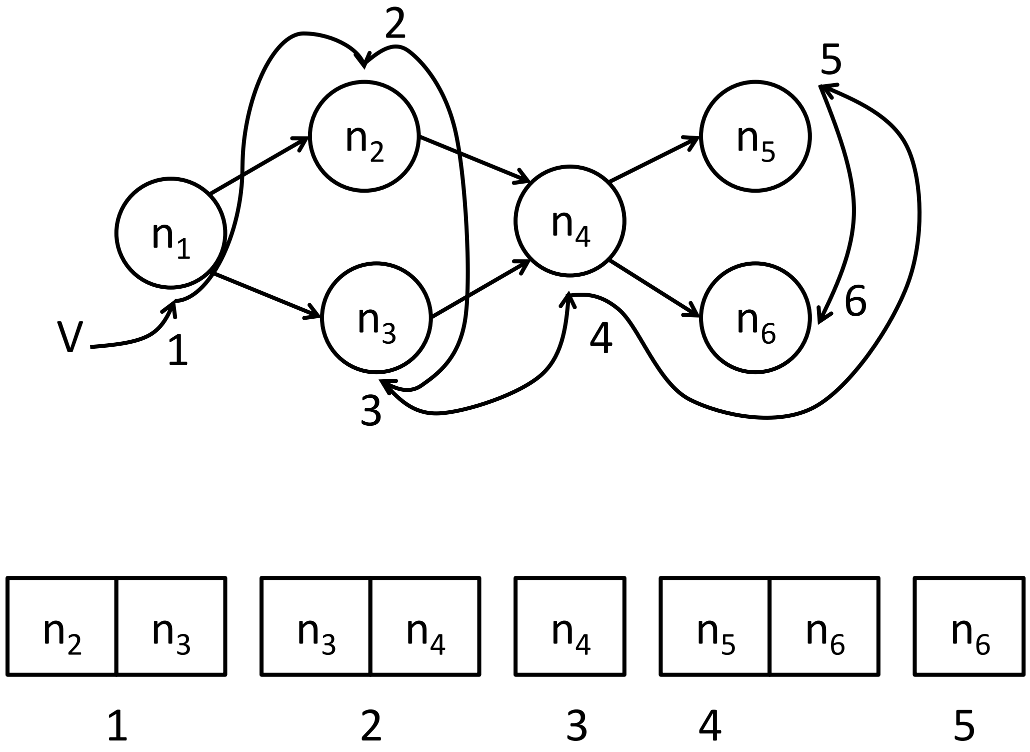




Graph Traversal /5

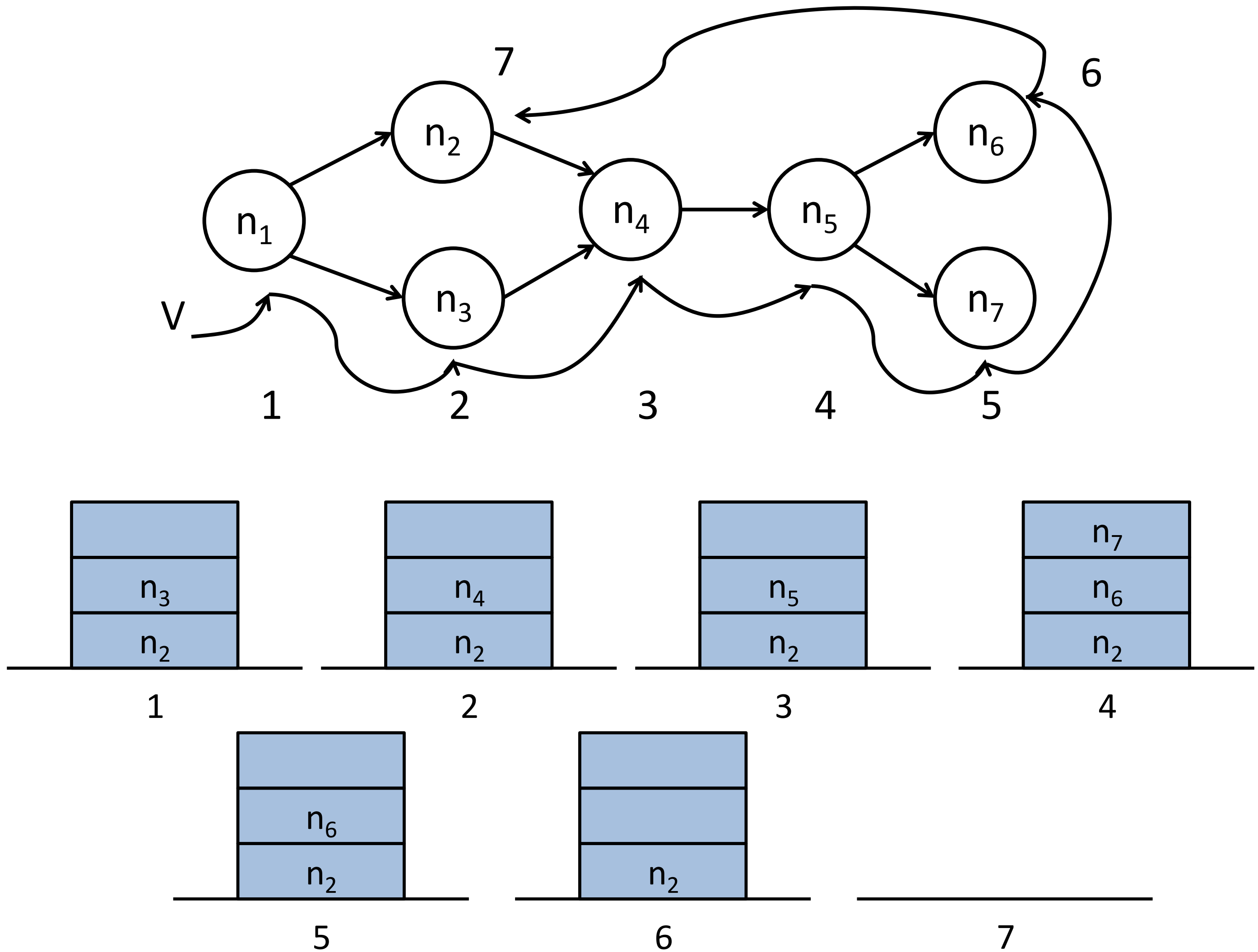
- **Graph Traversal and Container Selection:**
  - Breadth-First Search (BFS):
    use Queue ADT (FIFO principle) as the container
  - Depth-First Search (DFS):
    use a Stack ADT (LIFO principle) as the container

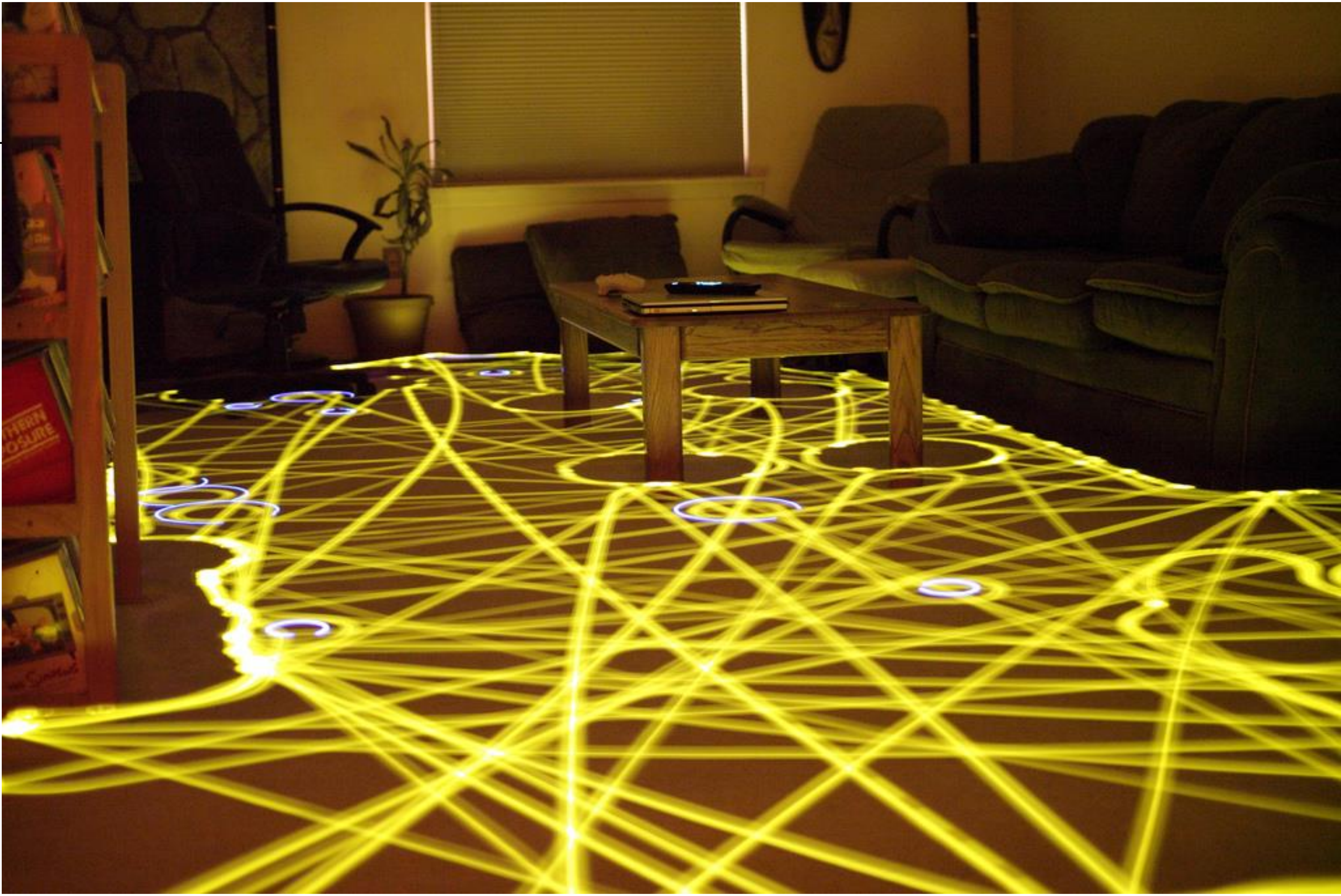
- **Breadth-First Search (BFS) Example:**



Graph Traversal /6

- **Depth-First Search (DFS) Example:**





Long exposure of Roomba trajectory

https://en.wikipedia.org/wiki/Roomba#/media/File:Roomba_time-lapse.jpg

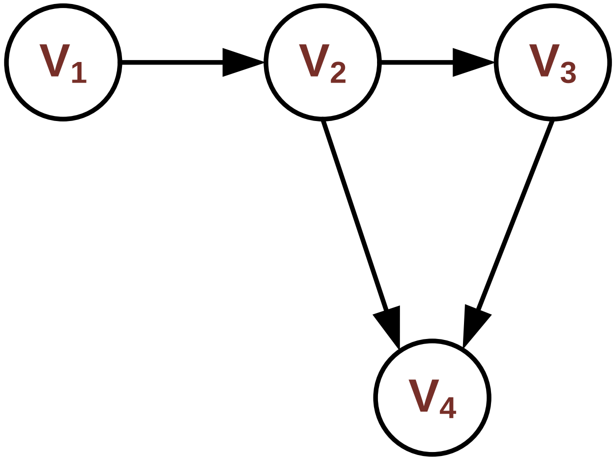
Path Finding in Graphs /1

- **One of the more interesting and challenging problems involving the use of graphs is path finding**
  - “Automatically find directions between physical locations”
  - This is crucial for a large number of robotic navigation and artificial intelligence applications
  - *Many* other applications (circuit optimization, telecommunication networks, etc.)

- **Among the path-finding problems, the shortest path problem is one of the most fundamental**
  - The goal is to identify a path between a starting node and a destination node, so that the sum of weights of the edges included in the path is minimized

Path Finding in Graphs /2

- **Example: path finding in unweighted directed graph**
  - Find the shortest path from v_1 to v_4



  - Answer: $v_1 – v_2 – v_4$ that has the path length of 2

- **A well-known approach to finding the shortest path is the Dijkstra's Algorithm**

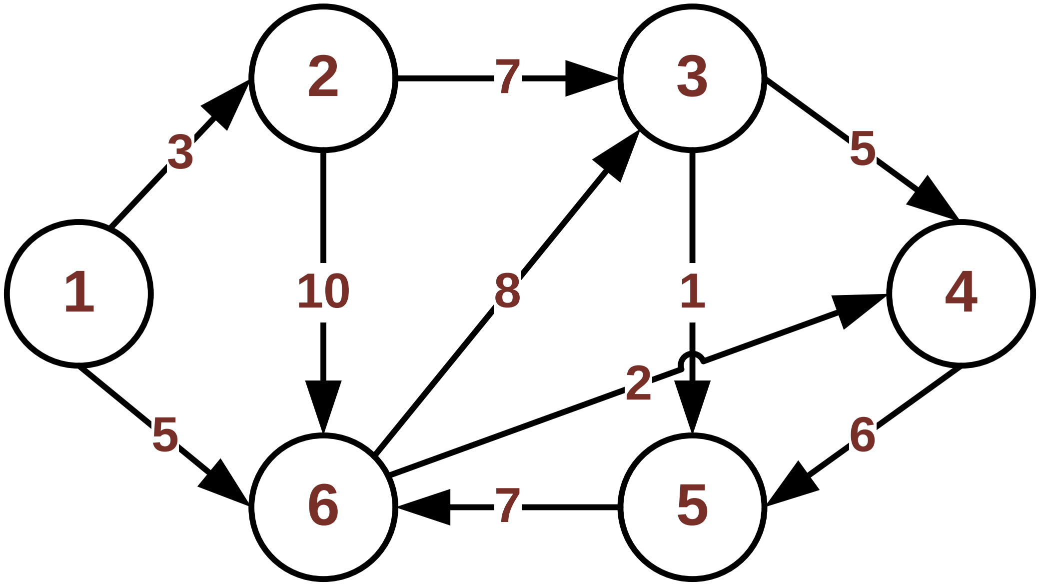
Path Finding in Graphs /3

- **Dijkstra's Algorithm for finding the shortest path in a graph:**
  1. Construct the shortest paths from the starting node to its closest nodes
  2. Keep track of the visited nodes and update the shortest path information to each unvisited node
  3. Extend the paths until the destination node is reached

- **Dijkstra's Algorithm is a greedy algorithm**
  - It chooses a solution that appears to be the best solution at the time, then worry about possible new solutions later
  - It has a runtime efficiency of $O(n^2)$, where n is the number of nodes in a graph

Path Finding in Graphs /4

- **Let us explain Dijkstra's Algorithm by applying it to the following graph**
  - The goal is to find the shortest path from node 1 to every other node in the graph



Path Finding in Graphs /5

- **Step1. Setup**
  - The first step is to construct and set up data structure representation of the graph
  - If we make use of the matrix representation, for a weighted directed graph of 6 nodes where $\mathbf{V} = \{1, 2, 3, 4, 5, 6\}$, the corresponding matrix $\mathbf{M}$ is as follows:

	1	2	3	4	5	6
1	0	3	∞	∞	∞	5
2	∞	0	7	∞	∞	10
3	∞	∞	0	5	1	∞
4	∞	∞	∞	0	6	∞
5	∞	∞	∞	∞	0	7
6	∞	∞	8	2	∞	0

Path Finding in Graphs /6

- **Step2. Initialization**
  - The second step is to construct a framework for keeping track of the shortest paths at each iteration
  - This includes the sum of weights for each shortest path, as well as the visited and unvisited nodes
  - The following data will be kept:
    - V: visited nodes
    - U: unvisited nodes
    - C: current closest node
    - d_c: the sum of weights of path from the starting node to the closest node C
    - d_i: the sum of weights of path from the starting node to node i

Iteration	V	U	C	d_c	d_1	d_2	d_3	d_4	d_5	d_6
1	1	2,3,4,5,6	-	-	0	3	∞	∞	∞	5

Path Finding in Graphs /7

- **Step3. Update**
  - First, we need to identify the node C from the set of unvisited nodes U that has the minimum distance from the starting node
  - From node 1, the closest node is node 2, which has the minimum distance of 3, so $\mathrm{C} = 2$ and $\mathrm{d_c} = 3$
  - Next, the closest node C is removed from the unvisited nodes U and placed into the set of visited nodes V
  - Hence, $\mathrm{V} = \{1{,}2\}$ and $\mathrm{U}{=}\{3{,}4{,}5{,}6\}$

Path Finding in Graphs /8

- **Step3. Update Continued**
  - Finally, the shortest distances from the starting node to the nodes in U are updated relative to the closest code C
  - To do so, for each node i, we compute the total distance from the starting node to C to node i, and then compare it with the current d_i
  - For the path from the starting node to C, if there is no direct path, use the shortest path from the starting node to C
  - If this newly computed distance is shorter than the current d_i, then this new path is the shortest path, so replace the current d_i

Path Finding in Graphs /9

- **Step3. Update Continued**
  - It follows that:
    - Node 2: (1) – (2) (d = 3) or (1) – (2) – (2) (d = 3), so shortest path: (1) – (2) (d_2 = 3)
    - Node 3: (1) – (3) (d = ∞) or (1) – (2) – (3) (d = 10), so shortest path: (1) – (2) – (3) (d_3 = 10)
    - Node 4: (1) – (4) (d = ∞) or (1) – (2) – (4) (d = ∞), so shortest path: (1) – (4) (d_4 = ∞)
    - Node 5: (1) – (5) (d = ∞) or (1) – (2) – (5) (d = ∞), so shortest path: (1) – (5) (d_5 = ∞)
    - Node 6: (1) – (6) (d = 5) or (1) – (2) – (6) (d = 13), so shortest path: (1) – (6) (d_6 = 5)
  - Updated algorithm information at the end of the cycle is as follows:

Iteration	V	U	C	d_c	d_1	d_2	d_3	d_4	d_5	d_6
2	1,2	3,4,5,6	2	3	0	3	10	∞	∞	5

Path Finding in Graphs /10

- **Step4. Repeat**
  - Repeat Step3 until all nodes have been visited
  - Iteration3: C = 6 since it has the next shortest distance to starting node ($d_c = 5$)
  - It follows that:
    - Node 3: (1) – (2) – (3) (d = 10) or (1) – (6) – (3) (d = 13), so shortest path: (1) – (2) – (3) ($d_3 = 10$)
    - Node 4: (1) – (2) – (4) ($d = \infty$) or (1) – (6) – (4) (d = 7), so shortest path: (1) – (6) – (4) ($d_4 = 7$)
    - Node 5: (1) – (5) ($d = \infty$) or (1) – (6) – (5) ($d = \infty$), so shortest path: (1) – (5) ($d_5 = \infty$)

Iteration	V	U	C	d_c	d_1	d_2	d_3	d_4	d_5	d_6
3	1,2,6	3,4,5	6	5	0	3	10	7	∞	5

Path Finding in Graphs /11

- **Step4. Repeat Continued**
  - Iteration4: C = 4 since it has the next shortest distance to starting node (d_c = 7)
  - It follows that:
    - Node 3: (1) – (2) – (3) (d = 10) or (1) – (6) – (4) – (3) (d = ∞),
      so shortest path: (1) – (2) – (3) (d_3 = 10)
      (use the shortest path from (1) to (4): (1) – (6) – (4))
    - Node 5: (1) – (5) (d = ∞) or (1) – (6) – (4) – (5) (d = 13),
      so shortest path: (1) – (6) – (4) – (5) (d_5 = 13)
      (use the shortest path from (1) to (4): (1) – (6) – (4))

Iteration	V	U	C	d_c	d_1	d_2	d_3	d_4	d_5	d_6
4	1,2,6,4	3,5	4	7	0	3	10	7	13	5

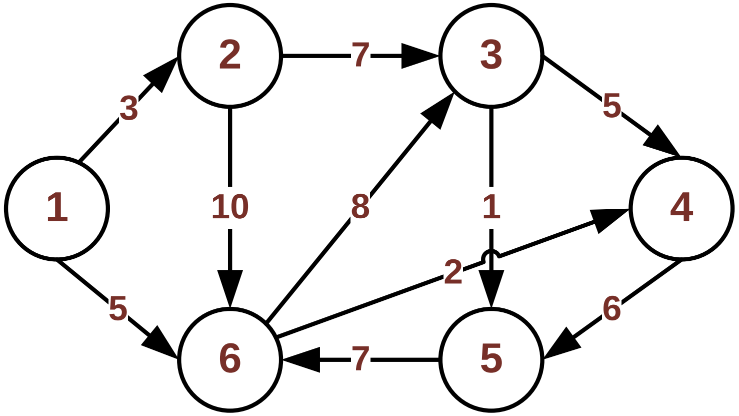
Path Finding in Graphs /12

- **Step4. Repeat Continued**
  - Iteration5: C = 3 since it has the next shortest distance to starting node (d_c = 10)
  - It follows that:
    - Node 5: (1) – (6) – (4) – (5) (d = 13) or (1) – (2) – (3) – (5) (d = 11), so shortest path: (1) – (2) – (3) – (5) (d_5 = 11)
      (use the shortest path from (1) to (3): (1) – (2) – (3))

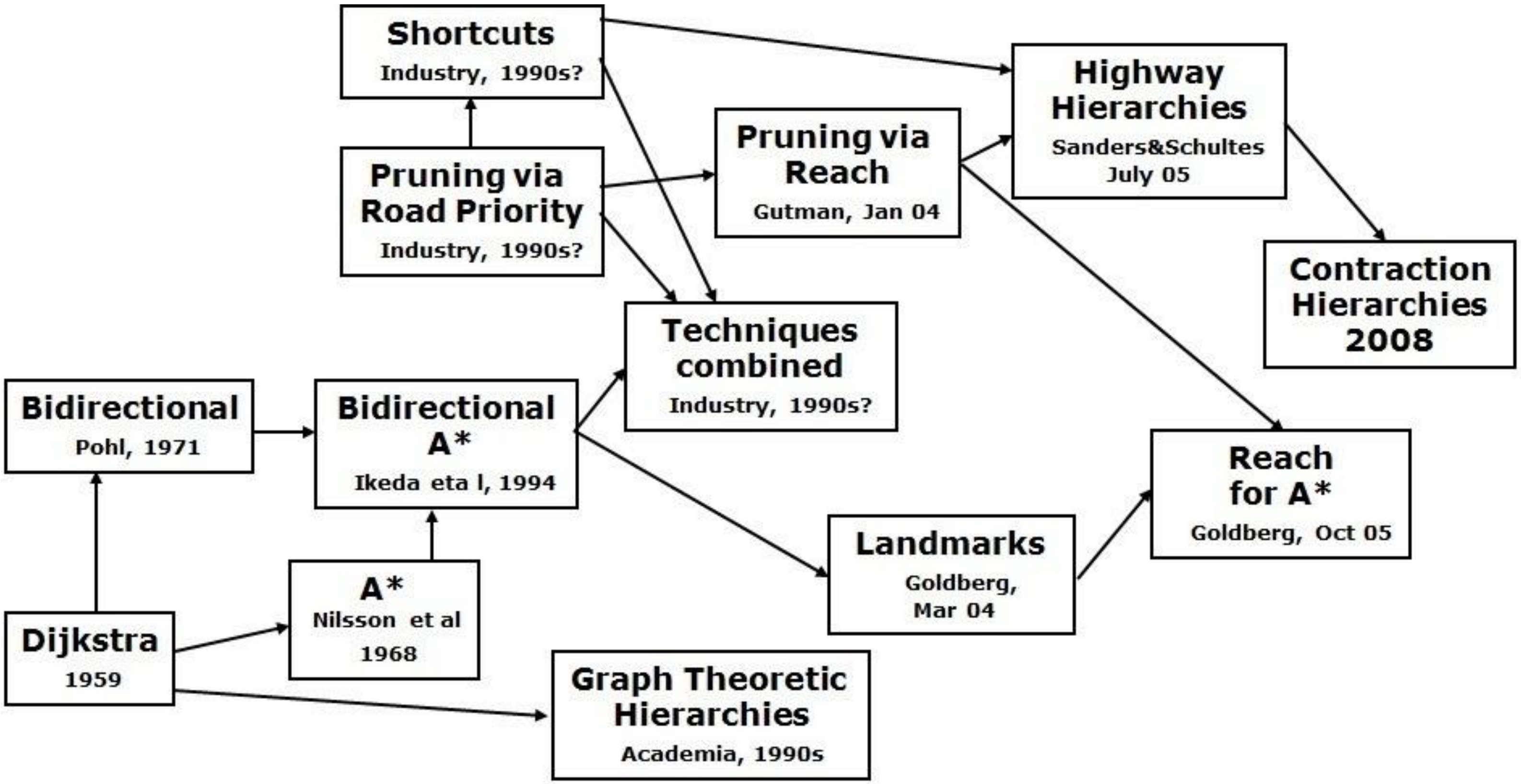
Iteration	V	U	C	d_c	d_1	d_2	d_3	d_4	d_5	d_6
5	1,2,6,4,3	5	3	10	0	3	10	7	11	5

Path Finding in Graphs /13

- **Step4. Repeat Continued**
  - Iteration6: C = 5 with $d_c = 11$
  - It follows that:
    - Node 5: (1) – (2) – (3) – (5) (d = 11) or (1) – (2) – (3) – (5) – (5) (d = 11) so shortest path: (1) – (2) – (3) – (5) ($d_5 = 11$)
    - Hence, no change from Iteration5
  - The final shortest paths are as follows:
    - Node 2: (1) – (2) ($d_2 = 3$)
    - Node 3: (1) – (2) – (3) ($d_3 = 10$)
    - Node 4: (1) – (6) – (4) ($d_4 = 7$)
    - Node 5: (1) – (2) – (3) – (5) ($d_5 = 11$)
    - Node 6: (1) – (6) ($d_6 = 5$)



(Brief) History of Graph Traversal Algo’s

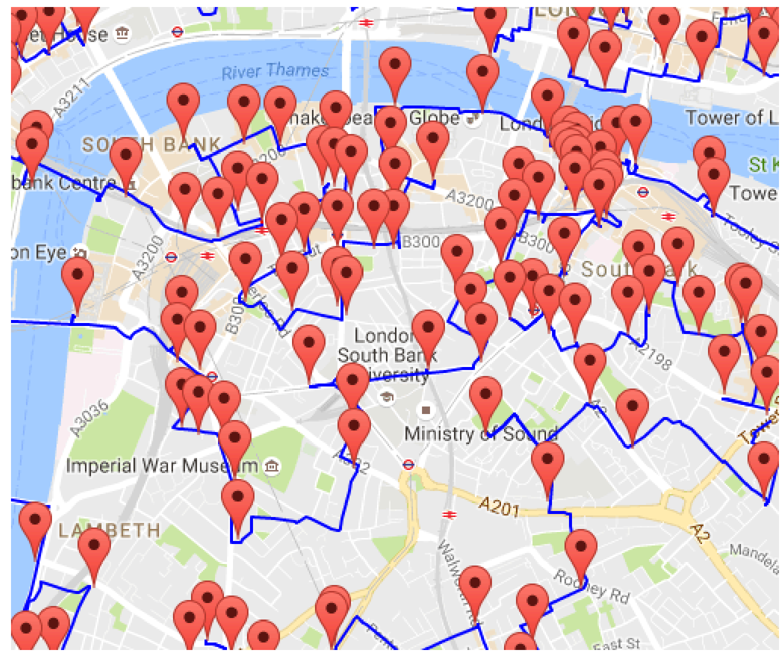


https://www.quora.com/How-does-the-algorithm-of-Google-Maps-work

Aside: Graphs in Practice

- Route optimization
  - Traveling salesperson problem: “"Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?" ”

Pokemon Go optimization



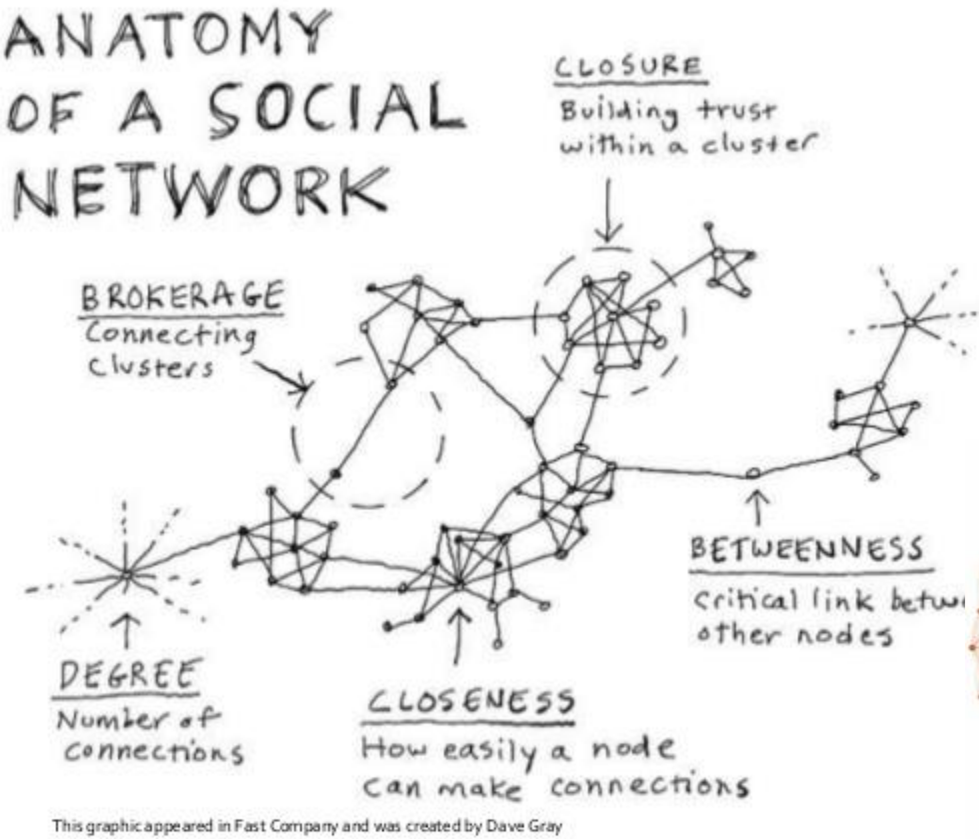
UK Pub Crawl optimization (warning: 24,727 pubs; don't try this at home)

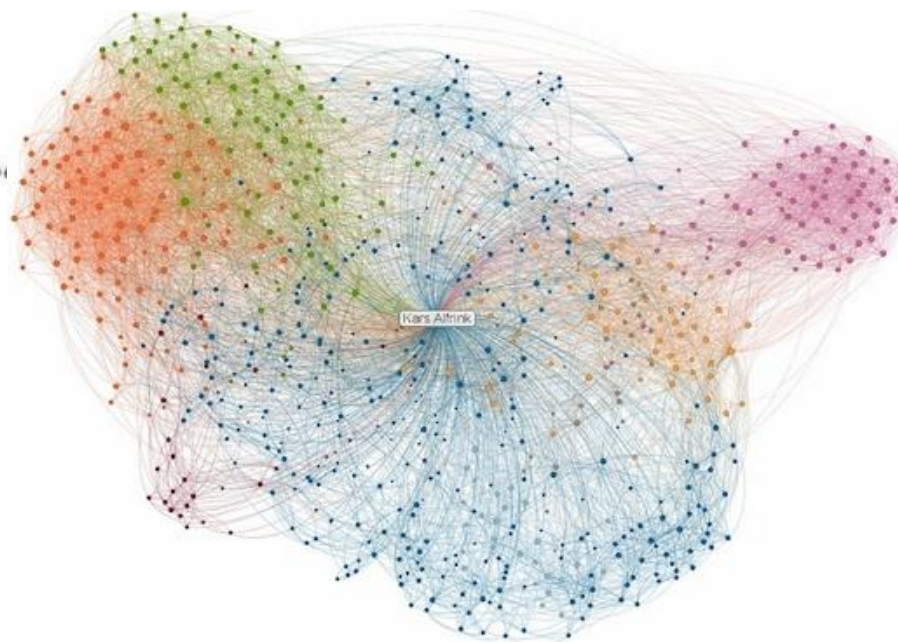
Aside: Graphs in Practice

- Social networks

Massive, *massive* research in this field

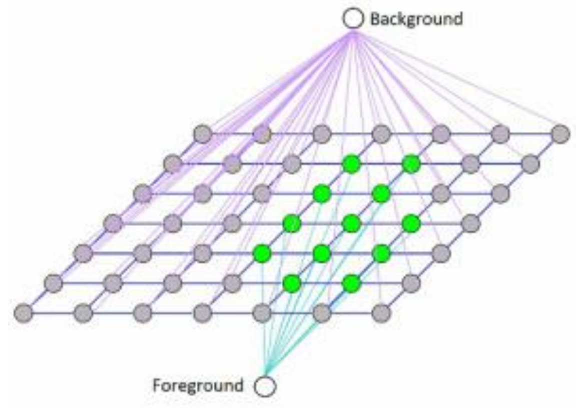
- **Friend/job/event suggestion**
- **Clique formation**
- **Network analysis**
- **Ads ($$$)**

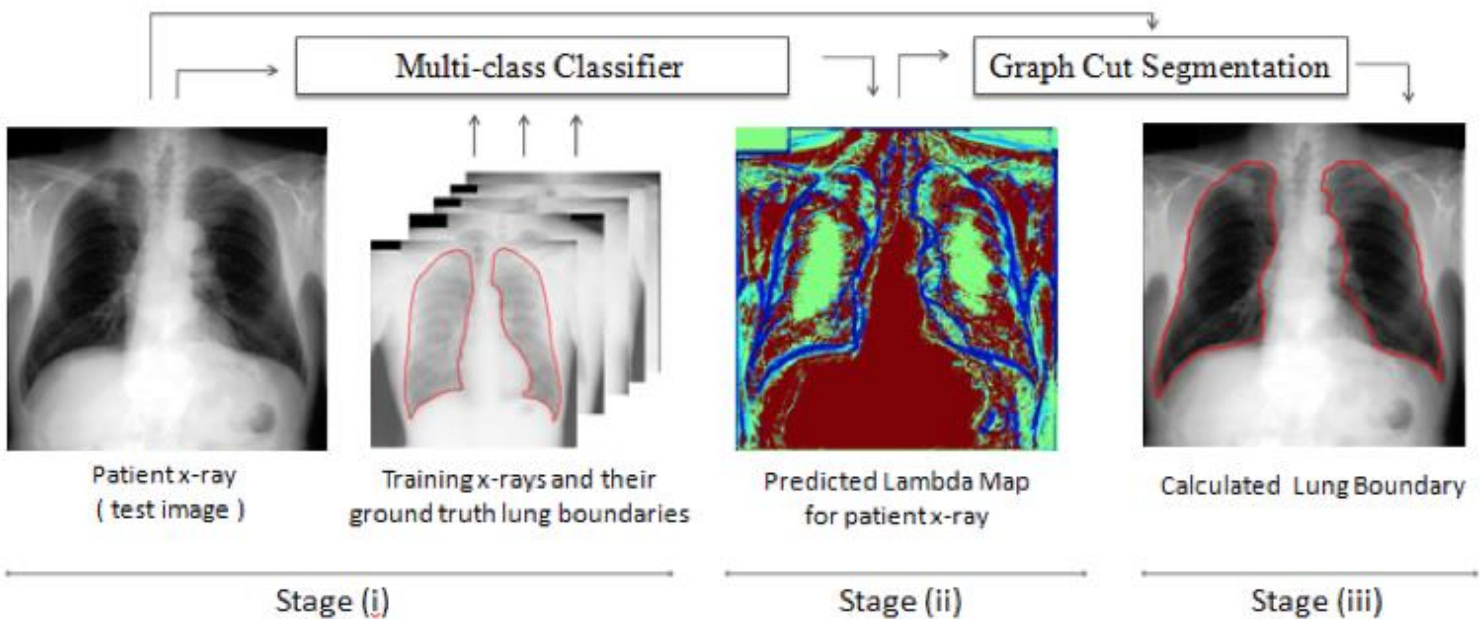




Aside: Graphs in Practice

- GraphCut image segmentation
  - GrabCut demo





Lecture Notes Summary

- **What do you need to know?**
  - Graph definition and basic applications
  - Graph formal definition
  - Graph properties
  - Graph representation
  - Graph traversals
  - Path finding and Dijkstra’s algorithm

Food for Thought

- **Read:**
  - Chapter 8 (Graphs) from the course handbook

- **Additional Readings:**
  - Chapter 15 (Graphs) from “Data Structures and Other Objects Using C++” by Main and Savitch
  - Amit Patel, *Introduction to A*, From Amit’s Thoughts on Pathfinding*, 2017. [Online]. http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html