Week 8 - GRAPHS

AD 325 - 2022

Contents

Learning Outcomes

- overview of graphs
- directed & undirected graphs
- graph traversal
- graph implementation

Reading & Videos

- Carrano & Henry: Ch. 29, 30
- https://www.coursera.org/learn/algorithms-part2/home/week/1

Reference

- https://algs4.cs.princeton.edu/41graph/
- https://algs4.cs.princeton.edu/42digraph/
- https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/
- https://www.geeksforgeeks.org/topological-sorting/

Overview

In computer science, a **Graph** is a non-linear data structure consisting of **vertices (nodes)** and **edges** - lines that connect any two vertices.

Graphs are useful for solving real-world network problems. Some examples, with vertices in parens:

- road maps (towns)
- social network (people)
- airline routes (airports)
- mazes (corners & ends)
- college degree programs (courses)

Unlike other data structures, graphs are not modified after being created. Instead they are used to answer questions about how vertices are related.

All trees are **connected graphs** with hierarchical order and without cycles.

Edges & Paths

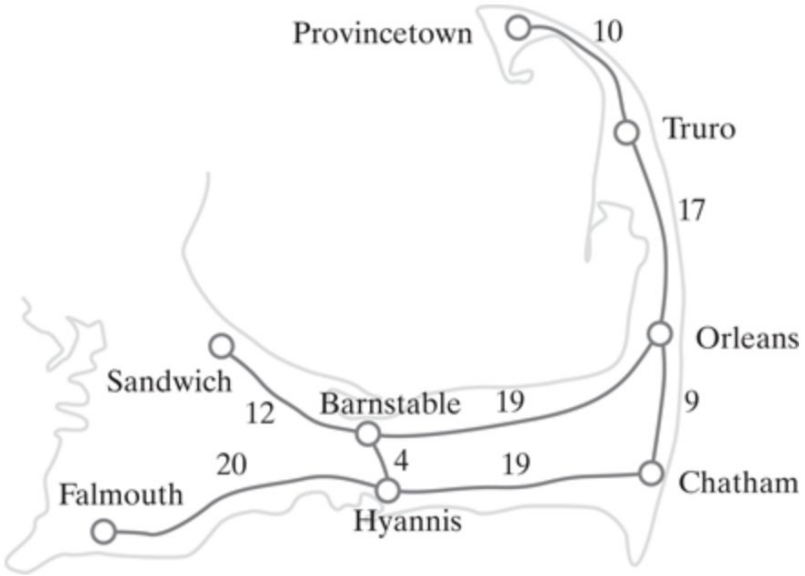


Figure 29-3
A weighted graph

An edge may be **undirected** if there is a two-way relationship between the vertices it connects, or **directed** if the relationship is one-way (e.g. one-way streets).

A graph with directed edges is called a **directed graph** or digraph.

Edges may have an associated weight or cost (e.g. road distance, driving time, etc.).

A **path** between two vertices in a graph is a sequence of edges, where path length is the number of edges.

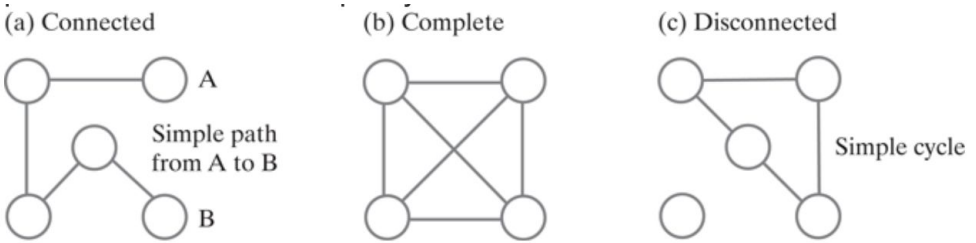
Edges & Paths, cont.

A **cycle** is a path that begins and ends at the same vertex and a **simple cycle** passes through other vertices only once each.

The weight of a path in weighted graph is the sum of its edge weights. Algorithms can use weights to choose from among multiple valid paths (e.g. to minimize cost).

A **connected graph** has a path between every pair of vertices and is complete if every vertex is connected to every other vertex. A disconnected graph has some vertices that can't be reached from all other vertices.

Two vertices in an undirected graph are adjacent (aka neighbors) if joined by an edge.



Traversal

Graph applications focus on the connections between vertices, rather than the contents of vertices.

Graph traversal can begin from any vertex (origin) and visits only vertices reachable from the origin.

Visited vertices are marked to avoid repeated visits.

The order in which neighbors are visited can vary according a graph's implementation.

Breadth-first Search (BFS)

A breadth-first search visits each of a vertex's neighbors before visiting neighbors of neighbors and so on. Level-order tree traversal is an example of BFS.

BFS uses a queue to hold the visited vertices and traversal order is the order that vertices are added to the queue.

Applications:

- Shortest path for an unweighted graph
- Peer-to-peer networks
- Web crawlers
- Social networks
- Garbage collection (in programming)
- Cycle detection

Depth-first Search (DFS)

A depth-first search follows a single path as deeply as possible before following other paths. Inorder, preorder, and postorder traversal are examples of DFS.

DFS traversal can be recursive and uses a queue to track visited vertices. The traversal order is the order in which vertices are added to the queue.

DFS is useful to determine the path between two vertices.

Applications:

- Detecting a cycle
- Path finding
- Topological sorting
- Solving mazes

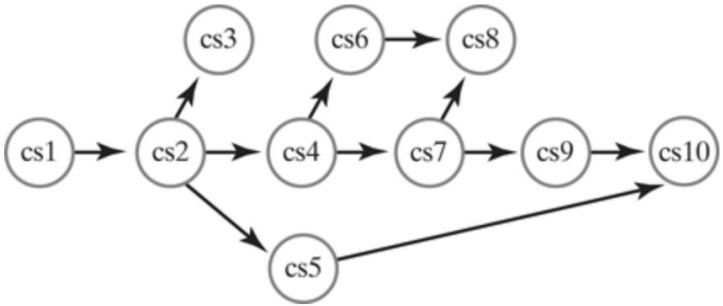
Topological Order

Vertices in a directed graph without cycles can be placed in topological order (in order of their precedence).

Such a graph may have more than one valid topological order.

Graphs with a cycle cannot have a topological order because this would result in circular logic.

Topological sort is the process for discovering the topological order for vertices in a graph. It uses a stack to hold vertices that have no successor or whose neighbors have been visited.



Implementation - Adjacency Matrix

Graphs can be implemented as a two-dimensional array of size V rows and V columns, where V is the number of vertices in the graph.

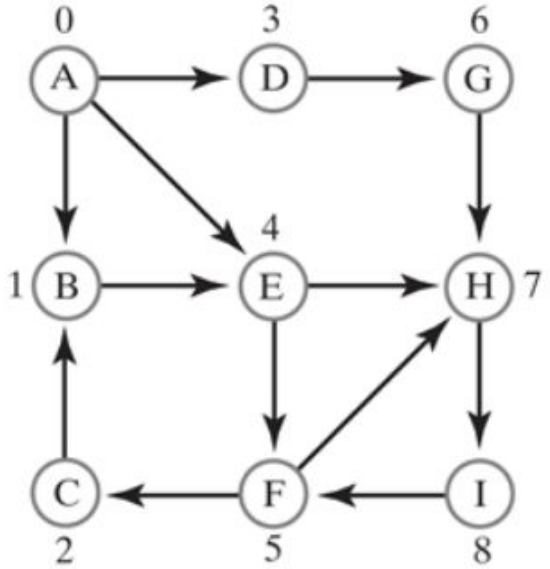
For an unweighted graph, the adjacency matrix would have boolean values for each edge. For a weighted graph, the matrix value would be the edge weight or infinity if no edge exists.

Determining if an edge exists for two vertices is a O(1) operation, but finding all the neighbors of a vertex is O(V).

The adjacency matrix requires fixed space for all possible edges, even though graphs are usually sparse. It can be a good choice for a dense graph.

Adjacency Matrix

(a) A graph



(b) The graph's adjacency matrix

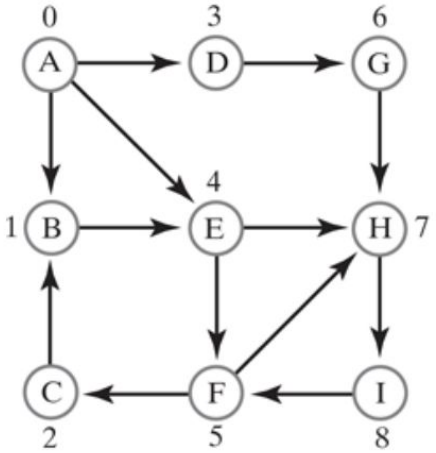
	A	B	C	D	E	F	G	H	I	
A		T		T	T					0
B					T					1
C		T								2
D							T			3
E						T		T		4
F			T					T		5
G								T		6
H									T	7
I						T				8
	0	1	2	3	4	5	6	7	8	

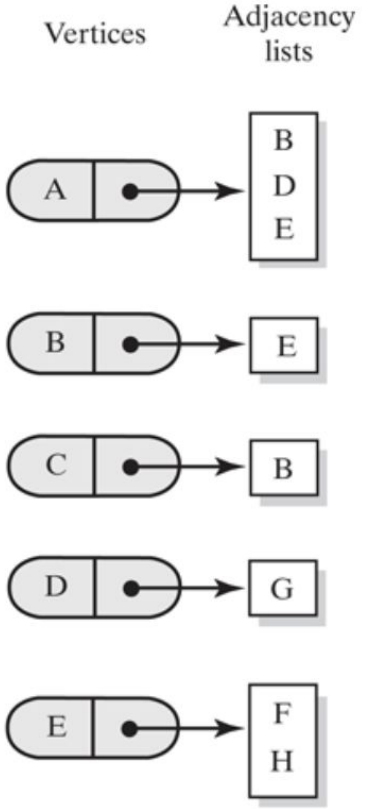
Implementation - Adjacency List

An adjacency list maintains a vertex-indexed array of lists. For each vertex, the array contains a list of its neighbors and represents the edges that originate from this vertex.

Space is reserved only for edges that exist, so this approach uses less memory.

Determining if an edge exists between two vertices, or finding all the neighbors of a vertex, is O(n) at worst but faster on average.





Efficiency

Implementation	Space	Add edge	Find edge btw v and w	Iterate vertices adjacent to v
Adjacency matrix	V^2	O(1)	O(1)	V
Adjacency list	E+V	O(1)	degree(v)	degree(v)

V = number of vertices
E = number of edges

Glossary

- **Graph** - a collection of vertices and edges
- **Vertex** - a node or entity in the graph
- **Edge** - a line that connects two vertices
- **Degree** - number of edges connecting a vertex to neighbors
- **Directed** edge - one-way relationship between two vertices
- **Undirected** edge - two-way relationship between vertices
- Directed graph (**digraph**) - a graph with directed edges
- **Weighted** edge - an edge with a weight or cost value
- **Path** - a sequence of edges between two vertices
- **Cycle** - a path that begins & ends at the same vertex

Glossary, cont.

- **Connected** - graph has a path between every vertex
- **Complete** - every vertex is connected to every other vertex
- **Disconnected** - some vertices can't be reached from all other vertices
- **Breadth-first search** - visit each of a vertex's neighbors before visting neighbors of neighbors
- **Depth-first search** - follow a single path to its end before following other paths
- **Topological order** - vertices of a digraph listed in order of precedence
- **Adjacency matrix** - two-dimensional array of graph edges
- **Adjacency list** - graph edges stored as a vertex-indexed array of lists