Graph notes

Prof Bill - Apr 2020

These notes introduce graphs and various data structures used to represent them.

Sections:

1. Undirected graphs
2. Data structures
3. Directed graphs

Reading. Happily, our online resources are pretty strong here:

- ➢ Princeton Chapter 4 graphs is good, algs4.cs.princeton.edu/40graphs
- ➢ Animated graph algorithms (from our favorite site), www.cs.usfca.edu/~galles/visualization/Algorithms.html

Sedgewick also has both **slides** and **videos** in support of his data structures text. Here's the index to all of it:

algs4.cs.princeton.edu/lectures

If **video learning** is your thing, scroll down for these graph lectures: Lecture 12 Undirected Graphs, and Lecture 13 Directed Graphs.

I prefer **slides** (sometimes) without the video. Scroll down (again), to "Table of lectures". The slide PDF's are on the left.

Coming soon...we'll hit as many cool graph algorithms as we can chomp: shortest path, min spanning tree, search/traverse, topological sort, bipartite, etc.

thanks… yow, bill

1. Undirected graphs

Sedgewick reading:

- ❖ Section 4.1 Undirected Graphs, algs4.cs.princeton.edu/41graph
- ❖ Slides, algs4.cs.princeton.edu/lectures/keynote/41UndirectedGraphs-2x2.pdf

Terms: (from Sedgewick)

- graph, edges, vertices, *adjacent* vertices, edge *incident* on vertices, subgraph
- self-loop, parallel edges, vertex degree
- path, simple path, cycle, simple cycle, path/cycle length, connected vertices, connected graph
- acyclic graph, tree, forest, spanning tree, bipartite graph

More terms (not in Sedgewick):

- ➔ **weighted graph** - a graph where edges have an associated weight (example: a graph of cities, edge weights are distance between cities)

/* shorthand: verts = vertices */

Undirected Graph API (pretty dreamy: simple, efficient)

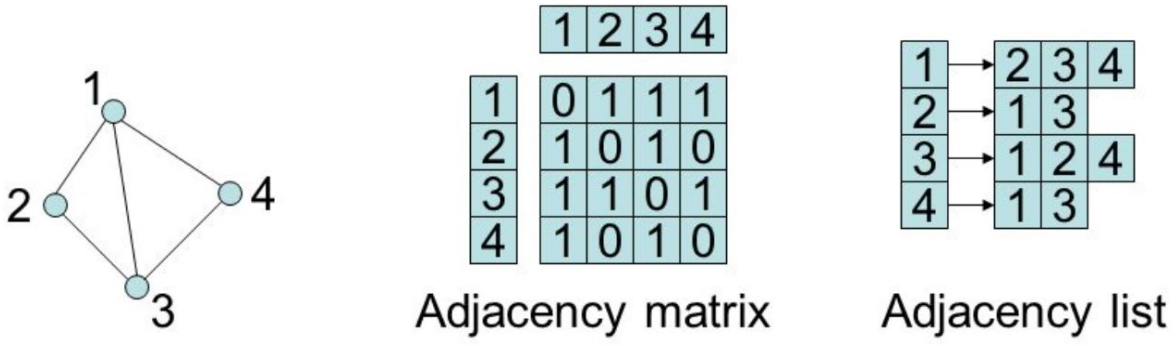
public class	Graph	
	Graph(int V)	*create a V-vertex graph with no edges*
	Graph(In in)	*read a graph from input stream* in
int	V()	*number of vertices*
int	E()	*number of edges*
void	addEdge(int v, int w)	*add edge* v-w *to this graph*
Iterable<Integer>	adj(int v)	*vertices adjacent to* v
String	toString()	*string representation*

API for an undirected graph

2. Data structures

Three most common graph data structures:

1. **adjacency list** - each vertex holds list of connected vertices
2. **adjacency matrix** - 2D array, size = (#verts x #verts), array slot[x,y] = 1 if edge exists between verts x and y
3. **edge list** - linked list (or ArrayList) of edges, each edge is a vert pair: (u, v)



Source: bournetocode.com/projects/AQA_A_Theory/pages/graph.html

For example above, edge list is: `{ (1,2), (1, 3), (1, 4), (2, 3), (3, 4) }`

Sparse graphs: use adjacency list. Dense graph: use adjacency matrix.

Sparse graph = large num verts, small average vert degree.

If verts have names, use symbol table (hash table) to get int from vert name

3. Directed graph

Princeton reading:

- ❖ Section 4.2 Directed Graphs, algs4.cs.princeton.edu/42digraph
- ❖ Slides: algs4.cs.princeton.edu/lectures/keynote/42DirectedGraphs-2x2.pdf

Terms:

- in-degree, out-degree
- directed path, directed cycle, length of a path (# edges), reachable vertex, strongly connected
- dag = directed acyclic graph, topological sort

Directed graph data structure and API - very similar to undirected… but edges have direction.