Week 8 Notes

Prof Bill - May 2018

Week 8 is one lecture... Intro to graphs.

Zee plan.

- Week 8 - define terms and data structures for graphs
- Week 9, 10 - *g*raph algorithms like shortest path, min spanning tree, search, topological sort, bipartite, etc

We're leaving Muganda-land… graphs aren't covered in the Muganda text.
Fortunately, our online resources are strong here:

- Princeton Chapter 4 graphs is good, algs4.cs.princeton.edu/40graphs
- Princeton lecture notes (slide) are very nice, too, algs4.cs.princeton.edu/lectures
- Animated graph algorithms (from our favorite site), www.cs.usfca.edu/~galles/visualization/Algorithms.html

thanks… yow, bill

A. Graphs

** Online: Princeton Chapter 4 is excellent, algs4.cs.princeton.edu/40graphs/
** Animation: www.cs.usfca.edu/~galles/visualization/RedBlack.html

4.1 Undirected Graphs

Princeton Reading:

- ❖ Section 4.1 Undirected Graphs, algs4.cs.princeton.edu/41graph/
- ❖ Section 4.1 slides, algs4.cs.princeton.edu/lectures/41UndirectedGraphs.pdf
    - ➢ 4 slides/page, algs4.cs.princeton.edu/lectures/41UndirectedGraphs-2x2.pdf

Terms: (from Princeton reading)

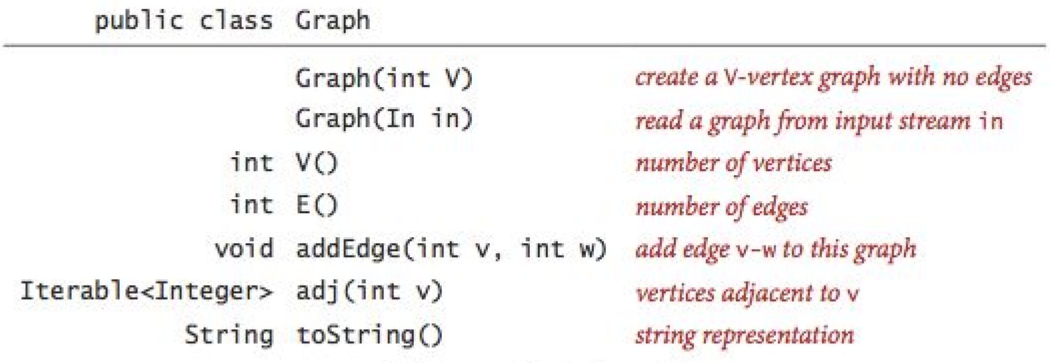
- graph, edges, vertices, *adjacent* vertices, edge *incident* on vertices, subgraph
- self-loop, parallel edges, vertex degree
- path, simple path, cycle, simple cycle, path/cycle length, connected vertices, connected graph
- acyclic graph, tree, forest, spanning tree, bipartite graph

More terms (not in Princeton):

- ➔ **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



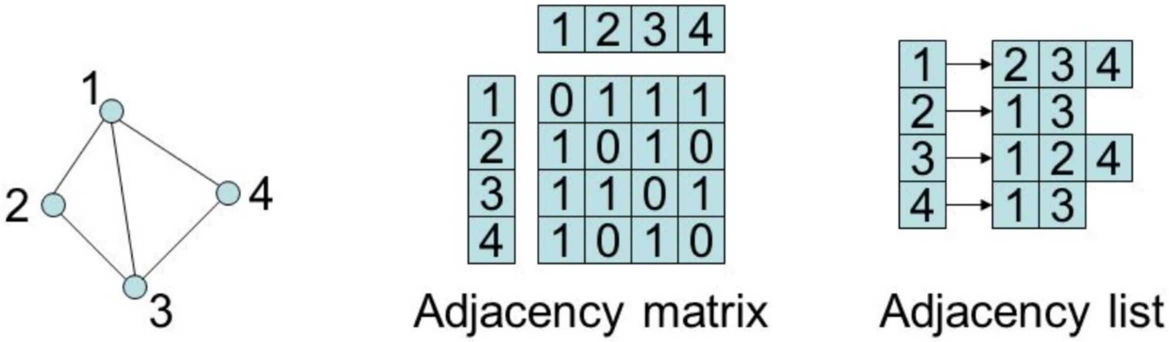
public class	Graph	
	Graph(int V)	*create a* V*-vertex graph with no edges*
	Graph(In in)	*read a graph from input stream* in
int	V()	*number of vertices*
int	E()	*number of edges*
void	addEdge(int v, int w)	*add edge* v-w *to this graph*
Iterable<Integer>	adj(int v)	*vertices adjacent to* v
String	toString()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

API for an undirected graph

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)



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

Adjacency matrix

1 →	2 3 4
2 →	1 3
3 →	1 2 4
4 →	1 3

Adjacency list

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

Traversal

Traversal/search = visit all verts in the graph or all connected verts (subgraph)

Depth-first search (DFS) - key concept: it's recursive!
Pseudocode:

```
// mark vertex v as visited, then recursively visit all connected verts
// prior to first dfs call, mark all verts as not visited
dfs( vertex v)
    mark v as visited
    for each vert w: adjacent to v {
        if w not visited
            dfs( w)
    }
```

Animation: www.cs.usfca.edu/~galles/visualization/DFS.html

Breadth-first search (BFS) - key concept - use a queue!

Pseudocode:

```
// use queue to do a breadth-first traversal of graph
bfs( vertex v)
    mark all verts not visited
    q = new queue
    q.enqueue( v)
    mark v as visited
    while ! q.isEmpty() {
        v2 = q.dequeue()
        for each vert w: adjacent to v2 {
            if w not visited
                q.enqueue( w)
                mark w as visited
        }
    }
```

Animation: www.cs.usfca.edu/~galles/visualization/BFS.html

Question: In earlier DFS pseudocode, can we remove recursion?
Answer: Yes! Use a stack, similar to the use of a queue in BFS,
www.mathcs.emory.edu/~cheung/Courses/171/Syllabus/11-Graph/dfs.html

4.2 Directed Graphs

Princeton reading:

- Section 4.2 Directed Graphs, algs4.cs.princeton.edu/42digraph
- Section 4.2 slides, algs4.cs.princeton.edu/lectures/42DirectedGraphs.pdf
  - 4 slides/page, algs4.cs.princeton.edu/lectures/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 - practically the same as undirected… but edges have direction.