Section 6.4: Traversal Algorithms

March 23, 2005

Abstract

We've already examined some tree traversal algorithms (pre-order, in-order, post-order), and considered their relative advantages. We now want to open the notion of traversal to all graphs (we certainly might want to write out the nodes of an arbitrary graph!). We examine and compare two recursive methods: depth-first and breadth-first.

Note: we're only covering 6.4 through Practice 16 (p. 453).

Important Convention: for the problems, we should stick with the convention that, given a choice, we should choose nodes in alphabetic order. This assures that we all end up with the same answer, which maximizes sanity....

1 Depth-First versus Breadth-First Traversal

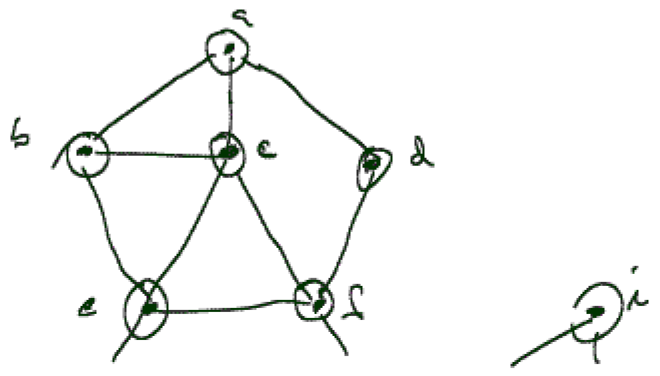
1.1 Depth-First

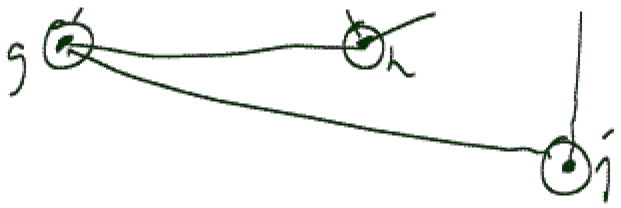
The idea behind the depth-first strategy is to burrow down into the graph, rather than spread out as one will in a breadth-first traversal. The depth-first algorithm is recursive. Have a look at the algorithm on p. 448.

1. Pick (mark and write) the start node;
2. Find its neighbor nodes (ordering them lexigraphically, again for sanity's sake!);

write

d
a
b
c
e
f
h





g
j
i

3. For each unmarked neighbor x, DepthFirst(G,x)

Exercise #3, p. 456

1.2 Breadth-First

Examine the breadth-first algorithm on p. 450. It uses a queue to traverse the nodes, popping elements off the queue as all of their adjacent nodes are also marked.

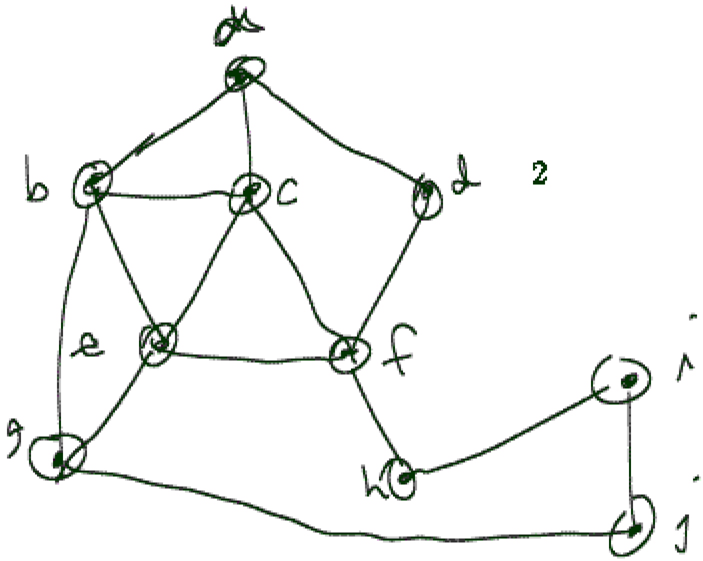
1. Pick (mark, write, and enqueue) the start node; then, while the queue is non-empty,
2. Find the front-of-the-queue's neighbor nodes (ordering them lexigraphically to be kind);
3. Mark, write, and enqueue those which are as yet unmarked;
4. Dequeue the front element of the queue;
5. Continue until the queue is empty.

Exercise #13, p. 457

2 How do these graph traversal algorithms behave for trees?

Look at an example (try a binary tree).

- Depth-first equates to preordering;
- Breadth-first does just what you'd expect! From the root on down, by depth.



M/W	Q
d	~~d~~
a	~~a~~
f	~~f~~
b	~~b~~
c	~~c~~
e	~~e~~
h	~~h~~
g	~~g~~

[illegible] [illegible]

3 Depth-First Application

These types of traversal algorithms are useful for operating on graphs. For example, I wrote this lisp code to find the shortest distance between two nodes x and y, using a depth-first algorithm (recursively). The algorithm is not particularly good; it was implemented because a student brainstormed it in a previous class, and it was a neat (albeit not particularly efficient) idea. It works like this:

- Start at the begin node;
- Find all adjacent nodes;
- Find the shortest distance from each of those nodes (recursively) to the destination node using a trimmed graph in which the start node has been eliminated (marked), and marking each one as finished once its shortest distance has been determined.

Note that I used the adjacency matrix representation, which is good for "full" graphs, but wasteful for sparse ones.

To test my algorithm, I ran it on the graph of Exercise #15, p. 444 for every pair of nodes (to compare the result with Floyd's algorithm). You can try out this procedure using this this web script.

#11