Section 7.4: Traversal Algorithms

March 30, 2021

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 simple, connected 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 graph traversals. In the end, we'll see how they work on trees, and how they relate to those algorithms. In a sense, we're creating a tree by writing out the nodes of a graph without repeats (without cycling).

Note: we're only covering 7.4 through Practice 16 (p. 602).

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

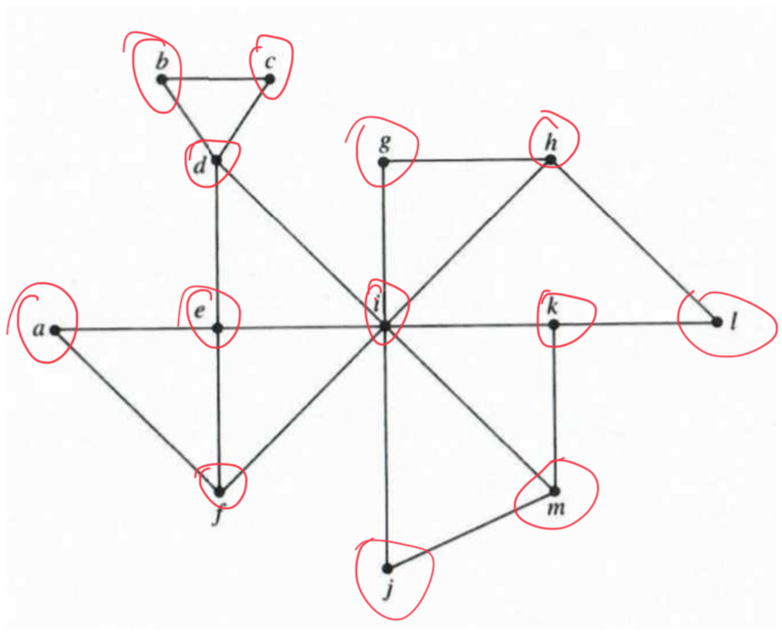
1.1 Depth-First

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. 597.

a. Pick (mark and write) the start node;

b. Find its neighbor nodes (ordering them lexigraphically, again for sanity's sake!);

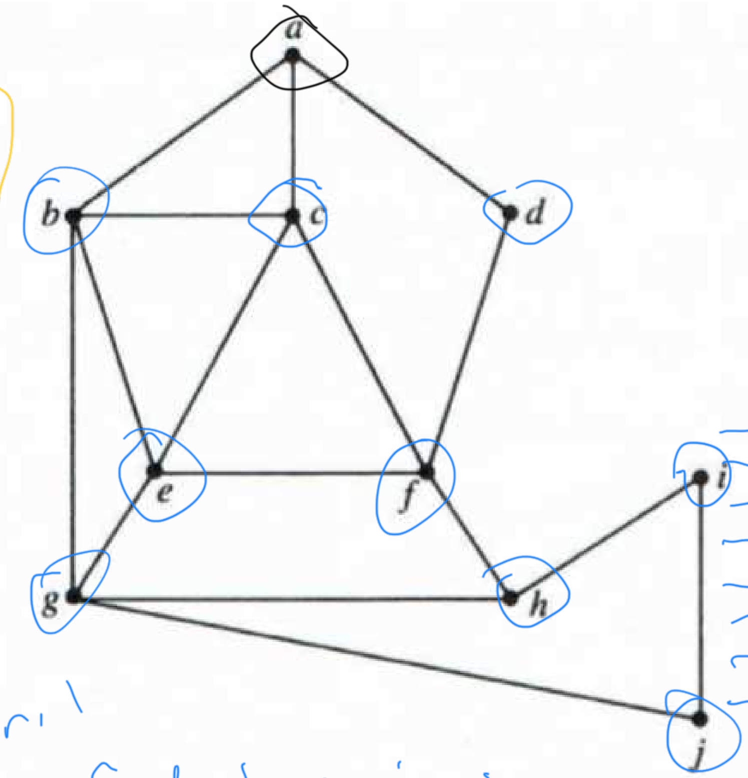
c. For each unmarked neighbor x, DepthFirst(G,x)



Marked	Adj
a	(~~e~~, ~~f~~)
e	(~~d~~, ~~f~~, ~~i~~)
d	(~~b~~, ~~c~~, ~~i~~)
b	(~~c~~)
c	nil
i	(~~f~~, ~~g~~, ~~j~~)
f	nil
g	(~~h~~)
h	(~~l~~)
l	(~~k~~)
k	(~~m~~)
m	(~~j~~)
j	nil

Figure 1: Practice 14, p. 598. Write the nodes in a depth-first search beginning from node a.

(b, c, d)
(c, e, g)
(e, f)
(f, g)
(~~d~~, h)
nil
(g, i)
(j)
(i)
nil



a, b, c, e, f, d, h, g, j, i

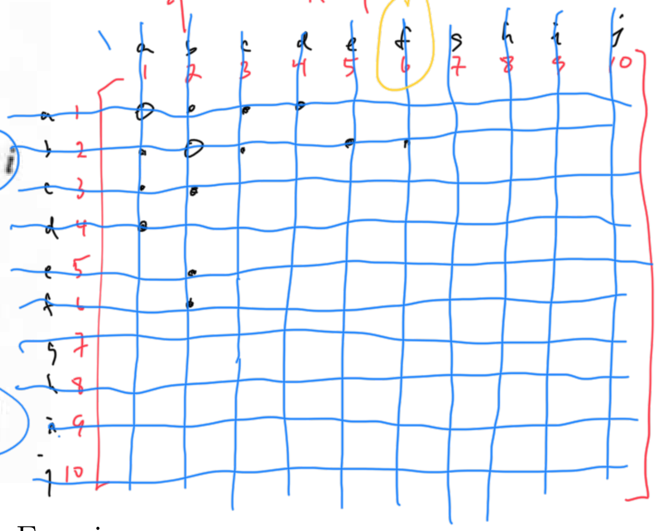


Figure 2: Exercise #1, p. 604. Ditto Practice 14. (Graph for Exercises 1-6, p. 604)

$$C_n = kC_{n-1} + b$$

1.2 Breadth-First

Examine the breadth-first algorithm on p. 599. It uses a queue to traverse the nodes, popping elements off the queue as all of their adjacent nodes are also marked.

a. Pick (mark, write, and enqueue) the start node; then, while the queue is non-empty,

b. Find the front-of-the-queue's neighbor nodes (ordering them lexigraphically to be kind);

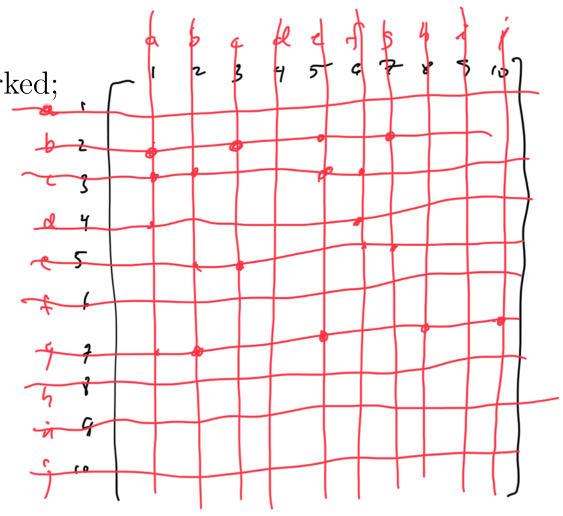
c. Mark, write, and enqueue those which are as yet unmarked;

d. Dequeue the front element of the queue;

e. Continue until the queue is empty.

Mark/write

~~a~~
~~b~~
~~c~~
~~d~~
~~e~~
~~g~~
~~f~~
~~h~~
~~j~~
~~i~~



Example: #11, p. 605

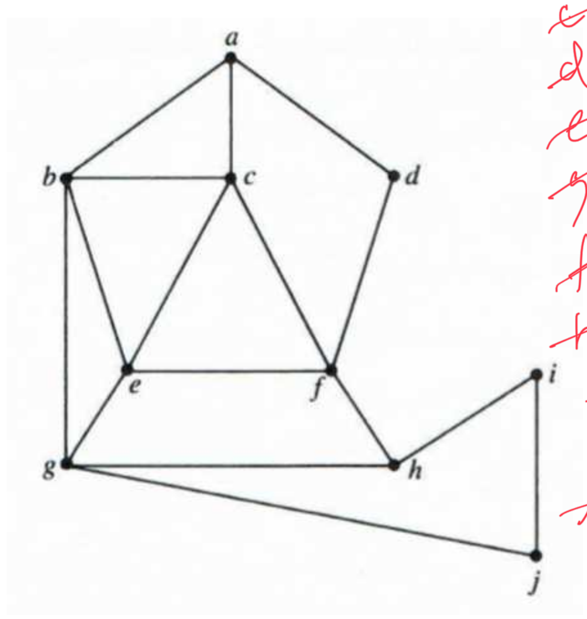


Figure 3: Graph for Exercises 1-6, p. 604

2 How do these graph traversal algorithms behave for trees?

Let's look at an example: the tree of Figure 6.46, p. 517.

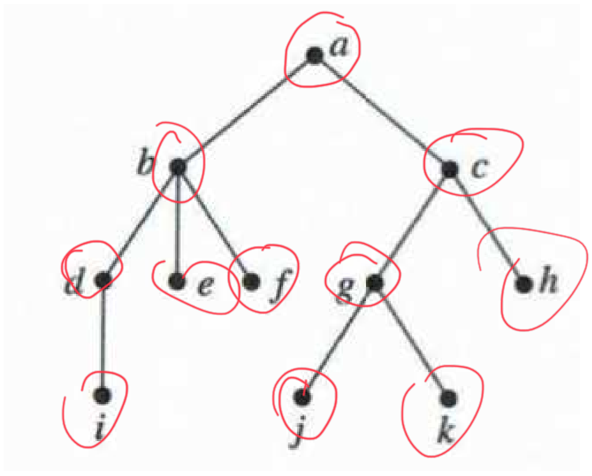
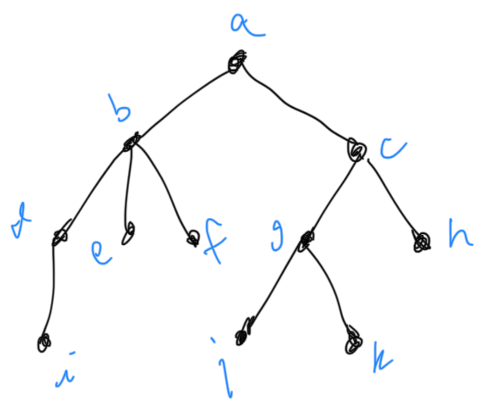


Figure 4: Figure 6.46, p. 517

- Depth-first equates to preordering;

a, b, d, i, e, f, c, g, j, k, h



Marked/write

~~a~~
~~b~~
~~c~~
~~d~~
~~e~~
~~f~~
~~g~~
~~h~~
~~i~~
~~j~~
~~k~~

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