CS250 Winter 2003

Week 10, Lecture 2:
Breadth-first search (BFS)
and depth-first search (DFS)

Goodrich & Tamassia, Section 10.3 (2nd ed)
or any intro to graph theory text
Basic graph algorithms

Graphs are data structures which are present throughout computer science, and algorithms for working with them are fundamental to the field. There are hundreds of interesting computational problems defined in terms of graph.

Searching, or traversing, a graph means systematically following the edges of the graph so as to visit its vertices. Searching is often used to discover features of the structure of a graph. There are two main ways of searching a graph, which are breadth-first search (BFS) and depth-first search (DFS).

In turn, many graph algorithms are applications of either type of searching. Sometimes, they start off by searching the input graph to get information about its structure. Other graph algorithms are elaborations of basic graph-searching algorithms.
Breadth-first search (BFS)

The strategy followed by breadth-first search is “fan out” from a source (starting) vertex. In other words, the explored paths grow by selecting frontier edges as close to the source vertex as possible. Breadth-first search is so named because it expands the frontier between undiscovered and discovered vertices uniformly across the breadth of the frontier.

Metaphor: paintbrush at the end of a string that is anchored at a source vertex. When all vertices at a level of string are painted, unroll the string one more level and paint the next vertices.

BFS is one of the simplest algorithms for searching a graph and is the archetype for many important graph algorithms.
BFS: standard vertex-labeling

Label the vertices of graph $G$ starting from source vertex $s$, in the order visited by a BFS.

```
BFSvertexLabel(G, s) {

    label[s] = 0
    T = {}  // init. set of traversed edges
    F = {frontier edges if s is in T}
    i = 1    // init. label counter

    while not all vertices in G are in T {
        e = (v,w) edge in F : smallest label on v
        label[w] = i
        T = T U {e}
        F = F U {neighbours of e not in T} \ {e}
        i = i + 1
    }
    return T
}
```

A frontier edge of $T$ is defined as an edge with one vertex in $T$ and the other not in $T$. 
BFS: vertex-labeling: illustration

\begin{itemize}
  \item \textbf{BFS:} vertex-labeling: illustration
\end{itemize}
BFS: level of exploration from source

Keep track of BFS progress by colouring the vertices white, gray or black.

- **White**: undiscovered vertices.

- **Non-white**: discovered vertices.
  - **Black**: all vertices adjacent to black ones have been discovered.
  - **Gray**: these vertices may have some white adjacent ones, i.e. they are the *frontier* between discovered and undiscovered vertices.

This traversal can be implemented using a queue. Why is this ADT a natural choice?
BFSvertexLevel(G, s) {
    for each u in V[G] \ {s} {
        colour[u] = white
        distance[u] = infinity
        predecessor[u] = nil
    }
    colour[s] = gray
    distance[s] = 0
    predecessor[s] = nil
    Q = {s}  // init. a queue Q to s

    while Q is non-empty {
        u = top(Q)
        for each v adjacent to u
            if colour[v] = white {
                colour[v] = gray
                distance[v] = distance[u] + 1
                predecessor[v] = u
                enqueue(Q, v)
            }
        dequeue(Q)
        colour[u] = black
    }
}
BFS: level of exploration: illustration
Depth-first search (DFS)

The strategy deployed by depth-first search is, as its name implies, to search “deeper” in the graph whenever possible. In DFS, the next edge to be explored is picked out of the most recently discovered vertex $v$ that still has unexplored edges leaving it. When all the edges of $v$ have been explored, the search “backtracks” to explore edges leaving the vertex from which $v$ was discovered. This process is continued until all reachable vertices have been discovered. If there are unreachable vertices remaining, the process is repeated by selecting one of them as a new source.

Metaphor: More adventurous than BFS, depth-first search is comparable to wandering in a labyrinth with a string and a can of paint, hence without getting lost.

DFS is, as well as BFS, one of the simplest algorithms for searching a graph and is the archetype for many important graph algorithms.
DFS: standard edge-labeling

Label the vertices of graph $G$ starting from source vertex $s$, as “discovery edges” and “back edges” in a DFS. Suppose here that $G$ is connected.

```c
DFSedgeLabel(G,v) {
    for each edge e incident on v {
        if e = (v,w) is unexplored
            if w is unexplored {
                label[e] = discovery
                DFSlabel(G,w)
            }
            else
                label[e] = back
    }
}
```

This is a recursive implementation of a DFS algorithm.
DFS: edge-labeling: illustration

The vertex labels are timestamps, including backtracks.
DFS: standard vertex-labeling

Label the vertices of graph $G$ starting from source vertex $s$, in the order visited by a DFS.

$$\text{DFSvertexLabel}(G,s) \{$$
$$\text{DFnumber}[s] = 0 \quad i = 1 \quad \text{// init. label counter}$$
$$T = \{\} \quad \text{// init. set of traversed edges}$$
$$S = \{\text{stack of frontier edges if } s \text{ is in } T\}$$

while not all vertices in $G$ are in $T$ {
$$\text{do } e = \text{pop}(S) \text{ while } e \text{ is not a frontier edge}$$
$$T = T \cup \{e\}$$
$$\text{DFnumber}[\text{unlabeled endpoint of } e] = i$$

for each frontier edge $f$
$$\text{if } \text{DFnumber}[\text{labeled endpoint of } f] = i$$
$$\text{push}(S, f)$$
$$i = i + 1$$
$$\}$$
$$\text{return DFnumber[ ]}$$
$$\}$$

This traversal can be implemented using a stack. Why is this ADT a natural choice?
DFS: vertex-labeling: illustration
Algorithms that use a B/DFS strategy

Some problems solved by B/DFS algorithms:
- Is graph $G$ connected?  - Is graph $G$ bipartite: can its
vertices can be partitioned into two sets with no edges
within them?  - Is there a cycle in $G$?  If so, what is it?

In general, both BFS and DFS are potentially useful whenever every vertex of a graph
needs to be visited, but either may be a bet-
ter, or only, strategy in solving some problems.

BFS: To find the shortest path between ver-
tices $s$ and $u$ in graph $G$, run BFSvertexLevel($G$, $s$), then $\text{length} = d[u]$ (proof omitted). Also,
to get out of a maze that may be infinite...
- What is the diameter of tree $T$, i.e. what is the length
of the largest of all shortest paths in $T$?

DFS: To find a topological sort of dag $G$, i.e. a
linear ordering of all its vertices such that if $G$
contains an edge $(u, v)$ then $u$ appears before $v$
in the ordering. Used to indicate precedences
among events, e.g.: wear socks before shoes.
Comparison of BFS and DFS

BFS:

DFS: