Computer Science 331

Graph Search: Depth-First Search

Mike Jacobson

Department of Computer Science
University of Calgary

Lecture #31

Outline

Connected Components

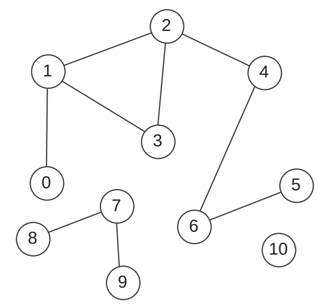
Recall that a graph $G = (V, E)$ is a *connected graph* if there is a path from u to v for every pair of vertices $u, v \in V$.

More generally, a *connected component* of a graph G is an induced subgraph $\widehat{G} = (\widehat{V}, \widehat{E})$ of G such that

- $\widehat{G}$ is a connected graph, and
- $\widehat{G}$ is "maximal:" It is not possible to add any more vertices or edges of G to $\widehat{G}$ in order to produce a larger subgraph of G that is still connected

Connected Components: An Example

Suppose G is the following graph



G has three connected components with vertex sets $\{0, 1, 2, 3, 4, 5, 6\}$, $\{7, 8, 9\}$, and $\{10\}$.

Computational Problems

Given an undirected graph $G = (V, E)$,

- Decide whether G is connected (ie, whether it has only *one* connected component, consisting of the entire graph)
- Find (and list) each of the connected components of G

Given an undirected graph $G = (V, E)$ and a vertex $s \in V$,

- Find the connected component of G that contains s
- Find a *spanning tree* for the connected component of G that contains s

"Graph Search" Problems

Graph Search — or *Graph Traversal* — refers to problems in which we wish to visit some, or all, of the vertices in a graph in a particular order.

Algorithms for graph search are important mainly because we need to use them to solve *other* (more interesting) problems.

In the version of a "graph search" problem to be given next the output will be a structure associated with a search — a *spanning tree* including the edges that have been followed in order to reach all the nodes that can be visited.

The primary application we will consider in these notes will be the discovery of a connected component of a graph; other applications are mentioned in suggested readings.

"Graph Search:" Specification of Requirements

Precondition:

- $G = (V, E)$ is an undirected graph and $s \in V$

Postcondition:

- The value returned is (a representation of) a function $\pi : V \to V \cup \{NIL\}$
- The predecessor subgraph $G_p = (V_p, E_p)$ corresponding to the vertex s and the function π is a spanning tree for the connected component of G that includes the vertex s
- The graph G has not been changed

Depth-First Search

Algorithm to search a graph in *depth-first* order:

- Given a graph G and a vertex s, the algorithm finds the *depth-first tree*, that is, a tree with root s whose edges are chosen by searching as deeply down a path as possible before "backtracking."

Applications include:

- finding the connected components of a graph
- solving puzzles (including some mazes) that have only one solution

Problem and Idea to Overcome It

Problem: graphs can have *cycles* and we need to avoid following cycles (resulting in infinite loops)

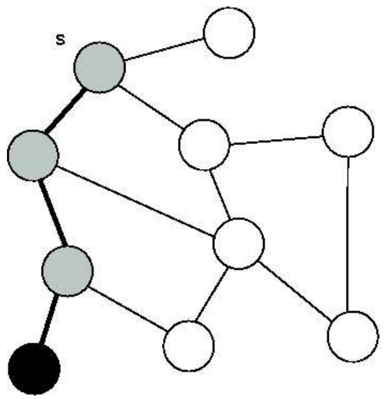
Solution: keep track of the nodes that have been visited already, so that we don't visit them again

Details:

- initially all vertices are **white**
- carry out the following steps, beginning with node s.
  - Colour a node grey when a search from the node begins:
  - recursively search from each white neighbour (reachable by following an edge in the "forward" direction)
  - end the search by colouring the node black.

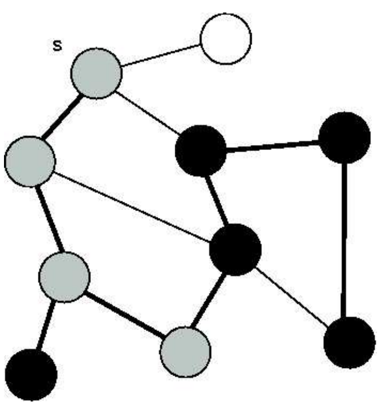
Typical Search Pattern

Pattern Near Beginning of Search:



Typical Search Pattern

Pattern Farther Along in Search:



Data and Pseudocode

The following information is maintained for each $u \in V$:

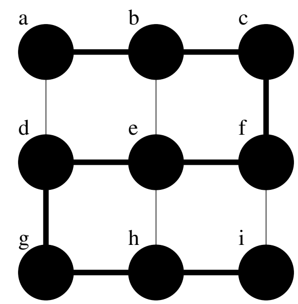
- $colour[u]$: Colour of u
- $\pi[u]$: Parent of u in tree being constructed

DFS$(G = (V, E), s)$
  {Initialization — all nodes initially white (undiscovered)}
  for each vertex $u \in V$ **do**
    $colour[u]$ = white
    $\pi[u]$ = NIL
  end for
  {Visit all vertices reachable from s}
  DFS-Visit(s)
  return π

Pseudocode, Continued

DFS-Visit(u)
  colour$[u]$ = grey $\{u$ is discovered, but not all neighbours$\}$
  for each $v \in Adj[u]$ **do**
    if *colour*$[v]$ == white **then**
      $\pi[v] = u$ {record parent of newly-discovered vertex}
      DFS-Visit(v) {visit each undiscovered neighbour recursively}
    end if
  end for
  colour$[u]$ = black $\{u$ finished (all children discovered)$\}$

Example



	a	b	c	d	e	f	g	h	i
π	NIL	a	b	e	f	c	d	g	h

Behaviour of DFS-Visit

Let $u \in V$.

- If DFS-Visit is ever called with input u then *colour*$[u]$ = white immediately *before* this function is called with this input, and *colour*$[u]$ = black on termination, if this function terminates.

The following notation will be useful when discussing properties of this algorithm.

- Consider the *colour* function just **before** DFS-Visit is called with input u. Let
  - $V_u = \{v \in V \mid colour[v] = \text{white}\}$,
  - $G_u = (V_u, E_u)$ be the induced subgraph of G corresponding to the subset V_u.

Behaviour of DFS-Visit

Additional Useful Notation:

- Consider the function π immediately **after** this call to DFS-Visit terminates (if it terminates at all).
  - Let $\pi_u : V_u \to V_u \cup \{\text{NIL}\}$ such that, for a node $v \in V_u$,

$$\pi_u(v) = \begin{cases} \pi(v) & \text{if } v \neq u, \\ \text{NIL} & \text{if } v = u. \end{cases}$$

  - Let $G_{p,u} = (V_{p,u}, E_{p,u})$ be the predecessor subgraph of G_u corresponding to the function π_u and the vertex u.

Behaviour of DFS-Visit

Theorem 1

Suppose that this execution of DFS-Visit terminates. Then

- $G_{p,u}$ *is a depth-first tree for the graph* G_u *and the vertex* u.
- *The graph* G *has not been changed by this execution of DFS-Visit.*
- *If* $v \in V_u$ *then* $colour[v] = black$ *if* $v \in V_{p,u}$, *and* $colour[v] = white$ *otherwise*
- *If* $v \in V$ *but* $v \notin V_u$ *then neither* $colour[v]$ *nor* $\pi[v]$ *have been changed by this execution of DFS-Visit.*
- $\pi[u]$ *has not been changed by this execution of DFS-Visit.*

Method of proof.

Induction on $|V_u|$. □

Partial Correctness of DFS

Theorem 2

If DFS is executed with an input graph G *and vertex* $s \in G$ *then either the post-condition of the "Search" problem is satisfied on termination (with the spanning tree corresponding to the output having been produced in a "depth-first" manner) or the algorithm does not terminate at all.*

Method of Proof.

Notice that this follows by inspection of the code, using the result about DFS-Visit that has just been established. □

Termination and Running Time

Theorem 3

Suppose $G = (V, E)$ *is a directed or undirected graph, and suppose DFS is run on* G *and a vertex* $v \in S$. *Then the algorithm terminates after* $\Theta(|V| + |E|)$ *operations.*

Sketch of Proof.

- **DFS-Visit** is called exactly once for each $u \in V$
  - only called if a vertex is white
  - vertex u is coloured grey when **DFS-Visit**(u) is executed, and never coloured white again.
- Total cost of **DFS-Visit**, minus recursive calls, is linear in $1 + \deg u$
  - $\sum_{u \in V} (\deg u + 1) = 2|E| + |V|$

Thus, total running time is $\Theta(|V| + |E|)$. □

Iteration in Depth-First Order

Some applications require that the vertices in a graph that are reachable from a vertex s be visited in a "depth-first" order.

One such ordering, called "discovery order" or "preordering," can be produced by modifying our algorithm as follows:

- Delete references to the array π (this is no longer needed)
- Visit a node as soon as it is coloured `grey`

The worst-case cost is in $\Theta(|V| + |E|)$ once again

Another useful ordering, called "finish order" or "postordering," is obtained by visiting each node when its colour is changed to `black`. The algorithm could also be changed to produce this ordering without significantly changing its worst-case running time.

References

Text, Section 13.3

Introduction to Algorithms, Section 22.3: More details about the version of the algorithm presented here.