Graph Search

CS 214, Fall 2019
Questions we might ask about graphs

• Is there a path from $v$ to $u$?
• What’s the shortest path from $v$ to $u$?
• Are there any cycles?
Graph search: basic idea

To answer whether there’s a path (among other things), we can use:

- Depth-first search (DFS): go as far as you can along a path, then go back and try anything you haven’t tried yet
- Breadth-first search (BFS): explore all the successors of a vertex before exploring their successors in turn
DFS example
DFS example
DFS example

![DFS example diagram](image-url)
DFS example
DFS example
DFS example
DFS example
DFS example
DFS example
DFS example
DFS example
DFS example
DFS example
DFS example
DFS example
DFS example
Recursive DFS algorithm (one source)

Procedure DFS(graph, start) is
    seen ← new array (same size as graph, filled with false);

Procedure Visit(v) is
    if not seen[v] then
        seen[v] ← true;
        for u in Successors(graph, v) do
            Visit(u)
        end
    end
end

Visit(start);
return seen
Recursive DFS algorithm (one source, lifted)

Procedure $\text{Visit}(graph, seen, v)$ is
  if not $seen[v]$ then
    $seen[v] \leftarrow$ true;
    for $u$ in $\text{Successors}(graph, v)$ do
      $\text{Visit}(graph, seen, u)$
    end
  end
end

Procedure $\text{DFS}(graph, start)$ is
  $seen \leftarrow$ new array (same size as graph, filled with false);
  $\text{Visit}(graph, seen, start)$;
  return $seen$
end
Recursive DFS algorithm (1 src., builds tree)

Procedure DFS(graph, start) is
  preds ← new array (same size as graph, filled with false);
  Procedure Visit(pred, v) is
    if not preds[v] then
      preds[v] ← pred;
      for u in Successors(graph, v) do
        Visit(v, u)
      end
    end
  end
  Visit(true, start);
  return preds
end
Recursive DFS algorithm (full)

Procedure DFS(graph) is
    preds ← new array (same size as graph, filled with false);
    Procedure Visit(pred, v) is
        if not preds[v] then
            preds[v] ← pred;
            for u in Successors(graph, v) do
                Visit(v, u)
            end
        end
    end
    for v in Vertices(graph) do
        Visit(true, v)
    end
    return preds
end
Iterative DFS algorithm

Procedure DFS($\text{graph, start}$) is

$preds$ ← new array (same size as graph, filled with false);
$todo$ ← new stack;

$preds[start]$ ← true;
Push($todo$, start);

while $todo$ is not empty do

$v$ ← Pop($todo$);
for $u$ in Successors($\text{graph, v}$) do

if not $preds[u]$ then

$preds[u]$ ← $v$;
Push($todo$, $u$)

end

end

end

return $preds$
Running DFS on a digraph

tree
back
cross
forward
Running DFS on a digraph

tree

back

cross

forward
Running DFS on a digraph

tree
back
cross
forward
Running DFS on a digraph

tree
back
cross
forward
Running DFS on a digraph
Running DFS on a digraph

tree
back
cross
forward

g ← f ← e ← h

c ← d ← b ← a
Running DFS on a digraph

tree

back

cross

forward
Running DFS on a digraph

tree
back
cross
forward

10
Running DFS on a digraph

tree

back

cross

forward
Running DFS on a digraph

tree

back

cross

forward

10
Running DFS on a digraph

tree ➝
back ➟
cross ➟
forward ➟
Running DFS on a digraph

tree

back

cross

forward
Running DFS on a digraph

tree
back

cross
forward
Running DFS on a digraph

tree

back

cross

forward

g
f
h
e
a
b
c
d
Running DFS on a digraph

tree
back
cross
forward
Running DFS on a digraph

tree
back
cross
forward
Running DFS on a digraph

tree
back
cross
forward
Running DFS on a digraph

- **tree**
- **back**
- **cross**
- **forward**
A DFS tree

tree
back
cross
forward
DFS for cycle detection

Procedure FindCycle(graph) is
    started ← new array (same size as graph, filled with false);
    finished ← new array (same size as graph, filled with false);

Procedure Visit(v) is
    if not finished[v] then
        if started[v] then
            | we found a cycle!
        end
        started[v] ← true;
        for u in Successors(graph, v) do
            | Visit(u)
        end
        finished[v] ← true;
    end

for v in Vertices(graph) do
    Visit(v)
end
Breadth-first search

Procedure BFS(graph, start) is

preds ← new array (same size as graph, filled with false);
todo ← new queue;

preds[start] ← true;
Enqueue(todo, start);

while todo is not empty do
    v ← Dequeue(todo);
    for u in Successors(graph, v) do
        if not preds[u] then
            preds[u] ← v;
            Enqueue(todo, u)
        end
    end
end

return preds
end
Running BFS on a digraph
Running BFS on a digraph
Running BFS on a digraph
Running BFS on a digraph
Running BFS on a digraph
Running BFS on a digraph
Running BFS on a digraph
Running BFS on a digraph
Running BFS on a digraph
Running BFS on a digraph
Generic graph search

If \textit{todo} is a stack we get DFS; if \textit{todo} is a queue we get BFS:

Procedure \texttt{Search}(\textit{graph, start}) is

\begin{verbatim}
  \textit{preds} \leftarrow \text{new array (same size as graph, filled with false)};
  \textit{todo} \leftarrow \text{new collection};

  \textit{preds}[\textit{start}] \leftarrow \text{true};
  \text{Add}(\textit{todo, start});

  \textbf{while} \textit{todo} is not empty \textbf{do}
    \textit{v} \leftarrow \text{Remove}(\textit{todo});
    \textbf{for} \textit{u in Successors}(\textit{graph, v}) \textbf{do}
      \textbf{if} not \textit{preds}[\textit{u}] \textbf{then}
        \textit{preds}[\textit{u}] \leftarrow \textit{v};
        \text{Add}(\textit{todo, u})
      \textbf{end}
    \textbf{end}

  \textbf{end}

  \textbf{return} \textit{preds}
\end{verbatim}

end
Next time: shortest paths