Graph Search

EECS 214, Fall 2018
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
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 \texttt{Visit}(graph, seen, v) is
\begin{verbatim}
if not seen[v] then
  seen[v] ← true;
  for u in Successors(graph, v) do
    Visit(graph, seen, u)
end
\end{verbatim}
end

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

Procedure DFS(\texttt{graph, start}) is
\begin{align*}
\textit{preds} & \leftarrow \text{new array (same size as graph, filled with false)}; \\
\text{Procedure Visit}(\textit{pred}, \texttt{v}) & \text{is} \\
& \text{if not } \textit{preds}[\texttt{v}] \text{ then} \\
& \quad \textit{preds}[\texttt{v}] \leftarrow \textit{pred}; \\
& \quad \text{for } u \text{ in Successors(\texttt{graph, v}) do} \\
& \quad \quad \text{Visit}(v, u) \\
& \quad \text{end} \\
& \text{end} \\
\text{Visit}(\texttt{true, start}); \\
\text{return } \textit{preds}
\end{align*}
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\( (\text{graph}, \text{start}) \) is
\[
\begin{align*}
\text{preds} & \leftarrow \text{new array (same size as graph, filled with false)}; \\
\text{todo} & \leftarrow \text{new stack}; \\
\text{preds}[\text{start}] & \leftarrow \text{true}; \\
\text{Push}(\text{todo, start}); \\
\text{while } \text{todo is not empty do} \\
& \quad \text{v} \leftarrow \text{Pop}(\text{todo}); \\
& \quad \text{for } u \text{ in } \text{Successors}(\text{graph, v}) \text{ do} \\
& \quad \quad \text{if not } \text{preds}[u] \text{ then} \\
& \quad \quad \quad \text{preds}[u] \leftarrow v; \\
& \quad \quad \quad \text{Push}(\text{todo, u}) \\
& \quad \text{end} \\
& \quad \text{end} \\
& \text{return } \text{preds} \\
\end{align*}
\]
Running DFS on a digraph

tree

back

cross

forward

g ← f ← e ← a

g ← f ← c ← d

g ← f ← c ← b

h ← e ← d

h ← e ← c ← b

h ← e ← c ← b

h ← e ← a ← c ← b

Running DFS on a digraph
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 ← e → h

c ← b ← a
Running DFS on a digraph

tree

back

cross

forward
Running DFS on a digraph

tree

back

cross

forward

g ← f

d ← c ← b ← a

e ← h
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

g

h

e

f

a

c

d

b

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
Running DFS on a digraph

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

tree
back
cross
forward

g → f → a → e → h
f → a → e → h
b → c → d → f

Running DFS on a digraph

tree

back

cross

forward
A DFS tree
DFS for cycle detection

Procedure FindCycle(graph) is

\(\text{started} \leftarrow \text{new array (same size as graph, filled with false)};\)
\(\text{finished} \leftarrow \text{new array (same size as graph, filled with false)};\)

Procedure Visit(v) is

if not \(\text{finished}[v]\) then
    if \(\text{started}[v]\) then
        \(\text{we found a cycle!}\)
    end
    \(\text{started}[v] \leftarrow \text{true};\)
    for \(u\) in Successors(graph, \(v\)) do
        Visit(u)
    end
    \(\text{finished}[v] \leftarrow \text{true};\)
end

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

Procedure \( \text{BFS}(\text{graph}, \text{start}) \) is

\[
\begin{align*}
\text{preds} & \leftarrow \text{new array (same size as graph, filled with false)}; \\
\text{todo} & \leftarrow \text{new queue}; \\
\text{preds}[\text{start}] & \leftarrow \text{true}; \\
& \text{Enqueue(} \text{todo}, \text{start} \text{);} \\
& \text{while } \text{todo is not empty do} \\
& \quad v \leftarrow \text{Dequeue(} \text{todo} \text{);} \\
& \quad \text{for } u \text{ in } \text{Successors(} \text{graph}, v \text{) do} \\
& \quad \quad \text{if not } \text{preds}[u] \text{ then} \\
& \quad \quad \quad \text{preds}[u] \leftarrow v; \\
& \quad \quad \quad \text{Enqueue(} \text{todo}, u \text{)} \\
& \quad \text{end} \\
& \text{end} \\
& \text{return } \text{preds} \\
\end{align*}
\]
Running BFS on a digraph
Running BFS on a digraph

The graph shows the Breadth-First Search (BFS) traversal starting from node 'a'. The visited nodes are marked in yellow ('a', 'b', 'e', 'h', 'g', 'f', 'c', 'd'). The traversal order is as follows: 'a' → 'b' → 'e' → 'h' → 'g' → 'f' → 'c' → 'd'.
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 *todo* is a stack we get DFS; if *todo* is a queue we get BFS:

Procedure `Search(graph, start)` is

1. `preds ← new array (same size as graph, filled with false);`
2. `todo ← new collection;`
3. `preds[start] ← true;`
4. `Add(todo, start);`

5. **while** `todo` is not empty do
6. 6.1 `v ← Remove(todo);`
7. 6.2 **for** `u in Successors(graph, v)` do
8. 6.2.1 if not `preds[u]` then
9. 6.2.2.1 `preds[u] ← v;`
10. 6.2.2.2 `Add(todo, u)`
11. 6.2.2 end
12. 6.2 end
13. 6.1 end

14. return `preds`
Next time: shortest paths