Graph Search
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
\[
\text{seen} \leftarrow \text{new array (same size as graph, filled with false)};
\]

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

Visit\((start)\);

return \text{seen}

end
Recursive DFS algorithm (one source, lifted)

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

Procedure DFS(graph, start) is
    seen ← new array (same size as graph, filled with false);
    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
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(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(graph, v) do
        if not preds[u] then
            preds[u] ← v;
            Push(todo, u)
        end
    end
end

return preds
end
Running DFS on a digraph

tree
back
cross
forward

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

diagram with nodes a, b, c, d, e, f, g, h and directed edges indicating tree, back, cross, and forward relationships.
Running DFS on a digraph

tree

back

cross

forward

tree

back

cross

forward

10
Running DFS on a digraph

tree

back

cross

forward

g

f

e

h

a

b

c

d
Running DFS on a digraph

- **tree**
- **back**
- **cross**
- **forward**

Diagram:
- Node a with directed edges to b, c, e, and h.
- Node b with directed edges to c and d.
- Node c with directed edges to b and d.
- Node d with directed edges to e.
- Node e with directed edge to f.
- Node f with directed edge to g.
- Node h with directed edge to e.
Running DFS on a digraph

tree

back

cross

forward

10
Running DFS on a digraph
Running DFS on a digraph

tree

back

cross

forward

10
Running DFS on a digraph
Running DFS on a digraph

tree

back

cross

forward
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
Running DFS on a digraph
Running DFS on a digraph
Running DFS on a digraph

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

dot diagram with nodes labeled a, b, c, d, e, f, g, h, and arrows indicating back, cross, and forward edges.
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
    end

    for v in Vertices(graph) do
        Visit(v)
    end
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

e h c

g ← f

h ← e

e ← a

b ← c

d ← a
Running BFS on a digraph
Running BFS on a digraph

c f
Running BFS on a digraph

A digraph is a directed graph, which means that the edges have a direction. In the diagram, nodes a, b, c, d, e, f, g, and h are connected by directed edges. The path taken during BFS is highlighted in blue and red.
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}(graph, start) is

\begin{itemize}
    \item $preds \leftarrow \text{new array (same size as graph, filled with false)}$;
    \item $todo \leftarrow \text{new collection}$;
    \item $preds[start] \leftarrow \text{true}$;
    \item \texttt{Add}(todo, start);
    \item \textbf{while} \texttt{todo} is not empty \textbf{do}
        \begin{itemize}
            \item $v \leftarrow \texttt{Remove}(todo)$;
            \item \textbf{for} $u$ in \texttt{Successors}(graph, $v$) \textbf{do}
                \begin{itemize}
                    \item \textbf{if} \texttt{not} $preds[u]$ \textbf{then}
                        \begin{itemize}
                            \item $preds[u] \leftarrow v$;
                            \item \texttt{Add}(todo, $u$)
                        \end{itemize}
                \end{itemize}
        \end{itemize}
    \item \textbf{return} $preds$
\end{itemize}
Next time: shortest paths