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
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
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
Running DFS on a digraph

tree

back

cross

forward

10
Running DFS on a digraph

- Tree edges: a -> b, b -> c, c -> d, d -> e, e -> f, f -> g
- Back edges: b -> a
- Cross edges: a -> h, h -> e, e -> a

1. Start at a.
2. Visit a, then b, then c, then d.
3. Back edge from b to a.
4. Cross edge from a to h.
5. Visit e.
6. Cross edge from e to a.
7. Visit f.
8. Cross edge from f to g.
9. Visit h.
10. Back edge from h to e.
11. Visit e, then a, then b, then c, then d.
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

Diagram:

- Nodes: a, b, c, d, e, f, g, h
- Edges:
  - a -> b
  - b -> c
  - c -> d
  - d -> a
  - e -> h
  - f -> g

Types of edges:
- Tree edges
- Back edges
- Cross edges
- Forward edges
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

g

h

e

f

a

b

c

d

10
Running DFS on a digraph
Running DFS on a digraph
Running DFS on a digraph

- Tree
- Back
- Cross
- Forward
Running DFS on a digraph

tree

back

cross

forward

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

tree

back

cross

forward

10
Running DFS on a digraph

Tree

Back

Cross

Forward

10
Running DFS on a digraph

tree

back

cross

forward
A DFS 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

  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

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