

Graph Search

EECS 214, Fall 2017

Questions we might ask about graphs

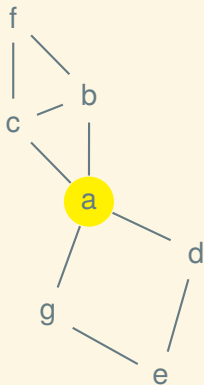
- Is there a path from v to u ?
- What's the shortest path from v to u ?
- Are there any cycles?

Graph search: basic idea

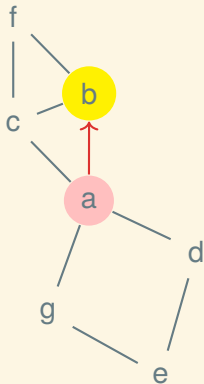
To answer whether there's a path (among other things), we can use:

- 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

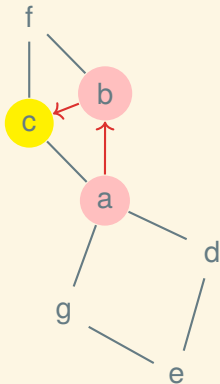
DFS example



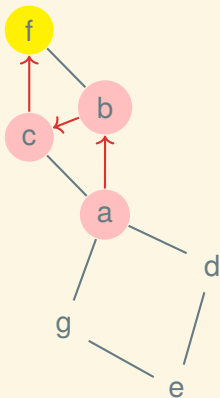
DFS example



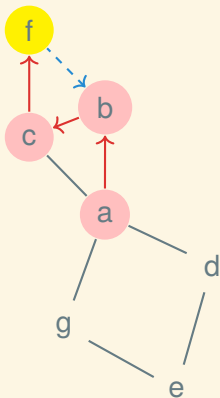
DFS example



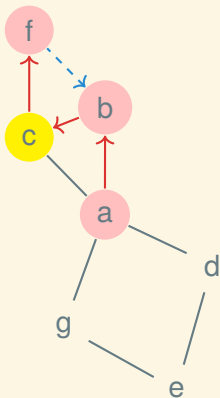
DFS example



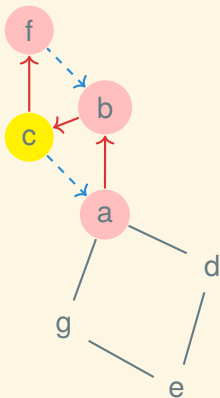
DFS example



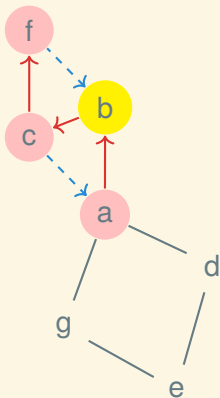
DFS example



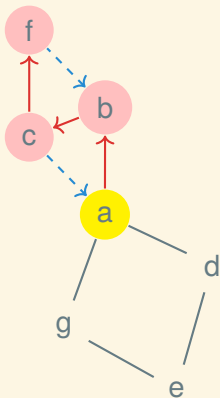
DFS example



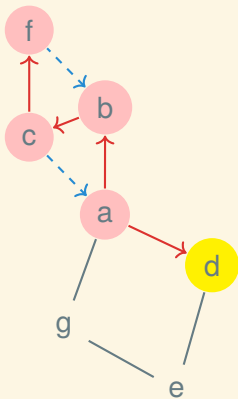
DFS example



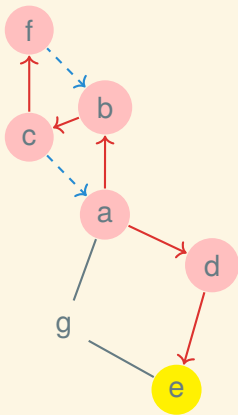
DFS example



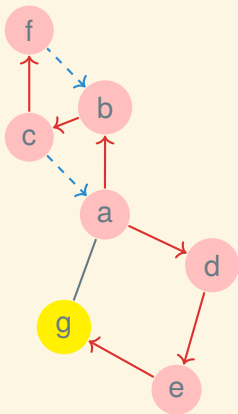
DFS example



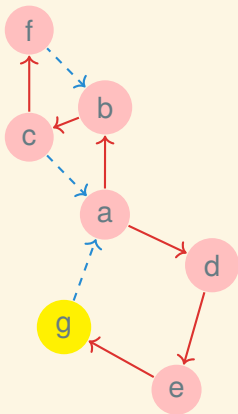
DFS example



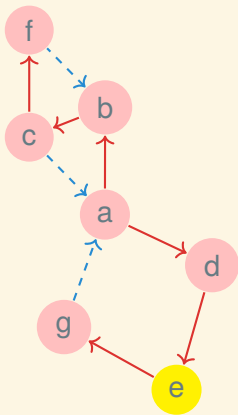
DFS example



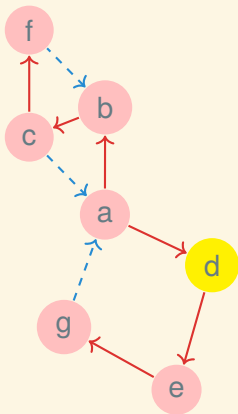
DFS example



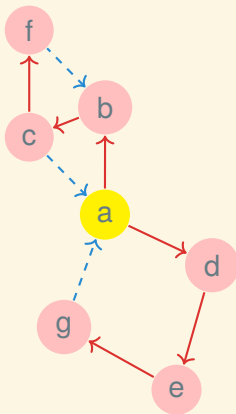
DFS example



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
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(*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

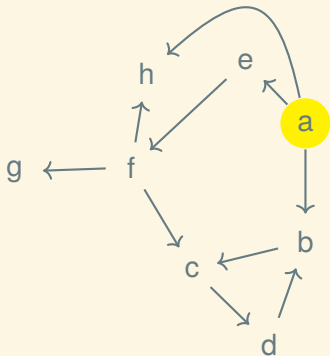
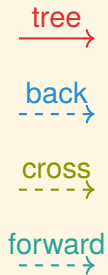
 end

 end

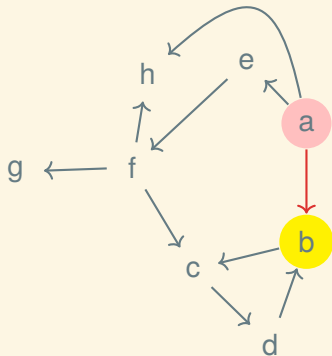
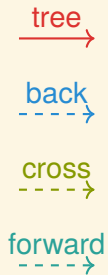
 return *preds*

end

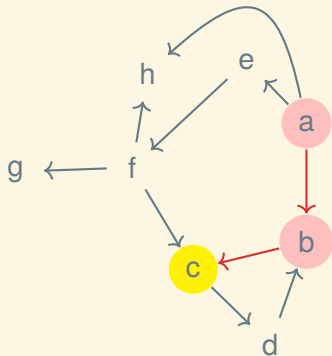
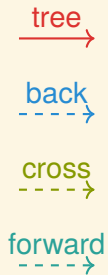
Running DFS on a digraph



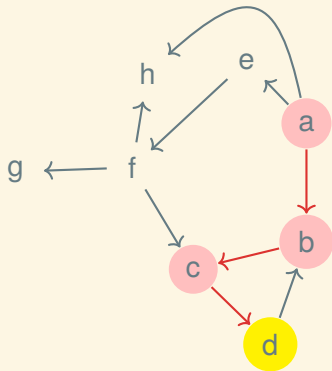
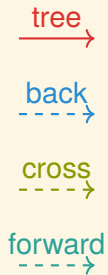
Running DFS on a digraph



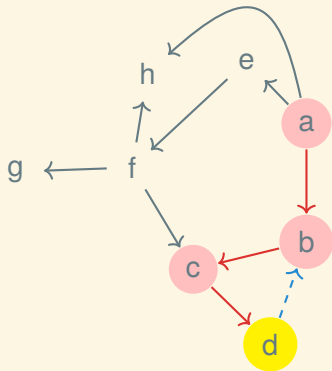
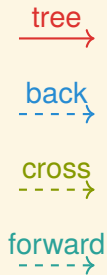
Running DFS on a digraph



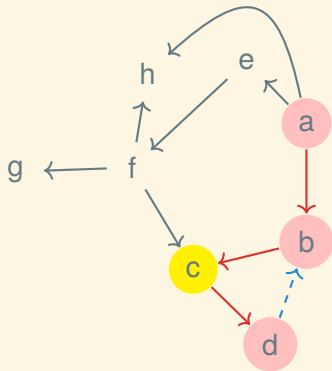
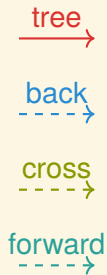
Running DFS on a digraph



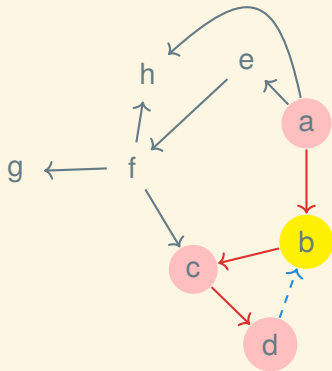
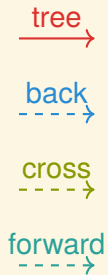
Running DFS on a digraph



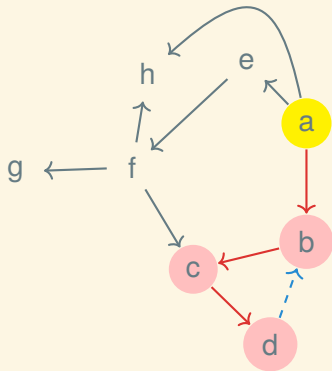
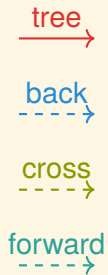
Running DFS on a digraph



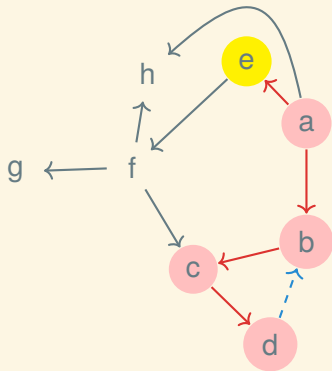
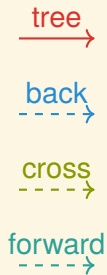
Running DFS on a digraph



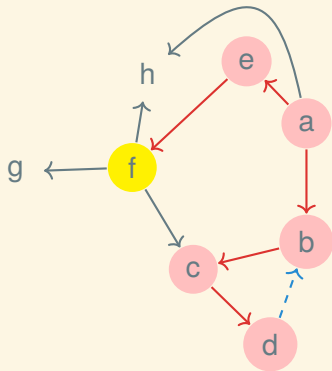
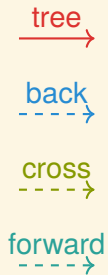
Running DFS on a digraph



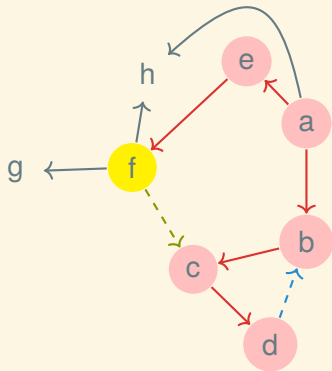
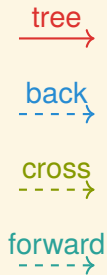
Running DFS on a digraph



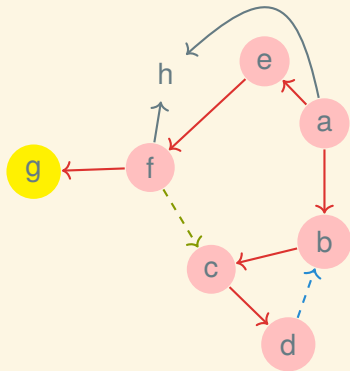
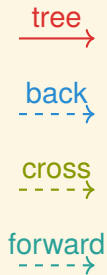
Running DFS on a digraph



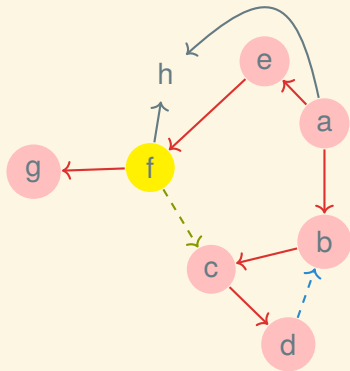
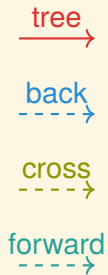
Running DFS on a digraph



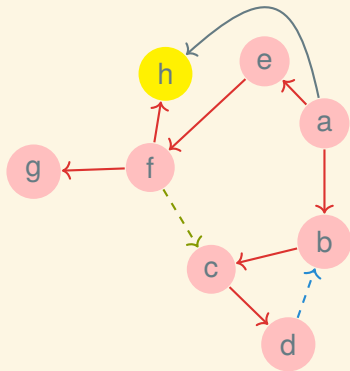
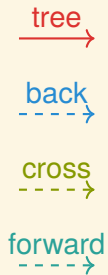
Running DFS on a digraph



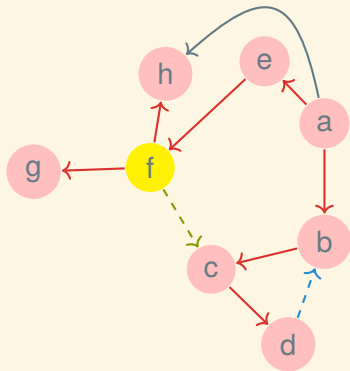
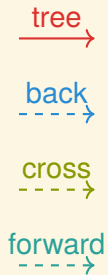
Running DFS on a digraph



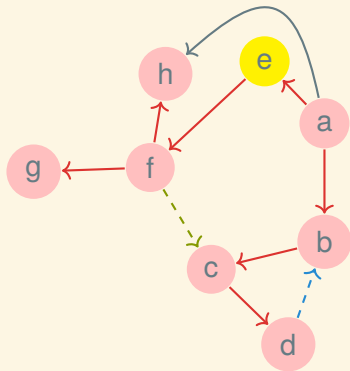
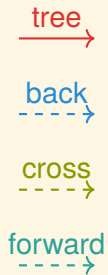
Running DFS on a digraph



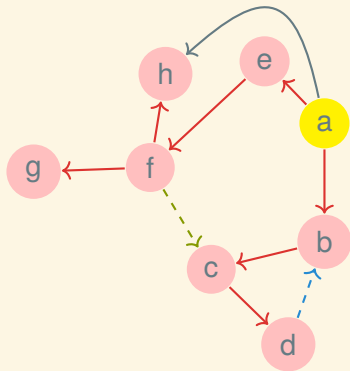
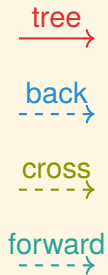
Running DFS on a digraph



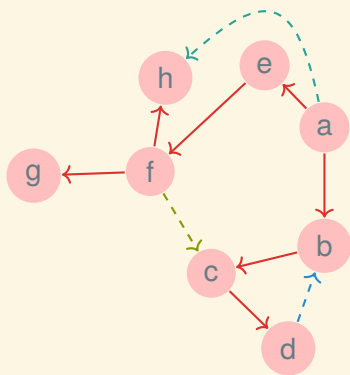
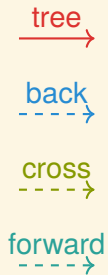
Running DFS on a digraph



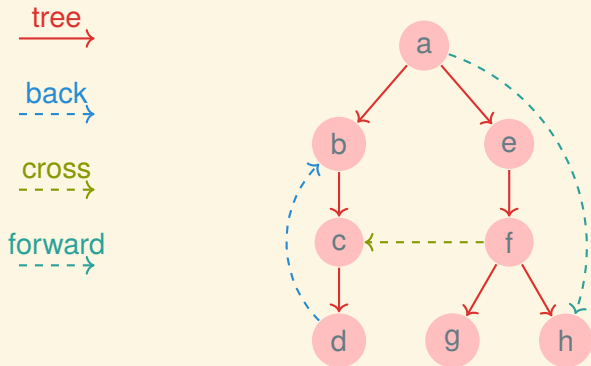
Running DFS on a digraph



Running DFS on a digraph



A DFS tree



DFS for cycle detection

Procedure FindCycle(*graph*) is

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

finished \leftarrow 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*] \leftarrow true;

 for *u* in Successors(*graph*, *v*) do

 Visit(*u*)

 end

finished[*v*] \leftarrow true;

 end

 end

 for *v* in Vertices(*graph*) do

 Visit(*v*)

 end

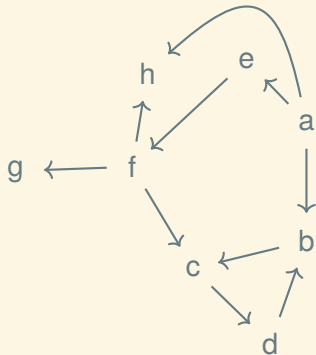
end

Breadth-first search

Procedure `BFS(graph, start)` is

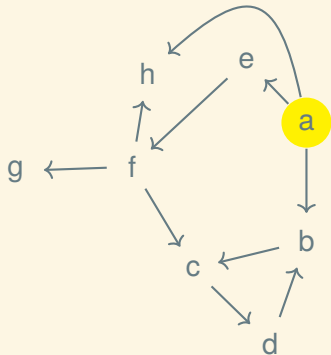
```
  preds ← new array (same size as graph, filled with false);  
  todo ← new queue;  
  
  preds[start] ← true;  
  Enqueue(todo, start);  
  
  while todo is not empty do  
    v ← Dequeue(todo);  
    for u in Successors(graph, v) do  
      if not preds[u] then  
        preds[u] ← v;  
        Enqueue(todo, u)  
      end  
    end  
  end  
  return preds  
end
```

Running BFS on a digraph



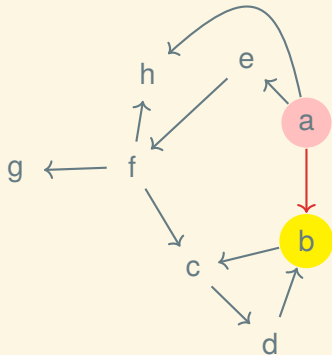
a

Running BFS on a digraph



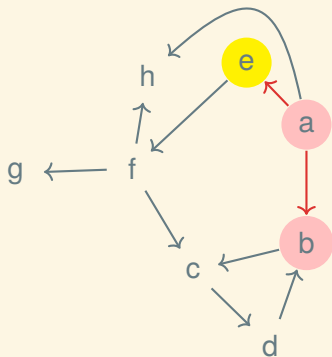
b e h

Running BFS on a digraph



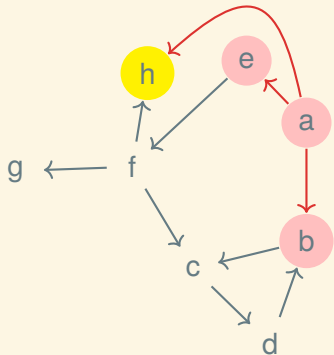
e h c

Running BFS on a digraph



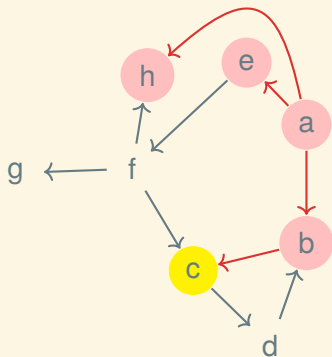
h c f

Running BFS on a digraph



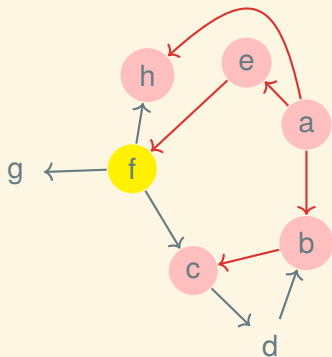
c f

Running BFS on a digraph



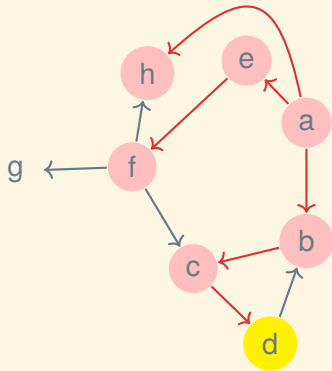
f d

Running BFS on a digraph



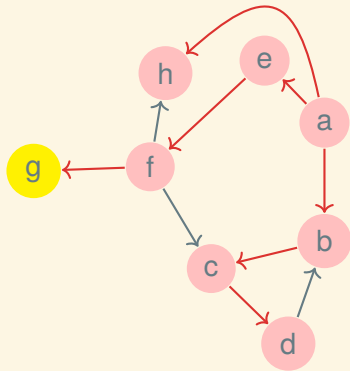
d g

Running BFS on a digraph

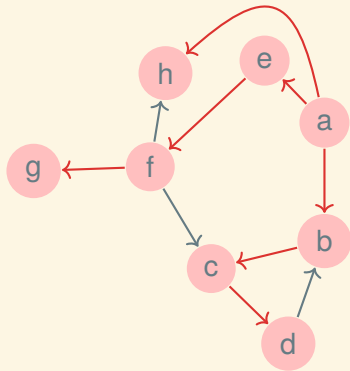


g

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

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

```
todo ← new collection;
```

```
preds[start] ← true;
```

```
Add(todo, start);
```

```
while todo is not empty do
```

```
    v ← Remove(todo);
```

```
    for u in Successors(graph, v) do
```

```
        if not preds[u] then
```

```
            preds[u] ← v;
```

```
            Add(todo, u)
```

```
        end
```

```
    end
```

```
end
```

```
return preds
```

```
end
```


Next time (after the exam): shortest paths