Depth first search

1. Background
2. Data structures
3. Procedure
4. Results and interpretation

1. Background

At start: we have a digraph *G = (V,E)* and a starting node (source node) *s* from the digraph.

Goal: we want to know all nodes that are reachable from *s*.

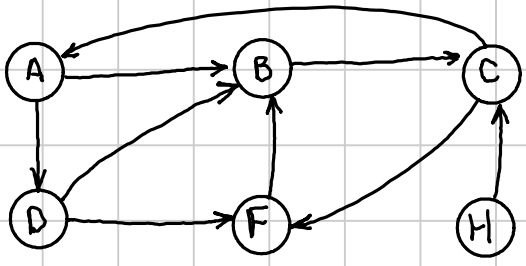
One way to do this is by performing a graph search.

If all other nodes are reachable, then the search is also a traversal.

Reminders

- y is reachable from x when there is a directed path from x to y
- y is adjacent to x when edge (x, y) exists
- a simple cycle is a path $\langle a_0, a_1, \ldots a_k \rangle$ where $a_k = a_0$ and no other nodes are repeated

Example



x	A	B	C	D	F	H
adjacent to x	D, B	C	A, F	B, F	B	C
reachable from x	D, B, C, A, F	C, A, D, B, F	C, A, F, B, D	B, F, C, A, D	B, C, A, D, F	C, A, F, D, B

some cycles: $\langle A, B, C, A \rangle$, $\langle A, D, F, B, C, A \rangle$ □

Results from Depth-first-search (DFS):

- all reachable nodes from s
- detection of simple cycles in the digraph
- a rooted tree whose root is s which includes all reachable nodes from s (the DF-tree)

2. Data structures

In DFS a node can be in one of 4 states:
(i) undetected, (ii) detected and undiscovered, (iii) discovered or (iv) discovered and handled.

Q: What do we mean when we say a node *x* has been discovered?

A: We mean that a path from *s* to *x* has been found.

Q: What do we mean when we say a node *x* has been detected and undiscovered?

A: We mean that *x* is adjacent to a node that has been discovered, but that *x* has not yet been discovered.

Q: What do we mean when we say a node *x* has been discovered and handled?

A: We mean that *x* has been discovered and all nodes adjacent to *x* have been discovered.

In DFS

- start with all nodes undetected and undiscovered, except *s*

- progress is made by moving along edges and discovering nodes that have been undetected and/or undiscovered

- the status of a node is monitored using a stack and colors
  - If *x* is undetected, then it is white and it is not on the stack.
  - If *x* is detected but undiscovered, then it is white and it is on the stack.
  - If *x* is discovered, then it is gray and it is on the stack.
  - If *x* is discovered and handled, then it is black and it is not on the stack.

Note: progression of a node:
(white, not on stack) $\rightarrow$ (white, on stack) $\rightarrow$ (gray, on stack) $\rightarrow$ (black, not on stack)

To perform DFS, for each node we have the following attributes:

- $x.colour$ = color of node
- $x.\pi$ = parent of node x in DF tree
- $x.Adj$ set containing nodes that are adjacent to x

In DFS we maintain a stack of gray and white nodes.

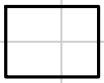
A stack is a one-dimensional data structure that has top.

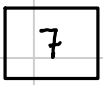
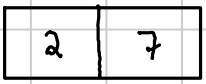
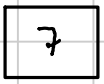
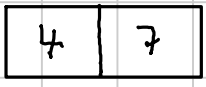
A stack *S* has two basic operations;

- PUSH(*S,x*): places (pushes) item *x* onto top of stack
- POP(*S*): removes and returns item from top of stack

A stack is said to function on a last-in-first-out (LIFO) basis.

Example

Start with empty stack: *S*: top 

operation	stack
PUSH(*S*, 7)	top 
PUSH(*S*, 2)	top 
POP(*S*)	top 
PUSH(*S*, 4)	top 

□

3. Procedure

Description of DFS:

DEPTH–FIRST–SEARCH

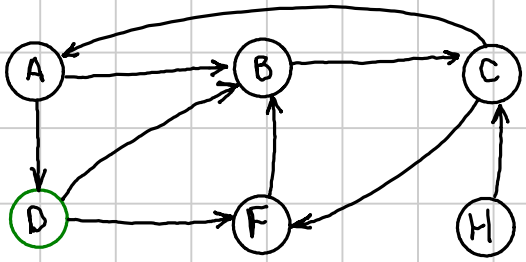
1. Mark s as discovered. Set $x = s$.
2. Mark x as discovered.
3. If x has at least one undiscovered adjacent node y, then move along edge (x, y) to y and set $x = y$. Return to step 2.
4. If x has no undiscovered adjacent nodes and $x = s$, then stop.
5. If x has no undiscovered adjacent nodes and $x \neq s$, then move back to u along edge (u, x) via which x was discoverd. Set $u = x$ and return to step 3.

When we perform step 3, we are moving forward.

When we perform step 5, we are moving backward (or backtracking).

Example:

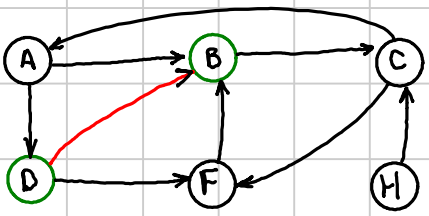
starting node: *s* = D



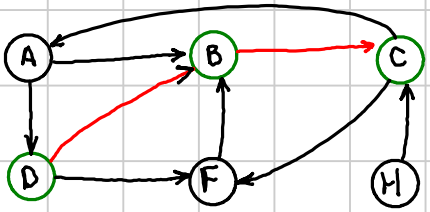
x x discovered (gray)

x x discovered and handled (black)

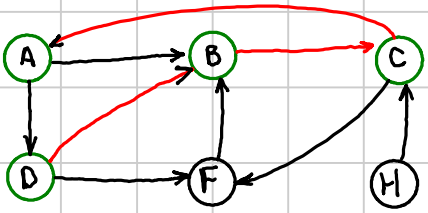
1. forward:
(D,B)



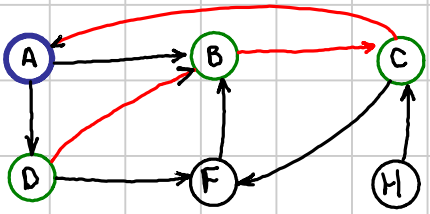
2. forward:
(B,C)



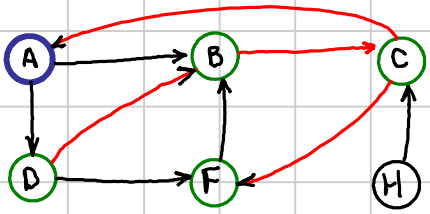
3. forward:
(C,A)



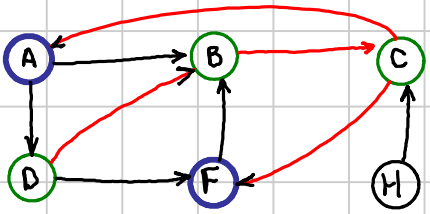
4. backward:
(C,A)



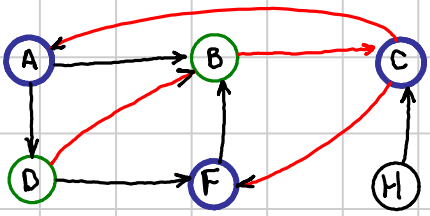
5. forward:
(C,F)



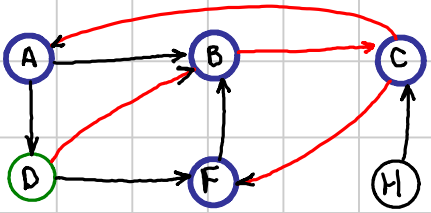
6. backward:
(C,F)



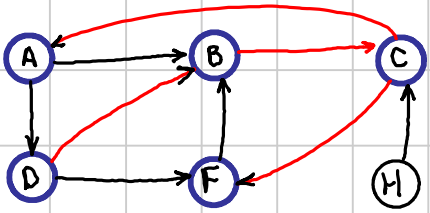
7. backward:
(B,C)



8. backward:
(D,B)



9. D is
handled



Pseudocode:

```
1   DEPTH–FIRST–SEARCH(s, G)
2   executes a depth first search on graph G starting from
3   source node s
4
5   forEach node x in G
6     x.color = white, x.π =NIL
7   end
8
9   ▷ initialize a stack in S
10  PUSH(S,s)
11  while S is not empty
12    x = POP(S)
13
14    /* If x is white, then it has not yet been discovered.*/
15    if x.color == white then
16    /* x is discovered. Put x back into the stack and
         investigate nodes adjacent to x.*/
17      x.color = gray, PUSH(S,x)
18
19        forEach node y in x.Adj
20          if y.color == white then
21            PUSH(S,y), y.π = x
22          else if y.color == gray then
23                ▷ a cycle that includes edge (x,y) exists
24              end
25          end
26        end
27    else
28      x.color = black
29    end
30  end
```

Remarks

1. Choose any node *x* from the stack *S*. The path from *s* to *x* can always be found from nodes lower down in the stack *S*.

2. All gray nodes in the stack *S* form a linear path starting from *s* out to the gray node of the greatest 'depth'.

3. The parent of a node may be set several times at line 21. The final time when it is set corresponds to two possible situations:

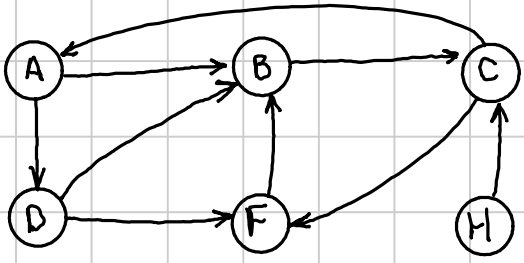
- When moving forward: *x* is colored gray at one iteration of the while-loop and *y* is colored gray on the following iteration.

- When moving backwards: we POP a white node *x* at line 12 that is adjacent to the deepest gray node *z* currently on the stack.

4. In each iteration of the while loop
   - Either *x* is white and it is colored gray, or *x* is gray and it is colored black
   - A node will be added to the stack as a gray node only once.
   - A black node is never added to the stack.

Example

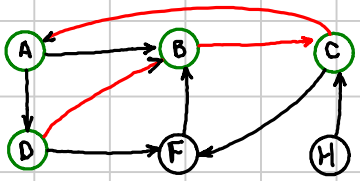
Execute DEPTH-FIRST-SEARCH with *s* = *D*



black node

gray node

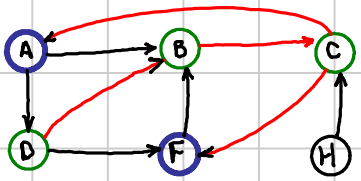
item	while-loop iteration 0	1	2	3	4
S	D	B,F,D	C,B,F,D	A,F,C,B,F,D	A,F,C,B,F,D
x		D	B	C	A
A.π	NIL	NIL	NIL	C	C
B.π	NIL	D	D	D	D
C.π	NIL	NIL	B	B	B
D.π	NIL	NIL	NIL	NIL	NIL
F.π	NIL	D	D	C	C
H.π	NIL	NIL	NIL	NIL	NIL
A.color	white	white	white	white	gray
B.color	white	white	gray	gray	gray
C.color	white	white	white	gray	gray
D.color	white	gray	gray	gray	gray
F.color	white	white	white	white	white
H.color	white	white	white	white	white
cycle edge					(A,D), (A,B)



black node

gray node

item	while-loop iteration 4	5	6	7
S	A,F,C,B,F,D	F,C,B,F,D	F,C,B,F,D	C,B,F,D
x	A	A	F	F
A.π	C	C	C	C
B.π	D	D	D	D
C.π	B	B	B	B
D.π	NIL	NIL	NIL	NIL
F.π	C	C	C	C
H.π	NIL	NIL	NIL	NIL
A.color	gray	black	black	black
B.color	gray	gray	gray	gray
C.color	gray	gray	gray	gray
D.color	gray	gray	gray	gray
F.color	white	white	gray	black
H.color	white	white	white	white
cycle edge	(A,D), (A,B)		(F,B)	



Final results:

x	A	B	C	D	F	H
$x.\pi$	C	D	B	NIL	C	NIL
x.color	black	black	black	black	black	white

edges causing cycles: (A,D), (A,B), (F,B)

4. Results and interpretation

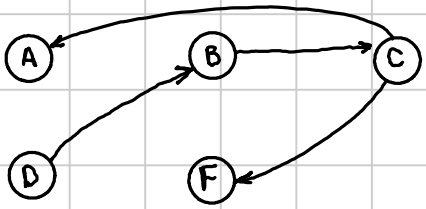
Q: How do we know what nodes are reachable from *s*?

A: If *x* is reachable from *s* then *x.color* is black.

Q: How can we produce the DF-tree?

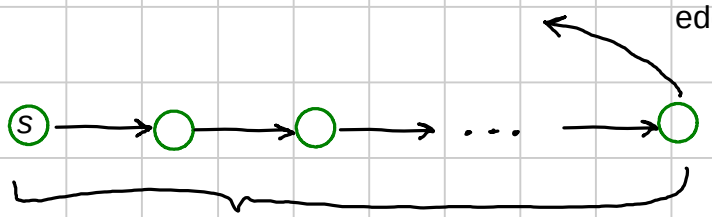
A: The DF-tree can be made using the parent attributes $x.\pi$ of the nodes.

DF-tree:

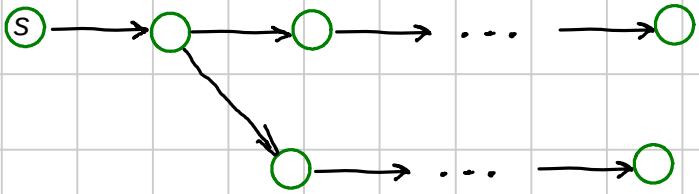


Q: How do we know that the edges found at line 23 produce cycles?

A: This is a consequence of the observation that the gray nodes in the stack form a linear path from *s*.



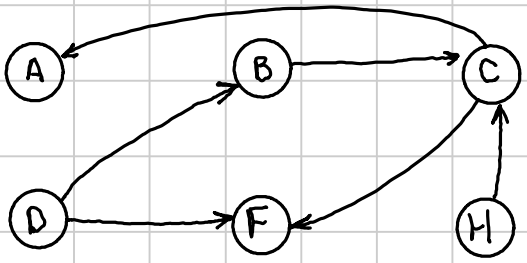
Why is this not possible for gray nodes in S?



Q: What do we obtain if we remove all the cycle edges found in line 23 from the original graph?

A: We obtain an acyclic graph.

edges causing cycles: (A,D), (A,B), (F,B)



Q: Can we use Depth-first-search on an undirected graph?

A: Yes.

Remarks for undirected graph:
- if *x* belongs to *y.Adj*, then *y* belongs to *x.Adj*

- if *x* is reachable from *s*, then *s* is reachable from *x*

- all edges in graph are either tree edges or edges that cause (undirected) cycles