Depth First Search

Search C455

Document last modified: 03/12/2014 00:58:52

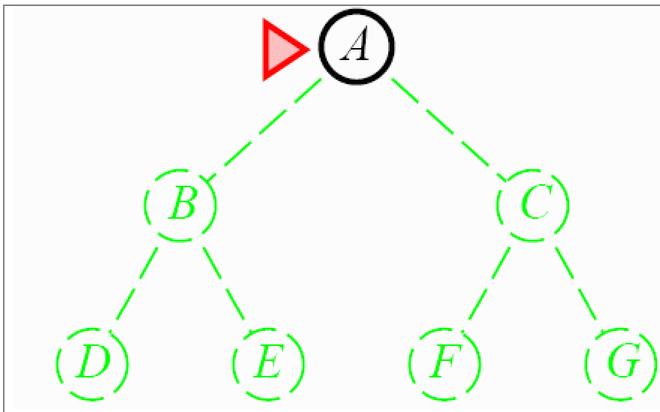
Overview

Key difference between BFS and DFS is the order *discovered* (adjacent) vertices are explored.

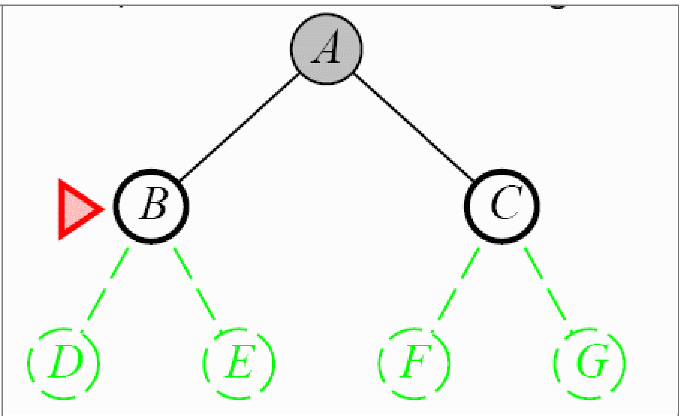
BFS places discovered vertices in FIFO *queue*, exploring vertices in the order discovered.

DFS places discovered vertices in LIFO *stack*, exploring vertices as discovered.

BFS

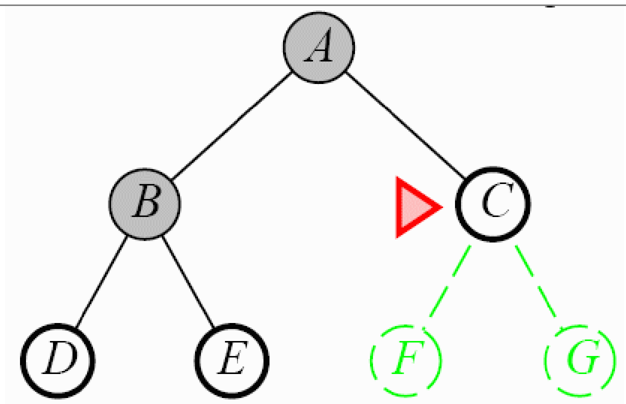


Start with A



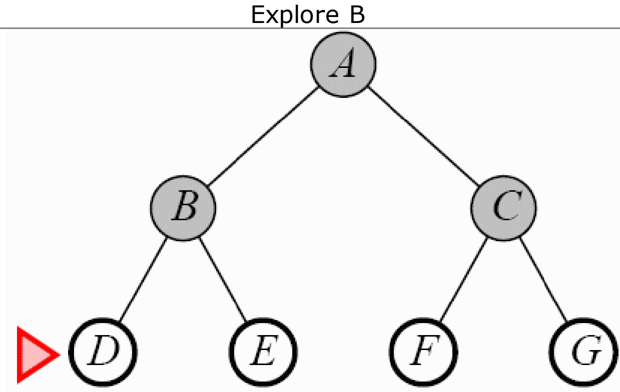
Discovered [**B, C**]

Explore B



Discovered [C, **D, E**]

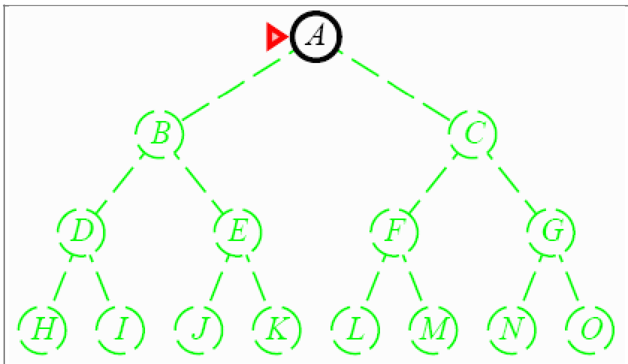
Explore C



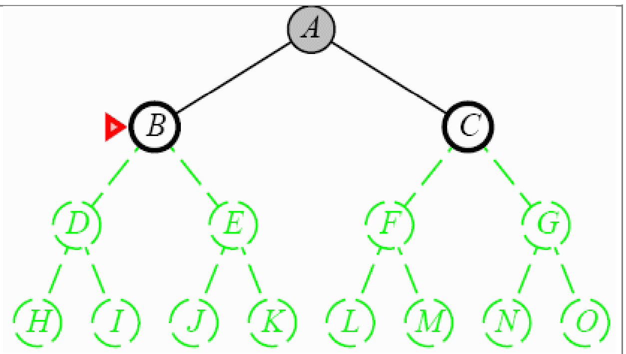
Discovered [D, E, **F, G**]

Explore D

DFS

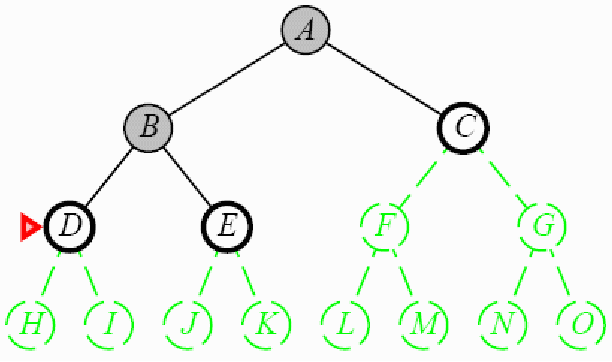


Start with A



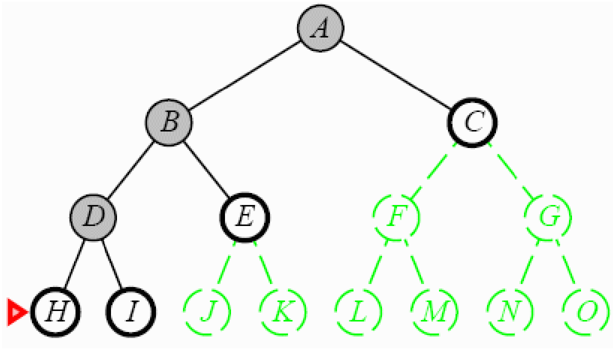
Discovered [**B, C**]

Explore B



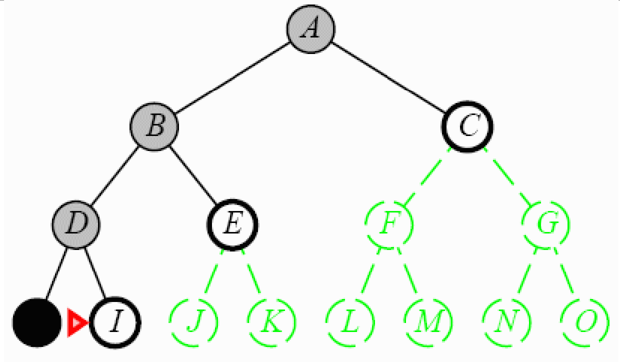
Discovered [**D, E**, C]

Explore D



Discovered [**H, I**, E, C]

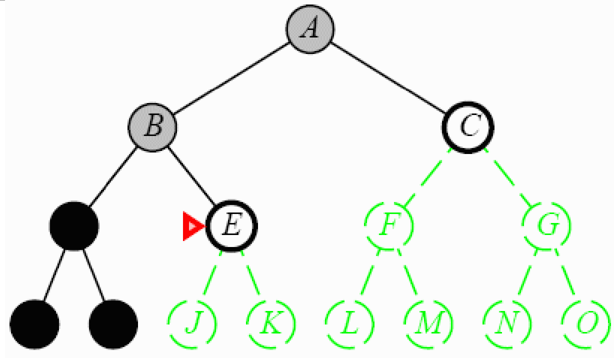
Explore H



Finished H

Discovered [I, E, C]

Explore I



Finished I, D

Discovered [E, C]

Input:

G = (V, E), directed or undirected. No source vertex given!

Output:

2 timestamps on each vertex, useful for other algorithms examined later:

- v.d = discovery time
- v.f = finishing time

Also computes v.Π, the predecessor subgraph.

DFS algorithm: Uses a global timestamp *time*.

```
DFS (G)
    -- WHITE - not discovered
    -- GRAY - discovered & being explored
    -- BLACK - discovered & finished
    -- Initialize arrays
    for each vertex u ∈ G.V do
        u.color ← WHITE
        u.Π ← NIL
    time ← 0
    -- Perform depth first search
    for each vertex u ∈ G.V do
        if u.color = WHITE then
            DFS-Visit (G, u)
```

```
DFS-Visit (G, u)
-- pre: u.color=WHITE
-- post: u.color=BLACK
--    ∀v  u.d<v.d<v.f<u.f, etc.
    u.color ← GRAY
    time ← time + 1
    u.d ← time
    for each vertex v ∈ G.Adj[ u ] do
        if v.color = WHITE then
            v.Π ← u
            DFS-Visit (G, v)
    u.color ← BLACK
    u.f ← time ← time + 1
```

Will methodically explore every edge.

- Start from some undiscovered vertex, DFS line 7.
- As soon as a new vertex discovered, explore it recursively, DFS-Visit line 7.
- Unlike BFS, which puts a vertex on a queue so that we explore from it later.

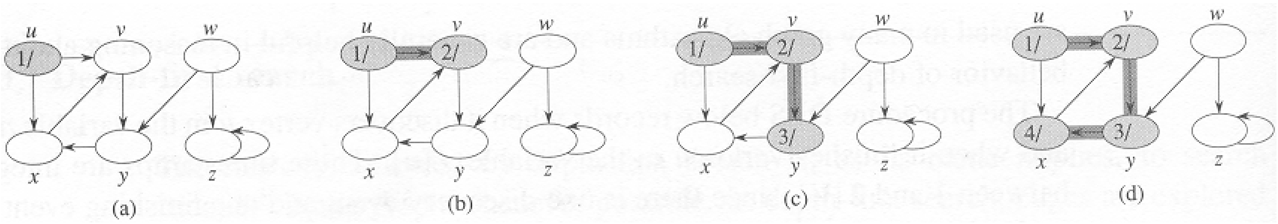
As DFS progresses, every vertex has a color:

- WHITE = undiscovered
- GRAY = discovered, but not finished (not done exploring from it)
- BLACK = finished (have found everything reachable from it)

Discovery and finish times:

- Unique integers from 1 to 2|V|.
- For all v, v.d < v.f $\qquad 1 \le v.d < v.f \le 2|V|$.

Example - Vertices labeled as *discovery/finish* time. Vertices selected alphabetically at **for** statements.

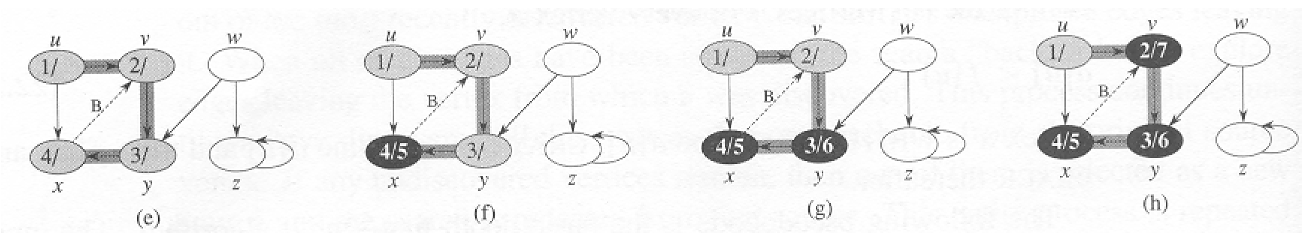


State prior to DFS-Visit(G, x) line 8

	u	v	w	x	y	z
Π	NIL	u	NIL	y	v	NIL
d	1	2		4	3	
f						
color	GRAY	GRAY	WHITE	GRAY	GRAY	WHITE

```
DFS (G)
     -- Initialize arrays
1    for each vertex u ∈ G.V do
2       u.color ← WHITE
3       u.Π ← NIL
4    time ← 0
5    for each vertex u ∈ G.V do
6       if u.color = WHITE then
7          DFS-Visit (G, u)
```

```
DFS-Visit (G, u)
1    u.color ← GRAY
2    time ← time + 1
3    u.d ← time
4    for each vertex v ∈ G.Adj[ u ] do
5       if v.color = WHITE then
6          v.Π ← u
7          DFS-Visit (G, v)
8    u.color ← BLACK
9    u.f ← time ← time + 1
```



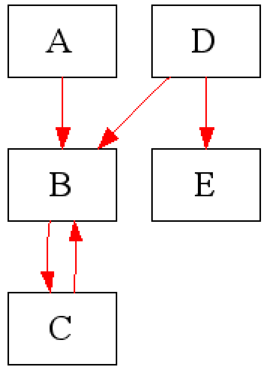
State prior to DFS-Visit(G, u) line 8

	u	v	w	x	y	z
Π	NIL	u	NIL	y	v	NIL
d	1	2	0	4	3	0
f		7		5	6	
color	GRAY	BLACK	WHITE	BLACK	BLACK	WHITE

Question 22.11

1. Where is the DFS?
2. Suppose G.V taken in order of {u, v, w, x, y, z}.

   Figure (h) above shows DFS-Visit(G, v) completed, BLACK. What is the next value of *u* of DFS(G, u) used in calling DFS-Visit(G, u) at line 7?
3. Finish DFS taking vertices in order of {u, v, w, x, y, z}.
4. Where is the LIFO stack that maintains the discovered but not finished vertices?



Question 22.12 - Use graph at right.

a. Give the DFS(G) solution starting at A, visiting vertices in alphabetical order at Line 5 and DFS-Visit(G, u) at Line 4.

b. Is every vertex visited for an undirected graph? a directed graph?

c. Is every path explored? Explain using graph.

d. BFS requires a queue to maintain a list of those vertices discovered but not yet visited, DFS does not. Why not?

```
DFS (G)
     -- Initialize arrays
1    for each vertex u ∈ G.V do
2        u.color ← WHITE
3        u.Π ← NIL
4    time ← 0
5    for each vertex u ∈ G.V do
6        if u.color = WHITE then
7            DFS-Visit (G, u)
```

```
DFS-Visit (G, u)
1    u.color ← GRAY
2    time ← time + 1
3    u.d ← time
4    for each vertex v ∈ G.Adj[ u ] do
5        if v.color = WHITE then
6            v.Π ← u
7            DFS-Visit (G, v)
8    u.color ← BLACK
9    u.f ← time ← time + 1
```

Loop-invariant - for loop of DFS(G) Lines 5-7

loop invariant:

Every vertex *v* visited by DFS-Visit(G, v) is BLACK (i.e. fully explored, finished)

initialization:

No vertex has been visited.

maintenance:

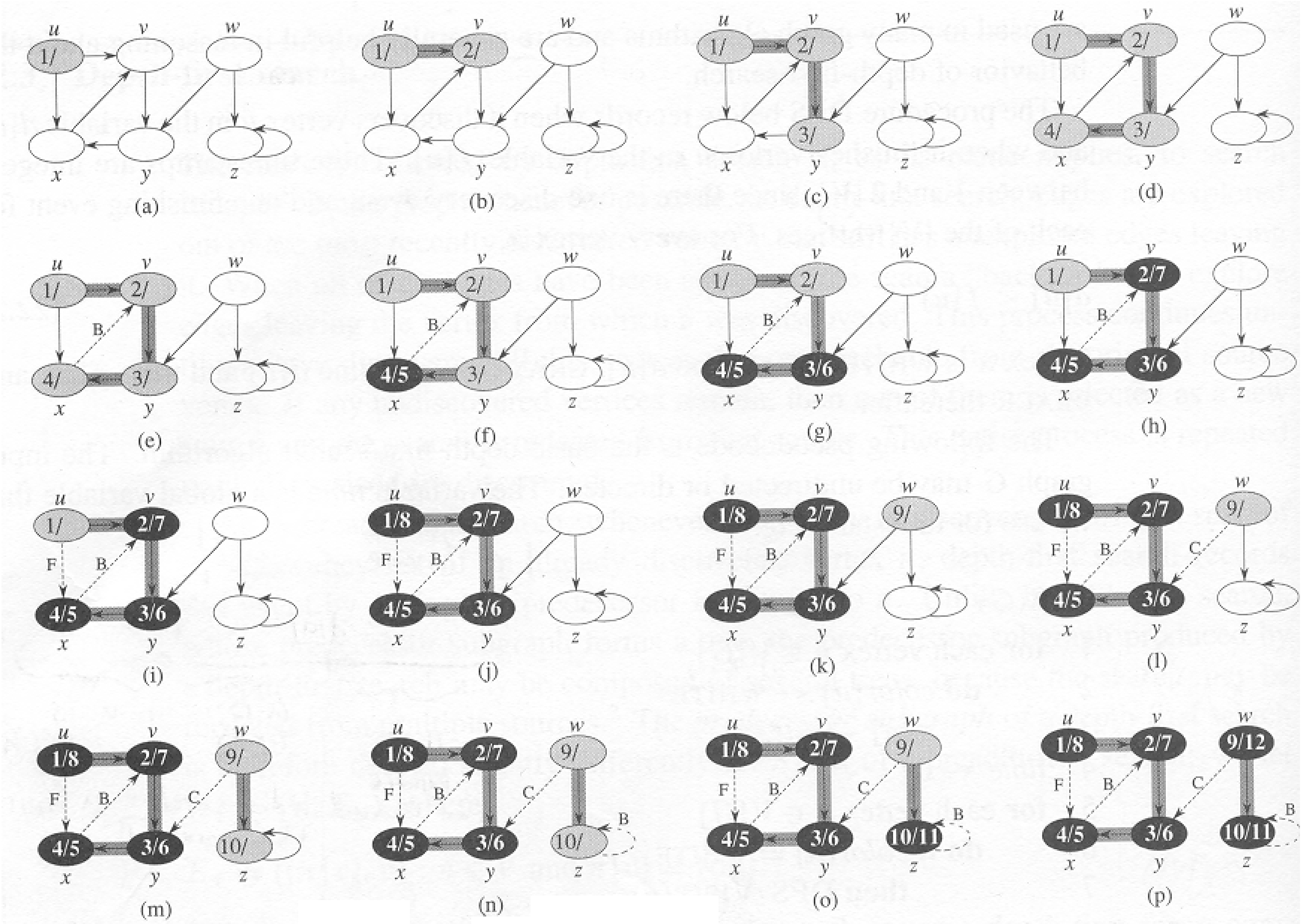
- We should show this by proving that DFS-Visit post condition holds when the precondition holds.

  -- pre: *u*.color=WHITE
  -- post: *u*.color=BLACK
  -- $\forall v \ \ u.d < v.d < v.f < u.f$ where $v.\Pi = u$

- DFS-Visit(G, u) only called when vertex *u* has not been discovered (WHITE) Line 6
- DFS-Visit(G, u) assigns u.color = BLACK, $\forall v \ \ u.d < v.d < v.f < u.f$ where $v.\Pi = u$
  - $u.d < v.d$ because *u.d* ← *time* Line 3 before *v.d* ← *time* Line 3
  - $v.d < v.f$ because *v.d* ← *time* Line 3 before *v.f* ← *time* Line 9
  - $v.f < u.f$ because *v.f* ← *time* Line 9 before *u.f* ← *time* Line 9

termination:

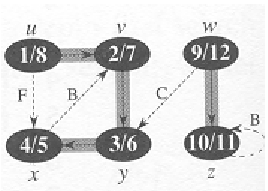
- The loop terminates after iterating through every vertex of V. By maintenance, the loop invariant holds.



Predecessor subgraph Π

$G_{\Pi}=(V, E_{\Pi})$

$E_{\Pi}=\{(v.\Pi, v) : v \in V \text{ and } v.\Pi \neq NIL \}$



Example: (x.Π, x) = (y, x)

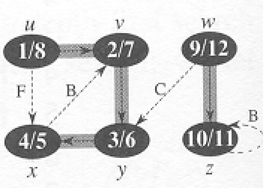
$E_{\Pi}=\{(u, v), (v, y), (y, x), (w, z)\}$ for above figure

	u	v	w	x	y	z
Π	NIL	u	NIL	y	v	w
d	1	1	9	4	3	10
f	8	7	12	5	6	11

Question 22.13 - For above graph.

a. Trace the path using the predecessor subgraph **Π** starting at x.
b. Draw the resulting depth-first tree(s).
c. With 10 vertices in a graph, what is the maximum finish time, *f* ?
d. Start DFS(G) at w. Will different predecessor subgraphs be established?
e. Do you agree or disagree with the statement "Depth first search is a good way of establishing the nature of the connectivity in a graph. You can build a table out of the results of a search showing which vertices are reachable from which other vertices."?

Classification of edges



1. **Tree edges** are edges in the depth-first forest of G_Π.

   Edge (u, v) is a tree edge if *v* is *first* discovered by exploring edge (u, v).

   Example: (y, x) is a tree-edge but (u, x) is not because x was not discovered exploring (u, x).

2. **Back edges** are those edges (u, v) connecting a vertex *u* to an ancestor *v* in a depth-first tree.

   Self-loops, which may occur in directed graphs, are considered to be back-edges.

   Example: (x, v) and v.d < x.d, *v* is an ancestor of *x*

3. **Forward edges** are those non-tree edges (u, v) connecting a vertex *u* to a descendant *v* in a depth-first tree.

   Example: (u, x) is not in G_Π and u.d < x.d, *x* is a descendant of *u*

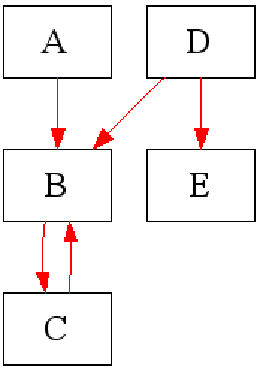
4. **Cross edges** are all other edges.

   Can go between vertices of same depth-first tree as long as one is not the ancestor of the other (cycle) or between vertices of different depth-first trees.

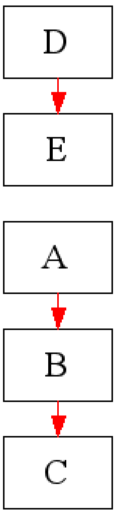
   Example: (w, y) between different depth-first trees.

   Question 22.14 - Classify each edge of the graph below left for the DFS results of below right.

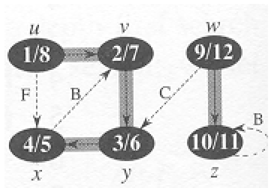
Graph G



DFS Trees

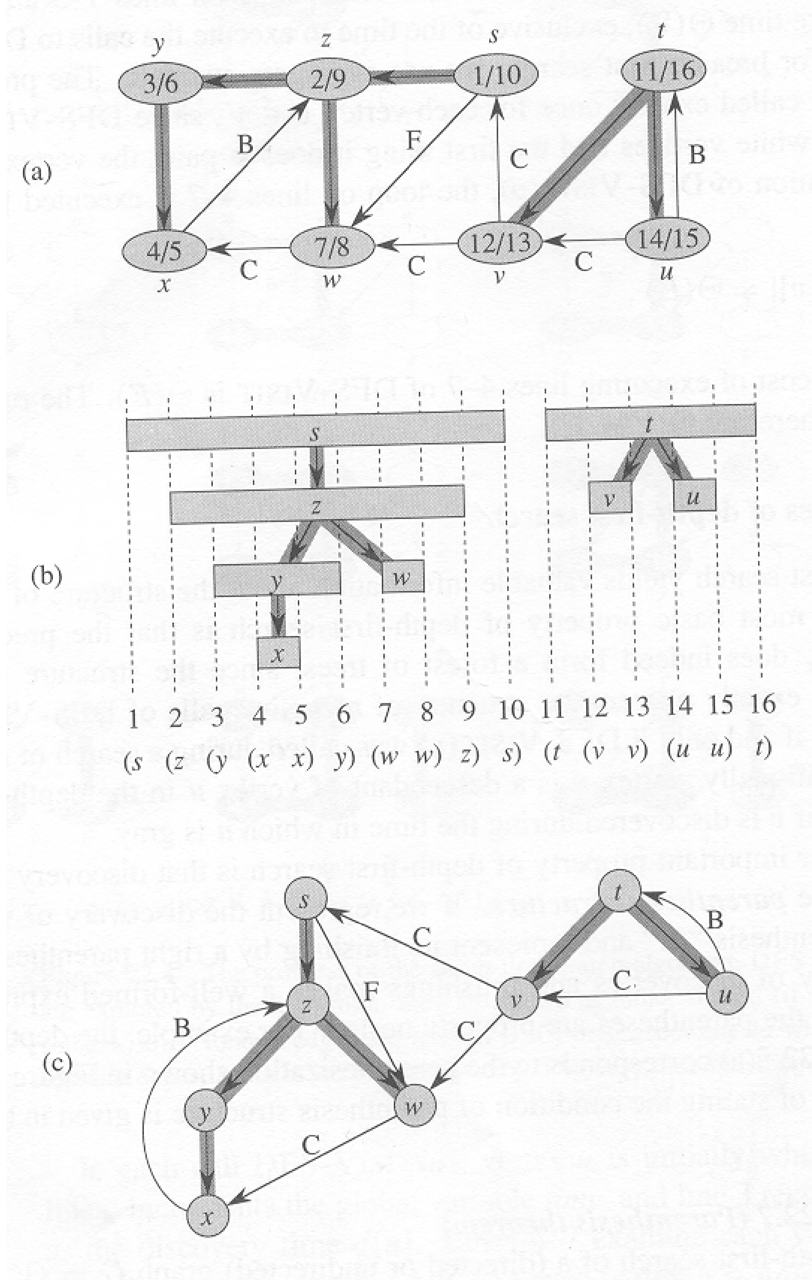


Properties of DFS



- G_Π forms a forest of disjoint trees.   Example at right: u → v → y → x and w → z
- Note that roots of a depth-first tree occur where *v*.Π = NIL for v ∈ V. Why is that?
- **Parenthesis structure**

	In the figures at left: a. DFS result of directed graph. Vertices time-stamped and edge types indicated (heavy lines are the DFS tree). b. Intervals for discovery and finish times



The DFS tree(s) above.

for each vertex correspond to the given parenthesization.

Each rectangle spans discovery and finish times of vertex.

When two intervals overlap one is nested within the other, and vertex of the smaller interval is a descendant of the larger.

For example: vertex z discovery and finish times span those of y, y is then a descendant of z.

> z.d < y.d < y.f < z.f implies y is a descendant of z.
>
> > 2 < 3 < 6 < 9
> >
> > (**z** (y (x x) y) (w w) **z**)

c. Graph of (a) redrawn with all tree and *forward* edges going down within a depth-first tree and all *back* edges going up from a descendant to an ancestor.

Parenthesis theorem

For all u and v, exactly one of the following holds:

1. u and v are in different branches or trees, neither of u and v is a descendant of the other:

   u.d < **u.f** < v.d < v.f

   or

   v.d < v.f < **u.d** < **u.f**

2. v is a descendant of u:

   u.d < v.d < v.f < **u.f**

3. u is a descendant of v:

   v.d < u.d < u.f < **v.f**

So

> u.d < v.d < u.f < v.f cannot happen

As with parentheses:

- OK: ( ) [ ] ([ ]) [( )]
- Not OK: ([ )] [( ])

Question 22.15 - What is implied by:

- s.d < w.d < w.f < s.f
- x.f < w.d

Analysis Similar to BFS analysis.

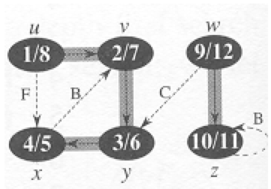
BFS is O(V + E)

Not all vertices may be examined in a directed graph.

Ω(1) with source *z* in graph at right.

DFS is Θ(V + E)

Θ, not just O, since guaranteed to examine every vertex and edge.



DFS (*G*)	**DFS-Visit** (*G, u*)

-- Initialize arrays
for each vertex *u* ∈ *G.V* **do**
  u.color ← WHITE
  u.Π ← NIL
time ← 0
for each vertex *u* ∈ *G.V* **do**
  if *u.color* = WHITE **then**
    DFS-Visit (*G, u*)
	u.color ← GRAY
	time ← *time* + 1
	u.d ← *time*
	for each vertex *v* ∈ *G.Adj*[*u*] **do**
	  if *v.color* = WHITE **then**
	    v.Π ← *u*
	    DFS-Visit (*G, v*)
	u.color ← BLACK
	u.f ← *time* ← *time* + 1