Graphs – Depth First Search
Graph Search Algorithms
Outline

- DFS Algorithm
- DFS Example
- DFS Applications
Outline

- DFS Algorithm
- DFS Example
- DFS Applications
Depth First Search (DFS)

- Idea:
  - Continue searching “deeper” into the graph, until we get stuck.
  - If all the edges leaving \( v \) have been explored we “backtrack” to the vertex from which \( v \) was discovered.
  - Analogous to Euler tour for trees

- Used to help solve many graph problems, including
  - Nodes that are reachable from a specific node \( v \)
  - Detection of cycles
  - Extraction of strongly connected components
  - Topological sorts
The DFS algorithm is similar to a classic strategy for exploring a maze:

- We mark each intersection, corner and dead end (vertex) visited.
- We mark each corridor (edge) traversed.
- We keep track of the path back to the entrance (start vertex) by means of a rope (recursion stack).
Depth-First Search

**Input:** Graph $G = (V,E)$ (directed or undirected)

- Explore every edge, starting from different vertices if necessary.
- As soon as vertex discovered, explore from it.
- Keep track of progress by colouring vertices:
  - Black: undiscovered vertices
  - Red: discovered, but not finished (still exploring from it)
  - Gray: finished (found everything reachable from it).
DFS Example on Undirected Graph

- A: unexplored
- B: being explored
- C: finished
- D: unexplored edge
- E: discovery edge
- F: back edge
Example (cont.)
DFS Algorithm Pattern

DFS(G)

Precondition: G is a graph

Postcondition: all vertices in G have been visited

for each vertex \( u \in V[G] \)

\[
\text{color}[u] = \text{BLACK} \quad // \text{initialize vertex}
\]

for each vertex \( u \in V[G] \)

if color[u] = BLACK //as yet unexplored

DFS-Visit(u)
DFS Algorithm Pattern

DFS-Visit \((u)\)

Precondition: vertex \(u\) is undiscovered

Postcondition: all vertices reachable from \(u\) have been processed

\[
\text{colour}[u] \leftarrow \text{RED}
\]

\[
\text{for each } v \in \text{Adj}[u] \quad //\text{explore edge } (u,v)
\]

\[
\text{if } \text{color}[v] = \text{BLACK}
\]

\[
\text{DFS-Visit}(v)
\]

\[
\text{colour}[u] \leftarrow \text{GRAY}
\]
Properties of DFS

Property 1

\(DFS-V\text{isit}(u)\) visits all the vertices and edges in the connected component of \(u\)

Property 2

The discovery edges labeled by \(DFS-V\text{isit}(u)\) form a spanning tree of the connected component of \(u\)
DFS Algorithm Pattern

DFS(G)
Precondition: G is a graph
Postcondition: all vertices in G have been visited

for each vertex \( u \in V[G] \)
    color[\( u \)] = BLACK //initialize vertex

for each vertex \( u \in V[G] \)
    if color[\( u \)] = BLACK //as yet unexplored
        DFS-Visit(\( u \))

\[ \text{total work} = \theta(V) \]
DFS Algorithm Pattern

DFS-Visit \( (u) \)

Precondition: vertex \( u \) is undiscovered

Postcondition: all vertices reachable from \( u \) have been processed

\[
\text{colour}[u] \leftarrow \text{RED}
\]

\[
\text{for each } v \in \text{Adj}[u] \quad \text{//explore edge } (u,v)
\]

\[
\begin{align*}
&\quad \text{if colour}[v] = \text{BLACK} \\
&\quad \quad \text{DFS-Visit}(v)
\end{align*}
\]

\[
\text{colour}[u] \leftarrow \text{GRAY}
\]

Thus running time = \( \theta(V + E) \)

(assuming adjacency list structure)
Variants of Depth-First Search

- In addition to, or instead of labeling vertices with colours, they can be labeled with **discovery** and **finishing** times.

- ‘Time’ is an integer that is incremented whenever a vertex changes state
  - from **unexplored** to **discovered**
  - from **discovered** to **finished**

- These **discovery** and **finishing** times can then be used to solve other graph problems (e.g., computing strongly-connected components)

**Input:** Graph $G = (V, E)$ (directed or undirected)

**Output:** 2 timestamps on each vertex:
- $d[v] = \text{discovery time.}$
- $f[v] = \text{finishing time.}$

1 $\leq d[v] < f[v] \leq 2 \mid V \mid$
DFS Algorithm with Discovery and Finish Times

DFS(G)

Precondition: G is a graph

Postcondition: all vertices in G have been visited

for each vertex $u \in V[G]$

\[
\text{color}[u] = \text{BLACK} \quad \text{//initialize vertex}
\]

\[
\text{time} \leftarrow 0
\]

for each vertex $u \in V[G]$

if color[$u$] = BLACK //as yet unexplored

DFS-Visit($u$)
DFS Algorithm with Discovery and Finish Times

DFS-Visit \((u)\)

Precondition: vertex \(u\) is undiscovered

Postcondition: all vertices reachable from \(u\) have been processed

\[
\begin{align*}
\text{colour}[u] & \leftarrow \text{RED} \\
\text{time} & \leftarrow \text{time} + 1 \\
\text{d}[u] & \leftarrow \text{time} \\
\text{for each } v \in \text{Adj}[u] & \text{ //explore edge } (u,v) \\
& \quad \text{if color}[v] = \text{BLACK} \\
& \quad \text{DFS-Visit}(v) \\
\text{colour}[u] & \leftarrow \text{GRAY} \\
\text{time} & \leftarrow \text{time} + 1 \\
\text{f}[u] & \leftarrow \text{time}
\end{align*}
\]
Other Variants of Depth-First Search

- The DFS Pattern can also be used to
  - Compute a forest of spanning trees (one for each call to DFS-visit) encoded in a predecessor list $\pi[u]$
  - Label edges in the graph according to their role in the search (see textbook)
    - **Tree edges**, traversed to an undiscovered vertex
    - **Forward edges**, traversed to a descendent vertex on the current spanning tree
    - **Back edges**, traversed to an ancestor vertex on the current spanning tree
    - **Cross edges**, traversed to a vertex that has already been discovered, but is not an ancestor or a descendent
Outline

- DFS Algorithm
- DFS Example
- DFS Applications
DFS

Note: Stack is Last-In First-Out (LIFO)

Found
Not Handled
Stack

<node,# edges>
DFS

Found Not Handled
Stack

\(<\text{node,}\#\text{ edges}>\)

s,0
DFS

Found
Not Handled
Stack

<node,# edges>

- a,1
- b,1
- c,0
- d,1
- e,1
- f,0
- g,1
- h,1
- i,2
- j,1
- k,1
- l,1
- m,1
- n,0

a,1
b,1
c,0
d,1
e,1
f,0
g,1
h,1
i,2
j,1
k,1
l,1
m,1
n,0
DFS

Found
Not Handled
Stack

<node,# edges>

ah,0
c,1
a,1
s,1

Stack

<node,# edges>

1/

a,1
c,1
h,0

2/
a

3/

c

4/
h

Node

s

b

d

e

i

f

j

m

k

l
DFS

Found
Not Handled
Stack

<node,# edges>

a,1
b,1
c,1
d,1
e,1
f,1
g,1
h,1
i,1
j,1
k,0
l,1
m,1
n,1
s,1
DFS

Path on Stack

Tree Edge

Found
Not Handled
Stack

<node,# edges>

1/ a,1
c,1
h,1
d,1
f,1
b,1
g,1
j,1
i,1

Not Handled
Stack

<node,# edges>

1/ s,1
h,1
c,1
a,1
s,1
DFS

Found
Not Handled
Stack

<node,# edges>

[Diagram of a graph with nodes labeled a, b, c, d, e, f, g, h, i, j, k, l, m, and a stack list showing nodes and edges]

Not Handled

Stack

Found

<node,# edges>

c,1
a,1
s,1

[Node labels and edge counts indicated on the graph]
Cross Edge to handled node: d[h] < d[i]

DFS

Stack

<node, # edges>

Found

Not Handled

s, 1

a, 1

c, 2

i, 1

j, 1

b, /

g, /
e, /
d, /

h, 7/4

k, 6/5

l, 1

m, /
DFS

Not Handled

Stack

<node,# edges>

s, 1

a, 1

c, 2

d, 2

e, 2

f

i, 2

c, 2

j

a, 1

b

k

h

l, 1

m

Found

1/

3/

2/

4/7

5/6

8/

/
DFS

Found
Not Handled
Stack

<node,# edges>

1,0
i,3
c,2
a,1
s,1
DFS

Found
Not Handled
Stack

<node,# edges>

1,1
i,3
c,2
a,1
s,1

Stack

<node,# edges>

1/  
2/  
3/  
4/7  
5/6  
6/  
7/  
8/  
9/  
10/
DFS

Found
Not Handled
Stack

<node,# edges>

s

1/

a

2/

c

3/

d

/

f

/

g

/

i, 3

c, 2

h

4/7

b

/

j

/

m

/

k

5/6

l

9/10

- 33 -
DFS

Stack

<node, # edges>

s, 1
a, 1
c, 2
i, 4
g, 0
e, 0
b, /
d, /

Found
Not Handled

h, 4
f, /

k, 6
j, /
m, /

1/ 2/ 3/ 4/ 5/ 6/ 7/ 8/ 9/ 10/ 11/
DFS

Found
Not Handled
Stack

<node,# edges>

j,0
g,1
i,4
c,2
a,1
s,1
DFS

Not Handled
Stack

<node, # edges>

m, 0
j, 2
g, 1
i, 4
c, 2
a, 1
s, 1
DFS

Not Handled
Stack

<node,# edges>

a,1
c,2
f,1
h,7
i,4
k,2
m,1
n,1
s,1
de,2
d,1
g,1
j,2

DFS

Found
Not Handled
Stack

<node,# edges>

j,2
g,1
i,4
c,2
a,1
s,1
DFS

Not Handled

Stack

<node,# edges>

s, 1
a, 1
c, 2
f, 1
i, 4
d, 1
b, 1
e, 1
g, 1
j, 1
h, 1
k, 1
m, 1
l, 1

1/ 2/ 3/ 4/ 5/ 6/ 7/ 8/ 9/ 10/ 11/ 12/ 13/ 14/
DFS

Stack

<node, # edges>

Found
Not Handled

s

NOT HANDLED

a

b

c

d

e

f

g

h

i

j

k

l

m

n

1/1

2/2

3/3

4/4

5/5

6/6

7/7

8/8

9/9

10/10

11/11

12/12

13/13

14/14

15/15

16/16

17/17

18/18

CSE 2011
Prof. J. Elder

Last Updated 2014-03-31 10:34 PM
DFS

Found
Not Handled
Stack

<node,# edges>

Forward Edge

s

1/

a

2/

f 17/18

3/

c,3

h

4/7

k

5/6

j

12/15

m

13/14

b

/ 11/16

e

/ 12/15

i

17/18

8/19
DFS

Found
Not Handled
Stack

<node,# edges>

s

a

f

i

k

h

m

b

d

e

j

\(a,1\)

\(i,1\)

\(j,1\)

\(s,1\)
DFS

Found
Not Handled
Stack

<node,# edges>

a, 2
s, 1

CSE 2011
Prof. J. Elder
DFS

Found Not Handled
Stack

<node,# edges>

s, l
DFS

Found
Not Handled
Stack

<node,# edges>

d,0
s,2

a
s
f
h
k

b
g
j

1/8/19
21/2/20
17/18
12/15
13/14
5/6
4/7
9/10
11/16
3/19
2/20
DFS

Found
Not Handled
Stack

<node,# edges>

s

a

b

d

f

e

g

j

i

h

k

l

CSE 2011
Prof. J. Elder
DFS

Found Not Handled Stack

<node,# edges>

e,0
d,3
s,2
DFS

Found
Not Handled
Stack

<n node,# edges>

s,2
DFS

Found
Not Handled
Stack

<node,# edges>

b,0
s,4

- 58 -

Last Updated 2014-03-31 10:34 PM
DFS

Found
Not Handled
Stack

<node,# edges>

b,1
s,4
DFS

Found
Not Handled
Stack

<node,# edges>

a(2/20)
b(25/)
c(3/19)
d(21/24)
e(22/23)
f(17/18)
g(11/16)
h(4/7)
i(-60)
j(12/15)
k(5/6)
l(9/10)
m(13/14)

s(1/)

b(2,)
s(4,

CSE 2011
Prof. J. Elder

Last Updated 2014-03-31 10:34 PM
DFS

Found
Not Handled

Stack

<node,# edges>

b,3
s,4
DFS

Found
Not Handled
Stack

Stack

Not Handled
Stack

<node, # edges>

s, 4

CSE 2011
Prof. J. Elder

Last Updated 2014-03-31 10:34 PM
DFS

Found
Not Handled
Stack

<node, # edges>

Tree Edges
Back Edges
Forward Edges
Cross Edges
END OF LECTURE
MAR 27, 2014
Classification of Edges in DFS

1. **Tree edges** are edges in the depth-first forest $G_\pi$. Edge $(u, v)$ is a tree edge if $v$ was first discovered by exploring edge $(u, v)$.

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

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

4. **Cross edges** are all other edges. They can go between vertices in the same depth-first tree, as long as one vertex is not an ancestor of the other.
Classification of Edges in DFS

1. **Tree edges**: Edge \((u, v)\) is a tree edge if \(v\) was **black** when \((u, v)\) traversed.

2. **Back edges**: \((u, v)\) is a **back edge** if \(v\) was **red** when \((u, v)\) traversed.

3. **Forward edges**: \((u, v)\) is a **forward edge** if \(v\) was **gray** when \((u, v)\) traversed and \(d[v] > d[u]\).

4. **Cross edges** \((u,v)\) is a **cross edge** if \(v\) was **gray** when \((u, v)\) traversed and \(d[v] < d[u]\).

Classifying edges can help to identify properties of the graph, e.g., a graph is acyclic iff DFS yields no **back** edges.
In a depth-first search of an *undirected* graph, every edge is either a *tree edge* or a *back edge*.

Why?
DFS on Undirected Graphs

- Suppose that $(u,v)$ is a **forward edge** or a **cross edge** in a DFS of an undirected graph.

- $(u,v)$ is a **forward edge** or a **cross edge** when $v$ is already handled (grey) when accessed from $u$.

- This means that all vertices reachable from $v$ have been explored.

- Since we are currently handling $u$, $u$ must be **red**.

- Clearly $v$ is reachable from $u$.

- Since the graph is undirected, $u$ must also be reachable from $v$.

- Thus $u$ must already have been handled: $u$ must be **grey**.

- **Contradiction!**
Outline

- DFS Algorithm
- DFS Example
- DFS Applications
DFS Application 1: Path Finding

- The DFS pattern can be used to find a path between two given vertices $u$ and $z$, if one exists.
- We use a stack to keep track of the current path.
- If the destination vertex $z$ is encountered, we return the path as the contents of the stack.

DFS-Path $(u, z, stack)$

Precondition: $u$ and $z$ are vertices in a graph, stack contains current path.
Postcondition: returns true if path from $u$ to $z$ exists, $stack$ contains path.

1. $colour[u] \leftarrow \text{RED}$
2. Push $u$ onto $stack$.
3. If $u = z$
   - Return TRUE.
4. For each $v \in \text{Adj}[u]$ //explore edge $(u,v)$
   - If $color[v] = \text{BLACK}$
     - If DFS-Path$(v, z, stack)$
       - Return TRUE.
5. $colour[u] \leftarrow \text{GRAY}$
6. Pop $u$ from $stack$.
7. Return FALSE.
DFS Application 2: Cycle Finding

- The DFS pattern can be used to determine whether a graph is acyclic.
- If a back edge is encountered, we return true.

DFS-Cycle ($u$)

Precondition: $u$ is a vertex in a graph $G$
Postcondition: returns true if there is a cycle reachable from $u$.

1. $\text{colour}[u] \leftarrow \text{RED}$
2. For each $v \in \text{Adj}[u]$, explore edge $(u,v)$
   - If $\text{color}[v] = \text{RED}$, then return true
   - Else if $\text{color}[v] = \text{BLACK}$
     - If DFS-Cycle($v$) is true, then return true
3. $\text{colour}[u] \leftarrow \text{GRAY}$
4. Return false
Why must DFS on a graph with a cycle generate a back edge?

- Suppose that vertex $s$ is in a connected component $S$ that contains a cycle $C$.
- Since all vertices in $S$ are reachable from $s$, they will all be visited by a DFS from $s$.
- Let $v$ be the first vertex in $C$ reached by a DFS from $s$.
- There are two vertices $u$ and $w$ adjacent to $v$ on the cycle $C$.
- wlog, suppose $u$ is explored first.
- Since $w$ is reachable from $u$, $w$ will eventually be discovered.
- When exploring $w$’s adjacency list, the back-edge $(w, v)$ will be discovered.
DFS Application 3. Topological Sorting  
(e.g., putting tasks in linear order) 

Note: The textbook also describes a breadth-first TopologicalSort algorithm (Section 13.4.3)
A directed acyclic graph (DAG) is a digraph that has no directed cycles.

A topological ordering of a digraph is a numbering \( v_1, \ldots, v_n \) of the vertices such that for every edge \((v_i, v_j)\), we have \( i < j \).

Example: in a task scheduling digraph, a topological ordering is a task sequence that satisfies the precedence constraints.

**Theorem**

A digraph admits a topological ordering if and only if it is a DAG.
Topological (Linear) Order

underwear ➔ pants ➔ socks ➔ shoes ➔ underwear

socks ➔ underwear ➔ pants ➔ socks ➔ shoes

pants ➔ socks ➔ shoes ➔ underwear ➔ pants

shoes ➔ socks ➔ underwear ➔ pants ➔ shoes

underwear ➔ pants ➔ socks ➔ shoes ➔ underwear

socks ➔ underwear ➔ pants ➔ socks ➔ shoes

pants ➔ socks ➔ shoes ➔ underwear ➔ pants

shoes ➔ socks ➔ underwear ➔ pants ➔ shoes
Topological (Linear) Order

underwear -> pants -> shoes -> socks

Invalid Order:
underwear -> shoes -> pants -> socks
Algorithm for Topological Sorting

Note: This algorithm is different than the one in Goodrich-Tamassia

**Method** TopologicalSort(G)

\[ H \leftarrow G \] // Temporary copy of G

\[ n \leftarrow G.numVertices() \]

**while** \( H \) is not empty **do**

Let \( v \) be a vertex with no outgoing edges

Label \( v \leftarrow n \)

\[ n \leftarrow n - 1 \]

Remove \( v \) from \( H \) //as well as edges involving \( v \)
Linear Order

Pre-Condition:
A Directed Acyclic Graph (DAG)

Post-Condition:
Find one valid linear order

Algorithm:
• Find a terminal node (sink).
• Put it last in sequence.
• Delete from graph & repeat

Running time: $\sum_{i=1}^{V} i = O\left(|V|^2\right)$

Can we do better?

\[ \text{Can we do better?} \]
Linear Order

Alg: DFS

Found
Not Handled
Stack

Stack:

f
g
e
d
When node is popped off stack, insert at front of linearly-ordered “to do” list.

Linear Order:

- a
- b
- c
- d
- e
- g
- f
- h
- i
- j
- k
- l

Found
Not Handled
Stack:

Alg: DFS
Linear Order

Alg: DFS

Found
Not Handled
Stack

g
ed

Linear Order:
l,f
Linear Order

Alg: DFS

Found
Not Handled
Stack

g, l, f

Linear Order:
Linear Order

Alg: DFS

Found
Not Handled
Stack

d

Linear Order:

e, g, l, f
Linear Order

Alg: DFS

Linear Order:

d, e, g, l, f
Linear Order:

```
-85-  d,e,g,l,f
```

Found
Not Handled
Stack

```
k
j
i
```
Linear Order
Alg: DFS

Found
Not Handled
Stack

j
i

Linear Order: k,d,e,g,l,f
Linear Order
Alg: DFS

Linear Order: j,k,d,e,g,l,f

Found
Not Handled
Stack

i
Linear Order: $a, b, h, c, i, d, e, k, g, f, l$
Linear Order
Alg: DFS

Linear Order: $i, j, k, d, e, g, l, f$
Linear Order

Alg: DFS

Found
Not Handled
Stack

b

Linear Order:  c,i,j,k,d,e,g,l,f
Linear Order
Alg: DFS

Found
Not Handled
Stack

Linear Order: b,c,i,j,k,d,e,g,l,f
Linear Order
Alg: DFS

Linear Order: b,c,i,j,k,d,e,g,l,f
Linear Order
Alg: DFS

Linear Order: h, b, c, i, j, k, d, e, g, l, f
Linear Order

Alg: DFS

Linear Order: a, h, b, c, i, j, k, d, e, g, l, f  Done!
DFS Algorithm for Topological Sort

Makes sense. But how do we prove that it works?
Linear Order

Proof: Consider each edge

• Case 1: u goes on stack first before v.
  • Because of edge, v goes on before u comes off
  • v comes off before u comes off
  • v goes after u in order. 😊

\[ \begin{array}{c}
\text{Found} \\
\text{Not Handled}
\end{array} \]

Stack

\[ \begin{array}{c}
v \\
\vdots \\
u \\
\vdots
\end{array} \]

\[ u \rightarrow v \]

u ... v ...
Linear Order

Proof: Consider each edge

- Case 1: u goes on stack first before v.
- Case 2: v goes on stack first before u.
  v comes off before u goes on.
- v goes after u in order. 😊
Linear Order

Proof: Consider each edge
• Case 1: u goes on stack first before v.
• Case 2: v goes on stack first before u.
  v comes off before u goes on.
Case 3: v goes on stack first before u.
  u goes on before v comes off.
• Panic: u goes after v in order. 😞
• Cycle means linear order
  is impossible 😊

The nodes in the stack form a path starting at s.

u • ——> v
Linear Order

Algorithm: DFS

Analysis: $\Theta(V+E)$

Linear Order: a,h,b,c,i,j,k,d,e,g,l,f  Done!

Found
Not Handled
Stack
DFS Application 3. Topological Sort

Topological-Sort(G)
Precondition: G is a graph
Postcondition: all vertices in G have been pushed onto stack in reverse linear order

for each vertex $u \in V[G]$
    color[$u$] = BLACK //initialize vertex
for each vertex $u \in V[G]$
    if color[$u$] = BLACK //as yet unexplored
        Topological-Sort-Visit($u$)
DFS Application 3. Topological Sort

Topological-Sort-Visit \((u)\)

Precondition: vertex \(u\) is undiscovered

Postcondition: \(u\) and all vertices reachable from \(u\) have been pushed onto stack in reverse linear order

\[
\text{colour}[u] \leftarrow \text{RED}
\]

\[
\text{for each } v \in \text{Adj}[u] \quad //\text{explore edge } (u,v)
\]

\[
\text{if } \text{color}[v] = \text{BLACK}
\]

\[
\text{Topological-Sort-Visit}(v)
\]

\[
\text{push } u \text{ onto stack}
\]

\[
\text{colour}[u] \leftarrow \text{GRAY}
\]
Outline

- DFS Algorithm
- DFS Example
- DFS Applications