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
Depth-First Search

- 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 (Discovered 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
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 \( \text{color}[u] = \text{BLACK} \quad //\text{as yet unexplored} \)

DFS-Visit(\( u \))
DFS Algorithm Pattern

DFS-Visit \((u)\)

Precondition: vertex \(u\) is undiscovered

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

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

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

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

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

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

Property 1

$DFS-Visit(u)$ visits all the vertices and edges in the connected component of $u$

Property 2

The discovery edges labeled by $DFS-Visit(u)$ 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

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

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

DFS-Visit \( (u) \)
Precondition: vertex \( u \) is undiscovered
Postcondition: all vertices reachable from \( u \) have been processed

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

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

\[
\text{if } \text{color}[v] = \text{BLACK} \\
\text{DFS-Visit}(v)
\]

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

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:

- \( d[v] = \) discovery time.
- \( f[v] = \) finishing time.

\[
1 \leq d[v] < f[v] \leq 2 |V|
\]
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 \( \text{color}[u] = \text{BLACK} \quad //\text{as yet unexplored} \)

\[
\text{DFS-Visit}(u)
\]
DFS Algorithm with Discovery and Finish Times

DFS-Visit \((u)\)

Precondition: vertex \(u\) is undiscovered
Postcondition: all vertices reachable from \(u\) have been processed

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

Discovered
Not Finished
Stack

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

Discovered
Not Finished
Stack

〈node,# edges〉

s,0
DFS

Discovered
Not Finished

Stack

\[ \langle \text{node}, \# \text{edges} \rangle \]

- \( a, 0 \)
- \( s, 1 \)
DFS

Discovered
Not Finished

Stack

<node,# edges>

s

1/

a

2/

b

/

c,0

d

/

a,1

e

/

s,1

f

/

gh

/

i

/
j

m

/
DFS

Discovered
Not Finished
Stack

\langle\text{node}, \# \text{edges}\rangle

s, 1
c, 1
h, 1
a, 1
k, 0

Stack

\langle\text{node}, \# \text{edges}\rangle

s, 1
a, 1
c, 1
d, 0
e, 0
f, 0
h, 1
i, 0
j, 0
k, 0
l, 0
m, 0

Discovered
Not Finished
DFS

Discovered
Not Finished
Stack

Path on Stack

Discovery tree Edge

- 26 -

Last Updated December 3, 2015
DFS

Discovered
Not Finished
Stack

<node,# edges>

Stack

<node,# edges>

c, 1
a, 1
s, 1
Discovered
Not Finished
Stack

<node, # edges>

DFS

s

Stack

<node, # edges>

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

h

k

2/
3/
4/7
5/6
6/7
8/

1/
/
Cross Edge to Finished node: d[h]<d[i]

Discovered
Not Finished
Stack

<node,# edges>

a,1
b,1
c,2
d,1
e,1
f,1
g,1
h,1
i,1
j,1
k,1
l,1
m,1
n,1
o,1
p,1
q,1
r,1
s,1
t,1
DFS

Discovered
Not Finished

Stack

<node,# edges>

1,0
c,2
a,1
s,1

Stack:

a,1
c,2
i,3
b,1
g,0
h,1
1,0
s,1
l,1
k,1
f,2
d,1
3/1
2/1
4/7
5/6
8/1
9/1
1/1
/1
/1
/1
/1
/1
DFS

Discovered
Not Finished

Stack

<node,# edges>

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

Discovered
Not Finished

Stack

<node,# edges>

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

1, 3
2, 3
1, 1
1, 1
DFS

Discovered
Not Finished
Stack

<node,# edges>

a, 1
b, / 
c, 2
d, / 
e, / 
f, / 
g, 0
h, 4
i, 4
j, / 
k, 2
l, / 
m, 1

Stack

s

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

Discovered
Not Finished
Stack

<node,# edges>

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

[Diagram of a graph with nodes and edges labeled with numbers indicating discovered and not finished states.]

Last Updated December 3, 2015
DFS

Discovered
Not Finished

Stack

\[\text{Back Edge to node on Stack:}\]

\[\text{Stack}: \langle \text{node, \# edges}\rangle\]

\[\text{a, 1}, \text{c, 2}, \text{i, 4}, \text{g, 1}, \text{j, 1}, \text{a, 1}, \text{s, 1}\]
DFS

Discovered
Not Finished

Stack

<node,# edges>

\[
\begin{array}{c}
\text{a,1} \\
\text{b} \\
\text{c,2} \\
\text{d} \\
\text{e} \\
\text{f} \\
\text{g,1} \\
\text{h} \\
\text{i,4} \\
\text{j,2} \\
\text{k,2} \\
\text{m,0} \\
\text{s,1}
\end{array}
\]
DFS

Discovered
Not Finished
Stack

<node,# edges>

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

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

Discovered
Not Finished

Stack

<node,# edges>

\[
\begin{align*}
& j,2 \\
& g,1 \\
& i,4 \\
& c,2 \\
& a,1 \\
& s,1
\end{align*}
\]
DFS

Discovered
Not Finished

Stack

<node,# edges>

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

Discovered
Not Finished
Stack

<(node, # edges)>
DFS

Discovered
Not Finished

Stack

<node,# edges>

a,1
c,2
d,1

b,1
e,1

f,1
g,1

h,1

i,5

j,1

k,1

l,1

s,1

Stack:

<node,# edges>

1/ a
2/ c
3/ f
4/ i
5/ c
6/ j
7/ b
8/ e
9/ i
10/ a
11/ f
12/ b
13/ g
14/ j
15/ s
16/ h
17/ k
18/ l

Discovered:

1/ a
2/ c
3/ f
4/ h
5/ i
6/ j
7/ b
8/ e
9/ i
10/ a
11/ f
12/ b
13/ g
14/ j
15/ s
16/ h
17/ k
18/ l
DFS

Discovered
Not Finished

Stack

<node,# edges>

1/

a,1

b

c,2

d

1/1

17/18

e

11/16

f

1/2

i,5

c,2

j

12/15

k

5/6

h

3/3

4/7

l

9/10

m

13/14

n

EECS 2011
Prof. J. Elder

Last Updated December 3, 2015
DFS

Not Finished

Stack

<node,# edges>

c,2
a,1
s,1

Discovered

Stack

<node,# edges>

1/

2/

3/

4/7

5/6

8/19

11/16

12/15

13/14

14/18

15/17

16/11

17/18

18/
End of Lecture

Dec 3, 2015

The final exam will concern material only up to this point.
DFS

Discovered
Not Finished
Stack

<node,# edges>

s, l
DFS

Discovered
Not Finished
Stack

<node,# edges>

d,0
s,2

Example Diagram of a DFS traversal with labeled nodes and edges.
DFS

Discovered
Not Finished

Stack

<node, # edges>

d, 1
s, 2
DFS

Discovered
Not Finished
Stack

<node,# edges>

d,2
s,2

- 53 -
DFS

Discovered
Not Finished

Stack

<node,# edges>

e,0
d,3
s,2
DFS

Discovered: 1
Not Finished:

Stack: <node, # edges>

- s
- d
- j
- e

a, 2/20
b, / e, 22/

1/
21/
11/16
22/
3/19
17/18
8/19
12/15
4/7
13/14
5/6
9/10
14/13
15/12
16/11
17/10
18/9
19/8
20/7
21/6
22/5

Deep First Search (DFS)

Discovered: Not Finished

Stack: 

<node, # edges>

- a, 2/20
- b
- c, 3/19
- d, 21/20
- e, 22/23
- f, 17/18
- g, 11/16
- h, 8/19
- i
- j, 12/15
- k, 5/6
- l, 9/10
- m, 13/14
- s, 1/1
- d, 3
- s, 2
DFS

Discovered
Not Finished
Stack

<node,# edges>

s,2
DFS

Discovered
Not Finished
Stack

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

s,3
DFS

Discovered:

Not Finished:

Stack:

<node,# edges>

b,0
s,4
Discovered
Not Finished
Stack

<node,# edges>

DFS

Stack

<node,# edges>

a, 8/19
b, 25/1
s, 1/
c, 3/19
d, 21/24
e, 22/23
f, 17/18
g, 11/16
h, 4/7
i, 8/19
j, 12/15
k, 5/6
l, 9/10
m, 13/14

b, 2
s, 4
DFS

Discovered
Not Finished
Stack

<node,# edges>

b,3
s,4
DFS

Discovered
Not Finished
Stack

\[
<\text{node,\# edges}>
\]

s,4
DFS

Discovered
Not Finished
Stack

Tree Edges
Back Edges
Forward Edges
Cross Edges

Finished!

<node,# edges>
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**.
DFS on Undirected Graphs

- 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 **Finished** (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 Finished: \(u\) must be **grey**.

- **Contradiction!**
Outline

- DFS Algorithm
- DFS Example
- DFS Applications
DFS Algorithm Pattern

DFS-Visit \((u)\)

Precondition: vertex \(u\) is undiscovered

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

\[
\begin{align*}
\text{colour}[u] & \leftarrow \text{RED} \\
\text{for each } v \in \text{Adj}[u] & \text{ //explore edge } (u,v) \\
\text{if color}[v] = \text{BLACK} & \\
& \text{DFS-Visit}(v) \\
\text{colour}[u] & \leftarrow \text{GRAY}
\end{align*}
\]

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

(assuming adjacency list structure)
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.

- \( colour[u] \leftarrow \text{RED} \)
- push \( u \) onto \( stack \)
- if \( u = z \)
  - return \( \text{TRUE} \)
- for each \( v \in \text{Adj}[u] \) //explore edge \((u,v)\)
  - if \( \text{color}[v] = \text{BLACK} \)
    - if DFS-Path\((v,z,stack)\)
      - return \( \text{TRUE} \)
  - \( colour[u] \leftarrow \text{GRAY} \)
- pop \( u \) from \( stack \)
- return \( \text{FALSE} \)
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 $\text{colour}[v] = \text{RED}$  //back edge
     - return true
   - else if $\text{colour}[v] = \text{BLACK}$
     - if $\text{DFS-Cycle}(v)$
       - 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.
Outline

- DFS Algorithm
- DFS Example
- DFS Applications