Lecture 11:
• Depth-First Search
• Checking for cycles

Announcements
• PS-3 due
• HW-2 out, due Thurs

Depth-First Search (DFS)

Basic Idea: similar to how we typically search a tree ...
• start with a vertex s
• find an unvisited path reachable from s
• backtrack to find other unexplored reachable paths from s
• repeat until all such paths are explored

Two standard ways to implement DFS:
• recursively
• iteratively using a stack ... HW-3

We’ll (briefly) go over both
Recursive BFS

**Algorithm:** Recursive DFS

**Input:** A graph $G = (V, E)$ and a source vertex $s \in V$

**Result:** Dictionary mapping discovered vertices to their parent vertices

1 begin
2 mark all vertices $v \in V$ as not visited
3 set parent of $s$ to -1 (denoting the root)
4 DFS-Visit($G$, $s$, visited list, search tree)

---

**Algorithm:** DFS-Visit

**Input:** Graph $G = (V, E)$, source vertex $s \in V$, visited list, and search tree

**Result:** Search tree and visited list filled in

1 begin
2 mark $s$ as visited
3 for $v$ adjacent to $s$ in $E$ do // or out nodes if directed
4 if $v$ is not visited then
5 set $v$’s parent to $s$
6 DFS-Visit($G$, $v$, visited list, search tree)

---

Check-In

1. Trace the DFS algorithm starting from vertex $s$

2. Trace again from $s$ but assume the graph is undirected
DFS Analysis

Analysis
- similar to BFS, $O(V + E)$, $O(V)$ init visited, visit all edges $O(E)$
- note that like BFS, assumes checking visited is $O(1)$

Check In:
- Can we use DFS instead of BFS to find (weakly) connected components?
- Can we use DFS to find shortest paths?

Implementing DFS using Iteration

Basic Idea:
- similar to BFS
- but use a stack instead of a queue ... LIFO versus FIFO
- note this is what the recursion does “for us” ... plus call overhead
- watch out for self edges!

Check In: repeat example but using iteration (i.e., a stack)

Note: Must implement DFS iteratively for HW-3
**DFS Edge Types**

*DFS Edge Types* ... based on the resulting DFS search tree

1. **Tree edges** \((u, v)\) are search tree edges (i.e., \(v\) discovered from \(u\))
2. **Back edges** \((v, u)\) connect \(v\) to ancestor vertex \(u\) in the search tree
3. **Forward edges** \((u, v)\) connect \(u\) to descendent vertex \(v\) in the search tree
4. **Cross edges** \((u, v)\) go between vertices in the search tree

Note often DFS defined over entire graph (not just a single source vertex)
- in which case, we have a search “forest” (collection of search trees)
- the same idea can be applied to BFS (which we’ve already seen)

---

**Modified Recursive DFS**

**Algorithm:** Recursive DFS  
**Input:** A graph \(G = (V, E)\) and a source vertex \(s \in V\)  
**Result:** Dictionary mapping discovered vertices to their parent vertices

```
begin
1 assign each vertex \(v \in V\) the color WHITE
2 set parent of \(s\) to -1 (denoting the root)
3 DFS-Visit(G, s, visited list, search tree)
end
```

**Algorithm:** DFS-Visit  
**Input:** Graph \(G = (V, E)\), source vertex \(s \in V\), visited list, and search tree  
**Result:** Search tree and visited list filled in

```
begin
1 mark \(s\) the color GRAY
2 for \(v\) adjacent to \(s\) in \(E\) do // or out nodes if directed
3 if \(v\) is assigned the color WHITE then
4 set \(v\)’s parent to \(s\)
5 DFS-Visit(G, v, visited list, search tree)
6 assign \(v\) the color BLACK
```
Check-In

Trace the following graph using the modified DFS algorithm

What do you notice regarding vertex coloring and edge types?

- if we come across a node that’s WHITE, found a tree edge
- if we come across a node that’s GRAY, found a back edge ... a cycle!
- if we come across a node that’s BLACK, either forward or cross edge

Cycle Checking

For HW-3: checking if a graph is acyclic ... no cycles

- re-implement iterative DFS to check for back edges
- that is, during traversal check to see if we arrive at a GRAY vertex
- if found, graph is cyclic (i.e., not acyclic)
- results in an $O(V + E)$ algorithm to check for cycles