# CMPSCI 250: Introduction to Computation

Lecture #25: Depth-First and Breadth-First Search on Graphs

David Mix Barrington

28 March 2012

# Depth-First and Breadth-First Search on Graphs

- Storing the Entire Search Space
- The DFS Tree of a Undirected Graph
- The DFS Tree of a Directed Graph
- Four Kinds of Edges
- The BFS Tree of a Undirected Graph
- The BFS Tree of a Directed Graph

## Storing the Entire Search Space

- In CMPSCI 311 you'll spend considerable time on search problems where the entire graph is given to you, usually as an **adjacency list** where for each node we have a list of the edges out of it.

- Given two nodes s and t in the graph, we can ask *whether* there is a path from s to t, *how long* the *shortest* path from s to t might be (measured by number of edges or measured by the total cost of the edges), or whether s and t remain connected if certain edges are deleted.

- With the whole graph stored (or just with a closed list) we avoid processing the same node twice.

- Both DFS and BFS on graphs will allow us to create a **tree** from the graph, which will allow us to address these problems more easily.

# The DFS Tree of an Undirected Graph

- Recall that our DFS algorithm places nodes onto a stack when they are discovered, and processes all their edges when they are taken off the stack.
- Our DFS tree will have a **tree edge** from s to t if we encounter t for the first time while we are processing s, that is, if we discover t through its edge from s. The tree edges form a tree that gives a path from the start node to each node that is reachable from it.
- If we defined the DFS recursively, the DFS tree would be essentially the call tree, because if (s, t) were a tree edge we would make the recursive call with parameter t in the course of processing the call with parameter s.
- A DFS of an undirected graph searches the entire **connected component** of the start node. What can we tell about the edges that aren't tree edges?

# Tree Edges and Back Edges

- Let G be a connected undirected graph and let T be its DFS tree. If G were a graph-theoretic tree, T and G would be the same graph (more precisely, T would be the rooted tree made from G with the start node as root).

- But if while processing node s, we find an edge to a node t that is *not* new, that edge does not go into T. (We'll ignore the reverse directions of tree edges.) Note that the processing of t must still be going on at this point, because we don't finish processing t until we've finished all the nodes reachable from it, including s. So t must be an **ancestor** of s in the tree, and the edge (s, t) is thus called a **back edge**.

- We'll see an example on the board where the undirected graph G becomes a rooted tree T together with some back edges.

- An **articulation point** is a node whose removal disconnects the graph. Can you tell what condition on the tree and back edges makes t such a point?

# The DFS Tree of a Directed Graph

- When we make a DFS of a directed graph, we still reach every node that is reachable from the start node. But it's no longer guaranteed that any or all of those nodes have paths back to the start point -- we no longer have a connected component to search.

- On HW#6 you'll work out how to use the DFS algorithm to find the **strongly connected components** of a directed graph -- the equivalence classes of the equivalence relation $P(x, y) \wedge P(y, x)$. If there is a back edge from a node t to an ancestor u, then all the nodes on the tree path from u down to t are in the same strongly connected component because they lie on a directed cycle.

- We can no longer guarantee that all the edges are either tree edges or back edges -- what are the other possibilities?

# Four Kinds of Edges

- Let (u, v) be an arbitrary edge in a directed graph G. In what different ways could (u, v) be encountered in a DFS of G?

- If we find u before v and first find v through the edge (u, v), it is a **tree edge**.

- If we find u before v, but find v through one of its siblings before we look at the edge (u, v), then (u, v) becomes a **forward edge** from u to a descendant.

- If we find v before u, and find u while we are still processing v, then the edge (u, v) becomes a **back edge** just as in the undirected case.

- If we find v before u and finish v before finding u (because there is no path from v to u), then (u, v) becomes a **cross edge**.

# The BFS Tree of an Undirected Graph

- A breadth-first search gives rise to tree edges in the same way -- (u, v) is a tree edge if we encounter v during the processing of u, and put v on the queue. The **BFS tree** is made up of all the tree edges, and is a rooted tree giving a shortest path (in number of edges) from the start node to each edge. (If there are multiple shortest paths, the algorithm will choose one as the tree path.)

- If u is at level k of the tree, and (u, v) is a non-tree edge, we know that v has already been put on the queue before the edge is seen. If it is still on the queue, it must be also at level k. If it has been finished, it must be at level k-1, because otherwise (in an undirected graph) we would have missed a shorter path from the start node to u by way of v.

- An undirected graph is **bipartite** if and only if we never get an edge from one node to another at the same level. (This follows from the theorem that an undirected graph is bipartite if and only if it has no **odd-length cycles**.)

# The BFS Tree of a Directed Graph

- In a BFS of a directed graph, the BFS tree will arrange the nodes into levels, based on their shortest-path distance from the start node (where again "shortest" means "fewest edges").

- If u is at level k and we find v for the first time while processing u, then (u, v) will be a tree edge and v will be at level k + 1.

- But if v has already been seen, it might be at *any* existing level of the tree from 0 to k or even k + 1, or might even not be in the tree at all! Remember that if a DFS or BFS finishes without reaching all the nodes, we start a new tree at a new start point. The node v might be in an earlier tree, which didn't contain a path *to* u, but still have an edge *from* u.