Harvard University
CS 124

Lecture 3

Lecture Notes by Anupa Murali

February 2, 2015

1 Graphs

How do we represent a graph on a computer? Most popular forms of representation are,

1. Adjacency matrix: n is the number of vertices. It is an $n \times n$ matrix, entry is 1 if (i, j) is in E, 0 otherwise.
2. Adjacency list: An array A of linked lists. $A[i]$ is a list of all j such that $(i, j) \in E$.

Graphs can be weighted.

	Adj. Matrix	Adj. List
space	n^2	$\Theta(n+m)$
$(i,j) \in E$?	$\Theta(1)$	$O(n)$ or $\Theta(\deg(i))$

How can we modify the Adjacency List representation such that it doesn't take $\Theta(\deg(i))$ to find an element and to insert or delete an element? One possibility is to use hashing, which gives $O(1)$ expected time. Another is to use balanced binary search trees, which ail give $O(\log \deg(i))$ search time.

2 Graph Algorithms

What are some things we want to do on a graph? Maybe your graph represents a road network for islands, and on the islands are cities connected by islands. One might want to figure out which cities are reachable by each other. We want to separate it into connected components. How can we explore the graph?

1. Depth-first search (DFS)
2. Breadth-first search (BFS)

Both do the same job but visit the vertices in different orders.

2.1 Depth-First Search

Suppose we have a global array called **visited** of length n, the number of vertices. Also suppose we have a procedure called search that is given an input v, a vertex. Following is the search() procedure.

search(v):
    visited[v] = True
    for u s.t. $(v, u) \in E$
        if visited[u] = False
            then search[u].

DFS:
dfs($G(V, E)$):
    visited = [False, False,. . . ,False]
    for $u \in V$:
        if visited[u] = False
            search(u)

We can define intervals for each vertex capturing when it is being processed, Let pre(v) be preorder and post(v) be postorder. For each vertex v we give a [pre(v), post(v)] pair. Preorder is when we begin processing the edge, postorder is when we finish processing it (its recursion stack is empty).

Runtime of DFS: Suppose every vertex was connected to one another. Then it would be n^2 time. So it is at most $O(n^2)$. In the adjacency list representation, it is $\Theta(n + m)$. In the adjacency matrix representation it is $\Theta(n^2)$.

DFS can solve:

1. Articulation points
2. biconnected components
3. strongly connected components
4. planarity testing
5. isomorphism of planar graphs
6. max-weighted matching

Claim: For any vertices $u \neq v$, their [pre,post] intervals are either disjoint or one is contained in the other.

Proof: Either one vertex is the ancestor of the other, or they're in different "trees". Containment happens because of an ancestor-child relationship.

More Notation: We can categorize edges into various types based on behavior of DFS. What are the types?

1. **forest edge**: Edge of the graph that appears in the DFS forest.
2. **forward edge**: Goes from ancestor to descendant. Doesn't show up in forest.
3. **back edge**: Goes from descendant to ancestor
4. **cross edge**: None of the above.

Claim: Suppose $(u, v) \in E$. Then the post-order number of u is at most the post-order number of v if and only if (u, v) is a back edge.

Proof: Proving that if it is a back edge, the post order number of u is at most the post order number of v is trivial. The post order number is just the time at which the function call is popped off the recursion stack, so the ancestor will be popped off after the descendant.

When you call search on u, you are going to call search on the edge from u to v and recurse on v if necessary. Suppose they are disjoint. Since $\text{post}(u) \leq \text{post}(v)$, their edges look like [][] where u is the red one and v is the blue one. So we visited u first. This means we haven't visited v yet, so we haven't done the recursive call. So this is impossible. This means that they must be nested. If they're nested, they look like [[]]. So (u, v) is a back edge.

□

Claim: G has a cycle if and only if G has at least one back edge.

Proof: Suppose (u, v) is a back edge. Then if you draw the DFS forest, v is the ancestor of u, so we know that there is some path from v to u. But then there is a back edge (u, v), so there is also an edge from u to v. This is a cycle.

Now let's prove the other direction. Suppose G has a cycle. Consider a cycle $\{v_1, v_2, \ldots, v_l, v_1\}$. Let the vertex v_i have the **smallest** post order in the cycle. Then v_i, v_{i+1} is a back edge since v_{i+1} has post-order less than or equal to that of v_i.

□