Algorithms in Systems Engineering
IE170

Lecture 15

Dr. Ted Ralphs

References for Today's Lecture

- Required reading
  - CLRS Chapter 22
- References
  - R. Sedgewick, *Algorithms in C++* (Third Edition), 1998.

Searching a Graph

- In the last lecture, we introduced a method of *searching* a graph using a technique called *depth-first search* (DFS).
- *Graph search* is a generalization of this method that is used to study the structure of a graph.
- We have already used graph search on several occasions.
- In the next few lectures, we will consider several methods of searching a graph.
- Each method will reveal something different about the structure of the graph.
- Many, many algorithms are based on this general framework.
  - Finding a (shortest) path between two vertices in a graph.
  - Determining whether a graph has a cycle.
  - Determining a minimal set of edges that connect all the vertices.
  - Determining whether there is a single edge/vertex whose removal disconnects the graph.

General Graph Search

- *Graph search* consists of systematically *processing* the vertices of a graph to discover some property of the graph.
- To search a single component:
  - Choose a start vertex and add it to the list of unprocessed vertices.
  - Repeat until no vertices remain on the list.
    * Choose a vertex v from the list of unprocessed vertices.
    * Process v.
    * Add all the neighbors of v to the list of unprocessed vertices.
- To search multiple components, we must have a method of finding a start vertex in each component.
- Note that generally each vertex only needs to be processed once, but may be placed on the list more than once.
- Typically, however, we only allow each vertex to be added to the list once.
- What do we need to specify to actually implement graph search?

Types of Graph Search

- Note that we have left three basic components unspecified in our description of graph search.

- The way in which these three steps are implemented determines the overall running time of the algorithm.
- The various options result in a rich class of algorithms that can answer many interesting questions about a given graph.

Depth-first Search

- In the last lecture, we introduced the depth-first search algorithm for determining the components of a graph.
- In DFS, the vertices are processed in last-in, first-out (LIFO) order.
- How do we implement this?

- Recall the maze exploration program from Lab 3.
  - The maze can be viewed as a graph (how?).
  - We used a stack implementation to explore this graph using DFS.
- To avoid adding a vertex to the list more than once, we can mark it the first time it is added to the list.
- In order to completely specify the algorithm, we still need to determine the order in which the neighbors of a vertex are added to the list.

Running Time of Depth-first Search

- The running time of DFS depends essentially on the running time of the processing step.
  - Assuming that the processing time for one vertex is in $\Theta(f(m, n))$, the total processing time is in
  - The time spent maintaining the list of unprocessed vertices is
  - To determine a starting vertex for each component, we must do a linear search for a total time in
- This gives a total running time of
- Note that in practice, it is almost always the situation that $n = O(m)$.

Using DFS

- Determining the components of a graph.
  - In the last lecture, we used DFS to determine the components of a graph.
  - The processing step consisted of marking each vertex with a component number (constant time).
- Finding a path from one vertex to another.
  - In this situation, the processing step consists of checking to see whether the destination vertex has been reached.
  - We must also keep track of the path itself.
  - The path can be tracked using a stack, as in Lab 3.
- Determining whether a graph has a cycle can be accomplished by trying to find a path from a vertex to itself.
- The total running time for all these is $\Theta(m + n)$.

Trees

- We've already discussed trees in several contexts, but now we can give a more rigorous definition.
- In graph terminology, a *tree* is a connected graph with no cycles and a *forest* is a graph consisting of a collection of trees.
- Properties of trees
  - Every tree has exactly $n - 1$ edges.
  - In a tree, there is a unique path from any given vertex to any other vertex.
- A tree that has a specified *root vertex* is called a rooted tree.
  - In a rooted tree, there is a unique path from the root to every other vertex.
  - We can therefore uniquely define the parent of a vertex v as the vertex that immediately precedes it on the path from the root to v.
  - Hence, we are justified in thinking of trees in the way that we had previously, as a set of hierarchical relationships between the vertices.

Search Trees and Forests

- Consider searching a connected undirected graph $G = (V, E)$.
- The process of searching G can be captured by constructing a tree T called the *search tree*.
- T is constructed as the search evolves by adding an edge connecting the vertex currently being processed to any vertex not yet processed.
- This graph must be connected and acyclic, and hence is a tree.
- We can view it as a rooted tree by taking the root to be the start vertex.
- In graphs with multiple components, we can similarly obtain *search forests*.
- The term depth-first search derives from the observation that the next vertex to be processed is the vertex at maximum depth in this tree.
- DFS tends to produce very deep search trees.
- We can also consider other graph search algorithms.

Pre-order and Post-order

- The order in which the vertices are encountered and processed can be used to create a sequence.
- *Pre-order* is the order in which the vertices are first encountered and added to the list to be processed.
- *Post-order* is the order in which the vertices are actually processed.