Algorithms in Systems Engineering
IE170

Lecture 19

Dr. Ted Ralphs
References for Today’s Lecture

- Required reading
  - CLRS Chapter 22-24

- References
Another View of Prim’s Algorithm

- Last time, we derived Prim’s Algorithm as a special case of graph search.
- The algorithm can also be viewed as a special case of another general class of algorithms called greedy algorithms.
- A greedy algorithm is one that makes the choice at each step that looks the best “at the moment” and doesn’t reconsider that choice later.
- We can view the construction of an MST as a greedy algorithm, but first we must define some terminology.
- Given an undirected graph $G = (V, E)$, a cut is a set $S \subset V$ that defines a partition of $V$ into two nonempty subsets, $S$ and $V \setminus S$.
- An edge is said to cross the cut if it connects a node in $S$ to a node in $V \setminus S$.
- Our goal is to build a spanning tree by adding one edge at a time to a set $T$ in a “greedy” fashion.
- Basically, we just need to somehow guarantee ourselves that at each step, the current set can be “extended” to an MST.
- How do we do that?
Safe Edges

- Let’s assume that our current set of edges $T$ already satisfies the property that $T$ can be extended to an MST.

- **Question**: What edges can we add to $T$ to maintain the property?

- **Answer**:

- **Rationale**:

- We will call such edge a *safe edge* if it also doesn’t create a cycle when added to $T$.

- How do we find such an edge?
Generic Greedy Algorithm for Building an MST

• Generic greedy algorithm for constructing a spanning tree.
  – Set $T = \emptyset$.
  – Select a safe edge and add it to $T$.
  – Repeat until $T$ is a spanning tree.

• This is guaranteed to work, no matter how the safe edges are selected.
Kruskal’s Algorithm

- **Kruskal’s Algorithm** takes a more global view.

- At each step, we consider *all edges* that do not form a cycle when added to the current set $T$.

- The minimum such edge is guaranteed to be safe (why?).

- As edges are added, we’ll keep track of the current set of components using

- At each step, we’ll add the cheapest edge to $T$ that doesn’t connect two nodes currently in the same component.

- Implementing Kruskal’s Algorithm
Running Time of Kruskal’s Algorithm

- Kruskal’s Algorithm consists of two stages.
  - Sorting the edges by weight.
  - Performing $m$ `find()` and $n - 1$ `union()` operations.

- The first step takes
- The second step takes
- The total running time is
Directed Graphs

- Up until now, we’ve concentrated on undirected graphs.

- In a directed graph, or *digraph*, the connections between the vertices are ordered pairs called *arcs*.

- The set of vertices is typically denoted $N$ and the set of arcs is denoted $A$ with $m = |A| \leq n(n - 1)$.

- A directed graph $G = (N, A)$ is then composed of a set of vertices $N$ and a set of arcs $A \subseteq V \times V$.

- If $a = (i, j) \in A$, then
  - $i$ is called the *tail* of $a$ and $j$ is called the *head* of $a$,
  - $a$ is said to be *incident from* $i$ and *incident to* $j$, and
  - $i$ and $j$ are said to be *adjacent* vertices.

- For a given digraph, there is an *underlying undirected graph* obtained by ignoring the directions of the arcs (and eliminating parallel edges).
More Terminology

• Let \( G = (N, A) \) be a digraph.

• A subgraph of \( G \) is a digraph composed of an arc set \( A' \subseteq A \) along with all incident vertices.

• A subset \( V' \) of \( V \), along with all incident arcs is called an induced subgraph.

• A directed path in \( G \) is a sequence \( v_0, \ldots, v_p \) of vertices such that for each \( i \in 0, \ldots, p - 1 \), \( (v_i, v_{i+1}) \in A \).

• A directed path is simple if no vertex occurs more than once in the sequence.

• A directed cycle is a directed path that is simple, except that the first and last vertices are the same.

• A directed tour is a directed cycle that includes all the vertices.
Data Structures for Digraphs

- Data structures for digraphs are similar to those for undirected graphs.
- As before, there are two basic choices
  - Adjacency matrix
  - Adjacency lists
- These are implemented in similar fashion, except that
  - In the case of an adjacency matrix, the matrix is no longer symmetric.
  - In the case of an adjacency list, each arc appears only on the adjacency list of its tail vertex.
Connectivity in Digraphs

- A digraph is *connected* if the underlying undirected graph is connected.
- A digraph is *strongly connected* if for each pair of vertices \( i \) and \( j \), there is a directed path from \( i \) to \( j \) and a directed path from \( j \) to \( i \).
- A digraph that is not strongly connected consists of a set of *strongly connected components* that are the maximal strongly connected subgraphs.
- Given a digraph, one of the most basic questions one can ask is whether there is a path from vertex \( i \) to vertex \( j \).
- We can answer this question as we did before using a modified version of graph search.
- In *graph search* for directed graphs, we only examine the arcs that are incident from the vertex being processed.
- Performing graph search starting at vertex \( r \) results in the processing of all the vertices to which there is a path from \( r \).
Graph Search for Digraphs

- Graph search doesn’t have the same interpretation in the directed case because we cannot use it directly to find the (strongly) connected components.

- Graph search from vertex $r$:
  - As before, this graph search process results in construction of a directed search tree in which there is a path from $r$ to each other vertex.
  - Such a directed tree is said to be directed away from $r$.
  - This graph search algorithm can easily be adapted to find the shortest directed paths from $r$ to each other vertex.
  - It can also be used to find a minimum spanning tree directed away from $r$. 
Directed Acyclic Graphs

• Often, directed graphs are used to represent precedence relations.

• Such *precedence graphs* should be both directed and acyclic.

• Given a directed acyclic graph (DAG), one thing we would like to be able to do is determine an ordering of vertices that obeys the precedence relations.

• This is called a *topological sort*.

• Given a DAG, DFS can be used to perform a topological sort.
  – Each vertex is added to the front of a linked list after all of its neighbors have been processed.
  – The resulting linked list is a topological sort.
  – Why?