Algorithms in Systems Engineering
ISE 172

Lecture 22

Dr. Ted Ralphs

References for Today's Lecture

- Required reading
  - Chapter 7
- References
  - CLRS Chapter 22-24
  - R. Sedgewick, *Algorithms in C++* (Third Edition), 1998.

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?).
- However, we need a data structure to keep track of what edges form a cycle when added to the existing ones.

Connected Components Algorithm

- Suppose the graph is specified simply as a a list of edges.
- Algorithm
  - Start with each vertex in its own subset.
  - While there are still edges left on the list,
    * Read the endpoints `i` and `j` of the next edge.
    * Call `find(i, j)` to determine whether they are in the same subset.
    * If so, then call `union(i, j)`.
- After reading in all the edges, a call to `find(i, j)` will determine whether `i` and `j` are in the same connected component.
- The advantage of this method is that we never have to actually store the list of edges.
- We will also consider more efficient methods that require storing the edges.

Quick Find Implementation of Union-Find

- The simplest implementation involves an array of length n.
- We will maintain the array such that two items are in the same subset if and only if the array entries are equal.
- This makes the `find(i, j)` constant time, so we call this implementation *quick find*.
- How do we implement `union(i, j)`?
- What is the running time?
- Note that this could also be implemented using linked lists, as described in CLRS.

Quick Union Implementation of Union-Find

- To speed up the union operation, we maintain the array in a different fashion.
- We will consider the i^{th} entry of the array to be a pointer to another item.
- We start with the i^{th} entry of the array equal to i, i.e., all items pointing to themselves.
- To perform `find(i, j)`,
  - Follow the pointers from nodes i and j until reaching a node that points to itself, called the *representative*
  - If the same representative is reached from both nodes i and j, then they are in the same subset.
- To perform `union(i, j)`, perform the find operation and then point the representative for i to the representative for j.
- What is the performance now?

Weighted Quick Union

- Note that the quick union algorithm essentially builds a tree out of the nodes in each component, with the root begin the representative.
- As in binary search, the running time of the find operation depends on the depth of the trees.
- Each union operation essentially connects two trees together by pointing the root of one tree to the root of the other.
- One way to limit the depth of the tree is to always point the smaller tree to the larger one.
- This ensures that each find takes less than $\lg n$ steps.
- Note that we must now keep track of the number of nodes in each tree, but that's easy to do.
- Another approach is to keep track of the height of each tree and always point the shorter tree to the taller one.

Path Compression

- Ideally, we would like each item to point directly to the representative of its subset.
- One possibility is to simply keep track of all the nodes encountered in the path to the root.
- After reaching the root, set all the nodes on the path to point to the root.
- This is easy to implement recursively and doesn't change the asymptotic running time.
- An easier method to implement is *compression by halving*, which is setting each node to point to its grandparent.
- Combining weighted quick union with path compression yields a total running time for connected components of approximately $O(m)$.

Kruskal's Algorithm: Overall Implementation

- As edges are added, we will keep track of the current set of components using a union-find data structure.
- 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
  - Before beginning, sort the edges by weight and set $T = \emptyset$.
  - While there are unexamined edges on the list
    - Call the find() operation on the endpoints of each edge until an edge e is found for whose endpoints are in different components.
    - After adding e to T, call the union() operation to combine the components containing its endpoints into a single component.

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 $\Theta(m \lg m) = \Theta(m \lg n)$ time.
- The second step takes $O(m \lg n)$ time.
- The total running time is $O(m \lg n)$.
- Hence, Kruskal's Algorithm and Prim's Algorithm have the same running time.