Greedy Algorithms: Prim's Algorithm

- Read section 9.1 (pages 315-322)
- In order for an algorithm to be characterized as **greedy**, there are three important properties regarding the choice made by the algorithm at each point. What are those properties?
- Does a greedy algorithm always provide an optimal solution to the problem?
- What would a greedy algorithm to the coin-making problem be? Show by an example that it is not the optimal solution.
- There are three techniques that are typically used in order to prove that a greedy algorithm does in fact produce the optimal solution. Describe an example of each:
  - We can use induction to prove that the partially constructed solution at any step can in fact be extended to an optimal solution.
  - We can show that at each step the algorithm does at least as well as any other algorithm.
  - We can show that the final result obtained by the greedy algorithm is in fact the optimal result, based purely on that result.
- What is a **minimum spanning tree** for an undirected connected graph?
- How does **Prim's Algorithm** construct a minimum spanning tree?
- Demonstrate a run of Prim's algorithm on the graph in page 9.3, but starting from the vertex d instead.
- Write pseudocode for Prim's algorithm.
- A key part of Prim's algorithm is the ability to select a minimum-weight edge from a set of eligible edges. The paragraph following the algorithm describes the main way to approach this selection process. Describe this way in more detail:
  - Maintain a priority queue structure with all the vertices that are yet to be connected to the spanning tree, along with the weights of the edges that connect them to the tree's vertices.
  - When a new vertex is connected to the tree, remove it from this structure and adjust the weights of any edges that are adjacent to it.
- Show how a min-heap can be used to maintain this structure. How can we adjust the weight of an entry in the heap while maintaining the heap property?
  - If the weight goes up, percolate value down (min-heap).
  - If the weight goes down, bubble value up.

- What is the time efficiency of Prim's algorithm?
- Prove that Prim's algorithm produces a minimum spanning tree by proving that the edge added at each level must be part of a minimum spanning tree, as follows:
  - We can assume that the tree T_{i-1} built after $i-1$ steps is part of a minimum spanning tree T, and we now consider extending T_{i-1} with an edge e_i, which has the smallest weight out of all edges we can use at that point.
  - If that edge is not in T, then adding it to T forms a cycle. This cycle crosses from T_{i-1} to the rest of the graph in at least one other place via an edge v.
  - Replacing v with e_i in the graph T produces a spanning tree of smaller total weight, which is not possible.
- Practice problems: 9.1.1, 9.1.9, 9.1.15