

CS231 Fall 2013 Lecture 34

Stephanie R Taylor

November 22, 2013

1 Administrative Topics

- We take the quiz.

2 Traversing a graph

In many of the problems, we need to use some graph traversal algorithm where we are visiting all nodes. For example, all three problems for your project involve traversals. All these traversals can be done recursively or with stacks or queues. During the traversal, there are three categories of nodes: Nodes already visited, nodes that are unvisited neighbors of visited nodes, and the remaining nodes. We will mark the nodes when visited and we will keep a collection C of unvisited neighbors of visited nodes.

General graph traversal algorithm:

```
unmark all nodes
create a new empty collection C of nodes
choose a starting node and add it to C
while C is not empty do:
    remove a node n from C
    mark n as visited
    for all neighbors p of n:
        if p is unmarked and not already in C:
            add p to C
```

Dijkstra's shortest path algorithm

Dijkstra's algorithm determines the shortest path from a given node to all other nodes. The strategy is similar to a traversal in that it visits nodes and add neighbors to a queue. When removing elements from the queue, it picks the unvisited vertex with the lowest-distance, calculates the distance through it to each unvisited neighbor, and updates the neighbor's distance if smaller. It marks a node as visited when all its neighbors have been dealt with.

If the edges in the graph are not weighted, then Dijkstra's algorithm follows the same pattern as the breadth first search (BFS) we have outlined above. It is much more interesting to consider the case in which the edges are weighted. To do that, we need to develop a mechanism for storing the edge weights. I am going to do that by making an class called EdgeTo. It will store the weight of the edge and a reference to the GraphNode at the other end of the edge.

```
Given: a graph G and starting vertex v0 in G
Initialize all vertices in G to be unmarked and have infinite cost
Create a priority queue, q, that orders vertices by lowest cost
Set the cost of v0 to 0 and add it to q
while q is not empty:
    let v be the vertex in q with lowest cost
    remove v from q
    mark v as visited
    for each vertex w that neighbors v:
        if w is not marked and v.cost + v_w_edge_weight < w.cost:
            w.cost = v.cost + v_w_edge_weight
            add w to q
Output: the cost of each vertex v in G is the shortest distance from v0 to v.
```