CSCI 2270

Data Structures & Algorithms

Gabe Johnson

Lecture 33 Apr 10, 2013

Help on Dijkstra's Algorithm

Lecture Goals

1. Pseudocode for isDAG and Dijkstra routines
2. Tree and TreeNode Classes
3. STL Priority Queue Issues

Upcoming Homework Assignment

HW #9 **Due: Friday, Apr 12**

Dijkstra's Algorithm

This is mostly about doing Dijkstra's Algorithm. There's one question about DAGs that is otherwise unrelated.

isDAG

Is the graph directed?

Iterate through all nodes in the graph, and run a DFS on each (be sure to clear first). After each, iterate through the edges and look for BACK_EDGEs. They indicate cycles.

Dijkstra (start)

Don’t clear the graph. Driver will do that.
Set start’s path distance (pd) to 0.
Create a new Tree **tree** and set start to the root.
Make **PQ** containing *all* nodes in the graph.

Now we do a big while loop (next page)

Dijkstra (start)

```
While PQ isn't empty:
    current = PQ.pop (peek and remove)
    if current.pd is infinity, return tree
    adj_edges = edges adjacent to current
    for each edge edge in adj_edges:
        n = other end of edge
        if n is not in PQ, stop processing for n.
        d = current.pd + edge.weight
        if d < n.pd:
            n.pd = d
            update tree such that n is now current's child
            n.predecessor = current
            jiggle n's location in the PQ so it is accurate
return tree
```

Tree and TreeNode

```
class TreeNode {
public:
  TreeNode(Node* n);
  Node* node;
  vector<TreeNode*> children;
};

class Tree {
private:
  TreeNode* root;
  vector<TreeNode*> members;
  void print_tree(TreeNode* tn, int lvl);
public:
  TreeNode* find(Node* data);
  void update(Node* child, Node* old_parent, Node* new_parent);
  void setRoot(Node* root);
  TreeNode* getRoot();
  void print_tree();
};
```

Notice these are all public. I could have used a struct.

Using Tree

```
Tree* tree = new Tree;
tree->setRoot(a);
tree->update(b, NULL, a); // a -> b. No prior parent
tree->update(c, NULL, a); // a -> c
tree->update(d, NULL, b); // b -> d
// now display the tree
tree->print_tree();
```

```
a dist: 0
 . b dist: 0
 .   . d dist: 0
 . c dist: 0
```

Using Tree

```
Tree* tree = new Tree;
tree->setRoot(a);
tree->update(b, NULL, a); // a -> b. No prior parent
tree->update(c, NULL, a); // a -> c
tree->update(d, NULL, b); // b -> d
tree->update(d, b, c); // c -> d. inform b.
tree->print_tree();
```

```
a dist: 0
 . b dist: 0
 . c dist: 0
 .   . d dist: 0
```

STL Priority Issues

The priority_queue in C++'s standard template library is cool but it doesn't have a 'contains' function. You need to be able to tell if a variable has been reached.

A few ways of doing this:

1. Use a vector instead of a PQ, and just make sure you keep things in the right order.
2. Use a priority queue and a set, and add/remove nodes in parallel.

STL Priority Issues

3. Use the node's color. I actually didn't try this but I am confident it will work.

White = still in the PQ

Gray = popped from PQ