

Computer Science 62

Lab 13

Wednesday, April 28, 2010

This lab will be a warm-up for Assignment 11. You are to write a single-source shortest path function using breadth-first search. In the assignment itself, you will replace the queue with a priority queue to obtain a full implementation of Dijkstra's algorithm.

Getting Started

Read Assignment 11. Make a new directory for this lab. Copy either our version of `priorityqueue62` into your working directory from the Assignment 11 starter or copy your version.

Dijkstra's

Review Dijkstra's algorithm from class. A copy of the high-level algorithm is at the end of this document. Write a shortest paths function that takes a graph and returns a parent map:

```
map<int,int> shortestpaths(int start,
                          const map<int,list<int> > & graph);
```

Notice that this is slightly different than the more general version written in class (and shown below). The version below accepts a weighted graph. In our case, we'll be running Dijkstra's on a graph where all the weights are implicitly 1:

```
map<int, list<pair<int, int> > > versus map<int, list<int> >
```

String processing

As part of our next assignment, you will be doing some string processing when reading in the movie data file. Search for the `istringstream` class in the web page from our C++ reference link on the course web page. `istringstream`s allow us to do processing of strings without having to do character-level processing. Look at the constructor. You can create a new `istringstream` as follows:

```
string movie_line = "32:197,4;615,4;680,1;";
istringstream in(movie_line, istringstream::in);
```

The first parameter is the `string` we want to process and the second tells it that we're going to be reading from this `string`.

Now, look at the `operator>>` method of `istringstream`. Notice that there are *many* overloaded versions of this operator. How this benefits us is that depending on what is on the right hand side of the `>>` operator, the stream will read as many characters as possible that fit the type of the variable. For example,

```
int num;
in >> num;
```

would result in `num` containing "32". If we then did:

```
char c;
in >> c;
in >> num;
```

What are the values of `c` and `num`?

Once you're comfortable with this, write a method:

```
pair<int, list<pair<int,int>>> >> parse_line(string line)
```

That takes a line formatted like our movie review file and returns a `pair` consisting of the reviewer id and a list containing the pairs of movie id and movie review.

Like other streams we've seen, you can check for when the `istringstream` is at the end of the `string` using `in.eof()`.

```

/*
 * Dijkstra's single-source shortest path algorithm,
 * as discussed in class on April 28.
 *
 * Arguments: a starting vertex
 *            a weighted graph presented as an adjacency map
 *
 * Result: a map of parents in a tree of shortest paths
 *
 * Rett Bull
 * April 28, 2009
 * Modified by Dave 4/23/2010
 */
map<int,int> shortest_paths(int start,
                           const map<int,list<pair<int,int> > > & graph) {
    map<int,int> parents;
    priorityqueue62 frontier;

    parents[start]=start;
    frontier.push(start, 0);

    while (!frontier.is_empty()) {

        int v = frontier.top_serialnumber();
        int p = frontier.top_priority();
        frontier.pop();

        for (the neighbors (n,w) of v)
            if (n == parents[v])
                ; // do nothing
            else if (n is not in the frontier and has not been visited) {
                parents[n] = v;
                frontier.push(n, p + w);
            } else if (p + w < frontier.get_priority(n)) {
                parents[n] = v;
                frontier.reduce_priority(n, p + w);
            }

        } // end while

    return parents;
}

```