Lab 5 - *ThesaurusLab*

Due: *Mon, Mar 3rd, 2014 @ 11:59 pm*

Parallel and Sequential Data Structures and Algorithms 15-210 (Spring '14)

1 Introduction

In this assignment, you will implement an interface for finding shortest paths in unweighted graphs. Shortest paths are used all over the place, from finding the best route to the nearest Starbucks to numerous less-obvious applications. You will then apply your solution to the thesaurus problem, which is: given any two words, find all of the shortest synonym paths between them in a given thesaurus. Some of these paths are quite unexpected! :-)

2 Files

After downloading the assignment tarball from Autolab, extract the files by running:

```
tar -xvf thesauruslab-handout.tgz
```

from a terminal window. Some of the files worth looking at are listed below. You should only modify the files denoted by *, as these will be the only ones handed in by the submission script.

1. `Makefile`
2. * `MkAllShortestPaths.sml`
3. * `MkThesaurusASP.sml`
4. * `Tests.sml`

Additionally, you should create a file called:

```
written.pdf
```

which contains the answers to the written parts of the assignment.

3 Submission

To submit your assignment to Autolab, open a terminal, `cd` to the `thesauruslab` folder, and run:

```
make
```

Alternatively, run `make package`, open the Autolab webpage and submit the `handin.tgz` file via the "*Handin your work*" link.

4 Unweighted Shortest Paths

The first part of this assignment is to implement a general-purpose interface for finding shortest paths. This will be applied to the thesaurus path problem in the next section, but could also be applied to other problems. Your interface should work on directed graphs; to represent an undirected graph you can just include an edge in each direction. Your task in this assignment is to implement the interface given in `support/ALL_SHORTEST_PATHS.sig`. Before you begin, carefully read through the specifications for each function there.

4.1 Graph Construction

For these tasks, you may assume that you will be working with *directed*, *simple*, *connected* graphs (*directed, simple*: no self-loops and no more than one directed edge in each direction between any two vertices; *connected*: each graph is comprised of a single component). You may also assume that the graph has at least 1 node.

Task 4.1 (5%). In `MkAllShortestPaths.sml`, implement the function

```
val makeGraph : edge seq -> graph
```

which generates a graph based on an input sequence E of directed edges. The number of vertices in the resulting graph is equal to the number of vertex labels in the edge sequence. For full credit, `makeGraph` must have $O(|E|\log|E|)$ work and $O(\log^2|E|)$ span.

Note that you need to define the type `graph` that would allow you to implement the functions within the required cost bounds.

4.2 Graph Analysis

Task 4.2 (6%). In `MkAllShortestPaths.sml`, implement the functions

```
val numEdges : graph -> int
val numVertices : graph -> int
```

which return the number of directed edges and the number of unique vertices in the graph, respectively.

Task 4.3 (5%). In `MkAllShortestPaths.sml`, implement the function

```
val outNeighbors : graph -> vertex -> vertex seq
```

which returns a sequence V_{out} containing all out neighbors of the input vertex. In other words, given a graph $G = (V, E)$, `outNeighbors` $G\ v = \{w : (v, w) \in E\}$. If the input vertex is not in the graph, `outNeighbors` returns an empty sequence.

For full credit, `outNeighbors` must have $O(|V_{\text{out}}| + \log|V|)$ work and $O(\log|V|)$ span.

4.3 All Shortest Paths Preprocessing

Task 4.4 (20%). In `MkAllShortestPaths.sml`, implement the function

```
val makeASP : graph -> vertex -> asp
```

where `makeASP G v` generates an `asp` A, which contains information about all of the shortest paths from the input vertex v to all other reachable vertices. If v is not in the graph, then A should be empty.

For full credit, `makeASP` must have $O(|E| \log |V|)$ work and $O(D \log^2 |V|)$ span, where D is the *longest shortest path* (i.e. the shortest distance to the vertex that is the farthest from v).

Note that you need to define the type `asp` that would allow you to implement the functions within the required cost bounds.

4.4 All Shortest Paths Reporting

Task 4.5 (14%). In `MkAllShortestPaths.sml`, implement the function

```
val report : asp -> vertex -> vertex seq seq
```

where `report A u` evaluates to a `vertex seq seq` which contains all shortest paths from u to v (the input vertex of `makeASP`), represented as a sequence of paths (each path is a sequence of vertices). If no such path exists, the function evaluates to the empty sequence.

For full credit, `report` must have $O(|P||L| \log |V|)$ work and span, where V is the set of vertices in the graph, P is the number of shortest paths from u to v, and L is the length of the shortest path.

4.5 Testing

You do not have to submit test cases for this part of the lab, but as usual it is always a good idea to thoroughly test your code locally before submitting! To test your code:

```
$ smlnj
Standard ML of New Jersey v110.xx
- CM.make "sources.cm";
...
- Tester.testNumEdges ();
...
- Tester.testNumVertices ();
...
- Tester.testOutNeighbors ();
...
- Tester.testReport (); ...
```

5 Thesaurus Paths

Now that you have a working implementation for finding all shortest paths from a vertex in an unweighted graph, you will use it to solve the Thesaurus problem. You will implement `THESAURUS` in the functor `ThesaurusASP` in `ThesaurusASP.sml`. We have provided you with some utility functions to read and parse from input thesaurus files in `ThesaurusUtils.sml`.

5.1 Thesaurus Construction

Task 5.1 (5%). In `MkThesaurusASP.sml`, implement the function

```
val make : (string * string seq) seq -> thesaurus
```

which generates a thesaurus given an input sequence of pairs (w, S) such that each word w is paired with its sequence of synonyms S. You must define the type `thesaurus` yourself.

5.2 Thesaurus Lookup

Task 5.2 (4%). In `MkThesaurusASP.sml`, implement the functions

```
val numWords : thesaurus -> int
val synonyms : thesaurus -> string -> string seq
```

where `numWords T` counts the number of distinct words in T, and `synonyms T s` evaluates to a sequence containing the synonyms of s in T if $s \in T$, and an empty sequence otherwise.

5.3 Thesaurus All Shortest Paths

Task 5.3 (8%). In `MkThesaurusASP.sml`, implement the function

```
val query : thesaurus -> string -> string -> string seq seq
```

such that `query T x y` returns all shortest paths from x to y as a sequence of strings $\langle x, \ldots, y \rangle$, i.e. x first and y last. If no such path exists, `query` returns the empty sequence.

For full credit, **your implementation must be *staged*.** For example:

```
val earthlyConnection = query thesaurus "EARTHLY"
```

should generate the `thesaurus` value with cost proportional to `makeASP`, and then

```
val poisonousPaths = earthlyConnection "POISON"
```

should find the paths with cost proportional to `report`.

Invariably, a good number of students each semester fail to stage `query` appropriately. Don't let this happen to you! Staging is not automatic in SML; ask your TA if you are unsure about SML evaluation semantics or how a properly staged function is implemented.

5.4 Testing

Task 5.4 (3%). Add test cases in `Tests.sml` to test your `THESAURUS` implementation. As in the previous section, run the following in the SMLNJ REPL:

```
- Tester.testNumWords ();
- Tester.testSynonyms ();
- Tester.testQuery ();
```

Note that `testQuery` merely prints out the shortest paths your code produces for the given test cases (we do not test against a reference solution, as our naïve implementation is too slow!). Ask on Piazza to see if you are producing correct results.

The thesaurus used is defined in `input/thesaurus.txt` where each line is an entry associating the first word in the line with the rest of the words in the line.

As reference, our implementation returned only find one path of length 10 from "CLEAR" to "VAGUE" as well as from "LOGICAL" to "ILLOGICAL". However, there are two length 8 paths from "GOOD" to "BAD".

We wouldn't try "EARTHLY" to "POISON" if we were you :-).

6 Fire! Fire! Fire!

While you were busy analyzing thesaurus paths and surviving your first 15–210 exam (**congrats** by the way!), the City of Pittsburgh saw a power struggle in its Bureau of Fire. As it turns out, you've been chosen by the people to be the new *Chief of Department* of the Bureau.

During your first day of training, you're briefed on some technical terms (yes... the Chief needs to know all the in-and-outs of the City, but let's just say you're on a *need-to-know* basis). You're told that Pittsburgh is modeled as a set of roads and a set of intersections where the roads meet. Of the intersections, the fire station is a special one.

Every road in Pittsburgh takes 1 minute to travel through. We say that Pittsburgh is *robust* if when one road becomes blocked by traffic, the time to get to any intersection from the fire station increases by *at most* 1 minute. Since you, the Chief of Department, are filled with Tartan pride, you immediately brainstorm ways to algorithmically determine if Pittsburgh is *robust*.

Let G be the city road graph of Pittsburgh, where intersections are vertices, and roads are edges. There are n intersections and m roads in Pittsburgh, where $n < m$. Let SP be a single-source shortest path algorithm that finds the shortest paths from a given source in G to all reachable vertices, and let $W_{SP}(G)$ and $S_{SP}(G)$ denote its work and span respectively. Note that $W_{SP}(G)$ is lower-bounded by $\Omega(n+m)$.

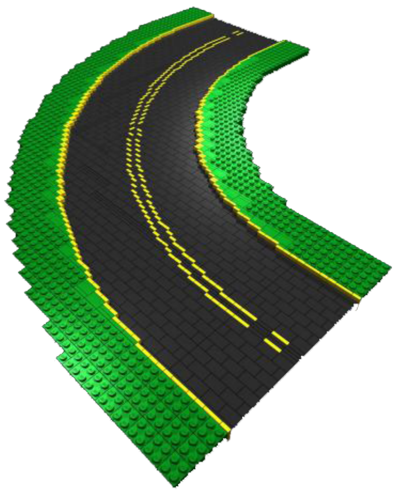
For each part in this question, please give pseudocode, NOT prose. This encourages you to think more critically about the algorithm, and makes your solutions less hand-wavey! Chapters 10–12 of the lecture notes provide examples of graph algorithms using the type of pseudocode we expect. Please also **state the graph representation** you'll be using; this will affect the costs of the various operations you'll be doing in your algorithms.

Task 6.1 (5%). Give a simple algorithm to determine if Pittsburgh is *robust*, with work $\Theta(mW_{SP}(G))$ and span $\Theta(S_{SP}(G))$. Also state what standard implementation of a single-source shortest path algorithm SP could be.

Task 6.2 (8%). The Mayor of Pittsburgh (who happens to be as mathematically-inclined as you), points out that there might be a better way to solve this problem, since he wants to be able to check on Pittsburgh's robustness more quickly. Impress him on your first day in Office by improving your algorithm above to have work $\Theta(nW_{SP}(G))$ without increasing its span (note the tight Θ bound given).

Task 6.3 (10%). Mr. Mayor is a hard man to please. He says, "That's pretty darn fast now, but I'm thinking of adding a bunch more roads and intersections to Oakland by 2017, so wouldn't it be awesome if we could solve this problem *regardless of how many roads we have*?" Slightly frustrated, but intrigued at this prospect, your task is now to improve your algorithm yet again. To meet his demands, your algorithm needs to have asymptotically better work than the one above. What is the resulting work of your improved algorithm?

Task 6.4 (7%). The Mayor is now happy, and gives you a raise, as well as your next task. He wants to make sure that in the event of a fire emergency in town, the brave men of your department are able to speed their fire trucks through to the destination as quickly as possible, avoiding the effects of heavy traffic (of the I-376 and I-279 for example).



So he asks you to suggest new roads that he should instruct the *Department of Public Works* to build in order to solve this problem. Armed with the city road map G and your speedy new *robust*-ness algorithm, you set your mind to drafting up a strategy for him. You need to ensure that when a road segment is blocked, it takes your fire trucks (which start out at the fire station of course) *at most* $d + 1$ minutes to reach a vertex that was reachable in d minutes before the blockage occured (that is, minimizing the increase in time to get there). Keep in mind that each road takes exactly 1 minute for your fire trucks to drive through.

Remember that Pittsburgh roads are *ever so narrow*, so you can only add at most 2 roads to each intersection. The Department of Public Works has limited resources, so they tell you that no two roads should share the same intersection endpoints.

What do you tell the Mayor? *Oh and by the way,* ***thank you*** *for your service to the City :)*.