Lab 5 - *ThesaurusLab*

Due: *Mon, Oct 13th, 2014 @ 11:59 pm*

Parallel and Sequential Data Structures and Algorithms   15-210 (Fall '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 v (the input vertex of `makeASP`) to u, 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 v to u, 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!). Feel free to share your paths with each other on Piazza to gain confidence in your own solution!

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!

Introduction: Congratulations! You've been appointed *Chief Analyst* for the Pittsburgh Bureau of Fire. The Mayor has assigned your first task: determine if Pittsburgh's road system is *robust*. Specifically, he wants to make sure that if any one road becomes unusable, the time it takes for fire trucks to reach any particular destination will increase by at most 1 minute.

After sifting through a multitude of highly inaccurate information compiled by your intern, you've decided to model Pittsburgh as an undirected graph where roads are edges and intersections are vertices. And of course, every road requires exactly 1 minute of travel time. The intern also gives you a little program he wrote for finding shortest paths. Unfortunely, he wrote it as practice for a code obfuscation competition – you'll simply have to assume it's correct. You set out on designing an algorithm for computing robustness.

The Problem: Let G be a connected, undirected, unweighted graph with n vertices and m edges. One vertex is marked as the fire station. We say that G is *robust* if, when any one edge is removed, each shortest path beginning from the fire station increases by at most 1.

You are given an algorithm SP which solves the single-source shortest path problem for unweighted graphs. SP returns a mapping of vertices to their shortest paths (which you may assume are sequences of vertices, if you'd like). SP run on a graph G from a particular source vertex has work $W_{\text{SP}}(G)$ and span $S_{\text{SP}}(G)$.

The exact representation of the graph G is flexible.

Task 6.1 (30%). Describe an algorithm for determining robustness on a graph. Your algorithm should use SP internally, and should have $\Theta(S_{\text{SP}}(G) + \log n)$ span. Depending on work of your algorithm, you will receive varying levels of credit:

- An algorithm which has $\Theta(m(W_{\text{SP}}(G) + n))$ work will receive at most 8 points.
- An algorithm which has $\Theta(n(W_{\text{SP}}(G) + n))$ work will receive at most 15 points.
- An algorithm which has $\Theta(W_{\text{SP}}(G) + m)$ work will be eligible for full credit.

Some additional notes:

- You will be graded both on clarity and brevity. Answers exceeding 350 words will likely lose points.
- Remember to explicitly mention what computation is happening in parallel.
- We expect an explanation in prose – please avoid using code. Include implementation details only if it is necessary to justify the cost bounds.