

Advanced Algorithms

Assignment 2

1 Graph Representation and Shortest Path Algorithms

In Lecture 8, we discussed three different graph representations:

- adjacency matrix
- adjacency list
- edge list

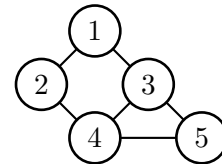
1.1 Implementation

Define an interface to load and modify an undirected (potentially weighted¹) graph $G = (V, E)$. Implement the interface for all three memory representations. The input will be a file with a list of edges as follows (adapted from the 9th DIMACS implementation challenge²): the first line is formatted as `p sp N M`, where N denotes the number of nodes and M denotes the number of edges. The set of nodes is $V = \{1, 2, \dots, N\}$. The next M lines are of the form `a N1 N2 W`, where $N1$ and $N2$ are the first and the second node of the edge, respectively, and W is the edge weight (from \mathbb{N}^+).

Example:

```
p sp 5 6
a 1 2 1
a 1 3 1
a 2 4 1
a 3 4 1
a 3 5 1
a 4 5 1
```

specifies the unweighted graph



1.2 Evaluation

Compare the running times and memory requirements for the different representations with respect to different graph sizes. Operations you may consider include (but are not limited to)

- loading a graph (you may download graph files from the Internet and/or generate your own graph files)
- querying the weight of a specific edge / listing all the edges of a given node
- adding / removing an edge
- adding / removing a vertex

¹You may assume that edge weights are positive integers.

²<http://www.dis.uniroma1.it/~challenge9/format.shtml#graph>

2 All Pairs Shortest Paths Algorithms

For the following questions, the output of your program shall be a $N \times N$ distance matrix. Outputting shortest paths is *optional*.

2.1 Using a Single Source Shortest Path Algorithm

One way to compute graph distances between all pairs of nodes is to run a single source shortest path algorithm for each and every node of the graph.

2.1.1 Implementation

Implement Breadth-First-Search and Dijkstra's algorithm.

2.1.2 Evaluation

Evaluate your implementations using graphs of different sizes. Optional (you will earn brownie points): You may want to compare the performance of different priority queues.

2.2 Floyd-Warshall Algorithm

Implement and evaluate the Floyd-Warshall algorithm for graphs of different sizes. Does it help to store the adjacency matrix of the graph in row-major order *and* in column-major order? Why? Why not?

2.3 Seidel's Algorithm

We discussed Seidel's algorithm³ to compute shortest paths between all pairs of nodes.

2.3.1 Implementation

Implement Seidel's algorithm using integer matrix multiplication. You may implement matrix multiplication (optional: fast matrix multiplication) yourself but you may also use a library (optional: different libraries). Note: the part involving the computation of actual paths is optional.

2.3.2 Evaluation

Evaluate your implementation(s) using graphs of different sizes. Seidel's algorithm heavily uses integer matrix multiplication as a sub-routine. Again: does it help to store some of the matrices in row-major order *and* in column-major order?

2.4 Overall Comparison

Compare the performance of the three algorithms.

³See Chapter 10.1 of *Randomized Algorithms* by Rajeev Motwani and Prabhakar Raghavan. The reference to the original paper is: Raimund Seidel: On the All-Pairs-Shortest-Path Problem in Unweighted Undirected Graphs. *J. Comput. Syst. Sci.* 51(3): 400–403 (1995)

Submission

Assignment 2 consists of two parts. Solve *both* of them and submit your solution by email.

Format

Your solution shall be packed as a `zip` file named *your student number-your last name.zip*. Send your file as an attachment of an email to `csommer@gmail.com`. Make sure that you receive a confirmation message from me by the end of August. The contents of your solution are supposed to be the following:

1. a `txt` file named `README` (or `readme.txt`) with instructions on how to compile and run your code
2. a `pdf` file named *your student number-your last name.pdf* describing in sufficient detail
 - (a) your implementation(s)
 - (b) your experimental evaluation(s)
3. the code you wrote (in a programming language of your choice — if this happens to be a very exotic one please reconfirm by email first)
4. the libraries you used (hyperlinks to files on the web if they happen to be large)
5. the graphs you used for evaluation (hyperlinks to files on the web if they happen to be large)

Date

The final submission deadline is 27 August 2010, 11:59 pm, Japanese time. Earlier submission is highly encouraged.

Collaboration

You may work in teams of up to 3 students (it is of course OK to work alone). Each student must submit an individual report stating (in sufficient detail) individual contributions (with respect to implementation, evaluation, and writing). The description of the implementation(s) and evaluation(s) may be the same for all group members. Your submission will be evaluated with higher expectations if you choose to work in a group.