Course Announcements

- Maps UI due Friday.
Bacon Review

John Jannotti

/course/cs0320/www/docs/lectures/

Apr 10, 2018
Nice things we looked for

- +2 known correct implementation to test against
- Batches SQL selects to load movies with actors
- generic on Node, Edge type
- +2 clear separation of DB logic and Actor/Movie logic
- Uses the proxy pattern
- Does not rebuild the graph for new queries
- Moved URL handlers into their own files/package
- Avoids reading the same node/edge from the db twice
- Actor/Movie does not contain dijkstra info ("visited flag")
- Graph Nodes/Edges do not contain dijkstra info
- .equals() and .hashCode() on IDs for movie/actor
- Creates a "Path" class or other non-obvious useful abstractions
- Implements toString for Actor/Movie/Path
- Customized autocorrect for Bacon
- Autocorrect Maintains capitalization in Actor/Movie names
Some bad things we looked for

- Builds SQL query "by hand", (using "+" to concat vars) instead of the "?" syntax.
- Does the above for names of actors/movies. This would be really bad for names with apostrophes in them and make them prone to SQL injections.
Today’s inspiration

My dog, Dijkstra.
Performance

- When your code is slow, Profile!
- The JDK comes with a (mediocre) profiler, visualvm

<table>
<thead>
<tr>
<th>Hot Spots – Method</th>
<th>Self time [ % ]</th>
<th>Self time</th>
<th>Self time (CPU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.sqlite.core.NativeDB.prepare[native]</td>
<td></td>
<td>8,389 ... (34.5%)</td>
<td>8,389 ms</td>
</tr>
<tr>
<td>org.sqlite.core.NativeDB.step[native] ()</td>
<td></td>
<td>5,530 ... (22.7%)</td>
<td>5,530 ms</td>
</tr>
<tr>
<td>org.sqlite.core.NativeDB._open[native] ()</td>
<td></td>
<td>2,643 ... (10.9%)</td>
<td>2,643 ms</td>
</tr>
<tr>
<td>org.sqlite.core.NativeDB._close[native] ()</td>
<td></td>
<td>2,210 ... (9.1%)</td>
<td>2,210 ms</td>
</tr>
<tr>
<td>org.sqlite.core.CoreConnection.open ()</td>
<td></td>
<td>1,490 ... (6.1%)</td>
<td>1,490 ms</td>
</tr>
<tr>
<td>org.sqlite.core.DB.newSQLException ()</td>
<td></td>
<td>923 ms (3.8%)</td>
<td>923 ms</td>
</tr>
<tr>
<td>edu.brown.cs.jj.orm.Db.getConnection ()</td>
<td></td>
<td>806 ms (3.3%)</td>
<td>806 ms</td>
</tr>
<tr>
<td>org.sqlite.core.NativeDB.free_functions[]</td>
<td></td>
<td>405 ms (1.7%)</td>
<td>405 ms</td>
</tr>
<tr>
<td>org.sqlite.core.NativeDB._exec[native] ()</td>
<td></td>
<td>404 ms (1.7%)</td>
<td>404 ms</td>
</tr>
<tr>
<td>org.sqlite.core.CoreStatement.&lt;init&gt; ()</td>
<td></td>
<td>303 ms (1.2%)</td>
<td>303 ms</td>
</tr>
</tbody>
</table>

Lots of time in opening/closing connections, preparing statements.
Two small optimizations

- Use a single connection (per thread!)
- Reuse prepared statements.

Now finally, there’s a problem in “business logic” (getFirstInitial)
No point in splitting

- We need the first initial quite often.
- Don’t split the name into an array.

Quite fast, very few changes.
Concurrent issues for Maps

- Your Dijkstra “bookkeeping” needs to exist outside the nodes.
- To use you use a single connection, consider `ThreadLocal<T>`
It’s great to get real types out of Dijkstra.
You’ll need mutually recursive types. (Nodes and Edges)

```java
public interface Vertex<V extends Vertex<V, E>,
                        E extends Edge<V, E>> {
    Collection<E> getEdges();
}

public interface Edge<V extends Vertex<V, E>,
                      E extends Edge<V, E>> {
    V getSource();
    V getDestination();
    double getWeight();
}
```