CS314 Software Engineering Sprint 3

Dave Matthews

Colorado State University

Sprint 3 Summary

- Add Level 2 and 3 software engineering processes/tools
  - Clean Code, Coverage, White Box Testing, Code Climate
- Learn some additional technologies
  - SQL (MariaDB)
  - Traveling Salesman Problem
- Add features
  - Produce shorter trips
  - Build trips from existing information

Internship Plan

Sprint	Processes	Tools	Technology	TripCo Epics
1	• Configuration Management • Project Management • Scrum, Planning Poker	• GitHub, ZenHub • CodePen • Unix	• Bootstrap 4 • HTML • JavaScript • ReactJS	• Make a mobile resume • Calculate geographic distances
2	• Continuous Integration • Test Driven Development • Black Box Testing	• Maven, Travis-CI • Webpack, Node.js • JUnit, IntelliJ	• Java Spark • REST API/HTTP • JSON, SVG	• Accept destination file • Show map and itinerary
3	• Clean Code • Code Coverage • White Box Testing	• Code Climate • Jacoco, …	• SQL • MariaDB	• Plan shorter trips • Modify destinations • Show information
4	• Code Smells • Refactoring		• KML	• Plan shorter trips • Add more information • Map operations
5	• Peer Reviews • Inspections • Metrics		• Concurrency	• Plan shorter trips • Plan trips faster • Finalize your resume

TFFI - version 3

- find destinations - /search
- optimization - /config, /trip

TFFI - find places

```
"type":      "search",
"version":   3,
"match":     "string",
"limit":     0,
"places":    []
```

- Places are the same format as trip objects.

SQL

```
MariaDB [cs314]> select id,name,municipality,type,latitude,longitude from airports limit 20;
+------+-----------------------------------------+-----------------+---------------+-------------------+---------------------+
| id   | name                                    | municipality    | type          | latitude          | longitude           |
+------+-----------------------------------------+-----------------+---------------+-------------------+---------------------+
| KBJC | Rocky Mountain Metropolitan Airport     | Denver          | medium_airport | 39.90879822       | -105.1169968        |
| KBKF | Buckley Air Force Base                  | Aurora          | medium_airport | 39.701698303200004 | -104.751998901      |
| KCOS | City of Colorado Springs Municipal Airport | Colorado Springs | large_airport | 38.805801391602   | -104.70099639893    |
| KDEN | Denver International Airport            | Denver          | large_airport  | 39.861698150635   | -104.672996521      |
| KEGE | Eagle County Regional Airport           | Eagle           | medium_airport | 39.64260101       | -106.9179993        |
| KGJT | Grand Junction Regional Airport         | Grand Junction  | medium_airport | 39.1223983765     | -108.527000427      |
| KPUB | Pueblo Memorial Airport                 | Pueblo          | small_airport  | 38.289100646972656 | -104.49700164794922 |
| 00CO | Cass Field                              | Briggsdale      | small_airport  | 40.62220001220703 | -104.34400177001953 |
| 01CO | St Vincent General Hospital Heliport    | Leadville       | heliport       | 39.24530029296875 | -106.24600219726562 |
| 02CO | Mc Cullough Airport                     | Monte Vista     | small_airport  | 37.64329910279999 | -106.04699707       |
| 03CO | Kugel-Strong Airport                    | Platteville     | small_airport  | 40.2125015259     | -104.744003296      |
| 04V  | Saguache Municipal Airport              | Saguache        | small_airport  | 38.0990833        | -106.1743889        |
| 05CO | Rancho De Aereo Airport                 | Mead            | small_airport  | 40.2149839        | -104.9844228        |
| 05V  | Blanca Airport                          | Blanca          | small_airport  | 37.41109848022461 | -105.552001953125   |
| 06CO | Jecan Airport                           | Branson         | small_airport  | 37.38750076293945 | -103.69100189208984 |
| 07CO | Comanche Creek Airport                  | Kiowa           | small_airport  | 39.26359939575195 | -104.427001953125   |
| 08CO | Terra Firma Airport                     | Rush            | small_airport  | 38.73249816894531 | -104.04100036621094 |
| 09CO | Cottonwood Field                        | Swink           | small_airport  | 38.055599212646484 | -103.65299987792969 |
| 0CD0 | Delta County Memorial Hospital Heliport | Delta           | heliport       | 38.7407989502     | -108.052001953      |
| 0CD1 | Colorado Plains Medical Center Heliport | Fort Morgan     | heliport       | 40.2610917        | -103.7963389        |
+------+-----------------------------------------+-----------------+---------------+-------------------+---------------------+
20 rows in set (0.00 sec)

MariaDB [cs314]>
```

SQL

```
# connect to the database from a shell using your eID
mysql –D cs314 –h faure --user=cs314–db --password=eiK5liet1uej
# show a list of tables
show tables;
# show the columns in a table
show columns from airports;
# count the number of records in the table
select count(*) from airports;
# show the first 5 entries in the airports table
select * from airports limit 5;
# show selected columns
select id,name,municipality from airports limit 20;
# show types
select distinct(type) from airports;
# show municipalities sorted
select distinct(municipality) from airports order by municipality;
```


SQL

```
# show all of the heliports
select name from airports where type = 'heliport';
# show all of the airports (large, medium, small)
select name from airports where type like '%airport%';

# show all records that refer to denver sorted by name
select id,name,municipality,type from airports where name like
'%denver%' or municipality like '%denver%' order by name;

# select airports by ids
select id,name,municipality,type from airports where id in
('19CO','26CO','77CO','CO23','CO24','K00V','KFNL','KDEN');
```

```
  // db configuration information
  private final static String myDriver   = "com.mysql.jdbc.Driver";
  private final static String myUrl      = "jdbc:mysql://faure.cs.colostate.edu/cs314";
  private final static String user="cs314-db";
  private final static String pass="eiK5liet1uej";
  // fill in SQL queries to count the number of records and to retrieve the data
  private final static String count      = "";
  private final static String search     = "";
  // Arguments contain the username and password for the database
  public static void main(String[] args){
    try  {
      Class.forName(myDriver);
      // connect to the database and query
      try (Connection conn = DriverManager.getConnection(myUrl, user, pass);
           Statement stCount = conn.createStatement();
           Statement stQuery = conn.createStatement();
           ResultSet rsCount = stCount.executeQuery(count);
           ResultSet rsQuery = stQuery.executeQuery(search)
      ) {
        printJSON(rsCount, rsQuery);
      }
    } catch (Exception e) {
      System.err.println("Exception: "+e.getMessage());
    }
```



```
  private static void printJSON(ResultSet count, ResultSet query) throws SQLException {
  System.out.printf("\n{\n");
  System.out.printf("\"type\": \"find\",\n");
  System.out.printf("\"title\": \"%s\",\n",search);
  System.out.printf("\"places\": [\n");
  // determine the number of results that match the query
  count.next();
  int results = count.getInt(1);
  // iterate through query results and print out the airport codes
  while (query.next()) {
    System.out.printf("  \"%s\"", query.getString("code"));
    if (--results == 0)
      System.out.printf("\n");
    else
      System.out.printf(",\n");
  }
  System.out.printf("  ]\n}\n");
}
```

TFFI - Optimization

- **"none", "short", "shorter", or "shortest"**
- **/config provides the supported optimization levels.**
- **/trip specifies the optimization level as an option.**

Traveling Salesman Problem

- Find the shortest Hamiltonian cycle in a graph.
  - O(n!)
  - heuristic algorithms gain speed at cost of tour quality
  - construction + improvement
- Construction
  - Nearest Neighbor
- Improvement
  - 2 opt
  - 3 opt

https://en.wikipedia.org/wiki/Nearest_neighbour_algorithm

Nearest Neighbor

- Does the answer change if I select a different starting city?

```
nearestNeighbor(cities) {
  1. Select a random city.
  2. Find the nearest unvisited city and go there.
  3. Are then any unvisited cities left?  If yes, repeat step 2.
  4. Return to the first city.
}
```


https://web.tuke.sk/fei-cit/butka/hop/htsp.pdf

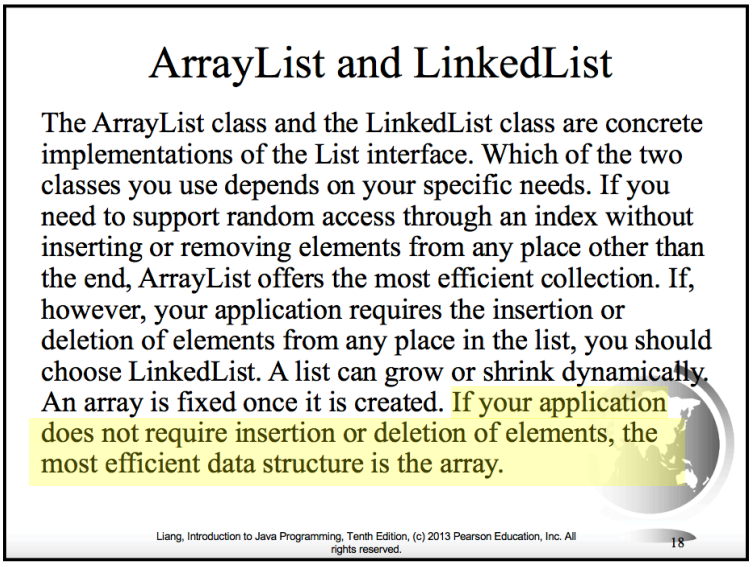
2-opt (from Wikipedia) - very slow

```
repeat until no improvement is made {
      start_again:
      best_distance = calculateTotalDistance(existing_route)
      for (i = 0; i < number of nodes eligible to be swapped – 1; i++) {
          for (k = i + 1; k < number of nodes eligible to be swapped; k++) {
              new_route = 2optSwap(existing_route, i, k)
              new_distance = calculateTotalDistance(new_route)
                  if (new_distance < best_distance) {
                      existing_route = new_route
                      goto start_again
                  }
          }
      }
}

2optSwap(route, i, k) {
     1. take route[1] to route[i-1] and add them in order to new_route
     2. take route[i] to route[k] and add them in reverse order to new_route
     3. take route[k+1] to end and add them in order to new_route
     return new_route;
}
```


https://en.wikipedia.org/wiki/2-opt

"ill-advised data structure use"



ArrayList and LinkedList

The ArrayList class and the LinkedList class are concrete implementations of the List interface. Which of the two classes you use depends on your specific needs. If you need to support random access through an index without inserting or removing elements from any place other than the end, ArrayList offers the most efficient collection. If, however, your application requires the insertion or deletion of elements from any place in the list, you should choose LinkedList. A list can grow or shrink dynamically. An array is fixed once it is created. If your application does not require insertion or deletion of elements, the most efficient data structure is the array.

18

2-opt (improved)

```
2optReverse(route, i1, k) { // reverse in place
    while(i1 < k) {
        temp = route[i1]
        route[i1] = route[k]
        route[k] = temp
        i1++; k--
    }
}

improvement = true
while improvement {
    improvement = false
    for (i = 0; i <= n-3; i++) {  // assert n>4
        for (k = i + 2; k <= n-1; k++) {
            delta = -dis(route,i,i+1)-dis(route,k,k+1)+dis(route,i,k)+dis(route,i+1,k+1)
            if (delta < 0) { //improvement?
                2optReverse(route, i+1, k)
                improvement = true
            }
        }
    }
}
```

2-opt inversions

route	23	15	3	9	7	…	21	11	5	4	23
index	0	1	2	3		…		n-3	n-2	n-1	n

i	i+1	k	k+1								
								i	i+1	k	k+1
		i	i+1	-	…	-		k	k+1		
i	i+1	-	-	-	…	-		-	-	k	k+1

0 <= i < i+1 < k < k+1 <=n