Submission

5206267	Luo, Zheng	3785/4 AEROAH
Submissions:-		
S 0	Tue Mar 30 11:25:05 2021	5206267 mon15b ass1 -4:-5
S 0+	Tue Mar 30 11:25:05 2021	5206267 mon15b ass1 -4:-6
Mon Apr	19 04:53:11 2021	## weber.orchestra.cse.unsw.EDU.AU ##

Listing

AVLTree.c

```
1 // Assignment 1 21T1 COMP2521: ADTs: FlightDb Using a Generic AVL Tree
 2 //
 3 // This program was written by Zheng Luo (z5206267@ad.unsw.edu.au)
 4 // on March/2021
 5
 6 #include <stdbool.h>
 7 #include <stdio.h>
 8 #include <stdlib.h>
9
10 #include "List.h"
11 #include "Record.h"
12 #include "AVLTree.h"
13
14 typedef struct node *Node;
15 struct node {
      Record rec;
16
      Node left;
17
18
      Node right;
19
      int
            height;
20 };
21
22 struct tree {
23
      Node
             root;
24
      int
             (*compare)(Record, Record);
25 };
26
// Auxiliary functions
28
29
30 static void doTreeFree(Node n, bool freeRecords);
31 static Node newNode(Record rec);
32 static Record doTreeSearch(Tree t, Node n, Record rec);
33 Node doTreeInsert(Tree t, Node n, Record rec, bool *result);
34 void doTreeSearchBetween(Tree t, Node n, Record lower, Record upper, List 1);
35 Record doTreeNextSearch(Tree t, Node n, Record r, Record *desiredRecord);
36 int TreeHeight(Node n);
37 Node rotateLeft(Node n);
  Node rotateRight(Node n);
39
   40
   static Node newNode(Record rec) {
41
      Node n = malloc(sizeof(*n));
42
      if (n == NULL) {
43
          fprintf(stderr, "error: out of memory\n");
44
          exit(EXIT_FAILURE);
45
      }
46
47
48
      n->rec = rec;
      n->left = NULL;
      n->right = NULL;
50
      n->height = 0;
51
52
      return n;
53 }
54
56
57 Tree TreeNew(int (*compare)(Record, Record)) {
      Tree t = malloc(sizeof(*t));
58
59
      if (t == NULL) {
          fprintf(stderr, "error: out of memory\n");
60
          exit(EXIT_FAILURE);
61
62
      }
63
      t->root = NULL;
64
65
      t->compare = compare;
      return t;
66
67 }
68
   70
71 void TreeFree(Tree t, bool freeRecords) {
```

```
doTreeFree(t->root, freeRecords);
72
73
       free(t);
74 }
75
76
    static void doTreeFree(Node n, bool freeRecords) {
       if (n != NULL) {
77
78
           doTreeFree(n->left, freeRecords);
79
           doTreeFree(n->right, freeRecords);
           if (freeRecords) {
80
81
              RecordFree(n->rec);
82
           }
           free(n);
83
       }
84
85
   }
86
    87
88
    Record TreeSearch(Tree t, Record rec) {
89
90
       return doTreeSearch(t, t->root, rec);
91 }
92
    static Record doTreeSearch(Tree t, Node n, Record rec) {
93
94
       if (n == NULL) {
95
           return NULL;
96
       }
97
       int cmp = t->compare(rec, n->rec);
98
99
       if (cmp < 0) {
100
           return doTreeSearch(t, n->left, rec);
101
       } else if (cmp > 0) {
102
           return doTreeSearch(t, n->right, rec);
103
       } else {
           return n->rec;
104
105
106
107
108
   109
   /* IMPORTANT:
110
111
      Do NOT modify the code above this line.
      You must not modify the 'node' and 'tree' structures defined above.
112
      You must not modify the functions defined above.
113
114 */
116
118 /**
    * Inserts the given record into the AVL tree.
119
    * The time complexity of this function must be O(log n).
120
121
     * Returns true if the record was inserted successfully, or false if
     * there was already a record that compares equal to the given record in
122
     * the tree (according to the comparison function).
123
124
125 bool TreeInsert(Tree t, Record rec) {
126
       bool result = false;
127
       t->root = doTreeInsert(t, t->root, rec, &result);
128
       return result;
129
130
   Node doTreeInsert(Tree t, Node n, Record rec, bool *result) {
131
       // Ending condition when successful found the desired location.
132
       if (n == NULL) {
133
           *result = true;
134
           return newNode(rec);
135
136
       // If repeated number has been found in the tree.
137
       else if (t->compare(n->rec, rec) == 0) {
138
           *result = false;
139
140
       }
141
       else {
           // Find the location first.
142
```

```
if (t->compare(n->rec, rec) > 0) {
143
144
                n->left = doTreeInsert(t, n->left, rec, result);
145
            }
            else {
146
147
                n->right = doTreeInsert(t, n->right, rec, result);
148
            }
149
            // Check the balance of the tree,
150
            // and rotate if necessary.
            int LHeight = TreeHeight(n->left);
151
            int RHeight = TreeHeight(n->right);
152
153
            if (LHeight - RHeight > 1) {
154
155
                if (t->compare(n->left->rec, rec) < 0) {</pre>
                    return rotateLeft(n);
156
157
                }
158
                else {
159
                     return rotateRight(n);
                 }
160
161
            }
            else if (RHeight - LHeight > 1) {
162
                if (t->compare(n->right->rec, rec) > 0) {
163
                    return rotateRight(n);
164
                }
165
                else {
166
167
                    return rotateLeft(n);
168
169
            }
170
        }
171
        return n;
172
    }
173
174
    // This function computes the tree height,
    // return the height of the tree as integer.
175
    int TreeHeight(Node n) {
176
        if (n == NULL) {
177
178
            return -1;
179
        }
180
181
        int lh = TreeHeight(n->left);
182
        int rh = TreeHeight(n->right);
        return 1 + ((lh > rh) ? lh : rh);
183
184
185
    // This function rotates the tree towards left,
186
    // return the node after rotation.
187
188
    Node rotateLeft(Node n) {
189
        if (n == NULL || n->right == NULL) {
            return n;
190
191
        }
192
        Node n1 = n->right;
        n->right = n1->left;
193
194
        n1->left = n;
195
        return n1;
196 }
197 // This function rotates the tree towards right,
198 // return the node after rotation.
199 Node rotateRight(Node n) {
        if (n == NULL || n->left == NULL) {
200
201
            return n;
202
        }
203
        Node n2 = n \rightarrow left;
        n->left = n2->right;
204
205
        n2 - right = n;
206
        return n2;
207 }
208
209
211
212 /**
213 * Searches for all records between the two given records, inclusive
```

```
214
     * (according to the comparison function) and returns the records in a
215
     * list in order. Assumes that `lower` is less than `upper`.
     * The time complexity of this function must be O(\log n + m), where m is
216
     * the length of the returned list.
217
218
     */
219
    List TreeSearchBetween(Tree t, Record lower, Record upper) {
220
221
        List 1 = ListNew();
222
        doTreeSearchBetween(t, t->root, lower, upper, 1);
223
        return 1;
224 }
225
    void doTreeSearchBetween(Tree t, Node n,
226
                                    Record lower, Record upper, List 1) {
227
228
        // Ending condition
229
        if (n == NULL) {
230
            return;
231
232
        int lowerCmp = t->compare(n->rec, lower);
233
        int upperCmp = t->compare(n->rec, upper);
234
235
        // For sorted list: Inorder search must applied.
236
        // In order to visit minimum nodes:
        // No necessary to visit left subtree,
237
238
        // as left subtree will all smaller than current, which < lower,
239
        // so go to right if current is smaller than lower
240
        if (lowerCmp < 0) {</pre>
241
            doTreeSearchBetween(t, n->right, lower, upper, 1);
242
243
        // Go to left if current is larger than upper
244
        else if (upperCmp > 0) {
245
            doTreeSearchBetween(t, n->left, lower, upper, 1);
246
        }
        else {
247
248
            doTreeSearchBetween(t, n->left, lower, upper, 1);
249
            ListAppend(1, n->rec);
250
            doTreeSearchBetween(t, n->right, lower, upper, 1);
251
        }
252
        return;
253
254 }
255
256
    257
     * Returns a smallest record greater than or equal to the given record r
258
259
     * (according to the comparision function).
     * Time complexity of the function must be O(\log n).
260
     */
261
262
   Record TreeNext(Tree t, Record r) {
263
        Record desiredRecord = NULL;
264
        return doTreeNextSearch(t, t->root, r, &desiredRecord);
265 }
266
267
    Record doTreeNextSearch(Tree t, Node n, Record r, Record *desiredRecord) {
268
        // There is no exact match in the tree whatsoever.
        if (n == NULL && *desiredRecord != NULL) {
269
             return *desiredRecord;
270
271
        // This is designed for the situation of no flights
272
273
        // in the rest of the week,
        // Then the function will search for the upcoming weeks.
274
275
        // For example,
        // when searching the flight on late Sunday evening,
276
        // if there is no any flight availble in later Sunday,
277
278
        // then this function will start searching from Monday 0am.
        else if (n == NULL && *desiredRecord == NULL) {
279
280
             Record new = RecordNew(RecordGetFlightNumber(r),
281
             RecordGetDepartureAirport(r), RecordGetArrivalAirport(r), 0, 0, 0, 0);
282
283
             return doTreeNextSearch(t, t->root, new, desiredRecord);
284
```

```
285
        }
286
        int cmp = t->compare(r, n->rec);
287
288
289
        // The next possible available flight will be recorded as desiredRecord,
        // which will updated as the tree searching progress,
290
        // desired record will be returned after searching reached the leaf.
291
        if (cmp < 0) {
292
293
             *desiredRecord = n->rec;
294
            return doTreeNextSearch(t, n->left, r, desiredRecord);
295
        }
296
        else if (cmp > 0) {
297
             return doTreeNextSearch(t, n->right, r, desiredRecord);
298
        }
299
        else {
300
             return n->rec;
301
        }
302 }
```

FlightDb.c

```
1 // Assignment 1 21T1 COMP2521: ADTs: FlightDb Using a Generic AVL Tree
 2 //
 3 // This program was written by Zheng Luo (z5206267@ad.unsw.edu.au)
 4 // on March/2021
 5
   #include <stdio.h>
 7 #include <stdlib.h>
   #include <string.h>
 9
10 #include "List.h"
11 #include "FlightDb.h"
12 #include "AVLTree.h"
13
14 #define CMPFLIGHTNUM strcmp(RecordGetFlightNumber(r1), \
15 RecordGetFlightNumber(r2))
   #define CMPDEPAIRPORT strcmp(RecordGetDepartureAirport(r1), \
16
17 RecordGetDepartureAirport(r2))
18 #define CMPDAY RecordGetDepartureDay(r1) - RecordGetDepartureDay(r2)
19 #define CMPHOUR RecordGetDepartureHour(r1) - RecordGetDepartureHour(r2)
20 #define CMPMIN RecordGetDepartureMinute(r1) - RecordGetDepartureMinute(r2)
21 #define MINDAY 0
22 #define MAXDAY 6
23 #define MINHOURS 0
24 #define MAXHOURS 23
25 #define MINMINS 0
26 #define MAXMINS 59
27 #define MINDURATION 0
28 #define MAXDURATION 9999
   #define MINCHAR ""
29
   #define MAXCHAR "zzzzzzz"
31
   struct flightDb {
32
           Tree byFlightNumber;
33
           Tree byDepartureAirportAndDay;
34
35
           Tree byTimeRange;
           Tree byNext;
36
37 };
38
39
   // Comparison functions
40
41
42 // This function compares the records in the order of
43 // flight number, departure airport, day, hour and minute.
44 // This function will return positive integer if r1 is greater than r2,
   // return negative integer if r1 is smaller than r2,
46 // or return 0 if r1 is equal to r2.
    int compareByFlightNumber(Record r1, Record r2) {
47
           if (CMPFLIGHTNUM == 0) {
48
49
                   // Departure airport
                   if (CMPDEPAIRPORT == 0) {
50
                           // day
51
                           if (CMPDAY == 0) {
52
                                   // Hour
53
                                  if (CMPHOUR == 0) {
54
55
                                          // Minite
56
                                          return CMPMIN;
57
                                   }
                                  else {
58
                                          return CMPHOUR;
59
60
                                   }
                           }
61
62
                           else {
63
                                  return CMPDAY;
64
                   }
                   else {
66
                           return CMPDEPAIRPORT;
67
68
           }
70
           else {
                   return CMPFLIGHTNUM;
71
```

```
72
             }
73 }
 74
    // This function compares the records in the order of
75
    // departure airport, day, hour, minute, and flight number.
77
    // This function will return positive integer if r1 is greater than r2,
    // return negative integer if r1 is smaller than r2,
    // or return 0 if r1 is equal to r2.
79
     int compareByDepartureAirportAndDay(Record r1, Record r2) {
80
             if (CMPDEPAIRPORT == 0) {
81
 82
                     if (CMPDAY == 0) {
                             if (CMPHOUR == 0) {
 83
                                     if (CMPMIN == 0) {
 84
 85
                                              return CMPFLIGHTNUM;
 86
                                      }
 87
                                     else {
                                              return CMPMIN;
 88
 89
                                      }
 90
                             }
91
                             else {
 92
                                      return CMPHOUR;
 93
                             }
 94
                     }
                     else {
 95
96
                             return CMPDAY;
 97
                     }
98
             }
             else {
99
100
                     return CMPDEPAIRPORT;
101
             }
102 }
103
    // This function compares the records in the order of
104
    // day, hour, minute, and flight number.
105
    // This function will return positive integer if r1 is greater than r2,
106
107
    // return negative integer if r1 is smaller than r2,
    // or return 0 if r1 is equal to r2.
108
     int compareByTimeRange(Record r1, Record r2) {
109
             if (CMPDAY == 0) {
110
                     if (CMPHOUR == 0) {
111
112
                             if (CMPMIN == 0) {
113
                                      return CMPFLIGHTNUM;
114
                             }
                             else {
115
116
                                      return CMPMIN;
117
                             }
118
                     }
                     else {
119
120
                             return CMPHOUR;
121
                     }
122
             }
             else {
123
124
                     return CMPDAY;
125
             }
126 }
127
128 // This function compares the records in the order of
129 // departure airport, day, hour, and minute.
130 // This function will return positive integer if r1 is greater than r2,
131 // return negative integer if r1 is smaller than r2,
    // or return 0 if r1 is equal to r2.
    int compareByNext(Record r1, Record r2) {
             if (CMPDEPAIRPORT == 0) {
134
                     if (CMPDAY == 0) {
135
                             if (CMPHOUR == 0) {
136
                                      return CMPMIN;
137
138
                             }
                             else {
139
140
                                      return CMPHOUR;
141
                             }
142
                     }
```

```
else {
143
144
                            return CMPDAY;
145
                    }
            }
146
147
            else {
148
                    return CMPDEPAIRPORT;
149
            }
150
151
152
    // End of comparison functions
153
    154
    // Creates a new flight DB.
155
    FlightDb DbNew(void) {
156
            FlightDb db = malloc(sizeof(*db));
157
158
            if (db == NULL) {
159
                    fprintf(stderr, "error: out of memory\n");
            exit(EXIT_FAILURE);
160
161
            }
162
163
            db->byFlightNumber = TreeNew(compareByFlightNumber);
            db->byDepartureAirportAndDay = TreeNew(compareByDepartureAirportAndDay);
164
            db->byTimeRange = TreeNew(compareByTimeRange);
165
166
            db->byNext = TreeNew(compareByNext);
167
            return db;
168
169
170
    // Frees all memory allocated to the given flight DB.
171
172
    void
             DbFree(FlightDb db) {
173
            TreeFree(db->byFlightNumber, true);
174
            TreeFree(db->byDepartureAirportAndDay, false);
175
            TreeFree(db->byTimeRange, false);
            TreeFree(db->byNext, false);
176
177
            free(db);
178
179
180
     * Inserts a flight record into the given DB if there is not already
181
182
     * record with the same flight number, departure airport, day, hour and
183
     * minute.
     * If inserted successfully, this function takes ownership of the given
184
     * record (so the caller should not modify or free it).
185
186
     * Returns true if the record was successfully inserted, and false if
187
     * the DB already contained a record with the same flight number,
188
     * departure airport, day, hour and minute.
     * The time complexity of this function must be O(\log n).
189
     */
190
             DbInsertRecord(FlightDb db, Record r) {
191
    bool
192
            if (TreeInsert(db->byFlightNumber, r) == true) {
                    TreeInsert(db->byDepartureAirportAndDay, r);
193
                    TreeInsert(db->byTimeRange, r);
194
195
                    TreeInsert(db->byNext, r);
196
                    return true;
197
            }
198
            else {
199
                    return false;
200
            }
201
202
203
204
     * Searches for all records with the given flight number, and returns
     * them all in a list in increasing order of (day, hour, min). Returns
205
     * an empty list if there are no such records.
206
207
     * The records in the returned list should not be freed, but it is the
     * caller's responsibility to free the list itself.
208
     * The time complexity of this function must be O(\log n + m), where m is
209
     * the length of the returned list.
210
     */
211
212 List
             DbFindByFlightNumber(FlightDb db, char *flightNumber) {
            Record dummyLower = RecordNew(flightNumber, MINCHAR, MINCHAR, MINDAY,
213
```

```
MINHOURS, MINMINS, MINDURATION);
214
215
             Record dummyUpper = RecordNew(flightNumber, MAXCHAR, MAXCHAR, MAXDAY,
             MAXHOURS, MAXMINS, MAXDURATION);
216
217
218
             List 1 = TreeSearchBetween(db->byFlightNumber, dummyLower, dummyUpper);
219
220
             RecordFree(dummyLower);
221
             RecordFree(dummyUpper);
222
223
             return 1;
224 }
225
226 /**
227
     * Searches for all records with the given departure airport and day of
     * week (0 to 6), and returns them all in a list in increasing order of
228
229
     * (hour, min, flight number).
     * Returns an empty list if there are no such records.
230
231
     * The records in the returned list should not be freed, but it is the
232
     * caller's responsibility to free the list itself.
233
     * The time complexity of this function must be O(log n + m), where m is
234
     * the length of the returned list.
     */
235
236
    List
             DbFindByDepartureAirportDay(FlightDb db, char *departureAirport,
237
                                          int day) {
238
             Record dummyLower = RecordNew(MINCHAR, departureAirport,
             MINCHAR, day, MINHOURS, MINMINS, MINDURATION);
239
240
             Record dummyUpper = RecordNew(MAXCHAR, departureAirport,
             MAXCHAR, day, MAXHOURS, MAXMINS, MAXDURATION);
241
242
243
             List 1 = TreeSearchBetween(db->byDepartureAirportAndDay,
244
             dummyLower, dummyUpper);
245
             RecordFree(dummyLower);
246
             RecordFree(dummyUpper);
247
248
249
             return 1;
250 }
251
252
253 /**
254
     * Searches for all records between (day1, hour1, min1) and (day2,
     * hour2, min2), and returns them all in a list in increasing order of
255
     * (day, hour, min, flight number).
256
257
     * Returns an empty list if there are no such records.
258
     * The records in the returned list should not be freed, but it is the
259
     * caller's responsibility to free the list itself.
     * The time complexity of this function must be O(log n + m), where m is
260
     * the length of the returned list.
261
     */
262
263
    List
             DbFindBetweenTimes(FlightDb db,
264
                                 int day1, int hour1, int min1,
                                 int day2, int hour2, int min2) {
265
             Record dummyLower = RecordNew(MINCHAR, MINCHAR, MINCHAR, day1,
266
267
             hour1, min1, MINDURATION);
268
             Record dummyUpper = RecordNew(MAXCHAR, MAXCHAR, MAXCHAR, day2,
269
             hour2, min2, MAXDURATION);
270
             List 1 = TreeSearchBetween(db->byTimeRange, dummyLower, dummyUpper);
271
272
273
             RecordFree(dummyLower);
274
             RecordFree(dummyUpper);
275
276
             return 1;
277
278
279
     * Searches for and returns the earliest next flight from the given
280
     * departure airport, on or after the given (day, hour, min).
281
282
     * The returned record must not be freed or modified.
283
     * The time complexity of this function must be O(\log n).
284
     */
```

```
Record
             DbFindNextFlight(FlightDb db, char *departureAirport,
285
                               int day, int hour, int min) {
286
             Record dummy = RecordNew(MINCHAR, departureAirport, MINCHAR, day,
287
288
             hour, min, MINDURATION);
289
             Record n = TreeNext(db->byNext, dummy);
290
291
             RecordFree(dummy);
292
293
294
             return n;
295 }
296
```

!dryrun_record

```
1 Dryrun log for z5206267
   3 ** Testing Assignment 1
    -----
   5 ** Compiling AVLTree.c and FlightDb.c
   7 gcc -Wall -Werror -Wno-unused-function -g -o test test.c FlightDb.c AVLTree.c List.c Record.c
     gcc -Wall -Werror -Wno-unused-function -g -o testAss1 testAss1.c FlightDb.c AVLTree.c List.c Record.c
   9
  10 ** Compiles OK
  11
    ______
  13 ** Testing your program using the provided Test 1
  14 ** Test 1 PASSED
    -----
    ** Testing your program using the provided Test 2
  17 ** Test 2 PASSED
  18 -----
  19 ** Testing your program using the provided Test 3
  20 ** Test 3 PASSED
    -----
  22 ** Testing your program using the provided Test 4
  23 ** Test 4 PASSED
  24 -----
  25 ** Testing your program using the provided Test 5
  26 ** Test 5 PASSED
  27 -----
  28 ** Testing your program using the provided Test 6
  29 ** Test 6 PASSED
  30 -----
  31
  32 -----
  33 ** SUMMARY
  34
  35 ** You passed the provided simple tests 1, 2, 3, 4, 5 and 6.
    ** Automarking will use different test cases to extensively test your program.
     -----
gcc -Wall -Werror -Wno-unused-function -g -o testAss1 testAss1.c FlightDb.c AVLTree.c List.c Record.c
```

```
** Compiles OK
```

Tests

```
** Test 1: TreeInsert
** Test failed (student's output on left, expected on right). Output difference:-
tree is unbalanced: left height was 2, right height was
                                       The tree is balanced
The tree is not balanced
tree is unbalanced: left height was 2, right height was
                                       The tree is balanced
The tree is not balanced
-----
** Test 2: TreeSearchBetween
-----
** Test passed
-----
** Test 3: TreeSearchBetween
** Test passed
-----
** Test 4: TreeSearchBetween
-----
** Test passed
** Test 5: TreeSearchBetween
-----
** Test passed
-----
** Test 6: TreeNext
** Test passed
-----
** Test 7: TreeNext
-----
** Test passed
-----
** Test 8: TreeNext
-----
** Test passed
-----
** Test 9: TreeNext
-----
** Test passed
-----
** Test 10: TreeNext
** Test passed
----
** Test 11: TreeNext
-----
** Test passed
-----
** Test 12: TreeNext
** Test passed
-----
** Test 13: TreeNext
-----
** Test failed (student's output on left, expected on right). Output difference:-
Found flight:
                                       No flights found
H1|H|Z|Monday 0007|2
** Test 14: DbInsertRecord
-----
** Test passed
-----
** Test 15: DbInsertRecord
_____
** Test passed
** Test 16: DbInsertRecord
_____
** Test passed
** Test 17: DbInsertRecord
```

```
** Test passed
_____
** Test 18: DbInsertRecord
-----
** Test passed
** Test 19: DbFindByFlightNumber
-----
** Test passed
-----
** Test 20: DbFindByFlightNumber
-----
** Test passed
** Test 21: DbFindByFlightNumber
** Test passed
-----
** Test 22: DbFindByFlightNumber
** Test passed
** Test 23: DbFindByDepartureAirportDay
-----
** Test passed
** Test 24: DbFindByDepartureAirportDay
-----
** Test passed
-----
** Test 25: DbFindByDepartureAirportDay
-----
** Test passed
-----
** Test 26: DbFindByDepartureAirportDay
** Test passed
** Test 27: DbFindBetweenTimes
-----
** Test passed
-----
** Test 28: DbFindBetweenTimes
_____
** Test passed
** Test 29: DbFindBetweenTimes
----
** Test passed
-----
** Test 30: DbFindBetweenTimes
_____
** Test passed
-----
** Test 31: DbFindNextFlight
-----
** Test passed
** Test 32: DbFindNextFlight
_____
** Test passed
** Test 33: DbFindNextFlight
----
** Test passed
-----
** Test 34: DbFindNextFlight
** Test failed (student's output on left, expected on right). Output difference:-
Found flight:
                                       No flights found
```

Assessment

```
!!perftab
              ** PERFORMANCE ANALYSIS **
Test 1 (2.4)
              TreeInsert .. .. .. !!FAILed (-2.4)
Test 2 (0.2)
              TreeSearchBetween
                                           !!PASSed
Test 3 (0.2)
              TreeSearchBetween
                                 .. .. !!PASSed
Test 4 (0.2)
              TreeSearchBetween
                                       .. !!PASSed
              TreeSearchBetween
Test 5 (0.2)
                                           !!PASSed
Test 6 (0.2)
              TreeNext . .. .. !!PASSed
Test 7 (0.2)
              TreeNext . .. .. ..
                                           !!PASSed
Test 8 (0.2)
              TreeNext .
                                            !!PASSed
              TreeNext . .. .. .. ..
Test 9 (0.2)
                                           !!PASSed
Test 10 (0.2)
              TreeNext . .. .. .. !!PASSed
Test 11 (0.2)
              TreeNext .
                                           !!PASSed
              TreeNext . .. .. .. ..
Test 12 (0.2)
Test 13 (0.2)
              TreeNext .
                        .. .. .. .. !!FAILed (-0.2)
Test 14 (0.16) DbInsertRecord
                                           !!PASSed
Test 15 (0.16)
              DbInsertRecord
                                           !!PASSed
Test 16 (0.16)
              DbInsertRecord .. .. .. !!PASSed
Test 17 (0.16)
              DbInsertRecord .. .. .. !!PASSed
Test 18 (0.16)
              DbInsertRecord .. .. ..
                                           !!PASSed
Test 19 (0.4)
              DbFindByFlightNumber . .. !!PASSed
Test 20 (0.4)
              DbFindByFlightNumber . .. !!PASSed
Test 21 (0.4)
              DbFindByFlightNumber . .. !!PASSed
Test 22 (0.4)
              DbFindByFlightNumber . .. !!PASSed
Test 23 (0.4)
              DbFindByDepartureAirportDay .. !!PASSed
Test 24 (0.4)
              DbFindByDepartureAirportDay .. !!PASSed
Test 25 (0.4)
              DbFindByDepartureAirportDay .. !!PASSed
              DbFindByDepartureAirportDay .. !!PASSed
Test 26 (0.4)
Test 27 (0.4)
              DbFindBetweenTimes .. .. !!PASSed
Test 28 (0.4)
              DbFindBetweenTimes .. .. !!PASSed
Test 29 (0.4)
              DbFindBetweenTimes .. .. !!PASSed
Test 30 (0.4)
              DbFindBetweenTimes .. .. !!PASSed
Test 31 (0.2)
              DbFindNextFlight . .. .. !!PASSed
Test 32 (0.2)
              DbFindNextFlight . .. .. !!PASSed
Test 33 (0.3)
              DbFindNextFlight . .. .. !!PASSed
Test 34 (0.3)
              DbFindNextFlight . .. .. !!FAILed (-0.3)
              DbFindNextFlight . .. .. !!FAILed (-0.3)
Test 35 (0.3)
Test 36 (0.3)
              DbFindNextFlight . .. .. !!PASSed
!!perfmark
               ** TOTAL PERFORMANCE MARK:
                                          10/12
                                                   <== mark altered (original mark was 8.8)</pre>
!!marktab
                  MARKER'S ASSESSMENT **
                  Style and Complexity (3)
 + Great job in regards to style, you have
 + appropriate variable names, consistent
 + indentation and sensible comments.
 + I restored 1.2/2.4 automarks lost due to not
 + correctly balancing in treeInsert. While you
 + correctly check for balance and rotate, you do +
 + not update the heights after rotations.
 + The issue in dbfindnextflight is likely that
 + you did not consider the case where the time
 + wraps around.
 + ============ +
!!finalmark
              ** FINAL ASSIGNMENT MARK:
                                            13/15
5206267 Luo, Zheng
                                            3785/4 AEROAH
Marked by z5074990 on Mon May 3 20:42:35 2021
```