

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 {
16     Record rec;
17     Node left;
18     Node right;
19     int height;
20 };
21
22 struct tree {
23     Node root;
24     int (*compare)(Record, Record);
25 };
26
27 ///////////////////////////////////////////////////////////////////
28 // Auxiliary functions
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 l);
35 Record doTreeNextSearch(Tree t, Node n, Record r, Record *desiredRecord);
36 int TreeHeight(Node n);
37 Node rotateLeft(Node n);
38 Node rotateRight(Node n);
39 ///////////////////////////////////////////////////////////////////
40
41 static Node newNode(Record rec) {
42     Node n = malloc(sizeof(*n));
43     if (n == NULL) {
44         fprintf(stderr, "error: out of memory\n");
45         exit(EXIT_FAILURE);
46     }
47
48     n->rec = rec;
49     n->left = NULL;
50     n->right = NULL;
51     n->height = 0;
52     return n;
53 }
54
55 ///////////////////////////////////////////////////////////////////
56
57 Tree TreeNew(int (*compare)(Record, Record)) {
58     Tree t = malloc(sizeof(*t));
59     if (t == NULL) {
60         fprintf(stderr, "error: out of memory\n");
61         exit(EXIT_FAILURE);
62     }
63
64     t->root = NULL;
65     t->compare = compare;
66     return t;
67 }
68
69 ///////////////////////////////////////////////////////////////////
70
71 void TreeFree(Tree t, bool freeRecords) {

```

```

72     doTreeFree(t->root, freeRecords);
73     free(t);
74 }
75
76 static void doTreeFree(Node n, bool freeRecords) {
77     if (n != NULL) {
78         doTreeFree(n->left, freeRecords);
79         doTreeFree(n->right, freeRecords);
80         if (freeRecords) {
81             RecordFree(n->rec);
82         }
83         free(n);
84     }
85 }
86
87 ///////////////////////////////////////////////////////////////////
88
89 Record TreeSearch(Tree t, Record rec) {
90     return doTreeSearch(t, t->root, rec);
91 }
92
93 static Record doTreeSearch(Tree t, Node n, Record rec) {
94     if (n == NULL) {
95         return NULL;
96     }
97
98     int cmp = t->compare(rec, n->rec);
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 {
104         return n->rec;
105     }
106 }
107
108
109 ///////////////////////////////////////////////////////////////////
110 /* IMPORTANT:
111    Do NOT modify the code above this line.
112    You must not modify the 'node' and 'tree' structures defined above.
113    You must not modify the functions defined above.
114 */
115 ///////////////////////////////////////////////////////////////////
116
117 ///////////////////////////////////////////////////////////////////
118 /**
119  * Inserts the given record into the AVL tree.
120  * The time complexity of this function must be O(log n).
121  * Returns true if the record was inserted successfully, or false if
122  * there was already a record that compares equal to the given record in
123  * the tree (according to the comparison function).
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
131 Node doTreeInsert(Tree t, Node n, Record rec, bool *result) {
132     // Ending condition when successful found the desired location.
133     if (n == NULL) {
134         *result = true;
135         return newNode(rec);
136     }
137     // If repeated number has been found in the tree.
138     else if (t->compare(n->rec, rec) == 0) {
139         *result = false;
140     }
141     else {
142         // Find the location first.

```

```

143     if (t->compare(n->rec, rec) > 0) {
144         n->left = doTreeInsert(t, n->left, rec, result);
145     }
146     else {
147         n->right = doTreeInsert(t, n->right, rec, result);
148     }
149     // Check the balance of the tree,
150     // and rotate if necessary.
151     int LHeight = TreeHeight(n->left);
152     int RHeight = TreeHeight(n->right);
153
154     if (LHeight - RHeight > 1) {
155         if (t->compare(n->left->rec, rec) < 0) {
156             return rotateLeft(n);
157         }
158         else {
159             return rotateRight(n);
160         }
161     }
162     else if (RHeight - LHeight > 1) {
163         if (t->compare(n->right->rec, rec) > 0) {
164             return rotateRight(n);
165         }
166         else {
167             return rotateLeft(n);
168         }
169     }
170 }
171 return n;
172 }
173
174 // This function computes the tree height,
175 // return the height of the tree as integer.
176 int TreeHeight(Node n) {
177     if (n == NULL) {
178         return -1;
179     }
180
181     int lh = TreeHeight(n->left);
182     int rh = TreeHeight(n->right);
183     return 1 + ((lh > rh) ? lh : rh);
184 }
185
186 // This function rotates the tree towards left,
187 // return the node after rotation.
188 Node rotateLeft(Node n) {
189     if (n == NULL || n->right == NULL) {
190         return n;
191     }
192     Node n1 = n->right;
193     n->right = n1->left;
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) {
200     if (n == NULL || n->left == NULL) {
201         return n;
202     }
203     Node n2 = n->left;
204     n->left = n2->right;
205     n2->right = n;
206     return n2;
207 }
208
209
210 ///////////////////////////////////////////////////
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`.
216 * The time complexity of this function must be  $O(\log n + m)$ , where  $m$  is
217 * the length of the returned list.
218 */
219
220 List TreeSearchBetween(Tree t, Record lower, Record upper) {
221     List l = ListNew();
222     doTreeSearchBetween(t, t->root, lower, upper, l);
223     return l;
224 }
225
226 void doTreeSearchBetween(Tree t, Node n,
227                         Record lower, Record upper, List l) {
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:
237     // No necessary to visit left subtree,
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) {
241         doTreeSearchBetween(t, n->right, lower, upper, l);
242     }
243     // Go to left if current is larger than upper
244     else if (upperCmp > 0) {
245         doTreeSearchBetween(t, n->left, lower, upper, l);
246     }
247     else {
248         doTreeSearchBetween(t, n->left, lower, upper, l);
249         ListAppend(l, n->rec);
250         doTreeSearchBetween(t, n->right, lower, upper, l);
251     }
252     return;
253 }
254 }
255
256 ///////////////////////////////////////////////////
257 /**
258 * Returns a smallest record greater than or equal to the given record r
259 * (according to the comparison function).
260 * Time complexity of the function must be  $O(\log n)$ .
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.
269     if (n == NULL && *desiredRecord != NULL) {
270         return *desiredRecord;
271     }
272     // This is designed for the situation of no flights
273     // in the rest of the week,
274     // Then the function will search for the upcoming weeks.
275     // For example,
276     // when searching the flight on late Sunday evening,
277     // if there is no any flight available in later Sunday,
278     // then this function will start searching from Monday 0am.
279     else if (n == NULL && *desiredRecord == NULL) {
280
281         Record new = RecordNew(RecordGetFlightNumber(r),
282                                RecordGetDepartureAirport(r), RecordGetArrivalAirport(r), 0, 0, 0, 0);
283
284         return doTreeNextSearch(t, t->root, new, desiredRecord);

```

```
285     }
286
287     int cmp = t->compare(r, n->rec);
288
289     // The next possible available flight will be recorded as desiredRecord,
290     // which will updated as the tree searching progress,
291     // desired record will be returned after searching reached the leaf.
292     if (cmp < 0) {
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
6 #include <stdio.h>
7 #include <stdlib.h>
8 #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))
16 #define CMPDEPAIRPORT strcmp(RecordGetDepartureAirport(r1), \
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
29 #define MINCHAR ""
30 #define MAXCHAR "zzzzzz"
31
32 struct flightDb {
33     Tree byFlightNumber;
34     Tree byDepartureAirportAndDay;
35     Tree byTimeRange;
36     Tree byNext;
37 };
38
39 ///////////////////////////////////////////////////////////////////
40 // Comparison functions
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,
45 // return negative integer if r1 is smaller than r2,
46 // or return 0 if r1 is equal to r2.
47 int compareByFlightNumber(Record r1, Record r2) {
48     if (CMPFLIGHTNUM == 0) {
49         // Departure airport
50         if (CMPDEPAIRPORT == 0) {
51             // day
52             if (CMPDAY == 0) {
53                 // Hour
54                 if (CMPHOUR == 0) {
55                     // Minute
56                     return CMPMIN;
57                 }
58                 else {
59                     return CMPHOUR;
60                 }
61             }
62             else {
63                 return CMPDAY;
64             }
65         }
66         else {
67             return CMPDEPAIRPORT;
68         }
69     }
70     else {
71         return CMPFLIGHTNUM;

```

```

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

```



```

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

```

```

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

```

```

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

```

!dryrun_record

```

1 Dryrun log for z5206267
2 -----
3 ** Testing Assignment 1
4 -----
5 ** Compiling AVLTree.c and FlightDb.c
6
7 gcc -Wall -Werror -Wno-unused-function -g -o test test.c FlightDb.c AVLTree.c List.c Record.c
8 gcc -Wall -Werror -Wno-unused-function -g -o testAss1 testAss1.c FlightDb.c AVLTree.c List.c Record.c
9
10 ** Compiles OK
11
12 -----
13 ** Testing your program using the provided Test 1
14 ** Test 1 PASSED
15 -----
16 ** 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
21 -----
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.
36 ** Automarking will use different test cases to extensively test your program.
37 -----
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
```

```
AB123|HBA|MEL|Monday 0000|75
-----
** Test 35: DbFindNextFlight
-----
** Test failed (student's output on left, expected on right). Output difference:-
CD456|PER|SYD|Wednesday 0930|260      |      JQ560|MEL|BNE|Monday 0900|85
-----
** Test 36: DbFindNextFlight
-----
** Test passed
-----
```

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
Test  5 (0.2)  TreeSearchBetween  ..  ..  ..  !!PASSEd
Test  6 (0.2)  TreeNext .  ..  ..  ..  ..  !!PASSEd
Test  7 (0.2)  TreeNext .  ..  ..  ..  ..  !!PASSEd
Test  8 (0.2)  TreeNext .  ..  ..  ..  ..  !!PASSEd
Test  9 (0.2)  TreeNext .  ..  ..  ..  ..  !!PASSEd
Test 10 (0.2)  TreeNext .  ..  ..  ..  ..  !!PASSEd
Test 11 (0.2)  TreeNext .  ..  ..  ..  ..  !!PASSEd
Test 12 (0.2)  TreeNext .  ..  ..  ..  ..  !!PASSEd
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
Test 26 (0.4)  DbFindByDepartureAirportDay ..  !!PASSEd
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)
Test 35 (0.3)  DbFindNextFlight .  ..  ..  ..  !!FAILED (-0.3)
Test 36 (0.3)  DbFindNextFlight .  ..  ..  ..  !!PASSEd

!!perfmark      ** TOTAL PERFORMANCE MARK:    10/12    <== mark altered (original mark was 8.8)

!!marktab      **  MARKER'S  ASSESSMENT  **

                                Style and Complexity  (3)   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

```