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

```c
// Assignment 1 21T1 COMP2521: ADTs: FlightDb Using a Generic AVL Tree
//
// This program was written by Zheng Luo (z5206267@ad.unsw.edu.au)
// on March/2021

#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

#include "List.h"
#include "Record.h"
#include "AVLTree.h"

typedef struct node *Node;
struct node {
    Record rec;
    Node   left;
    Node   right;
    int    height;
};

struct tree {
    Node    root;
    int     (*compare)(Record, Record);
};

////////////////////////////////////////////////////////////////////////
// Auxiliary functions

static void doTreeFree(Node n, bool freeRecords);
static Node newNode(Record rec);
static Record doTreeSearch(Tree t, Node n, Record rec);
Node doTreeInsert(Tree t, Node n, Record rec, bool *result);
void doTreeSearchBetween(Tree t, Node n, Record lower, Record upper, List l);
Record doTreeNextSearch(Tree t, Node n, Record r, Record *desiredRecord);
int TreeHeight(Node n);
Node rotateLeft(Node n);
Node rotateRight(Node n);
////////////////////////////////////////////////////////////////////////

static Node newNode(Record rec) {
    Node n = malloc(sizeof(*n));
    if (n == NULL) {
        fprintf(stderr, "error: out of memory\n");
        exit(EXIT_FAILURE);
    }

    n->rec = rec;
    n->left = NULL;
    n->right = NULL;
    n->height = 0;
    return n;
}

////////////////////////////////////////////////////////////////////////

Tree TreeNew(int (*compare)(Record, Record)) {
    Tree t = malloc(sizeof(*t));
    if (t == NULL) {
        fprintf(stderr, "error: out of memory\n");
        exit(EXIT_FAILURE);
    }

    t->root = NULL;
    t->compare = compare;
    return t;
}

////////////////////////////////////////////////////////////////////////

void TreeFree(Tree t, bool freeRecords) {
    doTreeFree(t->root, freeRecords);
    free(t);
}

static void doTreeFree(Node n, bool freeRecords) {
    if (n != NULL) {
        doTreeFree(n->left, freeRecords);
        doTreeFree(n->right, freeRecords);
        if (freeRecords) {
            RecordFree(n->rec);
        }
        free(n);
    }
}

////////////////////////////////////////////////////////////////////////

Record TreeSearch(Tree t, Record rec) {
    return doTreeSearch(t, t->root, rec);
}

static Record doTreeSearch(Tree t, Node n, Record rec) {
    if (n == NULL) {
        return NULL;
    }

    int cmp = t->compare(rec, n->rec);
    if (cmp < 0) {
        return doTreeSearch(t, n->left, rec);
    } else if (cmp > 0) {
        return doTreeSearch(t, n->right, rec);
    } else {
        return n->rec;
    }
}


////////////////////////////////////////////////////////////////////////
/* IMPORTANT:
   Do NOT modify the code above this line.
   You must not modify the 'node' and 'tree' structures defined above.
   You must not modify the functions defined above.
*/
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
/**
 * Inserts the given record into the AVL tree.
 * The time complexity of this function must be O(log n).
 * Returns true if the record was inserted  successfully,  or  false  if
 * there was already a record that compares equal to the given record in
 * the tree (according to the comparison function).
 */
bool TreeInsert(Tree t, Record rec) {
    bool result = false;
    t->root = doTreeInsert(t, t->root, rec, &result);
    return result;
}

Node doTreeInsert(Tree t, Node n, Record rec, bool *result) {
    // Ending condition when successful found the desired location.
    if (n == NULL) {
        *result = true;
        return newNode(rec);
    }
    // If repeated number has been found in the tree.
    else if (t->compare(n->rec, rec) == 0) {
        *result = false;
    }
    else {
        // Find the location first.
        if (t->compare(n->rec, rec) > 0) {
            n->left = doTreeInsert(t, n->left, rec, result);
        }
        else {
            n->right = doTreeInsert(t, n->right, rec, result);
        }
        // Check the balance of the tree,
        // and rotate if necessary.
        int LHeight = TreeHeight(n->left);
        int RHeight = TreeHeight(n->right);

        if (LHeight - RHeight > 1) {
            if (t->compare(n->left->rec, rec) < 0) {
                return rotateLeft(n);
            }
            else {
                return rotateRight(n);
            }
        }
        else if (RHeight - LHeight > 1) {
            if (t->compare(n->right->rec, rec) > 0) {
                return rotateRight(n);
            }
            else {
                return rotateLeft(n);
            }
        }
    }
    return n;
}

// This function computes the tree height,
// return the height of the tree as integer.
int TreeHeight(Node n) {
    if (n == NULL) {
        return -1;
    }

    int lh = TreeHeight(n->left);
    int rh = TreeHeight(n->right);
    return 1 + ((lh > rh) ? lh : rh);
}

// This function rotates the tree towards left,
// return the node after rotation.
Node rotateLeft(Node n) {
    if (n == NULL || n->right == NULL) {
        return n;
    }
    Node n1 = n->right;
    n->right = n1->left;
    n1->left = n;
    return n1;
}
// This function rotates the tree towards right,
// return the node after rotation.
Node rotateRight(Node n) {
    if (n == NULL || n->left == NULL) {
        return n;
    }
    Node n2 = n->left;
    n->left = n2->right;
    n2->right = n;
    return n2;
}


////////////////////////////////////////////////////////////////////////

/**
 * Searches  for  all  records  between the two given records, inclusive
 * (according to the comparison function) and returns the records  in  a
 * 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
 * the length of the returned list.
 */

List TreeSearchBetween(Tree t, Record lower, Record upper) {
    List l = ListNew();
    doTreeSearchBetween(t, t->root, lower, upper, l);
    return l;
}

void doTreeSearchBetween(Tree t, Node n,
                                Record lower, Record upper, List l) {
    // Ending condition
    if (n == NULL) {
        return;
    }
    int lowerCmp = t->compare(n->rec, lower);
    int upperCmp = t->compare(n->rec, upper);

    // For sorted list: Inorder search must applied.
    // In order to visit minimum nodes:
    // No necessary to visit left subtree,
    // as left subtree will all smaller than current, which < lower,
    // so go to right if current is smaller than lower
    if (lowerCmp < 0) {
        doTreeSearchBetween(t, n->right, lower, upper, l);
    }
    // Go to left if current is larger than upper
    else if (upperCmp > 0) {
        doTreeSearchBetween(t, n->left, lower, upper, l);
    }
    else {
        doTreeSearchBetween(t, n->left, lower, upper, l);
        ListAppend(l, n->rec);
        doTreeSearchBetween(t, n->right, lower, upper, l);
    }
    return;

}

////////////////////////////////////////////////////////////////////////
/**
 * Returns a smallest record greater than or equal to the given record r
 * (according to the comparision function).
 * Time complexity of the function must be O(log n).
 */
Record TreeNext(Tree t, Record r) {
    Record desiredRecord = NULL;
    return doTreeNextSearch(t, t->root, r, &desiredRecord);
}

Record doTreeNextSearch(Tree t, Node n, Record r, Record *desiredRecord) {
    // There is no exact match in the tree whatsoever.
    if (n == NULL && *desiredRecord != NULL) {
        return *desiredRecord;
    }
    // This is designed for the situation of no flights
    // in the rest of the week,
    // Then the function will search for the upcoming weeks.
    // For example,
    // when searching the flight on late Sunday evening,
    // if there is no any flight availble in later Sunday,
    // then this function will start searching from Monday 0am.
    else if (n == NULL && *desiredRecord == NULL) {

        Record new = RecordNew(RecordGetFlightNumber(r),
        RecordGetDepartureAirport(r), RecordGetArrivalAirport(r), 0, 0, 0, 0);

        return doTreeNextSearch(t, t->root, new, desiredRecord);
    }

    int cmp = t->compare(r, n->rec);

    // The next possible available flight will be recorded as desiredRecord,
    // which will updated as the tree searching progress,
    // desired record will be returned after searching reached the leaf.
    if (cmp < 0) {
        *desiredRecord = n->rec;
        return doTreeNextSearch(t, n->left, r, desiredRecord);
    }
    else if (cmp > 0) {
        return doTreeNextSearch(t, n->right, r, desiredRecord);
    }
    else {
        return n->rec;
    }
}
```

FlightDb.c

```c
// Assignment 1 21T1 COMP2521: ADTs: FlightDb Using a Generic AVL Tree
//
// This program was written by Zheng Luo (z5206267@ad.unsw.edu.au)
// on March/2021

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "List.h"
#include "FlightDb.h"
#include "AVLTree.h"

#define CMPFLIGHTNUM strcmp(RecordGetFlightNumber(r1), \
RecordGetFlightNumber(r2))
#define CMPDEPAIRPORT strcmp(RecordGetDepartureAirport(r1), \
RecordGetDepartureAirport(r2))
#define CMPDAY RecordGetDepartureDay(r1) - RecordGetDepartureDay(r2)
#define CMPHOUR RecordGetDepartureHour(r1) - RecordGetDepartureHour(r2)
#define CMPMIN RecordGetDepartureMinute(r1) - RecordGetDepartureMinute(r2)
#define MINDAY 0
#define MAXDAY 6
#define MINHOURS 0
#define MAXHOURS 23
#define MINMINS 0
#define MAXMINS 59
#define MINDURATION 0
#define MAXDURATION 9999
#define MINCHAR ""
#define MAXCHAR "zzzzzzz"

struct flightDb {
        Tree byFlightNumber;
        Tree byDepartureAirportAndDay;
        Tree byTimeRange;
        Tree byNext;
};

////////////////////////////////////////////////////////////////////////
// Comparison functions

// This function compares the records in the order of
// flight number, departure airport, day, hour and minute.
// 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.
int compareByFlightNumber(Record r1, Record r2) {
        if (CMPFLIGHTNUM == 0) {
                // Departure airport
                if (CMPDEPAIRPORT == 0) {
                        // day
                        if (CMPDAY == 0) {
                                // Hour
                                if (CMPHOUR == 0) {
                                        // Minite
                                        return CMPMIN;
                                }
                                else {
                                        return CMPHOUR;
                                }
                        }
                        else {
                                return CMPDAY;
                        }
                }
                else {
                        return CMPDEPAIRPORT;
                }
        }
        else {
                return CMPFLIGHTNUM;
```

```c
	}
}

// This function compares the records in the order of
// departure airport, day, hour, minute, and flight number.
// 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.
int compareByDepartureAirportAndDay(Record r1, Record r2) {
	if (CMPDEPAIRPORT == 0) {
		if (CMPDAY == 0) {
			if (CMPHOUR == 0) {
				if (CMPMIN == 0) {
					return CMPFLIGHTNUM;
				}
				else {
					return CMPMIN;
				}
			}
			else {
				return CMPHOUR;
			}
		}
		else {
			return CMPDAY;
		}
	}
	else {
		return CMPDEPAIRPORT;
	}
}

// This function compares the records in the order of
// day, hour, minute, and flight number.
// 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.
int compareByTimeRange(Record r1, Record r2) {
	if (CMPDAY == 0) {
		if (CMPHOUR == 0) {
			if (CMPMIN == 0) {
				return CMPFLIGHTNUM;
			}
			else {
				return CMPMIN;
			}
		}
		else {
			return CMPHOUR;
		}
	}
	else {
		return CMPDAY;
	}
}

// This function compares the records in the order of
// departure airport, day, hour, and minute.
// 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.
int compareByNext(Record r1, Record r2) {
	if (CMPDEPAIRPORT == 0) {
		if (CMPDAY == 0) {
			if (CMPHOUR == 0) {
				return CMPMIN;
			}
			else {
				return CMPHOUR;
			}
		}
```

```
                else {
                        return CMPDAY;
                }
        }
        else {
                return CMPDEPAIRPORT;
        }
}

// End of comparison functions
////////////////////////////////////////////////////////////////////////

// Creates a new flight DB.
FlightDb DbNew(void) {
        FlightDb db = malloc(sizeof(*db));
        if (db == NULL) {
                fprintf(stderr, "error: out of memory\n");
        exit(EXIT_FAILURE);
        }

        db->byFlightNumber = TreeNew(compareByFlightNumber);
        db->byDepartureAirportAndDay = TreeNew(compareByDepartureAirportAndDay);
        db->byTimeRange = TreeNew(compareByTimeRange);
        db->byNext = TreeNew(compareByNext);
        return db;
}


// Frees all memory allocated to the given flight DB.
void     DbFree(FlightDb db) {
        TreeFree(db->byFlightNumber, true);
        TreeFree(db->byDepartureAirportAndDay, false);
        TreeFree(db->byTimeRange, false);
        TreeFree(db->byNext, false);
        free(db);
}

/**
 * Inserts  a  flight  record  into the given DB if there is not already
 * record with the same flight number, departure airport, day, hour  and
 * minute.
 * If  inserted successfully, this function takes ownership of the given
 * record (so the caller should not modify or free it).
 * Returns true if the record was successfully inserted,  and  false  if
 * the  DB  already  contained  a  record  with  the same flight number,
 * departure airport, day, hour and minute.
 * The time complexity of this function must be O(log n).
 */
bool     DbInsertRecord(FlightDb db, Record r) {
        if (TreeInsert(db->byFlightNumber, r) == true) {
                TreeInsert(db->byDepartureAirportAndDay, r);
                TreeInsert(db->byTimeRange, r);
                TreeInsert(db->byNext, r);
                return true;
        }
        else {
                return false;
        }
}

/**
 * Searches  for  all  records with the given flight number, and returns
 * them all in a list in increasing order of  (day, hour, min).  Returns
 * an empty list if there are no such records.
 * The  records  in the returned list should not be freed, but it is the
 * caller's responsibility to free the list itself.
 * The time complexity of this function must be O(log n + m), where m is
 * the length of the returned list.
 */
List     DbFindByFlightNumber(FlightDb db, char *flightNumber) {
        Record dummyLower = RecordNew(flightNumber, MINCHAR, MINCHAR, MINDAY,
        MINHOURS, MINMINS, MINDURATION);
        Record dummyUpper = RecordNew(flightNumber, MAXCHAR, MAXCHAR, MAXDAY,
        MAXHOURS, MAXMINS, MAXDURATION);

        List l = TreeSearchBetween(db->byFlightNumber, dummyLower, dummyUpper);

        RecordFree(dummyLower);
        RecordFree(dummyUpper);

        return l;
}

/**
 * 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
 * (hour, min, flight number).
 * Returns an empty list if there are no such records.
 * The  records  in the returned list should not be freed, but it is the
 * caller's responsibility to free the list itself.
 * The time complexity of this function must be O(log n + m), where m is
 * the length of the returned list.
 */
List     DbFindByDepartureAirportDay(FlightDb db, char *departureAirport,
                                     int day) {
        Record dummyLower = RecordNew(MINCHAR, departureAirport,
        MINCHAR, day, MINHOURS, MINMINS, MINDURATION);
        Record dummyUpper = RecordNew(MAXCHAR, departureAirport,
        MAXCHAR, day, MAXHOURS, MAXMINS, MAXDURATION);

        List l = TreeSearchBetween(db->byDepartureAirportAndDay,
        dummyLower, dummyUpper);

        RecordFree(dummyLower);
        RecordFree(dummyUpper);

        return l;
}


/**
 * Searches  for  all  records  between  (day1, hour1, min1)  and (day2,
 * hour2, min2), and returns them all in a list in increasing  order  of
 * (day, hour, min, flight number).
 * Returns an empty list if there are no such records.
 * The  records  in the returned list should not be freed, but it is the
 * caller's responsibility to free the list itself.
 * The time complexity of this function must be O(log n + m), where m is
 * the length of the returned list.
 */
List     DbFindBetweenTimes(FlightDb db,
                            int day1, int hour1, int min1,
                            int day2, int hour2, int min2) {
        Record dummyLower = RecordNew(MINCHAR, MINCHAR, MINCHAR, day1,
        hour1, min1, MINDURATION);
        Record dummyUpper = RecordNew(MAXCHAR, MAXCHAR, MAXCHAR, day2,
        hour2, min2, MAXDURATION);

        List l = TreeSearchBetween(db->byTimeRange, dummyLower, dummyUpper);

        RecordFree(dummyLower);
        RecordFree(dummyUpper);

        return l;
}

/**
 * Searches  for  and  returns  the  earliest next flight from the given
 * departure airport, on or after the given (day, hour, min).
 * The returned record must not be freed or modified.
 * The time complexity of this function must be O(log n).
 */
```

```
Record   DbFindNextFlight(FlightDb db, char *departureAirport,
                          int day, int hour, int min) {
        Record dummy = RecordNew(MINCHAR, departureAirport, MINCHAR, day,
        hour, min, MINDURATION);

        Record n = TreeNext(db->byNext, dummy);

        RecordFree(dummy);

        return n;
}

```

!dryrun_record

```
Dryrun log for z5206267
-------------------------------------------------
** Testing Assignment 1
-------------------------------------------------
** Compiling AVLTree.c and FlightDb.c

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

** Compiles OK

-------------------------------------------------
** Testing your program using the provided Test 1
** Test 1 PASSED
-------------------------------------------------
** Testing your program using the provided Test 2
** Test 2 PASSED
-------------------------------------------------
** Testing your program using the provided Test 3
** Test 3 PASSED
-------------------------------------------------
** Testing your program using the provided Test 4
** Test 4 PASSED
-------------------------------------------------
** Testing your program using the provided Test 5
** Test 5 PASSED
-------------------------------------------------
** Testing your program using the provided Test 6
** Test 6 PASSED
-------------------------------------------------

-------------------------------------------------
** SUMMARY

** 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
```

```
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
```