Отчет к лабораторной работе №2

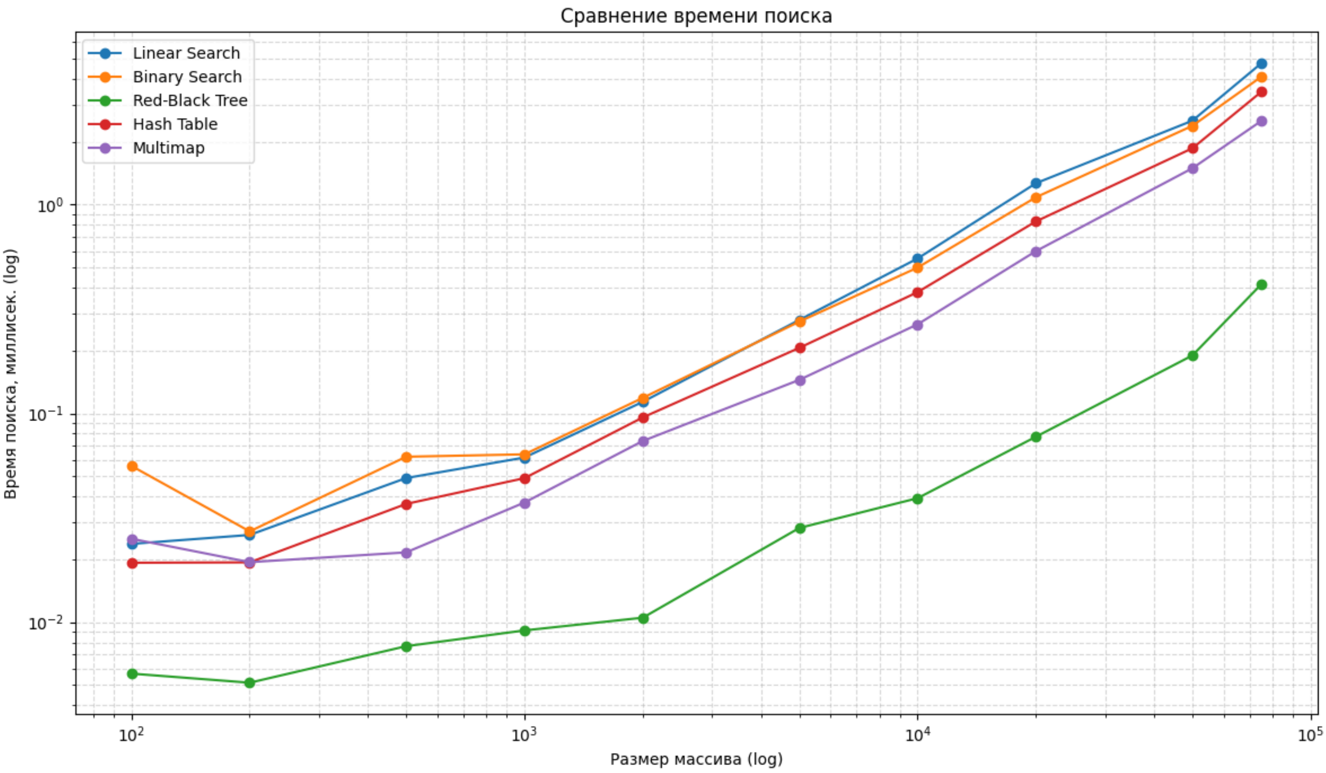
Вариант №21

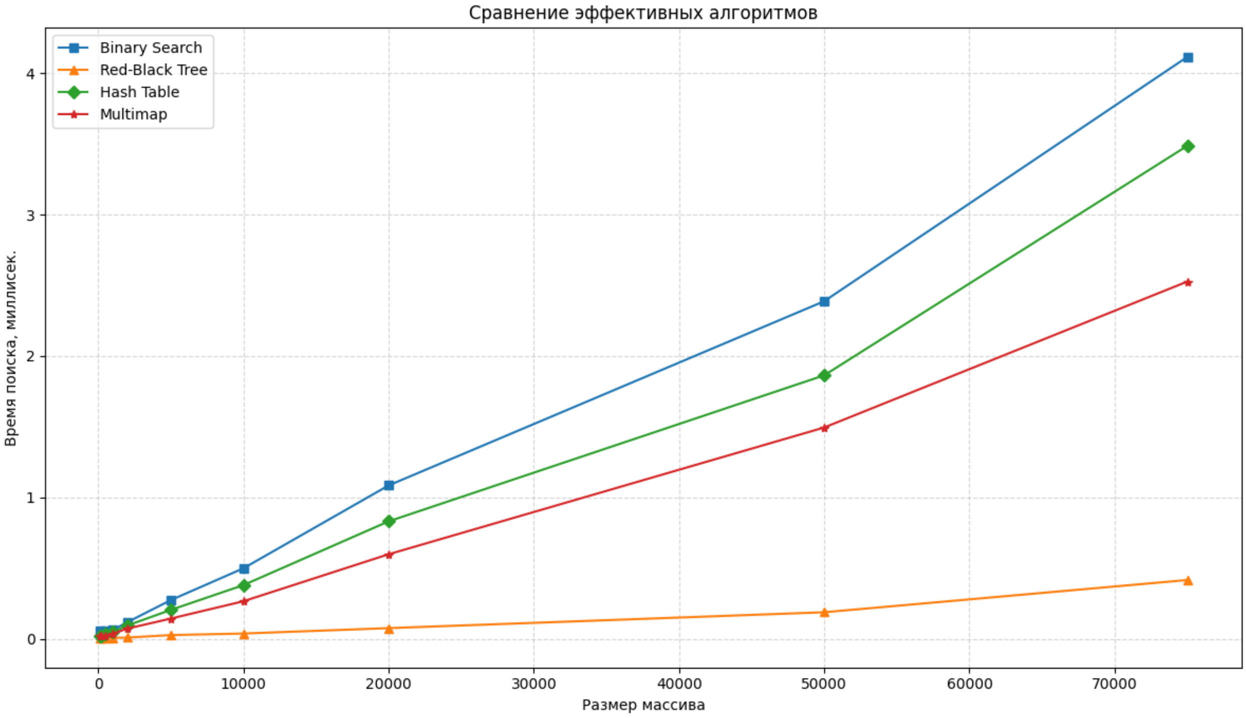
Труфанова Анастасия СКБ222

Ссылка на исходный код программы в репозитории

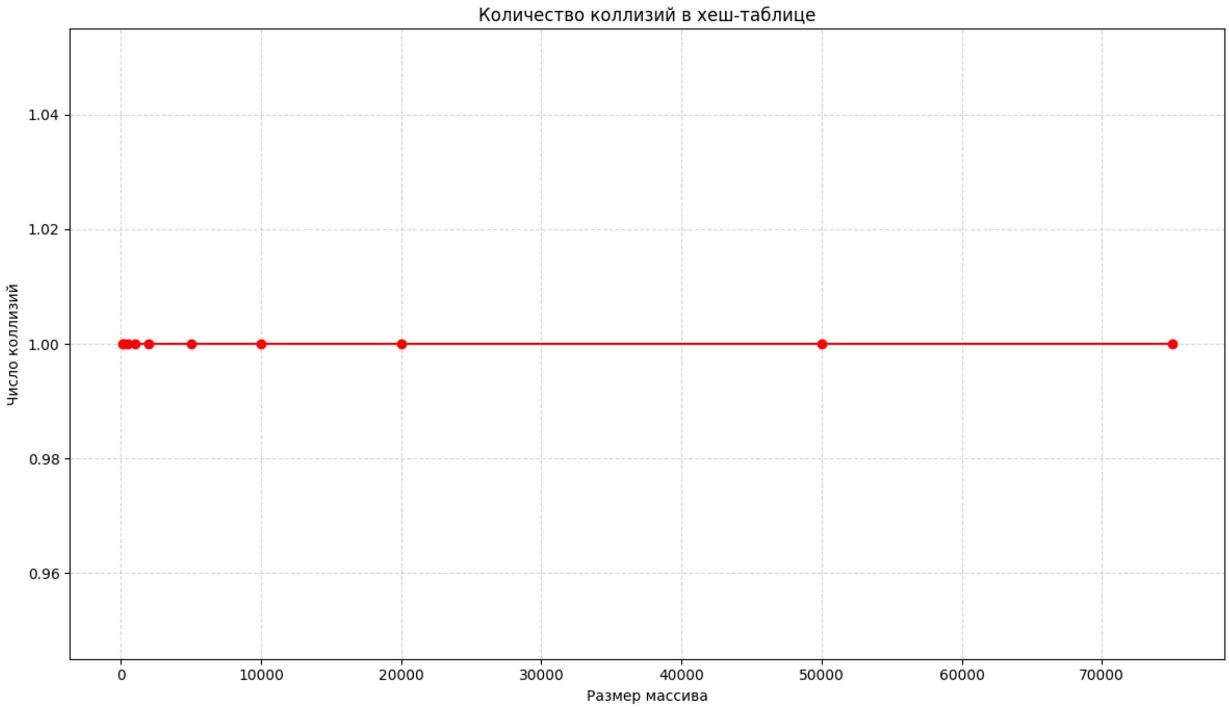
https://github.com/neko-nyashka/lab2_data_search_algorithms/tree/develop/src

Графики зависимости времени поиска от размерности массива





Графики зависимости числа коллизий от размерности массива



My Project

Generated by Doxygen 1.9.5

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 BSTree Class Reference

Binary Search Tree implementation for Flight objects.

```
#include <bstree.h>
```

Public Member Functions

- void Insert (Flight key)

  Inserts a flight into the tree.
- int Search (Flight key, std::vector< Flight > &result)

  Searches for flights matching criteria.
- BSTree ()

  Default constructor.
- BSTree (TreeNode ∗root_)

  Constructor with root initialization.
- ∼BSTree ()

  Destructor - cleans up all nodes.
- const TreeNode ∗ GetRoot () const

  Gets root node.
- void SetRoot (TreeNode ∗root_)

  Sets new root node.

3.1.1 Detailed Description

Binary Search Tree implementation for Flight objects.

Definition at line 28 of file bstree.h.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 BSTree() [1/2]

```
BSTree::BSTree ( )
```

Default constructor.

Definition at line 21 of file bstree.cpp.

3.1.2.2 BSTree() [2/2]

```
BSTree::BSTree (
            TreeNode * root_ )
```

Constructor with root initialization.

Parameters

root←↩ _	Node to set as initial root

Definition at line 25 of file bstree.cpp.

3.1.2.3 ∼BSTree()

```
BSTree::∼BSTree ( )
```

Destructor - cleans up all nodes.

Definition at line 57 of file bstree.cpp.

3.1.3 Member Function Documentation

3.1.3.1 GetRoot()

```
const TreeNode * BSTree::GetRoot ( ) const
```

Gets root node.

Returns

const TreeNode∗ Read-only pointer to root

Definition at line 8 of file bstree.cpp.

3.1.3.2 Insert()

```
void BSTree::Insert (
            Flight key )
```

Inserts a flight into the tree.

Parameters

key	Flight object to insert

Definition at line 61 of file bstree.cpp.

3.1.3.3 Search()

```
int BSTree::Search (
            Flight key,
            std::vector< Flight > & result )
```

Searches for flights matching criteria.

Parameters

key	Flight object with search criteria
result	Vector to store matching flights

Returns

Number of matches found

Definition at line 65 of file bstree.cpp.

3.1.3.4 SetRoot()

```
void BSTree::SetRoot (
            TreeNode * root_ )
```

Sets new root node.

Parameters

root↩ _	Pointer to new root node

Definition at line 5 of file bstree.cpp.

The documentation for this class was generated from the following files:

- /Users/anastasiatrufanova/Desktop/lab2_data_search_algorithms/src/bstree.h
- /Users/anastasiatrufanova/Desktop/lab2_data_search_algorithms/src/bstree.cpp

3.2 Flight Class Reference

Represents flight information.

```
#include <flight.h>
```

Public Member Functions

- Flight ()

  Default constructor.
- Flight (std::string flight_number_, std::string airline_, std::string arrival_date_, std::string arrival_time_, int passengers_)

  Parameterized constructor.
- ∼Flight ()=default

  Default destructor.
- std::string Get_flight_number () const
- std::string Get_airline () const
- std::string Get_arrival_date () const
- std::string Get_arrival_time () const
- int Get_passengers () const
- void Set_flight_number (std::string)
- void Set_airline (std::string)
- void Set_arrival_date (std::string)
- void Set_arrival_time (std::string)
- void Set_passengers (int)
- bool operator> (const Flight &other) const
- bool operator< (const Flight &other) const
- bool operator<= (const Flight &other) const
- bool operator>= (const Flight &other) const
- bool operator== (const Flight &other) const
- Flight & operator= (const Flight &other)=default

  Assignment operator.

Friends

- std::ostream & operator<< (std::ostream &os, const Flight &flight)

  Output stream operator.

3.2.1 Detailed Description

Represents flight information.

Definition at line 10 of file flight.h.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 Flight() [1/2]

```
Flight::Flight ( )
```

Default constructor.

Definition at line 4 of file flight.cpp.

3.2.2.2 Flight() [2/2]

```
Flight::Flight (
             std::string flight_number_,
             std::string airline_,
             std::string arrival_date_,
             std::string arrival_time_,
             int passengers_ )
```

Parameterized constructor.

Parameters

*flight_number_*	Flight number
*airline_*	Airline name
*arrival_date_*	Arrival date
*arrival_time_*	Arrival time
*passengers_*	Passenger count

Definition at line 13 of file flight.cpp.

3.2.2.3 ∼Flight()

```
Flight::∼Flight ( )  [default]
```

Default destructor.

3.2.3 Member Function Documentation

3.2.3.1 Get_airline()

```
std::string Flight::Get_airline ( ) const
```

Definition at line 24 of file flight.cpp.

3.2.3.2 Get_arrival_date()

```
std::string Flight::Get_arrival_date ( ) const
```

Definition at line 27 of file flight.cpp.

3.2.3.3 Get_arrival_time()

```
std::string Flight::Get_arrival_time ( ) const
```

Definition at line 31 of file flight.cpp.

3.2.3.4 Get_flight_number()

```
std::string Flight::Get_flight_number ( ) const
```

Definition at line 21 of file flight.cpp.

3.2.3.5 Get_passengers()

```
int Flight::Get_passengers ( ) const
```

Definition at line 35 of file flight.cpp.

3.2.3.6 operator<()

```
bool Flight::operator< (
             const Flight & other ) const
```

Definition at line 66 of file flight.cpp.

3.2.3.7 operator<=()

```
bool Flight::operator<= (
             const Flight & other ) const
```

Definition at line 70 of file flight.cpp.

3.2.3.8 operator=()

```
Flight & Flight::operator= (
             const Flight & other )  [default]
```

Assignment operator.

Parameters

other	Flight to assign

Returns

Reference to current object

3.2.3.9 operator==()

```
bool Flight::operator== (
             const Flight & other ) const
```

Definition at line 74 of file flight.cpp.

3.2.3.10 operator>()

```
bool Flight::operator> (
             const Flight & other ) const
```

Definition at line 58 of file flight.cpp.

3.2.3.11 operator>=()

```
bool Flight::operator>= (
             const Flight & other ) const
```

Definition at line 62 of file flight.cpp.

3.2.3.12 Set_airline()

```
void Flight::Set_airline (
             std::string airline_ )
```

Definition at line 42 of file flight.cpp.

3.2.3.13 Set_arrival_date()

```
void Flight::Set_arrival_date (
             std::string arrival_date_ )
```

Definition at line 45 of file flight.cpp.

3.2.3.14 Set_arrival_time()

```
void Flight::Set_arrival_time (
             std::string arrival_time_ )
```

Definition at line 49 of file flight.cpp.

3.2.3.15 Set_flight_number()

```
void Flight::Set_flight_number (
             std::string flight_number_ )
```

Definition at line 39 of file flight.cpp.

3.2.3.16 Set_passengers()

```
void Flight::Set_passengers (
             int passengers_ )
```

Definition at line 53 of file flight.cpp.

3.2.4 Friends And Related Function Documentation

3.2.4.1 operator<<

```
std::ostream & operator<< (
             std::ostream & os,
             const Flight & flight )   [friend]
```

Output stream operator.

Parameters

os	Output stream
flight	Flight to output

Returns

Reference to output stream

Definition at line 79 of file flight.cpp.

The documentation for this class was generated from the following files:

- /Users/anastasiatrufanova/Desktop/lab2_data_search_algorithms/src/flight.h
- /Users/anastasiatrufanova/Desktop/lab2_data_search_algorithms/src/flight.cpp

3.3 HashTable Class Reference

Hash table implementation with chaining.

```
#include <hash_table.h>
```

Public Member Functions

- HashTable ()=default

  Default constructor.
- HashTable (int size_)

  Constructor with size initialization.
- ∼HashTable ()=default

  Default destructor.
- void insert (const Flight &value)

  Inserts flight into table.
- int search (const std::string &key, std::vector< Flight > &result) const

  Searches flights by airline name.
- int get_collision_count () const

  Gets collision count.

3.3.1 Detailed Description

Hash table implementation with chaining.

Definition at line 11 of file hash_table.h.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 HashTable() [1/2]

```
HashTable::HashTable ( )  [default]
```

Default constructor.

3.3.2.2 HashTable() [2/2]

```
HashTable::HashTable (
              int size_ )
```

Constructor with size initialization.

Parameters

*size_*	Number of buckets

Definition at line 5 of file hash_table.cpp.

3.3.2.3 ∼HashTable()

```
HashTable::∼HashTable ( )  [default]
```

Default destructor.

3.3.3 Member Function Documentation

3.3.3.1 get_collision_count()

```
int HashTable::get_collision_count ( ) const
```

Gets collision count.

Returns

Number of collisions

Definition at line 2 of file hash_table.cpp.

3.3.3.2 insert()

```
void HashTable::insert (
              const Flight & value )
```

Inserts flight into table.

Parameters

value	Flight to insert

Definition at line 19 of file hash_table.cpp.

3.3.3.3 search()

```
int HashTable::search (
            const std::string & key,
            std::vector< Flight > & result ) const
```

Searches flights by airline name.

Parameters

key	Airline name to search
result	Vector for results

Returns

Number of matches

Definition at line 41 of file hash_table.cpp.

The documentation for this class was generated from the following files:

- /Users/anastasiatrufanova/Desktop/lab2_data_search_algorithms/src/hash_table.h
- /Users/anastasiatrufanova/Desktop/lab2_data_search_algorithms/src/hash_table.cpp

3.4 RBTree Class Reference

Red-Black Tree implementation with balancing.

`#include <rbtree.h>`

Public Member Functions

- void Insert (Flight key)

  Public insert interface.
- int Search (Flight key, std::vector< Flight > &result)

  Public search interface.
- RBTree ()

  Default constructor.
- RBTree (RBTreeNode ∗root_)

  Constructor with root initialization.
- ∼RBTree ()

  Destructor - cleans up all nodes.
- const RBTreeNode ∗ GetRoot () const

  Gets root node.
- void SetRoot (RBTreeNode ∗root_)

  Sets new root node.

3.4.1 Detailed Description

Red-Black Tree implementation with balancing.

Definition at line 31 of file rbtree.h.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 RBTree() [1/2]

```
RBTree::RBTree ( )
```

Default constructor.

Definition at line 139 of file rbtree.cpp.

3.4.2.2 RBTree() [2/2]

```
RBTree::RBTree (
            RBTreeNode * root_ )
```

Constructor with root initialization.

Parameters

root↩ _	Node to set as initial root

Definition at line 143 of file rbtree.cpp.

3.4.2.3 ∼RBTree()

```
RBTree::∼RBTree ( )
```

Destructor - cleans up all nodes.

Definition at line 174 of file rbtree.cpp.

3.4.3 Member Function Documentation

3.4.3.1 GetRoot()

```
const RBTreeNode * RBTree::GetRoot ( ) const
```

Gets root node.

Returns

const RBTreeNode∗ Read-only root pointer

Definition at line 11 of file rbtree.cpp.

3.4.3.2 Insert()

```
void RBTree::Insert (
             Flight key )
```

Public insert interface.

Parameters

key	Flight to insert

Definition at line 178 of file rbtree.cpp.

3.4.3.3 Search()

```
int RBTree::Search (
             Flight key,
             std::vector< Flight > & result )
```

Public search interface.

Parameters

key	Flight with search criteria
result	Vector for results

Returns

Number of matches

Definition at line 183 of file rbtree.cpp.

3.4.3.4 SetRoot()

```
void RBTree::SetRoot (
            RBTreeNode * root_ )
```

Sets new root node.

Parameters

root← _	Pointer to new root

Definition at line 8 of file rbtree.cpp.

The documentation for this class was generated from the following files:

- /Users/anastasiatrufanova/Desktop/lab2_data_search_algorithms/src/rbtree.h
- /Users/anastasiatrufanova/Desktop/lab2_data_search_algorithms/src/rbtree.cpp

3.5 RBTreeNode Struct Reference

Node structure for Red-Black Tree.

```
#include <rbtree.h>
```

Public Member Functions

- RBTreeNode (Flight key_)

  Constructs a new RBTreeNode.

Public Attributes

- Flight key

  Flight data.
- std::vector< Flight > flights

  Vector for duplicate keys.
- RBTreeNode * left

  Left child pointer.
- RBTreeNode * right

  Right child pointer.
- RBTreeNode * parent

  Parent pointer.
- int color

  Node color (RED or BLACK)

3.5.1 Detailed Description

Node structure for Red-Black Tree.

Definition at line 12 of file rbtree.h.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 RBTreeNode()

```
RBTreeNode::RBTreeNode (
            Flight key_ )
```

Constructs a new RBTreeNode.

Parameters

key↩_	Flight object to initialize node

Definition at line 2 of file rbtree.cpp.

3.5.3 Member Data Documentation

3.5.3.1 color

```
int RBTreeNode::color
```

Node color (RED or BLACK)

Definition at line 18 of file rbtree.h.

3.5.3.2 flights

```
std::vector<Flight> RBTreeNode::flights
```

Vector for duplicate keys.

Definition at line 14 of file rbtree.h.

3.5.3.3 key

```
Flight RBTreeNode::key
```

Flight data.

Definition at line 13 of file rbtree.h.

3.5.3.4 left

```
RBTreeNode* RBTreeNode::left
```

Left child pointer.

Definition at line 15 of file rbtree.h.

3.5.3.5 parent

```
RBTreeNode* RBTreeNode::parent
```

Parent pointer.

Definition at line 17 of file rbtree.h.

3.5.3.6 right

```
RBTreeNode* RBTreeNode::right
```

Right child pointer.

Definition at line 16 of file rbtree.h.

The documentation for this struct was generated from the following files:

- /Users/anastasiatrufanova/Desktop/lab2_data_search_algorithms/src/rbtree.h
- /Users/anastasiatrufanova/Desktop/lab2_data_search_algorithms/src/rbtree.cpp

3.6 TreeNode Struct Reference

Node structure for Binary Search Tree.

```
#include <bstree.h>
```

Public Member Functions

- TreeNode (Flight key_)

  Constructs a new TreeNode.

Public Attributes

- Flight key

  Flight data stored in the node.
- TreeNode ∗ left

  Pointer to left child node.
- TreeNode ∗ right

  Pointer to right child node.

3.6.1 Detailed Description

Node structure for Binary Search Tree.

Definition at line 12 of file bstree.h.

3.6.2 Constructor & Destructor Documentation

3.6.2.1 TreeNode()

```
TreeNode::TreeNode (
            Flight key_ )
```

Constructs a new TreeNode.

Parameters

key← _	Flight object to initialize the node

Definition at line 2 of file bstree.cpp.

3.6.3 Member Data Documentation

3.6.3.1 key

```
Flight TreeNode::key
```

Flight data stored in the node.

Definition at line 13 of file bstree.h.

3.6.3.2 left

```
TreeNode* TreeNode::left
```

Pointer to left child node.

Definition at line 14 of file bstree.h.

3.6.3.3 right

```
TreeNode* TreeNode::right
```

Pointer to right child node.

Definition at line 15 of file bstree.h.

The documentation for this struct was generated from the following files:

- /Users/anastasiatrufanova/Desktop/lab2_data_search_algorithms/src/bstree.h
- /Users/anastasiatrufanova/Desktop/lab2_data_search_algorithms/src/bstree.cpp

Chapter 4

File Documentation

4.1 /Users/anastasiatrufanova/Desktop/lab2_data_search_algorithms/src/bstree.cpp File Reference

```
#include "bstree.h"
```

4.2 bstree.cpp

Go to the documentation of this file.

```
#include "bstree.h"
TreeNode::TreeNode(Flight key_) {
    key = key_;
}
void BSTree::SetRoot(TreeNode* root_) {
    root = root_;
}
const TreeNode* BSTree::GetRoot()const {return root;}
void BSTree::Insert(TreeNode*& node, Flight key) {
    if (node == nullptr) {
        node = new TreeNode(key);
    }
    else if (key < node->key) {
        Insert(node->left, key);
    }
    else {
        Insert(node->right, key);
    }
}

BSTree::BSTree() {
    root = nullptr;
}

BSTree::BSTree(TreeNode* root_) {
    root = root_;
}

int BSTree::Search(TreeNode* node, Flight key, std::vector<Flight>& v) {
    int count = 0;
    if (node != nullptr) {
        if (key < node->key) {
            count = Search(node->left, key, v);
        }
        else {
            if (key == node->key) {
                v.push_back(node->key);
                count = 1;
            }
            count += Search(node->right, key, v);
        }
    }
    return count;
}

void  BSTree::DeleteTree(TreeNode *& node) {
    if (node != nullptr) {

        DeleteTree(node->left);
        DeleteTree(node->right);

        delete node;
        node = nullptr;
    }
}

BSTree::~BSTree() {
    this->DeleteTree(root);
}

void BSTree::Insert(Flight key) {
    this->Insert(root, key);
}

int BSTree::Search(Flight key, std::vector<Flight>& result) {
    return this->Search(root, key, result);
}
```

4.3 /Users/anastasiatrufanova/Desktop/lab2_data_search_algorithms/src/bstree.h File Reference

```
#include <vector>
#include "flight.h"
#include <iostream>
```

Classes

- struct TreeNode

    Node structure for Binary Search Tree.

- class BSTree

    Binary Search Tree implementation for Flight objects.

4.4 bstree.h

Go to the documentation of this file.

```
#ifndef BSTREE_H
#define BSTREE_H

#include <vector>
#include "flight.h"
#include <iostream>

struct TreeNode {
    Flight key;
    TreeNode* left;
    TreeNode* right;

    TreeNode(Flight key_);
};

class BSTree {
private:
    TreeNode* root;

    void Insert(TreeNode *& node, Flight key);

    int Search(TreeNode* node, Flight key, std::vector<Flight>& result);

    void DeleteTree(TreeNode *& node);

public:
    void Insert(Flight key);

    int Search(Flight key, std::vector<Flight>& result);

    BSTree();

    BSTree(TreeNode* root_);

    ~BSTree();

    const TreeNode* GetRoot() const;

    void SetRoot(TreeNode* root_);
};

#endif
```

4.5 /Users/anastasiatrufanova/Desktop/lab2_data_search_algorithms/src/flight.cpp File Reference

```
#include "flight.h"
```

Functions

- std::ostream & operator<< (std::ostream &os, const Flight &flight)

4.5.1 Function Documentation

4.5.1.1 operator<<()

```
std::ostream & operator<< (
            std::ostream & os,
            const Flight & flight )
```

Parameters

os	Output stream
flight	Flight to output

Returns

Reference to output stream

Definition at line 79 of file flight.cpp.

4.6 flight.cpp

Go to the documentation of this file.

```
#include "flight.h"


Flight::Flight() {
    flight_number = "";
    airline = "";
    arrival_date = "";
    arrival_time = "";
    passengers = 0;
}


Flight::Flight(std::string flight_number_, std::string airline_, std::string arrival_date_,
std::string arrival_time_, int passengers_) {
    flight_number = flight_number_;
    airline = airline_;
    arrival_date = arrival_date_;
    arrival_time = arrival_time_;
    passengers = passengers_;
}

std::string Flight::Get_flight_number()const{
    return flight_number;
}
std::string Flight::Get_airline()const{
    return airline;
}
std::string Flight::Get_arrival_date()const{
    return arrival_date;
}

std::string Flight::Get_arrival_time()const{
    return arrival_time;
}

int Flight::Get_passengers()const{
    return passengers;
}

void Flight::Set_flight_number(std::string flight_number_) {
    flight_number = flight_number_;
}
void Flight::Set_airline(std::string airline_) {
    airline = airline_;
}
void Flight::Set_arrival_date(std::string arrival_date_) {
    arrival_date = arrival_date_;
}

void Flight::Set_arrival_time(std::string arrival_time_) {
    arrival_time = arrival_time_;
}

void Flight::Set_passengers(int passengers_) {
    passengers = passengers_;
}


bool Flight::operator>(const Flight& other)const {
    return airline > other.airline;
}

bool Flight::operator>=(const Flight& other)const {
    return  airline >= other.airline;
}

bool Flight::operator<(const Flight& other)const {
    return airline < other.airline;
}

bool Flight::operator<=(const Flight& other)const {
    return airline <= other.airline;
}

bool Flight::operator==(const Flight& other)const {
    return airline == other.airline;
}


std::ostream& operator«(std::ostream& os, const Flight& flight) {
    os « flight.flight_number « " " « flight.airline « " " « flight.arrival_date « " " «
flight.arrival_time « " " « flight.passengers;
```

```
    return os;
}

```

4.7 /Users/anastasiatrufanova/Desktop/lab2_data_search_algorithms/src/flight.h File Reference

```
#include <string>
#include <iostream>
```

Classes

- class Flight

  Represents flight information.

4.8 flight.h

Go to the documentation of this file.

```
#ifndef FLIGHT_H
#define FLIGHT_H
#include <string>
#include <iostream>

class Flight {
private:
    std::string flight_number;
    std::string airline;
    std::string arrival_date;
    std::string arrival_time;
    int passengers;

public:
    Flight();

    Flight(std::string flight_number_, std::string airline_,
           std::string arrival_date_, std::string arrival_time_, int passengers_);

    ~Flight() = default;

    // Accessor methods
    std::string Get_flight_number() const;
    std::string Get_airline() const;
    std::string Get_arrival_date() const;
    std::string Get_arrival_time() const;
    int Get_passengers() const;

    // Mutator methods
    void Set_flight_number(std::string);
    void Set_airline(std::string);
    void Set_arrival_date(std::string);
    void Set_arrival_time(std::string);
    void Set_passengers(int);

    // Comparison operators
    bool operator>(const Flight& other) const;
    bool operator<(const Flight& other) const;
    bool operator<=(const Flight& other) const;
    bool operator>=(const Flight& other) const;
    bool operator==(const Flight& other) const;

    Flight& operator=(const Flight& other) = default;

    friend std::ostream& operator«(std::ostream& os, const Flight& flight);
};

#endif
```

4.9 /Users/anastasiatrufanova/Desktop/lab2_data_search_algorithms/src/hash_table.cpp File Reference

```
#include "hash_table.h"
```

4.10 hash_table.cpp

Go to the documentation of this file.

```
#include "hash_table.h"
int HashTable::get_collision_count()const {
    return collision_count;
}
HashTable::HashTable(int size_) {
    size = size_;
    table.resize(size_);
}

int HashTable::hash_function(const std::string& key)const {
    unsigned long h = 0;
    const unsigned long P = 31;
    for (unsigned char c :  key) {
        h = h * P + c;
    }
    return h % size;

}
void HashTable::insert(const Flight& value) {
    std::string key = value.Get_airline();
    int h = hash_function(key);
    bool key_exists = false;
    bool other_key_exists = false;

    for (const Flight& flight :  table[h]) {
        if (flight.Get_airline() == key) {
            key_exists = true;
        } else {
            other_key_exists = true;
        }
    }


    if (!key_exists && other_key_exists) {
        collision_count++;
    }
    table[h].push_back(value);

}

int HashTable::search(const std::string& key, std::vector<Flight>& result)const {
    int h = hash_function(key);
    for(int i = 0; i < (int) table[h].size(); ++i) {
        if(table[h][i].Get_airline() == key) result.push_back(table[h][i]);
    }
    return result.size();


}

```

4.11 /Users/anastasiatrufanova/Desktop/lab2_data_search_algorithms/src/hash_table.h File Reference

```
#include <string>
#include "flight.h"
#include <vector>
```

Classes

- class HashTable

    Hash table implementation with chaining.

4.12 hash_table.h

Go to the documentation of this file.

```
#ifndef HASH_TABLE_H
#define HASH_TABLE_H
#include <string>
#include "flight.h"
#include <vector>

class HashTable {
private:
    std::vector<std::vector<Flight» table;
    int size = 0;
    int collision_count = 0;

    int hash_function(const std::string& key) const;

public:
    HashTable() = default;

    HashTable(int size_);

    ~HashTable() = default;

    void insert(const Flight& value);

    int search(const std::string& key, std::vector<Flight>& result) const;

    int get_collision_count() const;
};
#endif
```

4.13 /Users/anastasiatrufanova/Desktop/lab2_data_search_algorithms/src/rbtree.cpp File Reference

```
#include "rbtree.h"
```

4.14 rbtree.cpp

Go to the documentation of this file.

```
#include "rbtree.h"
RBTreeNode::RBTreeNode(Flight key_) {
    key = key_;
    color = RED;
    flights.push_back(key_);
}

void RBTree::SetRoot(RBTreeNode* root_) {
    root = root_;
}
const RBTreeNode* RBTree::GetRoot()const {return root;}

void RBTree::rotateLeft(RBTreeNode*& node) {
    RBTreeNode* rightChild = node->right;
    if(rightChild) {
        RBTreeNode* grandparent = node->parent;
        RBTreeNode* grandson = rightChild->left;


    node->right = grandson;
    if (grandson)
        grandson->parent = node;


    rightChild->left = node;
    rightChild->parent = grandparent;


    if (!grandparent) {
        root = rightChild;
    } else if (node == grandparent->left) {
        grandparent->left = rightChild;
    } else {
        grandparent->right = rightChild;
    }

    node->parent = rightChild;

    node = rightChild;


    }
}

void RBTree::rotateRight(RBTreeNode*& node) {
    RBTreeNode* leftChild = node->left;
    if(leftChild) {
        RBTreeNode* grandparent = node->parent;
        RBTreeNode* grandson = leftChild->right;


    node->left = grandson;
    if (grandson)
        grandson->parent = node;


    leftChild->right = node;
    leftChild->parent = grandparent;


    if (!grandparent) {
        root = leftChild;
    } else if (node == grandparent->left) {
        grandparent->left = leftChild;
    } else {
        grandparent->right = leftChild;
    }

    node->parent = leftChild;

    node = leftChild;


    }

}

void RBTree::Balance(RBTreeNode*& node) {
    while (node != root && node->parent && node->parent->color == RED && node->parent->parent) {
        RBTreeNode* parent = node->parent;
        RBTreeNode* grandparent = parent->parent;


        RBTreeNode* uncle = (parent == grandparent->left) ?  grandparent->right :  grandparent->left;

        if (uncle && uncle->color == RED) {
            // Case 1:  Uncle is RED -- recolor
            parent->color = BLACK;
            uncle->color = BLACK;
            grandparent->color = RED;
            node = grandparent;
        } else {
            if (parent == grandparent->left) {
                if (node == parent->right) {
                    // Case 2:  Left-Right -- rotate left on parent
                    rotateLeft(parent);
                    node = parent;
                    parent = node->parent;
                }
                // Case 3:  Left-Left -- rotate right on grandparent
                parent->color = BLACK;
                grandparent->color = RED;
                rotateRight(grandparent);
            } else {
                if (node == parent->left) {
                    // Case 2 mirror:  Right-Left
                    rotateRight(parent);
                    node = parent;
                    parent = node->parent;
                }
                // Case 3 mirror:  Right-Right
                parent->color = BLACK;
                grandparent->color = RED;
                rotateLeft(grandparent);
            }
        }
    }
    root->color = BLACK;
}


RBTreeNode* RBTree::Insert(RBTreeNode*& node, Flight key, RBTreeNode* parent) {
    if (node == nullptr) {
        node = new RBTreeNode(key);
        node->parent = parent;
        return node;
    }
    else if (key < node->key) {
        return Insert(node->left, key, node);
    }
    else if (key > node->key) {
        return Insert(node->right, key, node);
    } else {
        node->flights.push_back(key);
        return node;
    }
}

RBTree::RBTree() {
    root = nullptr;
}

RBTree::RBTree(RBTreeNode* root_) {
    root = root_;
    root->color = BLACK;
}

int RBTree::Search(RBTreeNode* node, Flight key, std::vector<Flight>& v) {
    int count = 0;
    if (node != nullptr){
        if (key < node->key) {
            count = Search(node->left, key, v);
        } else if (key > node->key) {
            count = Search(node->right, key, v);
        } else {
            v.insert(v.end(), node->flights.begin(), node->flights.end());
            count = node->flights.size();
        }
    }
    return count;
}

void  RBTree::DeleteTree(RBTreeNode *& node) {
    if (node != nullptr) {

        DeleteTree(node->left);
        DeleteTree(node->right);

        delete node;
        node = nullptr;
    }
}

RBTree::~RBTree() {
    this->DeleteTree(root);
}

void RBTree::Insert(Flight key) {
    RBTreeNode* inserted = this->Insert(root, key, nullptr);
    Balance(inserted);
}

int RBTree::Search(Flight key, std::vector<Flight>& result) {
    return this->Search(root, key, result);
}
```

4.15 /Users/anastasiatrufanova/Desktop/lab2_data_search_↩ algorithms/src/rbtree.h File Reference

```
#include "flight.h"
```

Classes

- struct RBTreeNode

  Node structure for Red-Black Tree.
- class RBTree

  Red-Black Tree implementation with balancing.

Macros

- #define BLACK 0

  Black node color constant.
- #define RED 1

  Red node color constant.

4.15.1 Macro Definition Documentation

4.15.1.1 BLACK

```
#define BLACK 0
```

Black node color constant.

Definition at line 5 of file rbtree.h.

4.15.1.2 RED

```
#define RED 1
```

Red node color constant.

Definition at line 6 of file rbtree.h.

4.16 rbtree.h

Go to the documentation of this file.

```
#ifndef RBTREE_H
#define RBTREE_H
#include "flight.h"

#define BLACK 0
#define RED 1

struct RBTreeNode {
    Flight key;
    std::vector<Flight> flights;
    RBTreeNode* left;
    RBTreeNode* right;
    RBTreeNode* parent;
    int color;

    RBTreeNode(Flight key_);
};

class RBTree {
private:
    RBTreeNode* root;

    RBTreeNode* Insert(RBTreeNode *& node, Flight key, RBTreeNode* parent);

    int Search(RBTreeNode* node, Flight key, std::vector<Flight>& result);

    void DeleteTree(RBTreeNode *& node);

    void Balance(RBTreeNode*& node);

    void rotateLeft(RBTreeNode*& node);

    void rotateRight(RBTreeNode*& node);

public:
    void Insert(Flight key);

    int Search(Flight key, std::vector<Flight>& result);

    RBTree();

    RBTree(RBTreeNode* root_);

    ~RBTree();

    const RBTreeNode* GetRoot() const;

    void SetRoot(RBTreeNode* root_);
};
#endif
```

4.17 /Users/anastasiatrufanova/Desktop/lab2_data_search_algorithms/src/search.cpp File Reference

```
#include "search.h"
#include <fstream>
#include "flight.h"
#include "bstree.h"
#include "rbtree.h"
#include <vector>
#include "hash_table.h"
#include <chrono>
#include <map>
```

Functions

- int parseCSV (std::string filename, std::vector< Flight > &result)

Parses CSV file into flight vector.
- int linear_search (const std::vector< Flight > &flights, std::vector< Flight > &result)

  Linear search implementation.
- BSTree ∗ binary_insert (const std::vector< Flight > &flights)

  Builds BST from flights.
- RBTree ∗ rb_insert (const std::vector< Flight > &flights)

  Builds RBTree from flights.
- HashTable ∗ hash_table_insert (const std::vector< Flight > &flights, int &collisions)

  Builds HashTable from flights.
- std::multimap< std::string, Flight > ∗ multimap_insert (const std::vector< Flight > &flights)

  Builds multimap from flights.
- int main ()

4.17.1 Function Documentation

4.17.1.1 binary_insert()

```
BSTree * binary_insert (
            const std::vector< Flight > & flights )
```

Builds BST from flights.

Parameters

flights	Flight vector

Returns

Pointer to constructed BST

Definition at line 62 of file search.cpp.

4.17.1.2 hash_table_insert()

```
HashTable * hash_table_insert (
            const std::vector< Flight > & flights,
            int & collisions )
```

Builds HashTable from flights.

Parameters

flights	Flight vector
collisions	Output collision count

Returns

Pointer to constructed HashTable

Definition at line 90 of file search.cpp.

4.17.1.3 linear_search()

```
int linear_search (
            const std::vector< Flight > & flights,
            std::vector< Flight > & result )
```

Linear search implementation.

Parameters

flights	Flight vector to search
result	Vector for results

Returns

Number of matches

Definition at line 50 of file search.cpp.

4.17.1.4 main()

```
int main ( )
```

Definition at line 109 of file search.cpp.

4.17.1.5 multimap_insert()

```
std::multimap< std::string, Flight > * multimap_insert (
            const std::vector< Flight > & flights )
```

Builds multimap from flights.

Parameters

flights	Flight vector

Returns

Pointer to constructed multimap

Definition at line 101 of file search.cpp.

4.17.1.6 parseCSV()

```
int parseCSV (
            std::string filename,
            std::vector< Flight > & result )
```

Parses CSV file into flight vector.

Parameters

filename	CSV file path
result	Vector to store flights

Returns

0 on success, error code otherwise

Definition at line 12 of file search.cpp.

4.17.1.7 rb_insert()

```
RBTree * rb_insert (
            const std::vector< Flight > & flights )
```

Builds RBTree from flights.

Parameters

flights	Flight vector

Returns

Pointer to constructed RBTree

Definition at line 76 of file search.cpp.

4.18 search.cpp

Go to the documentation of this file.

```
#include "search.h"
#include <fstream>
#include "flight.h"
#include "bstree.h"
#include "rbtree.h"
#include <vector>
#include "hash_table.h"
#include <chrono>
#include <map>


int parseCSV(std::string filename, std::vector<Flight> &result) {
    int error = 0;
    std::ifstream in(filename);
    std::string line, flight_number_, airline_, arrival_date_, arrival_time_;
    int i1 = 0, i2 = 0, i3 = 0, i4 = 0, passengers_;

    if (in.is_open()) {
        while (std::getline(in, line)) {
            i1 = i2 = i3 = i4 = 0;
            for (int i = 0; i < (int)line.size(); i++) {
                if (line[i] == ',') {
                    if (i1 == 0) i1 = i;
                    else if (i2 == 0) i2 = i;
                    else if (i3 == 0) i3 = i;
                    else if (i4 == 0) i4 = i;
                }
            }
            flight_number_ = line.substr(0, i1);
            airline_ = line.substr(i1 + 1, i2 - i1 - 1);
            arrival_date_ = line.substr(i2 + 1, i3 - i2 - 1);
            arrival_time_ = line.substr(i3 + 1, i4 - i3 - 1);

            try {
                passengers_ = std::stoi(line.substr(i4 + 1, line.size() - i4 - 1));
            } catch (...)  {
                passengers_ = 0;
            }

            Flight f(flight_number_, airline_, arrival_date_, arrival_time_, passengers_);
            result.push_back(f);
        }
        in.close();
    } else {
        error = 1;
    }
    return error;
}

int linear_search(const std::vector<Flight>& flights, std::vector<Flight>& result) {
    int found = 0;
    for (int i = 0; i < (int)flights.size(); ++i) {
        if (flights[i].Get_airline() == KEY) {
            result.push_back(flights[i]);
            found++;
        }
    }
    return found;
}


BSTree* binary_insert(const std::vector<Flight>& flights) {
    if (flights.empty()) return nullptr;

    TreeNode* root = new TreeNode(flights[0]);
    BSTree* tree = new BSTree(root);

    for (int i = 1; i < (int)flights.size(); i++) {
        tree->Insert(flights[i]);
    }

    return tree;
}


RBTree* rb_insert(const std::vector<Flight>& flights) {
    if (flights.empty()) return nullptr;

    RBTreeNode* root = new RBTreeNode(flights[0]);
    RBTree* tree = new RBTree(root);

    for (int i = 1; i < (int)flights.size(); i++) {
        tree->Insert(flights[i]);
    }

    return tree;
}
```

```


HashTable* hash_table_insert(const std::vector<Flight>& flights, int& collisions) {
    HashTable* table = new HashTable(7);

    for (int i = 0; i < (int)flights.size(); i++) {
        table->insert(flights[i]);
    }

    collisions = table->get_collision_count();
    return table;
}

std::multimap<std::string, Flight>* multimap_insert(const std::vector<Flight>& flights) {
    auto* mmap = new std::multimap<std::string, Flight>();
    for (int i = 0; i < (int)flights.size(); i++) {
        mmap->insert({flights[i].Get_airline(), flights[i]});
    }
    return mmap;
}

int main() {
    std::vector<std::string> filenames = {
        "100.csv", "200.csv", "500.csv", "1000.csv", "2000.csv",
        "5000.csv", "10000.csv", "20000.csv", "50000.csv", "75000.csv"
    };

    std::string path = "/Users/anastasiatrufanova/Desktop/lab2_data_search_algorithms/data/";
    std::ofstream
out("/Users/anastasiatrufanova/Desktop/lab2_data_search_algorithms/plt/results.csv");

    if (!out.is_open()) {
        std::cerr « "Failed to open output file!" « std::endl;
    } else {

    out « "Array Size,Linear,Binary,RBTree,HashTable,Collisions,Multimap\n";

    for (const auto& filename :  filenames) {
        std::string full_path = path + filename;
        std::cout « "Processing:  " « full_path « std::endl;

        std::vector<Flight> flights;
        if (parseCSV(full_path, flights) != 0) {
            std::cerr « "Error parsing:  " « full_path « std::endl;
            continue;
        }

        std::vector<Flight> res1, res2, res3, res4, res5;
        int collisions = 0;

        auto t1_start = std::chrono::high_resolution_clock::now();
        linear_search(flights, res1);
        auto t1_end = std::chrono::high_resolution_clock::now();
        double time_linear = std::chrono::duration<double, std::milli>(t1_end - t1_start).count();


        BSTree* bst_tree = binary_insert(flights);

        auto t2_start = std::chrono::high_resolution_clock::now();
        if (bst_tree) {
            bst_tree->Search(Flight("", KEY, "", "", 0), res2);
        }
        auto t2_end = std::chrono::high_resolution_clock::now();
        double time_binary = std::chrono::duration<double, std::milli>(t2_end - t2_start).count();
        delete bst_tree;


        RBTree* rb_tree = rb_insert(flights);

        auto t3_start = std::chrono::high_resolution_clock::now();
        if (rb_tree) {
            rb_tree->Search(Flight("", KEY, "", "", 0), res3);
        }
        auto t3_end = std::chrono::high_resolution_clock::now();
        double time_rbtree = std::chrono::duration<double, std::milli>(t3_end - t3_start).count();
        delete rb_tree;

        HashTable* hash_table = hash_table_insert(flights, collisions);
        auto t4_start = std::chrono::high_resolution_clock::now();
        if (hash_table) {
            hash_table->search(KEY, res4);
        }
        auto t4_end = std::chrono::high_resolution_clock::now();
        double time_hash = std::chrono::duration<double, std::milli>(t4_end - t4_start).count();
        delete hash_table;


        auto* mmap = multimap_insert(flights);

        auto t5_start = std::chrono::high_resolution_clock::now();
        if (mmap) {
            auto range = mmap->equal_range(KEY);
            for (auto it = range.first; it != range.second; ++it) {
                res5.push_back(it->second);
            }
        }
        auto t5_end = std::chrono::high_resolution_clock::now();
        double time_multimap = std::chrono::duration<double, std::milli>(t5_end - t5_start).count();
        delete mmap;


        out « flights.size() « ","
            « time_linear « "," « time_binary « "," « time_rbtree « ","
            « time_hash « "," « collisions « "," « time_multimap « "\n";
    }

    out.close();
    }

    return 0;
}
```

4.19 /Users/anastasiatrufanova/Desktop/lab2_data_search_algorithms/src/search.h File Reference

```
#include "flight.h"
#include "bstree.h"
#include "rbtree.h"
#include <map>
#include "hash_table.h"
```

Macros

- #define KEY "Ural Airlines"

  Default search key.

Functions

- int parseCSV (std::string filename, std::vector< Flight > &result)

  Parses CSV file into flight vector.
- int linear_search (const std::vector< Flight > &flights, std::vector< Flight > &result)

  Linear search implementation.
- BSTree ∗ binary_insert (const std::vector< Flight > &flights)

  Builds BST from flights.
- RBTree ∗ rb_insert (const std::vector< Flight > &flights)

  Builds RBTree from flights.
- HashTable ∗ hash_table_insert (const std::vector< Flight > &flights, int &collisions)

  Builds HashTable from flights.
- std::multimap< std::string, Flight > ∗ multimap_insert (const std::vector< Flight > &flights)

  Builds multimap from flights.

4.19.1 Macro Definition Documentation

4.19.1.1 KEY

```
#define KEY "Ural Airlines"
```

Default search key.

Definition at line 9 of file search.h.

4.19.2 Function Documentation

4.19.2.1 binary_insert()

```
BSTree * binary_insert (
            const std::vector< Flight > & flights )
```

Builds BST from flights.

Parameters

flights	Flight vector

Returns

Pointer to constructed BST

Definition at line 62 of file search.cpp.

4.19.2.2 hash_table_insert()

```
HashTable * hash_table_insert (
            const std::vector< Flight > & flights,
            int & collisions )
```

Builds HashTable from flights.

Parameters

flights	Flight vector
collisions	Output collision count

Returns

Pointer to constructed HashTable

Definition at line 90 of file search.cpp.

4.19.2.3 linear_search()

```
int linear_search (
            const std::vector< Flight > & flights,
            std::vector< Flight > & result )
```

Linear search implementation.

Parameters

flights	Flight vector to search
result	Vector for results

Returns

Number of matches

Definition at line 50 of file search.cpp.

4.19.2.4 multimap_insert()

```
std::multimap< std::string, Flight > * multimap_insert (
            const std::vector< Flight > & flights )
```

Builds multimap from flights.

Parameters

flights	Flight vector

Returns

Pointer to constructed multimap

Definition at line 101 of file search.cpp.

4.19.2.5 parseCSV()

```
int parseCSV (
            std::string filename,
            std::vector< Flight > & result )
```

Parses CSV file into flight vector.

Parameters

filename	CSV file path
result	Vector to store flights

Returns

0 on success, error code otherwise

Definition at line 12 of file search.cpp.

4.19.2.6 rb_insert()

```
RBTree * rb_insert (
            const std::vector< Flight > & flights )
```

Builds RBTree from flights.

Parameters

flights	Flight vector

Returns

Pointer to constructed RBTree

Definition at line 76 of file search.cpp.

4.20 search.h

Go to the documentation of this file.

```
#ifndef SEARCH_H
#define SEARCH_H
#include"flight.h"
#include "bstree.h"
#include "rbtree.h"
#include <map>
#include "hash_table.h"

#define KEY "Ural Airlines"

int parseCSV(std::string filename, std::vector<Flight> &result);

int linear_search(const std::vector<Flight>& flights, std::vector<Flight>& result);

BSTree* binary_insert(const std::vector<Flight>& flights);

RBTree* rb_insert(const std::vector<Flight>& flights);

HashTable* hash_table_insert(const std::vector<Flight>& flights, int& collisions);

std::multimap<std::string, Flight>* multimap_insert(const std::vector<Flight>& flights);

#endif
```

Index