Отчет к лабораторной работе №2

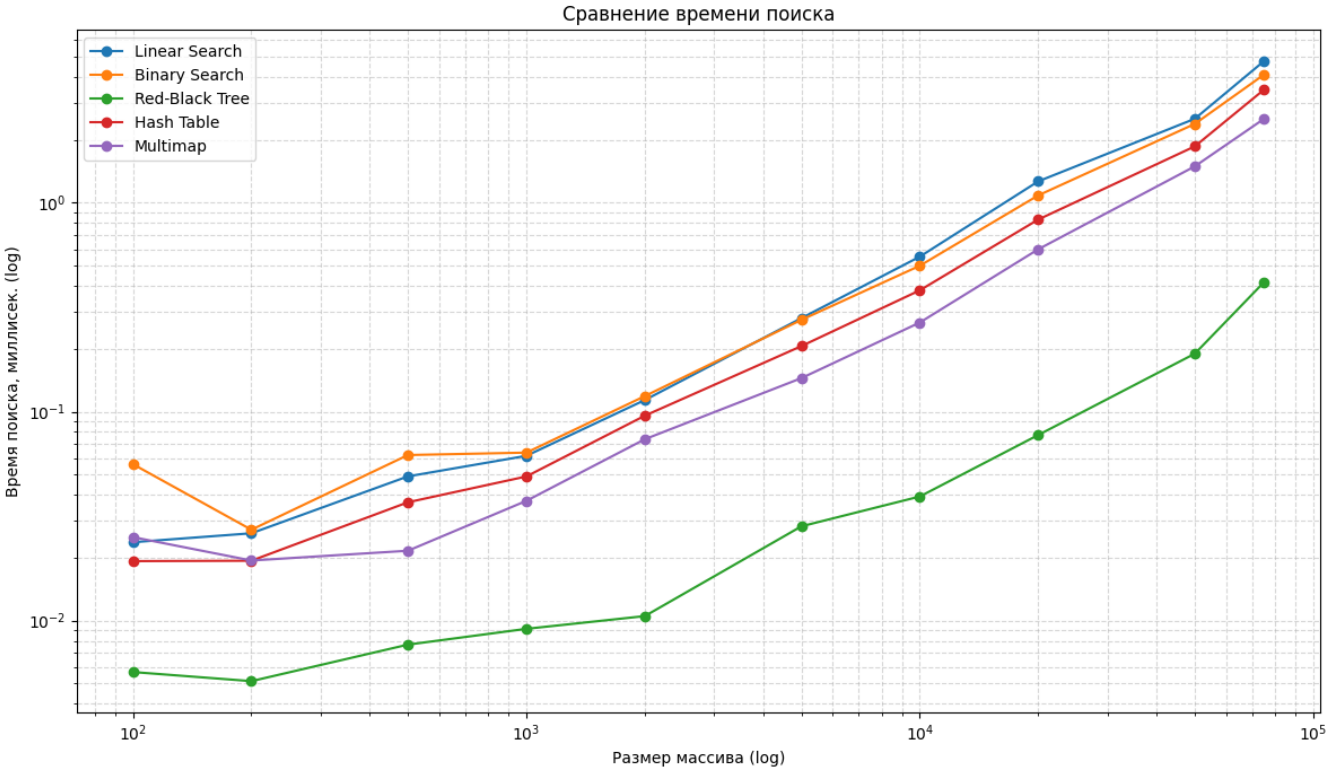
Вариант №21

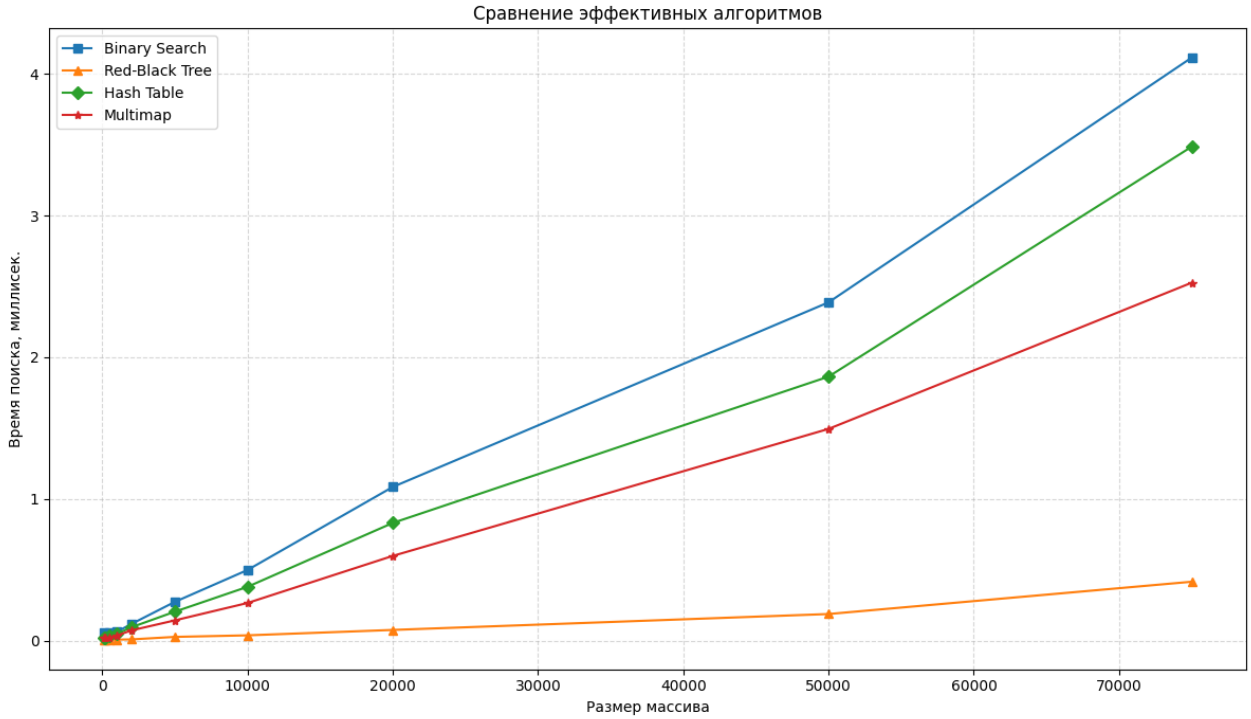
Труфанова Анастасия СКБ222

Ссылка на исходный код программы в репозитории

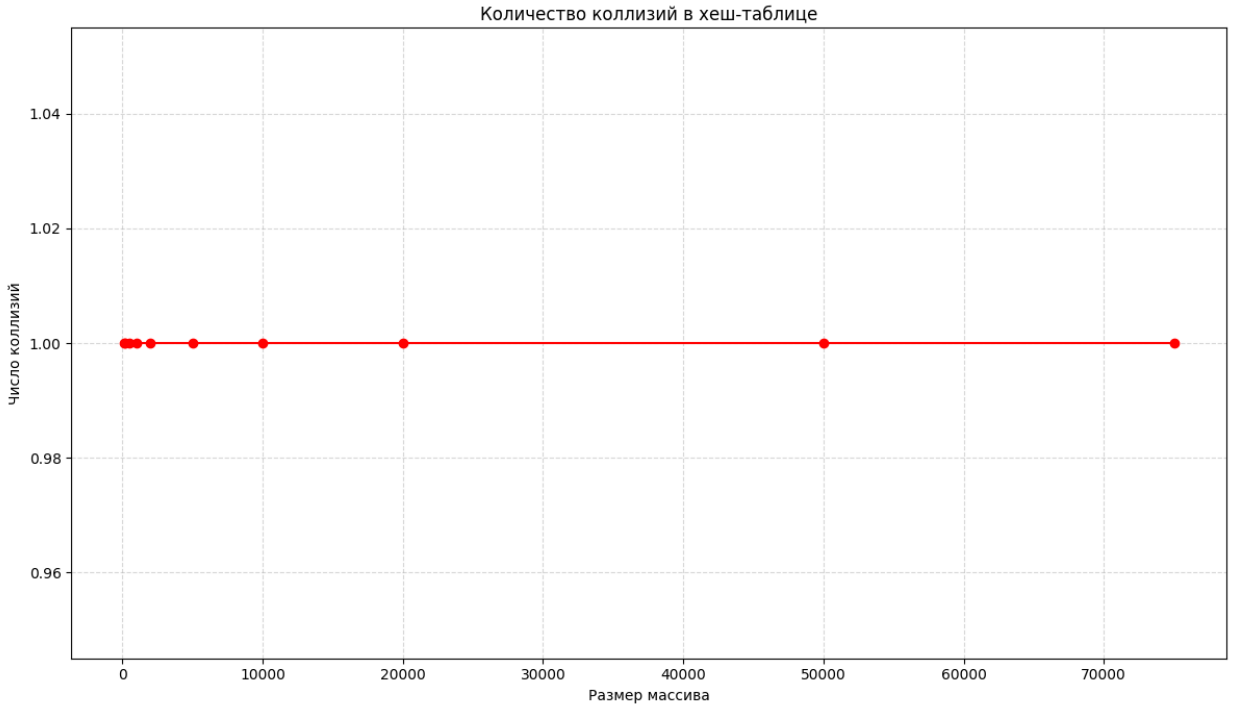
https://github.com/neko-nyashka/lab2_data_search_algorithms/tree/develop/src

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.

```
00001 #include "bstree.h"
00002 TreeNode::TreeNode(Flight key_) {
00003     key = key_;
00004 }
00005 void BSTree::SetRoot(TreeNode* root_) {
00006     root = root_;
00007 }
00008 const TreeNode* BSTree::GetRoot()const {return root;}
00009 void BSTree::Insert(TreeNode*& node, Flight key) {
00010     if (node == nullptr) {
00011         node = new TreeNode(key);
00012     }
00013     else if (key < node->key) {
00014         Insert(node->left, key);
00015     }
00016     else {
00017         Insert(node->right, key);
00018     }
00019 }
00020
00021 BSTree::BSTree() {
00022     root = nullptr;
00023 }
00024
00025 BSTree::BSTree(TreeNode* root_) {
00026     root = root_;
00027 }
00028
00029 int BSTree::Search(TreeNode* node, Flight key, std::vector<Flight>& v) {
00030     int count = 0;
00031     if (node != nullptr) {
00032         if (key < node->key) {
00033             count = Search(node->left, key, v);
00034         }
00035         else {
00036             if (key == node->key) {
00037                 v.push_back(node->key);
00038                 count = 1;
00039             }
00040             count += Search(node->right, key, v);
00041         }
```

```
00042     }
00043     return count;
00044 }
00045
00046 void  BSTree::DeleteTree(TreeNode *& node) {
00047     if (node != nullptr) {
00048
00049         DeleteTree(node->left);
00050         DeleteTree(node->right);
00051
00052         delete node;
00053         node = nullptr;
00054     }
00055 }
00056
00057 BSTree::~BSTree() {
00058     this->DeleteTree(root);
00059 }
00060
00061 void BSTree::Insert(Flight key) {
00062     this->Insert(root, key);
00063 }
00064
00065 int BSTree::Search(Flight key, std::vector<Flight>& result) {
00066     return this->Search(root, key, result);
00067 }
```

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.

```
00001 #ifndef BSTREE_H
00002 #define BSTREE_H
00003
00004 #include <vector>
00005 #include "flight.h"
00006 #include <iostream>
00007
00012 struct TreeNode {
00013     Flight key;
00014     TreeNode* left;
00015     TreeNode* right;
00016
00021     TreeNode(Flight key_);
00022 };
00023
00028 class BSTree {
00029 private:
00030     TreeNode* root;
00031
00037     void Insert(TreeNode *& node, Flight key);
00038
```

```
00046     int Search(TreeNode* node, Flight key, std::vector<Flight>& result);
00047
00052     void DeleteTree(TreeNode *& node);
00053
00054 public:
00059     void Insert(Flight key);
00060
00067     int Search(Flight key, std::vector<Flight>& result);
00068
00072     BSTree();
00073
00078     BSTree(TreeNode* root_);
00079
00083     ~BSTree();
00084
00089     const TreeNode* GetRoot() const;
00090
00095     void SetRoot(TreeNode* root_);
00096 };
00097
00098 #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.

```
00001 #include "flight.h"
00002
00003
00004 Flight::Flight() {
00005     flight_number = "";
00006     airline = "";
00007     arrival_date = "";
00008     arrival_time = "";
00009     passengers = 0;
00010 }
00011
00012
00013 Flight::Flight(std::string flight_number_, std::string airline_, std::string arrival_date_,
      std::string arrival_time_, int passengers_) {
00014     flight_number = flight_number_;
00015     airline = airline_;
00016     arrival_date = arrival_date_;
00017     arrival_time = arrival_time_;
00018     passengers = passengers_;
00019 }
00020
00021 std::string Flight::Get_flight_number()const{
00022     return flight_number;
00023 }
00024 std::string Flight::Get_airline()const{
00025     return airline;
00026 }
00027 std::string Flight::Get_arrival_date()const{
00028     return arrival_date;
00029 }
00030
00031 std::string Flight::Get_arrival_time()const{
00032     return arrival_time;
00033 }
00034
00035 int Flight::Get_passengers()const{
00036     return passengers;
00037 }
00038
00039 void Flight::Set_flight_number(std::string flight_number_) {
00040     flight_number = flight_number_;
00041 }
00042 void Flight::Set_airline(std::string airline_) {
00043     airline = airline_;
00044 }
00045 void Flight::Set_arrival_date(std::string arrival_date_) {
00046     arrival_date = arrival_date_;
00047 }
00048
00049 void Flight::Set_arrival_time(std::string arrival_time_) {
00050     arrival_time = arrival_time_;
00051 }
00052
00053 void Flight::Set_passengers(int passengers_) {
00054     passengers = passengers_;
00055 }
00056
00057
00058 bool Flight::operator>(const Flight& other)const {
00059     return airline > other.airline;
00060 }
00061
00062 bool Flight::operator>=(const Flight& other)const {
00063     return  airline >= other.airline;
00064 }
00065
00066 bool Flight::operator<(const Flight& other)const {
00067     return airline < other.airline;
00068 }
00069
00070 bool Flight::operator<=(const Flight& other)const {
00071     return airline <= other.airline;
00072 }
00073
00074 bool Flight::operator==(const Flight& other)const {
00075     return airline == other.airline;
00076 }
00077
00078
00079 std::ostream& operator«(std::ostream& os, const Flight& flight) {
00080     os « flight.flight_number « " " « flight.airline « " " « flight.arrival_date « " " «
      flight.arrival_time « " " « flight.passengers;
```

```
00081     return os;
00082 }
00083
```

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.

```
00001 #ifndef FLIGHT_H
00002 #define FLIGHT_H
00003 #include <string>
00004 #include <iostream>
00005
00010 class Flight {
00011 private:
00012     std::string flight_number;
00013     std::string airline;
00014     std::string arrival_date;
00015     std::string arrival_time;
00016     int passengers;
00017
00018 public:
00022     Flight();
00023
00032     Flight(std::string flight_number_, std::string airline_,
00033            std::string arrival_date_, std::string arrival_time_, int passengers_);
00034
00038     ~Flight() = default;
00039
00040     // Accessor methods
00041     std::string Get_flight_number() const;
00042     std::string Get_airline() const;
00043     std::string Get_arrival_date() const;
00044     std::string Get_arrival_time() const;
00045     int Get_passengers() const;
00046
00047     // Mutator methods
00048     void Set_flight_number(std::string);
00049     void Set_airline(std::string);
00050     void Set_arrival_date(std::string);
00051     void Set_arrival_time(std::string);
00052     void Set_passengers(int);
00053
00054     // Comparison operators
00055     bool operator>(const Flight& other) const;
00056     bool operator<(const Flight& other) const;
00057     bool operator<=(const Flight& other) const;
00058     bool operator>=(const Flight& other) const;
00059     bool operator==(const Flight& other) const;
00060
00066     Flight& operator=(const Flight& other) = default;
00067
00074     friend std::ostream& operator<<(std::ostream& os, const Flight& flight);
00075 };
00076
00077 #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.

```
00001 #include "hash_table.h"
00002 int HashTable::get_collision_count()const {
00003     return collision_count;
00004 }
00005 HashTable::HashTable(int size_) {
00006     size = size_;
00007     table.resize(size_);
00008 }
00009
00010 int HashTable::hash_function(const std::string& key)const {
00011     unsigned long h = 0;
00012     const unsigned long P = 31;
00013     for (unsigned char c :  key) {
00014         h = h * P + c;
00015     }
00016     return h % size;
00017
00018 }
00019 void HashTable::insert(const Flight& value) {
00020     std::string key = value.Get_airline();
00021     int h = hash_function(key);
00022     bool key_exists = false;
00023     bool other_key_exists = false;
00024
00025     for (const Flight& flight :  table[h]) {
00026         if (flight.Get_airline() == key) {
00027             key_exists = true;
00028         } else {
00029             other_key_exists = true;
00030         }
00031     }
00032
00033
00034     if (!key_exists && other_key_exists) {
00035         collision_count++;
00036     }
00037     table[h].push_back(value);
00038
00039 }
00040
00041 int HashTable::search(const std::string& key, std::vector<Flight>& result)const {
00042     int h = hash_function(key);
00043     for(int i = 0; i < (int) table[h].size(); ++i) {
00044         if(table[h][i].Get_airline() == key) result.push_back(table[h][i]);
00045     }
00046     return result.size();
00047
00048
00049 }
00050
```

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.

```
00001 #ifndef HASH_TABLE_H
00002 #define HASH_TABLE_H
00003 #include <string>
00004 #include "flight.h"
00005 #include <vector>
00006
00011 class HashTable {
00012 private:
00013     std::vector<std::vector<Flight» table;
00014     int size = 0;
00015     int collision_count = 0;
00016
00022     int hash_function(const std::string& key) const;
00023
00024 public:
00028     HashTable() = default;
00029
00034     HashTable(int size_);
00035
00039     ~HashTable() = default;
00040
00045     void insert(const Flight& value);
00046
00053     int search(const std::string& key, std::vector<Flight>& result) const;
00054
00059     int get_collision_count() const;
00060 };
00061 #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.

```
00001 #include "rbtree.h"
00002 RBTreeNode::RBTreeNode(Flight key_) {
00003     key = key_;
00004     color = RED;
00005     flights.push_back(key_);
00006 }
00007
00008 void RBTree::SetRoot(RBTreeNode* root_) {
00009     root = root_;
00010 }
00011 const RBTreeNode* RBTree::GetRoot()const {return root;}
00012
00013 void RBTree::rotateLeft(RBTreeNode*& node) {
00014     RBTreeNode* rightChild = node->right;
00015     if(rightChild) {
00016         RBTreeNode* grandparent = node->parent;
00017         RBTreeNode* grandson = rightChild->left;
00018
00019
00020     node->right = grandson;
```

```
00021     if (grandson)
00022         grandson->parent = node;
00023 
00024 
00025     rightChild->left = node;
00026     rightChild->parent = grandparent;
00027 
00028 
00029     if (!grandparent) {
00030         root = rightChild;
00031     } else if (node == grandparent->left) {
00032         grandparent->left = rightChild;
00033     } else {
00034         grandparent->right = rightChild;
00035     }
00036 
00037     node->parent = rightChild;
00038 
00039     node = rightChild;
00040 
00041 
00042     }
00043 }
00044 
00045 void RBTree::rotateRight(RBTreeNode*& node) {
00046     RBTreeNode* leftChild = node->left;
00047     if(leftChild) {
00048         RBTreeNode* grandparent = node->parent;
00049         RBTreeNode* grandson = leftChild->right;
00050 
00051 
00052     node->left = grandson;
00053     if (grandson)
00054         grandson->parent = node;
00055 
00056 
00057     leftChild->right = node;
00058     leftChild->parent = grandparent;
00059 
00060 
00061     if (!grandparent) {
00062         root = leftChild;
00063     } else if (node == grandparent->left) {
00064         grandparent->left = leftChild;
00065     } else {
00066         grandparent->right = leftChild;
00067     }
00068 
00069     node->parent = leftChild;
00070 
00071     node = leftChild;
00072 
00073 
00074     }
00075 
00076 }
00077 
00078 void RBTree::Balance(RBTreeNode*& node) {
00079     while (node != root && node->parent && node->parent->color == RED && node->parent->parent) {
00080         RBTreeNode* parent = node->parent;
00081         RBTreeNode* grandparent = parent->parent;
00082 
00083 
00084         RBTreeNode* uncle = (parent == grandparent->left) ?  grandparent->right :  grandparent->left;
00085 
00086         if (uncle && uncle->color == RED) {
00087             // Case 1:  Uncle is RED -- recolor
00088             parent->color = BLACK;
00089             uncle->color = BLACK;
00090             grandparent->color = RED;
00091             node = grandparent;
00092         } else {
00093             if (parent == grandparent->left) {
00094                 if (node == parent->right) {
00095                     // Case 2:  Left-Right -- rotate left on parent
00096                     rotateLeft(parent);
00097                     node = parent;
00098                     parent = node->parent;
00099                 }
00100                 // Case 3:  Left-Left -- rotate right on grandparent
00101                 parent->color = BLACK;
00102                 grandparent->color = RED;
00103                 rotateRight(grandparent);
00104             } else {
00105                 if (node == parent->left) {
00106                     // Case 2 mirror:  Right-Left
00107                     rotateRight(parent);
```

```
00108                     node = parent;
00109                     parent = node->parent;
00110                 }
00111                 // Case 3 mirror:  Right-Right
00112                 parent->color = BLACK;
00113                 grandparent->color = RED;
00114                 rotateLeft(grandparent);
00115             }
00116         }
00117     }
00118     root->color = BLACK;
00119 }
00120
00121
00122 RBTreeNode* RBTree::Insert(RBTreeNode*& node, Flight key, RBTreeNode* parent) {
00123     if (node == nullptr) {
00124         node = new RBTreeNode(key);
00125         node->parent = parent;
00126         return node;
00127     }
00128     else if (key < node->key) {
00129         return Insert(node->left, key, node);
00130     }
00131     else if (key > node->key) {
00132         return Insert(node->right, key, node);
00133     } else {
00134         node->flights.push_back(key);
00135         return node;
00136     }
00137 }
00138
00139 RBTree::RBTree() {
00140     root = nullptr;
00141 }
00142
00143 RBTree::RBTree(RBTreeNode* root_) {
00144     root = root_;
00145     root->color = BLACK;
00146 }
00147
00148 int RBTree::Search(RBTreeNode* node, Flight key, std::vector<Flight>& v) {
00149     int count = 0;
00150     if (node != nullptr){
00151         if (key < node->key) {
00152             count = Search(node->left, key, v);
00153         } else if (key > node->key) {
00154             count = Search(node->right, key, v);
00155         } else {
00156             v.insert(v.end(), node->flights.begin(), node->flights.end());
00157             count = node->flights.size();
00158         }
00159     }
00160     return count;
00161 }
00162
00163 void  RBTree::DeleteTree(RBTreeNode *& node) {
00164     if (node != nullptr) {
00165
00166         DeleteTree(node->left);
00167         DeleteTree(node->right);
00168
00169         delete node;
00170         node = nullptr;
00171     }
00172 }
00173
00174 RBTree::~RBTree() {
00175     this->DeleteTree(root);
00176 }
00177
00178 void RBTree::Insert(Flight key) {
00179     RBTreeNode* inserted = this->Insert(root, key, nullptr);
00180     Balance(inserted);
00181 }
00182
00183 int RBTree::Search(Flight key, std::vector<Flight>& result) {
00184     return this->Search(root, key, result);
00185 }
```

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.

```
00001 #ifndef RBTREE_H
00002 #define RBTREE_H
00003 #include "flight.h"
00004
00005 #define BLACK 0
00006 #define RED 1
00007
00012 struct RBTreeNode {
00013     Flight key;
00014     std::vector<Flight> flights;
00015     RBTreeNode* left;
00016     RBTreeNode* right;
00017     RBTreeNode* parent;
00018     int color;
00019
00024     RBTreeNode(Flight key_);
00025 };
00026
00031 class RBTree {
00032 private:
00033     RBTreeNode* root;
00034
00042     RBTreeNode* Insert(RBTreeNode *& node, Flight key, RBTreeNode* parent);
00043
00051     int Search(RBTreeNode* node, Flight key, std::vector<Flight>& result);
00052
00057     void DeleteTree(RBTreeNode *& node);
00058
00063     void Balance(RBTreeNode*& node);
00064
00069     void rotateLeft(RBTreeNode*& node);
00070
00075     void rotateRight(RBTreeNode*& node);
00076
00077 public:
00082     void Insert(Flight key);
00083
00090     int Search(Flight key, std::vector<Flight>& result);
00091
00095     RBTree();
00096
00101     RBTree(RBTreeNode* root_);
00102
00106     ~RBTree();
00107
00112     const RBTreeNode* GetRoot() const;
00113
00118     void SetRoot(RBTreeNode* root_);
00119 };
00120 #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.

```
00001 #include "search.h"
00002 #include <fstream>
00003 #include "flight.h"
00004 #include "bstree.h"
00005 #include "rbtree.h"
00006 #include <vector>
00007 #include "hash_table.h"
00008 #include <chrono>
00009 #include <map>
00010
00011
00012 int parseCSV(std::string filename, std::vector<Flight> &result) {
00013     int error = 0;
00014     std::ifstream in(filename);
00015     std::string line, flight_number_, airline_, arrival_date_, arrival_time_;
00016     int i1 = 0, i2 = 0, i3 = 0, i4 = 0, passengers_;
00017
00018     if (in.is_open()) {
00019         while (std::getline(in, line)) {
00020             i1 = i2 = i3 = i4 = 0;
00021             for (int i = 0; i < (int)line.size(); i++) {
00022                 if (line[i] == ',') {
00023                     if (i1 == 0) i1 = i;
00024                     else if (i2 == 0) i2 = i;
00025                     else if (i3 == 0) i3 = i;
00026                     else if (i4 == 0) i4 = i;
00027                 }
00028             }
00029             flight_number_ = line.substr(0, i1);
00030             airline_ = line.substr(i1 + 1, i2 - i1 - 1);
00031             arrival_date_ = line.substr(i2 + 1, i3 - i2 - 1);
00032             arrival_time_ = line.substr(i3 + 1, i4 - i3 - 1);
00033
00034             try {
00035                 passengers_ = std::stoi(line.substr(i4 + 1, line.size() - i4 - 1));
00036             } catch (...)  {
00037                 passengers_ = 0;
00038             }
00039
00040             Flight f(flight_number_, airline_, arrival_date_, arrival_time_, passengers_);
00041             result.push_back(f);
00042         }
00043         in.close();
00044     } else {
00045         error = 1;
00046     }
00047     return error;
00048 }
00049
00050 int linear_search(const std::vector<Flight>& flights, std::vector<Flight>& result) {
00051     int found = 0;
00052     for (int i = 0; i < (int)flights.size(); ++i) {
00053         if (flights[i].Get_airline() == KEY) {
00054             result.push_back(flights[i]);
00055             found++;
00056         }
00057     }
00058     return found;
00059 }
00060
00061
00062 BSTree* binary_insert(const std::vector<Flight>& flights) {
00063     if (flights.empty()) return nullptr;
00064
00065     TreeNode* root = new TreeNode(flights[0]);
00066     BSTree* tree = new BSTree(root);
00067
00068     for (int i = 1; i < (int)flights.size(); i++) {
00069         tree->Insert(flights[i]);
00070     }
00071
00072     return tree;
00073 }
00074
00075
00076 RBTree* rb_insert(const std::vector<Flight>& flights) {
00077     if (flights.empty()) return nullptr;
00078
00079     RBTreeNode* root = new RBTreeNode(flights[0]);
00080     RBTree* tree = new RBTree(root);
00081
00082     for (int i = 1; i < (int)flights.size(); i++) {
00083         tree->Insert(flights[i]);
00084     }
00085
00086     return tree;
00087 }
```

```
00088
00089
00090 HashTable* hash_table_insert(const std::vector<Flight>& flights, int& collisions) {
00091     HashTable* table = new HashTable(7);
00092
00093     for (int i = 0; i < (int)flights.size(); i++) {
00094         table->insert(flights[i]);
00095     }
00096
00097     collisions = table->get_collision_count();
00098     return table;
00099 }
00100
00101 std::multimap<std::string, Flight>* multimap_insert(const std::vector<Flight>& flights) {
00102     auto* mmap = new std::multimap<std::string, Flight>();
00103     for (int i = 0; i < (int)flights.size(); i++) {
00104         mmap->insert({flights[i].Get_airline(), flights[i]});
00105     }
00106     return mmap;
00107 }
00108
00109 int main() {
00110     std::vector<std::string> filenames = {
00111         "100.csv", "200.csv", "500.csv", "1000.csv", "2000.csv",
00112         "5000.csv", "10000.csv", "20000.csv", "50000.csv", "75000.csv"
00113     };
00114
00115     std::string path = "/Users/anastasiatrufanova/Desktop/lab2_data_search_algorithms/data/";
00116     std::ofstream
    out("/Users/anastasiatrufanova/Desktop/lab2_data_search_algorithms/plt/results.csv");
00117
00118     if (!out.is_open()) {
00119         std::cerr « "Failed to open output file!" « std::endl;
00120     } else {
00121
00122     out « "Array Size,Linear,Binary,RBTree,HashTable,Collisions,Multimap\n";
00123
00124     for (const auto& filename :  filenames) {
00125         std::string full_path = path + filename;
00126         std::cout « "Processing:  " « full_path « std::endl;
00127
00128         std::vector<Flight> flights;
00129         if (parseCSV(full_path, flights) != 0) {
00130             std::cerr « "Error parsing:  " « full_path « std::endl;
00131             continue;
00132         }
00133
00134         std::vector<Flight> res1, res2, res3, res4, res5;
00135         int collisions = 0;
00136
00137         auto t1_start = std::chrono::high_resolution_clock::now();
00138         linear_search(flights, res1);
00139         auto t1_end = std::chrono::high_resolution_clock::now();
00140         double time_linear = std::chrono::duration<double, std::milli>(t1_end - t1_start).count();
00141
00142
00143         BSTree* bst_tree = binary_insert(flights);
00144
00145         auto t2_start = std::chrono::high_resolution_clock::now();
00146         if (bst_tree) {
00147             bst_tree->Search(Flight("", KEY, "", "", 0), res2);
00148         }
00149         auto t2_end = std::chrono::high_resolution_clock::now();
00150         double time_binary = std::chrono::duration<double, std::milli>(t2_end - t2_start).count();
00151         delete bst_tree;
00152
00153
00154         RBTree* rb_tree = rb_insert(flights);
00155
00156         auto t3_start = std::chrono::high_resolution_clock::now();
00157         if (rb_tree) {
00158             rb_tree->Search(Flight("", KEY, "", "", 0), res3);
00159         }
00160         auto t3_end = std::chrono::high_resolution_clock::now();
00161         double time_rbtree = std::chrono::duration<double, std::milli>(t3_end - t3_start).count();
00162         delete rb_tree;
00163
00164         HashTable* hash_table = hash_table_insert(flights, collisions);
00165         auto t4_start = std::chrono::high_resolution_clock::now();
00166         if (hash_table) {
00167             hash_table->search(KEY, res4);
00168         }
00169         auto t4_end = std::chrono::high_resolution_clock::now();
00170         double time_hash = std::chrono::duration<double, std::milli>(t4_end - t4_start).count();
00171         delete hash_table;
00172
00173
```

```
00174         auto* mmap = multimap_insert(flights);
00175
00176         auto t5_start = std::chrono::high_resolution_clock::now();
00177         if (mmap) {
00178             auto range = mmap->equal_range(KEY);
00179             for (auto it = range.first; it != range.second; ++it) {
00180                 res5.push_back(it->second);
00181             }
00182         }
00183         auto t5_end = std::chrono::high_resolution_clock::now();
00184         double time_multimap = std::chrono::duration<double, std::milli>(t5_end - t5_start).count();
00185         delete mmap;
00186
00187
00188         out « flights.size() « ","
00189             « time_linear « "," « time_binary « "," « time_rbtree « ","
00190             « time_hash « "," « collisions « "," « time_multimap « "\n";
00191     }
00192
00193     out.close();
00194     }
00195
00196     return 0;
00197 }
```

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.

```
00001 #ifndef SEARCH_H
00002 #define SEARCH_H
00003 #include"flight.h"
00004 #include "bstree.h"
00005 #include "rbtree.h"
00006 #include <map>
00007 #include "hash_table.h"
00008
00009 #define KEY "Ural Airlines"
00010
00017 int parseCSV(std::string filename, std::vector<Flight> &result);
00018
00025 int linear_search(const std::vector<Flight>& flights, std::vector<Flight>& result);
00026
00032 BSTree* binary_insert(const std::vector<Flight>& flights);
00033
00039 RBTree* rb_insert(const std::vector<Flight>& flights);
00040
00047 HashTable* hash_table_insert(const std::vector<Flight>& flights, int& collisions);
00048
00054 std::multimap<std::string, Flight>* multimap_insert(const std::vector<Flight>& flights);
00055
00056 #endif
```

Index