

**Algoritmos e Estruturas de Dados (MIEI) – 2017/2018**

**Relatório Trabalho Final**



**Realizado pelo alunos do turno P7:**

**43612 – Luís Maria Soares Rosa**

**44640 – Diogo Almeida Pereira**

Índice

[Introdução 4](#_Toc500101577)

[Modelo dos Tipos Abstratos de Dados 5](#_Toc500101578)

[TAD – HomeAway 5](#_Toc500101579)

[TAD – SafeHome 8](#_Toc500101580)

[TAD – Home 9](#_Toc500101581)

[TAD – SafeRegion 10](#_Toc500101582)

[TAD – Region 11](#_Toc500101583)

[TAD – SafeUser 12](#_Toc500101584)

[TAD – User 13](#_Toc500101585)

[TAD – Travel 14](#_Toc500101586)

[Modelo das Estruturas de Dados 15](#_Toc500101587)

[Modelo das Estruturas de Dados(cont.) 16](#_Toc500101588)

[Complexidades temporais e descrição das operações efetuadas 17](#_Toc500101589)

[Estudo da complexidade espacial 21](#_Toc500101590)

# Introdução

Com este trabalho pretende-se implementar de um sistema de aluguer de casas e apartamentos (propriedades) online, permitindo aos utilizadores do mesmo constituírem-se como viajantes e/ou anfitriões relativamente a locais que façam parte do sistema. Um anfitrião é um utilizador que possui uma propriedade para alugar. Um viajante é também um utilizador do sistema que procura viajar para um determinado lugar e que pretende alugar uma propriedade nessa região. O sistema deverá permitir a gestão das propriedades para aluguer associadas aos anfitriões que as possuem. Deve ainda permitir a avaliação das propriedades pelos viajantes, de forma a permitir a ordenação das mesmas, de acordo com a qualidade da estadia, após o término da viagem.

Nós implementámos a versão J, utilizando as estruturas de dados e comparadores fornecidos no package java.util. Apesar das dataStructures estarem desenvolvidas ao longo do semestre na cadeira Algoritmos e Estruturas de Dados (AED), tivemos algumas dificuldades na implementação da última fase do trabalho, usando assim Hashmap, LinkedList para as classes do package do HomeAway.

A nossa aplicação ao arrancar começa por ler os comandos de entrada padrão (System.in), processando-os um a um e enviando o resultado para a saída padrão (System.out), o sistema de aluguer de casas e apartamentos (propriedades) online, termina quando for atingido um fim do ficheiro na entrada padrão. A persistência dos dados é assegurada entre execuções consecutivas.

# Modelo dos Tipos Abstratos de Dados

Para resolução deste problema utilizamos 8 tipos abstratos de dados.

## TAD – HomeAway

HomeAway é a nossa TAD principal onde se efetua de aluguer de casas e propriedade. Permite realizar as seguintes operações como inserir, remover, atualizar e obter dados de um utilizador. Com o registo de um utilizador no sistema, podemos adicionar, remover, obter os dados de uma propriedade. Com estas entradas feitas temos acesso às outras funcionalidades como adicionar estadias do utilizador nas propriedades, classificando-as se este utilizador não for o dono da propriedade. Assim podemos listar as viagens de um utilizador da mais recente para a mais antiga, as listar as casas de uma região ordenadas pela classificação dada pelas estadias feitas e listar as casas com uma dada capacidade de uma região.

Como variáveis:

regions = **new** HashMap<String, Region>();

users = **new** HashMap<String, User>();

homes = **new** HashMap<String, Home>();

Métodos:

//Verifies if a SafeUser with the given userID exists in the system

**boolean** **hasUser**(**String** userID);

//Verifies if a SafeHome with the given homeID exists in the system

**boolean** **hasHome**(**String** homeID);

//Verifies if a SafeHome with the given homeID exists in the properties of the SafeUser with the given userID

**boolean** **userOwnsHome**(**String** userID, **String** homeID)

**throws** **UserNotFoundException**, **HomeNotFoundException**;

//Verifies if the SafeHome with the given homeID has had any Travel to it.

**boolean** **hasTravel**(**String** homeID) **throws** **HomeNotFoundException**;

//Verifies if a SafeRegion with the given regionID exists in the

**boolean** **hasRegion**(**String** regionID);

//Verifies if the SafeRegion with the given regionID can house the given capacity.

**boolean** **regionHasCapacity**(**String** region, **int** capacity) **throws** **RegionNotFoundException**;

//Verifies if the SafeUser with the given userID has travelled.

**boolean** **isTraveller**(**String** userID) **throws** **UserNotFoundException**, **UserNotTravellerException**;

//Verifies if the SafeUser with the given userID is an owner.

**boolean** **isOwner**(**String** userID) **throws** **UserNotFoundException**, **HomeNotFoundException**;

//Returns the SafeRegion with the given identifier.

SafeRegion **getRegion**(**String** regionID) **throws** **RegionNotFoundException**;

//Returns the SafeUser with the given userID.

SafeUser **getUser**(**String** userID) **throws** **UserNotFoundException**;

//Lists all Travels of the SafeUser identified by userID.

Iterator<Travel> **listTravels**(**String** userID) **throws** **UserNotFoundException**, **UserNotTravellerException**;

//Searches SafeHomes in the SafeRegion identified by the given regionID able to house the given number of people.

Iterator<SafeHome> **pollHome**(**int** capacity, **String** regionID)

**throws** **RegionNotFoundException**, **NoCapacityException**;

//Lists all SafeHomes available in the SafeRegion identified by the given regionID, sorted from best to worst

Iterator<SafeHome> **listBest**(**String** region)

**throws** **RegionNotFoundException**, **HomeNotFoundException**;

// Inserts a SafeUser in the system.

**void** **insertUser**(**String** userID, **String** email, **String** phone, **String** name, **String** nationality, **String** address) **throws** **DuplicateException**;

//Updates a SafeUser's information in the system, name and nationality

**void** **updateUser**(**String** userID, **String** email, **String** phone, **String** address) **throws** **UserNotFoundException**;

//Removes SafeUser in the system identified by the given userID.

**void** **removeUser**(**String** userID)

**throws** **UserNotFoundException**, **UserIsOwnerException**;

//Adds a SafeHome to rent in the system.

**void** **addHome**(**String** homeID, **String** userID, **int** price, **int** person, **String** region, **String** description, **String** address)

**throws** **DuplicateException**, **UserNotFoundException**;

//Removes a SafeHome from the system.

**void** **removeHome**(**String** homeID) **throws** **HomeNotFoundException**;

// Returns the SafeHome with the given homeID

SafeHome **getHome**(**String** homeID) **throws** **HomeNotFoundException**;

//Adds a Travel made by the SafeUser identified by the given userID

**void** **addTravel**(**String** userID, **String** homeID, **int** score) **throws** **UserNotFoundException**, **HomeNotFoundException**;

//Lists all SafeHomes of the SafeUser identified by userID, sorted by homeID

Iterator<SafeHome> **listHome**(**String** userID)

**throws** **UserNotFoundException**, **HomeNotFoundException**;

## TAD – SafeHome

A SafeHome serve para a passagem para a main e como tal não é alterável. Modela uma propriedade modela uma propriedade, que por sua vez é representado pelo nome da propriedade inserido no sistema, representação do homeID(homeKey), nome da região, uma descrição, uma morada, preço da estadia, capacidade das pessoas pode ter uma propriedade, pontuação atribuída a essa propriedade das estadias feitas. Listagem de estadias feitas numa propriedade.

A TAD SafeHome tem como variáveis:

**private** **String** homeID, homeKey, region, description, address;

**private** **int** price, capacity, totalScore;

**private** SafeUser owner;

**private** List<Travel> travels;

Metodos:

**return** true if this SafeHome has had at least one Travel, false if not

**boolean** **hasTravel**();

**return** the homeID of this SafeHome

**String** **getID**();

**return** the description of this SafeHome

**String** **getDescription**();

**return** the address of this SafeHome

**String** **getAddress**();

**return** the SafeUser owner of this SafeHome

SafeUser **getUser**();

**return** the price of a Travel to this SafeHome

**int** **getPrice**();

**return** the number of people this SafeHome can house.

**int** **getCapacity**();

**return** the accumulated score of this SafeHome.

**int** **getTotalScore**();

**return** the unique region identifier of the SafeRegion where this SafeHome is located

**String** **getRegion**();

**return** the key associated with this SafeHome

**String** **getKey**();

## TAD – Home

Este TAD descreve um objecto Home, que é uma instancia alterável de um objeto SafeHome, com visibilidade limitada à package HomeAway.

Metodos:

//Adds a Travel to this Home.

**void** **addTravel**(**int** score, Travel travel);

);

## TAD – SafeRegion

A SafeRegion serve para a passagem para a main e como tal não é alterável. Modela uma região, com um identificador(key), nome da região, a capacidade máxima das propriedades dessa região adicionadas no sistema.

Tem como variáveis:

**private** **String** key, name;

**private** **int** maxCapacity;

**private** Map<String, SafeHome> homes;

Metodos:

//Lists all SafeHomes available for renting in this SafeRegion, sorted by accumulated score, from highest rated to lowest rated.

//If two or more SafeHomes have the same score, they are sorted by their unique identifier.

Iterator<SafeHome> **listBest**();

//Lists all SafeHomes available for renting in this SafeRegion that can accommodate the given number of people.

//Two SafeHomes with the same capacity are sorted by their unique identifier.

Iterator <SafeHome> **pollHome**(**int** capacity);

//**return** true if SafeRegion has at least one SafeHome, false if not.

**boolean** **hasHome**();

//**return** the maximum capacity of the SafeHome with the most capacity in this SafeRegion

**int** **getRegionCapacity**();

//**return** the unique SafeRegion identifier (name)

**String** **getRegionName**();

//**return** key associated with this SafeRegion

**String** **getKey**();

## TAD – Region

Este TAD descreve um objecto Region, que é uma instancia alterável de um objeto SafeRegion, com visibilidade limitada à package HomeAway.

//Adds a SafeHome to this Region.

**void** **addHome**(SafeHome home);

//Removes a SafeHome from this Region.

**void** **removeHome**(SafeHome home);

}

## TAD – SafeUser

A SafeUser serve para a passagem para a main e como tal não é alterável. Modela um user, que por sua vez é representado pelo nome do user inserido no sistema, representação do userID(userKey), nome do user, a nacionalidade, uma morada, email e telefone. O SafeUser pode ser viajante ou dono de uma propriedade(SafeHome) ou ambas.

Tem como Variaveis:

**private** **String** userID, userKey, name, nationality, address, email, phone;

**private** SortedMap<String, SafeHome> userHomes;

**private** Deque<Travel> travels;

Métodos:

//**return** true if user has travelled, false if not

**boolean** **hasTravel**();

//**return** true if user owns a home, false if not

**boolean** **hasHome**();

//Verifies if a given SafeHome is owned by this SafeUser.

**boolean** **hasHome**(**String** homeID);

//**return** userID of this SafeUser

**String** **getID**();

//**return** email of this SafeUser

**String** **getEmail**();

//**return** phone of this SafeUser

**String** **getPhone** ();

//**return** name of this SafeUser

**String** **getName**();

//**return** nationality of this SafeUser

**String** **getNationality**();

//**return** address of this SafeUser

**String** **getAddress**();

//returns this SafeUser's SafeHome with the given key.

SafeHome **getHome**(**String** homeID);

//Lists all Travels this SafeUser has made, sorted from most recent to oldest.

Iterator <Travel> **listTravels**();

//Lists all SafeHomes this SafeUser owns, SafeHomes are listed in order according to their unique identifier.

Iterator <SafeHome> **listHomes**();

//**return** SafeUser key

**String** **getKey**();

## TAD – User

Este TAD descreve um objecto User, que é uma instancia alterável de um objeto SafeUser, com visibilidade limitada à package HomeAway.

Metodos:

//Updates this User's email, phone and address to the given fields.

**void** **updateUser**(**String** email, **String** phone, **String** address);

//Adds a Travel to the given SafeHome.

**void** **addTravel**(**int** score, SafeHome home);

//Adds a SafeHome to this User.

**void** **addHome**(SafeHome home);

//Removes a SafeHome from this User.

**void** **removeHome**(SafeHome home);

## TAD – Travel

Esta TAD tem como função registar uma viagem de um viajante numa propriedade e atribuir uma pontuação a essa viagem.

Tem como variáveis:

**private** **int** score;

**private** SafeHome home;

Metodos:

//**return** SafeHome of this Travel

SafeHome **getHome**();

//**return** Score of this Travel

**int** **getScore**();

# Modelo das Estruturas de Dados

TAD – HomeAway

regions = **new** HashMap<String, Region>();

users = **new** HashMap<String, User>();

homes = **new** HashMap<String, Home>();

A estrutura de dados regions, guarda todas as regiões no HomeAway. Para a sua implementação foi escolhida a tabela de dispersão aberta HashMap, cuja chave é o uma representação em lowercase do seu identificador(String), e o seu valor é uma region(Region).

A estrutura de dados users, guarda todas as utilizadores no HomeAway. Para a sua implementação foi escolhida a tabela de dispersão aberta HashMap, cuja chave é o uma representação em lowercase do seu identificador(String), e o seu valor é uma user(User).

A estrutura de dados homes, guarda todas as propriedades no HomeAway. Para a sua implementação foi escolhida a tabela de dispersão aberta HashMap, cuja chave é o uma representação em lowercase do seu identificador(String), e o seu valor é uma home(Home);

TAD – SafeHome, Home

travels = **new** LinkedList<Travel>();

A estrutura de dados travels guarda todas as estadias(travels) feitas para esta casa. Para a sua implementação foi escolhida a lista duplamente ligada, que irá sempre inserir   
à cauda sempre que for introduzida uma estadia.

# Modelo das Estruturas de Dados(cont.)

TAD – SafeRegion, Region

homes = **new** HashMap<String, SafeHome>();

A estrutura de dados homes, guarda todas as propriedades na Region. Para a sua implementação foi escolhida a tabela de dispersão aberta HashMap, cuja chave é o uma representação em lowercase do seu identificador(String), e o seu valor é uma home(SafeHome);

TAD – SafeUser, User

userHomes = **new** TreeMap<String, SafeHome>();

travels = **new** LinkedList<Travel>();

A estrutura de userHomes guarda todas as propriedades(SafeHome), que são propriedade deste user. Para a sua implementação esta estrutura de dados foi escolhida uma TreeMap, porque interessa-nos iterar esta estrutura por ordem das chaves., cuja a chave é uma representação em lowercase do seu identicador(String), e o seu valor é uma propriedade(SafeHome).

A estrutura de dados homes, guarda todas as propriedades no HomeAway. Para a sua implementação foi escolhida a tabela de dispersão aberta HashMap, cuja chave é o uma representação em lowercase do seu identificador(String), e o seu valor é uma home(Home);

A estrutura de dados travels guarda todas as estadias(travels) feitas por este user. Para a sua implementação foi escolhida a lista duplamente ligada, que irá sempre inserir   
à cauda sempre que for introduzida uma estadia.

# Complexidades temporais e descrição das operações efetuadas

**Legenda:**

U – quantidade de Users no HashMap users;

H – quantidade de Homes no HashMap homes;

R – quantidade de Regions no HashMap regions;

hr – quantidade de Homes no HashMap homes de uma dada Region;

hu – quantidade de Homes no HashMap homes de um dado User;

T – quantidade de Travels na LinkedList de um dado User;

|  |  |  |  |
| --- | --- | --- | --- |
| **InsertUser** | Melhor Caso | Pior Caso | Caso Esperado |
| Verificar que o User ainda não existe | O(1) | O(U) | O(1) |
| Criar um novo objecto User | O(1) | O(1) | O(1) |
| Inserir o User no HashMap | O(1) | O(U) | O(1) |

|  |  |  |  |
| --- | --- | --- | --- |
| **RemoveUser** | Melhor Caso | Pior Caso | Caso Esperado |
| Verificar que o User existe | O(1) | O(U) | O(1) |
| Remover o User do HashMap | O(1) | O(U) | O(1) |

|  |  |  |  |
| --- | --- | --- | --- |
| **GetUser** | Melhor Caso | Pior Caso | Caso Esperado |
| Verificar que o User existe | O(1) | O(U) | O(1) |
| Encontrar o User | O(1) | O(U) | O(1) |

|  |  |  |  |
| --- | --- | --- | --- |
| **AddHome** | Melhor Caso | Pior Caso | Caso Esperado |
| Verificar que o User onde se vai introduzir a Home existe | O(1) | O(U) | O(1) |
| Verificar que a Home a introduzir ainda não existe | O(1) | O(H) | O(1) |
| Verificar que a Region onde se vai introduzir a Home existe, e caso não exista, criar | O(1) | O(R) | O(1) |
| Criar um novo objecto Home | O(1) | O(1) | O(1) |
| Inserir Home no HashMap homes | O(1) | O(H) | O(1) |
| Inserir Home na Region que lhe corresponde | O(1) | O(hr) | O(1) |
| Inserir Home no User que lhe corresponde | O(1) | O(hu) | O(1) |

|  |  |  |  |
| --- | --- | --- | --- |
| **UpdateUser** | Melhor Caso | Pior Caso | Caso Esperado |
| Verificar se o User existe | O(1) | O(U) | O(1) |
| Encontrar o User | O(1) | O(U) | O(1) |
| Actualizar dados do User | O(1) | O(1) | O(1) |

|  |  |  |  |
| --- | --- | --- | --- |
| **GetHome** | Melhor Caso | Pior Caso | Caso Esperado |
| Verificar que a Home já existe | O(1) | O(H) | O(1) |
| Encontrar a Home | O(1) | O(H) | O(1) |

|  |  |  |  |
| --- | --- | --- | --- |
| **RemoveHome** | Melhor Caso | Pior Caso | Caso Esperado |
| Verificar que a Home existe | O(1) | O(H) | O(1) |
| Verificar que a Home não tem Travels | O(1) | O(H) | O(1) |
| Remover Home da Region em que está inserida | O(1) | O(hr) | O(1) |
| Remover Home do User em que está inserida | O(1) | O(hu) | O(1) |
| Remover Home do HashMap homes | O(1) | O(H) | O(1) |

|  |  |  |  |
| --- | --- | --- | --- |
| **AddTravel** | Melhor Caso | Pior Caso | Caso Esperado |
| Encontrar o User que realiza a Travel | O(1) | O(U) | O(1) |
| Encontrar a Home para a qual é realizada a Travel | O(1) | O(H) | O(1) |
| Inserir Travel no User | O(1) | O(1) | O(1) |
| Inserir Travel na Home | O(1) | O(1) | O(1) |

|  |  |  |  |
| --- | --- | --- | --- |
| **ListHome** | Melhor Caso | Pior Caso | Caso Esperado |
| Verificar que o User cujas Homes vamos listar existe | O(1) | O(U) | O(1) |
| Verifica que o User cujas Homes vamos listar é dono de pelo menos uma Home | O(1) | O(1) | O(1) |
| Itera sobre todas as Homes do User | O(hu) | O(hu) | O(hu) |

|  |  |  |  |
| --- | --- | --- | --- |
| **ListTravels** | Melhor Caso | Pior Caso | Caso Esperado |
| Verificar que o User cujas Travels vamos listar existe | O(1) | O(U) | O(1) |
| Verificar que o User realizou viagens | O(1) | O(1) | O(1) |
| Itera sobre todas as Travels do User | O(T) | O(T) | O(T) |

|  |  |  |  |
| --- | --- | --- | --- |
| **ListBest** | Melhor Caso | Pior Caso | Caso Esperado |
| Verificar que a Region procurada existe | O(1) | O(R) | O(1) |
| Verificar que a Region procurada tem Homes | O(1) | O(hr) | O(1) |
| Ordenar Homes por classificação | O(hr\*log(hr)) | O(hr\*log(hr)) | O(hr\*log(hr)) |
| Iterar sobre as Homes ordenadas da Region | O(hr) | O(hr) | O(hr) |

|  |  |  |  |
| --- | --- | --- | --- |
| **PollHome** | Melhor Caso | Pior Caso | Caso Esperado |
| Verificar que a Region procurada existe | O(1) | O(R) | O(1) |
| Verificar que a Region procurada tem Homes | O(1) | O(hr) | O(1) |
| Ordenar Homes por capacidade | O(hr\*log(hr)) | O(hr\*log(hr)) | O(hr\*log(hr)) |
| Iterar sobre as Homes ordenadas da Region | O(hr) | O(hr) | O(hr) |

# Estudo da complexidade espacial

U – Users do sistema

R – Regions do sistema

H – Homes do sistema

HU - Homes de um User

HR – Homes de uma Region

TH – Travels de uma Home

TU – Travels de um User

A complexidade espacial do projeto é dada pela seguinte fórmula:

U \* HU + U \* TU + R \* HR + H \* TH