

**`**

**Algoritmos e Estruturas de Dados – 2022/2023 1º Semestre**

Relatório de Projeto

**“ArtAuctions - Obras de Arte e Leilões”**

**Cadeira de Algoritmos e Estruturas de Dados, 2023/2024**

**Realizado por:**

**Joana Neves, 65441**

**Rui Xavier, 65815**

Índice

[1. Introdução 2](#_Toc152092589)

[2. Estrutura, hierarquia, e funcionamento do projeto 3](#_Toc152092590)

[2.1. Tipos Abstratos de Dados 4](#_Toc152092591)

[2.1.1. Interface AuctionManagement 4](#_Toc152092592)

[2.1.1.1. Estruturas de dados da classe 4](#_Toc152092593)

[2.1.2. Interface User 6](#_Toc152092594)

[2.1.2.1. Interface PrivateUser 6](#_Toc152092595)

[2.1.2.2. Interface Collector 6](#_Toc152092596)

[2.1.2.3. Interface Artist 6](#_Toc152092597)

[2.1.3. Interface Auction 7](#_Toc152092598)

[2.1.3.1. Estruturas de dados da classe 7](#_Toc152092599)

[2.1.4. Interface Bid 7](#_Toc152092600)

[2.1.5. Interface Artwork 8](#_Toc152092601)

[2.1.6. Interface ArtworkInAuction 8](#_Toc152092602)

[2.1.3.1. Estruturas de dados da classe 9](#_Toc152092603)

[3. Estudo da Complexidade Temporal 10](#_Toc152092604)

[3.1. Inserir utilizador 10](#_Toc152092605)

[3.2. Inserir artista 10](#_Toc152092606)

[3.3. Remover utilizador 11](#_Toc152092607)

[3.4. Inserir obra de arte 11](#_Toc152092608)

[3.5. Consultar dados do utilizador 12](#_Toc152092609)

[3.6. Consultar dados do artista 13](#_Toc152092610)

[3.7. Consultar dados da obra de arte 13](#_Toc152092611)

[3.8. Criar um leilão 14](#_Toc152092612)

[3.9. Inserir uma obra de arte num leilão 15](#_Toc152092613)

[3.10. Inserir uma bid (proposta para uma obra) 16](#_Toc152092614)

[3.11. Fechar um leilão 17](#_Toc152092615)

[3.12. Listar todas as obras de um leilão 18](#_Toc152092616)

[3.13. Listar todas as obras de um artista 19](#_Toc152092617)

[3.14. Listar todas as bids de uma obra de arte 20](#_Toc152092618)

[3.15. Listar todas as obras vendidas por preço 21](#_Toc152092619)

[3.16. Terminar a execução 21](#_Toc152092620)

[4. Estudo da Complexidade Espacial 22](#_Toc152092621)

[5. Conclusão 22](#_Toc152092622)

# Introdução

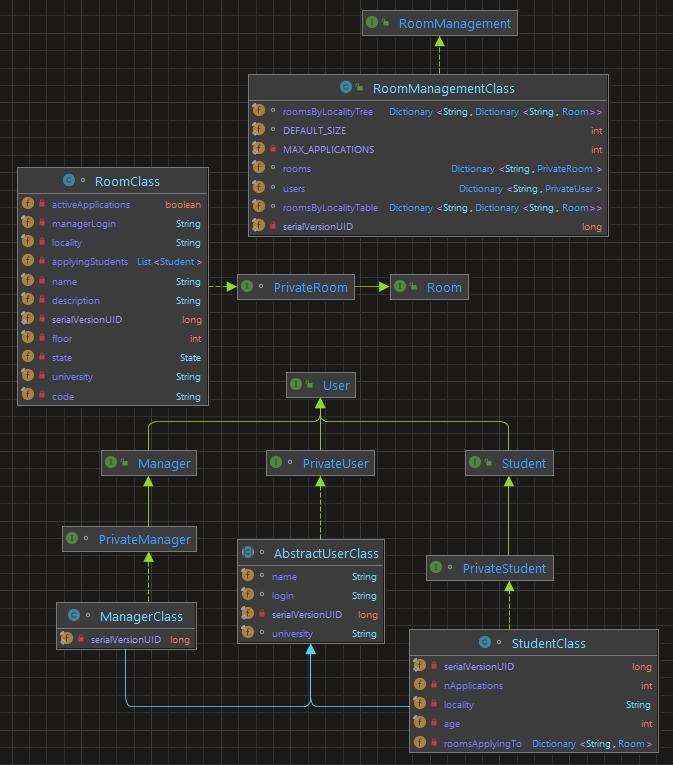
No âmbito da disciplina de Algoritmos e Estruturas de Dados do 2º ano de Licenciatura Informática, foi-nos proposta a realização do projeto **“ArtAuctions - Obras de Arte e Leilões”** de forma a aplicar os conhecimentos adquiridos ao longo do decorrer das aulas práticas e teóricas.

Este trabalho tinha como objetivo principal o desenvolvimento de um sistema de gerenciamento de obras de arte em leilões. O programa destina-se a ser usado por dois tipos de utilizadores: os **colecionadores** universitários **(*Collectors*)** e os **artistas** das residências **(A*rtists)***. Os colecionadores podem um valor de compra num determinado leilão para uma certa obra de arte (***Artwork***). Os artistas usam o sistema para introduzir obras de arte, introduzir o nome da obra, o ano da criação da mesma e o autor desta obra, neste caso ele mesmo.

O presente relatório destina-se à descrição detalhada de todo o processo de desenvolvimento do projeto, bem como à justificação de algumas das decisões tomadas relativamente à maximização da eficiência temporal e espacial do programa. Inclui ainda uma breve conclusão sobre a utilidade deste trabalho na melhor compreensão dos temas relevantes à realização do mesmo.

# Estrutura, hierarquia, e funcionamento do projeto

Abaixo encontra-se um diagrama de classes que visa dar um melhor entendimento da hierarquia das classes usadas no projeto, bem como da forma como estas se interligam. Como é possível interpretar pelo mesmo, todas as interfaces, à exceção da Interface ***RoomManagement***, são estendidas por uma subinterface com o prefixo “***Private***”. Este procedimento tem como objetivo a melhor divisão dos métodos declarados pelas interfaces, evitando a partilha desnecessária de informação com o exterior das classes que as implementam. Os métodos “***set***”, ou seja, que alteram de alguma forma os valores da classe, estão declarados na interface privada e só podem ser acedidos pela classe de topo, ficando assim inacessíveis diretamente pela classe main. Todos os restantes métodos estão declarados na classe pública e podem ser acedidos livremente.



## Tipos Abstratos de Dados

### Interface AuctionManagement

É implementada pela classe ***AuctionManagementClass***, que funciona como o sistema central do programa. Faz o tratamento dos dados e a comunicação entre a main e as restantes classes. Define 3 estruturas de dados e 2 constantes *int*:

* + - 1. Dictionary<String, PrivateUser> users
      2. Dictionary<String, PrivateArtwork> artworks
      3. Dictionary<String, PrivateAuction> auctions

Definem dicionários implementados por classes ***SepChainHashTable***. São utilizados para guardar, respetivamente, todos os utilizadores, obras de arte e leilões registados no sistema.

* + - 1. OrderedDictionary<Integer, OrderedDictionary<String, PrivateArtwork>> artworksRanking

Define um dicionário ordenado implementado pela classe ***AVL – Adelson-Velskii and Landis***. É utilizado para guardar todas obras de arte no sistema pelo ranking de venda. Em que o ***value*** de cada key neste dicionário é uma ***AVL***também mas ordenada por

* + - 1. final int AGE\_18 = 18

Define a idade mínima que um ***User*** deve ter.

#### Estruturas de dados da classe

Muitas das funcionalidades suportadas pelo programa requerem a inserção, remoção, pesquisa e modificação do estado de quartos e utilizadores várias vezes, não sendo necessário considerar a sua organização no sistema. Assim, achámos conveniente utilizar tabelas de dispersão para guardar os dados descritos acima nas alíneas **1.**, **2.**, **3.**, uma vez que a complexidade temporal da inserção, remoção, e pesquisa nesta estrutura é baixa e sempre constante (O(1)). Nestas usamos como ***key*** o login do utilizador ou identificador da obra de arte ou do leilão e como ***value*** uma instância da classe ***CollectorClass***, ***ArtistClass***, ***ArtworkClass***, ***AuctionClass***.

Sabemos ainda que outra funcionalidade do programa permite listar as artworks por ordem decrescente de acordo com o seu preço de venda mais alto, listando, assim o ranking das obras – ***listArtworksByValue()***. Para tal achávmos conveniente utilizar um Árvore de pesquisa Binária descrito acima na alínea **4.**.

Note-se, ainda, que é possível que exista mais de uma obra de arte com o mesmo preço de venda, e pretende-se que estas sejam então listadas por ordem decrescente dos respetivos preços de venda e em caso de empate, as obras são listadas por ordem lexiográfica do nome das respetivas obras. Para dar resposta a essa necessidade, esta estrutura têm uma particularidade: Cada nó da árvore (alínea **4.**) tem como ***key***,um Integer, e como ***value*** uma outra árvore de pesquisa binária cujos nós, por sua vez, contêm as obras de arte, com o mesmo preoço de venda, como ***values*** e os respetivos nomes como ***keys***, estas estão ordenadas por ordem lexicográfica (caso de empate entre preços de venda).

Na última alínea está definida uma constante que correspondente à idade mínima de um ***User***. Servem apenas para facilitar a interpretação do código e, caso seja necessário, uma eventual mudança dos valores definidos sem ter de alterar todos os métodos que os utilizam.

### Interface User

É estendida pelas sub-interfaces ***PrivateUser***, ***Collector*** e ***Artist***.

#### Interface PrivateUser

É implementada pela classe abstrata ***AbstractUserClass***, que é a classe mãe das subclasses ***CollectorClass*** e ***ArtistClass***. Cada ***User***, seja ele ***Collector*** ou ***Artist***, é um utilizador individual registado no sistema. Este possui um identificador único, um nome, um email e a sua idade. Os 3 primeiros elementos são definidos por constantes *String* e o último por uma constante *int*:

1. final String login

Inicializada no construtor como o identificador único do utilizador registado;

1. final String name

Inicializada no construtor como o nome do utilizador registado;

1. final String email

Inicializada no construtor como o email do utilizador registado;

1. final int age

Inicializada no construtor como a idade do utilizador registado.

1. Protected int countBidsActive

Inicializada no construtor como o counter das bids ativas do utilizador registado;

* + - 1. final int ZERO = 0

Define o número zero e é utilizada para inicializar o countBidsActive e verificar se este é igual a zero ou não.

#### Interface Collector

É estendida pela interface ***PrivateCollector***, que por sua vez é implementada pela classe ***CollectorClass***. Cada ***Collector*** é um tipo específico de ***User***, porém não precisa de mais dados para além daqueles que foram definidos na sua superclasse.

#### Interface Artist

É estendida pela interface ***PrivateArtist***, que por sua vez é implementada pela classe ***ArtistClass***. Cada ***Artist*** é um tipo específico de ***User*** com características particulares. Para além dos dados definidos na sua superclasse, possui ainda um nome artístico. Este é definido por uma constante *String*:

1. final String artistName

Inicializada no construtor como o nome artístico do artista registado.

1. protected int hasArtworkInAuction

Inicializada no construtor como o counter do número de obras de arte que o artista registado tem em leilões (Auctions).

1. final int ZERO = 0

Define o número zero e é utilizada para inicializar o hasArtworksInAuction e verificar se este é igual a zero ou não.

### Interface Auction

É estendida pela interface ***PrivateAuction***, que por sua vez é implementada pela classe ***AuctionClass***. Cada ***Auction*** representa um leilão registado no sistema e possui um identificador único, uma quantidade indefinida de obras de arte, definidos através de uma constante *String*, duas estruturas de dados.

* + - 1. final String id

Definida no construtor como o identificador único do leilão.

* + - 1. Dictionary<String, ArtworkInAuciont> artworkInAuctionDictionary

Define uma tabela de dispersão implementada por uma classe ***SepChainHashTable***. É utilizada para guardar todas as obras atualmente colocadas no leilão correspondente.

* + - 1. DoubleList<String, ArtworkInAuciont> artworkInAuctionList

Define uma lista implementada por uma classe ***DoubleList***. É utilizada para guardar todas as obras atualmente colocadas no leilão correspondente, estando ordenada por ordem de inserção.

#### 2.1.3.1. Estruturas de dados da classe

Pretende-se que o programa nos permita listar as obras de arte que estão atualmente registadas num determinado leilão, por ordem cronológica de inserção. Como tal, optámos por usar uma ***DoubleList***, uma vez que esta estrutura nos permite organizar os dados pela ordem pretendida, isto através da artworkInAuctionList pois é mais rápido para uma iteração. Por outro lado, também guardamos todas as obras de arte no leilão, neste caso numa tabela de dispersão para facilitar a pesquisa e acesso a uma determinada obra de arte, sabendo o hash desta.

### Interface Bid

É estendida pela interface ***PrivateBid***, que por sua vez é implementada pela classe ***BidClass***. Cada ***Bid*** representa uma proposta registada numa certa obra de arte (artwork) e possui o valor da proposta feita, um user, o que realizou a proposta em causa e um leilão (auction), que corresponde ao que esta bid foi feita, definidos através de uma constante *int*, de um ***User*** e de uma ***Auction***, respetivamente.

1. final int value

Inicializada no construtor como o valor da proposta de venda da obra de arte registado.

1. final User bidder

Inicializada no construtor como o ***User*** que realizou esta proposta para a obra no sistema.

1. final Auction auciton

Inicializada no construtor como a ***Auction*** em que foi realizada esta proposta para a obra no sistema.

### Interface Artwork

É estendida pela interface ***PrivateArtwork***, que por sua vez é implementada pela classe ***ArtworkClass***. Cada ***Artwork*** representa uma obra de arte e possui um identificador único, o ano em que foi feita, um nome, um ***User*** que a criou e um valor de venda mais alto. Estes dados estão definidos através de uma constante *String*, de 2 Strings, de um ***User*** e de um *int*, respetivamente.

1. final String id

Definida no construtor como o identificador único da obra de arte.

1. protected String year

Inicializada no construtor como o ano em que foi criada a obra de arte.

1. protected String name

Inicializada no construtor como o nome da obra de arte.

1. Protected User author

Inicializada no construtor como o ***User*** que criou a obra de arte.

1. protected int highestPriveValueSold

Inicializada a zero no construtor como o valor mais alto pelo qual a obra de arte foi vendida entre todos os leilões existentes.

### Interface ArtworkInAuction

É estendida pela interface ***PrivateArtworkInAuction***, que por sua vez é implementada pela classe ***ArtworkInAuctionClass***. Cada ***ArtworkInAuction*** representa uma obra de arte que se encontra num leilão e possui uma ***Artwork***, um iint, uma bid, um boolean e uma estrutura de dados. um valor mínimo de venda, uma ***Bid***, um sold, e , o ano em que foi feita, um nome, um ***User*** que a criou e um valor de venda mais alto. Estes dados estão definidos através de uma constante *String*, de 2 Strings, de um ***User*** e de um *int*, respetivamente.

1. final PrivateArtwork artwork

Definida no construtor como a obra de arte que se encontra num leilão.

1. final int minimumPrice

Inicializada no construtor como o preço mínimo pelo qual a obra de arte pode ser vendida neste leilão.

1. protected Bid highestBid

Inicializada no construtor como a bid mais alta relativa à obra de arte no respetivo leilão.

1. Protected boolean sold

Inicializada a falso no construtor como o *boolean* que informa se a obra de arte já foi vendia neste leilão.

1. DoubleList<Bid> bids

Define uma lista implementada por uma classe ***DoubleList***. É utilizada para guardar todas as ***Bids*** atualmente realizadas na obra de arte no leilão correspondente, estando ordenada por ordem de inserção.

#### 2.1.3.1. Estruturas de dados da classe

Pretende-se que o programa nos permita listar as ***Bids*** relativas a esta obra de arte que estão atualmente registadas num determinado leilão, por ordem cronológica de inserção. Como tal, optámos por usar uma ***DoubleList***, uma vez que esta estrutura nos permite organizar os dados pela ordem pretendida e ser mais rápido a iteração.

# Estudo da Complexidade Temporal

## Inserir utilizador

Este comando insere um novo estudante no sistema desde que não exista outro usuário com o mesmo login. Para fazer esta verificação fazemos um find no dicionário ***users*** para averiguar a existência de um ***User*** com o mesmo login, caso não se verifique criamos um novo objeto com os dados fornecidos pela Main e inserimo-lo no dicionário.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operação** | **Melhor Caso** | **Caso Esperado** | **Pior Caso.** |
| Pesquisa | O(1) | O(1 + ℷ) | O(n) |
| Criação | O(1) | O(1) | O(1) |
| Inserção | O(1) | O(1 + ℷ) | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Total** | **O(1)** | **O(1 +** ℷ**)** | **O(n)** |

Legenda :

* + - n - Número total de usuários no dicionário. (número de entradas da tabela)
    - ℷ - Fator de ocupação do dicionário, dado pela divisão do número de entradas da tabela pela dimensão da tabela.

## Inserir artista

Este comando lista as informações de um estudante, desde que o estudante esteja inserido no sistema. Para verificar se o estudante está inserido no sistema, fazemos uma pesquisa no dicionário users. Se encontrarmos o estudante, retornamos o mesmo para a Main como um objeto de interface pública.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operação** | **Melhor Caso** | **Caso Esperado** | **Pior Caso.** |
| Pesquisa | O(1) | O(1 + ℷ) | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Total** | **O(1)** | **O(1 +** ℷ**)** | **O(n)** |

Legenda :

* + - n - Número total de usuários no dicionário. (número de entradas da tabela)
    - ℷ - Fator de ocupação do dicionário, dado pela divisão do número de entradas da tabela pela dimensão da tabela.

## Remover utilizador

Este comando insere um novo gerente no sistema desde que não exista outro usuário com o mesmo login. Para fazer esta verificação fazemos um find no dicionário ***users*** para averiguar a existência de um utilizador com o mesmo login. Caso tal não se verifique, criamos um novo objeto com os dados fornecidos pelo utilizador e inserimo-lo no dicionário.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operação** | **Melhor Caso** | **Caso Esperado** | **Pior Caso.** |
| Pesquisa | O(1) | O(1 + ℷ) | O(n) |
| Criação | O(1) | O(1) | O(1) |
| Inserção | O(1) | O(1 + ℷ) | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Total** | **O(1)** | **O(1 +** ℷ**)** | **O(n)** |

Legenda:

* + - n - Número total de usuários no dicionário. (número de entradas da tabela)
    - ℷ - Fator de ocupação do dicionário, dado pela divisão do número de entradas da tabela pela dimensão da tabela.

## Inserir obra de arte

Este comando lista as informações de um gerente, desde que o gerente esteja inserido no sistema. Para verificar se tal acontece, fazemos uma pesquisa no dicionário ***users***. Se encontrarmos o gerente, retornamos o mesmo para a Main como um objeto de interface pública.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operação** | **Melhor Caso** | **Caso Esperado** | **Pior Caso.** |
| Pesquisa | O(1) | O(1 + ℷ) | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Total** | **O(1)** | **O(1 +** ℷ**)** | **O(n)** |

Legenda:

* + - n - Número total de usuários no dicionário. (número de entradas da tabela)
    - ℷ - Fator de ocupação do dicionário, dado pela divisão do número de entradas da tabela pela dimensão da tabela.

## Consultar dados do utilizador

Este comando insere um novo quarto no sistema, desde que:

* não exista outro quarto com o mesmo código.
* o login pertença a um gerente inserido no sistema
* a universidade fornecida coincidir com a universidade a qual o gerente está associado.

Para fazer esta verificação, fazemos uma pesquisa no dicionário ***rooms*** para averiguar a existência de um ***Room*** com o mesmo código, comparamos o login do gerente associado a este com o login do gerente dado pelo utilizador, e o nome da universidade fornecido com o nome da universidade a que o gerente pertence. Após as verificações, criamos um novo objeto com os dados fornecidos e inserimo-lo nos dicionários rooms, roomsByLocalityTree, roomByLocalityTable.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operação** | **Melhor Caso** | **Caso Esperado** | **Pior Caso.** |
| Pesquisa | O(1) | O(1 + ℷ) = z | O(n) = c |
| Criação | O(1) | O(1) | O(1) |
| Inserção rooms | O(1) | O(1 + ℷ) = z | O(n) = c |
| Pesquisar a localidade no dicionário roomsByLocalityTable | O(1) | O(1 + ℷ) = z | O(n1) = d |
| criação de uma BST | O(1) | O(1) | O(1) |
| inserção no resultado da pesquisa acima | O(1) | O(log(n2) = x | O(log(n2) = a |
| Inserção no roomsByLocalityTree | O(1) | O(log(n1) = y | O(log(n1) = b |
| Inserção no dicionário roomsByLocalityTable | O(1) | O(1 + ℷ) = z | O(n1) = d |

|  |  |  |  |
| --- | --- | --- | --- |
| **Total** | **O(1)** | **O(max(x,y,z ))** | **O(max(a,b,c,d)** |

Legenda:

* + - n - Número total de quartos no dicionário. (número de entradas dos 3 dicionários.)
    - ℷ - Fator de ocupação do dicionário, dado pela divisão do número de entradas da tabela pela dimensão da tabela.
    - n1 - número de localidades.
    - n2 - número de quartos na localidade

## Consultar dados do artista

Este comando lista as informações de um quarto, desde que o quarto esteja inserido no sistema. Para verificar se o quarto está inserido no sistema, fizemos uma pesquisa no dicionário rooms. Caso este seja encontrado, retornamo-lo para a Main como um objeto de interface pública.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operação** | **Melhor Caso** | **Caso Esperado** | **Pior Caso.** |
| Pesquisa | O(1) | O(1 + ℷ) | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Total** | **O(1)** | **O(1 +** ℷ**)** | **O(n)** |

Legenda:

* + - n - Número total de usuários no dicionário. (número de entradas da tabela)
    - ℷ - Fator de ocupação do dicionário, dado pela divisão do número de entradas da tabela pela dimensão da tabela.

## Consultar dados da obra de arte

Este comando modifica o estado de um quarto, desde que

* o quarto exista no sistema,
* o login fornecido seja coincidente com o login do gerente do quarto,
* o quarto não tenha candidaturas ativas
* o novo estado pretendido seja “OCUPADO”.

Para fazer estas verificações, começamos por fazer uma pesquisa no dicionário ***rooms***. Em seguida, averiguamos se o login do gerente do quarto coincide com o login fornecido pelo utilizador, e por fim fazemos um *if statement* que averigua se o novo estado é “OCUPADO” e se as candidaturas estão ativas. Após concluirmos as verificações, mudamos o estado do quarto.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operação** | **Melhor Caso** | **Caso Esperado** | **Pior Caso.** |
| Pesquisa | O(1) | O(1 + ℷ) | O(n) |
| Mudança de estado | O(1) | O(1) | O(1) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Total** | **O(1)** | **O(1 +** ℷ**)** | **O(n)** |

Legenda:

* + - n - Número total de usuários no dicionário. (número de entradas da tabela)
    - ℷ - Fator de ocupação do dicionário, dado pela divisão do número de entradas da tabela pela dimensão da tabela.

## Criar um leilão

Este comando remove um quarto do sistema. Para proceder a esta operação, temos que verificar:

* se o quarto está no sistema,
* se o login inserido pelo utilizador corresponde ao login do gerente do quarto
* se o quarto tem candidaturas ativas.

Para tal, fazemos uma pesquisa no dicionário rooms, seguido de uma comparação entre o login inserido pelo utilizador e o login guardado no objeto room, e por fim verificando a variável *boolean* que nos diz se o quarto tem candidaturas ativas. Se tudo correr como esperado, o comando passa à execução: o quarto obtido na primeira pesquisa é removido das tabelas e árvores que guardam os quartos do sistema.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operação** | **Melhor Caso** | **Caso Esperado** | **Pior Caso.** |
| Pesquisa | O(1) | O(1 + ℷ) = z | O(n) = c |
| Remoção do quarto de cada candidato. | O(1) | O(n1) = w | O(n1) |
| Remoção do quarto do dicionário rooms | O(1) | O(1 + ℷ) | O(n) |
| Pesquisar o quarto no dicionário roomsByLocalityTable | O(1) | O(1 + ℷ) | O(n2) |
| Remover o quarto da árvore binária (o resultado da pesquisa acima) | O(1) | O(log(n3)) = x | O(log(n3) =a |
| Remoção no roomsByLocalityTree | O(1) | O(log(n2) = y | O(log(n2) = b |
| Remoção no dicionário roomsByLocalityTable | O(1) | O(1 + ℷ) | O(n2) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Total** | **O(1)** | **O(max (z,x,y,w))** | **O(max(a,b,c))** |

Legenda:

* + - n - Número total de rooms no dicionário rooms. (número de entradas da tabela)
    - ℷ - Fator de ocupação do dicionário, dado pela divisão do número de entradas da tabela pela dimensão da tabela.
    - n1 - número de candidatos ao quarto.
    - n2 - número de localidades.
    - n3 - número de quartos com a mesma localidade que o quarto que vai se removido

## Inserir uma obra de arte num leilão

Este comando permite inserir uma nova candidatura de um estudante a um quarto. É feita uma pesquisa no dicionário ***users*** e ***rooms*** para encontrar o estudante e o quarto pretendidos, mas para o comando ser executado é necessário efetuar algumas verificações:

* se o estudante existe no sistema;
* se este já excedeu o limite máximo de candidaturas ativas;
* se o quarto existe no sistema;
* se este já se encontra ocupado;
* se o estudante já tem uma candidatura ativa a este quarto;

Caso alguma destas situações se verifiquem, é lançada a exceção correspondente ao erro, e caso contrário, o comando passa à execução. O estudante é inserido na lista de estudantes atualmente candidatos ao quarto, e o quarto é inserido na lista de quartos aos quais o estudante é atualmente candidato.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operação** | **Melhor Caso** | **Caso Esperado** | **Pior Caso.** |
| Pesquisa pelo quarto | O(1) | O(1 + ℷ) | O(n) |
| Pesquisa pelo estudante | O(1) | O(1 + ℷ) | O(n1) |
| Inserção do estudante na lista de candidatos ao quarto | O(1) | O(1) | O(1) |
| Inserção do quarto na lista de quartos aos quais o estudante é candidato | O(1) | O(1) | O(n2) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Total** | **O(1)** | **O(1 +** ℷ) | **O(max(n, n1, n2))** |

Legenda:

* + - n - Número total de rooms no dicionário ***rooms***. (número de entradas da tabela)
    - ℷ - Fator de ocupação do dicionário, dado pela divisão do número de entradas da tabela pela dimensão da tabela.
    - n1 - Número total de users no dicionário ***users***. (número de entradas da tabela)
    - n2 - Número de candidaturas de um estudante.

## Inserir uma bid (proposta para uma obra)

Este comando aceita uma candidatura a um quarto. Para que este comando tenha sucesso temos que verificar:

* se o quarto existe,
* se o login fornecido pelo utilizador corresponde ao login do gerente do quarto,
* se o estudante existe,
* se o estudante tem uma candidatura aquele quarto,

Para tal, fazemos pesquisa no dicionário ***rooms***, uma comparação entre o login dado e o login do gerente associado ao quarto, uma pesquisa no dicionário ***users***, e uma pesquisa na lista de quartos do estudante para verificar se este já é candidato ao quarto. Se as condições necessárias estiverem reunidas, o comando procede com a execução: Remove todas as candidaturas a outros quartos que o estudante possa ter e associa o estudante ao quarto pretendido mudando o estado para “OCUPADO” e removendo todas as candidaturas de outros estudantes.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operação** | **Melhor Caso** | **Caso Esperado** | **Pior Caso.** |
| Pesquisa pelo quarto | O(1) | O(1 + ℷ) | O(n) |
| Pesquisa pelo estudante | O(1) | O(1 + ℷ) | O(n) = e |
| Remoção do estudante na lista de candidaturas dos quartos. | O(1) | O(n1 \* n2) = a | O(n1 \* n2) = c |
| Remoção do quarto  da lista de candidaturas dos candidatos ao quarto. | O(1) | O(n2) = b | O(n2 \* n1) = d |

|  |  |  |  |
| --- | --- | --- | --- |
| **Total** | **O(1)** | **O(a)** | **O(max(c,d, e))** |

Legenda:

* + - n - Número total de rooms no dicionário rooms. (número de entradas da tabela)
    - ℷ - Fator de ocupação do dicionário, dado pela divisão do número de entradas da tabela pela dimensão da tabela.
    - n1 - número de candidaturas de um estudante.
    - n2 - número de candidatos ao quarto.

## Fechar um leilão

Este comando lista as candidaturas ativas a um quarto, devolvendo um iterador de quartos para a Main. Para o fazer temos que:

* verificar se o quarto existe,
* verificar se o login introduzido corresponde ao login do manager do quarto,
* verificar se o quarto tem alguma candidatura ativa.

Para tal é feita uma pesquisa no dicionário de quartos, uma comparação entre o login dado e o login do gerente associado ao quarto, e a verificação da variável *boolean* que nos indica se o quarto tem candidaturas ativas. Caso não existam problemas, o comando procede com a execução.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operação** | **Melhor Caso** | **Caso Esperado** | **Pior Caso.** |
| Pesquisa pelo quarto | O(1) | O(1 + ℷ) = a | O(n) = c |
| Criação do iterador | O(1) | O(1) | O(1) |
| Iteração de quartos | O(n1) | O(n1) = b | O(n1) = d |

|  |  |  |  |
| --- | --- | --- | --- |
| **Total** | **O(n1)** | **O(max (a,b))** | **O(max(c,d))** |

Legenda:

* + - n - Número total de rooms no dicionário rooms. (número de entradas da tabela)
    - ℷ - Fator de ocupação do dicionário, dado pela divisão do número de entradas da tabela pela dimensão da tabela.
    - n1 - número de candidatos ao quarto.

## Listar todas as obras de um leilão

Este comando lista todos os quartos do sistema por ordem lexicográfica da localidade dos mesmos. Para o comando ter sucesso tem que haver quartos inseridos no sistema.

|  |  |  |  |
| --- | --- | --- | --- |
| Operação | Melhor Caso | Caso Esperado | Pior Caso. |
| Criação do iterador | O(log(n)) | O(log(n)) | O(log(n)) |
| Iteração de quartos | O(n) | O(n) | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Total** | **O(n)** | **O(n)** | **O(n)** |

Legenda:

* + - n - Número total de quartos.

## Listar todas as obras de um artista

Este comando lista os quartos vagos de uma dada localidade. Para tal acontecer a localidade tem que existir e tem que existir quartos vagos na localidade pretendida. Para chegarmos a essa conclusão executamos uma busca no dicionário roomsByLocalityTable.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operação** | **Melhor Caso** | **Caso Esperado** | **Pior Caso.** |
| Pesquisa pela localidade | O(1) | O(1 + ℷ) | O(n) |
| Criação do iterador | O(log(n2) | O(log(n2)) | O(log(n2) |
| Iteração de quartos | O(n2) | O(n2) | O(n2) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Total** | **O(n2)** | **O(n2)** | **O(n2)** |

Legenda:

* + - n - Número total de localidades no dicionário roomsByLocalityTable. (número de entradas da tabela)
    - ℷ - Fator de ocupação do dicionário, dado pela divisão do número de entradas da tabela pela dimensão da tabela.
    - n2 - número de quartos com a localidade introduzidas.
    - total de quartos.

## Listar todas as bids de uma obra de arte

Este comando lista os quartos vagos de uma dada localidade. Para tal acontecer a localidade tem que existir e tem que existir quartos vagos na localidade pretendida. Para chegarmos a essa conclusão executamos uma busca no dicionário roomsByLocalityTable.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operação** | **Melhor Caso** | **Caso Esperado** | **Pior Caso.** |
| Pesquisa pela localidade | O(1) | O(1 + ℷ) | O(n) |
| Criação do iterador | O(log(n2) | O(log(n2)) | O(log(n2) |
| Iteração de quartos | O(n2) | O(n2) | O(n2) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Total** | **O(n2)** | **O(n2)** | **O(n2)** |

Legenda:

* + - n - Número total de localidades no dicionário roomsByLocalityTable. (número de entradas da tabela)
    - ℷ - Fator de ocupação do dicionário, dado pela divisão do número de entradas da tabela pela dimensão da tabela.
    - n2 - número de quartos com a localidade introduzidas.
    - total de quartos.

## Listar todas as obras vendidas por preço

Este comando lista as obras e arte vendidas por ordem descendente de preço mais alto de venda, e em caso de empates o critério de desempate é por ordem lexicográfica dos nomes das obras. Para tal acontecer tem de existir obras de artes vendidas, ou seja, o ***artworksRanking*** não pode estar vazio.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operação** | **Melhor Caso** | **Caso Esperado** | **Pior Caso.** |
| Verificação de existem obras vendidas | O(1) | O(1) | O(1) |
| Criação do iterador | O(n1n2) | O(n1n2) | O(n1n2) |
| Iteração de quartos | O(n1n2) | O(n1n2) | O(n1n2) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Total** | **O(n2)** | **O(n2)** | **O(n2)** |

Legenda:

* + - n1 - número de preços mais altos de venda diferentes existentes.
    - n2 – número de máximo de obras que têm o mesmo preço máximo de venda.

## Terminar a execução

Este comando termina a execução do programa.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operação** | **Melhor Caso** | **Caso Esperado** | **Pior Caso.** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Total** | **O(1)** | **O(1)** | **O(1)** |

# Estudo da Complexidade Espacial

A complexidade espacial é a soma do tamanho ocupado por todas as estruturas de dados:

**O(t1) + O(t2) + 2 \* O(n1 \* t5) + O(n2 \* t4) + O(n3 \* t3)**

Legenda:

* t1 - tamanho da ***SepChainHashTable*** “users”.
* t2 - tamanho da ***SepChainHashTable*** “artworks”.
* t3 - tamanho da ***AVLTreeWithComparator<Integer, AVLTree>*** “artworksRanking”.
* t4 - tamanho da ***SepChainHashTable*** “artworksAuctionDictionary”.
* t5 - tamanho da ***DoubleList*** “artworksAuctionList"
* n1 - número de user
* n2 - número de obra de arte
* n3 - número de leilão

# Conclusão

Tendo em conta a ideia base da disciplina - a aprendizagem das formas de utilização e funcionamento das várias estruturas de dados - sabíamos que o objetivo principal deste trabalho seria exatamente isso: fazer-nos pensar e refletir sobre a melhor forma de implementar as diferentes funcionalidades do programa, tendo sempre em mente a eficiência do código e poupança de tempo e recursos. Acreditamos que este objetivo foi alcançado na totalidade, e que somos agora capazes de desenvolver programas com melhor eficiência de tempo e memória.