
Projeto de Algoritmos e Estruturas de Dados - Segunda fase

# ArtAuctions - Obras de Arte e Leilões

Versão I completa

Tiago Rolo Da Costa - 64398

Diogo Filipe Caia Pinheiro - 65122

Turno Prático 3

NOVA School Of Science & Technology

# Ìndice

# 1. Tipos Abstratos De Dados (TADs) e implementações

Para a nossa solução do problema utilizámos diferentes TADs e implementações de acordo com os diferentes objetos e utilizações dos mesmos.

As TADs escolhidas foram:

- Dictionary<K,V> - Implementado por:
    - SepChainHashTable<K,V>;
    - AVLTree<K,V>;
- FindAndGetList<E> extensão de List<E> - implementado por:
    - FindAndGetDoubleList<E> extensão de DoubleList<E>;

Justificação das implementações:

### 1.1 SepChainHashTable<K,V>

Utilizada apenas na classe sistema AuctionHouseSystem com a função de armazenar os utilizadores e artistas, obras de arte e leilões.

Foram usadas 3 SepChainHashTables uma para os utilizadores e artistas que guarda interfaces User, outra para as obras de arte que guarda interfaces WorkOfArt e outra para leilões que guarda interfaces Auction.

Utilizamos esta implementação para esta situação pois a SepChainHashTable<K,V> é a única estrutura de dados que nos proporciona acesso em complexidade temporal O(1), e como o nosso programa tem de suportar em média 20 000 utilizadores, 10 000 obras e largas centenas de leilões qualquer outra estrutura de dados teria de realizar pesquisas sendo assim acesso O(1) a melhor opção para esta situação.

### 1.2 AVLTree<K,V>

Utilizada na classe sistema AuctionHouseSystem e na classe ArtistClass, com a função em AuctionHouseSystem de manter todas as artes já vendidas ordenadas por preço e quando este é igual por nome, e em ArtistClass de manter todas as artes criadas pelo artista ordenadas pelo nome.

Tanto em AuctionHouseSystem como em ArtistClass o nosso objetivo era manter ordem e para isso poderíamos simplesmente ordenar sempre que é necessário a estrutura de dados, mas isso seria uma solução muito cara temporalmente, então optamos por utilizar uma estrutura que está sempre ordenada, neste caso uma árvore binária de pesquisa, mas para melhorar ainda mais os tempos de acesso utilizamos uma árvore AVL que sempre encontra sempre balanceada.

### 1.3 FindAndGetDoubleList<E>

Utilizada na classe AuctionClass e na classe SingleArtAuctionClass, com a função em AuctionClass de guardar SingleArtAuctions, uma SingleArtAuction guarda a arte a ser vendida, o valor mínimo de venda e todas as propostas de compra. Em SingleArtAuctionClass tem a função de guardar todas as propostas de compra da arte desta SingleArtAuction.

Como nestes dois casos é importante mantermos a ordem de inserção optamos por utilizar a FindAndGetDoubleList<E> pois esta não só permite manter os dados ordenados por ordem de inserção, como nos permite encontrar elementos nela utilizando dummies através do método findAndGet().

## 2. Comandos, Complexidade Temporal e Estudo De Casos

### 2.1 addUser

Quando o comando addUser é chamado, ele passa inicialmente por uma verificação da idade introduzida, que, se for menor de 18, manda uma excepção. De seguida, verifica se o User já existe na base de dados, passando pelo comando find que é uma pesquisa na SepChainHashTable<K,V>. Por fim, se a idade é maior ou igual a 18, e, não existir na base de dados, é introduzido na SepChainHashTable<K,V>. Tanto a pesquisa como a introdução são O(1).

**Melhor caso** - O(1).

**Pior caso** - O(n) quando há colisões, sendo n o número de objetos no mesmo bucket.

**Caso esperado** - O(1).

### 2.2 addArtist

Quando o comando addArtist é chamado, ele passa inicialmente por uma verificação da idade introduzida, que, se for menor de 18, manda uma excepção. De seguida, verifica se o Artist já existe na base de dados, passando pelo comando find que é uma pesquisa na SepChainHashTable<K,V>. Por fim, se a idade é maior ou igual a 18, e, não existir na base de dados, é introduzido na SepChainHashTable<K,V>. Tanto a pesquisa como a introdução são O(1).

**Melhor caso** - O(1).

**Pior caso** - O(n) quando há colisões, sendo n o número de objetos no mesmo bucket.

**Caso esperado** - O(1).

### 2.3 removeUser

Quando o comando removeUser é chamado, ele pesquisa o utilizador inicialmente na SepChainHashTable<K,V> com o find, uma operação O(1). Se o find não encontrar, manda uma excepção. Se existir, de seguida, verifica se tem Bids, ou seja, se tem propostas de compra em leilões. Essa verificação passa por verificar se a variável dentro do próprio utilizador que incrementa quando faz propostas, está a zero. Por último verifica se o utilizador é um artista e, se for, verifica se o mesmo tem obras em venda, ou seja, se a variável de controlo de obras em venda é zero. No caso de se verificar, sendo ele artista, chama um método que remove todas as obras do artista em si. Esse mesmo método passa por um iterador sendo portanto O(n), em que n é o número de obras do artista, e realiza remoções O(1) sobre a SepChainHashTable<K,V> de artes. Após tal, se nenhuma das verificações mandar excepção, remove o utilizador, uma operação O(1) da SepChainHashTable<K,V>.

**Melhor caso -** O(1).

**Pior caso -** O(n), sendo n o número de artes do artista (no caso de o utilizador ser um artista).

**Caso esperado -** O(1) para utilizadores e O(n) para utilizadores que também sejam artistas.

### 2.4 addWork

Quando o comando addWork é chamado, ele passa inicialmente por uma verificação de se a arte em si que se pretende ser adicionada, existe. Essa verificação passa de uma pesquisa na SepChainHashTable<K,V> das obras de arte, que é uma pesquisa O(1). De seguida verifica se o utilizador em si, que se introduz como o artista da obra, existe no sistema, outra pesquisa O(1) na SepChainHashTable<K,V> de utilizadores. Por último, vê se o utilizador é ou não um artista. Todas as não verificações de condições enviam uma excepção para o sistema. Se não houver excepções a arte é adicionada na SepChainHashTable<K,V> e é inserida na lista de obras do artista, uma AVLTree<K,V>. Sendo a primeira O(n) e a segunda log n no melhor caso e (1.44) log n no pior caso.

**Melhor caso -** O(log n) sendo n o número de elementos na AVLTree.

**Pior caso -** O(n) sendo n o número de elementos nos buckets da SepChainHashTable

**Caso esperado -** O(log n) sendo n o número de elementos na AVLTree,

### 2.5 infoUser

Quando o comando infoUser é chamado, é feita uma pesquisa O(1) na SepChainHashTable<K,V> de utilizadores. Se retornar nula, manda excepção. Se não for nula, o programa apenas escreve no terminal as informações pedidas para o utilizador.

**Melhor caso -** O(1).

**Pior caso -** O(n), quando há pesquisa no bucket, sendo n o número de objetos no mesmo bucket (o bucket com mais que um objeto).

**Caso esperado -** O(1).

### 2.6 infoArtist

Quando o comando infoArtist é chamado, é feita uma pesquisa O(1) na SepChainHashTable<K,V> de utilizadores. Se retornar nula, manda excepção. Por fim, verifica se o User é Artist, no caso de não ser, manda uma excepção. Se for, o programa apenas escreve no terminal as informações pedidas para o utilizador.

**Melhor caso -** O(1).

**Pior caso -** O(n), quando há pesquisa no bucket, sendo n o número de objetos no mesmo bucket (o bucket com mais que um objeto).

**Caso esperado -** O(1).

### 2.7 infoWork

Quando o comando infoWork é chamado, é feita uma pesquisa O(1) na SepChainHashTable<K,V> de artes. Se retornar nula, manda excepção. Se não for nula, o programa apenas escreve no terminal as informações pedidas para o utilizador.

**Melhor caso -** O(1).

**Pior caso -** O(n), quando há pesquisa no bucket, sendo n o número de objetos no mesmo bucket (o bucket com mais que um objeto).

**Caso esperado -** O(1).

### 2.8 createAuction

Quando o comando createAuction é chamado, ele passa inicialmente por uma verificação de se a auction em si que se pretende ser adicionada, existe. Se não existir, ele adiciona-a na SepChainHashTable<K,V> de Auctions. Tanto a pesquisa como a inserção são O(1).

**Melhor caso** - O(1).

**Pior caso** - O(n) quando há colisões, sendo n o número de objetos no mesmo bucket.

**Caso esperado** - O(1).

### 2.9 addWorkAuction

Quando o comando addWorkAuction é chamado, ele inicialmente verifica se a auction em si existe. Depois, verifica se a arte a adicionar existe. Se a auction ou a arte não existirem, manda uma excepção. Ambas estas pesquisas em SepChainHashTable<K,V> de auctions e de artes, são O(1).
Após ambos passarem, a arte é inserida na auction, numa FindAndGetDoubleList<E>, no fim da mesma, sendo então uma operação O(1).

**Melhor caso** - O(1).

**Pior caso** - O(n) quando há colisões, sendo n o número de objetos no mesmo bucket.

**Caso esperado** - O(1).

### 2.10 bid

Quando o comando bid é chamado, começa por verificar se o utilizador existe, com uma pesquisa O(1) na SepChainHashTable<K,V> de utilizadores. Após isso, pesquisa na SepChainHashTable<K,V> de auctions se a auction dada existe. Por último, verifica se a auction tem a arte dada, uma pesquisa O(n) na FindAndGetDoubleList<E> de artes, na auction, em que n é o tamanho da lista. Caso algum dos 3 não se verifique são mandadas excepções. Por fim, a bid é adicionada, uma pesquisa O(n) dentro da FindAndGetDoubleList<E> na auction e a adição O(1) numa FindAndGetDoubleList<E> na SingularArtAuction respetiva á arte e à auction dadas.

**Melhor caso** - O(1).

**Pior caso** - O(n), sendo n o tamanho da FindAndGetDoubleList<E>.

**Caso esperado** - O(n), sendo n o tamanho da FindAndGetDoubleList<E>.

### 2.11 closeAuction

Quando o comando closeAuction é chamado começa por verificar se a auction existe com uma pesquisa O(1) na SepChainHashTable<K,V> de leilões, em seguida o leilão irá percorrer cada SingleArtAuction dentro dele para recolher para uma lista as propostas vencedoras de cada SingleArtAuction realizando assim uma pesquisa O(n).

A decisão da propostas vencedora dentro de cada SingleArtAuction consiste em primeiro verificar se existem propostas, se não houverem o contador de artes em leilão do artista criador da obra a ser leiloada é decrementado e em seguida e retornado uma proposta com valor -1 para propósitos de diferenciar propostas falhadas de não falhadas na listagem final do comando. Se houverem propostas na SingleArtAuction é feita uma pesquisa linear de todas as propostas e é escolhida a proposta mais antiga com o valor mais alto realizando assim outra pesquisa O(n).

Em seguida se a arte a ser leiloada já foi vendida no passado, então esta é removida da árvore AVL que guarda as artes já vendidas no sistema, pois esta árvore AVL usa como chave o próprio objeto da arte e como vamos atualizar o valor da arte precisamos de atualizar a sua posição na árvore, pois o seu ranking irá mudar.

Por fim atualizamos o valor na arte pelo novo valor de venda, decrementamos o contador do artista que conta as artes que este tem em leilão, apagamos o leilão do sistem, operação O(1) na SepChainHashTable<K,V> de leilões e listamos para o terminal as obras vendidas e não vendidas.

**Melhor caso** - O(1), quando o leilão não existe no sistema.

**Pior caso** - O($n^2$), pois tem de percorrer todas as SingleArtAuctions e propostas nestas.

**Caso esperado** - O($n^2$), pois tem de percorrer todas as SingleArtAuctions e propostas nestas.

### 2.12 listAuctionWorks

Quando o comando listAucionWorks é chamado, ele procura a auction dada na SepChainHashTable<K,V> de auctions, uma pesquisa O(1). De seguida, verifica se a auction tem artes, que é só verificar se a FindAndGetDoubleList<E> de artes da auction está vazia. Caso nada mande excepções, ele retorna um iterador para escrever no terminal as informações pretendidas, uma operação O(n).

**Melhor caso** - O(1), por mandar excepção antes de iterar.

**Pior caso** - O(n), em que n é o tamanho da FindAndGetDoubleList<E> a iterar.

**Caso esperado** - O(n), em que n é o tamanho da FindAndGetDoubleList<E> a iterar.

### 2.13 listArtistWorks

Quando o comando listArtistWorks é chamado, ele procura o utilizador dado na SepChainHashTable<K,V> de utilizadores, uma pesquisa O(1). De seguida, verifica se o user é um artista. Por fim, vê se o artista tem obras em seu nome, que é apenas verificar se o tamanho da AVLTree<K,V> de obras do artista é zero. Caso nada mande excepções, ele retorna um iterador para escrever no terminal as informações pretendidas, uma operação O(n).

**Melhor caso** - O(1), por mandar excepção antes de iterar.

**Pior caso** - O(n), em que n é o tamanho da AVLTree<K,V> a iterar.

**Caso esperado** - O(n), em que n é o tamanho da AVLTree<K,V> a iterar.

### 2.14 listBidsWork

Quando o comando listBidsWork é chamado, ele procura a auction dada na SepChainHashTable<K,V> de auctions, uma pesquisa O(1). De seguida, verifica se a auction tem a arte dada, passando por uma pesquisa O(n), na FindAndGetDoubleList<E> de singular auctions da auction. Por fim, verifica se a obra em si tem propostas, com uma variavel no objeto. Caso nada mande excepções, ele retorna um iterador para escrever no terminal as informações pretendidas, uma operação O(n).

**Melhor caso** - O(1), por mandar excepção antes de iterar ou pesquisar a arte.

**Pior caso** - O(n), em que n é o tamanho da FindAndGetDoubleList<E> a iterar.

**Caso esperado** - O(n), em que n é o tamanho da FindAndGetDoubleList<E> a iterar.

### 2.15 listWorksByValue

Quando o comando listWorksByValue é chamado, verifica-se se a AVLTree<K,V> de artes vendidas está vazia. Caso esteja, é mandada excepção, caso contrário, é iterada e devolvida, uma operação O(n).

**Melhor caso** - O(1), por mandar excepção antes de iterar.

**Pior caso** - O(n), em que n é o tamanho da AVLTree<K,V> a iterar.

**Caso esperado** - O(n), em que n é o tamanho da AVLTree<K,V> a iterar.

### 2.16 quit

Quando o comando quit é chamado, é escrito num ficheiro o objeto da classe sistema e o programa termina. Escrever o objeto é sempre complexidade O(1) mesmo que haja excepções.

**Melhor caso** - O(1).

**Pior caso** - O(1).

**Caso esperado** - O(1).

## 3. Estudo da complexidade espacial

A nossa solução proposta para o problema apresenta uma complexidade espacial esperada de 20 000 utilizadores e 10 000 artes. Em certas alturas poderá ter largas centenas de leilões abertos. Cada leilão terá apenas algumas dezenas de artes e também o número de propostas por arte será da ordem das dezenas. Os leilões só são conhecidos pelo sistema até encerrarem.

Também haverá uma complexidade adicional pois há o armazenamento das obras já vendidas e das obras que um artista criou.