

**Estruturas de Dados e Algoritmos II**

**Cestos de Compras**

**Elaborado por:**

João Calhau nº31621

José Pimenta nº31677

Indice

[1.Introdução 4](#_Toc419998536)

[2.Estruturas de dados 5](#_Toc419998537)

[2.1.Constituição das estruturas 5](#_Toc419998538)

[2.2.Diagrama e discrição das estruturas de dados usadas 6](#_Toc419998539)

[3.Formatos 8](#_Toc419998540)

[4.Funcionamento do programa 9](#_Toc419998541)

[4.1.Criação de cesto de compras 9](#_Toc419998542)

[4.2.Introdução de artigos num cesto de compras 9](#_Toc419998543)

[4.3.Retirada de artigos de um cesto de compras 9](#_Toc419998544)

[4.4.Visualização do conteúdo de um cesto de compras 10](#_Toc419998545)

[4.5.Realização da encomenda 10](#_Toc419998546)

[4.6.Cancelamento de um cesto de compras 10](#_Toc419998547)

[4.7.Limpeza dos cestos de compras 10](#_Toc419998548)

[4.8.Fim de execução 11](#_Toc419998549)

[5.Justificação das escolhas feitas 12](#_Toc419998550)

[5.1.Acessos a disco 12](#_Toc419998551)

[5.2.Estrutura de Dados 12](#_Toc419998552)

[5.2.1.HashTable 12](#_Toc419998553)

[5.3.Algoritmos 13](#_Toc419998554)

[5.3.1.Merge\_Sort 13](#_Toc419998555)

[5.3.2.Bubble\_Sort\_String 13](#_Toc419998556)

[5.3.3.Hash 13](#_Toc419998557)

[5.4.Funções 13](#_Toc419998558)

[5.4.1.Insert\_Item, Remove\_Item, Visualize\_Item e isEmpty 13](#_Toc419998559)

[5.4.2.Create\_HashTable 14](#_Toc419998560)

[5.4.3.Free\_HashTable 14](#_Toc419998561)

[5.4.4.Get\_Cesto 14](#_Toc419998562)

[5.4.5.Put\_Cesto 14](#_Toc419998563)

[5.4.6.Remove\_Cesto 14](#_Toc419998564)

[5.4.7.CC 14](#_Toc419998565)

[5.4.8.IC 14](#_Toc419998566)

[5.4.9.RC 15](#_Toc419998567)

[5.4.10.VC 15](#_Toc419998568)

[5.4.11.EC 15](#_Toc419998569)

[5.4.12.XC 15](#_Toc419998570)

[5.4.13.LC 15](#_Toc419998571)

[6.Código do programa 16](#_Toc419998572)

# 1.Introdução

O trabalho de Estruturas de Dados e Algoritmos II tem como objectivo simular um sistema online de comércio eletrónico que suporte a noção de cesto de compras, que o cliente vai enchendo durante a sua visita à loja virtual.

Iremos assim implementar o dito sistema de modo a podermos realizar operações sobre os cestos, incluindo a criação e destruição, alteração de artigos nos mesmos e realizações de encomendas.

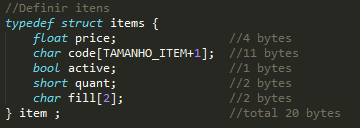
Para isso iremos implementar, através dos conhecimentos adquiridos na disiciplina de EDA2, algoritmos e estruturas de dados que sejam adequadas ao referido problema.

# 2.Estruturas de dados

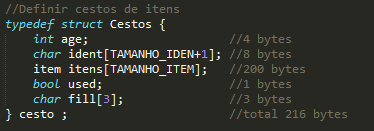
## 2.1.Constituição das estruturas

Para realizar o trabalho criámos as seguintes estruturas de dados:

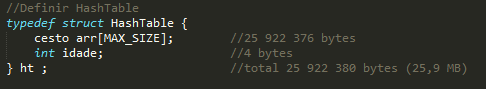
Items:



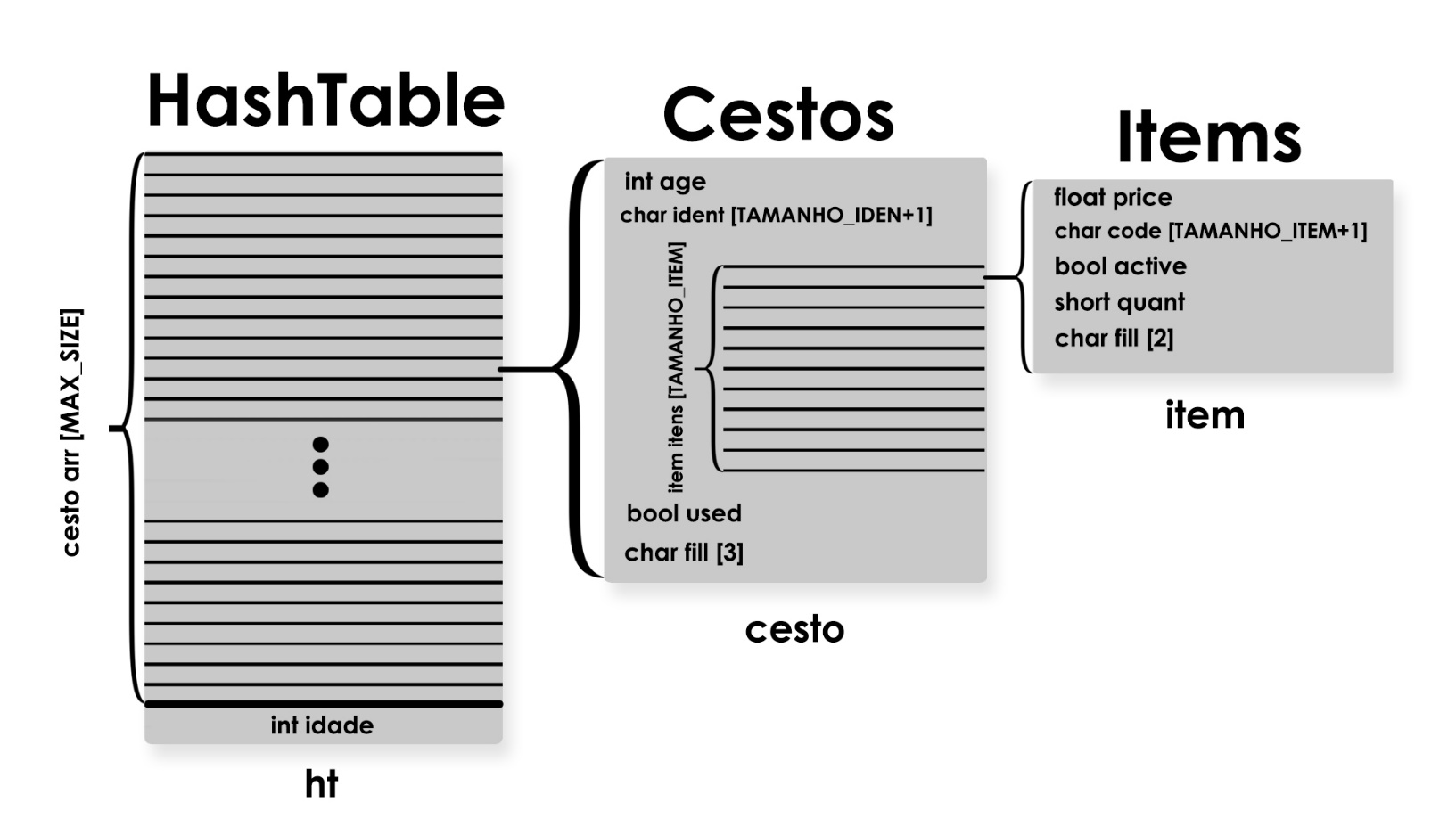
Cestos:



HashTable:



## 2.2.Diagrama e discrição das estruturas de dados usadas



Para a realização deste trabalho criamos as 3 estruturas referidas.

Na criação da HashTable incluímos uma variável, ***int idade***, que contem o tempo passado desde a criação da HashTable até ao momento atual. Para além disso a HashTable contem um array de cestos, ***cesto arr[MAX\_SIZE],*** em que *MAX\_SIZE* é a quantidade maxima de cestos possível na HashTable, sendo que o valor dessa variável, no nosso trabalho, é 200 003 pois é o primo mais próximo de 200 000, valor em que o factor de carga da HashTable seria 0,5 pois o máximo de cestos que podem ser inseridos é 100 000.

Na criação dos Cestos incluímos uma variável age, ***int age***, que guarda o instante em que ouve uma operação sobre cesto, incluímos uma variável ident, ***char ident[TAMANHO\_IDENT+1],*** que tem o id do cesto e que tem espaço suficiente para 7 caracteres e +1 para o caracter nulo (‘\0’). Temos ainda uma variável used, ***bool used***, que serve para dizer se o cesto se encontra activo, ou não, na HashTable, uma variável fill, ***char fill[3],*** que só serve para uma organização de memória e um endereçamento corretos. Finalmete temos ainda um array de itens***, item itens[TAMANHO\_ITEM],*** que tem espaço para 10 tipos de itens (visto que cada cesto só consegue ter 10 produtos diferentes).

Na criação dos Items incluímos uma variável price**, *float price***, que contem o preço unitário de cada item, uma variável code, ***char code[TAMANHO\_ITEM+1]*,** que contem o id do item de tamanho máximo 10 e +1 para o caracter nulo (‘\0’). Contem ainda uma variável active, ***bool active*,** que informa se o item que está no cesto está activo ou não, uma variável quant**, *short quant****,* que identifica a quantidade do item no cesto e uma variável fill, ***char fill[2],*** que, outra vez, só serve para uma organização de memória e um endereçamento corretos.

# 3.Formatos

Escolhemos guardar os dados num ficheiro binário porque esse tipo de ficheiro consegue guardar todo o tipo de dados, codificados em binário.

Quantidade de páginas totais para guardar em disco:

256MB = 1024 \* 1024 \* 256 = 268 435 456 Bytes

Páginas de 4096 Bytes

268 435 456 / 4096 = 65 536 Páginas

Quantidade de páginas utilizadas:

Tamanho da estrutura: 25 922 380 bytes

Páginas de 4096 bytes

25 922 380 / 4096 ≈ 6329 Páginas

Com esta estrutura de dados usamos apenas aproximadamente 1/10 das páginas totais.

# 4.Funcionamento do programa

No inicio do programa é criada uma tabela (é alocado espaço para a tabela, e são colocados todos os cestos, desativados, na tabela. O tempo da tabela é inicializado a 0).

O programa tenta ler de um ficheiro uma tabela já existente, caso não exista é utilizada a tabela criada anteriormente, senão é lida de disco e colocada no espaço alocado.

O programa vai receber os inputs do utilizador, e consoante estes vai escolher a operação correta a ser aplicada à tabela.

## 4.1.Criação de cesto de compras

Para criarmos um cesto de compras verificamos, a partir do nome do cesto, se este já se encontra na tabela, para isso fazemos hash do nome. Se na posição dada pelo hash se encontra um cesto ativo, verifica-se se o nome é igual. Em caso afirmativo o cesto já existe na tabela, caso contrário faz um novo hash. Se não se encontrar um cesto ativo é porque chegou a uma posição onde poderia estar, não estando insere-se nessa posição com uma idade igual ao tempo de inserção.

## 4.2.Introdução de artigos num cesto de compras

Para introduzir artigos num cesto, à semelhança da criação, vamos verificar se o cesto existe. Se não existe, não se insere nenhum item. Caso contrario, percorre os itens do cesto e vai guardando a ultima posição livre encontrada. Se encontrar o item, aumenta-lhe a quantidade, senão insere um item na última posição livre guardada anteriormente. É alterada ainda a idade do cesto para o tempo atual.

## 4.3.Retirada de artigos de um cesto de compras

Para retirar artigos de um cesto, à semelhança da criação, vamos verificar se o cesto existe. Se não existe, não se retira nada. Caso contrario, percorre os itens todos do cesto até encontrar o item desejado. Caso encontre, se a quantidade que se pretende remover for superior á que se encontra no cesto, não se remove, se for menor remove a quantidade desejada no cesto e se for igual remove o item do cesto. Caso não encontre o item no cesto, não se remove nada. É alterada ainda a idade do cesto para o tempo atual.

## 4.4.Visualização do conteúdo de um cesto de compras

Para visualizar os itens de um cesto, á semelhança da criação, vamos verificar se o cesto existe. Se não existe, não se faz nada. Caso contrario, percorre todos os itens do cesto e vai guardando os seus apontadores dos itens ativos para uma lista. Caso não encontre nenhum item ativo é porque o cesto está vazio, caso contrário ordena os itens da lista pela ordem crescente do código, percorre a lista, agora ordenada, e calcula o preço total de cada item e o preço total do cesto. É alterada ainda a idade do cesto para o tempo atual.

## 4.5.Realização da encomenda

Para realização da encomenda de um cesto, á semelhança da criação, vamos verificar se o cesto existe. Se não existe, não se faz nada. Caso contrário e se o cesto estiver vazio atualiza-se a idade e não se efetua mais nenhuma operação sobre o mesmo, se o cesto não estiver vazio, percorre todos os itens do cesto e “desativa-os”, estando isto feito, “desativa-se” também o cesto e é efetuada a venda do mesmo.

## 4.6.Cancelamento de um cesto de compras

Para o cancelamento de um cesto, á semelhança da criação, vamos verificar se o cesto existe. Se não existe, não se faz nada. Caso contrário percorre todos os itens do cesto e “desativa-os”, estando isto feito o cesto é “desativado” e assim cancelado.

## 4.7.Limpeza dos cestos de compras

Para fazer a limpeza dos cestos, através do tempo atual e da idade de remoção dada pelo utilizador, é calculado o tempo a partir do qual não são removidos cestos. Assim sabemos que cestos com tempo inferior a esse têm de ser removidos. São percorridos todos os cestos da tabela, e sempre que se encontre um cesto ativo e com tempo inferior ao calculado, é guardado o apontador para esse cesto numa lista e “desativa-se” o mesmo (O cesto está desativado, mas a informação da idade do mesmo ainda é acessível a partir dos apontadores, visto que os cestos nunca são apagados definitivamente). Após percorrer a tabela, ordena-se a lista por ordem decrescente da idade. Percorre-se a lista, agora ordenada, e imprime-se o nome dos cestos. Caso não encontre cestos, é mostrado que não existem cestos com idade superior à inserida pelo utilizador.

## 4.8.Fim de execução

Caso a instrução dada pelo utilizador não seja nenhuma das anteriores (XX), ou tenha chegado ao fim de ficheiro, é guardado o tempo atual na tabela e guardada a mesma em disco.

Finalmente o espaço previamente alocado para a tabela é libertado.

# 5.Justificação das escolhas feitas

## 5.1.Acessos a disco

No nosso programa só fazemos 2 acessos a disco, um no início para carregar a HashTable (ou não) para memória principal e um no fim para guardar a HashTable com as operações todas feitas sobre ela.

Os acessos ao disco são sempre constantes pois a quantidade de informação é sempre a mesma a ser guardada/lida, visto que a tabela tem sempre o mesmo tamanho (MAX\_SIZE, neste caso 200 003).

Decidimos fazer apenas 2 acessos à memória secundária visto que como não usamos apontadores na estrutura da tabela podemos guardar a tabela toda em memória secundária e quando vamos lê-la do disco carregamos a tabela toda para memória principal.

## 5.2.Estrutura de Dados

### 5.2.1.HashTable

Escolhemos uma HashTable para fazer este trabalho porque de todas as estruturas de dados que demos até agora era a mais simples de implementar e parecia ser a estrutura melhor para este tipo de problema, visto que vamos ter um máximo de 100 000 cestos e como a HashTable tem um acesso constante (no melhor caso quando só se faz hash 1 vez, O(1)) se dermos um tamanho grande o suficiente (neste caso os 200 003 cestos), muito raramente vão haver colisões, o que evita o pior caso (O(n)) e o facto de termos de fazer rehash à tabela toda, visto que assim o factor de carga será sempre 0,5.

## 5.3.Algoritmos

### 5.3.1.Merge\_Sort

Na função LC, inicialmente começámos por implementar o bubble sort (O(n²)), mas rapidamente nos apercebemos que para uma quantidade de cestos muito grande um algoritmo com uma complexidade temporal menor seria melhor (claro que também não funcionou no Mooshak). Por essa razão mudámos o algoritmo de ordenação para o Merge sort que tem uma complexidade temporal de O(n log n) o que baixou bastante o tempo de execução (desta vez já funcionou no Mooshak).

Nota: n é tamanho do array a ser ordenado (neste caso o número de

### 5.3.2.Bubble\_Sort\_String

Na função VC, implementamos o bubble sort adaptado a comparar strings, que tem uma complexidade temporal de O(n²), porque é mais simples de implementar e o tamanho máximo de produtos que vamos ordenar é sempre 10. Para números pequenos o tempo de ordenação, entre o Merge Sort e o Bubble sort não diferencia muito.

Nota: n é o tamanho do array a ser ordenado

### 5.3.3.Hash

Algoritmo de hash Djb2. Este algoritmo baseia-se no número “mágico” 33, o porquê de ser esse o número nunca foi bem fundamentado

Fonte: http://www.cse.yorku.ca/~oz/hash.html

## 5.4.Funções

### 5.4.1.Insert\_Item, Remove\_Item, Visualize\_Item e isEmpty

Complexidade O(m), devido ao ciclo for em que m é o tamanho do array de itens no cesto.

### 5.4.2.Create\_HashTable

Complexidade O(n), devido ao ciclo for, em que n é o tamanho do array de cestos na HashTable.

### 5.4.3.Free\_HashTable

Complexidade O(1), visto que é só um free da memória.

### 5.4.4.Get\_Cesto

Complexidade, no pior dos casos O(n) (por causa do ciclo while).

### 5.4.5.Put\_Cesto

Complexidade, no pior dos casos O(n+m) por causa do ciclo while e do ciclo for (em que n é o tamanho do array da tabela e m é o número máximo de itens).

### 5.4.6.Remove\_Cesto

Complexidade, no pior dos casos O(n\*m) por causa do ciclo for dentro do ciclo while (em que n é o tamanho do array da tabela e m é o número máximo de itens).

### 5.4.7.CC

Complexidade desta função é igual á do Put\_Cesto, O(n+m).

### 5.4.8.IC

Complexidade desta função é O(n+m) (n+m porque há 2 chamadas de funções com com complexidades O(m) e O(n), explicadas previamente).

### 5.4.9.RC

Complexidade desta função é O(n+m) (n+m porque há 2 chamadas de funções com com complexidades O(m) e O(n), explicadas previamente).

### 5.4.10.VC

Complexidade desta função é O(n+m) (n+m porque há 2 chamadas de funções com com complexidades O(m) e O(n), explicadas previamente).

### 5.4.11.EC

Complexidade desta função é O(n+m+n\*m) (porque há 3 chamadas de funções com complexidades O(n\*m), O(n) e O(m), explicadas previamente).

### 5.4.12.XC

Complexidade desta função é O(n+n\*m) (n+n\*m porque há 2 chamadas de funções com com complexidades O(n\*m) e O(n), explicadas previamente).

### 5.4.13.LC

Complexidade desta função é O(n²\*m+n\*log n) (n²\*m+n\*log n devido há chamada de uma função dentro de um ciclo for e há chamada de outra função, com complexidades O(n), O(n\*m) e O(n log n)).

Nota: Todas complexidades temporais são para os piores casos, no nosso trabalho quase nunca, ou nunca, chega a esses casos devido à estrutura da HashTable explicada anteriormente.

# 6.Código do programa

