Produtos

Um **produto** é definido como sendo uma string.

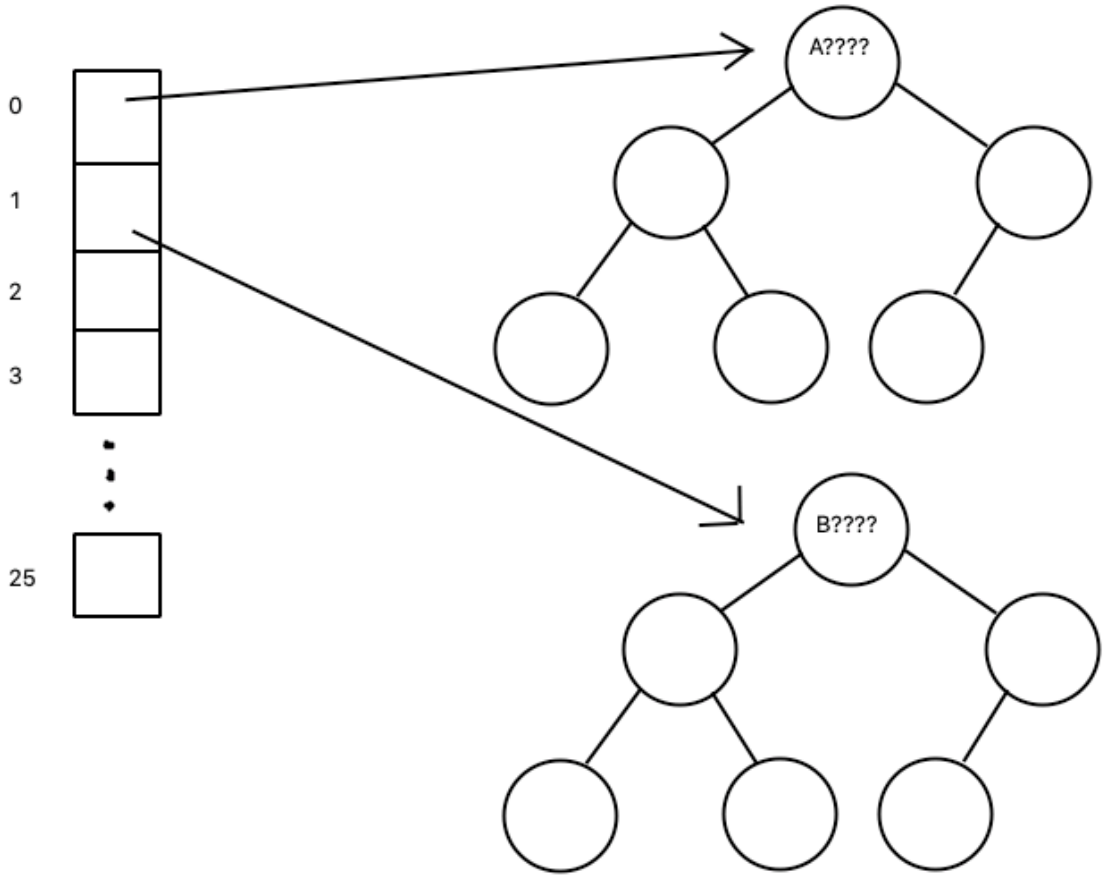
- char getChar0P(Produto p) - Retorna o primeiro caracter do produto.
- char* getCodProd(Produto p) - Retorna a string que o produto representa.
- Produto criaProd(char* codProd) - Retorna o produto criado a partir de uma string.
- int validaProduto(Produto p) - Testa se um determinado produto é válido.

Clientes

Um **cliente** é definido como sendo uma string.

- char getChar0C(Cliente c) - Retorna o primeiro caracter do cliente.
- char* getCodCliente(Cliente c) - Retorna a string que o cliente representa.
- Cliente criaCliente(char* codCliente) - Retorna o cliente criado a partir de uma string.
- int validaCliente(Cliente c) - Testa se um determinado cliente é válido.

Catálogos de produtos e clientes



Catálogo de Clientes e Produtos

Um **catálogo** é um array com 26 elementos cujos índices apontam para AVL's. A árvore num determinado índice do array contém apenas clientes cujo código começa por certa letra. As AVL's estão ordenadas de acordo com o abecedário.

Nota: Estes algoritmos são a versão do catálogo de clientes, dos produtos são semelhantes.

- int getIndexC(Cliente c) - Retorna o index do primeiro char do código de um cliente;
- Cat_Clientes inicializa_CatClientes() - Inicializa um catálogo de clientes;
- Cat_Clientes insereCliente(Cat_Cliente catc, Cliente c) - Insere um cliente num catálogo de clientes;
- int existeCliente(Cat_Clientes catc, Cliente c) - Procura um cliente no catálogo de clientes;
- char** getListaClientes(Lista_Clientes lc) - Retorna o array de strings(cada string é um cliente) de uma lista de clientes;
- void printCatCliente(Cat_Clientes cp) - Dá print do catálogo recebido(ordenado de forma pre-order);

Lista de Clientes e Produtos

Uma **lista** é definida por um array de Strings. Cada string corresponde a um cliente/produto.

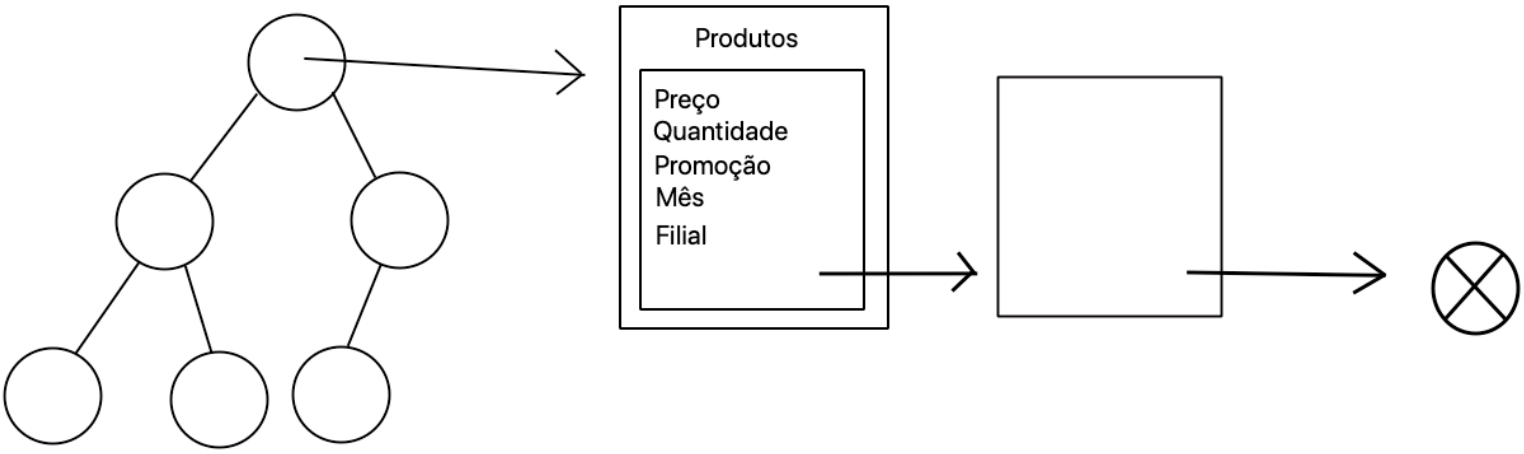
Nota: Estes algoritmos são a versão da lista de clientes, dos produtos são semelhantes.

- Lista_Clientes inOrderCLista(AVL root, Lista_Clientes lc, int* index) - Ordena uma AVL de clientes(todos começam por a mesma letra) de forma in order numa lst_clientes;
- Lista_Clientes initListaClientes() - Inicializa uma lista de clientes;
- Lista_Clientes listaPorLetraC(Cat_Clientes catc, char letra) - Vai à respectiva AVL do catálogo de clientes cuja primeira letra é a letra recebida como argumento e transforma essa AVL numa lista de clientes ordenada(in-order);
- Lista_Clientes catcToLista(Cat_Clientes catc) - Transforma o catálogo de clientes numa lista de clientes ordenada(in-order);
- void print Lista Clientes(Lista_Clientes lc) - Dá print da lista de clientes recebida;

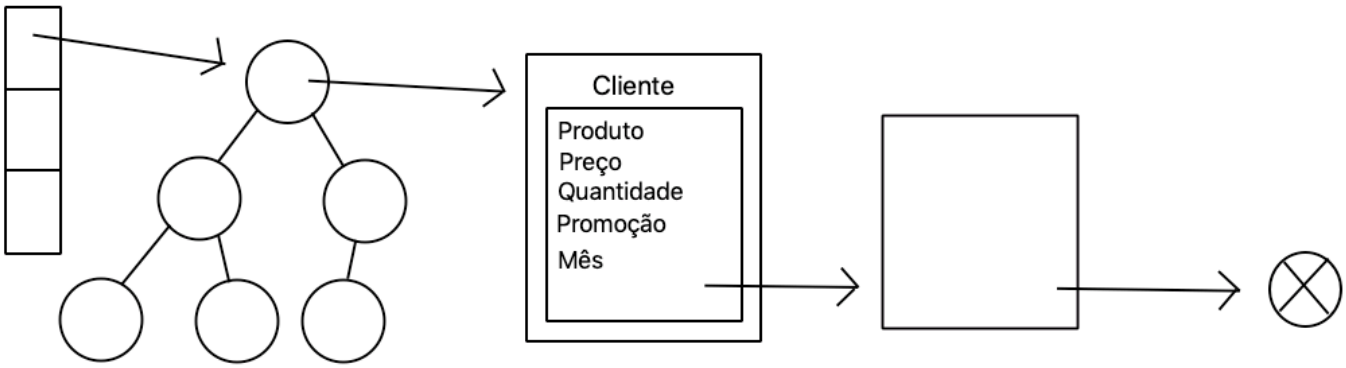
Geral

- void guardaClientes(FILE fp, Cat_Clientes catc, int* cLidos, int* cTotal) - Lê um ficheiro com códigos de clientes, inseres esses clientes num catálogo de clientes e guarda o número total de clientes lidos assim como o número total de clientes válidos;
- void guardaProdutos(FILE fp, Cat_Prods catp, int* pLidos, int* pTotal) - Lê um ficheiro com códigos de produtos, inseres esses produtos num catálogo de produtos e guarda o número total de produtos lidos assim como o número total de produtos válidos;

Facturação



Filial



A **facturação** é uma AVL organizada pelos códigos dos produtos. No entanto, em cada node não só tem o produto como também uma estrutura com o preço, quantidade, promoção, mês e filial e depois um endereço para uma próxima estrutura do mesmo tipo. Desta maneira conseguimos associar, para cada produto, todas as componentes anteriormente referidas quando estamos a ler as vendas, ou seja, teremos a facturação de uma venda (preço * quantidade) e qual foi a promoção, em que mês e em que filial foi realizada a tal compra. Podemos também dizer que o tamanho da lista ligada é o número de vezes que o produto foi comprado.

A **filial** é um array de tamanho 3 representado as 3 filiais. Cada elemento do array é uma AVL que é organizada pelos códigos do clientes. Em cada nodo não só tem o cliente como também uma estrutura com o código do produto, preço, quantidade, promoção e mês e depois um endereço para uma próxima estrutura do mesmo tipo. Desta maneira conseguimos associar, para cada cliente, todas as componentes anteriormente referidas quando estamos a ler as vendas, ou seja, teremos também a faturação, quantidade, promoção e o mês em que foi realizada a compra. Podemos dizer que o tamanho da lista ligada é a quantidade de compras que o cliente realizou.

Para além de todos os gets necessários para manter o encapsulamento…

Faturação

- void query8Aux(Facturacao f, int mes1, int mes2, int* totalVendas, float* totalFaturado) - Função auxiliar da query8 (é usada recursivamente, por isso não podíamos colocá-la na implementação da query).
- int procuraFilialNaInfo(Info i, int filial) - Testa se há algum bloco com uma filial.
- void query11Aux(Facturacao f, Lligada* a, int filial) - Função auxiliar da query11.
- Nodo mkNodoVenda(char* linhaVenda) - Cria um nodo a partir de uma linha de venda.
- Facturacao inicializa_Facturacao() - O nome diz tudo.
- Facturacao insertF(Facturacao node, Nodo n) - Insere um nodo na AVL e retorna o sítio
- Facturacao searchF(Facturacao root, char* produto) - Procura um produto na AVL.
- void preOrderF(Facturacao root) - Faz print da AVL em pre-order.

Filial

- void query9Aux(Filial fil, char* produto, List_Strings lsN, List_Strings lsP, int* indexN, int* indexP) - Função auxiliar da query9 (é usada recursivamente, por isso não podíamos colocá-la na implementação da query).
- Filial inicializa_Filial() - O nome diz tudo.
- NodoFil mkNodeVenda(char* linhaVenda) - Cria um nodo a partir de uma linha de venda.
- NodoFil editNodoFi(char* c) - Edita o cliente de um nodo.
- Filial insertFi(Filial node, NodoFil n) - Insere um nodo na AVL e retorna o sítio.
- Filial searchFi(Filial root, char* cliente) - Procura um cliente na AVL.
- void inOrderFi(Filial root) - Faz print da AVL em in-order.

Razões pelas quais a faturação foi assim estruturada:

- Acesso a um produto de forma eficiente.
- Vasta informação sobre as componentes de cada venda associada a cada produto.
- Como são listas de curta dimensão, conseguimos ter acesso a vários dados interligados entre si sem grandes custos em tempo de execução.

Razões pelas quais a filial foi assim estruturada:

- Acesso a um cliente de forma eficiente.
- Vasta informação sobre as componentes de cada venda associada a cada cliente.
- Como são listas de curta/média dimensão, conseguimos ter acesso a vários dados interligados entre si sem grandes custos em tempo de execução.

Páginas

As **páginas** são uma lista de strings em que cada página é uma porção dessa string. Cada página tem 10 linhas.

- List_Strings initListaStrings() - Inicializa uma lista de strings.
- List_Strings criaLsLp(Lista_Prods lp) - Transforma uma lista de produtos numa lista de strings.
- List_Strings criaLsLc(Lista_Clientes lc) - Transforma uma lista de clientes numa lista de strings.
- List_Strings addLinha(List_Strings ls, char* linha, int index) - Adiciona uma string a uma lista de strings na posição recebida.
- int sizeList_Strings(List_Strings ls) - Retorna o tamanho de uma lista de strings;
- void printList_Strings(List_Strings ls) - Dá print de uma lista de strings.
- void printNList(List_Strings ls, int k) - Dá print dos primeiros elementos de uma lista de strings.

Página

Cada **página** é uma porção de uma lista de strings.

- char* getLine(List_Strings ls, int i) - Retorna um produto de uma lista de strings(esta função é executada apenas no contexto onde a lista de strings corresponde a uma lista de strings com um produto e uma quantidade).
- List_Strings getPorcao(Pagina p) - Retorna uma porção de uma página;
- Pagina initPag(List_Strings ls) - Inicializa as páginas.
- Pagina getPagSeguinte(List_Strings ls) - Retorna a página a seguir à atual;
- void printPag(Pagina p) - Dá print dos elementos de uma página.

Navegador

O **navegador** permite ao user navegar pelas páginas escolhendo o número da página que pretende ler, bem como parar de ler quando assim achar necessário.

- void navegador(List_Strings ls) - Implementação do navegador onde são recebidas as Páginas(lista de strings) e permite ao user navegar pelas páginas.

AVL

Biblioteca auxiliar local de estruturas AVL que contém algoritmos básicos para o uso deste tipo de dados (inserção, procura, print, etc).

Razão da escolha desta estrutura:

- É a mais segura em termos de ratio de eficácia e probabilidade de bom funcionamento quando comparada a outras estruturas como HashTables, listas ligadas, etc.

IO

O ficheiro IO.c faz, tal como o nome indica, todas as instruções de leitura e escrita no ecrã. É usado para ler argumentos que vão ser usados nas queries e, depois, a escrita no ecrã dos resultados destas.

Linked Lists

Biblioteca auxiliar local de listas ligadas que contém algoritmos básicos para o uso deste tipo de dados (inserção, tamanho, ordenação, print, etc).

Sistema de Gestão de Vendas

O **SGV** contém ambos os catálogos, de produtos e de clientes, a facturação e também as filiais. A sua principal função é podermos ter acesso a todas estas estruturas ao mesmo tempo.

- SGV inicializa_SGV(Cat_Prods catp, Cat_Clientes catc, Facturacao fat, Filial filiais[3]) - dados as estruturas, inicializar o sistema de gestão de vendas.
- Cat_Prods getCatp(SGV sgv) - Retorna o catálogo de produtos do sistema.
- Cat_Clientes getCatc(SGV sgv) - Retorna o catálogo de clientes do sistema.
- Facturacao getFat(SGV sgv) - Retorna a faturação do sistema.
- Filial* getFilial(SGV sgv) - Retorna a filial do sistema.
- void setFat(SGV sgv, Facturacao fat) - Altera a facturação no sistema.

Complexidade das estruturas e optimizações realizadas e resultados dos testes realizados

- No início usávamos arrays de strings e a eficiência melhorou significativamente quando mudamos para AVL, principalmente no algoritmo de validação de vendas (apesar de agora ainda demorar alguns segundos).
- Em todas as queries, o tempo de execução é praticamente instantâneo com a excepção da query 11. Nesta, temos de aceder ao módulo da facturação e percorrer recursivamente esta estrutura pois queremos aceder a todos os produtos. Para cada um optamos por inserir numa lista ligada contendo apenas o código de produto e a quantidade de vezes que foi comprado. Ordenando depois esta lista com o algoritmo Merge Sort de acordo com a quantidade de vezes que o produto foi comprado, conseguimos facilmente ter acesso aos produtos que foram mais vendidos. Fazemos isto para cada filial, e, depois, para sabermos o número total de clientes é aceder de novo à faturação com os N produtos mais vendidos e ver para cada um qual o tamanho da sua lista ligada.
- Inicialmente tínhamos feito esta query ordenando apenas os primeiros N elementos da lista ligada com o algoritmo Selection Sort só que chegamos à conclusão que, a partir de um determinado N, o tempo de execução deste torna-se superior ao de se ordenar a lista por completo.