Universidade do Minho
Departamento de Informática

Laboratórios de Informática III

Mestrado Integrado em Engenharia Informática
2015/2016



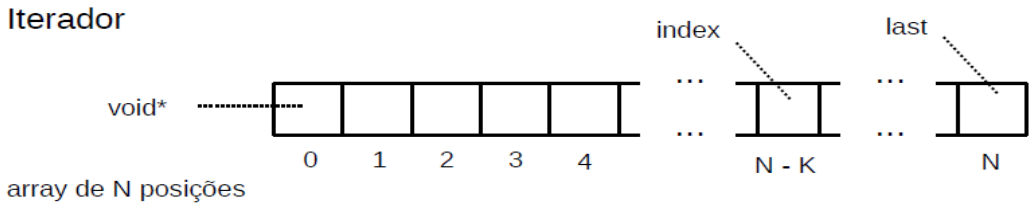
Grupo 11

Carlos Pereira - 61887
Diogo Silva - 76407
Rafael Braga - 61799

API

- **Módulo Iterator**

O módulo *iterator* contém um iterador implementado com um *array* dinâmico. Fornece suporte para a navegação de elementos e para a impressão destes. A estrutura deste é a seguinte:



Typedef no ficheiro iterator.h:

typedef struct iterator Iterator;*

Estrutura de dados principal deste módulo:

```
struct iterator {
    void **vec;                 /* Array de elementos do iterador. */
    unsigned int index;         /* Posição atual do iterador. */
    unsigned int capacity;      /* Capacidade do vetor. */
    unsigned int last;          /* Última posição do vetor. */
    unsigned int value_size;    /* Tamanho de um elemento do vetor. */

    void (*print)(const void*); /* Função para realizar a impressão de um
                                 * elemento. */
    void (*delete)(void*);      /* Função para apagar um elemento do iterador.
                                 * Esta função só é necessária para elementos
                                 * mais complexos. */
};

Iterator createIterator(const unsigned int value_size, void (*print)(const void*));
```

Aloca espaço para um iterador. O encapsulamento é garantido pela criação de uma nova estrutura dentro desta função, devolvendo-a no final.

void destroyIterator(Iterator it);

Liberta a memória ocupada pelo iterador. O utilizador apenas consegue fazê-lo através desta função do módulo, logo o encapsulamento é garantido.

unsigned int getSizeIterator(Iterator it);

Devolve o tamanho do iterador. Como se cria uma variável que vai guardar o valor da variável correspondente ao tamanho, dentro desta função, o encapsulamento é garantido.

unsigned int getIndexIterator(Iterator it);

Devolve o índice correspondente à última consulta no iterador. Como se cria uma variável que vai guardar o valor da variável correspondente ao índice, dentro desta função, o encapsulamento é garantido.

void nextIterator(Iterator it);

Avança uma posição no iterador. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

void prevIterator(Iterator it);

Recua uma posição no iterador. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

void goToIterator(Iterator it, const unsigned int index);

Navega para uma dada posição do iterador. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

*void setDeleteIterator(Iterator it, void (*delete)(void *value));*

Altera a função para libertar a memória de um elemento do iterador. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

*Iterator addIterator(Iterator it, const void *value);*

Adiciona um novo elemento ao iterador caso este exista.

Iterator addCollectionIterator(Iterator dest, Iterator source);

Adiciona um iterador a outro iterador. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

bool hasNextIterator(Iterator it);

Verifica se o iterador tem uma posição seguinte válida. Como se cria uma variável dentro da função, que depois será devolvida, o encapsulamento mantém-se.

bool hasPrevIterator(Iterator it);

Verifica se a posição anterior do iterador é válida. Como se cria uma variável dentro da função, que depois será devolvida, o encapsulamento mantém-se.

*void foldIterator(const Iterator it, void *acc, void (*f)(const void *value, void *acc));*

Cria uma qualquer estrutura de dados caso o iterador exista. O utilizador só precisa de fornecer o iterador, o acumulador e a função a aplicar. Esta função irá encarregar-se do resto, logo o encapsulamento mantém-se.

void printCurrentIterator(Iterator it);

Realiza a impressão da posição atual do iterador.
O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

void printAllIterator(Iterator it);

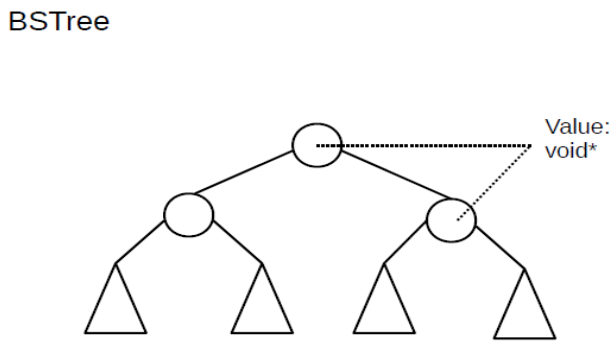
Realiza a impressão de todos os elementos do iterador. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

void printAtIterator(Iterator it, unsigned int index);

Realiza a impressão de uma certa posição do iterador caso esta seja válida. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

- **Módulo BSTree**

O módulo *BSTree* contém uma árvore binária de procura do tipo *AVL*. A sua estrutura é a seguinte:



Typedef no ficheiro bstree.h:

typedef struct bstree* BSTree;

Typedef no ficheiro bstree.c:

```
typedef struct node {
    void *value;        /* Valor a adicionar à arvore. */
    unsigned char height;    /* Altura da árvore.
                              * NOTA: A altura de uma árvore com cerca de 10G de
                              * elementos é 44, pelo que valores entre 0 e 255
                              * chegam para armazenar a altura de uma árvore com
```

```
                                * biliões de elementos. Ao armazenar este valor num
                                * inteiro estaríamos a gastar 4 vezes mais memória
                                * para este campo, o que é desnecessário. */
        struct node *left;      /* Sub-árvore à esquerda. */
        struct node *right;   /* Sub-árvore à direita. */
} Node;
```

Estrutura principal do módulo BSTree:

```
struct bstree {
        Node *root;                     /* Raiz da árvore. */
        unsigned int value_size;            /* Tamanho de cada elemento a
                                * adicionar na árvore. */
        unsigned int num_nodes;             /* Número de nós da árvore. */
        int (*cmp)(const void*, const void*); /* Função para efetuar a comparação
                                                * entre dois elementos. */
        void (*print)(const void*);         /* Função para realizar a impressão de
                                                * um elemento no ecrã. */
        void (*delete)(void*);      /* Função para apagar um elemento de
                                * um nó. Por defeito esta função será
                                * a função defaultDestroy que
                                * apaga todo o tipo de dados básicos.
                                * Para apagar tipos mais complexos,
                                * como por exemplo um array de
                                * qualquer tipo de dados, deve-se
                                * fornecer uma função o apagar. */
};

BSTree createBSTree(const unsigned int value_size,
                        int (*cmp)(const void *a,const void *b),
                        void (*print)(const void *value))
```

Função que inicializa o tipo concreto de dados. O encapsulamento é garantido pela criação de uma nova estrutura, devolvendo-a no fim.

```
void destroyBSTree(BSTree bst);
```

Liberta a memória ocupada pelos dados. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

```
unsigned int getHeightBSTree(const BSTree bst);
```

Devolve a altura da árvore caso esta exista. O encapsulamento é garantido porque se cria uma variável que vai guardar este valor, sendo devolvida no final.

```
unsigned int getSizeBSTree(const BSTree bst);
```

Devolve o tamanho da árvore. O encapsulamento é garantido porque se cria uma variável que vai guardar este valor, sendo devolvida no final.

*void setCmpBSTree(BSTree bst, int (*cmp)(const void *a, const void *b));*

Altera a função para comparar dois elementos da árvore. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

*void setPrintBSTree(BSTree bst, void (*print)(const void *value));*

Altera a função para realizar a impressão de um elemento. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

*void setDeleteBSTree(BSTree bst, void (*delete)(void *value));*

Altera a função para apagar um elemento de um nó. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

*BSTree addBSTree(BSTree bst, const void *value);*

Adiciona um elemento à árvore caso esta exista. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

*BSTree addWithCondBSTree(BSTree bst, const void *new_value,*
*void (*f)(void *acc, const void *new_value));*

Adiciona um elemento à árvore caso esta exista. É aplicada uma função para tratar casos de elementos repetidos. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

*bool containsBSTree(const BSTree bst, const void *value);*

Procura por um elemento na árvore caso esta exista. O encapsulamento é garantido porque se cria uma variável que vai guardar este valor, sendo devolvida no final.

*bool anyValueMatch(const BSTree bst, const void *pred,*
*bool (*f)(const void *value, const void *pred));*

Verifica se algum elemento da árvore satisfaz *f*. O encapsulamento é garantido porque se cria uma variável que vai guardar este valor, sendo devolvida no final.

*void delBSTree(BSTree bst, const void * value);*

Remove um elemento à árvore caso esta exista. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

*void mapBSTree(const BSTree bst, void (*f)(void * value));*

Aplica uma função a cada elemento da árvore caso esta exista. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

*void foldBSTree(const BSTree bst, void *acc, void (*f)(const void *value, void *acc));*

Constrói uma qualquer estrutura de dados a partir de todos os elementos da árvore binária. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

*void foldWithBSTree(const BSTree bst, void *acc, const void *data, const void *pred,*
*void (*f) (const void *data, const void *value, const void *pred, void *acc));*

Constrói uma qualquer estrutura de dados a partir de todos os elementos da árvore binária. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

Iterator toIteratorInOrderBSTree(const BSTree bst);

Devolve um iterador sobre a árvore binária de procura. O encapsulamento é garantido porque se cria um iterador que vai guardar estes valores, sendo devolvido no final.

Iterator toIteratorPostOrderBSTree(const BSTree bst);

Devolve um iterador sobre a árvore binária de procura. O encapsulamento é garantido porque se cria um iterador que vai guardar estes valores, sendo devolvido no final.

*Iterator filterBSTree(const BSTree bst, bool (*f)(const void *value));*

Devolve um iterador com todos os elementos da árvore que satisfaçam a função *f* recebida. O encapsulamento é garantido porque se cria um iterador que vai guardar estes valores, sendo devolvido no final.

*Iterator filterWithPredBSTree(const BSTree bst, const void *pred,*
*bool (*f)(const void *value, const void *pred));*

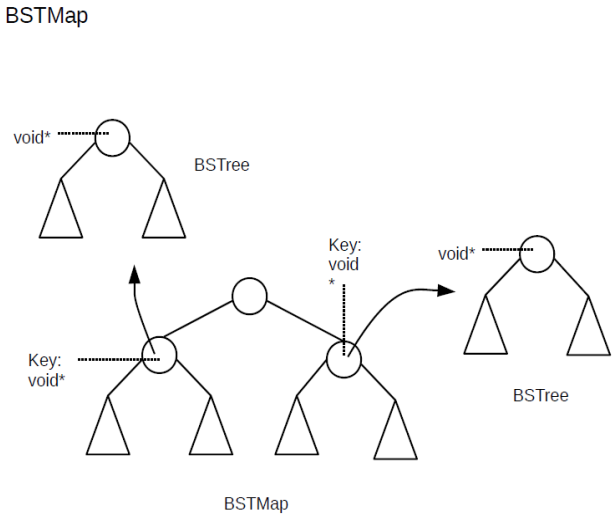
Devolve um iterador com todos os elementos da árvore que satisfaçam a função *f* recebida. O encapsulamento é garantido porque se cria um iterador que vai guardar estes valores, sendo devolvido no final.

*void filterToIteratorBSTree(const BSTree bst, Iterator it, bool (*f)(const void *value));*

Adiciona a um iterador já existente todos os elementos da árvore que satisfaçam a função *f* recebida. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

- **Módulo BSTMap**

A estrutura principal do módulo BSTMap é uma AVL em que os nós contêm uma chave e um apontador para uma outra AVL. A sua estrutura é a seguinte:



Typedef no ficheiro bstmap.h:

typedef struct bstMap BSTMap; /* Apontador para uma BSTMap. */*

Typedef no ficheiro bstmap.c:

```
typedef struct node {
        BSTree bst;          /* Apontador para uma árvore binária de procura do
                             * tipo AVL. */
        void *key;           /* Chave da BSTMap. */
        unsigned char height; /* Altura da BSTMap. */
        struct node *left;    /* Apontador para a sub-árvore à esquerda. */
        struct node *right;   /* Apontador para a Sub-árvore à direita. */
} Node;
```

Estrutura principal deste módulo:

```
struct bstMap {
        Node *root;                     /* Apontador para a raiz da
                                         * bstMap. */
        unsigned int num_nodes;              /* Número de nós. */
        unsigned int key_size;              /* Tamanho da chave. */
        int (*cmpKey)(const void*, const void*); /* Função para comparar duas chaves
                                                  * da BSTMap. */

        void (*printKey)(const void*);          /* Função para fazer a impressão de
                                * uma chave da BSTMap */
        void (*deleteKey)(void*);               /* Função para apagar um elemento de
                                * um nó. Por defeito esta função
                                * será defaultDestroyKey que
                                * apaga qualquer tipo de dados
                                * básicos. */
```

```
/* Os próximos elementos serão usados na createBSTree */
        unsigned int value_size;                /* Tamanho (em bytes) de um
                                                 * elemento a guardar na árvore */
        int (*cmpValue)(const void*, const void*); /* Compara dois valores de um
                                                    * BSTree */
        void (*printValue)(const void*);   /* Efetua a impressão de um valor
                                            * de uma BST. */
        void (*deleteValue)(void*);     /* Faz free de um ValueBST */
};
BSTMap createBSTMap(const unsigned int key_size, int (*cmpKey)(const void *a,
                                const void *b), void (*printKey)(const void *key),
                                unsigned int value_size,
                                int (*cmpValue)(const void *a, const void *b),
                                void (*printValue)(const void *value));
```

Função que inicializa o tipo concreto de dados. O encapsulamento é garantido pela criação de uma nova estrutura, devolvendo-a no fim.

void destroyBSTMap(BSTMap bstm);

Função que liberta toda a memória da *BSTMap*. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

*void setCmpBSTMap(*BSTMap *bstm, int (*cmp)(const void *a, const void *b));*

Altera a função para comparar duas chaves da *BSTMap*. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

*void setPrintBSTMap(BSTMap bstm, void (*print)(const void *key));*

Altera a função para realizar a impressão de uma chave da *BSTMap*. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

*void setDeleteBSTMap(BSTMap bstm, void (*delete)(void *key));*

Altera a função para apagar uma chave da *BSTMap*. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

*void setCmpValueBSTMap(BSTMap bstm, int (*cmp)(const void *a, const void *b));*

Altera a função para comparar dois valores da BSTree. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

*void setPrintValueBSTMap(BSTMap bstm, void (*print)(const void *value));*

Altera a função para realizar a impressão de um valor da BST. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

*void setDeleteValueBSTMap(BSTMap bstm, void (*delete)(void*));*

Altera a função de realocação da memória de um valor da *BST*. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

*BSTMap addBSTMap(BSTMap bstm, const void *key, const void *value);*

Adiciona uma chave e um elemento à árvore caso esta exista. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

*bool containsKeyBSTMap(const BSTMap bstm, const void *key);*

Procura por uma chave na árvore. O encapsulamento é garantido porque se cria uma variável que vai guardar este valore, sendo devolvida no final.

*bool containsValueBSTMap(const BSTMap bstm, const void *key, const void *value);*

Procura por um valor nas BST. O encapsulamento é garantido porque se cria uma variável que vai guardar este valore, sendo devolvida no final.

*void mapKeyBSTMap(const BSTMap bstm, void (*f)(void *key));*

Faz o map de uma função a todas as chaves da *BSTMap*. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

*void mapValueBSTMap(const BSTMap bstm, const void *key, void (*f)(void *value));*

Faz o *map* de uma função a todos os valores das *BST* de uma chave contida na *BSTMap*. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

*void foldBSTMap(const BSTMap bstm, void *acc, void (*f)(const void *value, void *acc));*

Faz o *fold* de uma função com acumulador a todos os valores das *BST*. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

*void foldKeyBSTMap(const BSTMap bstm, void *acc, void (*f)(const void *key, void *acc));*

Faz o *fold* de uma função com acumulador a todas as chaves da *BSTMap*. O utilizador apenas consegue fazer operações deste género através desta função do

módulo, logo o encapsulamento é garantido.

*void foldValueBSTMap(const BSTMap bstm, const void *key, void *acc, void (*f)(const void *value, void *acc));*

Faz o *fold* de uma função com acumulador a todos os valores de uma *BST* de uma chave contida na *BSTMap*. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

*void foldKeyValueWithBSTMap(const BSTMap bstm, void *acc, const void *pred, void (*f)(const void *key, const void *value, const void *pred, void *acc));*

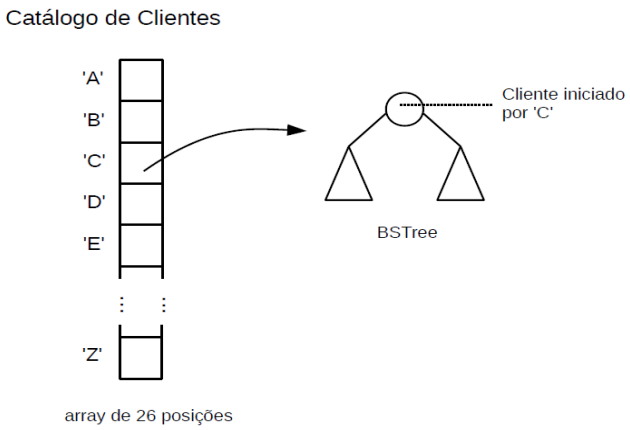
Faz o *fold* de uma função com acumulador a todos os valores da *BSTMap*. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

*Iterator toIteratorValuePostOrderBSTMap(const BSTMap bstm, const void *key);*

Procura um nó da *BSTMap* com chave *key*. O encapsulamento é garantido porque se cria um iterador que vai guardar estes valores, sendo devolvido no final.

- **Módulo Catálogo de Clientes**

A estrutura deste módulo é a seguinte:



Typedef no ficheiro catalogoClientes.h:

```
typedef char* ClientCode;              /* Tipo de dados armazenados -
                                        * Código de um cliente. */
typedef struct clientCtlg* ClientCtlg;  /* Array de árvores binárias -
                                        * Catálogo de clientes.*/
typedef void (*printClient)(const void*); /* Função para realizar a impressão
                                           * de um código de cliente. */
```

Estrutura principal deste módulo:

```
struct clientCtlg {
    BSTree *vec;          /* Array de BSTree's que irá conter os códigos
```

```
                    * de clientes.*/

    Iterator it;          /* Iterador sobre o catálogo de clientes. */

    unsigned int page_size; /* Tamanho de uma página de códigos de clientes. */
    unsigned int page;      /* Página atual. */
    printClient print;      /* Função de impressão de um cliente. */
};
ClientCtlg createClientCtlg(printClient print);
```

Cria e inicializa o catálogo de clientes. O encapsulamento é garantido pela criação de uma nova estrutura, devolvendo-a no fim.

void destroyClientCtlg(ClientCtlg catalog);

Liberta a memória ocupada pelo catálogo. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

unsigned int getNumClientPages(const ClientCtlg catalog);

Devolve o número total de páginas de códigos de cliente. O encapsulamento é garantido porque se cria uma variável que vai guardar este valor, sendo devolvida no final.

unsigned int getClientPageSize(const ClientCtlg catalog);

Devolve o tamanho de página de códigos de cliente. O encapsulamento é garantido porque se cria uma variável que vai guardar este valor, sendo devolvida no final.

void setClientPageSize(const ClientCtlg catalog, const unsigned int page_size);

Modifica o tamanho de uma página de códigos de cliente caso o catálogo tenha sido previamente inicializado. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

ClientCtlg addClient(ClientCtlg catalog, const ClientCode code);

Adiciona um cliente ao catálogo de clientes. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

*bool containsClient(const ClientCtlg catalog, const char *code);*

Verifica se existe um dado cliente no catálogo. O encapsulamento é garantido porque se cria uma variável que vai guardar este valor, sendo devolvida no final.

unsigned int clientsBeginningWithX(const ClientCtlg catalog, const ClientCode x);

Inicia o iterador com uma nova pesquisa no catálogo de clientes. Devolve o número de elementos guardados no *iterator.* O encapsulamento é garantido porque se cria uma

variável que vai guardar este valor, sendo devolvida no final.

*unsigned int filterClients(const ClientCtlg catalog, bool (*f)(const void*));*

Filtra para o iterador do catálogo todos os clientes que satisfaçam *f*. O encapsulamento é garantido porque se cria uma variável que vai guardar este valor, sendo devolvida no final.

unsigned int goToClientPage(const ClientCtlg catalog, const unsigned int num_page);

Vai para a página *num_page* de uma pesquisa realizada no catálogo de clientes. O encapsulamento é garantido porque se cria uma variável que vai guardar este valor, sendo devolvida no final.

unsigned int nextClientPage(const ClientCtlg catalog);

Mostra a próxima página de uma pesquisa no catálogo de clientes. O encapsulamento é garantido porque se cria uma variável que vai guardar este valor, sendo devolvida no final.

unsigned int previousClientPage(const ClientCtlg catalog);

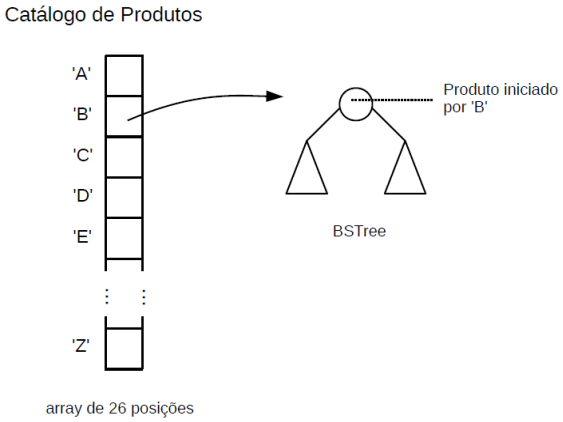
Mostra a página anterior da última pesquisa no catálogo de clientes. O encapsulamento é garantido porque se cria uma variável que vai guardar este valor, sendo devolvida no final.

*void foldClients(const ClientCtlg catalog, void *acc, void (*f)(const void *value, void *acc));*

Cria uma qualquer estrutura de dados caso o catálogo exista. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

- **Módulo Catálogo de Produtos**

A estrutura deste módulo é a seguinte:



Typedef no ficheiro catalogoProdutos.h:

```
typedef char* ProductCode;              /* Tipo de dados armazenados -
                                         * Código de um produto. */
typedef struct productCtlg* ProductCtlg; /* Array de árvores binárias -
                                          * Catálogo de produtos.*/
typedef void (*printProd)(const void*);   /*Função para realizar a impressão
                                           *de um código de produto. */
```

Estrutura principal deste módulo:

```
struct productCtlg {
    BSTree *vec;            /* Array de BSTree's que irá conter os códigos
                             * de produtos.*/
    Iterator it;          /* Iterador sobre o catálogo de produtos. */
    unsigned int page_size;  /* Tamanho de uma página de códigos de produtos. */
    unsigned int page;        /* Página atual. */
    printProd printP;       /* Função de impressão de um código de produto. */
};
```

ProductCtlg createProductCtlg(printProd printP);

Cria e inicializa o catálogo de produtos. O encapsulamento é garantido pela criação de uma nova estrutura, devolvendo-a no fim.

void destroyProductCtlg(ProductCtlg catalog);

Liberta a memória ocupada pelo catálogo. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

unsigned int getNumProductPages(const ProductCtlg catalog);

Devolve o número total de páginas de códigos de produto. O encapsulamento é garantido porque se cria uma variável que vai guardar este valor, sendo devolvida no final.

unsigned int getProductPageSize(const ProductCtlg catalog);

Devolve o tamanho de página de códigos de produto. O encapsulamento é garantido porque se cria uma variável que vai guardar este valor, sendo devolvida no final.

void setProductPageSize(const ProductCtlg catalog, const unsigned int page_size);

Modifica o tamanho de uma página de códigos de produto. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

ProductCtlg addProduct(ProductCtlg catalog, const ProductCode code);

Adiciona um produto ao catálogo de produtos. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é

garantido.

*bool containsProduct(const ProductCtlg catalog, const char *code);*

Verifica se existe um dado produto no catálogo. O encapsulamento é garantido porque se cria uma variável que vai guardar este valor, sendo devolvida no final.

unsigned int productsBeginningWithX(const ProductCtlg catalog, const ProductCode x);

Inicia o iterador com uma nova pesquisa no catálogo de produtos e devolve o número de elementos. O encapsulamento é garantido porque se cria uma variável que vai guardar este valor, sendo devolvida no final.

unsigned int goToProductPage(const ProductCtlg catalog, const unsigned int num_page);

Vai para a página *num_page* de uma pesquisa realizada no catálogo de produtos. O encapsulamento é garantido porque se cria uma variável que vai guardar este valor.

unsigned int nextProductPage(const ProductCtlg catalog);

Mostra a próxima página de uma pesquisa no catálogo de produtos. O encapsulamento é garantido porque se cria uma variável que vai guardar este valor, sendo devolvida no final.

unsigned int previousProductPage(const ProductCtlg catalog);

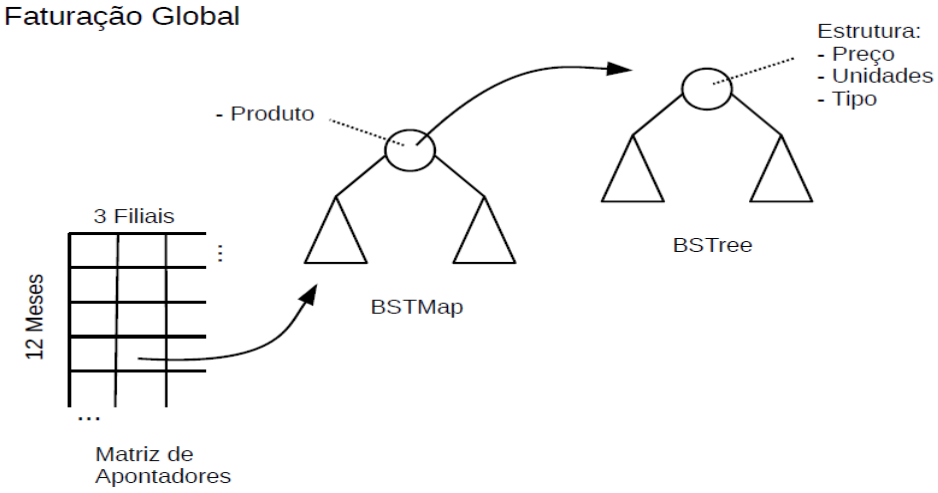
Mostra a página anterior de uma pesquisa no catálogo de produtos. O encapsulamento é garantido porque se cria uma variável que vai guardar este valor, sendo devolvida no final.

*void mapProductCtlg(const ProductCtlg catalog, void(*f)(void *product));*

Efetua uma função a cada código de produto presente no catálogo caso o catálogo exista. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

- **Módulo Faturação Global**

A estrutura deste módulo é a seguinte:



Typedef no ficheiro faturacaoGlobal.h:

```
typedef char ProductBill; /*Código do produto de uma venda*/
typedef struct bill *Bill; /*Estrutura de uma venda*/
typedef struct productUnitsClients ProductUnitsClients; /* Estrutura com o código
                                                          *de produto, o número de unidades
                                                          *vendidas e o número de clientes
                                                          *que compraram esse produto.*/
typedef struct globalBilled *GlobalBilled; /*Estrutura principal do módulo*/
typedef void (*printPBill)(const void*); /*Função impressão do código de produto*/
typedef void (*printPrdctUntClnt)(const void*); /*Função de impressão do código de
                                                  *produto, as unidades vendidas e o
                                                  *número de clientes que o
                                                  *compraram.*/
typedef void (*printGBill)(const void*); /*Função que imprime os campos de uma
                                           *venda.*/
```

Estrutura principal deste módulo:

```
struct globalBilled {
    BSTMap **b_matrix;      /* Uma matriz de BSTMap composta por 12 linhas e 3
                             *colunas, correspondendo, respetivamente,
                             * a meses e filiais.*/
    Iterator not_sold;      /* Todos os produtos que não foram vendidos
                             * encontram-se neste iterador. */
    Iterator *not_sold_in_branch; /* Um array de iteradores, onde cada posição
                                   * corresponde a uma filial */
    /* Estruturas e variáveis auxiliares. */
    unsigned int page;              /* Página atual do iterador.*/
    unsigned int page_size;         /* Número de elementos por página a ser
                                     *apresentada.*/
    unsigned int branch_page_size;  /* Número de elementos por página a ser
                                     *apresentada, quando se faz distinções de
                                     *resultados por filial.*/
    unsigned int *not_sold_in_branch_pages; /*Número máximo de páginas, por filial,
                                             *de cada iterador dos produtos que não
                                             *foram vendidos.*/
    unsigned int months;  /* Pode eventualmente ser alterado o valor por defeito
                           * de meses. Esta variável corresponde ao número de
                           * meses. */
    unsigned int branches; /* Número de filiais utilizadas no número de colunas
                            * de b_matrix */
    unsigned int product_size; /* Tamanho de um código de produto. */
    unsigned int *total_units; /* Total de unidades por mês. Ao ser inserida nova
                                * informação à estrutura, estes valores serão
                                * incrementados */
    double *total_billed;     /* Total faturado por mês. Ao ser inserida nova
                               *informação à estrutura, é calculado o novo
                               *valor faturado em cada mês. */
    printPBill printP; /* Função de impressão de um produto. */
    printPBill printB; /* Função de impressão de uma estrutura Bill. */
```

```
        BSTree *aux_most_sold; /* BSTree auxiliar de produtos mais vendidos, as
                                * estruturas em cada nó estão organizadas por ordem
                                * alfabética do produto. A cada nova inserção de dados
                                * é recalculado o número de unidades. */
        BSTree *most_sold;     /* BSTree que armazena os produtos mais comprados por
                                * ordem descendente do mais comprado. */
        Iterator *it_MS;       /* Iterador com os produtos mais vendidos. */
        unsigned int n;        /* Número de produtos que o cliente pretende
                                * visualizar no iterador it_MS. */
        printPrdctUntClnt printPUC; /* Função de impressão de uma estrutura
                                     * productUnitsClients. */
        unsigned int *it_MS_pages;  /* Número de páginas do iterador it_MS. */
};
```

Estruturas auxiliares:

```
/*Estrutura principal que armazena uma fatura de compras. */

struct bill {
        unsigned int units; /* Número de unidades de um produto numa venda. */
        double price;       /* Preço unitário de um produto numa venda. */
        char type;          /* Tipo de venda. */
};
/* Estrutura com o código de produto, o número de unidades vendidas e o número
 * de clientes que compraram esse produto.*/
struct productUnitsClients {
        char *product;        /* Código de um produto. */
        unsigned int units;   /* Número de unidades vendidas de um produto. */
        unsigned int clients; /* Número total de clientes que compraram um determinado
                               * produto. */
};
/* Estrutura auxiliar para obter facilmente e apenas numa travessia o número
 * de unidades em modo N e o preço em modo P. */
typedef struct auxBill {
        unsigned int units_N; /* Unidades de uma compra Normal. */
        unsigned int units_P; /* Unidades de uma compra Promocional. */
        double price_N;       /* Preço de uma compra Normal. */
        double price_P;       /* Preço de uma compra Promocional. */
} *AuxBill;
GlobalBilled createGlobalBilled(printPBill printP, printGBill printB, printPrdctUntClnt
                                printPUC, const unsigned int months,
                                const unsigned int branches,
                                const unsigned int product_size);
```

Cria a estrutura principal da faturação. O encapsulamento é garantido pela criação de uma nova estrutura, devolvendo-a no fim.

```
GlobalBilled defaultCreateGlobalBilled(printPBill printP, printGBill printB,
                                       printPrdctUntClnt printPUC);
```

Cria a estrutura principal da faturação de um modo pré-definido.O encapsulamento é garantido pela criação de uma nova estrutura, devolvendo-a no fim.

void destroyGlobalBilled(GlobalBilled g);

Destrói a estrutura principal. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

unsigned int getUnitsBill(Bill bill);

Devolve o campo de unidades de uma Bill. O encapsulamento é garantido porque se cria uma variável que vai guardar este valor, sendo devolvida no final.

double getPriceBill(Bill bill);

Devolve o campo de preço de um Bill. O encapsulamento é garantido porque se cria uma variável que vai guardar este valor, sendo devolvida no final.

char getTypeBill(Bill bill);

Devolve o campo do tipo de um Bill. O encapsulamento é garantido porque se cria uma variável que vai guardar este valor, sendo devolvida no final.

*char *getProductPUC(const ProductUnitsClients p);*

Devolve o campo produto de um ProductUnitsClients. O encapsulamento é garantido porque se cria uma variável que vai guardar este valor, sendo devolvida no final.

unsigned int getUnitsPUC(const ProductUnitsClients p);

Devolve o campo unidades de um ProductUnitsClients. O encapsulamento é garantido porque se cria uma variável que vai guardar este valor, sendo devolvida no final.

unsigned int getClientsPUC(const ProductUnitsClients p);

Devolve o campo produto de número de clientes de um *ProductUnitsClients*. O encapsulamento é garantido porque se cria uma variável que vai guardar este valor, sendo devolvida no final.

unsigned int getNotSoldInBranchPages(const GlobalBilled g, unsigned int branch);

Devolve a quantidade de páginas para apresentar os produtos não vendidos numa determinada filial. O encapsulamento é garantido porque se cria uma variável que vai guardar este valor, sendo devolvida no final.

unsigned int getMostBoughtPages(const GlobalBilled g, unsigned int branch);

Devolve a quantidade de páginas para apresentar os produtos mais vendidos no iterador *it_MS*. O encapsulamento é garantido porque se cria uma variável que vai guardar este valor, sendo devolvida no final.

unsigned int getNotSoldPageSize(const GlobalBilled g);

Devolve o número de elementos a apresentar por página do iterador *not_sold*. O encapsulamento é garantido porque se cria uma variável que vai guardar este valor, sendo devolvida no final.

unsigned int getNotSoldPages(const GlobalBilled g);

Devolve o número de páginas do iterador *not_sold*. O encapsulamento é garantido porque se cria uma variável que vai guardar este valor, sendo devolvida no final.

unsigned int getNotSold(GlobalBilled g);

Apresenta o número de produtos não vendidos em nenhuma das três filiais. O encapsulamento é garantido porque se cria uma variável que vai guardar este valor, sendo devolvida no final.

unsigned int getNotSoldInBranch(GlobalBilled g, const unsigned int branch);

Apresenta o número de produtos não vendidos numa nada filial. O encapsulamento é garantido porque se cria uma variável que vai guardar este valor, sendo devolvida no final.

unsigned int getTotalUnits(const GlobalBilled g, const unsigned int month);

Devolve as unidades vendidas num determinado mês. O encapsulamento é garantido porque se cria uma variável que vai guardar este valor, sendo devolvida no final.

double getTotalBilled(const GlobalBilled g, const unsigned int month);

Devolve total facturado num determinado mês de uma estrutura GlobalBilled. O encapsulamento é garantido porque se cria uma variável que vai guardar este valor, sendo devolvida no final.

void getNMostBought(const GlobalBilled g, const unsigned int n);

Calcula os produtos mais comprados por ordem descendente. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

GlobalBilled addBill(GlobalBilled g, const ProductBill product, const double price,
const unsigned int units, const char type, const unsigned int month,
const unsigned int branch);

Adiciona uma nova factura a GlobalBilled. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

GlobalBilled addNotSold(GlobalBilled g, const ProductBill product);

Adiciona um produto não comprado em nenhuma filial ao iterador *not_sold*. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

GlobalBilled addNotSoldInBranch(GlobalBilled g, const ProductBill product,
const unsigned int branch);

Adiciona um produto não comprado numa dada filial ao iterador *not_sold_in_branch*. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

bool productSoldInBranch(const GlobalBilled g, ProductBill product,
const unsigned int branch);

Verifica se um dado produto foi comprado numa determinada filial. O encapsulamento é garantido porque se cria uma variável que vai guardar este valor, sendo devolvida no final.

void billOfProductInMonth(const GlobalBilled g, const ProductBill product,
const unsigned int month, const unsigned int branch,
*double *price_N, double *price_P,*
*unsigned int *units_N, unsigned int *units_P);*

Determina o número de vendas e o preço de um mês numa dada filial. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

unsigned int goToNotSoldPage(const GlobalBilled g, const unsigned int num_page);

Altera a posição de pesquisa do iterador para a página *num_page.* O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

void goToNotSoldInBranchPage(const GlobalBilled g, const unsigned int num_page,
const unsigned int branch);

Altera a posição de pesquisa do iterador para a página *num_page.* O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

unsigned int nextNotSoldPage(const GlobalBilled g);

Mostra a próxima página da pesquisa no iterador de produtos não vendidos. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

void nextNotSoldInBranchPage(const GlobalBilled g, const unsigned int branch);

Mostra a próxima página da pesquisa no iterador de produtos não vendidos. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

unsigned int previousNotSoldPage(const GlobalBilled g);

Mostra a página anterior da pesquisa no iterador *not_sold*. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

void resetAllSearches(const GlobalBilled g);

Re-inicializa as pesquisas de todos os iteradores. O utilizador só consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

void goToMostBoughtPage(const GlobalBilled g, const unsigned int num_page,
const unsigned int branch);

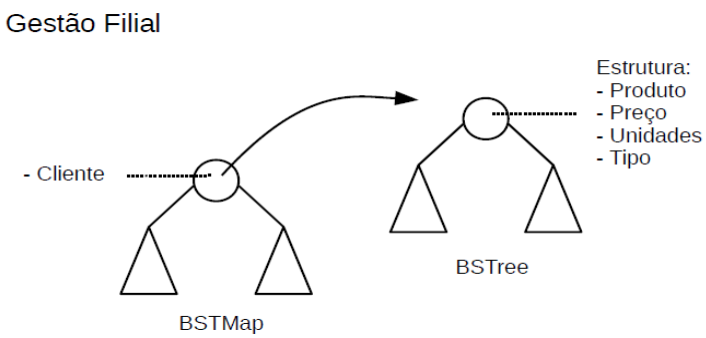
Altera a posição de pesquisa do iterador para a página num_page fornecida. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

void nextMostBoughtPage(const GlobalBilled g, const unsigned int branch);

Mostra a próxima página da pesquisa no iterador. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

- **Módulo Gestão de Filial**

  A estrutura deste módulo é a seguinte:



Typedef no ficheiro gestaoFilial.h:

typedef struct sale Sale;* ** Campos de uma venda:*
** - Codigo de produto.*
** - Unidades.*

```
                                                *       - Preço.
                                                *       - Tipo de venda. */
typedef struct branchCtlg* BranchCtlg;          /* Estrutura de um catálogo
                                                 * de vendas de uma filial. */
typedef void (*printSaleCode)(const void*);     /* Função para a impressão de
                                                 * um código de cliente. */
typedef void (*printSaleBranch)(const void*); /* Função para a impressão de
                                               * todos os campos de uma
                                               * venda. */
```

Estrutura principal do módulo:

```
struct branchCtlg {
        BSTMap *sales;                          /* Apontador para uma BSTMap em que a
                                                 * chave é um código de cliente. */
        Iterator mode_it[NUM_MODES];            /* Apontador para iterador onde a
                                                 * impressão tem de distinguir a venda
                                                 * por tipo (N ou P).*/
        unsigned int mode_pages[NUM_MODES]; /* Número de elementos por página do
                                             * de cada iterador modo N ou P. */
        unsigned int page_size;         /* Número de elementos numa página do
                                         * iterador de modos. */
        unsigned int page;              /* Página atual no iterador. */
        unsigned int size;              /* Número de meses da gestão de
                                         * filial .*/
        printSaleCode printSC;          /*Função que faz a impressão de um
                                         * código de produto. */
        printSaleBranch printS;         /* Função que faz a impressão de uma
                                         * venda. */
        unsigned int client_size;       /* Tamanho de um código de cliente. */
};
```

Estrutura auxilar do módulo:

```
struct sale {
        char *product;      /* Código de produto. */
        double price;       /* Preço de uma unidade de um produto. */
        unsigned int units; /* Número de unidades compradas de um produto. */
        char type;          /* Tipo de compra de um produto. */
};
BranchCtlg createBranchCtlg(printSaleCode printSC, printSaleBranch printS,
                            const unsigned int size);
```

Criação do catálogo de vendas. O encapsulamento é garantido pela criação de uma nova estrutura, devolvendo-a no fim.

```
BranchCtlg defaultCreateBranchCtlg(printSaleCode printSC, printSaleBranch printS);
```

Construtor por defeito da gestão de uma filial. O encapsulamento é garantido pela criação de uma nova estrutura, devolvendo-a no fim.

```
void destroyBranchCtlg(BranchCtlg catalog);
```

Destruição do catálogo de vendas. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

char getProduct(const Sale sale);*

Devolve o código produto de uma venda. O encapsulamento é garantido porque se cria uma variável que vai guardar este valor, sendo devolvida no final.

double getPrice(const Sale sale);

Devolve o preço de uma venda. O encapsulamento é garantido porque se cria uma variável que vai guardar este valor, sendo devolvida no final.

unsigned int getUnits(const Sale sale);

Devolve as unidades de uma venda. O encapsulamento é garantido porque se cria uma variável que vai guardar este valor, sendo devolvida no final.

char getType(const Sale sale);

Devolve o tipo de uma venda. O encapsulamento é garantido porque se cria uma variável que vai guardar este valor, sendo devolvida no final.

*unsigned int getTotalProducts(const BranchCtlg catalog, const char *client,*
unsigned int month);

Calcula o número total de produtos comprados por um cliente num dado mês. O encapsulamento é garantido porque se cria uma variável que vai guardar este valor, sendo devolvida no final.

unsigned int getBranchPageSize(const BranchCtlg catalog);

Devolve o número de elementos na página de códigos de cliente da filial. O encapsulamento é garantido porque se cria uma variável que vai guardar este valor, sendo devolvida no final.

unsigned int getNumModePages(const BranchCtlg catalog, const char mode);

Devolve o número de páginas da pesquisa de cada tipo de venda. O encapsulamento é garantido porque se cria uma variável que vai guardar este valor, sendo devolvida no final.

void setSalePageSize(const BranchCtlg catalog, const unsigned int page_size);

Modifica o número de elementos numa página de códigos. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

*BranchCtlg addSale(BranchCtlg catalog, char *product, const double price,*
*const unsigned int units, const char type, char *client,*
unsigned int month);

Adiciona uma venda a cada *array* de árvores *BSTMap*. O encapsulamento é

garantido através da cópia de todos os valores recebidos

*bool existClientInBranch(const BranchCtlg catalog, const char *client);*

Procura por um código de cliente na filial. O encapsulamento é garantido porque se cria uma variável que vai guardar este valor, sendo devolvida no final.

*void foldClientProductsInMonth(const BranchCtlg catalog, const char *client,*
*unsigned int month, void *acc,*
*void (*f)(const void *value, void *acc));*

Aplica uma função com acumulador a todos os produtos comprados por um cliente num mês na filial. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

*void mostSoldProducts(const BranchCtlg catalog, void *acc,*
*void (*f)(const void *value, void *acc));*

Aplica uma função com acumulador a todos os nós das árvores *BST* deste módulo. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

*void mostSpentByClient(const BranchCtlg catalog, const char *client, void *acc,*
*void (*f)(const void *v, void *acc));*

Aplica uma função com acumulador a todos os nós das árvores *BST* deste módulo. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

*void clientsThatBought(const BranchCtlg catalog, const char *product,*
*unsigned int *total_N, unsigned int *total_P);*

Calcula o número clientes que compraram um dado produto. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

void nextModePage(const BranchCtlg catalog, const char mode);

Passa para a página seguinte da pesquisa. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

void goToModePage(const BranchCtlg catalog, const char mode,
const unsigned int page);

Vai diretamente para uma página da pesquisa. O utilizador apenas consegue fazer operações deste género através desta função do módulo, logo o encapsulamento é garantido.

Estruturação do programa e IU

O programa principal está dividido nas seguintes secções:

– Leitura de dados a partir do teclado. Nesta secção existe uma função que lê os dados introduzidos pelo utilizador. O resultado é devolvido num *buffer* global sem o caractere *new line*.
– Funções de impressão. Contém todas as funções necessárias para a impressão dos campos das estruturas navegáveis dos módulos.
– Funções de validação de dados. Funções responsáveis pela verificação de todos os campos de uma venda.
– Menus. Contém os diferentes tipos de menus. Nesta aplicação existem dois tipos de menus, o menu principal da aplicação e o menu de navegação.
– Construtores e destrutores. Funções responsáveis pela inicialização e destruição de todas as estruturas de dados necessárias para o funcionamento deste programa.
– Funções de *queries*. Cada *query* contém a sua própria secção e todas as funções auxiliares para esta, nomeadamente funções para a obtenção de opções do utilizador.
– *Main*. Função responsável pela inicialização e terminação do programa.

A *IU* do programa *gereVendas* contém os seguintes componentes:

– Menu de iniciação. Pede ao utilizador como tratar dos ficheiros de clientes, produtos e de vendas. O utilizador pode introduzir cada um destes manualmente ou os ficheiros podem ser carregados com os valores por defeito (*Clientes.txt*, *Produtos.txt* e *Vendas_1M.txt*).
– Menu principal. Mostra todas as opções do programa *gereVendas* ao utilizador e pede ao mesmo a introdução de uma opção válida. Cada opção deste menu corresponde a uma *query*. Naturalmente o menu principal também contém a opção para o utilizador sair do programa.
– Menu de navegação. Este menu é chamado caso o número de elementos impressos numa *query* seja superior ao tamanho máximo de página pré-definido. O menu contém opções para avançar uma página, recuar uma página ou navegar para uma determinada página, caso esta seja válida.

Performance

	Vendas_1M.txt	*Vendas_3M.txt*	*Vendas_5M.txt*
Query 1	9.51690	30.75801	51.01363
Query 8	0.05083	0.14556	0.23114
Query 9	0.00002	0.00003	0.00003
Query 10	0.49042	0.64064	0.65005
Query 11	0.00016	0.00038	0.00042
Query 12	0.03090	0.02257	0.02084

Tabela 1 - Resultados das queries, em segundos.

A **Tabela 1** representa a média do tempo de execução do programa, dos nossos três computadores, nas *queries* 1, 8, 9, 10, 11 e 12. A tabela mostra também a diferença

dos tempos destas *queries* para os ficheiros de *Vendas_1M.txt*, *Vendas_3M.txt* e *Vendas_5M.txt*. A *Query 1* representa o carregamento e tratamento dos ficheiros. Observando a tabela verifica-se que o tratamento e carregamento de um milhão de vendas demora cerca de 10 segundos.

Da *query* 8 à *query* 12, verifica-se que a mais lenta é a Q*uery* 10. O tempo de execução desta query aumenta significativamente com o processamento de 3 milhões de vendas comparativamente ao processamento de 1 milhão de vendas. No entanto, o tempo da Q*uery* 10 é praticamente idêntico para o ficheiro de 3 milhões e para o ficheiro de 5 milhões de vendas. O tempo da *Query* 8 aumenta progressivamente com a leitura de ficheiros com um número superior de vendas. Já as q*ueries* 9, 11 e 12 mantêm um tempo praticamente constante independentemente do número de vendas lidas.

Makefile

Seguindo a sugestão do professor para a criação da *makefile*, procedemos com o seu desenvolvimento. Ou seja, na *makefile*, para cada módulo, juntamente com as suas dependências, criámos o ficheiro objecto correspondente. Criámos todos os objectos para serem posteriormente compilados e resultarem no nosso programa final.

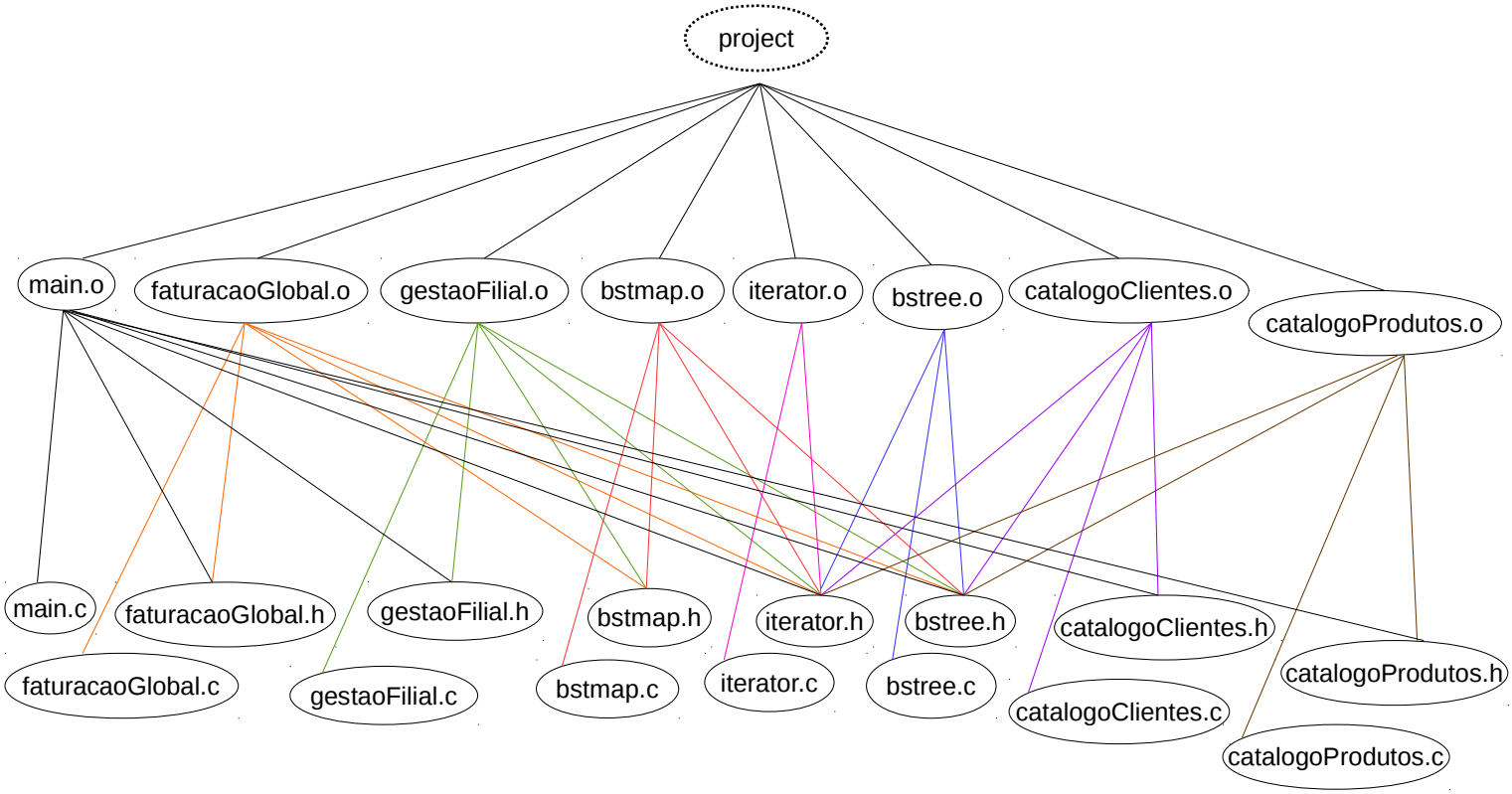
Para uma melhor consistência e qualidade de código, utilizámos as seguintes *flags* para compilar o mesmo: *Wall*, *Wextra*, *Wconversion*, *ansi* e *pedantic*. Além disso, achámos que seria vantajoso utilizar o nível de otimização 2 (*O2*) por predefinição. Assim, observámos que, de facto, os tempos foram melhorados sem sacrificar os resultados das *queries*. Ainda, utilizámos a *flag g* visto termos recorrido ao *gdb* para a correção de eventuais erros ou *bugs*.

Por outro lado, enquanto elaborávamos este relatório, apercebemos-nos que a *makefile* submetida continha dois erros que não foram detetados, já que não ocorreram erros de compilação. Um deles encontra-se na variável *OBJS.* Aqui, dizemos que *catalogoCientes.c* faz parte da mesma quando, claramente, seria *catalogoClientes.o*. E, ainda, ao compilar *catalogoClientes.o* referimos que o ficheiro *.c* correspondente é *catalogoCientes.c* quando devia ser *catalogoClientes.c*.

O nosso projeto contém, além da main, sete módulos, dos quais apenas *iterator.** é plenamente independente. Por outro lado, *bstree.** apenas apresenta uma dependência com *iterator.h*. Além disso, *bstree.**, dentro dos módulos com dependências, é aquele com menor número. Denote-se que *gestaoFilial.** e *faturacaoGlobal.** não dependem um do outro. Ainda, pelo facto do nosso projecto apresentar um número razoável de dependências, tentámos otimizar o grafo de dependências. Isto é, tentámos encontrar uma disposição das dependências favorável à fácil localização das mesmas, utilizando cores para as salientar.

Ao analisar o grafo de dependências apercebe-mos-nos de que uma das razões pelas quais temos um número razoável de dependências modulares deve-se ao uso de estruturas genéricas podendo ser utilizadas em vários contextos com diversos tipos de estruturas. Podemos observar no grafo que 6 módulos dependem do módulo do iterador (i*terator.h*). Assim, acreditarmos que o nosso programa apresenta uma boa reutilização de código (e nomeadamente dos módulos).

Grafo de Dependências



Conclusão

Com a realização deste trabalho concluímos que a divisão de um programa de larga escala em pequenos módulos apresenta inúmeras vantagens a nível de programação. Em particular, na criação de módulos independentes entre si, consegue-se muito facilmente detetar erros e *bugs*, para além de que sendo estes módulos independentes, podem ser reutilizados noutras aplicações.

Ao longo deste trabalho tentámos ser o mais genéricos possível a nível da criação de módulos, nomeadamente nos módulos que implementam estruturas de dados (*bstree*, *bstmap* e *iterator*). Para cada módulo criámos também, sempre que possível, funções genéricas. Assim, os módulos criados apresentam um nível de reutilização ótima.

Neste trabalho utilizámos tipos de dados opacos, ou seja, dados que só são acessíveis dentro do próprio módulo. Isto melhora a extensibilidade de cada módulo, pois consegue-se mudar as estruturas de dados de cada módulo sem qualquer impacto no cliente. Os tipos opacos impedem também a um cliente de aceder diretamente aos campos privados de uma estrutura, o que traz mais segurança ao bom funcionamento de um programa.

Também concluímos que as árvores binárias de procura do tipo AVL são ótimas estruturas para armazenar enormes quantidades de dados, pois são bastante eficientes no que toca à inserção de um elemento e procura deste.