Portuguese

Analytics - Estrutura de Dados

1.0.0-alpha

Gerado por Doxygen 1.8.13

Contents

Chapter 1

Analytics - Estrutura de Dados

1.1 Introdução

Trabalho final da cadeira de Estruturas de Dados - 2018/1
Instituto de Informática - UFRGS
O enunciado do trabalho pode ser encontrado aqui: `https://moodle.inf.ufrgs.br/pluginfile.↩ php/123790/mod_resource/content/0/trabalho2018-1.pdf`

1.2 Compilação

Para compilar o trabalho, abra a pasta do arquivo na linha de comando e digite `make` para rodar o makefile.

1.3 Utilização

Pela linha de comando, rode `./arvores data/entrada.txt data/operacoes.txt data/saida.↩ txt`
Podes criar um alias para isso, o que facilita bastante.

Chapter 2

Analytics - Estruturas de Dados

Trabalho final da cadeira de Estrutura de Dados - UFRGS - 2018/1 O enunciado do trabalho pode ser encontrado aqui.

Compilação

Para compilar o trabalho, abra a pasta do arquivo na linha de comando e digite `make` para rodar o makefile.

Utilização

Pela linha de comando, rode `./arvores data/entrada.txt data/operacoes.txt data/saida.txt` Podes criar um alias para isso, o que facilita bastante.

Chapter 3

Índice das estruturas de dados

3.1 Estruturas de dados

Lista das estruturas de dados com uma breve descrição:

Chapter 4

Índice dos ficheiros

4.1 Lista de ficheiros

Lista de todos os ficheiros documentados com uma breve descrição:

Chapter 5

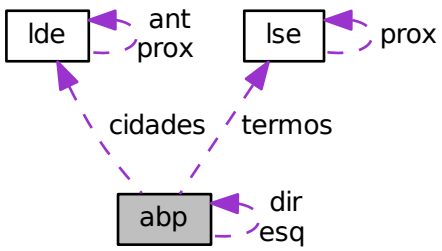
Documentação da classe

5.1 Referência à estrutura abp

Arvore binária.

`#include <struct.h>`

Diagrama de colaboração para abp:



Campos de Dados

- int qtdeAcessos
- int qtdeTermos
- LSE * termos
- LDE * cidades
- struct abp * esq
- struct abp * dir

5.1.1 Descrição detalhada

Arvore binária.

Arvore binária de pesquisa responsável por guardar todos os dados das consultas. Cada nodo da árvore representa uma consulta, armazenando os termos da consulta, além das cidades que realizaram essa consulta.

Aviso

A árvore é binária de pesquisa, PORÉM a chave dela não é tão útil para a realização das pesquisas.

5.1.2 Documentação dos campos e atributos

5.1.2.1 cidades

```
LDE* cidades
```

`LDE` contendo as cidades que realizaram essa pesquisa

5.1.2.2 dir

```
struct abp* dir
```

Ponteiro que aponta para o elemento-filho direito desse nodo

5.1.2.3 esq

```
struct abp* esq
```

Ponteiro que aponta para o elemento-filho esquerdo desse nodo

5.1.2.4 qtdeAcessos

```
int qtdeAcessos
```

Quantidade de vezes que essa consulta foi realizada. É a CHAVE da árvore

5.1.2.5 qtdeTermos

```
int qtdeTermos
```

Tamanho da lista representada por `termos`

5.1.2.6 termos

```
LSE* termos
```

LSE contendo os termos da pesquisa

A documentação para esta estrutura foi gerada a partir do seguinte ficheiro:

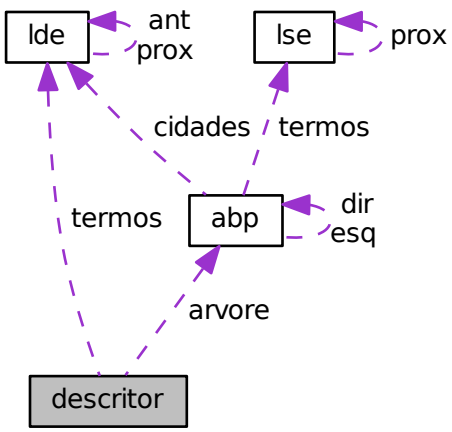
- struct.h

5.2 Referência à estrutura descritor

Estrutura principal do programa, responsável por guardar TODOS os dados.

```
#include <struct.h>
```

Diagrama de colaboração para descritor:



Campos de Dados

- Consulta * arvore
- LDE * termos

5.2.1 Descrição detalhada

Estrutura principal do programa, responsável por guardar TODOS os dados.

Estrutura principal do programa, que guarda tanto a árvore binária de pesquisa `arvore`, como a lista geral de termos consultados `termos`.

5.2.2 Documentação dos campos e atributos

5.2.2.1 arvore

```
Consulta* arvore
```

<struct abp> contendo todas as informações do arquivo

5.2.2.2 termos

```
LDE* termos
```

Lista duplamente encadeada contendo a lista geral de termos consultados

A documentação para esta estrutura foi gerada a partir do seguinte ficheiro:

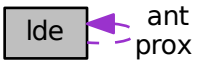
- struct.h

5.3 Referência à estrutura lde

Lista duplamente encadeada.

```
#include <struct.h>
```

Diagrama de colaboração para lde:



Campos de Dados

- char nome [100]
- int qtde
- struct lde * prox
- struct lde * ant

5.3.1 Descrição detalhada

Lista duplamente encadeada.

Lista duplamente encadeada, que pode ou não ser circular.

5.3.2 Documentação dos campos e atributos

5.3.2.1 ant

```
struct lde* ant
```

Ponteiro que aponta para o elemento anterior da lista

5.3.2.2 nome

```
char nome[100]
```

String de tamanho máximo 100

5.3.2.3 prox

```
struct lde* prox
```

Ponteiro que aponta para o próximo elemento da lista

5.3.2.4 qtde

```
int qtde
```

Inteiro que armazena quantas vezes esse nodo foi acessado e/ou chamado e/ou requisitado

A documentação para esta estrutura foi gerada a partir do seguinte ficheiro:

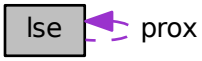
- struct.h

5.4 Referência à estrutura lse

Lista simplesmente encadeada.

`#include <struct.h>`

Diagrama de colaboração para lse:



Campos de Dados

- char termo [100]
- struct lse * prox

5.4.1 Descrição detalhada

Lista simplesmente encadeada.

5.4.2 Documentação dos campos e atributos

5.4.2.1 prox

```
struct lse* prox
```

Ponteiro que aponta para o próximo elemento da lista

5.4.2.2 termo

```
char termo[100]
```

String de tamanho máximo 100

A documentação para esta estrutura foi gerada a partir do seguinte ficheiro:

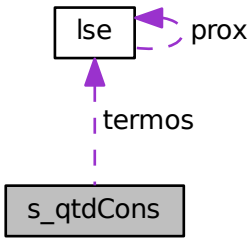
- struct.h

5.5 Referência à estrutura s_qtdCons

Estrutura utilizada para guardar uma LSE e um valor numérico arbitrário.

`#include <struct.h>`

Diagrama de colaboração para s_qtdCons:



Campos de Dados

- int **qtd**
- LSE * **termos**

5.5.1 Descrição detalhada

Estrutura utilizada para guardar uma LSE e um valor numérico arbitrário.

Estrutura que é utilizada somente na função consultasPorLocalidade() para poder armazenar quantas vezes uma `LSE*` foi chamada.

A documentação para esta estrutura foi gerada a partir do seguinte ficheiro:

- struct.h

Chapter 6

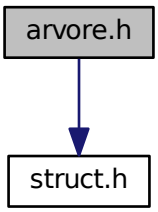
Documentação do ficheiro

6.1 Referência ao ficheiro arvore.h

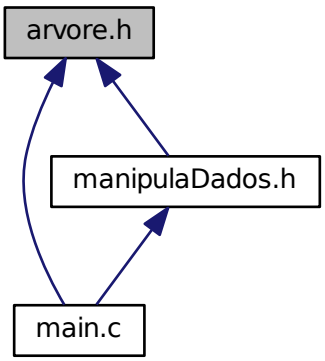
Arquivo que contém funções relacionadas a manipulação de dados em arvores.

```
#include "struct.h"
```

Diagrama de dependências de inclusão para arvore.h:



Este grafo mostra quais são os ficheiros que incluem directamente ou indirectamente este ficheiro:



Macros

- #define **QTD_TERMOS** 1
- #define **QTD_ACESSOS** 2

Funções

- Consulta ∗ criaArvore ()

  Cria árvore.
- Consulta ∗ insereNodoArvore (Consulta ∗arvore, LSE ∗listaTermos, int qtdTermos, char ∗cidade)

  Insere um nodo na árvore.
- int percorreArvore (Consulta ∗nodo, int nivel)

  Logging de informações sobre uma árvore.

6.1.1 Descrição detalhada

Arquivo que contém funções relacionadas a manipulação de dados em arvores.

Autor

Rafael Baldasso Audibert
Augusto Zanella Bardini

Data

11 Jul 2018

6.1.2 Documentação das funções

6.1.2.1 criaArvore()

```
Consulta* criaArvore ( )
```

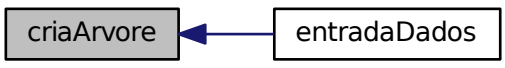
Cria árvore.

A função criaArvore() cria uma arvore vazia.

Retorna

`NULL` sempre é retornado.

Este é o diagrama das funções que utilizam esta função:



6.1.2.2 insereNodoArvore()

```
Consulta* insereNodoArvore (
            Consulta * arvore,
            LSE * listaTermos,
            int qtdTermos,
            char * cidade )
```

Insere um nodo na árvore.

A função insereNodoArvore() é responsável por inserir um nodo na árvore (ou incrementar ou contador, caso o nodo já exista)

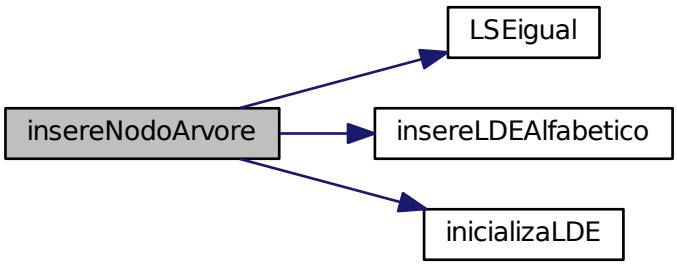
Parâmetros

arvore	Árvore na qual será inserido o novo nodo
listaTermos	Lista dos termos que esse novo nodo irá conter (consulta)
qtdTermos	Tamanho da `listaTermos`
cidade	String com o nome da cidade na qual foi realizada a consulta

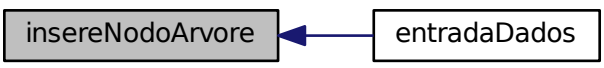
Retorna

`*Consulta` contendo a arvore recebida, com a adição/incrementação do novo nodo.

Grafo de chamadas desta função:



Este é o diagrama das funções que utilizam esta função:



6.1.2.3 percorreArvore()

```
int percorreArvore (
            Consulta * nodo,
            int nivel )
```

Logging de informações sobre uma árvore.

A função percorreArvore() é responsável por printar na tela do terminal, informações a respeito de cada nodo da árvore

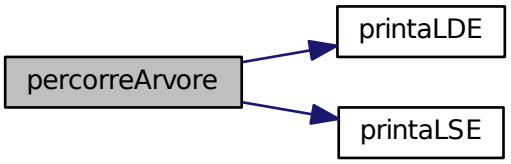
Parâmetros

nodo	Nodo inicial da árvore
nivel	Valor inicial para o nível da árvore (Para satisfazer ambas convenções de nível inicial = 0 ou 1)

Retorna

Altura da arvore - 1 + `nivel`

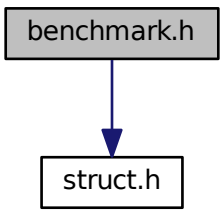
Grafo de chamadas desta função:



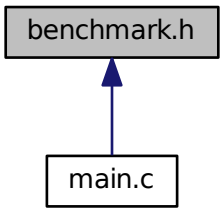
6.2 Referência ao ficheiro benchmark.h

Arquivo que contém funções relacionadas a medição do tempo gasto para realizar as operações de entrada//saida de dados.

`#include "struct.h"`
Diagrama de dependências de inclusão para benchmark.h:



Este grafo mostra quais são os ficheiros que incluem directamente ou indirectamente este ficheiro:



Funções

- Info ∗ infoBenchmark (Info ∗(∗function)(FILE ∗), FILE ∗entrada)

  Benchmark da entrada de dados.
- void operacoesBenchmark (int(∗function)(FILE ∗, FILE ∗, Info ∗), FILE ∗operacoes, FILE ∗saida, Info ∗dados)

  Benchmark da saida de dados.

6.2.1 Descrição detalhada

Arquivo que contém funções relacionadas a medição do tempo gasto para realizar as operações de entrada//saida de dados.

Autor

Rafael Baldasso Audibert
Augusto Zanella Bardini

Data

11 Jul 2018

6.2.2 Documentação das funções

6.2.2.1 infoBenchmark()

```
Info* infoBenchmark (
            Info *(*)(FILE *) function,
            FILE * entrada )
```

Benchmark da entrada de dados.

A função infoBenchmark() é responsável pela medição do tempo que a função `function` demora para realizar a inserção dos dados na estrutura.

Parâmetros

function	Função da qual será medido o tempo. OBS.: Ela precisa ser do tipo `Info*`, e receber como parâmetro `FILE*`
entrada	Arquivo que contém os dados de entrada. Será passado como parâmetro para `function`.

Retorna

`Info*` que é o mesmo que é retornado por `function`.

6.2.2.2 operacoesBenchmark()

```
void operacoesBenchmark (
             int(*)(FILE *, FILE *, Info *) function,
             FILE * operacoes,
             FILE * saida,
             Info * dados )
```

Benchmark da saida de dados.

A função operacoesBenchmark() é responsável pela medição do tempo que a função `function` demora para realizar as operações e escrever a saída dos dados em um arquivo.

Parâmetros

function	Função da qual será medido o tempo. OBS.: Ela precisa ser do tipo `int`, e receber como parâmetro <FILE*, FILE*, Info*>
operacoes	Arquivo que contém os dados das operações a serem realizadas em `dados`. Será passado como parâmetro para `function`.
saida	Arquivo onde serão impressos os dados de saída. Será passado como parâmetro para `function`.
dados	Dados onde serão realizadas as operações descritas em `operacoes`. Será passado como parâmetro para `function`.

Retorna

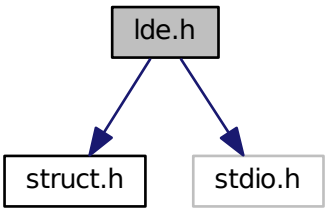
`int` que é a quantidade de operações realizadas com sucesso.

6.3 Referência ao ficheiro lde.h

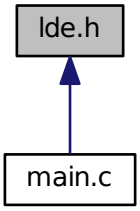
Arquivo que contém funções relacionadas a manipulação de listas duplamente encadeadas.

```
#include "struct.h"
#include <stdio.h>
```

Diagrama de dependências de inclusão para lde.h:



Este grafo mostra quais são os ficheiros que incluem directamente ou indirectamente este ficheiro:



Funções

- LDE ∗ inicializaLDE ()

  Cria uma lista duplamente encadeada.
- LDE ∗ insereLDEAlfabetico (LDE ∗lista, char ∗nome)

  Insere um item na LDE, deixando-a em ordem alfabética.
- LDE ∗ insereLDENumerico (LDE ∗lista, char ∗nome, int qtde)

  Insere um item na LDE, deixando-a em ordem numérica.
- void printaLDE (LDE ∗lista, int qtde, FILE ∗saida)

  Logging de informações sobre a lista.

6.3.1 Descrição detalhada

Arquivo que contém funções relacionadas a manipulação de listas duplamente encadeadas.

Autor

Rafael Baldasso Audibert
Augusto Zanella Bardini

Data

11 Jul 2018

6.3.2 Documentação das funções

6.3.2.1 inicializaLDE()

```
LDE* inicializaLDE ( )
```

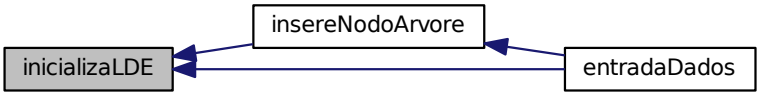
Cria uma lista duplamente encadeada.

A função inicializaLDE() cria uma lista duplamente encadeada vazia.

Retorna

NULL sempre é retornado.

Este é o diagrama das funções que utilizam esta função:



6.3.2.2 insereLDEAlfabetico()

```
LDE* insereLDEAlfabetico (
            LDE * lista,
            char * nome )
```

Insere um item na LDE, deixando-a em ordem alfabética.

A função insereLDEAlfabetico() é responsável por inserir um item na LDE, fazendo que com a LDE sempre fique em ordem alfabética. Caso um item seja igual a algum que já está inserido, é incrementado um contador, mostrando que aquele termo foi inserido mais de uma vez.

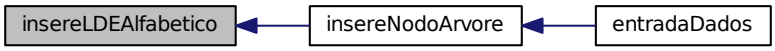
Parâmetros

lista	Lista Duplamente Encadeada na qual será inserido o novo item
nome	String contendo o item que será inserido na lista

Retorna

*LDE contendo a lista recebida, com a adição/incrementação do novo item.

Este é o diagrama das funções que utilizam esta função:



6.3.2.3 insereLDENumerico()

```
LDE* insereLDENumerico (
            LDE * lista,
            char * nome,
            int qtde )
```

Insere um item na LDE, deixando-a em ordem numérica.

A função insereLDENumerico() é responsável por inserir um item na LDE, fazendo que com a LDE sempre fique em ordem numérica decrescente. Caso um item tenha o mesmo valor numérico que outro, eles são ordenados de forma alfabética. Caso um item com a mesma quantidade numérica e tenha seu nome igual a algum que já está inserido, é incrementado um contador, mostrando que aquele termo foi inserido mais de uma vez.

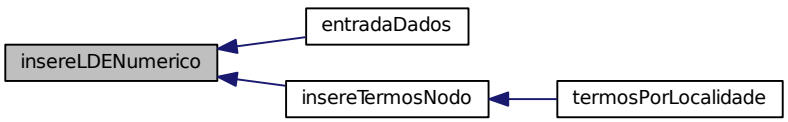
Parâmetros

lista	Lista Duplamente Encadeada na qual será inserido o novo item
nome	String contendo o item que será inserido na lista
qtde	Numero que representa a quantidade de vezes que ele foi chamado.

Retorna

*LDE contendo a lista recebida, com a adição/incrementação do novo item.

Este é o diagrama das funções que utilizam esta função:



6.3.2.4 printaLDE()

```
void printaLDE (
            LDE * lista,
            int qtde,
            FILE * saida )
```

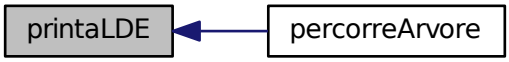
Logging de informações sobre a lista.

A função printaLDE() é responsável por printar em um arquivo, informações a respeito de `qtde` itens da lista. Se o parâmetro `qtde` for passado como 0, a lista inteira será printada. Essa informação é printada no formato "@<qtde, nome@>".

Parâmetros

lista	Lista que iremos printar
qtde	Quantidade de termos da lista que serão printados.
saida	Arquivo no qual será printado a informação (pode ser passado stdout, para printar no terminal)

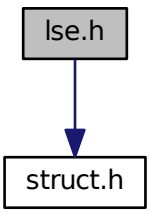
Este é o diagrama das funções que utilizam esta função:



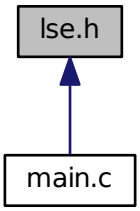
6.4 Referência ao ficheiro lse.h

Arquivo que contém funções relacionadas a manipulação de listas simplesmente encadeadas.

`#include "struct.h"`
Diagrama de dependências de inclusão para lse.h:



Este grafo mostra quais são os ficheiros que incluem directamente ou indirectamente este ficheiro:



Funções

- LSE ∗ inicializaLSE ()

  Cria uma lista simplesmente encadeada.
- LSE ∗ insereLSE (LSE ∗lista, char ∗termo)

  Insere um nodo em uma lista simplesmente encadeada.
- int LSEigual (LSE ∗lse1, LSE ∗lse2)

  Retorna 1 se duas LSE são identicas.
- void printaLSE (LSE ∗lista, FILE ∗saida)

  Logging de informações sobre a lista.
- char ∗ parseLSEtoString (LSE ∗lista, char ∗string)

  Transforma uma LSE em uma string.

6.4.1 Descrição detalhada

Arquivo que contém funções relacionadas a manipulação de listas simplesmente encadeadas.

Autor

Rafael Baldasso Audibert
Augusto Zanella Bardini

Data

11 Jul 2018

6.4.2 Documentação das funções

6.4.2.1 inicializaLSE()

```
LSE* inicializaLSE ( )
```

Cria uma lista simplesmente encadeada.

A função inicializaLSE() cria uma lista simplesmente encadeada vazia.

Retorna

NULL sempre é retornado.

Este é o diagrama das funções que utilizam esta função:

6.4.2.2 insereLSE()

```
LSE* insereLSE (
            LSE * lista,
            char * termo )
```

Insere um nodo em uma lista simplesmente encadeada.

A função insereLSE() insere um nodo em uma lista simplesmente encadeada já existente, mantendo-a ordenada em ordem alfabética.

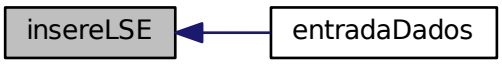
Parâmetros

lista	Lista na qual será inserida o novo termo
termo	String do termo que será inserido na lista

Retorna

`LSE*` contendo a lista recebido com a adição do novo termo.

Este é o diagrama das funções que utilizam esta função:



6.4.2.3 LSEigual()

```
int LSEigual (
              LSE * lse1,
              LSE * lse2 )
```

Retorna 1 se duas LSE são identicas.

A função LSEigual() é responsável por dizer se `lse1` e `lse2` são idênticas.

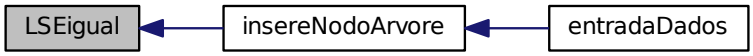
Parâmetros

lse1	Lista 1
lse2	LIsta 2

Retorna

`1` se as listas forem idênticas, `0` se as listas forem diferentes

Este é o diagrama das funções que utilizam esta função:



6.4.2.4 parseLSEtoString()

```
char* parseLSEtoString (
            LSE * lista,
            char * string )
```

Transforma uma LSE em uma string.

A função parseLSEtoString() é responsável por transformar uma LSE em uma string formatada. Seu formato é ""<termo; termo; termo; termo>".

Parâmetros

lista	Lista que iremos transformar em string
string	Ponteiro para o local onde guardaremos a string

Retorna

`*char` onde iremos guardar a string

Nota

Não é necessário utilizar o retorno dessa função, já que ela já recebe o ponteiro pra string na qual ela será guardada como parâmetro. Ela é retornada apenas por convenção.

6.4.2.5 printaLSE()

```
void printaLSE (
            LSE * lista,
            FILE * saida )
```

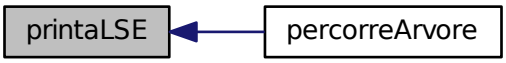
Logging de informações sobre a lista.

A função printaLSE() é responsável por printar em um arquivo, informações a respeito da lista. Essa informação é printada no formato "@<termo;termo;termo;...@>".

Parâmetros

lista	Lista que iremos printar
saida	Arquivo no qual será printado a informação (pode ser passado stdout, para printar no terminal)

Este é o diagrama das funções que utilizam esta função:

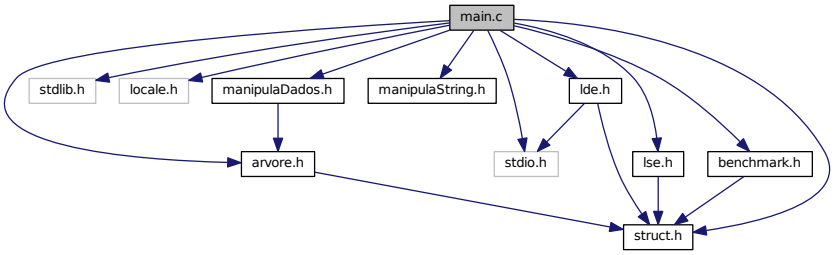


6.5 Referência ao ficheiro main.c

Arquivo que contém a main do programa Analytics.

```
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
#include "struct.h"
#include "manipulaString.h"
#include "arvore.h"
#include "lse.h"
#include "manipulaDados.h"
#include "lde.h"
#include "benchmark.h"
```

Diagrama de dependências de inclusão para main.c:



Macros

- #define **F_ENTRADA** "data/input.txt"
- #define **F_OPERACOES** "data/operations.txt"
- #define **F_SAIDA** "data/saida.txt"

Funções

- int main (int argc, char ∗∗argv)

  Função main do programa Analytics.

6.5.1 Descrição detalhada

Arquivo que contém a main do programa Analytics.

Autor

Rafael Baldasso Audibert
Augusto Zanella Bardini

Data

11 Jul 2018

6.5.2 Documentação das funções

6.5.2.1 main()

```
int main (
            int argc,
            char ** argv )
```

Função main do programa Analytics.

A função main() é a função principal do programa Analytics, e é por ela que o programa começa

Parâmetros

argc	Quantidade de argumentos passados para a execução do programa
∗∗*argv*	Argumentos passados para a execução do programa

Retorna

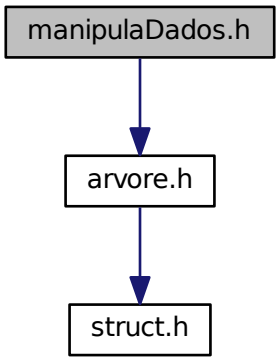
0 caso tudo ocorra sem problemas
1 caso o arquivo de entrada não possa ser aberto;
2 caso o arquivo de operações não possa ser aberto;
3 caso o arquivo de saída não possa ser criado;

6.6 Referência ao ficheiro manipulaDados.h

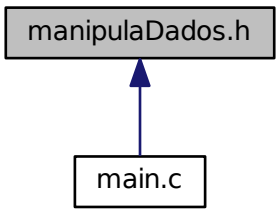
Arquivo que contém as principais funções do programa.

```
#include "arvore.h"
```

Diagrama de dependências de inclusão para manipulaDados.h:



Este grafo mostra quais são os ficheiros que incluem directamente ou indirectamente este ficheiro:



Funções

- Info ∗ entradaDados (FILE ∗entrada)

  Leitura dos dados do arquivo.
- int realizaOperacoes (FILE ∗operacoes, FILE ∗saida, Info ∗dados)

  Realização das operações nos dados recebidos.

6.6.1 Descrição detalhada

Arquivo que contém as principais funções do programa.

Autor

Rafael Baldasso Audibert
Augusto Zanella Bardini

Data

11 Jul 2018 Nesse arquivo encontramos as duas principais funções do programa, sendo elas responsáveis por controlar a leitura dos arquivos de entrada e operações, além de realizar toda a manipulação das estruturas e das outras funções.

6.6.2 Documentação das funções

6.6.2.1 entradaDados()

```
Info* entradaDados (
            FILE * entrada )
```

Leitura dos dados do arquivo.

A função entradaDados() realiza a leitura dos dados do arquivo de entrada, manipulando as outras funções do programa, fazendo com que cada valor termine em sua devida estrutura, pronta para ser utilizada pela função realizaOperacoes().

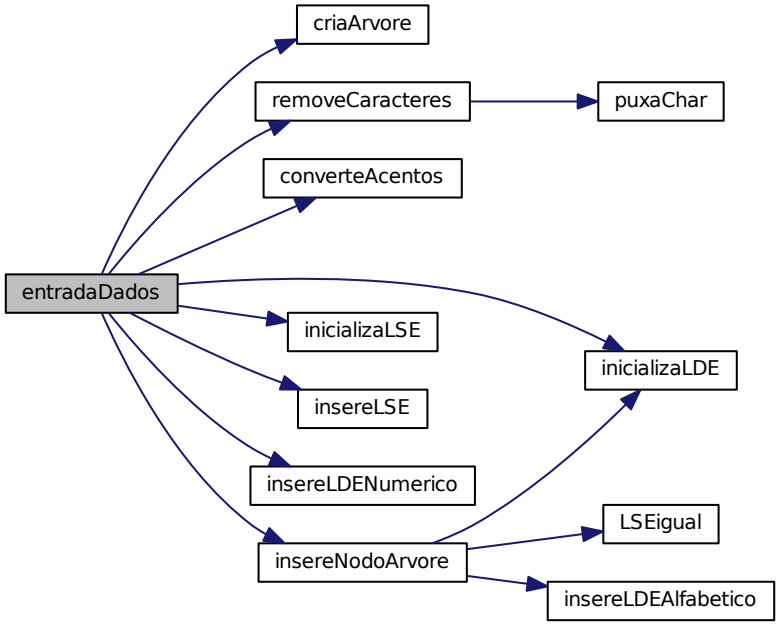
Parâmetros

entrada	Nome do arquivo de entrada

Retorna

`Info*` contendo a árvore com as consultas e a lista de termos totais do arquivo.

Grafo de chamadas desta função:



6.6.2.2 realizaOperacoes()

```
int realizaOperacoes (
            FILE * operacoes,
            FILE * saida,
            Info * dados )
```

Realização das operações nos dados recebidos.

A função realizaOperacoes() realiza a leitura das operacoes, realizando buscas nos dados fazendo com que toda a informação solicitada seja escrita no arquivo de saída.

Parâmetros

operacoes	Nome do arquivo que contém as operações a serem realizadas nos `dados`.
saida	Nome do arquivo no qual serão escritos os dados de saída
dados	Informações geradas pela função entradaDados() a partir dos dados que haviam no arquivo de entrada.

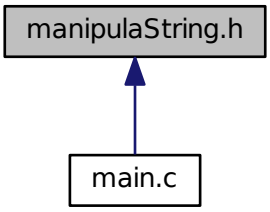
Retorna

`int` que representa a quantidade de operações realizadas.

6.7 Referência ao ficheiro manipulaString.h

Arquivo que contém funções relacionadas a manipulação de strings.

Este grafo mostra quais são os ficheiros que incluem directamente ou indirectamente este ficheiro:



Funções

- void converteAcentos (char ∗str)

  Tira os acentos de uma string.
- void removeCaracteres (char ∗str)

  Remove não-letras de uma string.
- void puxaChar (char ∗c)

  Remove um caractere de uma string.

6.7.1 Descrição detalhada

Arquivo que contém funções relacionadas a manipulação de strings.

Autor

Rafael Baldasso Audibert
Augusto Zanella Bardini

Data

11 Jul 2018

6.7.2 Documentação das funções

6.7.2.1 converteAcentos()

```
void converteAcentos (
            char * str )
```

Tira os acentos de uma string.

A função converteAcentos() tira todos os acentos de uma string, transformando, i.e. à -> a

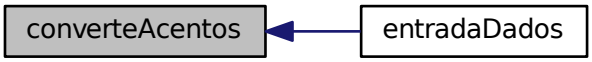
Parâmetros

str	String que será convertida

Aviso

A função funciona com strings escritas dentro do terminal, porém se for lido de um arquivo, pode ser que não funcione por causa das diferentes codificações.

Este é o diagrama das funções que utilizam esta função:



6.7.2.2 puxaChar()

```
void puxaChar (
            char * c )
```

Remove um caractere de uma string.

A função puxaChar() é a função auxiliar da removeCaracteres(). Ela remove um caractere de uma string, "puxando" todos os outros caracteres para não ficar um buraco no lugar do caractere a ser removido.

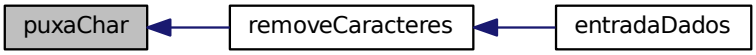
Parâmetros

c	Caractere a ser removido.

Nota

A função precisa que esse caractere esteja dentro de uma string, já que puxará todos os caracteres até encontrar um '\0'.

Este é o diagrama das funções que utilizam esta função:



6.7.2.3 removeCaracteres()

```
void removeCaracteres (
            char * str )
```

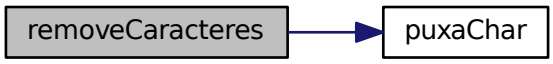
Remove não-letras de uma string.

A função removeCaracteres() tira todos os caracteres que não sejam letras e/ou números de uma string.

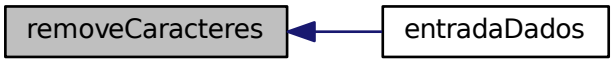
Parâmetros

str	String na qual será feita a remoção

Grafo de chamadas desta função:



Este é o diagrama das funções que utilizam esta função:

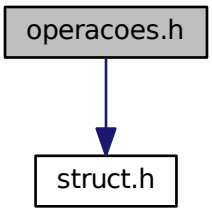


6.8 Referência ao ficheiro operacoes.h

Arquivo que contém funções executadas durante a leitura do arquivo de operacoes.

```
#include "struct.h"
```

Diagrama de dependências de inclusão para operacoes.h:



Macros

- #define **TAM_VET** 11000

Funções

- void consultasPorLocalidade (Consulta ∗arvore, char ∗cidade, int qtdConsultas, FILE ∗saida)

  Encontra e escreve em um arquivo as consultas mais realizadas em uma dada cidade.
- int consultasArquivo (Consulta ∗arvore, Consulta retorno[ ], int qtdConsultas)

  Encontra as consultas mais realizadas em todo o arquivo.
- LDE ∗ termosPorLocalidade (Consulta ∗arvore, LDE ∗lista, char cidade[ ])

  Encontra os termos consultados em uma localidade.
- LDE ∗ termosArquivo (LDE ∗listaTermos)

  Encontra os termos consultados em um arquivo.
- int mediaTamanhoConsultasLocalidade (Consulta ∗arvore, char ∗cidade)

  Retorna a média do tamanho das consultas realizadas em uma cidade.
- int mediaTamanhoConsultasArquivo (Consulta ∗arvore)

  Retorna a média do tamanho das consultas realizadas em todo o arquivo.
- LDE ∗ insereTermosNodo (LDE ∗lista, LSE ∗termos, int qtde)

  Concatena todos os termos de uma LSE em uma LDE.
- void auxiliarMediaTamanhoConsultasArquivo (Consulta ∗arvore, int ∗totTermos, int ∗totConsultas)

  Calcula a quantidade total de consultas e a quantidade total de termos no arquivo.
- void auxiliarMediaTamanhoConsultasLocalidade (Consulta ∗arvore, int ∗totTermos, int ∗totConsultas, char ∗cidade)

  Calcula a quantidade total de consultas e a quantidade total de termos em uma dada cidade.
- int temCidadeNaLista (char ∗cidade, LDE ∗lista)

  Retorna a "quantidade" de vezes que um termo aparece em uma LDE.
- int achaVetorReps (Consulta ∗arvore, int ∗vetor, int contador)

  Copia a quantidade de acessos de cada nodo da arvore pra posições de um vetor.
- int achaVetorRepsLocalidade (Consulta ∗arvore, int ∗vetor, int contador, char ∗cidade, Qtdcons ∗qtdCons)

  Encontra as repetições de uma consulta por localidade.
- int copiaArvore (Consulta ∗arvore, Consulta ∗retorno, int ∗vetor, int qtd, int pos, int vezesRep)

  Copia uma arvore para a outra, de maneira ordenada.
- void quick_sort (int ∗a, int left, int right)

  Quick sort de vetores int genéricos.

6.8.1 Descrição detalhada

Arquivo que contém funções executadas durante a leitura do arquivo de operacoes.

Autor

Rafael Baldasso Audibert
Augusto Zanella Bardini

Data

11 Jul 2018

6.8.2 Documentação das funções

6.8.2.1 achaVetorReps()

```
int achaVetorReps (
            Consulta * arvore,
            int * vetor,
            int contador )
```

Copia a quantidade de acessos de cada nodo da arvore pra posições de um vetor.

A função achaVetorReps() recebe uma arvore, e pra cada nodo dessa árvore encontrado recursivamente, copia a quantidade de acessos dele para um vetor

Parâmetros

arvore	Arvore que contem as informações a serem inseridas em vetor
vetor	Vetor com as quantidades de consultas realizadas.
contador	Conta em qual posição de `vetor` será inserido a quantidade

Retorna

`int` Quantidade de consultas encontradas

Nota

Essa função é uma função auxiliar para consultasArquivo()

6.8.2.2 achaVetorRepsLocalidade()

```
int achaVetorRepsLocalidade (
            Consulta * arvore,
            int * vetor,
            int contador,
            char * cidade,
            Qtdcons * qtdCons )
```

Encontra as repetições de uma consulta por localidade.

A função achaVetorRepsLocalidade() recebe uma estrutura especial, criada somente para ela, fazendo com que ela coloque em um vetor, todas as consultas realizadas em uma cidade.

Parâmetros

arvore	Arvore que contem as informações a serem inseridas em vetor
vetor	Vetor com as quantidades de consultas realizadas.
contador	Conta em qual posição de `vetor` será inserido a quantidade
cidade	Cidade a qual estamos procurando as consultas
qtdCons	Vetor de QtdCons onde serão guardados as consultas e sua quantidade

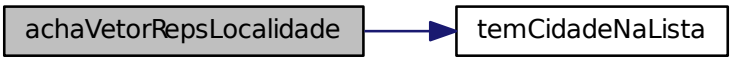
Retorna

`int` Quantidade de consultas encontradas

Nota

Essa função é uma função auxiliar para consultasPorLocalidade()

Grafo de chamadas desta função:



6.8.2.3 auxiliarMediaTamanhoConsultasArquivo()

```
void auxiliarMediaTamanhoConsultasArquivo (
            Consulta * arvore,
            int * totTermos,
            int * totConsultas )
```

Calcula a quantidade total de consultas e a quantidade total de termos no arquivo.

A função auxiliarMediaTamanhoConsultasArquivo() varre uma arvore buscando a quantidade total de consultas assim como a quantidade total de termos que possuem naquela arvore.

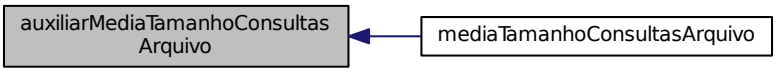
Parâmetros

arvore	Arvore a ser varrida
totTermos	Ponteiro para um inteiro que armazena a quantidade total de termos na árvore
totConsultas	Ponteiro para um inteiro que armazena a quantidade total de consultas na árvore

Nota

Essa função é uma função auxiliar para mediaTamanhoConsultasArquivo()

Este é o diagrama das funções que utilizam esta função:



6.8.2.4 auxiliarMediaTamanhoConsultasLocalidade()

```
void auxiliarMediaTamanhoConsultasLocalidade (
            Consulta * arvore,
            int * totTermos,
            int * totConsultas,
            char * cidade )
```

Calcula a quantidade total de consultas e a quantidade total de termos em uma dada cidade.

A função auxiliarMediaTamanhoConsultasLocalidade() varre uma arvore buscando a quantidade total de consultas assim como a quantidade total de termos que possuem naquela arvore e que tenham acontecido nas consultas de uma determinada cidade

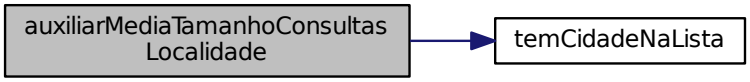
Parâmetros

arvore	Arvore a ser varrida
totTermos	Ponteiro para um inteiro que armazena a quantidade total de termos consultados pela cidade na árvore
totConsultas	Ponteiro para um inteiro que armazena a quantidade total de consultas realizadas pela cidade na árvore
cidade	String com o nome da cidade que está sendo procurada

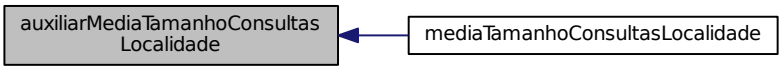
Nota

Essa função é uma função auxiliar para mediaTamanhoConsultasLocalidade()

Grafo de chamadas desta função:



Este é o diagrama das funções que utilizam esta função:



6.8.2.5 consultasArquivo()

```
int consultasArquivo (
            Consulta * arvore,
            Consulta retorno[],
            int qtdConsultas )
```

Encontra as consultas mais realizadas em todo o arquivo.

A função consultasArquivo() encontra as consultas que foram mais buscadas. Se 0 for passado como parametro para `qtdConsultas`, são mostradas todas as consultas realizadas.

Parâmetros

arvore	Arvore que contem os dados
retorno	Array de tamanho `qtdConsultas` (inicialmente), contendo as `qtdConsultas` mais realizadas em todo o arquivo
qtdConsultas	Quantidade de consultas que serão buscadas na `arvore`.

Retorna

`int` que é a quantidade de elementos em `retorno`.

Aviso

Essa função é extremamente custosa quando se está procurando TODAS as consultas do arquivo.

6.8.2.6 consultasPorLocalidade()

```
void consultasPorLocalidade (
            Consulta * arvore,
            char * cidade,
            int qtdConsultas,
            FILE * saida )
```

Encontra e escreve em um arquivo as consultas mais realizadas em uma dada cidade.

A função consultasPorLocalidade() encontra e escreve em um arquivo as `qtdConsultas` mais realizadas em uma dada cidade. Se 0 for passado como parametro para `qtdConsultas`, são mostradas todas as consultas realizadas naquela localidade.

Parâmetros

arvore	Arvore que contem os dados
cidade	String da cidade que estamos procurando os dados
qtdConsultas	Quantidade de consultas que serão printadas no arquivo
saida	Arquivo no qual serão printadas as `qtdConsultas` mais consultadas.

6.8.2.7 copiaArvore()

```
int copiaArvore (
            Consulta * arvore,
            Consulta * retorno,
            int * vetor,
            int qtd,
            int pos,
            int vezesRep )
```

Copia uma arvore para a outra, de maneira ordenada.

A função copiaArvore() copia a arvore `arvore`, para a arvore `retorno`, de maneira ordenada, seguindo o ordenamento dado por `vetor`

Parâmetros

arvore	Arvore que contem as informações
retorno	Arvore para a qual serao copiados os valores
vetor	Vetor ordenado, com o qual procuramos os nodos certos a serem inseridos
qtd	Até qual posição de `vetor` preciso preencher
pos	Minha posição de preenchimento atual de `vetor`
vezesRep	Vezes que a repetição ocorreu

Retorna

`int` Usado dentro das recursões para saber em que repetição da recursão estamos

Nota

Essa função é uma função auxiliar para consultasArquivo()

6.8.2.8 insereTermosNodo()

```
LDE* insereTermosNodo (
            LDE * lista,
            LSE * termos,
            int qtde )
```

Concatena todos os termos de uma LSE em uma LDE.

A função insereTermosNodo() concatena todos os termos que existem em `termos` em `lista`. É passado como parametro numérico da função insereLDENumerico() o que é recebido no parametro `qtde`.

Parâmetros

lista	Lista que receberá os termos
termos	Lista da qual são copiados os termos
qtde	Valor que será passado como parametro para a função insereLDENumerico(). É a quantidade de vezes que cada termo aparecia originalmente.

Retorna

`LDE*` que é a `lista` com a adição dos novos termos.

Nota

Essa função é uma função auxiliar para termosPorLocalidade()

Grafo de chamadas desta função:

Este é o diagrama das funções que utilizam esta função:

6.8.2.9 mediaTamanhoConsultasArquivo()

```
int mediaTamanhoConsultasArquivo (
            Consulta * arvore )
```

Retorna a média do tamanho das consultas realizadas em todo o arquivo.

A função mediaTamanhoConsultasArquivo() retorna a média do tamanho das consultas realizadas em no arquivo

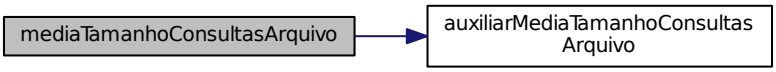
Parâmetros

arvore	Arvore com todas as consultas

Retorna

`int*` que é a média do tamanho das consultas realizadas no arquivo

Grafo de chamadas desta função:



6.8.2.10 mediaTamanhoConsultasLocalidade()

```
int mediaTamanhoConsultasLocalidade (
            Consulta * arvore,
            char * cidade )
```

Retorna a média do tamanho das consultas realizadas em uma cidade.

A função mediaTamanhoConsultasLocalidade() retorna a média do tamanho das consultas realizadas em uma determinada localidade

Parâmetros

arvore	Arvore com todas as consultas
cidade	Cidade a qual estamos procurando

Retorna

`int*` que é a média do tamanho das consultas realizadas em uma determinada cidade.

Grafo de chamadas desta função:

6.8.2.11 quick_sort()

```
void quick_sort (
            int * a,
            int left,
            int right )
```

Quick sort de vetores int genéricos.

Ordena um vetor de int genéricos usando o algoritmo de Quick Sort (https://pt.wikipedia.org/wiki/Quicksort)

Parâmetros

a	Vetor a ser ordenado
left	Detalhe de implementação relacionado ao algoritmo de Divide&Conqueror
right	Detalhe de implementação relacionado ao algoritmo de Divide&Conqueror

Nota

Essa função é uma função auxiliar

6.8.2.12 temCidadeNaLista()

```
int temCidadeNaLista (
            char * cidade,
            LDE * lista )
```

Retorna a "quantidade" de vezes que um termo aparece em uma LDE.

A função temCidadeNaLista() varre uma LDE, procurando um termo identico a `cidade`. Caso exista esse termo na LDE, retorna a "quantidade" de vezes que ele ocorre, caso contrário retorna 0. Essa <quantidade de vezes que ele ocorre> é dada pelo campo LDE::qtde da cidade.

Parâmetros

cidade	String a ser procurada em `lista`
lista	Lista que será varrida procurando por `cidade`

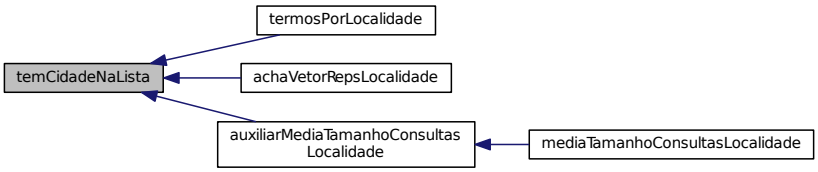
Retorna

`int` 1 se existe a string na lista e 0 se não existe a string na lista

Nota

Essa função é uma função auxiliar para achaVetorRepsLocalidade(), auxiliarMediaTamanhoConsultas↩ Localidade() e termosPorLocalidade()

Este é o diagrama das funções que utilizam esta função:



6.8.2.13 termosArquivo()

```
LDE* termosArquivo (
            LDE * listaTermos )
```

Encontra os termos consultados em um arquivo.

A função termosArquivo() encontra todos os termos que foram consultados em um arquivo.

Parâmetros

listaTermos	Lista de todos os termos consultados no arquivo
cidade	Cidade a qual estamos procurando os termos

Retorna

LDE* que é a lista de todos os termos consultados no arquivo.

Aviso

Essa função é apenas um placeholder, para manter uma convenção de sempre chamarmos uma função para as operações, já que já temos todos os termos do arquivo separados em uma estrutura própria.

6.8.2.14 termosPorLocalidade()

```
LDE* termosPorLocalidade (
            Consulta * arvore,
            LDE * lista,
            char cidade[] )
```

Encontra os termos consultados em uma localidade.

A função termosPorLocalidade() encontra todos os termos que foram consultados em uma localidade. Funciona RECURSIVAMENTE.

Parâmetros

arvore	Arvore que contem os dados
lista	LDE onde serão inseridos os termos do nodo atual da arvore.
cidade	Cidade a qual estamos procurando os termos

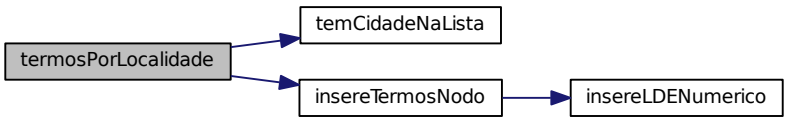
Retorna

LDE* que é a lista com todos os termos pesquisados naquela cidade.

Aviso

Na primeira chamada dessa função, a lista precisa ser VAZIA.

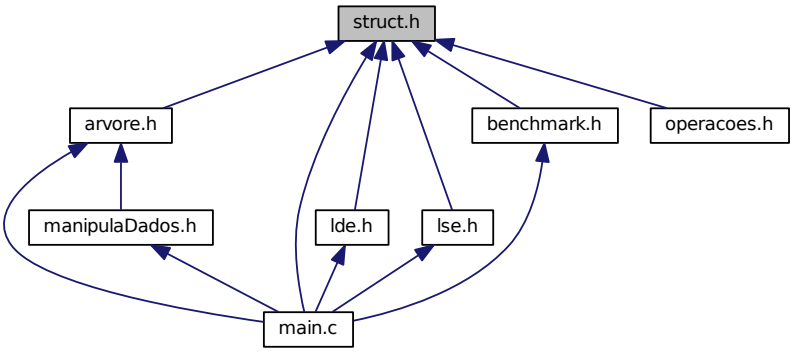
Grafo de chamadas desta função:



6.9 Referência ao ficheiro struct.h

Arquivo que contém as estruturas utilizadas no programa.

Este grafo mostra quais são os ficheiros que incluem directamente ou indirectamente este ficheiro:



Estruturas de Dados

- struct lde

  Lista duplamente encadeada.
- struct lse

  Lista simplesmente encadeada.
- struct abp

  Arvore binária.
- struct descritor

  Estrutura principal do programa, responsável por guardar TODOS os dados.
- struct s_qtdCons

  Estrutura utilizada para guardar uma LSE e um valor numérico arbitrário.

Definições de tipos

- typedef struct lde LDE

  Lista duplamente encadeada.
- typedef struct lse LSE

  Lista simplesmente encadeada.
- typedef struct abp Consulta

  Arvore binária.
- typedef struct descritor Info

  Estrutura principal do programa, responsável por guardar TODOS os dados.
- typedef struct s_qtdCons Qtdcons

  Estrutura utilizada para guardar uma LSE e um valor numérico arbitrário.

6.9.1 Descrição detalhada

Arquivo que contém as estruturas utilizadas no programa.

Autor

Rafael Baldasso Audibert
Augusto Zanella Bardini

Data

11 Jul 2018

6.9.2 Documentação dos tipos

6.9.2.1 Consulta

```
typedef struct abp Consulta
```

Arvore binária.

Arvore binária de pesquisa responsável por guardar todos os dados das consultas. Cada nodo da árvore representa uma consulta, armazenando os termos da consulta, além das cidades que realizaram essa consulta.

Aviso

A árvore é binária de pesquisa, PORÉM a chave dela não é tão útil para a realização das pesquisas.

6.9.2.2 Info

```
typedef struct descritor Info
```

Estrutura principal do programa, responsável por guardar TODOS os dados.

Estrutura principal do programa, que guarda tanto a árvore binária de pesquisa `arvore`, como a lista geral de termos consultados `termos`.

6.9.2.3 LDE

```
typedef struct lde LDE
```

Lista duplamente encadeada.

Lista duplamente encadeada, que pode ou não ser circular.

6.9.2.4 Qtdcons

```
typedef struct s_qtdCons Qtdcons
```

Estrutura utilizada para guardar uma LSE e um valor numérico arbitrário.

Estrutura que é utilizada somente na função consultasPorLocalidade() para poder armazenar quantas vezes uma `LSE*` foi chamada.