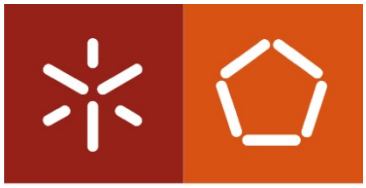


Universidade do Minho

Licenciatura em Engenharia Informática

Laboratórios de Informática III

Docentes: F. Mário Martins, João L. Sobral, João M. Fernandes

Relatório Projeto de C

Luís Miguel Carvalho Pinto

61049

Braga, Abril de 2013

Índice

1 Introdução

No âmbito da cadeira Laboratórios de Informática III, perante o problema irei desenvolver em C um programa que ajude no processamento estatístico de informação bibliográfica para artigos publicados num conjunto de revistas e/ou conferências (DBLP).

O objectivo é criar um programa que deverá processar informações armazenadas num ficheiro textual e gerar um conjunto de estatísticas.

Na primeira fase, desenvolvi um módulo reconhecedor de entradas capaz de verificar cada entrada dos ficheiros textuais e diferenciar os artigos válidos através da gramática pré definida no enunciado e assim gerar dados estatísticos básicos.

Na segunda fase, houve a necessidade de implementar estruturas de dados para guardar informações relativas aos artigos que foram considerados válidos, para posteriormente produzir um relatório estatístico mais detalhado.

2 Arquitetura dos módulos C

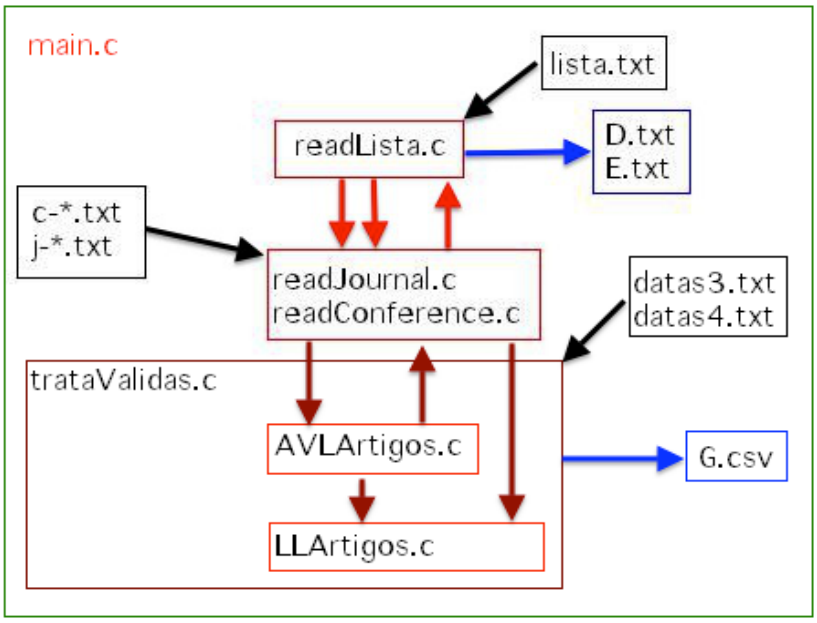


Imagem 1 – Desenho arquitetura

O meu programa é constituído por 7 módulos *.c* e os seus respectivos *.h*, excepto o *"main"*.

Breve descrição do que cada módulo faz:

- **main**

  Módulo responsável por gerir todo o programa.

  Primeiramente executa uma função do módulo *"readLista.h"*, depois de este módulo ter chegado ao fim é retornado uma árvore AVL. De seguida é chamada a função *"writeEstBasica",* por fim a função *"trataEstCompleta"* que faz o tratamento a AVL.

- **readLista**

  Módulo que controla a leitura do ficheiro *"lista.txt"* e a escrita do ficheiro de texto *"D.txt"* e a inicialização do *"E.txt"*.

  Após ler uma linha do ficheiro *"lista.txt"* testa através da função *"testaConfJour"* se é uma conferencia ou uma revista, e chama *"readConference"* ou *"readJournal",* respectivamente.

  As funções públicas deste módulo são:

```
NodoAVL readTCF(NodoAVL, FILE *, char *, int ) ;
NodoAVL readConference(char *, FILE*, NodoAVL) ;
NodoAVL readJournal(char *, FILE *, NodoAVL) ;
void readLista(NodoAVL *, FILE *) ;
void writeEstBasica() ;
FILE* inic_writeRejeitados(FILE *) ;
void close_writeRejeitados(FILE *) ;
```

- **readJournal e readConference**

Módulos que fazem a leitura de ficheiros "*c-*.txt"* e *"j-*.txt"* e o *parsing* de linhas de artigos através das funções "*treatsJournal"* e *"treatsConference"* que verifica se um artigo é válido e incrementa contadores (aceite/rejeitado) para a estatística e escreve depois de ler todas as linhas de um determinado ficheiro *"c-*.txt"* ou *"j-*.txt"* o número de artigos rejeitados no ficheiro "*E.txt".*

Caso um artigo seja considerado válido vai ser inserido na estrutura de dados uma data e o número de autores desse respectivo artigo.

As funções públicas a outros módulos em "*readJournal.h"* são:

```
int verificaInt(char *) ;
int verificaAuthors(char *,int, int *, LAutor *) ;
int verificaTitle(char *,int) ;
void writeRejeitados(FILE *, char *, int ) ;
```

As funções públicas a outros módulos em "*readConference.h"* são:

```
void writeRejeitados(FILE *, char *, int ) ;
```

De notar que este módulo usa as funções públicas de "*readJournal.h"*. No entanto as três primeiras funções públicas de "*readJournal.h"* deveriam ter sido criadas num módulo a parte destes dois e assim o encapsulamento seria muito melhor.

- **AVLArtigos e LLArtigos**

Módulos que possuem as funções de inserir dados ordenados na AVL e na Lista Ligada, *"inserirArtigoAVL"* e *"insereArtigoLL",* respectivamente.

API do módulo *"AVLArtigos.h"*:

```
NodoAVL inserirArtigoAVL(NodoAVL, int, int, int*) ;
NodoAVL balanceEsq (NodoAVL, int* ) ;
NodoAVL balanceDir (NodoAVL, int*) ;
NodoAVL roda_esq (NodoAVL ) ;
NodoAVL roda_dir (NodoAVL ) ;
```

API do módulo "LLArtigos.h":

```
LAutor insereAutor(LAutor, char *) ;
LArtigo criaArtigoLL(LArtigo, int) ;
LArtigo insereArtigoLL(LArtigo, int) ;
LFileg insereAutArt(LFileg, int, int) ;
```

- **trataValidas**

Módulo responsável pelo tratamento aos artigos que foram considerados válidos, leitura dos ficheiros com datas, *"datas3.txt"* e *"datas4.txt", e* produzir o ficheiro com a estatística completa *"G.csv".* Neste módulo tive de recorrer a uma estrutura auxiliar (LFileg) para guardar o número total de artigos por número de autor.

API do módulo "*trataValidas.h":*

```
NodoAVL trataEstCompleta(NodoAVL ) ;
```

3 Grafo de Dependências

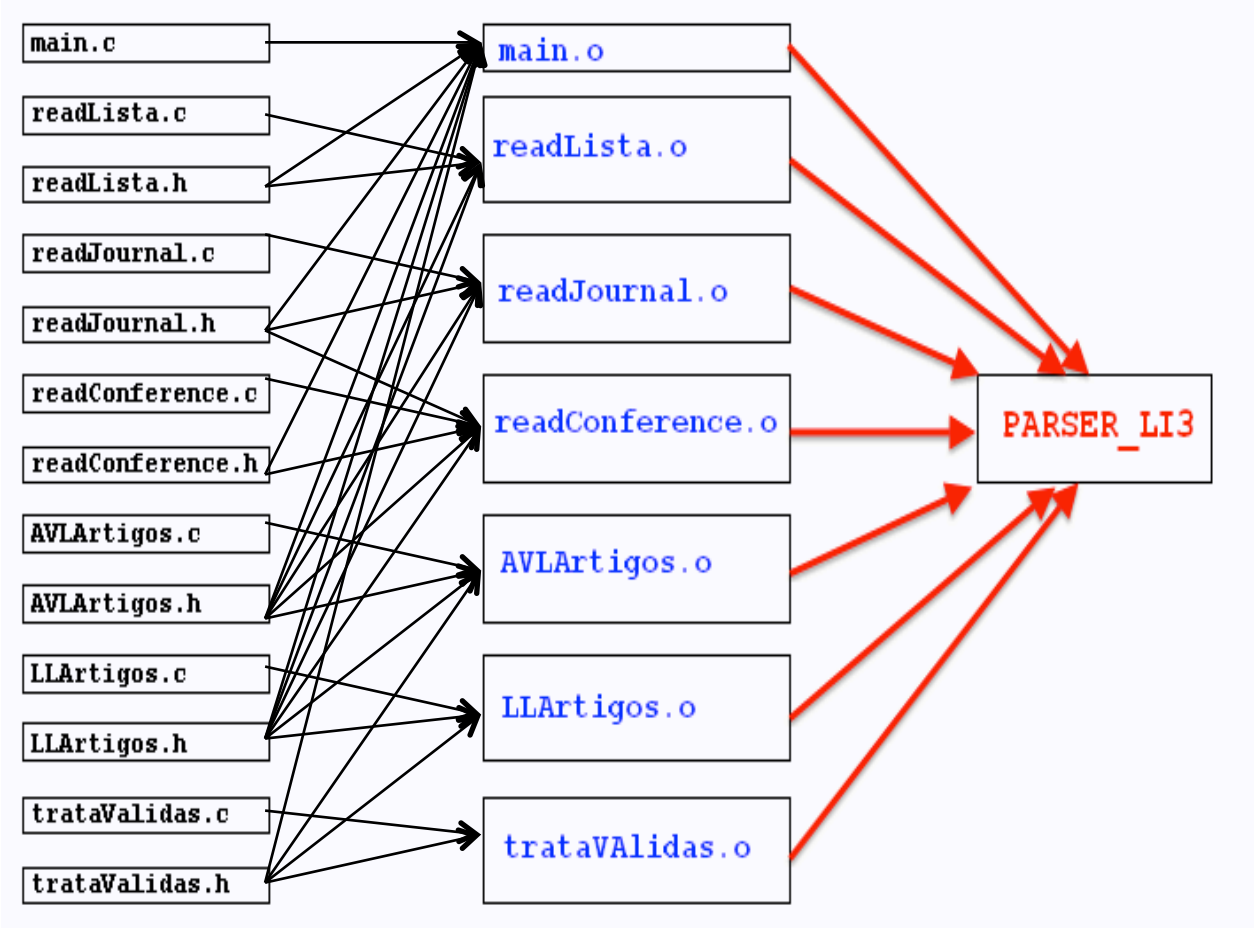


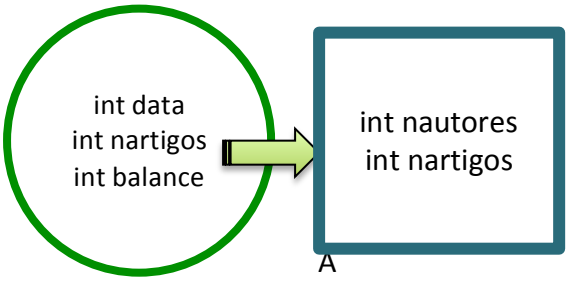
Imagem 2 - Grafo

Obtém-se assim o grafo de dependências do executável "*PARSER_LI3*", que a cada **.c* e **.h* nos quais existem dependências, na compilação dá origem a outros ficheiros objecto **.o* que se juntam e criam assim o programa executável.

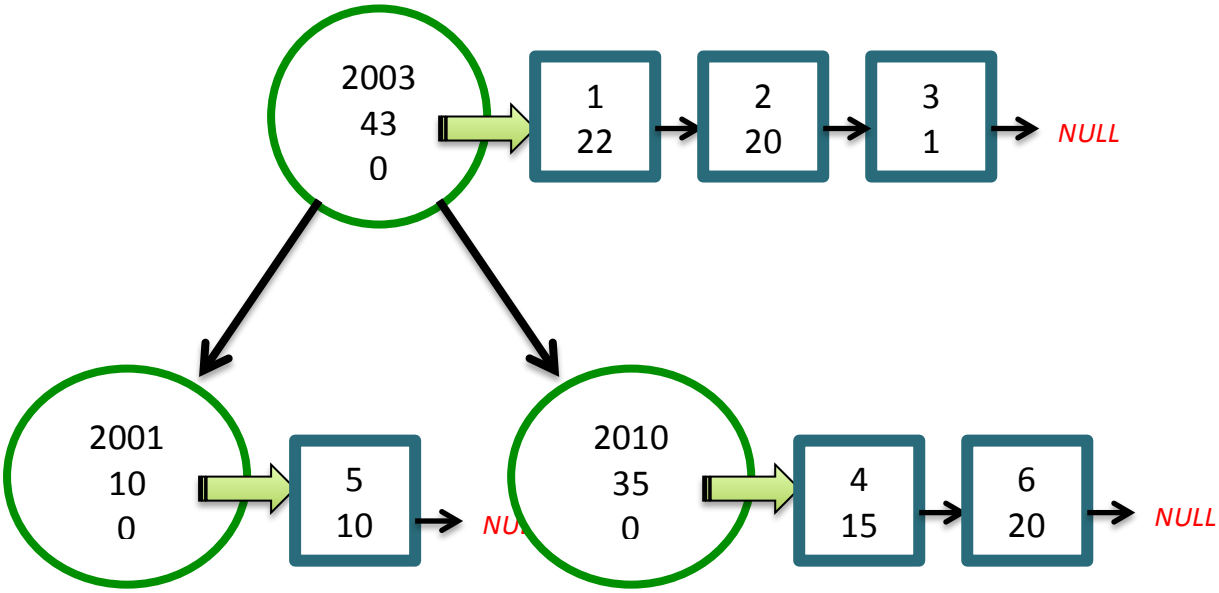
4 Estrutura de Dados

Para o desenvolvimento da fase 2 do projeto precisei de recorrer a estruturas de dados onde armazenei informação relativas aos artigos que foram dados como válidos pelo módulo reconhecedor de entradas, assim construí dois tipos de estruturas, AVL com lista ligada ordenada por "*nautores*", isto porque me pareceu a melhor forma para pesquisar por anos, ser mais eficiente, e dentro de cada nodo tem uma lista ligada que está ordenada para ser mais fácil percorrer por número de autores.

Exemplo da Estrutura:



Exemplo funcionamento da estrutura:



4.1 AVL

```
typedef struct sNodoAVL {
    int data;
    int nartigos;
    LArtigo la;
    int balance;
    struct sNodoAVL *esq, *dir;
} *NodoAVL, NNodoAVL;
```

Imagem 3 - Estrutura AVL

Módulo representativo de um nodo de uma AVL que está ordenado por uma dada data. De notar que cada nodo da árvore tem a capacidade de dizer quantos artigos existem naquele ano (*"int*

nartigos”), contém ainda uma Lista Ligada *(“LArtigo la”)* de artigos por número de autor, ordenada, e ainda um inteiro que diz se aquele nodo se encontra balanceado.

Sempre que é inserido um novo artigo é necessário verificar onde se vai introduzir aquele nodo, caso um determinado ano já existe na AVL é incrementado a variável *“nartigos”* e é inserido mais um artigo na Lista Ligada *“LArtigo”*, caso ainda não exista é então criado um novo nodo onde coloco a “*nartigos”* igual a 1 e crio a Lista Ligada para aquele nodo, nesse momento é necessário saber se este nodo se encontra no sítio certo da AVL, através das funções de balanceamento.

4.2 Lista Ligada

```
typedef struct sLArtigo {
    int nart;
    int nautores;
    struct sLArtigo *next;
} *LArtigo, NLArtigo;
```

Imagem 4

Módulo representativo de um nodo da lista ligada, que está ordenada por número de autores (“*nautores”)* e que a cada autor está associado quantos artigos existem com aquele número.

```
typedef struct sLFileg {
    int nautor;
    int nartigo;
    struct sLFileg *next;
} *LFileg, NLFileg;
```

Imagem 5

Estrutura auxiliar para ajudar a guardar a informação do número total de artigos por número de autores, que será usada para criar a segunda parte do ficheiro “*G.csv*”.

```
typedef struct sNomeAutor {
    char *nome;
}   NomeAutor;

typedef struct sLAutor {
    NomeAutor autor;
    struct sLAutor *next;
}   *LAutor, NLAutor;
```

Imagem 6

Estrutura usada para armazenar os nomes dos autores de cada artigo, está é usada no *parsing* da função que verifica se autores existem.

Pretende-se que seja usada na fase2b a quando da criação da *hash table* sempre que um artigo é válido.

5 Otimização

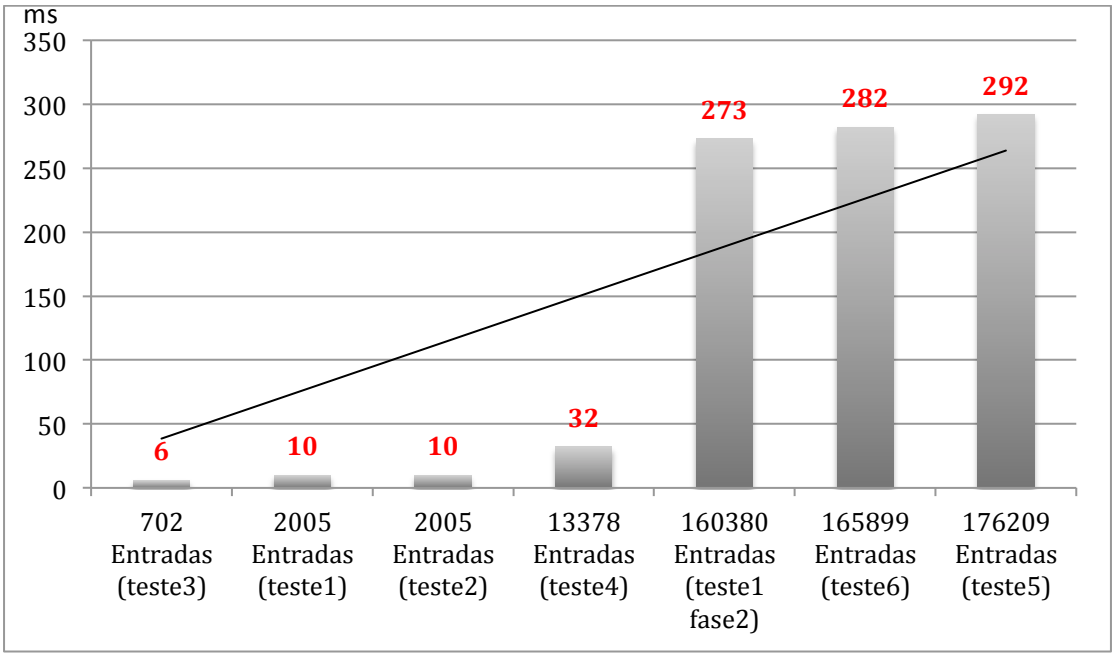
5.1 Tempo de Execução

Após realizar a seguinte instrução na consola *"$ time ./PARSER_LI3"* obtém-se três valores relacionados com a execução do programa.

```
real    0m0.014s
user    0m0.007s
sys     0m0.004s
```

No entanto, para se saber o tempo despendido a executar a aplicação temos de somar os valores de *"user"* e *"sys"*, pois o valor resultante é o tempo dedicado a este processo.

O gráfico que se segue mostra a variação do tempo resultante, em microssegundos (ms), pelo número de entradas. Cada teste foi executado três vezes, resultando no gráfico o tempo médio destas medições.



Como podemos constatar, com o aumento do número de entradas aumenta o tempo que é dedicado ao executar a aplicação.

5.2 Pesquisas por número de artigos

A minha estrutura de dados está de certa forma eficiente para se fazerem pesquisas por número de autores e assim se obter para um dado número de autor e para cada ano o número de artigos existentes.

6 Conclusão

Depois de ter concluído o projeto fico com a sensação que pus realmente em prática vários dos conhecimentos que aprendi noutras unidades curriculares.

Ao escolher trabalhar com AVL e listas ligadas como estruturas de dados em termos de execução, não é muito significativo. Através dos exemplos fornecidos pelos docentes nota-se que a diferença de datas dos artigos é de cerca 45 anos, logo a AVL no máximo teria 45 nodos, a mesma situação na lista ligada que pelos exemplos terá no máximo 44 autores. O que acontece normalmente tanto na AVL como LL é incrementar o número de artigos, assim, respectivamente.

O programa apresenta tempos quase imediatos em toda a sua estrutura, mas é natural que com o aumento do número de entradas o tempo de resposta seja maior.

Para criar o grafo de autores era necessário recorrer a *hash table* para ter melhores tempos de implementação.

Em geral, com as estruturas, modularidade e funções implementadas consegui ter um bom funcionamento do programa.