Algoritmos e Estruturas de Dados

Segundo Trabalho Prático

20 de Dezembro de 2019

Trabalho Realizado por:

* Daniel Gomes, Nmec 93015, Contributo:50%
* Rui Fernandes, Nmec 92952, Contributo: 50%

# Índice

[**Índice**](#_41czvovpmd74) **2**

[**Introdução**](#_xvp096t7zu4d) **3**

[Implementação do Problema](#_9t6rnm1ms9x2) **3**

[Porquê Hash Tables?](#_9tbboru36sdv) 3

[**Resultados obtidos**](#_qwz6bgbt5ko3) **4**

[TP2\_LL.c](#_v3plp2544wo6) 4

[TP2\_BT.c](#_qs4i49n5dvd3) 7

[**Testes Efetuados**](#_px6clypxe135) **9**

[**Conclusão**](#_u71bh2aab3ut) **10**

[**Código Utilizado**](#_mvoqcgtrjffs) **11**

[TP2\_LL.c](#_dh6xgw7z1ihx) 11

[TP2\_BT.c](#_5ps7277rzfv4) 16

# Introdução

Como objetivo deste segundo trabalho prático pretendia-se criar um programa que fosse capaz de contar o número de ocorrências de cada palavra diferente de um ficheiro de texto. Além disso, teria de apresentar as seguintes funcionalidades:

* Guardar a localização da primeira e última ocorrência de cada palavra distinta no ficheiro de texto;
* Guardar a menor, maior e média distância entre ocorrências consecutivas da mesma palavra.

Para armazenar a informação necessária, foi pedido que se utilizasse, como estrutura de dados, uma hash table implementada com o método de *separate chaining*, ou seja, cada entrada desta tabela deveria apontar para uma lista ligada( *linked list)* ou então para uma árvore binária ordenada,onde cada posição da respectiva sub estrutura iria conter toda a informação para uma dada palavra.

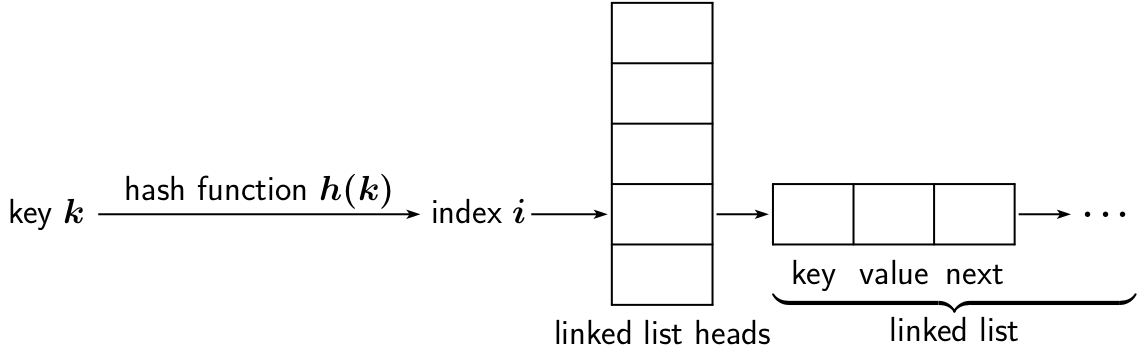
# Implementação do Problema

Por opção, decidiu-se, além da implementação de *Separate Chaining* com Listas Ligadas, uma implementação com Árvores Binárias. A primeira, encontra-se no programa **TP2\_LL.c** enquanto que a segunda no programa **TP2\_BT.c**. Pretendemos, com estas duas implementações, encontrar as respetivas vantagens e desvantagens de cada uma. Além disto, em ambos programas, era pretendido que a Hash Table fosse redimensionada de forma dinâmica.

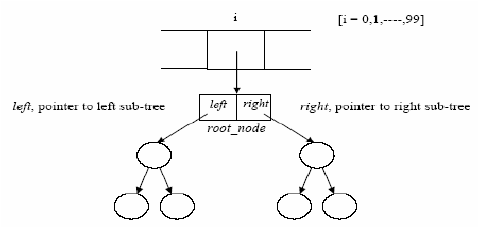
## Porquê Hash Tables?

Hash Tables, ou Tabelas de Dispersão, são uma estrutura de dados que associa chaves de pesquisa a valores. Estas chaves são um array de caracteres no nosso caso (a palavra a guardar) e ao serem passadas por uma Função de Dispersão é gerado o índice correspondente no array representativo da tabela, onde se vai guardar a informação pretendida. Assim através da hashtable e da função de dispersão, a pesquisa de informação torna-se muito mais rápida. Contudo, como cada índice gerado pela função de dispersão usada pode ser o mesmo para algumas palavras distintas (devido ao código ASCII associado a cada caractere) dão-se as chamadas colisões, ou seja neste caso, duas keys que apontam para a mesma posição da tabela. Assim, o uso de *Separate Chaining*, torna-se fundamental para que possamos lidar com estes casos, podendo ter várias chaves a apontar para o mesmo *index* sem perder informação.

Nas imagens,que se seguem encontram-se esquemas visuais de como se encontram os dados das formas que até agora temos abordado



Separate Chaining com Linked Lists



Separate Chaining com Binary Trees

# 

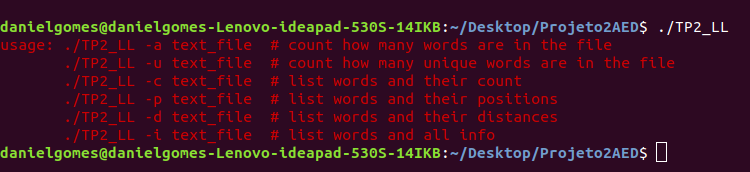
# Resultados obtidos

## TP2\_LL.c

Para podermos concluir se o nosso programa estaria de acordo com o expectável, decidiu-se criar um “menu”, idêntico ao que foi utilizado na Aula Prática 08 no programa **countwords.c**, onde o utilizador pode escolher se pretende:

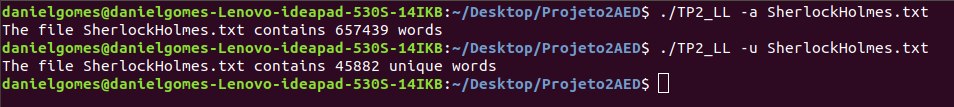
* visualizar o número de total de palavras;
* visualizar o número de palavras distintas;
* listar palavras distintas e o número respetivo de ocorrências;
* listar palavras distintas e as respetivas posições;
* listar palavras distintas e as respetivas distâncias;
* listar palavras distintas e informação completa acerca de cada uma.

Com vista à utilização correta deste programa, sempre que é executado introduzindo uma opção ou um ficheiro de texto inválidos, ou sem se introduzir algum dos dois, é impressa a seguinte mensagem.



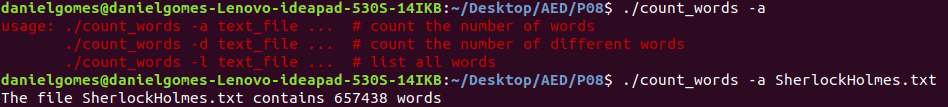
Menu Utilizado em **TP2\_LL.c**

Quanto às primeiras duas opções obtiveram-se os seguintes resultados,onde iremos comparar com o programa desenvolvido pelo Professor na Aula Prática já referida.



Resultados Obtidos em **TP2\_LL.c**

Assim, utilizando o mesmo ficheiro de texto conseguimos concluir que as duas primeiras opções têm um bom funcionamento, visto que os resultados obtidos são praticamente iguais aos do programa desenvolvido pelo professor (Aula P08):

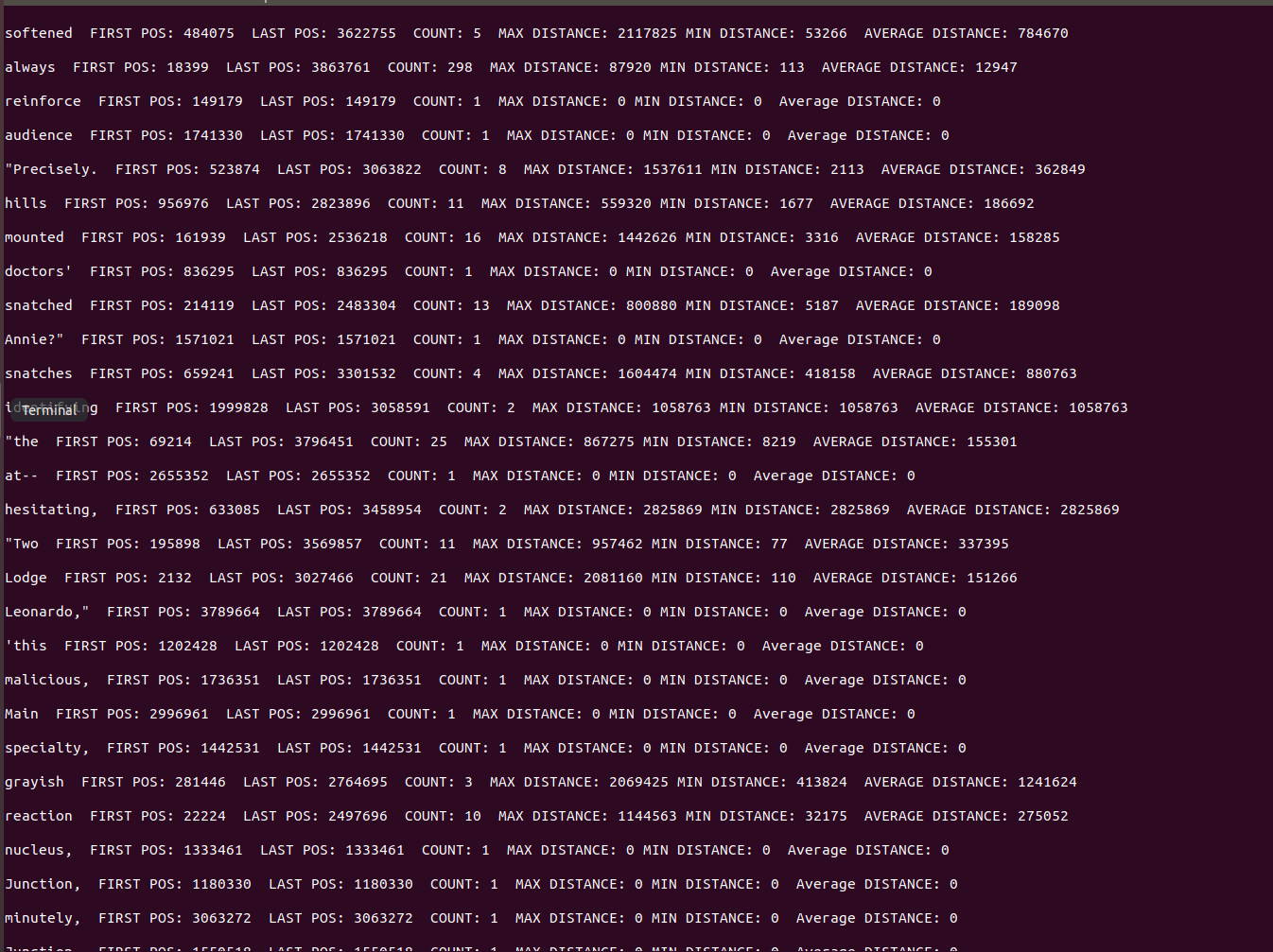


Resultados obtidos na Aula Prática

Ocorreu de facto uma diferença no número total de palavras de 1, o que se torna relativamente insignificante, e é provavelmente de fácil correção, apesar de não termos encontrado o motivo da diferença.

Passando para a opção mais importante, ***-i***, devido ao facto de conter toda a informação sobre as palavras do ficheiro de texto, tal como requerido no enunciado do projeto.

Foi utilizado o ficheiro de texto ***SherlockHolmes.txt*** ( ficheiro várias vezes utilizado para realização de testes ao longo trabalho), e obteve-se os seguintes resultados:



Excerto do resultado obtido da opção -i do programa **TP2\_LL.c**

Neste excerto de informação de algumas palavras do ficheiro mostra-se em cada linha a informação associada a uma dada palavra, tal que:

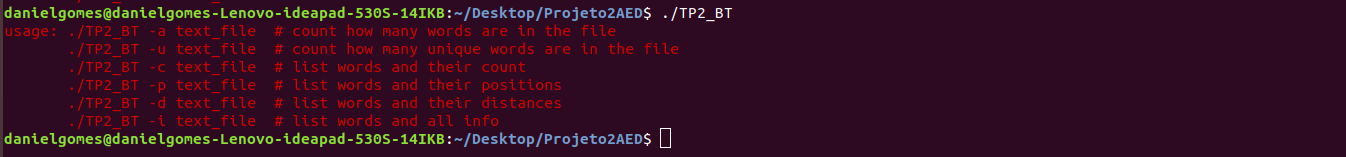
* Primeira coluna corresponde à palavra;
* Segunda e terceira coluna correspondem à primeira e última posição onde ocorreu a palavra no ficheiro (em número caracteres face ao início do ficheiro);
* Quarta coluna contém o número de aparições da palavra;
* As restantes colunas dizem respeito à máxima,mínima e média distâncias (em número caracteres).

Todas as palavras cujo número de ocorrências não fosse superior a uma, apresentaram distâncias máximas mínimas e médias de 0, enquanto que a distância média mostrou ser sempre um valor entre a distância mínima e máxima pelo que podemos aceitar os valores obtidos.

## TP2\_BT.c

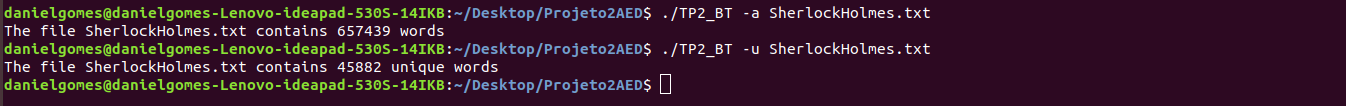
A estrutura deste programa é a mesma do que foi referido no ponto anterior,ou seja, apresenta-se, da mesma forma, um menu com as mesmas 5 opções:

* visualizar o número de total de palavras;
* visualizar o número de palavras distintas;
* listar palavras e o número respetivo de ocorrências;
* listar palavras distintas e as respetivas posições;
* listar palavras distintas e as respetivas distâncias;
* listar palavras distintas e informação completa acerca de cada uma.



Menu de Utilização do programa **TP2\_BT.c**

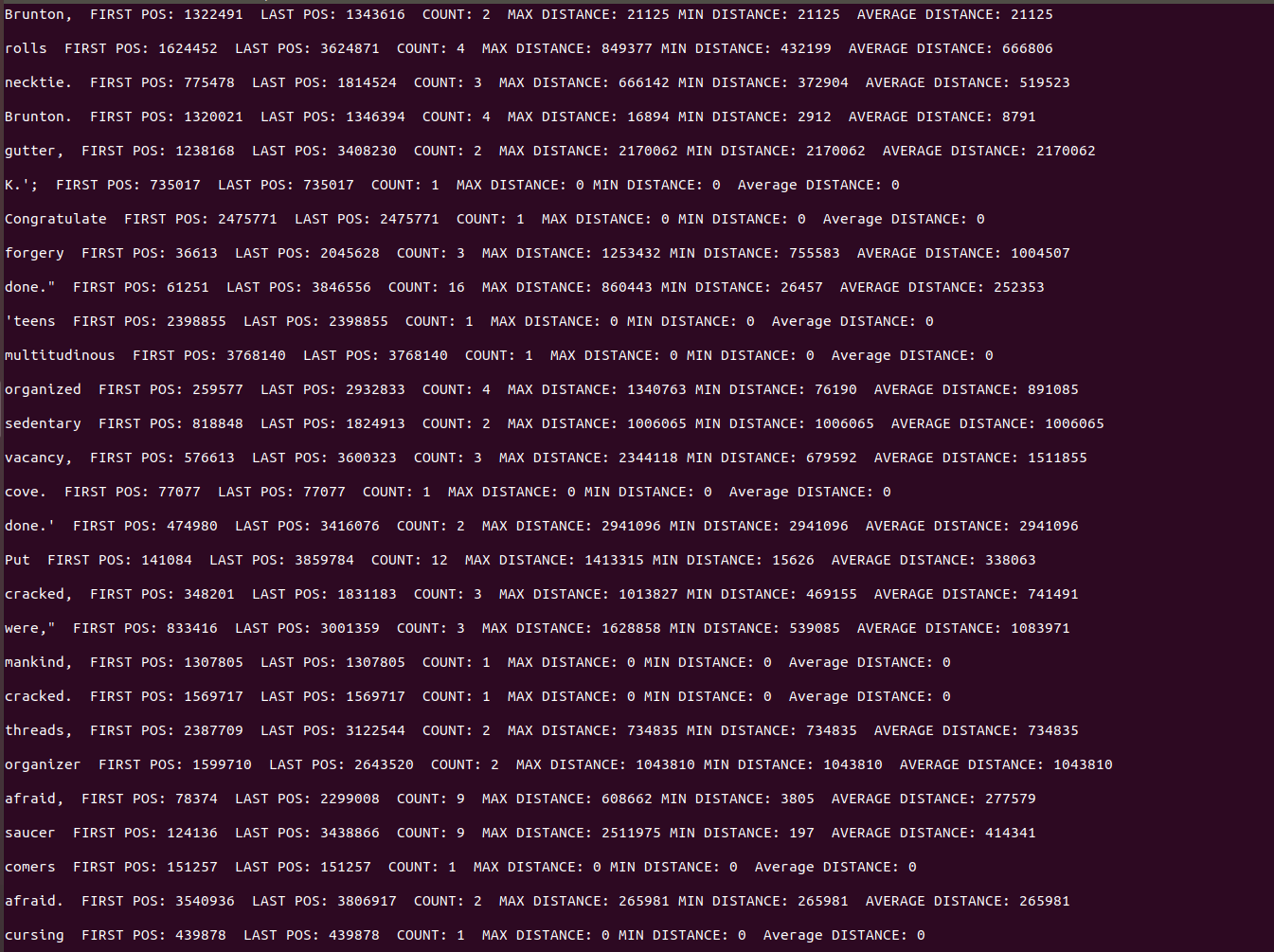
Comparando da mesma forma as primeira duas opções da implementação desenvolvida- contagem de palavras totais do ficheiro e a contagem de palavras únicas deste- às duas primeiras do ficheiro **countwords.c** já desenvolvido pelo Professor na Aula Prática, obtiveram-se os seguintes resultados:



Opção -a e -u do programa **TP2\_BT.c**

Tendo em conta que os valores para estas opções por parte do ficheiro **countwords.c** já foram enunciados, podemos concluir que mais uma vez o número de palavras analisadas foi um sucesso pois detetou-se 45882 palavras diferentes, ou seja, o mesmo valor, e 657439 palavras totais no ficheiro (diferença ligeira de uma palavra para o que foi determinado em **countwords.c**).

Da mesma forma que foi efetuado para TP2\_LL.c, iremos agora apresentar um excerto da execução, (visto que é extremamente extensa para ser apresentada por completo) da mesma opção, -i, para este programa,TP2\_BT.c:



Excerto da execução da opção -i do programa TP2\_BT.c

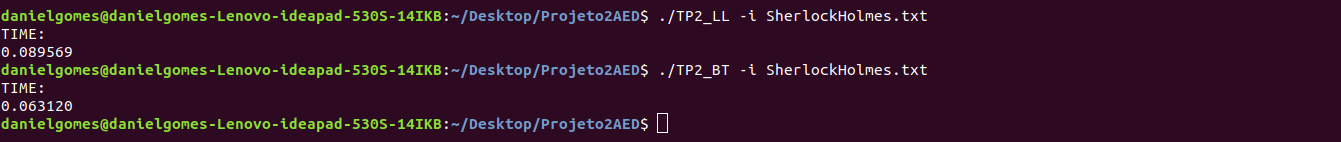
Tal como anteriormente, o resultado apresentado contém a mesma estrutura, ou seja, em cada linha do terminal está presente uma palavra diferente, seguida da primeira e última posição onde ocorreram no ficheiro de texto, assim como o número de aparições e distâncias máximas,mínimas e médias.

Também importante referir que, mais uma vez, as distâncias aparentam estar bem determinadas tendo em conta que se a palavra não passar de uma aparição singular, não apresentará qualquer valor paras as distâncias, mantendo-se em zero, além de que, em caso contrário, o valor médio de distância nunca encontra-se superior ou inferior à distâncias máxima e mínima, respetivamente.

# Testes Efetuados

Como forma de teste de eficiência para ambos os programas decidiu-se determinar o tempo de execução entre ambos ficheiros e em diversas condições (o tamanho inicial da tabela é de 8192 para todos estes testes). Assim, inicialmente comparou-se os programas nas condições já pré definidas, onde se obteve os seguintes tempos de execução:

* **TP2\_LL.c** demorou 0.089 segundos, utilizando a função resize() sempre que **80 por cento** da tabela estivesse ocupada;
* **TP2\_BT.c** demorou 0.063 segundos,utilizando a função de resize() sempre que **80 por cento** da tabela estivesse ocupada.

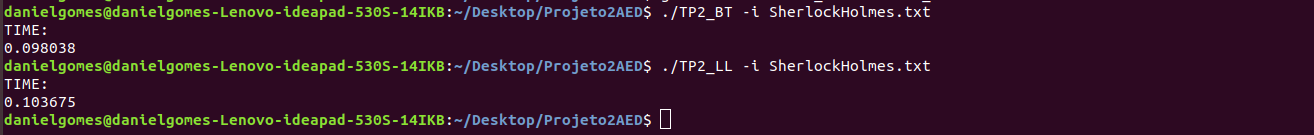


Tempo efetuado em cada Programa

Assim, revela-se que a execução utilizando separate chaining com árvores binárias torna-se ligeiramente melhor em termos de eficiência do que utilizando listas ligadas.

Alterando em cada um destes programas, a percentagem de preenchimento de tabela necessária para que fosse chamada a função de redimensionamento para **50 por cento** conseguiu-se obter os seguintes resultados:

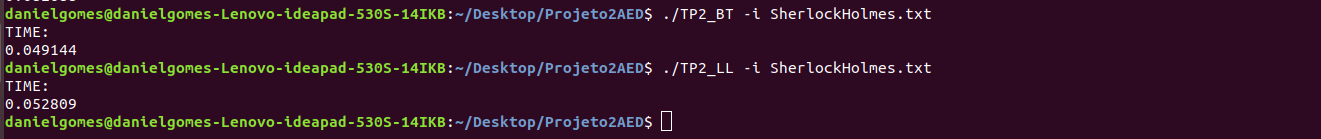
* **TP2\_LL.c** demorou 0.103 segundos
* **TP2\_BT.c** demorou 0.098 segundos.



Os valores são expectáveis tendo em conta que, ao diminuir a percentagem referida anteriormente, a função resize() é chamada mais vezes, e, por sua vez, aumenta o tempo necessário para correr o código.

Seguidamente, efetuou-se a realização de testes às duas implementações mas **sem efetuar qualquer redimensionamento à tabela de dispersão**, pelo que o *runtime* de cada uma foi o seguinte:

* **TP2\_LL.c** demorou 0.052 segundos
* **TP2\_BT.c** demorou 0.049 segundos.



Concluindo, sem ter de efetuar redimensionamento, que envolve copiar toda a informação de uma tabela para uma nova com maior tamanho, melhora-se o *runtime*. Contudo, não será de estranhar algumas vezes demorar um tempo semelhante àquele que apresentavam ambos programas nas condições iniciais visto que a tabela pode gerar mais colisões, o que por consequência envolve ter de percorrer a lista ligada, à qual aponta cada índice da tabela, até ao fim, para introduzir novas palavras.

# 

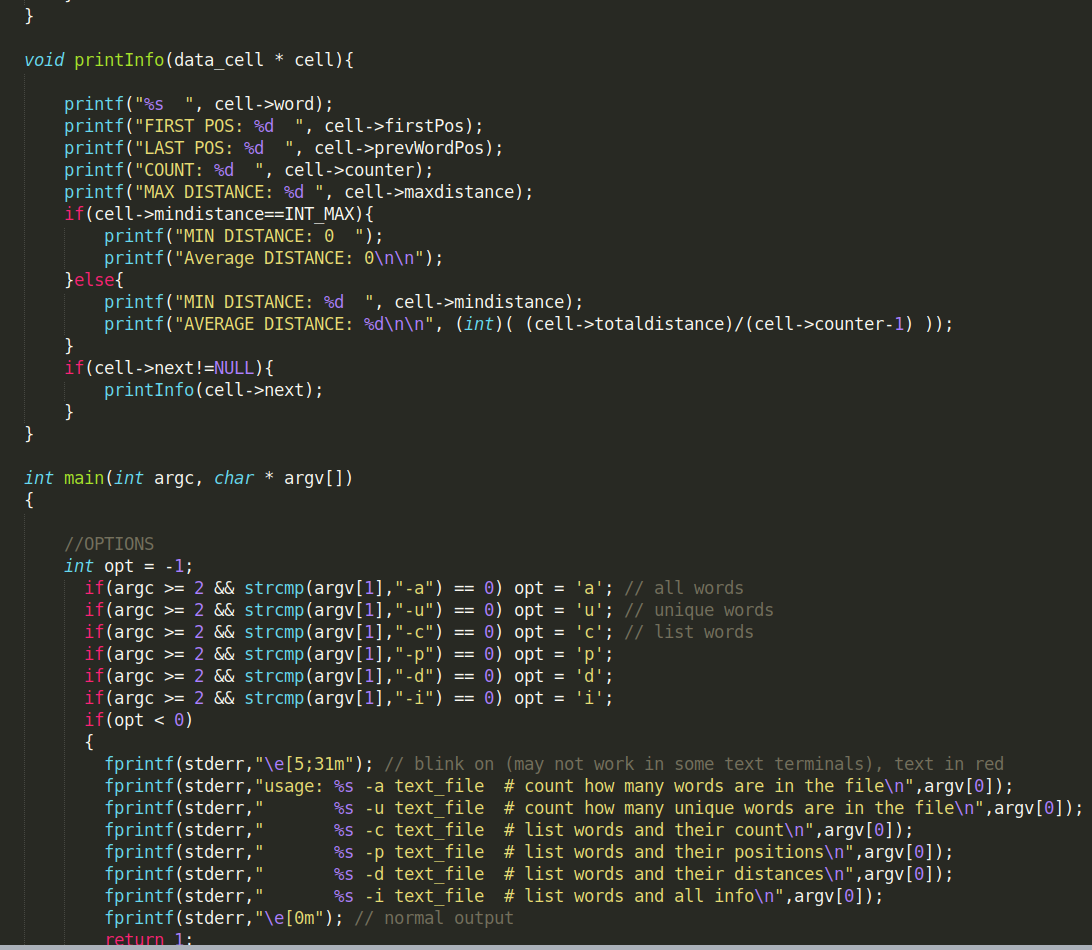
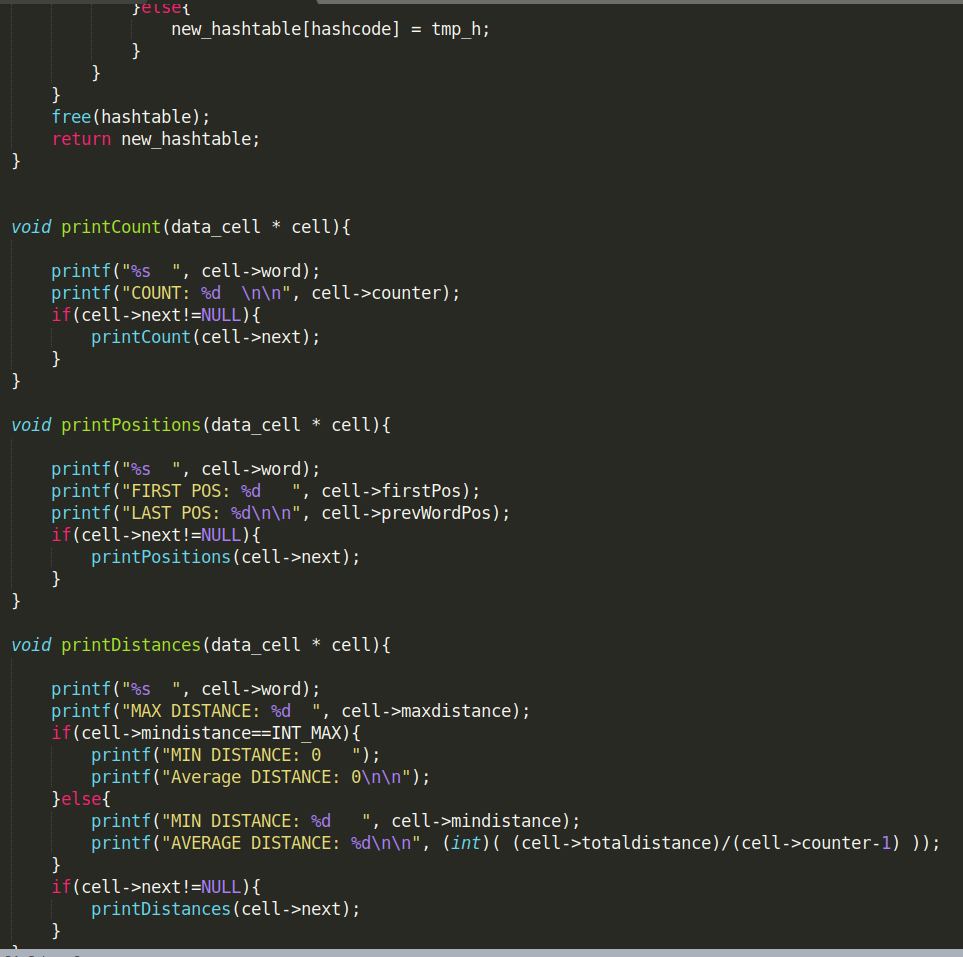
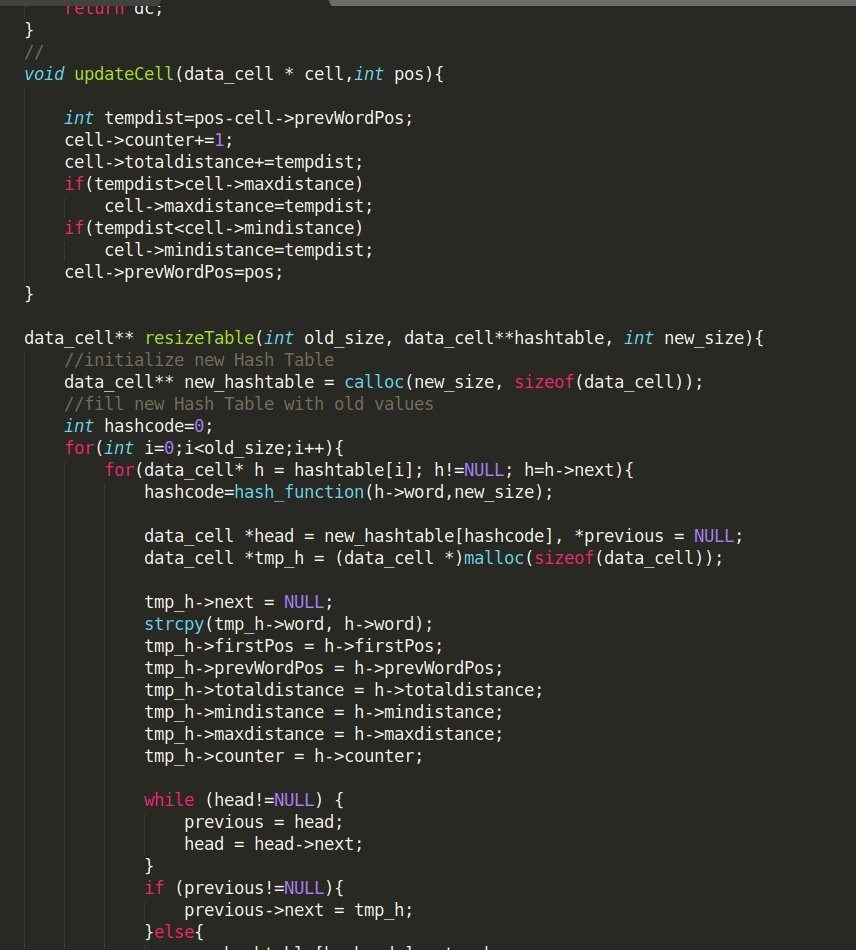
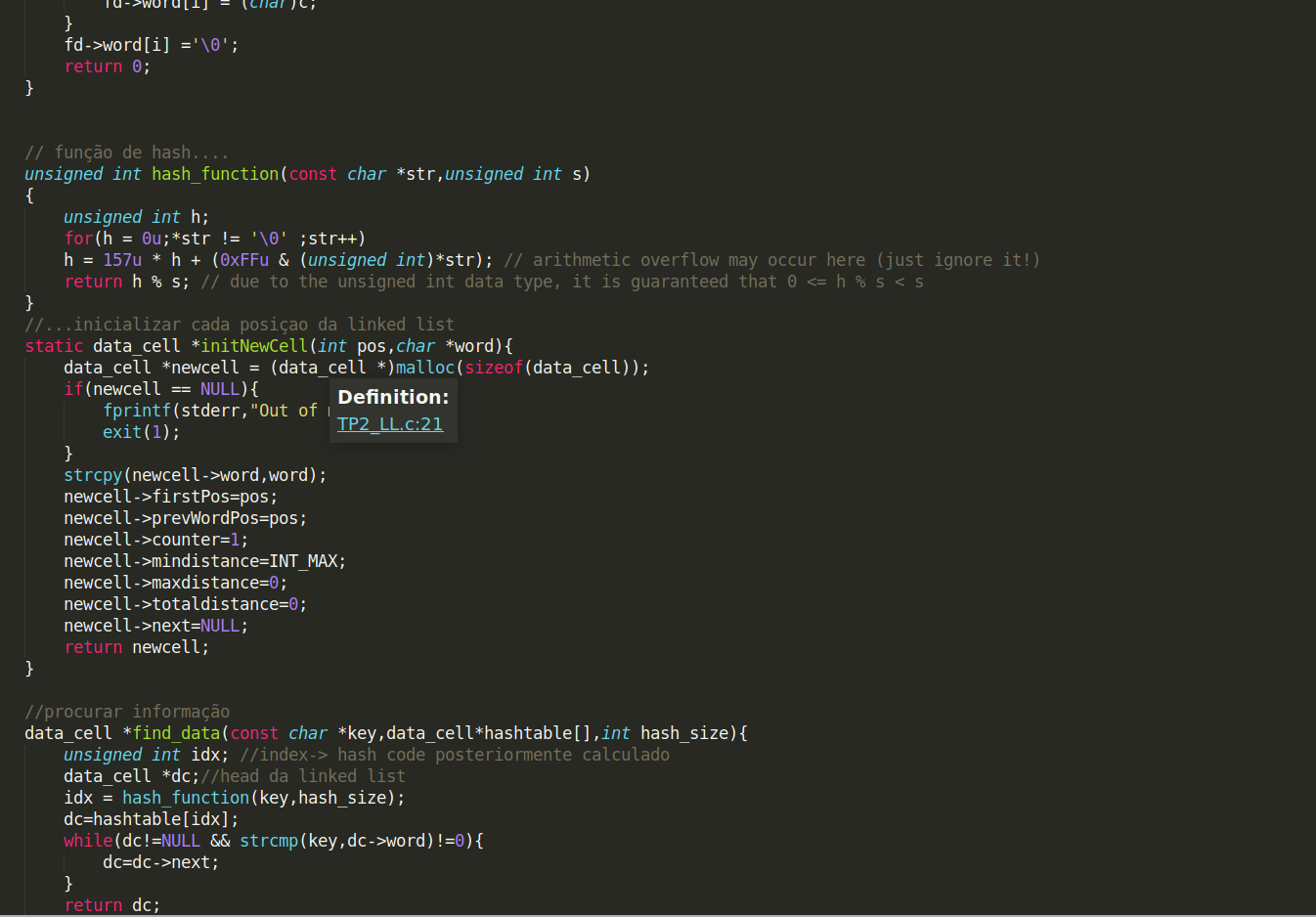
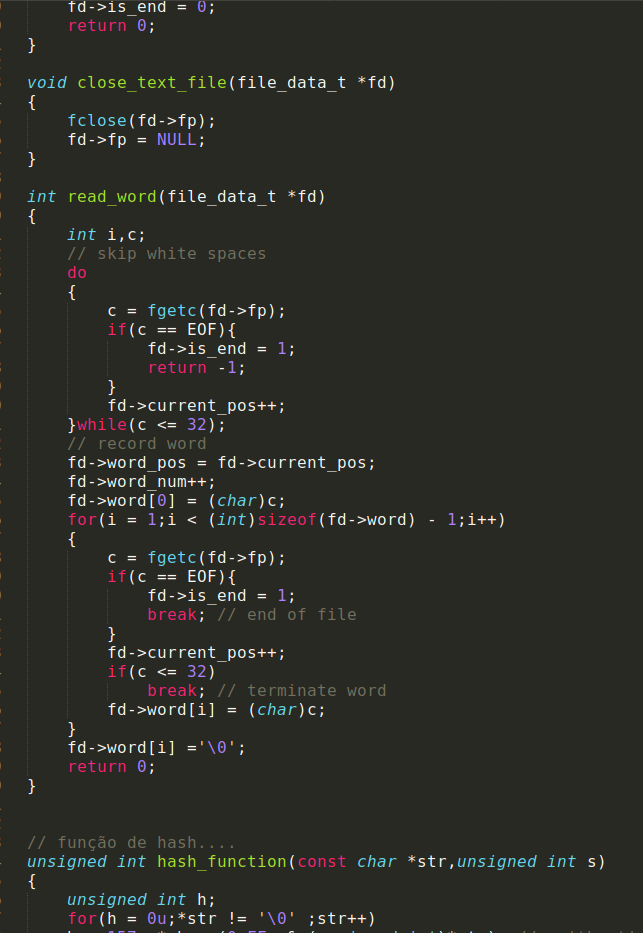
# Conclusão

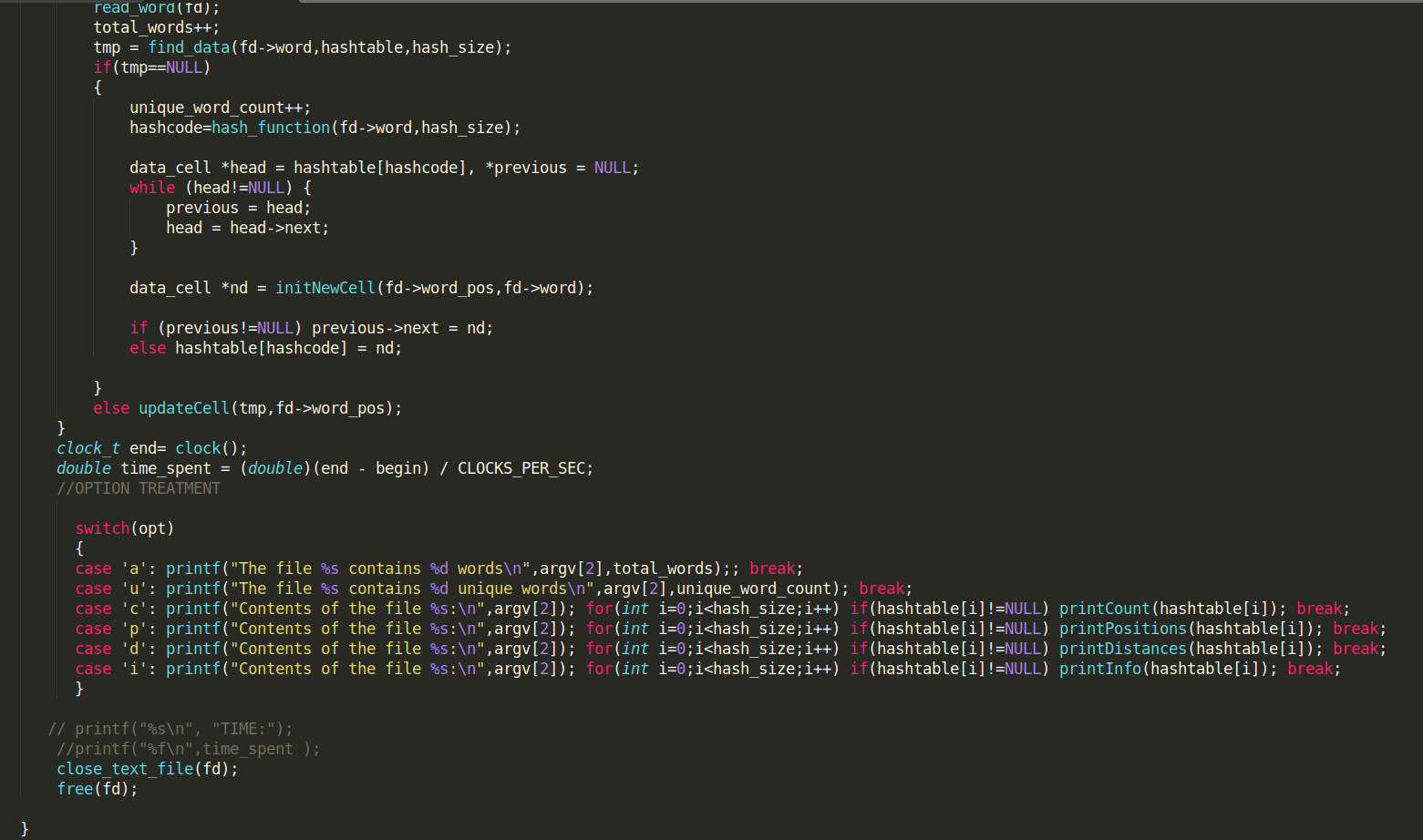
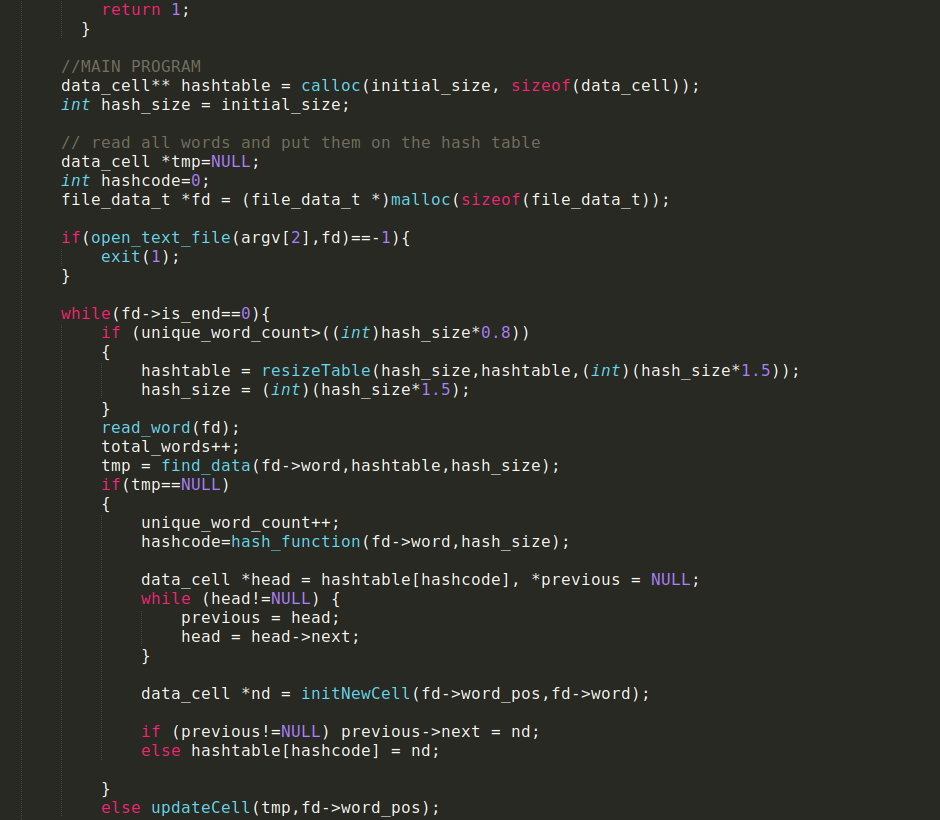
Este trabalho prático permitiu consolidar conhecimentos acerca de vários assuntos lecionados ao longo da disciplina de Algoritmos e Estruturas de Dados, nomeadamente *Hash Tables, Linked Lists e Ordered Binary Trees*. Além disso, este trabalho permitiu nos utilizar vários conceitos que estão por base na Linguagem de Programação usada ,***C*** , como alocação de memória. Aliado a tudo isto, concluiu-se que a utilização por listas ligadas torna-se mais lenta, mesmo que por pouco, quando comparada com árvores binárias além de que a utilização de Separate Chaining permite de forma relativamente fácil estruturar os nossos dados,evitando colisões.

Por fim, como base deste trabalho, é importante referir que de uma maneira eficiente e de fácil implementação, aprendemos uma nova forma de obter informação variada de um ficheiro de texto.

# Código Utilizado

# TP2\_LL.c





## TP2\_BT.c

## 

