

Universidade do Minho  
Licenciatura em Engenharia Informática  
Laboratórios de Informática III



## **Etapas II**

Ano Lectivo de 2009/2010

GRUPO 9  
54738 **João Gomes**  
54745 **André Pimenta**  
54825 **Daniel Santos**

25 de Abril de 2010

## **Resumo**

Neste relatório será apresentada a construção e funcionamento do projecto “SempreLigados” do âmbito da cadeira de Laboratórios de Informática III. Serão justificadas todas as opções e escolhas das varias implementações na construção do projecto e o seu funcionamento das mesmas. Será apresentada de forma global o funcionamento do projecto desenvolvido, as dificuldades no seu desenvolvimento e a motivação na construção deste.

# Conteúdo

<b>Conteúdo</b>	<b>1</b>
<b>1 Introdução</b>	<b>2</b>
1.1 Contextualização e apresentação do Caso de Estudo . . . . .	2
1.2 Motivação e objectivos . . . . .	2
1.3 Estrutura . . . . .	2
<b>2 Desenvolvimento</b>	<b>3</b>
2.1 Análise do Problema . . . . .	3
2.2 Solução do Problema . . . . .	4
2.2.1 Estruturas de dados . . . . .	4
2.2.2 Utilizadores . . . . .	5
2.2.3 Relações . . . . .	7
2.2.4 Mensagens . . . . .	8
2.3 Interface e Menus . . . . .	9
2.3.1 Menus . . . . .	10
2.4 Erros . . . . .	10
2.5 Funcionamento geral . . . . .	11
2.6 Complexidade e Desempenho . . . . .	11
2.6.1 Complexidade . . . . .	11
2.6.2 Desempenho . . . . .	12
2.7 Ferramentas utilizadas . . . . .	13
<b>3 Conclusão</b>	<b>14</b>
<b>4 Fotos</b>	<b>15</b>

# Capítulo 1

## Introdução

### 1.1 Contextualização e apresentação do Caso de Estudo

O presente relatório tem como objectivo explicar os passos dados pelo grupo na realização do projecto da Unidade Curricular de Laboratórios de Informática III, que tem como objectivo a construção de uma rede social “SempreLigados” usando a linguagem C.

Para a construção deste usamos varias ferramentas disponibilizadas pelos sistemas operativos Unix, como o GCC Debugger Valgrind que se revelaram muitos úteis na resolução de problemas que foram surgindo ao longo da construção do projecto.

Para a realização do projecto fui também muito útil os conhecimentos adquiridos nas disciplinas de Sistemas Operativos, Algoritmos e complexidade, Programação Imperativa e Laboratórios de Informática II.

### 1.2 Motivação e objectivos

Este projecto foi muito motivante para o nosso grupo, visto tratar se de uma rede social que é algo que está na moda nos dias de hoje.

Porem este não foi o factor que mais nos motivou, mas sim a utilização dos conhecimentos adquiridos em varias unidades curriculares .Onde podemos utiliza-los de forma pratica e perceber melhor a sua utilidade e funcionalidade.

### 1.3 Estrutura

O presente relatório é constituído por três capítulos: **Introdução**; **Desenvolvimento**, divido em duas secções, **Análise do Problema**, **Solução do Problema** e **Ferramentas utilizadas**, onde explicamos mais detalhadamente os passos dados pelo grupo durante a realização da etapa, assim como as dificuldades que foram surgindo e a forma em como as ultrapassámos; e, por fim, a **Conclusão**.

## Capítulo 2

# Desenvolvimento

### 2.1 Análise do Problema

No desenvolvimento do sempreLigados o principal problema colocado foi a organização e estruturação de dados e o seu armazenamento, de forma a que o programa não fica se lento na execução nem ao carregar/guardar os dados dos utilizadores da mesma, ou seja construir algo que fosse eficaz e organizado em todas as suas operações.

A construção de um projecto de forma modular também foi algo a ter em consideração, sendo muito importante para posteriores alterações de código mas também para a construção do mesmo. Esta torna se bastante importante especialmente quando se criar projectos de maior dimensão.

## 2.2 Solução do Problema

### 2.2.1 Estruturas de dados

Para o desenvolvimento do projecto decidimos usar dois tipos de dados tabelas de hash e grafos. Optamos então por utilizar tabelas de hash para o armazenamento de dados dos utilizadores. Criamos uma tabela para o nome e outra para o nif, onde ambas irão partilhar as informações sobre os utilizadores sem estar a repetias, mas organizando-as de forma diferente para que seja possível fazer pesquisas pelo nome e pelo nif de forma eficaz.

O principal factor que nos levou a escolher esta estrutura de dados para as informações dos utilizadores, foi o facto de conseguirmos obter com rapidez e bom desempenho os dados sempre que estes forem solicitados, visto que no melhor caso teremos tempos de execução perto de 1 e no pior caso tempos de execução perto de N.

```
//contem os dados de um perfil
typedef struct perfil{
    long long id;
    char nome[MAXNOME];
    long long nif;
    char cidade[MAXCIDADE];
    char estado_civil[MAXESTADOCIV];
    char email[MAXEMAIL];
    int apagado;
} Perfil;

typedef struct user_hash{
    Perfil *user;
    struct user_hash *next;
}User_h;

//Tabela de hash com apontadores para Perfis
//tabela para pesquisa por nome e nif
User_h *hash_nome[MAXNODES];
User_h *hash_nif[MAXNODES];
```

Figura 2.1: users

Para estruturar as relações optamos pela utilização de listas de adjacência de grafos, com tamanho dinâmico. Estas tornarão-se úteis graças ao campo `id` dos utilizadores, pois podemos aceder aos relacionamentos dos utilizadores directamente através do campo `id`. Sendo assim no pior caso teremos um Tempo de acesso a um utilizador relacionado de  $N$  ou no melhor caso um tempo de acesso de 1. Decidimos alterar um pouco a estrutura inicial atribuindo ao primeiro nodo, ou seja ao nodo de partida o `nif` a quem corresponde, para obtermos uma maior eficácia na leitura e escrita do ficheiro.

```

2 //estrutura para saber quais os adjacentes d
3 //posicao do adjacente no grafo (indice no a
4 //peso da ligação
5 //apontador para o próximo adajcente
6 typedef struct adj{
7     long long id;
8     long long nif;
9     int peso;
10    struct adj *next;
11 } Adj;
12
13
14 typedef struct info {
15     long long nif;
16     Adj *adjacentes;
17 } Info;
18
19 //grafos para as diferentes relações
20 //tem o numero de nodos existentes no grafo
21 //tem apontador para a estrutura Nodo (ver a
22 //tem apontador para os nodos adjacentes
23 typedef struct grafo{
24     int tamanho;
25     Info **nodos;
26 } Grafo;
27

```

Figura 2.2: relacoes

Para as mensagens usamos também um lista de adjacência de grafos com tamanho dinâmico. Vamos então ter uma lista de destinatários, onde cada destinatário possui o numero total de mensagens recebidas, para não passar de 20, e as ultimas 20 mensagens recebidas. Visto o numero de mensagens total do programa poder ser muito grande e como tal causar problemas de espaço em memoria ram, optamos por trabalhar directamente do ficheiro com as mensagens, carregando apenas as mensagens solicitadas, ficando assim poucas mensagens carregadas na memoria ram de cada vez.

```

3 typedef struct mensagens {
4     char *mensagem;
5     struct mensagens *next;
6 } Mensagens;
7
8 typedef struct remetente {
9     long long id;
10    int n_mensagens;
11    long long nif;
12    Mensagens *msg;
13    struct remetente *next;
14 }Remetente;
15
16 typedef struct destinatario {
17     int total_mensagens;
18     Remetente *remetentes;
19 } Destinatario;
20
21 typedef struct lista_destinatario{
22     long long nif_popular;
23     int num_mensagens;
24     int dimensao;
25     Destinatario **L_mensagens;
26 }Lista_destinatario;
27

```

**Figura 2.3:** mensagens

### 2.2.2 Utilizadores

Os utilizadores são constituídos pelo campos id,nome, morada,estado civil,email, nif ,e o campo apagado. Os primeiros serão referentes aos dados pessoais da pessoa enquanto este ultimo será um campo usado apenas internamente, pois ao apagar uma conta o utilizador continua na base de dados mas com o campo apagado activado, o que vai provocar com que o utilizador seja “ignorado” pelo programa até que este reactive a conta e então o campo apagado passa a estar desactivado.

Para a o tratamento dos utilizadores criamos três módulos diferentes, o modulo users, responsável pelas operações internas dos utilizadores na estrutura de dados, o modulo users-file responsável pelas as operações de escrita leitura de utilizadores no ficheiro e o modulo users-IO responsável pela interacção entre utilizador e as operações deste nas estrutura de dados.

#### Modulo users

Este modulo é responsável pelas operações sobre os dados do utilizador.

Para tratamos os dados pessoais dos utilizadores decidimos criar duas tabelas de hash uma destinada ao nome e outra destinada ao nif da pessoa, que por nos foi considerado o “login” do utilizador na rede social.

As tabelas são ambas de tamanho fixo (10000) e para a colisão de posições na tabela optamos por usar chaining pois assim não teremos um numero máximo de utilizadores.



Para o endereçamento dos utilizadores nas tabelas serão então considerados o campo nome para a tabela do nome e o nif para a tabela do nif.

Para que este endereçamento seja o mais distribuído possível e que torne a utilização de dados o mais eficaz e rápida possível(o mais próxima de 1) desenvolvemos duas funções de hash uma para o campo nome e outra para o campo nif.

Para o campo nif conseguimos desenvolver uma função que tendo em conta os diferentes dígitos do nif gera um índice aleatório, o que se torna mais importante no nosso caso pois o acesso aos utilizadores será quase sempre feito através do nif e logo teremos um acesso/pesquisa de utilizadores bastante rápido e eficaz..Esta função foi testada com os nif fornecidos pelo professor e por números gerados pela função rand e a utilização dos diferentes índices é de aproximadamente 80%.

Para o campo nome foi desenvolvida uma função diferente , onde são considerados os códigos da tabela ascii para o calculo do índice, esta função foi testada para os utilizadores fornecidos pelo professor e obtivemos uma taxa de 40% de utilização dos diferentes índices, o que torna o acesso aos utilizadores pelo nome um pouco mais lento(no pior caso  $\log(n)$  ) , o que no entanto não se torna muito relevante pois apenas é usado na pesquisa por nome de utilizadores feita pelo IO.

As operações de inserção,remoção,editar e apagar utilizador serão então feitas utilizando o nif para aceder ao utilizador na tabela caso este exista se não obteremos o “lugar” que este irá ocupar. Podemos então colocar o utilizador na estrutura de dados caso ele esteja a registar se pela primeira vez ou aceder ao utilizador e modificar os seus dados, e apagar a sua conta caso este já esteja registado.

### **Modulo users\_file**

Este modulo é responsável por tornar o programa capaz de guardar os utilizadores num ficheiro ao sair, para que estes não se percam e possam ser utilizados no futuro ao serem lidos e carregados quando se voltar a inicializar o programa.

Para tornar mais eficaz possível os processos de escrita/leitura do ficheiro optamos por escrever/ler em binário usando as funções fread e fwrite.

Desenvolvemos uma função que grava o ultimo id atribuído para que possa ser usado de novo, e todos os utilizadores registados no programa, e uma outra que vai ser capaz de ler o formato em que ficou escrita a base de dados.

Ficando desta forma sempre guardados os dados relativos aos utilizadores.

### **Modulo users\_IO**

Neste modulo encontram se todas as funções responsáveis pela interacção entre o programa e utilizador.

Encontram se as funções que fazem a ligação entre a estrutura / base de dados e o utilizador, como por exemplo a função de pesquisa de pessoas , onde será pedido o nif ou nome ao utilizador e posteriormente transmitido às operações de pesquisa na tabela de hash.

### 2.2.3 Relações

Existem três tipos de ligações/relações, de amizade, familiar e profissional. Uma relação no programa será uma ligação entre dois utilizadores que será caracterizada pelo campo peso, que quanto maior for mais distantes serão os utilizadores ligados entre si, id e nif do utilizador que se ligou a outro mais o peso dessa ligação, será por nós chamada adjacente. As relações poderão ser criadas apagadas e editadas.

#### Modulo relações

Neste modulo estão presentes as funções relativas às operações básicas sobre relações, tais como criar, apagar, editar relação. Começando pela função de criação de relação, esta actua de duas maneiras diferentes perante dois casos diferentes.

O primeiro caso é de um utilizador que está a criar uma nova relação e ainda não tem nenhuma relação no respectivo grafo, então será preciso criar a informação do utilizador na posição respectiva na tabela de adjacência que corresponde ao id do utilizador, usando a função `inserir_info_user` que irá colocar o nif do utilizador e só depois colocar o adjacente a ele(ou seja ligado a ele) as informações da ligação. O segundo caso acontece quando já existe mais relações do utilizador e então apenas é criado o adjacente(as informações da ligação) pela função `inserir_relacao`.

Ao criar se uma relação esta vai se criar duas vezes ou seja vai ficar nas relações dos dois utilizadores que se estão a ligar.

As operações de apagar e modificar relações já se tornam mais simples. A primeira acede à adjacente e apaga do grafo, a segunda apaga e depois cria uma nova relação com os dados que se pretendeu modificar.

Foram também criadas as funções de para inicializar os grafos e limpar grafos ao sair do programa, uma função para aumentar o tamanho dos grafos sempre que o numero de utilizadores no programa fique próximo do tamanho actual do grafo das relações, para desta forma aproveitar se o máximo possível de memoria.

Neste modulo estão presentes ainda as funções de caminhos entre utilizadores através das suas relações, para o seu desenvolvimento recorreremos ao algoritmo de dijkstra.

#### Modulo `relacoes_file`

Neste modulo estão as funções responsáveis pela escrita/leitura das relações no ficheiro. Optamos por usar uma forma de escrita/leitura parecida à usada para os dados dos utilizadores, mas adaptada para os dados que representam cada ligação, que são os nifs dos utilizadores que estão ligados e o peso da mesma.

As relações entre dois utilizadores apenas são guardadas uma vez no ficheiro de forma a poupar recursos, isto devesse à escolha que fizemos no tipo de estrutura de dados para as relações, ou seja deve se ao campo `info` que permite guardar/saber a informação de quem tem a relação e com quem tem a relação.

## Modulo `relacoes_IO`

Neste modo foram incorporadas e desenvolvidas as funções de interacção entre o utilizador e as operações sobre relações.

É constituído por um conjunto de funções que se limitam a retirar dados introduzidos pelo utilizador e a fornecer as operações de manutenção consoante o tipo de operação que o utilizador pretende fazer como é óbvio, a mostrar os resultados pretendidos aos utilizadores do programa.

### 2.2.4 Mensagens

As mensagens no programa são o conjunto de caracteres que um utilizador origem envia para um utilizador destino, ficando posteriormente guardada a mensagem no conjunto de mensagens que o utilizador destino recebe deste utilizador origem e onde só ficam guardadas as 20 ultimas de cada utilizador origem.

Para manter as mensagens guardadas e acessíveis optamos por trabalhar directamente do ficheiro carregando as mensagens temporariamente para a memoria ram, pois seria bastante difícil, se não impossível carregar as mensagens para memoria quando forem em numero elevado.

## Modulo `msg_operac`

Desenvolvemos neste modulo as funções que vão trabalhar com a memoria necessária para as mensagens solicitadas pelo utilizador. Sendo assim neste modulo estão presentes funções como a função que inicializa uma lista de destinatários, para onde serão carregadas as mensagens, que queria espaço físico para as mensagens, a função `act_tam_list_msg` que vai alocando mais espaço para a memoria sempre que tal seja necessário e uma outra que vai libertar espaço da memoria eliminando a mensagem de lá.

Estas são as funções básicas que trabalham com a memoria, sempre que é necessário carregar e ou usar mensagens na mesma.

## Modulo `mensagens_file`

Como já referimos o programa foi implementado de forma a trabalhar com as mensagens do ficheiro para que não se torne lento e incomodo ao trabalhar com ele.

Para tal foram desenvolvidas varias funções que vão “trabalhar” as mensagens a partir do ficheiro. Para que tal seja possível, criamos algumas funções que localizam o destinatário e mensagem que se pretende carregar para a memoria, após estas estarem localizadas irão ser lidas e carregadas para a memoria através de funções que trabalham com o ficheiro e a memoria.

Apesar destas serem as funções fulcrais em relação às mensagens, estão também presentes neste modulo um conjunto de funções que nos permitirá ver quem é o utilizador mais popular do programa ou seja, este será escrito também no ficheiro e sempre que algum utilizador recebe uma mensagem será comparado o numero de mensagens que esse utilizador recebeu com o numero de mensagem de do utilizador mais popular e ai então caso seja maior o numero de mensagens,

o nif do mais popular irá ser substituído por o deste. Desta forma conseguimos manter sempre actualizado o utilizador mais popular de uma forma simples e eficaz.

### **Modulo mensagem\_IO**

Tal como nos módulos anteriores de “IO” apresentam se neste modulo funções de interacção entre utilizador e operações básicas de funcionamento. Este modulo é então constituído por funções que irão permitir ao utilizador escrever e ler mensagens e também ver que é o utilizador mais popular.

## **2.3 Interface e Menus**

Para a interface do programa optamos por utilizar a biblioteca ncurses.

Causou nos sem duvida mais trabalho pois foi a primeira vez que trabalhamos com esta biblioteca e como tal tivemos de aprender o seu funcionamento,

No entanto achamos que foi bastante positivo termos usado esta biblioteca, pois para alem de a ficarmos a conhecer, também desenvolvemos uma interface mais bonita e agradável ao utilizador. Podemos então afirmar nem que seja como opinião pessoal que o programa ficou agradável ao uso do utilizador.

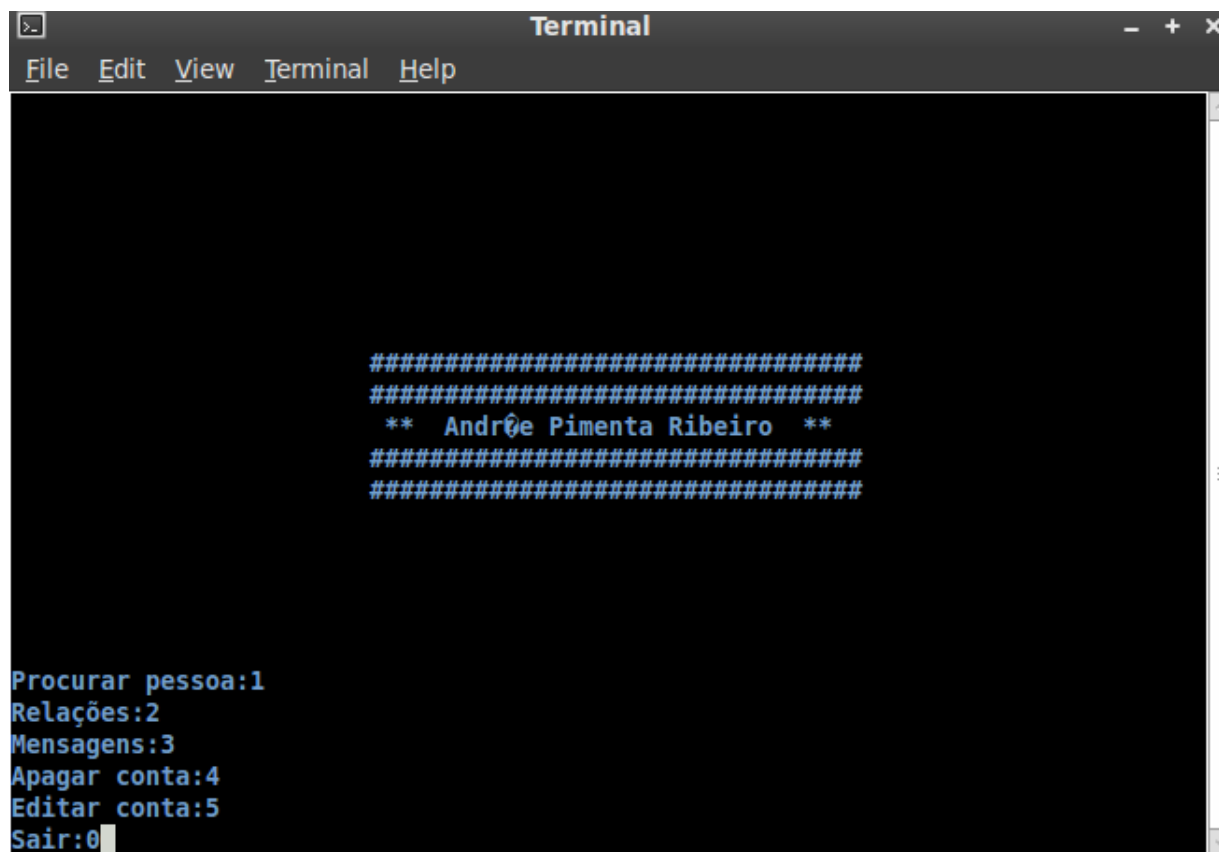
### **2.3.1 Menus**

Como já referimos antes a interface do programa do qual fazem parte os “menus”, foi desenvolvida usando a biblioteca ncurses, e como tal os menus foram desenvolvidos com a mesma.

Os menus serão a principal fonte de interacção entre o utilizador e o programa, e como tal importância que tem, tentamos desenvolver algo fácil de usar e agradável ao utilizador

Para torna isso possível optamos por fazer menus que são utilizados através de dígitos, ou seja as escolhas do utilizador para poder navegar no programa serão feitas através do input de dígitos.

Existem varias opções de navegação do programa, desde o login ao registar utilizador que se apresentam disponíveis ao utilizador, mas não nos alonguemos muito sobre esse assunto pois já foi falado e apresentado na segunda milestone do projecto.



```
#####
#####
**  André Pimenta Ribeiro  **
#####
#####

Procurar pessoa:1
Relações:2
Mensagens:3
Apagar conta:4
Editar conta:5
Sair:0
```

Figura 2.4: sempreLigados

## 2.4 Erros

Como uma boa programação modular exige, fizemos um modulo destinado a todo os tipos de erros possíveis do programa. Sempre que apareça um erro, sendo ele de ficheiro de utilizadores ou de nif inexistente entre outros, este será imprimido para o stderr e para a janela criada pelo ncurses.

## 2.5 Funcionamento geral

Após a apresentação dos diferentes módulos do programa pode se agora de forma sucinta apresentar e explicar o funcionamento geral do programa

Sempre que se iniciar o programa este irá carregar a base de dados das relações e dos utilizadores. Desta forma o utilizador terá logo acesso à base de dados e a todos os dados relativos aos utilizadores e as suas relações.

O utilizador terá então hipótese de se registar caso ainda não o tenha feito, se já se encontra registado poderá fazer o seu login, e a partir dai trabalhar directamente com os seus dados, o que no entanto não o impede de ver outros utilizadores mesmo que não estejam ligados directamente a si, pois o programa possui algoritmos de pesquisa de outros utilizadores e das

relações alheias. O mesmo não acontece com as mensagens, mas por opção nossa pois achamos que as mensagens podem ser consideradas de privadas e como tal cada um pode ver as suas mensagens e apenas as suas mensagens, isto é algo que acontece por exemplo nos mails. O utilizador terá então acesso a mensagens, relações, pesquisas e manipulação total da sua conta.

Ao sair do programa todas as alterações serão gravadas pois antes de este terminar serão invocadas funções que guardarão as alterações dos utilizadores e das relações. As mensagens vão sendo gravadas ao longo do programa e portanto vão se gravando, não sendo assim necessário uma função para as guardar quando o programa encerra.

## 2.6 Complexidade e Desempenho

Após a conclusão do programa fizemos vários teste ao programa e calculamos alguns tempos de execução de e espaço que este ocupa em memoria.

### 2.6.1 Complexidade

O programa ao carregar 18500 utilizadores (utilizadores fornecidos pelo professor mais alguns adicionados manualmente) ocupa cerca de 4mb de memoria ram, sendo se deve ao tipo de dados que atribuímos aos campos do utilizadores,

- long long id
- long long nif
- char nome[51]
- char cidade[21]
- char estado\_civil[12]
- char mail[51]
- int apagado

Ocupando em media cada utilizador 140kib de memoria ram

As relações fornecidas pelo professor irão ocupar cerca de 65mb de memoria ram, o que se deve a serem em numero muito maior cerca de 526674. As relações contêm os campos

- long long nif
- long long nif
- char\* tipo
- int peso

O programa com 18500 utilizadores e com 526674 relações irá ocupar cerca de 69mb de memoria, o que parece bastante aceitava, para a quantidade de dados que são carregados para a memoria.

### 2.6.2 Desempenho

Sendo as nossas estruturas de dados , tabelas de hash e grafos não orientados, os tempos de execução serão para as tabelas de hash no pior  $w(N)$ , caso todos os utilizadores fiquem com o mesmo índice( o que não está a acontecer como já referimos anteriormente) e no melhor caso teríamos um acesso de  $O(1)$ .

Ao carregar os 18500 utilizadores o programa está a demorar cerca de 0m0.044s, isto usando um Processador Intel Petium Dual CPU T2390 @1.86GHz Podemos então concluir que devido ao bom desempenho das funções de hash, especialmente a do nif conseguimos obter um bom tempo de execução para os utilizadores.

Relativamente às 526674 relações carregadas e testadas na mesma maquina demoraram cerca de 0m2.328s a carregar. Como estamos a usar grafos não orientados no pior caso teremos um tempo de acesso de  $w(N)$  para um determinado índice, isto acontecerá caso um utilizador esteja ligado com todos os outros utilizadores da rede social, o melhor caso será quando  $O(1)$  e acontecerá quando um utilizador não tiver ninguém ligado a si ou apenas uma pessoa ligada a ele.

Relativamente as mensagens, estas terão um tempo de execução um pouco diferente pois vai se trabalhar a partir do ficheiro, terão que ser localizadas e carregadas, uma de cada vez, conforme forem solicitadas.

No geral o nosso programa demora cerca de 0m3.628s a carregar as informações todas, e os vários tempos de execução ao longo da execução do programa podem variar de caso para caso , como já foi explicado anteriormente.

## 2.7 Ferramentas utilizadas

Durante esta etapa, as ferramentas mais utilizadas pelo grupo foram as seguintes:

**Kile** - Editor gráfico para LATEX;

**Valgrind**

**Debugger**

**NetBeans** - IDE de código aberto para desenvolver software nas linguagens Java, C/ C++ , PHP e outras;

**gedit** - editor oficial de texto plano para o Gnome.

É importante referir que todos os elementos do grupo utilizaram neste projecto a distribuição do Sistema Operativo Linux, **Ubuntu**.



## Capítulo 3

# Conclusão

Após a exposição dos passos de construção e do funcionamento do projecto por nós elaborado, podemos dizer que ao longo da construção do projecto surgiram algumas dificuldades, tais como o desenvolvimento de uma interface através da biblioteca ncurses, o uso de grande quantidade de informação e a forma de estruturar esta, o mais eficaz possível e sem perdas de memória. A construção deste projecto foi para todo grupo uma grande ajuda no desenvolvimentos de conhecimentos relativos de programação na linguagem C, no uso de estruturas de dados para grandes quantidades de informação e no uso de algumas ferramentas dos sistemas operativos unix. Relativamente ao projecto em si, tentamos cumprir todos os objectivos propôs e ainda acrescentar algo mais do que o pedido, exemplo disso foi o uso da biblioteca ncurses que nos tirou mais algum tempo no desenvolvimento do trabalho, porém desenvolvemos um projecto capaz de aguentar um grande numero de utilizadores de relações e de mensagens. Numa forma de auto-avaliação do trabalho desenvolvido pelo grupo, é considerado pelo mesmo que este foi efectuado de uma forma positiva, pois foram alcançados vários objectivos.

## Capítulo 4

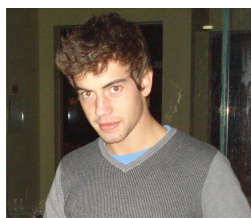
### Fotos



**Figura 4.1:** João Miguel



**Figura 4.2:** André Pimenta



**Figura 4.3:** Daniel Santos