Universidade do Minho
Licenciatura em Engenharia Informática
Laboratórios de Informática III

MilestoneII-ProjectoII

Ano Lectivo de 2009/2010

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

26 de Maio de 2010

Resumo

Neste relatório é apresentado as diferentes implementações propostas para o projecto sóAmigos. O projecto foi desenvolvido com três tipos de dados de estruturas diferentes, usando as Collections HashMap, TreeMap ,ArrayList e LinkedList.
Para sabermos qual a melhor implementação foi feita uma medição de tempos de execução e a comparação entre os mesmo para que se possa concluir quais as melhores opções para a estruturação de dados, foi feita também uma comparação entre os tempos de execução das tarefas em comum entre o projecto 1 em C e o projecto 2 desenvolvido em JAVA.
De forma sucinta vai então ser apresentados o desenvolvimento do projecto "sóAmigos" em JAVA qual a sua implementação e quais as melhores escolhas sobre as estruturas de dados.

Conteúdo

Capítulo 1

Introdução

1.1 Contextualização e apresentação do Caso de Estudo

Esta milestone tem como objectivo principal o uso de uma linguagem de programação diferente da usada no projecto anterior, para que se possa ver as diferenças de entre a performance da implementação em C do projecto anterior com a actual em JAVA, mas também a diferença de desempenho entre diferentes estruturas de dados. É também importante no aspecto do uso de uma linguagem de programação diferente, e com outro funcionamento, neste caso programação orientada a objectos.
. Pode se então afirmar que esta milestone se baseia na introdução de uma nova linguagem de programação e de uma pequena noção da diferença de performance no uso de diferentes linguagens de programação e no uso de diferentes estruturas de dados.

1.2 Motivação e objectivos

Esta milistone teve como grande motivação o uso de uma nova linguagem de programação(JAVA) ainda em estudo na unidade curricular de POO em projectos com alguma dimensão.
Apesar de ser o principal objectivo e aspecto motivacional , a diferença de tempos de execução para a linguagem C até agora usada e mesmo dentro do próprio JAVA usando diferentes estruturas e dados foi algo interessante e importante.

1.3 Estrutura

O presente relatório é constitúido por três capítulos: **Introdução**; **Desenvolvimento**, divido em , **Análise do Problema**, **Desenvolvimento do Projecto**,**Tempos de execução** 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

Para o desenvolvimento da rede social SóAmigos, uma rede social parecida ao FaceBook e ao Twiter foi usado a linguagem de programação JAVA proposta pelo docente da cadeira de Laboratórios de Informática III.
Tendo em consideração os aspectos de maior ênfase desta mesma rede social, as mensagens e os dados dos utilizadores foi dada uma maior importância a estes mesmos usando diferentes implementações dos mesmos para que se possa saber qual a mais eficaz.

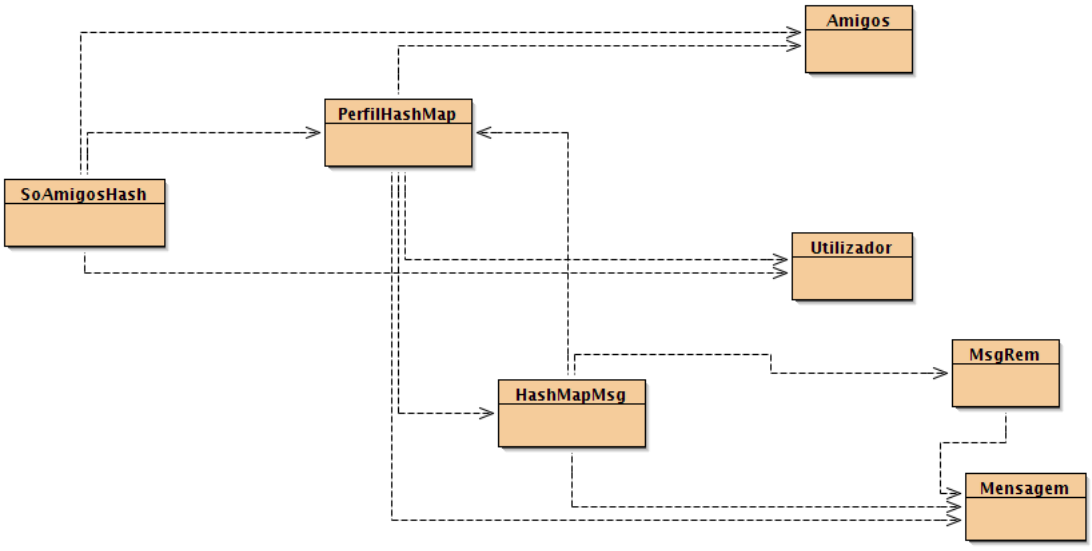

Figura 2.1: Estruturação soAmigos

2.2 Desenvolvimento do Projecto

2.2.1 Class's

Mensagens

Para a implementação do "sóAmigos" decidimos criar uma class Mensagens que tem como variáveis de instância uma String que é a mensagem em si , e data em que esta foi enviada, tendo então um int dia, int mês , int ano.
Nesta class apenas estão definidos os construtores e métodos básicos, pois é uma class básica no nosso projecto e não necessita de mais para o seu bom funcionamento.

No entanto a class Mensagem sozinha não consegue transmitir informação suficiente relativamente a quem pertence a mensagem ou de quem a enviou, paraa obtenção de algumas destas informações foi desenvolvida a class MsgRem.

A class MsgRem tem como variáveis de instância um o nif a quem pertence as mensagens e um arrayList de mensagens.
Optamos por esta implementação pois com um arrayList poderemos ordenar as mensagens por ordem de chegada, controlar o numero de mensagens que cada utilizador envia(neste caso 20 de cada utilizador) para outro utilizador.
No entanto estas duas classes continuam a serem insuficientes para termos as informações todas necessárias sobre quem enviou o que enviou e para quem enviou .Por isso é que se criou uma class posterior onde será armazenada as mensagens dos diferentes remetentes para um mesmo destinatário.
Esta class foi implementada de diferentes maneiras , em HashMap, TreeMap, ArrayList, e será explicada posteriormente.
Resumindo as mensagens estão estruturadas por "caixas" de vinte mensagens que um remetente envia para um destinatário e que serão guardadas juntamente com a informação de quem enviou na estrutura de mensagens do destinatário que contem todas as mensagens enviadas para este por diferentes utilizadores.

Amigos e Utilizadores

A class amigos não é mais do que um set de nifs de pessoas que se ligam a um utilizador. Usando A Collection Set, evitamos amigos repetidos e evitamos ter uma oredem de amigos pois tal não é necessário
A partir deste conjunto de nif de dados dos amigos poderemos aceder aos dados destes e ter acesso á sua informação e realizar operações sobre estes.

A class utilizadores contem como variáveis de instância o nif a morada e o nome do utilizador, ou seja vai conter apenas os dados das suas informações pessoais, esta é uma class muito simples.

Perfil

A class Perfil que está implementada de diferentes maneiras tal como a class que contem todas as mensagens de um destinatário, em HasMap, TreeMap, ArrayList, é de certa forma a class mais importante de projecto, pois vai ter como variáveis de instância um utilizador, amigos e as mensagens que este recebe dos outros utilizadores. Ou seja vai ser através da class Perfil que teremos acesso a todas informações e operações sobre um conta na rede social, vai ser através desta que teremos acesso as mensagens de um utilizador aos seus dados pessoais, aos seus amigos, ou seja a todos os dados que uma conta do "sóLigados" possui.
Esta class será posteriormente guardada em outra class que estará organizada em TreeMap, HashMap ou ArrayList, que por nós foi chamada de SoAmigos , podendo assim guardar se todas as contas dos diferentes utilizadores numa única classe e ter acesso a toda a informação da rede social nesta.

2.2.2 ArrayList

O uso de arrayList e de LinkedList por nós revelou se das diferentes implementações a mais lenta nos tempos de execução, provavelmente este facto deve se ao facto de estas necessitarem ser ordenadas usando um Compareted ao contrario do que se verifica nos Maps onde tal operações se efectua automaticamente.
Umas das principais características que se destaca nos tempos de execução é o facto de os tempos aumentarem com o aumento do numero de utilizadores em todas as operações, pois sendo a operação de comparação e o facto de ser necessário encontrar dentro destas o utilizador de forma meia manual.
Apesar dos maus desempenhos ambas as estruturas se revelam de forma surpreendente mais rapidas no tratamento de mensagens, e digamos de forma surpreendente pois em todos os campos de perfil esta se revelou mais lenta.
Apresenta se a seguir os resultados obtidos:

ArrayList

Carregar Base de Dados(ms)				
Tamanho	5000	10000	15000	18000
Tempo médio	6912.00	13239.00	34136.00	55369.00

Percorrer e imprimir todos os utilizadores(seg)				
Tamanho	5000	10000	15000	18000
Tempo médio	1.53	3.65	6.52	7.27

Percorrer a estrutura(ms)				
Tamanho	5000	10000	15000	18000
Tempo médio	892.00	1245.00	3421.00	3723.00

Inserir Utilizador(µs)				
Tamanho	5000	10000	15000	18000
Tempo médio	1067.00	1074.16	1074.37	1090.98

Procura por Nif(µs)				
Tamanho	5000	10000	15000	18000
Tempo médio	421.57	425.94	474.29	478.44

Procura por Nome(µs)				
Tamanho	5000	10000	15000	18000
Tempo médio	526.72	1171.44	1709.38	2119.02

Enviar msg (µs)				
Tamanho	5000	10000	15000	18000
Tempo médio	737.82	822.05	820.29	824.98

Figura 2.2: Tempos arrayList

2.2.3 HashMap

Para o armazenamento dos dados correspondentes aos perfis e ás mensagens usando HashMap, criamos a Class HashMapPerfil que irá guarda a base de dados da rede social seguindo a implementação em HashMap e as mensagens que cada utilizador recebe, seguindo os mesmos princípios.
Podemos afirmar que o uso de HashMap é relativamente fácil e com bons tempos de execução.
Ao usar HashMap as operações de pesquisa inserção e remoção são feitas em tempos constantes, e sendo a variação destes mesmo tempos mínima relativamente à mudança de quantidade de utilizadores que o programa contenha como poderemos ver na tabela relativa aos tempos de execução correspondeste a HashMap.
Através desta podemos ver mesmo que a operação de inserção é independente do tamanho da base de dados, mantendo sempre um valor constante e independente da quantidade de informação contida, o mesmo comportamento é esperado para a operação de remoção de dados pois tanto a operação de inserção e de remoção possuem mecanismos muito parecidos. Estes mesmo comportamento também é verificado nas mensagens pois também é realizado sobre elas a operação de inserção referida anteriormente.

HashMap

Carregar Base de Dados(ms)				
Tamanho	5000	10000	15000	18000
Tempo médio	11.00	23.00	36.00	43.00

Percorrer e imprimir todos os utilizadores(seg)				
Tamanho	5000	10000	15000	18000
Tempo médio	3.00	6.16	7.32	8.48

Percorrer a estrutura(ms)				
Tamanho	5000	10000	15000	18000
Tempo médio	1.00	3.00	3.60	4.00

Inserir Utilizador(µs)				
Tamanho	5000	10000	15000	18000
Tempo médio	2.23	2.24	2.26	2.25

Procura por Nif(µs)				
Tamanho	5000	10000	15000	18000
Tempo médio	1.60	1.69	1.71	1.75

Procura por Nome(µs)				
Tamanho	5000	10000	15000	18000
Tempo médio	1.64	1.67	1.71	1.78

Enviar msg (µs)				
Tamanho	5000	10000	15000	18000
Tempo médio	201.34	234.02	214.45	216.77

Figura 2.3: Tempos HashMap

Podemos então afirmar que o uso de HashMap possui a grande vantagem de a ordenação de dados ser desnecessária, pois o acesso aos dados contidos nesta será feito através da chave correspondeste ao campo de identificação dos dados a guarda na HasMap, e é este um dos principais motivos dos tempos apresentados na tabela anterior, pois a quantidade de informação guardada na HashMap não afectara muito a organização de dados da mesma.
Já as operações de carregamento da base de dados, imprimir e percorrer já têm comportamento diferente do anterior, pois agora o principal factor será a quantidade de informação a ser processada, dai a verificarmos um aumento de tempo de execução resultante de um aumento de quantidade de informação processada.
Concluindo o uso de HashMap é uma boa opção para o armazenamento de dados cujo o principal motivo seja aceder a dados por ordem independente, e a qualquer altura, para o acesso de dados que se encontrem em grande quantidade pois os tempos de execução não variam muito.

2.2.4 TreeMap

A TreeMap é uma estrutura de armazenamento que ordena os elementos através da chave por uma regra estabelecida. Quando esta ordem não é definida pela interface Comparable ou por um objecto Comparator1 o TreeMap busca a ordem natural dos elementos.
Usamos este tipo de armazenamento de dados e criamos a Class TreeMapPerfil que guardará os dados da rede social e as mensagens de cada utilizador usando também TreeMap.
Embora a TreeMap seja bastante parecida com a HashMap em termos de programação embora a TreeMap tornasse menos eficiente no que toca a operações sobre a estrutura em comparação com a HashMap, como podemos ver posteriromente na secção de Tempos de execução.
Ficam os tempos relativos a TreeMap na tabela seguinte.

TreeMap

Carregar Base de Dados(ms)				
Tamanho	5000	10000	15000	18000
Tempo médio	16.00	36.34	59.03	79.00

Percorrer e imprimir todos os utilizadores(seg)				
Tamanho	5000	10000	15000	18000
Tempo médio	2.72	3.69	5.32	5.97

Percorrer a estrutura(ms)				
Tamanho	5000	10000	15000	18000
Tempo médio	1.26	1.58	3.02	3.20

Inserir Utilizador(µs)				
Tamanho	5000	10000	15000	18000
Tempo médio	2.64	2.66	2.70	3.02

Procura por Nif(µs)				
Tamanho	5000	10000	15000	18000
Tempo médio	1.69	1.73	1.96	2.38

Procura por Nome(µs)				
Tamanho	5000	10000	15000	18000
Tempo médio	1.65	1.66	1.93	2.01

Enviar msg (µs)				
Tamanho	5000	10000	15000	18000
Tempo médio	259.33	234.39	247.57	261.23

Figura 2.4: Tempos TreeMap

2.3 Tempos de execução

2.3.1 Informações da máquina

```
Processador Intel Pentium Dual Core T2390 1.86Ghz - 1Mb cache L2 533Mhz
Disco Rigído - 250Gbs (5400RPM) SATA-II
Memória RAM - 2Gb DDR2 - 667Mhz
Sistema Operativo - Mint Elena 8
```

2.3.2 Metodologia experimental

Para a medição dos tempos era necessário usar mecanismos que permitissem medir o tempo com algum rigor como foi comunicado na palestra,para este projecto recorremos á class GregorianCalendar para a obtenção dos diferentes tempos de execução. Recorrendo então as várias funcionalidades desta class retiramos os diferentes tempos correspondentes ás diferentes estruturas de dados e ás diferentes quantidades de informção.

Detalhes:

- Para cada experiencia foram realizados 20 medições.
- Cada medição foi registada com base na media de 100 operações.
- Para cada experiencia é reportada a média.
- Nos casos em que existe discrepância entre valores esses valores são apresentados assim como a devida justificação de tais medições.
- Os tempos são reportados em microsegundos.
- As medições são baseadas num processador com resolução de 1/1.86GHz (0.53ns).
- Medições efectuadas para base de dados com 5000, 10000, 15000 e 18000 registos .

2.3.3 Diferença de tempos em JAVA

De entre as colecções utilizadas no trabalho (HashMap, TreeMap, ArrayList, LinkedList), e através dos tempos observamos vimos que correspondem em muitos dos casos às nossas expectativas, isto em relação às diferenças registadas entre entre os diferentes tipos mencionados. Por exemplo à partida sabiamos que uma implementação em ArrayList ou em LinkedList teria uma eficiencia muito reduzida em relaçao aos outros dois tipos (HashMap, TreeMap), pois sempre que realizamos uma operação de inserção, remoção ou procura temos de percorrer a lista e fazer as devidas comparações até encontrar a referência que procuramos. Isto leva a que o tempo de execussão de uma das operações que falamos atrás possa variar entre W(1) e O(N) pois se o objecto que queremos inserir na lista tem a chave com o maior valor da lista termos de percorrer toda a lista para o inserir. Sabiamos então que era entre as implementações em HashMap e ArrayList que estariam as melhores soluções para os nossos problemas. A implementação em HashMap é uma implementação que permite calcular códigos através de chaves fornecidas pelos utilizadores sendo uma forma mutio eficiente para aceder a dados espalhados em disco. Assim podemos prever que a inserção de um utilizador seja um processo bastante rápido pois o que é

preciso fazer é calcular a chave correspondente baseada no parametro nome ou nif dado como parametro e guardar um apontador para a estrutura nessa posição calculada. A solução TreeMap é uma solução que pode ser util do ponto de vista de procura visto que é uma solução que mantém as chaves ordenadas numa arvore binária. Assim encontrar um user é um processo bastante rápido pois é feito no máximo em tempo logaritmico.

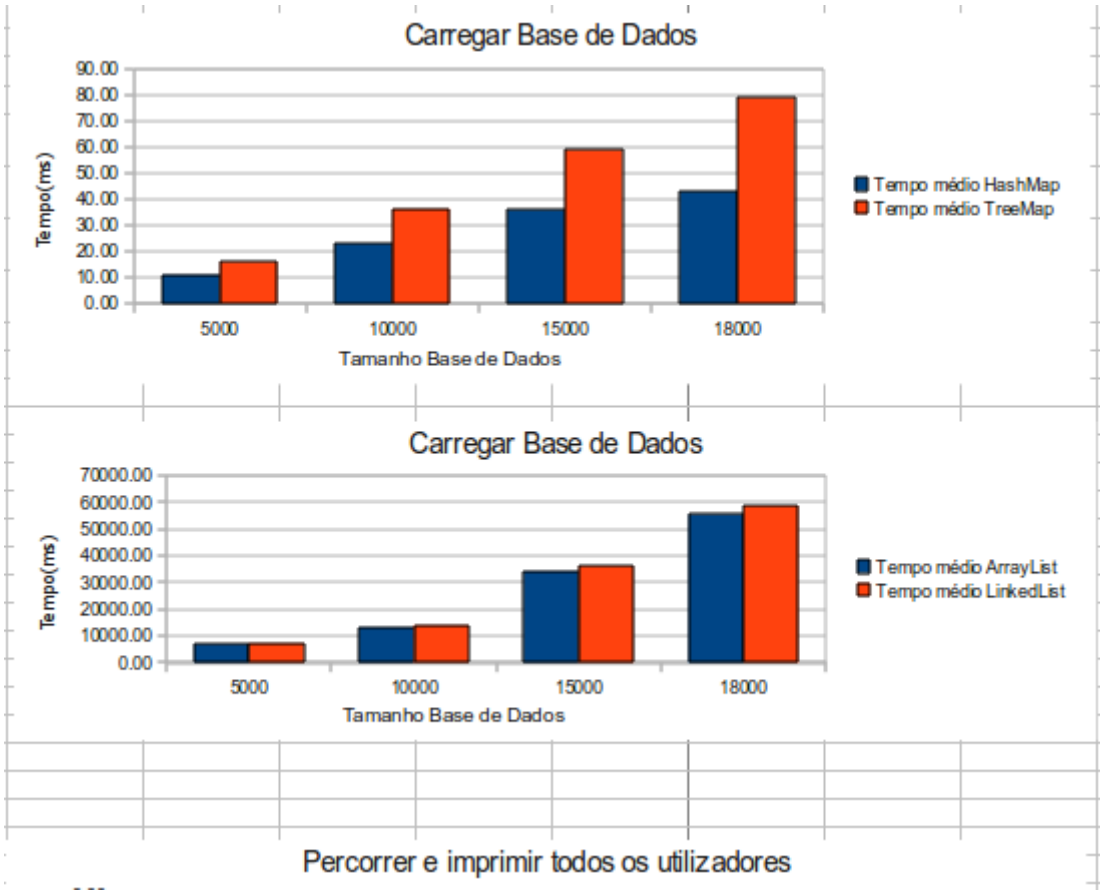

Figura 2.5: Diferença de tempos em JAVA

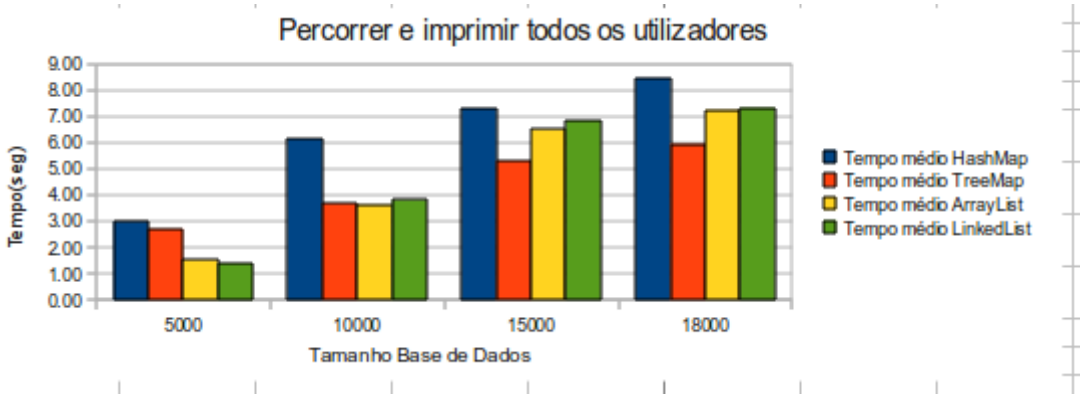

Figura 2.6: Diferença de tempos em JAVA

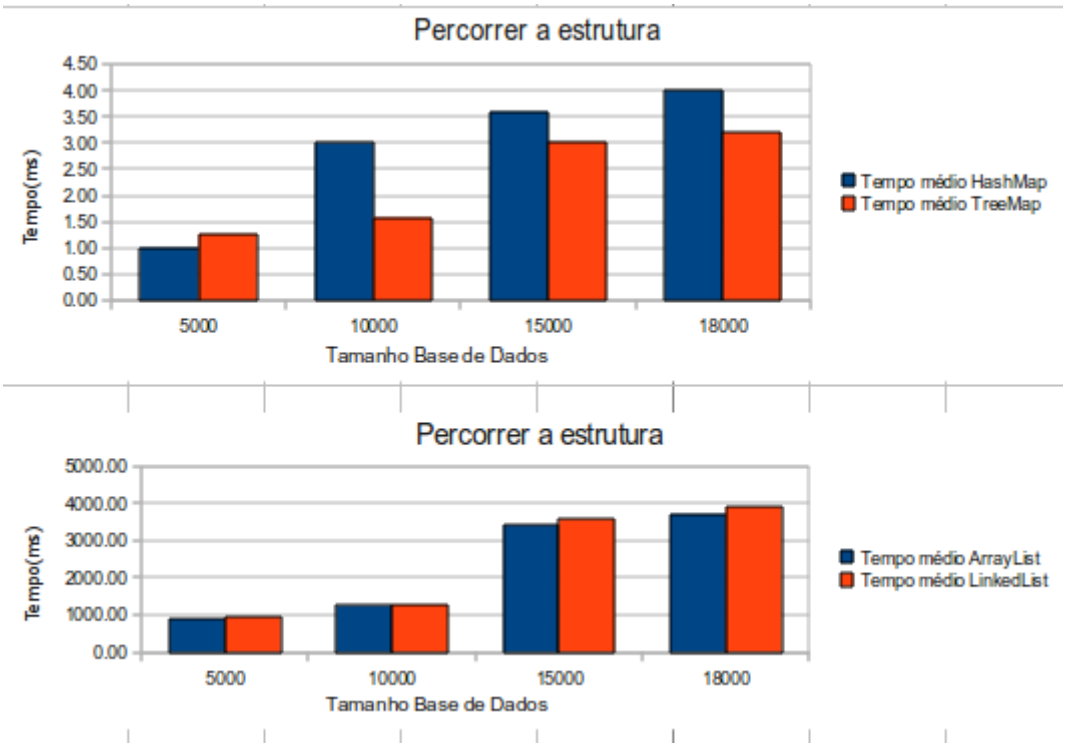

Figura 2.7: Diferença de tempos em JAVA

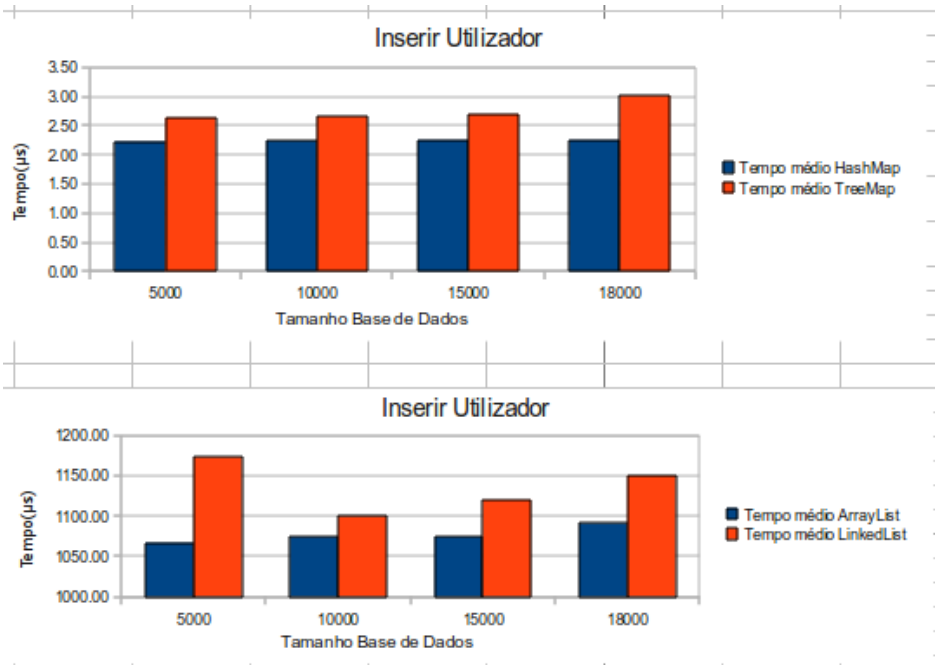

Figura 2.8: Diferença de tempos em JAVA

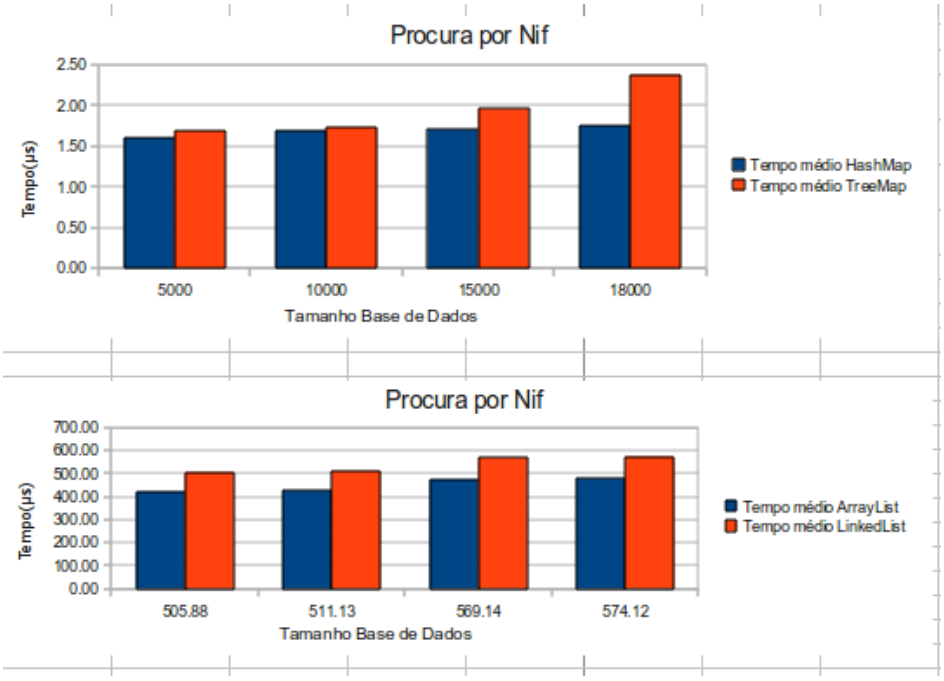

Figura 2.9: Diferença de tempos em JAVA

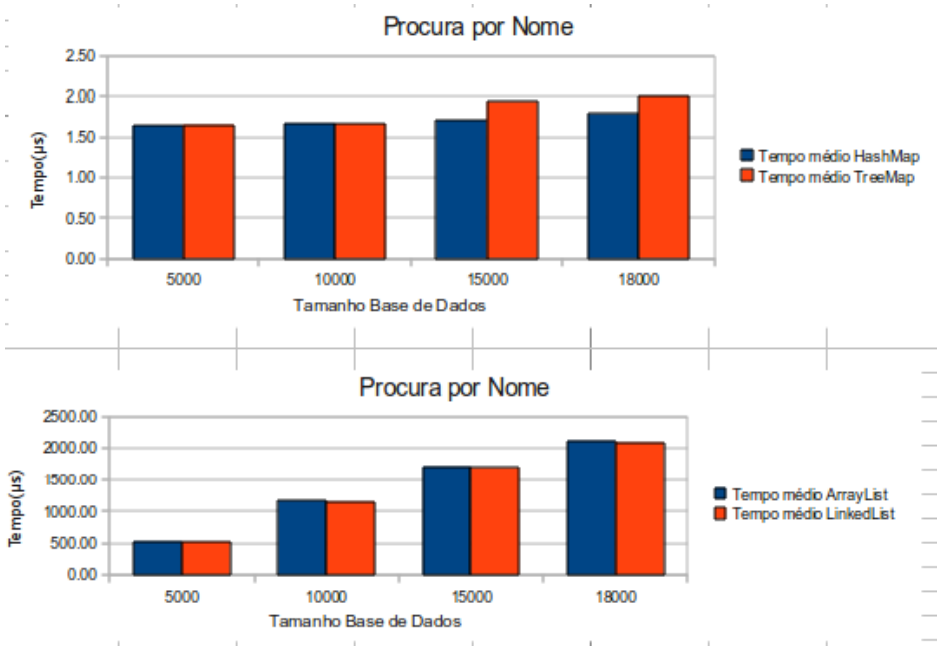

Figura 2.10: Diferença de tempos em JAVA

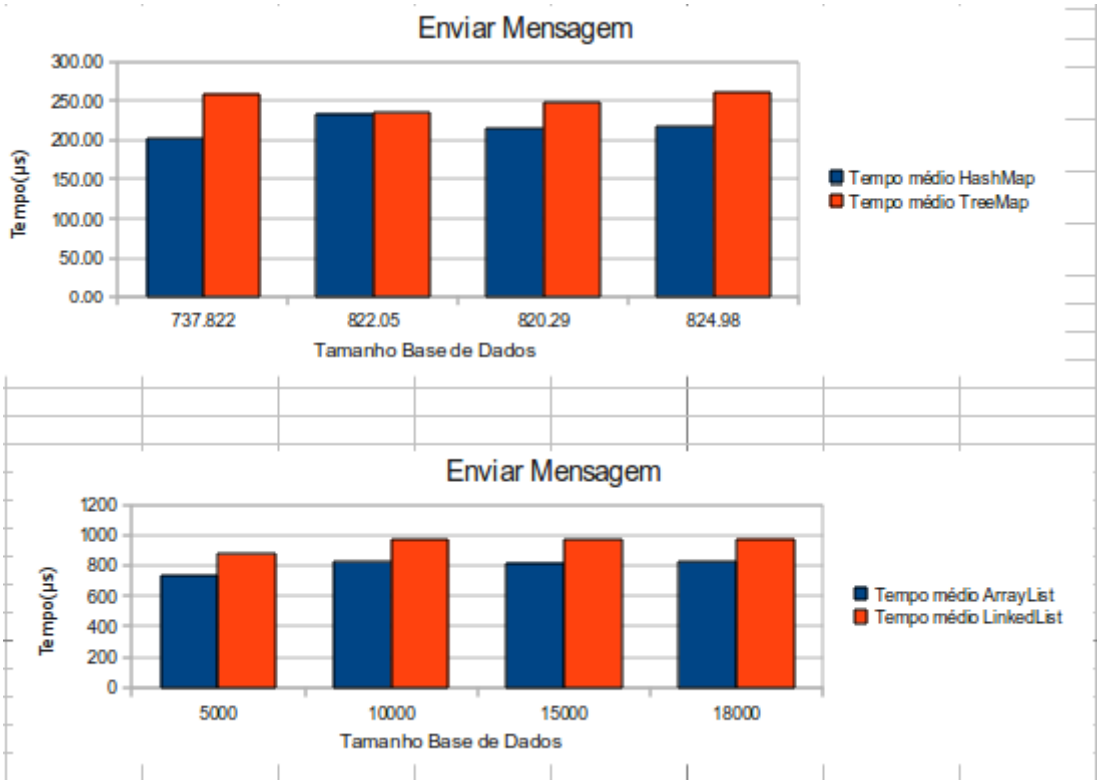

Figura 2.11: Diferença de tempos em JAVA

2.3.4 Diferença de tempos entre JAVA e C

Um dos objectivos e motivação por nós considerado foi a comparação e analise dos tempos das operações em comum entre os dois projectos. Para melhor se por analisar apresentamos em seguida uma tabela com os tempos retirados em C e com os tempos retirados em JAVA considerando a implementação HashMap que se verificou por nós a mais eficiente seguindo as nossas implementações próprias e considerando a base de dados com 18000 utilizadores.

Após a apresentação dos dados podemos afirmar que no geral a estrutura desenvolvida em

	Tempos para 18000 UTILZADORES	
	JAVA	C
inserir Utilizador(µs)	2.25	4.05
carregar Base de Dados(ms)	43	6890
Percorrer a estrutura e Imprimi	8.48	3.41
Procura por Nif(µs)	1.75	2
Procura por Nome(µs)	1.78	3.8
Enviar msg (µs)	216.77	81.8

Figura 2.12: Diferença de tempos entre JAVA e C

JAVA é mais eficiente e tem melhor desempenho, verificando se apenas melhores tempos na

operação de percorrer e imprimir a base de dados toda,e ao enviar mensagens. Apesar de a implementação em C estar mais eficiente nas mensagens isto muito se deve ao facto de em C ser apenas carregadas as mensagens de um único utilizador de cada vez para a memoria e de se trabalhar directamente do ficheiro ao contrario do que se passa na implementação em JAVA onde para já estão a ser carregadas todas as mensagens para a memoria.

Concluindo e resumindo podemos afirmar que o desempenho das mesmas operações em JAVA no geral é superior e que o seu desenvolvimento se revelou de certa forma menos trabalhoso e mais rápido.

2.4 Ferramentas utilizadas

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

Kile - Editor gráfico para LATEX;

BlueJ
NetBeans - IDE de código aberto para desenvoler 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 realização do relatório e de se analisarem todos os dados nele contido podemos retirar duas grandes conclusões.
Podemos então concluir que a melhor estrutura de dados a usar para o tratamento dos dados relativos aos perfis será a HashMap.
Apesar de a diferença de tempos de execução entre HashMap e TreeMap não ser muito grande e mesmo tendo a TreeMap melhores tempos ao percorrer a estrutura, a HasMap revela melhores tempos na operações mais importantes ou melhor nas operações com um potenciam maior de uso pelo utilizador, daí a escolha recair sobre HashMap.
Relativamente ao uso de arrayList ou arryList com uso de LinkedLis, está completamente fora de questão para o armazenamento de perfis, pois como podemos ver anteriormente revelou se o mais lento em todas ou quase todas as operações relativas aos perfis, isto provavelmente deve se à necessidade de ordenação de dados.
Relativamente à estrutura mais eficiente para o armazenamento e tratamento de dados revelou se curiosamente mais eficiente o uso de ArrayList, tal facto não se verificou para os perfis dos utilizadores antes pelo contrario. Podemos então afirmar que as estruturas mais eficientes serão HashMap para perfis e ArrayList para Mensagens no uso de JAVA.
A segunda grande conclusão que podemos retirar está relacionada com o uso das diferentes linguagens de programação neste caso C e JAVA, ou de pelo menos as estruturas escolhidas pelo programador para cada uma.
Apesar de os tempos de execução nas operações em comum nos dois projectos e dos resultados apresentados na milestnone anterior serem próximos dos apresentados pela estrutura com melhor desempenho desenvolvida por nós neste segundo projecto, existe uma ligeira diferença a ter em consideração ao carregar as bases de dados com vantagem para as estruturas desenvolvidas em JAVA.
Por sua vez revelou se mais rápida a implementação de mensagens usada em C, muito devido ao facto de apenas serem carregadas para memoria as mensagens correspondentes a um único utilizador, contrariamente ao que se verifica no projecto2.
O desenvolvimento das mesmas operações também se revelou menos trabalhoso e mais rápido em JAVA do que em C, apesar de não existir tanta liberdade de programação do que a linguagem C possui.

Capítulo 4

Fotos

Figura 4.1: João Miguel

Figura 4.2: André Pimenta

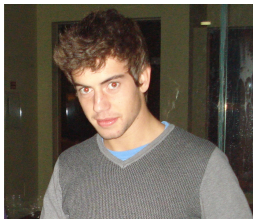

Figura 4.3: Daniel Santos