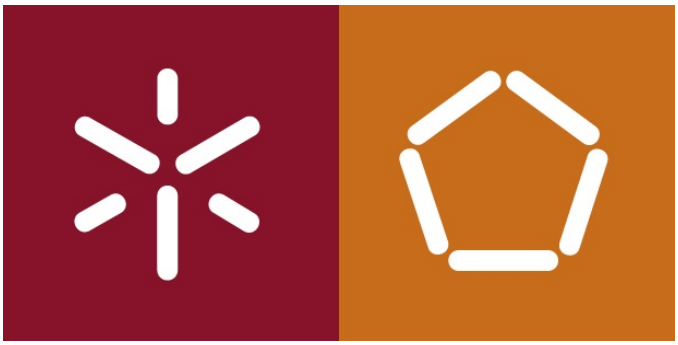
Universidade do Minho

Mestrado Integrado em Engenharia Informática



Stack Overflow

Laboratórios de Informática III

Grupo 51

Carla Cruz
A80564

Maria Dias
A81611

Pedro Freitas
A80975

June 12, 2018

Contents

Introdução

No âmbito da unidade curricular de Laboratórios de Informática III do segundo ano do Mestrado Integrado em Engenharia Informática foi proposto, numa segunda fase, o desenvolvimento de um projeto em linguagem Java, que visa o desenvolvimento de um sistema capaz de processar ficheiros XML que armazenam as várias informações utilizadas pelo Stack Overflow e a execução de um conjunto de interrogações de forma eficiente.

Este projeto já foi por nós abordado na linguagem C, e surge agora, no seguimento do anterior, o desafio de o converter (e, muito possivelmente, melhorar) utilizando a linguagem Java para que, no final, seja possível comparar os resultados obtidos em termos de performance, facilidade de programação, entre outros.

Leitura dos dados e Ferramentas usadas

Como foi previamente referido, este projeto, desenvolvido em Java, podia ser auxiliado de diversas ferramentas.

Em relação aos ficheiros .xml que foram disponibilizados, foi necessário proceder ao parsing destes. Este parsing foi realizado recorrendo à biblioteca *SAX* (Simple API for XML).

Para guardar os dados lidos após o parsing dos ficheiros .xml, o Java oferece um conjunto de classes que implementam as estruturas de dados mais utilizadas, que oferecem uma API consistente entre si, nomeadamente, *HashMaps* e *TreeMaps*.

Descrição das Coleções Utilizadas

Para a realização deste projeto, como referido acima, foi necessário proceder ao armazenamento de dados utilizando *HashMaps* e *TreeMaps*. Estes referidos tem como vantagem a sua fácil utilização, que permite uma fácil procura através das suas keys. Estas coleções também provêm de métodos de ordenação, inserção e remoção já pré-definidos, tornando a sua utilização ainda mais simples.

As principais diferenças entre estes são a sua organização dos dados em que um se baseia numa tabela hash enquanto que o outro se baseia numa árvore, sendo as *HashMaps*, as coleções mais usadas no desenvolvimento deste projeto.

Para guardar os Posts, debatemo-nos sobre a forma de o fazer pois no projeto em C tínhamos feito escolhas pobres relativamente à forma como armazenavamos posts da mesma data. Assim sendo, acabamos por seguir dois caminhos: primeiro, guardamo-los num HashMap com as correspondências id->Post, e depois guardamo-los também num TreeMap com as correspondências data->ArrayList<Post>, ou seja, numa estrutura que os guarda ordenadamente por data mais recente, sendo que a cada data é atribuído um array de Posts desse dia. Desta forma, deixamos de ter o problema que tínhamos na nossa estrutura anterior, em que sempre que queríamos os posts de uma dada data era calculado o valor de hash e depois, a partir disso, obtida a lista dos posts.

Quanto aos users, seguimos com um HashMap por considerarmos a estrutura mais eficiente para os guardar. Poderíamos também tê-los colocado num TreeMap, por exemplo, mas achamos que este tipo de dados é mais benéfico em situações que requerem fazer travessias, o que no contexto das interrogações propostas não acontece.

De igual forma, guardamos as Tags num HashMap.

A nossa estrutura consiste, então, num conjunto de 3 classes: a base de dados de users *UserBD*, a base de dados de posts *PostBD*, e a base de dados das tags *TagBD*, que consistem nas estruturas referidas acima.

Modularização e Abstração de Dados

Como já foi aprendido aquando da realização do projeto em C, é boa prática escrever código modular e manter tipos de dados abstratos.

Devido à forma como abstraímos o nosso código, podemos facilmente mudar algum detalhe na implementação do programa sem que isso prejudique todo o projeto, tendo de fazer pouquíssimas alterações, para que tudo funcione corretamente.

Ao longo do nosso programa, trabalhamos com clones de forma a não serem alterados os apontadores da estrutura interna.

O encapsulamento de dados também é muito facilitado pelos mecanismos que a linguagem oferece, tal como declarar a variáveis de instância como private.

Interrogações

Interrogação 1

Esta interrogação retorna o título e o nome do autor de um post, dado o id do mesmo. Caso o post dado seja uma resposta, deverá retornar as informações (título e autor) da pergunta correspondente.

Estratégia: Para responder a este problema primeiro pegamos no post cujo ID é nos dado como argumento. Depois de dado o id é feita a verificação se é do tipo 1 ou tipo 2. Caso seja do tipo 1, atráves do *OwnerUserId* conseguimos chegar ao id do User e depois ao seu nome, criando assim o *Pair* com o nome do user e o título da pergunta. Se for do tipo 2 através do *ParentId* conseguimos chegar ao post a qual responde, e a partir desse post procedermos da mesma maneira que o tipo 1.

Interrogação 2

Esta interrogação pretende obter o top N utilizadores com maior número de posts de sempre, tendo em conta todas as perguntas e respostas dadas por cada utilizador.

Estratégia: Para responder a esta interrogação, foi percorrida toda a estrutura dos users, sendo cada um adicionado a um Set ordenado pelo número de Posts correspondente a ele. De seguida é criada uma List de Users auxiliar colocando os id's dos N primeiros elementos dessa lista, num List<Long> final

Interrogação 3

Esta interrogação procura, dado um determinado intervalo de tempo, obter o número total de posts (identificando perguntas e respostas separadamente) efetuados nesse período.

Estratégia: Inicialmente começamos por pegar nos Posts organizados num *TreeMap<LocalDate,ArrayList<Post»* cuja key é um dia e o value é um ArrayList com todos os posts desse dia. Depois percorremos os dias que se encontram no intervalo de tempo dado, e a cada post do ArrayList fazemos a verificação se é do tipo 1 ou tipo 2. Se for do tipo 1 é incrementada a variável *per*, se for do tipo 2 é incrementada a variável *res*. Por fim introduzimos isso num *Pair* e devolvemos o mesmo.

Interrogação 4

Esta interrogação retorna os ids de todas as perguntas contendo uma dada tag, por ordem cronológica inversa da data das perguntas.

Estratégia: Inicialmente começamos por pegar nos Posts organizados num *TreeMap<LocalDate,ArrayList<Post»* cuja key é um dia e o value é um ArrayList com todos os posts desse dia. Percorremos os dias que se encontram nesse intervalo e cada post do ArrayList de cada dia retiramos as tags. Comparamos a *tag* passada como argumento com todas as tags de cada post (com o método *equals*) e caso sejam iguais , colocamos numa lista(*List<Long>*) o id do Post. No fim devolvemos essa lista.

Interrogação 5

Esta interrogação retorna a informação do perfil e as últimas dez publicações de um user do qual é dado o ID.

Estratégia: Primeiramente obtemos todos os Users e depois o User cujo o id é no dado em específico. Tendo o user, obtemos todos os posts dele e organizamos num *TreeSet<Post>* os posts ordenados pela Data (por cronologia inversa). Depois percorremos esse Set de posts e caso este seja do tipo 1 ou tipo 2 guardamos numa *List<Long>* auxiliar todos os posts. Por fim retiramos os 10 primeiros id's dessa mesma lista e devolvemos um par com a bio do User e a lista dos 10 últimos posts.

Interrogação 6

Esta interrogação procura, dado um determinado intervalo de tempo, obter os ids das N respostas com mais votos, por ordem descrescente do número de votos.

Estratégia: Começamos por pegar nos Posts organizados num *TreeMap<LocalDate,ArrayList<Post»* cuja key é um dia e o value é um ArrayList com todos os posts desse dia. Tendo esse TreeMap obtemos todas os posts dentro dessa data num ciclo *while* incrementado o dia e se o post for do tipo 2 colocamos num Set que, com um *Comparator*, organiza o Set pelo Score da resposta decrescentemente. Tendo esse Set com todas as respostas organizadas pelo Score, são colocadas numa Lista auxiliar de Posts e depois os id's dos N primeiros elementos dessa lista são colocados num List<Long> final.

Interrogação 7

Esta interrogação retorna os IDs das N perguntas com mais respostas num dado intervalo de tempo, por ordem decrescente do número de respostas.

Estratégia: Inicialmente organizamos os Posts num *TreeMap<LocalDate,ArrayList<Post»* cuja key é um dia e o value deste é um ArrayList com todos os posts desse dia. Guardamos entao neste TreeMap todos os Posts nesse intervalo de tempo e, de seguida, colocamos num Set que é ordenado com base nas respostas de cada post. Tendo esse Set ordenado, são colocados numa Lista auxiliar de Posts e sendo de seguida os id's dos N primeiros elementos dessa lista colocados num List<Long> final.

Interrogação 8

Esta interrogação devolve os IDs das N perguntas cujos títulos contêm uma dada palavra, por ordem cronológica inversa destas.

Estratégia: Para a resposta a esta interrogação, foi percorrida a estrutura dos Posts, e caso a palavra dada como parâmentro se encontrasse no título desse mesmo Post, este era adicionado a um Set, ordenado cronologicamente. No final, tendo já sida realizada esta seleção dos Posts, são colocados numa Lista auxiliar e de seguida os id's dos N primeiros elementos dessa lista colocados num List<Long>.

Interrogação 9

Esta interrogação devolve as últimas N perguntas (por ordem cronológica inversa) em que participaram dois utilizadores específicos.

Estratégia: Começamos por guardar todos os posts de cada user cujo ID nos foi passado (new ArrayList<>((this.com.getUsers().getUserMap().get(id1).getUserPosts())))

Interrogação 10

Esta interrogação devolve a melhor resposta dada a uma certa pergunta, da qual é passada o ID.

Estratégia: Primeiramente começamos por obter todos os Posts. Depois de todos os posts obtidos verifica-se se cada post é do tipo 2 e se o *OwnerId* do post é aquela passado como argumento. Se for esse o caso é guardado num Set temporariamente. Depois de obtidas todas as respostas do post, é calculado a pontuação através da fórmula estabelecida e devolvemos o id da melhor resposta(maior pontuação).

Interrogação 11

Esta interrogação retorna, dado um intervalo de tempo, os IDs das N tags mais utilizadas pelos N utilizadores com melhor reputação, naquele período.

Estratégia: Começamos por recolher o *HashMap* com todos o *Users* (*com.getUsers()*). Depois com o método *getNTopRep(users,N)* , que recebendo um HashMap e um inteiro (neste caso aquela passado como parâmetro à query) devolve-nos uma *List<Long>* com os N utilizadores com maior reputação. Para cada User dessa lista, obtém-se todos os posts do mesmo. Para cada post, se este pertencer ao intervalo dado, retiram-se as tags para uma *List<String>*. Depois de obtidas todas as tags, vamos percorrer a lista e com o método *addMapTag(Map<String,Integer> tags, String tag)* , que recebendo um Map<String,Integer> e uma String, adiciona ou incrementa o número de ocorrências dessa Tag no Map, sendo que a String corresponde ao nome da tag e o Integer ao número de ocorrências. Por último vamos a esse Map, procuramos a tag com o maior número de ocorrências adicionamos o seu ID numa List<Long> e retiramos a mesma do Map, repetindo o processo N vezes.

Resultados e Testes de Performance

Após o desenvolvimento e codificação do projeto em C, foram realizados alguns testes de performance, que consistem na obtenção dos tempos de execução das interrogações, usando os backups *android* e *ubuntu*, dos quais lemos os ficheiros Users.xml, Posts.xml e Tags.xml, usando a biblioteca standard de C *time.h*.

Em seguida são demonstrados os resultados dos mesmos:

Uma vez que a quantidade de dados aumenta do backup android para o backup ubuntu, é aceitável que os tempos de execução para os carregar, ou seja, a fase de leitura e inserção de dados, também aumente consideravelmente. Comparando os valores de tempo de execução das interrogações, é possível observar nos gráficos que o tempo de execução não varia significativamente, uma vez que são registadas variações na ordem dos milissegundos, devido a uma estruturação dos dados independente da quantidade de dados acumulada.

Estes testes foram realizados numa máquina com um processador de 3,1GHz, com uma memória RAM de 8GB.

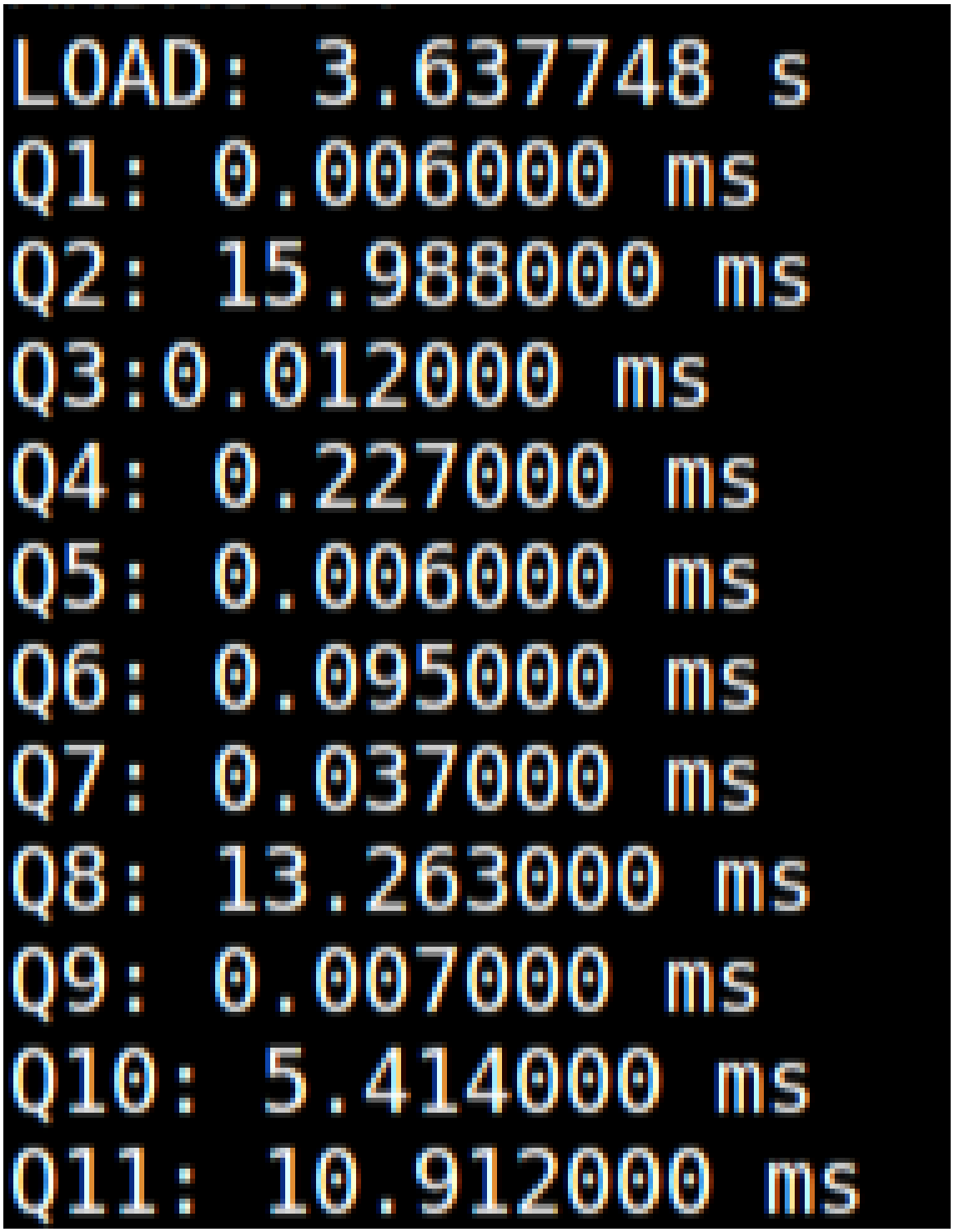


Figure 1: Tempos na dump android - C

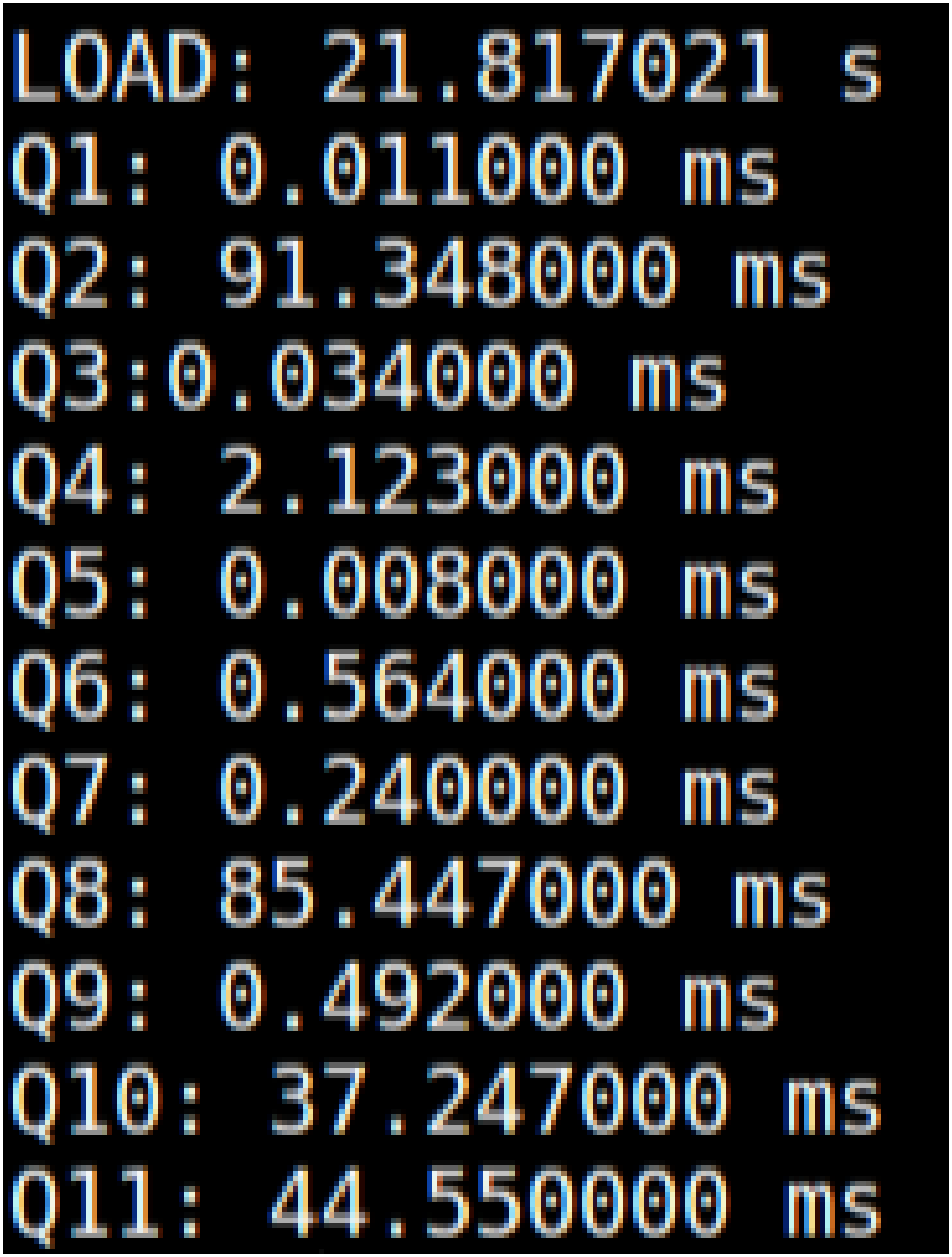


Figure 2: Tempos na dump ubuntu - C

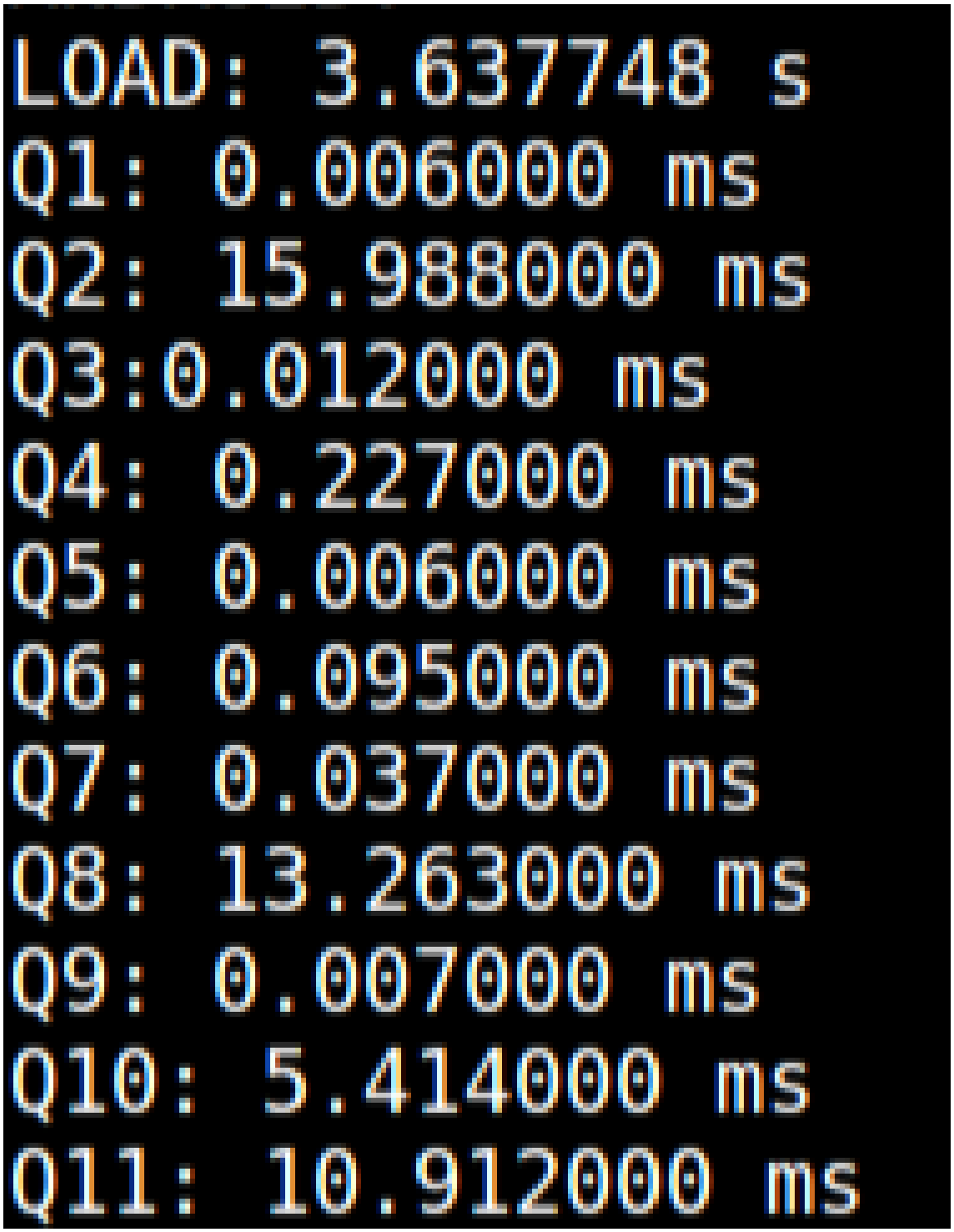


Figure 3: Tempos na dump android - C

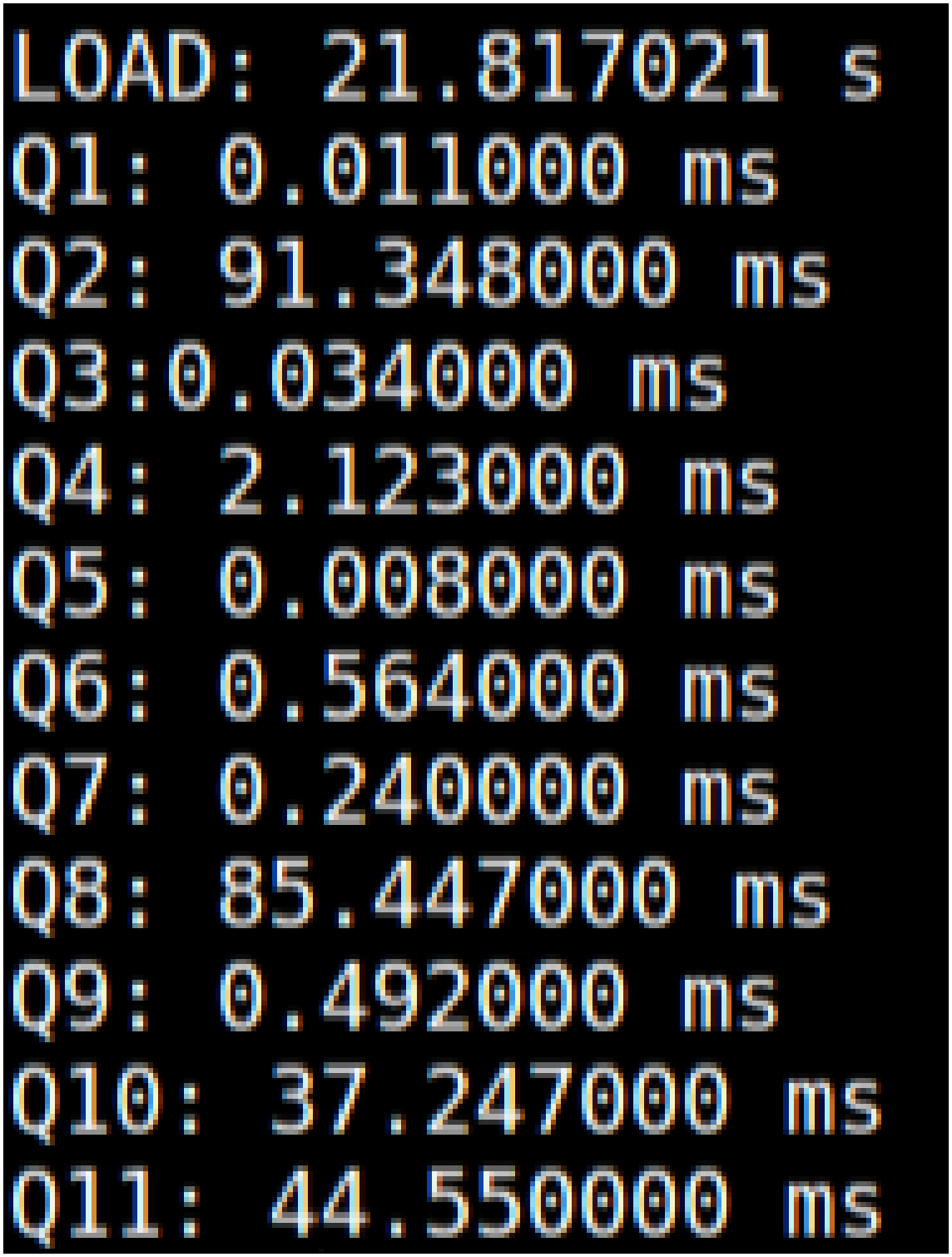


Figure 4: Tempos na dump ubuntu - C

Conclusão

Nesta segunda fase do projeto, sentimos um à vontade maior, dado que, na primeira fase termos já trabalhado com blocos de dados e já entendíamos melhor o conceito de abstração de dados e moldularidade, conseguindo assim, delimitar a vista do utilizador.

Na concessão deste projeto, sentimos uma maior facilidade em relação ao outro devido à API que a linguagem Java nos disponibiliza. Esta linguagem também tem a vantagem, em relação ao C, de correr numa máquina virtual, não tendo de haver uma preocupação existente na outra fase relativamente a leaks de memória.

Resta acrescentar que, comparando as duas fases do trabalho, podemos afirmar que este projeto se tornou numa tarefa muito mais simples de implementar do que foi verificado usando a linguagem C.