A89583 - Bruno Filipe de Sousa Dias
A89597 - Luís Enes Sousa
A85088 - Pedro Alferedo Fontes Machado

Laboratórios de Informática III

Projeto em C - Grupo 7

Introdução

Este trabalho foi realizado no âmbito da Unidade Curricular de Laboratórios de Informática III. O trabalho passa pela implementação de um sistema capaz de dar resposta a queries pedidas, sobre a leitura de um conjunto de dados fornecidos em formato de texto pelos docentes. No fundo, consiste na implementação de um Sistema de Gerenciamento de Vendas para uma cadeia de distribuição com 3 filiais.

A primeira fase do trabalho passaria pelo desafio de implementar este sistema na linguagem C, de forma a aumentar não só o conhecimento dos alunos em C, mas também, e talvez o aspeto mais importante do trabalho, apresentar as dificuldades que surgem (em qualquer que seja a linguagem) quando começamos a realizar programação em grande escala (aplicações com grandes volumes de dados e elevada estrutura algorítmica e estrutural).

Era permitido aos alunos (e recomendado) ter como recurso certos tipos de bibliotecas, como por exemplo glib (que utilizamos no trabalho realizado), havendo no entanto restrições a bibliotecas genéricas tais como LINQ.

Além dos aspetos referidos, o trabalho tinha também como focos prioritários a modularidade, o encapsulamento de dados e também o MVC (Modelo Vista Controlador).

Ao longo do trabalho, consideramos que os maiores desafios passaram por conseguir implementar as estruturas de dados de forma a respeitar a segurança e focos como a modularidade e o encapsulamento, e ao mesmo tempo conseguir responder a todas as queries da forma mais eficaz e instantânea possível.

Uma parte muito importante do trabalho foi a análise prévia e cuidada de como poderiam ser implementadas as diferentes estruturas, de modo a obter a melhor performance possível. Assim, passamos alguns dias antes mesmo de começar a "programar" e a implementar as conclusões a que chegamos, a desenhar diferentes tipos de estruturas que poderiam ser boas na resposta às finalidades do trabalho. Chegando a um desenho e a uma arquitetura final que pretendíamos implementar assim o fizemos. No entanto ao longo da realização do trabalho foram surgindo pormenores que poderiam ser ajustados e que otimizariam em grande quantidade a performance final e assim foi feito.

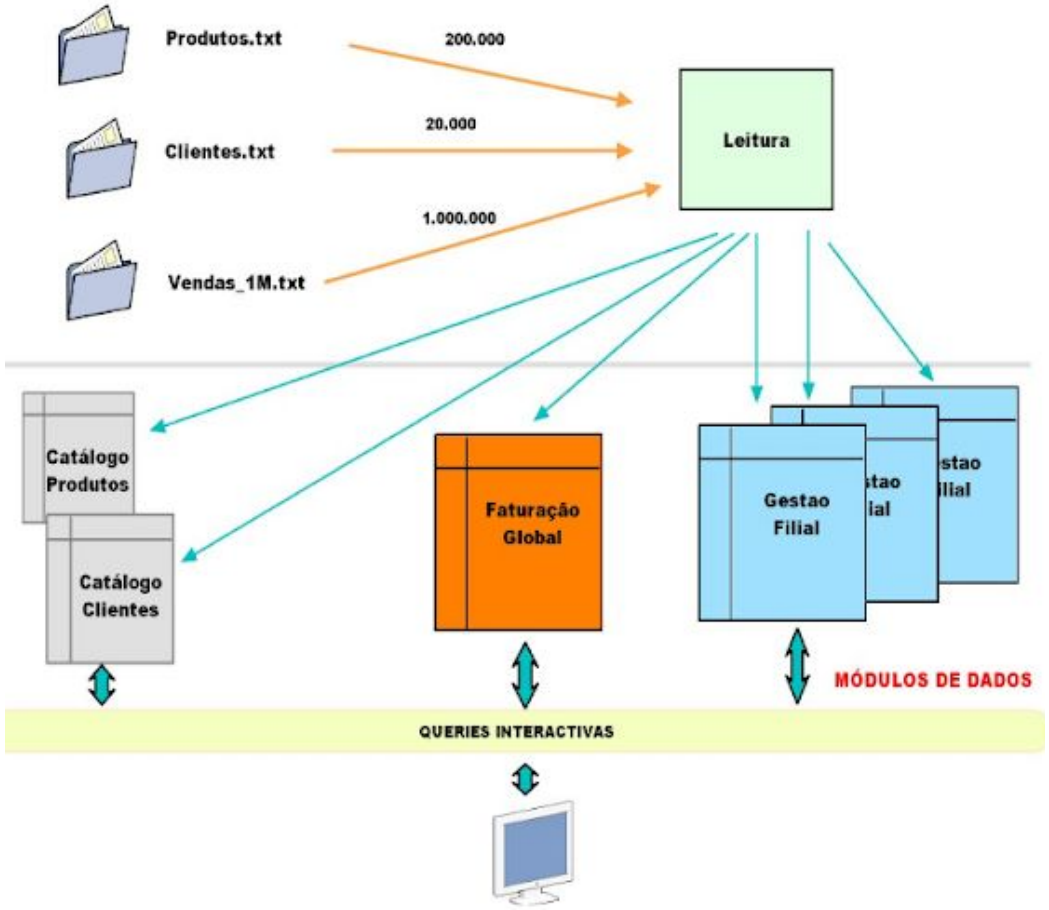
Ferramentas Utilizadas

Na realização deste trabalho, como referido, era possível a utilização de bibliotecas externas. Desta forma usamos estruturas da biblioteca glib, como por exemplo GPtrArray (Arrays de Apontadores) e GTree (AVL ou Balanced Binary Tree) pela sua robustez e qualidade. É importante realçar que achamos a forma como estas estruturas e esta biblioteca é apresentada ao público fraca em termos de interação com o utilizador e em termos de explicações a fundo de certos pormenores.

Foi utilizado também o software Valgrind, para deteção de memory leaks e para que tudo estivesse a funcionar corretamente.

Descrição da Estrutura do Projeto

A arquitetura do software é definida por 4 módulos principais: Cátalogos de Produtos e Clientes, Faturação Global e Vendas por filial cujos dados são retirados dos ficheiros de texto Produtos.txt, Clientes.txt e Vendas_1M.txt, uma interface que permite a comunicação máquina-utilizador (Modelo Vista Controlador).



Arquitetura de referência para o SGV

O ficheiro Produtos.txt contém cerca de 200.000 códigos de produtos. Cada linha é um código de Produto que é formado por duas letras maiúsculas seguidas de quatro dígitos.

O ficheiro Clientes.txt contém cerca de 20.000 códigos de clientes. Cada linha é um código de Cliente que é formado por uma letras maiúsculas seguidas de quatro dígitos.

O ficheiro Vendas.txt contém cerca de 1.000.000 de registo de vendas, onde cada código é formado por: código do Produto, código do Cliente, preço unitário do produto, número de unidades vendidas, o modo de compra (N - normal ou P - desconto) e o mês da venda.

Catálogo Clientes

typedef struct clientes *CLIENTES;

```
struct clientes {
    int * array[26];
    int size[26];

};
```

Os dados no Catálogo de Clientes são guardados num array de apontadores para inteiros com 26 posições, cada uma referente a uma posição do alfabeto (26 letras). Cada índice contém um apontador para um array de inteiros onde são guardados os dígitos presentes em cada código Cliente de forma ordenada crescentemente. Por exemplo o cliente A1234, faria com que o 1234 fosse introduzido no array de inteiros apontado pela primeira posição do array principal (array[0]). Como forma auxiliar existe também um array de inteiros com o número de clientes começados com uma certa letra, referida a uma certa posição.

A forma como a implementação desta estrutura foi feita permite-nos procuras mais rápidas por certos clientes, e de verificação de existência no catálogo. Uma vez que os número dos códigos estão ordenados e estes códigos separados pela letra inicial, podemos com uma procura binário chegar rapidamente a qualquer resposta.

Catálogo Produtos

typedef struct produtos *PRODUTOS;

```
struct produtos {
    int * matrix[26][26];
    int size[26][26];

};
```

O catálogo de Produtos foi implementado de maneira muito semelhante ao catálogo de Clientes, onde no entanto, em vez de ser utilizado um array é utilizada uma matriz. Esta matriz de apontadores para arrays de inteiros, tem como referência da linha a primeira letra do Produto e como referência de coluna a segunda letra do Produto. Assim será uma matriz 26x26. Cada posição desta matriz contém então um apontador para um array de inteiros onde são guardados os dígitos presentes em cada código Produto de forma ordenada crescentemente. Por exemplo o cliente AB1234, faria com que o 1234 fosse introduzido no

array de inteiros apontado pela posição da matriz referente á primeira linha e segunda colunas (matrix[0][1]). Como forma auxiliar existe também uma matriz de inteiros com o número de produtos começados por duas dadas letras (linha - 1ºLetra e coluna - 2ºLetra).

A forma como a implementação desta estrutura foi feita permite-nos procuras mais rápidas por certos produtos, e de verificação de existência no catálogo. Uma vez que os arrays de número dos códigos estão ordenados e estes arrays estão separados pelas duas letras iniciais por uma matriz (nas linhas a letra inicial, nas colunas a segunda letra) e dessa forma podemos com uma procura binário chegar rapidamente a qualquer resposta.

Faturação Global

typedef struct faturacao *FATURACAO;

```
struct faturacao {
    GTree* avlFatura;

};
```

```
struct nodoFatura { /* key = productID */
    PRODUTOS_FILIAL produtoFilial [3];
};
```

```
struct produtosFilial {
    INT_NP totalUnidades [12];
    FLOAT_NP totalFaturado [12];
    INT_NP totalVendas [12];
};
```

```
struct int_np {
    int normal;
    int promocao;
};
```

```
struct float_np {
    float normal;
    float promocao;
};
```

Este módulo contém as estruturas que serão responsáveis por responder de forma eficiente a questões quantitativas que relacionam os produtos às suas vendas mensais, em modo Normal(N) ou em Promoção(P).

O catálogo das faturas é constituído por uma AVL, mais propriamente uma GTree, da biblioteca glib, que facilita a ordenação constante dos dados e a dispensabilidade de ter de dar sorts a meio do código. Cada Nodo da AVL, tem como key/chave o código de um Produto, e possui um array de 3 posições (respetivo às 3 diferentes filiais) onde estão guardadas diferentes parâmetros relativos ao produto da key. Estes parâmetros são o número de unidades vendidas, o total faturado e o total de Vendas. Estes parâmetros estão todos eles divididos em arrays de 12 posições, sendo cada posição referente a uma certo mês. Cada posição tem então uma estrutura auxiliar onde que diferença o modo normal(N) e o modo promoção(P).

A utilização de GLib facilitou em grande parte a maneira como abordamos os diferentes problemas. Além disso, o facto de ser utilizada uma AVL melhora os tempos de procura, uma vez que estes são de ordem log(N). As estruturas que complementam a AVL principal foram sendo alteradas ao longo do trabalho de maneira a chegar a um ponto de equilíbro entre a segurança e a maneira de implementação dos dados e ao mesmo tempo a rapidez de resposta às diferentes queries.

O catálogo de Faturas, é iniciado (initFaturacao) fazendo quase um clone do catálogo dos Produtos, onde a key de cada nodo é um código de Produto e o conteúdo do Nodo é iniciado com todos os parâmetros (que são quantitativos) a 0. A função que liberta a

memória (freeFaturacao) recorre a uma função da glib g_tree_destroy, que dá free de forma automática, utilizando funções de libertação de memória anteriormente dadas na criação de cada nodo. A função de inserção de uma venda na Faturação (insereVendaFaturacao) procura um dado Produto na AVL e incrementa os valores (inicializados a 0) necessários e consoante as condições de dada venda a inserir (respeitando o tipo de compra, N ou P e a filial). A realização destas funções apoia-se em várias outras que são exclusivamente conhecidas por este módulo, tal como acontece noutros.

Filial

typedef struct filial *FILIAL;

```
struct filial {
    GTree* nodosFilial;

};
```

```
struct nodoFilial { /* key = clientID */
    GTree* produtosComprados;
};
```

```
struct int_np {
    int normal;
    int promocao;
};
```

```
struct float_np {
    float normal;
    float promocao;
};
```

```
struct produtosCliente { /* key = productID */
    INT_NP   totalProdutosComprados[12];
    FLOAT_NP dinheiroGasto[12];
};
```

Este módulo contém as estruturas que, para uma dada filial, serão responsáveis por responder de forma eficiente a de relacionamento entre Clientes e Produtos.

O catálogo das filiais é constituído por uma AVL, mais propriamente uma GTree, da biblioteca glib, que permitem manter uma ordem constante e sem necessidade de dar sorts. Cada Nodo da AVL, tem como key/chave o código de um Cliente, e possui uma outra AVL, onde estão contidos dados que dizem respeito às relações do cliente da key desse nodo e de todos os produtos que este comprou. Cada Nodo dessa AVL interna terá por ser vez como key um código de um Produto comprado por esse cliente, e vai conter uma estrutura onde estão guardadas diferentes parâmetros relativos à relação entre esse Cliente e esse Produto. Esses parâmetros são o número total de unidades vendidas e o dinheiro gasto nessa “relação”. Estes parâmetros estão, há semelhança da faturação, divididos em arrays de 12 posições, sendo cada posição referente a uma certo mês, onde cada posição vai ter uma estrutura auxiliar que diferença o modo normal(N) e o modo promoção(P).

Mais uma vez achamos o uso da AVL a melhor escolha neste módulo. Relembrando que usamos GLib o nos permite fazer com possamos dar diferentes valores às keys de cada nodo, e que neste trabalho essa funcionalidade foi uma mais valia. À semelhança do catálogo de Faturas, a Filial ao iniciar (initFilial), cria quase como um clone do catálogo de Clientes, onde a key do nodo é o código de Cliente e o conteúdo do Nodo inicia sem quaisqueres dados, tendo no entanto memória alocada para inserção dos mesmos. A função que dá free (freeFilial) utiliza o g_tree_destroy, função da GLib que mais uma vez torna mais fácil a realização de certos objetivos. A função de inserção de uma venda na

Filial (insereVendaFilial) incrementa os dados de uma relação Produto Cliente caso esta já exista e caso contrário, cria uma nova e insere-a no devido sítio.A realização destas funções apoia-se em várias outras que são exclusivamente conhecidas por este módulo, tal como acontece noutros.

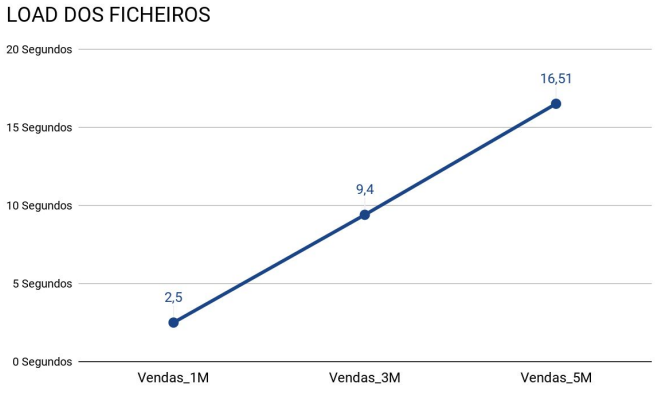
SGV - Sistema Geral de Vendas

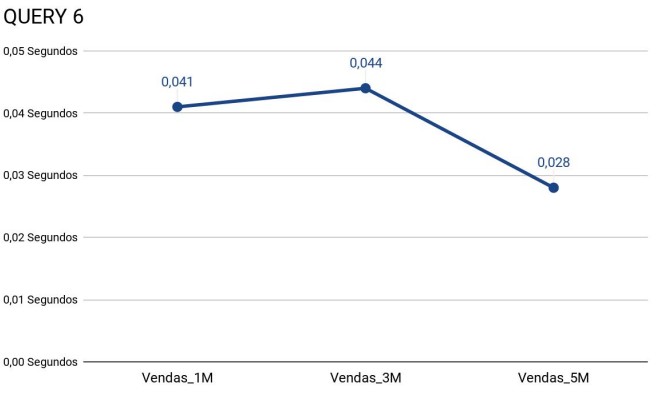
typedef struct sgv *SGV;

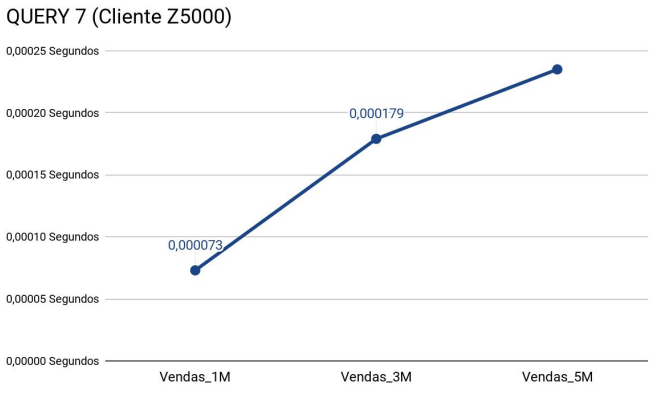
```
struct sgv {
    PRODUTOS produtos;
    CLIENTES clientes;
    FATURACAO faturacao;
    FILIAL filial [3];
};
```

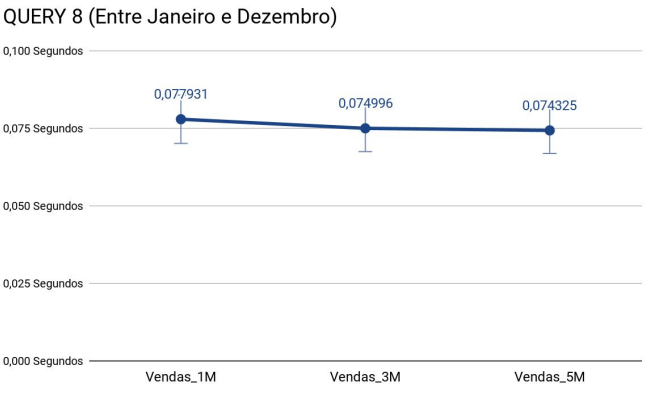
O SGV é uma entidade onde estão guardados os dados dos módulos referidos acima. Este SGV vai ser a estrutura entregue como parâmetro às queries, de modo a estas conseguir obter os dados necessários para a concretização e resposta das questões pedidas.

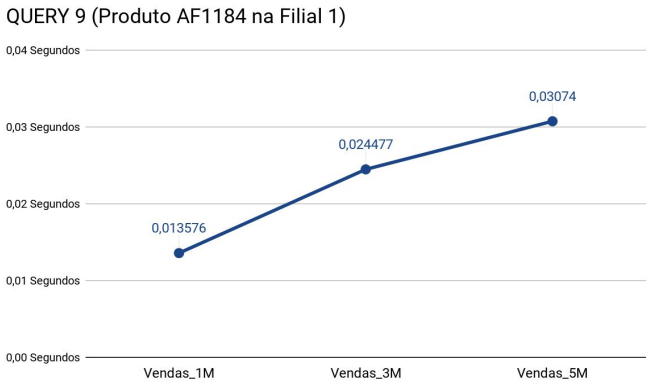
Testes de Performance

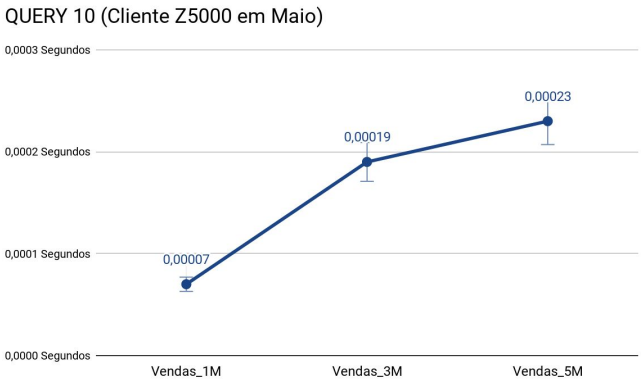


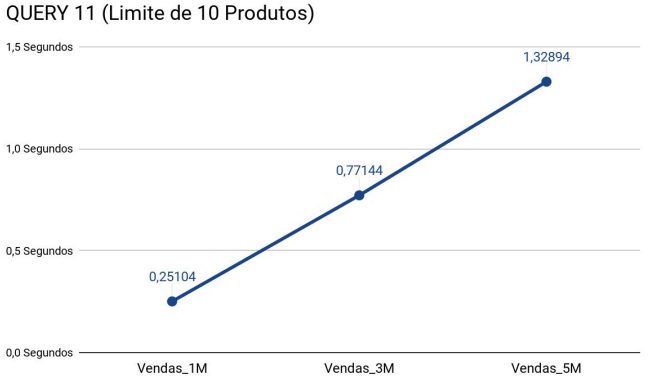


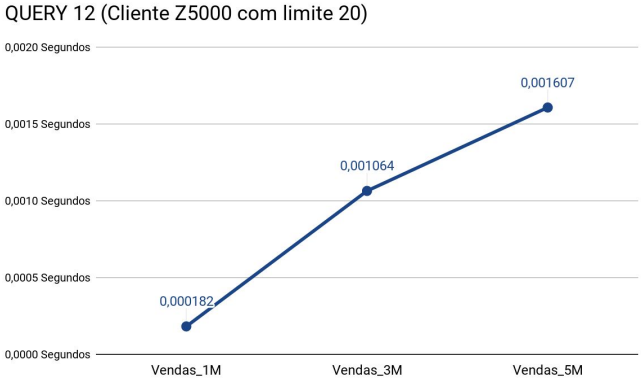












Foram realizados testes de performance, os quais se estão apresentado acima, que passam por obter tempos do load de ficheiros e das queries 6 a 12, usando os ficheiros Vendas_1M.txt (1 Milhão de vendas), Vendas_3M.txt (3 Milhões de vendas) e Vendas_5M.txt (5 Milhões de vendas), utilizando a bibioteca standard de C chamada time.h.

Estes tempos de teste foram registados numa máquina com um processador de 2.2GHz e com uma memória RAM de 16GB.

Com o aumento do número de vendas, é normal também que os números aumentem no que diz respeito aos tempos de execução na fase de leitura de dados e de inserção dos mesmos nas diferentes estruturas. Comparando contudo os vários valores de execução dos diferentes gráficos, que representam execuções de diferentes queries, podemos perceber que não existem grandes variações significativas, sendo estas na ordem dos milisegundos.

Dificuldades e Problemas Encontrados

Ao longo do trabalho foram surgindo vários desafios, aos quais tentamos responder da forma mais eficaz possível. Um desses problemas, que surgiu inicialmente, foi na implementação da biblioteca glib, na forma como poderíamos dar Cast da forma mais correta possível, e o modo como poderíamos construir as funções auxiliar de maneira harmonizada, uma vez que aqui trabalhamos com apontadores abstratos.

Um outro problema que não conseguimos detetar foi na query 8, uma vez que não retorna o resultados certo para a avaliação do ano todo (mês 1 a 12), no entanto, pensamos que seja por causa de certos arredondamentos. No load/leitura de ficheiros deparamo-nos também com outro problema. Aqui se realizarmos a leitura/load, em seguida dermos free, e dermos load novamente, o carregamento do catálogo dos Clientes (unicamente este) tem uma duração significativamente diferente do que na primeira vez que é feito o load. Esta diferença é quase imperceptível caso não fossem imprimidos os tempos com o time.h, uma vez que estas operações são quase instantâneas.

Outro problema foi os vários impactos com a dificuldade de otimização de certas queries consoante a estruturação de dados possuída, o que levou a remodelar algumas estruturas, ainda que muito poucas vezes.

Conclusão

Concluído o trabalho, e atingidos os objetivos do mesmo, ficamos a reter alguns pontos essenciais. Conseguimos perceber as dificuldades de programar em grande escala, e as complicações que isso pode trazer. Conseguimos também perceber que por vezes é difícil conseguir alcançar um balanço que satisfaça e agrade tanto a segurança dos dados (encapsulamento) por parte de terceiros, ou por vezes, até do próprio utilizador, bem como a rapidez de resposta a algumas questões.

Outra parte importante de reter foi a maneira como todas as estruturas de dados são organizadas, e a forma de como esta organização pode por vezes melhorar ou piorar um algoritmo, sendo necessário refazer a estrutura de modo a otimizar cada vez mais a performance, até se chegar a um *sweet spot*. Este ponto leva também à importância da reutilização de código e da generalização do mesmo, uma vez que com um bom código que respeitasse essas condições conseguiria-se reformular o trabalho depois de alterar as estruturas, apenas mudando alguns pontos chave em algumas funções de forma bastante simples.

Achamos contudo, e apesar de o grupo ter utilizado GLib que no fim valeu muito a pena, no entanto, achamos a API desta biblioteca por vezes bastante difícil de entender e não achamos que fosse tão interativa quanto desejávamos. Contudo, no fim de realizarmos o trabalho achamos a utilização desta Biblioteca uma mais valia para a realização do projeto, todavia, achamos que este pormenor deveria ser mais abordado nas aulas, já que os alunos são bastante incentivados pelos docentes a usarem o mesmo (pode ter sido apenas por causa da situação atual da sociedade, porém achamos importante referir).