Mestrado Integrado de Engenharia Informática

Laboratório de Informática III

PROJETO PROGRAM

(MIEI – LI3 20/21)

Grupo Nº40:

- Pedro Aquino Martins de Araújo - A90614

- Rafael dos Anjos Arêas - A86817

- Vitor Lelis Noronha Leite - A90707

Data de Entrega: 05/05/2021

ÍNDICE

Estruturas de dados utilizadas

Estrutura de dados utilizado para armazenar os dados dos ficheiros:

- **Qual?** Árvore balanceada, gTree importada da biblioteca glib.
- **Utilizado nos módulos:** Nos catálogos; business, review e users.
- **Motivo de escolha:** Utilizamos pela boa eficácia , realizando procura, inserção, remoção em tempo médio de O (log n). Também pela maior facilidade em manusear as funções do glib, em comparação às outras estruturas de dados.

Estrutura de dados utilizado para o Table:

- **Qual?** Array de ponteiros, gptrArray importada da biblioteca glib.
- **Utilizado nos módulos:** Table.
- **Motivo de escolha:** Utilizamos pela necessidade de possuir um array apontado a structs auxiliares (usado para guardar informações necessárias das querys) , e para strings (usado para fazer o print formatado para a interface).

Descrição dos Módulos

Main: Módulo responsável pela execução do programa. Cria as estruturas de dados que serão usadas e chama as funções dos outros módulos para manipular as determinadas estruturas.

Interface: Módulo utilizado para fazer toda ,ou quase maioria, de I/O do programa. Por exemplo, há funções responsáveis em recolher os argumentos necessários para as determinadas queries (*query2Interface*) e outras para imprimir o conteúdo da própria Table (*printQuery*).

AuxStructs: Módulo que contém uma struct auxiliar , necessária para a Table, contendo nesta, elementos necessários para armazenar informações requisitadas pelas queries.

Table: Módulo que contém a estrutura Table, utilizada pelas queries, para armazenar os respectivos dados requisitados das funções.

SGR: Módulo onde possui a estrutura SGR que a partir dela e de suas funções irá gerar a estrutura Table feita por cada query. A estrutura do SGR consiste, basicamente, na informação dos 3 catálogos juntos (Cada um em uma Árvore Balanceada gTree) para facilitar o cruzamento de informações entre os mesmos quando necessário.

Catálogos:

- Business: Neste módulo criamos uma estrutura de dados que armazena os dados específicos de um determinado negócio (Id, nome, cidade, estado e categoria) e colocamos cada negócio na árvore balanceada usando os Id's como parâmetro para o balanceamento.

- Users: Neste módulo adotou-se a mesma estratégia usada no Business para a criação de sua estrutura de dados, porém, os dados específicos para cada usuário são: Id, nome e amigos.

- Review: Assim como nos outros dois módulos, a estratégia de criação das estruturas de dados foi a mesma. Contudo, os dados específicos de cada avaliação foram: Id, id do usuário, id do negócio, estrelas, se era divertido, engraçado e/ou interessante, a data da avaliação e o texto. Diferente dos outros módulos, este possui os campos id do usuário e do negócio, logo, este era o principal meio para o cruzamento de informações.

Em todos os módulos é possível encontrar, além de seus próprios construtores, as funções auxiliares para a realização das queries. Funções estas que foram escritas lá para facilitar o acesso aos dados de cada módulo quando fosse necessário percorrer todas as estruturas.

Descrição das querys:

Query 1:

- **Status**: Concluída com sucesso.
- **Estratégia adotada**: Carregamento dos dados dos ficheiros através de fgets(leitura de cada linha do ficheiro) , strsep(filtro para os separadores de string, retornando a string no formato requerido) , atribuição dos valores para a struct respectiva e adição desta struct na árvore do respectivo catálogo .
- **Comentários**: Lentidão para carregamento de dados de ficheiros grandes, principalmente do catálogo users, o que já era considerado esperado dado o tamanho deste. Tempo de execução: 20 segundos.

Query 2:

- **Status**: Concluída com sucesso.
- **Estratégia adotada**: Percorrer a estrutura de dados de negócios, aqueles nomes que tiverem a primeira letra igual a letra solicitada pelo usuário, são guardados na estrutura TABLE.
- **Comentários**: Dificuldades no início para percorrer corretamente a estrutura de dados da árvore, após compreendida, foi resolvida facilmente. Tempo de execução: 0.05 segundos.

Query 3:

- **Status:** Concluída com sucesso.
- **Estratégia adotada** : Percorrer a estrutura de negócios e, o nodo em que tiver com mesmo valor de Id do business, salvamos na estrutura os seus dados como : nome, cidade e estado. Também percorremos a estrutura review para buscar os valores de stars e o número total de reviews que este ID fez.
- **Comentarios** : Tempo de execução: 0.36 segundos.

Query 4:

- **Status:** Concluída com sucesso.
- **Estratégia adotada** : Percorrer a estrutura de dados review e conferir o ID do users que fez negocios, então comparamos o user ID e o business ID. Guardamos tudo na estrutura TABLE. Depois usamos a estrutra AUX_STRUCT, que usamos para guardar o com o nome do negócio e o id e com isso fazer um print na tela as informações dos business ID e do nome do Business.
- **Comentarios** : Tempo de execução: 0.33 segundos.

Query 5:

- **Status**: Concluída com sucesso.
- **Estratégia adotada**: Percorrer a estrutura de dados de negócios, a procura dos nodos que possuem o mesmo valor para dada a determinada cidade. Caso seja encontrado, salva-se na estrutura TABLE o ID do nodo e o formato que será impresso (ID + nome do negócio). Após isto, é percorrido agora a estrutura de dados das reviews e monta-se uma lista auxiliar com os IDs dos reviews que possuem “n” ou mais estrelas. Tendo em mãos ambas as listas, usa-se uma função para cada ID na TABLE e checa se ele está na lista auxiliar. Quando estiver, significa que possui as estrelas desejadas e retira-se o ID mantendo apenas o formato para ser impresso, quando não, remove-o junto com o formato.

- **Comentários**: Dificuldade em cruzar as informações entre as estruturas de dados sem a lista auxiliar. Seja essa dificuldade por comparar os dados ou por percorrer a estrutura várias vezes. Tempo de execução: 0.65 segundos.

Query 6:

- **Status**: Não foi feita.
- **Comentários**: Primeiramente houve uma certa confusão em relação ao que era suposto fazer, e a primeira tentativa foi realizá-la de forma que era considerado os top n de todos os negócios desconsiderando o filtro por estado. Após a compreensão, percebeu-se que era mais complexo o problema, e era suposto realizá-la de forma mais sofisticada, então não nos restou tempo para resolvê-la posteriormente.
  Possibilidade de Estratégia: Na leitura dos ficheiros atribuir para cada Estado uma lista de Business, utilizando uma estrutura auxiliar como o hashTable. E após isto, calcular a média de cada business , e ordená-lo de forma decrescente, utilizando um min-heap(?).

Query 7:

- **Status**: Semi-concluída .
- **Estratégia adotada**: Recolher no catálogo review todos os ids dos usuários associado também ao id de negócio, através de um array de ponteiros do glib. Fazer sort desta lista a partir dos ids dos usuários, com a intenção de colocar os usuários repetidos juntos. Fazer um ciclo for, para verificar se o usuário visitou mais de um estado, incrementando um contador, e eliminando os usuário repetidos.
- **Comentários**: Funcionamento correto da função para quantidades menores de ciclos (usuários), mas muito ineficiente para a sua totalidade de usuários.
- **Possibilidade de otimização** : Usar uma estrutura de dados auxiliar na leitura do ficheiro review, associar para cada id user uma lista de ids business. Com isto iria eliminar users repetidos e não seria necessário percorrer a árvore do review. Usaria HashTable (?), associando a key como o user id , e seus valores os business id.

Query 8:

- **Status**: Não feita.
- **Possibilidade de estratégia**: Mudar a estrutura do catálogo business, no campo categories para um array de strings, guardando nela a lista de categorias sem vírgulas. Filtrar os negócios pela categoria solicitada. Determinar por ordem decrescente a média de cada negócio.

Query 9:

- **Status**: Concluída com sucesso.
- **Estratégia adotada**: Percorrer a estrutura de dados de reviews e acessar em cada nodo o campo de texto. Dentro do texto, procura-se a palavra desejada tomando a certeza que antes e depois da mesma só há espaços e/ou pontuações. Quando for encontrada a palavra é salvo na estrutura TABLE o ID deste review e impresso os primeiros "n" reviews que encontraram a mesma.
- **Comentários**:Além de usar a função "ispunct()" recomendada, foi usado também a "isspace()" com o intuito de garantir que estava sendo encontrado a própria palavra. Tempo de execução: 0.71 segundos.

Conclusões e desafios

Com o desenvolvimento deste projeto, ficamos cientes da dimensão e complexidade da ferramenta de leitura e armazenamento de dados. Diante desse desafio proposto, perdemos muito tempo a compreender e manusear estruturas de dados mais complexas.

A biblioteca Glib se mostrou muito útil nos tratamentos de dados com as suas estruturas já pré-definidas assim como suas funções. Contudo, no início do projeto tivemos uma certa dificuldade em compreender vossas ferramentas. Mas após a compreensão tornou-se muito pratico para a resolução dos problemas propostos do projeto.

Devido a perca de tempo nesta fase inicial do projeto, fomos limitados a não conseguir concluir o trabalho da forma que gostaríamos. Queríamos ter completado as querys restantes, otimizado funções, implementado a interface proposta no enunciado e a aplicação dos testes, mas pela complexidade do projeto e com o planejamento de tempo, não foi possível.

Concluímos que, o projeto nos proporcionou conhecimento acerca de estrutura de dados complexas que exigem armazenar uma grande quantidade de dados e saber manipulá-los.

Referências

- Biblioteca GLib: https://developer.gnome.org/glib/
- GTree: https://developer.gnome.org/glib/stable/glib-Balanced-Binary-Trees.html
- GPointerArray: https://developer.gnome.org/glib/stable/glib-Pointer-Arrays.html