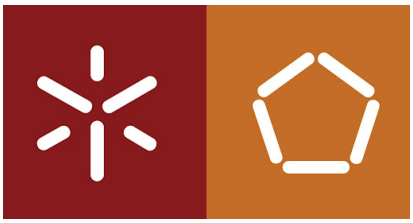

Universidade do Minho

Licenciatura em Engenharia Informática

Laboratórios de Informática III

Grupo 28

Daniel Azevedo (A93324) Rui Monteiro (A93179)
Rodrigo Rodrigues (A93201)

Ano Letivo 2021/2022

Capítulo 1

Exercício 1

Fazer a validação dos registos dos ficheiros de entrada (users.csv, commits.csv, repos.csv), i.e, descartar os registos que não respeitam os formatos dos campos e gerar ficheiros de saída com apenas os registos válidos (de acordo com as normas estabelecidas).

1.1 Users

Estratégia utilizada:

- Função *build_user* que constrói um GH_USER preenchendo a *structgh_user* com os dados obtidos na leitura de cada linha do ficheiro de entrada. Atribui o valor -1 ao campo id da *structgh_user* caso detete valores inválidos que não podem ser carregados para a estrutura.
- Função $userIsValid$ que verifica a validade de um GH_USER conforme os parâmetros estabelecidos. Verifica ainda se a função *build_user* indicou dados inválidos através do campo id da struct *gh_user*
- Função $add\_user2file$ que adiciona um user a um ficheiro com o formato tal como no ficheiro de entrada.
- Função *load_users* que lê o ficheiro dos Users, linha a linha, constrói os users e invoca a função userIsValid para averiguar a sua validade. Sendo válido, o user é adicionado ao ficheiro de saída.

1.2 Commits

- Função *build_commit* que constrói um GH_COMMIT preenchendo a *structgh_commit* com os dados obtidos na leitura de cada linha do ficheiro de entrada. Atribui o valor -1 aos campos relativos a $ID's$ da *structgh_commit* caso detete valores inválidos que não podem ser carregados para a estrutura.
- Função $commitIsValid$ que verifica a validade de um GH_COMMIT conforme os parâmetros estabelecidos. Verifica ainda se a função *build_commit* indicou dados inválidos através dos campos de id da struct *gh_commit*

- Função $add\_commit2file$ que adiciona um commit a um ficheiro com o formato tal como no ficheiro de entrada.
- Função *load_commits* que lê o ficheiro dos Commits, linha a linha, constrói os commits e invoca a função commitIsValid para averiguar a sua validade. Sendo válido, o commit é adicionado ao ficheiro de saída.

1.3 Repos

- Função *build_repos* que constrói um GH_REPOS preenchendo a *structgh_repos* com os dados obtidos na leitura de cada linha do ficheiro de entrada. Atribui o valor -1 aos campos relativos a $ID's$ da struct *gh_repos* caso detete valores inválidos que não podem ser carregados para a estrutura.
- Função $reposIsValid$ que verifica a validade de um GH_REPOS conforme os parâmetros estabelecidos. Verifica ainda se a função *build_repos* indicou dados inválidos através dos campos de id da struct *gh_repos*
- Função $add\_repos2file$ que adiciona um repositório a um ficheiro com o formato tal como no ficheiro de entrada.
- Função *load_repos* que lê o ficheiro dos Repos, linha a linha, constrói os *repos* e invoca a função reposIsValid para averiguar a sua validade. Sendo válido, o *repos* é adicionado ao ficheiro de saída.

Capítulo 2

Exercício 2

O Exercício 2 tem como objetivo efetuar o cruzamento de dados entre os novos ficheiros gerados no exercício 1, filtrando dados inválidos, de acordo com as filtragens estabelecidas.
Estratégia do grupo: utilizar estruturas de dados da GLib para armazenar $Users$, *Repos* e *commits*, para mais tarde aceder/remover *commits* e/ou *repositórios* consoante as filtragens estabelecidas.

2.1 Hashtables dos Users e dos Repositórios

Com o objetivo de armazenar os GH_USER, escolhemos como estrutura de dados a Hash Table, utilizando como chave o *id* do *user*. Escolhemos esta estrutura de dados por ser muito

eficiente nos acessos (acesso de tempo constante). Da mesma forma, foi ultilizada outra Hash Table para armazenar os GH_REPOS, utilizando como chave o *id* do *repos*.

2.2 Array dos Commits

Devido à inexistência de um *id* próprio dos *Commits*, escolhemos utilizar um Array para armazenar os *Commits*, que podemos iterar de forma a remover *Commits* conforme as regras de filtragem definidas.

2.2.1 Limitações

A utilização de um Array significa menos eficiência do que se teria com uma Hash Table.

2.3 Implementação

A função *exercicio*2() carrega as estruturas de dados construídas através da leitura dos novos ficheiros resultantes do exercicio-1 e procede a:

- Iterar pelo array dos commits e remover aqueles cujos *users* são inválidos (user_id committer_id) e remover aqueles cujo repositório não existe
- Consultar a Hash Table dos *repos* e remover os que referem *users* (owner_id) inválidos e remover os que não têm *commits*

2.4 Conclusão

A tarefa de fazer parsing de dados é essencial no desenvolvimento de um projeto em que é necessário processar ficheiros com milhões de dados. A utilização de estruturas de dados auxilia o armazenamento e eficiência dos dados para fácil acesso (inserir/remover). Pensamos ter concluído as tarefas exigidas para o guião-1 aplicando os conhecimentos de programação em C.