Relatório do Guião 1 do trabalho em C de LI3

Trabalho realizado por:

Hugo Dos Santos Martins a95125

João Bernardo Teixeira Escudeiro a96075

José Pedro Batista Fonte a91775

Introdução

•Este relatório é referente ao Guião 1, e nele são descritos os procedimentos realizados. Numa primeira fase pensamos em criar uma estrutura de dados para armazenar os diferentes campos relativos a cada um dos ficheiros, mas rapidamente percebemos que neste exercício não havia necessidade de armazenar as informações em memória.

Exercício 1

•Optamos por não criar nenhuma estrutura de dados em ambos os exercícios pois achamos que não havia necessidade. A estratégia utilizada na limpeza de cada um dos três ficheiros foi:

Users.csv

• O primeiro passo é abrir o ficheiro que está localizado na pasta de entradas. É criado um ciclo *while* (o primeiro dentro da *main*), que para quando a função "*fgets*" deixa de ler linhas, ou seja quando estamos numa linha vazia. A primeira linha que é o cabeçalho, é colocada de imediato no ficheiro resultado . À medida que cada uma das seguintes linhas são lidas, uma cópia dessas mesmas linhas é passada como argumento a uma outra função (*Verify_user*). Nesta função é onde todos os testes são feitos. Esta separa o conteúdo das diferentes colunas em parâmetros utilizando a função "*strtok*". Os testes que esta função realiza a cada um dos parâmetros são:

➔USER.ID -> Testa se é um número inteiro positivo;

➔USER.LOGIN -> Testa se é uma *string* válida, não vazia;

➔USER.TYPE -> Testa se é um dos três seguintes tipos: (BOT, ORGANIZATION, USER);

➔USER.CREATED_AT -> Testa se a data é válida (está entre 7 de abril de 2005 e a data atual) e se está no formato correto;

➔USER.FOLLOWERS -> Tem que ser um número inteiro positivo e esse número tem de coincidir com o tamanho da *follower_list*;

➔USER.FOLLOWING_LIST -> Tem que ser um número inteiro positivo e esse número tem de coincidir com o tamanho da *following_list*;

➔PUBLIC_GISTS -> Tem de ser um número inteiro maior ou igual a zero;

➔PUBLIC_REPOS-> Tem de sérum número inteiro maior ou igual a zero;

Caso os parâmetros não sejam válidos então as respetivas linhas são descartadas, caso todos sejam válidas então são colocadas no ficheiro final.

Repos.csv •À semelhança do que foi feito com o ficheiro de *users*, o mesmo foi feito com o ficheiro de *repos.* A diferença é que o ficheiro a utilizar é outro e os campos a verificar são diferentes . A primeira linha do ficheiro original é colocada de imediato no ficheiro final. Os seguintes Campos são testados na função "*verify_repos"*. O REPO.ID, OWNER.ID tem de ser um número inteiro positivo, FULL_NAME, LICENSE, DESCRIPTION, DEFAULT_BRANCH, LANGUAGE, tem de ser uma *string* válida, CREATED_AT, UPDATED_AT tem que ser uma data válida entre 7/04/2005 e a data atual e estar no formato correto, FORKS_COUNT, OPEN_ISSUES, STARGAZERS COUNT, SIZE tem de ser um número inteiro maior ou igual a zero. Se todos os parâmetros de uma dada linha passarem nos testes então esta linha é colocada no ficheiro resultado.

Commits.csv •Por fim este é o último ficheiro a ser tratado. À semelhança dos anteriores, existe um terceiro ciclo *while (terceiro da main)* que percorre o ficheiro original e testa os campos que necessitam de aprovação na função "*verify_commits*". Os parâmetros a ser testados nesta função são: REPO.ID/ AUTHOR.ID/ COMMITER.ID que têm que ser um número inteiro maior que zero, COMMIT.AT que tem que ser uma data válida entre 7/04/2005 e a data atual, e estar no formato correto e por último o parâmetro MESSAGE que tem de ser uma *string* válida.

Após todos estes testes e com os ficheiros finais encerrados com os seguintes nomes "users-ok.csv", "commits-ok.csv", "repos-ok.csv".

Exercício 2

•Nesta parte a tarefa mais desafiadora foi o entrelaçamento de dados entre os três ficheiros. A estratégia utilizada pelo grupo foi criar um *array* com tamanho igual ao máximo dito no enunciado (3 milhões) e apenas guardar o *user*.id de cada linha em cada posição do *array*. De seguida o *array* é ordenado com a função *mergesort*. A partir daí, começa o ciclo *while* que lê ,linha a linha o ficheiro de commits e verifica se o author.id e o *commiter*.id pertencem ao *array* de *users* através de uma procura binária (ferramenta utilizada pelo grupo para diminuir em grande parte o tempo de execução do programa) no array que tem os ids ordenados. Apenas se ambos os campos (*author* e *commiter*.id) pertencerem ao *array* de *users* essa linha é considerada válida e é colocada no ficheiro auxiliar que contém as linhas válidas de *repositórios*, mas ainda não é o final pois ainda vai ser alterado no futuro.

Posteriormente, e ainda com o *array* de *users* ordenado, é percorrido o ficheiro de *repos* da mesma forma que foi referida em cima, apenas o teste, entre parâmetros, é diferente. Neste caso é testado o parâmetro *repos_owner*. Em cada linha é verificado se o *repos_owner* pertence ao *array* de users_id e, caso pertença, a linha é colocada num ficheiro *csv* auxiliar, que ainda não é o final pois ainda irá ser alterado no futuro.

De seguida, é criado um *array* para os repositórios, da mesma forma que é para os *users*, mas este *array* é "carregado "com os ids dos repositórios e posteriormente ordenado. Com um novo ciclo while é percorrido o ficheiro de commits auxiliar resultante do ciclo anterior que já foi alterado relativamente aos users . Caso o parâmetro commits.repo.id pertença ao array de ids de repositórios essa linha é considerada válida e é colocada no ficheiro final de commits.

Por fim, é criado um array para os commits, que carrega o commits.repo.id e posteriormente ordenado. Com um ciclo while é lido o ficheiro de repos auxiliar que já foi alterado anteriormente quando comparado com os users e em cada linha é verificado se o repos.id tem alguma commit.repo.id associado. Caso tenha é validada e colocada no ficheiro de repos-final, se não tiver é descartada.

Conclusão

• Durante a realização do segundo exercício ocorreu algo um pouco estranho. O grupo optou por guardar todos os ids que seriam alvos de comparação em *arrays* na memória. Esta solução não se mostrou eficiente visto que o programa demorava cerca de uma hora a executar. Dado este problema, o grupo, decidiu optar por guardar unicamente, em memória, um array ordenado contendo apenas um dos campos passível de comparação e "correndo" o ficheiro linha a linha fazendo os devidos testes em cada linha. Após esta alteração o tempo melhorou significativamente e demora cerca de 10 segundos a executar a segunda parte.

Existem alguns erros no código, principalmente na limpeza do ficheiro *repos* no exercício 1.Surgiu uma anomalia devido à existência de um caractere estranho no fim da linha o que dificultou o processo e que fez com que os resultados fossem ligeiramente diferentes dos esperados. Na segunda parte também existe um erro , pois os resultados são diferentes ligeiramente dos esperados .Para a resolução destes problemas talvez devessem ser usadas outras bibliotecas de funções que não as ("strtok " ,"fputs" …) .Para a segunda parte talvez pudesse ser implementada uma lista ligada em vez de *arrays* para evitar alguns problemas de memória que foram surgindo à medida do trabalho.

O grupo concluiu que, apesar da existência de pequenas anomalias nos resultados, o objetivo geral foi cumprido.

NOTA: Um dos membros do grupo não conseguiu colocar o código por si produzido no repositório *git* , pois quando o grupo submeteu o código o membro a95125 Hugo dos Santos Martins apercebeu-se que o convite não tinha ficado aceite e que não tinha acesso ao repositório .Assim peço que não fique prejudicado em relação aos outros membros do grupo sendo que as tarefas foram realizadas ,de igual forma, por todos os membros .