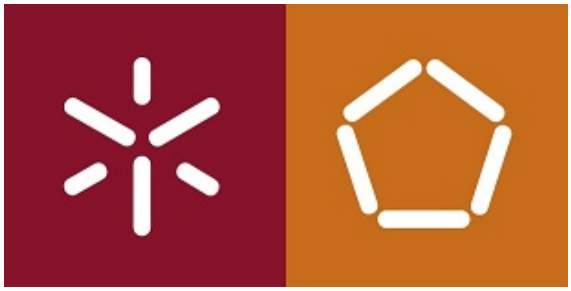

Universidade do Minho
Departamento de Informática

Github - Gestão de dados

LEI - Laboratórios de Informática 3

Grupo 9

13 de dezembro, 2021

Filipa Gomes
(a96556)

Ricardo Oliveira
(a96794)

João Loureiro
(a97257)

Introdução

Neste relatório descrevemos o nosso processo de desenvolvimento do guião 2, um sistema de gestão de usuários, commits e repositórios da plataforma Github, guardados em ficheiros *.csv*.

Exercicio

0.1 Estruturas de Dados

Para ser possível armazenar a informação de uma forma ordeira e organizada, foram criadas três estruturas principais de catálogos (tabelas de hash) de **Users**, **Repos** e **Commits**.

```
struct commit{
    int u_id;
    int a_id;
    int r_id;
    char *message;
    int date[3];
};

struct commit_entry{
    int id;
    Commit c;
    struct commit_entry *next;
};

struct catalog_c{
    CEntry commits[MAX_TABLE];
    int total;
};
```

*Exemplo do catálogo de commits.

0.2 Encapsulamento

Como pedido no guião, aplicou-se encapsulamento de dados. Para tal, foram criadas funções de *get_x* e *set_x*, de forma a que fosse possível aceder aos dados ao mesmo tempo que se cumpre o encapsulamento.

```

CEntry get_com_next (CEntry com) {
    return com->next;
}

int *get_com_date (Commit commit ) {
  int *dt=malloc(sizeof(int)*3);
  dt[0] = commit->date[0];
  dt[1] = commit->date[1];
  dt[2] = commit->date[2];
  return dt;
}

void set_com_next (CEntry com, CEntry cn) {
    com->next = cn;
}

void set_com_date (Commit commit, int *date) {
  commit->date[0] = date[0];
  commit->date[1] = date[1];
  commit->date[2] = date[2];
}
```

*Exemplo de *get_x* e *set_x* utilizadas para aceder aos dados relacionados com os catalogos de commits.

0.3 Importação de dados a partir de ficheiros

Foram criadas várias funções para importar os dados no projeto algumas delas são as funções *parse_hash_x*, onde x é o tipo a ser lido do ficheiro *.csv* e a ser colocado no catálogo respectivo.

```
int parse_hash_comm(CatalogC table,CatalogR r_table, CatalogU u_table, char *entrada,
    GStats gs){
  FILE *u = fopen(entrada, "r");
  char *buff = malloc(sizeof(char) * 500000);
  static int id = 300000;

  if (u == NULL){
    printf("Error when opening %s! \n", entrada);
    return 0;
  }else{fgets(buff, 500000, u);}

  while (fgets(buff, 500000, u)){
    char *strc;
    strc = strdup(buff);
    int r_id = verify_int(strtok(strc, ";"));
    int a_id = verify_int(strtok(NULL, ";"));
    int u_id = verify_int(strtok(NULL, ";"));
    char *cdt = strdup(strtok(NULL, ";"));
    char *message=strdup(strtok(NULL,";"));
    int j = 0, data[3];
    char *cd;
    cd=strtok(cdt,"- :");
    while (cd != NULL && j < 3)
    {
        data[j] = verify_int(cd);
        if (data[j] == -1)
            j = 3;
        j++;
        cd = strtok(NULL, "- :");
    }
```

*Codigo com omissões.

0.4 Estrutura de Estatísticas

Foi criada uma estrutura *GStats* que contém todos os campos necessários a guardar valores para a resolução das queries estatísticas. Inicializada toda a zeros, o preenchimento dos campos da estrutura *GStats*, é feito durante a leitura e parsing dos ficheiros.

```
struct global_stats {
    int t_users;
    int t_comms;
    int t_repos;
    int t_contribs;
    int u_type[3];
    int bot_repos;
};

GStats init_stats(){
    GStats s=malloc(sizeof(struct global_stats));
    s->t_users = 0;
    s->t_comms = 0;
    s->t_repos = 0;
    s->t_contribs = 0;
    s->bot_repos = 0;
    s->u_type[0] = 0;
    s->u_type[1] = 0;
    s->u_type[2] = 0;
```

```
    return s;
}
```

0.5 Queries Estatísticas

A resolução das queries estatíticas (1, 2, 3 e 4) é facilmente concretizada através do acesso dos campos da estrutura *GStats*. No caso da query dois, os colaboradores consideram-se como qualquer utilizador que seja author ou commiter id num commit de um dado repositório, podendo um dado utilizador contar várias vezes como colaborador, uma vez que, a média de colaboradores por repositorio simboliza o número de users que participam no mesmo, caso se excluissem utilizadores que já fossem colaboradores de outro repositorio essa média iria diminuir de forma irrealista, pois se esse repositorio só contesse esse user como colaborador seria representado como tendo apenas 0 colaboradores.

```
char * query_1 (GStats gs) {
    char *o = malloc(sizeof(char)*64);
    sprintf(o,"Bot : %d; Organization : %d; User : %d", get_type_n(gs,0), get_type_n(gs
    ,1), get_type_n(gs,2));
    return o;
}

char * query_2 (GStats gs) {
    char *o=malloc(sizeof(char)*8);
    double t_colab = get_t_contribs(gs);
    double t_repos = get_t_repos(gs);
    double x = (t_colab)/(t_repos);
    sprintf(o,"%0.2f", x);
    return o;
}
```

*Exemplo da query 1 e query 2.

0.6 Queries Parameterizadas

Houve algumas dificuldades em organizar o raciocinio de algumas queries(query 5 e 6), mas no final foi possível resolver. A dificuldade encontrava-se no tempo de execução, chegando a ser de 10 minutos ou mais. No caso da query 10, apesar de devolver commits com as maiores mensagens por repositório podem existir vários commits do mesmo user, uma vez que maior parte dos repositórios acaba por ter apenas um colaborador, sendo então mais útil retornar o top N, caso exista, das maiores mensagens e os seus users, independente de esses users poderem já ter aparecido. Foi criada uma query extra denominada de query 11, na qual é calculado o top N commits com maiores mensagens.

```
char *query_11(CatalogU cu, CatalogC cc, int n){
    char *o = malloc(sizeof(char)*(32*n));
    int k=0, j,id,sz,t1,t2;
    CEntry re;
    Commit r;
    int bc[2][n];

    for (int i = 0; i < MAX_TABLE; i++){
        re = get_com_entry(cc,i);
        while (re != NULL){
            j = 0;
            r = get_entry_commit(re);
            id=get_com_u_id(r);
            char * msg=get_com_message(r);
            sz=utf8_strlen(msg);
            while (j < k)
            {
                if (sz>bc[1][j]){
                    t1=bc[0][j];
                    t2=bc[1][j];
```

```
                    bc[0][j]=id;
                    bc[1][j]=sz;
                    id=t1;
                    sz=t2;
                }
                j++;
            }
            if (k < n){
                bc[0][k]=id;
                bc[1][k]=sz;
                k++;
            }
            re= get_com_next(re);
        }
    }

    for (int g = 0; g < k; g++)
    {

        if (g == 0)
        {
            char *temp = malloc(sizeof(char) * (32));
            sprintf(temp, "%d;%s;%d\n", bc[0][g], check_user_login_by_id(cu, bc[0][g]),
    bc[1][g] - 1);
            strcpy(o, temp); // get_user_coms(u) );
            free(temp);
        }
        else
        {
            char *temp = malloc(sizeof(char) * (32));
            sprintf(temp, "%d;%s;%d\n", bc[0][g], check_user_login_by_id(cu, bc[0][g]),
    bc[1][g] - 1);
            strcat(o, temp); // get_user_coms(u) );
            free(temp);
        }
    }
    return o;
}
```

0.7 Leitor de comandos

Para a possível leitura dos ficheiros de input com os comandos, foi criada uma função *do_query_from_line* que separa a linha por um espaço (indicador da query) e seleciona a query a implementar. No caso das queries parameterizadas é feita uma separação dos parametros.

```
char *do_query_from_line(char *line, CatalogU u, CatalogR r, CatalogC c, GStats s){
    char *token = strtok(line," ");
    int query = verify_int(token),n;
    char *out;
    switch (query){
        case 1:
        out = query_1(s);
        break;

        ...

        case 5:
        token=strtok(NULL," ");
        if(token!=NULL){
            n=verify_int(token);
            token = strtok(NULL, " ");
            if(token!=NULL){
                char *d1=strdup(token);
                token=strtok(NULL," ");
                if(token!=NULL){
                    char *d2 =strdup( token);
                    out = query_5(c, u, n, d1, d2);
                }else{out=strdup("Error Parsing Queue 5");}
```

```
            }else{out=strdup("Error Parsing Queue 5");}
        }else{out=strdup("Error Parsing Queue 5");}
        break;
```

*Exemplo da leitura da query 1 e 5.

0.8 Ouput

Após a execução das queries lidas no ficheiro de input, o seu output é redirecionado e concatenado para um ficheiro.

```
void interpete_query_file(char * f_name, CatalogU u, CatalogR r, CatalogC c,GStats s){
    FILE *f = fopen(f_name, "r");
    char *tok = strtok(f_name,".");
    char *buff = malloc(sizeof(char) * 500000);
    int n=1;
    if (f == NULL){
        printf("Error when opening %s! \n", f_name);
    }
    while (fgets(buff, 500000, f)){
        char *out = malloc(sizeof(char)*64);
        strcpy(out,tok);
        char *op=malloc(sizeof(char)*24);
        sprintf(op,"_command%d_output.txt",n);
        strcat(out,op);
        FILE *o = fopen(out,"w");
        char *line;
        char *temp = strdup(buff);
        line = do_query_from_line(temp,u,r,c,s);
        free(temp);
        if(line!=NULL)fprintf(o,"%s",line);
        fclose(o);
        n++;
    }
    free(buff);
    fclose(f);
}
```

0.9 Ficheiro com funções auxiliares

Criou-se um ficheiro com todas as funções auxiliares para a resolução das queries.

```
void build_dt(char *dt, int *data) {
    int j = 0;
    while (dt != NULL && j < 3)
    {
        data[j] = verify_int(dt);
        if (data[j] == -1)
            j = 3;
        j++;
        dt = strtok(NULL, "- :");
    }
}
```

*Exemplo da build_dt que por sua vez transforma uma string com a data num array de ints.

0.10 Testes

Para testar o output e o tempo de execução, criou-se uma pasta *testes* e adicionou-se ficheiros de teste para cada query e um ficheiro de teste global, no qual o tempo de execução era monitorizado através do comando time de modo a que fosse possível melhorar o mesmo. Ficheiro de teste global:

```
1
2
3
4
5 100 2015-10-10 2017-10-10
6 100 Java
7 2018-01-01
8 100 2017-01-01
9 5
10 50
11 100
```

Tempo de execução

```
real  0m7,220s
user  0m6,970s
sysn  0m0,240s
```

0.11 Desafios na resolução do Guião 2

Ao longo do desenvolvimento deste guião depara-mo-nos com vários desafios.

Primeiramente, o encapsulamento das estrutas de dados foi uma das tarefas mais trabalhosas, sendo que era a primeira vez que nos deparavamos com tal tarefa. Isto demorou algum tempo e criou algumas dúvidas entre nós sobre qual a melhor abordagem, porém, com a ajuda dispponivel sentimos que fomos capazes de ultrapassae essas dificuldades.

Posteriormente, ao desenvolver as queries, surgiram mais dificuldades, sendo, desta vez, o problema da eficiência do programa. Ao inicio, apesar de certas queries executarem, a nosso ver, extraordinariamente rápido, outras nem tanto, causando dificuldades no desenvolvimento de melhores estratégias. A maior mais valia, no entanto, vinha da fase de Parsing dos ficheiros, uma vez que, como foi dedicada uma grande parte do tempo ao desenvolvimento das estruturas, todos os dados necessários já estavam propicios às funcionalidades das querys. Nem tudo é bom com este método, uma análise mais detalhada na leitura e aramazenamento de dados no catálogo levou a um aumento do tempo de execução de querys singulares, isto é, ao executar apenas uma query o seu tempo tende a demorar cerca de 5s (para as querys estatisticas), no entanto, ao executar vários pedidos o tempo não aumenta linearmente, isto é, o custo está na leitura de dados, que permanecem na memória até ao final da execução. Dado isto, uma das estratégias, para o aumento da eficincia foi o "trade-off"entre o tempo de leitura de dados pela redução do tempo de execução das querys.

Por fim, foi necessário vários dias de trabalho árduo para que se pudesse melhorar e aperfeiçoar a resolução das queries, uma vez que, apenas o melhor tratamento de dados não é suficiente para diminuir os tempos de execução. Esta foi talvez a parte mais gratificante, pois, a cada melhoria do programa conseguia-mos ver o nosso trabalho refletido num melhor tempo. Porém, isso levou-nos a conhecer novas barreiras do nosso projeto, sendo elas a "duplicação de strings". A utilização da função 'strdup()', de modo a manter o encapsulamento, influencia as funções que retornam strings, uma vez que a operação de alocação de espaço pode ser demorada para grandes Data Sets.

0.12 Conclusão

O grupo sentiu que a elaboração deste guião os levou a melhorar o seu conhecimento e capacidades relativamente a programacao e desenvolvimento de programas seguindo os conceitos de modularidade e modulação, aplicar conhecimentos de outras cadeiras, como por exemplo hash tables, criando uma ligação entre cadeiras diferentes como por exemplo Algoritmos e Complexidade e Programação Imperativa também como direcionada a Objetos.