Relatório: Trabalho 2 de BD

Paulo Mateus Luza Alves - GRR20203945 Erick da Silva Santos - GRR20182554

1 Introdução

O trabalho consiste em um algorítimo de detecção de conflitos em um agendamento S utilizando 2 algorítimos, sendo eles o *teste de conflito por serialidade* e o *teste por visão equivalente*.

2 Estruturas de dados

`typedef struct operation {} operation_t; -> Representação de um operação no agendamento`

`typedef struct schedule {} schedule_t; -> Representação de um agendamento. (lista duplamente ligada)`

`typedef struct escalations {} escalations_t; -> Representação da lista de escalonamentos.`

`typedef struct graph {} graph_t; -> Grafo utilizando matriz de adjacências`

`typedef struct reads {} reads_t; -> Relação leitura <-> escrita do escalonamento`

3 Estrutura de arquivos

3.1 schedule.h

```
/* Aloca o espaço em memória para um agendamento */
schedule_t *alloc_schedule ();

/* Aloca o espaço em memória para uma operação */
operation_t *create_operation (int timestamp, int transaction_id, char op_type,char attribute);

/* Adiciona a operação na ponta do agendamento (lista duplamente ligada) */
void add_schedule_operation (schedule_t *schedule, operation_t *operation);

/* Realiza a leitura do arquivo de entrada, construindo o agendamento */
int read_input_schedule (schedule_t **schedule);

/* Pega todos os ids das transações do agendamento */
int get_ids (schedule_t *schedule);

/* Libera a memória alocada para o agendamento */
void destroy_schedule (schedule_t *schedule);
```

3.2 escalation.h

```
/* Aloca espaço para a lista de escalonamentos */
escalations_t *alloc_escalations ();

/* A partir do agendamento inicial, o separa nos escalonamentos. O processo é dado remanejando os
ponteiros das listas ligadas de acordo com o inicio e fim de cada escalonamento, o qual é dado pelos
commits*/
int separete_schedule (schedule_t *schedule, escalations_t *escalations);

/* Libera a memória alocada para a lista de escalonamentos */
void destroy_escalations (escalations_t *escalations);
```

3.3 graphs.h

```
/* Aloca espaço para o grafo */
graph_t *alloc_graph (int vertices_qtd);

/* Insere uma aresta da origem para o target */
void insert_edge (graph_t *graph, int origin_v, int target_v);

/* Verifica se existe ciclo no grafo */
int has_cycle (graph_t *graph);

/* Função auxiliar de verificação de ciclo */
int check_cycle (graph_t *graph, int *accessed_vertices, int actual);

/* Libera o espaço alocado para o grafo */
void destroy_graph (graph_t *graph);
```

3.4 serializable.h

```
/* Função que checa se o grafo é serializável. constrói o grafo de acordo com as regras do algorítimo
e ao fim verifica se existe ciclo no grafo resultante.*/
int check_serializable (schedule_t *schedule);
```

3.5 equivalentView.h

```
/* Função que checa as visões do agendamento em busca de uma equivalente. Armazena as ultimas escritas
e as relações de leitura <-> escrita e chama a função de geração de visões*/
int check_equivalence (schedule_t *schedule);

/* Função gera as permutações do array de ids, construindo as respectivas visões e comparando
com o escalonamento original*/
int verify_visions (schedule_t *schedule, int *ids, int *last_writes, reads_t *reads, int size,
int lower_id);

/* Função que, a partir do array de ids, gera uma visão na sequência dos ids lidos.*/
int generate_vision (schedule_t *schedule, schedule_t **vision, int *ids);

/* Função que troca a posição de dois ids do array entre si.*/
void swap (int *ids, int font, int target);

/* Função que pega todas as relações de leitura <-> escrita do escalonamento*/
void get_all_reads_relations (schedule_t *schedule, reads_t *reads);

/* Função que pega todas as ultimas escritas do escalonamento*/
void find_last_write (schedule_t *schedule, int *last_writes, int lower_id);
```

3.6 utils.h

```
/* Função que verifica se o numero indicado já esta incluso no array.*/
int includes (int number, int size, int *array);
```