Disciplina: CI1218 - Banco de Dados
Orientação: Profa. Dr. Eduardo Almeida
Equipe: Alessandra Souza da Silva GRR20182558 alessandrasilva@ufpr.br
Marisa Sel Franco GRR20186556 marisafranco@ufpr.br

TESTE DE SERIABILIDADE QUANTO AO CONFLITO E DE VISÃO EQUIVALENTE PARA DETECÇÃO DE CONFLITOS DE ESCALONAMENTO DE TRANSAÇÕES

Introdução

Este relatório tem por objetivo descrever brevemente detalhes de implementação, em C, dos algoritmos de teste de serialidade quanto ao conflito e de visão equivalente para detecção de conflitos de escalonamento de transações concorrentes em Sistema de Gerenciamento em Banco de Dados (SGBD). O trabalho foi desenvolvido nos cursos de Bacharelado em Ciência da Computação e Informática Biomédica, Setor de Ciências Exatas, da Universidade Federal do Paraná (UFPR), em cumprimento aos requisitos parciais da disciplina de Banco de Dados, sob orientação do prof. Dr. Eduardo Almeida.

O **algoritmo de teste de seriabilidade quanto ao conflito** garante que o resultado da execução concorrente de transações é mesmo que o serial. Se houver conflito, as transações não são executadas em paralelo pelo SGBD. Esse algoritmo usa um grafo para representar as transações do escalonamento. Seus passos são:

1. Crie um nó para cada T do escalonamento S;
2. Aresta T_i -> T_j para cada r(x) em T_j depois de w(x) em T_i;
3. Aresta T_i -> T_j para cada w(x) em T_j depois de r(x) em T_i;
4. Aresta T_i -> T_j para cada w(x) em T_j depois de w(x) em T_i;
5. S é serial se não existe ciclo no grafo formado.

Já o **algoritmo de visão equivalente** é menos restritivo do que o algoritmo de seriabilidade quanto ao conflito. Dessa forma, ele permite que alguns conflitos entre transações “passem”. Dois agendamentos S e S', sendo S’ uma das versões seriais de S, são ditos visão equivalentes se atenderem às seguintes condições:

1. O mesmo conjunto de transações e operações participam em S e S';
2. Para cada r(x) de T_i, se o valor de x lido já foi escrito por w(x) de T_j, ou seja, houve w(x) antes de r(x) em transações distintas, a mesma coisa deve acontecer para r(x) de T_i em S';
3. Se o operador w(y) em T_k é a última escrita de y em S, então w(y) em T_i deve ser a última escrita em S'.

Bibliotecas utilizadas

Na implementação, foram usadas apenas bibliotecas padrão da linguagem C: stdio.h, stdlib.h e string.h.

Compilação e execução

No terminal, execute “make” para compilar o programa. Para executá-lo, basta executar no terminal “./escalona < teste.in > teste.out”.

Entrada e saída

Conforme apresentado na especificação do trabalho, a **entrada** é feita pela entrada padrão (stdin). O arquivo é formado por uma sequência de linhas, em que cada linha representa uma transação chegando. Cada linha tem quatro campos: o primeiro é o tempo de chegada, o segundo é o identificador da transação, o terceiro é a operação (R=read, W=write, C=commit) e o quarto o atributo que será lido/escrito. Estas linhas estão ordenadas pelo primeiro campo (tempos menores no início indicando a linha do tempo).

A saída do programa também é feita pela saída padrão (stdout). O arquivo é composto por uma sequência de linhas, sendo uma linha para cada escalonamento. Cada linha tem quatro campos separados por espaço (um único espaço entre cada par de campos). O primeiro campo é o identificador do escalonamento. O segundo campo é a lista de transações. E o terceiro apresenta o resultado do algoritmo da garantia da seriabilidade, em que SS e NS significam respectivamente

serial (SS) ou não serial (NS). O quarto campo é o resultado do algoritmo de teste de equivalência de visão, em que SV e NV significam respectivamente equivalente (SV) ou não equivalente (NV).

Detalhes da implementação

Quanto às **estruturas de dados** utilizadas para implementação dos algoritmos, além de vetores padrão, foram criadas as seguintes estruturas:

Estrutura	O que armazena	Tipos de dados e componentes
transacao	Armazena os dados de uma etapa de uma transação: timestamp, id da transação, operação realizada e item de dados manipulado na operação	- int timestamp - int id_transacao - char operacao - char item
agendamento	Vetor que armazena um conjunto de transações de um agendamento	- typedef transacao * agendamento
estado	Armazena os números de pré e pós-ordem de um vértice v, obtidos por meio de uma busca em profundidade (DFS) em um grafo	- int pre - int pos
grafo	Armazena a representação de um grafo, cujos vértices são as transações de um escalonamento: número de vértices, matriz de adjacência para representar os arcos do grafo, vetor de estados dos vértices e vetor de ordenação topológica, formada somente caso haja ciclo(s) no grafo	- int n - int *matriz_adj - estado *estado - int *ordem_topologica

Após receber um agendamento com um ou mais escalonamento de transações, o programa conta o número de linhas do agendamento, o número de transações e demarca o ponto limite, ou seja, o final de cada escalonamento.

Para cada escalonamento identificado, é criado um **grafo de transações** utilizado para executar o algoritmo de teste de seriabilidade quanto ao conflito. A seguir, é feita uma **busca em profundidade (DFS)** no grafo direcionado gerado para identificar a presença de ciclos. Grafos com ciclos apontam que há conflito entre as transações e, portanto, o escalonamento não é serializável.

São listadas ainda as transações entre as quais há conflito e aquelas que não apresentam conflito para cada escalonamento.

Como o algoritmo de teste de seriabilidade quanto ao conflito é mais restritivo do que o algoritmo de visão equivalente, **caso** o grafo do escalonamento **NÃO tenha ciclos**, o escalonamento é considerado **serializável** quanto a conflito **e equivalente por visão**.

Caso contrário, o escalonamento é considerado **não serializável** quanto a conflito e o programa **avalia se existe um escalonamento serial S' que atenda às condições previstas** no algoritmo de visão equivalente **para que** o escalonamento **seja considerado equivalente por visão**.

Quanto à **modularização**, o programa está dividido da seguinte forma:

1. **“escalona.c”** - programa principal;
2. **“auxiliar.c” e “auxiliar.h”** - contêm as funções auxiliares para leitura da entrada e geração da saída;
3. **“agendamentos.c” e “agendamentos.h”** - contêm as funções usadas nas etapas dos algoritmos de teste de seriabilidade quanto ao conflito e de equivalência por visão, além das definições das estruturas de dados “transacao” e “agendamento”;
4. **“grafos.c” e “grafos.h”** - contêm as funções usadas na criação do grafo e na busca por ciclos, incluindo a DFS, além das definições das estruturas de dados “estado” e “grafo”.

Outros detalhes sobre a implementação e funções utilizadas podem ser vistos nos **comentários do código**.