# Escalonamento de transações

## Estrutura dos grafos

Os vértices são representados pela estrutura 'vertex'. Cada vértice ‘v’ tem um ponteiro ‘adj’ para seu primeiro vizinho de saída ‘w’, representado pela estrutura 'node'. O vizinho contém um ponteiro para o próximo vizinho de saída de ‘v’. O grafo é representado por uma estrutura ‘graph’ que contém um vetor de apontadores para cada vértice do grafo.

## Funções para grafos

O grafo é criado passando o número de vértices ‘n’. O vetor de apontadores de tamanho ‘n’ é alocado e os ponteiros para cada vértice são salvos nas posições do vetor. Para inserir um arco (v,w) no grafo um novo vizinho w (estrutura ‘node’) é criado e inserido na lista de vizinhos de v. Observe que a estrutura ‘node’ contém também um ponteiro para o vértice w.

## Checando se há ciclo no grafo

Para checar o se há ciclo no grafo utiliza-se o seguinte processo: é feita uma busca em profundidade no grafo e salvos em cada vértice o número de pós e pré-number. Para efetuar a busca em profundidade é utilizada uma pilha (no arquivo stack.c). Depois disso, todas os arcos do grafo são percorridas para procurar ciclos. Se em um arco (v,w) v.pos < w.pos w é ancestral de v, portanto há um caminho P de w a v. Se existe um arco (v,w) então o caminho P + vw forma um ciclo. Isso é feito na função ‘temCiclo’.

## Estrutura das transações e agendamentos

Uma transação Ti é representada pela estrutura ‘Transactions’, que contém um ponteiro para a próxima transação com o mesmo id ou null. O agendamento é representado pela estrutura ‘Agendamento’, que contém um vetor de ponteiros para transações. Duas formas de guardar a transação no agendamento são utilizadas:

1. Se é necessário separar em transações T1, T2...TN: o ponteiro para a primeira transação com id Ti é salvo no vetor de ponteiros do agendamento, e o Ti.next aponta para o próxima transação com o mesmo id.
2. Se é necessário guardar a ordem de chegada: o ponteiro para a primeira transação é alocado no vetor de transações, e seu atributo ‘next’ aponta para as próximas transações em ordem de chegada.

A função ‘transforma’ transforma o forma de guardar de 1 para 2.

## Algoritmo de teste de seriabilidade quanto ao conflito

O algoritmo teste por conflito varre a transação Ti à procura de uma operação de leitura ou escrita. Dependendo da operação encontrada varre o agendamento de T1...TN (exceto Ti) à procura de um r() ou de um w(), se encontrada esta segunda operação, e atender aos requisitos da figura 1, é criado um arco no grafo. Após criar os arcos ‘temCiclo’ verifica se há ciclo no grafo referente às transações.

**Algoritmo de teste de seriabilidade quanto ao conflito:**

- Crie um nó para cada T do escalonamento S
- Aresta $T_i$ -> $T_j$ para cada r(x) em $T_j$ depois de w(x) em $T_i$
- Aresta $T_i$ -> $T_j$ para cada w(x) em $T_j$ depois de r(x) em $T_i$
- Aresta $T_i$ -> $T_j$ para cada w(x) em $T_j$ depois de w(x) em $T_i$
- S é serial se não existe ciclo no grafo

*Figura 1: Teste de seriabilidade*

Obs: a entrada é um agendamento na forma 1, já que necessária a separação de transações.

## Algoritmo de visão equivalente

Para gerar as diferentes visões S’ seriais é usado um algoritmo recursivo que troca as posições das transações T1...Tn no vetor de ponteiros ‘agendamento.vetTransacao’. Como o agendamento está na forma 1, então ao permutar as

posições do vetor de transação, todas as transações de Ti permutam junto. Assim, basta colocar as transações em sequencia (na mesma lista) com a função ‘transforma’.

Para checar se a visão serial é equivalente quanto à última escrita das variáveis, a função ‘ultimaEscrita’ retorna uma lista de últimas atualizações de cada variável. Se as listas ultimaEscrita de S e S’ têm os mesmo elementos (não necessariamente na mesma ordem) então o requisito de última escrita é atendido. ‘comparaHistory’ faz essa comparação de listas de última escrita.

Para checar se a visão serial é equivalente quanto a relação ‘leitura e depois escrita’, a função ‘leituraEscrita’ varre as transações à procura de uma operação r(x). Se encontrada, varre as transações seguintes à procura de uma transação de escrita. O campo tempo da transação de leitura e escrita são guardados em uma estrutura ‘Dependency’, já que o tempo identifica a transação. Uma lista de relações leitura-escrita é retornada pela função. Se as listas leitura-escrita de S e S’ têm os mesmos elementos, então o requisito de manter as relações ‘leitura e depois escrita’ é atendido.

A função ‘visaoEquiv’ faz o teste de requisitos para cada permutação S’. É comparada a lista de últimas escritas do agendamento original S (transformado na forma 2) com a lista de últimas escritas do agendamento S’. Também compara-se as listas de leitura-escrita. Caso os requisitos sejam satisfeitos a função retorna 1 (o agendamento é equivalente).

Fernando Zanutto M. Barbosa 22/07/2021