Universidade Federal de Minas Gerais
Departamento de Ciência da Computação
Algoritmos e Estruturas de Dados III
1º Semestre de 2013

Trabalho Prático 1 - Sistema de Memória Virtual para Árvore B

Esse trabalho prático tem como objetivo a implementação de um Sistema de Memória Virtual (*SMV*), o qual será usado pela aplicação da *Árvore B*. Nesse trabalho o aluno deverá implementar uma versão simplificada de um SMV de modo a avaliar políticas de reposição de páginas (LRU, LFU e FIFO) em uma aplicação que estrutura seus dados em memória secundária na forma de *Árvore B*.

Problema

Árvores B são comumente utilizadas em aplicações que manipulam grande quantidade de dados (Ex. banco de dados e sistemas de arquivo), pelo fato de apresentarem complexidade temporal logarítmica em suas operações básicas (inserção,remoção e consulta). O espaço de memória primária, como sabemos, além de ser bem limitado, é endereçado diretamente pelo processador. Sendo assim, ela é a memória visível para as aplicações. Por causa disto, existem dois grandes problemas: quando as aplicações necessitam de um espaço maior do que o disponível em memória primária e quando existem múltiplas aplicações sendo executadas ao mesmo tempo (Compartilhando memória). Para resolver este problema, *Sistemas de Memória Virtual* são criados. Eles utilizam estratégias de gerenciamento de memória primária, no intuito de abstrair para as aplicações, uma memória suficiente para todos os seus dados [Fig. 1].

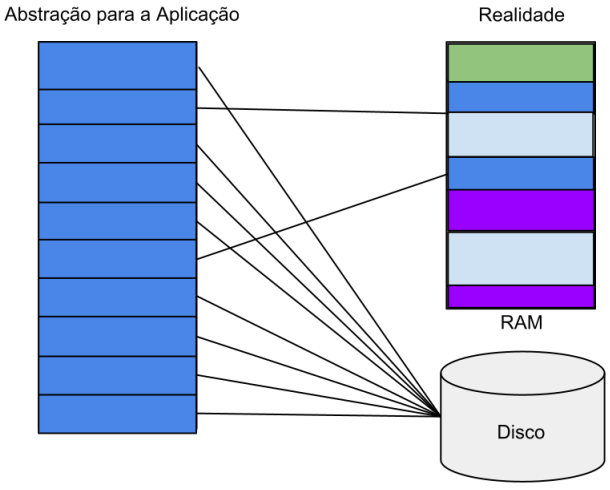

Figura 1: Abstração para a aplicação

O *SMV* a ser implementado, consiste em uma versão simplificada, que deve ser alocada dinamicamente de acordo com os parâmetros da entrada. A memória primária deve ser abstraída

de forma a armazenar estruturas do tipo *página*, que nada mais são do que blocos indivisíveis de dados. De forma geral, o funcionamento de um *SMV*, pode ser resumido por: Quando um registro de dados é requerido por uma aplicação, descobre-se em qual *página* o registro reside e verifica se a *página* se encontra em memória primária. Em caso positivo, acontece um *hit*, em caso negativo, ocorre um *page miss* e a *página* é carregada para a memória. Caso toda a memória esteja alocada, uma *página* é escolhida para ser substituída, seguindo uma política de reposição. Neste trabalho prático, três políticas devem ser implementas e comparadas:

- **FIFO (FirstIn, FirstOut)** - A página que está residente a mais tempo é escolhida para remoção.
- **LRU (Least Recently Used)** - A página acessada a mais tempo deve ser escolhida para remoção.
- **LFU (Least Frequently Used)** - A página com a menor quantidade de acessos deve ser escolhida para remoção.

No intuito de contextualizar sua implementação de *SMV*, uma aplicação deverá ser simulada. Tal aplicação será representada pela forma a qual seus dados são organizados em memória secundária. Neste trabalho prático, pede-se que esses dados sejam estruturados em forma de *Árvore B*, que são árvores balanceadas e desenvolvidas para otimizar o acesso ao armazenamento secundário. As *árvores B* são organizadas por nós, tais como os das árvores binárias de busca, mas estes apresentam um conjunto de chaves maior do que um e são usualmente chamados de páginas. As chaves em cada página são, no momento da inserção, ordenadas de forma crescente e para cada chave há dois endereços para páginas filhas, sendo que, o endereço à esquerda é para uma página filha com um conjunto de chaves menor e o à direita para uma página filha com um conjunto de chaves maior. O conceito de ordem (M) da árvore também deve ser esclarecido, defini-se ordem como sendo o número mínimo de chaves que uma página pode conter, com exceção da raiz. Diz-se que, em uma *Árvore B* de ordem M cada nó deve conter no mínimo M registros e no máximo $2M$, além disto, cada nó deve conter $2M + 1$ apontadores e no mínimo $M + 1$ descendentes e no máximo $2M + 1$. Um exemplo de *Árvore B* de ordem 2, pode ser visto na figura 2.

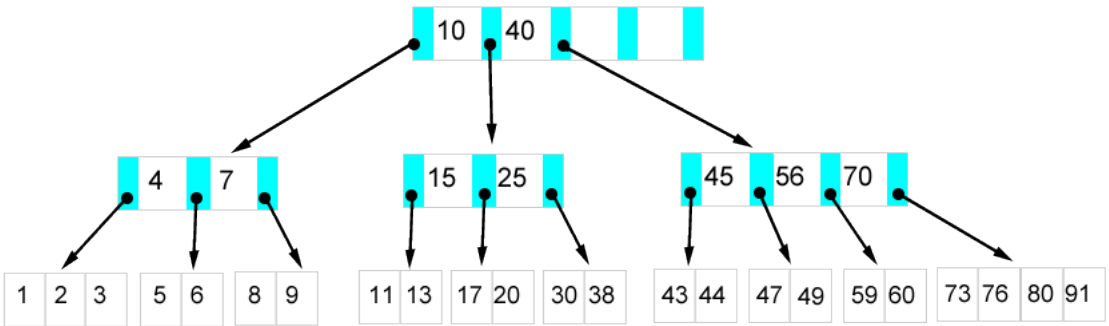

Figura 2: Árvore B de ordem $m = 2$

Como dito anteriormente, cada nó da árvore será tratado como uma página e sua estrutura deve conter um inteiro para informar o número de registros guardados, um vetor de inteiros de tamanho $2M$ e um vetor de apontadores para estrutura *pagina* de tamanho $2M+1$. Como sabemos

o tamanho em bytes dos apontadores em linguagem C se diferem de acordo com a arquitetura utilizada. Ponteiros em arquiteturas 32 bits são de tamanho 4 bytes, enquanto em arquiteturas 64 bits são o dobro. Portanto, utilizem uma *struct* para definir sua *página* e aloquem as páginas utilizando o comando *sizeof(pagina)*. A implementação da árvore pode ser encontrada no livro **Projeto de Algoritmos** [Zivianni 2010 cap. 6.3.1].

De maneira geral, em uma arquitetura de 32 bits, uma página de uma árvore de ordem 2, ocupará 40 bytes (2*2*4 + 2*2*4 + 4 + 4), pois apontadores e inteiros ocupam 4 bytes. Também considere que o tamanho da memória física é sempre um múltiplo do tamanho da página e que no início da simulação a memória primária está vazia, portanto qualquer acesso provoca uma falha.

Lembre-se que a consulta de qual *página* é responsável por um registro deve ser feita consultando a *árvore B*. Cada nó encontrado no caminhamento deve ser imediatamente carregado para a memória primária, respeitando as políticas de reposições exigidas. Isto se deve, pois precisamos avaliar as chaves contidas em cada nó durante a pesquisa do registro. Para fins de simplificação, consideramos que a página do SMV equivale em tamanho a uma página/nó da árvore B.

Entrada e Saída

O programa deverá solucionar múltiplas instâncias do problema em uma única execução. Serão repassados na entrada de dados informações para o SMV e os valores para a construção da Árvore B, valores para exclusão e consulta, além do limite de consultas a serem listadas no arquivo de saída. A saída será dada pelo número de falhas obtidas nas políticas implementadas, após a execução das consultas, além do caminhamento na Árvore de um número limitado de consultas. A entrada será lida de um arquivo e o resultado do programa deve ser impresso em outro arquivo de saída. Ambos arquivos devem ser passados por parâmetro na chamada do executável:

./tp1 input.txt output.txt

O arquivo de entrada possui um inteiro K na primeira linha onde K é o número de instâncias a serem simuladas. Em seguida, as K instâncias são definidas da seguinte forma. A primeira linha possui dois inteiros, o tamanho em bytes da memória física e um inteiro m indicando a ordem da Árvore B (lembre-se de que a memória primaria deve ao menos sustentar uma página), separados por espaço. A linha seguinte possui um inteiro N indicando quantos valores distintos serão inseridos na Árvore B ($1 \leq N \leq 100.000$). A próximas N linha possui os valores inteiros para cada item a ser inserido na Árvore, separados por espaço. Após isto, haverá uma nova linha contendo um inteiro M, que representa o número de itens a serem excluídos da Árvore B. A próxima M linha possui os valores distintos a serem excluídos separados por espaço. Em seguida, haverá outra linha contendo o número de inteiros a serem consultados na Árvore B. A linha seguinte possui a lista separada por espaço dos valores a serem consultados ($N \geq 1$). A próxima linha conterá o número S de consultas a serem exibidas no arquivo de saída e a linha seguinte os S inteiros separados por espaço.

Para cada instância, deve ser impresso no arquivo de saída, o número de falhas obtidas ao final da consulta, de acordo com as políticas FIFO, LRU e LFU, respectivamente, separados por espaço. O caminhamento de cada inteiro S, cujos valores carregados em memória devem ser separados por espaço. Considera-se que todos os itens pesquisados existam na Árvore B. O número de falhas sempre será maior do que zero, uma vez que a memória primária estará vazia ao ser iniciada a consulta.

Exemplo

A seguir temos um exemplo de funcionamento do programa para uma arquitetura de 32 bits:

Entrada:

```
1
80 2
18
10 5 7 20 9 13 18 32 15 38 40 8 60 27 17 12 37 25
2
32 20
9
15 25 40 8 7 12 37 8 13
2
40 37
```

Saída:

```
11 8 8
9 13 18 38 40 60
9 13 18 38 25 27 37
```

A Árvore B gerada após a inserção dos 18 inteiros pode se observada na Figura 3.

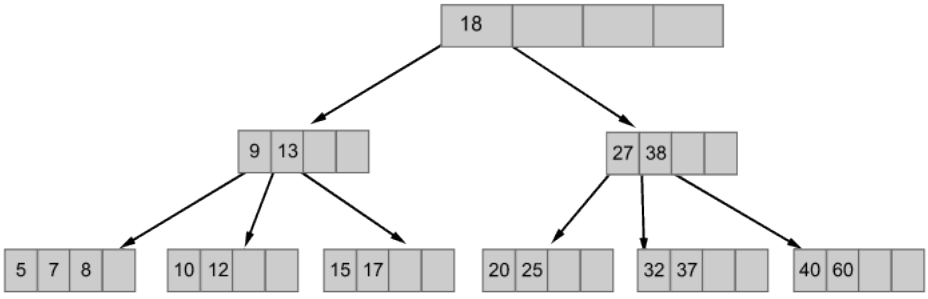

Figura 3: Exemplo da Árvore B após a inserção

Após a remoção dos itens indicados (32 e 20), a Árvore B estará com a configuração vista na Figura 4.

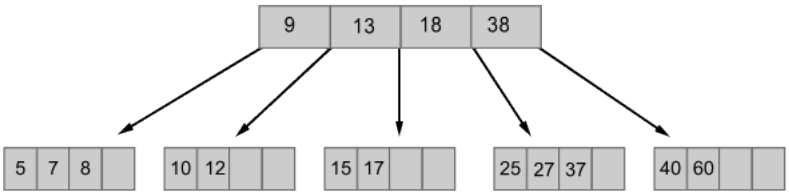

Figura 4: Exemplo da Árvore B após a remoção

Entrega

- A data de entrega desse trabalho é **22 de Abril**.
- A penalização por atraso obedece à seguinte fórmula $2^{d-1}/0.32\%$, onde d são os dias úteis de atraso.
- Submeta apenas um arquivo chamado <numero_matricula>_<nome>.zip. Não utilize espaços no nome do arquivo. Ao invés disso utilize o caractere '_'.
- Não inclua arquivos compilados ou gerados por IDEs. **Apenas** os arquivos abaixos devem estar presentes no arquivo zip.
  - Makefile
  - Arquivos fonte (*.c e *.h)
  - Documentacao.pdf
- Não inclua **nenhuma pasta**. Coloque todos os arquivos na raiz do zip.
- Siga rigorosamente o formato do arquivo de saida descrito na especificação. Tome cuidado com whitespaces e formatação dos dados de saída
- NÃO SERÁ NECESSÁRIO ENTREGAR DOCUMENTAÇÃO IMPRESSA!
- Será adotada **média harmônica** entre as notas da **documentação e da execução**, o que implica que a nota final será 0 se uma das partes não for apresentada.

Documentação

A documentação não deve exceder 10 páginas e deve conter pelo menos os seguintes itens:

- Uma **introdução** do problema em questão.
- **Modelagem e solução proposta** para o problema. O algoritmo deve ser explicado de forma clara, possivelmente através de pseudo-código e esquemas ilustrativos.
- **Análise de complexidade** de tempo e espaço da solução implementada.
- **Experimentos** variando-se o tamanho da entrada e quaisquer outros parâmetros que afetem significavelmente a execução.
  - Deverá ser contabilizada a quantidade de falhas de leitura para cada uma das políticas, além da sua relação com a quantidade de memória disponível, deve-se variar o tamanho do bloco da memória RAM e analisar o impacto na execução da Árvore B. Recomenda-se utilizar a proporção de 25%, 50% e 75% da memória utilizada.
  - Espera-se gráficos que explore a Localidade de Referência Espacial e Temporal para o problema da Árvore B variando as instâncias e suas respectivas análises.
- Especificação da(s) **máquina(s) utilizada(s)** nos experimentos realizados.
- Uma breve **conclusão** do trabalho implementado.

Código

- O código deve ser obrigatoriamente escrito na **linguagem C**. Ele deve compilar e executar corretamente nas máquinas Linux dos laboratórios de graduação.
- O utilitário ***make*** deve ser utilizado para auxiliar a compilação, um arquivo *Makefile* deve portanto ser incluído no código submetido.
- As estruturas de dados devem ser **alocadas dinamicamente** e o código deve ser **modularizado** (divisão em múltiplos arquivos fonte e uso de arquivos cabeçalho .h)
- **Variáveis globais** devem ser evitadas.
- Parte da correção poderá ser feita de forma automatizada, portanto **siga rigorosamente os padrões de saída especificados**, caso contrário sua nota pode ser prejudicada.
- O arquivo executável deve ser chamado tp0.
- **Legibilidade e boas práticas** de programação serão avaliadas.