MAC–IME–USP

Carlos Eduardo Ferreira
Sala 297A Tel.: 818 6140
e-mail cef@ime.usp.br

MAC 323 – Estruturas de Dados

Primeiro semestre de 1999

Exercício-Programa 4 – Entrega: **23 de junho de 1999**

B-árvores

1 Para os alunos matriculados em MAC 211

No terceiro EP de MAC 211 vocês implementaram o básico para um pequeno gerenciador de bancos de dados. Entretanto dois operadores eram particularmente ineficientes: o `exactMatchQuery` e a `RangeQuery`. O objetivo deste exercício-programa é implementar estes operadores eficientemente (para algum campo indexado do seu banco de dados), utilizando para isso uma árvore 2-3 (B-árvore de ordem 2).

2 Para os alunos que não fazem MAC 211

Em um sistema operacional o gerenciamento da memória é um dos aspectos mais fundamentais, e organizá-lo de forma eficiente é muito importante para o desempenho do sistema. Nesse exercício–programa vamos implementar um sistema de gerenciamento de memória. O objetivo **não** é mostrar a estrutura utilizada nos sistemas mais modernos (e, de fato, a estrutura mostrada não é a mais utilizada, como vocês vão aprender no ano que vem). Bons algoritmos de gerenciamento de memória são tema de pesquisa interessante, envolvendo problemas como partilhamento da memória, coleta de lixo, etc. (veja o capítulo 12 do livro [1]).

A seguir descrevemos o esquema de gerenciamento de memória a ser implementado.

O sistema será de multiprocessamento, ou seja, diversos processos compartilharão a memória. O sistema não é capaz de manipular pedaços não contíguos de memória. Assim, se um processo deseja uma quantidade de memória que não está disponível no momento, fica suspenso até que seja possível alocar uma área daquele tamanho. Os tamanhos dos blocos requisitados podem variar (são estabelecidos pelos processos).

A entrada do seu programa deverá ser um arquivo com o seguinte formato. Na primeira linha é dada a quantidade inicial de bytes disponíveis. A seguir, vêm diversas linhas, e em cada uma delas, um pedido da seguinte forma:

```
R n
D n
```

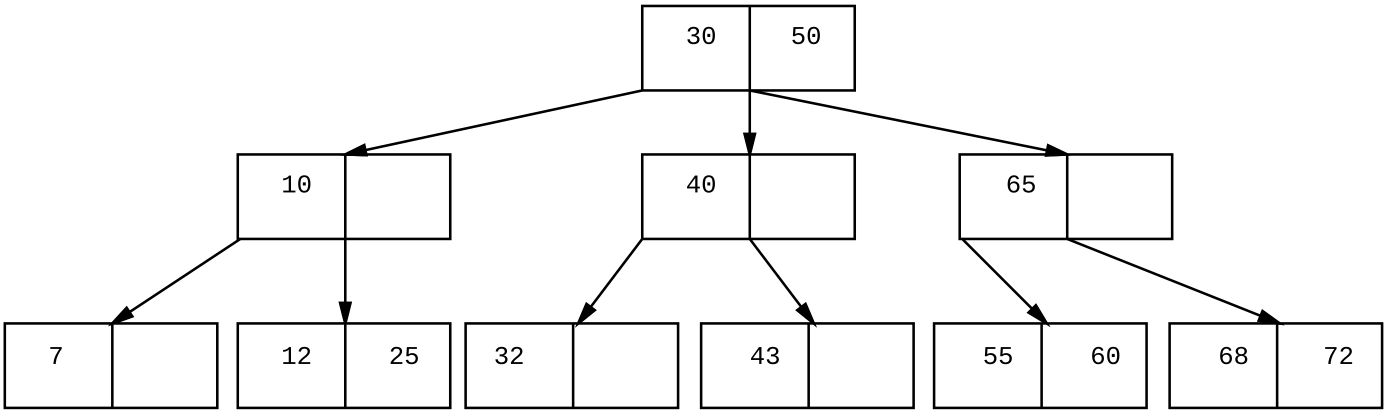
Se o pedido foi um 'R' o processo está alocando os n Kbytes (onde n é um número inteiro), se for um 'D', ele está devolvendo.

Os pedaços disponíveis de memória estarão armazenados em uma árvore 2-3 (uma B-árvore em que armazenamos no máximo 2 elementos por nó), em que a chave será o tamanho em Kbytes (ou MBytes) do segmento (no caso em que dois segmentos têm o mesmo tamanho, use o menor endereço de início do segmento para desempate). Assim, cada nó da árvore 2-3 terá o seguinte formato:

P_1	K_1	B_1	P_2	K_2	B_2	P_3

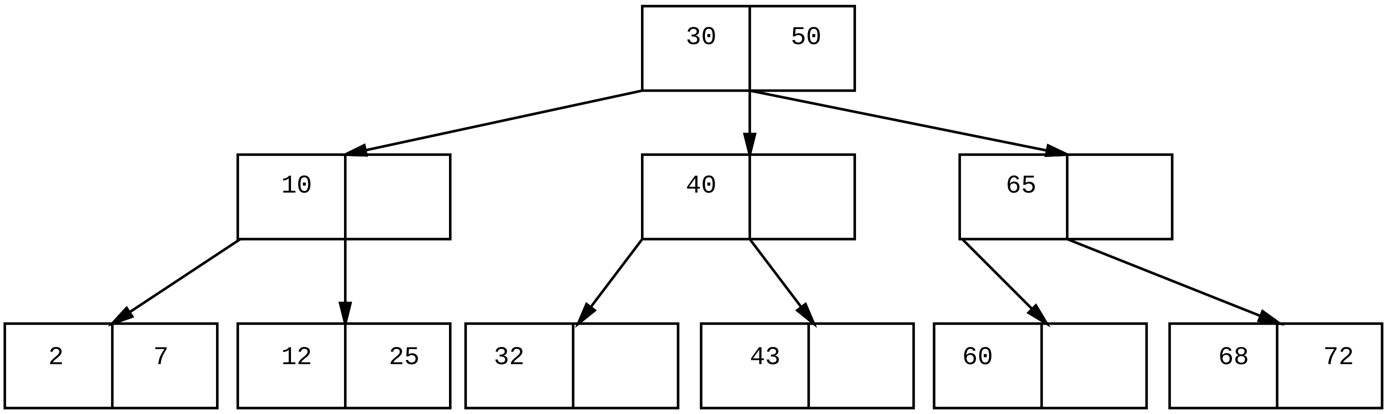
- P_1, P_2 e P_3 são ponteiros para os nós filhos;
- K_1 e K_2 são as chaves deste nó (tamanhos de segmentos contíguos de memória disponíveis);
- B_1 e B_2 são as posições de memória em que estes segmentos iniciam.

Observe o exemplo abaixo.



Quando requisitamos uma porção de memória, que é possível de se satisfazer no momento, uma remoção e, em geral, uma inserção serão necessárias. Veja o exemplo a seguir, em que alocamos o bloco requerido do menor bloco disponível maior que o pedido feito.

Ex.: Usando a árvore do exemplo anterior, pedimos um bloco de 53 kbytes.



Quando um bloco é devolvido, em geral é muito ineficiente tentar juntá-lo com blocos vizinhos, pois isso acarreta remoções e inserções na árvore. Assim, faremos esse rearranjo apenas quando tivermos algum pedido que não pôde ser satisfeito.

Seu exercício–programa pode implementar outras estratégias para escolha do bloco de que alocar a memória requisitada, e pode utilizar a estrutura que desejar para facilitar o rearranjo de memória. Você deverá fazer um pequeno relatório mostrando o desempenho do seu sistema para os testes que você fez (por exemplo, quantas vezes foi necessário fazer o rearranjo, quantos processos ficaram suspensos esperando memória, etc). Durante a execução emita mensagens informando ao usuário qual o estado do gerenciador.

Referências

[1] A.V. Aho, J.E. Hopcroft, J.D. Ullman, *Data Structures and Algorithms*, Addison-Wesley (1985).