Algoritmos e Estruturas de Dados II
2º semestre de 2017 - Turma 03
Lista de exercícios

1. Explique por que o tempo de acesso a um disco rígido (HD) é significativamente maior do que o tempo de acesso à memória principal (RAM).
2. Explique como pode ser decomposto o tempo de acesso a um disco rígido.
3. Por que a leitura de setores contíguos em um disco é mais eficiente do que a leitura de setores aleatórios, mesmo que o volume total de dados transferidos seja o mesmo?
4. Descreva como é implementada a alocação de arquivos em sistemas de arquivos baseados em:
   a) alocação contínua.
   b) alocação encadeada.
   c) alocação indexada.
5. Cite e explique vantagens e limitações de cada uma das formas de alocação de arquivos descritas na questão 4.
6. A forma de acesso (sequencial ou aleatória) que estará disponível para um arquivo armazenado em um HD irá depender tanto das características do dispositivo físico, quanto do modo de alocação implementado pelo sistema de arquivos. Esta afirmativa está correta? Justifique.
7. Quais as vantagens e desvantagens de se usar registros de tamanho fixo para organizar os dados em um arquivo?
8. Quais as vantagens e desvantagens de se usar registros de tamanho variável para organizar os dados em um arquivo?
9. Considere um arquivo composto por 40000 blocos de 4096 bytes (tamanho dos blocos usados nas transferências entre disco e memória RAM) que armazena registros (de tamanho fixo) com 256 bytes. Assumindo que as funções de leitura/escrita apenas são capazes de ler/escrever blocos inteiros, e que o cursor de leitura/escrita apenas pode ser posicionado para o primeiro byte de cada bloco, responda (justificando suas respostas):
   a) Qual o tamanho total (em bytes) do arquivo?
   b) Qual o número total de registros armazenados no arquivo?
   c) Qual o número total de acessos (quantidade de blocos lidos e escritos) a disco necessários para ordenar tal arquivo usando o *Keysorting*? Assuma a "versão completa" do *Keysorting* vista em aula, isto é, aquela que gera um arquivo de saída contendo os registros ordenados.

d) Qual o número total de acessos (quantidade de blocos lidos e escritos) a disco necessários para gerar um índice denso para tal arquivo? Assuma que o índice denso cabe integralmente em memória.

e) Qual será tamanho (em bytes) do arquivo de índice denso gerado?

f) Quantos acessos a disco serão necessários para buscar um registo do arquivo usando o índice denso? Assuma que o índice denso cabe integralmente em memória e que já está carregado antes da busca.

g) Quantos acessos a disco serão necessários para gerar um índice esparso em que cada entrada aponte para a primeira entrada de cada bloco do índice denso?

h) Qual será o tamanho (em bytes) do índice esparso gerado?

i) Quantos acessos a disco serão necessários para buscar um registro do arquivo usando o índice esparso? Assuma que o índice esparso cabe integralmente em memória e que já está carregado antes da busca.

j) Assumindo que um máximo de 60000 registros podem ser mantidos em memória, qual será o número de acessos a disco necessários para ordenar o arquivo com o *K-way merge*? Quantos arquivos intermediários serão gerados durante o processo de ordenação? Qual o tamanho (em bytes) destes arquivos intermediários?

10. Assumindo que uma biblioteca para manipulação de arquivos disponibiliza as seguintes funções:

```
#define TAMANHO_BLOCO 4096

// abre o arquivo para leitura e/ou escrita
Arquivo * abre(char * nome_arquivo);

// fecha um arquivo aberto
void fecha(Arquivo * a);

// reposiciona o cursor de leitura/escrita para um determinado bloco do arquivo
void procura(Arquivo * a, int num_bloco);

// lê o próximo bloco de bytes do arquivo, a partir da posição atual do cursor.
// Os bytes lidos são armazenados em buffer e ao término da leitura o cursor
// avança para o início do próximo bloco.
void le_bloco(Arquivo * a, unsiged char * buffer);

// escreve os bytes armazenados em buffer para o bloco do arquivo apontado pelo cursor.
// Ao término da escrita o cursor avança para o início do próximo bloco.
void escreve_bloco(Arquivo * a, unsigned char * buffer);
```

e que seu programa esteja manipulando arquivos de registros declarados da seguinte forma:

```
typedef struct {
  int id;
  unsigned char conteudo[128 - sizeof(int)];
} Registro;
```

Implemente as seguintes funções:

a) `Registro * le_registro(Arquivo * a, int indice_do_registro);`

b) `void escreve_registro(Arquivo * a, int indice_do_registro, Registro * r);`

c) `void imprime_registros(Arquivo * a);`

d) `void copia_arquivo(Arquivo * origem, Arquivo * destino);`

e) `void ordena_arquivo(Arquivo * origem, Arquivo * destino);`

f) `void gera_indice_denso(Arquivo * a, Arquivo * arq_indice); // indexado por id`

g) `Registro * busca_registro(Arquivo * a, Arquivo * arq_indice, int id);`