Estrutura de Dados II - 2020/EARTE

Trabalho Prático T3

19 de novembro de 2020

Leia atentamente **todo** esse documento de especificação. Certifique-se de que você entendeu tudo que está escrito aqui. Havendo dúvidas ou problemas, fale com o professor o quanto antes. As dúvidas podem ser sanadas usando o fórum de dúvidas do AVA.

1 Objetivo

O objetivo deste trabalho é implementar um índice, utilizando uma Árvore B em memória principal, para acesso rápido a registros armazenados em um arquivo binário.

2 Historinha de motivação e Problema

Todo trabalho precisa de uma historinha de motivação. Então vamos tentar...
Você acabou de ser contratado/a em uma empresa e sua tarefa é melhorar o sistema de cadastro utilizado pelos funcionários. Atualmente, os operadores do sistema estão reclamando que algumas operações estão demorando demais para executar e que esse tempo está impactando a eficiência da empresa. Após algum tempo, você descobriu que:

- O problema está no arquivo (ou tabela) que armazena informações de clientes. Esse é um arquivo binário e você não poderá modificar a forma como ele é organizado.
- O arquivo em questão pode ser visto como uma coleção de registros e cada registro armazena as informações de um único cliente.
- Para cada cliente, é armazenado no arquivo: um identificador único, composto por letras (maiúsculas e/ou minúsculas) e números; e uma sequência de caracteres, também alfanumérica, que representa os dados do cliente.
- Cada registro também tem uma porção referente a metadados: a quantidade de bytes utilizada pelos dados úteis do registro; e um indicador binário que informa que o registro deve ou não ser considerado como presente na base de dados. A Figura 1 representa a organização de cada registro. Um registro nunca pode ter mais de 4096 bytes (contanto dados e metadados).
- Você também descobriu que toda vez que um novo registro é escrito no arquivo, as informações são escritas seguindo o padrão do código abaixo:

```
void write_record(char *identificador, char *valor, FILE *f) {
    int n = strlen(identificador) + strlen(valor) + 1;
    int flag = 1;
    char v = ',';

    fwrite(&n, 4, 1, f);          // 4 bytes com valor de 'n'
    fwrite(&flag, 1, 1, f);       // Esse registro e valido
    fwrite(identificador, 1, strlen(identificador), f); // Escreve identificador
    fwrite(&v, 1, 1, f);          // Escreve a virgula
    fwrite(valor, 1, strlen(valor), f); // Escreve o valor
}
```

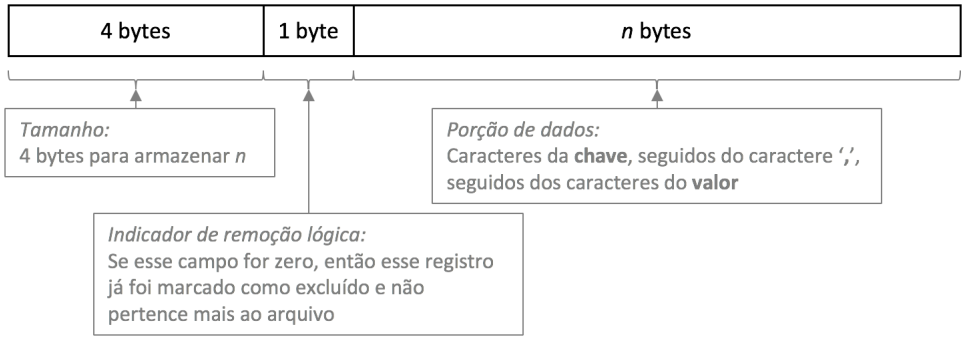

Figura 1: Organização de um registro no arquivo. Repare que o caractere '`,`' (vírgula) é utilizado para separar o *identificador* de seu *valor* associado dentro do registro. Suponha por exemplo que um registro tenha chave `ed2` e valor `nota100`, então a sequência de bytes (em hexadecimal) representando esse registro seria `0b 00 00 00 01 65 64 32 2c 6e 6f 74 61 31 30 30`. Os quatro primeiros bytes, `0b 00 00 00`, indicam que a porção de dados do registro tem 11 bytes; o byte seguinte, `01`, indica que o registro não foi removido do arquivo; os três próximos bytes, `65 64 32`, representam a *string* `ed2`; o byte seguinte, `2c`, representa o caractere '`,`'; e por fim, os próximos 7 bytes, `6e 6f 74 61 31 30 30`, representam a *string* `nota100`.

Após conversar com sua chefia, vocês decidiram que sua tarefa será implementar um índice para esse arquivo. Em outras palavras, você deverá implementar uma tabela de símbolos que mapeie os **identificadores** dos clientes para as **posições no arquivo em que os respectivos registros começam**. Dessa forma, para acessar as informações de um cliente, é necessário apenas que busque-se o cliente no índice (i.e., tabela de símbolos), descubra-se em qual parte do arquivo o registro correspondente se encontra e, por fim, efetuar a leitura dessa porção do arquivo. Além disso, ficou decidido que, devido a urgência da tarefa, você poderia inicialmente fazer esse índice em memória principal.

2.1 Algoritmo

Dada a problemática da seção anterior, sua tarefa é implementar o índice e algumas operações sobre o arquivo. Para implementar o índice, vocês devem usar uma Árvore B. Geralmente, esse tipo de estrutura de dados é implementado em arquivos, mas no nosso caso, nós vamos simplificar e permitir que ela fique em memória principal. Vocês podem (e eu recomendo que o façam) se basear na implementação disponível no livro texto (*Algorithms in C, 3rd edition, Sedgewick*).
Além do índice, vocês devem implementar as seguintes operações de forma eficiente (i.e., usando o índice):

1. `GET <chave>`: Dado um identificador de cliente, recuperar as informações no arquivo sobre esse cliente; se for um registro ativo, apresentar as informações na tela; se o registro não existir no índice ou estiver marcado como inativo no arquivo, não mostrar nada na tela. O formato de saída deve ser (sem o '.' no final) `identificador: valor`.

2. `DELETE <chave>`: Dado um identificador de cliente, remover esse cliente do arquivo. Nesse caso, a remoção no arquivo será lógica e não deve alterar o índice. Mais especificamente, se o identificador não estiver no índice, nada deve ser feito; caso o identificador esteja no índice e esteja marcado como inativo no arquivo, nada deve ser feito; caso o identificador esteja no índice e marcado como ativo no arquivo, marque-o como inativo (veja que o registro não é removido do arquivo, apenas o quinto byte do registro é modificado para o valor 0 – por isso o nome remoção lógica).

   Repare que você não precisa implementar a operação de remoção da árvore B. Essa tarefa é mais complicada.

3. `INSERT <chave> <valor>`: Insere um cliente no arquivo. Veja que há vários casos a considerar:

   - O identificador do cliente não está no índice: insira o cliente no final do arquivo e atualize o índice;
   - O identificador do cliente está no índice, mas o registro correspondente no arquivo está marcado como inativo: insira o cliente no final do arquivo e atualize o índice;

- O identificador do cliente está no índice e o registro correspondente no arquivo está marcado como ativo: marque o registro correspondente no arquivo como inativo; escreva o novo registro no final do arquivo; e atualize o índice para apontar para o novo registro.

4. `RUNDOWN <arquivo_saida>`: Cria um arquivo de texto com a informação de todos os clientes ativos da empresa. Os seguintes requisitos devem ser satisfeitos:

   - cada linha do arquivo deverá ter as informações de um registro, seguindo o formato (sem o ';' no final) `identificador: valor`;
   - as linhas do arquivo deverão estar ordenadas de acordo os identificadores. Veja que essa tarefa pode ser feita de forma fácil, uma vez que um índice com árvore B permite operações ordenadas sobre as chaves.

2.2 Exemplo ilustrativo

Vamos considerar um exemplo para deixar as coisas mais claras. Inicialmente, o arquivo tem o conteúdo sobre os clientes listados na Tabela 1 e a ordem de inserção é a ordem da tabela.

Tabela 1: Conteúdo do cadastro de clientes

Identificador	Valor	Ativo no cadastro?
ed2	facil	sim
paa	deboa	sim
tc	suave	não
cd	jafoi	sim

A sequência de bytes (em representação hexadecimal) do arquivo referente ao conteúdo da Tabela 1 é dada na Figura 2.

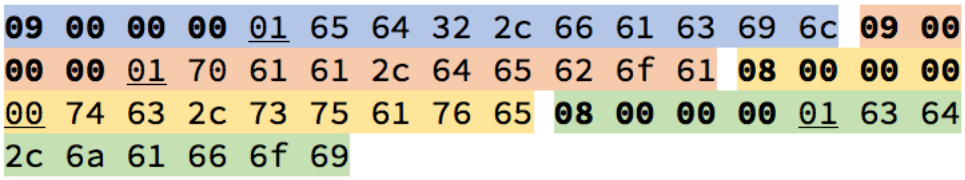

Figura 2: Bytes (em hexadecimal) do arquivo com conteúdo da Tabela 1. Pegue um café, uma tabela ASCII e analise com calma. Cores meramente ilustrativas para identificar os registros diferentes; em negrito, a parte dos metadados que identifica o tamanho da porção de dados de cada registro; sublinhado, o byte que indica se o registro está ativo ou não no cadastro.

O índice, em memória principal, deve ter conteúdo (mas em uma Árvore B) de acordo com a Figura 3.

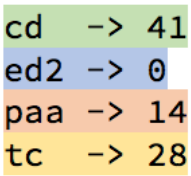

Figura 3: O registro com chave `ed2` começa no byte `0`; o registro com chave `paa` começa no byte `14`...

Se nesse arquivo fizermos a operação "`GET ed2`", o resultado será uma linha contendo a *string* "`ed2: facil`". Se fizermos a operação "`GET tc`" ou "`GET pp`", nada será impresso na tela.
A Figura 4 apresenta o estado do arquivo após executar a operação "`DELETE ed2`". Já a Figura 5 mostra o resultado após a operação "`INSERT es hard`". Por fim, a Figura 6 contém o estado do índice e arquivo após a operação "`INSERT ed2 quasela`".

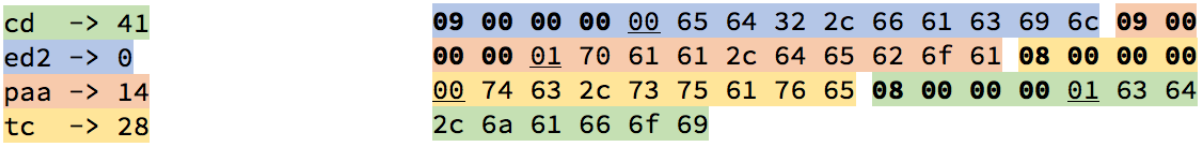

Figura 4: Índice (à esquerda) e conteúdo do arquivo (à direita) após remover a chave `ed2`. Veja que a única coisa que mudou foi o quinto byte do registro em azul, que passou a ser 0.

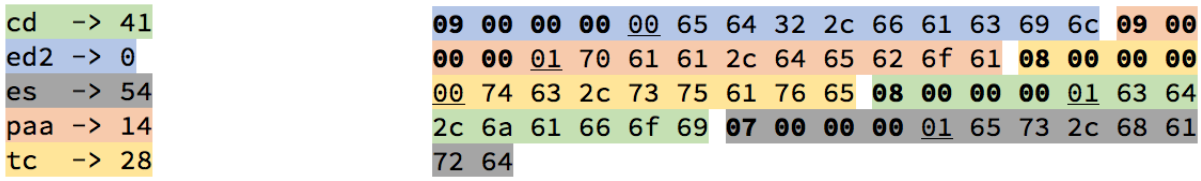

Figura 5: Índice (à esquerda) e conteúdo do arquivo (à direita) após inserir a chave `es` com o valor `hard`. Veja que o novo registro foi inserido no fim do arquivo e que o índice foi atualizado.

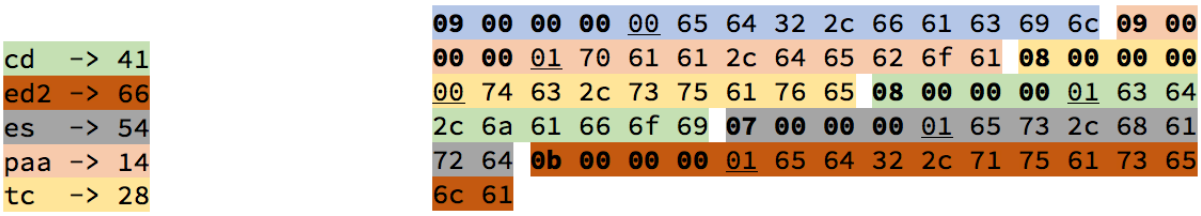

Figura 6: Índice (à esquerda) e conteúdo do arquivo (à direita) após inserir a chave `ed2` com o valor `quasela`. Veja que o novo registro foi inserido no fim do arquivo e que o índice foi atualizado.

Se executarmos o comando "`RUNDOWN out.txt`", o arquivo de texto `out.txt` será criado com o seguinte conteúdo:

```
cd: jafoi
ed2: quasela
es: hard
paa: deboa
```

3 Execução do trabalho, Entrada e Saída

Para testar seu trabalho, o professor executará comandos seguindo o seguinte padrão.

```
tar -xzvf <nome_arquivo>.tar.gz
make
./trab3 M <arquivo>
```

Onde:

- `M` é o parâmetro M da árvore B, como discutido na aula e apresentado no livro texto;
- `<arquivo>` é o nome do arquivo binário que possui as informações do cadastro. Há dois casos:
  - Se o arquivo não existir, um novo arquivo binário deverá ser criado. Nesse ponto, um índice vazio deverá também ser criado.
  - Se o arquivo já existir, o arquivo deverá ser lido (registro por registro) e as chaves de todos os registros válidos deverão ser inseridas no índice (Árvore B).

Após esse momento, seu programa deve entrar em laço esperando comandos do usuário. Os comandos válidos são `GET`, `INSERT`, `DELETE` e `RUNDOWN` assim como previamente explicados no texto. Quando o usuário digitar `STOP`, o programa deve terminar.
Observação 1: vocês podem assumir que as linhas de comando estarão bem formatadas e corretas.
Observação 2: sigam rigorosamente o padrão de entrada e saída apresentado nessa especificação. Havendo dúvidas, postem no AVA.

4 Detalhes de implementação

A seguir, alguns detalhes, comentários e dicas sobre a implementação. Muita atenção aos usuários do Sistema Operacional Windows.

- o trabalho deve ser implementado em C. A versão do C a ser considerada é a presente no Sistema Operacional Ubuntu 18.04.
- o caractere de nova linha será o `\n`.
- Seu programa deve ser, obrigatoriamente, compilado com o utilitário `make`. Crie um arquivo `Makefile` que gera como executável para o seu programa um arquivo de nome `trab3`.
- A linguagem C possui algumas funções que podem ser úteis na leitura dos arquivos. Em especial, sugere-se o estudo cuidadoso das funções `fseek`, `ftell`, `fwrite` e `fread`. Consulte da documentação da biblioteca `stdio.h` para mais detalhes sobre a manipulação de arquivos de texto e arquivos binários.
- A meta do trabalho é ser eficiente. Evite fazer muitas leituras de blocos pequenos de dados. Organize seu código para sempre (tentar) ler blocos de 4096 bytes (tamanho máximo de um registro).
- Trabalhar com arquivos binários pode ser complicado. O utilitário `xxd` do Linux permite fazer a conversão dos bytes para hexadecimal, o que torna a depuração mais simples.
- Ao longo do desenvolvimento do trabalho, certifique-se que o seu código não está vazando memória testando-o com o `valgrind`. Não espere terminar o código para usar o `valgrind`, incorpore-o no seu ciclo de desenvolvimento. Ele é uma ferramenta excelente para se detectar erros sutis de acesso à memória que são muito comuns em C. Idealmente o seu programa deve sempre executar sem nenhum erro no `valgrind`.

5 Regras para desenvolvimento e entrega do trabalho

- **Data da Entrega**: O trabalho deve ser entregue até as 23:59h do dia 10/12/2020. Não serão aceitos trabalhos após essa data.
- **Grupo**: O trabalho pode ser feito em grupos de até três pessoas.
- **Como entregar**: Pela atividade criada no AVA. Envie um arquivo compactado, no formato `.tar.gz`, com todo o seu trabalho. A sua submissão deve incluir todos os arquivos de código e um `Makefile`, como especificado anteriormente. **Somente uma pessoa do grupo deve enviar o trabalho no AVA. Coloque a matrícula de todos integrantes do grupo (separadas por vírgula) no nome do arquivo do trabalho.**
- **Recomendações**: Modularize o seu código adequadamente. Crie códigos claros e organizados. Utilize um estilo de programação consistente. Comente o seu código extensivamente. Não deixe para começar o trabalho na última hora.

6 Relatório de resultados

Esse trabalho não demandará entrega de relatório.

7 Avaliação

- O uso da primitiva **goto** e variáveis globais não são permitidos.
- Assim como especificado no plano de ensino, o trabalho vale 10 pontos.
- A parte de implementação será avaliada de acordo com a fração e tipos de casos de teste que seu trabalho for capaz de resolver de forma correta. Casos *pequenos* e *médios* (5 pontos) serão utilizados para aferir se seu trabalho está correto. Casos *grandes* (3 pontos) serão utilizados para testar a eficiência do seu trabalho. Casos *muito grandes* (2 pontos) serão utilizados para testar se seu trabalho foi desenvolvido com muito cuidado e tendo eficiência máxima como objetivo. Todos os casos de teste serão projetados para serem executados em poucos minutos (no máximo 2) em uma máquina com 16GB de RAM.
- Trabalhos com erros de compilação receberão nota zero.
- Trabalhos que gerem *segmentation fault* para algum dos casos de teste disponibilizados no AVA serão severamente penalizados na nota.
- Trabalhos com *memory leak* (vazamento de memória) sofrerão desconto na nota.
- Organização do código e comentários valem nota. Trabalhos confusos e sem explicação sofrerão desconto na nota.
- Caso seja detectado cópia (entre alunos ou da Internet), todos os envolvidos receberão nota zero. Caso as pessoas envolvidas em suspeita de cópia discordem da nota, amplo direito de argumentação e defesa será concedido. Neste caso, as regras estabelecidas nas resoluções da UFES serão seguidas.
- A critério do professor, poderão ser realizadas entrevistas com os alunos, sobre o conteúdo do trabalho entregue. Caso algum aluno seja convocado para uma entrevista, a nota do trabalho será dependente do desempenho na entrevista. (Vide item sobre cópia, acima.)