Trabalho Prático 1: Biblioteca do Filipe

Algoritmos e Estruturas de Dados III – 2016/2
Entrega: 26/09/2016

1 Introdução

Filipe é um bibliotecário famoso do ICEx. Ex-aluno de Ciência da Computação, ele deseja manter todo o registro de livros do acervo de forma que, quando chegue um pedido de livro, ele possa saber em pouco tempo se o livro está disponível ou não. Caso esteja disponível, Filipe quer saber também em qual estante o livro se encontra, para poder indicá-lo ao aluno.

Cada livro da biblioteca possui um título, e pode estar emprestado ou não. Normalmente, bibliotecas empregam taxonomias para organizar os livros em estantes. Filipe, porém, decidiu simplificar as coisas e ordenar todos os livros em ordem lexicográfica dos títulos, independente do tema do livro. A biblioteca possui E estantes idênticas, numeradas de 1 a E, cada uma com capacidade para guardar L livros. Filipe quer ordenar os livros pelo título, de forma que os primeiros L livros estejam na primeira estante, os próximos L na segunda estante, e assim por diante. Além disso, os livros em cada estante também devem estar ordenados, para que Filipe possa fazer busca binária para localizar um livro. Caso Filipe consiga colocar todos os livros em K estantes, com $K < E$, ele usará apenas as K primeiras, deixando as demais $E - K$ vazias.

Porém, antes de poder aplicar busca binária, Filipe precisa saber em qual estante está o livro. Para isto, ele construirá um *índice* para lhe ajudar. O índice conterá E registros, um para cada estante. O i-ésimo registro do índice contém o título do primeiro e do último livros da estante i. Com isto, para saber se um livro está em uma estante, Filipe apenas precisa verificar se o nome está entre os dois nomes do registro correspondente àquela estante.

Filipe precisa de um programa para lhe ajudar, já que são muitos livros na biblioteca. Por sorte, ele pode usar seu computador de trabalho. Porém, os computadores da biblioteca não são muito potentes. Mais precisamente, eles apenas podem armazenar os dados de M livros em memória ao mesmo tempo. Em disco, seu computador possui um arquivo com dados de todos

os N livros da biblioteca: o título, e se está atualmente emprestado ou não. Basicamente, as tarefas que Filipe deseja realizar são:

- Ordenar seu arquivo com todos os N livros, sem ultrapassar o limite M de livros que podem estar em memória ao mesmo tempo;
- Separar o arquivo em E arquivos, um para cada estante (com isso, Filipe vai fisicamente organizar os livros de acordo com este arquivo);
- Construir um arquivo com seu *índice*: um registro para cada estante contendo os nomes do primeiro e último livros;
- Responder consultas de alunos que pedem um livro, podendo dizer:

  O livro está na posição P na estante X caso o livro esteja disponível no acervo e seja o P-ésimo livro na estante X depois da ordenação que foi feita,

  O livro está emprestado caso a biblioteca possua o livro, mas ele esteja atualmente emprestado, ou

  A biblioteca não possui este livro caso a biblioteca não possua o livro.

Desde que começou em seu novo emprego, Filipe não tem programado muito. Assim, ele pediu a sua ajuda para realizar essas tarefas.

2 Entrada e saída

A entrada é a padrão do C, stdin, não é necessário abertura de arquivos para leitura da entrada.

Entrada A primeira linha da entrada é composta por cinco números inteiros: N, M, E, L e K, que correspondem ao número de livros da biblioteca, o número de livros que podem estar em memória, o número de estantes, o número de livros suportados por cada estante e o número de consultas a serem feitas pelos alunos, respectivamente.

A entrada segue com N linhas, cada uma com o nome de um livro que a biblioteca possui, cada linha termina com um número inteiro. Os nomes dos livros são sempre escritos no alfabeto latino do padrão ISO que consiste em 26 letras *a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z* e *underlines*, todas as letras minúsculas. Também é garantido que o titulo de um livro não tem mais que 50 caracteres. O número inteiro após o nome do

livro indica se o livro está ou não disponível: o número *1* indica que o livro está disponivel na biblioteca e o número *0* indica que o livro está atualmente emprestado.

A entrada termina então com K linhas, cada uma com o nome de um livro que os alunos do ICEx querem pegar. Os nomes dos livros seguem as mesmas restrições mencionadas acima (apenas letras minúsculas e *underscores*, com no máximo 50 caracteres).

Exemplo de entrada

```
10 2 4 5 3
projeto_de_algoritmos 0
algoritmos_teoria_e_pratica 0
kleinberg_and_tardos_algorithm_design 1
data_mining_and_analysis 0
analise_real_funcoes_de_uma_variavel 1
harry_potter 1
os_elementos 0
cplusplus_para_leigos 1
a_origem_das_especies 1
o_universo_numa_casca_de_noz 0
data_mining_and_analysis
o_grande_projeto
cplusplus_para_leigos
```

Saída Para este trabalho, são necessários três resultados de saída.

1. E arquivos, cada um contendo os livros da biblioteca ordenados. Os arquivos devem ser nomeados como *"estanteX"*, onde X representa qual estante é representada, com número de 0 a E - 1. Filipe é formado em Computação: ele começa a contar de 0. Estes arquivos devem ser binários, para permitir que a consulta seja feita eficientemente. Isto será melhor explicado no exemplo de saída dado logo abaixo.

2. Um arquivo de texto, nomeado como *"indice"*. Esse arquivo deverá ter E linhas, cada uma contendo os nomes do primeiro e ultimo livros de cada estante, separados por um espaço.

3. Imprimir na saída padrão (em tela) o resultado de cada consulta feita pelos alunos, um em cada linha, que pode ter os seguintes formatos:

    - **disponivel na posicao P na estante X** (sem acento na palavra "disponível") caso a biblioteca tenha o livro e ele não está empres-

tado, onde P é a posição do livro na estante (começando de 0) e X é a estante que o livro está,

- **emprestado** caso faça parte do acervo o livro, mas este está emprestado; ou
- **livro nao encontrado** caso o livro não faça parte do acervo da biblioteca.

Exemplo de saída
Para a entrada passada na seção 2, teremos a execução:

1. Os arquivos serão ordenamos usando algum método de ordenação externa. O resultado da ordenação deverá ser colocado no arquivo de nome **livros_ordenados**, que terá o seguinte conteúdo:

```
a_origem_das_especies 1
algoritmos_teoria_e_pratica 0
analise_real_funcoes_de_uma_variavel 1
cplusplus_para_leigos 1
data_mining_and_analysis 0
harry_potter 1
kleinberg_and_tardos_algorithm_design 1
o_universo_numa_casca_de_noz 0
os_elementos 0
projeto_de_algoritmos 0
```

   É importante notar que deve estar salvo em um arquivo, uma vez que não cabem tantos livros em memória dadas as restrições do problema ($M = 2$).

2. Separar os livros nas estantes. Vemos que a entrada informa que temos 4 estantes, cada uma suportando 5 livros. Como temos apenas 10 livros para alocar, só usaremos as duas primeiras estantes.

   Os arquivos que representam as estantes ficarão da seguinte forma:

   - Arquivo **estante0**

```
a_origem_das_especies 1
algoritmos_teoria_e_pratica 0
analise_real_funcoes_de_uma_variavel 1
cplusplus_para_leigos 1
data_mining_and_analysis 0
```

- Arquivo **estante1**

```
harry_potter 1
kleinberg_and_tardos_algorithm_design 1
o_universo_numa_casca_de_noz 0
os_elementos 0
projeto_de_algoritmos 0
```

- Arquivo **estante2**: arquivo vazio.
- Arquivo **estante3**: arquivo vazio.

Porém, cada livro deverá ocupar o mesmo número de bytes no arquivo; ou seja, serão arquivos binários com registros de tamanho fixo, um para cada livro. Isto é necessário porque cada consulta dos alunos deve ser respondida com busca binária. Para aplicar busca binária, você precisa conseguir acessar o livro em um índice i em $O(1)$ (ou seja, sem percorrer todos os livros anteriores). Porém, se o arquivo for dirigido por linhas, isto não é possível, já que você não consegue saber onde a primeira linha acabará antes de ler toda a linha (o mesmo para as demais, mas usamos o exemplo de saber onde começa a segunda linha para exemplificar).

Cada registro deve então ter o mesmo tamanho. Como sabemos que o nome de todo livro terá no máximo 50 caracteres (porque o enunciado garante isto), podemos representar cada livro com a seguinte estrutura em C:

```
typedef struct Livro {
  // 50 + 1 por causa do caractere '\0'
  // que termina o titulo.
  char titulo[51];
   // Um byte para dizer se esta disponivel ou nao.
  char disponivel;
} Livro;
```

Assim, cada livro será representado por 52 bytes. Este é apenas um exemplo de como representar os livros; a única restrição é de que o tamanho deverá ser fixo.

Isto será feito através das funções de entrada e saída binária em arquivos em C. Em particular, destacamos as seguintes funções que poderão ser usadas:

fwrite(P, T, R, F) escreve R registros de tamanho T bytes no arquivo F, estando todos em sequencia no ponteiro P. Por exemplo, se temos um arquivo de nome "arquivo", podemos escrever um livro nele da seguinte forma:

```
Livro l;
// ... le titulo e flag disponivel em l

// note que a flag de abertura b indica que
// o arquivo e' binario.
FILE *f = fopen("arquivo", "wb");
// escreve o registro l no arquivo F.
fwrite(&l, sizeof(l), 1, f);
```

fread(P, T, R, F) lê R registros de tamanho T bytes do arquivo F, estando todos em sequencia no ponteiro P. Por exemplo, se temos um arquivo "arquivo"que foi escrito da forma descrita acima, podemos ler 10 livros dele da seguinte forma:

```
Livro l[10];

// note que a flag de abertura b indica que
// o arquivo e' binario.
FILE *f = fopen("arquivo", "rb");
// le 10 registros do arquivo no array l.
fread(l, sizeof(Livro), 10, f);
```

fseek(F, O, flag) muda o cursor do arquivo F para o offset O relativo à posição indicada pela flag. Ou seja, após a chamada a esta função, qualquer leitura começará na nova posição do cursor. A flag SEEK_SET é o início do arquivo; ou seja, com esta flag, podemos colocar o cursor em qualquer posição do arquivo. Isto será usado para acessar qualquer livro durante a busca binária. Por exemplo, o seguinte código lê o quinto livro do arquivo "arquivo":

```
Livro l;
FILE *f = fopen("arquivo", "rb");
// 0 leria o primeiro livro; 1 o segundo,
// ..., 4 le o quinto livro, que comeca
// no byte 4 * 52 do arquivo.
fseek(f, 4 * sizeof(Livro), SEEK_SET);
fread(&l, sizeof(Livro), 1, f);
```

ftell(F) retorna a posição do cursor do arquivo F. Será usada para saber quantos registros há em um arquivo, também na busca binária, em conjunto com a flag SEEK_END da função fseek. A ideia é ir até o final do arquivo, pegar a posição do cursor e dividir pelo tamanho de cada livro para saber quantos livros há no arquivo.

Note que, ao abrir os arquivos de cada estante num editor de texto comum, **você não verá o conteúdo da mesma forma como um arquivo texto**. Ou seja, os arquivos da estante mostrados acima **não são exatamente o que você verá** quando seu programa escrevê-los. Os exemplos acima foram apenas para mostrar quais livros devem estar em cada estante e em que ordem. Isto é esperado.

3. Criar o arquivo de índice. Para o nosso exemplo, o arquivo de indices seria:

```
algoritmos_teoria_e_pratica data_mining_and_analysis
harry_potter projeto_de_algoritmos
#
#
```

É importante notar que para as estantes sem livros o indice no arquivo deverá ser o caracter '#'

4. Imprimir em tela o resultado das pesquisa. Lembre-se que essa pesquisa deverá ser feita usando busca binaria sobre o arquivo dos livros da estante. A busca binária deverá ser feita diretamente sobre o arquivo, para que seja eficiente. Ou seja, você não pode importar o arquivo todo para a memória e só então fazer busca binária, pois isto seria inútil (sua complexidade já foi $O(n)$ por ler todos os livros).

   No nosso exemplo, temos a pesquisa de 3 livros, que são:

```
data_mining_and_analysis
o_grande_projeto
cplusplus_para_leigos
```

   Pelo arquivo de indices, descobrimos quais são as estantes de cada um e então é feito a busca binária para descobrir o estado dos livros. A resposta para essas pesquisas é:

```
emprestado
livro nao encontrado
disponivel na posicao 3 na estante 0
```

3 O que deve ser entregue

Deverá ser submetido um arquivo **.zip** contendo somente uma pasta chamada `tp1` e dentro desta deverá ter: (i) Documentação **em formato PDF** e (ii) Implementação.

Documentação Poderá ter no máximo 10 páginas e deverá seguir tanto os critérios de avaliação discutidos na Seção 4.1, bem como as diretrizes sobre a elaboração de documentações disponibilizadas no *moodle*. Além disso, a documentação deverá conter análise experimental dos seguintes aspectos:

1. Análise do tempo da ordenação inicial com relação à variação da memória disponivel;
2. Análise do tempo com relação ao número de estantes e suas devidas capacidades;
3. Análise do tempo com relação ao número de pesquisas a serem feitas.

Implementação Código fonte do seu TP (*.c* e *.h*)

Makefile Inclua um *makefile* na submissão que permita compilar o trabalho. É obrigatório o uso das *flags*: **-Wall -Wextra -Werror -std=c99 -pedantic** na compilação.

4 Avaliação

Eis uma lista **não exaustiva** dos critérios de avaliação que serão utilizados.

4.1 Documentação

Introdução Inclua uma breve explicação do problema que está sendo resolvido no seu trabalho e um resumo da sua solução.

Solução do Problema Você deve descrever a solução do problema de maneira clara e precisa, detalhando e justificando os algoritmos e estruturas de dados utilizados. Para tal, artifícios como pseudo-códigos, exemplos ou diagramas podem ser úteis. Note que documentar uma solução não é o mesmo que documentar seu código. **Não** é necessário incluir trechos de código em sua documentação nem mostrar detalhes de sua implementação, exceto quando estes influenciem o seu algoritmo principal, o que se torna interessante.

Análise de Complexidade Inclua uma análise de complexidade de tempo e espaço dos principais algoritmos e estrutura de dados utilizados. Cada complexidade apresentada deverá ser devidamente **justificada** para que seja aceita.

Avaliação Experimental Sua documentação deve incluir os resultados de experimentos que avaliem o tempo de execução de seu código em função de características da entrada. Cabe a você gerar entradas para esses experimentos. Por exemplo: se esse trabalho fosse sobre ordenação, seria interessante mostrar como o tempo de execução de cada algoritmo varia quando o número de items a serem ordenados aumenta. Para tal, um gráfico mostrando o tempo de execução em função do tamanho da entrada pode ser interessante. Você também deve interpretar os resultados obtidos. Comente sobre cada gráfico ou tabela que você apresentar mostrando o que é possível concluir a partir dele.

4.2 Implementação

Linguagem & Ambiente O seu programa deverá ser implementado na linguagem **C** e poderá fazer uso de funções da biblioteca padrão da linguagem. Trabalhos que utilizem qualquer outra linguagem de programação e/ou que façam uso de outras bibliotecas que não a padrão serão zerados. Além disso, certifique-se que seu código compile e funcione corretamente nas máquinas **Linux** dos laboratórios do DCC.

Casos de teste A sua implementação passará por um processo de correção automatizado, portanto, o formato da saída do seu programa deve ser idêntico aquele descrito nessa especificação. Saídas com qualquer divergência serão consideradas erradas, mesmo que as divergências sejam *whitespaces*. e.g. espaços, *tabs*, quebras de linha, etc. Para auxilia-lo na depuração do seu código, será fornecido um pequeno, **não-exaustivo**, conjunto de entradas e suas respectivas saídas. É seu dever certificar-se que seu código funciona corretamente para qualquer entrada válida.

Alocação Dinâmica Algoritmos e estruturas de dados deverão fazer uso de memória alocada dinamicamente (`malloc()` ou `calloc()`). Certifique-se que seu programa utiliza essas regiões de memória corretamente, pois os monitores penalizarão implementações que realizam *out-of-bounds access* e que tenham vazamento de memória (não desalocar memoria dinâmica).
A alocação dinâmica deverá fazer uso das funções malloc() ou calloc() da

biblioteca padrão C, bem como liberar tudo o que for alocado utilizando free(), para gerenciar o uso da memória. **DICA:** Utilize `valgrind` antes de submeter o seu TP.

Qualidade do código Seu código também será avaliado no quesito de legibilidade, dando atenção, porém não limitando-se, aos seguintes items: (i) **INDENTAÇÃO**; (ii) nomes de variável e função descritivos e claros; (iii) Modularização adequada; (iv) Comentários dentro de funções, explicando o que certos trechos mais complicados fazem; (v) Comentários fora de funções, explicando, em alto-nível, o que as funções mais importantes fazem; (vi) funções concisas que desempenham somente uma tarefa; (vii) **Proibido uso de variáveis globais**.

Atrasos Trabalhos poderão ser entregues após o prazo estabelecido, porém sujeitos a uma penalização regida pela seguinte fórmula:

$$\Delta_p = \frac{2^{d-1}}{0.32}\,\%$$

Por exemplo, se a nota dada pelo corretor for 70 e você entregou o TP com 4 dias corridos de atraso, sua penalização será de $\Delta_p = 25\%$ e, portanto, a sua nota final será: $N_f = 70 \cdot (1 - \Delta_p) = 52.2$. Note que a penalização é exponencial e 6 dias de atraso resultam em uma penalização de 100%.

5 Consideração Final

Assim como em todos os trabalhos dessa disciplina é estritamente proibida a copia parcial ou integra de códigos, seja da internet ou de colegas. Utilizaremos o algoritmo *MOSS* para detecção de plágio em trabalhos, seja honesto. Você não aprende nada copiando código de terceiros nem pedindo a outra pessoa que faça o trabalho por você. Se a cópia for detectada, sua nota será zerada e os professores serão informados para que as devidas providências sejam tomadas.

HAVE FUN!!!