Trabalho Prático 2: Índice Invertido

Algoritmos e Estruturas de Dados III – 2017/1
Entrega: 14/06/2017

1 Introdução

Hetelberto Topperson passa muito tempo no ZipZop, um aplicativo de troca de mensagens. Ele sempre procura ter o máximo de contatos no aplicativo pois ele se preocupa demais com sua vida social.

No entanto Hetelberto é muito esquecido, ele tem um sério problema para se lembrar de pessoas. Ele consegue se lembrar de um assunto que conversava no passado muito bem, no entanto não consegue se lembrar com quem que ele conversou. Alguns acreditam que isso se deve a sua obsessão em ficar adicionando muitos contatos.

Hetelberto queria que o aplicativo permitisse buscas de modo a retornar um trecho de conversa com um contato dado uma palavra qualquer que apareceu na conversa.

Como o código fonte do aplicativo é aberto, Hetelberto pode fazer modificações no mesmo e implementar seu pequeno sistema de buscas.

Sua amiga Inês sugeriu que ele fizesse um índice invertido com as palavras dos assuntos que ele conversava. Uma vez que o buscador já havia sido implementado por ela, ele só precisaria de um índice invertido.

Hetelberto não entendeu muito bem como aquilo ajudaria ele, mas ele seguiu a dica de Inês e elaborou um esquema para construir seu índice invertido.

1.1 Índice Invertido

Um índice invertido é uma estrutura de índice que armazena registros que mapeiam um conteúdo em documentos de uma coleção. Tais índices servem principalmente como forma de se realizar buscas rápidas em uma base de texto muito grande.

O conteúdo geralmente é textual, ou seja, mapeia-se texto em documentos. Em uma coleção de documentos extremamente grande torna-se inviável

busca textual utilizando algoritmos tradicionais de processamento de caracteres e faz-se necessário uma estrutura para facilitar tal busca.

Uma implementação simples de um índice invertido consiste construir registros no formato $< w, d, f, p >$, onde w é uma palavra, d é o documento onde encontramos aquela palavra, f é a frequência da palavra naquele documento, p é a posição em bytes daquela palavra no documento (Algumas implementações armazenam a frequência da palavra em toda a coleção, mas não será o caso deste trabalho).

Estes registros são gerados para todas as palavras no texto e ordenados em um grande arquivo. Este arquivo servirá como entrada para uma máquina de busca. O mecanismo de busca a ser utilizado dependerá da necessidade do usuário da máquina de busca e a implementação do mesmo não está no escopo desta disciplina (Há uma disciplina avançada de Recuperação de Informação e Máquinas de Busca no DCC para os interessados).

1.2 Construção do Índice

Como Inês já tem uma algoritmo de busca implementado, resta para Hetelberto construir uma estrutura de índice para servir como entrada para a mesma.

A construção do índice seguirá o modelo simples mostrado na seção anterior seguindo o formato $< w, d, f, p >$.A saída será um arquivo onde será escrito um registro por linha. Os documentos serão números inteiros que correspondem a um identificador de contato das pessoas com que Hetelberto conversa.

Hetelberto começou a implementar seu algoritmo indexador. Ele teve primeiramente a seguinte ideia:

> Vou simplesmente armazenar todo o texto em um vetor de registros contando a frequência em cada conversa, marcar as posições e ordenar com um quicksort, não é nada muito complicado.

Quando ele executou seu código ele percebeu que o seu celular não daria conta de indexar todo o texto de conversa dele devido a memória (Sim, Hetelberto é uma pessoa que fala muito e gosta de puxar assunto, o que pode ser até irritante às vezes de acordo com Inês).

Ele não pode guardar todos os registros na limitada memória do seu celular e não sabe como vai fazer para indexar suas conversas. Hetelberto possui bastante dinheiro (seus pais, na verdade), e está disposto a pagar para quem indexar suas conversas no celular.

Você, estudante, precisa de dinheiro rápido e topou com Hetelberto no corredor da faculdade. Ele te explicou a situação e você aceitou resolver o problema dele.

2 Entrada e saída

Entrada A entrada começa com uma linha contendo 2 números inteiros D e M e duas strings E e S. Onde D é o número de conversas (Chamaremos de documentos), M é o tamanho da memória primária em bytes do celular de Hetelberto e E e S são caminho para diretórios de entrada e saída respectivamente.

O diretório P será conterá conversas que serão arquivos numerados de 1 até D onde cada arquivo será composto de texto puro com caracteres ASCII minúsculos (a-z) e espaços em branco (espaço e quebra de linha). As palavras não será maiores do que 20 caracteres.

Exemplo de entrada

```
3 40 /diretorio/de/conversas/ /diretorio/de/indice/
```

Conteúdo do arquivo 1

```
oi tudo bem
oi tudo sim
ontem foi top voce curtiu
nao
```

Conteúdo do arquivo 2

```
nao foi voce que disse
foi sim
ta bem entao
```

Conteúdo do arquivo 3

```
quanto ta tudo isso
cem tudo
top
voce que entao
sim
```

Saída Neste trabalho você não deve imprimir nada na saída padrão, e sim armazenar o seu índice dentro do diretório S com o nome *index*. Você escreverá registros nesse arquivo no formato $< w, d, f, p >$ explicado anteriormente. Lembre-se que a posição da palavra é dada em bytes e que o caractere

de quebra de linha conta como byte. A ordenação deverá ser feita lexicograficamente para as palavras como primeiro critério e os demais termos serão ordenados como valores inteiros não negativos.

Exemplo de saída

```
bem,1,1,8
bem,2,1,34
cem,3,1,20
curtiu,1,1,41
disse,2,1,17
entao,2,1,38
entao,3,1,42
foi,1,1,28
foi,2,2,4
foi,2,2,23
isso,3,1,15
nao,1,1,48
nao,2,1,0
oi,1,2,0
oi,1,2,10
ontem,1,1,22
quanto,3,1,0
que,2,1,13
que,3,1,38
sim,1,1,18
sim,2,1,27
sim,3,1,48
ta,2,1,31
ta,3,1,7
top,1,1,32
top,3,1,29
tudo,1,2,3
tudo,1,2,13
tudo,3,2,10
tudo,3,2,24
voce,1,1,36
voce,2,1,8
voce,3,1,33
```

3 O que deve ser entregue

Você já deve ter percebido que este trabalho demanda a implementação de Ordenação Externa. Foi dado um valor M de limite (em bytes) de memória principal. Obviamente este não será o limite de memória do seu programa, mas um valor não muito maior que este será usado para limitar o quanto de memória estará disponível para executar o seu código. Lembre-se que um caractere ocupa 1 byte na memória e um inteiro ocupa 4.

Qualquer algoritmo de Ordenação Externa apresentado em sala de aula poderá ser utilizado, ou mesmo qualquer algoritmo de ordenação externa apresentado no livro texto adotado (ZIVIANI, N).

Deverá ser submetido um arquivo **.zip** contendo somente uma pasta chamada `tp2` 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 validando as complexidades de tempo e espaço.

Implementação Código fonte do seu TP (*.c* e *.h*), com solução baseada em **Ordenação Externa**.

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. 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!!!