UNIVERSIDADE FEDERAL DE MINAS GERAIS
Departamento de Ciência da Computação

Disciplina	Curso	Turno	Período
Algoritmos e Estruturas de Dados II	—	—	2°

Professores
Camillo Oliveira, Gisele Pappa, Ítalo Cunha, Loïc Cerf e William Schwartz

Trabalho Prático 3
Inversão Baseada em Ordenação

O objetivo é montar um *índice de busca* de granularidade fina capaz de indicar cada *termo* existente numa base de dados, um conjunto de *documentos*. Uma das técnicas para fazer isso é a inversão baseada em ordenação. Na inversão baseada em ordenação criamos *trincas* da forma $< t, d, p >$, onde t é um termo, d é o nome do documento onde o termo t aparece e p é um número que indica a posição do termo t no documento d. O índice de busca é obtido ordenando-se as trincas primeiro pelos termos e depois pela posição do termo no documento.

Dada uma base de documentos com 4 documentos como exemplo:

- DOC1: A casa é bela.
- DOC2: A casa é uma casa bela.
- DOC3: A bela casa amarela.
- DOC4: É uma amarela casa bela.

A primeira coisa a ser feita na base de documentos é padronizar os dados. A transformações a serem realizadas para padronização são: eliminar qualquer caractere não alfabético (qualquer caractere que não é letra) e transformar todos os caracteres alfabéticos para minúsculos, mantendo os espaços.

O texto dos documentos ficariam:

- DOC1: a casa é bela
- DOC2: a casa é uma casa bela
- DOC3: a bela casa amarela
- DOC4: é uma amarela casa bela

Aplicadas as transformações de texto, extraem-se uma trinca para cada termo (palavra) de cada um dos documentos. Para cada documento, obtém-se uma série não ordenada. A seguir ilustram-se as séries não ordenadas obtidas para cada um dos documentos da base de documentos em questão.[1]

1. <a,1,1><casa,1,2><é,1,3><bela,1,4>
2. <a,2,1><casa,2,2><é,2,3><uma,2,4><casa,2,5><bela,2,6>
3. <a,3,1><bela,3,2><casa,3,3><amarela,3,4>
4. <é,4,1><uma,4,2><amarela,4,3><casa,4,4><bela,4,5>

[1] Os símbolos '<' e '>' que aparecem nos exemplos anteriores são apenas para melhor compreensão e legibilidade do exemplo.

Obtidas as séries não ordenadas para os documentos da base, faz-se a ordenação obtendo-se as séries ordenadas pelo termo, pelo documento e pela posição do termo no documento. As séries ordenadas para cada documento da base são mostradas abaixo. Você deve implementar um método de ordenação cuja complexidade para o caso médio seja $O(n \lg n)$.

1. <a,1,1><bela,1,4><casa,1,2><é,1,3>
2. <a,2,1><bela,2,6><casa,2,2><casa,2,5><é,2,3><uma,2,4>
3. <a,3,1><amarela,3,4><bela,3,2><casa,3,3>
4. <é,4,1><amarela,4,3><bela,4,5><casa,4,4><uma,4,2>

A seguir, ilustram-se as intercalações das séries ordenadas a cada dois documentos, podendo ser feito com uma intercação *in-place*.

- Intercalação da série ordenada do DOC1 com a série ordenada do DOC2:
  <a,1,1><a,2,1><bela,1,4><bela,2,6><casa,1,2><casa,2,2><casa,2,5><é,1,3><é,2,3><uma,2,4>
- Intercalação da série ordenada do DOC3 com a série ordenada do DOC4;
  <a,3,1><amarela,3,4><amarela,4,3><bela,3,2><bela,4,5><casa,3,3><casa,4,4><é,4,1><uma,4,2>

Realizando-se todas as intercalações necessárias tomadas duas a duas séries ordenadas, obtém-se a série final intercalada.

- <a,1,1><a,2,1><a,3,1><amarela,3,4><amarela,4,3><bela,1,4><bela,2,6><bela,3,2><bela,4,5>
  <casa,1,2><casa,2,2><casa,2,5><casa,3,3><casa,4,4><é,1,3><é,2,3><é,4,1><uma,2,4><uma,4,2>

A partir da série final intercalada obter um índice utilizando uma estrutura de *hash* perfeito ou uma árvore binária balanceada, onde cada entrada (no *hash* ou na árvore) refere-se a um termo e cada entrada possui uma lista possui uma lista (bloco de memória contíguo) com as informações dos documento onde o termo aparece.

Consultas

A partir do índice construído, podem ser feitas consultas, tais como:

- t — recupera a lista de documentos onde o termo t ocorre.
- t_1 `E` t_2 — recupera a interseção entre as listas de documento que possuem o ambos termos t_1 e t_2.
- t_1 `OU` t_2 — recupera a união entre as listas de documentos que possuem o termo t_1 e o termo t_2.
- `NAO` t — recupera todos os documentos que não possuem o termo t.

O que deve ser feito

- Implementar um programa em linguagem C capaz de gerar o índice para que sejam realizadas consultas simples (com apenas um termo).
- Definição do tipo estrutura:

```
typedef struct {
    char *termo;
    char *documento;
    int posicao;
} TRINCA;
```

O tipo estrutura `TRINCA` possui três elementos: um ponteiro para uma sequência de caracteres (*string*) para o termo, o nome do arquivo onde o termo ocorre e a posição do termo no documento.

Como o tamanho do termo pode variar e não querendo gastar espaços em demasia, para o elemento ponteiro termo do tipo estrutura `TRINCA` deve ser alocado apenas a quantidade de *bytes* suficiente para armazenar sequência de caracteres (e o caractere de terminação).

Os nomes dos arquivos são sempre iniciados com o prefixo DOC, seguido de números e com extensão `.txt`. Por exemplo: `DOC1.txt, DOC10.txt, DOC234.txt` e serão armazenados no campo documento do tipo estrutura TRINCA.

O argumento `argv[1]` é o nome de um arquivo texto que contém os nomes dos arquivos da base de dados de onde serão extraídas as palavras. Por exemplo: o nome do arquivo poderia ser `basedados.txt` que teria os nomes dos arquivos em cada linha.

```
DOC1.txt
DOC2.txt
DOC3.txt
DOC4.txt
```

O seu programa será executado da seguinte maneira:

```
./programa basedados.txt TERMO
```

Seu programa deve imprimir na tela todos os documentos e as respectivas posições onde `TERMO` aparece. Por exemplo, na base de dados composta pelos documentos DOC1–4 temos:

```
./programa basedados.txt casa
DOC1.txt,2
DOC2.txt,2
DOC2.txt,5
DOC3.txt,3
DOC4.txt,4
```

O que deve ser entregue

- Deve ser submetido o código fonte, seus arquivos .h (com as declarações das funções) e seus arquivos .c (com a implementação das funções). O programa deve ser ANSI, ou seja, o código fonte do programa deve ser compilável no Linux, não devendo ser utilizada nenhuma biblioteca que seja específica do sistema operacional MS-Windows.
- Documentação sobre o trabalho (em .pdf), incluindo as decisões tomadas relativas aos casos e detalhes de especificação que porventura estejam omissos no enunciado. Um exemplo de documentação pode ser encontrado no Moodle. Porém, ela deve conter, entre outras coisas:
  1. Introdução: descrição sucinta do problema a ser resolvido e visão geral sobre o funcionamento do programa.
  2. Implementação: descrição sobre a implementação do programa. Deve ser detalhada a estrutura de dados utilizada (de preferência com diagramas ilustrativos), o funcionamento das principais funções e procedimentos utilizados, o formato de entrada e saída de dados, compilador utilizado, bem como decisões tomadas relativas aos casos e detalhes de especificação que porventura estejam omissos no enunciado.
  3. Estudo de complexidade: estudo da complexidade do tempo de execução dos procedimentos implementados e do programa como um todo (notação O), considerando o número de experimentos E e o número de dependências D.
  4. Testes: descrição dos testes realizados e listagem da saída (não edite os resultados).

5. Conclusão: comentários gerais sobre o trabalho e as principais dificuldades encontradas em sua implementação.
6. Bibliografia: bibliografia utilizada para o desenvolvimento do trabalho, incluindo sites da Internet se for o caso.

- O trabalho será entregue através do sistema de submissão de trabalhos práticos: http://aeds.dcc.ufmg.br, em um arquivo compactado (.zip) contendo os arquivos fonte (arquivo .c, o arquivo .h, o main.c). A documentação deve ser enviada em pdf, no link especial para sua submissão no sistema.

Ponto extra

Aqueles que implementarem consultas adicionais, receberão ponto extra. Sugere-se as seguintes consultas:

- t_1 `E` t_2 — recupera a interseção entre as listas de documento que possuem o ambos termos t_1 e t_2.
- t_1 `OU` t_2 — recupera a união entre as listas de documentos que possuem o termo t_1 e o termo t_2.
- `NAO` t — recupera todos os documentos que não possuem o termo t.
- t_1 `ADJ` t_2 — recupera todos os documentos onde o termo t_1 aparece adjacente ao termo t_2