GA-024: Segundo Trabalho Prático
Arquivos Invertidos

Antônio Tadeu A. Gomes
Laboratório Nacional de Computação Científica (LNCC/MCTI)
atagomes@posgrad.lncc.br
https://classroom.google.com/c/NDYzODk1NjA2NTU2
Sala 2C-01

Prazo: sexta-feira da semana de prova

1 Introdução

Arquivos invertidos são constituídos de duas partes: *vocabulário* e *ocorrências*. O vocabulário é o conjunto de todas as palavras distintas no texto. Para cada palavra distinta, uma lista de posições onde ela ocorre no texto é armazenada. O conjunto das listas é chamado de ocorrências. As posições podem referir-se a caracteres, palavras, linhas ou páginas de um texto.

Um bom exemplo de arquivo invertido é um **índice remissivo**, que corresponde a uma lista em ordem alfabética de palavras relevantes do texto (*palavras-chave*), com a indicação dos locais no texto onde cada palavra-chave ocorre. O objetivo deste trabalho é implementar um TAD em C que mantenha, a partir de um conjunto de palavras-chave e um arquivo de texto de entrada, um índice remissivo para esse texto indicando as linhas em que cada palavra-chave ocorre no texto.

Como exemplo, suponha um arquivo com o seguinte texto:

```
Good programming is not learned from
generalities, but by seeing how significant
programs can be made clean, easy to
read, easy to maintain and modify,
human-engineered, efficient, and reliable,
by the application of common sense and
by the use of good programming practices.
```

Assumindo como palavras-chave: `programs`, `easy`, `by`, `and`, `be` e `to`, o índice remissivo gerado pode ser representado como:

```
and: 4, 5, 6
be: 3
by: 2, 6, 7
easy: 3, 4
programs: 3
to: 3, 4
```

2 Descrição

O trabalho consiste em codificar um TAD de índices remissivos como uma tabela Hash. Para simplificar a solução, considere somente o uso de caracteres ASCII de 8 bits nos textos, e palavras-chave de no máximo 16 caracteres (ou seja, 128 bits) no índice remissivo. Use `Index` como nome do TAD e implemente as operações básicas especificadas abaixo. Toda operação deve retornar 0 em caso de sucesso e $\neq 0$ em caso de erro:

```
int index_createfrom(
                const char *key_file,
                const char *text_file,
                Index **idx )
```

lê do arquivo de nome `key_file` as palavras-chave a serem usadas na construção do índice remissivo do texto contido no arquivo texto de nome `text_file`. O índice criado deve ser retornado em `idx`. As palavras-chave em `key_file` devem estar cada uma em uma linha separada, como ilustrado abaixo:

```
programs
easy
by
and
be
to
```

```
int index_get( const Index *idx,
               const char *key,
               int **occurrences,
               int *num_occurrences )
```

devolve em `occurrences` um vetor de inteiros (alocado dinamicamente pela operação) contendo todas as ocorrências (linhas) da palavra-chave `key` armazenadas pelo índice remissivo apontado por `idx`. O número de entradas de `occurrences` é devolvido em `num_occurrences`.

```
int index_put( Index *idx,
               const char *key )
```

inclui no índice remissivo a palavra-chave `key`, associando-a a todas as ocorrências presentes no arquivo texto usado na construção do índice. Caso a palavra-chave já exista no índice remissivo, as ocorrências da mesma no arquivo texto devem ser atualizadas (útil quando o arquivo texto é alterado durante a execução de um programa que usa o TAD).

```
int index_print( const Index *idx )
```

imprime para `stdout` o índice remissivo completo, em ordem alfabética, referenciado por `idx`. O formato de saída desse índice deve ser como ilustrado abaixo:

```
and: 4, 5, 6
be: 3
...
```

3 Resolução e Testes

A resolução do trabalho deve ser feita **individualmente**. Para o trabalho ser considerado como completo, deve ser entregue via Google Classroom, **até o prazo informado no início deste documento**:

1. Arquivo-fonte do módulo em C (`index.c`).

A resolução deve ser testada, **minimamente**, com o programa de teste a seguir, usando o texto e as palavras-chave ilustrados anteriormente como entradas do programa.

```
#include <stdio.h>
#include "index.h"

int main( int argc, char **argv ) {
  if( argc < 3 ) {
    fprintf( stderr, "Erro:    numero insuficiente de parametros:\n" );
    fprintf( stderr, "Sintaxe: %s "
            "key_file_name txt_file_name\n", argv[0] );
    return 1;
  }

  Index *idx;
  if( index_createfrom( argv[1], argv[2], &idx ) ) {
    fprintf( stderr, "Erro: criacao do indice\n" );
    return 1;
  }

  char keyword[17];
  printf( "Qual a palavra-chave a procurar?\n" );
  scanf( " %16[^\n]", keyword );
  int *occurrences;
  int n_occurrences;
  if( index_get( idx, keyword, &occurrences, &n_occurrences ) )
    fprintf( stderr, "Erro: palavra nao pertence ao indice\n" );
  else {
    if( n_occurrences <= 0 )
      printf( "Nao ha ocorrencias de %s\n", keyword );
    else {
      printf( "%d ocorrencias de %s: ", n_occurrences, keyword );
      int i;
      for( i=0; i<n_occurrences-1; i++ )
        printf( "%d, ", occurrences[i] );
      printf( "%d\n", occurrences[n_occurrences-1] );
    }
```

```
  }

  printf( "Indice completo:\n" );
  if( index_print( idx ) ) {
    fprintf( stderr, "Erro: impressao do indice\n" );
    return 1;
  }

  char new_keyword[17];
  printf( "Qual a palavra-chave a inserir?\n" );
  scanf( " %16[^\n]", new_keyword );
  if( index_put( idx, new_keyword ) ) {
    fprintf( stderr, "Erro: insercao no indice\n" );
    return 1;
  }

  printf( "Novo indice completo:\n" );
  if( index_print( idx ) ) {
    fprintf( stderr, "Erro: impressao do novo indice\n" );
    return 1;
  }

  return 0;
}
```

É importante destacar que os testes acima representam um **conjunto mínimo**, servindo de guia para a resolução; outros testes adicionais podem ser efetuados pelo professor.