GA-024: Segundo Trabalho Prático
Arquivos Invertidos

Antônio Tadeu A. Gomes
Laboratório Nacional de Computação Científica (LNCC/MCT)
atagomes@gmail.com
http://martin.lncc.br
Sala 2C-01

April 14, 2009

1 Descrição

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 uma estrutura de dados 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:

```
1: Good programming is not learned from
2: generalities, but by seeing how significant
3: programs can be made clean, easy to
4: read, easy to maintain and modify,
5: human-engineered, efficient, and reliable,
6: by the application of common sense and
7: 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
```

Use uma estrutura de **Árvore Patricia** na implementação do índice remissivo. Para simplificar a solução, considere somente o uso de caracteres ASCII de 8 bits nos textos, e palavras-chave de no máximo 8 caracteres (ou seja, 64 bits) no índice remissivo. Use `PTree` como nome do TAD e implemente as operações básicas especificadas abaixo:

`PTree* ptree_create( char *key_file, char *text_file )`: lê do arquivo de nome `key_file` as palavras-chave a serem usadas na construção do índice remissivo do texto contido no arquivo de nome `text_file`. As palavras-chave em `key_file` devem estar cada uma em uma linha separada, como ilustrado abaixo:

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

`int ptree_getoccurrences( PTree *p, char *key, int **occurrences )`: devolve em `occurrences` um vetor de inteiros (alocado dinamicamente pela operação) contendo todas as ocorrências (linhas) da palavra-chave `key` armazenadas pela árvore Patricia referenciada por `p`. O valor de retorno da função corresponde ao tamanho de entradas de `occurrences`.

`void ptree_print( PTree *p )`: imprime para `stdout` o índice remissivo completo, em ordem alfabética, referenciado por `p`. O formato de saída desse índice deve ser como ilustrado abaixo:

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

Dica: Árvores Patricia manipulam chaves como conjuntos de bits; pode ser útil, portanto, a implementação de uma função auxiliar

```
int bit( int i, char *key )
```

que retorna o valor (0 ou 1) do `i`-ésimo bit da chave `key`. Lembre-se que uma string é um vetor de caracteres, e cada caractere tem uma representação inteira (no caso deste trabalho, o código ASCII de 8 bits associado a esse caractere).

2 Resolução e Testes

A resolução do trabalho pode ser feita **individualmente** ou **em dupla**. Para o trabalho ser considerado como completo, deverão ser apresentadas:

1. Listagem do programa em C.
2. Listagem dos testes executados.
3. Demonstração da resolução junto ao professor.

A resolução deve ser obrigatoriamente testada com o seguinte programa:

```
#include "ptree.h"

int main( int argc, char **argv ) {
  /* Inicializacao da aplicacao ... */

  PTree *p = ptree_create( argv[1], argv[2] );
  char keyword[9];
  printf( "Qual a palavra-chave a procurar?\n" );
  scanf( " %8[^\n]", keyword );
  int *occurrences;
  int n_occurrences = ptree_getoccurrences( p, keyword, &occurrences );
  if( n_occurrences <= 0 )
    printf( "Nao ha ocorrencias de %s\n", keyword );
  else {
    printf( "Ocorrencias de %s: ", 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" );
  ptree_print( p );
  return 0;
}
```

usando o texto e palavras-chave ilustrados anteriormente como entradas do programa.