Instituto Superior de Engenharia de Lisboa

LEETC – LEIRT

Programação II

2019/20 – 2.º semestre letivo

Terceira Série de Exercícios

Nesta série de exercícios complementa-se a funcionalidade do programa especificado na série de exercícios anterior, passando a dispor de comandos de pesquisa, suportados em estruturas de dados dinâmicas. Pretende-se, também, melhorar o aproveitamento da memória, evitando o desperdício de espaço ocupado com réplicas do mesmo nome (de álbum ou de intérprete).

Tal como especificado na série de exercícios anteriores, o programa destina-se a catalogar um conjunto de faixas de áudio, armazenadas em ficheiros com formato MP3, com base na respetiva *tag*, se existir, permitindo listagens ordenadas e agora também pesquisa por uma palavra pertencente ao título de uma faixa.

O programa recebe, como argumento de linha de comando, o nome de um ficheiro de texto contendo uma lista com os nomes dos ficheiros MP3 a catalogar. Após a leitura da lista e dos ficheiros que ela indica, apresenta ciclicamente um *prompt* e espera, de *standard input*, um dos comandos seguintes.

- `a` – (*artists*) apresenta a lista de faixas áudio ordenada por *Artist*. Se houver várias faixas do mesmo intérprete, aplica, sucessivamente, a ordem por *Album* e *Title*;
- `t` – (*titles*) apresenta a lista de faixas áudio ordenada por *Title*. Se houver várias faixas com título idêntico, aplica, sucessivamente, a ordem por *Artist* e *Album*;
- `s` *word* – (search) apresenta a lista das faixas áudio cujo título contém a palavra indicada. A ordenação é idêntica à do comando `t`;
- `q` – (*quit*) terminar.

As listas são produzidas em *standard output*, pela execução dos comandos, e a sua ordem é alfabética crescente. É exibida a informação de cada faixa áudio numa linha, contendo os campos da *tag* e o nome do ficheiro MP3 correspondente.

1. Para implementar o comando de pesquisa, `s`, valorizando a eficiência, propõe-se a utilização de uma árvore binária de pesquisa. Cada nó da árvore identifica uma palavra e referencia, com uma lista ligada, o conjunto de faixas que contêm essa palavra no título.

   Sugere-se que a criação da árvore seja feita na função `setReady` do módulo `setdata.c`, após o preenchimento da estrutura de dados principal com os dados de todas as faixas e a respetiva ordenação. Para melhorar a eficiência das pesquisas, é conveniente o balanceamento da árvore. Em anexo é indicado um algoritmo para esse fim.

   1.1. Identifique as modificações a realizar à solução da SE2, para adicionar as estruturas de dados e o código de suporte ao novo comando `s`, e caraterize os elementos seguintes:

   - Os novos módulos de código. Propõe-se que considere dois – um para implementar as listas ligadas que referenciam conjuntos de faixas e outro para implementar a árvore binária.

- A forma de integrar a nova funcionalidade, mantendo a relação do módulo de aplicação com o módulo de gestão de dados, `setdata.c`, e sendo este a utilizar os novos módulos.

1.2. Desenvolva o módulo de listas ligadas.

- Defina o tipo para representar o nó de lista, capaz de referenciar uma faixa, apontando uma das *tags* referenciadas a partir do descritor `tagSet_t`.

- Escreva o código do módulo e o respetivo *header file*, contendo as declarações necessárias para a sua utilização. Deve conter nomeadamente as assinaturas das funções de interface.

1.3. Desenvolva o módulo de árvore binária de pesquisa.

- Defina o tipo para representar o nó de árvore, capaz de referenciar uma palavra, com alojamento dinâmico, e uma lista ligada de referências, conforme especificado.

- Escreva o código do módulo e o respetivo *header file*, contendo as declarações necessárias para a sua utilização. Deve conter nomeadamente as assinaturas das funções de interface.

1.4. Desenvolva o código para implementar o comando de pesquisa, explorando a árvore binária.

- Modifique o *makefile* para integrar os novos módulos.

- Modifique o tipo `tagSet_t`, adicionando um ponteiro raiz para a árvore binária.

- Adicione ao módulo de gestão de dados a implementação da nova funcionalidade.

- Modifique e teste o código da aplicação, de acordo com a especificação.

2. Modifique o código de gestão de dados para melhorar a utilização da memória, evitando ocupação com réplicas de cada nome de álbum ou de intérprete:

2.1. Desenvolva o módulo para armazenamento de *strings* sem repetição, `strset.c`, e o respetivo *header file* `sterset.h`.

A implementação deste módulo deve ser baseada numa *hash-table*, com motivação na eficiência e para demonstrar o funcionamento deste tipo de estrutura de dados.

O módulo referido usa o tipo `strSet_t` como descritor principal e deve disponibilizar as seguintes funções de interface:

```
strSet_t *strSetCreate( void );
```

Esta função cria, em alojamento dinâmico, o descritor para o conjunto de *strings*. Retorna o endereço do descritor criado.

```
char *strSearchAdd( strSet_t *set, const char *str );
```

Esta função procura no conjunto `set` a *string* indicada. Se não existir, cria-a em alojamento dinâmico e adiciona-a ao conjunto. Retorna o endereço da *string* armazenada no conjunto.

```
void strSetDelete( strSet_t *set );
```

Esta função elimina o conjunto de *strings* `set`, libertando a memória de alojamento dinâmico usada pelo descritor e pelas *strings*.

2.2. Escreva um programa de teste que use todas as funções do módulo desenvolvido e verifique o armazenamento de *strings* iguais ou diferentes. Execute o teste, de modo a corrigir eventuais erros.

2.3. Modifique o tipo `MP3Tag_t` para que os campos com o nome de álbum e de artista sejam ponteiros, de modo a usar as *strings* armazenadas sem repetição.

```
char *artist;
char *album;
```

2.4. Redefina a assinatura e o código da função de interface do módulo `tag.c`, adicionando um parâmetro para acesso ao conjunto de *strings*.

```
MP3Tag_t *tagRead( char *fileName, int *resPtr, strSet_t *sSet );
```

2.5. Redefina a assinatura e o código da função de interface do módulo `listfiles.c`, adicionando um parâmetro para acesso ao conjunto de *strings*.

```
int listScan( char *listName, TagSet_t *data, strSet_t *sSet );
```

2.6. Modifique o *makefile* para integrar o módulo de conjunto de *strings*.

Rescreva o módulo de aplicação de modo a usar estas funcionalidades. No início deve criar o descritor de conjunto de *strings* e passá-lo à função `listScan`; no final deve eliminá-lo.

Verifique se há impacto destas alterações nos outros módulos. Se forem necessárias adaptações, implemente-as.

ISEL, 2020.06.16

Anexo

Identificação das palavras nos títulos

Para construir a árvore binária de pesquisa é necessário identificar, e copiar para alojamento dinâmico, cada uma das palavras existentes nos títulos das faixas. Propõe-se que use, em alternativa, uma das funções `strtok` ou `sscanf` da biblioteca normalizada. Os exemplos de código seguintes ilustram o uso destas funções.

```
void exampleSplit1( const char str[] ){
    char sc[MAX_STR];
    strcpy( sc, str );
    char *p = strtok( sc, " \t\n" );
    while( p != NULL ){
          printf( "%s\n", p );
          p = strtok( NULL, " \t\n" );
    }
}

void exampleSplit2( const char str[] ){
    char word[MAX_WORD];
    int i = 0, n;
    while( sscanf( str + i, "%s%n", word, &n ) == 1 ){
          printf( "%s\n", word );
          i += n;
    }
}
```

Balanceamento da árvore binária

Para uma utilização eficiente, as árvores binárias devem ser balanceadas. Propõe-se, para simplificar, que as crie sem manter permanentemente o balanceamento, realizando-o através da função `Tbalance`, a intervalos de várias inserções e, principalmente, no final. O código proposto abaixo considera o nó de árvore com o tipo `Tnode` os campos de ligação com os nomes `left` e `right`.

Para implementar a função `Tbalance`, propõe-se a técnica de balanceamento em dois passos:

1. Transformar a árvore binária numa árvore degenerada em lista ordenada, ligada pelo campo `right`, usando o algoritmo seguinte.

```
Tnode *treeToSortedList( Tnode *r, Tnode *link ){
    Tnode * p;
    if( r == NULL ) return link;
    p = treeToSortedList( r->left, r );
    r->left = NULL;
    r->right = treeToSortedList( r->right, link );
    return p;
}
```

2. Conhecido o número de elementos, transformar a lista numa árvore, usando o algoritmo seguinte.

```
Tnode* sortedListToBalancedTree(Tnode **listRoot, int n) {
    if( n == 0 )
          return NULL;
    Tnode *leftChild = sortedListToBalancedTree(listRoot, n/2);
    Tnode *parent = *listRoot;
    parent->left = leftChild;
    *listRoot = (*listRoot)->right;
    parent->right = sortedListToBalancedTree(listRoot, n-(n/2 + 1) );
    return parent;
}
```