Instituto Superior de Engenharia de Lisboa

LEETC

Programação II

2019/20 – 1.º semestre letivo

Terceira Série de Exercícios

Esta série de exercícios tem como objetivo principal a exploração de estruturas de dados dinâmicas com diversas topologias: lista ligada; árvore binária de pesquisa; *hash-table*. São aproveitados, como base, os módulos de programa desenvolvidos na Série de Exercícios 2. Propõe-se a construção de versões alternativas para parte do código, usando as estruturas de dados referidas.

A solução usada na SE2 para implementar o índice de palavras é baseada em *array* dinâmico, redimensionado por `realloc`. Esta forma levanta preocupações de: eficiência, devido ao peso computacional do realojamento e da ordenação; flexibilidade, devido à necessidade de dispor, em *heap*, de um bloco único de grandes dimensões para conter o *array* de palavras.

Com os propósitos de melhorar a flexibilidade e de avaliar, comparativamente, a eficiência de outras estruturas de dados, propõe-se nos exercícios seguintes a construção de outras versões do índice, baseadas em árvore binária ou em *hash-table*.

1. Pretende-se o desenvolvimento de uma versão do índice, para que se propõe o nome `index3.c`, utilizando uma árvore binária de pesquisa para armazenar as palavras. É necessário definir os tipos para representar a nova estrutura de dados, nomeadamente o nó de árvore, cujo campo de dados deve do tipo `Word`.

   O método sugerido para a construção da árvore, com o propósito de simplificar a implementação, é a inserção nas folhas. No entanto, para obter pesquisas eficientes, é importante que a árvore seja balanceada. Por isso, no final da construção, deve ser feito o balanceamento. É proposto, em anexo, um algoritmo para esse efeito.

   1.1. Escreva o novo módulo `index3.c`, com a implementação do índice em árvore binária.

   1.2. Teste o código realizado, adicionando para isso, ao *makefile*, uma versão do executável, `wts4`, para testar o módulo `index3.c`.

2. Pretende-se o desenvolvimento de uma versão do índice, para que se propõe o nome `index4.c`, utilizando uma *hash-table* para armazenar as palavras. A chave de indexação deve ser a palavra a pesquisar. As colisões são resolvidas por listas ligadas. É necessário definir os tipos para representar os elementos da *hash-table*, nomeadamente o nó de lista, cujo campo de dados deve ser do tipo `Word`.

   2.1. Escreva o novo módulo `index4.c`, com a implementação do índice em *hash-table*.

   2.2. Teste o código realizado, adicionando para isso, ao *makefile*, uma versão do executável, `wts5`, para testar o módulo `index4.c`.

3. Faça um ensaio comparativo das diversas versões, usando o comando `time` para obter os tempos de execução. Propõe-se que, além dos executáveis e ficheiros de texto a processar, prepare ficheiros com palavras a pesquisar, para usar com redireccionamento de *standard input*, de modo a facilitar a aplicação das mesmas pesquisas aos diferentes executáveis. Sugere-se também que redirecione o *standard output* para outro ficheiro, de modo a exibir na consola apenas os tempos pretendidos. Exemplo de comando:

   ```
   time wts texto1 texto2 texto3 < ficheiro_palavras > resultados
   ```

   Pretende-se ainda distinguir o tempo de preparação do índice face ao tempo total, de modo a determinar o tempo relativo às pesquisas. Para isso, propõe-se que cada executável seja executado em, pelo menos, dois cenários:
   a) Usando um ficheiro de palavras vazio (tempo nulo dedicado a pesquisas);
   b) Usando um ficheiro de palavras com conteúdo, de preferência extenso.

   Tendo em conta a possibilidade de repetir algumas vezes este ensaio, é conveniente escrever um *shell-script* com os comandos de execução das diversas versões nos dois cenários indicados. Um *shell-script* é um ficheiro de texto, preparado no editor, cujas linhas são comandos reconhecidos pelo interpretador (*shell*). Para que possa ser executado, o ficheiro de *script* tem de ser marcado com atributo de execução, através do comando `chmod`.

Anexo – algoritmo de 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 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 o balanceamento, propõe-se o método em dois passos:

1.º 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.º 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;
}
```

ISEL, 2019.12.10