Disciplina: Estruturas de Dados e Algoritmos II

Docentes: Professora Irene Rodrigues e Professor Pedro Patinho

Curso: Engenharia Informática

Sistema de Pesquisa Numa Base de Textos

Trabalho Realizado Por:

Marlene Oliveira Nº 25999

Pedro Mateus Nº 26048

João Aiveca Nº 26175

Ano Lectivo 2010/2011

1. Estruturas de Dados utilizadas pelo sistema de pesquisa

O nosso sistema de pesquisa utiliza as seguintes estruturas de dados:

- ***Btree***: o sistema utiliza uma Btree de ordem 3 para guardar as palavras que constam em cada história.
- ***Struct Historia***: esta estrutura armazena os apontadores para os elementos que constituem a história que contém a mesma, ou seja, armazena os apontadores para o título da história, a listagem das personagens, as conclusões da história, o texto que compõe a história. Para além disto também é armazenado nesta estrutura o nome do ficheiro.
- ***Struct Node***: nó da Btree, que contém um apontador correspondente a uma palavra e o inteiro que armazena a frequência com que a mesma aparece no texto.

2. Operações Implementadas

No decurso da implementação do nosso sistema foram implementas as seguintes operações:

- ***novaHistoria***: a operação *novaHistoria* recebe como argumento um apontador para uma estrutura do tipo Historia. Esta operação tem como objectivo, tal como o nome indica, inicializar uma estrutura do tipo Historia, baseada num nome de ficheiro dado pelo utilizador (Se este é válido). A complexidade temporal desta operação é proporcional ao número de caracteres do input, que são copiados através de strcpy, e é corrrido file_ops (cuja complexidade é referida abaixo). A complexidade espacial é 128*sizeof(char) (O array de caracteres que representa o input do user).
- ***file_ops***: a operação *file_ops* recebe como argumentos uma string correspondente ao nome do ficheiro e um ficheiro FILE. Esta operação tem como objectivo armazenar os elementos contidos no ficheiro FILE numa estrutura do tipo História, ou seja, armazena os dados correspondentes ao título, às personagens, às conclusões e o texto numa estrutura do tipo Historia. A complexidade temporal desta operação é 2*n, onde n=número de caracteres do texto, já que é percorrido uma vez para calcular o tamanho do texto (para reservar memória), e ma segunda para guardar os dados. A complexidade espacial é n*sizeof(char), ou seja, o número de caracteres do texto.
- ***getString***: esta operação recebe como argumentos um apontador para um ficheiro FILE, um apontador para uma string e um inteiro correspondente ao tamanho da string. Esta função tem como objectivo copiar os dados que constam no ficheiro para a string que corresponde ao apontador. A complexidade temporal desta operação é n, ou seja, o número de caracteres da string argumento. Dado receber todos os argumentos directamente, não é adicionada complexidade espacial por esta função.
- ***calcsize***: esta operação recebe como argumentos um apontador para um ficheiro do tipo FILE e um inteiro que indica se a função deve ignorar o carácter New Line ou não. A operação retorna um inteiro correspondente ao tamanho dos dados contidos no ficheiro. A complexidade temporal desta operação é n, o número de caracteres da string argumento. A complexidade espacial é sizeof(int), o único inteiro que guarda o valor que se pretende calcular.
- ***savewords***: esta operação recebe como argumentos um apontador para uma string e uma Btree. A operação *savewords* tem como objectivo armazenar na Btree as palavras que constam na string a que corresponde o apontador. Esta operação retorna uma Btree onde constam as palavras armazenadas. A complexidade temporal da

operação é n+log(p), sendo n o número de caracteres do texto passado como argumento, e log(p) a complexidade da inserção numa B-Tree de uma palavra (string) p. A complexidade espacial corresponde a sizeof(B-Tree)+sizeof(char)*(n-m), ou seja, o espaço ocupado pela B-Tree (descrito abaixo) + o espaço dos caracteres do texto (n) excluindo espaços, vírgulas, pontos finais, etc. (qualquer elemento que não componha uma palavra).

- ***calc_est***: a operação *calc_est* recebe como argumentos um apontador para o padrão a ser procurado, o estado actual e o carácter do estado que se prentende calcular. A operação tem como objectivo calcular o estado para o carácter indicado. A complexidade temporal da operação é, no pior caso, o número de caracteres do padrão (caso seja necessário recuar estados).
- ***aut***: a operação *aut* recebe como argumentos os apontadores correspondentes ao padrão a ser procurado e ao texto no qual irá ser procurado o referido padrão. A operação retorna uma tabela com todos valores correspondentes aos estados do autómato finito. A complexidade da operação é n, sendo n o número de caracteres do texto.
- ***nova***: esta operação tem como principal objectivo inicializar uma Btree. A operação recebe como argumento o endereço da estrutura do tipo Btree a ser inicializada. A complexidade temporal da operação é maxfilhos, para inicializar todos os apontadores dos filhos da B-Tree. A complexidade espacial é maxfilhos*sizeof(pointer).
- ***novoNo***: esta operação recebe como argumento uma estrutura do tipo Node. A operação *novoNo* tem como principal função inicializar um novo nó, que mais tarde irá conter um valor correspondente a uma palavra e um inteiro correspondente à frequência com que a palavra aparece no texto. A complexidade temporal da operação é 1, e a espacial 30*sizeof(char) (Aceita palavras até 30 caracteres).
- ***inserir***: esta operação recebe como argumentos dois apontadores, um para uma estrutura do tipo Btree e outro para uma string. Esta operação tem como principal objectivo inserir a string na Btree. A complexidade temporal da operação é log(n), sendo n o número de chaves na B-Tree. A complexidade espacial da inserção na B-Tree (que engloba os métodos de inserção todos: inserir, insertNotFullLeaf, insertNotFullNonLeaf, split) é n*(maxchaves * sizeof(node) + sizeof(int) + maxfilhos * sizeof(pointer)), sendo n o número de chaves na B-Tree.
- ***insertNotFullLeaf***: a operação *insertNotFullLeaf* tem como principal funcionalidade inserir a string dada numa folha da Btree que ainda não está cheia. Esta operação, à semelhança da operação inserir, recebe como argumentos dois apontadores – um para uma estrutura do tipo Btree e outro para uma string. A complexidade temporal da operação é n, sendo n o número de chaves actualmente na B-Tree.
- ***insertNotFullNonLeaf***: a operação *insertNotFullNonLeaf* tem como principal funcionalidade inserir a string dada num nó da Btree que ainda não se encontra cheio. Esta operação, à semelhança da operação inserir, recebe como argumentos dois apontadores – um para uma estrutura do tipo Btree e outro para uma string. A complexidade temporal desta operação é log(n)*maxchaves no pior cenário.
- ***listar***: esta operação recebe como argumento uma estrutura do tipo Btree. A operação listar tem, tal como o nome indica, como objectivo listar os elementos que constam na Btree. A complexidade temporal da operação é n, sendo n o número de nós na B-Tree.
- ***split***: esta operação recebe como argumentos dois apontadores para estruturas do tipo Btree (uma correspondente ao nó da Btree que está cheio e outro correspondente ao pai desse nó) e um inteiro que indica a posição. A operação *split* tem como

principal funcionalidade partir um nó da Btree que se encontra cheio quando se pretende inserir dados naquela zona da Btree. A complexidade temporal desta operação é maxfilhos+novoNo()+nova(); o número total de operações implica re-distribuir os (maxfilhos) filhos de um nó. A complexidade espacial corresponde ao espaço ocupado por uma B-Tree nova.

- ***compareString***: esta operação recebe como argumentos dois apontadores para duas strings. A operação *compareString* tem como principal funcionalidade comparar as duas strings dadas. A complexidade da operação é n, sendo n o número de caracteres da string menor.
- ***searchPersonagem***: esta operação tem como principal funcionalidade listar os títulos e os nomes dos ficheiros de todas as histórias que têm a personagem dada. A operação *searchPersonagem* recebe como argumentos dois apontadores (um para uma string a procurar e outro para uma struct do tipo Historia). A complexidade da operação é n no pior caso, sendo n o tamanho do texto.
- ***searchConclu***: esta operação tem como principal funcionalidade listar todas as conclusões que têm a palava dada. A operação *searchConclu* recebe como argumentos dois apontadores (um para a palavra a procurar e outro para uma struct do tipo Historia). A complexidade da operação é n no pior caso, sendo n o tamanho do texto.
- ***searchTexto***: esta operação tem como principal funcionalidade listar todos os títulos e os nomes dos ficheiros cujas histórias contêm a palavra no texto. A operação *searchTexto* recebe como argumentos dois apontadores (um para a palavra a procurar e outro para uma struct do tipo Historia). A complexidade da operação é n no pior caso, sendo n o tamanho do texto.

3. Espaço Ocupado Pelo Sistema de Pesquisa:

- ***Espaço Ocupado em Disco***:

Dado que o programa não escreve directamente no disco, apenas recebe dados, o sistema em si não ocupa espaço no disco. No entanto, os ficheiros que são abrangidos pela pesquisa ocupam sizeof(char)*nCaracteres cada.

(*Nota:* Dado que não são guardados valores em disco durante a execução do sistema, considera-se apenas o espaço ocupado pelos ficheiros onde consta o código necessário ao funcionamento do mesmo e o espaço ocupado pelo makefile.)

- ***Espaço Ocupado em Memória***:

O sistema ocupa nFicheiros*(6*sizeof(pointer)+nCaracteres) + nPalavrasNãoRepetidas * (sizeof(char) *30 +1) em disco, ou seja, o espaço da B-Tree preenchida, e o espaço de cada história (sendo que cada história é composta por 5 apontadores e pelo texto + o apontador para a própria história).