Relatório

Estruturas de Dados e Algoritmos II

23/04/2016

Trabalho realizado por:
Marcelo Babau Nº 30372
Paulo Martins Nº 30218

Introdução

Neste trabalho foi-nos proposto desenvolver um sistema de benchamark para tipos abstratos de dados do tipo árvore como a **Trie** e a **Btree**, devendo ter em atenção ao desempenho de **inserção**, **remoção** e **pesquisa** de um número relativamente grande de elementos.

A trie, dada e testada nas aulas, suporta operações de criar uma nova, inserir, remover e pesquisar strings. Também se pode chamar de prefix tree. As operações insere remove e pesquisa têm complexidade O(m), em que m é o tamanho da string.

A btree, árvores de pesquisa balanceadas com n filhos. Uma btree tem um grau t, então cada nó á excepção da raiz, terá entre t-1 e 2t-1 chaves e cada nó interno excepto a raiz terá entre t e 2t filhos.

Densolvimento

Para primeiro ponto do trabalho desenvolvemos uma função que recebe o nome do ficheiro a ser lido, e retorna uma "string" ou array de char's, a que chamamos buffer, com todas as palavras lidas do ficheiro.

```
char * leFicheiro(char *fileName){
    char *buffer;
    int size;
    file= fopen(fileName, "r");

    if(!file) {
      fputs("File error.\n", stderr);
      exit(1);
      }

    fseek(file, 0, SEEK_END);
    size = ftell(file);
    fseek(file, 0, SEEK_SET);
    buffer = malloc(size);
    fread(buffer, 1, size, file);

    fclose(file);
    return buffer;
}
```

E a partir dessa string com todas as palavras, a que chamamos, **buffer**, vamos realizar a inserção.

Para introduzir as palavras na estrutura, primeiro aplicamos o **strtok** no **\n** ao **buffer**, e depois percorremos todas as palavras e inserimos, parando no número definido no inicio do programa.

```
token = strtok(buffer, "\n");


int i=0;
while( token != NULL )
  {
    if(i==numero){
       break;
    }
    token = strtok(NULL, "\n")
    inseret(t, token);
    //printf( " %s\n", token );
    i=i+1;
  }
```

Os métodos utilizados para calcular o tempo total que demorou uma certa operação, em que colocamos o start antes da operação, e o end no fim, e subtrai-se para termos o tempo total.

```
clock_t start = clock();
// operação de inserir ou pesquisar ou remover
clock_t end = clock();
float seconds = (float)(end - start) / CLOCKS_PER_SEC;
```

Para encontrar as palavras repetidas na estrutura utilizamos a função para abrir e ler o ficheiro e guardar no buffer. Em seguida aplicar o **strtok** para termos cada palavra, chamar o metodo **existe** com essas palavras para saber se a palavra é encontra na estrutura.

```
token2 = strtok(buffer2, "\n");

int f=0;
int encontrou=0;
while( token2 != NULL )
  {
    if(f==numero/10){
       break;
    }


    token2 = strtok(NULL, "\n");
    if (existet(t, token2) == 1){
       encontrou+=1;
    }
    f=f+1;
  }
printf("Encontrou palavras %d \n" ,encontrou);
```

Para remover tudo, passamos como argumento cada palavra que esta na estrutura, proveniente do buffer.

```
char *token3 = strtok(buffer3, "\n"){
    int z=0;
   while(token3 != NULL && z < numero){
       token3 = strtok (NULL,"\n");
       trieremove(t, token3);
       z++;
   }
```

Conclusão

Output:

para 50000 palavras:

TRIE

Tempo a inserir 0.034340

Encontrou palavras- 2870

Tempo que demorou a encontrar 0.006402

Tempo a remover 0.006237

BTREE

Tempo a inserir 0.032792

Encontrou palavras- 2870

Tempo que demorou a encontrar 0.010526

Conseguimos quase a totalidade das funcionalidades a implementar, excepto a remoção por parte da btree.

Concluimos que a estrutura mais rápida a inserir é a btree por pouca diferença, mas a pesquisar a trie é bem mais rapida.

Para a remoção não temos tempo de comparação, pois só realizamos esta funcionalidade para a trie.