LABORATÓRIOS DE INFORMÁTICA III

Grupo 47

Síntese

Relatório sobre o trabalho pratico de C

António Chaves A75870
Carlos Campos A74745
Pedro Fonseca A74166

• Introdução

Neste relatório será abordado toda a nossa estratégia para resolver este trabalho pratico, desde o desenho da estrutura de dados, até a escolha de certos algoritmos para resolver as interrogações propostas no enunciado. Serão também abordados, bastantes factos e matérias relacionadas com a cadeira Algoritmos e Complexidade, mais especificamente a analise dos tempos assintóticos.

• Desenho da estrutura de dados

Visto que as três primeiras interrogações do enunciado, são só a apresentação de 3 números, decidimos fazer o calculo dessas 3 mesmas interrogações no carregamento dos dados, e assim quando fizéssemos a chamada das funções que nos dão essas respostas, apenas teríamos de apresentar esses números.

De modo a responder de forma logarítmica à maior parte das interrogações, decidimos guardar todos os artigos numa AVL, essa mesma ordenada em relação ao id dos artigos. Mas como em diferentes backup's, existem artigos com o mesmo id e com revisões diferentes, e já que existem algumas interrogações que se preocupam com o estado mais recente do artigo, como exemplo as interrogações 7 e 9, e porque no enunciado é dito que a extração dos dados, é feita respeitando a ordem temporal, decidimos guardar em cada nodo, uma lista ligada dos artigos com o mesmo id, em que a inserção nessa mesma lista é feita à cabeça, e assim o primeiro elemento é o artigo mais recente.

Por ultimo, decidimos fazer à parte, uma lista ligada dos colaboradores, especificamente para a interrogação do top 10 dos colaboradores com mais contribuições, em que a lista ligada está ordenada por contribuições de cada colaborador. Cada elemento da lista tem também uma arvore binaria de ids de revisão que correspondem às contribuições desse colaborador. Esta é a estrutura definida no ficheiro mystuct.c:

```c
typedef struct TCD_istruct
{
    TAD_artigos artigos;
    long numeroArtigos;
    long numeroArtigosUnicos;
    long numeroRevisoes;
    TAD_colaboradores colaboradores;
} TCD_istruct;
```

Esta é a estrutura da AVL de listas ligadas definida no ficheiro Artigos.c:

```c
typedef struct TCD_artigos
{
    long id;
    TAD_paginas paginas;
    TAD_artigos esq;
    TAD_artigos dir;
    int altura;
} TCD_artigos;

typedef struct TCD_paginas
{
    TAD_pagina cabeca;
    TAD_paginas prox;
}TCD_paginas;
```

Como vemos cada nodo tem id, um apontador para esquerda e para a direita, a altura, e um apontador para a lista ligada, em que cada nodo tem um apontador para um artigo que vai ser mostrado a seguir, e um apontador para o próximo elemento.

Esta é a estrutura do artigo referida, anteriormente definida no ficheiro Pagina.c:

```c
typedef struct TCD_pagina
{
    char* titulo;
    TAD_revisao rev;
} TCD_pagina;
```

Nota: pode haver uma certa confusão por parte do leitor, em relação a nomenclatura pagina, pagina é referente ao artigo inteiro, e não é uma pagina do artigo, este nome foi escolhido pelo facto que as tags dos backups que delimitam um artigo, são <page>, e assim serve também para distinguir da estrutura paginas, que por sua vez foi escolhido para se diferenciar da estrutura artigos.

Este é a estrutura da revisão, definida no ficheiro Revisão.c:

```c
typedef struct TCD_revisao
{
    long id;
    char* data;
    long nbytes;
    long npalavras;
    TAD_colaborador colab;
} TCD_revisao;
```

Visto que os textos ocupam bastante espaço na memoria, e como apenas precisamos do seu numero de bytes e o seu numero de palavras, para responder as interrogações 6 e 8 respetivamente, decidimos não guardar os textos, e calcular esses números no parsing dos mesmos.

Esta é estrutura do colaborador, definida no ficheiro Colaborador.c:

```c
typedef struct TCD_colaborador
{
    long id;
    char* username;
} TCD_colaborador;
```

E termina assim a definição em cadeia da AVL, é de notar que, temos uma estrutura apenas para os colaboradores, e como não existe nenhuma interrogação que peça a ligação entre um colaborador e uma revisão que tenha sido feita por si, era de esperar que não houvesse necessidade de guardar o colaborador na estrutura da revisão. Mas fizemos assim, para que seja mais fácil implementar futuras interrogações, como por exemplo, saber o autor de uma determinada revisão.

Esta é a estrutura da lista ligada dos colaboradores, definida em Colaboradores.c:

```c
typedef struct TCD_ids
{
    long idRevisao;
    TAD_ids esq;
    TAD_ids dir;
} TCD_ids;

typedef struct TCD_colaboradores
{
    long idColab;
    int contribuicoes;
    TAD_ids ids;
    TAD_colaboradores prox;
} TCD_colaboradores;
```

Termina assim a definição da nossa estrutura de dados.

• Algoritmos de resposta

Resposta nº1: para contar todos os artigos na nossa estrutura, temos que contar todos os elementos da lista ligada de cada nodo, como essa operação é linear de acordo com o numero de artigos existente, decidimos fazer esse calculo no load, e assim como só temos de apresentar um numero, esta resposta torna-se constante. **T(1).**

Resposta nº2: para contar os artigos únicos, basta calcular o numero de nodos da AVL, independentemente do numero de elementos das listas ligadas. Aqui também foi adotado o mesmo pensamento para a resposta nº1, sendo assim esta resposta também contante. **T(1).**

Resposta nº3: para contar o numero de revisões, o nosso raciocínio foi contar o numero de vezes que encontramos id's de revisão repetidos, e subtrair ao número total de artigos. Para saber as repetições, apenas temos de comparar um id de revisão, com o id do próximo elemento da lista ligada, onde nos encontramos. Esta resposta também é constante. **T(1).**

Resposta nº4: como é no load em que se faz a ordenação e preenchimento da lista de colaboradores, esta resposta limita-se a dar os 10 primeiros elementos da lista, sendo assim de tempo constante. **T(1).**

Resposta nº5: apesar de existir uma estrutura de dados dedicada para os colaboradores, respondemos a esta interrogação atravessando a AVL, visto que em termos de probabilidade, é muito provável encontrarmos o colaborador, apenas no lado esquerdo da AVL, visto que cada colaborador contribui varias vezes no mesmo snapshot. Tirando a parte em que se procura um autor que não exista, em termos de caso medio, esta resposta é quase logarítmica. **T(log N).**

Resposta nº6: o algoritmo usado para esta interrogação funciona de tal forma que, em cada lista ligada procura pelo texto mais comprido, ou seja, com mais bytes, e faz a inserção nos arrays de resposta, a começar pelo fim, em que puxa para trás todos os elementos dos arrays, até encontrar um numero maior, ou igual, mas que o id do texto seja menor, quando encontrada a posição insere os valores nos seus respetivos arrays. Esta resposta é de tempo linear consoante o numero total de artigos na AVL. **T(N).**

Resposta nº7: aqui nota-se claramente um bom uso da estrutura de dados, visto que, basta fazer a travessia pela AVL com o id do artigo fornecido, e basta fornecer a cabeça da lista do nodo encontrado. Assim o tempo desta resposta é logarítmica. **T(log N);**

Resposta nº8: o funcionamento desta resposta é idêntico à resposta nº6, em que cada lista ligada procura pelo texto com mais palavras, e insere no array, os N textos com mais palavras, sendo que o N, é fornecido pelo utilizador. Esta resposta também é de tempo linear consoante o numero total de artigos na AVL. **T(N).**

Resposta nº9: esta resposta, usufrui da propriedade da lista ligada, em que só compara o titulo da cabeça da lista, e se o titulo tiver prefixo que é fornecido, o array é realocado. Esta função é linear, mas é em relação ao numero de artigos únicos. **T(N).**

Resposta nº10: esta função faz a travessia por ordem em relação ao identificador do artigo, em seguida procura na lista ligada até encontrar a revisão. Esta função, a nível assintótico é **T(N*log N).**

• Analise de tempos

```
init() -> 0.000000 ms
load() -> 15357.316000 ms
all_articles() -> 0.000000 ms
unique_articles() -> 0.000000 ms
all_revisions() -> 0.000000 ms
top_10_contributors() -> 0.002000 ms
contributor_name(28903366) -> 0.021000 ms
contributor_name(194203) -> 0.001000 ms
contributor_name(1000) -> 3.287000 ms
top_20_largest_articles() -> 3.322000 ms
article_title(15910) -> 0.001000 ms
top_N_articles_with_more_words(30) -> 3.361000 ms
article_title(25507) -> 0.001000 ms
article_title(1111) -> 0.001000 ms
titles_with_prefix(Anax) -> 1.474000 ms
article_timestamp(12,763082287) -> 0.002000 ms
article_timestamp(12,755779730) -> 0.000000 ms
article_timestamp(12,4479730) -> 0.000000 ms
clean() -> 11.012000 ms
```

Como esperado, o load é a função mais lenta, visto que para além de fazer o parsing dos backups, faz também o calculo de 3 interrogações e a ordenação da lista de colaboradores. Sendo que as respostas relacionadas são de tempo constante. A topN e a top20, destacam-se como as funções menos rápidas, tal como esperado no tópico anterior, mas estamos falando da casa dos 3 milissegundos.

Em seguida vem função dos títulos com determinado prefixo, sendo que esta, como as duas top anterior, são de tempo linear, mas esta é em relação ao numero de artigos únicos, o que é um numero muito menor do que o numero de artigos totais, daí o tempo ser cerca de um terço menor. Já que existem mais ou menos 3 artigos para 1 artigo único.

A função que retorna o nome de um autor tem especial destaque, porque como previsto, para autores que existam na AVL, possui tempos como as outras funções logarítmicas, mas isso já não acontece para autores que não existam, tendo que percorrer todos os elementos de todas as listas ligadas, de todos os nodos da AVL, dai possuir um tempo semelhante as topN e top20.

• Conclusão

Concluindo, a nível de pontos fracos, existe uma dependência, da aplicação em relação ao parsing dos backups por ordem cronológica, caso isto não aconteça, algumas respostas podem até ser erradas, como a interrogação 7. Mas visto que essa inserção respeita sempre a sua ordem natural, como é dito no enunciado, não haverá grande problema.

Existe também uma performance relativamente mais lenta em relação as respostas de tempo linear, mas visto que este trabalho foi sempre testado com 3 snapshots, em que existem dezenas de milhares de artigos, mesmo que cresça muito mais o numero, será difícil de fazer essas mesmas apresentar tempos na casa dos segundos.

Em contrapartida, existem algumas respostas de tempo constante, e a maioria de tempo logarítmico, o que sugere que independentemente do numero de artigos, os seus tempos serão sempre muito reduzidos.

Por fim, outro dos pontos fortes deste trabalho, é a sua modularidade e o seu forte encapsulamento, visto que cada objeto foi programado como seria programado em java, o que dá uma certa sustentabilidade, e segurança, o que permite alterar partes importantes do programa, sem ter que mudar o código fonte em vários lados, mesmo não sabendo o conteúdo de certos objetos de estudo.