Relatório da 2º Fase do Trabalho Prático de

Estruturas de dados e Algoritmos II

Junho de 2020

José Santos (nº43017) Miguel Azevedo (nº36975)

O programa deverá ser avaliado no Problema T.

1 Estruturas de dados

Neste trabalho, escolhemos utilizar como estruturas de dados uma tabela de *Hash*. A razão pela qual nos levou a decidir esta estrutura, foi a vantagem de ter acesso direto a cada um dos seus elementos, visto que os países têm um identificador único.

De forma a reduzir o número de colisões, achamos apropriado utilizar um fator de carga inferior a 0.7. Para isso a Tabela de *Hash* de países terá um tamanho de 1019.

A Tabela tem como elementos *Struct* país, esta estrutura é constituída por um *char* [] ID, composto por 3 caracteres, 2 para constituir o ID mais 1 para o caracter nulo (3 *Bytes*), três *unsigned int*, estudantes_ativos, abandonou e terminou (12 *Bytes*). Indicando assim o número de estudantes que estão ativos, que abandonaram e terminaram respetivamente, esta estrutura ocupa no total 15 *Bytes*.

A Tabela de *Hash* estará localizada em memória central e ocupará

15*1019 = 15 285 *Bytes*.

Temos também uma *Struct* estudante, sendo que esta servirá para armazenar as informações relativas a cada estudante. É constituída por um *char* [] id, composto por 7 caracteres, 6 para o ID do estudante mais 1 para o caracter nulo (7 *Bytes*), outro *char* [] pais que identifica o país a que o estudante está associado (3 *Bytes*), e ainda 3 *Booleanos*, ativo, terminou e existe, indicando se o estudante está ativo, se já terminou o curso ou se existe, respetivamente (3 *Bytes*), esta estrutura ocupa no total 13 *Bytes*.

Para armazenarmos a informação relativamente aos estudantes iremos utilizar uma "simulação" de uma Tabela de *Hash* em memória secundária. Esta tabela terá dimensão 18000041 com um fator de carga inferior a 0.6 e irá ocupar 18000041*(13+3) = 288 000 656 *Bytes*.

Isto é uma das alterações feitas em relação à primeira fase do trabalho, devido à possibilidade de existir excesso de memória central utilizada, e eliminar a necessidade de percorrer, a cada execução do programa, o ficheiro dos estudantes para gerar os indicadores.

Em ambas as Tabelas de Hash é utilizado *DuploHashing* como tratamento de colisões.

2 Ficheiro de Dados

Iremos utilizar dois ficheiros, um para guardar os países e outro para os estudantes.

Ficheiro Países, é constituído por *Struct* Paises:

País 1	País 2	País 3	País 4
char [] id unsigned int estudantes_ativos unsigned int abandonou unsigned int terminou	char [] id unsigned int estudantes_ativos unsigned int abandonou unsigned int terminou	…	…

As estruturas estarão organizadas sequencialmente.

O tamanho deste ficheiro é variável e irá ocupar, no máximo, 16*676 = 10 816 *Bytes*.

Ficheiro Estudantes, é constituído por Struct Estudantes:

Estudante 5		Estudante 3	Estudante 7	
char [] id char [] pais bool ativo bool terminou bool existe		char [] id char [] pais bool ativo bool terminou bool existe	char [] id char [] pais bool ativo bool terminou bool existe	

Este ficheiro será uma "simulação" de uma Tabela de *Hash*.

Para o seu funcionamento, o estudante é inserido consoante o *HashCode* gerado pelo seu *id.*

O tamanho deste ficheiro é variável, e no máximo irá ocupar 16*18000041= 288 000 656 *Bytes*.

3 Operações

Para a execução das operações foram criadas funções auxiliares, tais como:

- procPosEstudanteDISK (char *estudanteID, FILE *file) – esta função tem como objetivo, dado um estudante, descobrir a posição onde este está localizado no ficheiro, ou, se ainda não existir, que posição irá ocupar. Começa por calcular o *HashCode*, e é lido o estudante nessa posição. Se o *id* lido for "\0" significa que não está ocupada, e, portanto, esta será a posição onde será inserido. Caso o *id* do estudante que estamos a ler não for "\0" significa que, ou encontramos o estudante, ou houve uma colisão. Se for o estudante, é devolvida a sua posição. Se não for, através de *DuploHashing*, repete-se o processo, até ser encontrado ou o *id* ser "\0", devolvendo, por fim, essa posição.
- getEstudanteDISK (unsigned int pos, FILE *file) – esta função tem como objetivo, dada uma posição, devolver o estudante do disco associado a esta posição pos.
- insereEstudanteDISK (Struct estudante *estudante, FILE *file, unsigned int pos) – esta função tem como objetivo inserir o estudante na posição do ficheiro.

1. Introduzir um novo estudante

Nesta operação é recebido, através do *scanf*, um identificador de um estudante e o código de um país. Com o identificador recebido, obtém-se, através da função *procPosEstudanteDISK,* uma posição do disco, e, de seguida, por meio da função *getEstudanteDISK*, obtém-se o estudante que está nessa mesma posição. Assim, verifica-se, através da estrutura lida do disco, se o booleano existe está a *true* e se o *id* é igual ao recebido pelo *scanf*, verificando, assim, se o estudante já existe ou não.

Caso o estudante não exista, é criado um estudante com aquele identificador e com o código do país. Através da função *hash_procPais*, obtemos o país onde queremos adicionar o estudante. Se o país não existir, é criado um país com o código recebido, é incrementada a variável estudantes_ativos, e insere-se na Tabela de *Hash* de países. Contudo, se existir é apenas incrementada a variável. Por fim, através da função *insereEstudanteDisk*, insere-se o estudante na posição de disco que tinha sido obtida anteriormente.

2. Remover um identificador

Nesta operação é recebido, através do *scanf*, um identificador de um estudante e o código de um país. Com o identificador recebido, obtém-se, através da função *procPosEstudanteDISK,* uma posição do disco, e, de seguida, por meio da função *getEstudanteDISK*, obtém-se o estudante que está nessa mesma posição. Assim, verifica-se, através da estrutura lida do disco, se o booleano existe está a *true* e se o *id* é igual ao recebido pelo *scanf*, verificando, assim, se o estudante já existe ou não.

Caso o estudante exista, é verificado se está ativo. Se for o caso, são alterados os booleanos existe e ativo, do estudante, para *false*. Decrementa-se os estudantes ativos daquele país e, através da função *insereEstudanteDisk*, atualiza-se o estudante no disco na mesma posição recebida anteriormente.

Se o estudante existe, mas não está ativo, significa que já terminou ou abandonou o curso.

3. Assinalar que um estudante terminou o curso

Nesta operação é recebido, através do *scanf*, um identificador de um estudante e o código de um país. Com o identificador recebido, obtém-se, através da função *procPosEstudanteDISK,* uma posição do disco, e, de seguida, por meio da função *getEstudanteDISK*, obtém-se o estudante que está nessa mesma posição. Assim, verifica-se, através da estrutura lida do disco, se o booleano existe está a *true* e se o *id* é igual ao recebido pelo *scanf*, verificando, assim, se o estudante já existe ou não.

Caso o estudante exista, é verificado se está ativo. Se estiver, altera-se os booleanos, ativo e terminou, do estudante, para *false* e *true*, respetivamente. Incrementa-se os estudantes que terminaram o curso daquele país e decrementa-se os estudantes ativos. Posteriormente, voltamos a escrever no disco o mesmo estudante com a informação atualizada.

Se o estudante existir, mas não estiver ativo, significa que já terminou ou abandonou o curso.

4. Assinalar o abandono de um estudante

Nesta operação é recebido, através do *scanf*, um identificador de um estudante e o código de um país. Com o identificador recebido, obtém-se, através da função *procPosEstudanteDISK,* uma posição do disco, e, de seguida, por meio da função *getEstudanteDISK*, obtém-se o estudante que está nessa mesma posição. Assim, verifica-se,

através da estrutura lida do disco, se o booleano existe está a *true* e se o *id* é igual ao recebido pelo *scanf*, verificando, assim, se o estudante já existe ou não.

Caso o estudante exista, é verificado se está ativo. Se estiver, altera-se o booleano ativo, do estudante, para *false*. Incrementa-se os estudantes que abandonaram o curso daquele país e decrementa-se os estudantes ativos. De seguida, voltamos a escrever no disco o mesmo estudante com a informação atualizada.

Se o estudante existir, mas não estiver ativo, significa que já terminou ou abandonou o curso.

5. Obter os dados de um país

Nesta operação é recebido, através do *scanf*, o código de um país. Com esse código é procurado o respetivo país na sua Tabela de *Hash,* e, caso exista, é realizado o output dos dados relativos ao país.

Das operações 1 até à 4, os acessos feitos ao disco são os mesmos. Um por cada função *insereEstudanteDISK* e *getEstudanteDISK*. Na função *procPosEstudanteDISK,* o número de acessos pode variar consoante o número de colisões que existirem.

Na operação 5, não existe qualquer acesso ao disco.

4 Início e fim da execução

No início da execução do programa, é aberto o ficheiro que contém os países, se já existir, senão, é criado. O mesmo acontece para o ficheiro com os estudantes.

Se o ficheiro com os países existir, são carregados todos os países que lá estão e são inseridos na sua Tabela de Hash, reduzindo, deste modo, o número de acessos ao disco por parte dos países.

No final da execução do programa, é reescrito em disco todas as posições ocupadas na Tabela de Hash, fechando, assim, o ficheiro dos países. Relativamente ao ficheiro dos estudantes, basta fechar o ficheiro, dado que todas as alterações são imediatamente escritas em disco.

5 Bibliografia

A função de *Hash* não é da nossa autoria e foi consultada para a realização do trabalho através deste link http://www.cse.yorku.ca/~oz/hash.html

A função de *Hash* utilizada foi a djb2.