Relatório da 1º Fase do Trabalho Prático de Estruturas de dados e Algoritmos II

José Santos (nº43017) Miguel Azevedo (nº36975)

Maio de 2020

1 Estruturas de dados

Neste trabalho escolhemos utilizar como estruturas de dados duas Tabelas 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 tanto os estudantes como os países têm um identificador único, este pode ser utilizado para calcular o *HashCode*. Também devido ao número elevado de estudantes, achamos adequado o uso desta estrutura.

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 Estudantes terá um tamanho de 15 000 017 (número primo mais próximo de 15 000 000). Para a Tabela de Hash dos países (676 que é o numero de combinações possíveis com 2 letras) terá um tamanho de 1019 (primo mais próximo de 1014).

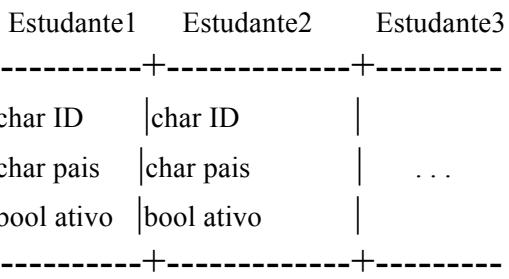
Através dos estudantes retiramos o ID do país, com este, geramos o HashCode para aceder a tabela países para obter/modificar informação de um determinado país.

As tabelas estão localizadas em memória central. A tabela de Hash dos estudantes, não vai conter a informação diretamente, mas sim apenas um inteiro que aponta para a posição de memória secundária onde a respetiva informação irá estar. Em relação à tabela *país*, esta terá estruturas *país* que irão ter, um *char ID* único de cada país (3 bytes), e *três inteiros* (15 bytes). Indicando o número de estudantes ativos, estudantes que desistiram e estudantes que completaram o curso, respetivamente. Dando assim um total de 15 bytes por cada estrutura.

Como a estrutura *estudante_simples* vai conter apenas um indicador para memória secundária, a tabela de Hash dos estudantes vai ocupar 15 000 017 x 4 (bytes) = 60 000 068 bytes. Para a tabela de países vamos alocar espaço para 1019 posições logo 1019 x 15 (bytes) = 15285 bytes. Dando um total de 60 051 353 bytes (aproximadamente 60 MegaBytes).

2 Ficheiros de dados

O ficheiro com as estruturas *estudante* contém um *char ID*, um *char Pais* e *um booleano* ativo.

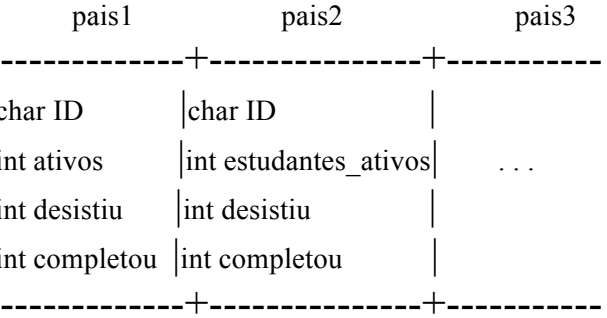


O estudante tem dois *char's* e um booleano:

ID é um *array* de 6 caracteres, juntando ainda o caracter nulo no fim, ficando assim um *array char* de 7 posições. Serve como identificador único de cada estudante.
Pais é um array de 2 caracteres, juntando ainda o caracter nulo no fim, ficando assim um *array char* de 3 posições. Serve para associar o estudante a um determinado país.
Ativo tem como função indicar se o estudante ainda está ativo no ensino superior. Este irá estar *false* caso o estudante desista, ou acabe o curso. É inicialmente inicializado a *true*.

Como cada ***char*** ocupa 1 *byte*, o char ID juntamente com o **char** pais ocupam 7 + 3 = 10 *bytes*.
Um **booleano** ocupa 1 *byte*, fazendo assim um total de 10 + 1 = 11 *bytes* por estrutura estudante.
Visto que o tamanho do ficheiro é constante e calculável (10 000 000 x 11 = 110 000 000) irá ser necessário por volta de 110 MegaBytes de memória secundaria para guardar no máximo 10 milhões de estudantes.

O outro ficheiro com as estruturas *pais* contém um char ID, um *int* estudantes_ ativos, um *int* desistiu e um *int* completou.



O pais tem um *char* e 3 inteiros:

ID é um *array* de 2 caracteres, juntando ainda o caracter nulo no fim, ficando assim um *array char* de 3 posições. Serve como identificador único de cada país.
Ativos serve para contar o número de estudantes ativos de um determinado país. É inicializado a 0 e incrementado a cada estudante que é adicionado a um país, e decrementado quando algum estudante completa o curso ou desiste.
Desistiu serve para contar, o número de estudantes que desistiram do ensino superior, de um determinado país. É inicializado a 0 e incrementado a cada estudante que desiste.
Completou serve para contar, o número de estudantes que completaram o ensino superior, de um determinado país. É inicializado a 0 e incrementado a cada estudante que completa o curso.

Como cada *inteiro* ocupa 4 bytes logo 4 x 3 = 12 *bytes*. Adicionando ainda os 3 bytes do ID dando assim um total de 15 *bytes* por estrutura.
Visto que o tamanho é constante e calculável, tendo no máximo 676 países, iremos necessitar de 676 x 15 = 10140 *bytes* em memória secundária.
Este ficheiro apenas irá ser aberto para leitura no inicio do programa, carregando assim as estruturas para a respetiva tabela de Hash, e no final do programa irá escrever a informação atualizada de cada país fechando assim o ficheiro.

3 Operações

1. Introduzir um novo estudante

A inserção de um novo estudante na tabela de Hash é feita através dos seguintes passos:

1. Criação de uma estrutura estudante, com o *ID* e *país* dados e ainda colocando o *ativo* a *true*.
2. É gerado o respetivo *HashCode*, para saber a posição na Tabela de Hash que o respetivo estudante vai estar. Na tabela de Hash, vão estar apenas estruturas *estudante_simples*, estas contêm um indicador para a posição onde a estrutura estudante está no disco. Sendo assim, caso existem colisões vamos observar as posições do disco indicadas pela nossa tabela de Hash *estudantes_simples* e comparar o *ID* com o recebido. Se o *ID* não corresponder, vai ser criada então uma estrutura *estudante_simples* com um novo indicador que aponta para a posição de disco, e colocando a nova estrutura estudante nessa posição de memória secundária.
3. Através do país dado, vamos calcular outro *HashCode* para colocar na tabela de *Hash* de países. Caso o país não estiver na tabela, criamos uma estrutura país com o ID correspondente, também incrementando o *estudantes_ativos*. Se o país estiver já estiver na tabela, apenas iremos incrementar os *estudantes_ativos* associados ao mesmo.

2. Remover um identificador

1. Utilizando o *ID*, vamos a posição correspondente na tabela de Hash estudante, que nos indicará a posição do disco onde irá estar o respetivo estudante, caso a posição da tabela hash esteja vazia, este estudantes não existe.
2. Carregamos a estrutura estudante para memória principal, para comparar com o *ID* (caso apenas seja uma colisão).
3. Quando encontramos a estrutura, através do país, vamos a gerar um HashCode com essa componente, e vamos a tabela de hash país a posição correspondente.
4. Ao encontrarmos o país, iremos decrementar o numero de estudantes ativos correspondentes ao mesmo.
5. De seguida, libertamos a posição na tabela de Hash estudantes e também a posição do disco associada.

3. Assinalar que um estudante terminou o curso

1. Utilizando o *ID*, vamos a posição correspondente na tabela de Hash estudante, que nos indicará a posição do disco onde irá estar o respetivo estudante, caso a posição da tabela hash esteja vazia, este estudantes não existe.
2. Carregamos a estrutura estudante, colocando a variável ativo a *false*, e guardando o país associado ao estudante, voltando a colocar a estrutura no disco.
3. Através do país vamos à tabela de Hash pesquisando pelo mesmo, para decrementar o numero de estudantes ativos, e incrementando o numero de estudantes que completaram o ensino superior.

4. Assinalar o abandono de um estudante

1. Utilizando o *ID*, vamos a posição correspondente na tabela de Hash estudante, que nos indicará a posição do disco onde irá estar o respetivo estudante, caso a posição da tabela hash esteja vazia, este estudantes não existe.
2. Carregamos a estrutura estudante, colocando a variável ativo a *false*, e guardando o país associado ao estudante, voltando a colocar a estrutura no disco.
3. Através do país vamos à tabela de Hash pesquisando pelo mesmo, para decrementar o numero de estudantes ativos, e incrementando o numero de estudantes que abandonaram o ensino superior.

5. Obter os dados de um país

1. Utilizando o *ID*, vamos a posição correspondente na tabela de Hash país, caso a posição da tabela hash esteja vazia, este país não existe.
2. Quando o ID recebido corresponder ao ID do pais, mostramos as variáveis estudantes_ativos, completaram e abandonaram.

A complexidade temporal de todas estas operação irá ser a mesma, visto estarmos a trabalhar com uma tabela de Hash. Isto indica-nos uma complexidade O(1), sendo no pior caso O(n) em que ***n*** será o número de colisões de cada elemento. Todas as outras operações que não envolvam a tabela de Hash serão de custo constante.

Como algoritmo de Hash utilizaremos o djb2*. As colisões serão tratadas linearmente.

4 Inicio e fim da execução

No início da execução do programa, é aberto os ficheiros que contém as estruturas, se já existir, senão, são criados os ficheiros.

Através do *ID* de cada estrutura, recalculamos o *HashCode* para ambas as tabelas, países e estudantes.

No caso da tabela estudantes, criamos uma estrutura *estudante_simples* para cada uma, indicando assim apenas a posição do disco em que as mesmas estão.

No caso da tabela países, colocamos as estruturas na tabela de Hash.

No fim da execução do programa, atualizamos a informação dos países em disco, e fechando ambos os ficheiros.

5 Bibliografia

*http://www.cse.yorku.ca/~oz/hash.html