UNIVERSIDADE DE ÉVORA

Engenharia Informática
Estrutura de Dados e Algoritmos II

Trabalho Prático - 1ªFase

Professor: Vasco Pedro

Sarah Simon Luz 38116
Ana Ferro 39872

7 Maio, Ano Letivo 2019/2020

Índice

1 Estruturas de Dados **2**
1.1 HashTables 2
1.1.1 HashTable Alunos 3
1.1.2 HashTable Países 4

2 Ficheiros de dados **5**
2.1 Nós país 5
2.2 Nós aluno 6

3 Operações **7**
3.1 Introduzir um novo estudante 8
3.2 Remover um identificador 9
3.3 Assinalar que um estudante terminou o curso 9
3.4 Assinalar o abandono de um estudante 10
3.5 Obter os dados de um país 10
3.6 Complexidade das operações 11
3.7 Acessos ao disco 11

4 Início e fim da execução **12**

5 Bibliografia **13**

1 Estruturas de Dados

1.1 HashTables

A estrutura de dados principal escolhida para a resolução deste trabalho foi uma hash table. Uma hash table guarda os dados num formato de um array, onde cada valor tem o seu índice único, chamado de hash ou hash code, que foi criado através de uma hash function.

Vai ser criada uma hash tables, uma para guardar informação sobre país e uma estrutura semelhante, com uma *hash function* na memória principal que calcula um *"hash code"* que irá representar uma posição de memória no ficheiro, para guardar nós com a informação dos alunos mas também dos países.

Escolhemos esta estrutura de dados devido à facilidade de implementação e também pelas suas primitivas, em média, terem uma complexidade de O(1).

1.1.1 HashTable Alunos

Um aluno quando introduzido no sistema terá de ter: um id único de identificação, que consiste numa sequência de 6 caracteres, de entre letras maiúsculas e algarismos decimais; um código que identifica um país, numa sequência de duas letras maiúsculas; se está activo; se não estiver activo é necessário saber se já terminou ou se abandonou o ensino.

A informação dos alunos irá ser encapsulada numa struct com os respectivos parâmetros: *idaluno*, um array com 7 posições incluindo o carácter nulo; *idpais*, um array com 3 posições incluindo o carácter nulo; três booleanos, *ativo*, *concluido*, *abandonou*.

Para saber o espaço máximo que cada struct aluno irá ocupar é feita a seguinte conta:

$$7 + 3 + 1 + 1 + 1 = 13bytes$$

Mas este valor não irá ficar alinhado na memória, para tal temos de acrescentar mais 3 bytes, para ficar 16 bytes. Visto que pode haver até 10 000 000 (10 milhões) de alunos introduzidos no sistema, ficará num total de 160 000 000 bytes que em MB serão 160MB.

No entanto, devido às restrições da memória principal apenas são permitidos até 64 MB, não podendo ser possível guardar todas as structs de aluno na memória principal.

Para resolver o problema da falta de espaço na memória principal, decidimos guardar a informação dos alunos num ficheiro em memória secundária que irá funcionar como um array. A posição das struct aluno em memória será atribuída através de um método semelhante à estrutura de dados, hash table. Produzindo um *"hash code"* com o identificador do aluno usando uma *"hash function"*. Sempre que for necessário aceder a informação de um aluno iremos calcular de novo o seu hash code.

Visto que iremos tratar os dados em memória como uma *hash table*, é necessário saber o comprimento da mesma. O comprimento de uma *hash table* tem de ser 1.3 vezes o número máximo de elementos que realmente estarão na tabela e ao mesmo tempo tem de ser um número primo.

Como o número máximo de alunos é 10 000 000 e

$$10000000 * 1.3 = 13000000$$

Com o número primo mais próximo, 13 000 001, o comprimento será 13 000 001.

1.1.2 HashTable Países

É necessário saber a informação básica de todos os 195 países pertencentes à UNESCO, tal como: identificação do país, número total de alunos, número de alunos ativos, número de alunos que completaram os estudos, número de alunos que abandonaram os estudos.

Esta informação vai ser guardada no mesmo ficheiro onde está a informação dos alunos, usando o mesmo princípio de encontrar o índice com o identificador, mas neste caso, o identificador do país. Os dados de cada país estarão encapsulados numa struct, chamada de *pais*. A informação numérica será guardada em inteiros, que ocupam 4 bytes e serão necessários 4 variáveis para guardar a informação, para o identificador do país será utilizado um *array* com 3 posições. Cada struct de um pais irá ocupar:

$$4 + 4 + 4 + 4 + 3 + 13(alinhamento\ no\ ficheiro) = 32\ bytes$$

Existem 195 países pertencentes à UNESCO, mas o comprimento da hash table tem de ser 1.3 vezes maior que o máximo de casos que iremos adicionar e também tem de ser primo:

$$195países * 1.3 = 253.5$$

Sendo o número primo mais próximo, 257, logo o comprimento ficará 257.

Para diminuir os acessos ao ficheiro, os dados dos países também irão ser guardados na memória principal. Mas ocupando apenas, 19 bytes, pois não é necessário o alinhamentos da memória. Cada país irá ter a sua entrada numa *hash table* chamada de *paises*. O comprimento será de 257. O *hash code* será calculado usando o identificador de cada país. Sempre que é adicionado, removido ou alterado os dados dos alunos, o mesmo irá acontecer com os dados dos países em memória principal e no final da execução do programa as alterações serão adicionadas no ficheiro.

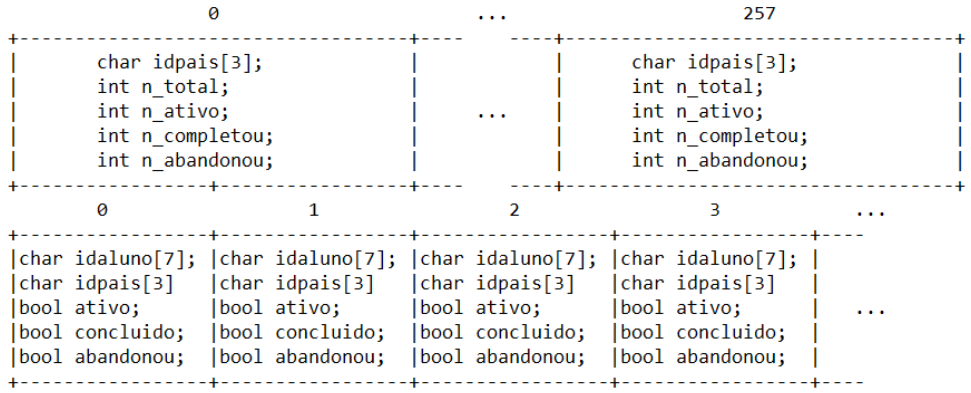
2 Ficheiros de dados

O ficheiro de dados contém uma lista de nós *país* e uma lista de nós *aluno*.

Na primeira execução do programa é criado um ficheiro binário vazio. O conteúdo do ficheiro consiste numa sequência de nós *país*, em que cada índice contém a informação de um país, ocupando os primeiros 8224bytes do ficheiro, seguido de uma sequência de nós *aluno*, em que cada índice contém a informação de um aluno.

Os nós *país* são escritos de 32 em 32 bytes e os nós *aluno* são escritos de 16 em 16 bytes para impedir problemas na leitura de elementos, evitando que informação fique repartida entre 2 páginas.

Figura ilustrativa do conteúdo do ficheiro



Neste caso, o tamanho do ficheiro é constante visto que retém dados de duas lista estáticas, logo conseguimos calcular o espaço que vai ser utilizado.

2.1 Nós país

Cada nó é constituído por:

- *idpais* - Código do país (tipo char);
- *n_total* - Número total de estudantes que frequentaram ou frequentam o ensino superior nesse país (tipo int) ;
- *n_ativo* - Número de estudantes ativos (tipo int);
- *n_completou* - Número de estudantes que completaram o curso (tipo int);
- *n_abandonou* - Número de estudantes que abandonaram o ensino (tipo int).

Feitas as contas cada nó *país* ocupa

$$4 + 4 + 4 + 4 + 3 = 19\ bytes$$

aos quais lhe acrescentamos 13 bytes perfazendo um total de 32 bytes. Multiplicando pelo máximo número de elementos (especificado na secção 1.1.2) obtemos

$$257 * 32 = 8224\ bytes < 1MB$$

2.2 Nós aluno

Cada nó é constituído por:

- *idaluno* - Identificador de aluno (tipo char);
- *idpais* - Código do país (tipo char);
- *ativo* - Indicador de aluno ativo ou desativo (tipo bool);
- *concluido* - Indicador que o aluno acabou o curso (tipo bool);
- *abandonou* - Indicador que o aluno desistiu do curso (tipo bool).

Feitas as contas cada nó *aluno* ocupa

$$7 + 3 + 1 + 1 + 1 = 13\ bytes$$

aos quais lhe acrescentamos 3 bytes para obter um alinhamento correto no ficheiro. Multiplicando pelo máximo número de elementos obtemos

$$13\ 000\ 001 * 16 = 208\ 000\ 016\ bytes$$

que equivale a 208MB, 2/5 do espaço do espaço total disponibilizado para a memória secundária(514MB).

3 Operações

Antes de verificar as operações é necessário falar das flags e dos seus significados. Existem três flags: *ativo*, *concluido* e *abandonou*. A flag *ativo* diz se o estudante está activo ou não. A flag *concluido* diz se o estudante já terminou ou não. A flag *abandonou* diz se o estudante abandonou ou não.

Se a flag *ativo* estiver marcada como *false*, pode significar várias coisas. Para saber qual o seu significado é necessário verificar as restantes flags.

1. Está desativado, mas a flag *concluido* está *true*. Significa que concluiu os estudos;

2. Está desativado, mas a flag *abandonou* está *true*. Significa que abandonou os estudos;

3. Está desativado e as outras duas flags também, significa que o aluno foi removido do sistema. E o seu identificador poderá ser reutilizado.

3.1 Introduzir um novo estudante

Dados o identificador de um estudante e o código de um país, a inserção de um novo estudante no ficheiro é feita executando os seguintes passos:

1. Calcular o *"hash code"* do identificador do aluno.

2. Procurar no ficheiro a posição calculada pela *"hash function"*, verificar se a posição está vaga:

   2.1. Em caso negativo, pode haver três opções possíveis:

   2.1.1. Os identificadores são iguais mas é possível reutilizar, é então encontrada uma posição vaga;

   2.1.2. Os identificadores são iguais mas não é possível reutilizar. Neste caso, não é possível adicionar o aluno e é enviada uma mensagem de erro.

   2.1.3. O identificador é diferente. Neste caso, irá ser feito um *"double hashing"* do identificador do aluno e os passos serão repetidas até chegar a uma passo terminal.

   2.2. Em caso positivo, é encontrada uma posição vaga.

3. Quando encontrada uma posição vaga, será criado uma struct aluno com os dados do aluno e adicionado no ficheiro na posição calculada, também serão alterados os dados do país correspondente, mas apenas na memória principal, sendo criada uma struct para esse mesmo país caso ainda não tenha sido feito e colocado na memória principal com o *hash code* calculado, ou *double hashing* até encontrar uma posição, se necessário.

3.2 Remover um identificador

Dado um identificador de um estudante ativo, o sistema apaga a informação relativa a esse identificador, que poderá voltar a ser usado. São feitos os seguintes passos:

1. Calcular o *"hash code"* do identificador de aluno;
2. Procurar no ficheiro a posição calculada pela *hash function*, verificar se a posição está vaga:

   2.1. Em caso afirmativo, não foi encontrado nenhum aluno com o identificador e é enviada uma mensagem de erro;

   2.2. Em caso negativo, é necessário verificar se o identificador é o desejado, se não for é necessário fazer *"double hashing"* e repetir os passos até chegar a uma passo terminal;
3. Quando o identificador idêntico é encontrado;

   3.1. Para o remover, as flags são todas passadas para *false*. E as alterações necessárias no país correspondente serão feitas.

3.3 Assinalar que um estudante terminou o curso

Dado um identificador de um estudante ativo, o sistema guarda a informação de que esse estudante completou o curso. O estudante deixa de estar ativo, mas o identificador não poderá ser reutilizado. São feitos os seguintes passos:

1. Calcular o *"hash code"* do identificador de aluno;
2. Procurar no ficheiro a posição calculada pela *hash function*, verificar se a posição está vaga:

   2.1. Em caso afirmativo, significa que não existe nenhum aluno com aquele identificador e é enviada uma mensagem de erro;

   2.2. Em caso negativo, verifica se o identificador no ficheiro na posição correspondente é igual ao identificador recebido, se não for repete o processo até encontrar o identificador desejado e que está activo;
3. Quando o identificador idêntico é encontrado;

   3.1. Para assinalar que o estudante terminou a flag *ativo* é marcada como *false* e a flag *terminou* é marcada como *true*. Sendo feitas as alterações necessárias no país correspondente.

3.4 Assinalar o abandono de um estudante

Dado um identificador de um estudante ativo, o sistema guarda a informação de que esse estudante abandonou o sistema de ensino. O estudante deixa de estar ativo, mas o identificador não poderá ser reutilizado. São feitos os seguintes passos:

1. Calcular o *"hash code"* do identificador de aluno;
2. Procurar no ficheiro a posição calculada pela *hash function*, verificar se a posição está vaga:

   2.1. Em caso afirmativo, significa que não existe nenhum aluno com aquele identificador e é enviada uma mensagem de erro;

   2.2. Em caso negativo, verifica se o identificador no ficheiro na posição correspondente é igual ao identificador recebido, se não for repete o processo até encontrar o identificador desejado e que está activo;
3. Quando o identificador idêntico é encontrado;

   3.1. Para assinalar que o estudante abandonou o ensino a flag *ativo* é marcada como *false* e a flag *abandonou* é marcada como *true*. Sendo feitas as alterações necessárias no país correspondente.

3.5 Obter os dados de um país

Dado o código de um país, o sistema mostra o número total de estudantes que frequentaram ou frequentam o ensino superior nesse país, o número de estudantes ativos, o número de estudantes que completaram o curso e o número de estudantes que abandonaram o ensino.

Sempre que um aluno é introduzido ou alterações são feitas a um aluno, os dados do país onde esse aluno estuda também são alterados na memória principal. Para obter esses dados, são feitos os seguintes passos:

1. Calcular o *hash code* do país;
2. Aceder à *hash table* com o *hash code* e ver se a entrada corresponde ao país:
   2.1. Se o identificador corresponder ao país recebido, aceder aos dados desejados e devolvidos na consola;

   2.2. Se não corresponder, repetir o processo até encontrar o país desejado, depois aceder aos dados desejado, devolvendo-os na consola;

   2.3. Se a entrada estiver vazia. Mandar uma mensagem de erro, pois o país não existe em memória.

3.6 Complexidade das operações

A complexidade destas operações é, em média, O(1), no entanto, este valor irá depender do número de colisões. No pior dos casos, teremos que percorrer a tabela toda até encontrar um espaço vazio (no caso de inserção de aluno) ou encontrar a posição correta (para as restantes operações), o que vai corresponder a uma complexidade de O(n).

Com o uso do algoritmo de hash *djb2* esperamos uma boa distribuição dos dados e, consequentemente, uma diminuição de colisões.

3.7 Acessos ao disco

Considerando que a leitura e escrita correspondem a um acesso ao disco cada uma, concluímos que, no melhor caso, isto é, quando não existe colisões, cada operação descrita acima à exceção da "Obter dados de um país"acede duas vezes ao disco, uma para verificar se a posição é a correta (leitura) e outra para realizar as alterações necessárias (escrita).

No pior caso, isto é, quando há colisões, cada operação acede $n+2$ vezes, sendo n o número de colisões e, consequente, leituras necessárias para verificar que se encontrou a posição correta, mais 2 acessos, já na posição certa, para verificação(leitura) e alteração de dados(escrita).

4 Início e fim da execução

No início da execução do programa, é aberto um ficheiro, caso ainda não tenha sido criado um, este é criado.

Se o ficheiro existe, é lido os primeiros 8224 bytes, que correspondem ao espaço utilizado pelos nós *país* (equivalente a 257 acessos ao disco). Esta leitura permite reconstruir a hash table países que passa a ficar carregada em memória principal. Ao guardar estes dados em memória principal permite-nos diminuir o tempo de execução, evitando acessos desnecessários à memória secundário.

No final da execução do programa, é necessário atualizar/reescrever os nós *país*. De seguida, o ficheiro é fechado, visto que todas as alterações relativamente aos nós *aluno* são feitas em disco ao longo do programa.

5 Bibliografia

- *http://www.cs.unc.edu/ plaisted/comp550/Neyer%20paper.pdf*
- *https://stackoverflow.com/questions/9214353/hash-table-runtime-complexity-insert-search-and-delete*