Relatório Fase 2 Trabalho EDAII

g24:
Luís Maurício - 37722
João Galvão - 37814

2019/2020

O nosso grupo é o g24 no mooshak e para efeito de avaliação usaremos o programa submetido em último no problema T, mais especificamente a submissão 221.

1 Alterações em relação à primeira fase

1.1 Hashtable Estudantes

Como descrito mais abaixo cada posição desta tabela ocupa 11 Bytes em vez dos 9 referidos no relatório da primeira fase, faltava contabilizar os bytes de alinhamento.
Além disso o campo referente ao estado passou a poder também representar um aluno Removido e Vazio, na primeira fase era só para Ativo, Termino ou Abandono.

1.2 Hashtable Países

No relatório da 1ª fase dissemos que esta hash table teria apenas 397 posições visto que só existem 195 países no mundo, mas ao chegar ao problema E reparámos que isso não era um fator e mudámos o número de posições para 1361, que é o número primo maior e mais próximo do dobro do número de permutações possíveis com 2 letras maiúsculas.
Ainda alterámos também a forma de escrever o conteúdo desta hash table em ficheiro, uma vez que só precisamos de ler e escrever o ficheiro no início e no fim, não interessa o ficheiro estar em forma de hash table, passou a ser só uma lista de países seguidos.

1.3 Acessos a ficheiros

O número de acessos a ficheiros no relatório da 1ª fase estava mal calculado, não tinha em conta que depois de se ler uma posição precisariamos de mais um acesso para reposicionar e mais um para escrever. Essa alteração foi feita e alguns sítios que referiam 2 acessos no melhor caso passaram a referir 4.

2 Estruturas Utilizadas

Para a implementação deste trabalho decidimos optar por utilizar hashtables como a única estrutura utilizada.
Para utilizar hashtables é necessário escolher uma função de hash que evite demasiadas colisões, escolhemos a função djb2 por ser para strings e mostrar resultados bastante diversos, e assim usamos a mesma função de hash tanto para identificadores de estudantes como para identificadores de países.

A escolha foi motivada por ter de escolher alguma estrutura que permita a procura rápida de um aluno específico de forma a saber se o código identificador já está em uso ou não. Da mesma forma é necesário encontrar rapidamente os dados correspondentes a cada país.

2.1 Hashtable Estudantes

Utilizámos uma hashtable em memória secundária para guardar a informação referente aos estudantes. Esta hashtable tem tamanho fixo visto que sabemos o número máximo de estudantes que o sistema chega a ter. Sabendo que o número máximo de estudantes é de 10 000 000 escolhemos 20 000 003 como o número de posições para esta hashtable visto que é o número primo maior que o dobro de estudantes mais próximo.

Cada posição ocupa 11 Bytes, sendo que é composta de 9 caracteres + 2 bytes de alinhamento. Os primeiros 6 caracteres correspondem ao identificador do aluno, os próximos 2 correspondem ao código do país e o último caracter representa o estado em que o aluno se encontra(Activo, Terminado, Abondono, Removido, Vazio).
Para encontrar um estudante basta aceder ao offset (multiplicando o código hash por 11) no ficheiro e ler as 11 posições seguintes.

Cada aluno vazio é representado por um identificador vazio, codigo de país vazio e estado 'E'(Empty). Assim permite funcionar como flag para avançar na hash table até encontrar uma posição vazia.

Já um aluno removido mantém o identificador e o código de país mas altera o estado para 'R'(Removido). Assim acontece que mesmo um estudante removido já tem uma posição da hash table que lhe pertence. Isto é, se voltar a introduzir um estudante com esse código apenas se sobrepõe ao que lá estava, enquanto que um estudante com código diferente mas que tem o mesmo hash code tem de procurar outra posição quadraticamente.

Table 1: Hashtable para guardar informação dos estudantes

Posição	Identificador	País	Estado
00 000 000	ABC123	PT	A
00 000 001	ABC124	CA	D
...	...	...	...
20 000 002	65VD1U	FR	T

2.2 Hashtable Países

Para evitar estar a percorrer todos os estudantes e contar quantos pertencem a cada país, utilizamos ainda uma hashtable que serve para guardar os países e o número de estudantes em cada estado associados a este. Esta hashtable é guardada em memória principal para evitar o número de acessos a disco na pesquisa dos dados de cada país, mas para manter a persistência entre utilizações do programa é também guardada em memória secundária.

Decidimos utilizar listas ligadas para lidar com colisões por sabermos que não estamos a usar muita memória principal visto que só os países e um único estudante se encontram nesta. Assim os bytes extra do ponteiro para um próximo não têm problema e também não se mete o impedimento de termos de percorrer muito em listas já que escolhemos uma boa função de hash e podemos sempre aumentar a capacidade da hash table de forma a reduzir colisões.

Cada elemento desta tabela é composto por o código de país (char[3]), 3 valores inteiros e um ponteiro, fazendo com que cada elemento ocupe 23B (3 + 4 + 4 + 4 + 8).
Os três valores inteiros representam o número de estudantes em cada estado associados ao país (Ativos, Terminos e Abandonos).
O ponteiro é o que se utiliza em caso de colisão numa posição.

Sabendo que os códigos de país contém 2 caracteres e o alfabeto é só letras maiúsculas(26), sabemos que o número máximo de permutações são 676 (26*26). Então escolhemos o número primo superior mais próximo do dobro de 676. Ou seja a tabela terá 1361 posições.

Com isto podemos calcular que a tabela ocupa cerca de 30.57KiB ou 31.3KB (1361 * 23B) em memória principal, sendo que pode aumentar até mais 23B * o número de colisões.

Table 2: Hashtable para guardar informação dos países

Posição	Código País	Ativos	Terminados	Desistentes	*next
0000	PT	223173	101806	56671	NULL
0001	US	184802	120306	12182	*Table 3
...	...	...	...	...	...
1360	CA	264941	215149	150364	NULL

Table 3: *Lista para colisões

Elementos	1	2	...	X
Código País	PT	TR	...	RW
Ativos	6453451	12435	...	54346
Terminados	2431432	77564	...	21543
Abandonos	6587	12354	...	32433
*next	*2	*3	...	NULL

3 Ficheiros de Dados

3.1 Ficheiro de estudantes

Table 4: Exemplo de ficheiro com informação dos estudantes

ABC123	PT	A	ABC124	CA	D	...	...	...	65VD1U	FR	T
0			1			...			20 000 002		

Cada posição da hashtable corresponde a 11 carateres seguidos, que representam os 3 campos seguintes:

ID: Contém o identificador do aluno que tem 6 caracteres mais um de alinhamento.

País: Contém o identificador do país em que o aluno frequenta/ou o ensino superior, tem 2 caracteres mais um de alinhamento.

Estado: Representa o estado do aluno em relação ao ensino ou serve de flag para vazios e removidos, neste caso terá 5 possiveis valores, A – Activo, D – Abandono (Letra D pois Desistência é sinónimo), T - Terminado, E - Vazio (E de Empty), R - Removido.

Este ficheiro tem um tamanho fixo de 209.8MiB ou 220MB. É fixo porque mesmo na sua criação são escritos 20 000 003 alunos com a flag de vazio, e o que acontece é substituir posições introduzindo dados numa posição já antes existente.

3.2 Ficheiro de países

Supondo que temos N países na hash table, o ficheiro é como no seguinte exemplo:

Table 5: Exemplo de ficheiro com informação dos países

PT	223173	101806	56671	*next	US	184802	120306	12182	*next	...
0					1					...

...	*	*	*	*	*	CA	264941	215149	150364	NULL
...	N-1					N				

Cada posição da hashtable corresponde à seguinte divisão:

País: Contém o identificador do país que tem 2 caracteres mais um de alinhamento.

Ativo: Contador de estudantes ativos o ensino superior.

Terminado: Contador de estudantes que terminaram o ensino superior.

Abandono: Contador de estudantes que abandonaram o ensino superior.

Este ficheiro não tem um tamanho fixo, depende do número de países existentes. É sempre 23B * o número de países existentes. Se suposermos o máximo de países como o máximo de permutações de duas letras maiúsculas (26*26) 676, este ficheiro terá um tamanho máximo de 15.18KiB ou 15.55KB (676 * 23B).

4 Operações

4.1 Inserir um novo Estudante

Ao inserir o código identificador de um estudante acontecem as seguintes operações:

1. Calcular o hashcode do código do estudante.
2. Calcular o offset da posição no ficheiro dos estudantes desse código.
   Caso essa posição esteja ocupada lidar com a colisão como explicado na descrição das estruturas e procurar na próxima possível.
3. Aceder a essa posição do ficheiro e escrever os dados do aluno.
   Caso o aluno exista na hashtable será emitido um output com a informação de que esse aluno já existe.
4. Calcular o hashcode do país.
5. Pesquisar esse hashcode na hashtable referente aos países.
   Caso não exista o país especificado, adicioná-lo à hashtable respetiva.
6. Incrementar o número de estudantes ativos nesse país.

O número de acessos ao disco é 4 no melhor caso, 1 para encontrar a posição correta, 1 para a leitura da mesma, 1 para o reposicionamento nessa posição e por último 1 para a escrita. Se tiver ocorrido alguma colisão o número de acessos para encontrar a posição irá aumentar até encontrar o identificador correto ou uma posição vazia.

Ambas as estruturas são hashtables e apenas acontece pesquisa e inserção, logo esta ação tem complexidade temporal de $O(n)$ no pior caso e $O(1)$ no melhor.

4.2 Remover um identificador

Para remover um identificador ocorrem as seguintes operações:

1. Calcular o hashcode do código do estudante.
2. Calcular o offset da posição no ficheiro dos estudantes desse código.
3. Aceder a essa posição do ficheiro e carregar os dados desse aluno para memória principal.
   Caso o aluno não exista na hashtable será emitido um output com a informação de que esse aluno não existe (logo não pode ser removido).
4. Substituir o aluno por ele mesmo mas com o estado 'R' para indicar que está removido, guardar o país a que o aluno pertence e o estado para fazer alterações ao país.
5. Na hashtable dos países decrementar o contador referente ao estado em que o estudante se encontrava.

Serão feitos 4 acessos a disco no melhor caso, 1 para encontrar e 1 para ler, 1 para reposicionar na posição encontrada e por último 1 para rescrever o aluno com a alteração no estado.

No pior dos casos esta ação tem uma complexidade temporal de $O(n)$ e no melhor caso $O(1)$ referente à pesquisa e inserção (mesmo que vazia) numa hashtable.

4.3 Assinalar que um estudante terminou o curso

Para assinalar que um estudante terminou o curso os passos para a execução são:

1. Calcular o hashcode do código do estudante.
2. Calcular o offset da posição no ficheiro dos estudantes desse código.
3. Aceder a essa posição do ficheiro e carregar os dados desse aluno para memória principal.
   Caso o aluno não exista na hashtable será emitido um output com a informação de que esse aluno não existe, caso o aluno já esteja com o estado de terminado será emitido um output com a indicação de que essa informação já está atualizada.
4. Alterar o estado do aluno para T(Terminado).
5. Substiuir os dados do aluno pelo novos atualizados.
6. Encontrar o país do estudante e incrementar o contador de estudantes Terminados, decrementando o contador referente ao estado anterior do aluno.

A complexidade temporal desta operação pode ser no melhor caso O(1) ou O(n) no pior caso, de acordo com a complexidade de pesquisa e inserção em hashtables.

O número de acessos ao disco é 4 no melhor caso, 1 para encontrar a posição, 1 para ler, 1 para reposicionar na posição depois de confirmar que é a correta e 1 para a escrita. Se tiver ocorrido alguma colisão o número de acessos para encontrar a posição irá aumentar até encontrar o identificador correto ou uma posição vazia.

4.4 Assinalar o abandono de um estudante

Para assinalar que um estudante abandonou o curso os passos para a execução são:

1. Calcular o hashcode do código do estudante.
2. Calcular o offset da posição no ficheiro dos estudantes desse código.
3. Aceder a essa posição do ficheiro e carregar os dados desse aluno para memória principal.
   Caso o aluno não exista na hashtable será emitido um output com a informação de que esse aluno não existe, caso o aluno já esteja com o estado de abandono será emitido um output com a indicação de que essa informação já está atualizada.
4. Alterar o estado do aluno para D(Desistência).
5. Substiuir os dados do aluno pelo novos atualizados.
6. Encontrar o país do estudante e incrementar o contador de estudantes Terminados, decrementando o contador referente ao estado anterior do aluno.

A complexidade temporal desta operação pode ser no melhor caso O(1) ou O(n) no pior caso, de acordo com a complexidade de pesquisa e inserção em hashtables.

O número de acessos ao disco é 4 no melhor caso, 1 para encontrar a posição, 1 para a ler, 1 para reposicionar na posição depois de confirmar que é a correta e 1 para a escrita. Se tiver ocorrido alguma colisão o número de acessos para encontrar a posição irá aumentar até encontrar o identificador correto ou uma posição vazia.

4.5 Obter os dados de um país

Para a obtenção dos dados do país os passos para a execução são:

1. Calcular o hashcode do código do país.
2. Pesquisar esse hashcode na hashtable referente aos países.
3. Retornar o código do país e os contadores referentes.

Para a obtenção dos dados, caso não exista o código referente irá ser dado um output indicando que o código pesquisado não se encontra na hashtable, ou seja não existem dados desse país.

A complexidade temporal desta operação pode ser no melhor caso O(1) caso seja encontrado à primeira, ou O(n) no pior caso em que haja colisões.

5 Início e fim da Execução

5.1 Início

Para começar inicializa-se a hash table presente em memória principal(países) para se poder depois fazer as leituras necessárias dos ficheiros para manter a persistência.
A leitura para o preenchimento desta hash table ocorre da seguinte forma:

Ficheiro já existe Lê-se o ficheiro país a país e introduz-se na hash table já inicializada, pode então fechar-se o ficheiro.

Ficheiro não existe Utiliza-se apenas a hash table que já está inicializada sem preencher posições nenhumas.

Para inicializar a hash table dos estudantes é um pouco diferente visto que esta hash table é o ficheiro mesmo e não tem nenhuma parte (a não ser o estudante em que se está a trabalhar atualmente) em memória principal. Esta inicialização/leitura ocorre da seguinte forma:

Ficheiro já existe Apenas abri-lo em modo 'r+' que permite a leitura do que já existe e a escrita sem apagar, e assim temos o ficheiro/hash table pronto a manipular.

Ficheiro não existe Abrir o ficheiro em modo 'w' para permitir criá-lo e depois preencher 20 000 003 posições com estudantes vazios.

5.2 Fim

No fim da execução o ficheiro dos estudantes apenas precisa ser fechado uma vez que todas as alterações que foram acontecendo foram diretamente feitas no ficheiro.

O ficheiro para manter a persistência dos países precisa de ser reescrito então é aberto em modo 'w' (de forma a apagar tudo o que houvesse antes) e um ciclo que corre toda a hash table dos países vai escrevendo um a um no ficheiro.

Depois disto podemos fechar o ficheiro e libertar o espaço que a hash table dos países tinha alocado em memória principal.

6 Bibliografia

Para o estudo das complexidades da Hash Tables:
`https://en.wikipedia.org/wiki/Hash_table`

Para cálculo de tamanhos e limites:
`https://www.tutorialspoint.com/cprogramming/c_data_types.htm`

Para manipulação de ficheiros:
`https://www.moodle.uevora.pt/1920/mod/page/view.php?id=34562`

Para exemplo de hash table em memória secundária
`https://github.com/rjcf18/Trabs_Exerc_Univ/tree/master/BSc/EDA2/TrabalhoEDA2/TrabEDA2Final/corrector_final`

Base para hash table com lista ligada:
`https://gist.github.com/phsym/4605704`

Função de Hash (djb2):
`http://www.cse.yorku.ca/~oz/hash.html`