Organização de Arquivos - Turma C
Projeto 01: Indexação de Arquivos e utilização de técnicas com Lista Invertida 2/2017

Gabriel Correia de Vasconcelos - 16/0120781
Universidade de Brasília
Campus Darcy Ribeiro - Departamento de Ciência da Computação
gcvasconcelos98@gmail.com

Pedro Augusto Nunes - 16/0141044
Universidade de Brasília
Campus Darcy Ribeiro - Departamento de Ciência da Computação
pedropacn@gmail.com

Abstract

Projeto de Organização de Arquivos com intuito de botar em prática os conceitos do procedimento de indexação de arquivos, tanto na organização de índices primários, quanto secundários. Para isso foram utilizadas técnicas de lista invertida e ordenação heapsort, conceitos aprendidos em sala de aula. As implementações foram feitas em dupla e foi utilizada a linguagem Python como ferramenta de desenvolvimento.

1. Introdução

O processo de indexação de arquivos consiste na organização de uma lista de registros por meio de índices, para facilitar seu acesso e ordenação. Um índice simples é formado por uma chave primária, que identifica cada registro unicamente, de forma que nunca se repete, e por um campo de referência, que indica a posição de seu respectivo registro no arquivo de dados.

Este mecanismo viabiliza a ordenação de séries de registros na memória primária, sem a necessidade de rearranjá-los, tornando a operação menos custosa. Outro ponto positivo da indexação é permitir o acesso direto, via chave, à posição de registros no arquivo de dados. Desta maneira, a busca sequencial se limita apenas ao arquivo de índices, no qual ocorre uma pesquisa binária.

Os índices são inicialmente armazenados em um *array*, na memória primária, em que cada registro possui um índice correspondente. Para se garantir sua persistência, os dados deste *array* são armazenados no chamado arquivo de índices na memória secundária. Este arquivo é consideravelmente menor que seu arquivo de dados referente e é sempre formado por registros de tamanho fixo, facilitando bastante sua manipulação, visto que as operações são menos custosas.

A operação de inserção de um registro funciona de forma que, a cada inclusão no arquivo de dados, é adicionado seu índice corresponde no arquivo de índices, formado pelo par chave primária e campo de referência. Esta inclusão pode acontecer tanto no final do arquivo, como em algum espaço vazio no meio dele. Sempre que um registro é adicionado, o arquivo de índices é mantido ordenado. Este, inteiramente na memória primária, não necessita de acessos ao arquivo de dados. A operação de exclusão é similar a de inserção, mas de forma que elimina tanto o registro no arquivo de dados, quanto seu índice no arquivo de índices e ainda reordena o arquivo.

Já para se atualizar algum registro, existem dois cenários: o que há mudança na chave primária e no que não acontece esta mudança. Caso algum campo que forme a chave primária seja modificado, é necessário atualizar o arquivo de índices, excluindo o índice antigo e adicionando o índice modificado. O registro no arquivo de dados é modificado normalmente. Agora se não há alguma mudança que afete a chave primária e se a alteração cabe no registro, nenhuma mudança é feita no arquivo de índices. Se não cabe, deve-se excluir o registro do arquivo de dados e colocar o novo registro em algum lugar que o caiba. Também é necessário alterar o campo de referência do registro no arquivo de índices.

Apesar de mais fáceis, buscas por uma chave primária são mais raras. A busca mais comum utiliza um tipo de chave se caracteriza por poder ser associada a mais de um

registro, as chaves primárias. Uma solução para esta situação é a implementação de índices secundários. Este recurso permite a visualização de um mesmo arquivo de diferentes formas, visto que pode-se combinar e buscar suas uniões e intersecções de resultados.

O arquivo de índices secundários tem registros fixos, que contém, para cada registro do arquivo de dados, um índice formado por sua chave secundária e um campo de referência para sua a chave primária correspondente, e não a sua posição. Dentre os problemas desse método esta a repetição de chaves secundárias, ocasionando perda de espaço, e a reorganização desnecessária do arquivo de índices. Uma das técnicas para se solucionar este problema é a aplicação de listas invertidas. Neste procedimento, o arquivo de índices secundários é formado pela chave secundária e por um *array* de campos de referência para aquela chave secundária. Desta forma, as operações realizadas no arquivo de índice secundário são facilitadas.

Para a operação de adição, agora são adicionados os índices secundários das entradas nos arquivos de índices secundários. Na remoção, são removidas todas as referências, tanto no arquivo de índices primários, quanto nos secundários, e reorganizar os remanescentes. Na operação de atualização existem três cenários: se a chave secundária muda, o arquivo de índices secundários deve ser reordenado; se a chave primária muda, primeiramente o arquivo de índices primário é reordenado e depois o arquivo de índices secundários é atualizado; se mudar outro campo qualquer, nada é afetado se o campo alterado cabe no espaço do registro, caso contrário, há então um exclusão seguida de uma inclusão.

As operações de ordenação aqui citadas, utilizam a técnica do *heapsort*. Esse algoritmo faz parte da família de ordenação por seleção, e utiliza a estrutura da *heap* para ordenar os índices do arquivo de índices com base em sua chave primária, a medida que são inseridos novos índices. A *heap* pode ser implementada como uma árvore quase completa, binária e com algumas propriedades especiais, ou como um vetor, onde cada nó n tem os filhos $2n$ e $2n + 1$ e pai $\frac{n}{2}$. Pode ser de mínimo (ordenação decrescente) ou de máximo (ordenação crescente), tal que a raiz desta árvore é o valor mínimo ou máximo, respectivamente. Uma vantagem do deste método é permitir a ordenação em arquivos com vários registro sem perder sua eficiência, visto que sua complexidade é igual tanto no pior, quanto no caso médio, O($nlog_2(n)$).

2. Materiais e Métodos

Os algoritmos para a indexação de arquivos com com índices primários e secundários explicados na Introdução foram desenvolvidos na linguagem *Pyhton*, versão 2.7, sem o auxílio de bibliotecas externas. O sistema operacional utilizado foi o Linux, utilizando o editor de texto *Visual Studio Code*. Como ferramenta de desenvolvimento, foi o utilizado o sistema de versionamento *git*, e a plataforma *GitHub* para facilitar a elaboração do código entre o grupo.

O primeiro algoritmo implementado foi o de ordenação *heapsort*, em que a estrutura da *heap* de mínimo foi desenvolvida como um vetor. O algoritmo utiliza um função auxiliar, que troca os valores de dois índices do vetor, e uma função que *'heapifica'*, ou seja, faz com que o vetor se comporte como uma *heap*, dado um índice de começo e fim. Esta ordenação é aplicada sempre que algum índice é adicionado, removido ou atualizado (caso sua chave primária mude) no arquivo de índices primários. A função $ordena\_indices()$ efetua o ordenamento nos índices e escreve no arquivo de índices.

Antes da implementação das funções de operação, foram desenvolvidas funções que inicialização.

A função $inicializa\_registros()$ recebe o arquivo de dados como parâmetro, percorre todas as linhas desse arquivo, ou seja, registro a registro, e separa cada um pelas suas informações de matrícula, nome, op, curso e turma em um dicionário. Cada dicionário é adicionado a uma lista de dicionários que contem as informações de todos os registros, justamente o variável de retorno da função.

A função $inicializa\_indices()$ recebe agora o arquivo de índices, ainda em branco, e a lista de registros. Para cada registro é criado um índice, com uma chave primária, formada pela concatenação dos campos de matrícula e nome, e com o valor da posição (número da linha) deste registro no arquivo de dados. O índice é escrito no arquivo de índices e também é armazena em uma lista de dicionários de índices.

Um pouco mais complexa, a função $inicializa\_indice\_secundario()$, recebe o arquivo do índice secundário, as listas de registros e índices, e o índice secundário a ser formado. Primeiramente, por meio da função auxiliar $opcoes\_secundario()$ é retornado uma lista com todos os valores possíveis daquele índice secundário. Desta forma, também é inicializado o *HEAD* da lista invertida, do mesmo tamanho da lista de opções, mas com os valores iniciais de -1. Agora, a cada índice, é buscado seu registro correspondente, para se ter acesso ao valor do índice secundário desejado utilizando a função $busca\_registro()$. Assim, seu índice é adicionado ao arquivo de índices secundários, e sua posição sempre se encontra no *HEAD*. O primeiro valor é sempre -1 e os seguintes são as posições do índices primários correspondentes.

Foi também desenvolvida a função $printa\_arquivo()$ para se mostrar os valores dos registros dos arquivos de dados, índices primários e índices secundários.

Dentre as operações, o primeiro algoritmo desenvolvido foi a função $adicionar\_registro()$ recebe os arquivos de dados, de índices primários e secundários, as listas de registros e índices e por fim o novo registro. O novo registro

é um dicionário com as mesmas características do outros registros, e é adicionado a lista de registros e depois escrito no arquivo de dados, sua posição é sempre salva numa variável global, também utilizada na função de inicialização dos registros. Aqui também é criado o índice primário correspondente a esse registro, que também é adicionado a lista de índices e no arquivo de índices após ser ordenado. Os arquivos de índices secundários são reinicializados, levando em conta os novos índices primários.

A função $remover\_registro()$ tem quase os mesmos parâmetros que a última, porém, ao invés de receber um novo registro, recebe a matrícula do registro que será removido. Dentro da função, primeiramente é encontrada a posição do registro a ser removido. Caso não seja encontrado, uma mensagem de erro é mostrada na tela. Caso contrário, o registro é apagado. Em seguida, o arquivo de contento os registros e o contendo os índices é escrito sem o elemento escolhido. Por último, é impressa na tela a nova lista de registros.

Para atualizar um registro, em primeiro lugar é mostrado todos os registros na tela. Em seguida, é perguntado a matrícula do elemento que se deseja atualizar. Ocorre então, uma checagem para garantir que a matrícula esta no registro. Caso não esteja, uma mensagem de erro aparece e a execução é terminada. Se a matrícula existe, o usuário entra com qual campo do registro quer atualizar e com o novo valor. É criado um novo registro com apenas o campo alterado. Em seguida, ocorre uma composição das funções $adicionar\_registro()$ e $remover\_registro()$. Dessa forma, a atualização funciona como uma remoção seguida de uma adição. No final, os registros são mostrados.

Para a interação com o usuário, foi desenvolvida uma interface no terminal. Inicialmente é feita a pergunta se o usuário deseja que as alterações sejam feitas no arquivo de dados *'lista1.txt'* ou *'lista2.txt'*. Tomada a decisão, é apresentado o menu de opções.

A primeira opção permite o usuário visualizar os arquivos, e se selecionada, da a opção de visualizar o arquivo de dados, arquivo de índices primários, arquivo de índices secundários 'op' ou 'turma'. Esta opção simplesmente mostra todos os registros linha a linha, com seus campos separados por tabulações na saída de vídeo. Se o usuário digita alguma opção inválida, é impresso na tela um aviso e o programa interrompe.

A segunda opção permite incluir registros. Primeiramente, o arquivo de índices atual é mostrado na tela. Após isso são coletadas as informações do novo registro para ser adicionado aos arquivos de dados e índices primários e secundários. Então o novo registro é de fato adicionado e o arquivo de índices primários é mostrado novamente na tela, agora reordenado devido a inclusão.

A próxima opção permite a exclusão de registro, com base na matrícula. Inicialmente, o arquivo de índices é impresso na tela. Após isso é pedida a matrícula do registro a ser removido e são executadas as funções de remoção, tanto do registro no arquivo de dados, quanto seu índice no arquivo de índices primários e suas referencias no arquivo de índices secundários. Finalmente, o novo arquivo de índices é mostrado na tela.

Por último, existe a opção de atualizar as informações de algum registro. Como nas outras opções, o arquivo de índices é inicialmente impresso na tela e então é perguntado ao usuário o qual campo ele deseja atualizar. Se ele escolhe o campo da matrícula ou nome, a chave primária desse registro muda, tal que é necessário mudar essa informação no arquivo de índices primários além de atualizar o campo no arquivo de dados. Se há mudança de algum índice secundário, deve-se atualizar essa informação também no arquivo de índices secundários, mudando o comportamento da lista invertida. Caso a mudança não afete nem o índice primário, nem o secundário, acontece uma única atualização no arquivo de dados.

3. Resultados e Discussão

Os resultados da atividade de laboratório são bem intuitivos, visto que as consequências das funcionalidades implementadas já são seus resultados. A função que imprime o conteúdo dos arquivos, de fato os imprime na saída de vídeo.

Quando um registro é incluído, o algoritmo de fato o inclui no arquivo de dados, seja em alguma posição vazia no meio do arquivo, seja no final do arquivo. A observação dos arquivos de índice nos leva a concluir também que na inclusão do registro, são adicionados seus índices primário e secundário correspondentes, causando o efeito esperado.

Quando um registro é excluído, acontece então a exclusão dele no arquivo de dados, dado a sua matrícula. Suas referências nos arquivos de índice primário e secundário também são excluídas, de modo que atendem as exigências da especificação.

Por último, o programa consegue buscar o registro por matrícula e atualiza o campo escolhido. Essa atualização consegue cobrir os casos de mudança na chave primária e mudança de chave secundária, estas refletidas em mudanças no arquivo de índices primário e secundário respectivamente.

4. Conclusões

A implementação dos requisitos da especificação do projeto assistiram bastante na compreensão dos conceitos sobre indexação aprendidos em sala de aula. Os problemas que surgem durante o desenvolvimento do código são fundamentais para se entender o motivo do uso as técnicas apresentadas. A implementação dessas técnicas, como o

heapsort ajuda a entender a fundo o funcionamento da ordenação. Outro ponto positivo foi conhecer melhor as vantagens e, principalmente, as desvantagens destas técnicas, já que nem sempre são entendidas de fato no campo teórico.

A escolha da linguagem de desenvolvimento *Python* ajudou muito no desenvolvimento, essencialmente na possibilidade de se criar listas e bibliotecas facilmente, além da facilitada manipulação de strings. Apesar dessas facilidades foi fácil entender porque a linguagem de programação C é mais eficiente, visto que permite ao desenvolvedor controlar diretamente o uso de memória e do *offset* dos arquivos.

References

[1] M. Folk and B. Zoellick. File structures. 1992.

[2] T. R. Harbron. *File systems: structures and algorithms*. Prentice-Hall, Inc., 1987.

[3] R. Kaushik, R. Krishnamurthy, J. F. Naughton, and R. Ramakrishnan. On the integration of structure indexes and inverted lists. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 779–790. ACM, 2004.

[4] R. Schaffer and R. Sedgewick. The analysis of heapsort. *Journal of Algorithms*, 15(1):76–100, 1993.

[1] [2] [4] [3]