Universidade Federal da Bahia

MATA54 - Estruturas de Dados e Algoritmos II

Trabalho 1

Manipulação de arquivos em C++ com Hash Duplo

Aluno: Felipe Carvalho Passos
Matricula: 201515282
Professor: Flávio Morais de Assis Silva

Data : 22/03/2019

1 Organização do Arquivo

O programa opera sobre um arquivo binário chamado *"random.bin"*, e mantém o tamanho desejado do arquivo em uma constante chamada *FSIZE*.

Existe um arquivo auxiliar chamado *auxi.bin* que guarda apenas um valor, que informa o programa na inicialização se o arquivo principal ja foi inicializado. Caso o arquivo não tenha sido inicializado, a função $initFile()$ será executada. **ver observação na explicação da função $printFile()$ para alterar o tamanho do arquivo*

O registro possui três campos: chave (única para cada registro), nome, e idade.

O programa é capaz de consultar, inserir e remover registros nesse arquivo *"random.bin"*, que é interpretado como uma tabela hash. Os registros são inseridos, consultados e removidos do arquivo, sempre através de sua *chave*.

A posição que um registro deverá ocupar no arquivo é dada pela função de hash, *hash1(int chave)*. Caso essa posição já esteja ocupada, a posição que o registro deverá ocupar é calculada somando o valor de *hash2(int chave)* ao ponteiro do arquivo, até achar uma posição vazia (no caso da inserção) ou achar o registro desejado (caso consulta e remoção).

O programa possui uma váriavel global do tipo $FILE$, que é atribuída pelo arquivo nas funções $abreArquivo()$ e $abreArquivoAux()$. Também utiliza duas variáveis globais auxiliares, uma chamada *Registro*, que é utilizada para escrever e ler nas funções $escreveRegistro()$ e $leRegistro()$, e a outra chamada *reg*, utilizada para armazenar os dados na inserção. Por último, uma variável global do tipo *char* chamada *entrada*, que é utilizada para determinar qual operação será feita no arquivo, de acordo com as especificações do trabalho.

2 Funções Importantes

1. ***void printFile()***
   A função *printFile*() imprime no console cada registro do arquivo, linha por linha, da posição 0 até $FSIZE$. Cada linha da saída vai ser impressa da seguinte maneira
   Caso posição tenha registro:

   Posicao: "posição atual" - Chave: "chave";Nome: "nome";Idade: "idade"

   Caso posição esteja vazia:

   Posicao: "posiçao atual" - Registro vazio

   Sendo os valores que estão dentro das aspas, os campos de cada registro

   *** Observação:** O programa inicialmente está setado para o tamanho de arquivo $FSIZE = 11$. Caso desejar aumentar ou diminuir o tamanho do arquivo, basta alterar FSIZE manualmente, mas isso pode gerar inconsistências no arquivo. Para garantir o perfeito funcionamento do arquivo ao alterar seu tamanho, basta apagar os dois arquivos "*random.bin*" e "*auxi.bin*" para que eles sejam reinicializados.

2. *void* ***InitFile()***
   Função de inicialização, que escreve em todas as posições do arquivo (0 até $FSIZE$) o registro vazio.

3. *int* ***hash1(int chaveAux)***
   Simples função de hash, que retorna o resto da divisão da chave pelo tamanho do arquivo ($FSIZE$).

4. *int* ***hash2(int chaveAux)***
   Função secundária de hash que retorna o resto da divisão do piso da chave dividido pelo tamanho do arquivo. Utilizada para resolução de colisões.

5. *int* ***escreveRegistro(int jump)***
   A função escreve o registro armazenado em *Registro* na posição calculada através do *jump*, a partir do início do arquivo. Retorna *true* se consegue escrever na posição, e $false$ caso contrario.

6. *int* ***leRegistro(int jump)***
   A função lê o registro armazenado na posição do arquivo, calculada através do *jump*, a partir do início do arquivo. Retorna *true* se consegue ler posição, e $false$ caso contrario.

7. *void* ***consultaChave(int chaveAux)***
   Se a chave buscada estiver na posição do $hash1(chaveAux)$, a função retorna os valores que estão armazenados no registro. Caso contrário, se houver colisão, a função passa a responsabilidade para $resolveColisaoConsulta(chaveAux)$.

8. *void* ***resolveColisaoConsulta(int chaveAux)***
   Soma-se ao valor de $hash1(chaveAux)$, o valor de $hash2(chaveAux)$ quantas vezes for necessário para achar o registro com a chave desejada (Aplica-se o resto da divisão desse valor pelo tamanho do arquivo, para o arquivo se comportar como um ciclo na consulta). Caso essa quantidade de vezes ultrapasse o tamanho do arquivo, significa que o ponteiro do arquivo já "andou"por todas as posições e não existe um registro com essa chave no arquivo. Se o ponteiro achar uma posição vazia, também significa que não existe essa chave no arquivo. Quando o ponteiro acha o registro desejado, o programa imprime os dados daquele registro.

9. *void* ***insereChave()***
   Utiliza como parametro "implícito"as variáveis globais reg, que é o registro a ser inserido, e *Registro*. Se na posição $hash1(reg.chave)$ o registro está vazio, pode inserir reg nela. Caso contrário, se houver colisão, a função passa a responsabilidade para $resolveColisaoInsercao()$.

10. *void* ***resolveColisaoInsercao()***
    Soma-se ao valor de $hash1(reg.chave)$, o valor de $hash2(reg.chave)$ quantas vezes for necessário para achar o registro com a chave desejada (Aplica-se o resto da divisão desse valor pelo tamanho do arquivo, para o arquivo se comportar como um ciclo na inserção). Caso essa quantidade de vezes ultrapasse o tamanho do arquivo, significa que o ponteiro do arquivo já "andou"por todas as posições e o arquivo está cheio. Se o ponteiro achar uma posição vazia, pode inserir reg nela. A função também trata o caso de haver algum registro com a mesma chave a ser inserida nas posições sondadas.

11. ***void removeChave(int chaveAux)***
    Funciona de maneira muito parecida a $consultaChave()$. Se o registro com a chave buscada estiver na posição do $hash1(chaveAux)$, a função substitui essa chave pelo valor -2 e apaga o valor dos outros atributos do registro, marcando-o como apagado.

12. ***void resolveColisaoRemocao(int chaveAux)***
    Funciona de maneira muito parecida a $resolveColisaoConsulta()$. Porém,

quando acha o registro com a chave desejada, ao invés de retorná-lo, substitui sua chave pelo valor -2 e apaga o valor dos outros atributos do registro, marcando-o como apagado.