Organização e Recuperação de Dados

2º Trabalho Prático

O 2º trabalho prático da disciplina **Organização e Recuperação de Dados** consiste na construção de um programa a ser desenvolvido em conformidade com as especificações abaixo. O programa deverá ser escrito na linguagem Python e poderá ser feito em equipes de até 3 alunos.

A equipe deverá enviar o código fonte pelo *Google Classroom* em um arquivo nomeado com os nomes dos integrantes da equipe. **O trabalho deverá ser apresentado pela equipe em data e horário agendado pela professora.**

Especificação

O objetivo deste trabalho é implementar um ***hashing* extensível** para armazenar e gerenciar chaves numéricas (inteiros de 4 bytes). Diferentemente do trabalho anterior, o foco será exclusivamente nas chaves, sem registros de dados associados, para simplificar a implementação da estrutura de indexação.

A estrutura será mantida em dois arquivos binários:

1. **diretorio.dat**: Arquivo que armazena o diretório do *hashing*.
2. **buckets.dat**: Arquivo que armazena os *buckets* de tamanho fixo com as chaves.

O programa deverá ser controlado por linha de comando, sem interface interativa com o usuário, para realizar operações de busca, inserção e remoção de chaves, além de funcionalidades de diagnóstico e manutenção da estrutura. O tamanho dos *buckets* (número de chaves que pode armazenar) deve ser uma constante facilmente configurável no código (p.e.: TAM_MAX_BUCKET), permitindo que o programa seja facilmente testado com *buckets* de diferentes tamanhos.

A estrutura interna dos *buckets* e do diretório devem ser similares às vistas em aula. Os *buckets* deverão ser criados e mantidos no arquivo de **buckets.dat**. O diretório será armazenado no arquivo de **diretorio.dat**, mas será construído e manipulado em memória.

Funcionalidades e operações

Como no 1º trabalho, as funcionalidades do programa serão acionadas por meio de *flags* na linha de comando.

1. Execução de operações (-e)

Esta é a principal funcionalidade do programa. Ela processa um arquivo de texto que contém uma sequência de operações de inserção, busca e remoção.

A execução do arquivo de operações será acionada pela linha de comando, no seguinte formato:

```
$ python programa.py -e arquivo_operacoes.txt
```

Para simplificar o processamento do arquivo de operações, considere que o arquivo de operações sempre será fornecido corretamente (i.e., o seu programa não precisa verificar a integridade desse arquivo).

Formato do arquivo de operações

O arquivo de operações terá um comando por linha, consistindo em um caractere identificador da operação seguido de um espaço e a chave (um número inteiro).

- i <chave>: **Insere** a chave no *hashing*. Não será permitida a inserção de chaves duplicadas.
- b <chave>: **Busca** pela chave, informando se foi encontrada e em qual bucket ela está.
- r <chave>: **Remove** a chave do *hashing*.

A seguir é exemplificado o formato de um arquivo de operações.

```
i 20
i 4
i 12
i 20
b 12
r 4
b 4
r 99
```

Com base no arquivo de operações mostrado acima, o programa deverá apresentar a seguinte saída:

```
> Inserção da chave 20: Sucesso.
> Inserção da chave 4: Sucesso.
> Inserção da chave 12: Sucesso.
> Inserção da chave 20: Falha – Chave duplicada.
> Busca pela chave 12: Chave encontrada no bucket 2.
> Remoção da chave 4: Sucesso.
> Busca pela chave 4: Chave não encontrada.
> Remoção da chave 99: Falha – Chave não encontrada.
```

Note que durante a execução das operações o **diretório** do *hashing* estará carregado na memória. Ao final da execução das operações, os arquivos **diretorio.dat** e **buckets.dat** devem estar atualizados e consistentes.

2. Impressão do diretório (-pd)

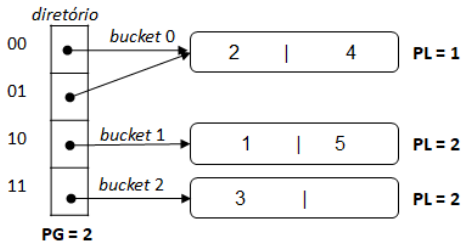
Essa funcionalidade exibe o estado atual do **diretório**. A impressão do diretório também será acessada via linha de comando, no seguinte formato:

```
$ python programa.py -pd
```

Note que, para essa execução, o *hashing* extensível precisa ser inicializado/carregado para a memória e os arquivos **buckets.dat** e **diretorio.dat** devem existir.

Sempre que ativada, essa funcionalidade apresentará na tela o conteúdo de todas as células do diretório, além das seguintes informações: (a) profundidade; (b) tamanho atual; e (c) número total de *buckets* referenciados.

Como exemplo, considere o *hashing* extensível com *buckets* de tamanho 2 mostrado abaixo.



Nesse caso, seu programa deverá apresentar as seguintes informações sobre o diretório:

```
----- Diretório -----
dir[0] = bucket(0)
dir[1] = bucket(0)
dir[2] = bucket(1)
dir[3] = bucket(2)

Profundidade = 2
Tamanho atual = 4
Total de buckets = 3
```

3. Impressão dos *buckets* (-pb)

Essa funcionalidade exibe o conteúdo dos ***buckets*** ativos no arquivo **buckets.dat**. Ela também será acessada via linha de comando, no seguinte formato:

```
$ python programa.py -pb
```

Note que, para essa execução, o *hashing* extensível precisa ser inicializado/carregado para a memória e os arquivos **buckets.dat** e **diretorio.dat** devem existir.

Como exemplo, considere o *hashing* extensível mostrado anteriormente, supondo que o *bucket* de RRN = 4 tenha sido removido. Nesse caso, seu programa deverá apresentar as seguintes informações sobre os *buckets*:

```
----- Buckets -----
Bucket 0 (Prof = 1):
Conta_chaves = 2
Chaves = [2, 4]

Bucket 1 (Prof = 2):
Conta_chaves = 2
Chaves = [1, 5]

Bucket 2 (Prof = 2):
Conta_chaves = 2
Chaves = [3, -1]

Bucket 4 -- Removido
```

Remoção de chaves e *buckets* vazios

Quando a remoção de uma chave resultar na concatenação de *buckets* amigos, um deles será removido logicamente e deverá ser **sinalizado como inativo**. Esse *bucket* inativo poderá ser mantido como fragmentação externa e, nesse caso, aparecerá como 'Removido' na impressão do arquivo de *buckets*. A implementação de uma PED para a reutilização de *buckets* inativos é um desafio opcional e não obrigatório para a avaliação do trabalho.

BOM TRABALHO!