

TRABALHO PRÁTICO 1

IBD 2022/1

Sergio Mergen

BufferManager

- A interface BufferManager permite definir como será realizado o gerenciamento de memória de cada tabela

```
public interface BufferManager {  
  
    public static final int BUFFER_SIZE = 6;  
  
    public void clear();  
    public Block[] getBufferedBlocks();  
    public Block getBlock(Long block_id, TableIO databaseIO) throws Exception;  
  
}
```

BufferManager

- A interface BufferManager permite definir como será realizado o gerenciamento de memória de cada tabela

Limpa todas as estruturas internas usadas para realizar o gerenciamento

```
public interface BufferManager {  
  
    public static final int BUFFER_SIZE = 6;  
  
    public void clear();  
    public Block[] getBufferedBlocks();  
    public Block getBlock(Long block_id, TableIO databaseIO) throws Exception;  
  
}
```

BufferManager

- A interface BufferManager permite definir como será realizado o gerenciamento de memória de cada tabela

Retorna um vetor contendo os blocos que estão na memória.

Esta função é chamada quando for necessário salvar no arquivo todas as modificações feitas na tabela

```
public interface BufferManager {  
  
    public static final int BUFFER_SIZE = 6;  
  
    public void clear();  
    public Block[] getBufferedBlocks();  
    public Block getBlock(Long block_id, TableIO databaseIO) throws Exception;  
  
}
```

BufferManager

- A interface BufferManager permite definir como será realizado o gerenciamento de memória de cada tabela

Retorna um bloco a partir do seu id.

É o principal método, pois é a partir dele que o gerenciamento de memória é realizado

```
public interface BufferManager {  
  
    public static final int BUFFER_SIZE = 6;  
  
    public void clear();  
    public Block[] getBufferedBlocks();  
    public Block getBlock(Long block_id, TableIO databaseIO) throws Exception;  
  
}
```

Estratégia LRU

- Para poder usar uma estratégia, devem ser criadas extensões que implementem os métodos desta interface.
- O protótipo conta com uma implementação padrão
 - LRUBufferManager

```
public class LRUBufferManager implements BufferManager{  
  
    protected Hashtable<Long, Block> blocksBuffer = new Hashtable();  
  
    LinkedList<Block> blocksList = new LinkedList();  
  
    //funções publicas  
    public void clear(){...}  
  
    public Block[] getBufferedBlocks() {...}  
  
    public Block getBlock(Long block_id, TableIO tableIO) throws Exception {...}
```

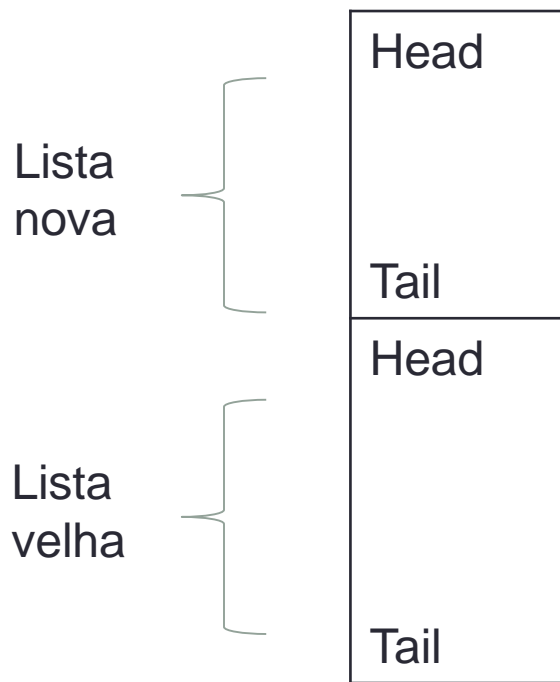
Estratégia LRU

- Basicamente, a função `getBlock` faz o seguinte:
- Verifica se um bloco solicitado está na memória
- Caso esteja
 - retorna o bloco
- Caso contrário
 - Verifica se o buffer está cheio
 - Caso esteja
 - remove o bloco que está a mais tempo sem ser usado
 - Adiciona o bloco solicitado ao buffer, na primeira posição
 - Retorna o bloco

MidPoint Insertion

- Existem estratégias mais adequadas para cenários em que alguns blocos sejam mais comumente solicitados do que outros
 - Por exemplo, o MySQL usa o MidPoint Insertion.
 - Uma variação do LRU que divide o buffer em duas listas:
 - Lista nova e lista velha

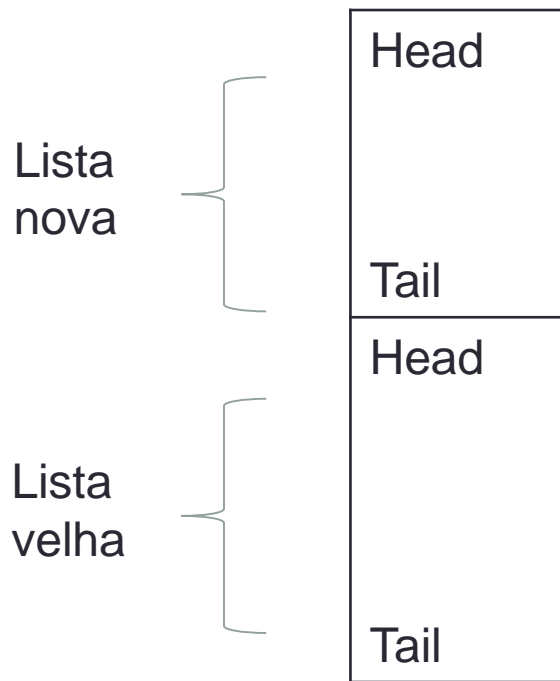
MidPoint Insertion



Lista nova: contém blocos recentemente acessados

Lista velha: contém blocos acessados há mais tempo

MidPoint Insertion

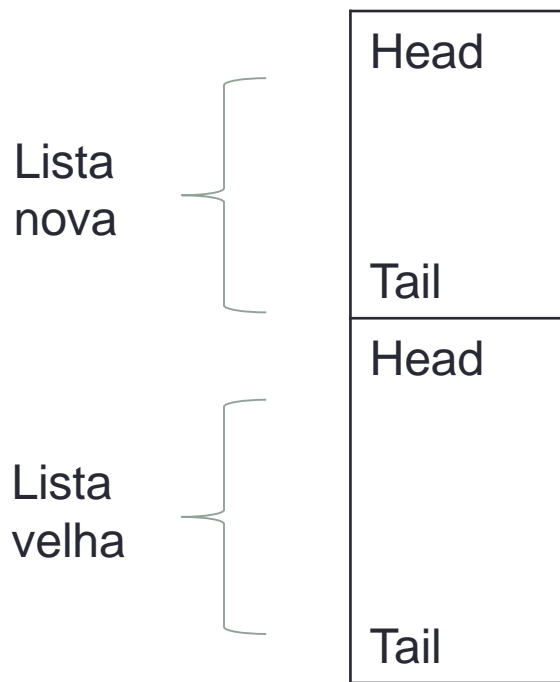


Caso o bloco já esteja no buffer

Ele é movido para o começo da **lista nova**

Isso faz com que os demais blocos 'envelheçam'

MidPoint Insertion



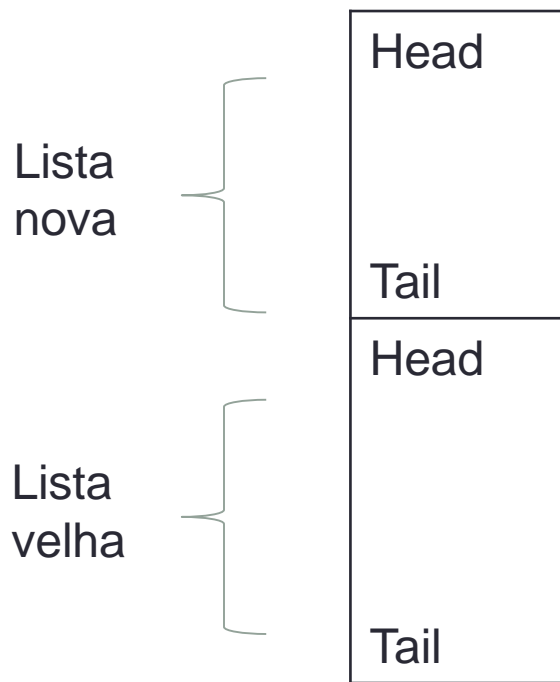
Caso um bloco não esteja no buffer

Ele é inserido no começo (**head**) da **lista velha**

Isso faz com que os demais blocos dessa lista 'envelheçam'

Quem sai é o bloco que está no final (**tail**) da lista

MidPoint Insertion



Caso um bloco seja usado apenas uma vez, ele nunca entra na lista nova

Assim, ele não prejudica os blocos comumente acessados presentes na lista nova

MidPoint Insertion

À esquerda, a lista contendo os blocos a serem acessados

Os blocos, por ordem de frequência são:

B1: 4 vezes

B2: 3 vezes

B5: 2 vezes

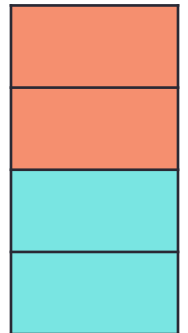
B4: 1 vez

B3: 1 vez

B6: 1 vez

B7: 1 vez

B1
B2
B1
B4
B2
B3
B2
B5
B1
B6
B5
B7
B1



MidPoint Insertion

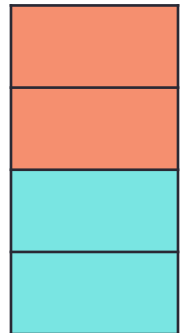
À direita, a lista encadeada de blocos na memória (buffer).

O buffer está dividido em duas sublistas:

- Lista velha: abaixo
- Lista nova: acima

Inicialmente, as listas estão vazias

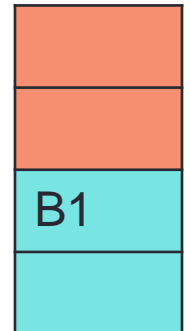
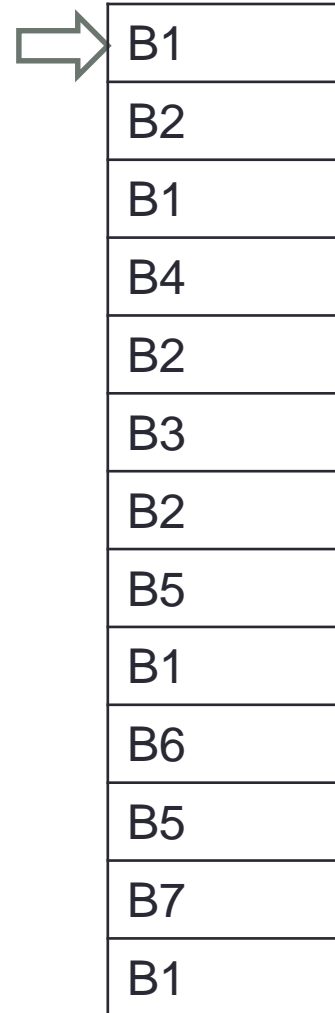
B1
B2
B1
B4
B2
B3
B2
B5
B1
B6
B5
B7
B1



MidPoint Insertion

B1 não está na lista.

É inserido no começo da lista velha

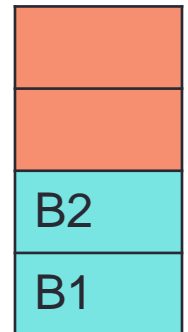
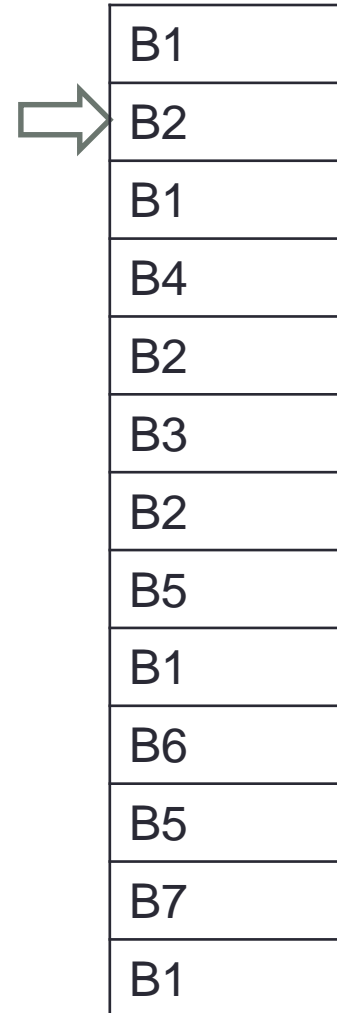


MidPoint Insertion

B2 não está na lista.

É inserido no começo da lista velha

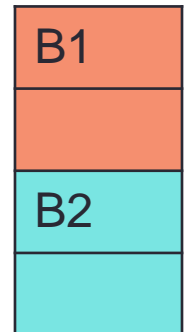
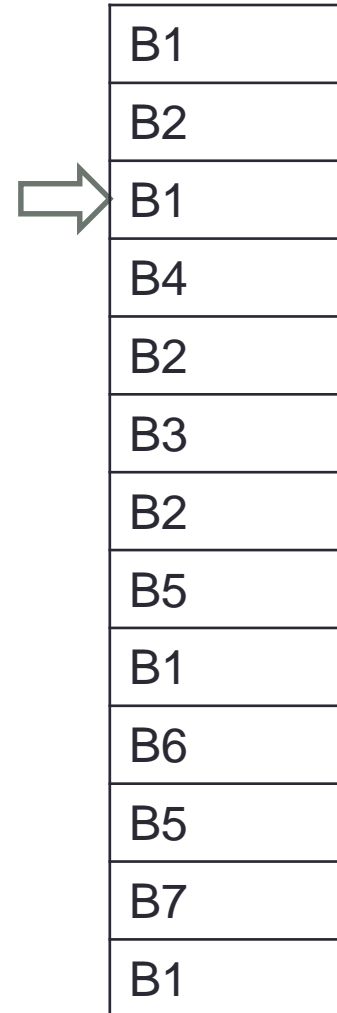
B1 envelhece



MidPoint Insertion

B1 já está na lista.

É promovido para o começo da lista nova

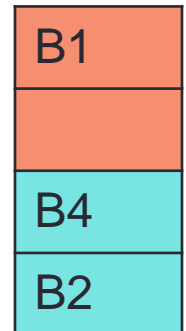
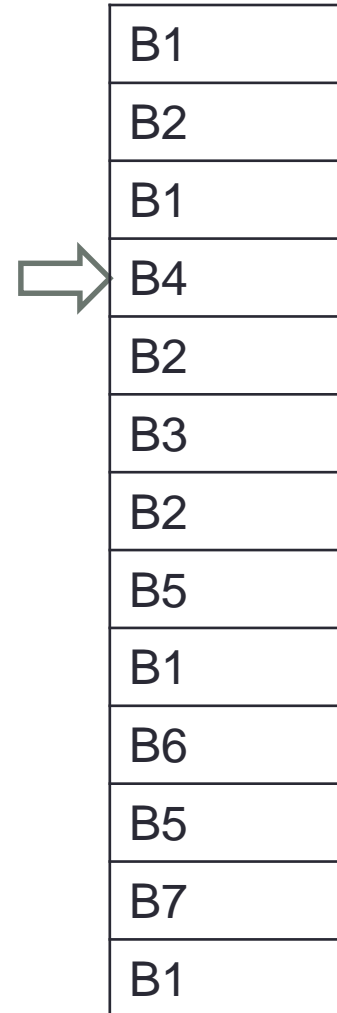


MidPoint Insertion

B4 não está na lista.

É inserido no começo da lista velha

B2 envelhece

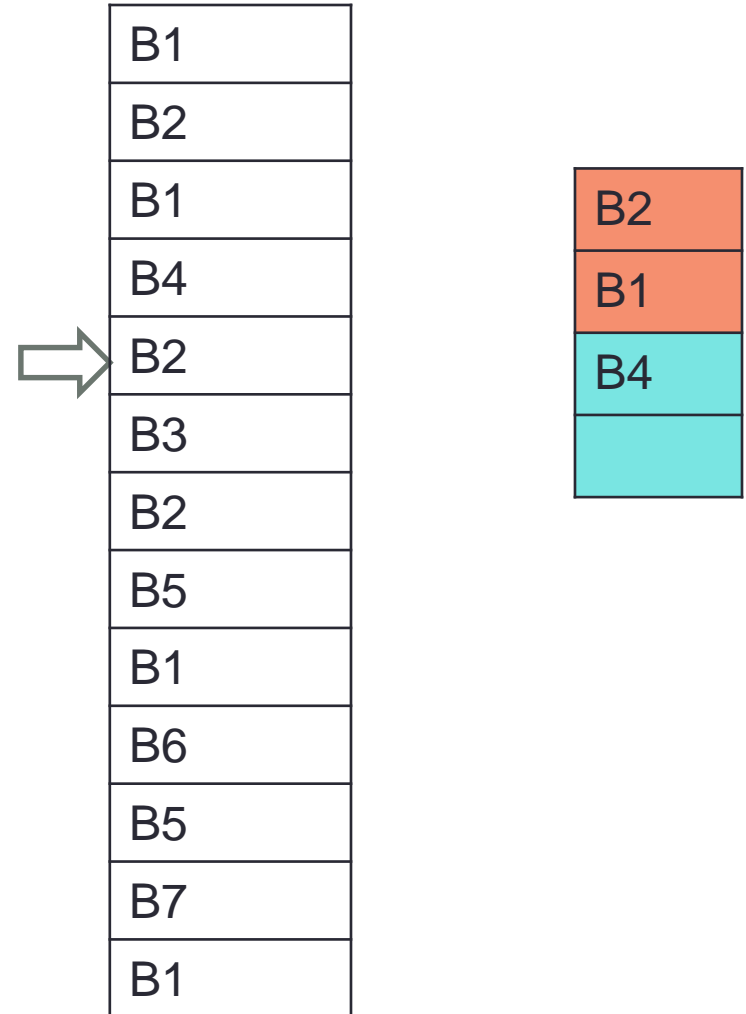


MidPoint Insertion

B2 já está na lista.

É promovido para o começo da lista nova

B1 envelhece

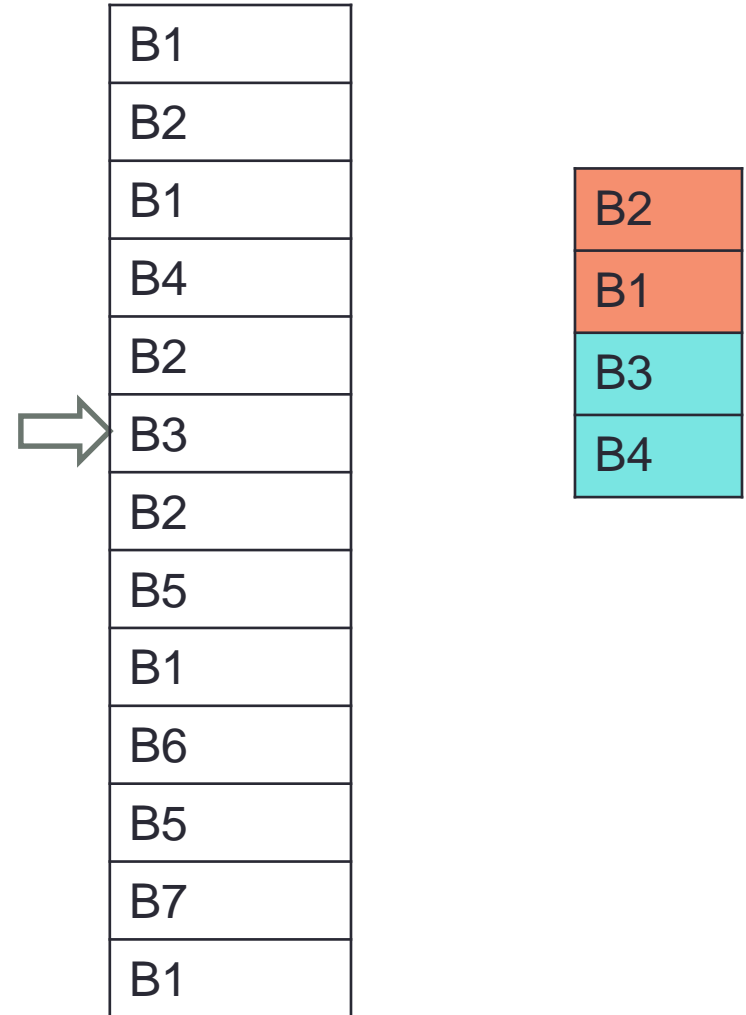


MidPoint Insertion

B3 não está na lista.

É inserido no começo da lista velha

B4 envelhece

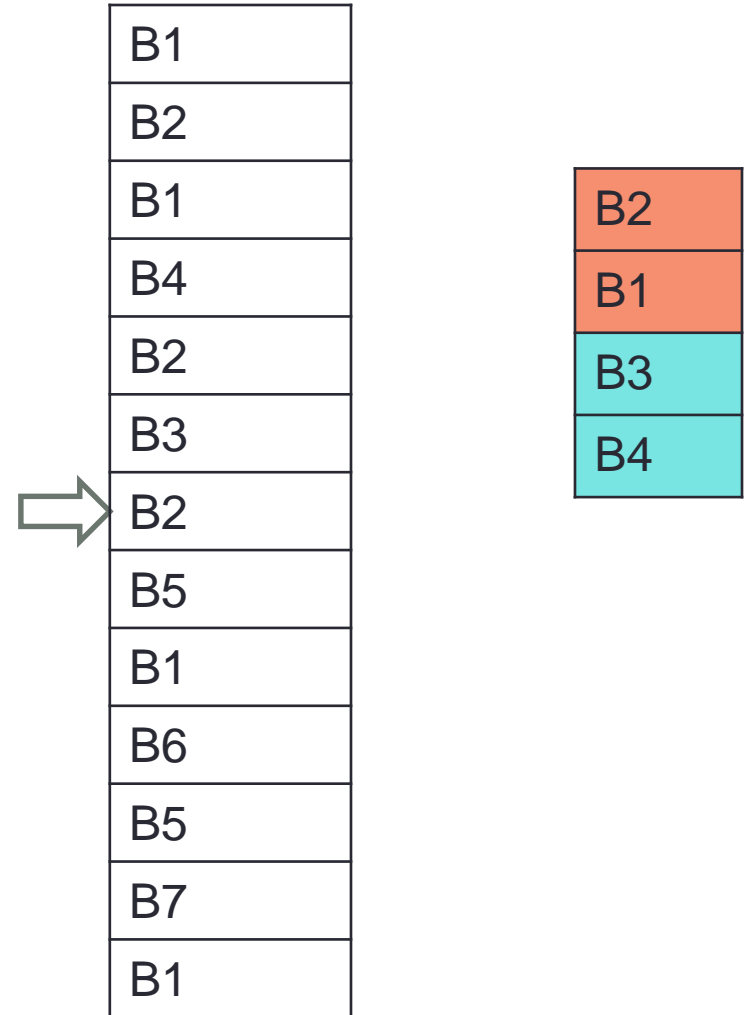


MidPoint Insertion

B2 já está na lista.

Ele já está no começo da lista nova

Nada é feito



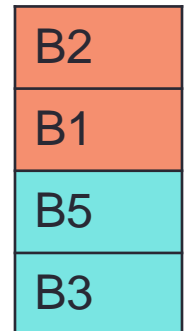
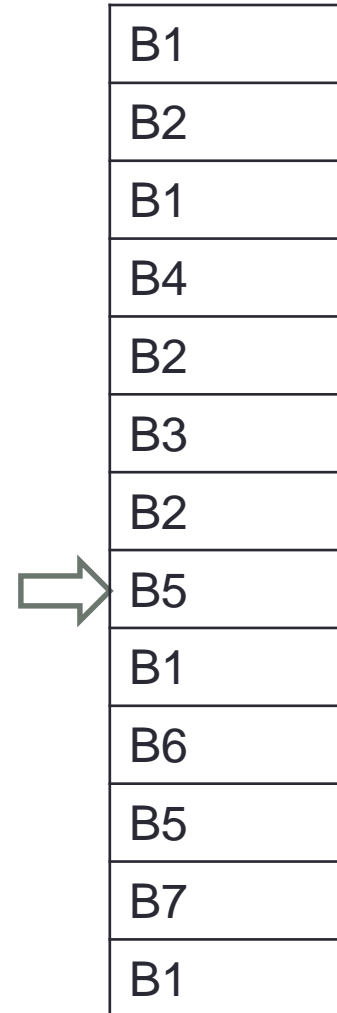
MidPoint Insertion

B5 não está na lista.

É inserido no começo da lista velha

B3 e B4 envelhece

B4 é despejado

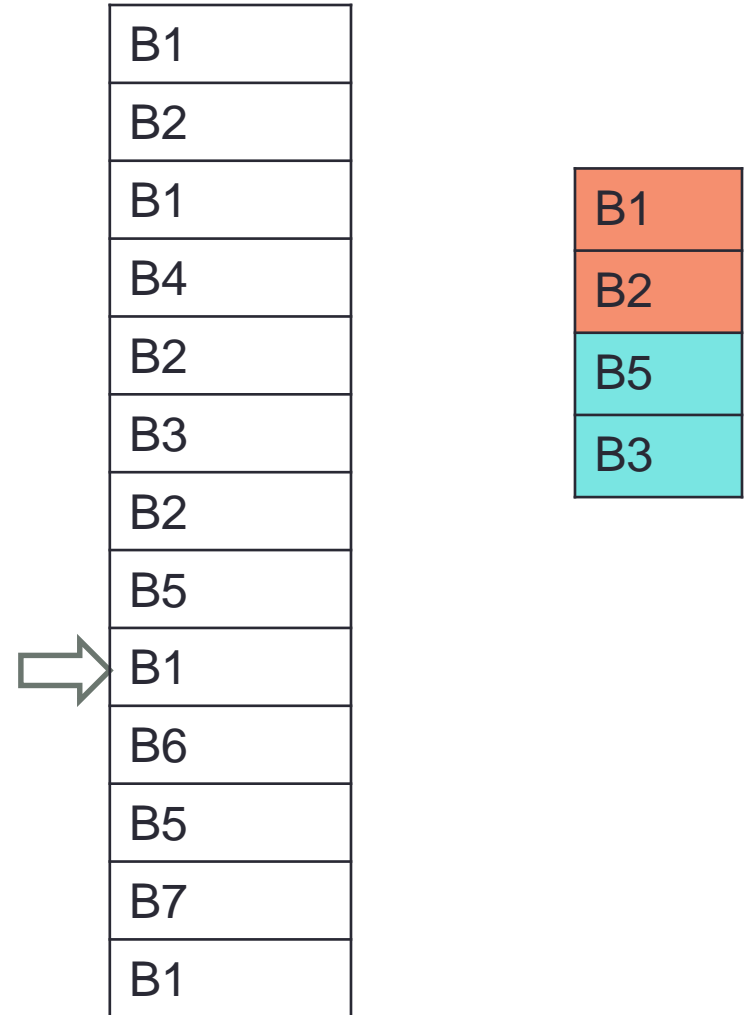


MidPoint Insertion

B1 já está na lista.

É promovido para o começo da lista nova

B2 envelhece



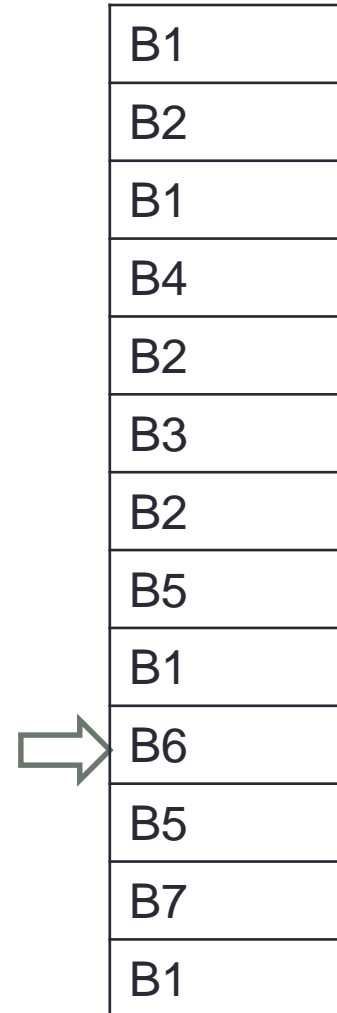
MidPoint Insertion

B6 não está na lista.

É inserido no começo da lista velha

B5 e B3 envelhecem

B3 é despejado



B1
B2
B1
B4
B2
B3
B2
B5
B1
B6
B5
B7
B1

B1
B2
B6
B5

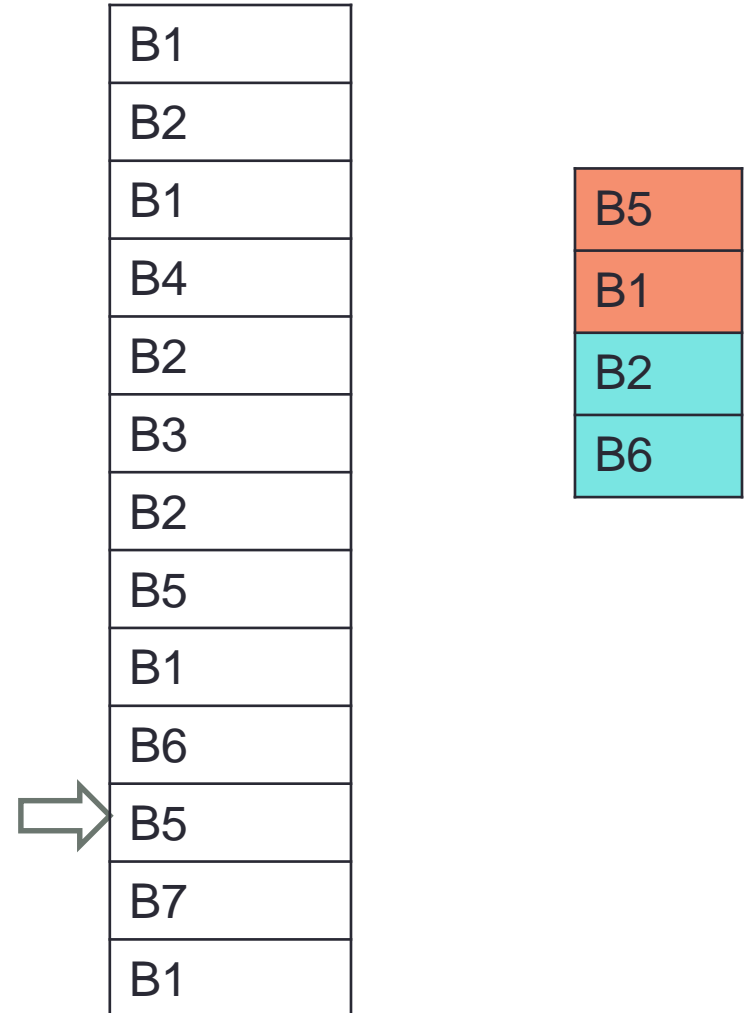
MidPoint Insertion

B5 já está na lista.

É promovido para o começo da lista nova

B1, B2 e B6 envelhecem

B2 sai da new lista



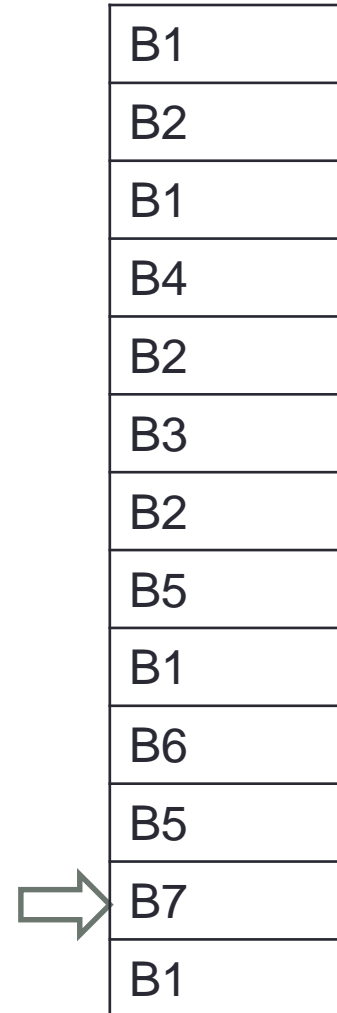
MidPoint Insertion

B7 não está na lista.

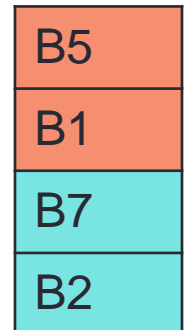
É inserido no começo da lista velha

B2 e B6 envelhecem

B6 é despejado



B1
B2
B1
B4
B2
B3
B2
B5
B1
B6
B5
B7
B1



B5
B1
B7
B2

MidPoint Insertion

B1 já está na lista.

É promovido para o começo da lista nova

B5 envelhece

B1
B2
B1
B4
B2
B3
B2
B5
B1
B6
B5
B7
B1

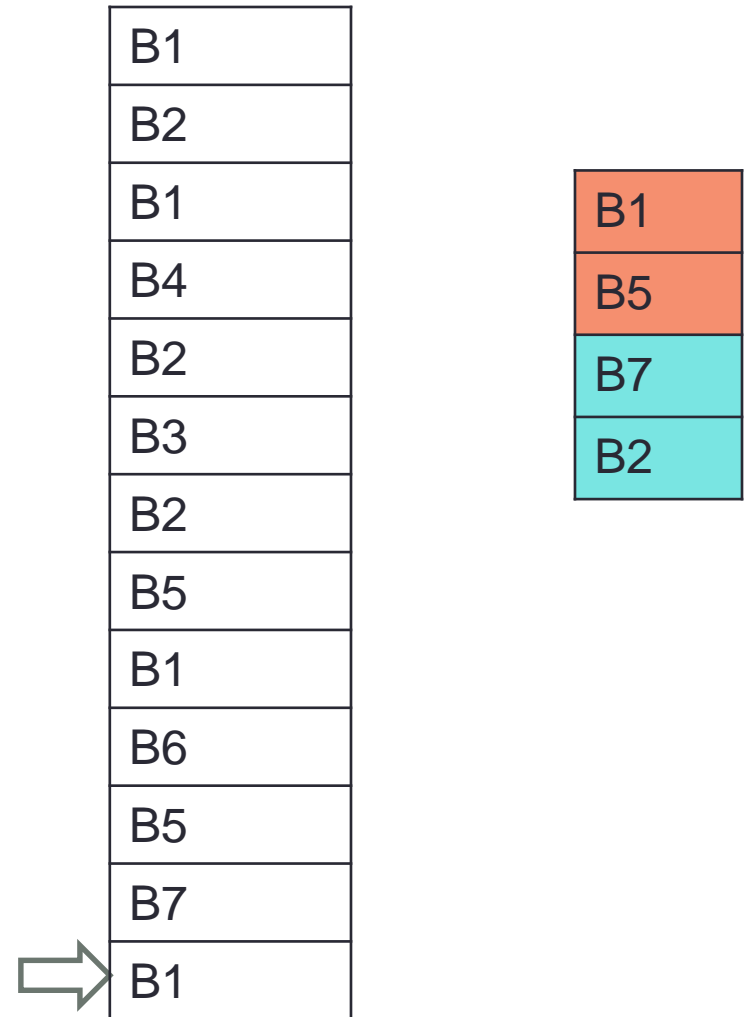


B1
B5
B7
B2

MidPoint Insertion

Perceba que os blocos mais frequentemente acessados não saíram do buffer (B1, B2 e B5)

B5 demorou a entrar, mas como começou a ser acessado com frequência, foi logo promovido



Trabalho Prático

- O objetivo do trabalho é implementar a estratégia MidPoint Insertion, estendendo a interface BufferManager
- O nome da classe deve ser **XXXMidPointBufferManager**,
 - Onde **xxx** é o nome do aluno
 - O pacote da classe deve ser `ibd.block`.

```
public class XXXMidPointBufferManager implements BufferManager{  
  
    //funções publicas  
    public void clear(){...}  
  
    public Block[] getBufferedBlocks() {...}  
  
    public Block getBlock(Long block_id, TableIO tableIO) throws Exception {...}
```

Trabalho Prático

- Todo o código desenvolvido deve ficar restrito à classe criada
- As três funções públicas devem ser devidamente implementadas
 - Crie funções privadas complementares, caso necessário
- Não modifique o comportamento classes pré-existentes
- É permitido usar qualquer estrutura de dados para solucionar o problema, inclusive mais do que uma
 - Deste que todo o código fique restrito à classe criada

Trabalho Prático

Dicas:

- Analise o código da classe LRUBufferManager
 - Ela pode disponibilizar trechos importantes para a nova implementação
- Estude as estruturas de dados disponibilizadas pelo próprio Java. Elas podem ser úteis para resolver o problema

Testes

- As funções para teste encontram-se na classe `ibd.block.Main` disponibilizada no moodle junto com o trabalho
- Insira a classe `Main` dentro do pacote correto:
 - `ibd.block`
- Funções usadas
 - **`execMultipleInsertions`**
 - **`execMultipleSearches`**
 - **`generateRecordIDs`**

Testes

- public void **execMultipleInsertions** (Table table, int amount, boolean ordered, boolean display).
 - Insere registros em uma tabela
- Parâmetros:
 - **table**: tabela alvo
 - **amount** quantidade de registros a serem inseridos
 - **ordered**: indica se os registros serão inseridos em ordem
 - **display**: exibe mensagem para cada registro inserido
- Ex. o trecho abaixo insere 1000 registros desordenados na tabela table.ibd e não exibe mensagem para cada registro inserido

```
Main m = new Main();  
Table table = Directory.getTable("c:\\teste\\ibd", "table.ibd");  
m.execMultipleInsertions(table, 1000, false, false);
```

Testes

- public void **execMultipleSearches** (Table table, Long[] recIds, boolean display).
 - Acessa registros de uma tabela
- Parâmetros:
 - **table**: tabela alvo
 - **recIds**: lista de ids de registros a serem acessados
 - **display**: exibe mensagem para cada registro recuperado
- Ex. o trecho abaixo procura na tabela table.ibd pelos registros com ids 6,2,8 e 14, e exibe mensagem conforme os registros vão sendo recuperados

```
Main m = new Main();  
Table table = Directory.getTable("c:\\teste\\ibd", "table.ibd");  
Long[] recIds = new Long[]{6L,2L,8L,14L};  
m.execMultipleSearches(table, recIds, true);
```

Testes

- `public Long[] generateRecordIDs(int blocksAmount1, int blocksAmount2,`
- `int recordsAmount1, int recordsAmount2)`
- Gera uma lista aleatória de ids, mesclando ids de dois conjuntos de blocos (1 e 2) de tamanhos variáveis
- Parâmetros:
 - **blocksAmount1**: quantidade de blocos do conjunto 1.
 - **blocksAmount2**: quantidade de blocos do conjunto 2.
 - **recordsAmount1**: quantidade de ids de registros de blocos do conjunto 1
 - **recordsAmount2**: quantidade de ids de registros de blocos do conjunto 2

Testes

- `public Long[] generateRecordIDs(int blocksAmount1, int blocksAmount2,`
- `int recordsAmount1, int recordsAmount2)`
- Ex. o trecho abaixo cria 1000 ids de 6 blocos (Conjunto 1) e outros 100 ids de outros 6 blocos (Conjunto 2).
 - Os blocos de conjunto 1 possuem 10x vezes mais acessos do que os blocos do conjunto 2

```
generateRecordsIDs(6, 6, 1000, 100);
```

- Ex. o trecho abaixo cria 1000 ids de 6 blocos (Conjunto 1) e outros 100 ids de outros 60 blocos (Conjunto 2).
 - Os blocos de conjunto 1 possuem 10x vezes mais acessos do que os blocos do conjunto 2
 - Há 10 vezes menos blocos no conjunto 1 do que no conjunto 2

```
generateRecordsIDs(6, 60, 1000, 100);
```

Testes

- Parâmetros de medição
 - BLOCKS_LOADED = quantidade de blocos carregados para a memória
 - TIME = o tempo total empregado na busca;

Testes

Gera a lista de Ids
Use os parâmetros que aparecem no slide

```
Main m = new Main();  
Long[] recIDs = m.generateRecordIDs(6,60, 1000, 100);  
  
Table table1 = Directory.getTable("c:\\teste\\ibd", "table.ibd");  
  
m.executeMultipleInsertions(table1, (int)(Block.RECORDS_AMOUNT *  
Table.BLOCKS_AMOUNT), true, false);  
  
table1.bufferManager = new LRUBufferManager();  
  
Params.BLOCKS_LOADED = 0;  
long time = m.executeMultipleSearches(table1, recIDs, false);  
System.out.println("BLOCKS_LOADED "+Params.BLOCKS_LOADED);  
System.out.println("time "+time);
```

Testes

Instancia uma tabela

```
Main m = new Main();  
Long[] recIDs = m.generateRecordIDs(6,60, 1000, 100);  
  
Table table1 = Directory.getTable("c:\\teste\\ibd", "table.ibd");  
  
m.executeMultipleInsertions(table1, (int)(Block.RECORDS_AMOUNT *  
Table.BLOCKS_AMOUNT), true, false);  
  
table1.bufferManager = new LRUBufferManager();  
  
Params.BLOCKS_LOADED = 0;  
long time = m.executeMultipleSearches(table1, recIDs, false);  
System.out.println("BLOCKS_LOADED "+Params.BLOCKS_LOADED);  
System.out.println("time "+time);
```

Testes

Certifique-se de que a tabela possua registros executando este trecho pelo menos uma vez

```
Main m = new Main();  
Long[] recIDs = m.generateRecordIDs(6,60, 1000, 100);  
  
Table table1 = Directory.getTable("c:\\teste\\ibd", "table.ibd");  
  
m.executeMultipleInsertions(table1, (int)(Block.RECORDS_AMOUNT *  
Table.BLOCKS_AMOUNT), true, false);  
  
table1.bufferManager = new LRUBufferManager();  
  
Params.BLOCKS_LOADED = 0;  
long time = m.executeMultipleSearches(table1, recIDs, false);  
System.out.println("BLOCKS_LOADED "+Params.BLOCKS_LOADED);  
System.out.println("time "+time);
```


Testes

Configuração do BufferManager

```
Main m = new Main();  
Long[] recIDs = m.generateRecordIDs(6,60, 1000, 100);  
  
Table table1 = Directory.getTable("c:\\teste\\ibd", "table.ibd");  
  
m.executeMultipleInsertions(table1, (int)(Block.RECORDS_AMOUNT *  
Table.BLOCKS_AMOUNT), true, false);  
  
table1.bufferManager = new LRUBufferManager();  
  
Params.BLOCKS_LOADED = 0;  
long time = m.executeMultipleSearches(table1, recIDs, false);  
System.out.println("BLOCKS_LOADED "+Params.BLOCKS_LOADED);  
System.out.println("time "+time);
```

Testes

Execução da busca e impressão dos resultados

```
Main m = new Main();  
Long[] recIDs = m.generateRecordIDs(6,60, 1000, 100);  
  
Table table1 = Directory.getTable("c:\\teste\\ibd", "table.ibd");  
  
m.executeMultipleInsertions(table1, (int)(Block.RECORDS_AMOUNT *  
Table.BLOCKS_AMOUNT), true, false);  
  
table1.bufferManager = new LRUBufferManager();  
  
Params.BLOCKS_LOADED = 0;  
long time = m.executeMultipleSearches(table1, recIDs, false);  
System.out.println("BLOCKS_LOADED "+Params.BLOCKS_LOADED);  
System.out.println("time "+time);
```

Testes

Executa a mesma coisa usando outro
BufferManager

```
Table table2 = Directory.getTable("c:\\teste\\ibd", "table.ibd");  
  
table2.bufferManager = new XXXMidPointBufferManager();  
  
Params.BLOCKS_LOADED = 0;  
long time = m.executeMultipleSearches(table2, recIDs, false);  
System.out.println("BLOCKS_LOADED "+Params.BLOCKS_LOADED);  
System.out.println("time "+time);
```

Testes

A tabela é a mesma(table.ibd), mas muda a instância (table2)

```
Table table2 = Directory.getTable("c:\\teste\\ibd", "table.ibd");  
  
table2.bufferManager = new XXXMidPointBufferManager();  
  
Params.BLOCKS_LOADED = 0;  
long time = m.executeMultipleSearches(table2, recIDs, false);  
System.out.println("BLOCKS_LOADED "+Params.BLOCKS_LOADED);  
System.out.println("time "+time);
```

Testes

- Use as medições de BLOCKS_LOADED e TIME para medir o desempenho da estratégia criada
- A nota depende do quão efetiva for a estratégia criada
 - O esperado é que ela vença a estratégia padrão
- EM um dos testes, o desempenho foi
 - LRU: 267 blocos carregados
 - MidPoint Insertion: 231 blocos carregados
- Para a análise de desempenho, não altere as configurações do generateRecordIDs
 - Elas conseguem explorar as vantagens da estratégia MidPoint.

Controle de Buffer

- O buffer foi configurado para ter tamanho 6
 - Não altere o tamanho
 - Ele deve ser baixo para que seja possível perceber ganho de performance ao trocar de uma estratégia para a outra
- Divida as listas nova e velha em tamanhos iguais
 - Metade do tamanho do buffer para cada um
- Não permita que o buffer cresça mais do que o seu limite
 - Isso pode levar a anulação da nota

Trabalho Prático

- Além dos Ids gerados pela generateRecordIDs, use outras listas menores para depurar o código, e descobrir se o gerenciador se comporta de maneira adequada.

```
Long[] reclds = new Long[]{6L,2L,8L,14L};
```

- Para garantir que o registro pertença a um bloco específico, multiplique o id pelo número de registros do bloco
- Ex.
 - 0* Block.RECORDS_AMOUNT : acessa o registro 0, que é o primeiro registro do bloco 0
 - 1* Block.RECORDS_AMOUNT : acessa o registro 31, que é o primeiro registro do bloco 1

Avaliação

- Os principais aspectos que serão analisados para a avaliação do trabalho são
 - O código deve estar funcional
 - Todas funções públicas foram implementadas de forma adequada
 - As funções complementares foram criadas como privadas
 - O arquivo correto foi enviado (.java)
 - O pacote da classe foi especificado da forma correta
 - Os blocos se movem dentro do buffer conforme a especificação
 - O algoritmo se sobressai ao LRU no teste de desempenho

Entrega

- Entrega pelo moodle
- Não entregue o projeto inteiro
 - Apenas a classe java solicitada
- O trabalho é **individual**
 - O compartilhamento de código entre alunos leva à anulação da nota

Entrega

- A nota máxima possível depende do dia em que for feita a entrega

Prazo	Nota máxima
13/05 23h59min (sexta)	100%
14/05 23h59min (sábado)	80%
15/05 23h59min (domingo)	60%
16/05 23h59min (segunda)	40%

- Entregas feitas após o dia 16/05 não serão avaliadas