# Banco de dados II

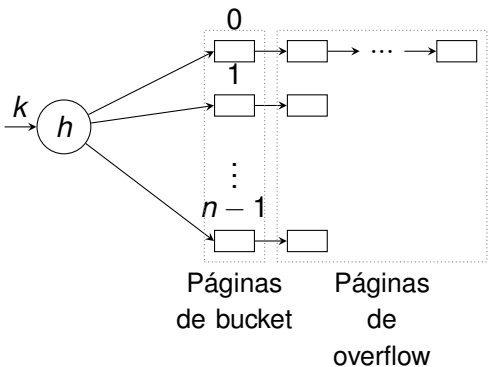
## *4 – Indexação baseada em hash*

Marcos Roberto Ribeiro

Departamento de Engenharia e Computação (DEC)
Curso de Engenharia de Computação
2024

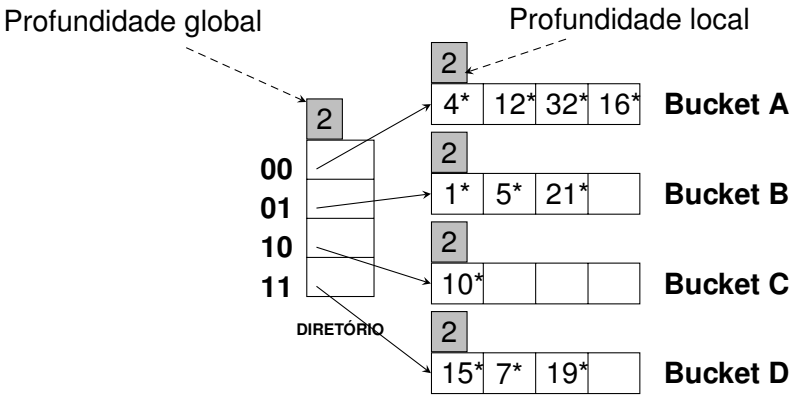
INSTITUTO FEDERAL
Minas Gerais
Campus Bambuí

# *Hash* estático




- $h(k) = k$ **mod** $n$ onde $n$ é o número de *buckets* (ou $\log_2 n$ últimos bits de $k$)
- Páginas de *overflow* são criadas quando não há espaço na página de *bucket*
- O índice é estático porque o número de páginas de *buckets* não muda (são alocados assim que o índice é criado)
- O problema é que podem aparecer longas cadeias de *overflow* causando lentidão nas pesquisas

# *Hash* extensível




- O *hash* extensível utiliza um diretório de ponteiros para os *buckets*
- Esse diretório é duplicado se ocorrer algum *overflow*

# Inserções em *hash* extensível

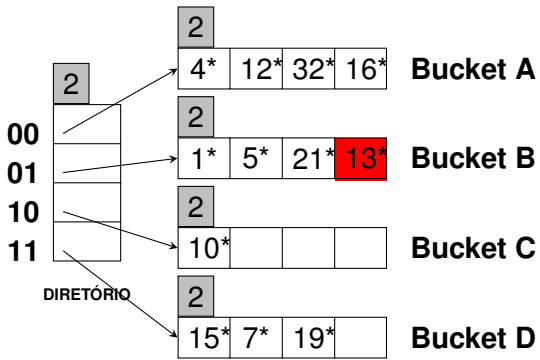
- Para inserir uma entrada, calculamos a função *hash* sobre a chave considerando os últimos *n* bits, onde *n* é a profundidade local




- Exemplo: Inserir a entrada **13***, temos $h(13) = 01$
- Nesse caso, a entrada cabe no *bucket*, então apenas gravamos a entrada na posição livre

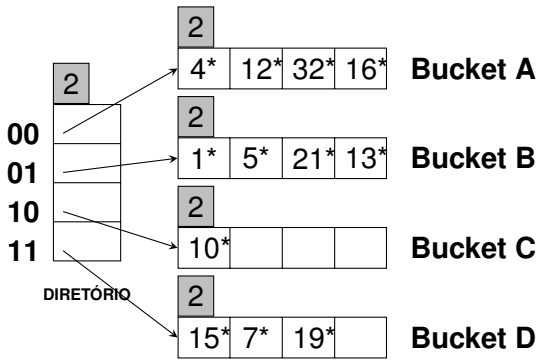
# Inserções em *hash* extensível

- Para inserir uma entrada, calculamos a função *hash* sobre a chave considerando os últimos *n* bits, onde *n* é a profundidade local



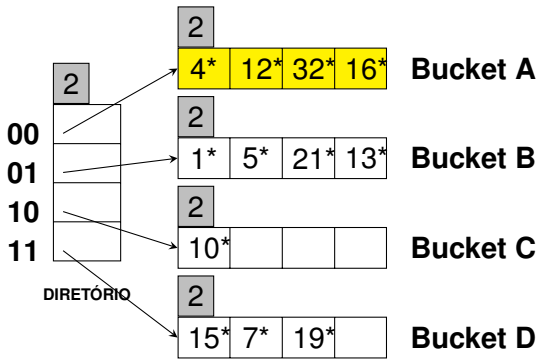

- Exemplo: Inserir a entrada **13***, temos $h(13) = 01$
- Nesse caso, a entrada cabe no *bucket*, então apenas gravamos a entrada na posição livre

# Inserção com *overflow*




- Agora, considere a inserção da entrada **20***
- $h(20) = 00$ (*bucket* A)
- Entretanto, o *bucket* A está cheio

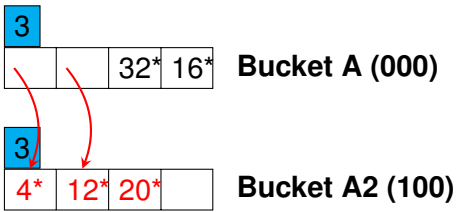
# Inserção com *overflow*



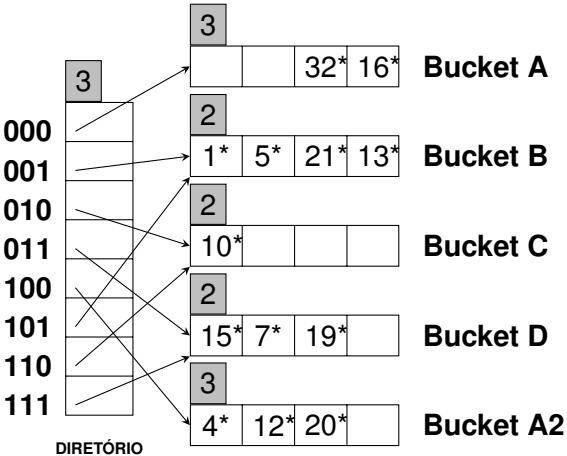

- Agora, considere a inserção da entrada **20***
- $h(20) = 00$ (*bucket* A)
- Entretanto, o *bucket* A está cheio

# Inserção com *overflow* - novo *bucket*

- Aumentamos a profundidade local do *bucket* A para 3
- Criamos o novo *bucket* A2 e redistribuímos os registros usando a nova profundidade (3 últimos bits)



# Inserção com overflow - duplicação do tamanho do diretório




- Devemos duplicar o tamanho do diretório sempre que há uma profundidade local maior do que a profundidade global
- Os *buckets* que não foram divididos permanecem com a profundidade local inalterada e recebem mais ponteiros

# Exclusões de *buckets*

- Durante as exclusões, poderíamos excluir os *buckets* vazios
- Porém, na prática, isto não é viável porque os arquivos tendem a crescer

# *Hash* linear

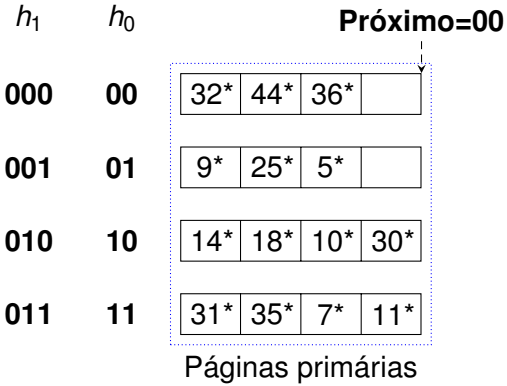
- O *hash* linear reduz as divisões e proporciona maior ocupação dos *buckets* do que o *hash* extensível
- Esse tipo de *hash* utiliza uma família de funções *hash* $h_0$, $h_1$, $h_2$, ...
- O intervalo de $h_{i+1}$ é o dobro do intervalo de $h_i$
- Por exemplo, supondo 32 *buckets* iniciais ($N_0 = 32 = 2^5$):

  $h_0(k) = k$ **mod** $32 \Rightarrow d_0 = 5$ (5 bits)

  $h_1(k) = k$ **mod** $64 \Rightarrow d_1 = d_0 + 1 = 5 + 1 = 6$ (6 bits, $N_1 = 64$)

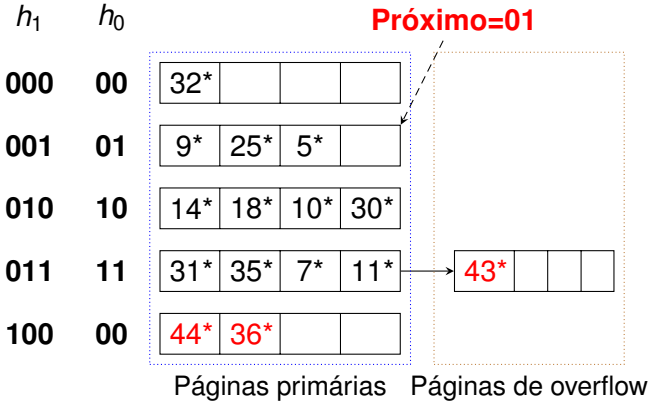
  $\vdots$

# Exemplo de *hash* linear



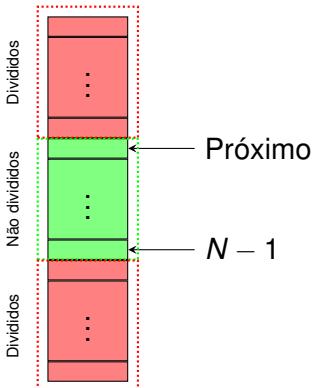

- Iniciamos com 4 buckets ($N_0 = 4$) e nível 0 (zero)
- $N = N_0 * 2^{\text{nível}} = 4 * 2^0 = 1$ (número de *buckets* atual)
- Podem ocorrer páginas de *overflow*
- Dividimos o *bucket* apontando por **próximo** sempre que ocorre *overflow*
- Redistribuímos as entradas usando $h_{\text{nível}+1}$ e incrementamos o apontador **próximo**

# Inserção com *overflow*



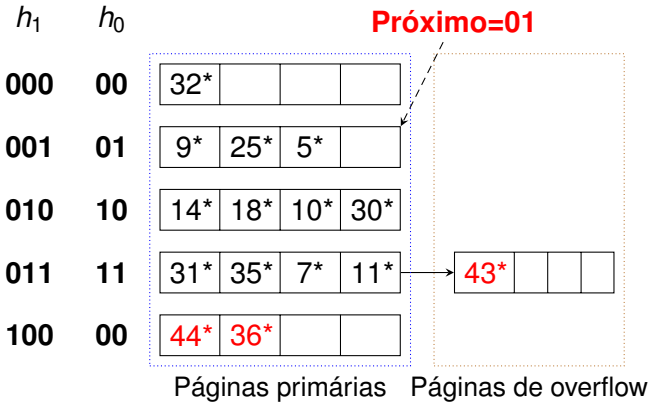

- Ao inserir **43***, temos $h_0(43) = 11$
- Como o *bucket* **11** está cheio, criamos um página de *overflow* para incluir a entrada **43***
- Dividimos o *bucket* **00** apontado por próximo e incrementamos próximo para **01**

# Rodadas de divisão




- O apontador próximo circula pelos **buckets** até que todos sejam divididos
- Quando isto acontece, passamos a usar as funções **hash** do próximo nível

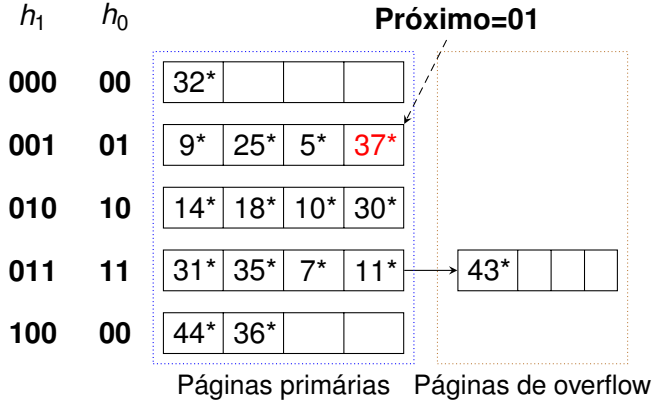
# Pesquisa



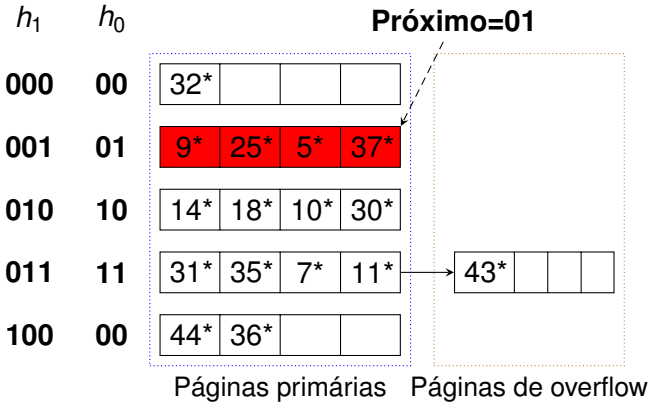

- Primeiro usamos a função $h_{\text{nível}}$
- Se **próximo** $\leq h_{\text{nível}} < N_{\text{nível}}$, já encontramos o *bucket*. Exemplo
  $h_0(18) = 10(2)$,
  **próximo** $\leq 10 < 100$, (bucket 10)
- Caso contrário, aplicamos $h_{\text{nível}+1}$. Exemplo:
  $h_0(32) = 00$, $00 <$ **próximo**,
  $h_1(32) = 000$
  $h_0(44) = 00$, $00 <$ **próximo**,
  $h_1(44) = 100$

# Inserção sem *overflow*

- Nem sempre ocorre divisão, por exemplo, na inserção da entrada **37***

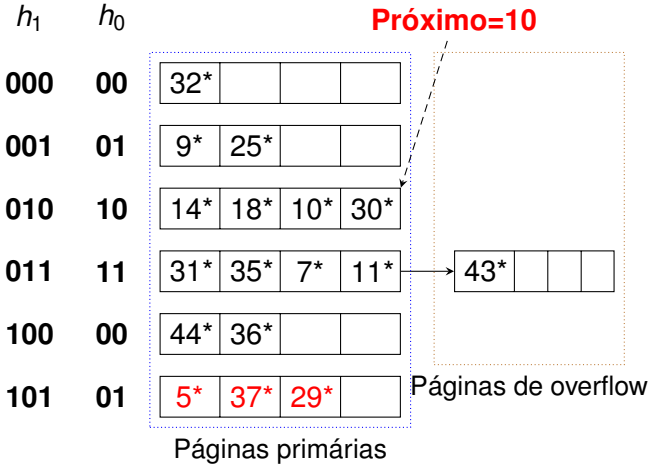


# Inserção com divisão e sem *overflow*




- Se o *bucket* que causar *overflow* for aquele apontado pelo **próximo**, dividimos o *bucket* e evitamos este *overflow*
- Exemplo: inserir a entrada 29*, $h_0(29) = 01$ (está cheio e é o próximo)

# Inserção com divisão e sem *overflow*

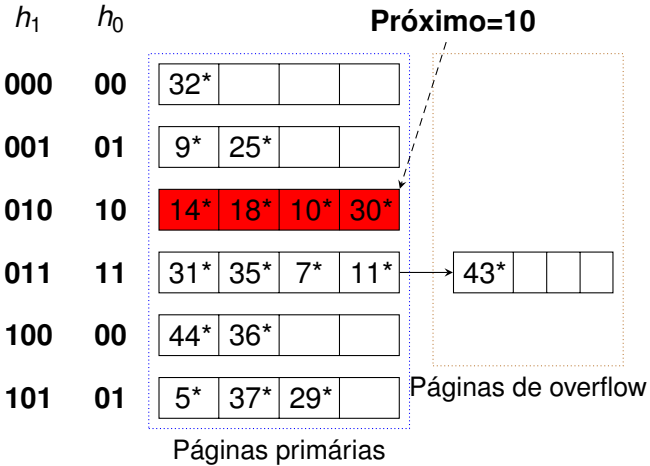



- Se o *bucket* que causar *overflow* for aquele apontado pelo **próximo**, dividimos o *bucket* e evitamos este *overflow*
- Exemplo: inserir a entrada 29*, $h_0(29) = 01$ (está cheio e é o próximo)

# Exercícios

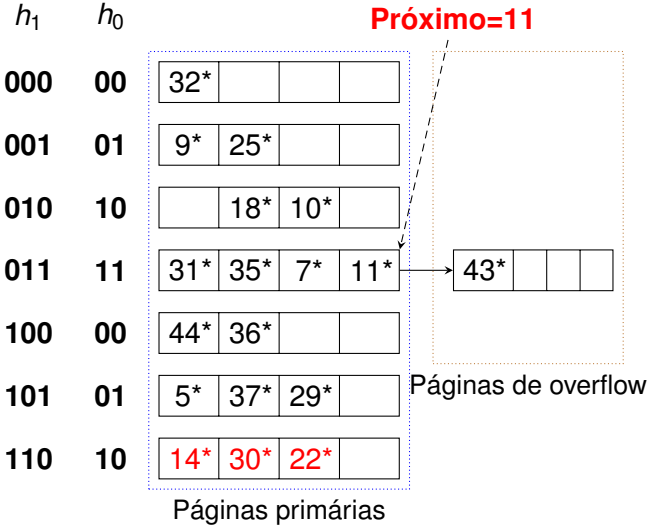
- Inserir as entradas 22*, 66* e 34*

# Inserção da entrada 22*



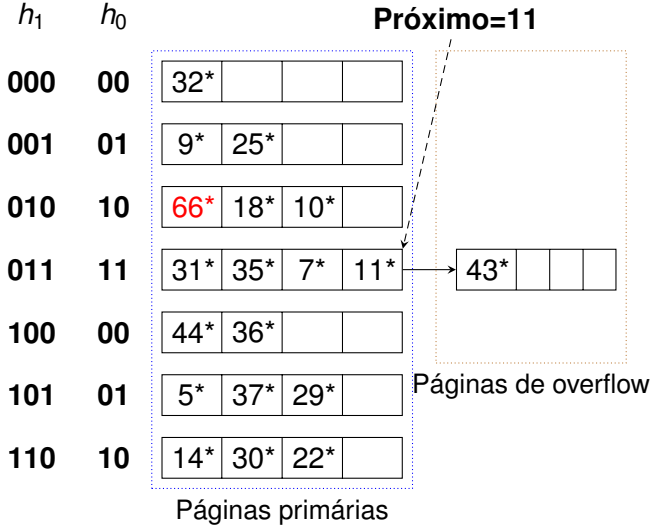

- $h_0(22) = 10$ (está cheio e é o próximo)

# Inserção da entrada 22*



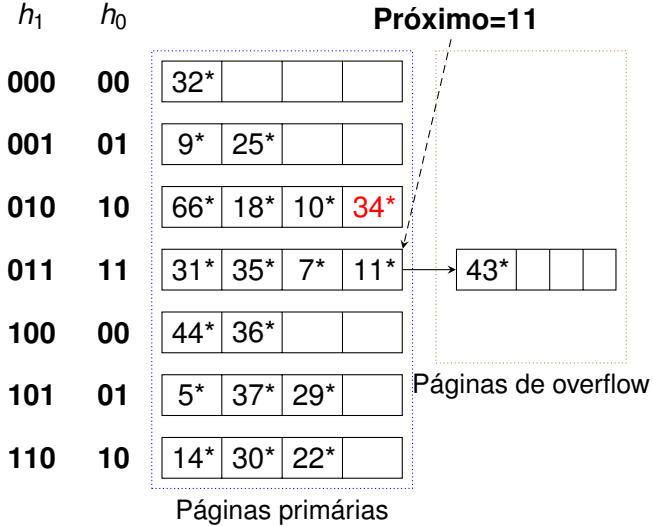

- $h_0(22) = 10$ (está cheio e é o próximo)

# Inserção da entrada 66*



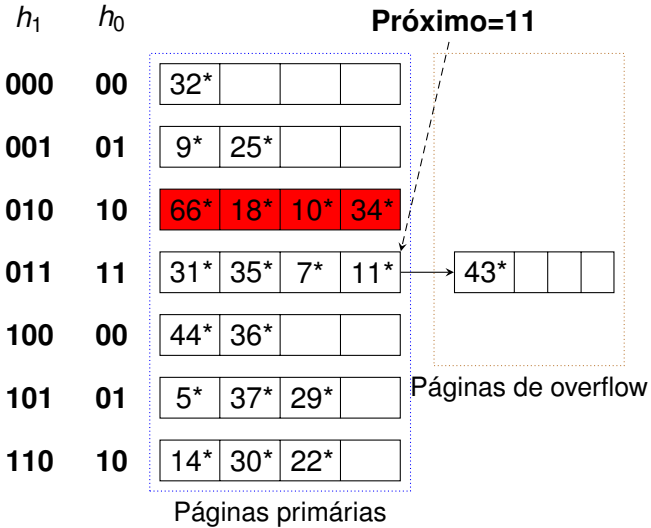

- $h_0(66) = 10$, como $01 <$ **próximo**, aplicamos $h_1$
- $h_1(66) = 010$

# Inserção da entrada 34*



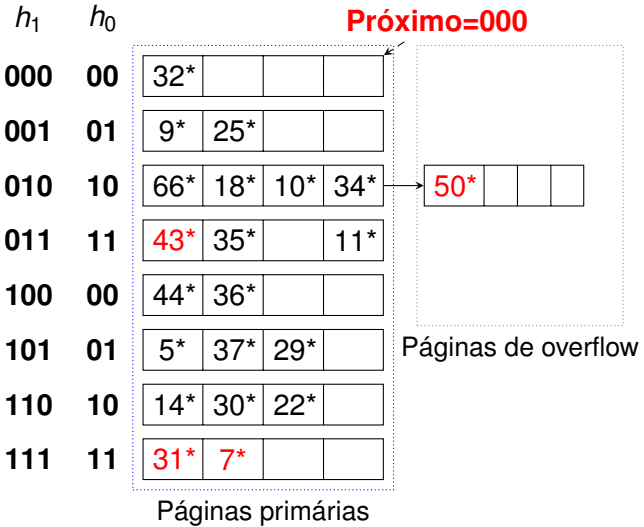

- $h_0(34) = 10$, como $01 <$ **próximo**, aplicamos $h_1$
- $h_1(34) = 010$

# Início de nova rodada



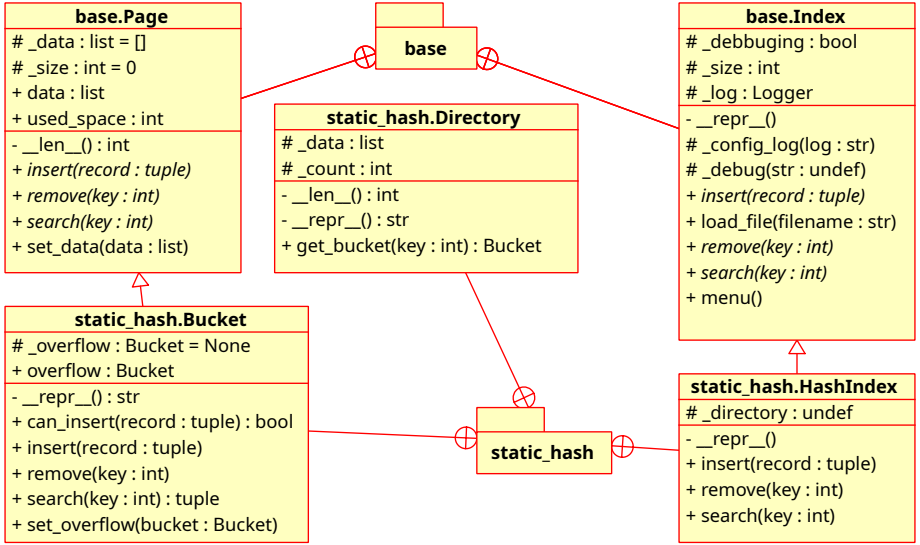

- Vamos inserir a entrada 50*, $h_0(50) = 10$, como $01 <$ **próximo**, aplicamos $h_1$
- $h_1(50) = 010$, mas *bucket* 010 está cheio, precisamos de overflow
- O **próximo** é último *bucket* do nível 0, passamos então para o próximo nível e mudamos o **próximo** para o primeiro *bucket* novamente

# Início de nova rodada




- Vamos inserir a entrada 50*, $h_0(50) = 10$, como $01 <$ **próximo**, aplicamos $h_1$
- $h_1(50) = 010$, mas *bucket* 010 está cheio, precisamos de overflow
- O **próximo** é último *bucket* do nível 0, passamos então para o próximo nível e mudamos o **próximo** para o primeiro *bucket* novamente

# Implementação de *hash* estático



# static_hash.py I

```
1  #!/usr/bin/python3
2  # -*- coding: utf-8 -*-
3
4  from base import Page, Index, get_size, get_arguments
5
6  LOG = '/tmp/static_hash.log'
7
8  class Bucket(Page):
9      '''
10     Data bucket
11     '''
12     def __init__(self, size):
13         # Initialize page for bucket
14         super().__init__(size)
15         # Start without overflow
16         self._overflow = None
```

# static_hash.py II

```
17
18      @property
19      def overflow(self):
20          '''Overflow'''
21          return self._overflow
22
23      def set_overflow(self, bucket):
24          '''Set overflow'''
25          self._overflow = bucket
26
27      def search(self, key):
28          '''
29          Search for key
30          '''
31          # Result list
32          result_list = []
```

# static_hash.py III

```
33          current = self
34          while current is not None:
35              # For each stored record
36              for record in current.data:
37                  # Check if record has the searched key
38                  if record[0] == key:
39                      # Add record to result list
40                      result_list.append(record)
41              current = current.overflow
42          return result_list
43
44      def can_insert(self, record):
45          '''Check if insertion is possible'''
46          # Check if record size is greater than page size
47          if get_size(record) > self._size:
48              raise Exception('Record greater than page size')
```

# static_hash.py IV

```
49          # Check if available space is enough to store record
50          if get_size(record) <= self._size - self.used_space:
51              return True
52          return False
53
54      def insert(self, record):
55          # Try to insert into current bucket
56          if self.can_insert(record):
57              self._data.append(record)
58          else:
59              # If current bucket is full, go to overflow
60              if self._overflow is None:
61                  # If there is no overflow, create it
62                  self._overflow = Bucket(self._size)
63              # Insert into overflow
64              self._overflow.insert(record)
```

# static_hash.py V

```
65
66      def remove(self, key):
67          '''Remove records with given key'''
68          current = self
69          while current is not None:
70              # Data to kept after remove
71              keep_data = []
72              for record in current.data:
73                  # Check if record will be not removed
74                  if record[0] != key:
75                      # Keep record in new data
76                      keep_data.append(record)
77              # Update stored data
78              current.set_data(keep_data)
79              current = current.overflow
80
```

# static_hash.py VI

```
81      def __repr__(self):
82          text = ''
83          if len(self._data) > 0 and hasattr(self._data[0], '__getitem__'):
84              # Representation for str()
85              text = '|'.join([str(record[0])
86                               for record in self._data])
87          text = '[' + text + ']'
88          if self._overflow is not None:
89              text += '->' + str(self._overflow)
90          return text
91
92  class Directory():
93      '''
94      Directory of buckets
95      '''
96      def __init__(self, size):
```

# static_hash.py VII

```
97          # List of buckets
98          self._data = []
99          # Number of buckets
100         self._count = size // get_size(Bucket(size))
101         # Create a bucket for each slot
102         for _ in range(self._count):
103             self._data.append(Bucket(size))
104
105     def get_bucket(self, key):
106         '''
107         Get bucket for a key
108         '''
109         return self._data[key % self._count]
110
111     def __repr__(self):
112         # Representation for str()
```

# static_hash.py VIII

```
113         text = 'Directory: ' + str(len(self._data)) + ' buckets'
114         for pos, bucket in enumerate(self._data):
115             if len(bucket) > 0:
116                 text += '\n' + str(pos) + ': ' + str(bucket)
117         return text
118
119     def __len__(self):
120         # Representation for len()
121         return len(self._data)
122
123
124 class HashIndex(Index):
125     '''
126     Hashing index
127     '''
128     def __init__(self, size, log, debugging=False):
```

# static_hash.py IX

```
129         super().__init__(size, log, debugging)
130         self._debug('Creating directory...')
131         # Create directory
132         self._directory = Directory(size)
133         self._debug('%d buckets created', len(self._directory))
134 
135     def insert(self, record):
136         '''
137         Insert record
138         '''
139         self._debug('Inserting: %s', record)
140         bucket = self._directory.get_bucket(record[0])
141         bucket.insert(record)
142 
143     def remove(self, key):
144         '''
```

## static_hash.py X

```
145         Remove record
146         '''
147         self._debug('Removing: %s', key)
148         bucket = self._directory.get_bucket(key)
149         bucket.remove(key)
150
151     def search(self, key):
152         '''
153         Search for key
154         '''
155         bucket = self._directory.get_bucket(key)
156         return bucket.search(key)
157
158     def __repr__(self):
159         # Representation for str(), same of Directory
160         return str(self._directory)
```

# static_hash.py XI

```
161
162 def main():
163     '''
164     Main function
165     '''
166     args = get_arguments('Static Hashing Index')
167     index = HashIndex(args.page_size, LOG, args.debug)
168     if args.file:
169         index.load_file(args.file)
170     else:
171         index.menu()
172
173 if __name__ == '__main__':
174     main()
```

# Referências

DATE, C. J. **Introdução a sistemas de bancos de dados**. Rio de Janeiro: Elsevier, 2004.

ELMASRI, R.; NAVATHE, S. B. **Sistemas de banco de dados**. 7. ed. São Paulo: Pearson Addison Wesley, 2018.

RAMAKRISHNAN, R.; GEHRKE, J. **Sistemas de gerenciamento de banco de dados**. 3. ed. São Paulo: McGrawHill, 2008.

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. **Sistema de bancos de dados**. 3. ed. São Paulo: Campus, 2007.