CAPÍTULO 12

Mapas com Espalhamento

"Se está muito difícil encontrar o caminho, faça-o"
— Autor desconhecido

12.1 - INTRODUÇÃO

A implementação de Mapa usando Listas não é eficiente pois em todas as operações precisamos percorrer todo Conjunto de associações e isso pode ficar muito custoso a medida que o número de associações cresce.

Novamente podemos utilizar a técnica de Espalhamento para obter resultados melhores. É importante notar que os Mapas são semelhantes aos Conjuntos. Nos Conjuntos, os elementos não podem se repetir. Nos Mapas, as chaves das associações não podem se repetir.

Para utilizar a técnica de Espalhamento precisamos definir basicamente a Função de Espalhamento e a Tabela de Espalhamento.

Vimos que o Código de Espalhamento não é gerado pela estrutura de dados, no caso pelo Mapa, e sim pelo próprio elemento que vamos trabalhar, no caso as chaves. No Java, temos o método `hashCode()` para calcular este código.

A Tabela de Espalhamento é implementada como uma Lista de Lista de Associações. Analogamente ao que fizemos para implementar Conjuntos.

```
public class MapaEspalhamento {
  private List<List<Associacao>> tabela =
      new ArrayList<List<Associacao>>();
}
```

Além disso, precisamos inicializar cada posição da Tabela. Isso será feito no construtor.

```
public MapaEspalhamento() {
  for (int i = 0; i < 100; i++) {
    this.tabela.add(new LinkedList<Associacao>());
  }
}
```

Para ajustar o código gerado pelo método `hashCode()` e gerar um índice válido para a Tabela, vamos definir o método `calculaIndiceDaTabela(String)`

```
private int calculaIndiceDaTabela(String placa) {
  return Math.abs(placa.hashCode()) % this.tabela.size();
}
```

12.2 – Operações

Vamos ver as operações em mapas.

12.3 – Verificando se uma chave existe

Como estamos utilizando a técnica de Espalhamento, para verificar se uma chave existe no Mapa, basta calcular o índice correto da Tabela e procurar na Lista correspondente.

```
public boolean contemChave(String placa) {
  int indice = this.calculaIndiceDaTabela(placa);
  List<Associacao> lista = this.tabela.get(indice);

  for (int i = 0; i < lista.size(); i++) {
    Associacao associacao = lista.get(i);
    if (associacao.getPlaca().equals(placa)) {
      return true;
    }
```

```
  }
  return false;
}
```

12.4 – Removendo uma associação dado uma chave

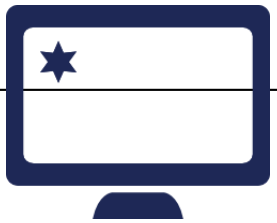
Este procedimento é simples, calculamos o índice e procuramos a chave na Lista correspondente. Ao achar a chave, removemos a associação, se a chave não for achada podemos lançar uma exceção ao usuário.

```
public void remove(String placa) {
  int indice = this.calculaIndiceDaTabela(placa);
  List<Associacao> lista = this.tabela.get(indice);

  for (int i = 0; i < lista.size(); i++) {
    Associacao associacao = lista.get(i);
    if (associacao.getPlaca().equals(placa)) {
      lista.remove(i);
      return;
    }
  }

  throw new IllegalArgumentException("A chave não existe");
}
```

12.5 – Adicionando uma associação dado uma chave

Ao adicionar um associação nova pode ser que a chave da mesma já exista no Mapa. Neste caso, vamos retirar a associação antiga antes de colocar a nova. Isso deve ser feito porque o Mapa não permite chaves repetidas.

```
public void adiciona(String placa, Carro carro) {
  if (this.contemChave(placa)) {
    this.remove(placa);
  }

  int indice = this.calculaIndiceDaTabela(placa);
  List<Associacao> lista = this.tabela.get(indice);
  lista.add(new Associacao(placa, carro));
}
```



Conheça a Alura.

12.6 – Recuperando o valor associado a uma determinada chave

O principal objetivo do Mapa é oferecer uma forma rápida de acessar o valor de uma chave dada.

Então, procuramos a associação pela chave na Lista adequada e devolvemos o valor correspondente. Se a chave não existe então lançamos uma exceção para o usuário.

```
public Carro pega(String placa) {
  int indice = this.calculaIndiceDaTabela(placa);
  List<Associacao> lista = this.tabela.get(indice);

  for (int i = 0; i < lista.size(); i++) {
    Associacao associacao = lista.get(i);
    if (associacao.getPlaca().equals(placa)) {
      return associacao.getCarro();
    }
  }

  throw new IllegalArgumentException("A chave não existe");
}
```

12.7 – Performance das operações

Utilizando a técnica de Espalhamento podemos obter um consumo de tempo médio constante para todas as operações. Isso é muito melhor do que obtivemos implementando Mapas com Listas.

Porém ainda precisaríamos tratar o problema da sobrecarga das Listas da Tabela de Espalhamento. Assim como fizemos no capítulo de Conjuntos com Espalhamento.

Deveríamos transformar a nossa Tabela de Espalhamento em uma tabela dinâmica, ou seja, uma tabela que aumenta e diminui de tamanho conforme a carga do Mapa.

12.8 – Generalização e Parametrização

O nosso Mapa está atrelado fortemente a associações entre `String` e `Carro`. Podemos generalizá-lo e parametrizá-lo para reutilizá-lo em diversas situações.

O primeiro passo é tornar as associações genéricas e parametrizadas.

```
public class Associacao<C, V> {
  private C chave;
  private V valor;

  public Associacao(C chave, V valor) {
    this.chave = chave;
    this.valor = valor;
  }

  public C getChave() {
    return chave;
  }

  public V getValor() {
    return valor;
  }

  @Override
  public String toString() {
    return "{" + this.chave + " -> " + this.valor + "}";
  }
}
```

Depois modificamos a classe `MapaEspalhamento`.

```
public class MapaEspalhamento<C, V> {

  private List<List<Associacao<C, V>>> tabela =
      new ArrayList<List<Associacao<C, V>>>();

  public MapaEspalhamento() {
    for (int i = 0; i < 100; i++) {
      this.tabela.add(new LinkedList<Associacao<C, V>>());
    }
  }

  public boolean contemChave(C chave) {
    int indice = this.calculaIndiceDaTabela(chave);
    List<Associacao<C, V>> lista = this.tabela.get(indice);

    for (int i = 0; i < lista.size(); i++) {
      Associacao<C, V> associacao = lista.get(i);
      if (associacao.getChave().equals(chave)) {
        return true;
      }
    }
    return false;
  }

  public void remove(C chave) {
```

```
    int indice = this.calculaIndiceDaTabela(chave);
    List<Associacao<C, V>> lista = this.tabela.get(indice);

    for (int i = 0; i < lista.size(); i++) {
      Associacao<C, V> associacao = lista.get(i);
      if (associacao.getChave().equals(chave)) {
        lista.remove(i);
        return;
      }
    }

    throw new IllegalArgumentException("A chave não existe");
  }

  public void adiciona(C chave, V valor) {
    if (this.contemChave(chave)) {
      this.remove(chave);
    }

    int indice = this.calculaIndiceDaTabela(chave);
    List<Associacao<C, V>> lista = this.tabela.get(indice);
    lista.add(new Associacao<C, V>(chave, valor));
  }

  public V pega(C chave) {
    int indice = this.calculaIndiceDaTabela(chave);
    List<Associacao<C, V>> lista = this.tabela.get(indice);

    for (int i = 0; i < lista.size(); i++) {
      Associacao<C, V> associacao = lista.get(i);
      if (associacao.getChave().equals(chave)) {
        return associacao.getValor();
      }
    }

    throw new IllegalArgumentException("A chave não existe");
  }

  private int calculaIndiceDaTabela(C chave) {
    return Math.abs(chave.hashCode()) % this.tabela.size();
  }

  private List<Associacao<C, V>> pegaTodas() {
    ArrayList<Associacao<C, V>> associacoes =
        new ArrayList<Associacao<C, V>>();
    for (List<Associacao<C, V>> lista : this.tabela) {
      associacoes.addAll(lista);
    }
    return associacoes;
  }

  @Override
  public String toString() {
    return this.pegaTodas().toString();
  }
}
```

Agora, quando utilizarmos a nossa implementação de Mapa podemos escolher o tipo das chaves e o tipo dos valores.

```
MapaEspalhamento<String, Carro> mapa =
    new MapaEspalhamento<String, Carro>();
```


12.9 - API do Java

Na biblioteca de coleções do Java existem algumas implementações de Mapa. Há duas que utilizam a técnica de Espalhamento (`HashMap` e `Hashtable`).

O uso destas classes é simples e bem parecido com o que fizemos aqui.

```
public class TesteHashMap {
  public static void main(String[] args) {

    HashMap<String, Carro> mapa = new HashMap<String, Carro>();
    mapa.put("abc1234", new Carro("a"));
    System.out.println(mapa);
    mapa.put("abc1234", new Carro("b"));
    System.out.println(mapa);
    mapa.put("def1234", new Carro("c"));
    System.out.println(mapa);

    System.out.println(mapa.containsKey("abc1234"));
    System.out.println(mapa.get("abc1234"));
    mapa.remove("abc1234");
    System.out.println(mapa);
  }
}
```

CAPÍTULO ANTERIOR:

Mapas com Lista

Você encontra a Caelum também em:

Blog Caelum

Cursos Online

Facebook

Newsletter

Casa do Código

Twitter