Benchmarking Java Collections

Geovane Fedrecheski[1], Mariana Cordeiro dos Santos[1]

[1]Departamento de Ciência da Computação
Universidade Estadual do Centro-Oeste (UNICENTRO)
Rua Padre Salvador, 875 – CEP 85015-430
Guarapuava – PR – Brasil – Phone: (42) 3621-1000

geo.arrob@gmail.com, mariana.cordeiro.15@gmail.com

Resumo. *Realizar um Benchmark consiste em executar determinados processos em um computador, avaliando suas performances através da realização de vários testes. Neste artigo, vamos analisar um Benchmark realizado sobre as principais estruturas Java que implementam as interfaces* `List`*,* `Set` *e* `Map`*, sendo que todas fazem parte do Framework Collection.*

Abstract. *Perform a Benchmark consists in run some computer process, evaluating their performances through performing several tests. In this paper, we are going to evaluate a benchmark accomplished on the main Java structures that implements the* `List`*,* `Set` *and* `Map` *interfaces, which are part of the Collection Framework.*

1. Descrição do Benchmark Criado

O trabalho desenvolvido, consiste em um benchmark de tempos de inserção de dez mil elementos aleatórios (gerados e armazenados previamente em arquivos), busca de cem elementos aleatórios na faixa de zero a dez mil (idem) e a remoção destes elementos, nas principais estruturas do Framework Collection do Java. As estruturas utilizadas foram: `ArrayList`, `Vector` e `LinkedList` (Interface `List`); `HashSet`, `LinkedHashSet` e `TreeSet` (Interface `Set`); `HashMap`, `LinkedHashMap` e `TreeMap` (Interface `Map`).

1.1. A marcação do tempo

Para a realização da medição do tempo, uma classe `Benchmark` foi criada, de modo que possa ser extendida por qualquer outra, sendo portanto possível medir o tempo de execução de qualquer código. Assim, temos um método abstrato **exec()**, o qual será escrito pela classe que desejar utilizar-se deste Benchmark. A medição do tempo acontece ao chamar-se o método **start()**, o qual em seu interior marcará o início da contagem, executará o método **exec()** e marcará o final da contagem. O cógido abaixo tem a intenção de tornar mais clara toda esta ideia:

```
1 public abstract void exec();
2
3 public void start() {
```

```
4     time.init();
5     exec();
6     time.end();
7   }
```

O objeto **time**, instância da classe `Time`, encarrega-se de marcar os tempos de relógio e cpu, do instante imediatamente anterior à execução e do momento posterior ao seu término.

1.2. Executando e Salvando Resultados

O procedimento que será descrito a seguir foi realizado para todas as implementações do Framework Collection descritas no primeiro parágrafo desta seção.
São gerados os arquivos especificados para a realização do benchmark e em seguida os valores neles contidos são utilizados para as operações de inserção, busca e remoção — as quais se deseja medir o desempenho. Esse procedimento é repetido n vezes, e na sequencia é tirada a média de cada tempo de cada estrutura. Estes resultados são então gravados em arquivos .dat que servirão para a construção dos gráficos.

2. As Interfaces

Interfaces formam o conjunto de interfaces disponíveis, onde Collections e todas as classes concretas irão derivar de uma ou mais interfaces. List e Set são um tipo de Collection, cada um com suas particularidades. Já um Map não é do mesmo tipo dos demais mas também manipula coleções de elementos.

2.1. List

Representa uma coleção ordenada (ordem de inserção) e que permite duplicatas, suas implementações são:

ArrayList: Pode ser visto como um array (vetor) porém dinâmico. Ele é organizado pelo índice, ou seja, temos alguma garantia quanto a ordem que encontraremos os elementos.

Vector: É basicamente um ArrayList, no entanto seus métodos são sincronizados o que significa que o acesso por vários processos simultaneamente é coordenado.

LinkedList: Muito similar as duas coleções vistas anteriormente, porém todos os elementos são ligados entre si. Seu desempenho é superior aos do ArrayList e Vector quando necessitamos inserir elementos no início da coleção, no entanto ao precisar obter algum elemento pelo índice o desempenho é inferior.

2.2. Set

Representa uma coleção que não pode conter duplicatas, implementa uma abstração dos conjuntos matemáticos, também contendo três implementações:

HashSet: Caracteriza-se por não aceitar duplicatas, característica derivada do Set, ser uma coleção desordenada e desorganizada, isto é, não há nenhuma garantia quanto a ordem que os elementos serão percorridos.

LinkedHashSet: É uma versão organizada do HashSet, ou seja, existe algum tipo de seqüência não-aleatória durante a iteração dos elementos, neste caso a ordem de inserção é respeitada. Por ser um Set, o LinkedHashSet não aceita duplicatas. Deve-se utilizar o LinkedHashSet ao invés do HashSet quando a ordem de iteração dos elementos é importante.

treeSet: É um Set ordenado e como tal não aceita duplicatas, no TreeSet os elementos inseridos serão percorridos de acordo com sua ordem natural e de forma ascendente.

2.3. Map

Implementa objetos que armazenam um elemento e o removem através da sua chave, não aceitam chaves duplicadas. Suas implementações são:

HashMap: É um Map desorganizado, isto é, a ordem de iteração dos elementos é desconhecida, e desordenado.

LinkedHashMap: É muito similar ao LinkedHashSet, porém esta é a versão que implementa a interface Map, logo ao armazenar os objetos é necessária uma chave. Ao contrário do HashMap, o LinkedHashMap é organizado, o que significa dizer que durante a iteração dos elementos ele respeita a ordem que estes foram inseridos na coleção.

TreeMap: ordena seus elementos através da chave por alguma regra. Quando esta ordem não é definida pela interface Comparable ou por um objeto Comparator1 o TreeMap busca a ordem natural dos elementos.

3. Sobre os Benchmarks

Na computação são necessários testes de hardware e software para gerenciamento de sua performance, esses testes são conhecidos como benchmarks. Benchmark é o ato de executar um programa de computador, um conjunto de programas ou outras operações, a fim de avaliar a performance relativa de um objeto, normalmente executando uma série de testes padrões e ensaios nele.

Benchmark é útil para o entendimento de como o gerenciador de banco de dados responde sob a variação de condições. Pode-se criar cenários que testam o tratamento de deadlock, performance dos utilitários, diferentes métodos de carregar dados, características da taxa de transição quando mais usuários são adicionados e ainda o efeito na aplicação usando uma nova versão do produto.

4. Resultados (Gráficos)

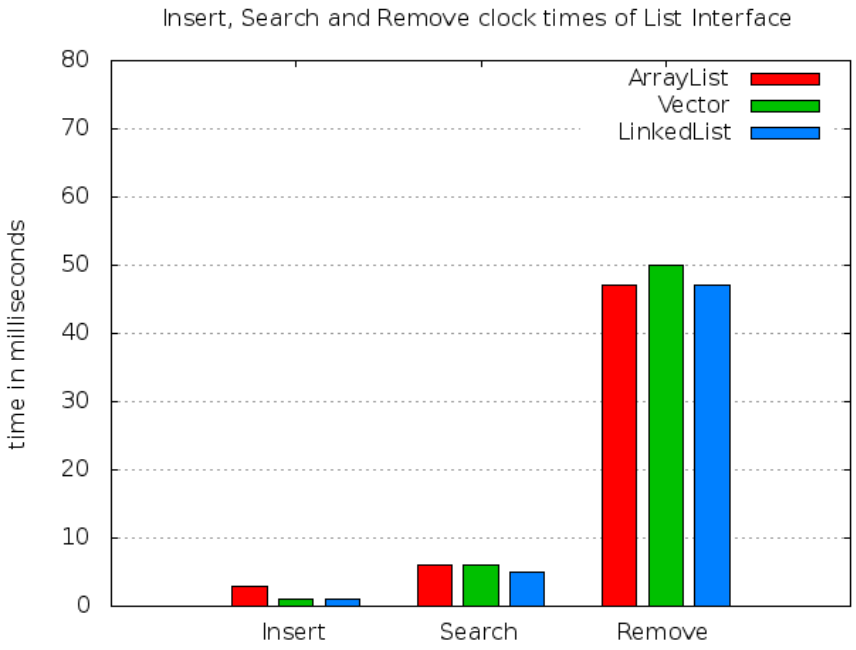

Figura 1. Interface List Clock times

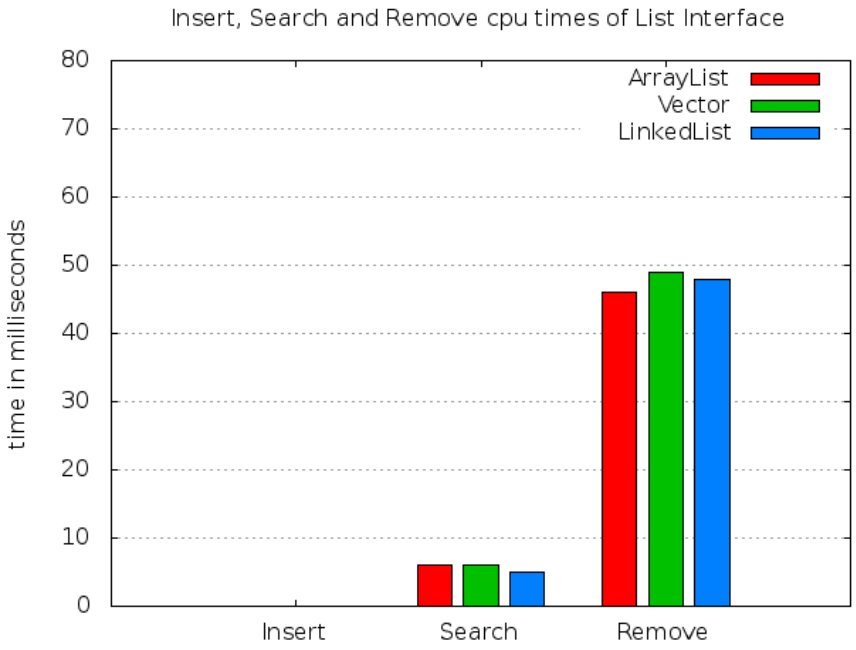

Figura 2. Interface List Cpu times

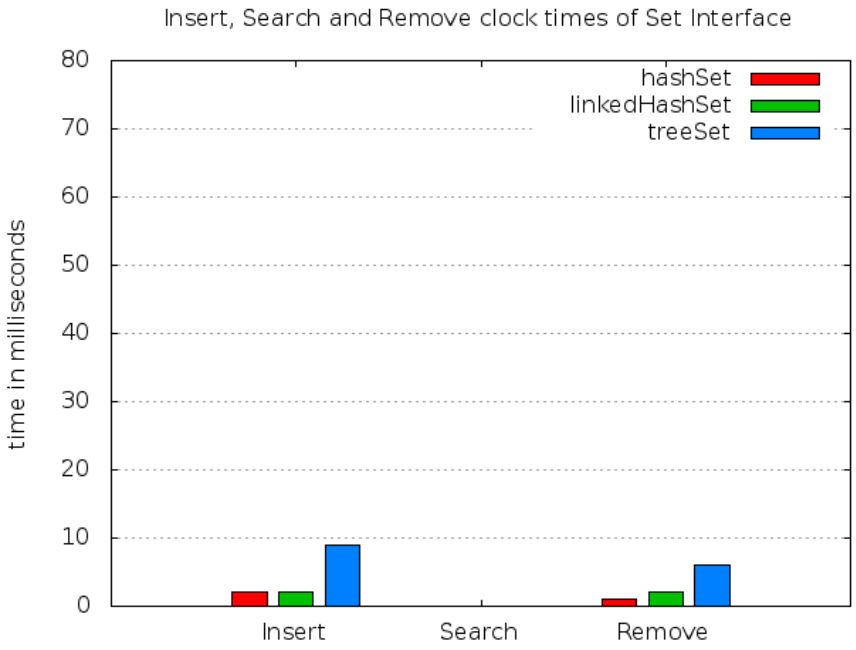

Figura 3. Interface Set Clock times

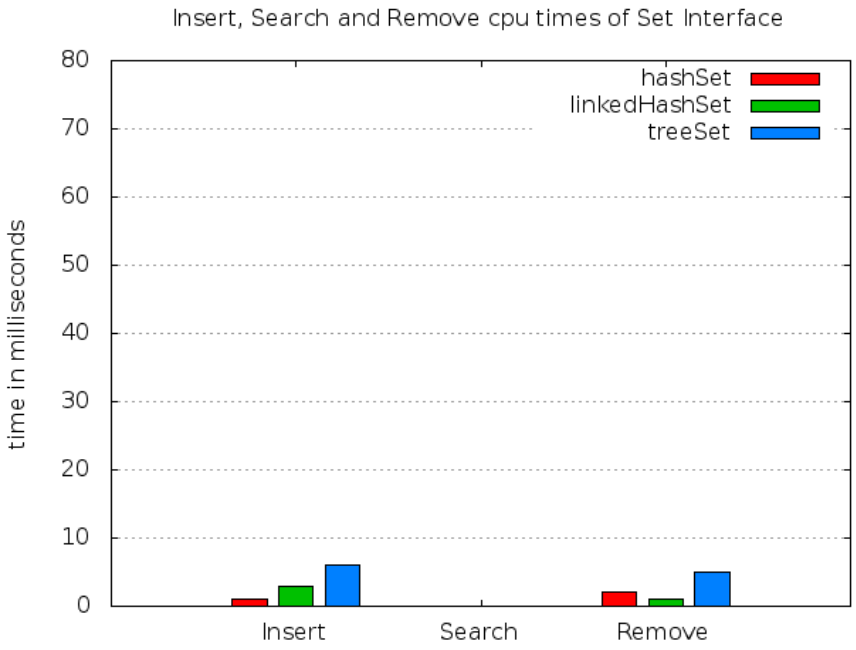

Figura 4. Interface Set Cpu times

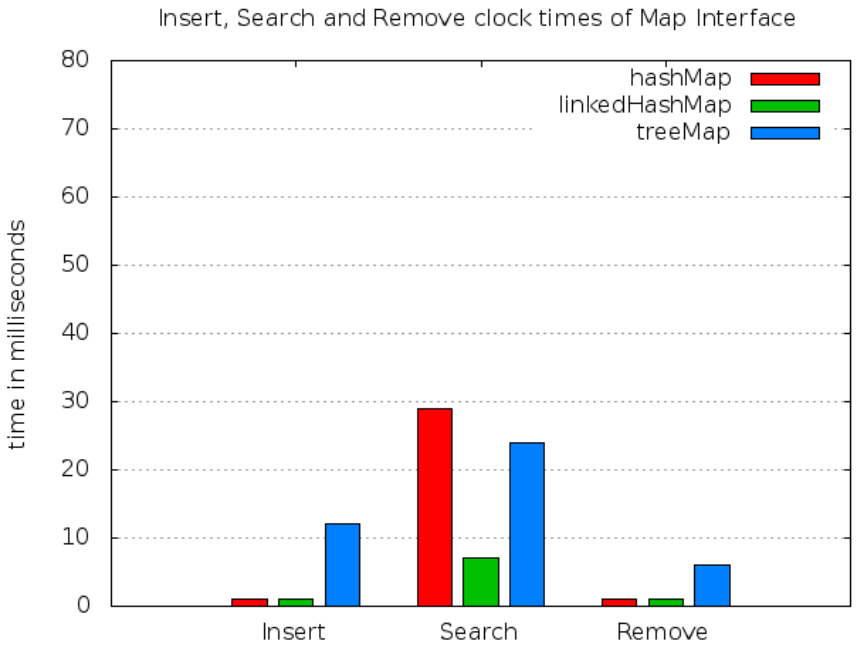

Figura 5. Interface Map Clock times

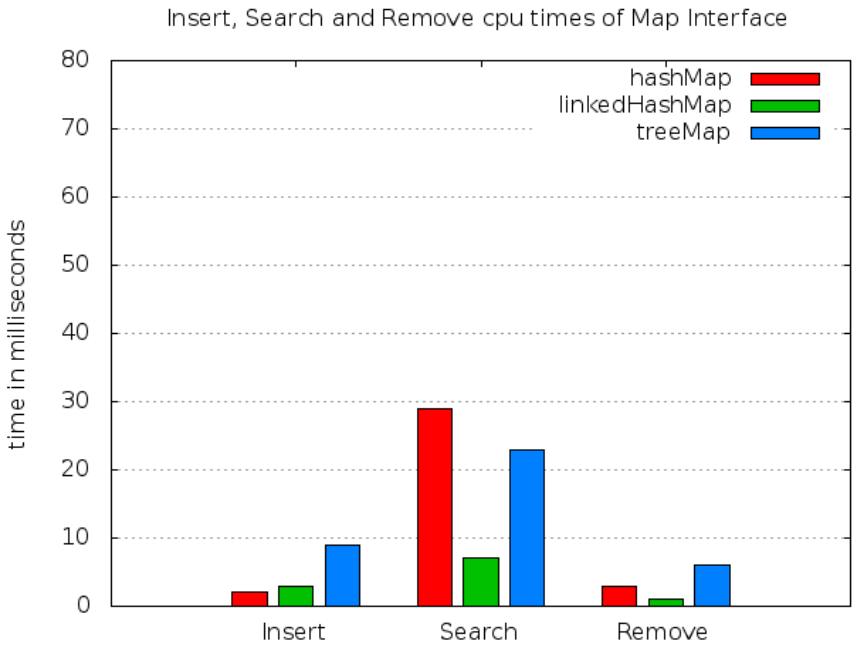

Figura 6. Interface Map Cpu times