Aula 03: 22/03/2017

Tópicos

- custo amortizado
- `Bags`
- iteradores

Pilhas com redimensionamento (aula passada)

Leitura: Pilhas, PF

O método privado `resize()` faz um redimensionamento: aumenta ou diminui o vetor que abriga a pilha.

Graças à maneira como o redimensionamento é invocado por `pop()` e `push()`, o vetor `a[]` sempre está pelo menos 25% cheio.

```
public class Stack<Item> {
    private Item[] a = null;
    private int n = 0;

    public Stack() {
        a = (Item[]) new Object[1];
        n = 0;
    }

    public boolean isEmpty() {
        return n == 0;
    }

    public int size() {
        return n;
    }

    public void push(Item item) {
        if (n == a.length) resize(2*a.length);
        a[n++] = item;
    }

    public Item pop() {
        Item item = a[--n];
        a[n] = null; // Avoid loitering
        if (n > 0 && n == a.length/4) resize(a.length/2);
        return item;
    }

    private void resize(int max) {
        Item[] tmp = (Item[]) new Object[max];
        for (int i = 0; i < n; i++) {
            tmp[i] = a[i];
```

```
        }
        a = tmp;
    }
}
```

Custo amortizado

O **custo amortizado** de uma operação é o custo médio da operação quando considerada em uma sequência de operações do ADT (não se trata, portanto, da média de um conjunto aleatório de execuções da operação).

Consideremos uma sequência de m operações `push()`.

Sejam n_i e t_i o número de itens e o tamanho da pilha depois da operação i.

O custo real da i-ésima operação `push()` é 1 se há espaço e n_i se a pilha está cheia.

Custo de uma operação é $O(m)$. Portanto, o custo das m operações $O(m^2)$. Isso está correto, mas é um exagero.

Exemplo

operação (n)	t	custo
1	1	1
2	2	1 + 1
3	4	1 + 2
4	4	1
5	8	1 + 4
6	8	1
7	8	1
8	8	1
9	16	1 + 8
10	16	1
[...]	16	1
17	32	1 + 16
33	64	1 + 32

Custo total:

$\sum_{i=1}^{m} c_i = m + \sum_{i=0}^{k} 2^i = m + 2^{k+1} - 1 < m + 2m - 1 < 3m$, onde $k = \lfloor \lg(m-1) \rfloor$.

Análise de algoritmos

Leitura: Analysis of Algoriths, S&W

Escrevemos $\sim f(n)$ para representar qualquer função que, quando dividida por $f(n)$ vai para 1 quano $n \to \infty$.

$g(n) \sim f(n)$ indica que $g(n)/f(n) \to 1$ quando n cresce.

Note a diferença com a notação O().

$f(n)$ é $O(g(n))$ se existem constantes c e n_0 tais que $f(n) \leq g(n)$ para $n > n_0$.

Memória

Memória consumida por objetos primitivos

```
tipo      bytes
---------------
boolean     1
byte        1
char        2
int         4
float       4
long        8
double      8
```

```
int[] a = new int[n];
--------------------
overhead = 16 bytes
valor n = 4 bytes
total 24 + 4n

double[] c = new double[n];
---------------------------
overhead = 16 bytes
valor n = 4 bytes
total 24 + 8n

double[][] t = new double[n][m];
--------------------------------
valor   m = 4 bytes
m referências = 8m
valores n = 4m bytes
valores nm doubles = 8nm
Total = 24 + 8m + m*(24+8n) = 24 + 32m + 8mn

Resumo:

tipo       bytes
-------------------
int[]      ~4n
double[]   ~8n
double[][] ~8nm
```

Pilhas iteráveis

```
Iterator<String> it = s.iterator();
while (it.hasNext()) {
    StdOut.println(it.next());
}
```

Com *foreach statement*.

```
        for (String bla: s) {
            StdOut.println(bla);
        }
        StdOut.println("----\n");
```

Ingredientes:

- devemos implementar um método `iterator()` que retorna um objeto `Iterator<Item>`
- A classe `Iterator` deve incluir dois métodos `hasNext()` e `next()`

Receita

Passo 1: colocar `import java.util.Iterator;`;

Passo 2: colocar `implements Iterable<Item>` na declaração da classe

```
public interface Iterable<Item> {
    Iterator<Item> iterator();
}
```

Passo 3: escrever um método `Iterator<Item> iterator()`

```
public Iterator<Item> iterator() {
    return new ListIterator();
}
```

Passo 4: escrever a classe `ListIterator` que implementa `Iterator<Item`:

```
public interface Iterator<Item> {
    boolean hasNext();
    Item next();
    void remove();
}
```

implementação

```
import java.util.Iterator; // passo 1

public class Stack<Item> implements Iterable<Item> { // passo 2

    private Node first;
    private int n;

    private class Node {
        Item item;
        Node next;
    }

    public boolean isEmpty() {
        return first == null;
    }
```

```
public int size() {
    return n;
}

public void push(Item item) {
    Node oldfirst = first;
    first = new Node();
    first.item = item;
    first.next = oldfirst;
    n++;
}

public Item pop() {
    Item item = first.item;
    first = first.next;
    n--;
    return item;
}

// passo 3
public Iterator<Item> iterator() {
    return new ListIterator();
}

// passa 4
private class ListIterator implements Iterator<Item> {
    private Node current = first;

    public boolean hasNext() {
        return current != null;
    }

    public Item next() {
        Item item = current.item;
        current = current.next;
        return item;
    }

    public void remove() {
        throw new UnsupportedOperationException();
    }
}

// unit test
public static void main(String[] args) {
    Stack<String> s = new Stack<String>();

    while (!StdIn.isEmpty()) {
        String item = StdIn.readString();
        if (!item.equals("-")) s.push(item);
```

```
            else if (!s.isEmpty()) StdOut.println(s.pop() + " ");
        }

        StdOut.println("\nConteúdo usando while (...): ");
        StdOut.println("----");
        Iterator<String> it = s.iterator();
        while (it.hasNext()) {
            StdOut.println(it.next());
        }
        StdOut.println("----\n");

        StdOut.println("\nConteúdo usando foreach statement: ");
        StdOut.println("----");
        for (String bla: s) {
            StdOut.println(bla);
        }
        StdOut.println("----\n");

        StdOut.println("(" + s.size() + " left on stack)");
    }
}
```

Bags (sacos)

Leitura: Sacos, PF

API

```
public class Bag<Item> implements Iterable<Item>
-----------------------------------------------
               Bag()           cria um saco de Items vazio
          void add(Item item)  coloca item neste saco
Iterator<Item> iterator()      um iterador que percorre os itens do saco
       boolean isEmpty()       este saco está vazio?
           int size()          número de Items neste saco
```

Uso

```
Welcome to DrJava.  Working directory is /home/coelho/Dropbox/mac0323/2017/aulas/bags
> Bag<Integer> saco = new Bag<Integer>()
> saco.isEmpty()
true
> saco.add(11)
> saco.add(22)
> saco.add(33)
> saco.add(44)
> saco.add(33)
```

```
> saco.size()
5
> Iterator<Integer> it = saco.iterator()
Static Error: Undefined class 'Iterator'
> import java.util.Iterator; // auto-import
Iterator<Integer> it = saco.iterator()
> import edu.princeton.cs.algs4.StdOut;
> while (it.hasNext()) {
    int i = it.next();
    StdOut.println(i);
}
33
44
33
22
11
> for (int i: saco) StdOut.println(i);
33
44
33
22
11
>
```

implementação

```
import java.util.Iterator;

public class Bag<Item> implements Iterable<Item> {

    private Node first;
    private int  n;

    private class Node {
        private Item item;
        private Node next;
    }

    public Bag() {  // construtor
        first = null;
    }

    public void add(Item item) {
        Node oldfirst = first;
        first = new Node();
        first.item = item;
        first.next = oldfirst;
        n++;
    }
```

```
    public int size() {
        return n;
    }

    public boolean isEmpty() {
        return n == 0;
    }

    public Iterator<Item> iterator() {
        return new ListIterator();
    }

    private class ListIterator implements Iterator<Item> {

        private Node current = first;

        public boolean hasNext() {
            return current != null;
        }

        public Item next() {
            Item item = current.item;
            current = current.next;
            return item;
        }

        public void remove() {
            throw new UnsupportedOperationException();
        }
    }
}
```

Cliente

```
import edu.princeton.cs.algs4.StdIn;
import edu.princeton.cs.algs4.StdOut;

public class Stats {

   public static void main(String[] args) {
      Bag<Integer> bag = new Bag<Integer>();
      while (!StdIn.isEmpty()) {
         bag.add(StdIn.readInt());
      }

      int n = bag.size();
      int soma = 0; // inicialização é supérflua
      for (int x : bag) {
```

```
            soma += x;
        }
        double media = soma/n;
        soma = 0;
        for (int x : bag) {
            soma += (x - media)*(x - media);
        }
        double desvpad = Math.sqrt(soma/(n-1));
        StdOut.printf("Media: %.2f\n", media);
        StdOut.printf("Desv padrao: %.2f\n", desvpad);
    }
}
```