



Programação Avançada

2c

ADT Map | Impl. com Árvore Binária de Pesquisa

Bruno Silva, Patrícia Macedo

Sumário



- ADT Map
- Interface `Map<K,V>`
 - Implementação (fornecida) com `List` e exemplo
 - Implementação (a finalizar) com BST
 - Motivação
 - Algoritmos recursivos
- Exercícios

ADT Map

- O ADT Map consiste num contentor de mapeamentos *chave:valor*, vulgarmente também chamado de *dicionário*.
 - Não permite chaves duplicadas;
 - O mesmo valor pode estar associado a múltiplas chaves.

Key (Menu Item)	Value (Calories)
"Bacon & Cheese Hamburger"	790
"Chicken Salad with Grilled Chicken"	350
"French Fries (small)"	320
"Onion Rings (small)"	320
...	...

Interface `Map<K, V>`

A especificação do ADT Map na linguagem Java é descrito numa *interface*:

```
package pt.pa.adts;

/**
 * An object that maps keys to values. A map cannot contain duplicate keys; each key can map to at most one value.
 * @param <K> the type of keys maintained by this map
 * @param <V> the type of mapped values
 */
public interface Map<K, V> {
    V put(K key, V value) throws NullPointerException;
    V get(K key) throws NullPointerException;
    V remove(K key) throws NullPointerException;
    boolean containsKey(K key) throws NullPointerException;
    Collection<K> keys();
    Collection<V> values();
    int size();
    boolean isEmpty();
    void clear();
}
```

Versão comentada da interface disponível no projeto base:

https://github.com/pa-estsetubal-ips-pt/ADTMap_Template

Implementação (fornecida) com `List` e exemplo

- No **projeto base** (faça *git clone*) é fornecida uma implementação completa na classe `MapList` e um exemplo de utilização que mapeia números para o seu número de ocorrências.

```
int[] numbers = {1,4,3,7,4,8,9,1,4,6,4,7,6,9,5,3,6,8,4,6,9};

Map<Integer, Integer> uniqueCount = new MapList<>();

for(int num : numbers) {
    if(uniqueCount.containsKey(num)) {
        int curCount = uniqueCount.get(num);
        uniqueCount.put(num, curCount + 1);
    } else {
        uniqueCount.put(num, 1);
    }
}

//Do not use .toString() for this!
//TODO: 1. show only unique numbers
//TODO: 2. show unique numbers and how many times they occur
```

- **?** Complete o código em falta, utilizando as operações de `Map`.

Implementação (fornecida) com `List` e exemplo

Implementação fornecida utilizando uma estrutura de dados linear:

```
public class MapList<K,V> implements Map<K,V> {  
  
    private final List<KeyValue> mappings;  
  
    public MapList() {  
        mappings = new ArrayList<>();  
    }  
  
    @Override  
    ...  
  
    private class KeyValue {  
        private K key;  
        private V value;  
  
        public KeyValue(K key, V value) {  
            this.key = key;  
            this.value = value;  
        }  
    }  
}
```

- Guarda os mapeamentos `KeyValue` numa instância de `ArrayList` (já contém funcionalidades de manipulação de um array).

ADT Map | Impl. com BST

- Se notar, todos os métodos principais de `Map` envolvem a pesquisa de uma chave na estrutura de dados subjacente; aliás, basta olhar para o *javadoc* da interface.
- Como visto anteriormente, as **árvores binárias de pesquisa** permitem acelerar significativamente a pesquisa de elementos.
 - Estrutura de dados linear ➡ $O(n)$
 - Árvore binária de pesquisa ➡ $O(\log n)$
- **!** Temos então o objetivo de obter uma implementação de `Map` baseada nesta última estrutura de dados.
 - A classe `MapBST` no projeto base contém uma implementação praticamente total.

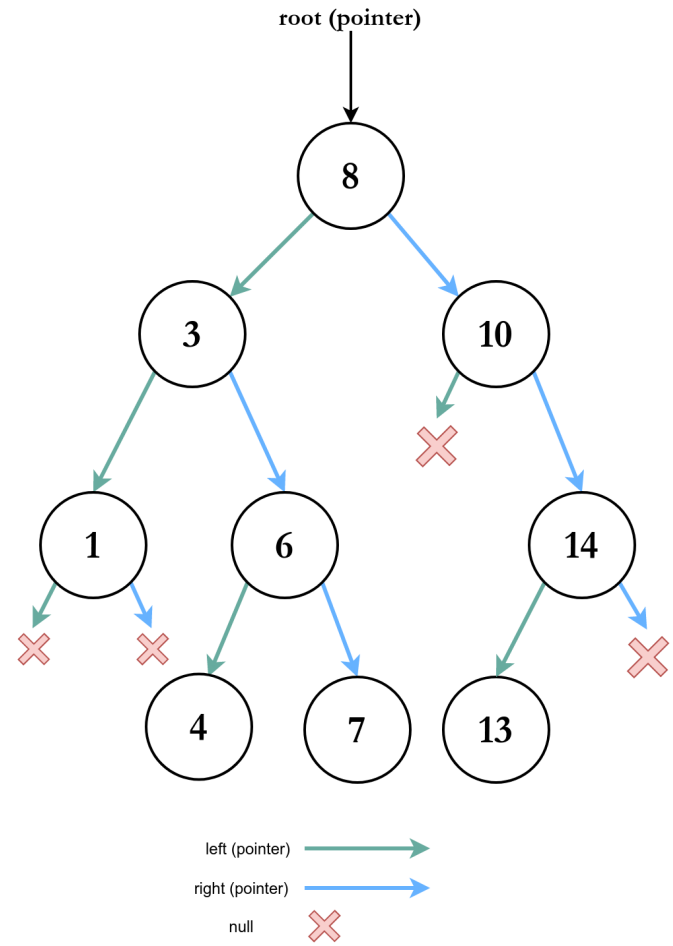
ADT Map | Impl. com BST

- Particularidade desta implementação: só poderá aceitar chaves que sejam comparáveis (por motivos que deverão ser óbvios).

```
public class MapBST<K extends Comparable<K>, V> implements Map<K,V> {  
  
    private BSTNode root;  
  
    public MapBST() {  
        this.root = null;  
    }  
  
    @Override  
    ...  
    private class BSTNode {  
        private K key;  
        private V value;  
  
        private BSTNode parent; //útil para operação de remoção  
        private BSTNode left;  
        private BSTNode right;  
  
        public BSTNode(K key, V value, BSTNode parent, BSTNode left, BSTNode right) {  
            ...  
        }  
    }  
}
```


ADT Map | Impl. com BST

- Exemplo da **estrutura de dados** contendo 9 números (**keys**) ➡
 - Ponteiros `parent` estão omitidos.
 - **values** estão omitidos.
- ⚠ A **abstração recursiva** de árvores permitirá a utilização de algoritmos *recursivos* em algumas situações.



ADT Map | Impl. com BST

- Implementações "óbvias":

```
public class MapBST<K extends Comparable<K>, V> implements Map<K,V> {  
  
    private BSTNode root;  
  
    public MapBST() {  
        this.root = null;  
    }  
  
    @Override  
    public boolean isEmpty() {  
        return (this.root == null);  
    }  
  
    @Override  
    public void clear() {  
        this.root = null;  
    }  
  
    ...  
}
```

ADT Map | Impl. com BST

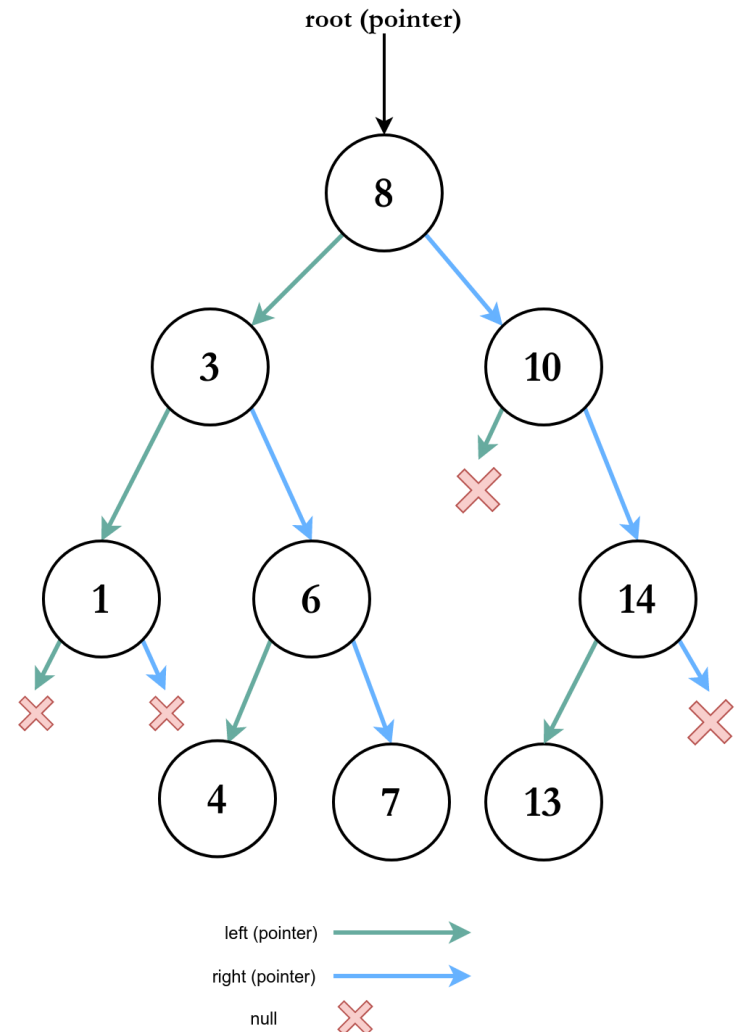
- Implementação *recursiva* de `size()` :

```
public class MapBST<K extends Comparable<K>, V> implements Map<K,V> {  
  
    @Override  
    public int size() {  
        return size(this.root);  
    }  
  
    private int size(BSTNode treeRoot) {  
        if(treeRoot == null) return 0;  
        else return 1 + size(treeRoot.left) + size(treeRoot.right);  
    }  
  
    ...  
}
```

- **!** Note que a utilização de um atributo `int size` é mais eficiente. O propósito aqui é o de **introduzir/utilizar a abstração recursiva**.

ADT Map | Impl. com BST

- Implementação *recursiva* de `size()` :
 - Simule o algoritmo anterior no diagrama.



ADT Map | Impl. com BST

- ? Por forma a poder testar a implementação da classe `MapBST` forneça a implementação dos seguintes dois métodos auxiliares:
 - `private BSTNode searchNodeWithKey(K key, BSTNode treeRoot)`
 - Dada a raiz de uma (sub-)árvore, pesquisa o nó que contém essa chave; `null` se não existir. **Forneça uma implementação recursiva.**
 - `private BSTNode getLeftmostNode(BSTNode treeRoot)`
 - Dada a raiz de uma (sub-)árvore, pesquisa o seu nó mais à esquerda (*contém a chave "mínima"*); `null` se não existir. **Forneça uma implementação recursiva ou iterativa.**

ADT Map | Impl. com BST

- ? Execute o método `main()` utilizando a implementação completa de `MapBST` ;
- ? Utilize o método `MapBST.toString()` que irá mostrar uma representação textual da árvore subjacente:

```
MapBST of size = 8:  
├── {key=9, value=3  
│   ├── {key=8, value=2  
│   │   ├── {key=7, value=2  
│   │   │   ├── {key=6, value=4  
│   │   │   └── {key=5, value=2  
│   │   │       ├── {key=4, value=5  
│   │   │       │   ├── {key=3, value=2  
│   │   │       │   └── {key=1, value=2
```

- ? Teste a remoção de mapeamentos, verificando as árvores resultantes.

Exercícios | Implementação

1. Altere a implementação por forma a que os métodos `keys()` e `values()` utilizem uma *travessia em-ordem* da árvore.
 - No caso de `keys()`, dado que são as chaves da árvore, a coleção irá conter esses elementos ordenados.

Exercícios | Implementação

2. Adicione ao *output* do método `toString()` informação sobre a **altura da árvore**, e.g.:

```
MapBST of size = 8 and height = 3:
├── {key=9, value=3
│   ├── {key=8, value=2
│   │   ├── {key=7, value=2
│   │   │   ├── {key=6, value=4
│   │   └── {key=5, value=2
│   └── {key=4, value=5
│       ├── {key=3, value=2
│       └── {key=1, value=2
```

- Implemente/utilize um método auxiliar recursivo:
 - `private int height(BSTNode treeRoot)`

Exercícios | Utilização

3. Crie uma classe `MainMenu` que, no método `main()` replique numa instância de ADT Map o menu de calorias apresentado no início da aula.
 - Adicionalmente, e utilizando as operações de `Map`, mostre apenas os *items* do menu com calorias superiores a um *threshold* t .