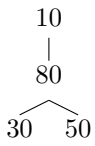
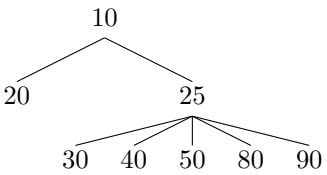
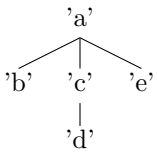
1 Introdução

Uma N-ÁRVORE é uma árvore com um número variável de filhos.

Vejamos alguns exemplos:







Para o projeto de AED vamos desenvolver uma variante deste tipo N-ÁRVORE onde todas as instâncias de árvore têm de respeitar a seguinte invariante:

A travessia prefixa de qualquer N-ÁRVORE *resulta sempre numa sequência crescente de valores.*

Nos exemplos acima, a segunda e terceira árvore são instâncias corretas mas a primeira não é, dado que o valor 80 surgiria antes dos valores 30 e 50 numa travessia prefixa.

2 O tipo árvore

A interface seguinte mostra os serviços públicos que queremos incluir na nossa implementação:

```
1  public interface NTree<T extends Comparable<T>>
2                          extends Iterable<T> {
3    public boolean isEmpty();
4    public boolean isLeaf();
5    public int size();
6    public int countLeaves();
7    public int height();
8    public T min();
9    public T max();
10   public boolean contains(T elem);
11   public void insert(T elem);
12   public void delete(T elem);
13   public List<T> toList();
14   public Iterator<T> iterator();
15 }
```

Ler o *javadoc* do ficheiro `NTree.java` para mais informações sobre as operações acima.

De notar que o tipo genérico `T` deve implementar o tipo `Comparable`. Ou seja, as nossas árvores apenas armazenam valores de tipos que implementam uma ordem total. Exemplos de tipos comparáveis incluem `Integer` e `String`.

Outra consequência de respeitar a invariante é que as nossas árvores não podem armazenar elementos repetidos. Assim, tentar inserir um elemento já existente não deve produzir qualquer efeito.

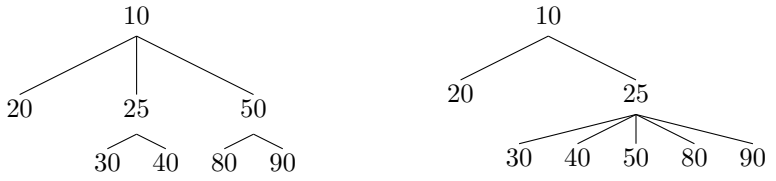
3 O que fazer

É-vos pedido que implementem esta interface na classe `ArrayNTree`.

A classe implementa uma versão limitada do tipo original, isto é, cada nó poderá guardar um máximo número de filhos, a que vamos chamar a *capacidade* dos nós.

Como o nome da classe indica, os filhos de um dado nó devem ser guardados num *array*. Assim, deverão incluir um atributo de tipo `ArrayNTree<T>[]`[1].

A vossa resolução deve incluir um método `clone` que cria uma instância igual da árvore e um método `equals` que compara a árvore com outro objecto dado. Considera-se que duas árvores do tipo `NTree` são iguais se contêm os mesmos elementos, não precisando de ter a mesma estrutura[2]. Por exemplo, as seguintes árvores são consideradas iguais:



A implementação do iterador deve devolver a sequência dos valores da árvore a partir de uma travessia prefixa. Assim, o iterador irá devolver uma sequência crescente de valores.

É-vos dado um esqueleto da classe `ArrayNTree` que devem preencher. Respeitem as assinaturas dos métodos fornecidos. Podem, naturalmente, incluir os métodos auxiliares que acharem adequados.

Dada a natureza recursiva do tipo árvore, espera-se que a maioria dos métodos públicos a implementar sejam recursivos.

Incluam comentários informativos e, nos *javadoc* dos vários métodos, descrevam as respetivas complexidades temporais do melhor caso e do pior caso para uma árvore com n elementos. Usem este formato nos *javadoc*:

```
1 /**
2  * @best-case  O(1)
3  * @worst-case O(1)
4  * ...
5  */
6 public boolean isEmpty();
```

[1] Nesta implementação um objeto do tipo árvore possui um *array* para mais objetos do tipo árvore. Não há necessidade de haver o tipo auxiliar `Node`, como usámos para construir as árvores binárias.

[2] Dois objectos do tipo `NTree` podem ser considerados iguais sem terem necessariamente de ser ambos do tipo `ArrayNTree`. Por exemplo, poderia haver uma segunda classe a implementar a interface `NTree`.

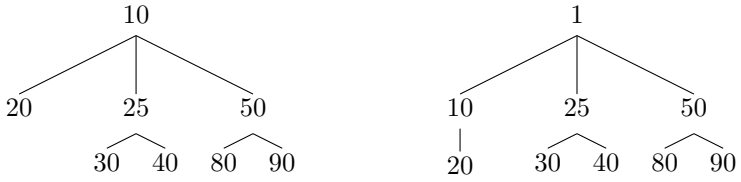
4 Sobre inserir e apagar elementos

A inserção e remoção de elementos são as operações mais difíceis de implementar, dado que é preciso manter a invariante. Nesta secção, vamos dar algumas pistas à sua realização. Nos exemplos seguintes consideramos árvores com capacidade de três filhos por nó.

Inserção. Podem ocorrer vários tipos de situação que devem ter atenção particular.

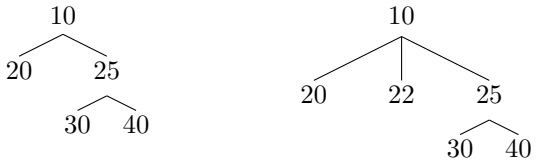
1. A árvore estar vazia.

2. Se o elemento a guardar for menor que a raíz, tem de ficar na raíz e devemos então inserir a raíz antiga no resto da árvore. No exemplo seguinte, inserimos na árvore da esquerda o valor 1:

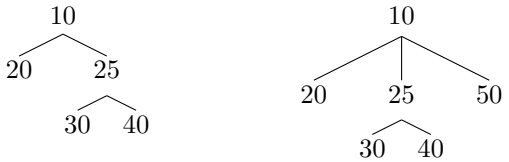


3. O elemento `E` a guardar num *array* com espaço livre: este terá de se colocar ou (a) no início ou meio do vetor (tendo-se de empurrar os restantes) ou (b) no fim do vetor. De notar que só se pode colocar `E` no vetor da raíz se o maior dos filhos do elemento anterior for menor que `E` (cf. o segundo exemplo dos seguintes).

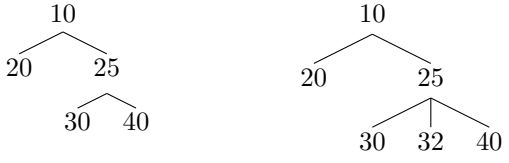
   a) Exemplo, inserir 22:



   b) Exemplo, inserir 50:

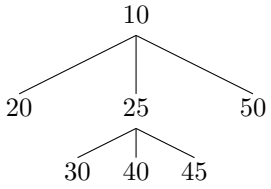


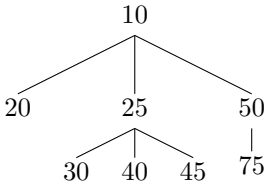
   b) Exemplo, inserir 32:



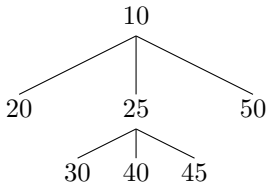
4. O elemento a guardar num *array* sem espaço livre.

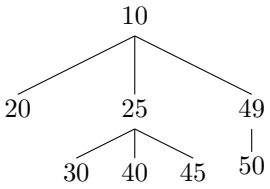
   Exemplo, inserir 75:



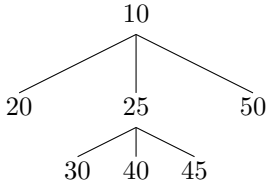


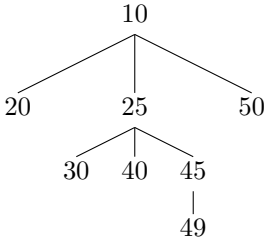
Exemplo, inserir 49:





De notar que neste caso também poderíamos resolver desta forma:

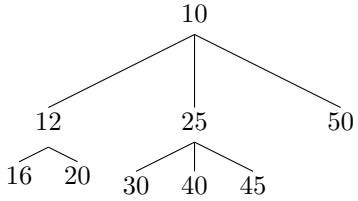


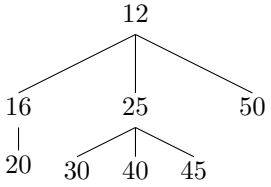


Existe alguma liberdade na forma como implementam a inserção. Não é necessário manter a árvore o mais balanceada possível mas devem aproveitar a capacidade disponível dos vários nós. A principal restrição é manter a invariante.

Remoção. Para remover é preciso ter atenção que pode ser necessário mexer em alguns dos outros valores. Ao remover a raíz é necessário ir buscar o menor filho para ocupar essa posição, e assim sucessivamente até se chegar a uma folha.

Exemplo, remover 10:





5 Entrega

Devem entregar um zip chamado `projetoAED_XXX`, sendo `XXX` o número de grupo. O zip deve apenas ter a classe `ArrayNTree`, devidamente implementada. O zip deve ser entregue via moodle até às 23:59 do dia 20 de Maio.

Identifiquem o número do grupo e os vossos números de aluno e nomes no *javadoc* da classe.