




9 DE NOVEMBRO DE 2022

# RELATÓRIO ESINF TRABALHO 2- PROJETO ESTATÍSTICAS FAO

1ºSEMESTRE 2022/2023

JOSÉ GOUVEIA - 1211089  
PEDRO PEREIRA – 1211131  
TIAGO OLIVEIRA – 1211128  
ALEXANDRE GERAÇÃO – 1211151  
RICARDO VENÂNCIO – 1210828  
DIOGO CARVALHO - 1200611



# Índice

Índice.....	1
Introdução.....	2
Problema.....	3
Diagrama de classes.....	4
Solução.....	5
Exercicio1 .....	5
Métodos para exercício 2 .....	5
Métodos para exercício 3 .....	5
Métodos para exercício 4 .....	7
Métodos para exercício 5 .....	7
Exercicio2 .....	9
Exercicio3 .....	10
Exercicio4 .....	11
Exercicio5 .....	13
Complexidade do Algoritmo .....	13
Melhoramentos possíveis .....	14

## Introdução

No contexto da unidade curricular de Estruturas de Informação (ESINF), foi-nos proposto desenvolver um projeto que incluía a criação de uma biblioteca de classes, respetivos métodos e testes que permitiam gerir a informação relativa aos dados de productos agrícolas e pecuários recolhidos pelo FAO.

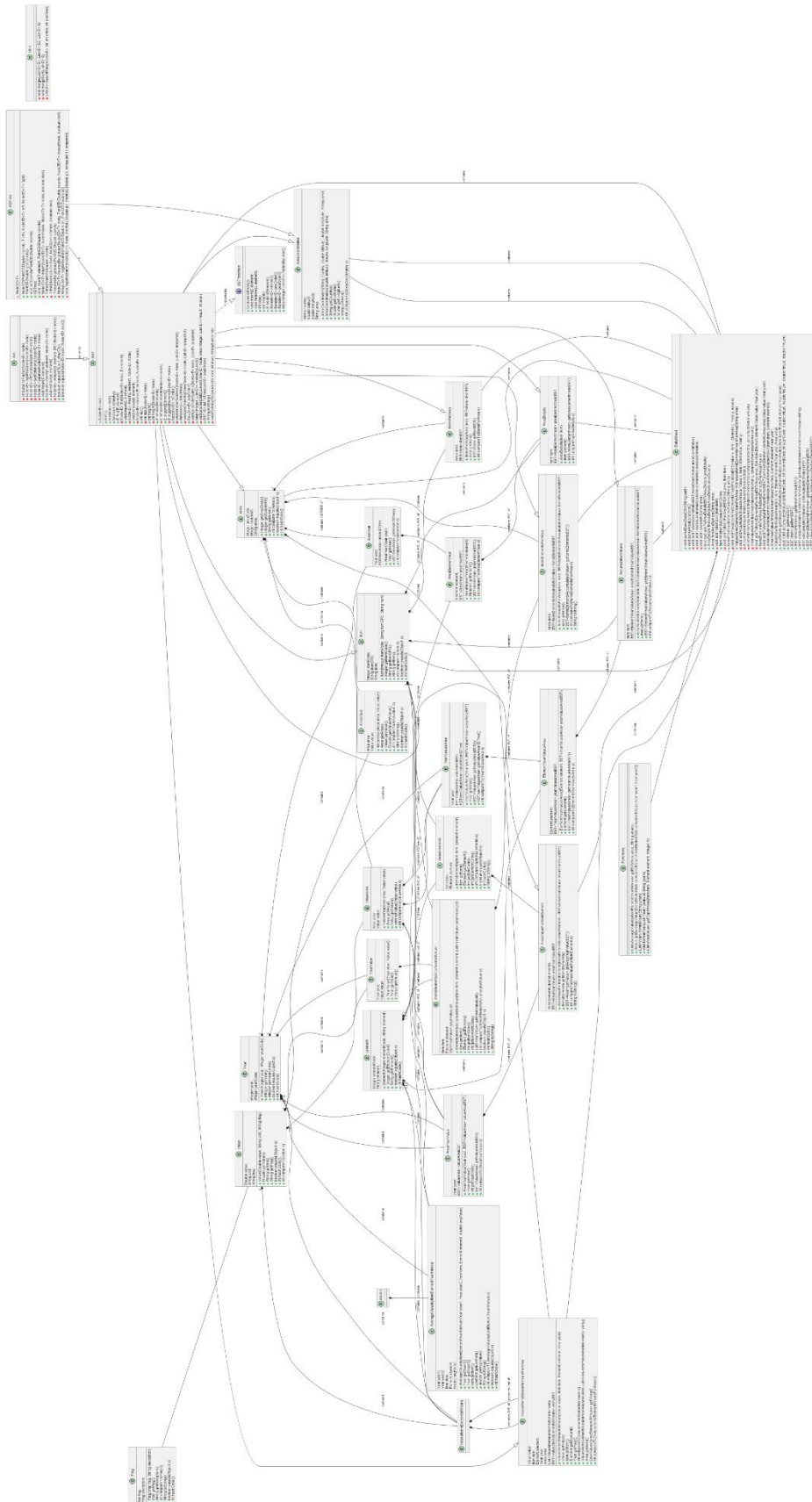
Ao longo deste relatório será apresentada o problema a em questão, a nossa visão acerca dele e a respetiva solução encontrada, assim como passos e decisões que nos levaram a essa decisão. Para além disso apresentamos a um diagrama de classes, a análise de complexidade de todas as funcionalidades implementadas e melhoramentos possíveis.

## Problema

Usando a classe árvore binária de pesquisa (BST) apresentada nas aulas, desenvolva da forma mais eficiente possível as seguintes funcionalidades:

1. Com recurso à classe árvore binária de pesquisa (BST), carregar a informação relativa aos dados da FAOSTAT apenas em árvores binárias de pesquisa (BST) que permitam obter os valores de produção {Value, Unit, Flag, Flag Description}. A pesquisa deverá ser efetuada não só através dos campos {Area Code, Item Code, Element Code, Year}, mas também e tirando partido da hierarquia de classes, permitir a pesquisa por outras combinações, p.ex.: {Area, Item, Element, Year}.
2. Para uma determinada Area, passada por parâmetro, devolver numa estrutura de dados, a média dos Value agregados por Item e Element num intervalo de anos passado por parâmetro, ex.: [1996, 2005] ordenado por ordem decrescente de valor.
3. Para um determinado Item e Element, obter as top-N Areas com maior valor no último ano registado no conjunto de dados para aquele Element. Nota: último ano entende-se o último ano em que foi registado valor no ficheiro de dados em análise.
4. Com recurso à 2d-tree devolva todos os detalhes da Area geograficamente mais próxima das coordenadas: latitude, longitude passadas por parâmetro. Deverá considerar apenas Areas com valores registados para um dado {Item, Element, Year}. Exemplo: Qual é a Area geograficamente mais próxima das coordenadas latitude: 41.14961, longitude: -8.61099 com produção de figos em 2018? {latitude: 41.14961, longitude: -8.61099, Item: Figs, Element: Production, Year: 2018}
5. Com recurso à 2d-tree devolva para um Item Code, Element Code e Year Code o acumulado dos valores de produção para uma área geográfica retangular dada por uma latitude inicial, latitude final, longitude inicial e longitude final

## Diagrama de classes



# Solução

## Exercicio1

### Métodos para exercício 2

O método `addToItemsElementsPerArea` vai adicionar a BST `itemsElementsPerAreaBST`, caso esta ainda seja nula vai criar, de seguida vai procurar um elemento `ItemsElementsPerArea` através da área, caso este seja null vai ser criado um objeto do tipo `ItemsElementsPerArea` e depois será inserido na arvore `itemsElementsPerAreaBST`.

Depois o algoritmo vai verificar se a BST contida em `itemsElementsPerArea` (BST do tipo `ItemsElementsAccumulatedValue`) é diferente de null, caso não seja null vai verificar se na arvore já existe algum objeto com o item e elemento passados como parâmetro, caso não exista será inserido estes valores na arvore, depois será criado um objeto do tipo `ItemsElementsAccumulatedValue` que será obtido dos dados inseridos na linha anterior e posteriormente vai ser guardado o Value referente ao ano numa lista contida no objeto da classe `ItemsElementsAccumulatedValue` contido na BST do `itemsElementsPerArea`. Caso o objeto do tipo `ItemsElementsAccumulatedValue` já estiver na arvore, só será acrescentado o valor do value referente aquele ano na lista contida em `itemsElementsAccumulatedValue`.

```
private void addToItemsElementsPerAreaBST(Area area, Item item, Element element, Value value, Year year) {
    if (itemsElementsPerAreaBST == null) itemsElementsPerAreaBST = new AVL<>();

    ItemsElementsPerArea itemsElementsPerArea = findItemsElementsPerArea(area);

    if (itemsElementsPerArea == null) {
        itemsElementsPerArea = new ItemsElementsPerArea(area, new BST<>());
        itemsElementsPerAreaBST.insert(itemsElementsPerArea);
    }

    if (itemsElementsPerArea.getItemsElementsBST() != null) {
        if (itemsElementsPerArea.getItemsElementsBST().find(new ItemsElementsAccumulatedValue(item, element, yearValueList: null)) == null) {
            itemsElementsPerArea.getItemsElementsBST().insert(new ItemsElementsAccumulatedValue(item, element, new ArrayList<>()));
            ItemsElementsAccumulatedValue itemsElementsAccumulatedValue = itemsElementsPerArea.getItemsElementsBST().find(new ItemsElementsAccumulatedValue(item, element, yearValueList: null));
            itemsElementsAccumulatedValue.getYearValueList().add(new YearValue(year, value));
        } else {
            ItemsElementsAccumulatedValue itemsElementsAccumulatedValue = itemsElementsPerArea.getItemsElementsBST().find(new ItemsElementsAccumulatedValue(item, element, yearValueList: null));
            itemsElementsAccumulatedValue.getYearValueList().add(new YearValue(year, value));
        }
    }
}

public ItemsElementsPerArea findItemsElementsPerArea(Area a) {
    return itemsElementsPerAreaBST.find(new ItemsElementsPerArea(a, itemsElementsPerAreaBST));
}
```

### Métodos para exercício 3

O método `addToAreaValuePerItemElementBST` irá receber um Item, Element, Area, Value e Year que terá de adicionar à `areaValuePerItemElementBST` e todas as outras árvores necessárias ao exercício. Primeiramente, se não existir uma BST criada irá criar uma nova. De seguida, vai verificar se já existe algum Item e Element guardados na árvore, se não existir irá inserir na `areaValuePerItemElementBST` o Item e o Element e, associados a estes, inserirá na `areaYearValueBST` o Year e, associado a este, irá inserir na `ValueAreaBST` a Area e Value. Caso exista vai verificar se já existe um Year guardado associado ao Item e Element. Se não existir irá guardar e criar uma nova BST associada ao Year que guarda a Area e o Value. Se já existir um Year, irá verificar se já existe uma Area e Value associadas ao Year. Se não existir irá criar uma nova `valueAreaBST` e guardará a Area e o Value. Caso já exista, irá por fim verificar se para uma determinada Area, o novo Value é maior do que o guardado. Se for irá substituir o guardado pelo novo.

```
private void addToAreaValuePerItemElementBST(Item item, Element element, Area area, Value value, Year year) {
    if (areaValuePerItemElementBST == null) areaValuePerItemElementBST = new AVL<>();

    //Item Element -> BST<Year,BST<AREA,VALUE>
    AreaValuePerItemElement areaValuePerItemElement = findAreaValuePerItemElementBST(item,element);

    //if 1st bst in null
    if(areaValuePerItemElement == null){
        // mudei de new BST<> para AVL<> - ricardo
        areaValuePerItemElement = new AreaValuePerItemElement(new ItemsElements(item,element),new AVL<>());
        areaValuePerItemElementBST.insert(areaValuePerItemElement);

        AreaYearValue areaYearValue = new AreaYearValue(year,new AVL<>());
        areaValuePerItemElement.getAreaYearValueBST().insert(areaYearValue);
        areaValuePerItemElement.getAreaYearValueBST().find(areaYearValue).getValueAreaBST().insert(new ValueArea(area,value));
    }else {
        // 2nd bst is null
        AreaYearValue areaYearValue = findAreaYearValue(item,element,year);
        if(areaYearValue==null){
            areaYearValue = new AreaYearValue(year,new BST<>());
            areaValuePerItemElement.getAreaYearValueBST().insert(areaYearValue);
            areaValuePerItemElement.getAreaYearValueBST().find(areaYearValue).getValueAreaBST().insert(new ValueArea(area,value));
        }
    }
}
```

```
    }else {
        //3rd bst is null
        if(findValueArea(item,element,year,area)==null){
            areaValuePerItemElement.getAreaYearValueBST().find(areaYearValue).getValueAreaBST().insert(new ValueArea(area,value));
        }else {
            ValueArea valueArea = areaValuePerItemElement.getAreaYearValueBST().find(areaYearValue).getValueAreaBST().find(new ValueArea(area,value));
            if (valueArea.getValue().getValue()<value.getValue()){
                valueArea.setValue(value);
            }
        }
    }
}
}
```

O método findAreaValuePerItemElementBST irá verificar se o determinado Item e Element existem na areaValuePerItemElementBST. Se existir, irá retornar o objeto da classe AreaValuePerItemElement cujos Item e Element são iguais aos passados por parâmetro. Se não existir, retornará nulo. Este método tem complexidade  $O(\log(n))$ .

```
public AreaValuePerItemElement findAreaValuePerItemElementBST(Item item, Element element){
    return areaValuePerItemElementBST.find(new AreaValuePerItemElement(new ItemsElements(item,element), areaYearValueBST: null));
}
```

O método findAreaYearValue irá verificar se o determinado Year existem na areaYearValueBST. Se existir, irá retornar o objeto da classe AreaYearValue cujos Item, Element e Year são iguais aos passados por parâmetro. Se não existir, retornará nulo. Este método tem complexidade  $O(\log(n))$ .

```
public AreaYearValue findAreaYearValue(Item item, Element element,Year year){
    return areaValuePerItemElementBST.find(new AreaValuePerItemElement(new ItemsElements(item,element),
        areaYearValueBST: null)).getAreaYearValueBST().find(new AreaYearValue(year, valueAreaBST: null));
}
```

O método findValueArea irá verificar se o determinado Area existem na valueAreaBST. Se existir, irá retornar o objeto da classe ValueArea cujos Item, Element, Year e Area são iguais aos passados por parâmetro. Se não existir, retornará nulo. Este método tem complexidade  $O(\log(n))$ .

```
public ValueArea findValueArea(Item item, Element element,Year year,Area area){
    return areaValuePerItemElementBST.find(new AreaValuePerItemElement(new ItemsElements(item,element),
        areaYearValueBST: null)).getAreaYearValueBST().find(new AreaYearValue(year, valueAreaBST: null)).getValueAreaBST().find(new ValueArea(area, value: null));
}
```

#### Métodos para exercício 4

O método `addToAreaDetails` vai receber como parâmetro uma `Area`, um `Item`, um `Element` e um `Year` e vai adicionar os valores ao `areaDetailsBST`. Primeiro passo é verificar se esta BST existe senão vai criar a BST. De seguida procura na BST se uma `AreaDetails` para a `Area` passada por parâmetro existe, se não existir cria uma nova e adiciona o valor de `AreaDetails` à BST, se existir procuramos pela BST de `AreaElementsYear` que faz parte de `AreaDetails` se não existir cria uma nova `AreaElementsYear` e de seguida adiciona o valor à BST. O próximo passo é verificar se dentro do `AreaElementsYear` existe uma BST de `AreaYear` se não existir segue o mesmo método dos anteriores onde cria uma nova BST e adiciona o novo valor de `AreaYear`.

O próximo passo é verificar se o `AreaCoordinates` existe na `areaLocationBST` se não existir criamos uma nova e adicionamos.

```
public void addToAreaDetailsBST(Area a, Item i, Element e, Year y){

    if(areaDetailsBST==null)
        areaDetailsBST=new AVL<>();

    AreaDetails ad = areaDetailsBST.find(new AreaDetails(i));
    if(ad==null){
        ad= new AreaDetails(i);
        areaDetailsBST.insert(ad);
    }

    AreaElementYear aey= ad.getAreaElementYearBST().find(new AreaElementYear(e));
    if(aey==null){
        aey=new AreaElementYear(e);
        ad.getAreaElementYearBST().insert(aey);
    }

    AreaYear ay= aey.getAreaYearBST().find(new AreaYear(y));
    if(ay==null){
        ay= new AreaYear(y);
        aey.getAreaYearBST().insert(ay);
    }

    AreaCoordinates ac= areaLocationsBST.find(new AreaCoordinates( country: "", latitude: 0, longitude: 0,a.getArea()));
    if(ac!=null){
        ay.getAreaKDTree().insert(a,new Point2D.Double(ac.latitude,ac.longitude));
    }

}
```



## Métodos para exercício 5

O método *addAccumulatedValues* vai adicionar à **BST** de *Accumulated Values* todos os valores através de combinações de *item codes*, *element codes* e *year codes*, vai ainda armazenar numa **2DTree** os valores e as áreas das respetivas combinações.

É verificado através de um *find* se já existe algum *Item*, *Element* ou *Year* (mediante o caso) e caso exista, simplesmente adiciona na **BST** de cada classe. Na ocasião de não ser encontrado algum *Item*, *Element* ou *Year*, vai ser criada a nova combinação.

```
private void addAccumulatedValues(Area a, Value v, Item i, Element e, Year y){

    if(accumulatedValuesBST == null) accumulatedValuesBST = new AVL<>();

    AccumulatedValues val = findAccumulatedValues(i);
    if(val == null){
        val = new AccumulatedValues(i, new AVL<>());
        accumulatedValuesBST.insert(val);
    }

    ElementYearValuesArea val2 = findElementYearValuesArea(e, val);
    if(val2 == null){
        val2 = new ElementYearValuesArea(e, new AVL<>());
        val.getElementYearValuesAreaBST().insert(val2);
    }

    YearValuesArea val3 = findYearValuesArea(y, val2);
    if(val3 == null){
        val3 = new YearValuesArea(y, new AVL<>());
        val2.getYearValuesAreaBST().insert(val3);
    }

    AreaCoordinates ac = areaLocationsBST.find(new AreaCoordinates( country: "", latitude: 0, longitude: 0, a.getArea()));
    if(ac == null)
        return;
    Point2D.Double coords = new Point2D.Double(ac.getLatitude(), ac.getLongitude());
    val3.getValueAreaKDTree().insert(new ValueArea(a, v), coords);
}
```

## Exercicio2

O método `getEx2` retorna uma lista contendo a media dos values agregados por Item e Element, o método recebe como parâmetro uma área e um intervalo de anos (no seguinte formato: “[1990,2010]”). Primeiro é criada uma lista `ex2List` a qual posterior mente será preenchida com os valores a devolver, depois o algoritmo vai converter o intervalo de anos para dois objetos do tipo `Year` os quais irão permitir uma melhor manipulação nos dados (O algoritmo usa tem complexidade  $O(n)$ ). Depois o algoritmo vai procurar um objeto do tipo `ItemsElementsPerArea` através do `findItemsElementsPerArea` (este método tem complexidade de  $O(\log(n))$  pois é utilizada uma AVL que garante este nível de complexidade), de seguida o programa vai criar um iterable com a posorder da AVL contida no `itemsElementsPerArea` (este método tem complexidade de  $O(n)$ ). Uma vez obtida este iterable vamos percorrê-lo e calcular o average através do método `getAverage` (complexidade  $O(n)$ ) e de seguida vai acrescentar este valor junto com a área, elemento e item a lista `ex2List`. Finalmente esta lista vai ser organizada em ordem decrescente usando o merge-sort (o merge-sort tem uma complexidade de  $O(n\log(n))$ ). Assim a complexidade deste método é de  $O(n)$ .

```
public List<AverageValuebyItemElementYearInterval> getEx2(Area area, String years){
    if (area == null || years == null || years.isEmpty()) return null;
    List<AverageValuebyItemElementYearInterval> ex2List = new ArrayList<>();
    List<Year> yearList = formatYear(years);
    Year year1 = yearList.get(0);
    Year year2 = yearList.get(1);
    ItemsElementsPerArea itemsElementsPerArea = d1.findItemsElementsPerArea(area);
    Iterable<ItemsElementsAccumulatedValue> itemsElementsPerAreaList = itemsElementsPerArea.getItemsElementsBST().posOrder();

    for (ItemsElementsAccumulatedValue c: itemsElementsPerAreaList){
        double average = getAverageValue(c, year1, year2);
        ex2List.add(new AverageValuebyItemElementYearInterval(year1, year2, c.getItem(), c.getElement(), average));
    }

    Utils.mergeSort(ex2List);
    return ex2List;
}
```

O método `getAverageValue` retorna um double com o valor medio de produção de um Item e Element num intervalo de anos, caso o `itemsElementsAccumulatedValue` ou a lista que esta contida neste objeto for nula o algoritmo retorna 0.

Primeiro é criada uma lista `yearValueList` do tipo `YearValue` com os valores de produção em cada ano, vamos iterar sobre esta lista toda e se o ano estiver dentro do intervalo especificado vamos somar o seu valor de produção a variável `average` posteriormente criada e vamos aumentar em um o `occ` (quantidade de valores somados), uma vez percorrida a lista toda vamos verificar se o numero de elementos divididos é igual a zero, neste caso o algoritmo retorna 0, caso contrario o algoritmo retorna o valor da divisão do `average` pelo `occ`. Este método tem uma complexidade  $O(n)$  devido ao ciclo implementado para o cálculo do valor medio.

```

public double getAverageValue(ItemsElementsAccumulatedValue itemsElementsAccumulatedValue, Year year1, Year year2){
    if( itemsElementsAccumulatedValue==null || itemsElementsAccumulatedValue.getYearValueList()==null){
        return 0;
    }
    List<YearValue> yearValueList = itemsElementsAccumulatedValue.getYearValueList();

    double average = 0.0;
    int occ = 0;

    for (YearValue c: yearValueList){

        if (year1.getYear().getYear() <= c.getYear().getYear() && c.getYear().getYear() <= year2.getYear()){
            average=average+c.getValue().getValue();
            occ++;
        }
    }

    if(occ==0){
        return 0;
    }else {
        return average/occ;
    }
}

```

O método `formatYear` retorna uma lista com o ano inicial e final (passados como parâmetro no método `getEx2`). Primeiro cria um array o qual será preenchido com o método `split` da classe `String` ficando no index 0 o primeiro ano e o index 1 o segundo ano. Depois da separação dos anos são criados dois objetos do tipo `Year` com cada valor dos anos e de seguida serão acrescentados a lista de retorno. Este método tem uma complexidade de  $O(1)$ .

```

public List<Year> formatYear(String years){
    List<Year> yearList = new ArrayList<>();
    formatYear(yearList, years);
    return yearList;
}

private void formatYear(List<Year> yearList, String years){
    String[] a = years.split( regex: "[ ]");
    a[0]=a[0].substring( beginIndex: 1);
    a[1]=a[1].substring(0,4);
    Year yearInitial = new Year(Integer.parseInt(a[0]),Integer.parseInt(a[0]));
    Year yearEnd = new Year(Integer.parseInt(a[1]),Integer.parseInt(a[1]));
    yearList.add(yearInitial);
    yearList.add(yearEnd);
}

```

### Exercicio3

Segundo um determinado Item e Element, o método `getTopNAreas`, vai retornar uma lista com as top-N Areas com maior Value no último ano registado no conjunto de dados para aquele Element. Primeiramente irá fazer a verificação dos parâmetros que, se algum deles for nulo, irá dar retorno nulo, pois não é possível prosseguir sem algum dos parâmetros. De seguida, cria uma lista que irá guardar objetos do tipo `AreaValue` (uma certa Area e o respetivo Value num certo ano) que no final do método, irá conter os resultados pretendidos. Depois é feita a chamada do método `findAreaValuePerItemElement`, onde irá ser feita a procura do determinado Item e Element na `areaValuePerItemElementBST`. O resultado da procura é guardado num objeto da classe `AreaValuePerItemElement` e se for nulo, o método retornará nulo, pois significa que não existe registo do Item e Element. Se não for nulo, invocaremos o método `getAreaYearValueBST` (esta BST terá todas as instâncias de `AreaYearValue` associados a um certo Item e Element) e logo de seguida o `biggestElement` para obtermos instância da classe `AreaYearValue` associada ao determinado Item e Element que tem o último ano em registo. De seguida é realizada uma nova procura na

areaYearValueBST através do ano que foi encontrado previamente e será retornado objeto que tem as Areas e os Values do último ano em registo. Com este objeto, iremos fazer getValueAreaBST para termos então as instâncias da classe ValueArea e iremos organizá-las em pós-ordem num Iterable areaValueIterable. Finalmente irá entrar no ciclo for que adicionará as N instâncias com maior Value à lista final areaValuesList que irá ser retornada.

Por fim, este método tem complexidade  $O(n)$  uma vez que os métodos find tem complexidade  $O(\log(n))$ , o biggestElement tem  $O(n)$  e o ciclo for  $O(n)$ .

```
public List<AreaValue> getTopNAreas(Item item, Element element, Integer n){
    if (item == null || element == null || n == null || n <= 0) return null;

    List<AreaValue> areaValuesList = new ArrayList<>();
    AreaValuePerItemElement areaValuePerItemElement = d1.findAreaValuePerItemElementBST(item, element);
    if (areaValuePerItemElement == null) return null;
    //sort
    AreaYearValue areaYearValue1 = areaValuePerItemElement.getAreaYearValueBST().biggestElement();

    AreaYearValue areaYearValue2 = areaValuePerItemElement.getAreaYearValueBST().find(new AreaYearValue(areaYearValue1.getYear(), valueAreaBST: null));
    Iterable<ValueArea> areaValueIterable = areaYearValue2.getValueAreaBST().posOrder();

    for (ValueArea c: areaValueIterable){
        if(n!=0) {
            areaValuesList.add(new AreaValue(c.getArea(), c.getValue()));
            n--;
        }
    }

    //get top n
    return areaValuesList;
}
```

## Exercicio4

Para o exercício 4 foi criada uma classe “AreaDetails” que pretende capturar as regras de negócio do que era pretendido procurar no método que resolve o problema.

Outras classes criadas foram “AreaElementYear” e “AreaYear” .

```
public class AreaDetails implements Comparable<AreaDetails>{
    private Item item;
    private BST<AreaElementYear> areaElementYearBST;

    public AreaDetails(Item item) {
        this.item = item;
        areaElementYearBST= new AVL<>();
    }

    public Item getItem() {
        return item;
    }

    public BST<AreaElementYear> getAreaElementYearBST() {
        return areaElementYearBST;
    }

    @Override
    public int compareTo(AreaDetails o) {
        return item.getItemCode().compareTo(o.getItem().getItemCode());
    }
}
```

Foi criada uma 2d-Tree "areaLocation2DTree “– utilizada também no exercício 5 – que armazena as Coordenadas de cada Área.

Foi Criada também uma BST que armazena objetos da classe “AreaDetails”, inicializada quando o documento é lido pela aplicação.

O método que soluciona o exercício é getCloseAreaDetails que recebe os recebe os parâmetros {double lat, double lon, int itemCode, int elementCode, int yearCode }, o seu objetivo é calcular a área mais próxima das coordenadas passadas como parâmetro que que contem os dados sobre o item e o elemento passados no respetivo ano.

O método começa por pesquisar o Item correspondente ao ItemCode passado por parametro, e faz uma filtragem dos outros elementos até atingir uma instancia da classe AreaYear que armazena uma 2Dtree com todas as coordenadas das Areas que tem os valores passados por parametros. Por fim, apenas chama a função getNearestNeighbour passando como parametros as coordenadas recebidas anteriormente, o retorno da função é a Area mais proxima das coordenadas passadas que contem o item, elemento e ano pretendido.

```
public Area getCloseAreaDetails(double lat,double lon ,int itemCode ,int elementCode,int yearCode){  
  
    AreaDetails ad = areaDetailsBST.find(new AreaDetails(new Item(itemCode, itemCPC: "", item: "")));  
    if(ad== null)  
        return null;  
    AreaElementYear aey = ad.getAreaElementYearBST().find(new AreaElementYear(new Element(elementCode, element: "")));  
    if(aey==null)  
        return null;  
    AreaYear ay = aey.getAreaYearBST().find(new AreaYear(new Year( year: 0,yearCode)));  
    if(ay==null)  
        return null;  
  
    return ay.getAreaKdTree().nearestNeighbour(new Point2D.Double(lat,lon));  
}
```

## Exercicio5

Para o exercício cinco, foi utilizada uma **BST** que guarda *Accumulated Values*. Objeto esse que tem como atributos *Item* e **BST** de *Element Year Values Area*. Utilizando a mesma lógica explicada acima, *Element Year Values Area* vai ter como atributos um *Element* e uma **BST** de *Year Values Area*.

Repete-se até *Value Area* que é um objeto que tem como campos *Value* e *Area*.

A funcionalidade faz pesquisas às árvores, mediante os valores que receber por argumento. Tendo em conta que as árvores implementadas são **AVLs**, as pesquisas são, definitivamente, de complexidade  $O(\log(n))$ .

A funcionalidade começa por pesquisar na **BST** que armazena *Accumulated Values* pelo *item code* recebido, na ocasião de existir, prossegue com o algoritmo, caso contrário vai retornar zero. As pesquisas às **BSTs** vão continuar até que todos os códigos (*item*, *element* e *year code*) se verifiquem presentes.

Ao entrar no *loop*, a **BST** de *Value Area* vai ser *iterada*. Em cada iteração é verificado se o país em questão está dentro da área geográfica restringida pelos valores recebidos por argumento, ou seja, faz uma pesquisa por região à **2DTree** e vai iterar somar apenas os valores elegíveis.

### Complexidade do Algoritmo

Todas as pesquisas efetuadas, dado que são executadas em **AVLs**, são de complexidade  $O(\log(n))$ . Apenas a *iteração da lista* de países que se encontra dentro da área filtrada é de complexidade  $O(n)$ . Logo, o método tem uma complexidade de  $O(n)$ .

```
public int accumulatedValues(int itemCode, int elementCode, int yearCode, double iniLat,
                             double iniLon, double finLat, double finLon){
    AccumulatedValues av = d1.getAccumulatedValuesBST().find(
        new AccumulatedValues(new Item(itemCode, itemCPC: "", item: ""),
                               itemsElementYearValuesBST: null));

    if(av == null)
        return 0;

    ElementYearValuesArea eyv = av.getElementYearValuesAreaBST().find(
        new ElementYearValuesArea(new Element(elementCode, element: ""),
                                    yearValuesAreaBST: null));

    if(eyv == null)
        return 0;

    YearValuesArea yv = eyv.getYearValuesAreaBST().find(
        new YearValuesArea(new Year( year: 0, yearCode),
                            valueAreaBST: null));

    if(yv == null)
        return 0;

    Point2D.Double p1, p2;
    p1 = new Point2D.Double(iniLat, iniLon);
    p2 = new Point2D.Double(finLat, finLon);

    List<ValueArea> list = yv.getValueAreaKdTree().regionSearch(p1, p2);

    int sum=0;
    for(ValueArea va : list)
        sum += va.getValue().getValue();

    return sum;
}
```

## Melhoramentos possíveis

- Melhorar organização de código;
- Reutilizar mais código.