

1º Projecto Prático

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Algoritmos e Estruturas de Dados**

2016/2017

João Afonso Póvoa Marques – XXXXXXXXXX

José Miguel Saraiva Monteiro – 2015235576

Leonardo Machado Alves Vieira – 2015236155

**Índice**

[**Índice** 2](#_Toc478235854)

[**1 – Introdução** 2](#_Toc478235855)

[**2 – Estruturas** 3](#_Toc478235856)

[Estrutura 1: 3](#_Toc478235857)

[Estrutura 2: 4](#_Toc478235858)

[Estrutura 3: 4](#_Toc478235859)

[**3 - Operações Implementadas** 5](#_Toc478235860)

[Estrutura 1: 5](#_Toc478235861)

[Estrutura 2: 5](#_Toc478235862)

[Estrutura 3: 6](#_Toc478235863)

[**4 – Casos de Teste** 6](#_Toc478235864)

[Estrutura 1: 6](#_Toc478235865)

[Estrutura 2: 6](#_Toc478235866)

[Estrutura 3: 6](#_Toc478235867)

[**5 – Comparações** 6](#_Toc478235868)

[**6 – Conclusões** 6](#_Toc478235869)

# **1 – Introdução**

Neste projeto foi nos proposta a tarefa de criar 3 estruturas de dados suportadas em abstrações diferentes que possam armazenar e manipular a informação disponibilizada. No contexto deste projeto, informação era fornecida através do ficheiro dados*.csv* sobre a evolução do acesso a redes elétricas pela população mundial desde 1960 até 2016.

Sobre todas estas estruturas foram definidas e implementadas funções de pesquisa, inserção, edição e remoção para ambos anos e países, que foram adaptadas e melhoradas de acordo com a estrutura e que podem ser facilmente utilizadas através de um interface funcional (igual para todas as estruturas) que é apresentado ao utilizador ao iniciar o programa.

Também foram criadas em todas as estruturas, com uma vertente de teste, funções auxiliares para realização de *benchmarks* em que são feitos testes extremos e contabilizado o tempo de resposta para cada um.

Todo o projecto foi realizado em python3.

As estruturas implementadas foram as seguintes:

* Listas Duplamente Ligadas de Listas Duplamente Ligadas
* Stacks de Stacks
* Árvores AVL de Árvores AVL

# **2 – Estruturas**

## Estrutura 1:

Stacks de Stacks

[Implementação base retirada das estruturas apresentadas nas aulas de AED]

## Estrutura 2:

Listas Duplamente Ligadas de Listas Duplamente Ligadas

[Implementação base retirada das estruturas apresentadas nas aulas de AED]

Uma das estruturas implementadas foi uma Lista Duplamente Ligada de Listas Duplamente Ligadas. Esta estrutura foi escolhida por diversas razões, sendo uma delas a relativa facilidade de implementação e o facto de já termos tido contacto com este tipo de estruturas previamente. Para além disso foi também tido em conta a complexidade *BigO* das várias funções a implementar sobre esta estrutura. Assim sendo, o facto de esta ter uma complexidade de Θ(1) para as operações (isoladas) de inserção e remoção foi bastante apelativo. Isso faz com que as operações referidas sejam uma vantagem desta estrutura.

No entanto, como no contexto do problema não faz sentido haver duplicação de dados e faz sentido alguns valores estarem ordenados algumas desvantagens são criadas. Isto é, para cada operação de adição é necessário uma primeira pesquisa para garantir que o valor a adicionar não se encontra previamente na lista. Se o valor estiver de alguma forma ordenado, será escusado a pesquisa completa da lista, no entanto com valores desordenados terá sempre que iterar por todos os nós da lista à procura de um com o valor igual. Com isto, facilmente se entende que uma das desvantagens desta estrutura será a pesquisa, pois está subentendida em todas as outras operações e tem uma complexidade de Θ(n) (sendo o *n* minimizado quando operando sobre valores ordenados).

Poderiam ter sido implementadas estruturas auxiliares para melhorar tempos e complexidades de certas operações. Por exemplo, adicionar uma estrutura auxiliar para cada ano em que se iria guardar o par país, percentagem desse país nesse ano. Assim, seria possível obter a informação de todos os países para um ano de uma maneira muito mais “económica” e sem piorar a complexidade espacial pois toda a informação aparentemente duplicada iria apontar para o mesmo sitio da memória.

## Estrutura 3:

Árvore AVL de Árvores AVL

[Implementação base retirada das estruturas apresentadas nas aulas de AED]

A terceira estrutura a ser implementada foi uma árvore AVL de árvores AVL. A escolha desta estrutura teve como base a ideia de que seria uma das melhores implementações para os dados em questão. Esta estrutura possibilita-nos operações (isoladas) de inserção, remoção e pesquisa com uma complexidade Θ(log n). Além disto todas as adições/remoções não fazem com que a arvore perca o seu ordenamento, pois esta esta sempre organizada (por ordem alfabética e crescente de anos no nosso caso especifico) e sempre balanceada, o que facilita operações de pesquisa. Estas características de facilidade de acesso e baixa complexidade fazem com que esta estrutura tenha vantagens, num aspeto mais geral, às apresentadas anteriormente.

Como auxiliar a esta estrutura, para evitar ter de criar uma nova árvore organizada pela *tags* e para evitar ter de percorrer toda a árvore para encontrar uma *tag* especifica, foi criado ainda um dicionário com os nomes de todos os países (*key*) e a *tag*  correspondente (*value*). Como o acesso a este tipo de estruturas é de Θ(1) não irá afetar a complexidade da pesquisa.

# **3 - Operações Implementadas**

## Estrutura 1:

Stacks de Stacks

* Pesquisa:
* Inserção:
* Edição:
* Remoção:

## Estrutura 2:

Listas Duplamente Ligadas de Listas Duplamente Ligadas

* Pesquisa:

Como referido anteriormente a pesquisa é um dos

pontos fracos desta estrutura.

Num caso base, a pesquisa teria sempre uma complexidade de Θ(n), pois teria de correr sempre toda a lista à procura do nó desejado. No entanto, na lista de anos, os anos encontram-se ordenados, o que faz com que para esta lista, o *n* seja reduzido. Para este caso temos num limite inferior uma complexidade de pesquisa de Θ(1) e num limite superior a complexidade base de Θ(n).

Foi implementada uma função de pesquisa na lista dos anos, que permite a pesquisa de anos dado um limite inferior e superior de percentagens. Esta função tem uma complexidade de Θ(n) pois esta vai correr toda a lista imprimindo apenas os anos cujo valor esteja dentro dos limites.

Para além disso, foi também implementado uma pesquisa *inversa* à primeira referida. Isto é, o utilizador dá *input* de um ano, e é lhe apresentado o valor de todos os países para esse ano. Esta função é obviamente má a nivel de complexidade, pois como tem de correr sempre a lista de países e dentro de cada nó, ainda correr a lista de anos, terá uma complexidade de Θ(n2) num limite superior e O(n) num limite inferior. Note-se que isto poderia ser melhorada

com uma estrutura auxiliar com as percentragens para todos os paises num certo ano. Isto seria no fundo uma estrutra *“espelho”* da que foi de facto implementada.

Dentro da lista de países foi apenas implementada uma pesquisa simples através de uma função Θ(n) que corre a lista toda à procura do nó correto que depois é retornado para ser usado noutras funções.

* Inserção:

Também já foi referida a vantagem da inserção neste tipo de estruturas.

Foram implementadas dois tipos de inserções. Para a inserção na lista (desordenada) de países, recorre-se primeiramente a uma pesquisa pelo nome e *tag* do país a inserir, para garantir que o país não está previamente na lista evitando assim duplicação de informação. Depois desta pesquisa, e em caso negativo de duplicação, procede-se à inserção do novo nó no final da list. Esta parte da operação é realizada com uma complexidade de Θ(1), visto que a pesquisa inicial retorna o ultimo nó da lista, de maneira a evitar precorrê-la novamente.

Será importante notar que quando é adicionado um novo país, a lista de anos encontra-se a *None.*

A outra implementação foi feita sobre a lista (ordenada) de anos de um país. À semelhança da anterior, também é necessário uma pesquisa prévia (que também retorna um nó) para garantir que não existe duplicação de informação. Porém, esta é uma inserção ordenada, o que implica que o nó retornado nem sempre é o ultimo, mas sim, caso exista, o primeiro com valor maior do que o nó a adicionar. No melhor caso esta inserção é ótima, com uma complexidade de Θ(1). Nos restantes dos casos será de uma complexidade de Θ(n), aumentado o *n* até ao pior caso, que é ter de percorrer toda a lista.

* Edição:

Existem sobre esta estrutura duas possibilidades de edição. Edição de um ano de um país, e de uma percentagem de um ano de um país.

Para a edição da percentagem, simplesmente faz-se uma pesquisa pelo país e pelo ano (2\*Θ(n)), e caso estes existam é feito uma mudança do valor da variável respetiva, sendo isto uma operação Θ(1).

A edição do ano em si já é mais complexa pois não se pode desprezar a ordem da lista de anos e também porque não se pode criar duplicação de anos. Assim sendo, para evitarmos isto tudo, a operação realiza-se nos seguintes paços. Primeiro procurar o país e garantir que ano a alterar existe e que o novo ano para o qual vai ser alterado não existe (3\*Θ(n)). Depois disto, caso “passe” o primeiro passo, iremos guardar numa variável auxiliar a percentagem do ano e remover esse ano. Finalmente é adicionado um novo nó com a percentagem antiga e com o novo ano. Isto é feito para garantir que o nó fica ordenado pois a função de inserção, como foi dito anteriormente, certifica-se disso. Assim conclui-se que esta operação tem uma complexidade de Θ(n) e é bastante dispendiosa.

* Remoção:

A implementação da operação de remoção foi bastante semelhante para a lista de anos e para a lista de países.

Tal como todas as outras, esta operação tambem requer uma pesquisa prévia para encontrar e garantir que existe o ano / país. Logo aqui limitamos a complexidade a (Θ(n)). De seguida, de acordo com a posição do nó pedido, são executadas uma serie de opreções Θ(1) para garantir que todas as ligações *next* e *prev* são bem feitas. Feito isto todas as variáveis do nó de um ano / país são colocadas a *None*.

## Estrutura 3:

Árvore AVL de Árvores AVL

* Pesquisa:

A pesquisa numa AVL pode ser feita tal como seria numa árvore binária não balanceada. No nosso caso em especifico pesquisa pelo nome de um país, a complexidade desta pesquisa seria Θ(log n).

Em adição à pesquisa pelo nome de um país foi ainda adicionada um método de pesquisa por anos. Este método na AVL de anos continuaria a ter complexidade Θ(log n), mas como para a pesquisa em todos em países por esse ano teríamos de percorrer toda a árvore, operação de complexidade Θ(n), a complexidade total da pesquisa de um ano em todos os países será Θ(n\*log n). Esta acaba por ser a operação de maior complexidade da estrutura da forma como está implementada e usada.

Como referido a cima as AVLs de anos tem apenas implementada uma pesquisa por anos de complexidade Θ(log n)

Outra opção de pesquisa implementada é a pesquisa, dentro de um país, de valores entre um certo limite. Esta opção foi implementada de forma semelhante à pesquisa de um ano em todos os países. Será então feita uma pesquisa pelo país desejado e após isso terá de se percorrer toda a árvore de anos. Tal como a pesquisa de um ano em todos os países a complexidade deste método será também Θ(n\*log n).

* Inserção:

O método de inserção utilizado é uma adaptação do código fornecido pelo professor nas aulas adaptado aos novos nós que serão inseridos na árvore. Como tal mantemos a vantagem de todas as inserções serem automaticamente ordenadas, devido a natureza balanceada da AVL. Mais uma vez voltamos a ter a complexidade característica das AVLs de Θ(log n).

* Edição:

Para esta estrutura poderiam ser implementados dois métodos para realizar a edição. Um deles passaria por pesquisar na árvore pelo nó que se desejava alterar e alterar um valor, embora a complexidade desta operação fosse a mesma das pesquisas, Θ(log n), poderíamos perder uma propriedade bastante útil desta estrutura que seria a sua organização, afetando assim pesquisas futuras e podendo mesmo resultar numa pesquisa falhada (pelo método implementado).

Para evitar isto e sem sacrificar a complexidade o que foi implementado é um processo de pesquisa, após termos encontrado o nó que se deseja alterar irá ser feita um copia da informação antiga para um novo nó com, juntamente com a nova informação a ser adicionada. O nó antigo será removido da árvore e o novo será inserido. Assim a árvore permanecerá ordenada e a inserção irá ser balanceada automaticamente. Estas três operações (pesquisa, remoção e adição) por serem chamadas de forma isolada vão permanecer com a complexidade Θ(log n).

* Remoção:

A operação de remoção tem a mesma base para países e anos, apenas com ligeiras diferenças em termos de código. Basta apenas fornecer o ano/país que se deseja remover da árvore e o método irá pesquisar pelo nó desejado remove-lo da árvore e prosseguir ao rebalanceamento da árvore tudo isto com a mesma complexidade de todas as outras operações isoladas, Θ(log n).

# **4 – Casos de Teste**

De modo a testar as nossas estruturas de modo válido e eficiente, foram adicionadas funções de *benchmark* a cada uma das estruturas de dados.

A funçoes de teste implementadas foram: inserção, pesquisa e remoção de anos e países. Isto é feito de forma a que todos os países / anos adicionados são posteriormente pesquisados e finalmente removidos

Estas funções foram implementadas de maneira semelhante em todas a estruturas, no entanto existem alguma diferenças. Nas listas duplamente ligadas testou-se a inserção na lista (ordenada) de anos em diferentes sitios da lista.

**Os resultados foram os seguintes:**

## Estrutura 1:

Stacks de Stacks

## Estrutura 2:

Listas Duplamente Ligadas de Listas Duplamente Ligadas

## Estrutura 3:

Árvore AVL de Árvores AVL

# **5 – Comparações**

# **6 – Conclusões**