FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

1º Projecto Prático

Algoritmos e Estruturas de Dados

2016/2017

João Afonso Póvoa Marques – 2015227041

José Miguel Saraiva Monteiro – 2015235572

Leonardo Machado Alves Vieira – 2015236155

Índice

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]

A primeira estrutura implementada foi um stack de stacks. Esta estrutura foi escolhida devido ao contacto que tivemos com ela durante as aulas, e à sua fácil implementação. Outro fator decisivo na escolha desta estrutura foi a "curiosidade" de saber o quão afetadas seriam as diversas operações, pela escolha de uma estrutura de dados ligeiramente menos eficiente.

As principais desvantagens desta estrutura são a pesquisa e o acesso aos seus valores, operações que possuem uma complexidade de $\Theta(n)$ (embora *n* seja minimizado quando trabalhando com dados ordenados), mas que em operações de adição e remoção a complexidade é meramente $\Theta(1)$, o que se torna num ponto a favor desta estrutura.

No contexto deste problema, para facilitar a pesquisa em anos, todas as stacks de anos estão ordenadas por ano (anos mais 'antigos' estão mais acima na pilha) isto é, não é necessário percorrer a stack inteira se o valor estiver logo no início, melhorando assim a performance. O mesmo não se verifica no stack dos países, o que poderá diminuir a performance do programa. É importante também referir que na stack dos anos não é feita verificação se existe duplicação de dados.

Poderia no entanto, ter sido implementado algum tipo de estrutura auxiliar para melhorar o desempenho.

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 de https://github.com/recluze/python-avl-tree/blob/master/simple_avl.py]

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 $\Theta(\log n)$. 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 $\Theta(1)$ não irá afetar a complexidade da pesquisa.

3 - Operações Implementadas

Estrutura 1:

Stacks de Stacks

- Pesquisa:

  Tal como já foi referido acima, tanto a pesquisa como o acesso a dados são os pontos menos favoráveis da estrutura tendo em conta que se a mesma não estiver ordenada, a complexidade será sempre Θ(n), pois será necessário percorrer a estrutura toda até encontrar o valor desejado. No entanto, na pilha de anos, os valores encontram-se ordenados, o que irá fazer com que o *n* para esta stack seja reduzido. Ficamos assim com Θ(1) no melhor cenário de complexidade e Θ(n) no melhor cenário.

  Foi também implementada uma função de pesquisa na stack dos anos, em que fornecendo como input dois valores, um mínimo e um máximo, irão ser impressos todos os anos com valores compreendidos entre o mínimo e o máximo. Esta função de pesquisa terá sempre uma complexidade de Θ(n), pois será sempre necessário percorrer a stack inteira para saber quais os anos com valores compreendidos entre os que são dados à função.

  Para além da função de pesquisa de valores num certo intervalo, foi também implementada uma função que pesquisa se existe informação de um determinado ano, em todos os países. Como para saber se existe esta informação para cada país existente é necessário percorrer a stack dos países, e posteriormente a stack dos anos à procura do valor, a complexidade desta função será menos boa, sendo esta igual a $\Theta(n^2)$ num limite superior, ou então Θ(n) num limite inferior. Estes valores poderiam ser melhorados, caso se optasse por utilizar uma estrutura de dados auxiliar.

  Na stack de países foi apenas implementada uma função de pesquisa de complexidade Θ(n), que desencadeia a maior

parte de métodos, quer para adicionar, pesquisar, remover ou até imprimir informação.

- Inserção:

  Para esta operação, foram implementados dois métodos, um para inserir novos elementos à stack de países (desordenadamente), e outro para adicionar novos elementos à stack de anos / valores (respeitando a ordem dos anos).

  No caso da inserção de novos elementos na stack de países, a função limita-se a adicionar as informações que o utilizador fornece (nome do país e código do mesmo, neste caso) e adicionar ao topo da pilha. É de notar que esta implementação não verifica se há duplicação de dados. Tal operação irá ter uma complexidade $\Theta(1)$.

  No caso da stack de anos, a implementação realizada preza manter a informação ordenada e não duplicada, isto é, cada vez que se adiciona um novo elemento, acontece uma pesquisa prévia à stack existente, que primeiramente serve para adicionar o novo valor na posição pretendida, e, caso este já exista, para não permitir a sua duplicação. A complexidade de tal operação, será $\Theta(1)$ num caso ótimo, e $\Theta(n)$ no pior caso possível, em que teremos que adicionar o valor no fim.

- Edição:

  Em relação à edição de valores, temos duas possibilidades implementadas. Podemos simplesmente editar o valor correspondente a um ano existente, ou então editar o ano em que um certo valor está associado.

  Para a edição do valor, é simplesmente feita uma pesquisa na stack de países, à procura do país, e posteriormente executada uma pesquisa na stack dos anos, até encontrar o ano do qual se pretende editar o valor. A complexidade de tal operação é $\Theta(n^2)$ no pior cenário, tendo em conta que a função para pesquisar o ano encontra-se dentro do ciclo de pesquisa do país.

Para editar o ano, o cenário é bastante parecido com o anterior, primeiro existe um ciclo que procura o país pretendido, depois disso chama a função para editar o valor do ano pretendido, que irá chamar a função para adicionar um novo ano. Sendo assim, a complexidade da função será $\Theta(n^3)$ no pior cenário possível, tornando esta estrutura possivelmente na pior para esta operação, devido à sua complexidade.

- Remoção:

  Para a remoção de um elemento, temos duas opções, ou remover um país e consequentemente toda a informação relativa a esse mesmo país, ou então simplesmente remover a informação relativa a um ano de um certo país.

  A implementação destes métodos foi bastante parecida, tanto para a stack de países, como para a stack de anos. Inicialmente realiza-se uma pesquisa na stack de países, (operação com complexidade $\Theta(n)$), e posteriormente, caso o elemento seja encontrado, faz-se o 'pop' do elemento, caso o objetivo seja remover o país, caso contrário chama-se outro método que irá percorrer os anos do país selecionado à procura do ano pretendido. A complexidade de tal função seria $\Theta(n)$ caso estejamos perante um caso ideal, ou então $\Theta(n^2)$ no pior cenário, tendo em conta que se iria ter que percorrer as 2 stacks.

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 $\Theta(n)$, 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 $\Theta(1)$ e num limite superior a complexidade base de $\Theta(n)$.

  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 $\Theta(n)$ 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 $\Theta(n^2)$ num limite superior e $O(n)$ num limite inferior. Note-se que isto poderia ser melhorada com uma estrutura auxiliar com as percentagens 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 $\Theta(n)$ 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 $\Theta(1)$, 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 $\Theta(1)$. Nos restantes dos casos será de uma complexidade de $\Theta(n)$, 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 ($\Theta(n)$), e caso estes existam é feito uma mudança do valor da variável respetiva, sendo isto uma operação $\Theta(1)$.

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 (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*. No entanto, na lista de países, como não se encontra ordenada, irá inserir sempre no final da lista.

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.

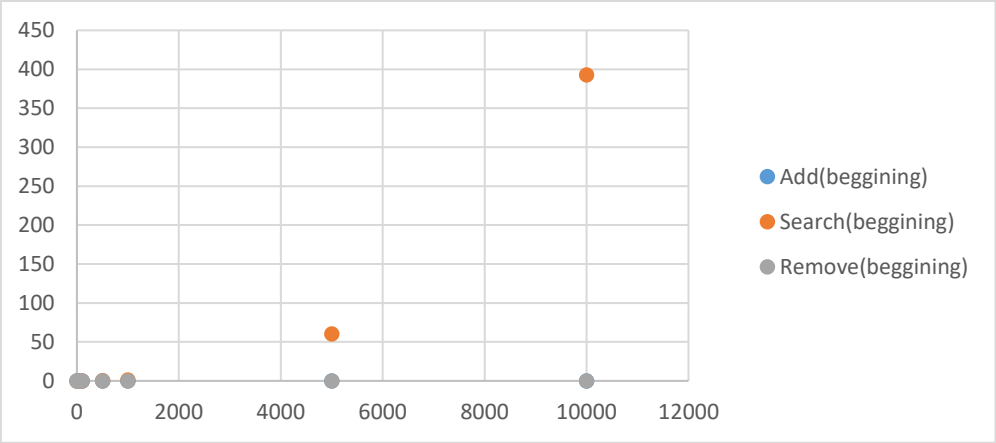
Os resultados foram os seguintes:

Nota: Todos os testes foram efetuados na mesma máquina e nas mesmas circunstancias.

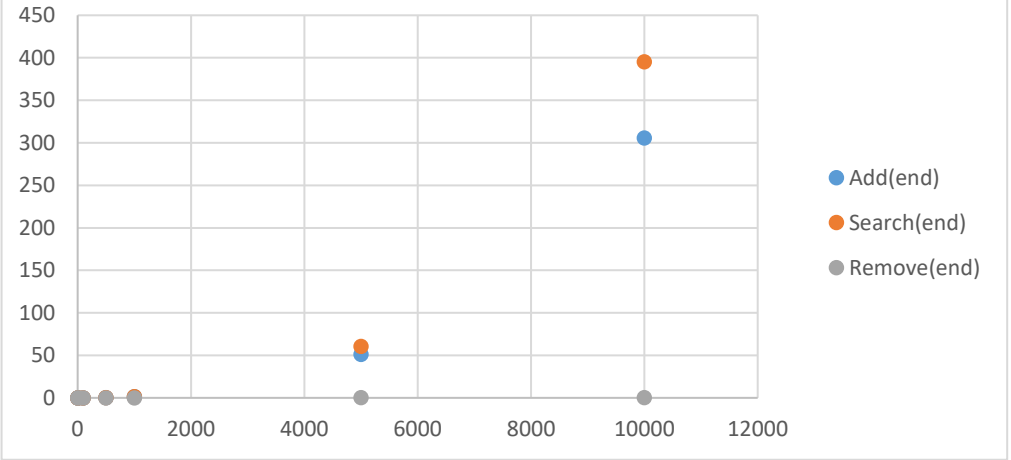
Estrutura 1:

Stacks de Stacks

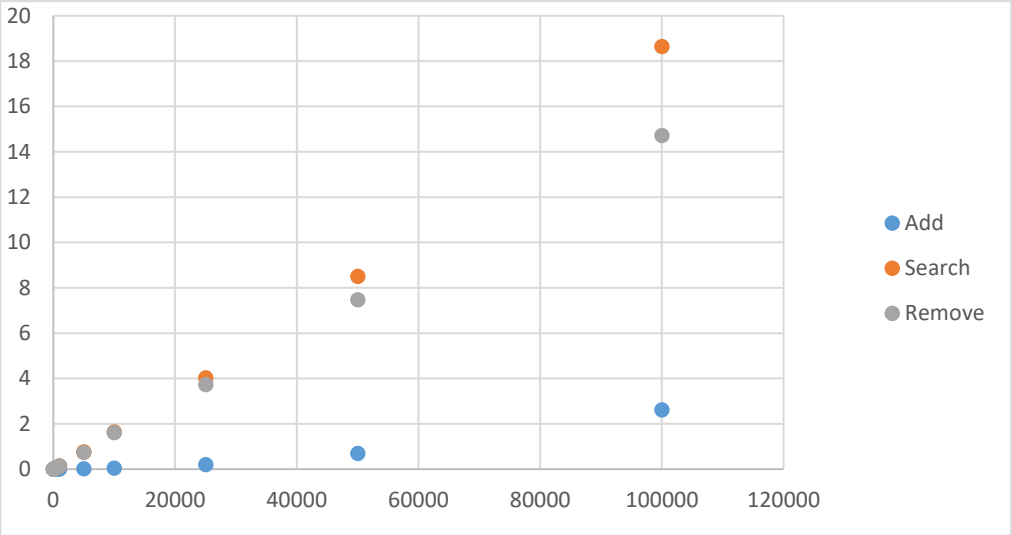
Years	Add(beggining)	Search(beggining)	Remove(beggining)
1	0	0	0
10	0	0,000501632	0
100	0	0,011052	0
500	0,00401677	0,2687836	0,001003
1000	0,011029	1,206995	0,003009
5000	0,022089	60,4348	0,016042
10000	0,0861306	392,7089	0,0365958



Years	Add(end)	Search(end)	Remove(end)
1	0	0,000502	0
10	0	0	0
100	0,013065	0,0115599	0,0009999
500	0,3044484	0,2769081	0,0050108
1000	1,289804	1,259831	0,0030086
5000	51,139271	60,4	0,0812137
10000	305,728926	395,14635	0,2716866



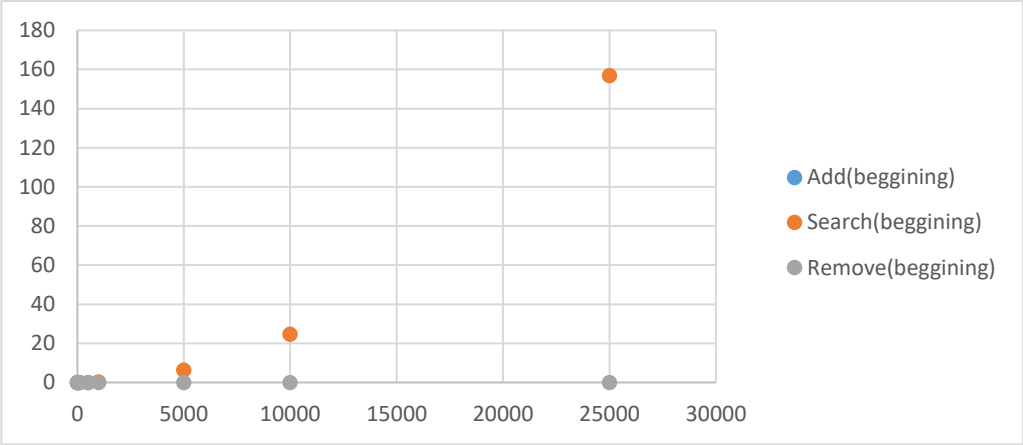
Countries	Add	Search	Remove
1	0	0	0
10	0	0,0014942	0,00150418
100	0,0005016	0,0165433	0,01554298
500	0,00099	0,076728	0,0731937
1000	0,0020051	0,1534416	0,1488938
5000	0,0215542	0,767116	0,74456358
10000	0,05314088	1,6488998	1,61632966
25000	0,1986258	4,0281	3,7279257
50000	0,690887	8,510396	7,480846
100000	2,6196351	18,64933	14,72559



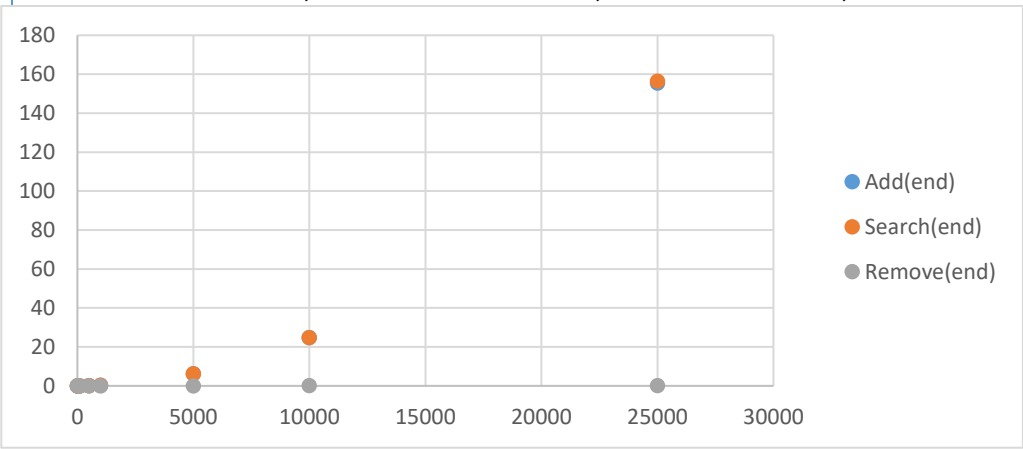
Estrutura 2:

Listas Duplamente Ligadas de Listas Duplamente Ligadas

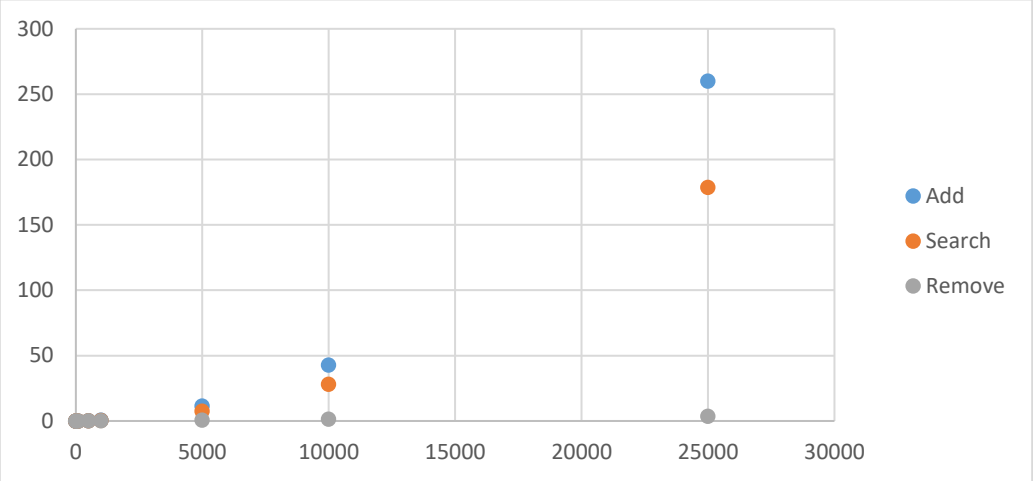
Years	Add(beggining)	Search(beggining)	Remove(beggining)
1	0	0	0
10	0	0	0
100	0	0,003	0,001
500	0,001	0,062196	0,002
1000	0,00298	0,261793	0,003
5000	0,012566	6,37418	0,016043
10000	0,022083	24,8015	0,029105
25000	0,059739	156,9113	0,073695



Years	Add(end)	Search(end)	Remove(end)
1	0	0	0
10	0	0	0
100	0,003	0,003	0
500	0,06317	0,06317	0,003
1000	0,2507989	0,25127	0,006
5000	6,195222	6,26639	0,0325
10000	24,7003	24,82472	0,055149
25000	155,40137	156,4717	0,1433839



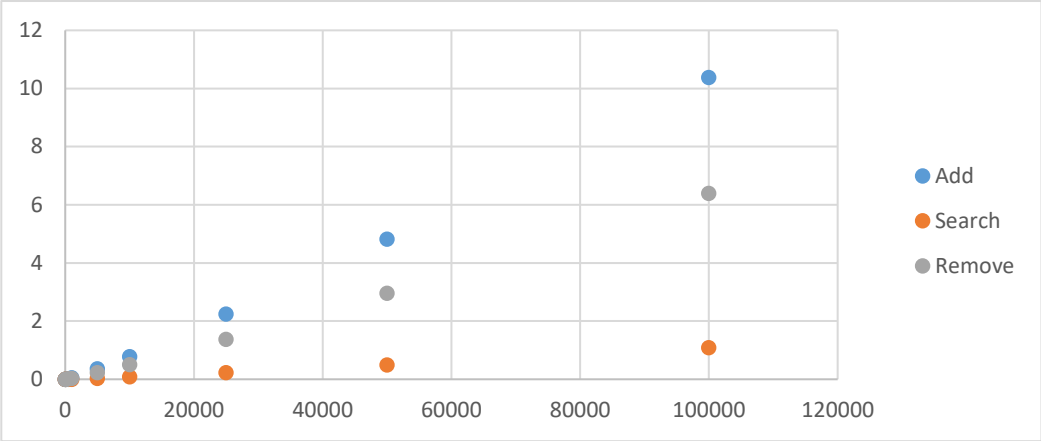
Countries	Add	Search	Remove
1	0	0	0,00100255
10	0,00302	0,0010023	0,0019862
100	0,02608323	0,017044544	0,015577555
500	0,209294795	0,133892297	0,07119131
1000	0,6187822	0,4151294	0,14787125
5000	11,332543	7,50117	0,7305088
10000	42,744488	28,08081	1,4304854
25000	259,953381	178,72584	3,6084306



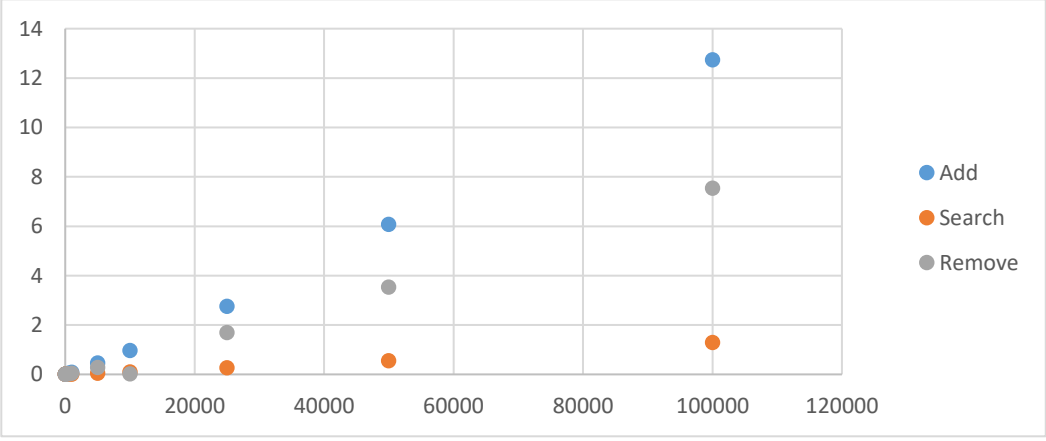
Estrutura 3:

Árvore AVL de Árvores AVL

Years	Add	Search	Remove
1	0	0	0
10	0	0	0
100	0,00452	0	0,00253677
500	0,02807	0,002007	0,02604247
1000	0,05924225	0,00601601	0,035063
5000	0,36556	0,0376644	0,2306759
10000	0,7851	0,0857932	0,5120484
25000	2,2499295	0,23262546	1,374312
50000	4,8180699	0,4989347	2,9591327
100000	10,3872	1,0888	6,400197



Countries	Add	Search	Remove
1	0	0	0
10	0,001003	0	0
100	0,00802	0,0010033	0,0030396
500	0,0380336	0,0030079	0,02105665
1000	0,0872643	0,00651717	0,04462208
5000	0,4560308	0,04210888	0,279415
10000	0,97002578	0,09174728	0,0169114
25000	2,7561097	0,2592139	1,685565
50000	6,072265	0,5550939	3,53233
100000	12,73679	1,29178	7,53485



5 – Análises e Comparações

Analisando e comparando os tempos da AVL começamos a ter uma ideia mais concreta dos seus benefícios face as outras estruturas implementadas, assim como as funções em que perde em termos de tempo mas que nem sempre se pode considerar um aspecto negativo. Quando foram executados os testes para a inserção, pesquisa e remoção de países na AVL as conclusões começaram a aparecer obvias os tempos de qualquer operação sobre países estava a ser inferior a qualquer outra estrutura implementada. Para a organização de paises a AVL demonstra ser superior às outras.

Quando passamos a analisar tempos das mesmas operações para os anos os números começam a contar uma história diferente. A adição de novos anos já não é a mais rápida quando comparada com a adição no inicio da lista nem quando comparada com a adição no fim da pilha. A história repete-se e a AVL não manifesta melhoria de tempo em relação a nenhuma outra estrutura, ainda assim estes tempos poderão não ser considerados como maus para além de ter a vantagem de manter a ordem durante todos os processos, coisa que as outras estruturas não estão a assegurar. Esta ordem é mais um ponto de ajuda nas pesquisas o que também se reflete no tempo de pesquisa que mais uma vez se demonstra superior em relação às outras estruturas. Assim, tanto para os anos como para os países, a AVL demonstra ser a estrutura ideal para pesquisas. No que toca a adições/remoções os casos que apresentam tempos mais altos não parecem ser melhoria suficiente para perder em tempos de pesquisa e perder o ordenamento inerte às AVLs.

Olhando para os tempos relativos à stack, penso que é seguro dizer que em termos de adição e/ou remoção, consegue bater os tempos da AVL, mas fica ligeiramente atrás das listas duplamente ligadas de listas duplamente ligadas, mas no que toca a pesquisa, é a pior das 3 estruturas. Comparando diretamente com as AVLs, a explicação para a diferença de tempos nestas operações, no caso de adição e de remoção, está diretamente associado com a diferença de complexidades, $\Theta(1) > \Theta(\log n)$, pois enquanto que na stack o novo elemento se adiciona diretamente no topo, as AVLs têm que se rebalancear, operação tal que irá consumir mais tempo, mas irá ajudar na pesquisa. Em relação às listas, ao adicionar países, como a pilha não verifica se o país que se pretende adicionar já existe na estrutura, mas as listas sim, as pilhas irão ter um tempo muito inferior às listas, pois se limitam a adicionar os elementos no topo, sem verificação de duplicação de

informação. Quando se fala em remover elementos, a situação é parecida, enquanto que as AVLs precisam de se rebalancear depois da operação, a pilha só precisa de voltar a receber os elementos guardados numa estrutura auxiliar, depois de remover os elementos pretendidos. Quando comparada às listas, as pilhas têm um tempo ligeiramente pior, tendo em conta que a stack tem que receber os elementos que foram para a estrutura auxiliar, e as listas apenas têm que mudar o apontador do elemento que precede o que irá ser removido, para o elemento que sucede o elemento a ser removido.

Olhando agora para os valores das listas duplamente ligadas, são apresentados tempos excelentes para a inserção de anos no inicio da lista. Isto deve-se ao facto dos calores estarem ordenados e da função de pesquisa não ter de correr toda a estrutura. No entanto para a pesquisa que se segue e da tabela *end*, os valores temporais são maiores. Isto deve-se ao facto da distancia do nó a procurar ao inicio da lista ir aumentando fazendo com que tenha de percorrer mais nós. O contrário verifica-se nas duas remoções em que os tempos são também muito bons comparativamente às outras estruturas implementadas. Isso porque o programa vai removendo os nós sempre iniciais ou perto fazendo assim com que a complexidade seja $\Theta(1)$ ou perto.

Relativamente aos países, como já foi referido, é feita uma verificação da existência previa do nó, e como esta lista não se encontra ordenada, teremos de iterar por todos os nós e só no fim, devolvido o ultimo nó da lista, inserir o novo país.

6 – Conclusões

Após a análise destes valores e prós e contras das várias estruturas estamos em boas condições para tomar uma decisão de qual a melhor estrutura implementada neste trabalho. Em geral a AVL demonstra ser a estrutura superior, as suas características permitem fácil organização e em geral bons tempos para pesquisas e adições, em muitos casos os melhores tempos das estruturas implementadas. Por outro lado, temos também as listas como uma opção viável devido à sua fácil implementação. Uma conclusão que se pode também tirar é que as stacks em poucos casos reais seriam uma opção preferível não só devido aos seus fracos tempos como a dificuldade de implementar e trabalhar com uma estrutura deste tipo. Ainda assim poderíamos explorar outras estruturas com outro tipo de implementações para melhorar as nossas situações concretas ainda assim pensamos que os resultados obtidos são bastante satisfatórios.