Trabalho final

Análise de sentimento

O presente relatório visa comparar o desempenho distinto de duas aplicações, as quais têm por objetivo analisar as sentenças em um arquivo .txt, para isso devíamos construir pelo menos dois modos diferentes para obter a análise de cada sentença. As sentenças a serem avaliadas pelas aplicações estão escritas linha a linha do arquivo .txt de sentenças e o arquivo de léxico é usado para aferir o valor de cada sentença e também é uma entrada em .txt necessária as duas aplicações. Em resumo, as entradas e saídas são iguais a ambas aplicações, logo o que difere é somente o modo como comparamos o que temos no nosso léxico passado na entrada com cada termo de cada linha do arquivo de sentenças. As aplicações escolhidas o foram de acordo com a sugestão dada no arquivo explicativo do trabalho, entre as sugestões estava ABP vs AVL. Optei por está pelo fato do desenvolvimento de uma ABP me parecer bem agradável e AVL por já conhecer as funções de rotação. Por fim, vou mostrar que convenientemente temos muitas funções comuns as duas aplicações assim como tipos de dados criados que ou são iguais ou diferem por muito pouco. Agora, vou iniciar falando de aspectos comuns aos dois modos escolhidos, podemos claramente ver que a diferença gritante está nas árvores e como ambas são implementadas, porém o restante do código é basicamente o mesmo. As duas aplicações são feitas cada uma em um TAD, esse é divido em três arquivos respectivamente, os conhecidos de protótipo e dados, de implementação e por fim o main. O main fez-me lembrar de que ambas as aplicações devem ser executadas via prompt de comando e para isso seguimos as recomendações do arquivo do trabalho e vídeo postado no moodle da info.

Inicialmente, podemos dizer que os tipos de dados utilizados são basicamente iguais, exceto uma pequena modificação que acabou por gerar um novo tipo de dado a partir do tipo ptNodoABP que é o ptNodoAVL, ambos tem mesma estrutura interna, todavia ptNodoAVL tem um campo a mais que é o int fb, pois é preciso para balancear a árvore durante seu processo de criação nas AVL's. O outro tipo de dado criado, e o último criado, foi o ptLSE, esse tipo é comum nas duas aplicações e extremamente útil, pois armazena na ordem correta e na quantia certo o número de escores para cada uma das linhas, segundo a proposta do trabalho. O tipo ptLSE serve para criação da lista simplesmente encadeada que será alocada durante o período de execução, para isso temos funções adequadas para lidar com esse tipo. Antes de tratar das funções que lidam com os tipos estruturados que criamos se pode observar que poderíamos ter usado o tipo fila, na verdade escolhi usar o tipo lista por parecer mais flexível e não ter de pensar num novo tipo estruturado, o descritor de início e fim usado em fila, para mim, lidar com menos tipos é melhor nesse caso, porém sei que poderia ter usado fila tranquilamente. Com isso, encerramos aqui e vamos olhar para as funções.

As funções inicializaLSE, insereNoFimLSE, removeDoInicio e destroiLSE, dizem respeito ao tipo de dado criado ptLSE, e são de fundamental importância para ambas as aplicações, porque permite o armazenamento na ordem precisa das

sentenças além de ter um tamanho ajustável as necessidades do input dado com o arquivo .txt de sentenças. Poderia entrar em detalhes aqui, contudo como são procedimentos relativamente simples de se desenvolver acredito não ser necessário, somente destaco o fato da função insereNoFimLSE ser recursiva.

As funções que vou colocar aqui são idênticas em ambas as aplicações, todavia possuem nomes distintos, isso foi feito com o intuito de as distinguir mais facilmente durante o processo de desenvolvimento. Por exemplo, inicializaArvoreABP é igual a inicializaArvoreAVL e assim por diante com insereNodo***, busca*** e destroi*** todas equivalentes em implementação, apenas com o final do nome diferente de ABP para AVL. Todas elas têm em comum o fato de serem recursivas, exceto inicializa, cabe destaque a função insereNodo***, pois usa o modo de inserção de árvores binárias de pesquisa mesmo quando sob o nome de insereNodoAVL, no caso de AVL o balanceamento é verificado somente posteriormente a inserção na árvore. Um último aspecto interessante a um desses procedimentos, especificamente a funcao busca***, é o fato de dentro dele se contar o número de acessos para encontrar a palavra procurada.

A função criaArvore***, tendo nome semelhante em ambas as aplicações, é a mais importante para distinção das aplicações, porque cria as árvores, de maneira significativamente diferente. A forma do procedimento é quase igual até chegar o conjunto, a mais, de funções desenvolvidas exclusivamente para a árvore AVL. As funções desenvolvidas para AVL são atualizaFB, fatorAVL, menorDesbalanceado, aplicandoRotacao. Além dessas temos as funções que são usadas dentro destas últimas, de modo composto, como por exemplo: alturaNodo, fatorBalanceamento, rotacao_direita, rotacao_esquerda, rotacao_dupla_direita, rotação_dupla_esquerda. A ideia de criaArvore*** é reunir em si todos os ingredientes necessário para a criação da árvore em um único local da main, assim ela acaba por executar muitas coisas e retornar a árvore completa. O processo interno para criar a ABP é bem mais simples em relação à AVL como foi dito anteriormente, foi necessário o desenvolvimento de muitas funções a fim de possibilitar a AVL. A ABP tem inserção na árvore segundo a ordem de disposição no arquivo de léxico e segundo a ordem lexicográfica de cada elemento a ser inserido, contudo nas AVL há, além dos critérios da própria ABP, os fatores de balanceamento de cada elemento não podendo ter modulo maior do que 1, assim é preciso ter muito mais cuidado ao inserir na árvore.

Há ainda umas funções comuns entre os programas que são a leTXT, escreveEscoresNova, escreveComparacao. A leTXT junto com a criaArvore*** são as mais fundamentais, pois uma obtêm a árvore e a outra consegue os escores numa lista adequada a uma utilização posterior, a escreveEscores e a escreveComparacao simplesmente escrevem no arquivo de resultado que queremos ao fim da execução.

Voltando a funções já citados, todavia não bem esclarecidas, como as rotações, foram obtidas dos slides da aula acerca das AVL's, porém as modifiquei e me retifiquei de seu funcionamento adequado para minha ideia de funcionamento da aplicação em questão. As modificações foram, basicamente em torno da não necessidade de alterar os fatores de balanceamento dentro dessas funções, pois já havia pensado em fazê-lo fora delas com uma função externa, a qual caberia obter os fatores para toda a arvore a atualizando a cada mudança, o nome dessa função é

atualizaFB. A observação de que outros procedimentos como fatorAVL, obtêm o maior fator em modulo da função, nos ajudam a saber quando a árvore está desbalanceada, ou seja, existe ao menos um nodo que está desbalanceado. Agora, talvez, a mais sofisticada das funções, menorDesbalanceado, essa função retorna o nodo desbalanceado de menor altura entre os nodos desbalanceados, no caso de existir mais de um, ela é muito relevante, porque trata algo que facilmente pode estragar todo o balanceamento da arvore que é a ordem das rotações a serem feitas, essa função cumpre esse papel. Ademais, cabe ressaltar que boa parte desses procedimentos dependem muito da estrutura adaptada feita dentro da criaArvoreAVL que modificou boa parte da estrutura inicialmente feita para ABP's, essa estrutura é composta unicamente pela adição de um {} num comando if já existente, além de um comando extra do-while que nos assegurou que novos nodos somente seriam inseridos em uma árvore previamente balanceada, logo o do-while é responsável por possíveis sequencias de rotações necessárias a uma árvore possivelmente grande.

O esquecido procedimento aplicandoRotacao, acabei esquecendo de citar junto com leTXT e criarArvore***, como também muito relevante no contexto das AVL dado que tem um fato um tanto surpreendente, que é de ser recursiva mesmo sendo grande e bem composta, essa função junto com as funções de rotação tiveram de ser repensadas após a verificação da necessidade de ter de trabalhar diretamente com retorno dos ponteiros, na verdade, tudo isso veio a tona com testes que fiz para debugar as funções, os quais em alguns testes houve a necessidade de alterar a própria raiz da árvore continuamente e não somente nodos internos e folha, sendo assim era impossível construir tais funções sem um retorno como eu havia pensado a priori. Para finalizar essa parte de descrição dos programas, depois veremos o desempenho dos dois comparados, vou dissertar a cerca da maior parte da criação que o ato de corrigir pequenos erros, geralmente ligados sintaxe, que demandou uma boa quantia de tempo e paciência conhecido como debugar.

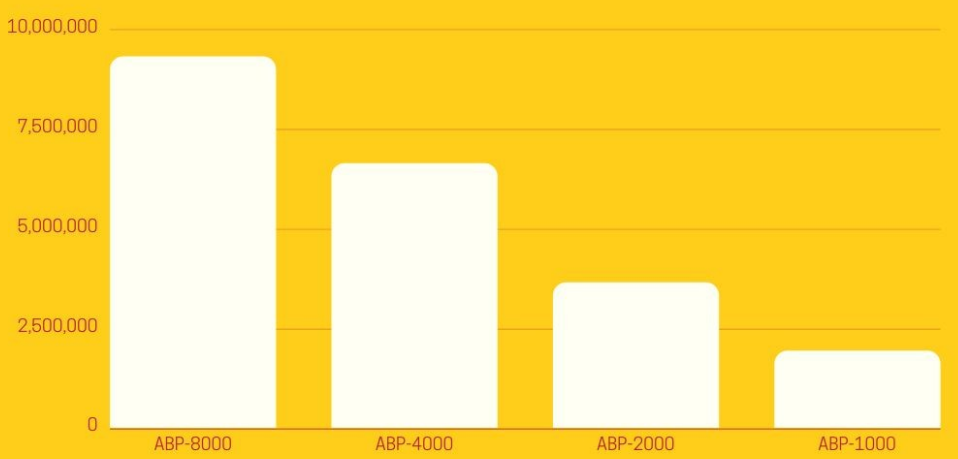
A parte base de AVL vem inteiramente da parte de ABP muito bem debugada, logo quando comecei a construir para AVL tinha certeza da ABP. Isso ocorre porque achei os erros na aplicação ABP, os quais a maior parte era de sintaxe e outra parte relacionada ao não entendimento correto do funcionamento de ponteiros e como lidar com funções aplicadas a arquivos .txt. Procurei documentar no código onde e como resolvi certos problemas e deixarei alguns testes que fiz durante o processo para evidenciar como os resolvi e porquê o fiz de tal modo. O que foi feito para ABP também o foi para AVL, entretanto nas AVL's tive de recorrer a testes de mesa para diferentes situações, coisa que não fiz nas ABP's dado que me pareceu mais simples e natural a criação dessas árvores em relação as anteriores. Acredito que não citei os problemas nas AVL, diferentemente das ABP's, as AVL's tinham problemas na lógica que empreguei, no retorno de certas funções e, também, sintáticos. Antes de irmos para comparação das aplicações tenho de falar da main, a qual quase não falei, que idêntica nas duas aplicações, com exceção da troca de nomes, porém tem exatamente a mesma forma. Agora, vamos falar sobre as diferenças de desempenho segundo certos critérios que serão adotados.

AplicacaoABP vs AplicacaoAVL

Vamos comparar as duas aplicações recebendo as mesmas entradas quanto a quantidade de comparações que cada um faz, variando o tamanho do léxico e variando o tamanho das sentenças. Vamos testar com o mesmo léxico para 8000, 4000, 2000 e 1000 palavras, sendo que léxico com menos estão contidos nos maiores, o mesmo vale para as sentenças, teremos de 1000, 500, 250 e 125 sentenças, sendo o arquivo de sentenças menor contido no maior.

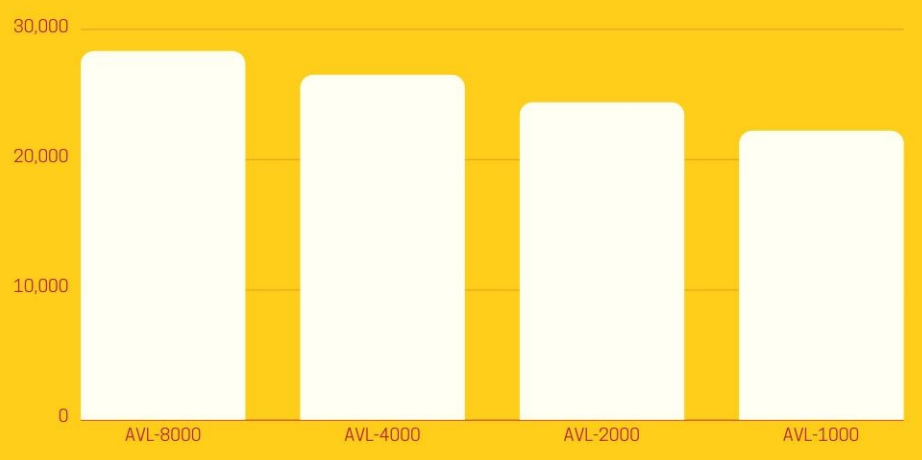
COMPARAÇÕES ABP

Quantidade de comparações para arquivo de sentenças com 125 sentenças, os valores ao lado de ABP- são o tamanho do léxico utlizado.



COMPARAÇÕES AVL

Quantidade de comparações para arquivo de sentenças com 125 sentenças, os valores ao lado de AVL- são o tamanho do léxico utlizado.



Vou postar uma tabela com os dados usados para plotar os gráficos e uns dados a mais de brinde.
A primeira tabela compara ABP e AVL para um arquivo de 1000 sentenças com a quantia de palavras do léxico variando, já a segunda tabela segue o mesmo esquema, porém com um arquivo de sentenças com 125 sentenças.

1000 - Sentenças

Tamanho do léxico	Total comparações - ABP	Total comparações - AVL
8000 palavras	70.300.508	218.700
4000 palavras	50.459.482	204.233
2000 palavras	27.874.704	187.818
1000 palavras	14.896.924	171.056

125 - Sentenças

Tamanho do léxico	Total comparações - ABP	Total comparações - AVL
8000 palavras	9.310.303	28.288
4000 palavras	6.631.665	26.461
2000 palavras	3.639.898	24.349
1000 palavras	1.924.392	22.166

Podemos também olhar para os dados no sentido de quantas vezes mais comparações são necessárias numa ABP do que numa AVL.

1000 - Sentenças

Tamanho do léxico	Total-ABP / Total-AVL
8000 palavras	321,4
4000 palavras	247
2000 palavras	148,4
1000 palavras	87

125 - Sentenças

Tamanho do léxico	Total-ABP / Total-AVL
8000 palavras	329,1
4000 palavras	250,6
2000 palavras	150,1
1000 palavras	86,8

Por fim, vamos analisar os gráficos de modo a comparar ABP com AVL, observe que as AVL's tem um número de comparações significativamente menor que ABP, por exemplo para 1000 sentenças com dicionário de 8000 palavras em ABP temos 70.300.518 comparações contra 218.700 comparações em AVL, ou seja, o número de comparações em ABP nesse caso chega a ser 322 vezes maior que em AVL, observando os gráficos acima fica bem claro a vantagem de se usar AVL quando o quesito é número de comparações necessárias para encontrar o que queremos. Ademais, o mais relevante dos dados é o fato deles apontarem para uma tendência, se olharmos a variação do tamanho do léxico desconsiderando o número de sentenças, ou seja, observarmos somente a variação de tamanho das árvores veremos que a proporção de comparações aumenta significativamente conforme a árvore cresce, logo é muito interessante usar AVL's em árvores cada vez maiores ao invés de ABP's. Logo, a conclusão que podemos tirar é bem simples, para árvores grandes ou muito grandes a eficiência das AVL's aumenta muito, então é muito interessante considerar utilizá-las.

Nome: Anderson Hemicha Alcara
Cartão: 00289464
Disciplina: Estrutura de dados
Professora: Viviane Morreira
Turma: B