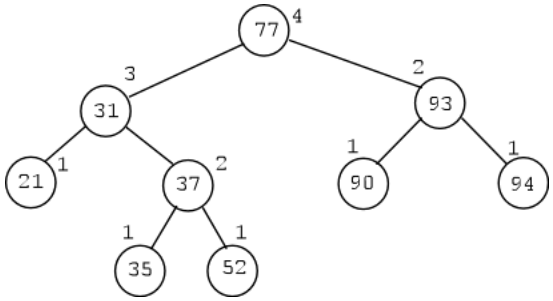
1 - Introdução

Árvores são estruturas de dados bastante eficientes para armazenamento de informações, sendo um ponto médio de eficiência dentre os vários requisitos, como eficiência de pesquisa, inserção, remoção, uso de memória, etc. Em particular, as árvores de pesquisa possuem otimizações em sua arquitetura que as tornam ainda mais eficientes, sobretudo nos processos de pesquisa.

Uma árvore é basicamente um grafo direcionado, acíclico, que pode conter apenas um nodo, ou então uma quantidade determinada de nodos filhos, sendo que cada um destes nodos filhos é por si só uma árvore também. Chamamos de árvore binária uma árvore em que cada nodo tem apenas dois filhos.

Exemplo de árvore binária:

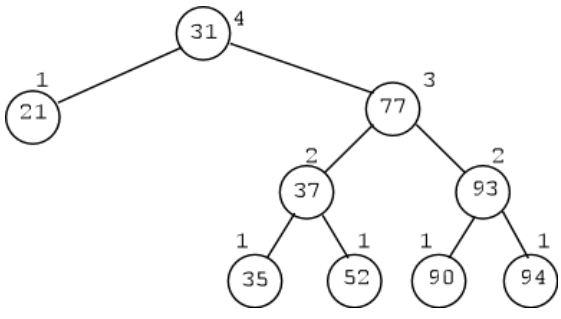


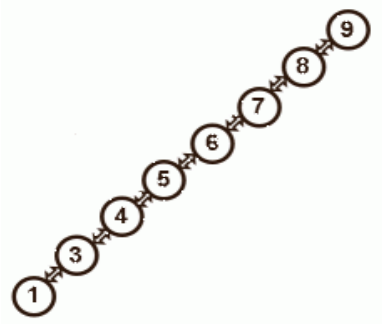
Árvores de pesquisa não são necessariamente binárias, podendo ter efetivamente ter quantidades variáveis de nodos-filhos, a árvore de pesqisa binária é porém uma forma bastante simples e eficiente de implementar essa estrutura de dados. Numa árvore binária de pesquisa a premissa é:

Para um determinado nodo X, sua sub-árvore da esquerda contém apenas nodos cujas chaves são lexicograficamente menores que a chave de X, enquanto na árvore da direita temos chaves lexicograficamente maiores do que a chave de X.

Dessa forma obtemos uma estrutura em que, sempre que percorrermos a árvore para esquerda, obteremos resultados menores, e para a direira vice-versa. A grande vantagem é que, como estamos trabalhando numa estrutura de duas dimensões, o tempo de pesquisa é drasticamente reduzido, se comparado a uma estrutura unidimensional como uma lista linear.

Exemplos de árvores desbalanceadas / degeneradas





Uma árvore porém pode perder sua eficiência se chegar num estado que chamado de "Árvore Degenerada". Nesse estado, a árvore está "desbalanceada". Uma árvore desbalanceada, ou degenerada é caracterizada por uma estrutura que, devido à ordem de inserção das informações, acabou por ter seus dados "horizontalizados". Nesse formato o custo de pesquisa da árvore se afasta do ideal, que é da ordem de $_{log}N$, assumindo um tempo de pesquisa de uma lista linear comum.

Para evitar a degeneração de árvores de pesquisa, existem várias técnicas e restrições que podem ser adotadas na construção destas árvores. Estes métodos porém não fazem parte do escopo deste documento e serão ignorados.

2 – Estruturas de Dados

A estrutura principal em que se baseia a aplicação é a de uma árvore de pesquisa binária, não balanceada. Os dados recebidos através do arquivo de entrada são indexados em duas dessas árvores, uma usando o CPF como chave, a outra usando o Nome da pessoa.

Optei por incluir além dos campos clássicos, um campo *parent*, que indica o nodo pai do nodo atual. Esse procedimento facilita os processos de remoção e inserção recursivos adotados. A parte disso, a implementação é a clássica para uma árvore binária de pesquisa, apenas adicionando os campos e métodos referentes ao tratamento das informações específicas da aplicação.

3 - Implementação

A aplicação recebe em linha de comando um arquivo texto com a seguinte estrutura:

```
nome1
endereço1
cpf1
idade1
sexo1
nome2
endereço2
cpf2
idade2
sexo2
endOfFile
```

Ao final do arquivo de entrada, para finalizar a entrada de dados, haver uma linha com palavra `endOfFile`, assim o programa pára a leitura e inicia o prompt para consulta de dados.

Ao rodar a aplicação será apresentado o seguinte menu:

```
PeopleIndexer(Arvores Binarias) - TPV - AEDSII
Aluno: Gustavo Campos Ferreira Guimaraes - 2005041291
Professor: Roberto Bigonha
-----------------------------------------------------

Inseridos X registros

Escolha uma opcao:
1 - Buscar um registro por CPF
2 - Buscar registros por nome
3 - Apagar um Registro
4 - Visualizar banco ordenado por CPF
5 - Visualizar banco ordenado por Nome
6 - Sair do Programa:
```

Na linha “Inseridos X registros”, uma referência para a quantidade de registros obtidos no

arquivo é mostrada.

Cada uma das opções executa uma tarefa especificada na proposta. Nos casos das buscas, (1 e 2) e da remoção (3), uma janela de prompt será mostrada para digitação do cpf/nome desejado.

Nas opções 4 e 5, será mostrada a lista de todos os registros presentes na memória, ordenados lexicograficamente pela chave escolhida.

Comentários:

Como pode ser visto nos códigos anexos, optei por usar os métodos compareToIgnoreCase() da classe java.lang.String para comparar lexicograficamente as chaves. Isso evita que seja necessário digitar a palavra exatamente como foi digitada na entrada, facilitando a pesquisa na árvore.

O método de pesquisa por nomes usa um arranjo de nodos de árvore para retornar todas as ocorrências de um determinado nome. O tamanho desse arranjo é fixado como o valor máximo de repetições que podem ocorrer, salvo na variável MAX_NAME_RESULTS. Seu valor arbitrário é foi escolhido em 40 resultados, afinal é o valor máximo de entradas.

4 - Anexos

Este documento contém dois anexos. O Primeiro contém os códigos fonte das classes.

O segundo contém a descrição gerada pelo Javadoc dos métodos da classe HumanTree.java