Nome: Renan Antonio Hammerschmidt Krefta

Descrição do Código:

Esse código foi feito e rodado no Apache NetBeans e foi postado diretamente no GitHub, portanto, talvez sejam necessárias algumas alterações mínimas para que possa ser rodado em outro ambiente fora o Apache NetBeans.

Foi implementado no código a arvore binária de busca e a arvore AVL.

Arvore Binária de Busca:

A arvore binária de busca possui um nó raiz e funções que implementam as operações de busca, inserção e remoção na arvore.

(Maior detalhamento das operações na seção que descreve o Node)

- A inserção checa se o nó raiz existe, se não existir é criado um nó raiz com a informação, e se existir, o elemento é inserido de forma recursiva na arvore.

- A remoção checa se o nó raiz existe e então remove recursivamente da arvore

- A busca checa se o nó raiz existe e então busca recursivamente na arvore.

Node:

O nó utilizado pelas arvores contém em si variáveis para guardar sua informação e os nós esquerdo e direito. Também estão incluídas funções recursivas para inserção, remoção e busca. E para a parte de balanceamento, estão incluídas funções para verificar a altura da arvore e também para checar o balanceamento do nó, funções para rotacionar a esquerda e direita, funções essas que também checam se há necessidade de rotação dupla.

- A função de inserção recebe o elemento a ser inserido e, primeiramente, compara com a informação contida no nó em questão, se o elemento for maior ou igual a informação contida no nó, ela passa a inserção para o nó da direita, se existir, se não existir ela cria um nó a sua direita. Se o elemento for menor

que sua informação, ela passa para a esquerda se existir, se não, ela cria um nó a sua esquerda com o elemento.

- A função de busca recebe um elemento a ser procurado na arvore e retorna o nó que contém aquele elemento, essa função primeiramente verifica se a informação do nó é igual ao elemento, se sim, ela retorna o seu nó, se o elemento for menor que sua informação, ela procura na esquerda recursivamente e retorna a busca de seu filho esquerdo, se o elemento for maior que sua informação, ela procura na direita e retorna a busca realizada em seu filho direito.

- A função de remoção recebe um elemento e o encontra na arvore para removê-lo, se a informação do nó for igual ao elemento, ela verifica se possui filhos na esquerda e direita, se possuir um filho na esquerda, mas não na direita, ela retorna seu filho na esquerda e vice-versa, mas se ela possuir um filho na esquerda e um filho na direita, ela procura o maior filho na subárvore esquerda e faz a substituição por ele, iniciando uma remoção recursiva na subárvore esquerda com o elemento que foi substituído como chave para a remoção.

  Se o elemento for menor que sua informação, ela passa a operação de remoção para o seu filho esquerdo, se ele existir.

  Se o elemento for maior que sua informação, ela passa a operação para o seu filho direito.

- A função para encontrar o maior elemento na subárvore esquerda, utilizada pela operação de remoção, verifica recursivamente o próximo nó direito na arvore esquerda, até que a direita seja nula, então, é retornado o último elemento, que será o maior da subárvore esquerda.

- **Funções para o rebalanceamento (Node):**

- Altura: uma função que retorna a altura de um nó verificando seu filho esquerdo e direito, sendo –1 para filhos nulos.

- Checagem de Balanceamento: uma função que verifica o balanceamento de um nó em questão através da subtração da altura de seu nó esquerdo pelo direito.

- Rotação à direita e rotação à esquerda: essas funções realizam a rotação do nó em questão quando é necessário um rebalanceamento, ambas as funções verificam o balanceamento de seus filhos, para verificar se há necessidade de uma rotação dupla

- Rotação à esquerda: checa se o balanceamento do filho direito, se ele existir, é igual a 1, se for igual a 1, rotaciona ele (seu filho direito) para a direita. A rotação do nó em questão define uma nova raiz como seu filho direito, verifica a existência do filho esquerdo da nova raiz e guarda-o em uma variável temporária e então define o filho esquerdo da nova raiz para o nó em questão, define o filho direito do nó em questão para a variável temporária e então retorna a nova raiz.

- Rotação à direita: checa se o balanceamento do filho esquerdo, se ele existir, é igual a 1, se for, rotaciona seu filho esquerdo para a esquerda. A rotação do nó em questão para a direita defina uma nova raiz como seu filho esquerdo, define então uma variável temporária que guarda o filho direito dessa nova raiz, então define o filho direito dessa nova raiz como o nó sendo rotacionado e, então, muda o filho esquerdo do nó em questão para a variável temporária definida anteriormente, então retorna a nova raiz.

- Balancear: uma função que verifica recursivamente os filhos do nó em questão, com o intuito de encontrar o nó desbalanceado e então realizar a rotação adequada para aquele nó.

Árvore AVL:

A arvore AVL, funciona de forma similar a arvore binária de busca, mas ela se diferencia pois em suas inserções e remoções, ela verifica o fator de balanceamento de seus nós, para encontrar um possível desbalanceamento na arvore e então, realiza o rebalanceamento para que suas buscas se tornem mais rápidas e eficientes.

Ela possui, no código, uma função para a checagem de balanceamento, que retorna o fator de balanceamento do nó raiz, e uma função para realizar o balanceamento começando pelo nó raiz.

Comparações:

Utilizei um método para medir o tempo em nanossegundos entre as operações da arvore binária de busca e da arvore AVL com 100, 500, 1000, 10000 e 20000 operações e notei que a arvore AVL em geral tende a ser mais lenta em inserções e remoções, contudo ela apresenta um resultado muito superior em operações de busca quando comparada à arvore binária de busca, que não possui balanceamento.

Devido ao fato de que a arvore AVL está constantemente balanceando a si mesma, é notável que suas inserções e remoções sejam mais lentas, isso se dá em prol da velocidade de busca, que é evidente nos testes a seguir:

Teste com 100 operações com números aleatórios:

```
Tempo da inserção na arvore AVL: 319400 ns
Tempo da inserção na arvore Binária: 376900 ns
Tempo da Busca na arvore AVL: 46400 ns
Tempo da Busca na arvore Binária: 75700 ns
Tempo da Remoção na arvore AVL: 106100 ns
Tempo da Remoção na arvore Binária: 75000 ns
```

Teste com 500 operações:

```
Tempo da inserção na arvore AVL: 1216800 ns
Tempo da inserção na arvore Binária: 590700 ns
Tempo da Busca na arvore AVL: 76100 ns
Tempo da Busca na arvore Binária: 244900 ns
Tempo da Remoção na arvore AVL: 625400 ns
Tempo da Remoção na arvore Binária: 280600 ns
```

É notada uma diferença de velocidade de um pouco mais que 3x com 500 operações

Teste com 1000 operações:

```
Tempo da inserção na arvore AVL: 2692100 ns
Tempo da inserção na arvore Binária: 732600 ns
Tempo da Busca na arvore AVL: 138600 ns
Tempo da Busca na arvore Binária: 480700 ns
Tempo da Remoção na arvore AVL: 1836900 ns
Tempo da Remoção na arvore Binária: 461100 ns
```

Teste com 10000 operações:

```
Tempo da inserção na arvore AVL: 362431100 ns
Tempo da inserção na arvore Binária: 2753800 ns
Tempo da Busca na arvore AVL: 1274700 ns
Tempo da Busca na arvore Binária: 2259100 ns
Tempo da Remoção na arvore AVL: 478356500 ns
Tempo da Remoção na arvore Binária: 2533700 ns
```

Teste com 20000 operações:

```
Tempo da inserção na arvore AVL: 1898641200 ns
Tempo da inserção na arvore Binária: 5221700 ns
Tempo da Busca na arvore AVL: 2860400 ns
Tempo da Busca na arvore Binária: 4048500 ns
Tempo da Remoção na arvore AVL: 2757503500 ns
Tempo da Remoção na arvore Binária: 5723800 ns
```

- Percebemos que, em todos os testes, a arvore AVL apresenta uma velocidade superior a arvore binária de busca.

(Teste adicional com 20000 operações sobre números inseridos em ordem crescente [1, 2, 3, 4, ...]).

```
Tempo da inserção na arvore AVL: 2582282100 ns
Tempo da inserção na arvore Binária: 1013859900 ns
Tempo da Busca na arvore AVL: 2325400 ns
Tempo da Busca na arvore Binária: 875499500 ns
Tempo da Remoção na arvore AVL: 531330100 ns
Tempo da Remoção na arvore Binária: 987200 ns
```

- Vemos aqui, neste último teste, uma diferença enorme entre as buscas das duas arvores, a arvore binária de busca acaba se tornando uma lista de números, enquanto a árvore AVL demonstra grande velocidade nesse caso devido a sua propriedade de balanceamento.

Conclusão:

A arvore AVL, apesar

Em cenários onde é esperado fazer mais buscas do que inserções, certamente será mais vantajoso utilizar a AVL visto que ela é sempre balanceada em sua altura e suas buscas tendem a ser muito mais rápidas que a árvore binária de busca.