Trabalho EBD2 - segunda unidade

Eduardo Marinho

abril 2024

1 Resumo

Esse relatório tem como objetivo explicar a implementação da ferramenta para auxílio do monitoramento do parque das dunas, a qual foi realizada em C++ utilizando uma AVL-tree para armazenar os valores.

2 Introdução

2.1 Cenário

TODO

2.2 Estrutura da árvore

A árvore utilizada foi a AVL-tree. Um tipo de árvore binária balanceada, ou seja, com a garantia de que, para todos os nodos da árvore, a diferença da altura da subarvore da esquerda para a altura da subarvore da direita não é maior que 1, fazendo com que a altura máxima da árvore seja log(n). Para manter essa propriedade a AVL-tree realiza rotações quando um node não estiver balanceado, de modo a torná-lo balanceado.

3 Modelagem

Foi criada uma classe "Dados" para encapsular todos os dados da execução, e implementar as funções auxiliares para ler e escrever os dados no arquivo, assim como as funções para consultar e alterar os dados, dentre elas: "incluir animal" para incluir um novo animal no banco de dados, "remover animal" para remover um animal, "consultar fauna" para consultar as infomações de um animal específico, "inserir monitoramento" para inserir dados de monitoramento para um animal específico, entre outras.

Além disso, foi criado structs auxiliares como DadosDoAnimal e DadosDeMonitoramento, para encapsular os respectivos dados e métodos de entrada e saida dos valores relacionados.

Ao início da execução o arquivo é lido pela class "Dados", onde todos os

dados são armazenados e então, é criado um loop, onde é lido a operação que o usúario deseja realizar e os dados necessários, caso hajam, o loop acaba quando o usúario escolher a opção de acabar a execução, fazendo com que o loop acabe e os dados sejam salvos no mesmo arquivo de entrada.

4 Algoritmos

Considere que o nodo contém a seguinte estrutura:

```
struct nodo {
    int chave;
    int fatorDeBalanceamento;
    nodo filhoDaEsquerda;
    nodo filhoDaDireita;
};
```

4.1 Inserção

4.1.1 Complexidade

Para realizar a inserção em uma AVL tree é necessário encontrar o local a qual se deve inserir o nodo, para isso deve percorrer um número de nodos equivalente a O(log(n)), onde n é o número de nodos da arvore. Ademais, sabe-se que as etapas auxiliares no algoritmo, como rotacionar a arvore para balanceá-la e colocar o nodo no local tem complexidade constantes. Portanto, a complexidade do algoritmo de inserção é O(log(n)).

4.1.2 Pseudocódigo

```
Inserir(raiz, chave)
    if raiz == null {
        raiz = new Nodo
        raiz.chave = chave
        retornar
    }
    if valor < raiz.valor {
        raiz.esquerda = InserirAVL(raiz.esquerda, valor)
        –raiz.fatorDeBalanceamento;
     else if valor > raiz.valor {
        raiz.direita = InserirAVL(raiz.direita, valor)
        ++raiz.fatorDeBalanceamento;
    } else {
        retornar raiz // Valor já esta inserido
    }
    if (raiz.fatorDeBalanceamento == 2) {
        rotaçãoEsquerda(raiz)
```

```
    } else if (raiz.fatorDeBalanceamento == -2)
        rotaçãoDireita(raiz)
    }
}
    RotacaoDireita(raiz) {
    novaRaiz = raiz.filhoDaEsquerda;
    if (raiz == raiz.pai.filhoDaEsquerda) {
        raiz.pai.filhoDaEsquerda = novaRaiz;
    } else {
        raiz.pai.filhoDaDireita = novaRaiz;
    }
    novaRaiz.pai = raiz.pai;
    raiz.pai = novaRaiz;
    raiz.filhoDaEsquerda = novaRaiz.filhoDaDireita;
    novaRaiz.filhoDaDireita = raiz;
    if (raiz.filhoDaEsquerda != null) {
        raiz.filhoDaEsquerda.pai = raiz;
    }
}
    RotacaoEsquerda(raiz) {
    novaRaiz = raiz.filhoDaDireita;
    if (raiz == raiz.pai.filhoDaEsquerda) {
        raiz.pai.filhoDaEsquerda = novaRaiz;
    } else {
        raiz.pai.filhoDaDireita = novaRaiz;
    }
    novaRaiz.pai = raiz.pai;
    raiz.pai = novaRaiz;
    raiz.filhoDaDireita = novaRaiz.filhoDaEsquerda;
    novaRaiz.filhoDaEsquerda = raiz;
    if (raiz.filhoDaDireita != null) {
        raiz.filhoDaDireita.pai = raiz;
    }
}
```

4.2 Procura

4.2.1 Complexidade

Como o máximo de operações realizadas é equivalente a altura da arvore e considerando que a altura maxima da arvore é log(n), tendo em vista que ela é balanceada, a complexidade do algoritmo é O(log(n)).

4.2.2 Pseudocódigo

```
procurar(alvo) {
    atual = raiz;
    while (atual != null) {

        if (atual.chave > alvo) {
            atual = atual.filhoDaEsqueda;
        } else if (atual.chave < alvo) {
            atual = atual.filhoDaDireita;
        } else {
            retorne atual;
        }
    }
    returne atual;
}
```

4.3 Remocão

4.3.1 Complexidade

Como é preciso realizar o algoritmo de procura para achar o nodo a ser removido, e como as operações de remoção e de rotação da árvore tem complexidades constantes o algoritmo de remoção tem complexidade de O(log(n)).

4.3.2 Pseudocódigo

TODO

5 Conclusão

TODO

6 Repositório

https://github.com/enfmarinho/EDB2-Trabalho-1