Relatório de C204

Tema: Pokédex (1)
Alunos: Arthur Openheimer Vilela, Bianca Ribeiro de Souza e Leonardo Luciano de Almeida Maia Júnior

Introdução

- A Pokédex é uma enciclopédia portátil que os treinadores de Pokémon transportam para registrar todas as espécies diferentes de Pokémon que são encontradas durante sua viagem como treinadores.
- Uma árvore binária é uma estrutura de dados que armazena elementos de forma hierárquica. Ela consiste em um conjunto de nós interligados por meio de arestas, daí veio o seu nome.
- Utilizando a estrutura citada, a finalidade do projeto é tornar a busca por um certo Pokémon na Pokédex mais ágil.

Desenvolvimento

O projeto é dividido em:

- Struct: armazena as informações de cada Pokémon.
- Class Pokedex: todas as funções que dizem respeito a Pokédex estão dentro dessa class.
    - inserirPokemon: pega as informações que foram entradas do Pokémon e encaixa na árvore.
    - verificarPokemon: verifica a existência do Pokemon.
    - imprimirEmOrdem: imprime as informações dos Pokémons em ordem alfabética.
    - contarPorTipo: conta a quantidade de Pokémon por tipo.
    - removerPokemon: remove o Pokémon da Pokédex.
    - encontrarMenor: encontra o nó como menor valor.
    - removerTodosPorPokebola: remove todos os pokémons com uma determinada pokebola da Pokédex.
    - imprimirPorPokebola:imprime todos os Pokémons com uma determinada Pokébola.
    - encontrarPokemon: encontra um Pokémon com base no seu número.
    - imprimirTodosEmOrdem: mostra todos os Pokémons em ordem alfabética.
    - imprimirPorTipo: mostra, em ordem alfabética, todos o Pokémons de um determinado tipo.
    - imprimirPorTipoOrdenado: mostra todos os Pokémons ordenados por tipo.
- Dentro da função main nós temos um while para escolher o que desejamos que o código execute, com base no número nós chamamos a função dentro de um switch.

```cpp
#include <iostream>
#include <string>
#include <algorithm>
#include <vector>
#include <locale.h>

using namespace std;

struct Pokemon
{
    string nome;
    string tipo;
    int numero;
    string pokebola;

    Pokemon *esquerda;
    Pokemon *direita;

    Pokemon(const string &nome, const string &tipo, int numero, const string &pokebola)
        : nome(nome), tipo(tipo), numero(numero), pokebola(pokebola), esquerda(NULL), direita(NULL)
    {
    }
};

class Pokedex
{
private:
    Pokemon *raiz;

    void inserirPokemon(Pokemon *&raiz, const Pokemon &pokemon)
    {
        if (raiz == NULL)
        {
            raiz = new Pokemon(pokemon.nome, pokemon.tipo, pokemon.numero, pokemon.pokebola);
        }
        else
        {
            if (pokemon.nome < raiz->nome)
            {
                inserirPokemon(raiz->esquerda, pokemon);
            }
            else
            {
                inserirPokemon(raiz->direita, pokemon);
            }
```

```
        }
    }

    bool verificarPokemon(const Pokemon *raiz, const string &nome)
    {
        if (raiz == NULL)
        {
            return false;
        }
        else if (raiz->nome == nome)
        {
            return true;
        }
        else if (nome < raiz->nome)
        {
            return verificarPokemon(raiz->esquerda, nome);
        }
        else
        {
            return verificarPokemon(raiz->direita, nome);
        }
    }

    void imprimirEmOrdem(const Pokemon *raiz)
    {
        if (raiz != NULL)
        {
            imprimirEmOrdem(raiz->esquerda);
            cout << "Nome: " << raiz->nome << ", Tipo: " << raiz->tipo
<< ", Numero: " << raiz->numero << ", Pokebola: " << raiz->pokebola <<
endl;
            imprimirEmOrdem(raiz->direita);
        }
    }

    int contarPorTipo(const Pokemon *raiz, const string &tipo)
    {
        if (raiz == NULL)
        {
            return 0;
        }

        int count = contarPorTipo(raiz->esquerda, tipo) +
contarPorTipo(raiz->direita, tipo);
        if (raiz->tipo == tipo)
        {
            count++;
        }
```

```
        return count;
    }

    Pokemon *removerPokemon(Pokemon *raiz, const string &pokebola)
    {
        if (raiz == NULL)
        {
            return NULL;
        }

        if (pokebola < raiz->pokebola)
        {
            raiz->esquerda = removerPokemon(raiz->esquerda, pokebola);
        }
        else if (pokebola > raiz->pokebola)
        {
            raiz->direita = removerPokemon(raiz->direita, pokebola);
        }
        else
        {
            if (raiz->esquerda == NULL)
            {
                Pokemon *temp = raiz->direita;
                delete raiz;
                return temp;
            }
            else if (raiz->direita == NULL)
            {
                Pokemon *temp = raiz->esquerda;
                delete raiz;
                return temp;
            }

            Pokemon *temp = encontrarMenor(raiz->direita);
            raiz->nome = temp->nome;
            raiz->tipo = temp->tipo;
            raiz->numero = temp->numero;
            raiz->pokebola = temp->pokebola;

            raiz->direita = removerPokemon(raiz->direita, temp->pokebola);
        }

        return raiz;
    }

    Pokemon *encontrarMenor(Pokemon *raiz)
    {
```

```
        Pokemon *atual = raiz;
        while (atual->esquerda != NULL)
        {
            atual = atual->esquerda;
        }
        return atual;
    }

    Pokemon *removerTodosPorPokebola(Pokemon *raiz, const string &pokebola)
    {
        if (raiz == NULL)
        {
            return NULL;
        }

        raiz->esquerda = removerTodosPorPokebola(raiz->esquerda, pokebola);
        raiz->direita = removerTodosPorPokebola(raiz->direita, pokebola);

        if (raiz->pokebola == pokebola)
        {
            if (raiz->esquerda == NULL)
            {
                Pokemon *temp = raiz->direita;
                delete raiz;
                return temp;
            }
            else if (raiz->direita == NULL)
            {
                Pokemon *temp = raiz->esquerda;
                delete raiz;
                return temp;
            }

            Pokemon *temp = encontrarMenor(raiz->direita);
            raiz->nome = temp->nome;
            raiz->tipo = temp->tipo;
            raiz->numero = temp->numero;
            raiz->pokebola = temp->pokebola;

            raiz->direita = removerPokemon(raiz->direita, temp->pokebola);
        }

        return raiz;
    }
```

```
    void imprimirPorPokebola(const Pokemon *raiz, const string &pokebola)
    {
        if (raiz != NULL)
        {
            imprimirPorPokebola(raiz->esquerda, pokebola);
            if (raiz->pokebola == pokebola)
            {
                cout << "Nome: " << raiz->nome << ", Tipo: " << raiz->tipo << ", Numero: " << raiz->numero << ", Pokebola: " << raiz->pokebola << endl;
            }
            imprimirPorPokebola(raiz->direita, pokebola);
        }
    }

    int encontrarPokemon(Pokemon *raiz, int numero, string &percurso)
    {
        if (raiz == NULL)
        {
            return -1;
        }

        if (raiz->numero == numero)
        {
            return raiz->numero;
        }
        else if (numero < raiz->numero)
        {
            percurso += "EmOrdem|"; // EmOrdem
            int result = encontrarPokemon(raiz->esquerda, numero, percurso);
            if (result != -1)
            {
                return result;
            }

            percurso += "PosOrdem|"; // PosOrdem
            result = encontrarPokemon(raiz->direita, numero, percurso);
            if (result != -1)
            {
                return result;
            }

            percurso += "PreOrdem|"; // PreOrdem
        }
        else
```

```
        {
            percurso += "PreOrdem|"; // PreOrdem
            int result = encontrarPokemon(raiz->direita, numero,
percurso);
            if (result != -1)
            {
                return result;
            }

            percurso += "PosOrdem|"; // PosOrdem
            result = encontrarPokemon(raiz->esquerda, numero,
percurso);
            if (result != -1)
            {
                return result;
            }

            percurso += "EmOrdem|"; // EmOrdem
        }

        return -1;
    }

public:
    Pokedex()
        : raiz(NULL)
    {
    }

    void inserirPokemon(const Pokemon &pokemon)
    {
        inserirPokemon(raiz, pokemon);
    }

    void verificarPokemon(const string &nome)
    {
        if (verificarPokemon(raiz, nome))
        {
            cout << "Pokemon encontrado!" << endl;
        }
        else
        {
            cout << "Pokemon nao encontrado!" << endl;
        }
    }

    int contarPorTipo(const string &tipo)
    {
```

```
        return contarPorTipo(raiz, tipo);
    }

    void removerTodosPorPokebola(const string &pokebola)
    {
        raiz = removerTodosPorPokebola(raiz, pokebola);
    }

    void imprimirPorPokebola(const string &pokebola)
    {
        imprimirPorPokebola(raiz, pokebola);
    }

    int encontrarPokemon(int numero, string &percurso)
    {
        return encontrarPokemon(raiz, numero, percurso);
    }

    void imprimirTodosEmOrdem()
    {
        imprimirEmOrdem(raiz);
    }

    void imprimirPorTipo(const Pokemon *raiz, const string &tipo)
    {
        if (raiz != NULL)
        {
            imprimirPorTipo(raiz->esquerda, tipo);
            if (raiz->tipo == tipo)
            {
                cout << "Nome: " << raiz->nome << ", Tipo: " << raiz->tipo << ", Numero: " << raiz->numero << ", Pokebola: " << raiz->pokebola << endl;
            }
            imprimirPorTipo(raiz->direita, tipo);
        }
    }

    void imprimirPorTipoOrdenado()
    {
        imprimirPorTipo(raiz, "Agua");
        imprimirPorTipo(raiz, "Fogo");
        imprimirPorTipo(raiz, "Normal");
    }
};

int main()
{
```

```
    setlocale(LC_ALL, "");

    Pokedex pokedex;
    string nome, tipo, pokebola;
    int numero;
    string percurso;
    int numeroEncontrado;
    int opcao = -1;

    while (opcao != 0)
    {
        cout << "--- POKEDEX MENU ---" << endl;
        cout << "Selecione uma opcao:" << endl;
        cout << "1. Inserir Pokemon" << endl;
        cout << "2. Imprimir todos os Pokemon em ordem" << endl;
        cout << "3. Imprimir todos os Pokemon ordenados por tipo" <<
endl;
        cout << "4. Verificar se um Pokemon existe" << endl;
        cout << "5. Contar Pokemon de um determinado tipo" << endl;
        cout << "6. Remover todos os Pokemon com uma determinada
pokebola" << endl;
        cout << "7. Imprimir todos os Pokemon com uma determinada
pokebola" << endl;
        cout << "8. Encontrar Pokemon pelo numero (descobrir percurso)"
<< endl;
        cout << "0. Sair" << endl;
        cout << "--- --- --- --- ---" << endl
             << endl;
        cin >> opcao;

        switch (opcao)
        {
        case 1:
            cout << "Digite o nome, tipo, numero e pokebola do
Pokemon:" << endl;
            cin >> nome >> tipo >> numero >> pokebola;
            pokedex.inserirPokemon(Pokemon(nome, tipo, numero,
pokebola));
            break;
        case 2:
            pokedex.imprimirTodosEmOrdem();
            break;
        case 3:
            pokedex.imprimirPorTipoOrdenado();
            break;
        case 4:
```

```
            cout << "Digite o nome do Pokemon que voce deseja
verificar:" << endl;
            cin >> nome;
            pokedex.verificarPokemon(nome);
            break;
        case 5:
            cout << "Digite o tipo de Pokemon que voce deseja contar:"
<< endl;
            cin >> tipo;
            cout << "Quantidade de pokemons do tipo '" << tipo << "': "
<< pokedex.contarPorTipo(tipo) << endl;
            break;
        case 6:
            cout << "Digite o tipo de pokebola que voce deseja
remover:" << endl;
            cin >> pokebola;
            pokedex.removerTodosPorPokebola(pokebola);
            break;
        case 7:
            cout << "Digite o tipo de pokebola para exibir os Pokemon:"
<< endl;
            cin >> pokebola;
            pokedex.imprimirPorPokebola(pokebola);
            break;
        case 8:
            cout << "Digite o nemero do Pokemon que voce deseja
encontrar:" << endl;
            cin >> numero;
            percurso;
            numeroEncontrado = pokedex.encontrarPokemon(numero,
percurso);
            if (numeroEncontrado != -1)
            {
                cout << "Pokemon encontrado com numero " <<
numeroEncontrado << " pelo percurso: |" << percurso << endl;
            }
            else
            {
                cout << "Pokemon nao encontrado." << endl;
            }
            break;
        default:
            cout << "Opcao invalida. Tente novamente." << endl;
            break;
        }
    }
    return 0;
}
```

Casos de teste utilizados

Caso 1

1
Charmander Fogo 4 Pokebola
1
Squirtle Agua 7 Masterball
1
Blastoise Agua 1 Greatball
1
Mudkip Agua 25 Ultraball
1
Eevee Normal 133 Pokebola
2
3
4
Charmander
4
Pidgey
5
Agua
5
Pidgey
6
Pokebola
2
7
Greatball
8
133

Caso 2

1
Vulpix Fogo 4 Pokebola
1
Horsea Agua 7 Masterball
1
Ditto Normal 133 Greatball
1
Wigglytuff Normal 40 Ultraball
2
3
5
Horsea
5
Fogo
6
Greatball
2

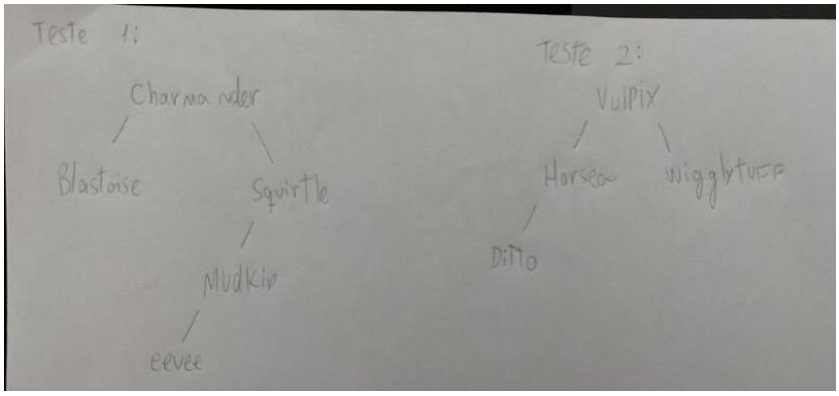
8
133

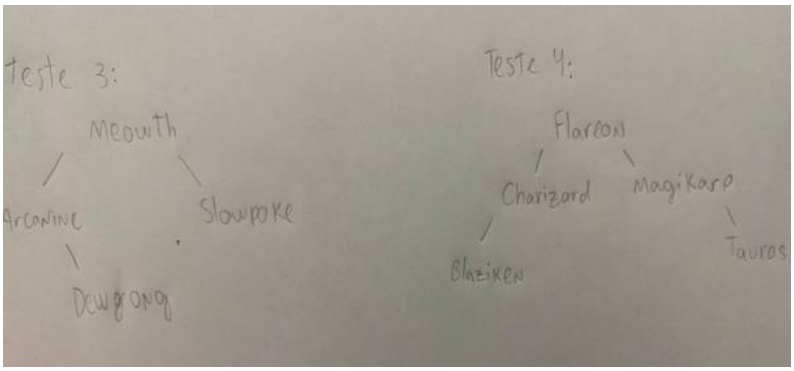
Caso 3

1
Meowth Normal 25 Pokebola
1
Slowpoke Agua 7 Greatball
1
Arcanine Fogo 6 Ultraball
1
Dewgong Agua 87 Masterball
2
3
4
Meowth
6
Masterball
2
8
25

Caso 4

1
Flareon Fogo 1 Greatball
1
Magikarp Agua 7 Pokebola
1
Charizard Fogo 6 Ultraball
1
Tauros Normal 133 Greatball
1
Blaziken Fogo 150 Masterball
2
4
Pidgey
6
Greatball
2
8
150





Análise de complexidade do código
Com base no cenário de teste fornecido, vamos analisar a complexidade de cada operação:

- No cenário dado, há um total de 5 chamadas para a função inserirPokemon, que tem uma complexidade de O(log N) no pior caso. Portanto, a complexidade para essas inserções seria O(log N) * 5.
- Há 1 chamada para a função imprimirTodosEmOrdem, que tem uma complexidade de O(N) no pior caso, onde N é o número de Pokémons na Pokédex.
- Há 1 chamada para a função imprimirPorTipoOrdenado, que internamente chama imprimirPorTipo três vezes. Cada chamada para imprimirPorTipo tem uma complexidade de O(N) no pior caso. Portanto, a complexidade geral para essa operação é O(N) * 3.
- Há 1 chamada para a função verificarPokemon, que tem uma complexidade de O(log N) no pior caso.
- Há 1 chamada para a função contarPorTipo, que tem uma complexidade de O(N) no pior caso.
- Há 1 chamada para a função removerTodosPorPokebola, que tem uma complexidade de O(N) no pior caso.
- Há 1 chamada para a função imprimirPorPokebola, que tem uma complexidade de O(N) no pior caso.
- Há 1 chamada para a função encontrarPokemon, que tem uma complexidade de O(log N) no pior caso.

Com base nessa análise, a complexidade geral do cenário de teste fornecido pode ser resumida da seguinte forma:

O(log N) * 5 + O(N) + O(N) * 3 + O(log N) + O(N) + O(N) + O(N) + O(log N)

O termo dominante nessa expressão é o termo O(N), portanto, a complexidade geral do cenário de teste pode ser aproximada como O(N).

Conclusão

Neste relatório, apresentamos um projeto de Pokédex implementado usando uma estrutura de dados de árvore binária. A Pokédex é uma enciclopédia portátil utilizada por treinadores de Pokémon para registrar as diferentes espécies encontradas durante suas jornadas. O objetivo do projeto era tornar a busca por um Pokémon na Pokédex mais ágil.

Desenvolvemos uma classe Pokedex que contém várias funções relacionadas à Pokédex, como inserir um Pokémon, verificar sua existência, imprimir os Pokémons em ordem alfabética, contar a quantidade de Pokémons por tipo, remover um Pokémon, encontrar o menor valor, remover todos os Pokémons de uma determinada Pokébola, imprimir Pokémons com uma determinada Pokébola, encontrar um Pokémon com base em seu número, imprimir todos os Pokémons em ordem alfabética, imprimir Pokémons de um determinado tipo em ordem alfabética e imprimir Pokémons ordenados por tipo.

Realizamos testes com diferentes casos, e a complexidade do código foi analisada. Com base na análise dos casos de teste, concluímos que a complexidade geral do código é aproximadamente O(N), onde N é o número de Pokémons na Pokédex.

Em resumo, foi implementada uma Pokédex utilizando uma estrutura de dados de árvore binária, proporcionando um desempenho rápido para buscar e manipular os Pokémons registrados.