

UNIVERSIDADE FEDERAL DO PARÁ

INSTITUTO DE CIENCIAS EXATAS E NATURAIS

FACULDADE DE COMPUTAÇÃO

CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

Trabalho Final (Breves Conclusões)

Disciplina: Estrutura de Dados I

Professor: Pedro Alberto Bento Gomes

Estudante: Pedro Paulo Lisboa de Sousa/ 201711140038

Belém/PA

2018

**ESTRUTURA DE DADOS - TRABALHO FINAL**

**O QUE FOI PEDIDO?**

**DATAS**

* Apresentações: 05/07/2018
* Entrega do Relatório: 05/07/2018

**REQUISITOS NÃO FUNCIONAIS**

* Linguagem livre desde que não use bibliotecas de Listas, pilhas, filas, ordenação e arvores.
* Entrega de um relatório em PDF e apresentação do código

**REQUISITOS DO PROJETO**

* Leitura de CSV
* Implementar árvores AVL e RB (apenas inserção, busca e "ordem simétrica")
* Implementar algoritmos de ordenação: Seleção, QuickSort, MergeSort, HeapSort e algum método de ordenação em tempo linear
* Implementar Lista, Pilha e Fila (lista encadeada ou matriz)

**Leitura do CSV**

De acordo com o que foi pedido, foi feito um código em JAVA que lê uma base de dados previamente fornecida e localizada dentro da pasta do projeto.

Para leitura foi utilizado um BuffReader para ler a linha completa, em seguida a linha foi dividida em colunas a cada vírgula encontrada. Tais campos foram adicionados a objetos jogador por meio de um loop utilizando Reflection API.

OBS: Foram identificados 2 registros com valores nulos para o peso, na base de dados. Para contornar o problema, na hora da leitura, esses campos são preenchidos randomicamente.

**Ordenação dos objetos criados a partir do CSV**

Os objetos foram ordenados utilizando o mesmo método, BubbleSort, para ordená-los por ID, Peso e Altura.

* ID: Forma mais rápida de ordenação, pois o ID não apresenta repetições
* Peso: Geralmente demora mais que a ordenação por ID, pois apresenta inúmeras repetições.
* Altura: Similar a ordenação pelo peso, ambas possuem tempo praticamente igual.

**Implementar Algoritmos de Ordenação**

Teste feito utilizando 100000 numeros.

Resultado médio dos tempos:

* **SelectionSort**: Teve tempo de 4333 ms (4s); não testado para 1000000
* **QuickSort**: Tempo de 111 ms; 9262 ms para 1000000
* **MergeSort**: Tempo de 38 ms; 301 ms para 1000000
* **HeapSort**: Tempo de 35 ms; 262 ms para 1000000
* **RadixSort**: Tempo de 25 ms, 163 ms para 1000000

Pude observar que o RadixSort teve tempo menor em praticamente todos os testes (10, 100, 1000, 10000 e 100000 números)

Em relação ao tempo de complexidade esperado obtém-se a seguinte tabela:

|  |  |  |  |
| --- | --- | --- | --- |
| Algoritmo | Complexidade | | |
|  | Melhor caso | Caso médio | Melhor Caso |
| SelectionSort | Ω(n^2) | θ(n^2) | O(n^2) |
| QuickSort | Ω(n log(n)) | θ(n log(n)) | O(n^2) |
| MergeSort | Ω(n log(n)) | θ(n log(n)) | O(n log(n)) |
| HeapSort | Ω(n log(n)) | θ(n log(n)) | O(n log(n)) |
| RadixSort | Ω(nk) | θ(nk) | O(nk) |

Implementação de Lista

Foi implementada lista simples utilizando matriz onde foram criados métodos de inserção e remoção de elementos da lista

Implementação de Fila