Organização TP1

29 de setembro de 2025

1 Introdução

Essa trabalho tem como objetivo aplicação da parte 1 de Estrutura de Dados. O problema trata de objetos que podem ser definidos dentro de certo intervalo onde podemos imaginar a interceção entre eles. Cada objeto apresenta um subintervalor fixo e podemos movelo, modificando assim o subintervalo x e y.

O objetivo do trabalho é avaliar a capacidade do aluno de aplicar o conteitos ensinados dentro da sala de aula, como ordenação de uma vetor, operações como inserção ou remoção nessas estruturas e análise de complexidade do algoritmo. Se espera do discente uma boa implementação para resolução do problema, buscando um algoritimo estável e eficiente.

A implementação tem como base armazenar os objetos da entrada em uma *lista encadeada* , comparamos e ordenamos os objetos desta lista com *mergesort*.

Ao final da introduc¸ ˜ao, procure apresentar como a documentac¸ ˜ao est´a organizada (“A sec¸ ˜ao 2 trada d

2 Implementação

Para um bom entendimento do problema comecei implementando as classes básicas, `objeto`, `movimento` e `cena`. Com as classes prontas, preciso de um *Tipo Abstrado de Dados* para organizar as entradas fornecidas. A escolha da *lista encadeada* é importante para esse problema pois precisamos realizar

3 operações basicas, **search** importante para busca na lista no caso em que iremos mover os objetos, **insert** o básico para armazenar os elementos na lista e **delete** como foi especificado no documento, o objeto ψ que altecede qualqueres outros objetos e tem o intervalo contido no outro, não aparece na cena, ele é ocluso, então deletamos esse objeto da lista.

3 Instruções de compilação e execução

4 Análise de complexidade

A escolha do TAD facilita a análise, pois conhecemos o custo das operações de **search**, **insert** e **delete**, também conhecemos o algoritmo *mergesort* e podemos chegar facilmente a complexidade.

4.1 Lista Encadeada

A insercerção tem complexidade $\mathcal{O}(1)$.

A operação `search` tem complexidade $\Theta(n)$. Usamo esse método para a classe `movimento`.

A operação `delete` tem complexidade $\mathcal{O}(1)$, pois não vou deletar o elemento usando `search`. Durante o método *quicsort* conseguimos identificar que ψ está dentro do intervalo de outro "objeto a sua frente". Nesse caso eu adiciono um método para identificar se essa condição é satisfeita.

4.2 Método Quicksort

Podemos ultilzar este algoritimo em lista encadeadas, para ordenar e comparar os elementos, desta forma identificamos a oclusão entre os elementos que tem profundidade menor que o um obejto ψ.

Esse algoritimo tem complexidade $\mathcal{O}$(n ·logn) para o melhor caso, pior caso e caso médio, logo podemos classificalo com $\Theta(n \cdot logn)$.

Temos um algorimo que independente da entrada, tem tempo de execução concistente.

4.3 Prova complexidade (tempo de execução)

Somamos todas as complexidades temos:

$$\mathcal{O}(1) + \Theta(n) + \mathcal{O}(1) + \Theta(n \cdot logn) = \Theta(n \cdot logn)$$

Logo, nosso algoritimo tem complexidade $\Theta(n \cdot logn)$.

4.4 Prova complexidade (espaço)

A lista armazena n elementos e o quicksorte precisa de um espaço $\mathcal{O}$(n)...

5 Seções adicionais

6 Conclusão