DOCUMENTAÇÃO – TRABALHO PRÁTICO 01

Nome: Lucca Silva Medeiros

Matrícula: 2019054773

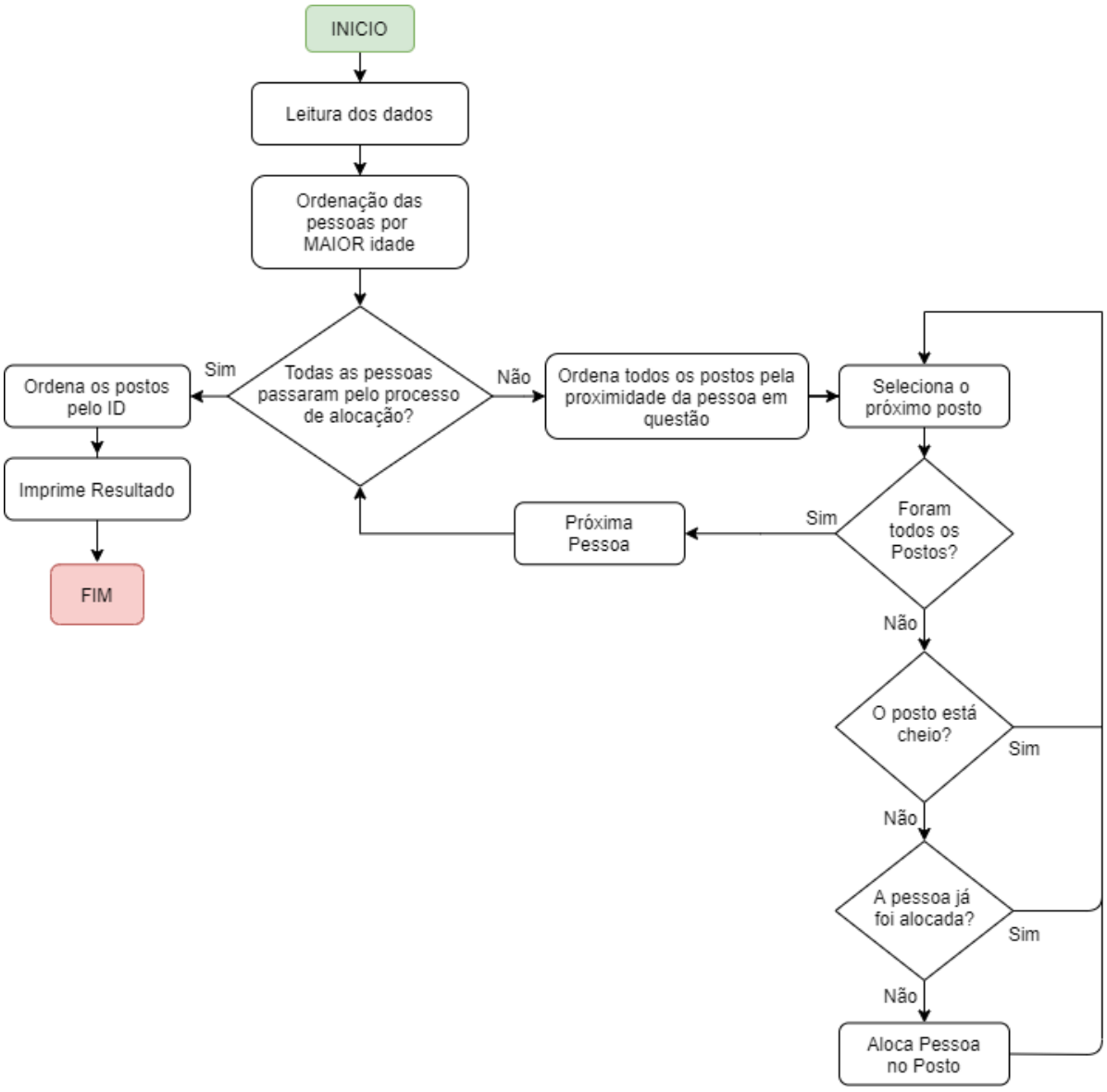
Disciplina: Algoritmos I

Introdução:

O trabalho prático consiste em desenvolver um programa, capaz de processar um conjunto numérico de entrada, que representa informações de postos de saúde e pessoas a serem vacinadas, e dar como saída a melhor alocação possível, tendo em vista fatores de prioridade nas idades das pessoas pelos postos mais próximos geograficamente.

Modelagem computacional do problema:

Para resolver o problema proposto a modelagem utilizada foi a descrita pelo fluxograma abaixo:



Estruturas de Dados:

Foram utilizadas duas estruturas de dados para solução do problema, são elas:

Person: representa uma pessoa que deve ser alocada para vacinação.

Atributos: Number, Age, xCord, yCord e HC.

Obs: O atributo HC representa o Posto de Alocação da pessoa, sendo o -1 a não alocação.

Métodos: Construtor, set/get Number, set/get Age, set/get x and y Cords e set/get HC.

HealthCenter: representa o posto de vacinação.

Atributos: Number, Total Capacity, Capacity, xCord, yCord e Priority.

Obs: Priority é um array de inteiros que armazena os ID's das pessoas que estão alocadas naquele posto.

Métodos: Construtor, set/get Number, set/get TotalCapacity, set/get Capacity, set/get x and y Cords, set Alocation, get Distance e deQueue Priority.

Obs: - set Alocation é o método responsável por alocar uma pessoa no posto de vacinação.

- get Distance é o método responsável por calcular a distância euclidiana entre um posto e uma pessoa.

- deQueue Priority é o método responsável por retornar uma pessoa na fila de alocação.

Algoritmos utilizados:

Os algoritmos utilizados para solução do problema podem ser classificados em dois tipos, sendo eles algoritmos de ordenação e algoritmo de casamento estável.

Para ordenar as pessoas pela idade foi implementado o Mergesort, haja vista que nessa situação é necessário um algoritmo de ordenação estável, em que as pessoas com o menor ID devem ter prioridade caso a idade seja a mesma. O Mergesort também é utilizado para ordenar os postos pelo ID para imprimir a saída de forma correta.

Já na ordenação dos postos pela distância foi utilizado o Quicksort, porém com uma modificação que obriga os postos com o menor ID terem prioridade em caso de empate na distância. Foi utilizado esse algoritmo por causa do baixo consumo de memória em relação ao Mergesort, pois ao rodar entradas com mais de 10 mil pessoas o Mergesort resultava problemas de má alocação de memória, o que foi solucionado com o Quicksort.

O algoritmo que realiza o casamento estável entre os postos e as pessoas foi uma adaptação do algoritmo de Gale–Shapley, pois na situação do problema a lista de prioridade de todos os postos é a mesma (sempre a pessoa mais velha vai ter prioridade), logo essa característica foi explorada, visando simplificar a solução.

O resultado implementado segue a lógica do pseudo-código abaixo:

Ordena as pessoas da mais velha para a mais nova;
Para todas as pessoas:
    Faz a lista de prioridade da pessoa com base a proximidade dos postos;
    Para todos os postos, seguindo a lista de prioridade da pessoa:
        O posto em questão tem vaga e a pessoa não está alocada?
            Caso sim > aloca a pessoa no posto em questão;
            Caso não > passa pro próximo posto;

Essa implementação foi a escolhida tendo em vista simplificar a solução. Como as pessoas mais velhas sempre vão ter prioridade de escolha, ao ordenar o vetor de pessoas basta calcular a lista de prioridade de cada pessoa e ir alocando diretamente seguindo sua fila de preferência, pois como as mais velhas sempre serão alocadas primeiro, elas sempre vão ficar com o posto disponível mais próximo geograficamente possível, garantindo a estabilidade do casamento.

Complexidade Assintótica de Tempo:

- A complexidade da leitura dos dados dos postos é **O(n)**, sendo n o número de postos.
- A complexidade da leitura dos dados das pessoas é **O(m)**, sendo m o número de pessoas.
- A complexidade do algoritmo de ordenação mergeSort é **O(n log n).**
- A complexidade do algoritmo de ordenação quickSort também é **O(n log n).**
- A complexidade da parte responsável pela impressão da saída é **O(n+m).**

A complexidade assintótica de tempo geral do programa é igual a maior complexidade dentre as estruturas do projeto, que no caso, é a parte do algoritmo responsável pelo casamento estável. Durante essa parte, o programa executa para cada pessoa um mergeSort responsável por ordenar o vetor de postos e depois mais um loop para cada posto existente nesse vetor, para conseguir alocar a pessoa.

Haja vista essa situação e que **m** é o número total de pessoas e **n** o número total de postos dados na entrada: A complexidade assintótica de tempo geral do programa é igual a **O(m*(n log n + n)).**