# Trabalho Prático 1: Controle Vacinação Sars-CoV-2

**Ricardo Avelar**

Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brazil
ricardodiasmode@hotmail.com

## 1. Introdução

Esta documentação lida com o problema de desenvolver um script com a funcionalidade de realizar a distribuição de pessoas de uma cidade entre postos de saúde, buscando manter as regras de prioridade propostas pela Secretaria de Saúde, tendo como prioridade a idade e a distância da pessoa até o posto, respeitando a preferência dos idosos.

Essa documentação tem como objetivo facilitar o entendimento do script construído com a finalidade de completar essa tarefa. Aqui serão explicados todos os conceitos sobre as funções do script e como foram tomadas as decisões para se resolver os problemas necessários.

Para resolver o problema citado, foi seguida uma abordagem de casamento estável, buscando alcançar uma situação em que a premissa do casamento estável seja satisfeita.

A seção 2 trata de uma visão sobre a implementação deste problema, tratando as decisões para a resolução dos problemas. Já na seção 3 é apresentado um sucinto passo-a-passo de como se executar o programa. A seção 4 faz a análise de complexidade do script e a seção 5 é a conclusão e visão final do projeto.

## 2. Implementação

O código desenvolvido para solução desse problema está organizado em:

**Main.cpp**: Aqui ocorre a inicialização do programa. Em um primeiro 'while', são lidas as informações necessárias para seu funcionamento. Após a leitura dos dados, um segundo 'while', que é um algoritmo de casamento baseado no algoritmo de Gale-Shapley é executado e o casamento é feito. Depois disso é impresso o resultado do casamento no terminal.

**Posto.h**: Essa é a classe que representa o posto de saúde. Aqui temos variáveis que representam cada característica do posto, sendo válido descrever uma em especial, que é definida como 'Pessoa** PessoasDentro = nullptr;'. Essa variável é um array da classe Pessoa que cada posição representa a pessoa que está dentro do posto.

**Posto.cpp**: Contém o construtor da classe Posto.h.

**Pessoa.h**: Essa é a classe que representa as pessoas. Aqui temos variáveis que representam cada característica das pessoas, além de duas em especial. A variável chamada 'Propostas' é um array que cada posição representa um posto, e assume verdadeiro ou falso se já propôs ou não para um posto, respectivamente. Outra variável se chama 'MeuPosto', que representa o posto em que a pessoa está.

**Pessoa.cpp**: Contém construtores da classe Pessoa.h.

**Matching.h**: Essa é a classe onde contém todos os métodos que são utilizados no algoritmo do casamento estável. Abaixo há uma breve descrição dos mais e não tão intuitivos.

**PrimeiroPostoDePreferenciaNaoProposto**: Esse método retorna o posto de preferência da pessoa passada por parâmetro, com a condição de que a pessoa ainda

não deve ter proposto para esse posto.

**TrocarPessoaDePosto**: Esse método recebe como parâmetro uma pessoa, nomeado 'PessoaRef', que é a pessoa que está entrando no posto em questão, e por meio de um ponteiro retorna a pessoa que está saindo - 'PessoaSaindoDoPosto'.

**TodosEstaoEmPostos**: Esse método retorna '-1' se todas as pessoas estiverem dentro de um posto, e caso contrário retorna o index da pessoa que não está em um posto.

**ChecarPreferenciaPosto**: Esse método checa o posto em questão prefere uma pessoa p1 a qualquer pessoa dentro dele. Se prefere p1, retorna true. Se prefere todas as pessoas dentro, retorna false.

**Configuração utilizada para testar o programa**:

- Sistema Operacional Windows 10;
- Linguagens C++;
- Compilador Mingw;
- Processador Ryzen 5 3600x e 16Gb RAM;

## 3. Instruções de compilação e execução

- Acesse o diretório do projeto (2019054960_RicardoDiasAvelar);
- Utilizando um terminal, execute o arquivo [ Makefile ] utilizando o seguinte comando: < make >;
- Com esse comando, devem ser criados os arquivos .o na pasta obj e o arquivo .exe na pasta bin;
- Proceda Utilizando o terminal para acessar o diretório bin, posicione os arquivos entrada.txt em qualquer pasta que queira utilizar e execute o arquivo [ tp01.exe ] utilizando o seguinte comando: < "tp01 < caminho/para/entrada.txt"> ;
- Dessa forma será impresso no prompt o relatório do processo;

## 4. Análise de complexidade

No main.cpp temos um while que enquanto houver linhas no arquivo a serem lidas ocorre o loop. Esse primeiro while tem complexidade de tempo O(2+n/3), pois em cada iteração do loop, com exceção de duas iterações, são lidas 2 linhas extras(o que explica o n/3). Ainda no primeiro while, a complexidade de espaço é Θ(n+m), pois há a alocação de n postos e m pessoas. Há um segundo while, que utiliza diversas funções do Matching.cpp, e portanto teremos a complexidade dele após calculá-las abaixo. No final do programa, temos um for loop dentro de outro for loop, um iterando de 0 a n e outro iterando de 0 a m, portanto a complexidade de tempo é Θ(n+m).

**static Posto* PrimeiroPostoDePreferenciaNaoProposto - complexidade de tempo:** essa função realiza operações constantes, em tempo O(1). Além disso, há um for loop que itera de 0 a n, com n sendo a quantidade de postos. Assim, sua complexidade é Θ(n).

**static Posto* PrimeiroPostoDePreferenciaNaoProposto - complexidade de espaço:** essa função realiza todas as operações considerando estruturas auxiliares unitárias O(1) e não aloca nenhum array, portanto sua complexidade de espaço é O(1).

**static void ColocarPessoaNoPosto - complexidade de tempo:** essa função realiza operações constantes, em tempo O(1). Além disso, há um for loop que itera de 0 a n, com n sendo a quantidade de pessoas dentro do posto. No entanto, o loop pode acabar antes de

iterar por completo. Assim, sua complexidade é O(n).

**static void ColocarPessoaNoPosto - complexidade de espaço:** essa função realiza todas as operações considerando estruturas auxiliares unitárias O(1) e não aloca nenhum array, portanto sua complexidade de espaço é O(1).

**static void TrocarPessoaDePosto - complexidade de tempo:** essa função realiza operações constantes, em tempo O(1). Além disso, há um for loop que itera de 0 a n, com n sendo a quantidade de pessoas dentro do posto. No entanto, o loop pode acabar antes de iterar por completo. Assim, sua complexidade é O(n).

**static void TrocarPessoaDePosto - complexidade de espaço:** essa função realiza todas as operações considerando estruturas auxiliares unitárias O(1) e não aloca nenhum array, portanto sua complexidade de espaço é O(1).

**static int TodosEstaoEmPostos - complexidade de tempo:** essa função realiza operações constantes, em tempo O(1). Além disso, há um for loop que itera de 0 a n, com n sendo a quantidade de pessoas existentes. Há um if que contém outro for loop dentro desse primeiro, portanto há um pior, médio e melhor caso. No pior caso, quando todas as pessoas não estão em postos, mas todas as pessoas já propuseram para todos os postos, temos a complexidade $\Theta(n*m)$, onde n é a quantidade total de pessoas e m é a quantidade total de postos. No melhor caso, temos a situação de que todas as pessoas já tem postos, portanto teremos a complexidade $\Theta(n)$. No caso médio, portanto, temos $\Theta(((n*m)+n)/2)$.

**static int TodosEstaoEmPostos - complexidade de espaço:** essa função realiza todas as operações considerando estruturas auxiliares unitárias O(1) e não aloca nenhum array, portanto sua complexidade de espaço é O(1).

**static bool ChecarPreferenciaPosto - complexidade de tempo:** essa função realiza operações constantes, em tempo O(1). Além disso, há um for loop que itera de 0 a n, com n sendo a quantidade de pessoas dentro do posto. Assim, sua complexidade é $\Theta(n)$.

**static bool ChecarPreferenciaPosto - complexidade de espaço:** essa função realiza todas as operações considerando estruturas auxiliares unitárias O(1) e não aloca nenhum array, portanto sua complexidade de espaço é O(1).

## 5. Conclusão

Este trabalho lidou com o problema de distribuição de pessoas entre postos para a vacinação, no qual a abordagem utilizada para resolução foi o desenvolvimento de um script que faz a leitura de um arquivo txt com os dados das pessoas e dos postos buscando fazer os casamentos necessários para se tornar estável.

Com a solução adotada, pode-se verificar que a tentativa utilizada foi a criação de três classes, Posto, Pessoa e Matching, e cada classe desempenha um papel especial como vimos no desenvolvimento da documentação.

Por meio da resolução desse trabalho, foi possível praticar os conceitos relacionados a casamentos estáveis, com a necessidade de associação das pessoas com os postos levando em consideração distância e idade, e também a resolução de problemas relacionados à alocação de ponteiros. Pela estratégia de resolução escolhida, além do algoritmo de casamento estável, pratiquei principalmente a lógica de alocação de ponteiros e sua utilização em funções externas.

## Referências

Slides virtuais da disciplina de Algoritmos 1. Disponibilizado via moodle. Departamento de Cieˆncia da Computação. Universidade Federal de Minas Gerais. Belo Horizonte.