Documentação do Projeto: Postos da Vacina

Augusto Carvalho Porto Pereira

Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG) Belo Horizonte – MG – Brazil

1. Introdução

O objetivo deste trabalho é criar um sistema que receberá dados referentes aos postos de vacinação de uma capital brasileira fictícia, assim como informações sobre os moradores da região, e será capaz de relacionar os postos de vacinação às pessoas na fila de prioridade da vacina através de um algoritmo de casamento estável. A ordem de prioridade é definida com base na idade do pretendente à vacina e na distância dele aos centros de vacinação, garantindo também que nenhum centro esteja sobrecarregado para além de sua capacidade, assim como não haja nenhum posto de vacinação vazio enquanto existam pessoas que ainda não tomaram a vacina.

Nas próximas seções, serão descritas a modelagem computacional do problema bem como a estrutura de dados e os algoritmos utilizados para a solução, seguidas da justificativa para o seu uso. Além disso, será apresentado a análise de complexidade de tempo assintótica da solução total e de trechos importantes do código fonte.

2. Modelagem Computacional

A estratégia utilizada no desenho da solução foi construir a estrutura utilizada para desenhar os casamentos à medida que os parâmetros de entrada são lidos do arquivo texto. Para isso, a criação desses dados no programa obedece a ordem que segue: A cada tupla de informações relacionada a um posto de vacinação é criado um registro na lista de postos. Em seguida, para cada pessoa no arquivo de entrada, é criado um registro com as suas informações e sua lista de preferência de postos baseada na distância entre a pessoa e cada um dos postos da lista anterior.

Por fim, a lista de pessoas é ordenada por idade decrescente e a regra de desempate considerando a ordem de inserção dos registros é respeitada.

Na sequência, após a criação no código de todas as listas necessárias para realizar a relação entre postos e pessoas, o algoritmo de casamento estável que será explicado na seção a seguir é implementado e os resultados da relação são preenchidos nos registros inseridos de cada pessoa e posto. Por fim, o resultado é impresso na tela e o programa se encerra.

3. Algoritmos e Estruturas de Dados

Buscando organizar o código e representar da melhor forma os conceitos reais em objetos, foram criadas as classes Posto e Pessoa que possuem os componentes referentes aos dados recebidos nos arquivos de entrada além de suas listas de prioridade e suas informações finais de quais os selecionados para vacinar no posto ou qual o posto de vacinação da pessoa, se existir. Existe também uma estrutura auxiliar utilizada para calcular a distância entre as pessoas e cada um dos postos chamada de *Distancia_Posto_Pessoa*.

Para as ordenações da lista de prioridade de pessoas para vacinação e da lista de postos mais próximos de cada pessoa, foi utilizada a função *stable_sort* da biblioteca *algorithm*, disponibilizada como ``*standard*'' e sendo portando padrão da linguagem. O uso de uma biblioteca já pronta se justifica pela menor complexidade em sua construção e maior garantia de bom desempenho visto o alto nível de revisão e controle de complexidade que bibliotecas muito utilizadas recebem. É importante reforçar também que foi utilizada a variante estável da função de ordenação pois era essencial pela regra de execução do sistema que a ordem de inserção dos registros fosse respeitada.

O algoritmo principal do programa, responsável por relacionar os postos e as pessoas, é uma variação do algoritmo de Gale-Shapley que funciona como descrito no pseudocódigo abaixo. É importante destacar que, pela mesma justificativa pela qual o algoritmo apresentado em sala apresenta a solução ótima para os hospitais, esta versão propõe uma solução ótima no ponto de vista das pessoas candidatas à vacinação.

ENQUANTO há vagas nos postos e pessoas desalocadas
    pe <- pessoa na lista de vacinação

```
        PARA CADA po <- posto de vacinação
                SE (pe não está alocado)
                        Aloca pe em po
                SENÃO, SE (pe prefere po ao posto atual po')
                        SE (vaga em po)
                                Desaloca pe de po' e aloca pe em po
                        SENÃO, SE (preferência de pe sobre algum pe' alocado)
                                Desaloca pe' de po, desaloca pe de po' e aloca pe em po
                SENÃO
                        pe não consegue vaga em po
FIM ALGORITMO
```

4. Complexidade assintótica de tempo

Seccionando o código pelas responsabilidades de cada um dos trechos do código, conseguimos descrever a complexidade assintótica de tempo das partes principais do sistema, bem como sua complexidade total. Funções de menor relevância ou pouquíssima complexidade de tempo serão desprezadas nesse espaço para fins de síntese.

O processo de leitura e inicialização dos postos é executado por um loop executado *n* vezes, sendo *n* o número de postos apresentado na primeira linha do arquivo. Portanto, a complexidade desse trecho é *O(n)*. Já a leitura e inicialização das pessoas ocorre de maneira diferente, pois para cada pessoa informada é realizado um loop que ocorre *n* vezes para o cálculo da distância dos postos e, após encontrar as medidas, é executada a função *stable_sort*, que possui complexidade *n log n.* Como essas etapas ocorrem para cada uma das *m* pessoas, a complexidade de tempo da inicialização das instâncias de uma pessoa é dada m * n log n, ou, para sintetizar, $O(n^2 \log n)$.

Após a leitura dos dados de pessoas, é realizada uma ordenação da lista de pessoas para respeitar a ordem de prioridade da fila de vacinação, executando o algoritmo *stable_sort* novamente, porém fora de qualquer laço, representando somente a complexidade *O(n log n)*. Por fim, a última seção apresentada no código é a execução do algoritmo Gale-Shapley adaptado. Com base na complexidade estudada no algoritmo visto em sala de aula e visto que os dados foram pré-processados criando as listas de prioridade, o número máximo de propostas que

podem ser realizadas até que chegue a solução final é m*n e, portando, a execução desse trecho é *O(n²).*

Visto que todos esses blocos são executados em sequência na rotina *main*, a complexidade de tempo de todo o sistema é definida pela maior complexidade de tempo dos blocos sequenciais. Portanto, a complexidade de tempo de todo o sistema é ditada pelo tempo gasto para a inicialização das instâncias de pessoas que será sempre ***O(n² log n).***