Trabalho Prático 1
Problema da Lista de Prioridade das Vacinas

Jean Lucas Almeida Mota

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte - MG - Brasil

jeanlk@ufmg.br

1. Introdução

O problema a ser enfrentado possui natureza fictícia com base na atual pandemia da covid-19. Neste problema, dado os postos de saúde e as pessoas que moram na região, deve-se alocar as pessoas nos postos. Sob as seguintes condições:

1. As pessoas com maior idade possuem preferência.
2. Os postos possuem capacidade máxima diária.
3. As pessoas devem ser alocadas para postos próximos a elas.
4. se uma pessoa não for alocada num posto mais próximo a ela, não deve haver naquele posto vagas não alocadas ou alocações de pessoas mais jovens que ela.
5. Caso exista um empate, a condição de desempate será o id da pessoa ou do posto.

2. Implementação

O programa foi desenvolvido na linguagem C++, compilado pelo compilador G++.

2.1 Estrutura de dados

A implementação do programa teve como base o problema do casamento estável. Dessa forma a modelagem foi feita a fim de facilitar a implementação do algoritmo de Gale-Shapley. O qual foi modificado para se adequar ao problema proposto. Pseudocódigo:

Par instável: uma pessoa p1 é alocada em um posto b1, mas há um posto b2 mais próximo que está alocando uma pessoa p2 mais nova que p1. Ou seja, b2 prefere p1 e p1 prefere b2 aos pares aos quais foram alocados.

INCIALIZE M como vazio.
WHILE (existir um posto com vaga E houver par instável)
    Pegue uma pessoa p1 que ainda não tem par e ainda não propôs pra todos os postos
    IF (o posto b1 tiver vaga)
        P1 é alocado para b1
    ELSE IF (b1 prefere p1 ao seu atual par p2)
        P1 é alocado para b1

```
            P2 fica sem par
        ELSE
            B1 rejeita p1
RETURN O casamento estável M
```

Assim foi usada a estrutura Lista Encadeada, a fim de otimizar o algoritmo e facilitar a implementação.

Observação: Está sendo levado em conta o casamento estável do ponto de vista das pessoas.

2.2 Classes

A fim de modularizar a implementação, o TAD lista encadeada foi implementado de forma independente e uma classe de mais alto nível faz uso dela.

Classe ListaEncadeada: É a implementação do TAD lista encadeada. Além dos métodos básicos de Inserção, Remoção, Impressão, Pesquisa. Foi implementado os métodos AndaProposta(), ReiniciaProposta(), ImprimeProposta(), que são utilizados para manipular o próximo posto a se propor na ordem de preferência de uma pessoa.

Classe ListaPreferencia: É a classe um nível a cima de ListaEncadeada, faz uso das listas encadeadas para interpretar as listas de preferências das pessoas e postos.

Classe Pessoa: É a classe que modela as informações referentes as pessoas, como por exemplo: idade, posição geográfica.

Classe Posto: É a classe que modela as informações referentes aos postos, como por exemplo: capacidade, posição geográfica.

Classe SistemaPrioridade: É a classe de mais alto nível, que faz uso de todas as classes anteriores. Possui os métodos:

- Leitura(), faz a leitura dos dados via linha de comando;
- CalculaDistancia(p1,p2), faz o cálculo da distancia entra uma pessoa e um posto;
- ConstruirPreferencias(), baseado nas distancias entre pessoas e postos, constrói as listas de preferencias das pessoas em relação aos postos e também constrói a lista de preferencia dos postos em relação as pessoas usando a idade delas;
- Matching(), é p método responsável por implementar o algoritmo de Gale-Shapley;

- Imprimir(), faz a impressão na tela do resultado obtido no método Matching().

Existem alguns outros métodos auxiliares usados para melhor visualização do código também para ordenação. Os métodos de ordenação usam uma versão do algoritmo de ordenação por Inserção. O método de inserção foi escolhido por sua boa velocidade e baixa complexidade de implementação.

3. Análise de Complexidade

3.1 Tempo

A entrada do sistema pode ser interpretada da seguinte forma: Sendo **n** o numero de postos de vacinação e **m** o numero de pessoas.

O sistema é pode ser dividido em 4 métodos principais de alto nível, logo somando a complexidade de tempo deles, teremos a complexidade total do sistema. Os métodos são:

Leitura(): Ao inicializar percorremos todas as pessoas e todos os postos. Temos o custo de inicialização **O(n+m)**.
ConstruirPreferencias(): Custo para montar as listas de preferencias tanto das pessoas quanto dos postos, logo **O(n*m) + O(m) = O(n*m)**
Métodos de ordenação: Sempre é usado o algoritmo de inserção, logo temos para o **Melhor caso: O(n)** e **Pior caso: O(n^2)**.
Matching(): Implementação do algoritmo de Gale-Shapley, temos: **O(m)+O(n)+O(n^2*m^2) = O(n^2*m^2)**.

Dessa forma, a complexidade total do sistema é dada por:
Melhor caso: **Max(O(n+m), O(n*m), O(n), O(n^2*m^2)) = O(n^2*m^2)**
Pior caso: **Max(O(n+m), O(n*m), O(n^2), O(n^2*m^2)) = O(n^2*m^2)**

3.2 Espaço

A quantidade de espaço ocupado pro sistema é o custo de armazenar as pessoas, os postos e suas listas de preferência, **O(n+m)**.

4. Conclusão

Após a leitura do problema proposto, nota-se o uso essencial da modelagem por casamento estável, o qual proporciona facilidade e velocidade na hora da implementação. Nota-se também que o algoritmo de Gale-Shapley possui complexidade O(n^2), no entanto, na hora de adapta-lo para o problema proposto, o mesmo ficou com uma complexidade muito alta O(n^2*m^2). Talvez com um estudo mais detalhado da implementação seja possível diminuir a complexidade do sistema para O(n^2).

Ao finalizar o sistema, ficou evidente a necessidade de se conhecer bem as estruturas de dados básicas, como filas, pilhas, lista, arvores e grafos. Uma vez que, o bom uso delas pode melhorar muito a implementação e o tempo de execução de um sistema.

5. Execução
O programa foi desenvolvido no Windows 10 e testado no Windows 10 e no Linux.

Para execução no Windows, entre no diretório onde está o Makefile e execute os seguintes comandos:
mingw32-make
tp01

Para execução no Linux entre no diretório onde está o Makefile e execute os seguintes comandos:
make
./tp01

Observação: Os testes deixados pelo professor foram testados usando o comando:
./tp01 < casosteste10/ct00-input.txt

References
Chaimowicz, Luiz e Prates, Raquel. Slide Lista Arranjo e Lista Encadeada de Pesquisa (2020). Disciplina Estrutura de Dados DCC205 UFMG.

Kleinberg, Jon. Livro Algorithm Design.

Post sobre a estrutura map. Disponível em https://www.delftstack.com/pt/howto/cpp/how-toiterate-over-map-in-cpp/ 01 nov. 2017. Acesso em: 19 jan. 2021.

TAD-ORDENACAO. GitHub: Jean Lucas Almeida Mota, 10 set. 2020. Disponível em: https://github.com/jeanhardzz/TAD-Ordenacao . Acesso em: 29 jun. 2021.