Trabalho Prático 1

Breno Cláudio de Sena Pimenta

[1]Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
DCC216 - Algoritmos I

1. Introdução

O trabalho prático consiste em realizar o casamento entre postos de vacinação e pessoas de uma dada comunidade. Os postos devem alocar as pessoas com maior idade e mais próximas. Se uma pessoa não for alocada num posto mais próximo a ela, isso significa que não deve haver naquele posto vagas não-alocadas ou alocações de pessoas mais jovens que ela.

A alocação deve ser simultaneamente a melhor possível para todas as pessoas, ou seja, o casamento deve ser ótimo do ponto de vista das pessoas. Sendo assim, a pessoa que tiver a maior idade sempre conseguirá ser atendida no posto mais perto. Para isso, foi realizada uma modelagem computacional e implementado um programa em C++ que será descrito a seguir que soluciona o dado problema. Além disso será apresentada a análise de complexidade de tempo assintótica do programa desenvolvido.

2. Modelagem Computacional

Para modelar a solução inicialmente foi estudado o algoritmo básico de Gale-Shapley. Para facilitar a implementação foram adotadas os seguintes passos para realizar o casamento estável entre postos de saúde e pessoas:

1. **Passo 1 - Ler dados da entrada:** o programa inicialmente deve ler as localidades geográficas e as idades das pessoas cadastradas para vacinação. Também, deve ler as localidades geográficas e as capacidades dos postos.
2. **Passo 2 - Obter tabela de preferências dos postos:** o programa deve gerar a tabela de preferência dos postos, ou seja, a lista de pessoas em ordem decrescente de idade e crescente de id (posição em que aparece na lista).
3. **Passo 3 - Obter tabela de preferências das pessoas:** o programa deve montar a tabela de preferência das pessoas, ou seja, uma matriz nxm em que n é o número de pessoas e m é o número de rankings necessários, ou seja, a quantidade total de postos de saúde disponíveis. Assim, se na posição [2][0] da matriz de preferência das pessoas existe o número 3 significa que o posto 3 é o posto mais preferido da pessoa 2. Essa matriz é formada a partir do calculo da distancia euclidiana de uma pessoa a cada posto da cidade.
4. **Passo 4 - Gerar a alocação de pessoas:** com as duas tabelas anteriores conhecidas, basta alocar a pessoa com mais idade ao posto de saúde com vaga mais próximo. Assim, será gerada uma nova tabela em que cada linha representa um posto de saúde e as colunas as pessoas alocadas para tal posto.
5. **Passo 5 - Imprimir na tela os resultados:** para finalizar, o programa deve imprimir a matriz gerada no passo anterior na tela para o usuário.

2.1. Estruturas de dados

Antes de implementar o algoritmo desenvolvido nos passos anteriores, foram criados três classes: Posto, Pessoa e CasamentoEstavel.

2.1.1. Classe Pessoa

A classe Pessoa representa uma pessoa que está se candidatando para ser vacinada. Ela possui os atributos idade e coordenadas x e y de sua residencia. Ela possui métodos para calcular a sua distancia a qualquer outro ponto conhecido, sabe informar se já está alocada para ser vacinada em um posto de saúde com vaga e métodos set e get para alterar seus atributos privados.

2.1.2. Classe Posto

A classe Posto representa um posto de saúde existente na cidade. Ela possui os atributos vagas e coordenadas x e y de sua residencia. Ela possui métodos set e get para alterar seus atributos privados.

2.1.3. Classe CasamentoEstavel

A classe CasamentoEstavel realiza a alocação de pessoas aos postos de saúde com vagas, de acordo com os critérios de idade e ids. Como essa classe é mais complexa que as outras, seus métodos e atributos serão explicados com mais detalhes a seguir. Os métodos da classe CasamentoEstavel são os seguintes:

- **Construtor:** o construtor da classe apenas define alguns atributos iniciais da classe como nulos.
- **Destrutor:** o destrutor da classe libera toda memória alocada dinamicamente para os atributos da classe.
- **static bool comparadorIdadesIndices(const pair<int,int>&pessoa1, const pair<int,int>&pessoa2):** O operador de comparação foi redefinido para utilizar a função sort da biblioteca STL para ordenar um vector formador por um par de inteiros (idade e id) na função ranquearPreferenciaDosPostos().
- **void lerDadosEntrada():** essa função lê os dados da entrada e insere no vector de pessoas e no vector de postos os dados lidos, por meio das funções auxiliares void inserirPessoa(int idade, int x, int y) e void inserirPosto(int vagas, int x, int y).
- **void ranquearPreferenciaDasPessoas():** a função calcula para cada pessoa cadastrada a distancia entre ela e cada cada posto existente. A função utiliza o vector de pessoas e o vector de postos que foram gerados pelos dados da entrada e retorna uma matriz nxm em que n é o número de pessoas e m é o número de rankings necessários, ou seja, a quantidade total de postos de saúde disponíveis. Cada linha da matriz representa o ranking de preferencia de uma pessoa, sendo a coluna 0 o posto preferido, e mais perto, e a coluna $m - 1$ o posto menos preferido, e consequentemente, o mais longe.

- **void ranquearPreferenciaDosPostos():** A função utiliza o vector de pessoas da entrada para criar uma nova matriz de 1 linha com os ids de cada pessoa ordenados da pessoa mais velha à pessoa mais nova, sendo utilizado o menor id como critério de desempate. Como os postos sempre preferem atender a pessoa mais velha não alocada, a lista de preferencias é a mesma para todos os postos de saúde.
- **void gerarListaAtendimento():** A função gera uma matriz na qual cada linha representa um posto de saúde e as colunas as pessoas alocadas para tal posto. Como essa função é central na resolução do problema proposto, a sua corretude será explicada na próxima seção.
- **void imprimirListaAtendimento():** A função imprime para o usuário a lista de atendimento gerada no passo anterior.

3. O Algoritmo de Alocação

Para encontrar a matriz A de alocação de pessoas, considera-se que temos uma matriz unidimensional I que contem os ids das pessoas ranqueadas da mais velha até a mais nova e uma matriz bidimensional D que contém os postos preferidos de cada pessoa. Essas estruturas de dados são ilustradas a seguir.

Matriz I - Preferencia dos Postos

Pessoa mais prioritária	...	Pessoa menos prioritária

Matriz D - Preferencia das Pessoas

Pessoa 0	Posto mais preferido	...	Posto menos preferido
Pessoa 1	...	...	...
...	...	...	...
Pessoa M-1	Posto mais preferido	...	Posto menos preferido

Matriz A - Alocação de Pessoas

Posto 0	Pessoa R	...	Pessoa I
Posto 1	...	...	...
...	...	...	...
Posto N-1	Pessoa U	...	Pessoa L

Algorithm 1: Algoritmo Gerar Lista de Atendimento

input : Matriz I de preferencia dos postos e Matriz D de preferencia das pessoas

output: A matriz A da lista de atendimento dos postos para vacinação

```
foreach pessoa i ∈ matriz I de preferencia dos postos do
    foreach posto j ∈ matriz D de preferencia da pessoa i do
        if pessoa i não está alocada and posto j tem vaga then
            A[j][final da fila + 1] ← pessoa i
            pessoa i é marcada como alocada
            break
return A
```

Note-se que apesar da função implementada ser void e não necessitar de parametros de entrada, esse padrão foi mantido no pseudo-código para facilitar o entendimento.

3.1. Demonstração Corretude do Algoritmo

Lemma 3.1. *A implementação do algoritmo de gerar lista de atendimento retorna o casamento estável entre pessoas e postos de saúde.*

Demonstração. Considere que uma pessoa i e um posto j formam um par que não está na lista de atendimento A* gerada.

- Caso 1: O posto j não possui vagas para alocar i. Logo, isso significa que o posto j aceitou a alocação de outra pessoa i' primeiro, portanto i' é mais velha ou possui um id menor que i. Logo, o par $i - j$ não é instável.
- Caso 2: O posto j possui vagas para alocar i. Logo, isso significa que i foi alocada para outro posto j' mais perto ou com menor id. Assim, o par $i - j$ não é instável.

■

Note que a alocação depende do número de vagas disponíveis e dos critérios de priorização das pessoas. Portanto, pode ocorrer de não ser alocada nenhuma pessoa para algum posto ou que alguma pessoa fique sem alocação. Porém, é garantido que a alocação será ótima do ponto de vista das pessoas, já que a pessoa que tiver a maior idade sempre conseguirá ser atendida na melhor opção de posto disponível para ela. Além disso, o algoritmo desenvolvido é um pouco diferente da solução de Gale-Shapley pois as preferencias dos postos de saúde são iguais, não necessitando que uma pessoa seja desalocada para dar lugar a outra.

4. Análise de Complexidade Assintótica de Tempo

Existem cinco passos que ocorrem para gerar a saída esperada. Contudo, os algoritmos centrais do código estão nos passos 2 - Obter tabela de preferências dos postos, 3 - Obter tabela de preferências das pessoas e 4 - Gerar a lista de atendimento. Logo, a análise de complexidade assintótica de tempo irá focar nessa parte do desenvolvimento. Para considerar o pior caso, suponha que o número N de pessoas e o número M de postos de vacinação sejam iguais a K e que todos os postos tem apenas uma vaga.

1. **Passo 2 - Obter tabela de preferências dos postos** Essa função depende unicamente de N. Inicialmente é feito um for para obter as idades e ids das pessoas. Em seguida é feita uma ordenação e por fim outro laço for para enviar cada id ordenado para o atributo da classe. Portanto a complexidade é $O(N+N*log(N+N))$, ou seja, $O(K * log(K))$.
2. **Passo 3 - Obter tabela de preferências das pessoas:** Essa função depende de N e de M. Para cada pessoa é necessário calcular a sua distancia a cada posto de saúde. Portanto, a função é $O(N*M)$. Além disso, após conhecida as distancias é necessário ordená-las usando a função std::sort em $O(M * log(M))$. Em seguida, o resultado da ordenação é passado para uma nova matriz, em $O(M)$. Portanto a complexidade é $O(N*(M+M*log(M)+M)) = O(K*K+K*K*log(K)+K*K)$, ou seja $O(K^2)$.

3. **Passo 4 - Gerar a lista de atendimento:** Essa função depende de N e M. Existem dois laços alinhados usados para encontrar o posto mais próximo e disponível para cada pessoa. Como estamos considerando o pior caso, temos que a busca por esse posto é $O(N * M)$, ou seja, $O(K^2)$.

Assim, podemos concluir que a complexidade assintótica de tempo geral do código desenvolvido é $O(N * M)$, ou seja $O(K^2)$.

5. Considerações finais

A partir do trabalho foi possível entender mais sobre o Problema do Casamento Estável na prática.A implementação do trabalho transcorreu sem maiores problemas e os resultados ficaram dentro do esperado. Foi possível cumprir com os objetivos do trabalho prático proposto. Não foram encontradas muitas dificuldades, o principal desafio foi entender como seria a estratégia de solução do problema.

Referências

Cormen, T. H. (2002). *Algoritmos: Teoria e Prática*. Editora Campus.

Jon Kleinberg, E. T. (2006). *Algorithm Design*. Pearson Education.

Ziviani, N. (2006). *Projetos de Algoritmos com Implementações em Java e C++*. Cengage Learning.