1 Introdução

Desde o ano de 2020, o vírus SARS-CoV2 se espalhou globalmente e foi decretado um estado de pandemia global por causa do mesmo, que causa a doença chamada de Covid-19. Um fantásico esforço global no avanço científico fez com que vacinas pudessem ser feitas e testadas em tempo recorde, e agora, após meses de espera, a população brasileira está sendo vacinada.

Essa documentação tem como objetivo descrever a modelagem computacional e o algoritmo que distribui pessoas entre vagas para vacinação em clínicas enquanto satisfaz certos critérios de prioridade entre as pessoas. O problema consiste em clínicas i que tem um número Q_i de vagas para pessoas que podem ser vacinadas ali, e devemos alocá-las considerando como critério, principalmente, a idade de cada uma delas(pessoas mais velhas tem maior prioridade na alocação para clínicas mais próximas).

Por fim, foi possível propor um algoritmo que garante que a alocação seja ótima do ponto de vista das pessoas, e, ainda, que ela seja a mais justa possível, respeitando as restrições(idade e o número de vagas por posto).

2 Execução do Código

2.1 Compilação e Execução

Pré-requisitos:

- Compilador moderno de C++(e.g: g++ 10.x.x)
- Programa *GNU Make* instalado (altamente recomendado)

Na pasta raiz do projeto, onde se localiza o *Makefile*, execute o comando *make* e então um arquivo chamado ***tp01*** será gerado no diretório */bin*, que também está na raiz do projeto.

```
$ make
$ cd bin
$ ./tp01 < entrada.txt
```

2.1.1 Testes Unitários

Grande parte do programa foi desenvolvida fazendo uso de testes unitários. Caso queira executar os mesmos é necessário que comente todo o código no arquivo *src/main.cpp* e descomente todo código no arquivo */tests/test.cpp*. Feito isso, basta executar o seguinte comando na raiz do projeto:

```
$ make test
```

Os testes serão feitos e os arquivos serão limpados automaticamente depois.

3 Modelagem Computacional

O problema tratado pode ser enxergado como um problema de 'casamento' entre pacientes e clínicas, onde os pacientes tem uma preferência por clînicas mais próximas e as clínicas tem preferência pelos pacientes mais velhos, sendo que, em segunda instância, ambos preferem quem tiver o seu número de identificação menor.

Enxergando o problema dessa forma, Foi possível desenvolver uma versão adaptada ao problema do algoritmo conhecido como *Gale-Shapley*, que oferece um método que garante um casamento estável, ou seja, o mais justo seguindo os critérios de preferência definidos, e que foi feito de tal forma que garante um resultado ótimo para as pessoas.

Após uma leitura em alto nível do problema, foi necessário analisar quais tipos de operações seriam necessárias e mais críticas(frequentes), para então escolher as estruturas mais adequadas que poderiam contribuir para uma maior eficiência do algoritmo.

Inicialmente, uma pessoa pode propor para múltiplas clínicas, e é importante que ela saiba se já conseguiu uma vaga, para quais clínicas ainda pode propor ou se ja propôs para todas. Do ponto de vista da clínica, é importante saber se ela está cheia ou se pode aceitar o paciente sem entraves, e, caso esteja cheia, precisa saber se existe alguém com menor prioridade que aquele quem está solicitando a vaga, pois se sim será necessária a troca. Nesse texto irei me referir à essas classes também como pessoa ou paciente, para a *Patient*, e clínica para a *Clinic*.

Após a leitura da entrada de informações, cada pessoa é criada recebendo seu número de identificação, sua posição e uma referência para um mapa que tem a informação associada à cada uma das clínicas. Cada pessoa só precisa saber a posição da clínica para calcular a sua ordem de preferência, então optei por utilizar apenas essa referência que permitirá gerar uma estrutura interna mantendo a única informação necessária, que é o identificador da clínica. A estrutura interna mencionada que armazena as preferências da pessoa é, na verdade, uma pilha, que foi montada de tal forma que uma pessoa pode saber qual é a próxima clínica para a qual ela irá propor seguindo a sua ordem de preferência em tempo constante, pegando o elemento no topo da sua pilha.

Da perspectiva da clínica, elas também são criadas com base na leitura da entrada, sendo que as informaçõs mais relevantes são a sua posição e a sua capacidade. A classe *Clinic* foi pensada de uma forma que visa reduzir o custo temporal das operações mais frequentes, e usa de alguns artifícios para reduzir a complexidade dessas operações. Uma operação frequente é a checagem se há alguém já alocado na clínica que tem prioridade menor do que alguém que está solicitando uma vaga, a solução ingênua seria comparar cada um dos pacientes da clínica e, caso fôsse encontrado, remover esse alguém da clínica para então adicionar quem solicitou a vaga. Para contornar essa custosa operação, eu optei por, toda vez que um paciente for inserido na clínica, ordenar de maneira crescente o vetor que os contém, de tal forma que sempre que for necessário verificar se existe um paciente com menor prioridade que alguém solicitando uma vaga, basta uma única comparação com o primeiro elemento do vetor de paciantes. Por fim, o problema da remoção em um vetor também pode vir a ser custosa, já que ao deletar um elemento pode ser necessário deslocar todos os elementos para continuar com o vetor em uma parte contínua de memória, e para solucioná-lo bastou tomar o cuidado de copiar as informações desse paciente no índice zero que será removido e, então, sobrescrevê-las com as informações do novo paciente, retornando as do antigo para que ele possa voltar a propor para clínicas. Não foi necessário armazenar a informação da preferência de cada clínica quanto aos pacientes pois todas preferem aqueles mais velhos e com menor número de identificação, então basta usar esse critério no momento de comparação.

Na classe *UtilsService*, fazendo uso dessas estruturas descritas anteriormente, foi criado o método que executa o algoritmo de Gale-Shapley e aloca os devidos pacientes nas clínicas.

Algorithm 1

Gale-Shapley (Pacientes com sua lista de preferência, Clinicas):
while (houver alguém p não alocado **E** que não propôs para todas as clínicas) **do**
c $\leftarrow$ primeira clínica em preferência para qual p ainda não propôs
Se c não está cheia **então** adicione p aos pacientes de c
Ou se c está cheia mas tem alguém v com menor prioridade **então** remova v, volte seu estado para não alocado, e adicione p aos pacientes de c
Ou p precisa tentar uma próxima clínica, caso não tenha proposto para todas
end while

Esse algoritmo evita os dois casos de instabilidade definidos, observando que: as pessoas propõem para todas as clínicas em ordem decrescente de preferência; que enquanto alguém não estiver alocado e ainda não tiver proposto para todas as clínicas o algoritmo não terminará; e que uma clínica com espaço aceita qualquer paciente.

- Caso 1: Existir uma pessoa sem alocação enquanto ainda há vagas em alguma clínica.

  Vamos supor que uma clínica c ficou com uma vaga sobrando ao final do algoritmo Gale-Shapley. Supondo também que um paciente p não estava alocado ao final da execução. Pelas observações, c não recebeu uma pedido de alocação de p, mas p propôs para todas as clínicas, já que está desalocado no final. Uma contradição.

- Caso 2: Uma pessoa estar alocada em um posto β mas há um posto α mais próximo que possui vagas disponíveis ou que está alocando uma pessoa de menor idade.

  Supondo que uma pessoa p foi alocada na clínica β, mas preferiria α. Existem apenas duas opções:

  - p propôs para α. Nesse caso p foi rejeitado(imediatamente ou posteriormente), significando que α deu prioridade para outra pessoa p_{dif} e portanto p-α não é um par instável.

  - p nunca propôs para α. Nesse caso, sabendo que p propõe em ordem decrescente de preferência, p prefere a clínica β e portanto o par p-α não é instável.

Por fim, podemos mostrar que o algoritmo retornará um casamento R^* que é ótimo da perspectiva da preferência das pessoas:

- Vamos supor que uma pessoa esteja alocada em uma clínica que não seja a sua melhor clínica possível.
- As pessoas propõe em ordem decrescente de preferência $\implies$ alguma pessoa foi rejeitada por uma clínica que era válida como "parceira".
- Vamos chamar essa pessoa de p e a primeira clínica que rejeita p de c.
- Seja R o casamento estável onde p e c estão "casados", ou seja, p está alocada em c.
- Em R^*, quando p foi rejeitada por c, c reafirmou sua preferência com uma pessoa p_d de maior prioridade, ou seja, c prefere p_d à p.
- Em R, chamaremos de c_d a clínica na qual p_d foi alocada.
- p_d não havia sido rejeitada por nenhuma clínica no momento em que c rejeitou p, por definição. Então p_d não havia proposto para c_d quando p_d propôs para $c \implies p_d$ prefere c à c_d
- Portanto, p_d-c é um par instável em R, uma contradição.

4 Análise de Complexidade Assintótica

A ênfase da análise será no algoritmo central proposto, mas é importante considerar que existe um pré-processamento na entrada, onde criamos cada pessoa/paciente e sua lista de preferência. Esse pré-processamento consiste no cálculo da distância, por cada um dos n pacientes, para cada uma das m clínicas, depois disso esse vetor é ordenado e cada paciente o atravessa para criar uma pilha. Esse pré-processamento para executar o algoritmo Gale-Shapley proposto devidamente tem complexidade $\mathrm{O}(n^2)$.

4.1 Gale-Shapley

O algoritmo foi implementado para funcionar com uma pilha de pessoas, de tal forma que ele consegue acessar em tempo constante a próxima pessoa que pode começar propor e também adicionar alguém ao topo em tempo constante, caso necessário.

Cada uma das n pessoas pode propor para até m clínicas. Cada proposta consiste em uma pessoa retirar à próxima pessoa da sua pilha de preferência, em tempo constante, e "perguntar"à ela se ela tem espaço, essa validação também acontece em tempo constante, já que a clínica mantém um contador. Caso haja espaço, uma inserção é feita na lista de pessoas da clínica e essa mesma lista é, então, ordenada, para possibilitar o tempo constante no momento de verificar se existe alguém com prioridade menor do que um solicitante.

Caso não haja espaço na clínica, é necessário verificar se a clínica tem alguém com menor prioridade que essa pessoa, e, como foi dito acima, esse processo também acontece em tempo constante, já que temos um vetor ordenado por prioridade, essa verificação sempre consistirá em comparar a pessoa solicitante apenas com a pessoa no índice 0 do vetor. Se a troca for necessária, os dados da pessoa solicitante serão copiados para o índice 0 e o vetor é ordenado novamente, enquanto os dados da pessoa removida são retornados e adicionados à pilha de pessoas que podem propor.

Portanto, o processo de uma proposta feita por uma pessoa para uma clínica, embora seja agilizado por várias operações em tempo constante, é limitado pela ordenação que pode acontecer, colocando um limite superior de $\mathrm{O}(n \cdot \log n)$.

Ademais, sabendo que cada clínica tem no mínimo 1 vaga, é possível que hajam mais pessoas do que vagas em clínicas, nesse caso as n pessoas irão propor para até m clínicas, sendo que $m < n$. Por outro lado, é possível que o número de vagas em clínicas seja maior ou igual ao de pessoas, então cada pessoa irá propor para, no máximo, m clínicas, sendo $m = n$, já que cada clínica contém ao menos 1 pessoa. Assim, o laço das propostas é executado até $\mathrm{O}(n^2)$ vezes.

Por fim, sabendo que o laço de propostas é executado até $\mathrm{O}(n^2)$ vezes e que cada proposta tem como limite superior $\mathrm{O}(n \cdot \log n)$, o algoritmo central proposto é $\mathrm{O}(n^3 \cdot \log n)$

$$f_1 = O(n^2), f_2 = O(n \cdot \log n) \longrightarrow f_1 f_2 = O(n^3 \cdot \log n)$$