Gale-Shapley Algorithm Application

Marcelo Mascarenhas Ribeiro de Araújo
2019110053

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

mascarenhassribeiro@gmail.com

1.Introdução:

Essa documentação tem a finalidade de apresentar diretrizes a serem seguidas na execução do programa, além de fornecer uma breve explicação da implementação realizada, em concomitância com uma análise de complexidade das principais funções utilizadas para a solução do problema proposto.

O projeto trata da implementação de um sistema responsável por alocar cidadãos à postos de vacinação para o recebimento de vacinas de imunização contra o vírus Sars-CoV-2, causador da doença covid-19, seguindo alguns critérios. O programa realiza a leitura das localidades geográficas e as idades das pessoas cadastradas, além de receber as posições dos postos e suas respectivas capacidades. Dito isso, os indivíduos são alocados nos postos mais próximos à sua localidade, priorizando as pessoas que tem as maiores idades. Caso exista mais que uma pessoa com a mesma idade, aqueles que possuem um menor número de identificação são alocados primeiro. Além disso, caso existam dois postos que contenham uma vaga e que sejam equidistantes de uma pessoa, o posto com o menor ID também será priorizado. Esse processo ocorre até que todas as vagas dos postos de vacinação sejam preenchidas, ou todos os indivíduos sejam alocados.

Nesse sentido, o projeto modela o problema computacional descrito e apresenta uma solução viável, que recebe os parâmetros através de um arquivo .txt, e realiza a leitura e a execução seguindo a lógica proposta na apresentação dos requerimentos do projeto.

Para informações mais detalhadas sobre o projeto, consulte as seções a seguir, sendo que:

- A seção 2 trata sobre as decisões tomadas para a resolução do problema proposto, bem como alguns detalhes de implementação, além do ambiente computacional que o trabalho foi desenvolvido.
- A seção 3 informa as instruções de compilação e execução.
- A seção 4 aborda uma breve análise de complexidade, de tempo e espaço, das principais funções e métodos utilizados no programa.
- A seção 5 conclui a documentação, fornecendo alguns detalhes adicionais sobre o processo de desenvolvimento e considerações em relação à solução proposta.

2.Implementação:

O programa foi desenvolvido e testado nas seguintes especificações:

-Linguagem: C++;

-Compilador: g++;

-OS e Versão do sistema: Linux Mint, versão 20.04 (Ulyssa).

-Configuração do computador: AMD Ryzen 5 3600, 16GB RAM.

O fluxograma do programa segue a seguinte ordem: No main, são criados dois std::vectors, de modo que um dos vetores contenha objetos do tipo ‘Persons’, e o outro contenha objetos do tipo ‘Stations’. Após isso, a função de leitura do arquivo passado é chamada, e os vetores são preenchidos com objetos criados de acordo com as informações disponibilizadas no arquivo de entrada. Depois desse processo, um método de ordenação estável é chamado, para que as pessoas que possuam as maiores idades e os menores ids fiquem dispostas primeiro, estabelecendo, portanto, a ordem de prioridade de alocação requisitada no detalhamento do problema. Em seguida, a função que aloca as pessoas nos postos é chamada – para conferir os detalhes de funcionamento do método, consultar as informações abaixo -. Após essa etapa, as listas de pessoas de cada posto são impressas, é feita a deleção dos objetos alocados no heap, e o programa se encerra.

O código está organizado em 5 diferentes arquivos, sendo eles:

Nota: Serão omitidos os parâmetros recebidos e os tipos retornados, salvo por quando forem citados na descrição, para fins de simplificação. Para mais detalhes, ou em caso de dúvidas, consulte o código que acompanha a documentação.

Main.cpp: Programa principal, responsável por declarar as estruturas de dados, chamar o método de leitura do arquivo, o método de preenchimento das pessoas nos postos, ordenar o vetor de pessoas, e deletar os objetos do heap.

Auxiliar.hpp/Auxiliar.cpp: Classe que contém alguns dos principais métodos responsáveis pelo funcionamento do programa. Segue abaixo o detalhamento dos mesmos:

```
void Auxiliar::readInput(std::vector<Person*> *people, std::vector<Station*> *vacination_posts)
```

Função que lê o arquivo de entrada. Ela lê as informações que estão disponíveis na entrada padrão(cin), e realiza a leitura de dois ‘blocos’ de informações. O primeiro corresponde aos postos, salvando a capacidade, posição e ID no posto em um objeto criado no heap. Após essa etapa, salva esse objeto no vetor ‘vacination_posts’. No segundo bloco, são criados os objetos correspondentes as pessoas, e elas são salvas no vetor ‘people’.
Obs: Para funcionar o método realizar a leitura corretamente, é preciso garantir que os arquivos passados estão no MESMO formato dos passados para os casos de teste.

```
void Auxiliar::orderStationList(Person* p, std::vector<Station*> &station_list)
```

Função responsável por calcular a ordem dos melhores postos de um indivíduo, Recebe um objeto pessoa, e o vetor contendo todos os postos de vacinação. Ordena o vetor utilizando um método, ‘stable_sort’, e usando lambda, para ordenar por distância e por idade. O resultado final é um vetor ordenado em ordem crescente de distância e em relação aos Ids, constituindo a lista de preferência do indivíduo p.

```
void Auxiliar::allocPersonsInPosts(std::vector<Person*> &people, std::vector<Station*> &vacination_post)
```

Função que realiza as respectivas alocações, podendo ser considerada o coração do programa. No começo, cria uma lista auxiliar que contém uma cópia de todos os objetos de vacination posts. Para cada pessoa no vetor ‘people’ (que já está ordenado, considerando que as primeiras pessoas são as com maiores idades e menores ids), ordena essa lista, chamando a função ‘orderStationList’. Após essa etapa, realiza a alocação das pessoas, num loop que testa cada posto em ordem de preferência. O programa caminha na lista de preferência de postos de uma pessoa p até encontrar um vazio. Depois, a pessoa é alocada, e o programa segue para a próxima pessoa. Isso ocorre até todas as pessoas serem alocadas. Caso a lista de preferências termine e um indivíduo p não tiver sido alocado, a função se encerra(Pois isso implica que todos os postos estão cheios, e que nenhuma pessoa será alocada).

Person.hpp: Classe base do programa, responsável por guardar a idade, localização e ID das pessoas passadas no arquivo de entrada. Além disso, contém um atributo bool que denota se a pessoa foi ou não alocada à algum posto. Possui apenas métodos simples, i.e, construtores, gets e sets.

Stations.hpp: Outra classe base, responsável por guardar os ids, localização, capacidade e uma lista de pessoas que foram alocadas aquela estação de vacinação -TAD std::list -. Também possui apenas métodos simples, i.e, construtores, gets, sets e alguns métodos de manipulação da lista.

- **Observação:** Os arquivos Person.hpp e Stations.hpp não possuem um arquivo .cpp, já que seus métodos são simples e suas implementações podem ser realizadas inline, dispensando a necessidade dos mesmos.

2.1. Estruturas de Dados Utilizadas:

Para a implementação do programa, duas estruturas de dados principais foram utilizadas: vetores e listas. No caso deste programa específico, a necessidade foi a utilização de uma estrutura que providenciasse uma fácil manipulação de uma lista sequencial de elementos. Embora hajam diferenças entre esses TADs, o programa poderia ser feito somente com vectors, ou somente com lists, sem impactar na performance geral do programa.

2.2. Pseudo-Código:

Segue o pseudo-código da parte principal do programa:

```
Begin:
    Initialize a Person vector. Initialize a Station vector
    Read information from the file, and allocate the objects in those vectors
    Order the person vector with a stable method by age in descending order.
    Initialize an auxiliar station list.
    Fullfil the aux list with a copy of all the objects of the station vector.

    For each person:
        Order the auxlist by the person's distance from a station, in ascending order, using the
        smallest station id in a comparison as a tiebreaker.

        For each item in aux list
            If item→list of persons allocated is full
                Continue
            Else
                Put a reference of the person in item→list of persons allocated
                Set person attribute is_allocated to True
                Break
        EndFor
        If person is_allocated == false
            Break
    Endfor

    For each post in Station
        Print post→list of persons allocated
    Endfor
End
```

3. Instruções de compilação e execução:

1. Realize a extração do zip para um diretório de sua preferência.

2. Acesse o diretório onde está o makefile, abra o terminal no diretório, e execute o comando make (Caso o comando não seja reconhecido no Windows, é necessário instalá-lo, através do pacote "Chocolatey"). Um executável com o nome "tp01" irá ser criado na pasta do makefile.

3. Abra o terminal.

No Windows:

- Execute o programa digitando o nome do executável, em concomitância com o caminho completo do arquivo que possui as informações dos postos e pessoas, com aspas, e o passando para a entrada padrão com o operador “<”. Veja o exemplo:

tp01 < “C:/Users/Marcelo/OneDrive/Documentos/Faculdade Stuff/TP1_ALG/input.txt”

No Linux:

- Execute o programa digitando o nome do executável precedido de um ./ , em concomitância com o caminho completo do arquivo que possui as informações dos postos e pessoas, e o passando para a entrada padrão com o operador “<”. Veja o exemplo:

./tp01 < /home/mmra/Documents/Faculdade/Algoritmos_I/Trabalho/input.txt

4. Análise de Complexidade:

Como os arquivos Person.hpp e Station.hpp só contém setters, getters, construtores e manipuladores de lista, a análise dos métodos dessas classes não serão realizadas. Segue a análise, portanto, do main e dos métodos auxiliares.

`void Auxiliar::readInput`

O método realiza algumas operações constantes, com complexidade O(1). Além disso, possui dois Whiles. O primeiro executa 2 vezes(sobre ambos os blocos, de pessoa e estações), e o segundo executa sobre o número de linhas sobre cada bloco. Isso equivale, na verdade, a dizer que a função lê as n linhas do arquivo, e em cada uma dessas n linhas, realiza diversas operações. Além disso, aloca n-2 objetos na memória (já que 2 linhas do arquivo dizem qual é o número de objetos de cada bloco). Dessa forma, podemos dizer que o método tem c**omplexidade de tempo e espaço de Θ(n)**.

`void Auxiliar::orderStationList`

O método realiza o sort estável de uma lista, de acordo com a distância de 2 postos em relação à pessoa. De acordo com a própria documentação da biblioteca do C++, apesar de não ser citado qual é o método de ordenação exato que foi utilizado no sort de uma lista, a complexidade de tempo referida do método é O(nlogn), tal que n seja o número de elementos ‘postos’ na lista. Portanto, a **complexidade de tempo** da função também ser **O(nlogn)**. Possui também **complexidade de espaço O(1)**, por possuir apenas declarações constantes que não variam de acordo com a entrada.

`void Auxiliar::allocPersonsInPosts`

Função principal do programa, que faz as alocações conforme solicitado na descrição do problema. Seja m o número de objetos no vetor de pessoas, e n o número de objetos no vetor de postos. A função começa alocando, numa lista auxiliar, os n objetos do vetor de postos. Após isso, para cada pessoa, a lista é ordenada, de acordo com a função orderStationList. Depois, percorre a lista auxiliar tentando alocar as pessoas em cada um dos postos referidos. Assim, podemos dizer que a complexidade de tempo da função é O(n+m*(nlogn + n)), o que implica numa complexidade de O(n+mn + m*nlogn).Assim, temos que a **complexidade de tempo** da função é **O(m*nlogn)**. A função aloca uma lista auxiliar que contém n objetos do tipo posto. Logo, a c**omplexidade de espaço** da função é **Θ(n).**

`int main(int argc, char* argv[])`

A função declara dois vectores, e chama a função readInput. Após isso, chama o método std::stable_sort para o vector de pessoas, que possui complexidade O(nlogn). Depois, chama a função allocPersonsInPosts. Por fim, realiza um loop, iterando sobre o vetor de vacinação para imprimir a lista de pessoas(Cada posto m possui uma fração das pessoas alocadas. Como no máximo os m postos podem imprimir ao todo n pessoas, a

complexidade desse loop é O(m+n).) alocadas em cada posto, e realiza um outro loop para deletar os objetos das pessoas. Assim, como as operações realizadas na função são de menor custo do que o método allocPersonsInPost, o main ‘herda’ a complexidade deste, possuindo **complexidade de tempo** também de **O(m*nlogn)**, tal que m é o número de pessoas no vetor Persons, e n é o número de postos no vetor de Stations. Como a função main também não faz alocações que dependam do número de entradas, ela ‘herda’ a complexidade da função que a faz, possuindo **complexidade de espaço Θ(L)**, tal que L é o número de linhas do arquivo passado. (Note que o custo de espaço da função readInput é maior que o custo de espaço da função allocPersonsInPosts, já que uma realiza L alocações, e a outra realiza alocação referente ao número de postos em uma linha auxiliar).

5. Conclusão:

Considerando todas as informações supracitadas, o problema apresentado, que consistia no desenvolvimento de um sistema que alocasse pessoas em postos de saúde de acordo com critérios pré-estabelecidos, foi resolvido de maneira satisfatória. Não foram detectados erros nos testes realizados, seja de execução – considerando tanto os criados pelo desenvolvedor, quanto os passados pela equipe de monitoria -, quanto em testes de memórias – o programa foi submetido à testes com a ferramenta ‘Valgrind’ antes da entrega -. Além disso, a resolução se demonstrou extremamente modularizada e não tão custosa, apesar de provavelmente existirem projetos com uma eficiência maior.

Referências:

Chaimowicz, L. and Prates, R. (2020). Modelos e Requerimentos de Documentações, fornecido na disciplina de Estruturas de Dados. Disponibilizado via Moodle. Departamento de Ciência da Computação. Universidade Federal de Minas Gerais. Belo Horizonte.