**Algoritmos e Estruturas de Dados I**

Sistema de monitoramento de qualidade de atendimento

Pedro Turik Firmino 22200699

Lucas Bastos Davi 22200307



**Estratégia para resolução:**

Optamos pela estratégia de modelar os elementos inicialmente em classes que modularizam o comportamento, e em seguida aplicar a lógica do processo. A principal estrutura de dados abstrata utilizada foi a Fila, implementada com uma lista encadeada.

A implementação com listas encadeadas possibilita que a inserção e retirada de elementos da fila sejam operações de complexidade constante (graças ao fato de mantermos referências tanto para o início, quanto para o fim da fila).

Além de funcionalidades normais de fila, a estrutura foi equipada com um método waitAllAndCheck, com o intuito de abstrair a chamada do método await dos objetos da classe Paciente, tanto quanto a checagem de quais fariam ouvidorias. “waitAllAndCheck” tem complexidade de O(n).

A classe Paciente representa os pacientes do sistema, e possui métodos e atributos para a identificação/armazenamento de seu grau de risco, para os cálculos de tempo de espera, e para checagem de ouvidoria.

A classe FilaPrioridade é um wrapper em torno de 4 filas internas, que distribui o paciente de acordo com seu grau de risco, e retira os pacientes da fila de acordo com a prioridade. Essa classe também mantém informações e métodos para o cálculo das médias exigidas, e para distribuir entre as filas internas os chamados para waitAllAndCheck para suas filas internas.

As classes de salaTriagem e salaAtendimento são semelhantes entre si. Elas armazenam o paciente no qual estão trabalhando, e os mantém pelo tempo necessário (decidido no momento de inserção). Todas as operações nessas salas têm complexidade constante.

Por fim, a salaEspera, que mantém uma fila armazenada de pacientes na sala de espera. Pode adicionar realizar enqueue e dequeue, chamar waitAllAndCheck na fila, e também calcular a média de tempo esperado.

**Execução da lógica:**

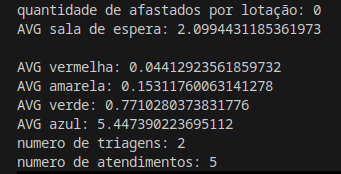
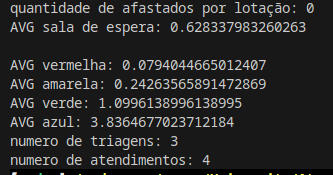
Na main, instanciamos os objetos que modelamos além de algumas variáveis necessárias, e adicionamos as salas de triagem e atendimento em arrays ( construídos com 1000 espaços fixos devido às proporções do problema mas poderiam ser de tamanho variado ), para que possamos manipular várias salas de maneira mais fácil.

O loop principal é definido, então, como a seguinte sequência:

* Primeiro adicionamos (com 50% de chance) um novo paciente na espera. Essa operação é de complexidade constante, e retorna false, caso um paciente tenha quisto entrar mas a sala estava cheia.
* Em seguida loopamos por todas as salas de triagem. Caso a sala de espera não esteja vazia e a triagem esteja, retiramos o paciente da espera e inserimos na triagem. Se a sala de triagem não estiver vazia, realizamos a triagem (método retorna true, quando a triagem está finalizada). Caso a triagem tenha sido finalizada resetamos o wait time do paciente, definimos seu grau de risco e inserimos ele nas filas adequadas (switch no método de inserção da filaPrioridade)
* DESTAQUE IMPORTANTE: A nossa implementação dessa atividade, como podemos ver, assume algumas coisas: que um paciente recém inserido na triagem já está apto para realizar a triagem em si (mesma lógica servirá para o atendimento), e que cada paciente é capaz que fazer uma reclamação por tempo de espera, por setor (uma vez para a sala de espera e uma vez para as filas, e por isso o reset do wait time). Além disso estamos levando em consideração que ouvidorias permanecem ativas, mesmo após a tratamento do Paciente ser finalizado (as auditorias só voltaram a zero quando chegarem a soma de 10, e resultarem em um aumento do número de salas)
* Agora que os pacientes já estão em suas filas adequadas, loopamos pelas salas de atendimento. Caso a fila não esteja vazia mas a sala de atendimento esteja, retiramos o paciente da fila e o inserimos na sala. Em seguida, caso a sala não esteja vazia, executamos o atendimento (retorna true em caso de atendimento finalizado). As operações nas salas de triagem e atendimento realizam pouco trabalho, mas possuem complexidade O(n) para n salas operando, podemos se tornam um pouco de maior tempo de execução para inputs de maior escala.
* Por fim, atualizamos as variáveis de ouvidorias, chamando waitAllAndCheck na sala de espera e nas filas, aumentando a quantidade de triagens ou atendimentos caso necessário. Novamente, por terem sido guardadas em arrays de tamanho fixo, a operação de criação é de complexidade constante, mas seria O(n) em caso de armazenamento variável. E como dito anteriormente, waitAllAndCheck se dá em O(n)

**Respostas e execução**

Segue aqui imagens da execução do programa contendo instruções para reprodução.





Para executar, escreva este comando na raiz do projeto