Trabalho Prático 2

Simulador de eventos discretos

Maria Eduarda Nunes e Carvalho de Vasconcelos Costa - 2023087982

1. Introdução

Nesse trabalho, o estudante foi contratado pelo Hospital Zambs (HZ) para implementar e otimizar um sistema de filas do hospital, que possui no total seis procedimentos (triagem, atendimento, medidas hospitalares, testes de laboratórios, exames de imagem, medicamentos/instrumentos) e cada procedimento tem um número de atendentes e um tempo de atendimento, além de três filas de prioridades: uma dos pacientes verdes, uma dos pacientes amarelos e uma dos pacientes vermelhos. A criação desse sistema consiste em basicamente a implementação de um simulador de eventos discretos, que controla o relógio do hospital, atualiza as estatísticas dos pacientes, e move os pacientes para filas ou para procedimentos, quando estão liberados, com base na hora de chegada em cada etapa e no grau de urgência.

2. Método

2.1. TAD Escalonador

O tipo abstrato de dado (TAD) chamado `Escalonador` é uma estrutura que representa um minHeap em um vetor. Ela armazena os `Pacientes` que estão sendo atendidos ou não chegaram no hospital no atributo `pacientes`, a quantidade desses pacientes no atributo `qntd` e a quantidade máxima de pacientes que o vetor armazena no atributo `tamMax`.

2.2. TAD Fila

O tipo abstrato de dado (TAD) chamado `Fila` é uma estrutura que representa uma fila, ou seja, o primeiro paciente a entrar na fila é o primeiro a sair. Ela armazena `frente`, que é a primeira posição válida do vetor; `tras`, que é a primeira posição inválida do vetor; `qntd` com a quantidade de pacientes da fila; `tamMax` com a quantidade máxima de pacientes que podem estar na fila e `pacientes`, o vetor de pacientes da fila. A versão implementada da fila é a fila circular, para evitar os custos de inserção e remoção de uma fila tradicional.

2.3. TAD Hospital

O tipo abstrato de dado (TAD) chamado `Hospital` é uma estrutura que representa o hospital do simulador. Ele armazena o relógio do hospital em `relogioHospital`; os pacientes lidos no atributo `pacientesHospital`; a quantidade de pacientes lidos em `qntdPacientes`; as filas em `filas`; os procedimentos em `procedimentos` e o escalonador em `escalonadorHospital`. Essa estrutura é a principal estrutura do simulador, sendo responsável por integrar todos os TADS e simular os eventos discretos do hospital. O atributo `filas` é parecido com uma matriz 6x3, sendo cada linha um procedimento (0 - triagem, 1 - atendimento, 2 - medidas hospitalares, 3 - testes de laboratórios, 4 - exames de imagem, 5 - medicamentos/instrumentos) e cada coluna a fila de urgência do

procedimento (0 – Verde, 1 – Amarelo, 2 – Vermelho), com exceção da fila de triagem, que é apenas uma fila única. O atributo `procedimentos` segue a mesma lógica, mas é um vetor 6x1 e cada linha é um procedimento, com os índices iguais ao atributo `filas`.

2.4. TAD Paciente

O tipo abstrato de dado (TAD) chamado `Paciente` é uma estrutura que representa um paciente do hospital. Ela armazena as informações lidas do arquivo e as informações necessárias para o simulador. Esses atributos armazenam as seguintes informações:

- `id`: o id do paciente;
- `alta`: 0 se não teve alta após ser atendido, 1 caso contrário;
- `dataAdmissao`: a data de entrada do paciente no hospital;
- `grauUrgencia`: o grau de urgência do paciente, importante para encaminhar às filas de urgência correta (0 – Verde, 1 – Amarelo, 2 – Vermelho).
- `quantidades`: um vetor de quatro posições com as quantidades de medidas hospitalares (posição 0), de testes de laboratórios (posição 1), de exames de imagem (posição 2) e de medicamentos/instrumentos (posição 3);
- `estado`: o estado em que o paciente está (1 a 14);
- `idUnidade`: o id da `Unidade` que ele está, ou –1 se estiver na fila;
- `dataFim`: armazena a data da mudança de estado do paciente (ou a data de chegada do hospital ou a data de saída de um procedimento);
- `tempoOcioso`: a quantidade em horas do tempo em que o paciente esperou em uma fila;
- `tempoAtendido`: a quantidade em horas do tempo em que o paciente esteve em atendimento.

2.5. TAD Procedimento

O tipo abstrato de dado (TAD) chamado `Procedimento` é uma estrutura que armazena as informações de um procedimento, sendo elas: o tempo médio do procedimento em `tempo`; a quantidade de unidades em `qntdAtendentes`; e as em `unidades`.

2.6. TAD Unidade

O tipo abstrato de dado (TAD) chamado `Unidade` é uma estrutura que armazena as informações de uma unidade de um procedimento, sendo elas: se está ocupada ou não em `ocupado`; o tempo que ficou desocupada em `tempoOcioso`; e a última data registrada em `ultimaData`.

2.7. Funções

O algoritmo implementado consiste basicamente em funções que permitem a implementação do simulador de eventos discretos. Para isso, foram implementadas as funções explicadas a seguir.

Funções relacionadas ao `Escalonador`:

- `inicializaEscalonador`: inicializa um novo `Escalonador` com a quantidade máxima de pacientes.
- `getAncestral`: retorna a posição do ancestral de uma posição qualquer do vetor.
- `getSucessorEsq`: retorna a posição do sucessor à esquerda de uma posição qualquer do vetor.
- `getSucessorDir`: retorna a posição do sucessor à direita de uma posição qualquer do vetor.
- `insereEvento`: insere um novo `Paciente` no `Escalonador` e o balanceia.
- `retiraProximoEvento`: remove o "menor" `Paciente` do `Escalonador` e o balanceia.
- `escalonadorVazio`: retorna 1 se o `Escalonador` estiver vazio, ou 0 caso contrário.
- `finalizaEscalonador`: finaliza um `Escalonador`, desalocando as memórias alocadas dinamicamente.

Funções relacionadas à `Fila`:

- `inicializaFilas`: inicializa todas as `Filas` do `Hospital` com a quantidade de pacientes lidas.
- `enfileira`: coloca um `Paciente` na última posição da `Fila` recebida.
- `desinfileira`: remove e retorna o primeiro `Paciente` da `Fila`.
- `filaVazia`: retorna 1 se a `Fila` estiver vazia, ou 0 caso contrário.
- `finalizaFilas`: finaliza as `Filas` do `Hospital`, desalocando todas as memórias alocadas dinamicamente.

Funções relacionadas ao `Hospital`:

- `preencheHospital`: lê do arquivo recebido todas as informações e inicializa os atributos do `Hospital` com as informações lidas.
- `determinaProcedimento`: retorna o `Procedimento` que um `Paciente` acabou de sair ou de entrar.
- `mudaEstado`: muda o `Paciente` para o próximo estado e atualiza as estatísticas dele, além de atualizar as unidades ocupadas do `Procedimento` em que o `Paciente` acabou de sair ou de entrar.
- `moveParaFila`: move um `Paciente` para a `Fila` de acordo com seu estado, seguindo seu grau de urgência.
- `moveParaAtendimento`: percorre todos os `Procedimentos` para verificar se tem algum que está livre e tem `Paciente` na `Fila`.
- `verificaProcedimento`: move um `Paciente` que estava em uma `Fila` para aquele `Procedimento` que está desocupado, seguindo a prioridade do grau de urgência.
- `simulaHospital`: realiza todos as transições de estados dos `Pacientes` até que nenhum `Paciente` do `Hospital` tenha algum procedimento a ser realizado.
- `imprimeHospital`: imprime todos os `Pacientes` do `Hospital` na ordem que foram lidos na entrada, mostrando as informações pedidas na especificação do trabalho.

- finalizaHospital: finaliza o Hospital, desalocando todas as memórias alocadas dinamicamente.

Funções relacionadas ao Paciente:

- inicializaPaciente: inicializa um Paciente com as informações lidas no arquivo de entrada.
- pacienteMenor: compara dois Pacientes, de acordo com a dataFim e o id, e retorna 1 se o primeiro for menor que o segundo ou 0 caso contrário.
- determinaQuantidade: retorna a quantidade de Procedimentos que o Paciente deve realizar, de acordo com seu estado.
- imprimePaciente: imprime o id, a data de admissão, a data de saída, a quantidade de horas que o Paciente esteve no Hospital, a quantidade de horas de atendimento e de horas de fila de um Paciente.

Funções relacionadas ao Procedimento:

- inicializaProcedimento: inicializa um Procedimento com o tempo e a quantidade de atendentes lidas no arquivo de entrada.
- inicializaHora: inicializa as horas da Unidades do Procedimento com a hora recebida.
- achaUnidade: retorna o id da primeira Unidade desocupada de um Procedimento ou –1, caso ele esteja cheio.
- procedimentoOcupado: retorna 1 se todas as unidades do Procedimento estão ocupadas ou 0 caso contrário.
- finalizaHora: faz a última atualização do tempo ocioso das Unidades de um Procedimento.
- finalizaProcedimentos: finaliza os Procedimentos do Hospital, desalocando todas as memórias alocadas dinamicamente.

Funções relacionadas a Unidade:

- inicializaUnidade: inicializa uma Unidade com os atributos zerados.

3. Análise de complexidade

As principais funções que impactam a complexidade do algoritmo são as funções relacionadas aos TADs Escalonador, Fila, Hospital e Procedimento. Para as análises a seguir são consideradas as seguintes variáveis: “n” a quantidade de pacientes lidos e “u” a quantidade de unidades de um procedimento.

As funções importantes relacionadas ao Escalonador são as seguintes:

- inicializaEscalonador: tem complexidade de tempo e de espaço $O(n)$, já que é inicializado um vetor com tamanho máximo de pacientes.
- insereEvento e retiraProximoEvento: têm complexidade de tempo $O(1)$, no melhor caso, ou $O(\log \quad n)$, no pior caso quando todos os pacientes estão no escalonador, e complexidade de espaço $O(1)$, pois utiliza uma quantidade fixa de variáveis auxiliares.

As funções importantes relacionadas à `Fila` são as seguintes:

- `inicializaFilas`: tem complexidade de tempo e de espaço $O(n)$, já que é inicializado três vetores para cada `Procedimento` com tamanho máximo de pacientes.
- `enfileira` e `desinfileira`: têm complexidade de tempo e espaço $O(1)$, pois são realizadas um número constante de operações e tem um número fixo de variáveis auxiliares.

As funções importantes relacionadas ao `Hospital` são as seguintes:

- `preencheHospital`: tem complexidade de tempo e de espaço $O(n)$, já que é inicializado os vetores de pacientes do `Hospital`, das `Filas` e dos `Procedimentos` com tamanho máximo de pacientes.
- `simulaHospital`: tem complexidade de tempo $O(g)$, sendo $g = \max(u_{max}, n \log n)$, uma vez que inicializa a hora de todas as unidades e escalona todos os pacientes para seus procedimentos (no mínimo, `2n` eventos e, no máximo, `6n` eventos), e complexidade de espaço $O(1)$.
- `imprimeHospital`: tem complexidade de tempo $O(n)$, pois imprime todos os pacientes do `Hospital`, e complexidade de espaço $O(1)$.
- `finalizaHospital`: tem complexidade de tempo $O(n)$, pois desaloca as memórias alocadas para os pacientes, e complexidade de espaço $O(1)$.

As funções importantes relacionadas ao `Procedimento` são as seguintes:

- `inicializaProcedimento`: tem complexidade de tempo e de espaço $O(u)$, já que é inicializado o vetor de unidades.
- `inicializaHora` e `finalizaHora`: tem complexidade de tempo $O(u)$ e de espaço $O(1)$.
- `achaUnidade` e `procedimentoOcupado`: no melhor caso, tem complexidade de tempo $O(1)$ (primeira Unidade desocupada) e, no pior caso, $O(u)$ (última unidade ou nenhuma desocupada). Tem complexidade de espaço $O(1)$.

A função `main` tem complexidade de tempo igual a maior complexidade do programa, ou seja, igual à complexidade da função `simulaHospital`, $O(g)$, sendo $g = \max(u_{max}, n \log n)$ e complexidade de espaço igual a $O(n + u_{max})$.

4. Estratégias de robustez

Para implementar exceções, foram definidas duas macros, uma chamada `avisoAssert` e outra chamada `erroAssert`. Essas duas estruturas foram utilizadas no código para escrever uma mensagem de aviso em `stderr` ou uma mensagem de erro em `stderr` e a finalização do código.

Em cada caso específico de anormalidade, foi decidido se era enviado um aviso ou um erro. O critério de decisão geral para isso foi: caso tenha um ponteiro nulo quando não se esperava (alocação de memória, entre outros) foi enviado um erro e terminado o

programa; caso tenha ponteiro nulo em situações que não são muito graves, como em `moveParaFila` ou `mudaEstado`, foi apenas enviado um aviso e encerrada a função.

Em caso de erro, foi escolhido a finalização do programa para evitar anormalidades, como ordenar um ponteiro nulo, acessar índices fora da posição permitida, tentar acessar índices de ponteiros nulos, entre outros. Já em caso de aviso, foi escolhido apenas enviar uma mensagem e a finalização da função por não ser muito grave e poder continuar o programa normalmente, apesar desses erros leves. Por exemplo, é possível continuar o programa mesmo que tente adicionar um paciente nulo ao escalonador (no caso, ele é apenas ignorado e não é adicionado).

5. Análise experimental

Para realizar a análise experimental, foram utilizadas as entradas do VPL de teste disponibilizadas no moodle e a entrada de exemplo do enunciado do trabalho. A partir desses arquivos, foram realizados diversos testes e os resultados foram apresentados nas tabelas abaixo.

O primeiro experimento foi realizado variando o número de atendentes disponíveis de cada procedimento. Para fazer esse teste, a primeira linha, na tabela abaixo, de cada arquivo foi realizada com as quantidades originais, a segundo aumentando os atendentes dos três atendimentos menos demorados para a metade do número de pacientes, a terceira aumentando os atendentes dos três atendimentos mais demorados para a metade do número de pacientes e a quarta aumentando os atendentes para o número de pacientes.

Arquivo	Triagem	Atendi.	Med. Hos	Test. Lab	Exa. Img.	Inst Med	Média de espera paciente	Tempo de execução
Arquivo de exemplo	2	2	2	2	2	2	1.14	0.002543604
	2	2	5	5	2	5	0.88	0.012180802
	5	5	2	2	5	2	0.33	0.017758406
	10	10	10	10	10	10	0.00	0.022256134
exemplo1.csv	20	20	20	20	20	20	1.42	0.029238596
	20	20	20	12	12	12	1.42	0.017855013
	12	12	12	20	20	20	9.41	0.025283192
	25	25	25	25	25	25	0.00	0.021118976
exemplo2.csv	2	10	20	3	50	22	0.23	0.028998871
	2	10	50	50	50	50	0.16	0.029958591
	50	50	20	3	50	22	0.07	0.007898453

	100	100	100	100	100	100	0.00	0.036567833
exemplo3.csv	1	1	1	1	1	1	50.62	0.032913074
	1	25	1	1	25	25	27.27	0.028797004
	25	1	25	25	1	1	34.37	0.002469815
	50	50	50	50	50	50	0.00	0.025398599

Ao analisar os resultados obtidos, podemos ver que o tempo de espera reduz mais se os números de atendentes forem aumentados dos procedimentos que demoram mais tempo, o que é lógico de se pensar já que esses procedimentos vão demorar mais tempo, na maior parte dos casos. Além disso, quando todos os atendentes eram da quantidade dos pacientes do hospital, nenhum paciente teve que esperar em nenhuma fila, porém muitas unidades dos procedimentos ficaram desocupadas simultaneamente, o que não é desejável, já que queremos que todas as unidades fiquem ocupadas e que os pacientes fiquem o menor tempo possível em uma fila.

Para o segundo experimento, foram modificados os graus de urgência dos pacientes. Na primeira linha da tabela de cada arquivo, foram utilizados os dados originais; na segunda, foram divididos os pacientes igualmente entre as urgências; na terceira, havia mais pacientes verdes; na quarta, havia mais pacientes amarelos; na quinta, havia mais pacientes vermelhos.

Arquivo	Verde	Amarelo	Vermelho	Média de espera paciente	Tempo de execução
Arquivo de exemplo	5	3	2	1.14	0.002543604
	4	3	3	0.88	0.024737775
	6	2	2	1.19	0.020861554
	2	6	2	1.09	0.017361983
	2	2	6	0.88	0.020809621
exemplo1.csv	6	19	0	1.42	0.029238596
	9	8	8	1.42	0.022655383
	19	3	3	1.42	0.027700974
	3	19	3	1.42	0.020970950
	3	3	19	1.42	0.028088597
exemplo2.csv	30	70	0	0.23	0.028998871

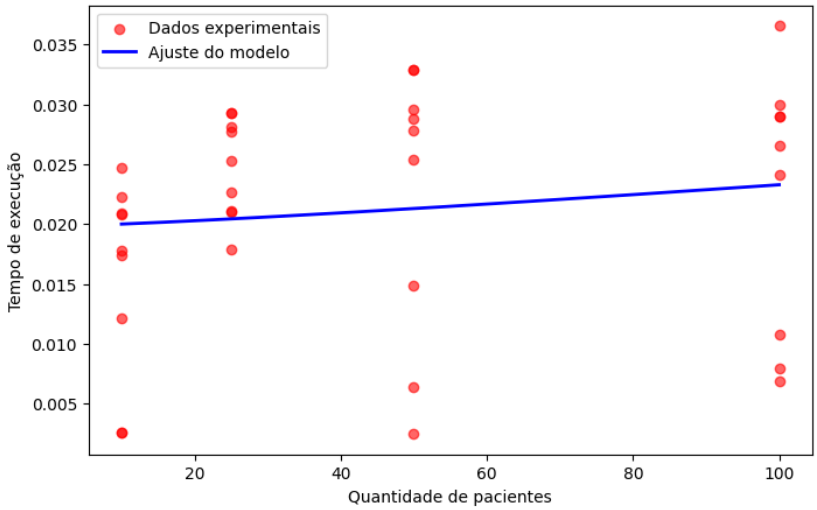
	34	33	33	0.23	0.010762757
	80	10	10	0.22	0.006897825
	10	80	10	0.23	0.026591589
	10	10	80	0.23	0.024137112
exemplo3.csv	5	5	40	50.62	0.032913074
	17	17	16	51.51	0.014906828
	40	5	5	53.68	0.027805118
	5	40	5	52.38	0.029541817
	5	5	40	34.56	0.006413106

Ao analisar os resultados obtidos, percebemos que a quantidade de pacientes de cada grau de urgência influencia diretamente no tempo. Podemos notar que se aumentarem o número de pacientes de cada grau o tempo de espera aumenta ou diminui, já que depende de qual paciente é mudado para o grau de urgência maior. Por exemplo, se mudarmos um paciente que era verde para vermelho, mas que foi o último a chegar no hospital, é provável que não afete muito o tempo de espera, já que os primeiros pacientes devem ter saído do hospital. Por isso, não podemos afirmar se aumenta ou diminui o tempo de espera quando aumenta o número de pacientes de um certo grau de urgência, pois depende de quais pacientes foram trocados de urgência.

A partir desses experimentos realizados, foi possível perceber que o tempo de espera dos pacientes dependem de diversos fatores. Tais fatores são a quantidade de unidades disponíveis, a quantidade de pacientes de cada urgência, a quantidade de procedimentos de cada paciente e todas suas combinações possíveis, além de outros, como a quantidade de pacientes que receberam alta após atendimento. Por exemplo, se aumentarmos a quantidade disponíveis dos procedimentos mais demorados e diminuirmos as quantidades desses procedimentos dos pacientes, é muito provável que o tempo de espera diminua. Diante disso, as análises realizadas nesse trabalho foram apenas uma pequena amostra das diversas combinações que influenciam o tempo de espera, mas não deixam de mostrar resultados significantes dos impactos no tempo de espera dos pacientes.

Quanto ao tempo de execução do programa, podemos perceber que as variações na quantidade das unidades e na quantidade de pacientes de cada urgência influenciaram em tal tempo. A partir dos tempos observados nas tabelas acima, foi, então, feita uma regressão para estimar o tempo estimado de execução do programa, dado um número n de pacientes do hospital. Essa regressão forneceu a seguinte equação $tempo = 4{,}08 \times 10^{-5} n + 0{,}02$, o que confirma a análise de complexidade da seção 3, já que o comportamento esperado por $O(n \log n)$, quando são menos unidades do que pacientes. Muito provavelmente, o programa não teve o pior caso e, por isso, aparentou ter um

comportamento linear em relação a n. A seguir, foi exibido o gráfico gerado por essa regressão, bem como a representação dos dados originais.



6. Conclusão

Em resumo, nesse trabalho foi feita a implementação de um simulador discreto de eventos e de prioridades de filas para que fosse possível calcular o tempo de espera e de atendimento de cada paciente no Hospital Zambs. Para isso, foram utilizadas estruturas de dados aprendidas durante a disciplina, bem como outros algoritmos e raciocínios que tiveram de ser descobertos para que o simulador funcionasse adequadamente, como a lógica de implementação do simulador discreto.

Um ponto importante desse trabalho foi a utilização de uma estrutura minHeap. Essa estrutura foi essencial para o controle dos eventos discretos, pois permitiu que o paciente com o próximo evento sempre fosse retirado para simular o hospital, ou seja, o paciente com a menor data ou menor id. Assim, essa aplicação no mundo real do minHeap possibilitou a melhor compreensão dessa estrutura e de seus benefícios na prática.

Além disso, outro ponto importante foi a implementação da prioridade entre pacientes. Essa prioridade foi essencial para que o paciente correto fosse retirado do escalonador ou que fosse selecionado o paciente correto para ir para o atendimento, ou seja, o paciente que tinha grau de urgência maior. Por isso, o uso da estrutura de uma fila, também aprendida na disciplina, foi fundamental durante esse trabalho.

Diante disso, é possível afirmar que esse trabalho consolidou os aprendizados teóricos da disciplina. Isso ocorreu, pois o uso do minHeap e da fila foram essenciais para o funcionamento do simulador discreto. Também foi importante para melhorar a resolução de problemas e o uso de boas práticas, como a programação defensiva. Por fim, fica evidente que esse trabalho foi essencial para o aprendizado e para possibilitar uma aplicação real da teoria aprendida no curso.

7. Bibliografia

Wagner Meira Jr e Anísio Lacerda. 2024. Slides virtuais da disciplina de Estruturas de Dados. Disponibilizados via moodle. Departamento de Ciência da Computação. Universidade Federal de Minas Gerais. Belo Horizonte.

Wagner Meira Jr e Anísio Lacerda. 2024. Práticas avaliativas. Disponibilizadas via moodle. Departamento de Ciência da Computação. Universidade Federal de Minas Gerais. Belo Horizonte.

8. Extra

Para melhorar o tempo de espera de cada `Paciente` no `Hospital`, foi feita uma mudança na lógica para entrar numa fila. Tal mudança foi: o `Paciente` só entra em uma `Fila` caso ele tenha procedimentos daquela fila a serem realizados, caso contrário ele vai para o próximo `Procedimento`. Essa mudança foi realizada na função `mudaEstado` do arquivo `Hospital.c`, comentada como “Extra”. A seguir, foi realizada as comparações do tempo de espera de um paciente sem e com a mudança.

Arquivo	Tempo de espera	
	Código original	Código do extra
Arquivo de exemplo	1.14	1.14
exemplo1.csv	1.42	1.42
exemplo2.csv	0.23	0.23
exemplo3.csv	50.62	34.56
teste1.csv	8.25	8.25
teste2.csv	0.11	0.11
teste3.csv	218.78	218.24
teste4.csv	39.80	39.80
teste5.csv	24.70	24.70
teste6.csv	94.48	94.48
teste7.csv	355.50	355.50

OBS.: os arquivos “testeX.csv” são os arquivos do segundo VPL de teste

Ao analisar os resultados, podemos perceber que apenas no exemplo3 e no teste3 houve uma melhora. Isso deve ocorrer, pois muitos pacientes devem ficar em filas para procedimentos que eles não têm exames. Nos demais casos, deve haver poucos pacientes com nenhum exame ou que realmente não tenha exame, mas fique esperando na fila para ser atendido. Assim, essa melhoria é observada em apenas alguns casos de entrada e não em todos os casos, mas pode ter um impacto significante na redução do tempo de espera dos pacientes, como vemos no caso do exemplo3.