FIAP – FACULDADE DE INFORMÁTICA E ADMINISTRAÇÃO PAULISTA

AUGUSTO DOUGLAS NOGUEIRA DE MENDONÇA - RM558371
GABRIEL VASQUEZ QUEIROZ DA SILVA – RM557056
GUILHERME ARAUJO DE CARVALHO – RM558926
GUSTAVO OLIVEIRA RIBEIRO – RM559163

CHALLENGE – DINAMIC PROGRAMMING

Sprint 3

SÃO PAULO
2025

SUMÁRIO

INTRODUÇÃO

A evolução da tecnologia tem impulsionado a modernização dos processos laboratoriais, tornando-os mais precisos, eficientes e integrados. No contexto da análise patológica, a etapa de macroscopia é fundamental para a correta identificação e documentação de amostras anatômicas, mas ainda enfrenta desafios devido à sua execução manual. O Projeto NOA (Núcleo de Otimização de Análise) surge como uma solução inovadora para esse problema, buscando automatizar e padronizar o processo de aferição de amostras através de uma estação digital compacta.

Este trabalho aborda o desenvolvimento de um sistema de gerenciamento de pacientes, aos quais fornecerão as amostras para os testes patológicos, focado na aplicação prática de estruturas de dados e algoritmos. A implementação detalhada da pilha e da fila é apresentada para organizar o fluxo de consultas e históricos, enquanto os algoritmos de busca (binária e sequencial) e ordenação (Merge Sort e Quick Sort) garantem a eficiência na manipulação e recuperação dos dados. O objetivo é demonstrar como essas abordagens clássicas da ciência da computação garantem a coerência e a performance na gestão das informações.

1 DESCRITIVO DO PROJETO NOA

1.1 Contexto do Problema

A etapa de macroscopia é a primeira análise realizada pelo patologista ao receber uma peça anatômica, como um órgão, tecido ou fragmento. Atualmente, esse processo é altamente manual e artesanal, envolvendo tinta para marcação de lateralidade, réguas para medições e registros em papel. Esse método tradicional apresenta diversas limitações, incluindo:

- Risco de erros humanos e falta de padronização, comprometendo a qualidade dos registros;
- Falta de rastreabilidade, impossibilitando auditorias precisas e a recuperação eficiente das informações;
- Perda de tempo valioso dos patologistas, reduzindo a produtividade e aumentando o tempo de análise;
- Nenhuma integração com sistemas digitais como NAVE, LIS e outros sistemas laboratoriais, dificultando a fluidez dos processos.

1.2 Solução Proposta

O Projeto NOA (Núcleo de Otimização de Análise) surge como uma solução tecnológica inovadora para modernizar e automatizar o processo de aferição de amostras patológicas. A proposta consiste em uma estação digital compacta que integra hardware e software, combinando captura de imagens, interface guiada e um assistente de inteligência artificial para transformar a macroscopia em um processo seguro, padronizado e eficiente.

Entre as principais funcionalidades do NOA estão:

- Captura automatizada de imagens das amostras, eliminando erros de registro;
- Medição precisa e automática, com sensores ópticos e visão computacional, garantindo precisão milimétrica nos laudos;
- Registro digital dos dados, substituindo os registros em papel e garantindo fácil rastreabilidade;
- Geração automática de relatórios, prontos para exportação e integração com sistemas laboratoriais;
- Digitalização dos fluxos laboratoriais, promovendo um armazenamento seguro e compatível com prontuários eletrônicos.

1.3 Benefícios da Solução

A implementação do NOA traz diversos ganhos para o setor da saúde, incluindo:

- Maior precisão nas medições, reduzindo erros manuais e retrabalho;
- Padronização do processo, garantindo uniformidade e confiabilidade nos registros;

- Otimização do tempo dos profissionais da saúde, permitindo maior dedicação às análises clínicas;
- Melhoria na segurança dos dados, com armazenamento digital e integração direta com sistemas eletrônicos;
- Aprimoramento da eficiência clínica, alinhando-se às melhores práticas tecnológicas da área da saúde.

Com sua abordagem inovadora, o NOA representa um grande avanço na área da patologia, automatizando processos críticos e contribuindo para um ambiente clínico mais eficiente e confiável.

2 NOTAÇÃO O GRANDE DAS FUNÇÕES

2.1 Funções de Ordenação

quicksort(lista):

- Melhor e Caso Médio: O(nlogn)
- Pior Caso: O(n2)
- Comentário: O Quicksort é um dos algoritmos de ordenação mais rápidos na prática. Seu pior caso ocorre quando a lista já está ordenada ou quase ordenada, o que pode levar a um desempenho lento.

mergesort(lista):

- Melhor, Médio e Pior Caso: O(nlogn)
- Comentário: O Mergesort tem uma complexidade de tempo mais estável do que o Quicksort, pois seu desempenho não é afetado pela ordenação inicial da lista. No entanto, ele requer mais espaço de memória (O(n)) para criar as sublistas durante a ordenação.

2.2 Funções de Busca

busca_sequencial(nome):

- Complexidade: O(n)
- Comentário: A busca sequencial percorre a lista elemento por elemento. No pior caso, ela precisa verificar todos os n pacientes para encontrar o desejado (ou determinar que ele não existe).

busca_binaria(lista_ordenada, alvo):

- Complexidade: O(logn)
- Comentário: A busca binária é extremamente eficiente, pois a cada passo ela elimina metade dos elementos restantes. No entanto, ela requer que a lista de entrada já esteja ordenada, o que tem um custo computacional. A função consultar_por_binaria primeiro ordena a lista de nomes com Quicksort, então

a complexidade total da operação é dominada pela ordenação, resultando em O(nlogn).

2.3 Funções de CRUD e Listagem

cadastrar_paciente():

- Complexidade: O(1) (tempo constante, em média)
- Comentário: Adicionar um item ao final de uma lista (.append()) é uma operação muito rápida.

listar_pacientes():

- Complexidade: O(n)
- Comentário: Esta função percorre todos os n pacientes para imprimi-los na tela.

atualizar_paciente() e excluir_paciente():

- Complexidade: O(n)
- Comentário: Para encontrar o paciente com o id correspondente, a função realiza uma busca linear. No pior caso, todos os n pacientes precisam ser verificados.

2.4 Funções de Estruturas de Dados

registrar_historico():

- Complexidade: O(n)
- Comentário: A função faz uma busca linear para encontrar o paciente pelo id, que é uma operação O(n). A adição do histórico à fila (deque) é uma operação O(1). A busca linear domina a complexidade.

listar_historicos():

- Complexidade: O(k) (onde k é o número de históricos na fila)
- Comentário: A função percorre todos os elementos da fila para imprimi-los.

consultar_paciente():

- Complexidade: O(n)
- Comentário: A busca linear por id é a operação mais custosa da função.

listar_consultas_recentes():

- Complexidade: O(c) (onde c é o número de consultas na pilha)
- Comentário: A função percorre os elementos da lista (consultas) para imprimi-los.

3 ESTRUTURAS E ALGORÍTMOS USADOS

3.1 Pilha

Para gerenciar as consultas recentes dos pacientes, utilizei uma pilha, que é uma estrutura de dados que segue o princípio LIFO (Last-In, First-Out), ou "último a entrar, primeiro a sair". No meu programa, a pilha é implementada com a lista consultas.

Aplicação: A função consultar_paciente() adiciona um paciente ao topo da pilha usando o método append(). Quando a função listar_consultas_recentes() é chamada, o reversed() garante que a consulta mais recente (a que está no topo da pilha) seja exibida primeiro, o que é o comportamento esperado para uma lista de itens recentes.

3.2 Fila

Para o histórico de análises, a escolha foi uma fila, que segue o princípio FIFO (First-In, First-Out), ou "primeiro a entrar, primeiro a sair". A implementação utiliza o deque da biblioteca collections, armazenado na variável historicos.

Aplicação: O registro de uma nova análise na função registrar_historico() é feito adicionando-o ao final da fila com o método append(). Essa escolha garante que as análises sejam processadas ou exibidas na ordem exata em que foram registradas, mantendo a sequência cronológica dos eventos.

3.3 Busca Sequencial

A busca sequencial é o método mais direto para encontrar um elemento em uma lista. Ela verifica cada elemento em sequência até encontrar o item desejado.

Aplicação: Minha função busca_sequencial() é usada para encontrar um paciente pelo nome. Percorro a lista completa de pacientes (pacientes) e comparo cada nome. Essa abordagem foi utilizada por ser simples e eficaz, especialmente quando a lista de dados não está ordenada.

3.4 Busca Binária

Para alcançar uma busca mais rápida, utilizei a busca binária, um algoritmo que tem uma complexidade de tempo muito menor, mas que exige que os dados estejam ordenados.

Aplicação: Na função consultar_por_binaria(), primeiro eu pego todos os nomes de pacientes e os ordeno com o quicksort. Só então aplico a busca binária. Isso me permite encontrar o nome do paciente de forma muito mais eficiente, reduzindo drasticamente o tempo de busca em listas grandes.

3.5 Merge Sort e Quick Sort

Ambos os algoritmos de ordenação, Merge Sort e Quick Sort, foram implementados para ordenar os dados, demonstrando métodos mais eficientes do que uma ordenação simples. Eles são baseados no conceito de "dividir e conquistar".

Aplicação:

O Quick Sort foi a escolha principal e é usado diretamente para ordenar a lista de nomes antes de realizar a busca binária. A sua eficiência, na maioria dos casos, o torna ideal para essa tarefa.

O Merge Sort foi implementado como uma opção alternativa no menu principal, servindo para demonstrar outro algoritmo de ordenação com uma complexidade de tempo mais estável.

CONCLUSÃO

O desenvolvimento deste sistema de gerenciamento de pacientes representa um avanço significativo na organização e no acesso à informação clínica. Ao estruturar a manipulação de dados com pilhas e filas, e otimizar a pesquisa e a ordenação com algoritmos como Merge Sort e Quick Sort, garantimos uma integração eficiente entre a lógica de processamento e a apresentação dos dados.

A automação desses processos manuais de organização proporciona maior precisão, segurança e agilidade na busca e na gestão das informações do paciente. A utilização de abordagens como a busca binária contribui para um ambiente de software mais confiável e alinhado às boas práticas da ciência da computação.

Com essa abordagem, o sistema não apenas resolve os desafios da gestão de dados de forma simples, mas também abre caminho para novas inovações, promovendo eficiência e qualidade na análise e no manuseio de registros de pacientes.

LINK GITHUB

https://github.com/guilhermearaujodec/dynamic-programming-sprint3