

Relatório Técnico: Gerenciamento de Pacientes com Algoritmos de Ordenação e Busca

Enrique Antonio Ponce Cruz RA 218197

Matheus Simioni Sversute RA 214981

Rychard Gimenez dos Santos RA 215587

INTRODUÇÃO

Este relatório técnico apresenta uma análise detalhada do código desenvolvido para o gerenciamento de pacientes em uma aplicação Qt utilizando C++. O sistema implementa funcionalidades essenciais como adição, remoção, consulta e geração de relatórios de pacientes, utilizando algoritmos de ordenação e busca para otimizar a eficiência das operações, assim como classificar o paciente de acordo com a gravidade dos sintomas apresentados. Foi utilizada a ferramenta *QTCreator* na versão 14.0.2 baseada na *stack* Qt 6.7.3.

DESCRIÇÃO GERAL DO SISTEMA

A aplicação visa gerenciar uma fila de pacientes que aguardam tratamento, permitindo ao usuário adicionar novos pacientes, atualizar informações de pacientes existentes, remover pacientes da fila e consultar informações específicas. Além disso, o sistema gera relatórios detalhados dos pacientes tratados.

Para garantir a eficiência nas operações de busca e manutenção da fila, o sistema implementa algoritmos de ordenação e busca como ***Selection Sort*** e ***Binary Search***, além de utilizar **Busca Sequencial** para consultas simples.

ESTRUTURA DOS DADOS

CLASSE *PATIENT*

A classe *Patient* representa os dados de cada paciente na aplicação. Ela encapsula as seguintes informações:

- **Atributos:**
 - int id: Identificador único do paciente.
 - QString name: Nome do paciente.
 - QString cpf: CPF do paciente.
 - QString email: Email do paciente.
 - QDate bday: Data de nascimento do paciente.
 - int idade: Idade do paciente, calculada com base na data de nascimento.
 - unsigned sintomas: Sintomas do paciente, representados por flags binárias.
- **Construtor:**
 - Inicializa os atributos da classe, calcula a idade do paciente com base na data atual e ajusta a idade se o aniversário ainda não ocorreu no ano corrente.

VETORES UTILIZADOS

O sistema utiliza três vetores principais para gerenciar os pacientes:

- std::vector<Patient> fila;
 - Representa a fila de pacientes aguardando tratamento.
 - Mantém os pacientes na ordem de prioridade baseada nos sintomas.
- std::vector<Patient> sorted_name_fila;

- Cópia ordenada da fila de pacientes baseada no nome.
 - Utilizado para otimizar as operações de busca binária.
 - `std::vector<Patient>` relatorio;
 - Armazena os pacientes que já foram tratados.
 - Utilizado para gerar relatórios detalhados do sintoma relatado e os dados de cadastro.
-

ALGORITMOS UTILIZADOS

SELECTION SORT

Selection Sort é um algoritmo de ordenação simples que divide o vetor em duas partes: a parte ordenada e a parte não ordenada. Repetidamente, seleciona o elemento mínimo da parte não ordenada e o move para a parte ordenada.

O algoritmo apresenta performance do pior caso de $O(n^2)$ comparações e $O(n)$ em trocas. No melhor caso, apresenta performance de $O(n^2)$ comparações e $O(1)$ trocas, que seria o vetor previamente ordenado. A performance média do algoritmo é $O(n^2)$ em comparação e $O(n)$ em trocas.

Implementação no Sistema:

- **Função:** `ordenar_paciente_sintoma()`
- **Objetivo:** Ordenar a fila de pacientes (fila) com base na prioridade dos sintomas, onde sintomas com valores mais altos têm maior prioridade.
- **Passos:**
 - Itera sobre cada elemento da fila.
 - Para cada posição, encontra o paciente com a maior prioridade na parte não ordenada.
 - Troca o paciente encontrado com o paciente na posição atual.

Considerações:

- **Complexidade de Tempo:** $O(n^2)$, o que pode ser ineficiente para grandes conjuntos de dados.
- **Uso no Sistema:** Adequado para pequenos a médios conjuntos de pacientes, onde a simplicidade do algoritmo é vantajosa.

BINARY SEARCH

Binary Search é um algoritmo eficiente para encontrar um elemento em um vetor ordenado. Ele divide repetidamente o intervalo de busca pela metade até encontrar o elemento ou determinar que ele não existe.

O algoritmo de busca binária apresenta performance no pior caso de $O(\log n)$ comparações. No melhor caso, realiza $O(1)$ comparação, que ocorre quando o elemento alvo está exatamente no meio do vetor. A performance média do algoritmo é $O(\log n)$ em comparação.

Implementação no Sistema:

- **Função:** `binary_search(const QString& name)`
- **Objetivo:** Encontrar o índice de um paciente pelo nome na fila ordenada por nome (`sorted_name_fila`).
- **Passos:**
 - Define os índices de início (*left*) e fim (*right*) do intervalo de busca.
 - Calcula o índice do meio (*mid*).
 - Compara o nome alvo com o nome do paciente no índice do meio.
 - Se igual, retorna o índice.
 - Se o nome alvo for maior, organiza o intervalo para a metade direita.
 - Se menor, organiza para a metade esquerda.
 - Repete até encontrar o paciente ou concluir que não existe.

Considerações:

- **Complexidade de Tempo:** $O(\log n)$, muito eficiente para grandes conjuntos de dados.
- **Uso no Sistema:** Utilizado quando a fila está ordenada por nome, proporcionando buscas rápidas e eficientes.

BUSCA SEQUENCIAL

Busca Sequencial é um método simples de busca onde cada elemento do vetor é verificado em ordem até encontrar o elemento desejado ou chegar ao final do vetor.

Implementação no Sistema:

- **Localização:** Dentro da função `on_consultarPatient_clicked()` para buscas que não utilizam a busca binária.
- **Objetivo:** Encontrar um paciente pelo CPF ou nome na fila não ordenada (fila).
- **Passos:**
 - Itera sobre cada paciente na fila.
 - Compara o CPF ou nome do paciente com o critério de busca.
 - Se encontrar uma correspondência, exibe as informações do paciente e interrompe a busca.

Considerações:

- **Complexidade de Tempo:** $O(n)$, menos eficiente que a busca binária para grandes conjuntos de dados.
 - **Uso no Sistema:** Adequado para buscas rápidas em pequenas filas ou quando a fila não está ordenada.
-

FLUXO DA APLICAÇÃO

ADICIONAR/ATUALIZAR PACIENTE

- **Função:** on_patientButton_clicked()
- **Objetivo:** Adicionar um novo paciente à fila ou atualizar as informações de um paciente existente.
- **Processo:**
 - **Validação de Entrada:**
 - Verifica se os campos CPF e Nome não estão vazios.
 - Valida o formato do CPF (XXX.XXX.XXX-XX).
 - **Captura de Sintomas:**
 - Verifica quais sintomas foram selecionados através dos checkboxes.
 - Utiliza operações bitwise para combinar os sintomas em uma única variável.
 - **Verificação de Paciente Existente:**
 - Percorre a fila para verificar se já existe um paciente com o CPF fornecido.
 - **Atualização ou Adição:**
 - Se o paciente existe, atualiza suas informações.
 - Se não existe, cria um novo objeto Patient, adiciona à fila e ao relatório.
 - **Ordenação:**
 - Ordena a fila por sintomas utilizando o selection sort.
 - **Limpeza da Interface:**
 - Limpa os campos de entrada e desmarca os checkboxes de sintomas.

REMOVER PACIENTE

- **Função:** on_removePatient_clicked()
- **Objetivo:** Remover um paciente da fila baseado no CPF fornecido.

- **Processo:**
 - **Validação de Entrada:**
 - Verifica se o campo CPF não está vazio.
 - **Busca e Remoção:**
 - Percorre a fila para encontrar o paciente com o CPF correspondente.
 - Remove o paciente encontrado da fila.
 - **Ordenação:**
 - Reordena a fila por nome para manter a consistência na fila ordenada.
 - **Feedback ao Usuário:**
 - Informa o usuário sobre a remoção bem-sucedida ou se o paciente não foi encontrado.

CONSULTAR PACIENTE

- **Função:** on_consultarPatient_clicked()
- **Objetivo:** Consultar informações de um paciente com base no nome ou CPF fornecido.
- **Processo:**
 - **Captura de Dados:**
 - Obtém o CPF e o nome inseridos pelo usuário.
 - **Verificação do Método de Busca:**
 - Se a busca binária está selecionada (buscaBinariaCheckbox está marcada):
 - Ordena a fila por nome.
 - Realiza a busca binária pelo nome.
 - Exibe as informações do paciente encontrado.
 - Caso contrário:
 - Realiza uma busca sequencial na fila pelo CPF ou nome.

- Exibe as informações do paciente encontrado.
- **Feedback ao Usuário:**
 - Informa se o paciente foi encontrado ou não.

ATUALIZAÇÃO DA INTERFACE

- **Funções:**
 - update_ui(): Atualiza a hora e a data na interface.
 - update_fila(): Atualiza a exibição da fila de pacientes e gerencia o tratamento atual.
- **Processo:**
 - Utiliza temporizadores (QTimer) para chamar essas funções periodicamente.
 - update_ui():
 - Atualiza os elementos da interface que mostram a hora e a data atuais.
 - update_fila():
 - Atualiza os campos que exibem os próximos pacientes na fila.
 - Controla o tempo de tratamento do paciente atual, atualizando uma barra de progresso e removendo o paciente após o término do tratamento.

GERAR RELATÓRIO

- **Função:** gerar_relatorio()
- **Objetivo:** Gerar um relatório detalhado dos pacientes tratados e exibir as informações na interface.
- **Processo:**
 - **Iteração sobre o Vetor de Relatório:**
 - Percorre cada paciente no vetor relatório.
 - **Formatação das Informações:**
 - Concatena as informações do paciente em uma string formatada.
 - **Exibição na Interface:**

- Adiciona as informações formatadas ao *TextBrowser* na interface para visualização.
-

DETALHAMENTO DO PROCESSO

TRATAMENTO DE PACIENTES

O sistema gerencia o tratamento dos pacientes da seguinte maneira:

- **Início do Tratamento:**
 - Quando `tempo_tratamento` está definido como -1, indica que nenhum tratamento está em andamento.
 - Seleciona o primeiro paciente da fila (`fila.at(0)`).
 - Atualiza o campo de chamado na interface com o CPF e ID do paciente.
 - Define `tempo_tratamento` com base nos sintomas do paciente (`fila.at(0).sintomas + 1`), representando a duração do tratamento.
 - Reseta a barra de progresso na interface.
- **Durante o Tratamento:**
 - A cada segundo, a função `update_fila()` é chamada pelo temporizador.
 - Decrementa `tempo_tratamento`.
 - Calcula a porcentagem de conclusão do tratamento e atualiza a barra de progresso.
- **Conclusão do Tratamento:**
 - Quando `tempo_tratamento` atinge 0, o tratamento é concluído.
 - Reseta `tempo_tratamento` para -1.
 - Reseta a barra de progresso.
 - Remove o paciente tratado da fila (`fila.erase(fila.begin())`).

Nota: A prioridade dos sintomas determina a duração do tratamento, com sintomas mais graves, provavelmente resultando em tratamentos mais longos.

INTERFACE

MainWindow

Bem vindo a Software de Gerenciamento Hospitalar

Sun Oct 6 2024

14:59:57

Ultimo Chamado : 0

0%

Aguardando paciente

Aguardando paciente

Aguardando paciente

Aguardando paciente

Aguardando paciente

Aguardando paciente

name

cpf

email

birthday

tosse

nausea

febre

acidente

avc

infeccao

covid

disenteria

Usar Busca Binaria

Consultar por CPF/Nome

Listar Pacientes

Gerar Relatorio

Figura 1.0: Interface no estado inicial

A interface na figura 1.0 é a única tela do sistema, onde todas as operações podem ser realizadas. Apresenta as funcionalidades de cadastro, alteração, consulta e remoção dos dados do paciente, onde de acordo com os sintomas é classificado para receber atendimento urgente ou é colocado em outras posições da fila. É possível gerar relatórios de todos os pacientes atendidos desde que começou o tempo de execução do programa, porém está faltando integração real com sistemas que persistam estes dados devido a estar fora do escopo do projeto solicitado.

Figura 1.1: Bloco de operações com consulta por CPF ou Nome, Listar pacientes na fila e Gerar Relatórios de atendimento.

Na figura 1.1 as operações disponíveis são de consulta por CPF ou Nome, que deverá ser preenchido no bloco de operações na figura 1.2 para mostrar os dados do paciente no textBox da figura 1.1. Listar Pacientes mostra no textBox os dados cadastrais dos pacientes atuais na fila. Gerar relatório apresenta os dados de todos os pacientes tratados no dia.

Figura 1.2: Bloco de operações com entradas utilizadas para cadastro, atualização, consulta e remoção de pacientes.

Na figura 1.2 os campos name, cpf, email, birthday são utilizados para preencher o nome, cpf, e-mail, e data de nascimento do paciente respectivamente, que será utilizado para operações de CRUD. As checkboxes de sintomas comuns são utilizadas

para classificar o nível de prioridade de atendimento desse paciente, e se necessário, atender primeiro.

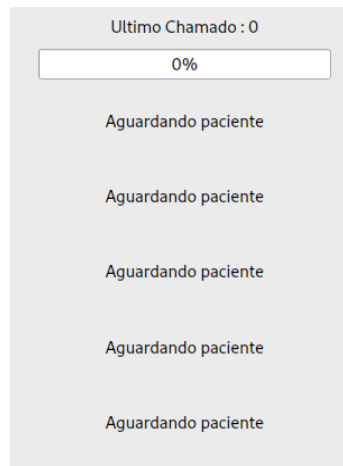


Figura 1.3: Elementos visuais mostrando o CPF do último paciente chamado normalmente e seu ID, a porcentagem de tratamento do paciente sendo atendido e o CPF dos próximos 4 pacientes a serem atendidos.

Na figura 1.3 vemos um bloco focado na interatividade com o usuário, onde consegue ser visualizado quão avançado está o tratamento do paciente atual, os próximos 4 pacientes que serão atendidos e se o último chamado foi por andamento normal da fila ou por emergência.

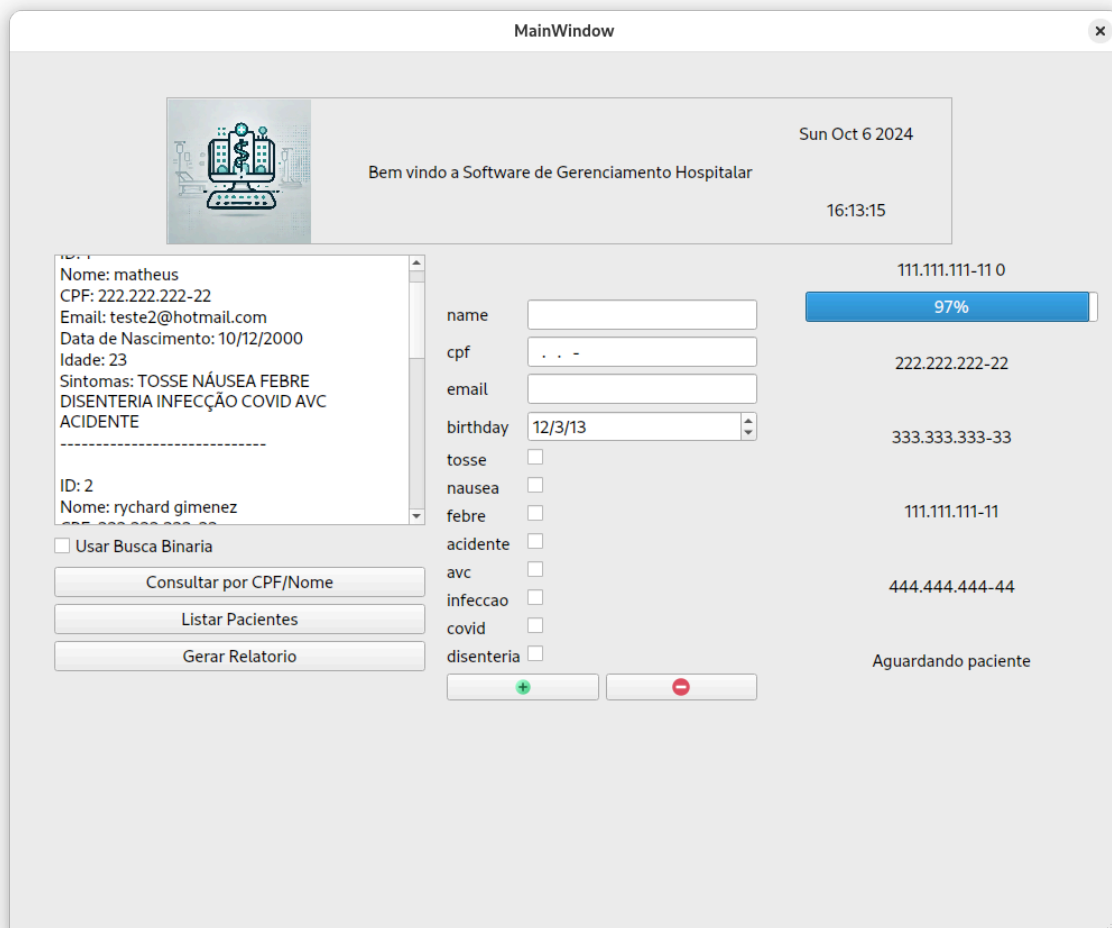



Figura 1.4: Interface no estado funcional, com 3 pacientes na fila e um sendo atendido com 97% de conclusão do tratamento.



Sun Oct 6 2024

Bem vindo a Software de Gerenciamento Hospitalar

16:14:50

Nome: Enrique Ponce

CPF: 111.111.111-11

Email: teste1@gmail.com

Data de Nascimento: 11/10/2000

Idade: 23

Sintomas: TOSSE NÁUSEA COVID ACIDENTE

ID: 3

Nome: tio patinhas

CPF: 444.444.444-44

Email: tiopatinhas@dinheiro.com

Data de Nascimento: 02/03/2003

☐ Usar Busca Binária

Consultar por CPF/Nome

Listar Pacientes

Gerar Relatorio

name

cpf

email

birthday

tosse

nausea

febre

acidente

avc

infeccao

covid

disenteria

. . -

12/3/13

☐

☐

☐

☐

☐

☐

☐

☐

333.333.333-33 2

43%

333.333.333-33

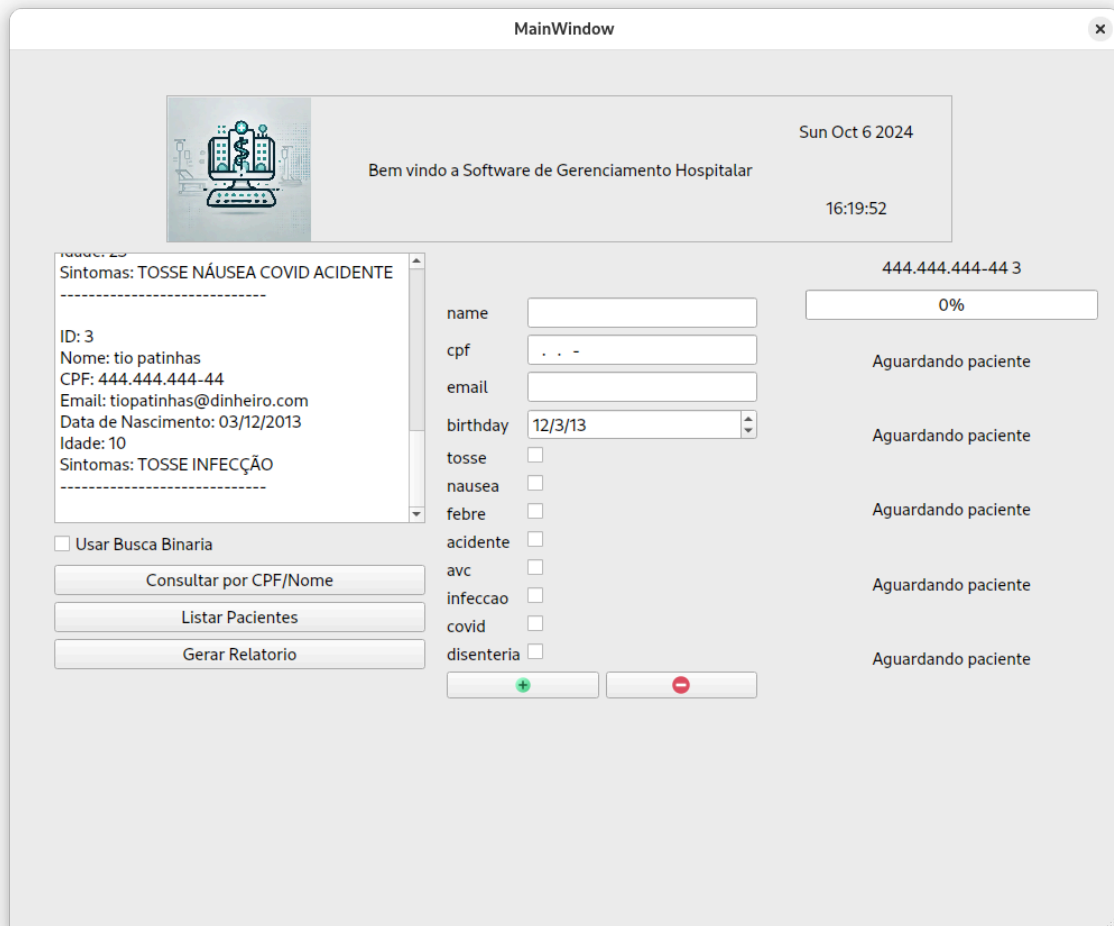
111.111.111-11

444.444.444-44

Aguardando paciente

Aguardando paciente

.Figura 1.5: Interface no estado funcional, após concluir o tratamento do primeiro paciente este foi removido da fila.



.Figura 1.6: Após concluir o tratamento dos pacientes a fila é limpa e o software espera o cadastro de novos pacientes ou a geração de relatórios.

CONCLUSÃO

Este relatório técnico detalhou a estrutura e o funcionamento de um sistema de gerenciamento de pacientes desenvolvido em C++ com a *framework Qt*. O sistema utiliza algoritmos de ordenação e busca, como *Selection Sort* e *Binary Search*, para otimizar operações de gerenciamento da fila de pacientes. A implementação inclui funcionalidades de adição, atualização, remoção e consulta de pacientes, bem como a geração de relatórios detalhados.

As decisões de *design*, como a utilização de vetores para armazenar pacientes e a implementação de algoritmos de busca e ordenação personalizados, foram adequadas para o escopo e complexidade do sistema. Contudo, para escalar adequadamente o sistema, outros algoritmos de ordenação e métodos de consulta devem ser implementados.

Além disso, a interface do usuário é atualizada dinamicamente para refletir as mudanças na fila de pacientes e no status do tratamento, proporcionando uma experiência interativa e informativa para o usuário.

REFERÊNCIA BIBLIOGRÁFICAS

“Qt GUI 6.7.3”, [s.d.]. Disponível em: <<https://doc.qt.io/qt-6/qtgui-index.html>>

“Qt Core 6.7.3”, [s.d.]. Disponível em: <<https://doc.qt.io/qt-6/qtcore-index.html>>

OPENAI, 2024. ChatGPT. Disponível em: <<https://openai.com/chatgpt/>>