Centro Universitário Braz Cubas

GERENCIAMENTO DE ASSISTÊNCIA TÉCNICA

Desenvolvedores:
Ana Luiza Guilherme - Ciência da Computação - 33911410
Luis Carlos de Oliveira Dias Maia - 34448195

Orientador:
Andrea Ono Sakai, Dr.

Mogi das Cruzes - SP
2024/1

1. Desenvolvimento do Cenário

Sistema de Gerenciamento de Assistência Técnica:

Descrição:
Nosso sistema de gerenciamento de assistência técnica foi projetado para otimizar o fluxo de trabalho em uma empresa de assistência técnica, oferecendo soluções eficazes para os desafios enfrentados no dia a dia. Ele visa agilizar o processo de registro, acompanhamento e resolução de problemas técnicos relatados pelos clientes, além de facilitar a gestão interna dos recursos da empresa.

Cenário:
Imagine uma empresa de assistência técnica para dispositivos eletrônicos, como smartphones, tablets e laptops. Diariamente, essa empresa lida com um grande volume de solicitações de clientes que variam desde problemas de hardware até questões de software e configuração. Os clientes entram em contato por meio de diversos canais, como telefone, e-mail ou pessoalmente, buscando assistência rápida e eficiente para seus dispositivos com defeito.

Problemas:
Com base no cenário da assistência técnica para dispositivos eletrônicos, alguns dos principais problemas a serem resolvidos pelo sistema de gerenciamento incluem:

1. **Gestão de Chamados:** Um dos desafios enfrentados por empresas de assistência técnica é a gestão eficiente de um grande volume de chamados de clientes. O sistema deve oferecer uma maneira organizada de registrar, priorizar e acompanhar o status de cada chamado, garantindo que nenhum problema seja negligenciado e que todos sejam tratados dentro do prazo esperado.
2. **Atribuição de Recursos:** A empresa precisa alocar recursos, como técnicos e peças de reposição, de forma eficaz para garantir um atendimento rápido e satisfatório aos clientes. O sistema deve ajudar na programação de serviços, atribuindo chamados aos técnicos disponíveis e gerenciando o estoque de peças para evitar atrasos desnecessários nos reparos.
3. **Comunicação com o Cliente:** Manter os clientes informados sobre o status de seus dispositivos é essencial para garantir a satisfação do cliente. O sistema deve facilitar a comunicação bidirecional entre a empresa e o cliente, permitindo o envio de atualizações automáticas sobre o progresso do reparo e oferecendo canais de contato para perguntas e feedbacks.
4. **Gestão de Estoque:** Uma gestão eficiente do estoque de peças e componentes é fundamental para evitar interrupções no processo de reparo. O sistema deve monitorar o nível de estoque de cada item, gerar alertas quando os estoques estiverem baixos e facilitar a reposição de forma oportuna para garantir a disponibilidade dos materiais necessários.

5. **Análise de Desempenho:** A empresa precisa avaliar constantemente seu desempenho para identificar áreas de melhoria e tomar decisões estratégicas. O sistema deve fornecer dados analíticos sobre o tempo médio de resolução, a satisfação do cliente, o volume de chamados atendidos, entre outros indicadores, para auxiliar na identificação de tendências e na implementação de melhorias contínuas nos processos de assistência técnica.

Funcionalidades:

1. **Registro de Chamados:** Os clientes podem abrir chamados de assistência técnica por meio de um formulário online, fornecendo detalhes sobre o problema enfrentado e informações de contato.
2. **Acompanhamento de Chamados:** Cada chamado é atribuído a um técnico responsável, que pode acompanhar o status do problema, registrar etapas de diagnóstico e solução, e manter contato com o cliente para atualizações.
3. **Agendamento de Serviços:** Os técnicos podem agendar serviços de manutenção e reparo de acordo com a disponibilidade de horários e recursos da empresa, garantindo um fluxo de trabalho organizado e eficiente.
4. **Gestão de Estoque:** O sistema permite o controle de estoque de peças e componentes, alertando automaticamente quando determinados itens estão em baixa quantidade, facilitando a reposição e evitando atrasos nos reparos.
5. **Geração de Relatórios:** A empresa pode gerar relatórios sobre o desempenho do serviço, tempo médio de resolução, satisfação do cliente, entre outros indicadores, auxiliando na tomada de decisões e no aprimoramento contínuo dos processos.

Estruturas Necessárias:
Para implementar esse sistema, serão necessárias as seguintes estruturas de dados:

1. **Fila:** Para organizar a ordem de atendimento dos chamados, garantindo que sejam tratados conforme a chegada.
2. **Lista:** Para armazenar informações dos clientes, técnicos, chamados abertos e histórico de serviços.
3. **Pilha:** Para registrar as etapas de diagnóstico e solução de problemas de cada chamado, permitindo um acompanhamento estruturado do processo.

Algoritmo de Ordenação:
Um algoritmo de ordenação que podemos usar é o **QuickSort**. Ele pode ser aplicado, por exemplo, para ordenar os chamados por prioridade ou tempo de espera, garantindo uma distribuição equitativa dos recursos da empresa e uma resposta mais rápida aos clientes em situações críticas. O QuickSort é eficiente para lidar com grandes conjuntos de dados, o que o torna uma escolha adequada para otimizar o desempenho do sistema em cenários de alta demanda.

TEORIA

2. Algoritmos de Ordenação

Quick Sort

O Quick Sort é um algoritmo de ordenação eficiente e amplamente utilizado. Ele é um algoritmo do tipo "divide e conquista" que funciona escolhendo um elemento como pivô e particionando os elementos ao redor desse pivô, de modo que os elementos menores fiquem antes do pivô e os elementos maiores, depois. O processo é repetido recursivamente para as sublistas à esquerda e à direita do pivô.

Descrição dos Princípios de Funcionamento do Algoritmo

1. Escolha do Pivô: Escolhe-se um elemento da lista para ser o pivô. Existem várias estratégias para escolher o pivô, como escolher o primeiro elemento, o último elemento, o elemento do meio ou um elemento aleatório.
2. Particionamento: Reorganiza-se a lista de modo que todos os elementos menores que o pivô fiquem à sua esquerda e todos os elementos maiores, à sua direita. O pivô, então, está em sua posição final.
3. Recursão: Aplica-se recursivamente o mesmo procedimento às sublistas à esquerda e à direita do pivô.
4. Combinação: Como o pivô já está na posição correta, nenhuma combinação é necessária. O algoritmo termina quando as sublistas estão ordenadas.

Comparação de Eficiência

O Quick Sort é escolhido frequentemente devido à sua eficiência, especialmente em listas grandes. Sua complexidade média é O(nlogn)O(n \log n)O(nlogn), onde nnn é o número de elementos na lista. Em detalhes:

- Melhor Caso: O(nlogn)O(n \log n)O(nlogn) – Ocorre quando a lista é dividida aproximadamente ao meio a cada partição.
- Pior Caso: O(n2)O(n^2)O(n2) – Ocorre quando o pivô escolhido é sempre o maior ou o menor elemento, resultando em partições extremamente desbalanceadas. Este caso pode ser evitado com boas escolhas de pivô, como pivôs aleatórios.
- Caso Médio: O(nlogn)O(n \log n)O(nlogn) – Na prática, o Quick Sort é geralmente muito rápido devido à pequena constante oculta em sua complexidade.

O Quick Sort é mais eficiente que outros algoritmos de ordenação como o Bubble Sort e o Insertion Sort, especialmente para listas grandes, devido à sua complexidade média de O(nlogn)O(n \log n)O(nlogn). No entanto, outros algoritmos como Merge Sort também têm complexidade O(nlogn)O(n \log n)O(nlogn) no pior caso, mas podem ser menos eficientes devido à necessidade de espaço adicional para a combinação das sublistas.

3. Estruturas de Dados

Pilha (Stack)

Conceito e Principais Características:

Uma pilha é uma estrutura de dados linear que segue o princípio LIFO (Last In, First Out), ou seja, o último elemento inserido é o primeiro a ser removido. É semelhante a uma pilha de pratos, onde você só pode adicionar ou remover o prato do topo da pilha.

Características:

- **LIFO (Last In, First Out):** O último elemento inserido é o primeiro a ser removido.
- **Inserção e Remoção no Topo:** Os elementos são inseridos e removidos apenas no topo da pilha.
- **Operações Básicas:** Empilhar (push) e Desempilhar (pop).

Operações Básicas

1. **Empilhar (push):** Adiciona um elemento ao topo da pilha.
2. **Desempilhar (pop):** Remove e retorna o elemento do topo da pilha.
3. **Topo (top/peek):** Retorna o elemento do topo sem removê-lo.
4. **Vazia (isEmpty):** Verifica se a pilha está vazia.

Fila (Queue)

Conceito e Principais Características

Uma fila é uma estrutura de dados linear que segue o princípio FIFO (First In, First Out), ou seja, o primeiro elemento inserido é o primeiro a ser removido. É semelhante a uma fila de pessoas, onde a primeira pessoa a entrar na fila é a primeira a ser atendida.

Características:

- **FIFO (First In, First Out):** O primeiro elemento inserido é o primeiro a ser removido.
- **Inserção na Cauda e Remoção na Cabeça:** Os elementos são inseridos na cauda (final) da fila e removidos da cabeça (início) da fila.
- **Operações Básicas:** Enfileirar (enqueue) e Desenfileirar (dequeue).

Operações Básicas

1. **Enfileirar (enqueue):** Adiciona um elemento ao final da fila.
2. **Desenfileirar (dequeue):** Remove e retorna o elemento do início da fila.
3. **Cabeça (front):** Retorna o elemento do início da fila sem removê-lo.
4. **Vazia (isEmpty):** Verifica se a fila está vazia.

Lista

Conceito e Principais Características

Uma lista é uma coleção de elementos ordenados que podem ser acessados por um índice. Pode ser implementada como um array (lista estática) ou como uma lista encadeada (lista dinâmica).

Características:

- **Acesso por Índice:** Elementos podem ser acessados diretamente através de um índice.
- **Inserção e Remoção:** Elementos podem ser inseridos e removidos em qualquer posição.
- **Tamanho Dinâmico (em listas encadeadas):** O tamanho da lista pode crescer ou diminuir conforme necessário.

Operações Básicas

1. **Inserção (insert):** Adiciona um elemento em uma posição específica.
2. **Remoção (delete):** Remove um elemento de uma posição específica.
3. **Acesso (get):** Retorna o elemento de uma posição específica.
4. **Tamanho (size):** Retorna o número de elementos na lista.

4. Continuação

Instruções de Uso

1. **Inicialização:** O sistema inicializa os dados necessários ao chamar a função inicializarDados().
2. **Registro de Clientes:** Utilize a função registrarCliente(char *nome, char *contato) para adicionar novos clientes ao sistema.
3. **Registro de Técnicos:** Utilize a função registrarTecnico(char *nome) para adicionar novos técnicos ao sistema.
4. **Registro de Chamados:** Utilize a função registrarChamado(char *descricao, int clienteId) para registrar novos chamados de assistência técnica.
5. **Atribuição de Técnicos:** Utilize a função atribuirTecnico(int chamadoId, int tecnicoId) para atribuir um técnico a um chamado específico.
6. **Atualização de Chamados:** Utilize a função atualizarChamado(int chamadoId, char *atualizacao) para adicionar atualizações aos chamados em andamento.
7. **Finalização de Chamados:** Utilize a função finalizarChamado(int chamadoId) para marcar um chamado como finalizado.
8. **Registro de Peças no Estoque:** Utilize a função registrarPeca(char *nome, int quantidade) para adicionar novas peças ao estoque.
9. **Atualização do Estoque:** Utilize a função atualizarEstoque(char *nome, int quantidade) para atualizar a quantidade de peças no estoque.

10. **Geração de Relatório:** Utilize a função gerarRelatorio() para imprimir um relatório dos chamados de assistência técnica.

O código está organizado em funções específicas para cada tarefa, o que facilita a manutenção e a reutilização do código. Cada função está documentada com comentários explicativos para descrever seu propósito e funcionamento.

5. Conclusão

Durante o desenvolvimento deste projeto, enfrentei desafios significativos, especialmente com a implementação e gestão de estruturas de dados.
A modularização do código foi um desafio adicional, mas crucial para tornar o programa mais organizado e fácil de manter. Aprendi a importância de escrever funções claras e bem definidas, o que facilitou a manutenção e a reutilização do código.
A documentação do código foi fundamental para compreender melhor o fluxo do programa e identificar pontos de melhoria. Esse processo de adicionar comentários explicativos tornou o código mais acessível para outros desenvolvedores e para mim mesma no futuro.
Em resumo, este projeto foi uma oportunidade valiosa para aplicar conceitos aprendidos em sala de aula, superar dificuldades com estruturas de dados e aprimorar minhas habilidades em programação.