Alunas: Ana Carolyne Pereira de Souza, Caroline Bohadana Rodrigues Viana, Beatriz Gerogiannis Mendonça, Geovanna cunha Andrade Silva

Relatório:

INTRODUÇÃO:

A escolha da aplicação recai sobre um gerenciador de turnos para a ala de emergência de um hospital pediátrico, operando por meio de uma estrutura de fila implementada como um Tipo Abstrato de Dado (TAD) em linguagem C. Compreendendo um conjunto de 23 funções interdependentes, seu propósito central é abordar de maneira eficaz os desafios inerentes ao escalonamento e à alocação otimizada dos profissionais de saúde na ala de emergência infantil.

Essa implementação tem como objetivo primordial resolver complexidades ligadas ao planejamento de horários, garantindo uma distribuição precisa dos recursos humanos, tais como médicos, enfermeiros e técnicos de enfermagem, com base nas prioridades e necessidades específicas de cada turno. Sua missão fundamental é assegurar que cada turno seja provido de uma equipe capacitada para enfrentar a demanda variável de pacientes, eliminando riscos de sobrecarga ou insuficiência de pessoal em horários críticos ou áreas específicas do hospital.

Os requisitos funcionais são:

inicia_lista: essa função inicia a lista de funcionários, sendo esta uma lista que tem seus nós alocados dinamicamente. Dentro dessa função, iniciamos cada lista utilizando a função *"strcmp"*, que inicia cada lista de funcionário de acordo com sua especialidade/cargo.

lista_vazia: essa função permite que o primeiro nó de cada lista aponte para NULL, fazendo com que as listas de funcionários estejam, inicialmente, vazias.

insercao_funcionario: essa função permite ao usuário inserir um funcionário novo a partir da inserção de informações básicas, como o nome do funcionário, especialização e a disponibilidade.

insere_turno: permite ao usuário inserir em qual turno o funcionário trabalhará a partir da inserção de informações básicas, como ID, especialização e a escolha do turno que o funcionário será alocado.

remove_turno: essa função permite ao usuário remover a partir da inserção do ID, especialização e turno que será removido.

troca_turno: possibilita o usuário trocar o turno do funcionário a partir da inserção de informações, como a especialização, ID, turno atual e o turno desejado.

restrição: a função restrição permite ao usuário checar se o funcionário já trabalhou a quantidade de horas máximas permitidas (60h semanais). Essa checagem é feita a partir da inserção das informações especialização e ID.

troca: essa função permite trocar os IDs dos médicos a partir do algoritmo de ordenação eficiente Quick Sort, recebendo como parâmetro os IDs e trocando-os.

particionar: essa função define o pivô e particiona o vetor.

QuickSort: função de ordenação Quick Sort que consiste, primeiramente, na escolha de um pivô, que é um elemento arbitrariamente escolhido do vetor. Após isso, particiona-se o vetor, ou seja, o vetor deve ser reorganizado de forma que os elementos menores que o pivô fiquem de um lado, e os maiores, de outro. Por ultimo, recursivamente, ordena-se as sub-listas abaixo e acima do pivô.

busca binária: a função de busca binária utiliza o algoritmo de busca eficiente que permite que, desde que o vetor esteja ordenado, seja possível encontrar determinado ID de um funcionário específico.

consulta_disponibilidade: essa função permite que o usuário consulte a disponibilidade de um determinado funcionário a partir da inserção da especialização e do ID do mesmo.

imprime: a função imprime permite que o usuário imprima quais funcionários estão trabalhando.

adicionar_horas: permite ao usuário adicionar as horas trabalhadas do funcionário através da inserção do ID, da especialização e a quantidade de horas que serão inseridas.

limpa_buffer: limpa o buffer do teclado.

ja_foi_inserido: Esta função evita que o usuário insira o mesmo ID para a mesma lista de funcionários.

libera_memoria: Libera a memória alocada para cada lista de departamentos e seus respectivos funcionários.

calcula_salario: proporciona ao usuário calcular o salário do funcionário a partir da inserção da função do mesmo. O valor de cada hora de acordo com sua função está no algoritmo, assim como o acesso das horas trabalhadas.

merge: essa função de fato vai ordenar de forma eficiente os nomes por ordem alfabética

mergeSort: Divide o vetor em duas partes iguais, que serão sucessivamente divididas até que reste apenas um ou dois elementos em cada parte.

ordena_nome: utiliza o merge sort (algoritmo de ordenação eficiente) para ordenar os nomes dos funcionários por ordem alfabética.

bubbleSort: algoritmo ineficiente de ordenação utilizada para ordenar os salários.

Em suma, o usuário deve ser capaz de inserir um novo funcionário, inserir um novo turno, remover turno, trocar turno, verificar se há restrições ao adicionar horas, consultar disponibilidade do funcionário, consultar quantas horas o

funcionário trabalhou, adicionar horas trabalhadas, imprimir quais funcionários estão trabalhando, calcular o salário de uma determinada especialidade/função e ordenar alfabeticamente os nomes dos funcionários.

OBJETIVO:

O objetivo desta aplicação é desenvolver um programa robusto para gerenciar a alocação de turnos na equipe de emergência de um hospital infantil. O programa visa eliminar falhas e inconsistências, garantindo um escalonamento eficiente dos profissionais de saúde. Ao otimizar a distribuição dos profissionais, evita-se sobrecarga ou falta de pessoal em horários ou áreas específicas, contribuindo para a operação fluida da ala de emergência e prevenindo a sobrecarga do hospital.

COMPLEXIDADE COMPUTACIONAL:

inicia_lista = O(1)

aloca_funcionario = O(1)

insercao_funcionario = O(n)

insere_turno = O(n)

remove_turno = O(n)

troca_turno = O(n)

lista_vazia = O(n)

restricao = O(n)

troca = O(1)

particionar = O(n)

quickSort = no melhor caso O(n log n), no pior caso O(n^2)

busca_binaria = O(log n)

consulta_disponibilidade = a complexidade da ordenação é O(n log n) em média, mas no pior caso pode ser O(n^2).

Imprime = O(n)

adicionar_horas = O(n)

limpa_buffer = O(1)

ja_foi_inserido = O(n)

libera_memoria = O(N_total), onde N_total é a soma dos números de funcionários em todas as listas de diferentes especializações.

calcula_salario = O(n^2)

merge = O(n)

mergeSort = O(n log n)

ordena_nome = O(n log n)

bubbleSort = O(n^2)

DESENVOLVIMENTO:

Implementação:

O programa implementa diversas funções que, juntas, são capazes de gerenciar os funcionários da ala de emergência de um hospital pediátrico. Ele usa uma estrutura de lista ligada para armazenar informações sobre funcionários, como nome, especialização, disponibilidade de turnos e horas trabalhadas.

Além disso, o programa possui um menu interativo que permite ao usuário realizar várias ações, incluindo adicionar um novo funcionário, inserir e remover turnos de trabalho, consultar a disponibilidade de um funcionário, trocar turnos, checar horas trabalhadas, adicionar horas trabalhadas, imprimir lista de funcionários que estão trabalhando, calcular salário e ordenar os funcionários alfabeticamente.

A implementação geral envolve alocação dinâmica de memória para a lista de funcionários, uso de funções para manipulação da lista e interação com o usuário através do menu. As informações dos funcionários são coletadas do usuário e armazenadas na lista. O programa fornece recursos para manipulação e consulta desses dados de funcionários de acordo com as opções escolhidas pelo usuário no menu.

RESULTADOS:

O código apresenta um sistema de gerenciamento de funcionários na área da saúde que é eficiente e abrangente. Ele permite a execução de várias operações importantes, como adicionar, consultar e atualizar informações dos funcionários. O sistema é notável por sua rápida execução. As saídas são diretas e relevantes, eliminando falhas e facilitando decisões informadas. Com uma interface amigável, desempenho ágil e resultados precisos, o código se destaca como uma ferramenta valiosa para o gerenciamento eficaz de recursos humanos na área da saúde.

CONCLUSÃO:

O projeto passou por um processo de refinamento detalhado e foi extensivamente testado em várias ocasiões. Essa abordagem meticulosa resultou em um sistema que opera de forma fluida e eficiente. Cada funcionalidade e operação foi submetida a iterações rigorosas de teste,

garantindo um desempenho constante e livre de falhas perceptíveis. O compromisso com a melhoria contínua se reflete na estabilidade e confiabilidade alcançadas pelo sistema.