UNIVERSIDADE
FEDERAL DO CARIRI

UNIVERSIDADE FEDERAL DO CARIRI
CENTRO DE CIÊNCIAS E TECNOLOGIA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Projeto de Implementação – Parte I

Professor: Carlos Vinícius Gomes Costa Lima

Wanderson Faustino Patricio - MAT: 2022005052
Francisco Anderson Maciel Cruz - MAT: 2022005876

Juazeiro do Norte, 22 de outubro de 2023

1 Apresentação

Nesse projeto foi implementado um sistema de fila de prioridades das espaçonaves presentes próximas à passagem de um túnel interdimensional, esperando a sua vez de passar. Como há muitas espaçonaves chegando, é preciso colocar em ordem de prioridade, ou seja, aquela com maior prioridade devem estar na primeira posição.

As naves podem ter suas prioridades alteradas, e, portanto, suas posições na fila alteradas.

A fila tem a propriedade de a nave de maior prioridade está na primeira posição, sempre enviando a nave mais necessitada no atual momento para fora do portal.

2 Apresentação da implementação das funções

2.1 Passageiro criarPassageiro(char* nome, int idade, int doente, int especializado, int origem, int id)

Esta função cria e retorna um novo objeto Passageiro com os atributos especificados, como nome, idade, estado de saúde, especialização, origem e ID.

2.2 Nave criarNave(int id, char recursos[][50], Passageiro passageiros[])

A partir das informações dos passageiros e dos recursos, bem como do id, uma nova nave é instanciada e retornada como valor da função.

2.3 Heap* criarHeap()

É criado um espaço na memória com a capacidade para a heap. O tamanho dela é inicialmente setado para zero. É retornado um ponteiro para o endereço de memória reservado.

2.4 void liberarHeap(Heap * h)

Essa função libera o espaço de memória reservado para a Heap h através da função free.

2.5 void infoPassageiro(Passageiro p)

Esta função exibe informações sobre um Passageiro (p). Ela imprime o nome, a idade, o ID, o planeta de origem, a especialização (caso possua) e o estado de saúde (doente ou não) do passageiro.

2.6 void infoNave(Nave n)

Esta função exibe informações detalhadas sobre uma Nave (n). Ela imprime o ID da nave, sua prioridade, a lista de recursos a bordo e informações sobre cada passageiro a bordo, chamando a função infoPassageiro para exibir os detalhes de cada passageiro.

2.7 void verificarPrioridade(Nave* n)

Esta função atribui uma nova prioridade aleatória a uma Nave (n) com uma certa probabilidade. Ela gera um número aleatório e, se esse número for igual a 10, atualiza a prioridade da Nave com um valor aleatório entre 1 e 100.

2.8 void ordenaHeap(Heap* h)

Esta função realiza uma ordenação de heap nos elementos da estrutura de Heap (h). Ela garante que os objetos Nave na fila estejam dispostos em ordem decrescente de prioridade. A função chama a função descer para manter a propriedade de heap.

2.9 void adiciona_passageiros(char* file_passageiros, Heap* h)

Esta função lê os dados dos passageiros de um arquivo CSV especificado (file_passageiros) e preenche uma estrutura de Heap (h) com esses dados. O programa processa o arquivo CSV linha por linha, criando objetos Passageiro e atribuindo-os às naves apropriadas na fila.

2.10 void adiciona_recursos(char* file_passageiros, Heap* h)

Similar à função adiciona_passageiros, esta função lê os dados dos recursos de um arquivo CSV especificado (file_recursos) e adiciona as informações de recursos às naves na fila.

2.11 void trocar(Heap* h, int i, int j)

Essa função troca as naves de índice i e j de posição na fila de prioridades.

2.12 void subir(Heap* h, int $index$)

A função subir utiliza-se da ideia de representar a fila de prioridades como uma árvore binária. Visto que $h[i].prioridade \geq h[2i].prioridade$ e $h[i].prioridade \geq h[2i+1].prioridade$ podemos considerar a heap como uma árvore binária em que o nó pai tem prioridade maior que os seus nós filhos.

Na função subir é verificado se o nó atual tem prioridade menor que ou igual ao seu pai, caso contrário ele é trocado de lugar com seu pai, e a função é chamada recursivamente, tomando o nó pai ($h\,[\lfloor index/2 \rfloor]$) como paramentro da função, até que o nó alcance a posição correta dele na fila.

2.13 void descer(Heap* h, int $index$)

Similar à função subir, é verificado se a prioridade do nó atual é maior que ou igual a seus filhos, caso contrário ele é trocado de posição com o seu filho de maior prioridade. A função é chamada recursivamente com seu filho de maior prioridade como paramentro da função, até que o nó alcance a posição correta dele na fila.

2.14 void inserirNave(Nave n, Heap* h)

Para a inserção de uma nova nave esta é colocada na primeira posição livre da heap ($h[h.tamanho + 1]$, caso possível), é aumentado o tamanho da fila em uma unidade, e após isso é utilizada a função subir na posição $h.tamanho$ até que a nova nave alcance a posição correta.

2.15 void removerNave(Heap* h)

Para a remoção sempre removeremos o elemento da primeira posição da heap, que tem a maior prioridade.

Trocaremos a primeira nave com a última da fila, diminuiremos o tamanho da heap e aplicaremos o algoritmo descer na primeira posição até que o nó alcance a posição correta dele na fila.

2.16 void exibirHeap(Heap* h)

Essa função exibe em ordem (da 1 até a $h.tamanho$) todas as naves presentes na fila, usando a função infoNave($h.nave[i]$).

2.17 void inicializar()

A função de inicialização é a função de interação com o usuário. A partir dessa função são expostas opções para a utilização da fila de prioridades de naves.

Ao iniciar esta função utilizamos as funções adiciona_recursos e adiciona_passageiros para inicializar a heap com algumas naves predefinidas, não sendo necessário inserir nave por nave para utilizar o programa.

O usuário tem 5 funcionalidades para utilizar:

caso 1: **Enviar a primeira nave pelo portal**.

É utilizada a função removerNave e exibida em tela as informações da nave que acabou de partir.

caso 2: **Exibir a informacao de todas as naves da fila**.

Todas as naves da fila são exibidas em tela com suas infomações a partir da função exibirHeap.

caso 3: **Alterar a prioridade de uma nave**.

O usuário escolhe o índice de alguma nave e digita o novo valor de prioridade para a nave escolhida. Dependendo se sua prioridade aumentou ou diminuiu utilizamos a função subir ou descer para ajeitar a sua posição na fila.

caso 4: **Inserir uma nave na fila**.

O usuário digita todas as informações necessárias para a nova nave. Após isso utilizamos a função inserirNave para adicionar a nova nave na fila.

caso 5: **Exibir as informacoes de uma nave**.

O usuário escolhe um índice (i) e as informações da nave $h.nave[i]$ são exibidas pela função infoNave($h.nave[i]$).

2.18 void finalizar()

Exibe uma mensagem de despedida ao usuário encerrar o programa.