

**Algoritmos e Estruturas de Dados I**

**Colegiado de Ciência da Computação - Salgueiro (PE)**

**Bacharelado em Ciência da Computação**

Filas e Listas

Atividade Avaliativa

Aluno(a):Pedro Vinícius da Silva Ferreira

Data: 23/09/2021

**INSTRUÇÕES**

* Cada questão será pontuada de 0 (zero) a 100 (cem) pontos. A nota da Atividade Avaliativa será a média aritmética simples de todas as questões.
* Essa Atividade Avaliação **DEVE** ser respondida individualmente. Qualquer resposta parecida com a de outro aluno (critério subjetivo que cabe ***apenas*** ao professor) será desconsiderada para a correção de **ambos** os alunos.
* Coloque as respostas das questões imediatamente abaixo do seu cabeçalho. Seja organizado. Questões mal formatadas serão desconsideradas para a correção.

**QUESTÕES**

1. Suponha uma fila de inteiros implementada em vetor, e as seguintes definições globais em C:

**int** queue[100]; // armazena os elementos da fila

**int** front, // front: ponteiro para o início da fila

rear; // rear: ponteiro para o final da fila

Implementamos em sala de aula a operação

**int** remove() {

// salva elemento que será removido da fila

**int** x = queue[0];

// desloca todos os elementos da fila uma posição abaixo

**for** (**int** i = 0; i < rear; i++)

queue[i] = queue[i+1];

// atualiza o ponteiro de final da fila

rear -= 1;

// retorna o elemento salvo (desenfileirado)

**return** x;

}

de maneira que quando um item é eliminado a fila inteira é deslocada no sentido do início do vetor. Dessa forma a fila não precisa mais ter um ponteiro front porque o elemento na posição 0 do vetor será sempre o elemento do início da fila. Nessa implementação, a fila vazia é representada com rear igual a -1.

Um método alternativo é continuar utilizando os ponteiros front e rear e adiar o deslocamento dos elementos no vetor até que rear seja igual ao último índice do vetor. Quando essa situação ocorre e faz-se uma tentativa de **inserir** um novo elemento na fila, a fila inteira é deslocada para baixo, de modo que, nesse momento, o primeiro elemento da fila fique na posição 0 do vetor.

Implemente as operações, em C,

**bool** empty() { ... }

**void** insert(**int** x) { ... }

**int** remove() { ... }

usando esse método alternativo.

**obs.:** o subprograma deve ser robusto e tratar todos os casos excepcionais que possam acontecer durante a sua execução. Comente o seu código indicando as suas escolhas de implementação. Códigos sem comentários não serão considerados para a correção.

**RESPOSTA:**

bool empty() {

// Se front for maior que rear retornará true, indicando que a fila está vazia

return (front > rear);

}

void insert (int x) {

// Se rear for diferente do último índice do vetor entrará no if

if(rear != 99){

// queue na posição rear + 1 receberá o valor a ser inserido

queue[rear+1] = x;

// rear será incrementado por 1

rear = rear + 1;

// Se rear for igual ao último índice do vetor entrará no else

} else {

// Se queue na posição 0 for igual a zero, que indica que a posição está vazia, entrará no if

if(queue[0] == 0){

// Desloca todos os elementos da fila uma posição abaixo, já que o espaço anterior está vazio.

for (int i = 0; i < rear; i++){

queue[i] = queue[i+1];

}

} else {

// Caso o espaço anterior não esteja vazio, terá um print alertando que não foi possível adicionar o valor.

printf("Não é possível adicionar um novo valor.");

}

}

}

int remove() {

// salva elemento que será removido da fila

int x = queue[front];

// queue na posição front receberá 0, que indicará que a posição está vazia

queue[front] = 0;

// front irá ser incrementado a cada vez que um item for removido

front = front+1;

// retorna o elemento salvo (desenfileirado)

return x;

}

1. Suponha uma lista simplesmente encadeada implementada com alocação dinâmica de memória e as seguintes definições globais em C:

// registro que representa um nó da lista encadeada

**struct** Node {

**int** info; // armazena a informação do nó

**struct** Node \*next; // armazena o ponteiro para o próximo elemento da lista encadeada

};

**typedef struct** Node node\_t;

Implemente a operação

**void** concat(node\_t \*\*l1, node\_t **\***l2) { ... }

responsável por concatenar duas listas encadeadas, acrescentando ao final da lista l1 a lista encadeada apontada por l2. O ponteiro nulo deve ser representado pelo valor **NULL**.

**obs.:** o subprograma deve ser robusto e tratar todos os casos excepcionais que possam acontecer durante a sua execução. Comente o seu código indicando as suas escolhas de implementação. Códigos sem comentários não serão considerados para a correção.

**RESPOSTA:**

void concat(node\_t \*l1, node\_t \*l2){

// Se a lista 1 for vazia, ela irá receber a lista 2

if (l1 == NULL) {

l1 = l2;

// Se a lista 1 não for vazia, entrará no else

} else {

// Variáveis auxiliares para percorrer a lista

node\_t\* aux1;

node\_t\* aux2;

// A variável auxiliar 1 receberá a lista 1

aux1 = l1;

// Enquanto a variável auxiliar 1, que receberá o próximo elemento apontado por ela, não for nulo, continuará no do

do {

aux2 = aux1;

aux1 = aux1->prox;

}

while(aux1 != NULL);

// O próximo valor da variável auxiliar 2 recebe a lista 2

aux2->prox = l2;

}

}

1. Suponha uma lista simplesmente encadeada implementada em vetor e as seguintes definições globais em C:

// registro que representa um nó da lista encadeada

**struct** Node {

**int** info; // armazena a informação do nó

**struct** Node \*next; // armazena o ponteiro para o próximo elemento da lista encadeada

};

**typedef struct** Node node\_t;

node\_t nodes[100]; // armazena o nós livres para a lista

Suponha que os nós livres (desalocados) do vetor nodes estão estruturados como uma lista encadeada. Esta lista de nós é referenciada através de um ponteiro global

node\_t \*free\_nodes = **NULL**;

Inicialmente, a lista de nós livres está estruturada em sua ordem natural, ou seja, o nó de índice 0 aponta para o nó de índice 1, o nó de índice 1 aponta para o nó de índice 2… e assim por diante até o último nó, de índice 99, que tem o valor **NULL** como campo next indicando um ponteiro nulo. Ou seja, a lista de nós livres é inicializada como

// inicializa a lista encadeada de nós livres

**for** (**int** i = 0; i < 99; i++) {

// inicializa o campo de informação

nodes[i].info = 0;

// inicializa o campo ponteiro, apontando para o próximo nó da lista

nodes[i].next = &(nodes[i+1]);

}

// inicializa o campo de informação do último nó da lista

nodes[99].info = 0;

// inicializa o campo ponteiro do último nó da lista, apontando para **NULL**

nodes[99].next = **NULL**;

// inicializa o ponteiro para a lista de nós livres

free\_nodes = &(nodes[0]);

Implemente a operações

node\_t \* alloc\_node() { ... }

**void** free\_node(node\_t \*n) { ... }

de forma a sempre manter os nós livres (desalocados) estruturados na lista encadeada referenciada por free\_nodes.

**obs.:** o subprograma deve ser robusto e tratar todos os casos excepcionais que possam acontecer durante a sua execução. Comente o seu código indicando as suas escolhas de implementação. Códigos sem comentários não serão considerados para a correção.

**RESPOSTA:**

node\_t \* alloc\_node(){

// Alocação de um novo nó, com utilizando malloc

node\_t \*aux = malloc(sizeof(node\_t));

if (aux == NULL) {

aux->info = 0;

aux->next = NULL;

}

return aux;

}

void free\_node(node\_t \*n){

// Se n for diferente de NULL entrará no if

if (n != NULL) {

//Uma variável ultimo\_no será criada

node\_t \* ultimo\_no;

// Se free\_nodes na posição next for NULL entrará no if

if (free\_nodes->next == NULL) {

ultimo\_no = free\_nodes;

} else {

ultimo\_no = free\_nodes->next;

// O último no será concatenado

do {

ultimo\_no = ultimo\_no->next;

} while (ultimo\_no->next != NULL)

}

// O último no na posição next recebe n

ultimo\_no->next = n;

}

}

1. Suponha uma lista simplesmente encadeada implementada com alocação dinâmica de memória, e as seguintes definições globais em C:

// registro que representa um nó da lista encadeada

**struct** Node {

**int** info; // armazena a informação do nó

**struct** Node \*next; // armazena o ponteiro para o próximo elemento da lista encadeada

};

**typedef struct** Node node\_t;

// aloca um novo nó para a lista encadeada. Retorna um ponteiro diferente de **NULL** se a alocação for bem sucedida, e **NULL** caso contrário.

node\_t \* alloc\_node() { ... }

Implemente a operação

node\_t \* search\_and\_insert(node\_t \*\*l, **int** x) { ... }

que procura em uma lista de inteiros o nó de informação x e retorna o seu endereço. Caso a lista não possua um nó com informação x, ele deve ser inserido na lista como o último elemento, e o seu endereço retornado pelo subprograma.

**obs.:** o subprograma deve ser robusto e tratar todos os casos excepcionais que possam acontecer durante a sua execução. Comente o seu código indicando as suas escolhas de implementação. Códigos sem comentários não serão considerados para a correção.

**RESPOSTA:**

node\_t \* search\_and\_insert(node\_t \*\*l, int x){

// For para percorrer a lista e procurar o valor

for (node\_t \*aux = (\*l); aux != NULL; aux = aux->next){

// Se o campo informação da variável aux for igual ao valor procurado, retornará o endereço da lista aux

if (aux->info == x){

return aux;

}

}

// Se o valor não foi encontrado, será adicionado através do insert

return insert(l, x);

}