Marcos Antônio Lommez Cândido Ribeiro - 771157

Tipos Abstratos de Dados Lineares – Lista

• Diferencie remoção lógica e física

Quando um arquivo é deletado o que é feito na verdade é apenas retira-lo da lista de memória que está sendo utilizada pelo computador, se tornando assim não mais acessível por meios normais. Mesmo tendo seu ponteiro desfeito as informações (bits) continuam na memoria sem nenhuma mudança ate que essa memoria seja utilizada novamente, a partir daí as informações são reescritas por cima desses dados. Esta é a exclusão lógica, um endereço de memoria é apenas abandonado para ser reescrito, por outro lado existe a remoção física que é a combinação da exclusão com a formatação do disco, onde os dados não são apenas excluídos, mas também limpos bit a bit da memória do sistema.

• Diferencie formatação lógica (rápida) e física

O termo formatação se refere ao ato de preparar uma memoria para pode ser utilizada, separando suas unidades e limpando partes necessárias para uso, etiquetando os lugares onde as informações serão guardadas, caso contrário o disco não poderá ser utilizado. A formatação logica se refere ao ato de apenas separar memoria livre para uso sem se preocupar com o estado atual do disco e possíveis lixos armazenados, apenas o sobrescrevendo-os, ou se resumindo em apenas habilitar o disco a receber um sistema operacional. A formatação física por outro lado é feita pelo próprio fabricante que é a preparação da memoria e sua etiquetagem em setores de memória para a utilização.

• No seu SO, o que acontece quando enviamos um arquivo para a lixeira? E quando o excluímos definitivamente?

A lixeira do sistema é uma sala de armazenamento para os arquivos que se deseja excluir da memória, seus ponteiros são alocados de forma que ainda possam ser recuperados de forma fácil ate que se tenha certeza que aqueles dados possam ser “perdidos”. Quando um dado é excluído da lixeira por outro lado ele tem apenas seu ponteiro excluído, não é feito a sua remoção física, mas os dados ainda podem ser recuperados através de outras maneiras caso não tenham sido sobrescritos .

Tipos Abstratos de Dados Lineares – Pilha

• Dado o código da lista (métodos II, IF, I, RI, RF e R), como podemos alterá-lo para criarmos uma pilha? Apresente as duas soluções possíveis. Por que a segunda não é interessante?

Para criarmos uma Pilha usamos **Inserir Fim** e **remover Fim** ou **Inserir Início** e **Remover Início**.

Dentro das duas possíveis soluções Inserir e Remover no Início são ruins porque requerem que toda a tabela precise ser realocada para cada inserção/remoção, tornando este método não eficiente

Tipos Abstratos de Dados Lineares – Fila

• Dado o código da lista (métodos II, IF, I, RI, RF e R), como podemos alterá-lo para criarmos uma fila? Apresente as duas soluções possíveis e mostre a desvantagem de cada uma

Para criarmos uma Fila usamos **Inserir Início** e **Remover Fim** ou **Inserir Fim** e **Remover Início**.

Dentro das duas possíveis a primeira **II** + **RF** gera o problema da inserção não eficiente, porque todas as veze que um dado for inserido o array inteiro precisa ser movido para uma posição a mais para que o novo dado possa ser alocado. Por contrapartida o método **IF + RI** gera o problema da remoção não eficiente, se fazendo da necessidade de realocar todo o array para cada remoção feita na fila.

Exercícios de Fila Circular

- Exercicio

- 0 % 5 = 0
- 1 % 5 = 1
- 2 % 5 = 2
- 3 % 5 = 3
- 4 % 5 = 4

- Faça o quadro de memória do programa

```
n = 0;
n = (n + 1) % 5;
n = (n + 1) % 5;
n = (n + 1) % 5;
n = (n + 1) % 5;
n = (n + 1) % 5;
n = (n + 1) % 5;
n = (n + 1) % 5;
n = (n + 1) % 5;
n = (n + 1) % 5;
```

Comando	Var	Valor
n = 0;	n	0
n = (n + 1) % 5;	n	1
n = (n + 1) % 5;	n	2
n = (n + 1) % 5;	n	3
n = (n + 1) % 5;	n	4
n = (n + 1) % 5;	n	0
n = (n + 1) % 5;	n	1
n = (n + 1) % 5;	n	2
n = (n + 1) % 5;	n	3
n = (n + 1) % 5;	n	4
n = (n + 1) % 5;	n	0

- Implemente o método boolean isVazio()

```
boolean isVazio(int i) {
    boolean vazio = false; // se estiver fora do range
    if (i >= 0 && i <= array.length) {
        vazio = true; // esta no range
        if (primeiro < ultimo) {
            if (i >= primeiro && i < ultimo) {
                vazio = false; // nao esta alocado
            }
        } else {
            if (i >= primeiro || i < ultimo ) {
                vazio = false; // nao esta alocado
            }
        }
    }
    return vazio;
}
```

- Implemente o método void mostrarRec() de forma recursiva

```
void mostrarRec() {
    System.out.print("[");
    mostrarRec(primeiro, ultimo);
    System.out.println("]");
}

void mostrarRec(int i, int j) {
    if (i != j) {
        System.out.print(array[i] + " ");
        mostrarRec((i + 1) % array.length, j);
    }
}
```

- Implemente o método boolean pesquisar(int elemento)

```
boolean pesquisar(int elemento) {
    int i = primeiro;
    boolean achei = false;
    while (i != ultimo) {
        if (array[i] == elemento) {
            achei = true;
        }
        i = (i + 1) % array.length;
    }
    return achei;
}
```

- Implemente o método int retornaPos(int posicao)

```
int retornaPos(int posicao) throws Exception{
    int resp = 0;
    if (posicao >= 0 && posicao <= array.length) {
        if (primeiro < ultimo) {
            if (posicao >= primeiro && posicao < ultimo) {
                resp = array[posicao];
            }
            else {
                throw new Exception("Area nao criada!");
            }
        } else {
            if (posicao >= primeiro || posicao < ultimo ) {
                resp = array[posicao];
            }
            else {
                throw new Exception("Area nao criada!");
            }
        }
    } else {
        throw new Exception("Fora do range!");
    }
    return resp;
}
```

• Implemente a fila circular sem o “+1” do construtor e garantindo a quantidade de elementos solicitada

O que fiz foi criar uma variável tamanho que gera a regra de tamanho do array para não haver sobreposição de itens. Resolvendo também o problema do array começar da posição 1.

```
class Fila {
    int[] array;
    int primeiro, ultimo, tamanho;

    Fila() {
        this(5);
    }

    Fila(int tamanho) {
        array = new int[tamanho];
        primeiro = ultimo = tamanho = 0;
    }

    void inserir(int x) throws Exception {
        if (tamanho + 1 >array.length) {
            throw new Exception("Erro!");
        }
        array[ultimo] = x;
        ultimo = (ultimo + 1) % (array.length);
        tamanho++;
    }

    int remover() throws Exception {
        if (tamanho -1 == 0)
            throw new Exception("Erro!");
        int resp = array[primeiro];
        primeiro = (primeiro + 1) % array.length;
        tamanho--;
        return resp;
    }

    void mostrar() {
        System.out.print("[");
        for (int i = 0, pos = primeiro; i < tamanho; i++) {
            System.out.print(array[pos] + " ");
            pos = (pos + 1) % array.length;
        }
        System.out.println("]");
    }
}
```