FICR
Faculdade Católica Imaculada Conceição

Estruturas de Dados

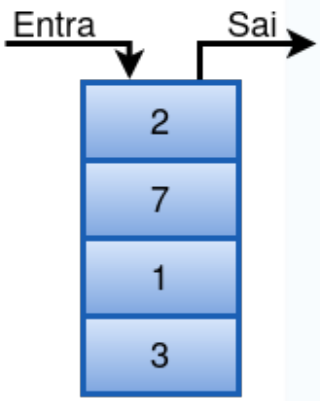
Mayrton Dias
mayrton.queiroz@p.ficr.edu.br

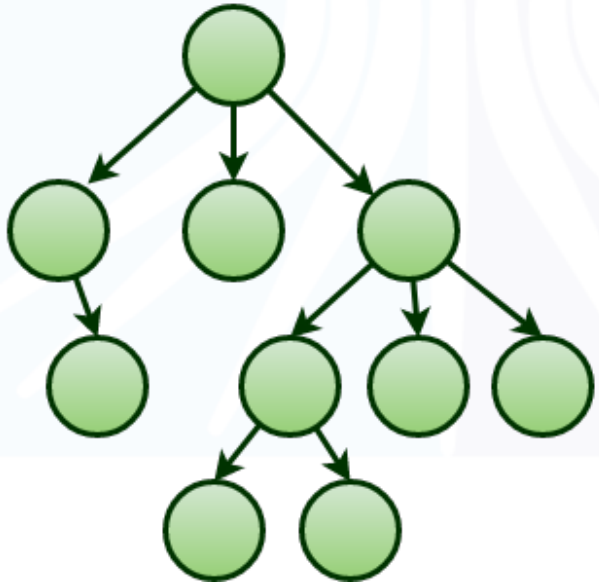
1. Apresentação
2. Abstração e Tipos Abstratos de Dados
3. Tipos Abstratos de Dados
4. TADs vs ED
5. Lista com vetores
6. Lista Duplamente Encadeada
7. Lista Simplesmente Encadeada e Circular

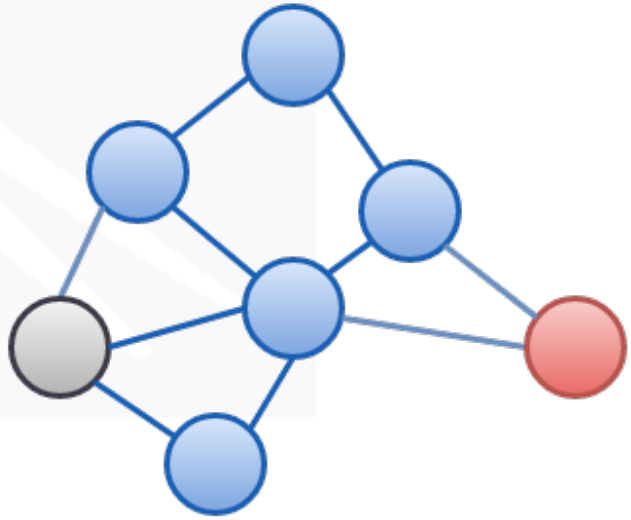
Apresentação

Apresentação

Estrutura de Dados: É o ramo da Computação que estuda os diversos mecanismos de organização de dados para atender aos diferentes requisitos de processamento.



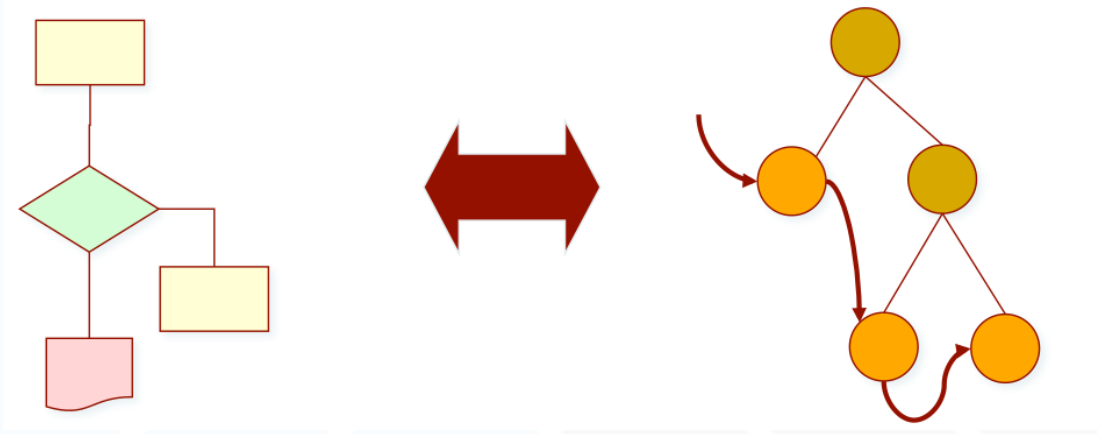




Apresentação

O objetivo da nossa disciplina será analisar as melhores alternativas para **manipular eficientemente os dados de um sistema computacional**.

Algoritmos geralmente trabalham sobre **Estrutura de Dados**



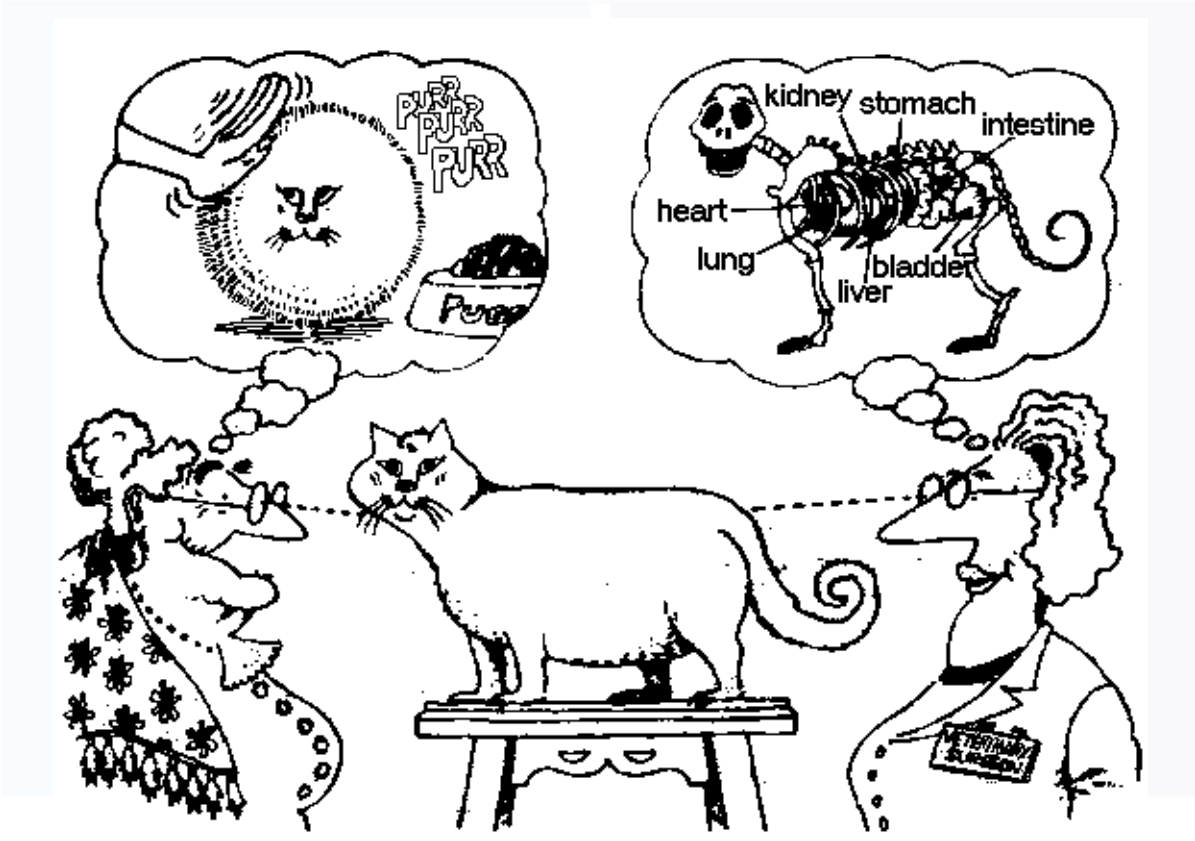
A escolha de uma **ED adequada** pode simplificar a implementação do algoritmo para um dado problema.

FICR
Faculdade Católica Imaculada Conceição

Abstração e Tipos Abstratos de Dados

Abstração

- Abstração:



Copyritght 1991 © Grady Booch

Abstração

Abstração:

- Permite-nos **dominar a complexidade do mundo real**
- **Modela-se uma entidade** ou **conceito de forma simplificada... focando apenas no que é interessante para resolver o problema.**
- Definição de Abstração:

"Uma abstração é uma visualização ou uma representação de uma entidade que **inclui somente os atributos de importância** em um **contexto particular**." Fonte: Robert Sebesta

Tipos Abstratos de Dados

Tipos Abstratos de Dados

FICR
Faculdade Católica Imaculada Conceição

Tipo Abstrato de Dados (TAD) é um **modelo** de **estruturação dos dados** que especifica:

- O tipo dos dados armazenados
- As operações definidas sobre esses dados
- Os tipos de parâmetros dessas operações

$$\text{TAD} = \text{encapsula dados} + \text{operações}$$

TADs vs ED

TADs vs ED

TAD:

- **Conceito matemático básico** que define o tipo de dado
- Define **O QUE** cada operação faz, **não como faz**
- Não se relaciona com **detalhes de implementação**

  Ex: Eficiência de espaço (uso memória) e tempo

Estrutura de Dados

- Método particular para **implementar um TAD** em uma Linguagem de Programação

Operações em TADs

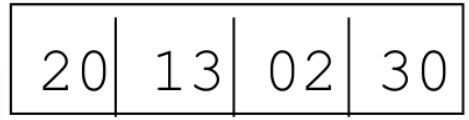
FICR
Faculdade Católica Imaculada Conceição

TAD:

- **Criação de um TAD**
- **Inclusões e remoções de dados no TAD**
- **Percurso no TAD** (varre todos os dados armazenados)
- **Busca** (busca algum dado dentro da estrutura).

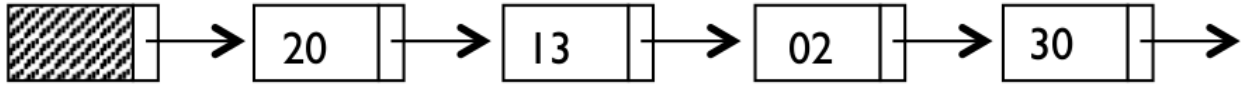
Operações em TADs

Implementação por Vetores:




```
void Insere(int x, Lista L) {
    for(i=0;...) {...}
    L[0] = x;
}
```

Implementação por Listas Encadeadas




```
void Insere(int x, Lista L) {
    p = CriaNovaCelula(x);
    L.primeiro = p;
    ...
}
```

Operações em TADs

Programa usuário do TAD

```
int main() {
    Lista L;
    int x;

    x = 20;

    Insere(x,L);
    ...
}
```

Lista com vetores

Inclusão das bibliotecas e definição da estrutura

```c
#include <stdio.h>
#include <stdlib.h>

#define TAM 5

typedef struct{
    int id;
    int *elementos;
} Lista;
```

Lista com vetores

Criar Lista

```
Lista* criarLista(){
    Lista *nova = (Lista*)malloc(sizeof(Lista));
    if(nova == NULL){
        printf("Nao tem espaco\n");
        return NULL;
    }
    nova->id = 0;
    nova->elementos = (int*)malloc(TAM*sizeof(int));
    if(nova->elementos == NULL){
        printf("Nao tem espaco\n");
        free(nova);
        return NULL;
    }
    return nova;
}
```

Lista com vetores

FICR
Faculdade Católica Imaculada Conceição

Insere Elemento (insere o elemento no final da lista)

```
int inserirElemento(Lista *lista, int valor){
    if(lista == NULL){
        printf("A lista nao foi criada\n");
        return 0;
    }
    if(lista->id < TAM){
        lista->elementos[lista->id] = valor;
        ++(lista->id);
    }else{
        printf("Espaco esgotado\n");
        return 0;
    }
    return 1;
}
```

Lista com vetores

Insere Elemento (insere o elemento no início da lista)

```
int inserirElementoInicio(Lista *lista, int valor){
    int i;
    if(lista == NULL){
        printf("A lista nao foi criada\n");
        return 0;
    }
    if(lista->id < TAM){
        for(i = lista->id; i > 0; --i){
            lista->elementos[i] = lista->elementos[i-1];
        }
        lista->elementos[0] = valor;
        ++(lista->id);
    }else{
        printf("Espaco esgotado\n");
        return 0;
    }
    return 1;
}
```

Lista com vetores

Insere Elemento no índice (posição) indicado

```
int inserirElementoID(Lista *lista, int valor, int posicao){
    int i;
    if(lista == NULL){
        printf("A lista nao foi criada\n");
        return 0;
    }
    if(lista->id < TAM){
        if(posicao < lista->id){
            for(i = lista->id; i > posicao; --i){
                lista->elementos[i] = lista->elementos[i-1];
            }
            lista->elementos[posicao] = valor;
            ++(lista->id);
        }else{
            printf("Posicao fora o intervalo permitido\n");
        }
    }else{
        printf("Espaco esgotado\n");
        return 0;
    }
    return 1;
}
```

Lista com vetores

Mostrar os Elementos da Lista

```
void imprimirElementos(Lista *lista){
    int i;
    if(lista == NULL){
        printf("A lista nao foi criada\n");
        return ;
    }
    if(lista->id == 0){
        printf("Lista vazia\n");
        return ;
    }
    for(i = 0; i < lista->id; ++i){
        printf("%d ", lista->elementos[i]);
    }
    printf("\n");
}
```

Remove Elemento

```
int removerElemento(Lista *lista, int valor){
    int i, j;
    for(i = 0; i < lista->id; ++i){
        if(lista->elementos[i] == valor){
            for(j = i; j < lista->id-1; ++j){
                lista->elementos[j] = lista->elementos[j+1];
            }
            --(lista->id);
            return 1;
        }
    }
    return 0;
}
```

Busca Elemento

```
int buscarElemento(Lista *lista, int valor){
    int i, j;
    for(i = 0; i < lista->id; ++i){
        if(lista->elementos[i] == valor){
            return i;
        }
    }
    return -1;
}
```

Atualiza Elemento

```
int atualizarElemento(Lista *lista, int busca, int valor){
    int i;
    for(i = 0; i < lista->id; ++i){
        if(lista->elementos[i] == busca){
            lista->elementos[i] = valor;
            return 1;
        }
    }
    return 0;
}
```

Exclui a Lista

```
Lista* excluirLista(Lista *lista){
    free(lista->elementos);
    free(lista);
    return NULL;
}
```

Lista com vetores

Função `main`, com as chamadas das funções apresentadas

```c
int main(){
    Lista *lista;
    lista = criarLista();
    inserirElemento(lista, 5);
    inserirElementoInicio(lista, 8);
    inserirElementoID(lista, 4, 1);
    removerElemento(lista, 4);
    imprimirElementos(lista);
    int busca = buscarElemento(lista, 2);
    if(busca != -1){
        printf("Elemento: %d\n",lista->elementos[busca]);
    }
    atualizarElemento(lista, 8, 6);
    lista = excluirLista(lista);
    return 0;
}
```

Lista Duplamente Encadeada

Lista Duplamente Encadeada

Inclusão das bibliotecas e definição da estrutura

```
typedef struct lista Lista;
typedef struct listaNo ListaNo;

struct lista{
    ListaNo *prim;
    ListaNo *ult;
};

struct listaNo{
    int valor;
    ListaNo *prox;
    ListaNo *ant;
};
```

Criar Lista

```
Lista* criarLista(){
    /*solicitando espaco para a lista*/
    Lista *nova = (Lista*)malloc(sizeof(Lista));

    /*Verificando se o espaco foi resevado*/
    if(nova == NULL){
        printf("Sem espaco\n");
        return NULL;
    }

    /*Preparando os dados iniciais da lista*/
    nova->prim = NULL;
    nova->ult = NULL;

    /*Retonando o espaco resevado*/
    return nova;
}
```

Lista Duplamente Encadeada

Insere Elemento (insere o elemento no final da lista)

```
int inserirElemento(Lista *lista, int valor){
    ListaNo *p;
    ListaNo *nova = (ListaNo*) malloc(sizeof(ListaNo));

    if(nova == NULL){
        printf("Sem espaco\n");
        return 0;
    }
    nova->valor = valor;
    nova->prox = NULL;
    nova->ant = NULL;

    if(lista->prim == NULL){
        lista->prim = nova;
        lista->ult = nova;
        return 1;
    }
    lista->ult->prox = nova;
    nova->ant = lista->ult;
    lista->ult = nova;

    return 1;
}
```

Lista Duplamente Encadeada

Faculdade Católica Imaculada Conceição

Mostrar os Elementos da Lista

```
void imprimirElementos(Lista *lista){
    ListaNo *p;
    if(lista == NULL){
        printf("A lista nao foi criada\n");
        return ;
    }
    if(lista->prim == NULL){
        printf("A lista esta vazia\n");
        return ;
    }

    for(p = lista->prim; p != NULL; p = p->prox){
        printf("%d ", p->valor);
    }
    printf("\n");
}
```

Lista Duplamente Encadeada

Faculdade Católica Imaculada Conceição

Mostrar os Elementos da Lista na ordem Inversa

```
void imprimirElementosOrdemInversa(Lista *lista){
    ListaNo *p;
    if(lista == NULL){
        printf("A lista nao foi criada\n");
        return ;
    }
    if(lista->prim == NULL){
        printf("A lista esta vazia\n");
        return ;
    }

    for(p = lista->ult; p != NULL; p = p->ant){
        printf("%d ", p->valor);
    }
    printf("\n");
}
```

Lista Duplamente Encadeada

FICR
Faculdade Católica Imaculada Conceição

Remove Elemento

```
int removerElemento(Lista *lista, int valor){
    ListaNo *p, *aux;
    if(lista == NULL){
        printf("A lista nao foi criada\n");
        return 0;
    }
    if(lista->prim == NULL){
        printf("A lista esta vazia\n");
        return 0;
    }
    if( lista->prim->valor == valor){
        aux = lista->prim;
        lista->prim->prox->ant = NULL;
        lista->prim = lista->prim->prox;
        free(aux);
        return 1;
    }
    //continua ...
```

Lista Duplamente Encadeada

Faculdade Católica Imaculada Conceição

Remove Elemento

```
    //...continuacao
    for(p = lista->prim; p->prox->prox != NULL; p = p->prox){
        /*Verificando se e o elemento que sera removido da lista*/
        if(p->prox->valor == valor){
            aux = p->prox;
            p->prox = p->prox->prox;
            p->prox->ant = p;
            free(aux);
            return 1;
        }
    }
    if(p->prox->valor == valor){
        aux = p->prox;
        p->prox = NULL;
        lista->ult = p;
        free(aux);
        return 1;
    }
    return 0;
}
```

Lista Duplamente Encadeada

FICR
Faculdade Católica Imaculada Conceição

Busca Elemento

```
ListaNo* buscarElemento(Lista *lista, int valor){
    ListaNo *p;
    if(lista == NULL){
        printf("A lista nao foi criada\n");
        return NULL;
    }

    if(lista->prim == NULL){
        printf("A lista esta vazia\n");
        return NULL;
    }

    for(p = lista->prim; p != NULL; p = p->prox){
        if(p->valor == valor){
            return p;
        }
    }
    return NULL;
}
```

Lista Duplamente Encadeada

Faculdade Católica Imaculada Conceição

Atualiza Elemento

```
int atualizarElemento(Lista *lista, int busca, int valor){
    ListaNo *p;
    if(lista == NULL){
        printf("A lista nao foi criada\n");
        return 0;
    }
    if(lista->prim == NULL){
        printf("A lista esta vazia\n");
        return 0;
    }

    for(p = lista->prim; p != NULL; p = p->prox){
        if(p->valor == busca){
            p->valor = valor;
            return 1;
        }
    }
    return 0;
}
```

Exclui a Lista

```
Lista* excluirLista(Lista* lista){
    ListaNo *aux;
    /*Liberando o espaco que foi reservado para os elementos*/
    while(lista->prim != NULL){
        aux = lista->prim;
        lista->prim = lista->prim->prox;
        free(aux);
    }
    /*Liberando o espaco que foi reservado para a lista*/
    free(lista);
    return NULL;
}
```

FICR
Faculdade Católica Imaculada Conceição

Lista Simplesmente Encadeada e Circular

Inclusão das bibliotecas e definição da estrutura

```
typedef struct lista Lista;
typedef struct listaNo ListaNo;

struct lista{
    ListaNo *prim;
};

struct listaNo{
    int valor;
    ListaNo *prox;
};
```

Criar Lista

```
Lista* criarLista(){
    Lista *nova = (Lista*)malloc(sizeof(Lista));

    if(nova == NULL){
        printf("Sem espaco\n");
        return NULL;
    }

    nova->prim = NULL;

    return nova;
}
```

Lista Simplesmente Encadeada e Circular

Faculdade Católica Imaculada Conceição

Insere Elemento (insere o elemento no final da lista)

```
int inserirElemento(Lista *lista, int valor){
    ListaNo *p;
    ListaNo *nova = (ListaNo*) malloc(sizeof(ListaNo));

    if(nova == NULL){
        printf("Sem espaco\n");
        return 0;
    }
    nova->valor = valor;
    nova->prox = NULL;
    if(lista->prim == NULL){
        nova->prox = nova;
        lista->prim = nova;
        return 1;
    }
    for(p = lista->prim; p->prox != lista->prim; p = p->prox);
    p->prox = nova;
    nova->prox = lista->prim;
    return 1;
}
```

Lista Simplesmente Encadeada e Circular

FICR
Faculdade Católica Imaculada Conceição

Mostrar os Elementos da Lista

```
void imprimirElementos(Lista *lista){
    ListaNo *p;

    if(lista == NULL){
        printf("A lista nao foi criada\n");
        return ;
    }
    if(lista->prim == NULL){
        printf("A lista esta vazia\n");
        return ;
    }
    p = lista->prim;
    do{
        printf("%d ", p->valor);
        p = p->prox;
    }while(p != lista->prim);

    printf("\n");
}
```

Lista Simplesmente Encadeada e Circular

Remove Elemento

```
int removerElemento(Lista *lista, int valor){
    ListaNo *p, *aux;

    if(lista == NULL){
        printf("A lista nao foi criada\n");
        return 0;
    }
    if(lista->prim == NULL){
        printf("A lista esta vazia\n");
        return 0;
    }

    if( lista->prim->valor == valor){
        aux = lista->prim;
        for(p = lista->prim; p->prox != lista->prim; p = p->prox);
        lista->prim = lista->prim->prox;
        p->prox = lista->prim;
        free(aux);
        return 1;
    }
    //continua ...
```

Remove Elemento

```
    //continuacao ...
    for(p = lista->prim; p->prox->prox != lista->prim; p = p->prox){
        if(p->prox->valor == valor){
            aux = p->prox;
            p->prox = p->prox->prox;
            free(aux);
            return 1;
        }
    }

    if(p->prox->valor == valor){
        aux = p->prox;
        p->prox = lista->prim;
        free(aux);
        return 1;
    }

    return 0;
}
```

Lista Simplesmente Encadeada e Circular

Busca Elemento

```
ListaNo* buscarElemento(Lista *lista, int valor){
    ListaNo *p;

    if(lista == NULL){
        printf("A lista nao foi criada\n");
        return NULL;
    }
    if(lista->prim == NULL){
        printf("A lista esta vazia\n");
        return NULL;
    }

    p = lista->prim;
    do{
        if(p->valor == valor){
            return p;
        }
        p = p->prox;
    }while(p != lista->prim);

    return NULL;
}
```

Lista Simplesmente Encadeada e Circular

Atualiza Elemento

```c
int atualizarElemento(Lista *lista, int busca, int valor){
    ListaNo *p;

    if(lista == NULL){
        printf("A lista nao foi criada\n");
        return 0;
    }
    if(lista->prim == NULL){
        printf("A lista esta vazia\n");
        return 0;
    }
    p = lista->prim;
    do{
        if(p->valor == busca){
            p->valor = valor;
            return 1;
        }
        p = p->prox;
    }while(p != lista->prim);

    return 0;
}
```

Exclui a Lista

```
Lista* excluirLista(Lista* lista){
    ListaNo *aux;

    while(lista->prim != lista->prim->prox){
        aux = lista->prim->prox;
        lista->prim->prox = lista->prim->prox->prox;
        free(aux);
    }
    free(lista->prim);
    free(lista);
    return NULL;
}
```

FICR
Faculdade Católica Imaculada Conceição

Estruturas de Dados

Mayrton Dias
mayrton.queiroz@p.ficr.edu.br