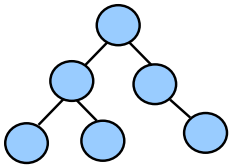
INF01203 – Estruturas de Dados

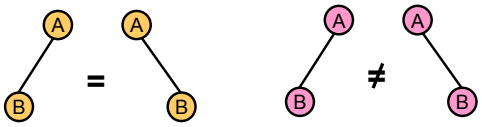
Árvores Binárias

Árvores Binárias

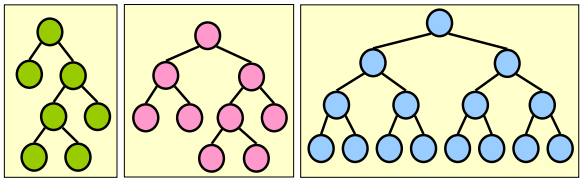
- grau dos nós
  - 0 – 1 – 2



- ordenadas
  - sub-árvore da esquerda ≠ sub-árvore da direita



Tipos de Árvores Binárias

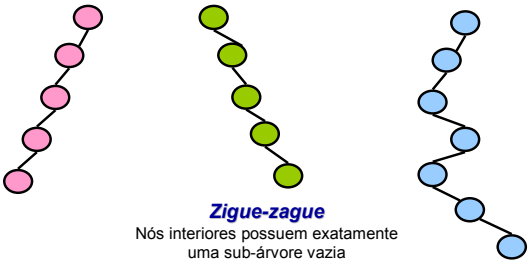


Estritamente Binária
0 ou 2 filhos

Binária Completa
Sub-árvores vazias no último ou penúltimo nível

Binária Cheia
Sub-árvores vazias somente no último nível

Tipos de Árvores Binárias



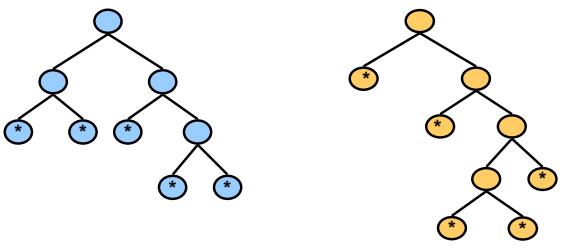
Zigue-zague
Nós interiores possuem exatamente uma sub-árvore vazia

Tipos de Árvores Binárias

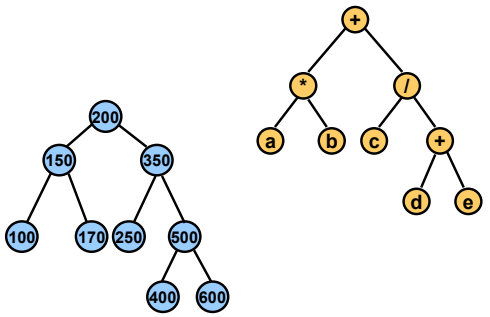
- Dados os tipos de árvores:
  - estritamente binária
  - completa
  - cheia
  - zigue-zague
- Qual árvore possui altura *máxima* ?
  - zigue-zague
- Qual árvore possui altura *mínima* ?
  - cheia

Árvore Binária

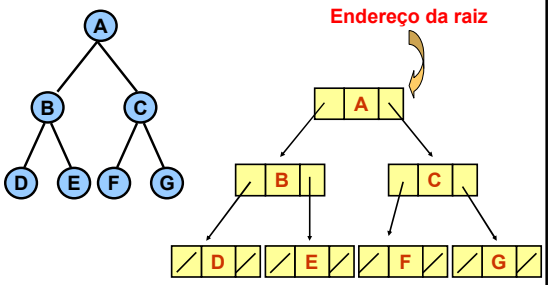
- Comprimento de caminho de uma árvore
  - tempo de execução



Exemplos de Árvores Binárias



Implementação

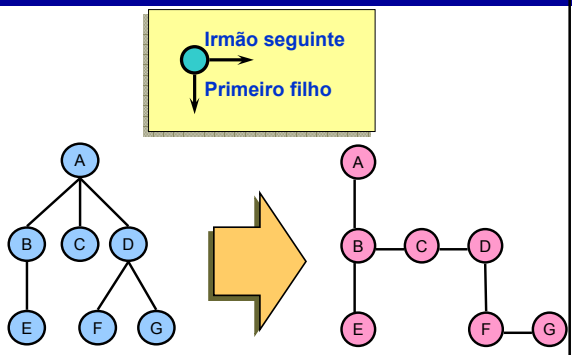


Árvores Binárias - vantagens

- Otimização de alocação de memória
- Mais fáceis de manipular
- Implementação de operações é muito simplificada

Transformar árvore **n-ária** em **binária**

Transformação: n-ária em binária

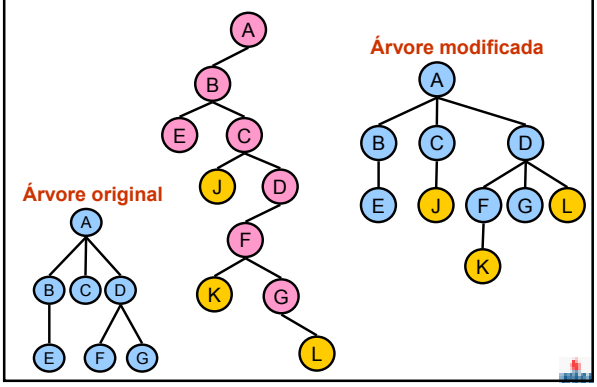


Transformação: n-ária em binária

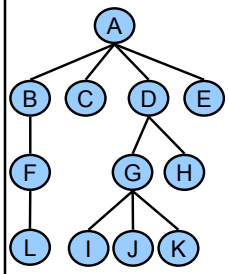
1. O primeiro filho de um nodo passa a ser seu filho à esquerda na árvore binária
2. Os demais filhos de um nodo passam a ser filhos à direita do seu irmão imediato à esquerda
3. Executar o mesmo processo para cada nodo da árvore

Filho à esquerda = primeiro filho
Filho à direita = irmão seguinte (está direita)

Reconstituição Árvore n-ária



Exercícios

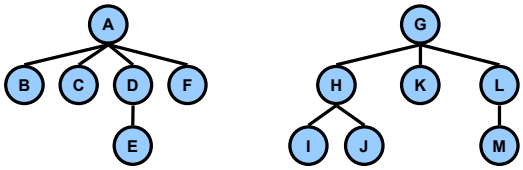


?

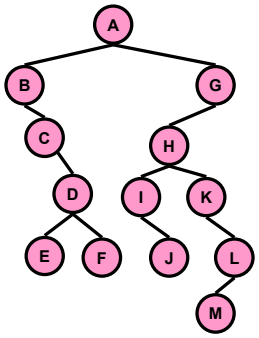
- Converter Binária
- inserir Y filho direita de A
- inserir X filho esquerda Y
- inserir Z filho direita Y
- converter para árvore grau n

Transformação de Floresta em Binária

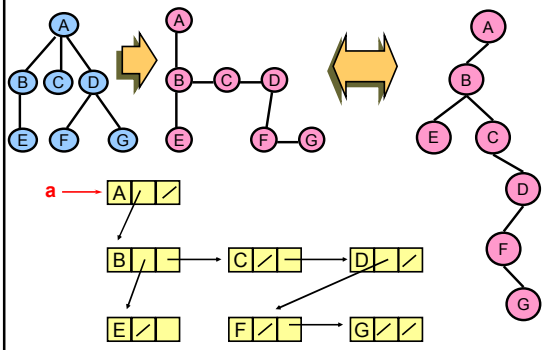
- Para converter uma floresta em árvore binária, basta considerar as raízes das árvores como nós irmãos e aplicar a conversão anterior



Transformação de Floresta em Binária

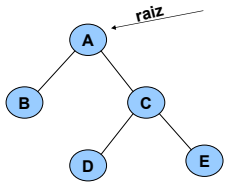


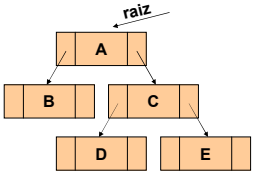
Transformação: n-ária em binária



Implementação de Árvores Binárias

TAD : Estrutura de Dados





Tipo

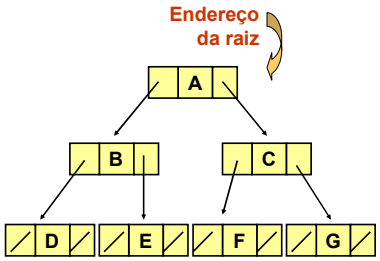
```
struct TNodoA{
   char info;
   struct TNodoA *esq;
   struct TNodoA *dir;
};

typedef struct TNodoA pNodoA;
```

Nodo

TAD : Operações

- Inicializa
- Insere
  - raiz
  - meio
  - folha
- Consulta
- Remove
- Destrói



TAD: Operações sobre Árvores Binárias

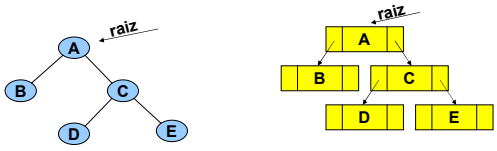
```
pNodoA* cria_arvore(void)
// cria uma árvore vazia

pNodoA* InsereRaiz(pNodoA *a, char ch)
// aloca nodo raiz e insere dado

pNodoA* InsereEsq(pNodoA * a, char info, char infopai)
// insere um nodo na sub-árvore esquerda

pNodoA* InsereDir(pNodoA * a, char info, char infopai)
// insere um nodo na sub-árvore direita
```

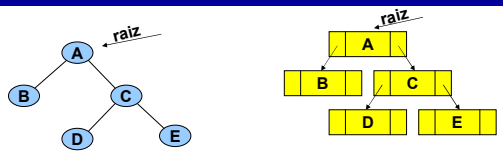
Inicializa



Criar uma árvore vazia

```
pNodoA* cria_arvore(void)
{
      return NULL;
}
```

Insere raiz



Alocar nodo raiz

```
pNodoA* InsereRaiz(pNodoA *a, char ch)
{
  if (a == NULL)
  {
    a = (pNodoA*) malloc(sizeof(pNodoA));
    a->info = ch;
    a->esq = NULL;
    a->dir = NULL;
    return(a);
  }
}
```

Insere filho à esquerda

```
pNodoA* InsereEsq(pNodoA * a, char info, char infopai)
{
    pNodoA *pai, *filho;
    pai = achaPai(a,infopai);  //função auxiliar para achar o pai de um nodo
    if (pai != NULL){
        if (pai->esq == NULL){
          filho = (pNodoA*) malloc (sizeof(pNodoA));
          filho->info = info;
          filho->esq=NULL;
          filho->dir=NULL;
          pai->esq=filho;
        }
        else
          printf("o nodo ja tem sub-arvore esquerda\n");
    }
    return a;
}
```

Função para achar o pai de um nodo

```
pNodoA* achaPai(pNodoA *a, char info)
{
   pNodoA *found = NULL;
    if (a != NULL){
        if (a->info == info)
          return a;
        else{
           found = achaPai(a->esq,info);
           if (!found)
             return achaPai(a->dir,info);
           else return found;
        }
    }
    else
      return NULL;
}
```

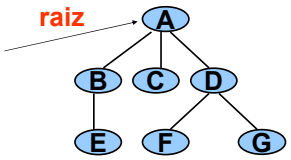
Destrói

```
pNodoA* destroi(pNodoA *a)
{
    if (a!= NULL)
    {
        destroi(a->esq);
        destroi(a->dir);
        printf("destruindo %c\n",a->info);
        free(a);
        return NULL;
    }
}
```

Caminhamentos em Árvores Binárias

Consulta Nodos

- acesso sempre através da raiz
- cada nodo deve ser "visitado" uma vez, e apenas uma vez

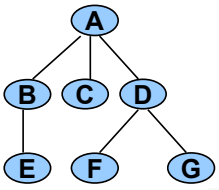


Visita **a um nodo**

acesso a um nodo
para realizar alguma operação

Caminhamentos

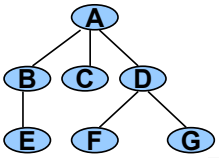
- método de percurso sistemático de todos os nodos de uma árvore, de modo a que cada nodo seja visitado exatamente uma vez



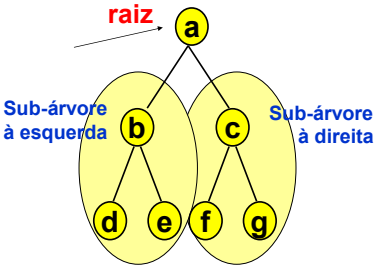
Caminhamentos

- um caminhamento define uma seqüência de nodos
- cada nodo passa a ter um nodo seguinte, ou um nodo anterior, ou ambos (exceto árvore com 1 só nodo)
- seqüência de nodos depende do caminhamento

Ex: Caminhamento 1:
A - B - C - D - E - F - G
Caminhamento 2:
A - B - E - C - D - F - G



Principais Caminhamentos



Caminhamentos

Pré-Fixado à esquerda
.Visita a raiz
.Percorre a sub-árvore esquerda
.Percorre a sub-árvore direita

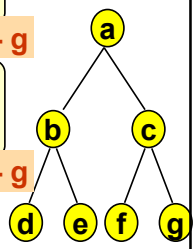
a - b - d - e - c - f - g

Central à esquerda
.Percorre a sub-árvore esquerda
.Visita a raiz
.Percorre a sub-árvore direita

d - b - e - a - f - c - g

Pós-Fixado à esquerda
.Percorre a sub-árvore esquerda
.Percorre a sub-árvore direita
.Visita a raiz

d - e - b - f - g - c - a



Caminhamentos

Pré-Fixado à direita
.Visita a raiz
.Percorre a sub-árvore direita
.Percorre a sub-árvore esquerda

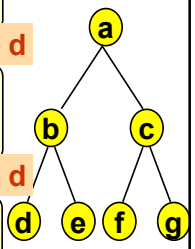
a - c - g - f - b - e - d

Central à direita
.Percorre a sub-árvore direita
.Visita a raiz
.Percorre a sub-árvore esquerda

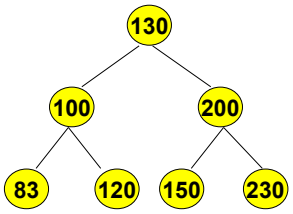
g - c - f - a - e - b - d

Pós-Fixado à direita
.Percorre a sub-árvore direita
.Percorre a sub-árvore esquerda
.Visita a raiz

g - f - c - e - d - b - a



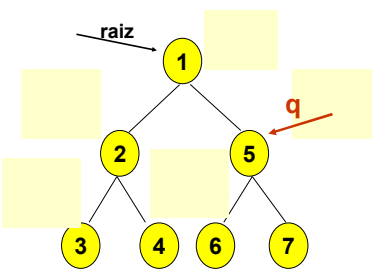
Exemplo 02



.Central à esquerda?

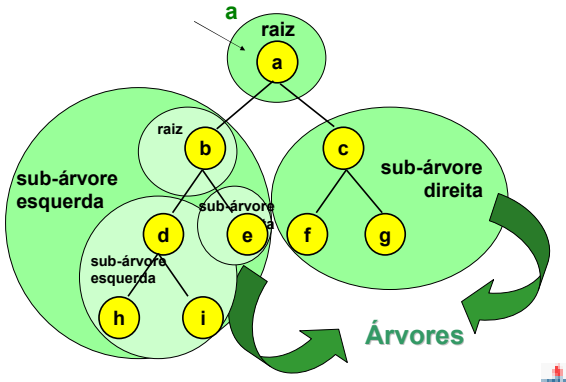
.Central à direita?

Percorrer: Pré-Fixado Esquerda



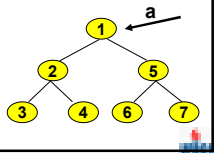
1 - 2 - 3 - 4 - 5 ...

Recursividade em Árvores



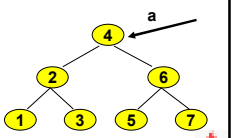
Pré-Fixado Esquerda

```
void preFixado(pNodoA *a)
{
   if (a!= NULL)
   {
      printf("%c\n",a->info);
      preFixado(a->esq);
      preFixado(a->dir);
   }
}
```



Central Esquerda

```
void central(pNodoA *a)
{
    if (a!= NULL)
    {
        central(a->esq);
        printf("%c\n",a->info);
        central(a->dir);
     }
}
```



Pós-Fixado a Esquerda

```
void posFixado(pNodoA *a)
{
    if (a!= NULL)
    {
        posFixado(a->esq);
        posFixado(a->dir);
        printf("%c\n",a->info);
     }
}
```