CES-11

Algoritmos e Estruturas de Dados

Carlos Alberto Alonso Sanches
Juliana de Melo Bezerra

CES-11

- Árvores
  - Operações sobre uma árvore
  - Estruturas para armazenar árvores
    - Contígua
    - Contígua melhorada
    - Encadeada direta
    - Listas de filhos
    - Encadeamento de pais e irmãos

CES-11

- Árvores
  - Operações sobre uma árvore
  - Estruturas para armazenar árvores
    - Contígua
    - Contígua melhorada
    - Encadeada direta
    - Listas de filhos
    - Encadeamento de pais e irmãos

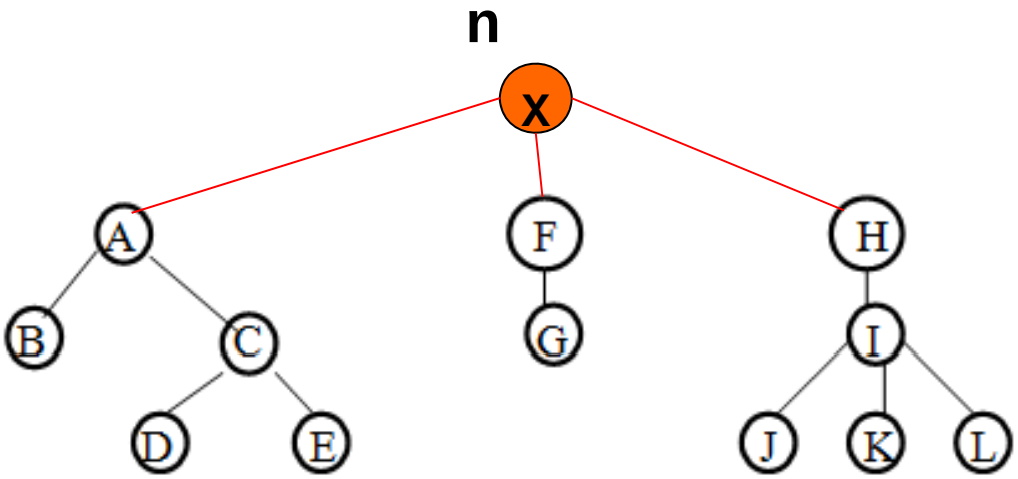
Operações sobre uma árvore

node é o tipo correspondente a um vértice da árvore

- **node Pai (n,A)**
  Retorna o pai do nó **n** da árvore **A**
  - Se **n** é raiz, **Pai** é **NULL**
- **node FilhoEsquerdo (n,A)**
  Retorna o filho esquerdo do nó **n** da árvore **A**
  - Se **n** é folha, **FilhoEsquerdo** é **NULL**
- **node IrmaoDireito (n,A)**
  Retorna o irmão direito do nó **n** da árvore **A**
  - Se **n** é caçula, **IrmaoDireito** é **NULL**
- **bool Cacula (n,A)**
  Verifica se o nó **n** é caçula na árvore **A**

Operações sobre uma árvore

- **`char Elemento (n,A)`**
  Retorna as informações do nó **`n`** da árvore **`A`**
  - Neste exemplo, os nós armazenam um caractere, mas poderiam ter qualquer outra informação
- **`arvore Criacao (x,ListaArvores)`**
  Coloca a informação **`x`** em um novo nó **`n`** e cria uma árvore com raiz **`n`** e sub-árvores de **`ListaArvores`** (uma lista de árvores)



Operações sobre uma árvore

- **`node Raiz (A)`**
  Retorna o nó raiz da árvore **A**
- **`void Esvaziar (A)`**
  Torna vazia a árvore **A**
- **`bool Vazia (A)`**
  Verifica se a árvore **A** é vazia

- As estruturas de dados utilizadas no armazenamento da árvore terão um claro impacto na eficiência das suas operações.

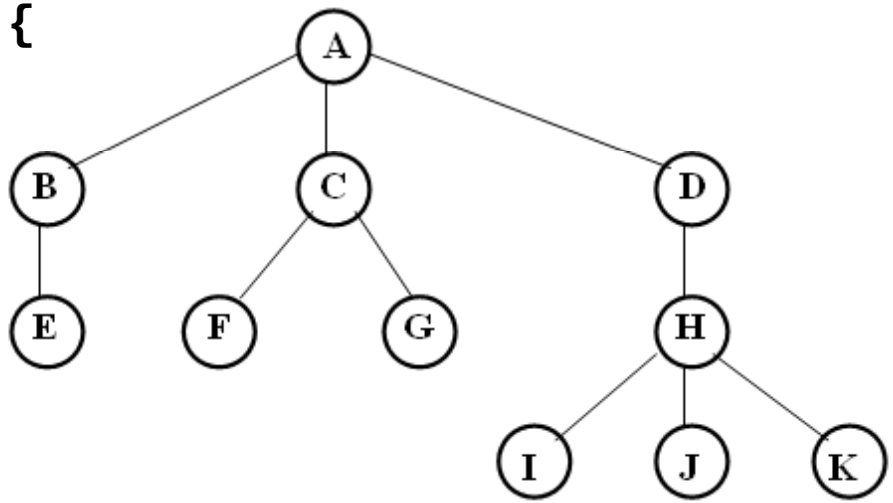
Operações sobre uma árvore

- Há ainda várias outras operações que poderiam ser definidas:
  - Alterar o conteúdo de um nó
  - Tornar um novo nó caçula de outro nó
  - Fazer com que uma árvore se torne sub-árvore de um nó de outra árvore
  - Eliminar uma sub-árvore de um nó
  - Eliminar um nó de uma árvore (mais complicado se não for folha)
  - Inserir um nó numa determinada posição de uma árvore
  - etc.

Operações sobre uma árvore

- Exemplo 1: ordenação pré-ordem

```
void PreOrdem (node n, arvore A) {
   node c;
   Escrever (Elemento (n, A));
   c = FilhoEsquerdo (n, A);
   while (c != NULL)  {
        PreOrdem (c, A);
        c = IrmaoDireito (c, A);
   }
}
```



PreOrdem (Raiz(A), A)

Operações sobre uma árvore

- Exemplo 2: cálculo da altura de uma árvore **A**
- Definição recursiva da altura de um nó **n**:
  - Se **n** é **NULL**, Altura (**n**) = -1;
  - Senão, se **n** é folha, Altura (**n**) = 0;
  - Senão, Altura (**n**) = 1 + max (alturas dos filhos de **n**)

```
Altura (Raiz(A), A)
```

Operações sobre uma árvore

- Exemplo 2: cálculo da altura de uma árvore A

```
int Altura (node n, arvore A) {
   int maxalt, aux; node f;
   if (n == NULL)
      return -1;
   f = FilhoEsquerdo (n, A);
   if (f == NULL)
      return 0;
   for (maxalt = 0, aux = 0; f != NULL; f = IrmaoDireito (f, A)){
     aux = Altura (f, A);
     if (aux > maxalt)
        maxalt = aux;
   }
    return maxalt + 1;
}
```

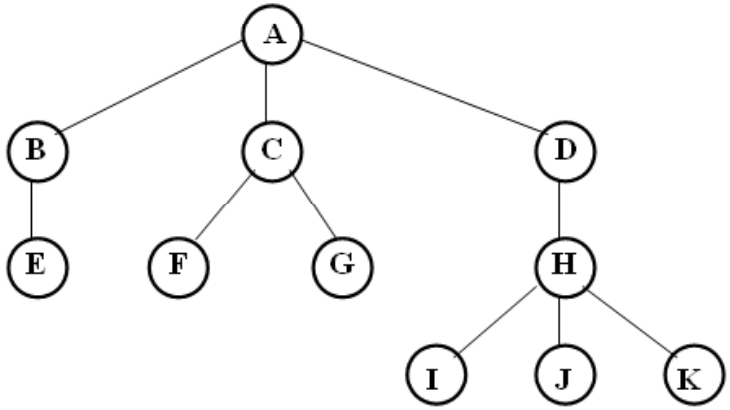
Importante: na definição genérica de árvore, todo filho único é considerado filho esquerdo

CES-11

- Árvores
  - Operações sobre uma árvore
  - Estruturas para armazenar árvores
    - Contígua
    - Contígua melhorada
    - Encadeada direta
    - Listas de filhos
    - Encadeamento de pais e irmãos

Contígua

- Os nós são armazenados em um vetor numa ordem convencionada (pré-ordem, por exemplo)
- Neste caso, **node** é um índice no vetor de células



Propositalmente vazio, a fim de começar com nó 1

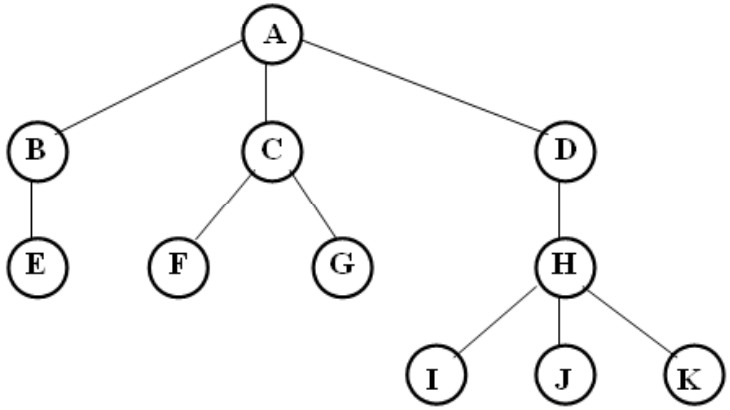
Elementos — **celula**

	#	A	B	E	C	F	G	D	H	I	J	K	
info	#	A	B	E	C	F	G	D	H	I	J	K	
grau	#	3	1	0	2	0	0	1	3	0	0	0	
node	0	1	2	3	4	5	6	7	8	9	10	11	

11 ncel

Contígua

```
const int max = 100;
typedef int node;
struct celula{
   char info;
    int grau;
};
struct arvore{
    celula Elementos[max+1];
    int ncel;
};
```



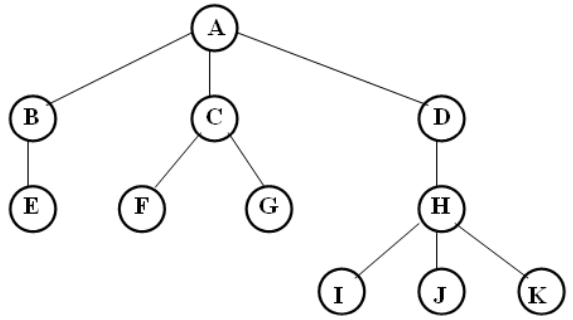
Elementos — celula

#	A	B	E	C	F	G	D	H	I	J	K	info
#	3	1	0	2	0	0	1	3	0	0	0	grau
0	1	2	3	4	5	6	7	8	9	10	11	node

11 ncel

Contígua

```
node Raiz (arvore A){
  if (A.ncel == 0) return NULL;
  return 1;
}
```




```
bool Vazia (arvore A){
  if (A.ncel == 0)
    return true;
  else
    return false;
}
```

```
void Esvaziar (arvore *A) {
  A->ncel = 0;
}
```

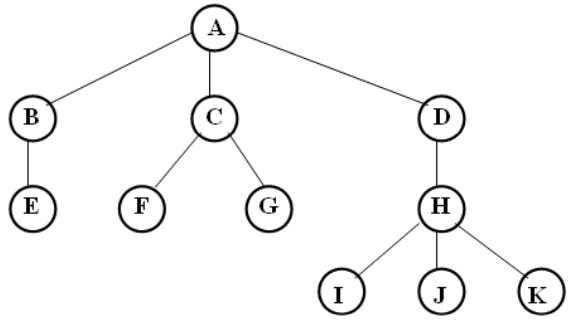
Elementos

#	A	B	E	C	F	G	D	H	I	J	K	info
#	3	1	0	2	0	0	1	3	0	0	0	grau
0	1	2	3	4	5	6	7	8	9	10	11	node

11 ncel

Contígua

```
char Elemento (node n, arvore A){
  if (n <= 0 || n > A.ncel) //ERRO
  else
    return A.Elementos[n].info;
}
node FilhoEsquerdo (node n, arvore A){
  if (n <= 0 || n > A.ncel) //ERRO
  else if (A.Elementos[n].grau == 0)
    return NULL;
  else return n + 1;
}
```



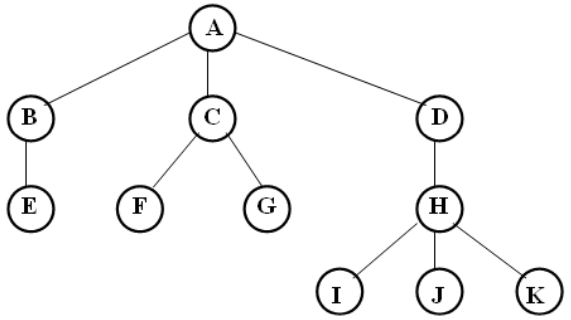
Elementos

#	A	B	E	C	F	G	D	H	I	J	K	info
#	3	1	0	2	0	0	1	3	0	0	0	grau
0	1	2	3	4	5	6	7	8	9	10	11	node

11 ncel

Contígua

- Como encontrar o pai de um nó?
- Exemplo: pai do nó 7 é o nó 1
- O pai de **n** é um nó que está à sua esquerda
- Entre **n** e seu pai, estão todos os irmãos de **n** mais à esquerda com seus respectivos descendentes
- O algoritmo não é imediato...



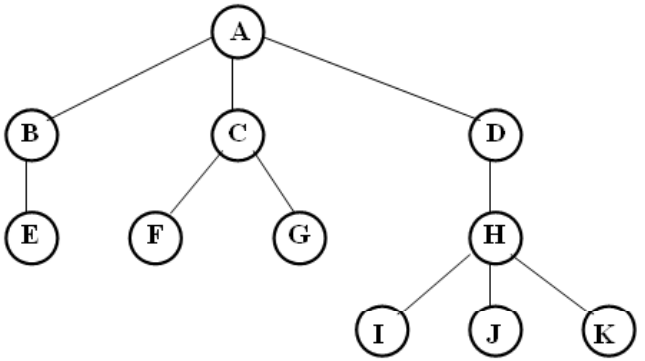
Elementos

#	A	B	E	C	F	G	D	H	I	J	K	info
#	3	1	0	2	0	0	1	3	0	0	0	grau
0	1	2	3	4	5	6	7	8	9	10	11	node

11 ncel

Contígua

- Como encontrar o irmão direito de um nó?
- Exemplo: o irmão direito do nó 4 é o nó 7
- Se **n** não for caçula, seu irmão direito é um nó que está à sua direita
- Entre **n** e seu irmão direito estão todos os seus descendentes próprios
- O algoritmo também não é imediato...



Elementos

#	A	B	E	C	F	G	D	H	I	J	K	info
#	3	1	0	2	0	0	1	3	0	0	0	grau
0	1	2	3	4	5	6	7	8	9	10	11	node

11 ncel

Contígua

- Pontos fortes
  - Gasta relativamente pouca memória
  - Serve para armazenamento permanente
- Ponto fraco
  - Possui operações ineficientes

Elementos

#	A	B	E	C	F	G	D	H	I	J	K	info
#	3	1	0	2	0	0	1	3	0	0	0	grau
0	1	2	3	4	5	6	7	8	9	10	11	node

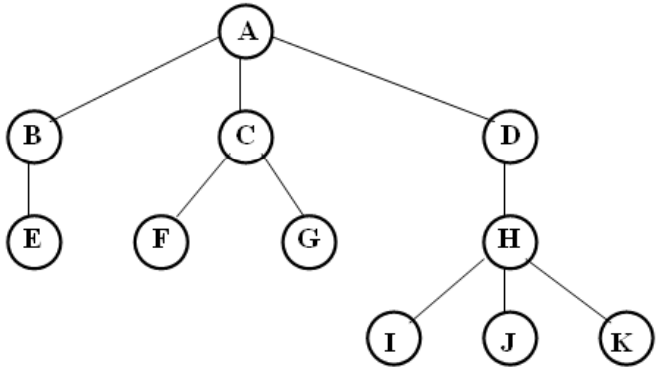
11 ncel

CES-11

- Árvores
  - Operações sobre uma árvore
  - Estruturas para armazenar árvores
    - Contígua
    - Contígua melhorada
    - Encadeada direta
    - Listas de filhos
    - Encadeamento de pais e irmãos

Contígua melhorada

- A estrutura contígua pode ser melhorada, incluindo-se nas células informações de caráter operacional
  - Concretamente: acesso ao pai de cada nó



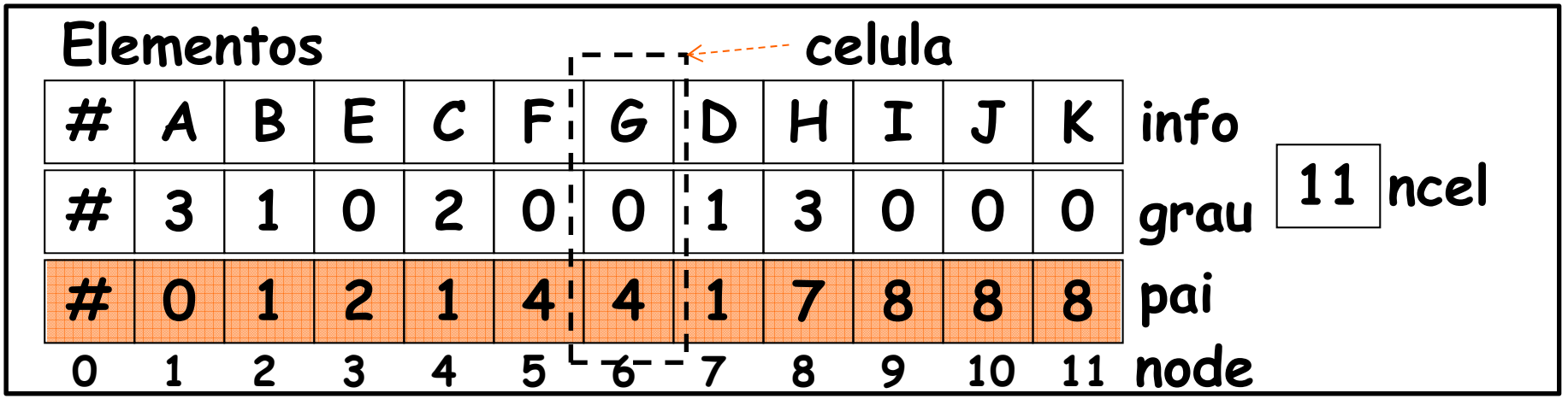
Elementos — celula

	0	1	2	3	4	5	6	7	8	9	10	11	
#	A	B	E	C	F	G	D	H	I	J	K		info
#	3	1	0	2	0	0	1	3	0	0	0		grau
#	0	1	2	1	4	4	1	7	8	8	8		pai

11 ncel

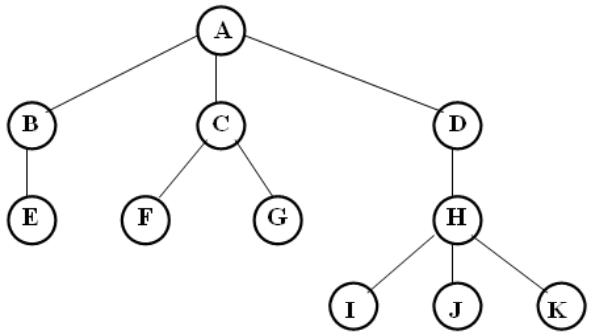
Contígua melhorada

```
const int max = 100;
typedef int node;
struct celula{
  char info;
  int grau, pai;
};
struct arvore{
  celula Elementos[max+1];
  int ncel;
};
```



Contígua melhorada

- Quem é o **pai** de um nó?
  - Exemplo: pai do nó 7 é o nó 1
  - Agora é elementar: a informação já consta na célula
- Quem é o **irmão direito** de um nó?
  - Exemplo: irmão direito do nó 4 é o nó 7
  - Basta encontrar o primeiro nó à sua direita que tenha o mesmo pai



Elementos — celula

#	A	B	E	C	F	G	D	H	I	J	K	info
#	3	1	0	2	0	0	1	3	0	0	0	grau
#	0	1	2	1	4	4	1	7	8	8	8	pai
0	1	2	3	4	5	6	7	8	9	10	11	node

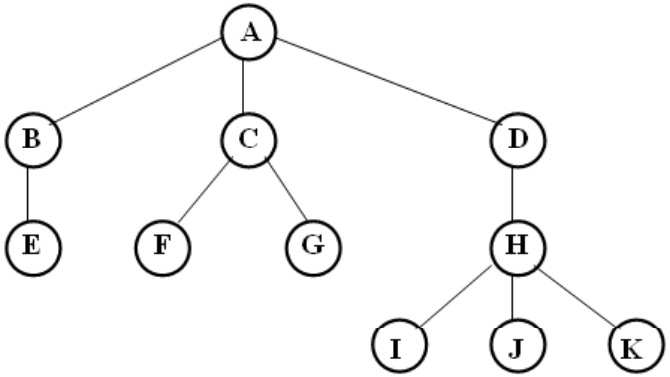
11 ncel

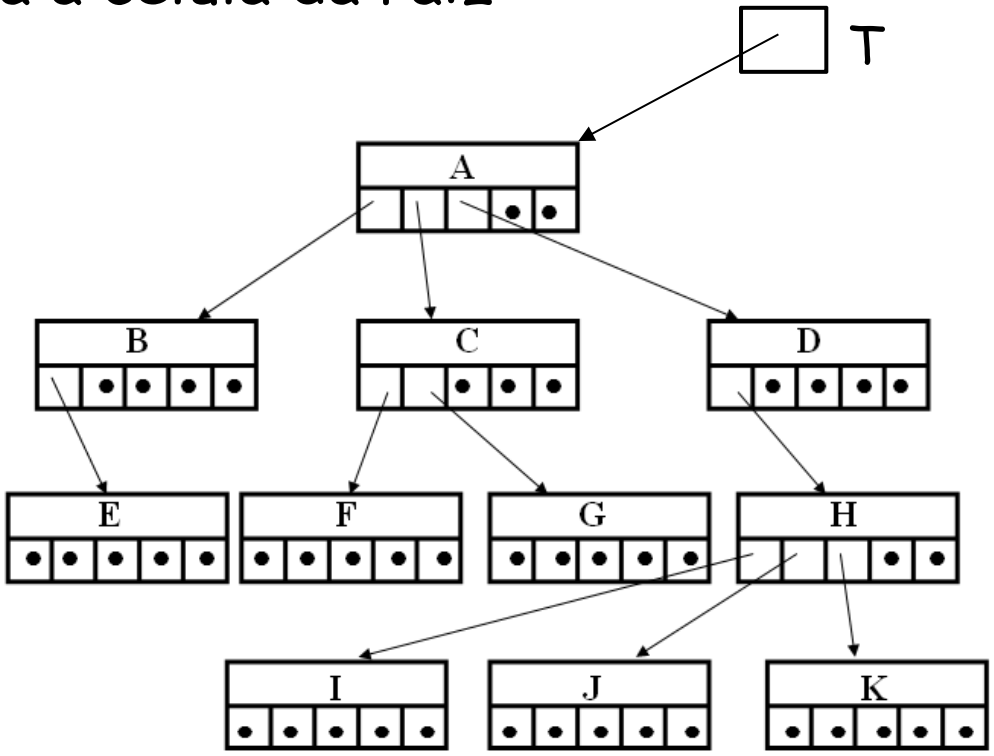
CES-11

- Árvores
  - Operações sobre uma árvore
  - Estruturas para armazenar árvores
    - Contígua
    - Contígua melhorada
    - Encadeada direta
    - Listas de filhos
    - Encadeamento de pais e irmãos

Encadeada direta

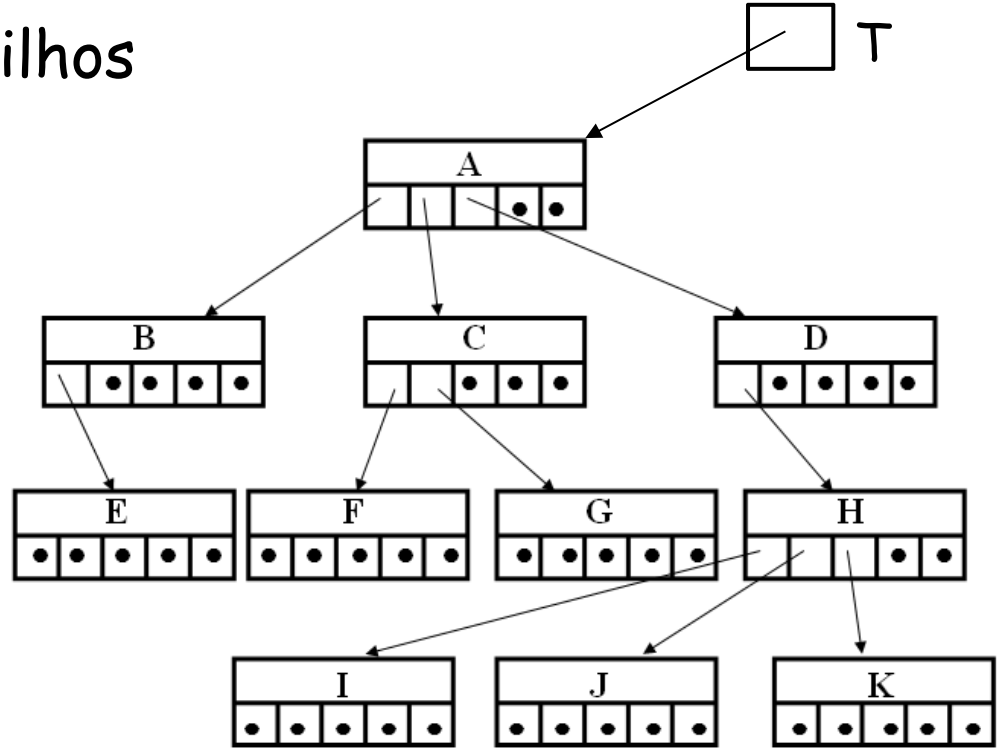
- Declarações:
  - **Nó**: ponteiro para célula
  - **Árvore**: ponteiro para a célula da raiz





Encadeada direta

- Pontos fracos:
  - Usa muitos ponteiros desnecessários (por exemplo, nas folhas)
  - Limita o número de filhos



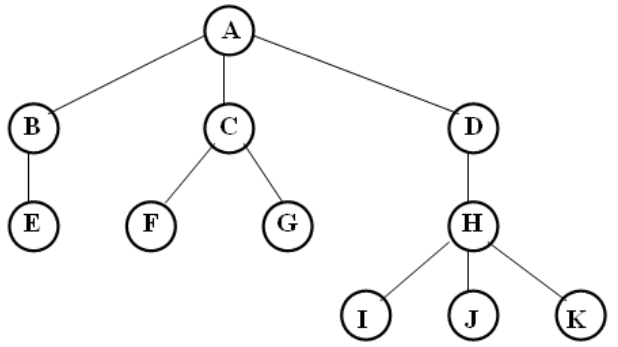

```
const int MaxFil = 5;
struct celula{
   char info;
   celula* filhos[MaxFil];
};
typedef celula *node;
typedef celula *arvore;

arvore T;
```

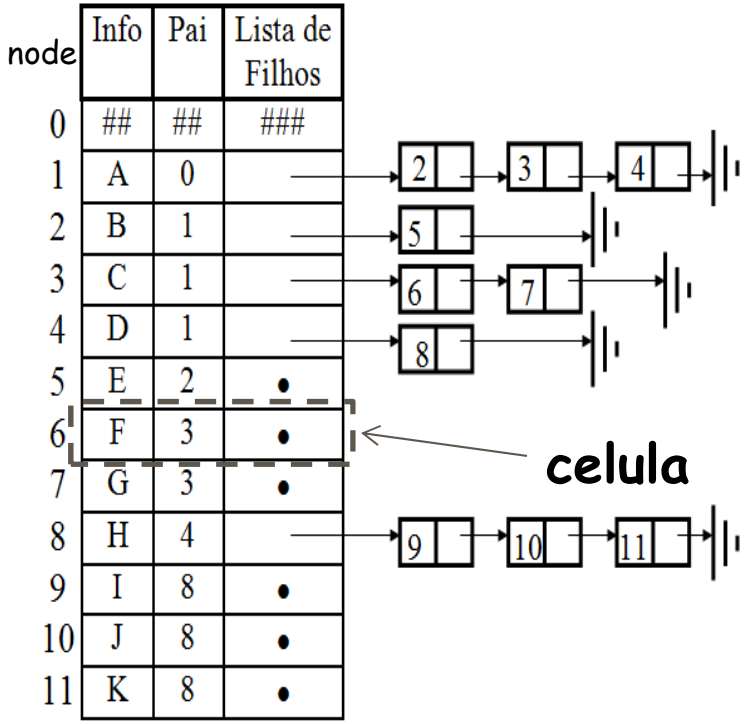
CES-11

- Árvores
  - Operações sobre uma árvore
  - Estruturas para armazenar árvores
    - Contígua
    - Contígua melhorada
    - Encadeada direta
    - Listas de filhos
    - Encadeamento de pais e irmãos

Lista de filhos

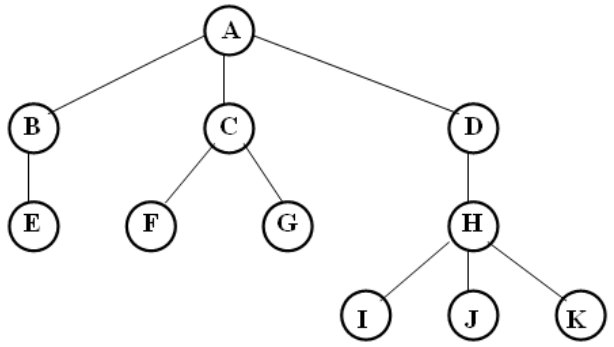
- Nó: índice em um vetor de células
  - Os nós podem ficar em qualquer posição no vetor, sem seguir nenhuma ordenação
  - A raiz não precisa ficar necessariamente no nó 1
- Filhos: ordenados da esquerda para a direita numa lista
  - Estrutura das listas: contígua, encadeada, etc.

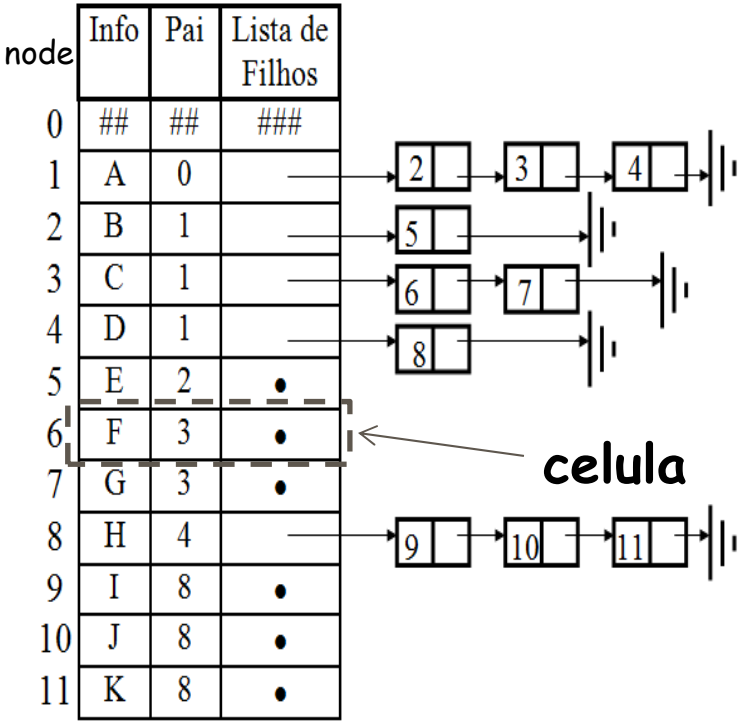




Lista de filhos

```
typedef int node;
struct celfilho  {
  node filho;
  celfilho *prox;
};
typedef celfilho *lista;
typedef celfilho *posicao;
struct celula {
  char info;
  node pai;
  lista listaFilhos;
};
struct arvore  {
  node raiz;
  celula Elementos[51];
  int ncel;
};
```



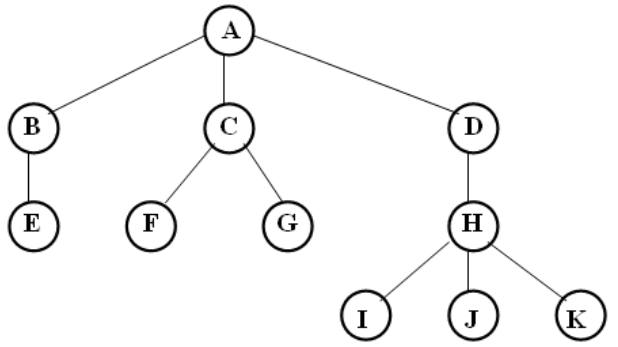


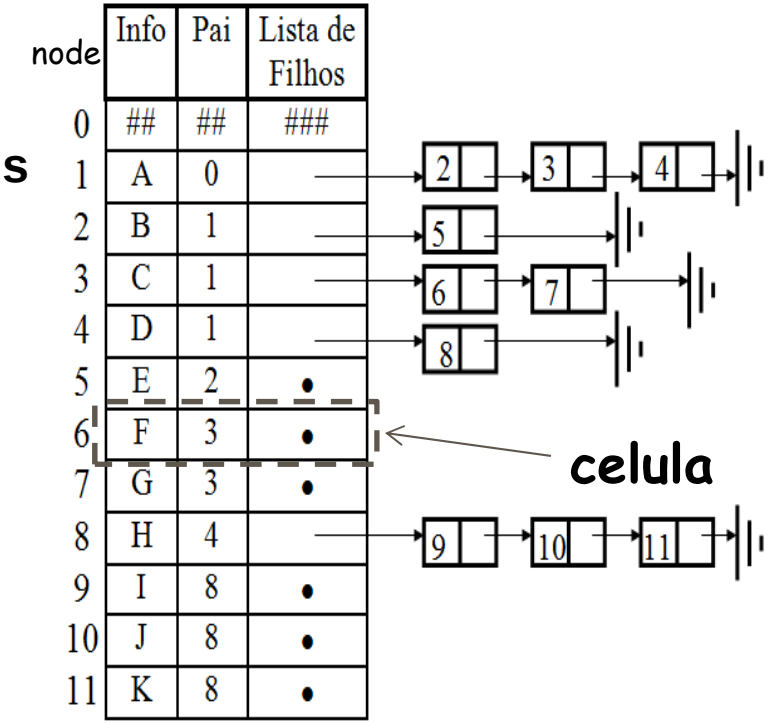
Lista de filhos

```
node Raiz (arvore A) {
  if (A.ncel == 0)
    return NULL;
  return A.raiz;
}
```

```
void Esvaziar (arvore *A){
  //Liberar todas as células de filhos
  A->raiz = NULL;
  A->ncel = 0;
}
```

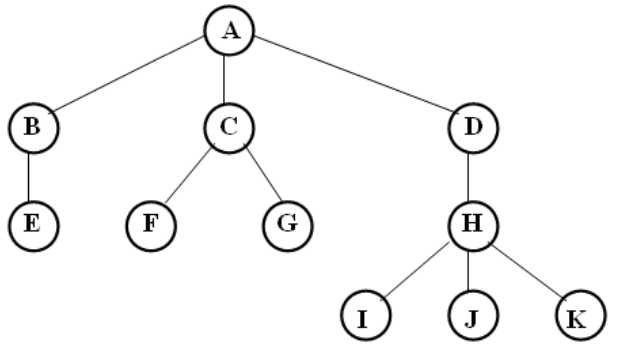
```
char Elemento (node n, arvore A) {
  return A.Elementos[n].info;
}
```



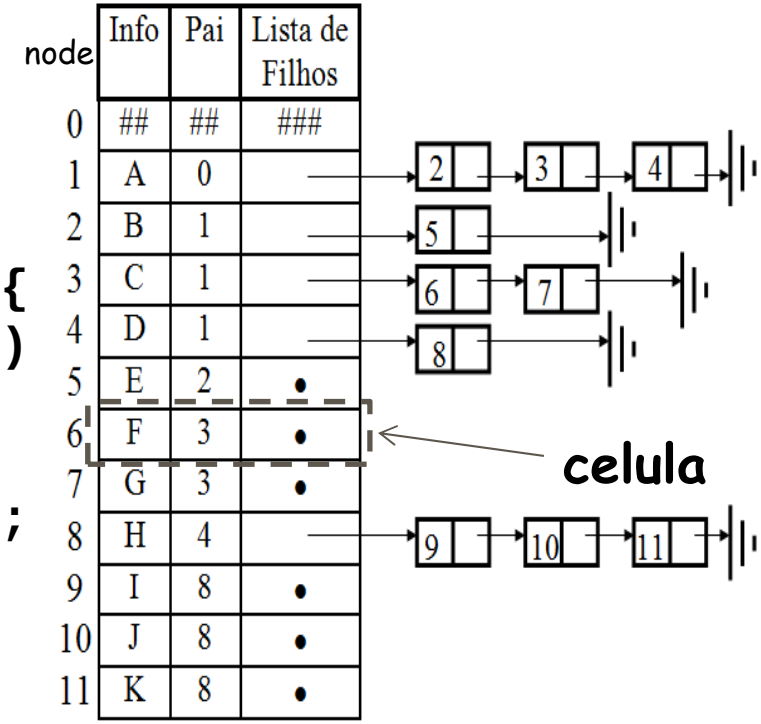


Lista de filhos

```
node Pai (node n, arvore A) {
        return A.Elementos[n].pai;
}
```

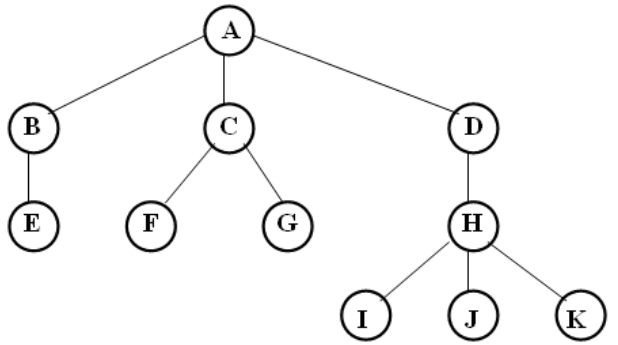


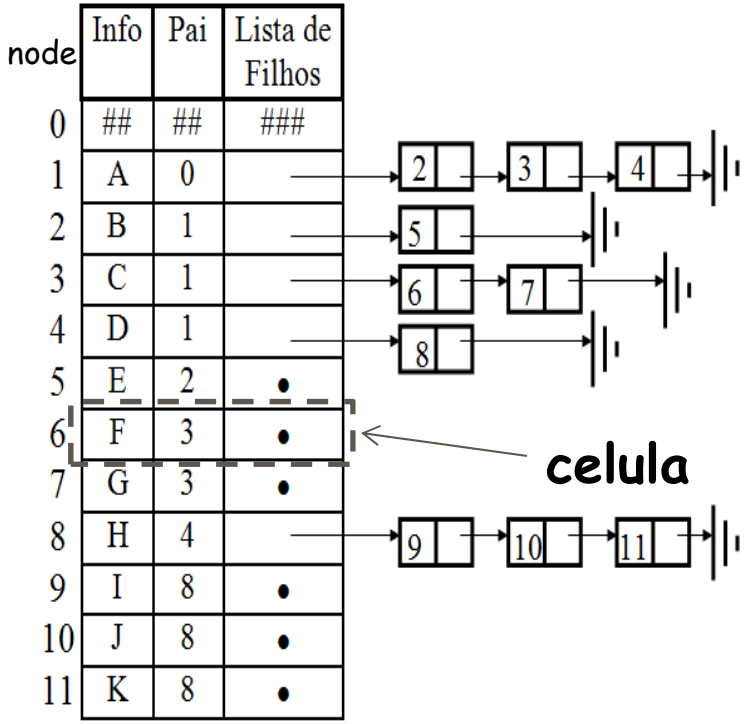

```
node FilhoEsquerdo (node n, arvore A) {
  if (A.Elementos[n].listaFilhos==NULL)
     return NULL;
  else return
     A.Elementos[n].listaFilhos->filho;
}
```



Lista de filhos

```
node IrmaoDireito (node n, arvore A)
{
    node pai;
    posicao idir, p;
    if (n == A.raiz)
        return NULL;
    pai = A.Elementos[n].pai;
    p = A.Elementos[pai].listaFilhos;
    while (p->filho != n)
        p = p->prox;
    idir = p->prox;
    if (idir == NULL)
        return NULL;
    else
        return idir->filho;
}
```





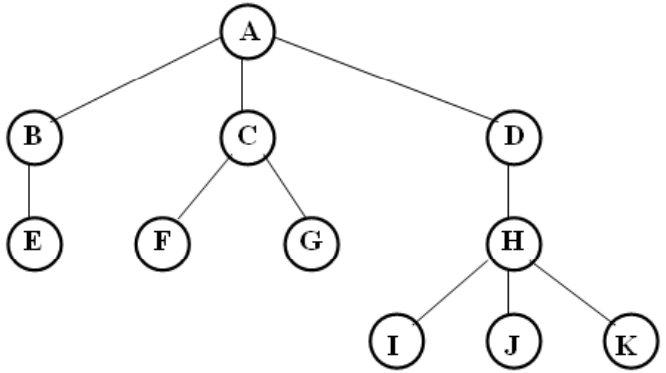
CES-11

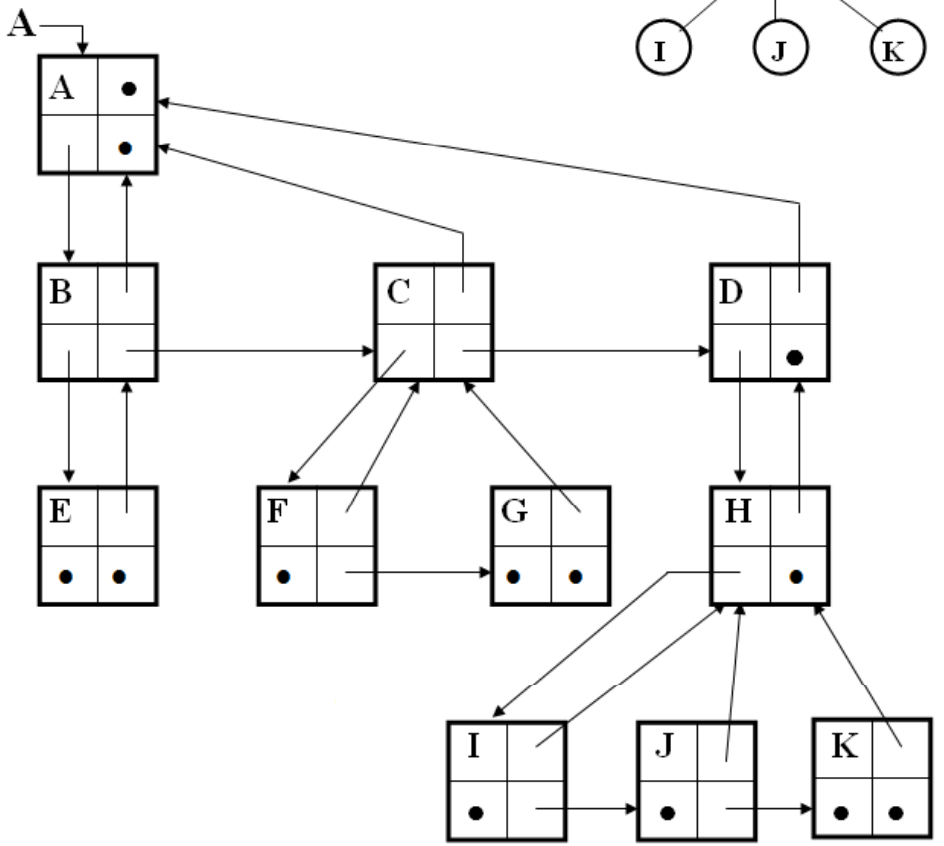
- Árvores
  - Operações sobre uma árvore
  - Estruturas para armazenar árvores
    - Contígua
    - Contígua melhorada
    - Encadeada direta
    - Listas de filhos
    - Encadeamento de pais e irmãos

Encadeamento de pais e irmãos

celula

info	pai
fesq	idir





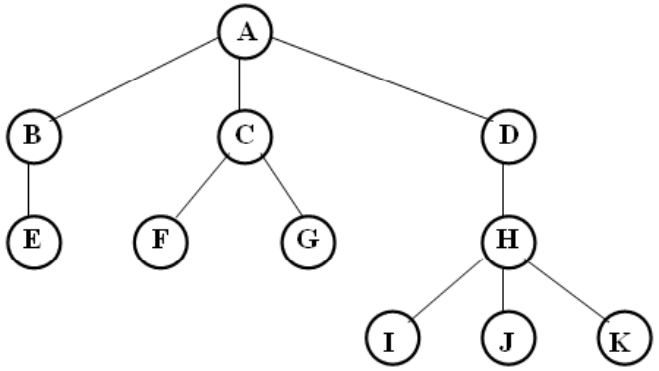

```
struct celula {
   char info;
   celula *pai, *fesq, *idir;
};

typedef celula *node;
typedef celula *arvore;

arvore A1, A2, A3;
```

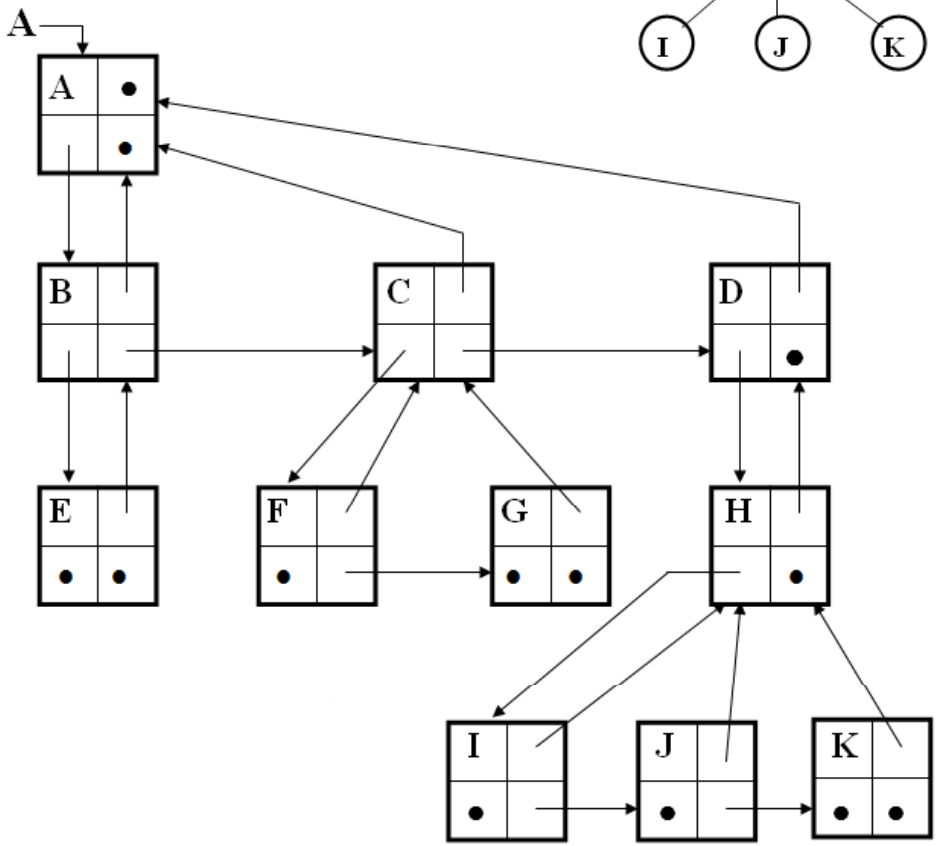
Encadeamento de pais e irmãos

celula

info	pai
fesq	idir

```
node FilhoEsquerdo (node n,arvore A){
    return n->fesq;
}

node IrmaoDireito (node n,arvore A){
    return n->idir;
}

char Elemento (node n,arvore A){
    return n->info;
}

node Pai(node n,arvore A){
    return n->pai;
}

node Raiz (arvore A){
    return A;
}
```

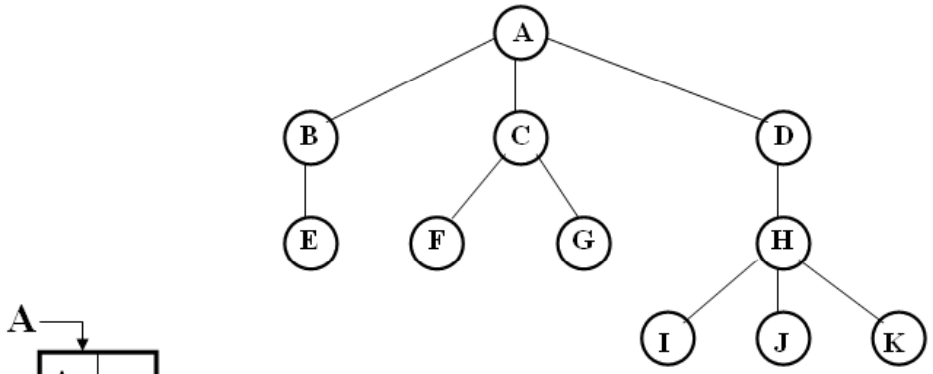


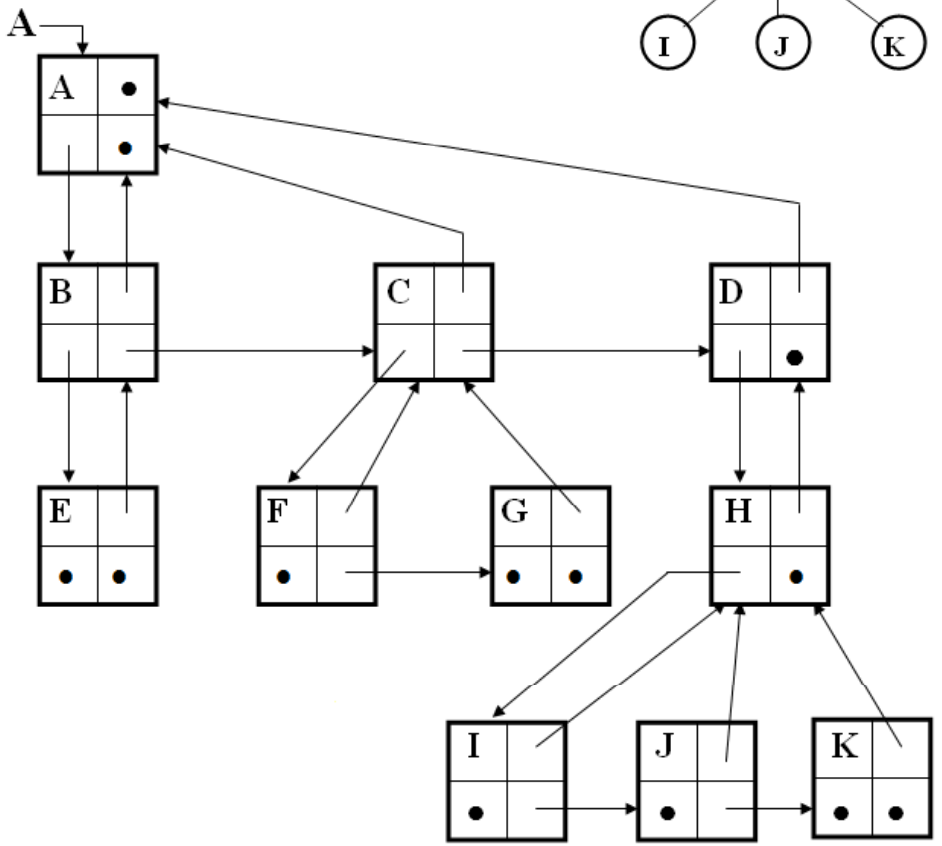


Encadeamento de pais e irmãos

celula

info	pai
fesq	idir

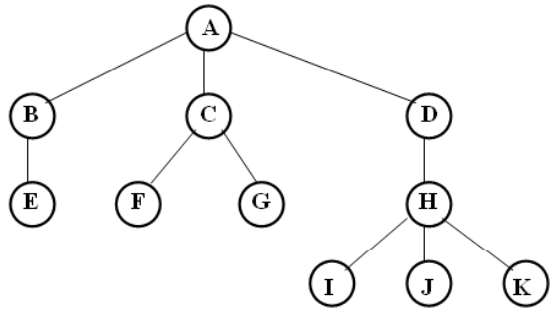





```
void Esvaziar (node *n) {
  if (*n == NULL)
    return;
  Esvaziar (&(*n)->fesq);
  Esvaziar (&(*n)->idir);
  free (*n);
  *n = NULL;
}
```

Encadeamento de pais e irmãos

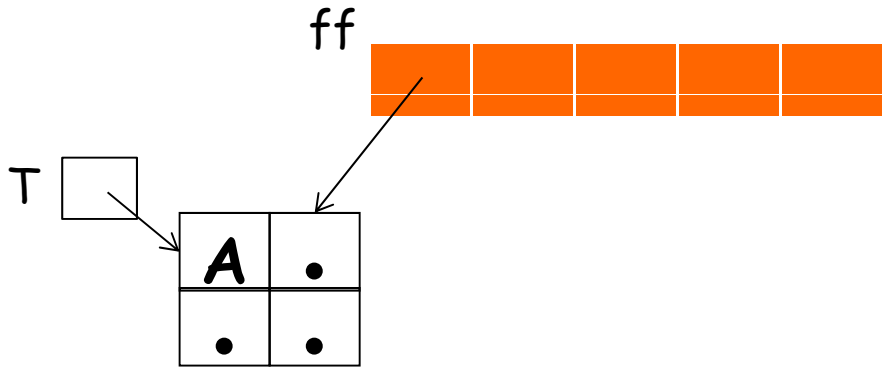
- Leitura iterativa de uma árvore

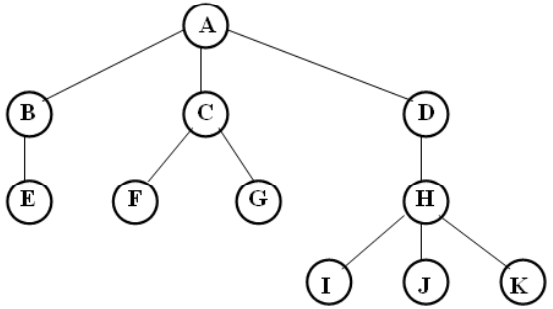



```
A arvore tem raiz? (s/n): s

Digite a informacao da raiz: A

Quantos filhos tem A? 3
1o filho de A: B
2o filho de A: C
3o filho de A: D

Quantos filhos tem B? 1
1o filho de B: E

Quantos filhos tem C? 2
1o filho de C: F
2o filho de C: G

Quantos filhos tem D? 1
1o filho de D: H

Quantos filhos tem E? 0

Quantos filhos tem F? 0

Quantos filhos tem G? 0

Quantos filhos tem H? 3
1o filho de H: I
2o filho de H: J
3o filho de H: K

Quantos filhos tem I? 0

Quantos filhos tem J? 0

Quantos filhos tem K? 0
```

Encadeamento de pais e irmãos

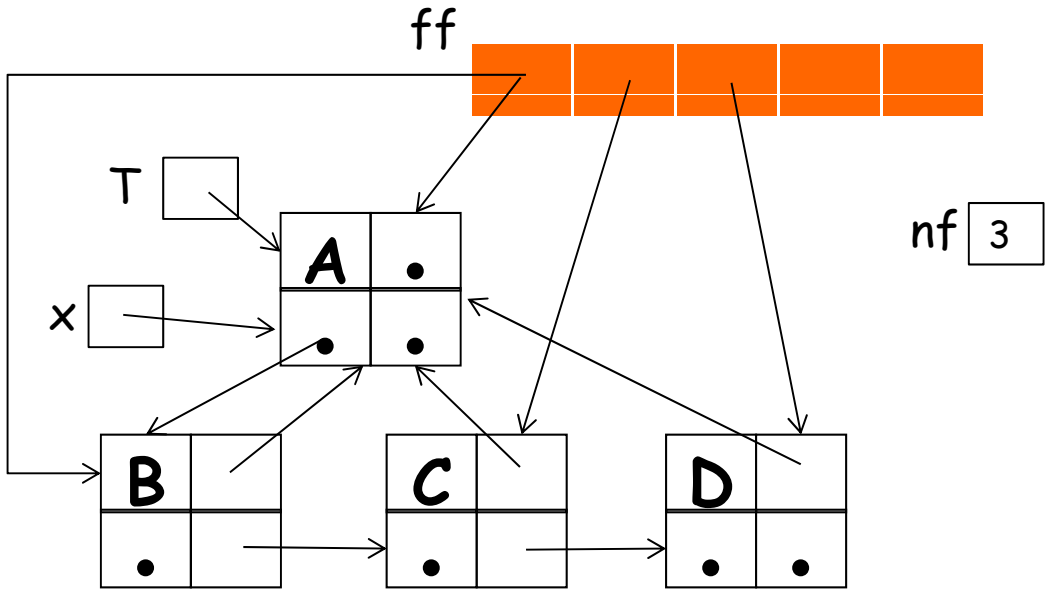
- Leitura iterativa de uma árvore
  - Ideia semelhante ao percurso em largura: usa-se uma fila **ff** para guardar os nós já introduzidos na estrutura, mas cujos filhos ainda não foram processados
  - Começa-se pela raiz, alocando-a e colocando-a em **ff**

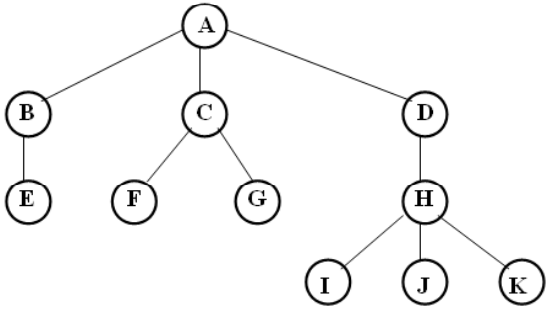




Encadeamento de pais e irmãos

- Leitura iterativa de uma árvore
  - Enquanto a fila **ff** não ficar vazia:
    - Eliminar o primeiro elemento de **ff**, guardando-o em **x**
    - Perguntar número de filhos deste nó, guardando-o em **nf**
    - Alocar cada filho na árvore e colocá-lo também na fila **ff**





Encadeamento de pais e irmãos

- Leitura iterativa de uma árvore
  - O *loop* na fila **ff** continua até que fique vazia: assim, todos os nós da árvore serão armazenados.
    - Código em C fica como exercício.
- Uma forma mais elegante de armazenar uma árvore é receber como entrada a sua forma parentética
  - Ex: (A (B (E)) (C (F) (G)) (D (H (I) (J) (K))))
    - Exercício: escrever código em C que leia a forma parentética e monte a árvore.