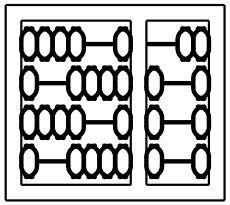


Universidade Estadual de Campinas - UNICAMP
Instituto de Computação - IC

Árvores: Parte 3

Tópicos

1. Nós com campo pai
2. Árvores binárias de busca
3. Implementação
4. Exemplos de funções

- Nós com campo pai

– Em algumas aplicações é necessário o acesso imediato ao pai de qualquer nó na árvore binária.

```
typedef struct No {
  int info ;      /* exemplo do conteudo de cada no */
  struct No *pai ;  /* apontador para o pai */
  struct No *esq, *dir ; /* apontadores para os filhos */
} ArvBin, NoArvBin ;
```

• Exemplo: Dado o endereço x de um nó, encontrar o endereço do nó seguinte na árvore percorrida em inordem.

Encontre 1 erro !!!

```
 /* Recebe um no x e devolve o no seguinte na arvore
 percorrida em inordem (supoe x != NULL) */

NoArvBin  *Seguinte (NoArvbin *x)  {
  if  (x->dir != NULL) {
    no *y = x->dir ;
    while (y->esq != NULL)  y = y->esq ;
    return y ;
  }
  while(x->pai != NULL && x->pai->dir == x)
    x = x->pai ;
 return x ;
}
```

• Exemplo: Dado o endereço x de um nó, encontrar o endereço do nó seguinte na árvore percorrida em inordem.

Encontre 1 erro !!!

```
 /* Recebe um no x e devolve o no seguinte na arvore
 percorrida em inordem (supoe x != NULL) */

NoArvBin  *Seguinte (NoArvbin *x)  {
  if  (x->dir != NULL) {
    no *y = x->dir ;
    while (y->esq != NULL)  y = y->esq ;
    return y ;
  }
  while(x->pai != NULL && x->pai->dir == x)
    x = x->pai ;
 return x ;
}
```

• Exemplo: Dado o endereço x de um nó, encontrar o endereço do nó seguinte na árvore percorrida em inordem.

Sem o erro !!!

```
 /* Recebe um no x e devolve o no seguinte na arvore
 percorrida em inordem (supoe x != NULL) */

NoArvBin  *Seguinte (NoArvBin *x)  {
  if  (x->dir != NULL) {
    NoArvBin *y = x->dir ;
    while (y->esq != NULL)  y = y->esq ;
    return y ;
  }
  /*subir na arvore enquanto x for filho direito de um no */
  while(x->pai != NULL && x->pai->dir == x)
    x = x->pai ;
  return x->pai ;     <-------
}
```

Árvores binárias de busca

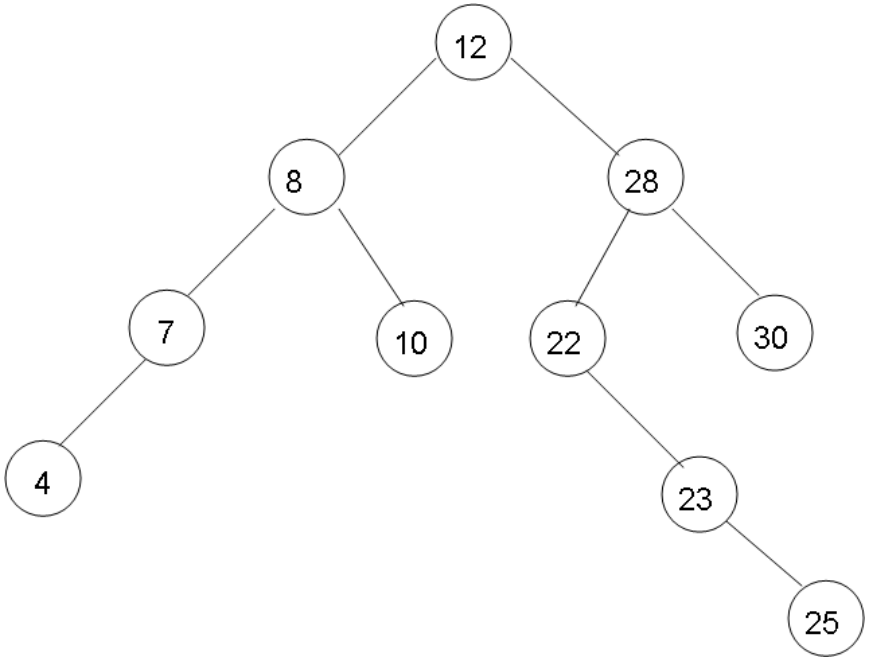
• Definição: Uma árvore binária é dita de busca, com relação a um campo **chave**, se cada nó X tem a seguinte propriedade:

– A chave de X é:

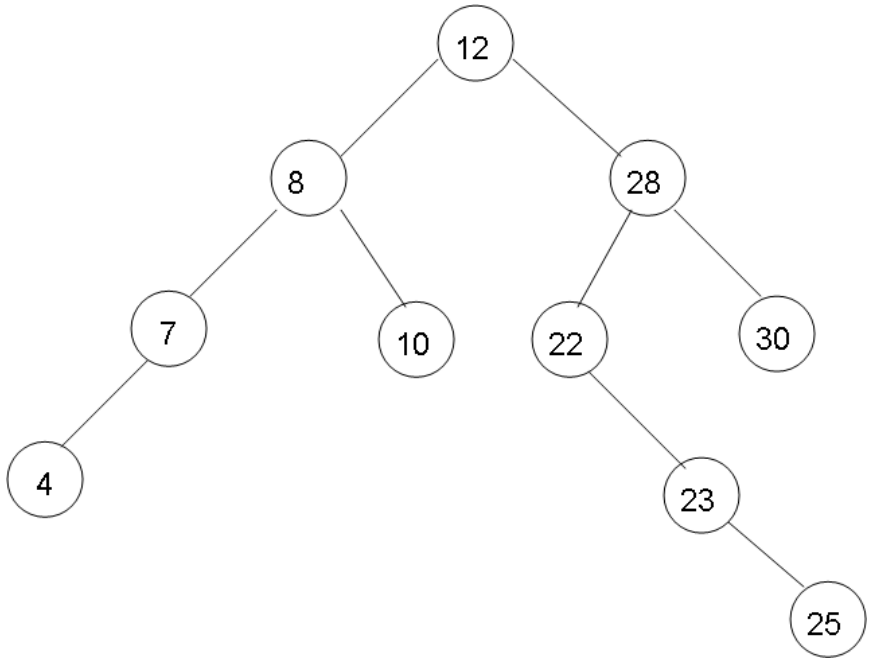
1. maior que a chave de qualquer nó na subárvore esquerda de X e

2. menor que a chave de qualquer nó na subárvore direita de X (menor ou igual caso se permita a repetição de nós de mesmo conteúdo na árvore).

- Exemplo:

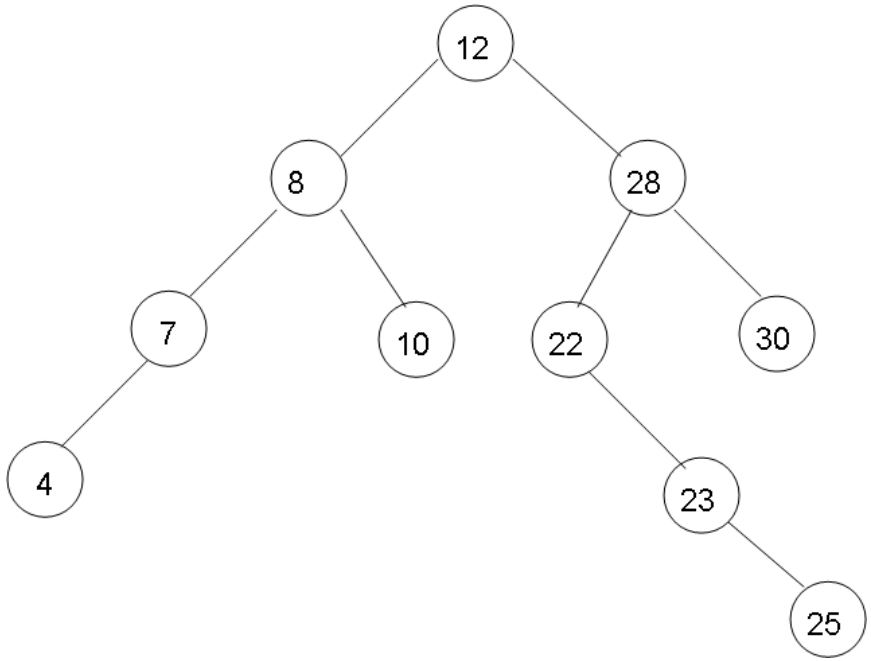


- Exemplo:



Como esq.chave $<$ X.chave $<$ dir.chave, então o percurso de tal árvore em *inordem*, com relação à respectiva chave, resulta numa visita em ordem crescente de seus nós.

- Exemplo:



Como esq.chave $<$ X.chave $<$ dir.chave, então o percurso de tal árvore em *inordem*, com relação à respectiva chave, resulta numa visita em ordem crescente de seus nós.

Percurso inordem: *4 7 8 10 12 22 23 25 28 30*

- Propriedade:

– A complexidade da busca em uma árvore binária pode variar de $\lceil log_2(n+1) \rceil$ a n.

- Implementação:

```
typedef struct No {
   int chave ;
   int info ;
   struct No *esq ;
   struct No *dir ;
} ArvBin, NoArvBin ;
```

– Neste caso, a chave é de um tipo linearmente ordenado como, por exemplo, *int* ou *char* e que naturalmente pode ser o próprio campo *info* de cada nó.

Exemplo de funções

- Criar uma árvore de busca vazia:

```
ArvBin *abb_cria(void) {
  return NULL ;
}
```

Exemplo de funções

- Imprimir os valores dos nós:

```
/* o campo info corresponde a propria chave */

ArvBin *abb_imprime(NoArvBin *p) {
  if ( p != NULL) {
    abb_imprime (p->esq) ;    /* visita nos em inordem */
    printf(''%d\n'', p->info) ;
    abb_imprime (p->dir) ;
  }
}
```

• As funções principais sobre uma árvore binária de busca são naturalmente:

1. busca: função que busca um elemento na árvore
2. insere: função que insere um novo elemento na árvore
3. remove: função que remove um elemento da árvore

- Buscar recursivamente um nó cuja *chave* tem um certo valor:

```
/* Recebe k e uma arvore binaria de busca p e devolve NULL
ou o no com chave igual a k */

NoArvBin *Busca (ArvBin *p, int k) {
 if (p == NULL || p->chave == k)
   return p ;
 if (p->chave > k)
   return Busca (p->esq, k) ;
 else
   return Busca (p->dir, k) ;
}
```

- Buscar iterativamente um nó cuja *chave* tem um certo valor:

- Buscar iterativamente um nó cuja *chave* tem um certo valor:

```
/* Recebe k e uma arvore binaria de busca p e devolve NULL
ou no com chave igual a k */

NoArvBin *BuscaI (ArvBin *p, int k) {
 while (p != NULL && p->chave != k) {
   if (p->chave > k) p = p->esq ;
   else p = p->dir ;
 }
 return p ;
}
```

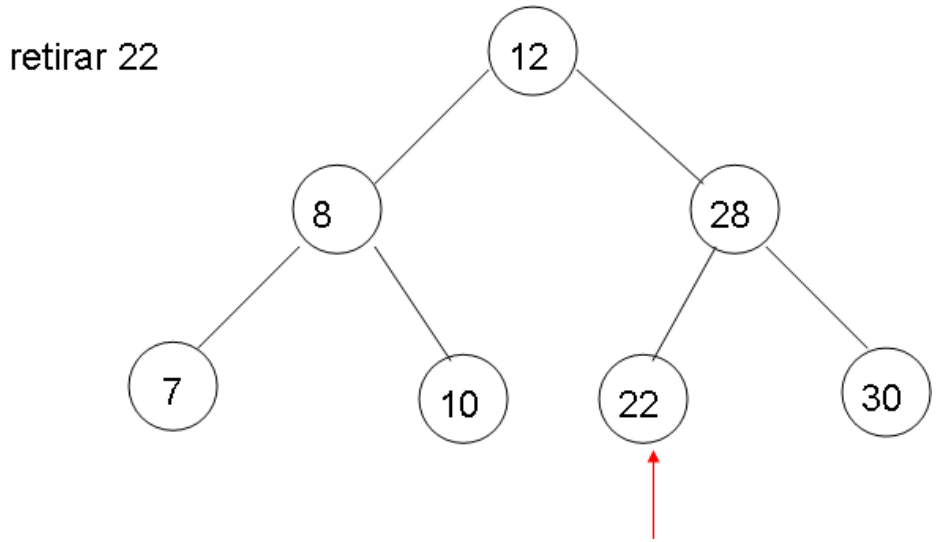
- Inserir um novo nó em uma árvore binária de busca:

```
/* Devolve o endereco da nova subarvore criada */

void Insere (int  k, ArvBin **p, boolean *pertence) {
  if (*p == NULL) {
    *pertence = FALSE ;
    *p = (NoArvBin *)malloc(sizeof(NoArvBin)) ;
    (*p)->chave = k ;
    (*p)->esq = NULL ;
    (*p)->dir = NULL ;
  }
  else
    if (k == (*p)->chave)  *pertence = TRUE ;
    else if (k < (*p)->chave)
      Insere (k, &(*p)->esq, pertence) ;
      else
        Insere (k, &(*p)->dir, pertence) ;
}
```

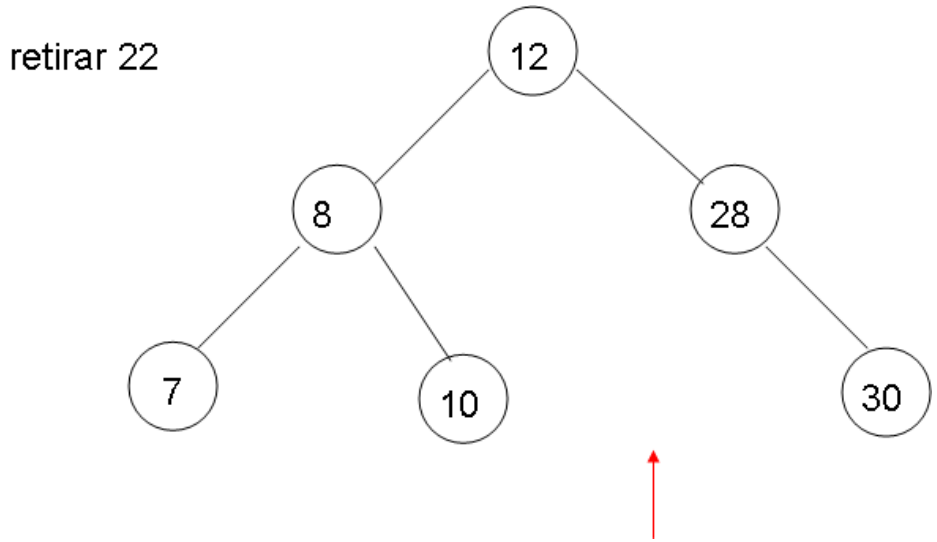
- Remover um nó com chave k da árvore binária:

– Caso 1: elemento a ser retirado é uma folha $\rightarrow$ retira sem problemas



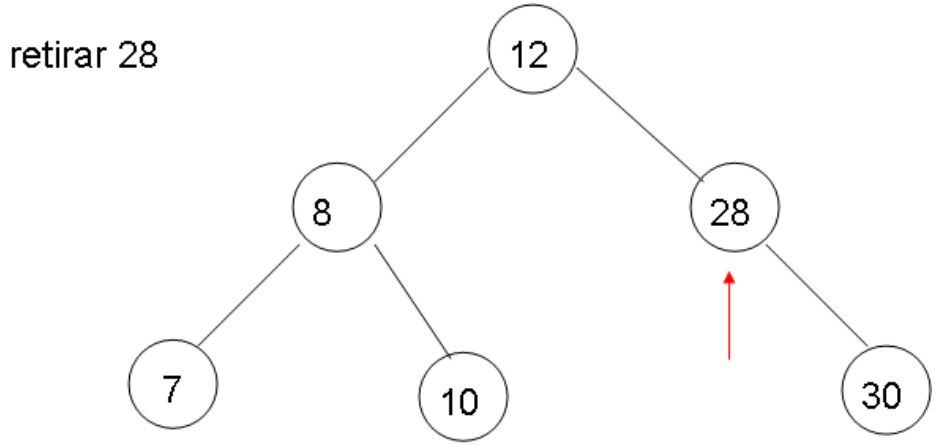
- Remover um nó com chave k da árvore binária:

– Caso 1: elemento a ser retirado é uma folha $\rightarrow$ retira sem problemas



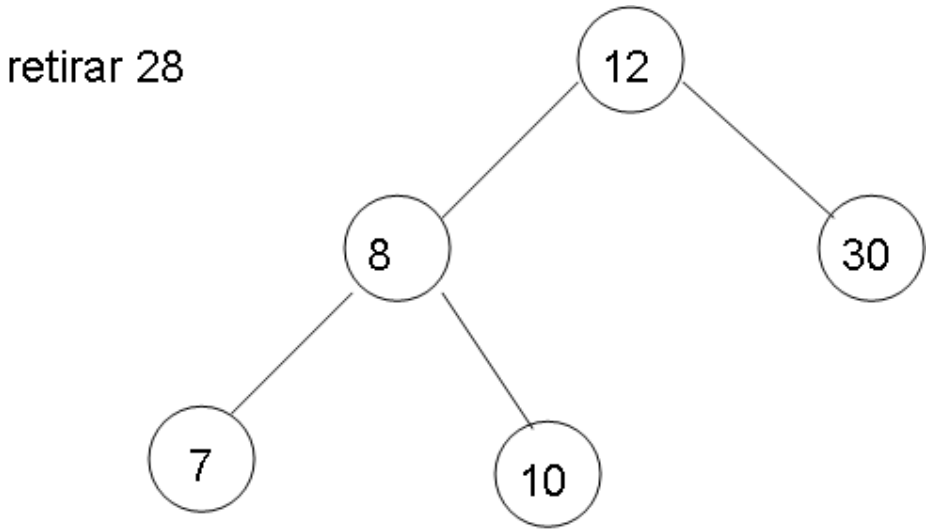
• Remover um nó com chave k da árvore binária:

– Caso 2: elemento a ser retirado tem no máximo 1 filho → o filho ocupa o lugar do nó a ser retirado:



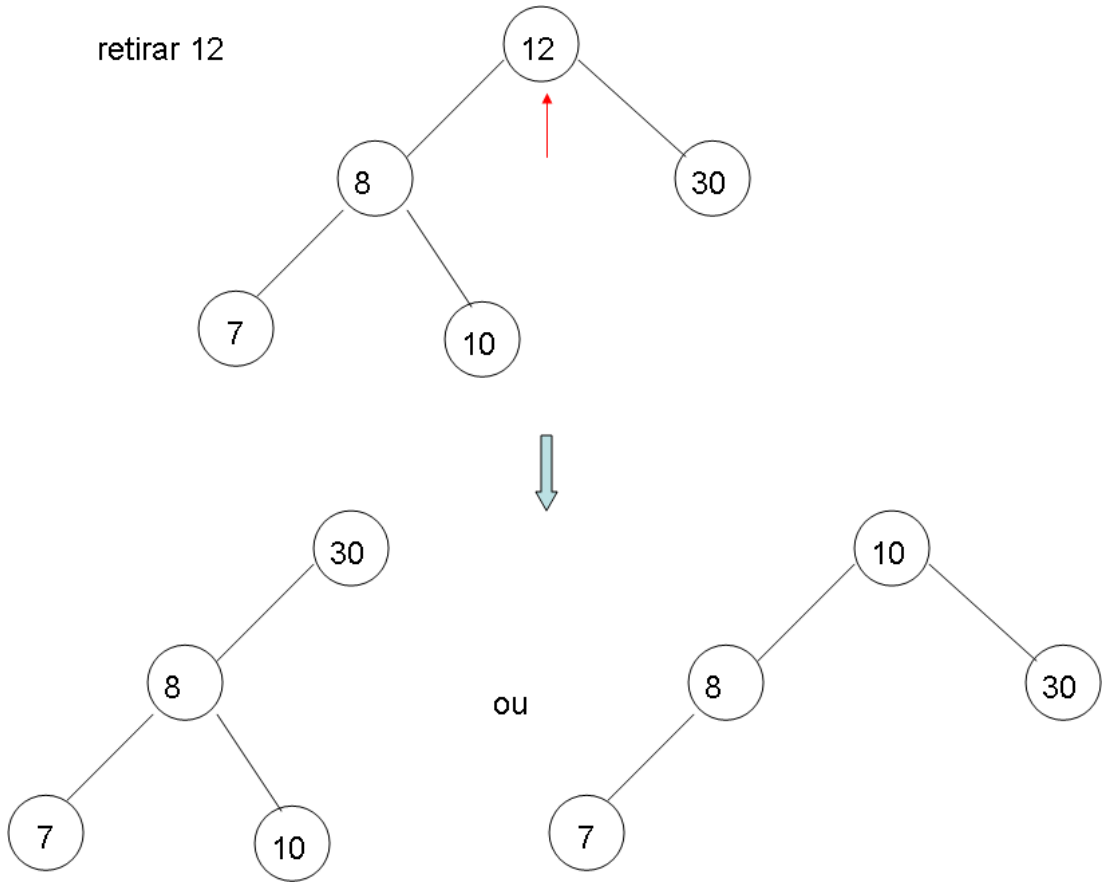
• Remover um nó com chave k da árvore binária:

– Caso 2: elemento a ser retirado tem no máximo 1 filho → o filho ocupa o lugar do nó a ser retirado:



• Remover um nó com chave k da árvore binária:

– Caso 3: elemento a ser retirado tem no máximo 2 filhos → neste caso, o elemento a ser retirado deve ser substituído pelo elemento mais à direita da subárvore esquerda ou pelo elemento mais à esquerda da subárvore direita.



```
NoArvBin *g ;           /* variavel global */
void Remove (int k, ArvBin  **p) {
  if (*p == NULL) printf("Nao ha chave k\n") ;
  else
    if (k < (*p)->info) Remove (k, &(*p)->esq) ;
    else
     if (k > (*p)->info Remove (k, &(*p)->dir) ;
     else {                 /* remover */
       g = *p ;
       if (g->dir == NULL) *p = g->esq ;    /* soh tem um...
       else                                   filho ? */
        if (g->esq == NULL) *p = g->dir ;
        else
       /* troca pelo mais a dir da subarvore esq */
         troca (&g->esq) ;
       free (g) ;
     }
}
```

- A função troca:

```
 void troca (NoArvBin **r) {
   if ((*r)->dir != NULL) troca (&(*r)->dir) ;
   else {
     g->info = (*r)->info ;
     g = *r ;
     *r = (*r)->esq ;
   }
}
```