

CES-11

Algoritmos e Estruturas de Dados

Carlos Alberto Alonso Sanches
Juliana de Melo Bezerra

CES-11

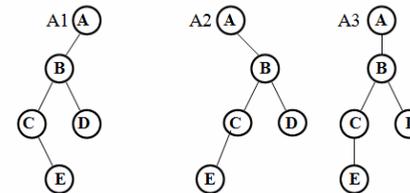
- Árvores binárias
- Árvores binárias de busca
 - Conceito
 - Operações

CES-11

- **Árvores binárias**
- Árvores binárias de busca
 - Conceito
 - Operações

Árvores binárias

- Uma árvore binária é:
 - uma árvore vazia;
 - ou uma árvore onde qualquer nó possui:
 - nenhum filho;
 - ou um filho esquerdo ou um filho direito;
 - ou os dois filhos citados.



A1 é binária? **Sim**

A2 é binária? **Sim**

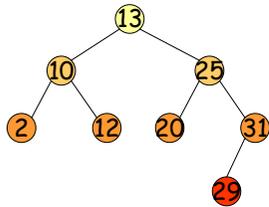
A3 é binária? **Não**

No nó A, não há distinção entre filho esquerdo e direito...

Árvores binárias

- Exemplo ordenação de nós numa árvore binária:

- Por nível (largura): 13 10 25 2 12 20 31 29
- Pré-ordem: 13 10 2 12 25 20 31 29
- Pós-ordem: 2 12 10 20 29 31 25 13
- Ordem-central: 2 10 12 13 20 25 29 31

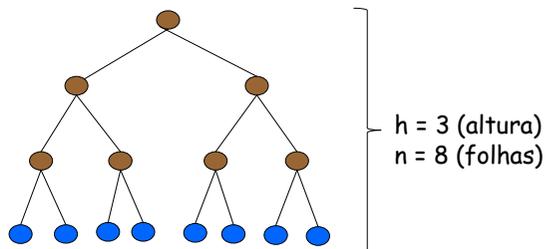


Árvores binárias

- Uma árvore binária é completa quando os nós internos têm exatamente dois filhos e as folhas têm a mesma distância da raiz.
- Uma árvore binária completa de altura h tem exatamente $2^h - 1$ nós internos e 2^h folhas.
- Numa árvore binária completa com n folhas, a distância da raiz até qualquer folha é $\lg n$.
- Prove!!

Árvores binárias

- Seja a árvore binária completa:



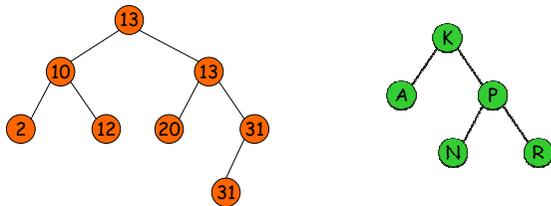
$$n = 2^h \longleftrightarrow h = \lg n$$

CES-11

- Árvores binárias
- Árvores binárias de busca
 - Conceito
 - Operações

Árvores binárias de busca

- Em cada nó de uma árvore binária de busca, os valores armazenados na sub-árvore esquerda não são maiores que o valor do próprio nó, e os valores armazenados na sub-árvore direita não são menores que o valor do próprio nó.

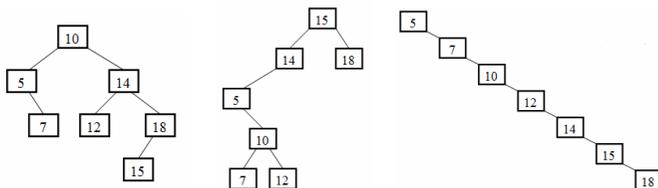


Árvores binárias de busca

- Uma utilização: implementar dicionários
 - Consiste em um conjunto de chaves ou elementos **distintos** (sem repetições)
 - De modo geral, cada chave tem uma informação associada
 - Operações comuns:
 - Busca de uma chave (teste de pertinência)
 - Inserção de uma chave
 - Eliminação de uma chave
 - Esvaziamento de uma árvore

Árvores binárias de busca

- Exemplo:
 - Seja o conjunto $C = \{5, 7, 10, 12, 14, 15, 18\}$
 - Há várias árvores binárias de busca possíveis:



CES-11

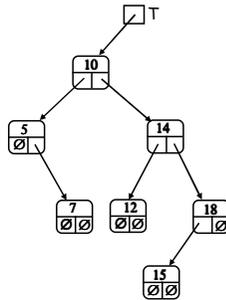
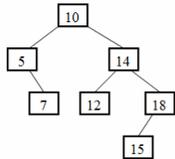
- Árvores binárias
- Árvores binárias de busca
 - Conceito
 - Operações

Operações

Declaração

```
struct Celula {
    int elem;
    Celula *fesq, *fdir;
};
typedef Celula *Dicionario;

Dicionario T = NULL;
```



Operações

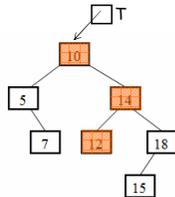
Operação membro(x,dic)

- Algoritmo de busca é simples e direto
- Repetição que começa na raiz:
 - Compare com o valor armazenado no nó.
 - Se for igual, a busca chegou ao fim.
 - Se for menor, vá para a sub-árvore esquerda.
 - Se for maior, vá para a sub-árvore direita.
 - Se não houver como continuar, o valor não está na árvore.

Operações

Operação membro(x,dic)

```
logic membro (int x, Dicionario dic) {
    if (dic == NULL)
        return FALSE;
    else if (x == dic->elem)
        return TRUE;
    else if (x < dic->elem)
        return membro(x, dic->fesq);
    else
        return membro(x, dic->fdir);
}
```



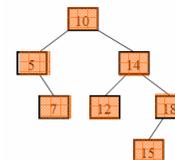
- Exemplo: membro(13,T)
- Pior caso: tempo proporcional à altura da árvore

Operações

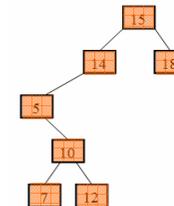
Operação inserir(x,&dic)

- Insera o elemento x no dicionário dic
- A **seqüência** de inserções determina o **formato** final da árvore

10, 14, 5, 12, 7, 18, 15



15, 14, 5, 18, 10, 12, 7



Operações

Operação inserir(x,&dic)

```
void inserir (int x, Dicionario *dic) {
    if (*dic == NULL) {
        *dic = (Celula *) malloc (sizeof(Celula));
        (*dic)->elem = x;
        (*dic)->fesq = NULL;
        (*dic)->fdire = NULL;
    }
    else if (x < (*dic)->elem)
        inserir (x, &(*dic)->fesq);
    else if (x > (*dic)->elem)
        inserir (x, &(*dic)->fdire);
}
```

Como o elemento não existe, um novo nó é criado

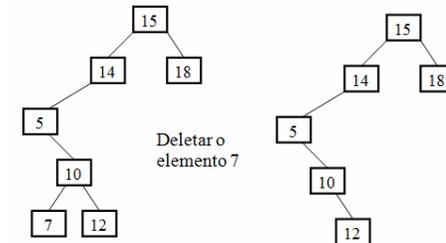
Chamada recursiva

E quando x é igual a (*dic)->elem? Não faz nada!

Operações

Operação eliminar(x,&dic)

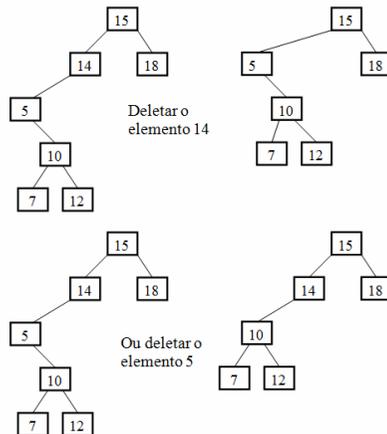
- Caso 1: Eliminar uma folha
- Basta apagá-la da estrutura



Operações

Operação eliminar(x,&dic)

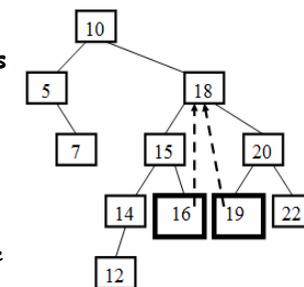
- Caso 2: Eliminar nó com um único filho
- Basta fazer esse filho ocupar a sua posição original



Operações

Operação eliminar(x,&dic)

- Caso 3: Eliminar nó com os dois filhos
- Um elemento deve ocupar o nó do elemento a ser eliminado.
- Duas opções:
 - Maior elemento da sub-árvore esquerda
 - Menor elemento da sub-árvore direita desse nó



Por convenção, será o escolhido

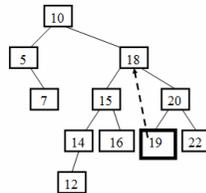
Operações

Operação eliminar(x,&dic)

- Utiliza-se o operador **eliminarMin (&dic)**
 - Esse operador elimina o menor elemento de dic, retornando o seu valor

No exemplo:

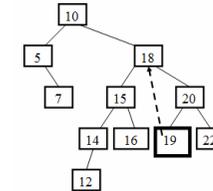
- Aplica-se esse operador à sub-árvore direita do nó **18**
- O valor retornado **19** é armazenado no nó que continha o **18**



Operações

Operador eliminarMin(&dic)

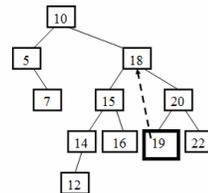
```
int eliminarMin (Dicionario *dic) {
    Celula *p;
    int ret;
    if ((*dic)->fesq == NULL) {
        ret = (*dic)->elem;
        p = *dic;
        *dic = (*dic)->fdir;
        free(p);
    }
    else ret = eliminarMin (&(*dic)->fesq);
    return ret;
}
```



Operações

Operação eliminar(x,&dic)

```
void eliminar (int x, Dicionario *dic) {
    Celula *p;
    if (*dic != NULL) {
        if (x < (*dic)->elem) eliminar (x, &(*dic)->fesq);
        else if (x > (*dic)->elem) eliminar(x, &(*dic)->fdir);
        else if ((*dic)->fesq == NULL && (*dic)->fdir == NULL) {
            free (*dic);
            *dic = NULL;
        }
        else if ((*dic)->fesq == NULL){
            p = *dic;
            *dic = (*dic)->fdir;
            free (p);
        }
        else if ((*dic)->fdir == NULL){
            p = (*dic);
            *dic = (*dic)->fesq;
            free (p);
        }
        else
            (*dic)->elem = eliminarMin (&(*dic)->fdir);
    }
}
```



Operações

- Nessas operações, manteve-se a ordenação própria de uma árvore binária de busca: a chave do pai é maior que a do filho esquerdo e menor que a do filho direito.
- No entanto, não foi possível garantir seu balanceamento.
- Seria desejável que a altura dessa árvore continuasse proporcional ao logaritmo do número de nós: assim, as operações seriam mais eficientes.
- Há diversos algoritmos para balanceamentos de árvores...

Fim

