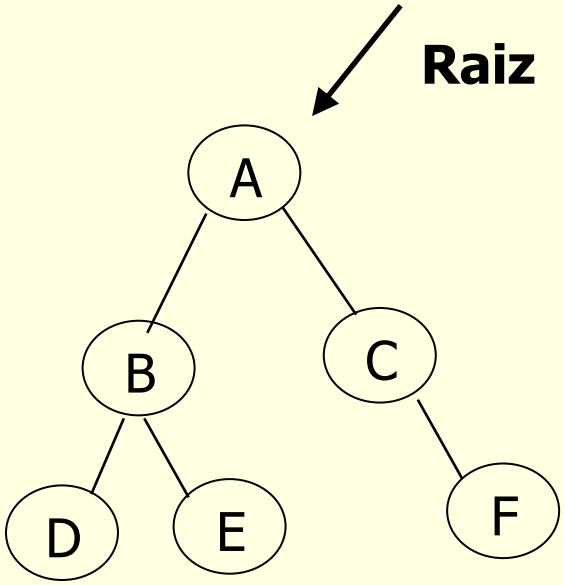
Árvores binárias de busca

SCC-202 – Algoritmos e Estruturas de Dados I

Lucas Antiqueira

Árvores binárias

- Árvores de grau 2, isto é, cada nó tem dois filhos, no máximo



Terminologia:

- filho esquerdo
- filho direito
- informação

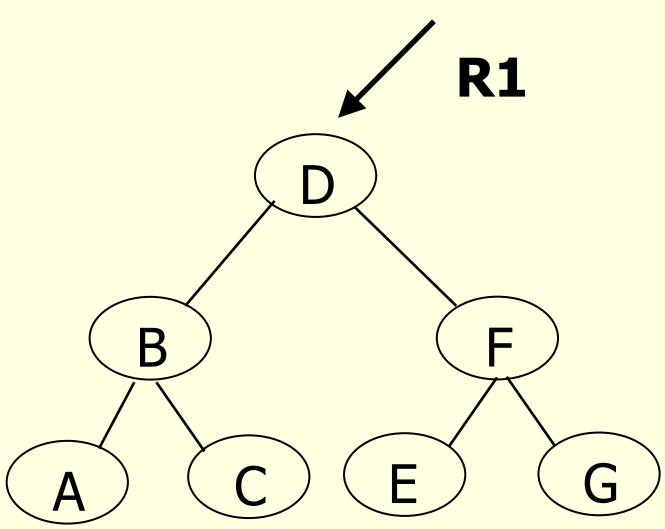
Árvores binárias de busca (ABB)

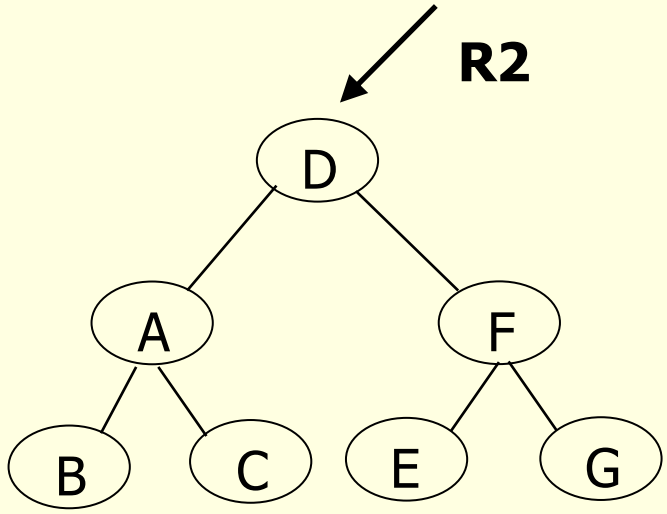
- Também chamadas “árvores de pesquisa” ou “árvores ordenadas”

- *Definição*
  - Uma árvore binária com raiz R é uma ABB se:
    - a chave (informação) de cada nó da subárvore esquerda de R é menor do que a chave do nó R (em ordem alfabética, por exemplo)
    - a chave de cada nó da subárvore direita de R é maior do que a chave do nó R
    - as subárvores esquerda e direita também são ABBs

ABB

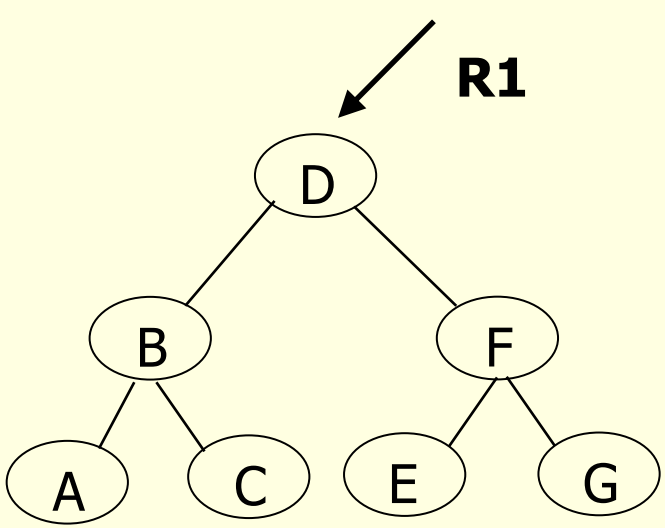
- Exemplos



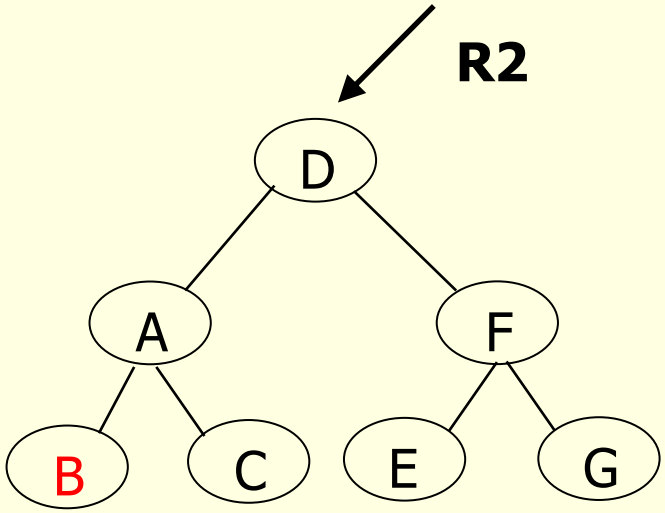


ABB

- Exemplos



é ABB



não é ABB

ABB

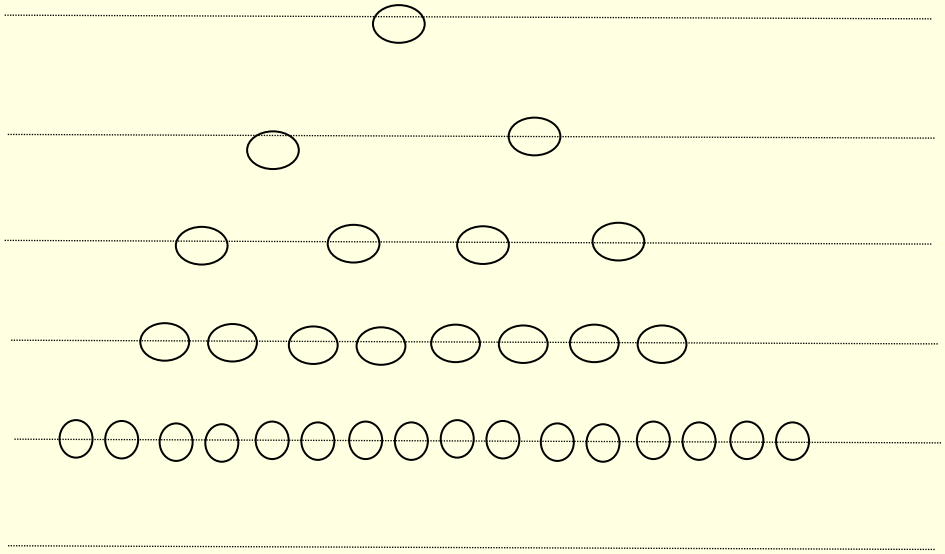
- Imagine a situação
  - Sistema de votação por telefone (“Você decide”)
    - Cada número só pode votar uma vez
    - Um sistema deve armazenar todos os números que já ligaram
    - A cada nova ligação, deve-se consultar o sistema para verificar se aquele número já votou; o voto é computado apenas se o número ainda não votou
    - A votação deve ter resultado on-line

ABB

- Solução com ABBs
  - Cada número de telefone é armazenado em uma ABB
  - Suponha que em um determinado momento, a ABB tenha 1 milhão de telefones armazenados
  - Surge nova ligação e preciso saber se o número está ou não na árvore (se já votou ou não)

ABB

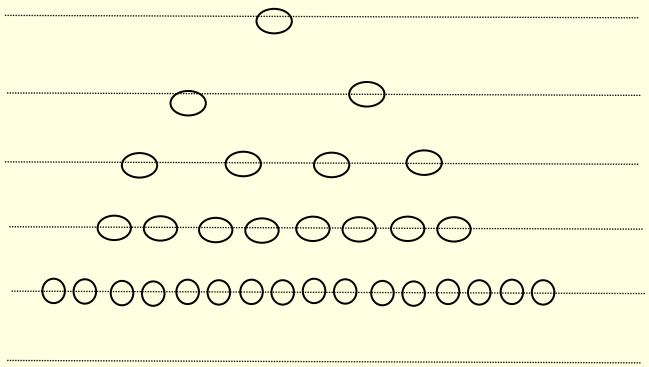
- Considere uma ABB com chaves uniformemente distribuídas (árvore cheia)



ABB

- Responda
  - Quantos elementos cabem em uma árvore de N níveis, como a anterior?
  - Como achar um elemento em uma árvore assim, a partir da raiz?
  - Quantos nós temos que visitar, no máximo, para achar o telefone na árvore, ou ter certeza de que ele não está na árvore?

ABB



nível 1

nível 2

nível 3

nível 4

nível 5

nível N

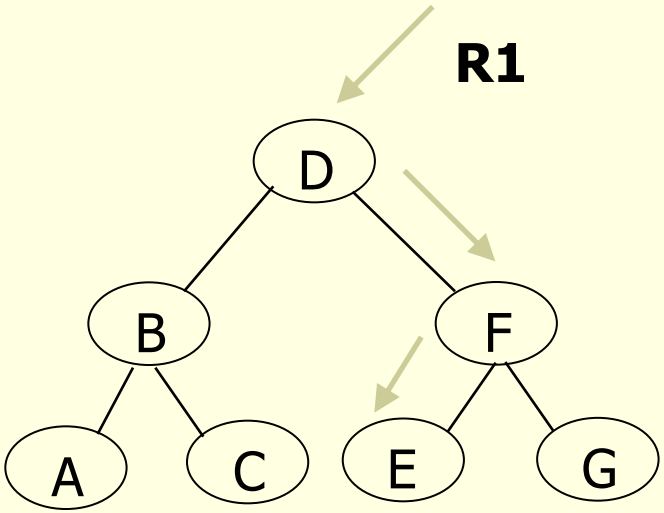
Nível	Quantos nós cabem?
1	1
2	3
3	7
4	15
10	1.023
13	8.191
16	65.535
18	262.143
20	1.048.575
30	1.073.741.823
...	...
N	**$2^N - 1$**

ABB

- Por que utilizar uma ABB é eficiente?
- Para se buscar em uma ABB
  - Em cada nó, compara-se o elemento buscado com o elemento presente
    - Se menor, percorre-se a subárvore esquerda
    - Se maior, percorre-se subárvore direita
  - Desce-se verticalmente até as folhas, no pior caso, sem passar por mais de um nó em um mesmo nível
  - Portanto, no pior caso, a busca passa por tantos nós quanto for a altura da árvore

ABB

- Exemplo: busca pelo elemento E na árvore abaixo



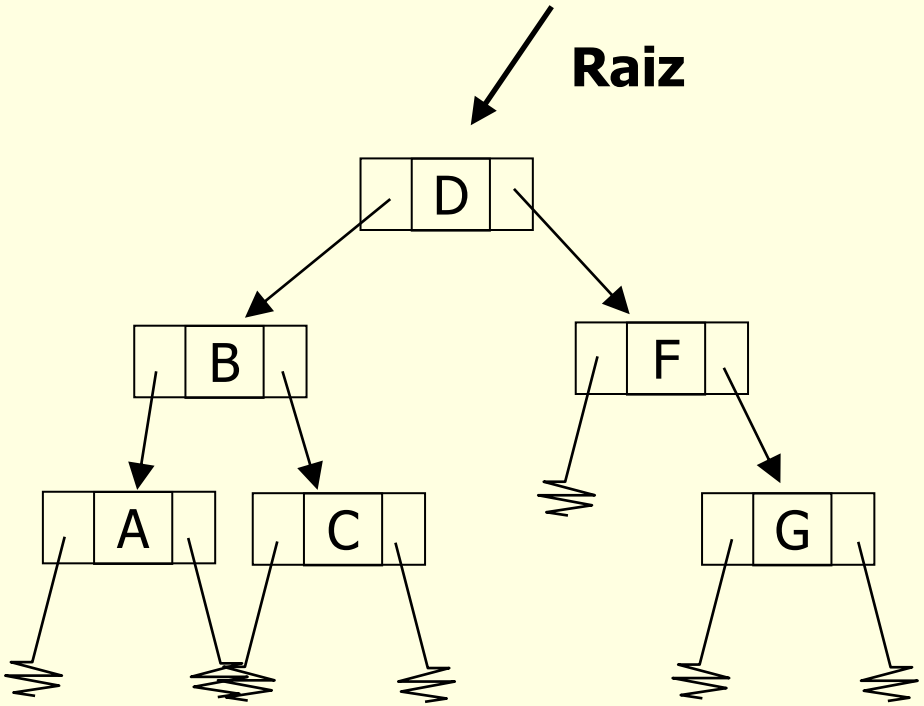
3 consultas

ABB

- Por que uma ABB é eficiente?
  - Buscas muito rápidas!!!

ABB

- Representação



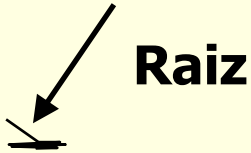


ABB vazia

ABB

- Declaração igual a das árvores binárias convencionais. Contudo, a manipulação é diferente

```
typedef int elem;

typedef struct no {
        elem info;
        struct no *esq, *dir;
} no;

typedef struct {
        no *raiz;
} ABB;
```

ABB

- Operações sobre a ABB
  - Devem considerar a ordenação dos elementos da árvore
    - Por exemplo, na inserção, deve-se procurar pelo local certo na árvore para se inserir um elemento

ABB

- Exercício
  - Construa a partir do início uma ABB com os elementos K, E, C, P, G, F, A, T, M, U, V, X, Z

TAD ABB

- *Operações básicas*
  - Busca
  - Inserção
  - Remoção

TAD ABB

- Busca
  - Comparando o parâmetro “chave” com a informação do nó “raiz”, 4 casos podem ocorrer
    - A árvore é vazia => a chave não está na árvore => fim do algoritmo
    - Elemento da raiz = chave => achou o elemento (está no nó raiz) => fim do algoritmo
    - Chave < elemento da raiz => chave pode estar na subárvore esquerda
    - Chave > elemento da raiz => chave pode estar na subárvore direita
  - Pergunta: quais os casos que podem ocorrer para a subárvore esquerda? E para a subárvore direita?

TAD ABB

- Busca
  - Comparando o parâmetro “chave” com a informação do nó “raiz”, 4 casos podem ocorrer
    - A árvore é vazia => a chave não está na árvore => fim do algoritmo
    - Elemento da raiz = chave => achou o elemento (está no nó raiz) => fim do algoritmo
    - Chave < elemento da raiz => chave pode estar na subárvore esquerda
    - Chave > elemento da raiz => chave pode estar na subárvore direita
  - Pergunta: quais os casos que podem ocorrer para a subárvore esquerda? E para a subárvore direita?

Os mesmos!

TAD ABB

- Exercício
  - Implementação da sub-rotina de busca de um elemento na árvore

TAD ABB

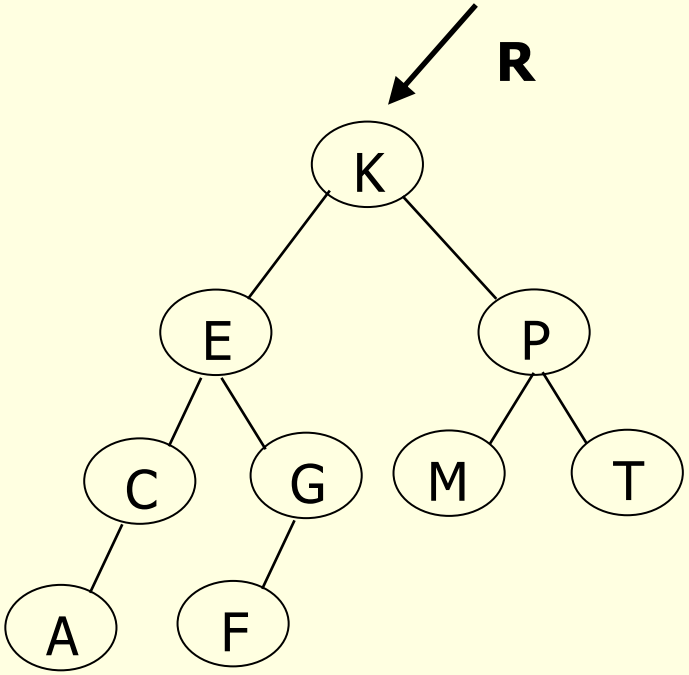
- Inserção
  - Estratégia geral
    - Inserir elementos como nós folha (sem filhos)
    - Procurar o lugar certo e então inserir
  - Comparando o parâmetro “chave” com a informação no nó “raiz”, 4 casos podem ocorrer
    - A árvore é vazia => insere o elemento, que passará a ser a raiz; fim do algoritmo
    - Elemento da raiz = chave => o elemento já está na árvore; fim do algoritmo
    - Chave < elemento da raiz => insere na subárvore esquerda
    - Chave > elemento da raiz => insere na subárvore direita

TAD ABB

- Exercício
  - Implementação da sub-rotina de inserção de um elemento na árvore

TAD ABB

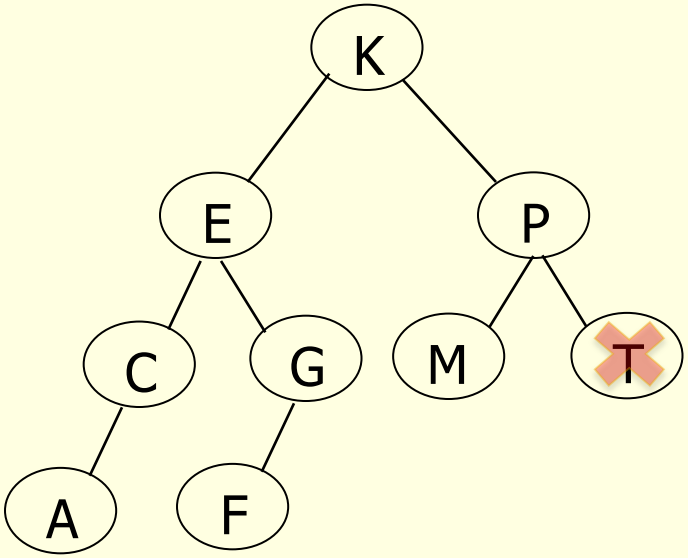
- Remoção
  - Para a árvore abaixo, remova os elementos T, C e K, nesta ordem



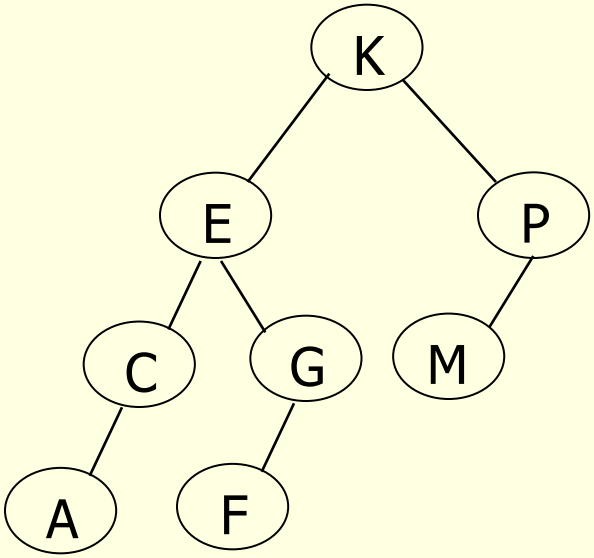
ABB

- Caso 1 (remover T): o nó p a ser removido não tem filhos
  - Remove-se o nó
  - p aponta para NULL

ABB



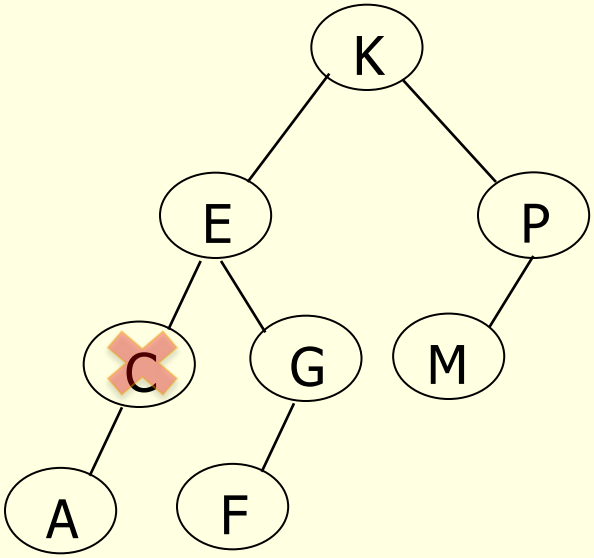
ABB



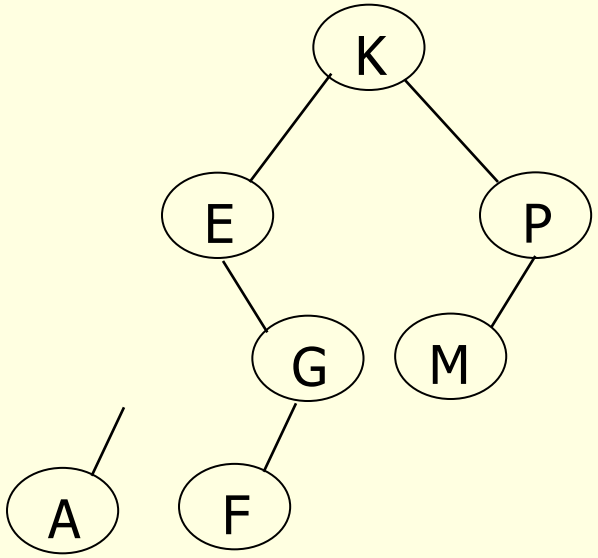
ABB

- Caso 2 (remover C): o nó a ser removido tem 1 único filho
  - Remove-se o nó
  - “Puxa-se” o filho para o lugar do pai

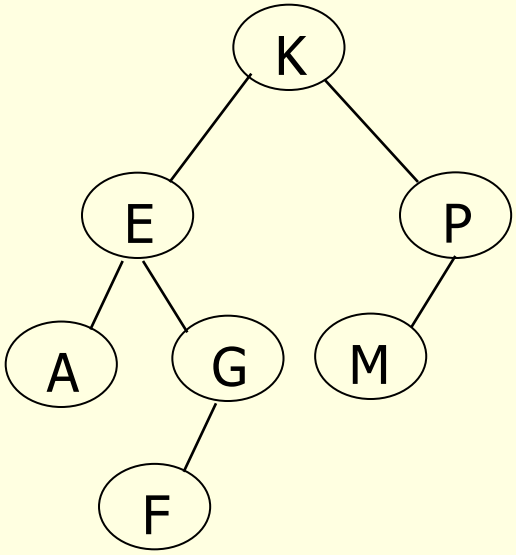
ABB



ABB



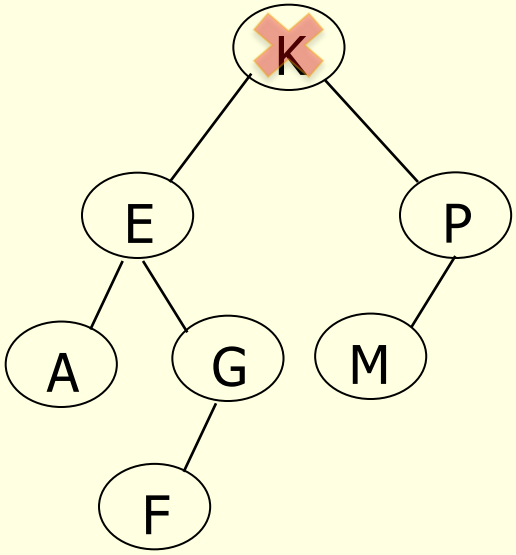
ABB



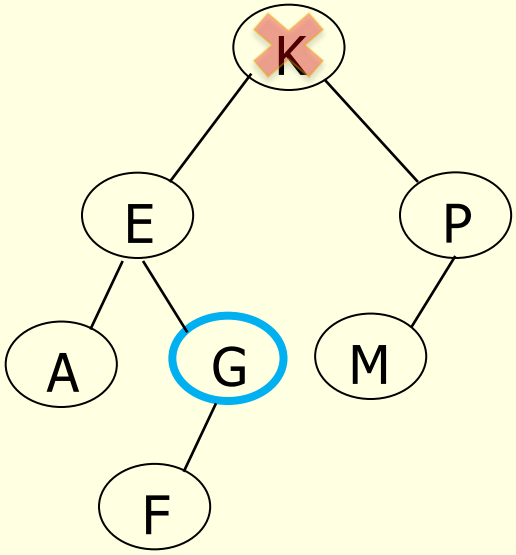
ABB

- Caso 3 (remover K): o nó p a ser removido tem 2 filhos
  - Acha-se a maior chave da subárvore esquerda
  - p recebe o valor dessa chave
  - Remove-se a maior chave da subárvore esquerda

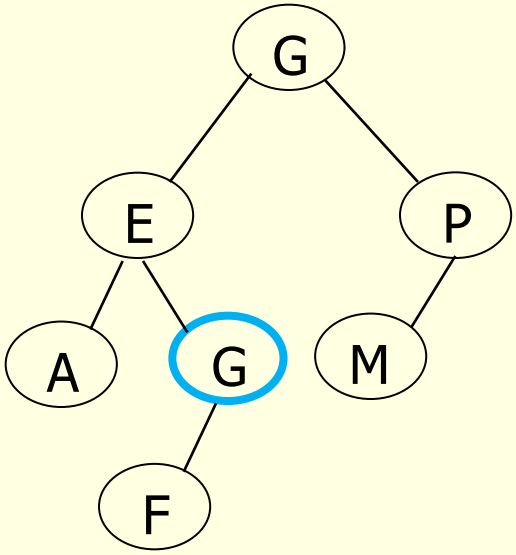
ABB



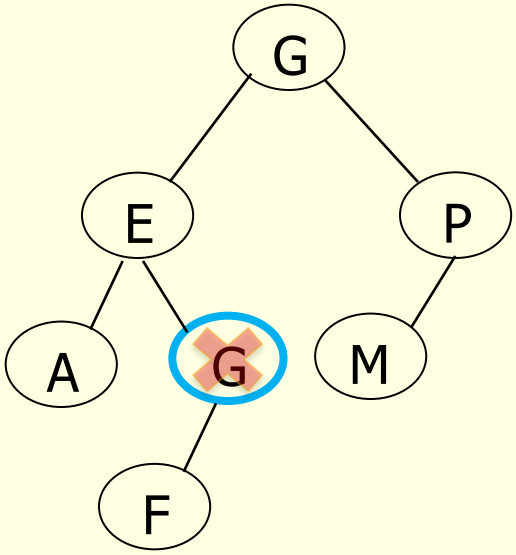
ABB



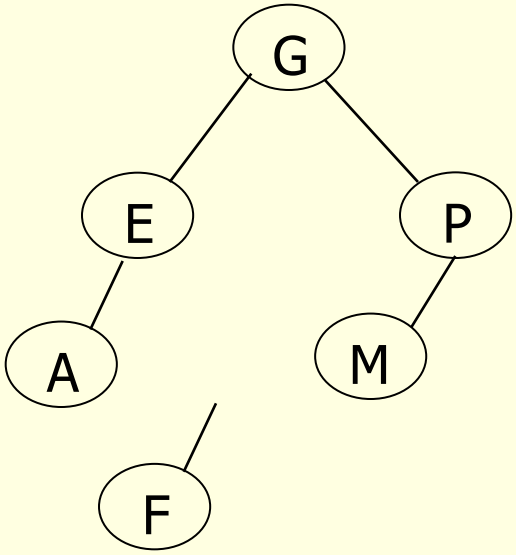
ABB



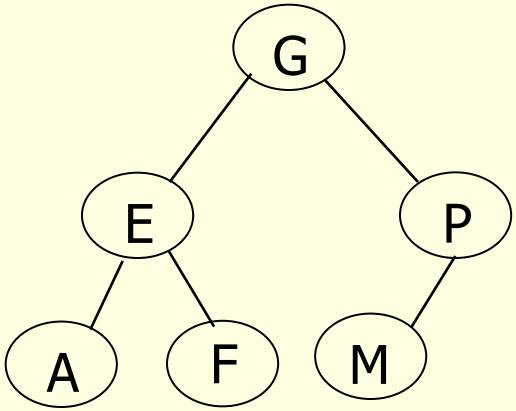
ABB



ABB



ABB



TAD ABB

- Exercício
  - Implementação da sub-rotina de remoção de um elemento da árvore

Créditos

- *Material gentilmente cedido pelo Prof. Thiago A. S. Pardo*