Árvores

Notas de aula da disciplina IME 04-10820
ESTRUTURAS DE DADOS I

Paulo Eustáquio Duarte Pinto
(pauloedp arroba ime.uerj.br)

junho/2018

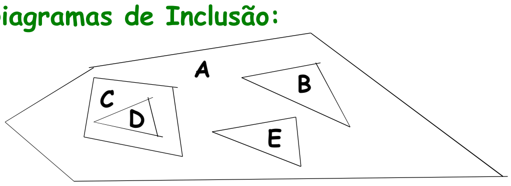
Conceitos sobre Árvores

Servem para representar estruturas hierarquiza-das, notadamente Bancos de Dados.

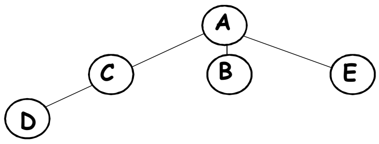
Def: Uma árvore enraizada T é um conjunto finito de vértices (nós) tais que:
- -T é um conjunto vazio, ou
- -existe um nó r (raiz) tal que os demais nós são particionados em m > 1 conjuntos não vazios, as subárvores de r e sendo cada um desses conjuntos uma árvore.

Representações de Árvores

a) Diagramas de Inclusão:



b) Representação Hierárquica:



Representações de Árvores

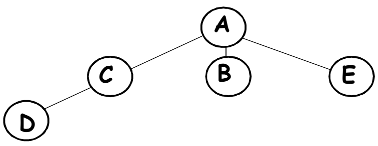
c) Diagramas de Barras:

```
A-------------
    C---------
        D-----
    B---------
    E---------
```

d) Parênteses Aninhados:

(A (C (D)) (B) (E))

Conceitos



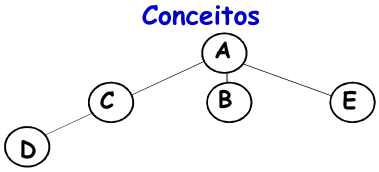
A é pai de C, B, E; C é pai de D; A é avô de D;

B é filho de A; D é filho de C; B é tio de D;

B é irmão de C; D é sobrinho de E;

D, B e E são folhas; A e C são nós intermediários;

Conceitos



A está no nível 1; D, no nível 3;

A altura da árvore é 3;

O caminho entre A e D tem comprimento 2; entre D e E, comprimento 3;

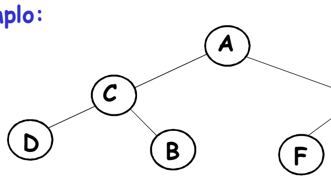
Uma Floresta é um conjunto de árvores;

Árvores Binárias

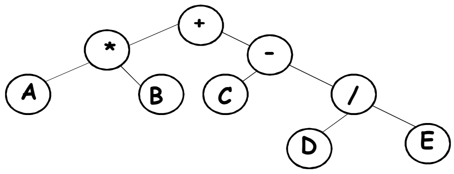
Def: Uma árvore binária enraizada T é um conjunto finito de vértices (nós) tais que:

- -T é um conjunto vazio, ou
- -existe um nó r (raiz) tal que os demais nós são particionados em 2 conjuntos, as subárvores esquerda (T_e) e direita (T_d) de r, sendo cada uma delas uma árvore binária.

Exemplo:



Conceitos de Árvores Binárias



Em uma árvore estritamente binária cada nó tem 0 ou 2 filhos.

Em uma árvore binária completa as subárvores nulas estão todas nos dois últimos níveis da árvore;

Em uma árvore binária cheia as subárvores (arestas, links) nulas estão todas no último nível;

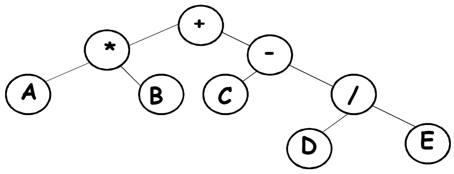
Exercícios

EXS10:

Desenhar todas as árvores binárias distintas para as chaves A B e C.

Desenhar uma árvore estritamente binária com 11 nós e altura 5.

Exercícios

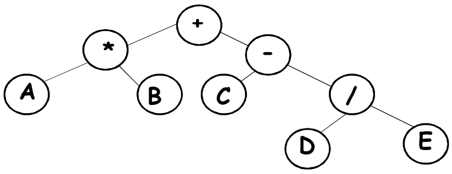


Quantos nós tem uma árvore binária cheia de altura h?

Qual o número mínimo de nós em uma árvore binária completa de altura h?

Quantas formas de árvores binárias completas distintas existem contendo n nós?

Conceitos

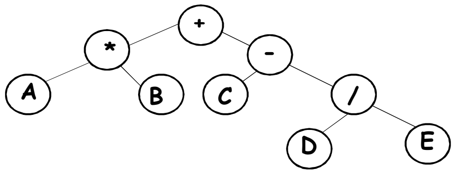


O número de subárvores nulas (links nulos) em uma árvore binária com n nós é n+1.

Existe um único caminho entre dois nós de uma árvore.

A árvore binária que representa uma expressão aritmética é uma árvore estritamente binária com os operandos nas folhas e operadores em nós intermediários.

Conceitos



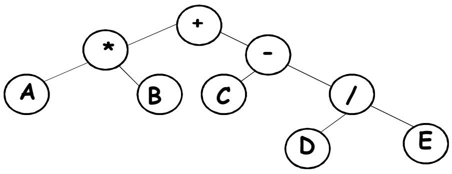
Altura h de uma árvore árvore binária completa com n nós:

$$2^{h-1} \le n \le 2^h - 1 \quad \Rightarrow \quad h = \lfloor \log_2 n \rfloor + 1 = \quad = \lceil \log_2 (n+1) \rceil$$

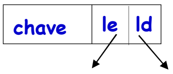
Altura de uma árvore binária com n nós:

$$\lfloor \log_2 n \rfloor + 1 \le h \le n$$

Representação



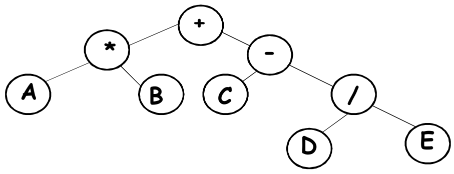
Representação computacional:



C/C++
```
typedef struct No* Arv;
struct No {char chave;
          Arv le,  ld;}
Arv p, q;
```

Tupy
```
tipo Arv:
      caracter chave
      Arv le, ld;
Arv p, q;
```

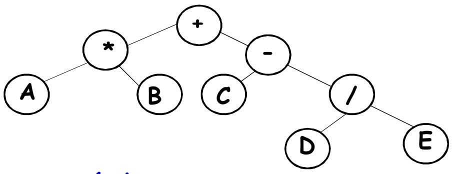

Percursos



Percursos em árvores binárias são formas sistemáticas de percorrer (visitar) os nós da árvore.

- -Percurso em nível
- -Percurso em pré-ordem
- -Percurso in-ordem (ordem simétrica)
- -Percurso em pós-ordem
- ...

Percursos

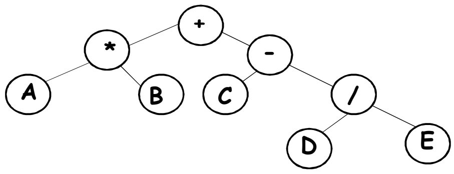


Percurso em nível:

a idéia é visitar um nível a cada vez, de cima para baixo e da esquerda para a direita

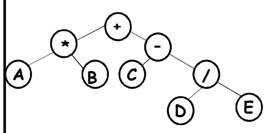
\+ * - A B C / D E

Percurso em nível




```
Percurso_nivel(T):
      Esvazia (Q);   Enfila (T);
      enquanto (f ≠ 0):
            p ← Q[f]
            Visita (p)
            Enfila (p.le)
            Enfila (p.ld)
            p ← Desenfila()
```

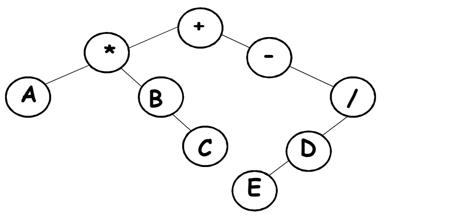

Percurso em nível




```
Percurso_nivel(T):
#Dados: Árvore binária T
     Esvazia (Q);   Enfila (T);

     enquanto (f ≠ 0):
          p ← Q[f]
          Visita (p)
          Enfila (p.le)
          Enfila (p.ld)
          p ← Desenfila ()
```

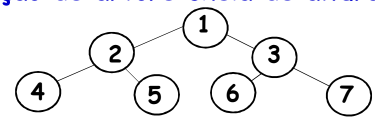
+	*	-					+
	*	-	A	B			*
		-	A	B	C	/	-
			A	B	C	/	A
				B	C	/	B
					C	/	C
D	E					/	/
D	E						D
	E						E

Exercício



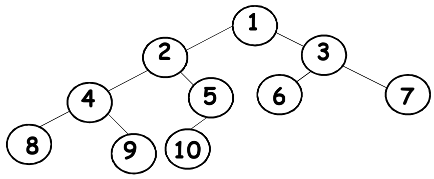
Exercício: mostrar a situação da fila na Busca em nível da árvore acima.

Construção de árvore cheia de altura h



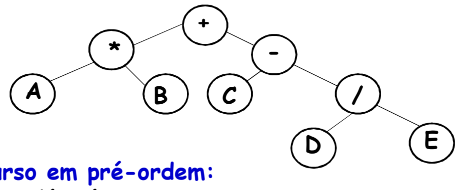

```
Constroi(h):
    #Dados: inteiro h
    Esvazia(Q);  Alocar(T);    Enfila(T);     n ← 2^(h-1) - 1
    T.c ← 1;   T.le ← nulo;    T.ld ← nulo;  i ← 1;

    enquanto (f ≠ 0):
          p ← Q[f]
          q ← Arv();   q.c ← i+1;   q.le ← nulo;   q.ld ← nulo;
          s ← Arv();   s.c ← i+2;    s.le ← nulo;   s.ld ← nulo;
          p.le ← q;     p.ld ← s;     i ← i+2;
          se (i ≤ n):
                Enfila (q)
                Enfila (s)
          p ← Desenfila ()
```


Exercício



Exercício: modificar o algoritmo de criação de árvore binária cheia, para a criação de uma **árvore binária completa**, para dado **n**.

Percursos

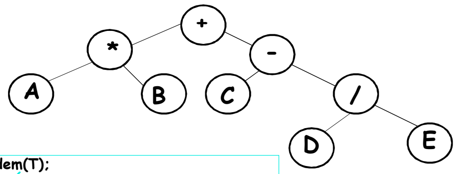


Percurso em pré-ordem:

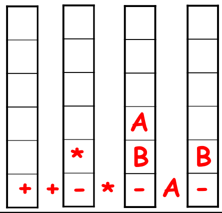
a idéia é visitar, sistematicamente, a raiz, em seguida toda a subárvore esquerda e depois toda a subárvore direita (RED).

+ * A B - C / D E

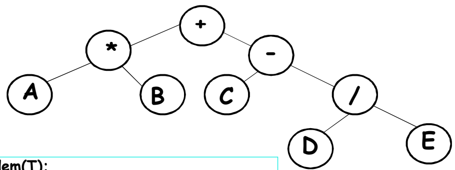
Percurso em pré-ordem




```
Pré-Ordem(T):
#Dados: Árvore binária T

        Esvazia(P);  PUSH (T);

        enquanto (topo ≠ 0):
                p ← POP ()
                Visita (p)
                PUSH (p.ld)
                PUSH (p.le)
```

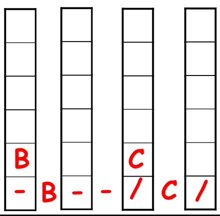


Percurso em pré-ordem

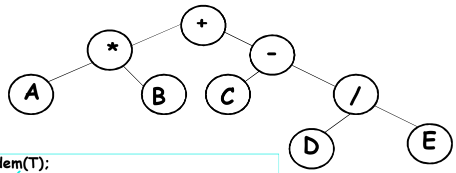



```
Pré-Ordem(T):
#Dados: Árvore binária T

        Esvazia(P);  PUSH (T);

        enquanto (topo ≠ 0):
                p ← POP ()
                Visita (p)
                PUSH (p.ld)
                PUSH (p.le)
```

+ * A

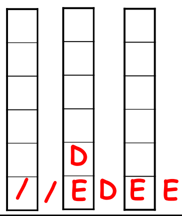


Percurso em pré-ordem

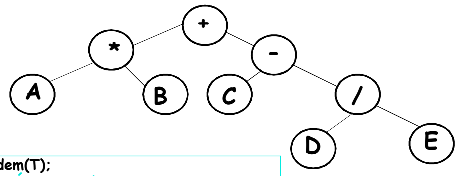



```
Pré-Ordem(T):
#Dados: Árvore binária T

        Esvazia(P);  PUSH (T);

        enquanto (topo ≠ 0):
                p ← POP ()
                Visita (p)
                PUSH (p.ld)
                PUSH (p.le)
```

+ * A B - C



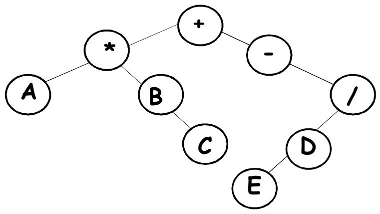
Percursos



```
Pré-Ordem(T);
#Dados: Árvore binária T

      Esvazia(P);  PUSH (T);

      enquanto (topo ≠ 0):
            p ← POP ()
            Visita (p)
            PUSH (p.ld)
            PUSH (p.le)
```

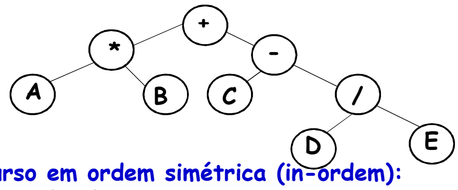
+ * A B - C / D E

Exercício



Exercício: mostrar a situação da pilha na Busca em pré-ordem da árvore acima.

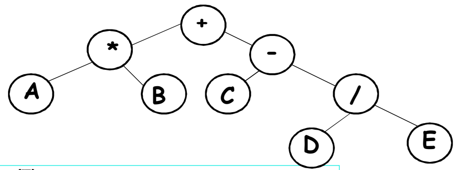
Percursos



Percurso em ordem simétrica (in-ordem):
a idéia é visitar, sistematicamente, toda a subárvore esquerda, a raiz, e em seguida toda a subárvore direita (ERD).

A * B + C - D / E

Percursos em ordem simétrica

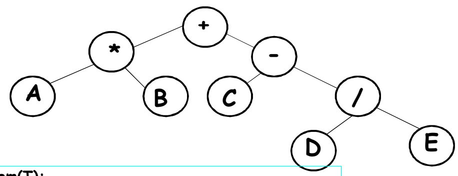


```
In-Ordem(T);
#Dados: Árvore binária T
      Esvazia(S);   p ← T;

      enquanto ((p ≠ nulo) ou (topo ≠ 0)):

            enquanto (p ≠ nulo):
                  PUSH (p)
                  p ← p.le

            p ← POP (); Visita (p);   p ← p.ld;
```

A

Pilha: A, *, +

Percurso em ordem simétrica

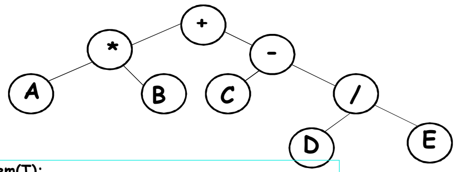


```
In-Ordem(T);
#Dados: Árvore binária T
      Esvazia(S);   p ← T;

      enquanto ((p ≠ nulo) ou (topo ≠ 0)):

            enquanto (p ≠ nulo):
                  PUSH (p)
                  p ← p.le

            p ← POP (); Visita (p);   p ← p.ld;
```

A * B

Pilha: B, +

Percurso em ordem simétrica

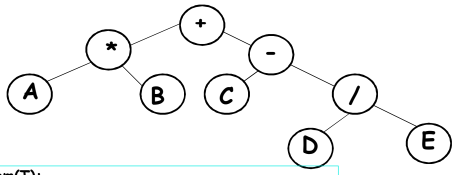


```
In-Ordem(T);
#Dados: Árvore binária T
      Esvazia(S);   p ← T;

      enquanto ((p ≠ nulo) ou (topo ≠ 0)):

            enquanto (p ≠ nulo):
                  PUSH (p)
                  p ← p.le

            p ← POP (); Visita (p);   p ← p.ld;
```

A * B + C

Pilha: C, -

Percurso em ordem simétrica

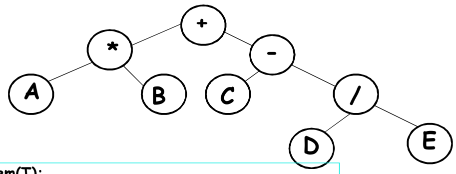



```
In-Ordem(T);
#Dados: Árvore binária T
        Esvazia(S);  p ← T;

        enquanto ((p ≠ nulo) ou (topo ≠ 0)):

                enquanto (p ≠ nulo):
                        PUSH (p)
                        p ← p.le

                p ← POP (); Visita (p);  p ← p.ld;
```

A * B + C - D

D
/

Percurso em ordem simétrica

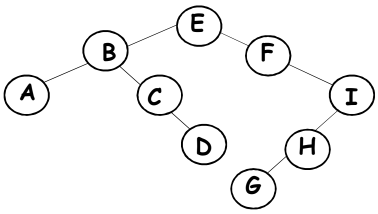



```
In-Ordem(T);
#Dados: Árvore binária T
        Esvazia(S);  p ← T;

        enquanto ((p ≠ nulo) ou (topo ≠ 0)):

                enquanto (p ≠ nulo):
                        PUSH (p)
                        p ← p.le

                p ← POP (); Visita (p);  p ← p.ld;
```

A * B + C - D / E

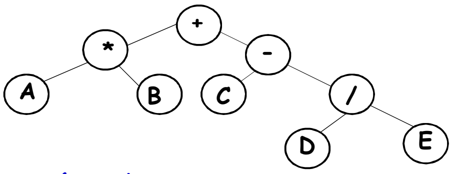
E

Exercício



Exercício: mostrar a situação da pilha na Busca in-ordem da árvore acima.

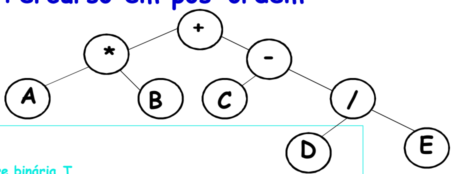
Percursos



Percurso em pós-ordem:
a idéia é visitar, sistematicamente, toda a subárvore esquerda, em seguida toda a subárvore direita e, por fim, a raiz (EDR).

A B * C D E / - +

Percurso em pós-ordem

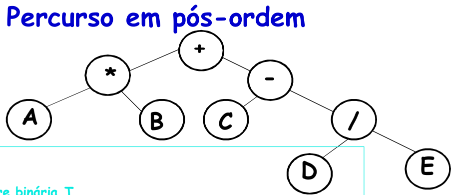



```
Pos-Ordem(T);
#Dados: Árvore binária T
        Esvazia(S);  p ← T;

        enquanto ((p ≠ Nulo) ou (topo ≠ 0)):
                enquanto (p ≠ nulo):
                        PUSH (p, 1);    p ← p.le;

                enquanto ((p = nulo) e (topo ≠ 0)):
                        (p, vez) ← POP();
                        se ((vez = 1) e (p.ld ≠ nulo)):
                           PUSH(p, 2);   p ← p.ld;
                        senão:
                           Visita (p);     p ← nulo;
```

A

A,1
*,1
+,1

Percurso em pós-ordem

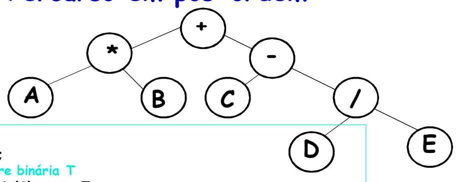



```
Pos-Ordem(T);
#Dados: Árvore binária T
        Esvazia(S);  p ← T;

        enquanto ((p ≠ Nulo) ou (topo ≠ 0)):
                enquanto (p ≠ nulo):
                        PUSH (p, 1);    p ← p.le;

                enquanto ((p = nulo) e (topo ≠ 0)):
                        (p, vez) ← POP();
                        se ((vez = 1) e (p.ld ≠ nulo)):
                           PUSH(p, 2);   p ← p.ld;
                        senão:
                           Visita (p);     p ← nulo;
```

A B,1

B,1
*,2
+,1

Percurso em pós-ordem

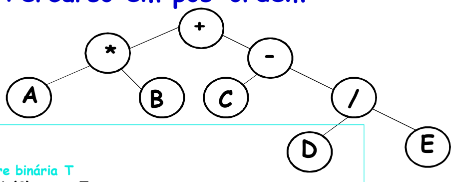



```
Pos-Ordem(T);
#Dados: Árvore binária T
    Esvazia(S);  p ← T;

    enquanto ((p ≠ Nulo) ou (topo ≠ 0)):
        enquanto (p ≠ nulo):
            PUSH (p, 1);   p ← p.le;

        enquanto ((p = nulo) e (topo ≠ 0)):
            (p, vez) ← POP();
            se ((vez = 1) e (p.ld ≠ nulo)):
                PUSH(p, 2);  p ← p.ld;
            senão:
                Visita (p);     p ← nulo;
```

A B *

+,1

Percurso em pós-ordem

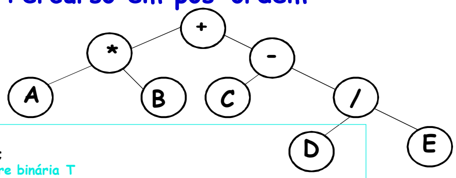



```
Pos-Ordem(T);
#Dados: Árvore binária T
    Esvazia(S);  p ← T;

    enquanto ((p ≠ Nulo) ou (topo ≠ 0)):
        enquanto (p ≠ nulo):
            PUSH (p, 1);   p ← p.le;

        enquanto ((p = nulo) e (topo ≠ 0)):
            (p, vez) ← POP();
            se ((vez = 1) e (p.ld ≠ nulo)):
                PUSH(p, 2);  p ← p.ld;
            senão:
                Visita (p);     p ← nulo;
```

A B * C,1

C,1
-,1
+,2

Percurso em pós-ordem

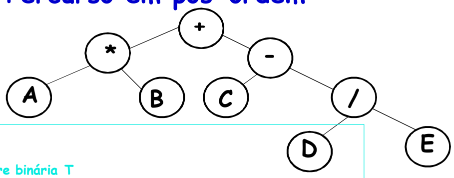



```
Pos-Ordem(T);
#Dados: Árvore binária T
    Esvazia(S);  p ← T;

    enquanto ((p ≠ Nulo) ou (topo ≠ 0)):
        enquanto (p ≠ nulo):
            PUSH (p, 1);   p ← p.le;

        enquanto ((p = nulo) e (topo ≠ 0)):
            (p, vez) ← POP();
            se ((vez = 1) e (p.ld ≠ nulo)):
                PUSH(p, 2);  p ← p.ld;
            senão:
                Visita (p);     p ← nulo;
```

A B * C D,1

D,1
/,1
-,2
+,2

Percurso em pós-ordem

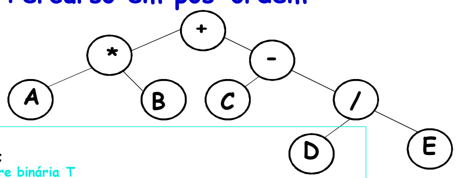



```
Pos-Ordem(T);
#Dados: Árvore binária T
    Esvazia(S);  p ← T;

    enquanto ((p ≠ Nulo) ou (topo ≠ 0)):
        enquanto (p ≠ nulo):
            PUSH (p, 1);   p ← p.le;

        enquanto ((p = nulo) e (topo ≠ 0)):
            (p, vez) ← POP();
            se ((vez = 1) e (p.ld ≠ nulo)):
                PUSH(p, 2);  p ← p.ld;
            senão:
                Visita (p);     p ← nulo;
```

A B * C D E,1

E,1
/,2
-,2
+,2

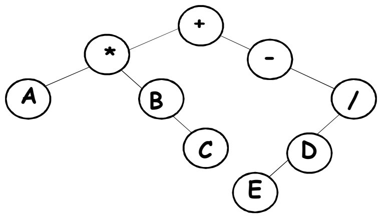
Percurso em pós-ordem




```
Pos-Ordem(T);
#Dados: Árvore binária T
    Esvazia(S);  p ← T;

    enquanto ((p ≠ Nulo) ou (topo ≠ 0)):
        enquanto (p ≠ nulo):
            PUSH (p, 1);   p ← p.le;

        enquanto ((p = nulo) e (topo ≠ 0)):
            (p, vez) ← POP();
            se ((vez = 1) e (p.ld ≠ nulo)):
                PUSH(p, 2);  p ← p.ld;
            senão:
                Visita (p);     p ← nulo;
```

A B * C D E / - +

Exercício



Exercício: mostrar a situação da pilha na Busca em pós-ordem da árvore acima.

Recursão

É uma técnica de solução de problemas (construção de algoritmos) que procura a solução subdividindo o problema em sub-problema(s) menor(es), de mesma natureza, e compõe a solução desses subproblemas, obtendo uma solução do problema original.

DIVIDIR PARA CONQUISTAR!!!

Recursão

```
Fatorial(p):
        se (p = 0):
                retornar 1
        senão:
                retornar p*Fatorial(p-1)
```

```
Fib(p):
        se (p ≤ 1 ):
                retornar p
        senão:
                retornar Fib(p-1)+Fib(p-2)
```


Recursão

Visões sobre Recursão:

a) Solução de problemas de trás para frente

enfatiza-se os passos finais da solução, após ter-se resolvido problemas menores. Mas a solução de problemas pequenos ("problemas infantís") tem que ser mostrada.

b) Analogia com a "Indução finita"

Indução Finita: prova-se resultados matemáticos gerais supondo-os válidos para valores inferiores a n e demonstrando que o resultado vale também para n. Além disso, mostra-se que o resultado é correto para casos particulares.

Recursão

Visões sobre Recursão:

c) Equivalente procedural de Recorrências

Recorrências são maneiras de formular funções para n, utilizando resultados da mesma função para valores menores que n. Além disso uma recorrência deve exibir resultados específicos para determinados valores.

d) Estrutura de um procedimento recursivo

Procedimentos recursivos "chamam a sí mesmos". Um procedcimento recursivo começa com um "Se", para separar subproblemas 'infantís" dos demais. O procedimento chama a si mesmo pelo menos uma vez. Sempre há uma chamada externa.

Recursão

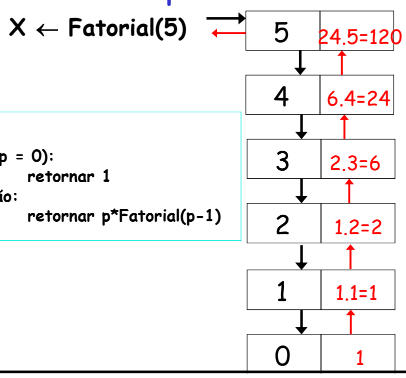
Dinâmica da execução de um procedimento recursivo.

Sempre que uma chamada recursiva é executada, o sistema operacional empilha as variáveis locais e a instrução em execução, desempilhando esses elementos no retorno da chamada.

Chamadas recursivas podem ser expressas através de uma árvore de recursão.

Recursão

Árvore de recursão para Fatorial




```
Fatorial(p):
        se (p = 0):
                retornar 1
        senão:
                retornar p*Fatorial(p-1)
```

Recursão

```
Fatorialrec(p):
        se (p = 0):
                retornar 1
        senão
                retornar p*Fatorial(p-1)
```

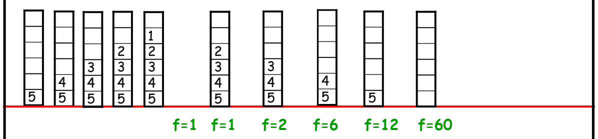
O algoritmo com uso de uma pilha dá uma idéia mais exata do que ocorre durante a execução do algoritmo recursivo.

```
Fatorialpil(p):
        Esvaziar(S);  i ← p;
        enquanto (i > 0):
             PUSH(i);  i ← i-1;
        f ← 1
        enquanto (topo > 0):
             f ← f*POP()
        retornar f
```


Recursão

O algoritmo com uso de uma pilha dá uma idéia mais exata do que ocorre durante a execução do algoritmo recursivo.

```
Fatorialpil(p):
        Esvaziar(S);  i ← p;
        enquanto (i > 0):
             PUSH(i);  i ← i-1;
        f ← 1
        enquanto (topo > 0):
             f ← f*POP()
        retornar f
```



Exercício

Impressão de lista Encadeada

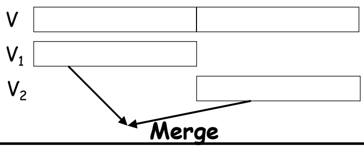
Exercício:
Escrever um algoritmo recursivo para imprimir uma lista encadeada em ordem reversa.

Recursão

MergeSort em Vetores:

Este é um importante método de ordenação, cuja idéia recursiva é a seguinte:

a) Dividir o vetor ao meio e ordenar separadamente cada um dos subvetores.
b) Fazer Merge dos subvetores ordenados.
c) O problema infantil é um subvetor de tamanho igual ou inferior a 1 quando, para ordená-lo, nda deve ser feito.



Recursão

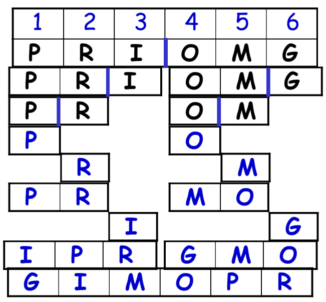
MergeSort em Vetores:

```
MergeSort (e, d):
        se (d > e):
                i ← ⌊(e+d)/2⌋
                MergeSort(e, i)
                MergeSort(i+1, d)
                Merge(e, i, d)
```

O algoritmo de Merge a ser usado usa dois vetores auxiliares que recebem metade do vetor inicial. O Merge é colocado no vetor principal.

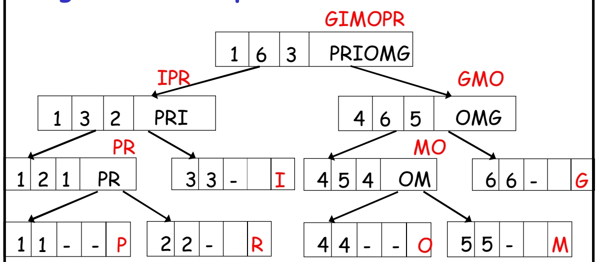
Recursão

MergeSort: exemplo



Recursão

MergeSort: exemplo da árvore de recursão



Exercício

MergeSort

Exercício:
Mostrar a árvore de recursão do Mergesort para o MIXSTRING (10 letras).

Análise do Mergesort:

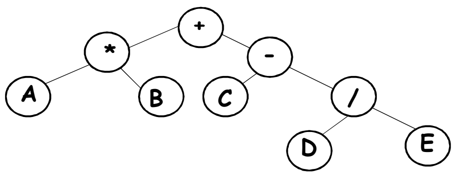
Complexidade:
Pior caso = melhor caso = caso médio: $O(n \log n)$

Estabilidade (manutenção da ordem relativa de chaves iguais):
Algoritmo estável

Memória adicional:
Vetor de tamanho n para merge intermediário

Usos especiais:
Ordenação de listas encadeadas
Minimização de I/O para arquivos grandes.

Percursos recursivos

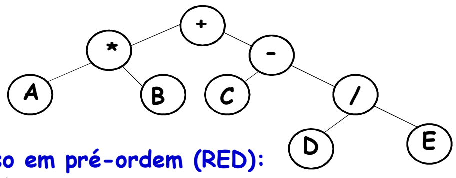


A maioria dos percursos em árvores binárias são intrinsecamente recursivos, pois cada percurso se constitui em visitar nós especiais e subárvores (problemas menores).

- -Percurso em pré-ordem
- -Percurso in-ordem (ordem simétrica)
- -Percurso em pós-ordem

O problema infantil é visitar uma árvore nula(nada é feito).

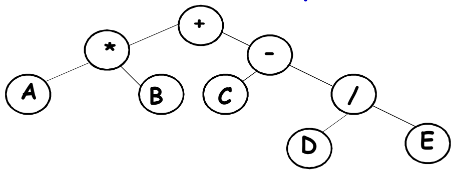
Percursos recursivos



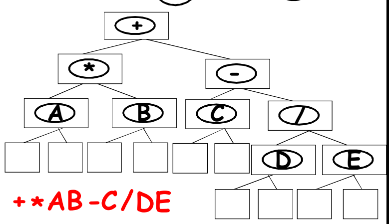
Percurso em pré-ordem (RED):
- -Visitar a raiz,
- -Visitar a subárvore esquerda em pré-ordem
- -Visitar a subárvore direita em pré-ordem.

+ * A B - C / D E

Percurso recursivo em pré-ordem



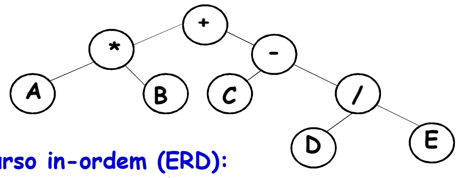
Pré-Ordem (p):
se ($p \neq$ nulo):
Visita (p)
Pré-Ordem (p.le)
Pré_Ordem (p.ld)



Externamente:
Pré-Ordem(T)

++AB-C/DE

Percursos recursivos

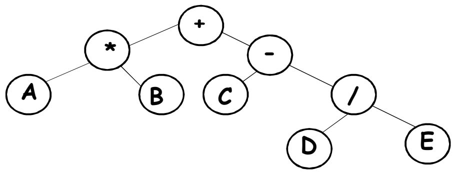


Percurso in-ordem (ERD):

- Visitar a subárvore esquerda in-ordem,
- Visitar a raiz,
- Visitar a subárvore direita in-ordem.

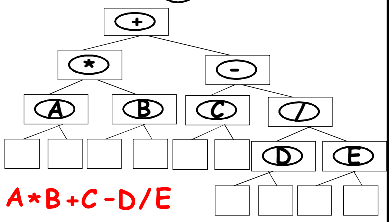
A * B + C - D / E

Percurso recursivo em ordem simétrica

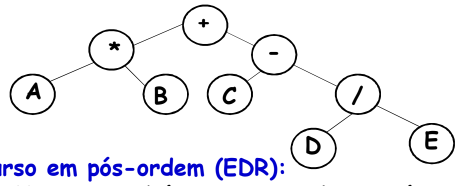



```
In-Ordem (p):
   se (p ≠ nulo):
      In-Ordem (p.le)
      Visita (p)
      In-Ordem (↑.ld)
```

Externamente:
In-Ordem(T)



Percursos recursivos

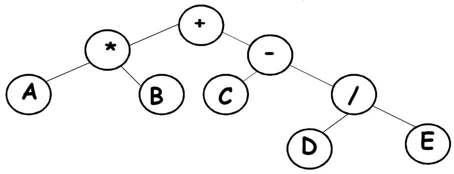


Percurso em pós-ordem (EDR):

- Visitar a subárvore esquerda em pós-ordem,
- Visitar a subárvore direita em pós-ordem,
- Visitar a raiz.

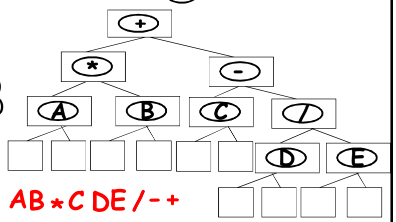
A B * C D E / - +

Percurso recursivo em pós-ordem



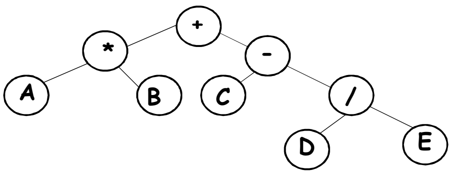

```
Pós-Ordem (p):
   se (p ≠ nulo):
      Pós-Ordem (p.le)
      Pós-Ordem (p.ld)
      Visita (p)
```

Externamente:
Pós-Ordem(T)



Exercício

Percursos recursivos

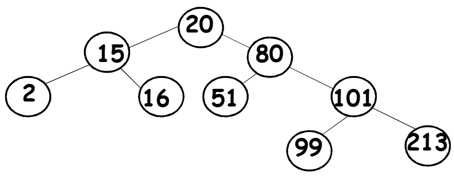


Exercício: Dada uma expressão aritmética com operadores binários, representada em uma árvore binária, escrever a expressão equivalente totalmente parentisada.

Árvores Binárias de Busca

São árvores constuídas com a regra forçada de que, para cada nó da árvore, todas as chaves da subárvore esquerda são menores que a chave desse nó e todas as da subárvore direita, maiores.

Exemplo:



Exercício

Exercício:

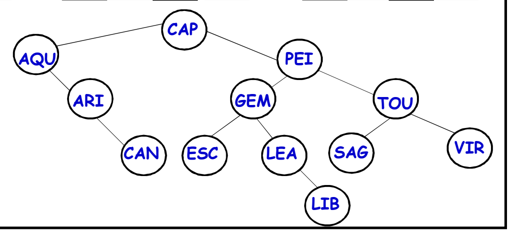
Desenhar todas as ABBs distintas para as chaves: A, B, C e D, com A na raiz.

Árvores Binárias de Busca

Inserções: sempre é possível inserir uma chave não existente, através de um processo simples, top-down.

Exemplo: Árvore dos Signos

CAP,	AQU,	PEI,	ARI,	TOU,	GEM,	CAN,	LEA,	VIR,	LIB,	ESC,	SAG



Exercício

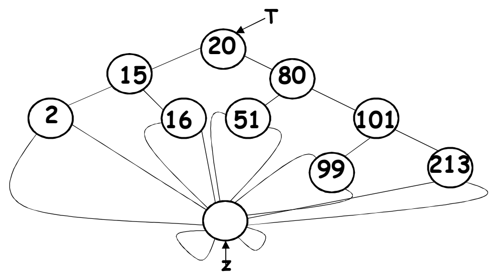
Desenhar ABBs para as chaves incluídas nas seguintes ordens:

a) AQU, ARI, CAN, CAP, ESC, VIR, TOU, SAG, GEM, PEI, LEA, LIB

b) AQU, VIR, ARI, TOU, CAN, SAG, CAP, PEI, LIB, ESC, GEM, LEA

Árvores Binárias de Busca

Convenção de nó terminal: pode-se usar um nó pré-alocado, z, como nó final, em substituição às subárvores nulas.



Busca em uma ABB (com nó terminal).

```
Busca(k);
#Dados: Árvore binária T, valor k

        z.chave ← k;  p ← T;

        enquanto (p.chave ≠ k):
                se (k < p.chave):
                        p ← p.le
                senão:
                        p ← p.ld

        se (p = z):
                p ← nulo

        retornar p
```


Inserção uma ABB (com nó terminal).

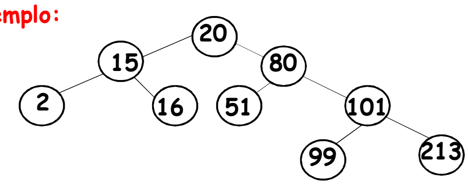
```
Inicializa;
        Alocar (z);   z↑.le ← z;   z↑.ld ← z;   T ← z;
Fim;
```

```
Insere(k):
#Dados: Árvore binária T, valor k
        z.chave ← k;   p ← T;   f ← T;

        enquanto (p.chave ≠ k):
                f ← p
                se (k < p.chave):
                        p ← p.le
                senão:
                        p ← p.ld

        se (p = z):
          p ← Arv();   p.chave ← k;   p.le ← z;   p.ld ← z;
          se (T = z):
                T ← p
          senão se (k < f.chave):
                f.le ← p
          senão:
                f.ld ← p
```

Propriedade fundamental de uma ABB:

O percurso in-ordem obtem as chaves ordenadas.

Exemplo:



In-ordem: 2 15 16 20 51 80 99 101 213

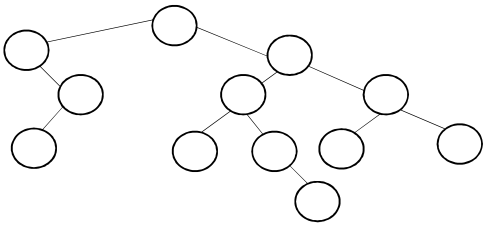
Essa propriedade conduz ao método de ordenação TreeSort!!!

Exercício

Criar uma ABB para o MIXSTRING (sem repetições) e checar a propriedade fundamental de uma ABB.

Exercício:

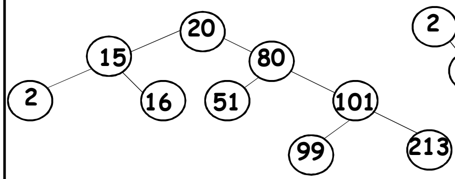
colocar as chaves A,B,C,D,E,F,G,H,I,J,K,L na estrutura abaixo, tal que tenhamos uma ABB.

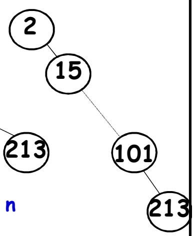


Altura de uma ABB:

Altura mínima: árvore completa.

$h = \lfloor \log_2 n \rfloor + 1 = \; = \lceil \log_2 (n+1) \rceil$





Altura máxima: árvore degenerada. $h = n$

Altura média (Knuth 73): $h \approx 1{,}4 \log_2 n$

Enumeração de ABBs.

T(n) = número de ABBs distintas para n chaves

T(0) = 1; T(1) = 1; T(2) = 2; T(3) = 5;

$T(n) = \sum Ti.T(n-1-i)$, $0 \le i < n$;

Idéia da recorrência: fixar progressivamente cada uma das n chaves como raiz e somar os produtos do número de ABBs distintas à esquerda com i chaves, pelo número de ABBS distintas à direita com n-1-i chaves.

T(n) = Comb(2n, n)/(n+1). (Número de Catalão!!)

Exercício:

Calcular T(6) = número de distintas ABBs c/ 6 chaves:

a) usando a recorrência

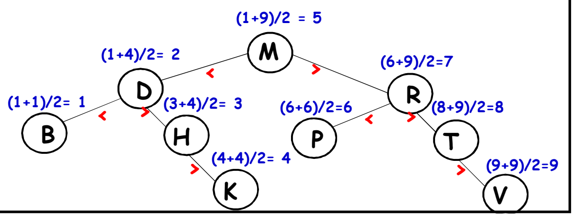
b) usando o número de Catalão.

Árvore de Decisão da Pesquisa Binária (ADPB)

A Pesquisa Binária pode ser melhor compreendida por uma ABB, que representa o processo de busca no vetor.

Exemplo:

1	2	3	4	5	6	7	8	9
B	D	H	K	M	P	R	T	V

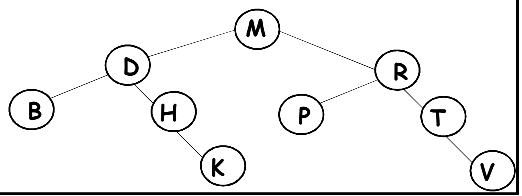


Árvore de Decisão da Pesquisa Binária (ADPB)

A Árvore de Decisão da Pesquisa Binária é uma árvore binária completa e pode ser construida recursivamente.

Exemplo:

1	2	3	4	5	6	7	8	9
B	D	H	K	M	P	R	T	V



Construção Recursiva da ADPB

```
Constroi(e, d, p):
        se (e > d):
                p ← nulo
        senão:
                i ← ⌊(e+d)/2⌋
                p ← Arv()
                p.chave ← V[i]
                Constroi(e, i-1, p.le)
                Constroi(i+1, d, p.ld)

Externamente:
        Constroi(1, n, T)
```


Exercício:

Desenhar a ADPB para n = 11.

Árvores Balanceadas

Para evitar a degeneração em ABB, foram criadas as **árvores balanceadas**. A idéia é a de, a cada inserção em uma ABB, checar se a árvore está tendendo à degeneração e acertar imediatamente, se necessário (**balancear** a árvore).

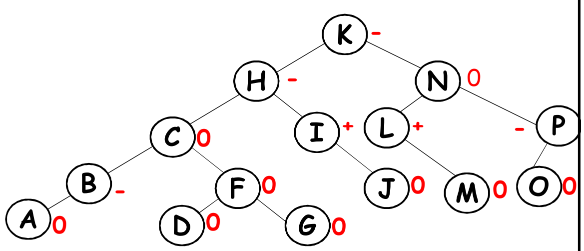
O balanceamento é baseado na altura.

Tipos de árvores balanceadas:

- AVL
- B/B+
- Rubronegras
- 2-3, etc

Árvores AVL (Adel'son-Vels'ii-Landis)

Árvores binárias onde, para **cada nó**, a **altura** das duas subárvores difere no máximo em **1**.



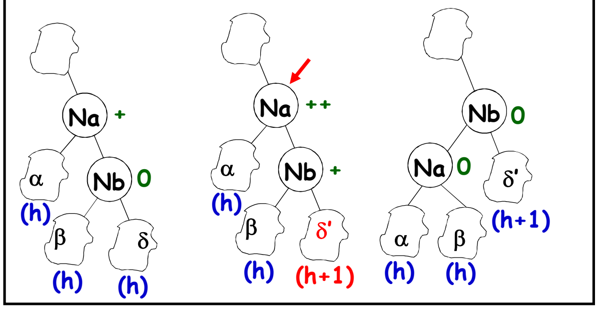
Exercício:

a) Desenhar uma árvore AVL de altura 6, com o menor número de nós possível.

b) Desenhar todas as AVLs para as chaves A, B, C, D.

Balanceamento de AVL's
Caso 1 - Rotação Simples

Sit. Inicial: Desbalanceamento Rotação Simples



Balanceamento de AVL's
Caso 1 - Rotação Simples -

a) Preservação da ordem de busca

Busca in-ordem antes: α.Na.β.Nb.δ'

depois: α.Na.β.Nb.δ'

b) Preservação da altura

Antes: max(h, max(h, h)+1)+1 = h+2

Depois: max(max(h, h) +1, h+1)+1 = h+2

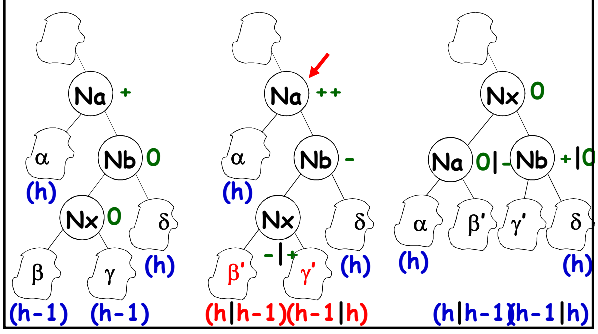
c) Balanceamento

Subárvores α, β e δ' balanceadas, por hipótese. Nós Na e Nb balanceados (esquema); restante da árvore balanceada, por b).

Árvores AVL

Balanceamento - Caso 2 - Rotação Dupla

Sit. Inicial: Desbalanceamento Rotação Dupla



Árvores AVL

Correção da Rotação Dupla

a) Preservação da ordem de busca

Busca in-ordem antes: α.Na.β'.Nx.γ'.Nb.δ

depois: α.Na.β'.Nx.γ'.Nb.δ

b) Preservação da altura

Antes: max(h, max(max(h-1, h-1)+1, h)+1)+1 = h+2

Depois: max(max(h, h| h-1)+1, max(h-1| h, h)+1)+1 = h+2

c) Balanceamento

Subárvores α, β', γ' e δ balanceadas, por hipó-tese. Nós Nx, Na e Nb balanceados (esquema); res-tante da árvore balanceada, por b).

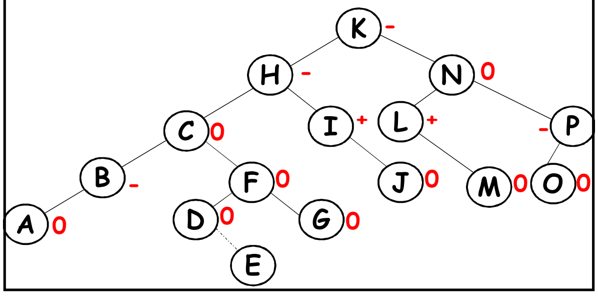
Árvores AVL

Exercício:

Criar e balancear, passo a passo, o MIXSTRING.

Árvores AVL

Exercício: balancear a árvore abaixo, após a inserção da chave E.



Árvores AVL

Resumo do algoritmo de Inserção/Balanceamento:

```
Busca_Insercao(k,  p,  h);
    se (p = nulo):
        Insere(k)
        h ← verdadeiro
    senão se (k < p.chave):
        Busca_Insercao(k, p.le, h);
        se (h):
            caso (p.bal):
                1:  p.bal ← 0;   h ← falso;
                0:  p.bal ← -1
               -1:  se (p.le.bal = -1):
                        RotSimplEsq(p)
                    senão:
                        RotDuplEsq (p)
                    p.bal ← 0;   h ← falso;
```

continua...

Árvores AVL

Resumo do algoritmo de Inserção/Balanceamento:

```
Busca_Insercao(k,  p,  h):
    ...
    senão se (k > p.chave):
        Busca_Insercao(k, p.ld, h)
        se (h):
            caso (p.bal):
               -1:  p.bal ← 0;   h ← falso;
                0:  p.bal ← 1
                1:  se (p.ld.bal = 1):
                        RotSimplDir(p)
                    senão
                        RotDuplDir (p)
                    p.bal ← 0;   h ← falso;
    senão:
        h ← falso
```

continua...

Árvores AVL

Resumo do algoritmo de Inserção/Balanceamento:

```
RotSimplEsq(p);
    pf ← p.le;
    p.le ← pf.ld;       pf.ld ← p;       p.bal ← 0;
    p ← pf;             p.bal ← 0;       h ← falso;

RotDuplEsq(p);
    pf ← p.le;          pn ← pf.ld;
    pf.ld ← pn.le;      pn.le ← pf;      p.le ← pn.ld;  pn.ld ← p;
    se (pn↑.bal = -1):
            p.bal ← 1
    senão:
            p.bal ← 0
    se (pn.bal = 1):
            pf.bal ← -1
    senão:
            pf.bal ← 0
    p ← pn;    p.bal ← 0;    h ← falso;
```


Árvores AVL

Árvores AVL Mínimas

Problema: determinar a **maior altura possível** para uma AVL com n chaves.

Solução: resolver o problema dual:
Determinar o número **mínimo de chaves** em uma AVL de altura h.

Árvores AVL

Árvores AVL Mínimas

Recorrência: T(h) =
núm. mínimo de chaves numa AVL de altura h

$T(0) = 0;\quad T(1) = 1;$
$T(h) = 1 + T(h-1) + T(h-2);\quad T(h) = Fib(h+2)-1;$

h	0	1	2	3	4	5	6	7	8	9
T(h)	0	1	2	4	7	12	20	33	54	88
Fib(h)	0	1	1	2	3	5	8	13	21	34

Árvores AVL

Árvores AVL Mínimas

Recorrência: T(h) =
núm. mínimo de chaves numa AVL de altura h

$T(0) = 0; \quad T(1) = 1;$
$T(h) = 1 + T(h-1) + T(h-2); \quad T(h) = Fib(h+2)-1;$

$\Rightarrow$ $h \leq 1{,}4 \log n$,
a altura máxima da AVL é 1,4 log n
a altura média da AVL é log n.

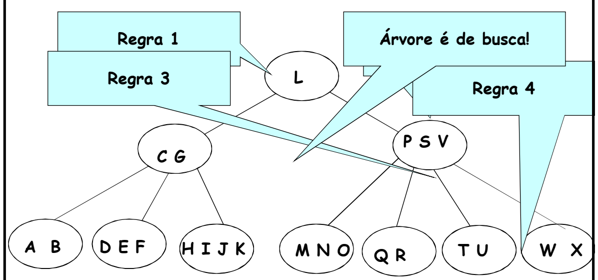
Árvores B (Bayer)

Árvores m-árias de busca (de ordem d) , para ar-mazenamento em disco, que obedecem a quatro regras básicas:

1) A raiz tem entre 1 e 2d chaves.
2) Nós intermediários têm um mínimo de d chaves e um máximo de 2d chaves.
3) Nó intermediário com k chaves tem k+1 filhos.
4) As folhas estão todas no mesmo nível.

Árvores B (Bayer)

Exemplo de árvore B de ordem 2:



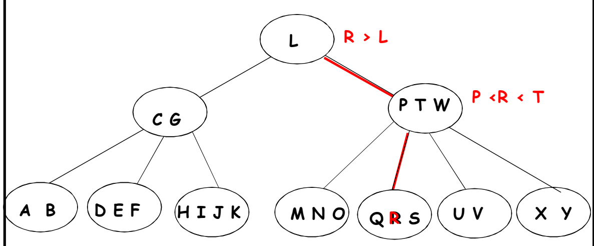
Árvores B (Bayer)

Exercício:

Desenhar uma árvore B de ordem 1, para as 26 letras do alfabeto, com altura 4.

Árvores B (Bayer)

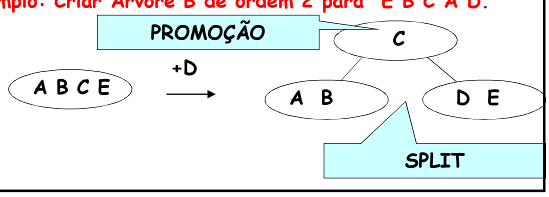
Busca na árvore: R



Árvores B (Bayer) - Inserções

A inserção é sempre feita nas folhas. Isso implica um processo de crescimento de baixo para cima e a necessidade de um método especial de inserção: Split e Promoção.

Exemplo: Criar Árvore B de ordem 2 para E B C A D.

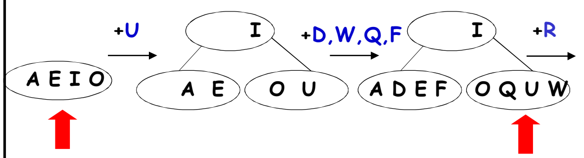


Arvores

Árvores B (Bayer) - Inserções (Regras)

1) a inserção é sempre feita numa folha. Se houver "overflow", faz-se Split e Promoção.

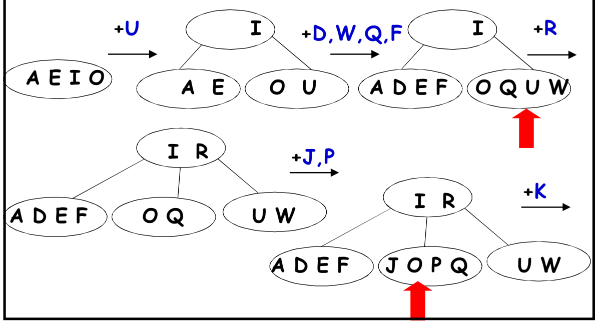
Ex: Criar árvore B de ordem 2 para:
A E I O U D W Q F R J P K T C B G H L M S V Z N Y X



Arvores

Árvores B (Bayer) - Inserções

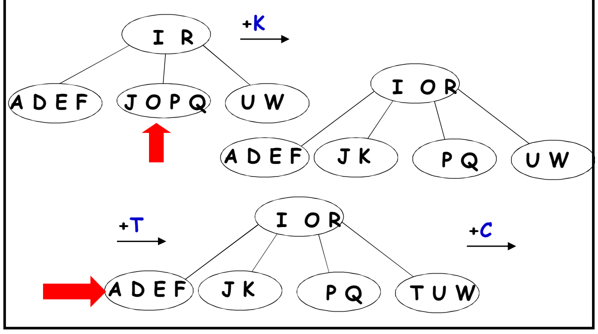
Ex: Criar árvore B de ordem 2 para:
A E I O U D W Q F R J P K T C B G H L M S V Z N Y X



Arvores

Árvores B (Bayer) - Inserções

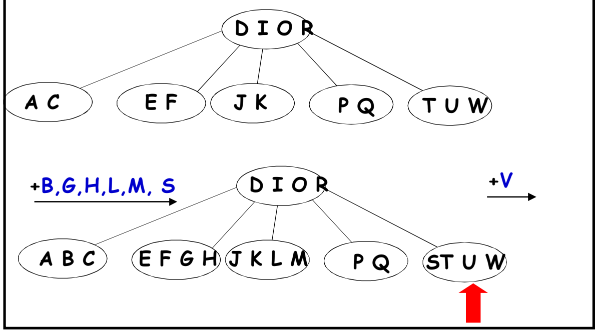
Ex: Criar árvore B de ordem 2 para:
A E I O U D W Q F R J P K T C B G H L M S V Z N Y X



Arvores

Árvores B (Bayer) - Inserções

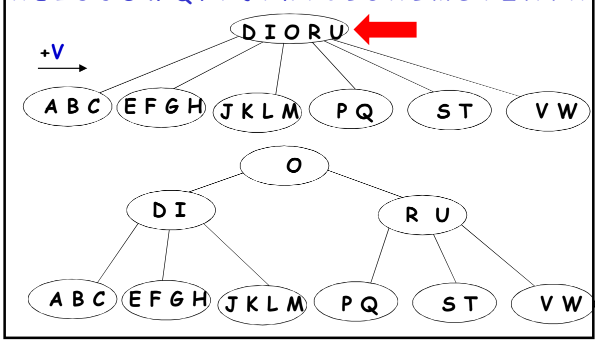
Ex: Criar árvore B de ordem 2 para:
A E I O U D W Q F R J P K T C B G H L M S V Z N Y X



Arvores

Árvores B (Bayer) - Inserções

Ex: Criar árvore B de ordem 2 para:
A E I O U D W Q F R J P K T C B G H L M S V Z N Y X

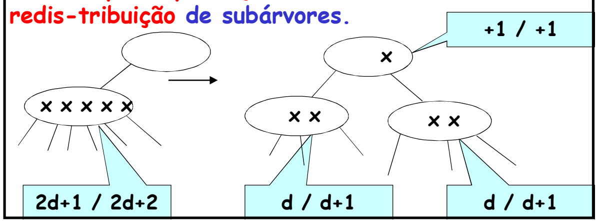


Arvores

Árvores B (Bayer) - Inserções

1) a inserção é sempre feita numa folha. Se houver "overflow", faz-se Split e Promoção.

2) Se houver "overflow" em nó intermediário, faz-se split e promoção no nível intermediário e redis-tribuição de subárvores.



Árvores B (Bayer) - Inserções

Exercício:

Criar, passo a passo, árvores B de ordens 1 e 2 para o MIXSTRING.

Árvores B (Bayer) - Inserções

Porque se usa árvore m-ária em disco?

R: Para otimizar acesso a disco. Quando se acessa disco para leitura ou gravação, sempre se trabalha com uma trilha inteira devido ao "seek" do disco.

A ordem da árvore é calculada a partir do tamanho da trilha.

Ex: tamanho da trilha = 10k
tamanho da chave + link = 50
→ d = 10000/2*50 = 100

Árvores B

Resumo do algoritmo de Inserção

```
Atualiza():
    T ← nulo;

    enquanto (houver chaves):
        Ler(x);
        BuscaInserção(x, T, h, u);
        se (h):
            q ← T;
            T ← Arv();
            T.m ← 1;   T.le ← q;  Vv[1] ← u;
```

continua...

Árvores B

Resumo do algoritmo de Inserção:

```
Busca_Insere(k, p, h, v);
    se (p = Nulo):
        v.c ← x;   v.ld ← nulo;
    senão:
        i ← PB(x, p)
        se (i ≠ nulo):
            h ← falso
        senão:
            se (i = 0):
                q ← p.le
            senão:
                q ← p.Vv[i].ld
            Busca_Insere(x, q, h, u)
            se (h):
                Insere();
```

continua...

Árvores B

Resumo do algoritmo de Inserção:

```
Insere():
    se (p↑.m < 2d):
        p.m++;   h ← falso;   Faz shift em Vv;  Vv[i] ← u;
    senão:
        b ← ArvB();
        v ← item a ser promovido;
        Transfere a metade direita de p para b;
        Acerta a metade esquerda de p e limpa sua metade direita;
        p.m ← d;  b.m ← d;  b.le ← v.ld;  v.ld ← b;
```


Árvores B (Bayer) - Inserções

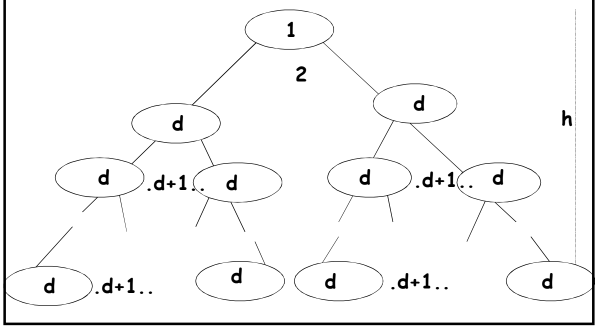
Árvores B Mínimas

Problema: determinar a maior altura possível para uma árvore B, ordem d, com n chaves.

Solução: resolver o problema dual:
Determinar o número mínimo de chaves em uma árvore B de ordem d e altura h.

Árvores B (Bayer) - Inserções

Árvores B Mínimas - Estrutura da árvore



Árvores B (Bayer) - Inserções

Árvores B Mínimas

Solução: resolver o problema dual: determinar o número **mínimo de chaves** em uma árvore B de ordem d e altura h.

$$n = 1 + 2(d + d(d+1) + \dots + d(d+1)^{(h-2)}) =$$
$$= 1 + 2d((d+1)^{(h-1)} - 1)/(d+1-1) =$$
$$2(d+1)^{(h-1)} - 1$$
$$\Rightarrow \quad h-1 = \log_{d+1} (n+1)/2 \quad \text{ou}$$

$$h \approx \log_{d+1} (n+1)/2$$

Árvores B (Bayer) - Inserções

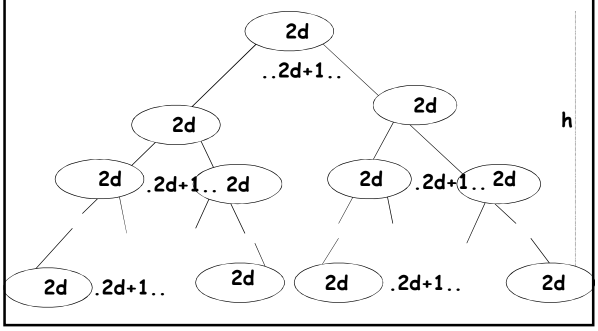
Árvores B Máximas

Problema: determinar a **menor altura possível** para uma árvore B, ordem d, com n chaves.

Solução: resolver o problema dual:
Determinar o número **máximo de chaves** em uma árvore B de ordem d e altura h.

Árvores B (Bayer) - Inserções

Árvores B Máximas - Estrutura da árvore



Árvores B (Bayer) - Inserções

Árvores B Máximas

Solução: resolver o problema dual: determinar o número **máximo de chaves** em uma árvore B de ordem d e altura h.

$$n = 2d + 2d(2d+1) + \dots + 2d(2d+1)^{(h-1)}) =$$
$$= 2d((2d+1)^h - 1)/(2d+1-1) = (2d+1)^h - 1$$
$$\Rightarrow$$
$$h = \log_{2d+1} (n+1)$$

Árvores B (Bayer) - Inserções

Árvores B - Alturas máxima e mínima

ÁRVORE MÍNIMA			
n\d	10	50	60
10^2	2	2	1
10^3	4	3	2
10^6	6	4	3

ÁRV. MÁXIMA		
10	50	60
2	2	1
4	3	2
6	4	3

mínima ~ máxima

valores baixos !!!

Árvores B (Bayer) - Inserções

Regras para Deleções

1. Se a chave está na folha e não há "underflow", a chave é deletada na folha.

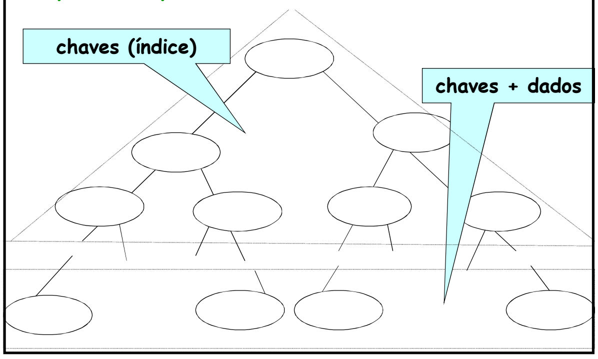
2. Se a chave está na folha e há "underflow", faz-se "fusão" ou "recombinação".

3. Se a chave está em nó intermediário, substitui a mesma pela sucessora(na folha) e elimina esta.

4. Se ocorrer "underflow" em nó intermediário, faz-se "fusão" ou "recombinação" no nível.

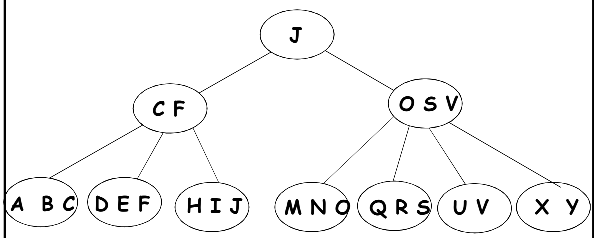
Árvores B+

Arquivos Sequenciais Indexados



Árvores B+

Árvores B+ de ordem 2

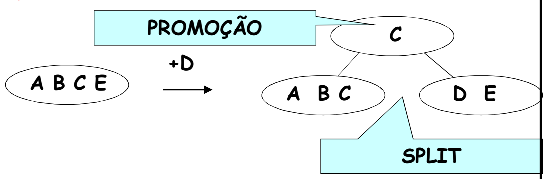


Árvores B+

Árvores B+ - Inserções

Semelhante à inserção na árvore B. A diferença é que, nos splits de folha, a chave promovida tam-bém permanece na folha.

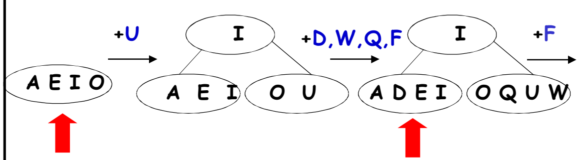
Exemplo: Criar Árvore B+ de ordem 2 para E B C A D.



Árvores B+

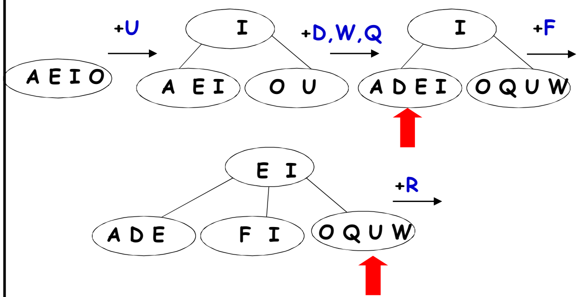
Árvores B+

Ex: Criar árvore B de ordem 2 para:
A E I O U D W Q F R J K T C B



Árvores B+

Ex: Criar árvore B+ de ordem 2 para:
A E I O U D W Q F R J K T C B

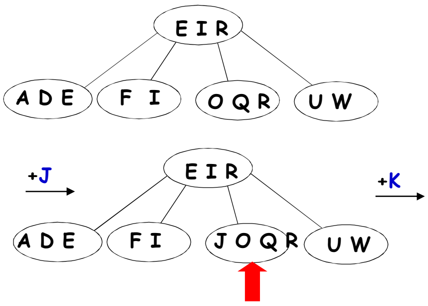


Árvores

Árvores B+

Ex: Criar árvore B+ de ordem 2 para:

A E I O U D W Q F R J K T C B

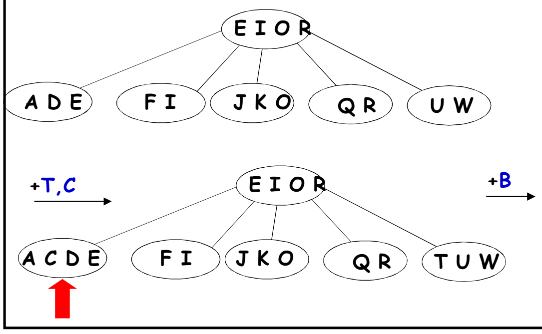


Árvores

Árvores B+

Ex: Criar árvore B+ de ordem 2 para:

A E I O U D W Q F R J K T C B

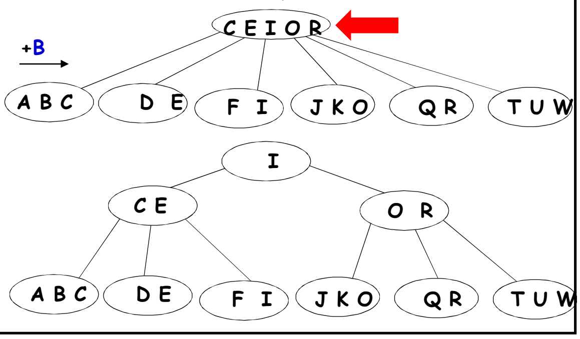


Árvores

Árvores B+

Ex: Criar árvore B de ordem 2 para:

A E I O U D W Q F R J K T C B

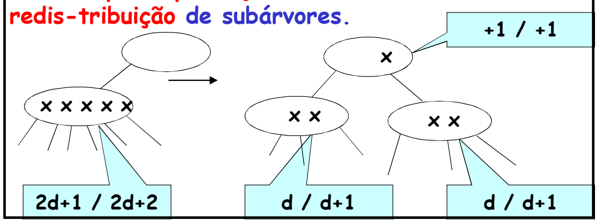


Árvores

Árvores B+ - Inserções

1) a inserção é sempre feita numa folha. Se houver "overflow", faz-se Split e Promoção, mantendo a chave promovida na folha.

2) Se houver "overflow" em nó intermediário, faz-se split e promoção no nível intermediário e redis-tribuição de subárvores.



Árvores

Árvores B+ - Inserções

Exercício:

Criar, passo a passo, árvores B+ de ordens 1 e 2 para o MIXSTRING.

Árvores

FIM