Árvores 2-3

O problema:

- Como implementar uma tabela de símbolos em uma BST de modo que a árvore permaneça balanceada qualquer que seja a sequência de buscas e inserções?

- O que é uma árvore balanceada?
  - refere-se a que? A altura da árvore

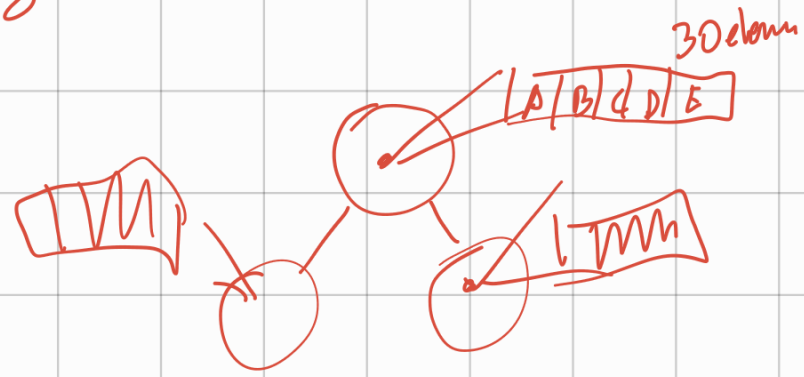
-

Destaques das árvores 2-3

- nós simples, duplos
- nós triplos temporários
- Operação de busca
- Operação de inserção } $\sim \lg N$
- balanceamento
- altura nunca passa de $c \lg N$

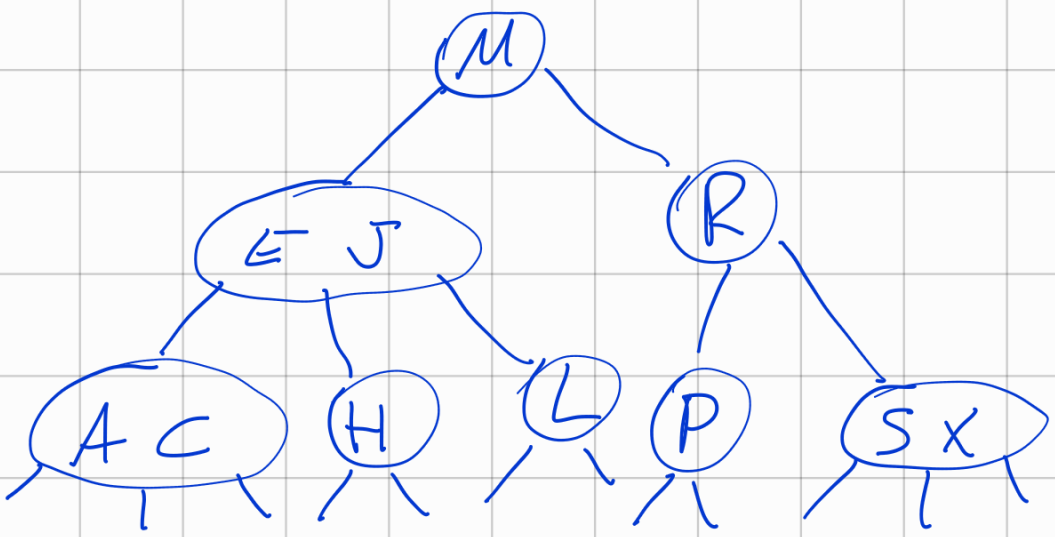
- Por que a altura cresce? Cresce com a inserção de novas chaves na árvore

- Solução: Cada nó armazene mais de 1 chave



- Difícil:

- O que vira a árvore?



- A árvore 2-3 de busca é:

  - Uma árvore vazia
  - ou um nó simples, que contém uma chave e dois links
    - Com a propriedade:

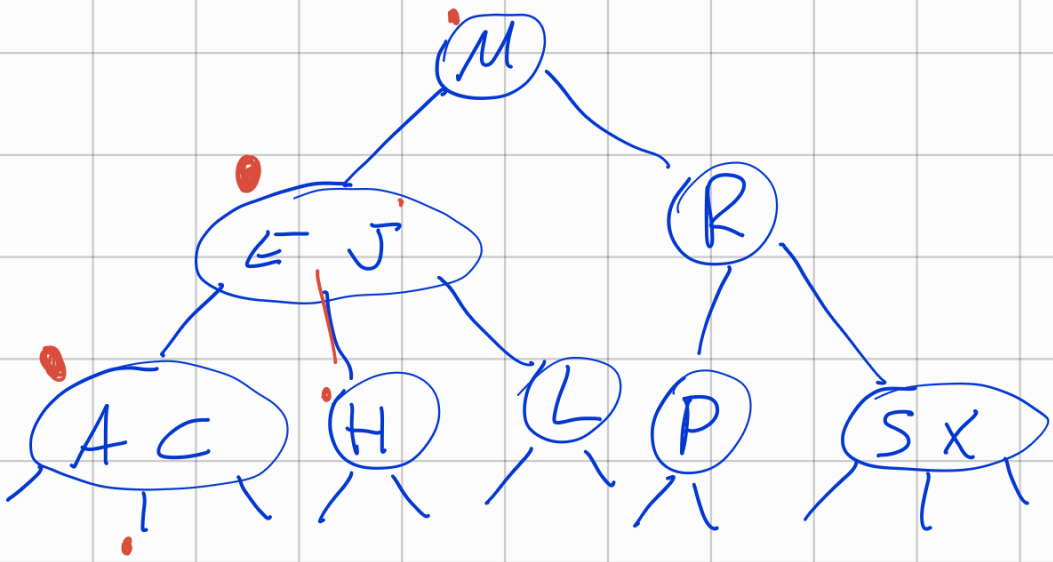
  - ou um nó duplo, que contém duas chaves e 3 links
    - Com a propriedade:

  - Toda árvore 2-3 é perfeitamente balanceada!
    Todos os links "NULL" (z) estão no mesmo nível

- Busca na árvore 2-3

H

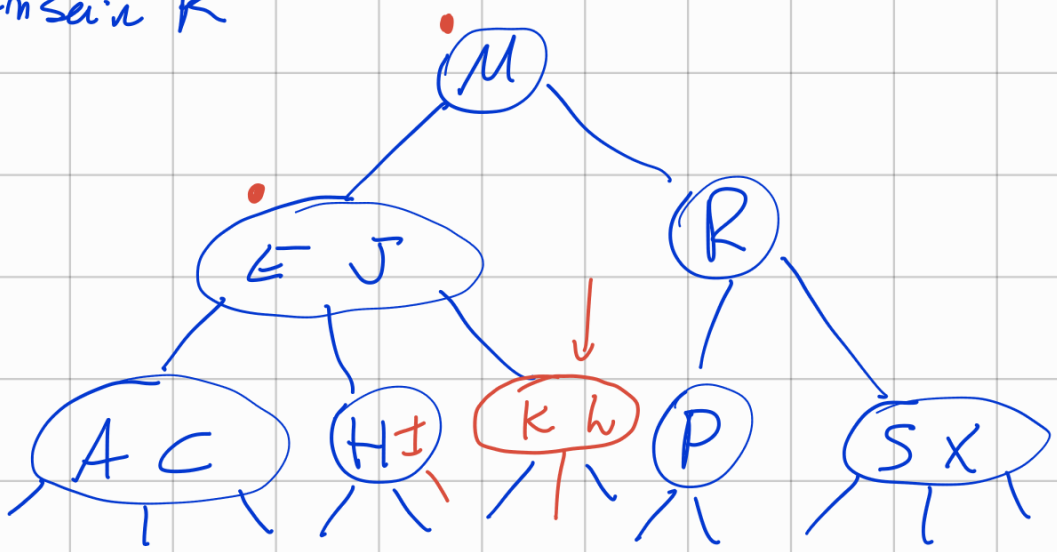
B



Como fica a busca pela chave "H"? e a "B"?

Inserção em árvores 2-3

→ Inserir "K"

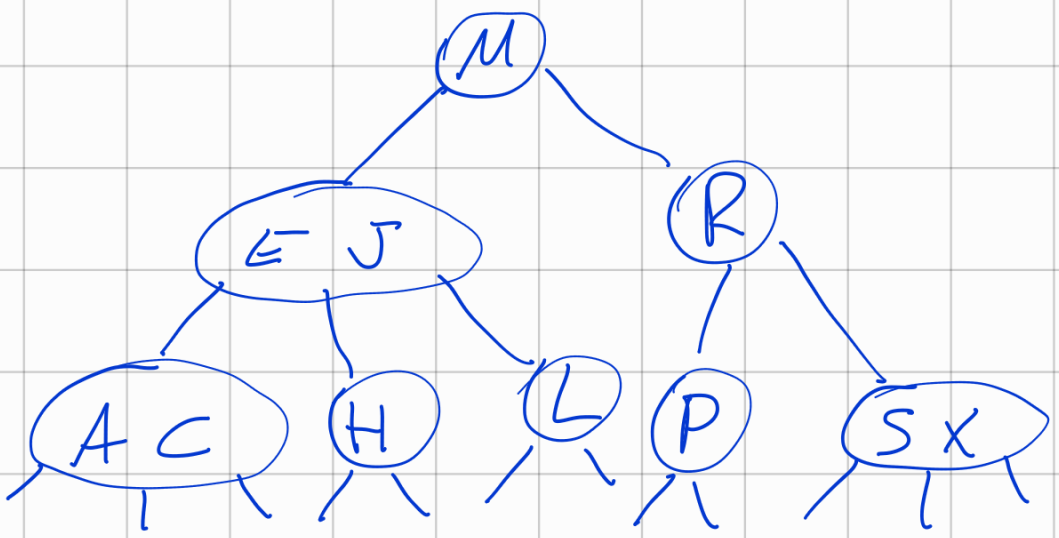


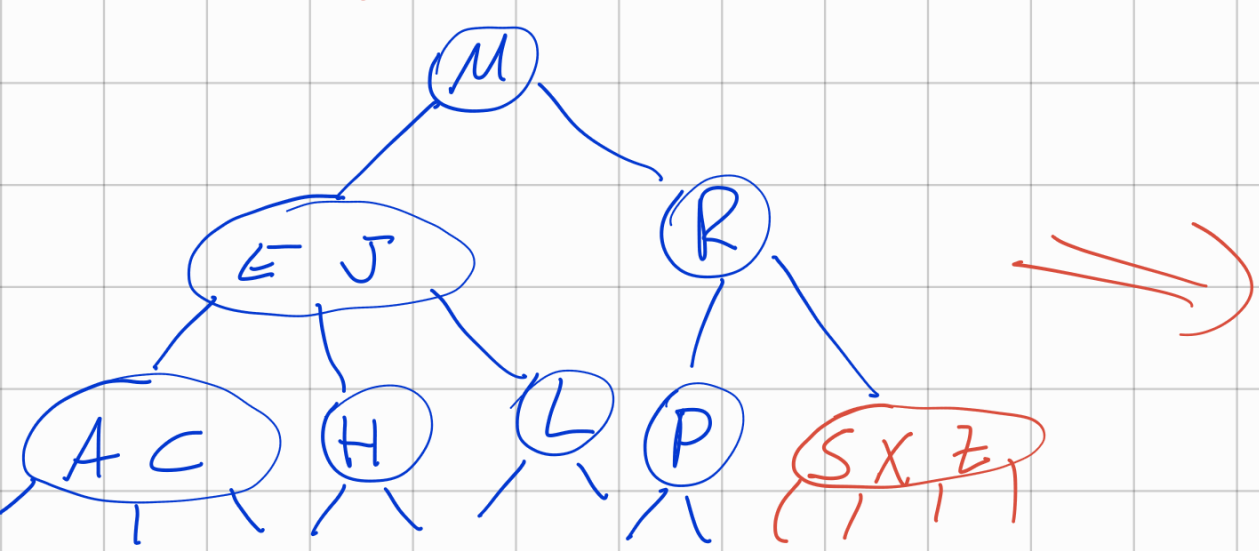
- tipo de inserção:
- Como fica o balanceamento?

Inserção em um nó duplo

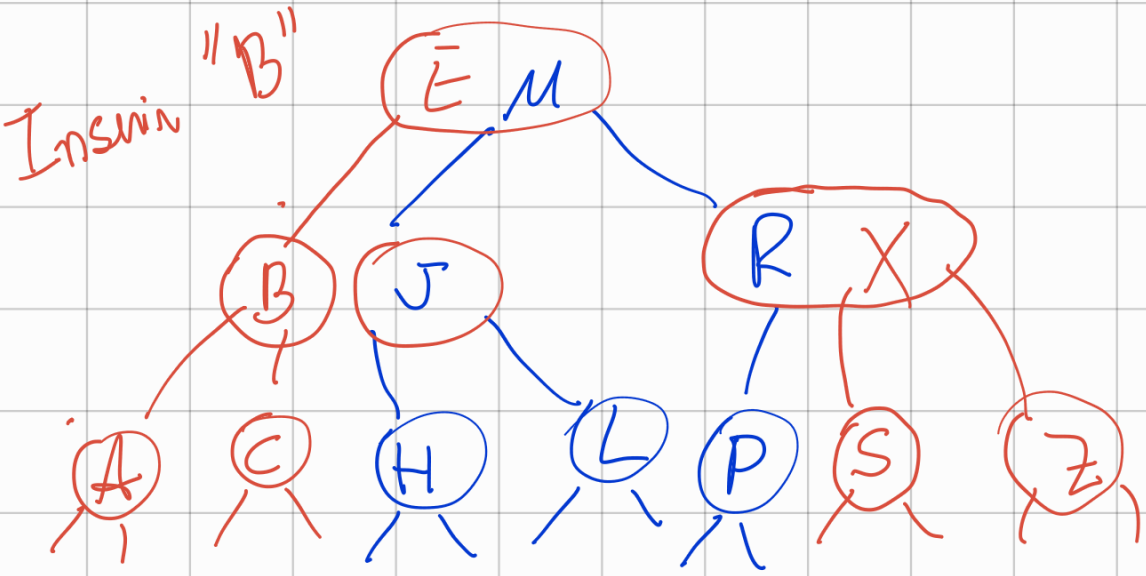
- Regra

→ Inserir "Z"



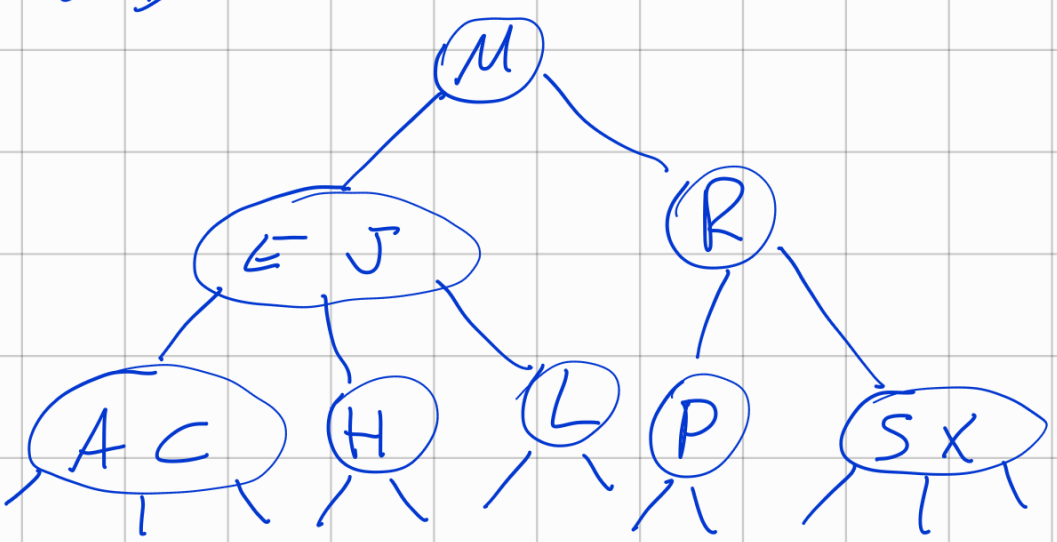


Inserir "B"



- E quando o pai é duplo?

→ Insiro "D"



É a divisão na raiz?

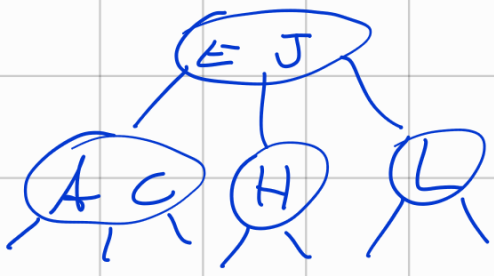
- Como fica o balenceamento?

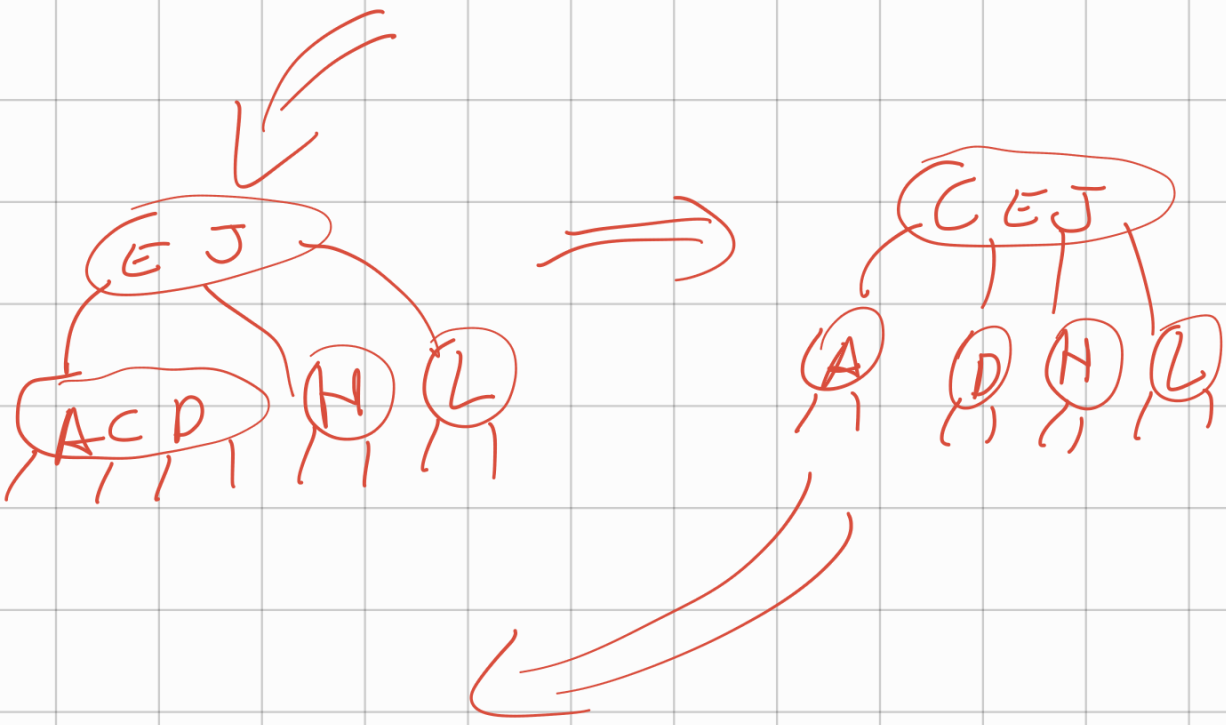
- e a altura?

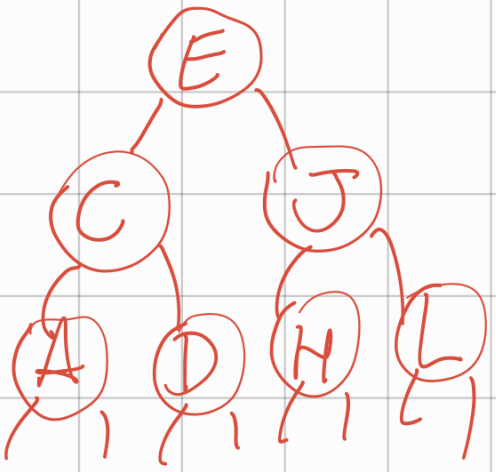
aumentou!

OK

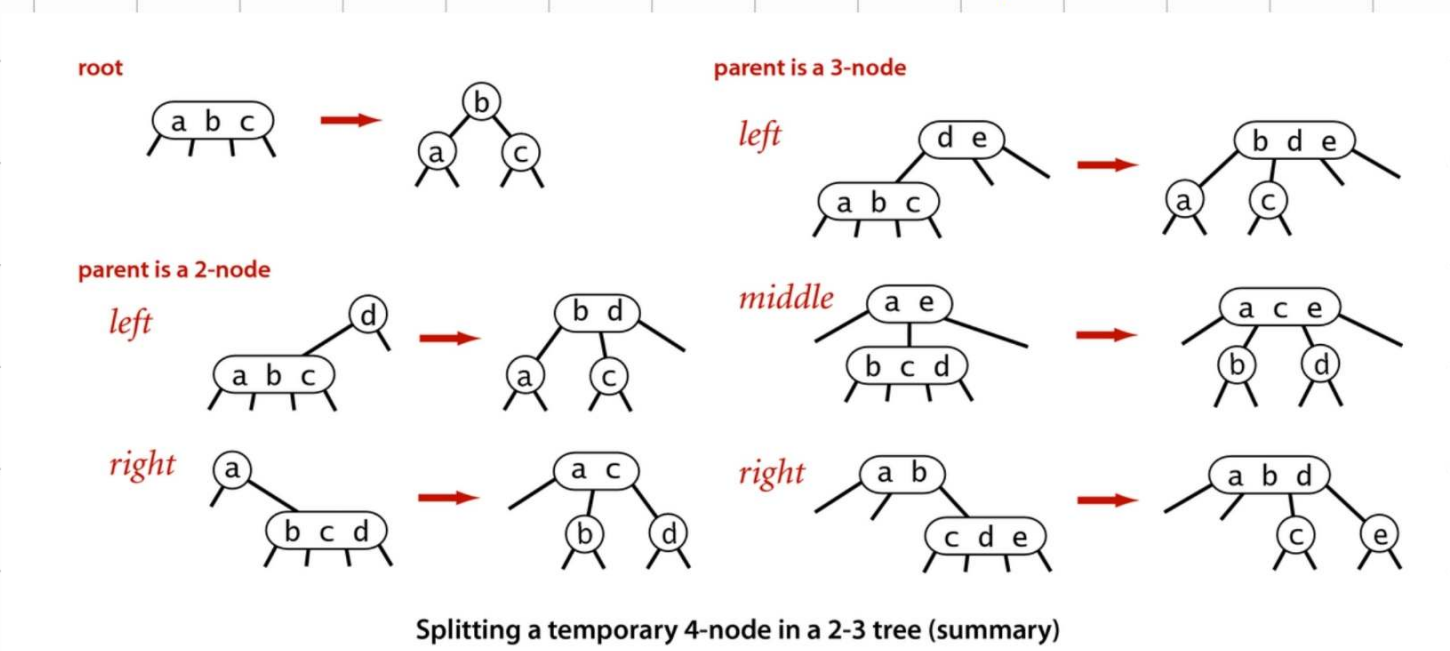
→ Inserir "D"





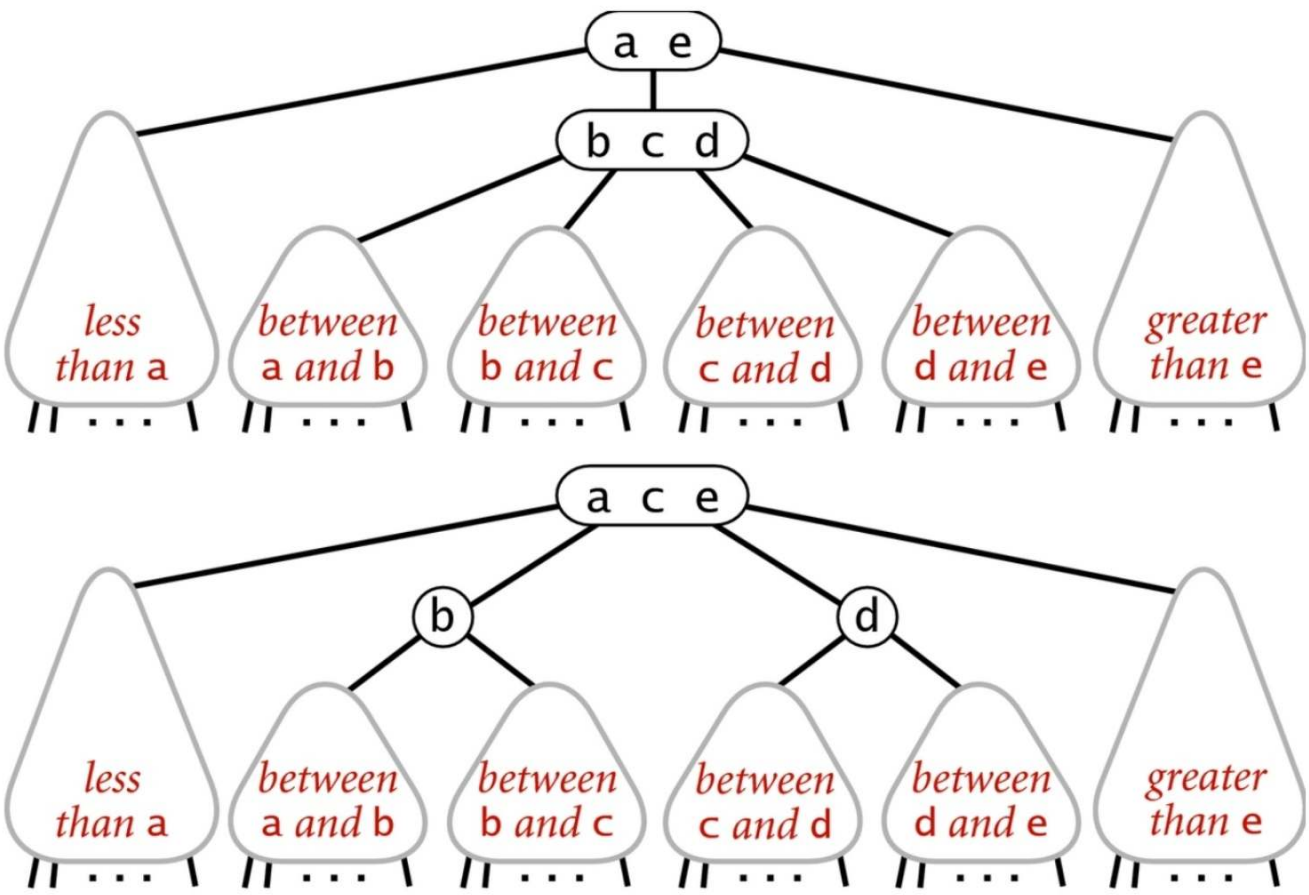


- Divisão de um nó ternário temporário envolve 6 transformações. As transformações são locais e portanto consomem tempo constante



Splitting a temporary 4-node in a 2-3 tree (summary)

- As transformações preservam as propriedades globais da árvore (a árvore continua em ordem e perfeitamente balanceada):



Splitting a 4-node is a local transformation that preserves order and perfect balance

- Construir uma árvore 2-3 de busca:

a) Inserir na ordem S E A R C H X M P L

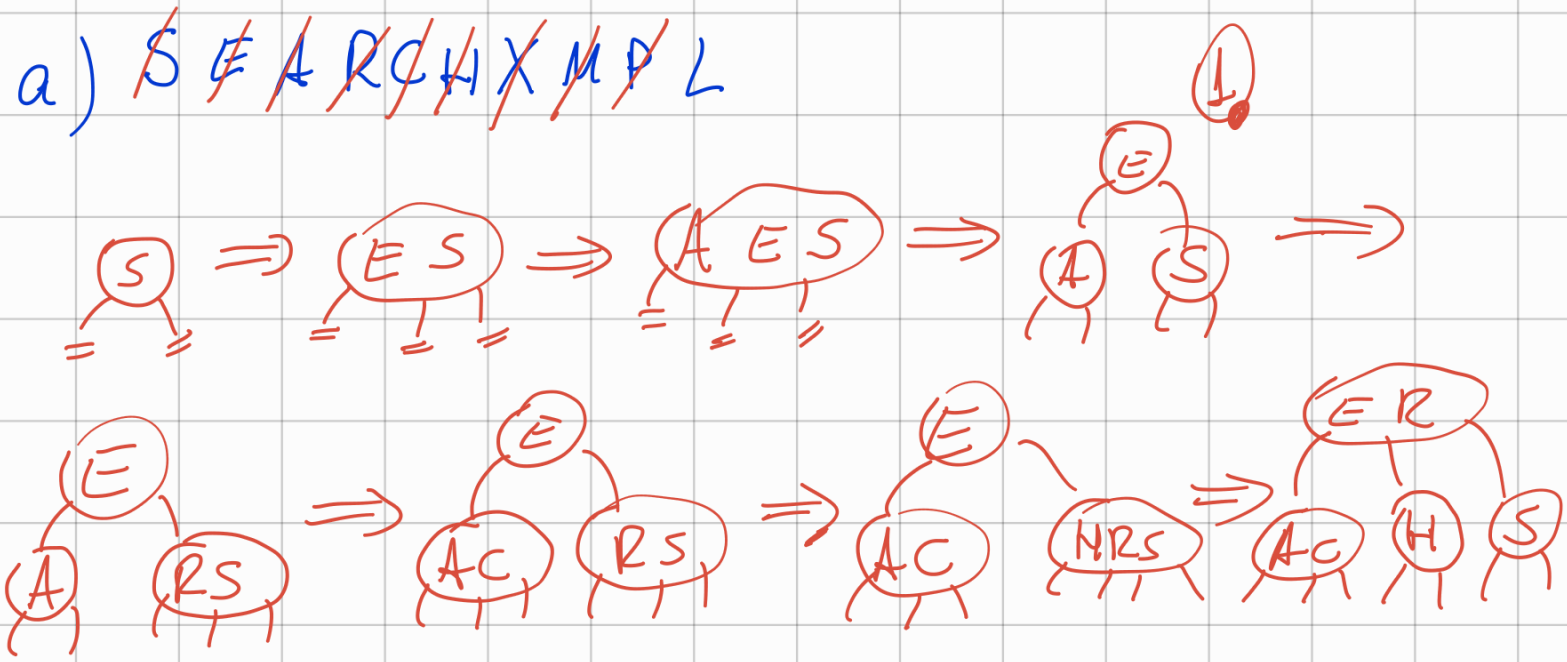
b) Inserir na ordem A C E H L M P R S X

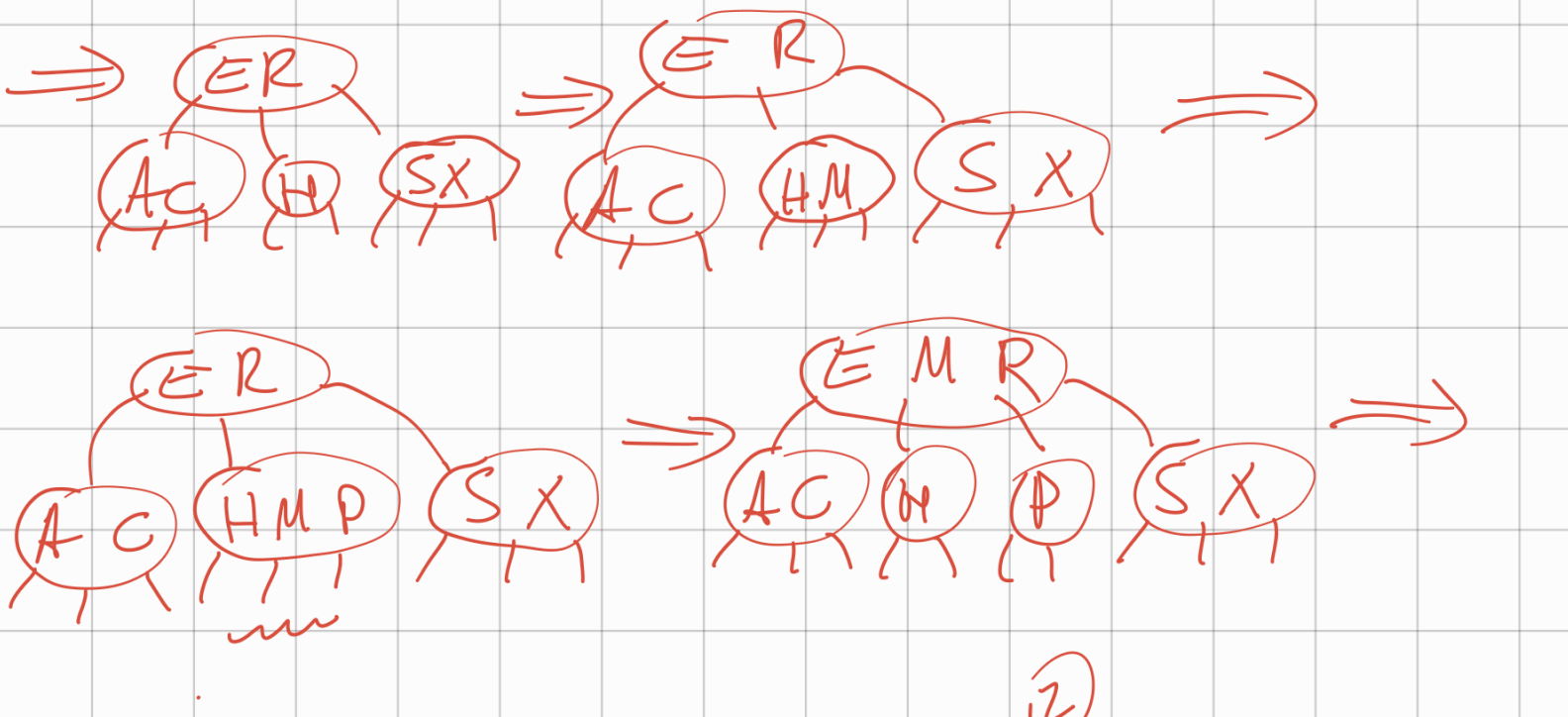
c) Inserir na ordem A B C D E F G H I J

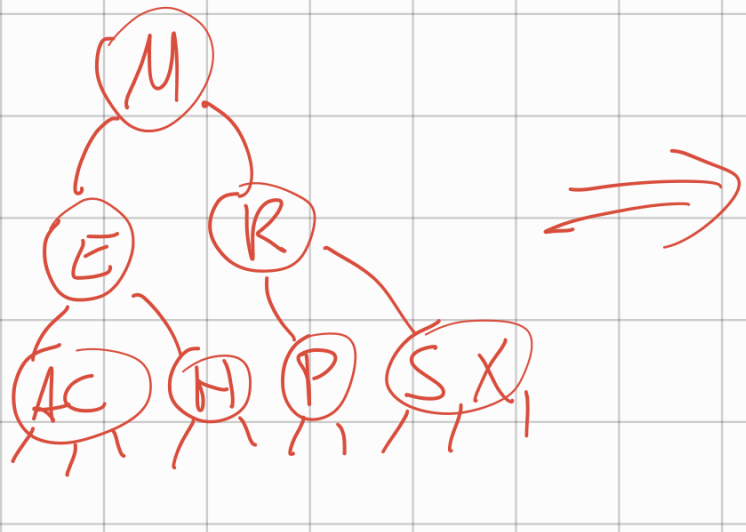
d) Inserir na ordem: J I H G F E D C B A

e) Há uma ordem de inserção para as chaves S E A R C H X M que resulte em uma árvore 2-3 de altura 1?

a) ~~S E A R C H X M P~~ L

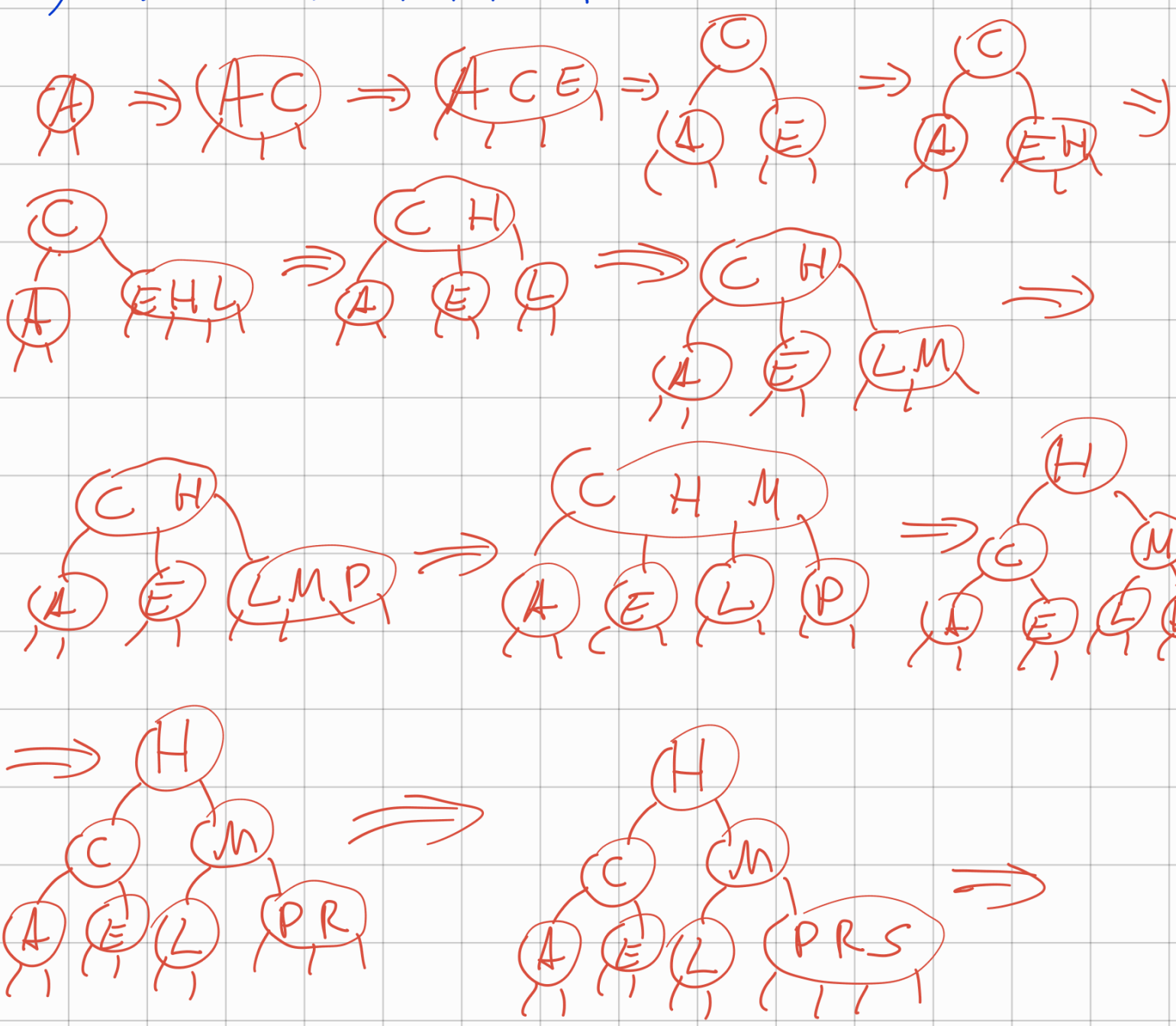




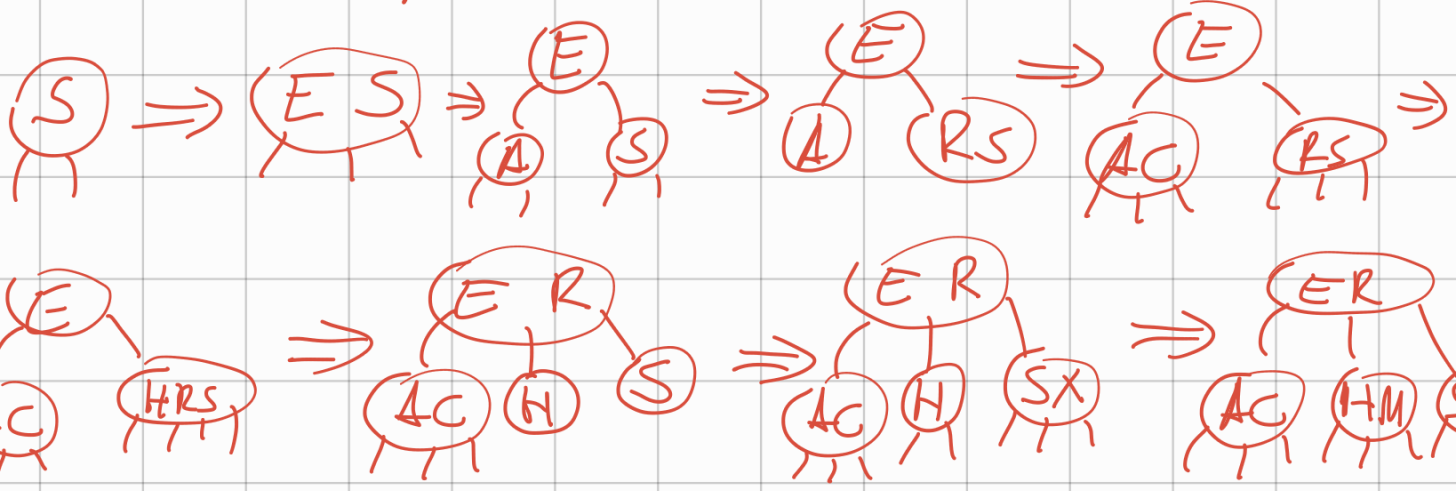


• A C E H L M P R S X

b) A C E H L M P R S X



c) ~~S~~ ~~E~~ ~~A~~ ~~R~~ ~~C~~ ~~H~~ ~~X~~ M



TC: Alguma ordem de inserção forme uma árvore 2-3 com altura diferente de 1?

- Desempenho do Pior Caso

  - Árvores 2-3 foram inventadas para garantir bom desempenho no pior caso

  - Considere uma árvore 2-3 com N nós.

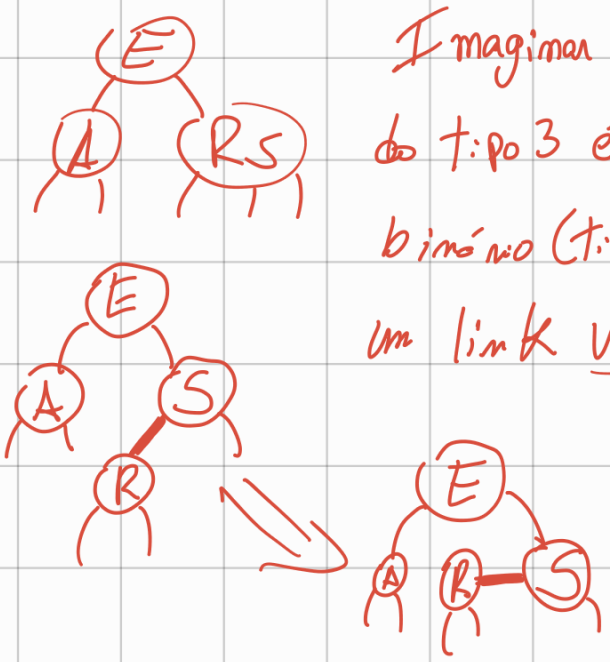
    - Se temos apenas nós simples, a altura será: $\lg N$

    - Se temos apenas nós duplos, a altura será: $\log_3 N$

  - Conclusão: $\log_3 N = \frac{\lg N}{\lg 3} = \frac{\lg N}{1.58} = 0{,}63 \lg N$

- Proposição: Em uma árvore 2-3 com N nós, busca e inserção nunca visitam mais que $\lg N$ nós, mesmo no pior caso

  - Cada visita fez no máximo 2 comparações de chaves



Imaginar que o nó do tipo 3 é um nó binário (tipo 2) com um link vermelho

Implementações

- Use 2 tipos de nós: um com uma chave e 2 links e outro com 2 chaves e 3 links

  ... como fica com as operações?

- Como fazer melhor?

Até agora temos:

	pior caso (tabela com N chaves)			caso médio (N chaves inseridas em ordem aleatória)		
	busca	inserção	delete	busca bem-sucedida	inseção	delete
busca sequencial em lista ligada	N	N	N	$N/2$	N	$N/2$
busca binária em vetor ordenado	$\lg N$	N	N	$\lg N$	$N/2$	$N/2$
busca binária em BST	N	N	N	$1.4 \lg N$	$1.4 \lg N$	?
árvore 2-3	$c \lg N$	$c \lg N$	$c \lg N$	$c \lg N$	$c \lg N$	$c \lg N$