Tabela de símbolos: árvore de busca balanceada

Marcelo K. Albertini

9 de Janeiro de 2014

Resumo de complexidades

Análises para operação efetuada após N inserções

	pior caso			caso médio			keys	chave
	get	put	delete	get	put	delete		
lista	N	N	N	N/2	N	N/2	não	equals()
vetores paralelos	log N	N	N	log N	N/2	N/2	sim	compareTo()
ABB	N	N	N	log N	log N	?	sim	compareTo()
meta	log N	log N	log N	log N	log N	log N	sim	compareTo()

Árvores balanceadas: $O(\log N)$ sempre

- árvore 2-3
- árvore rubro-negra
- árvore B

Árvore 2-3

Permite

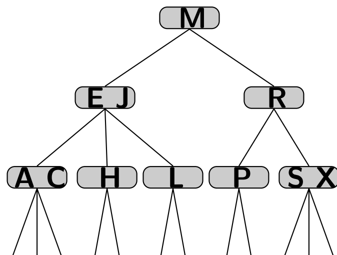
- 2-node: uma chave e dois filhos
- 3-node: duas chaves e três fihos

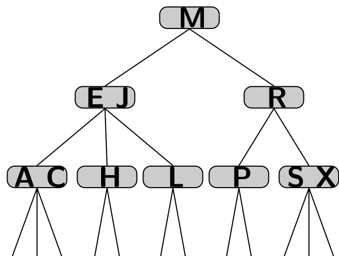
Balanceamento perfeito

Todo caminho da raiz para uma folha `null` tem o mesmo comprimento.

Ordem simétrica

Travessia in-order percorre chaves em ordem crescente



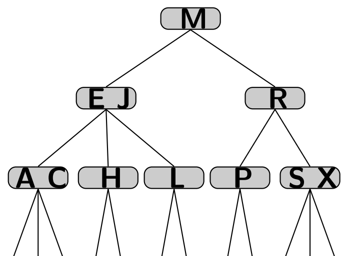


- A C é menor que E
- H está entre E e J
- L é maior que J

Árvore 2-3: operação get

Busca

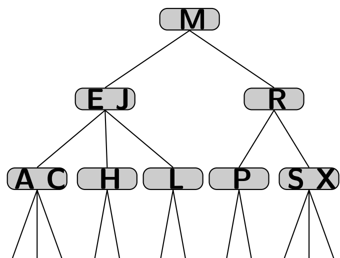
- Comparar chave de busca com chaves no nó
- Procurar intervalo contendo chave de busca
- Seguir nó associado recursivamente



Árvore 2-3: operação `put`

Inserir

- Buscar pela chave (como na ABB)
- Em um 2-node: torná-lo um 3-node
- Em um 3-node:
    - torná-lo um 4-node temporário
    - dividir o 4-node em dois 2-nodes
    - promover chave mediana para pai



Simulação: put

2

Simulação: put

Vai inserir 1

2

Simulação: put

Inseriu 1

1 2

Simulação: put

Vai inserir 5

1 2

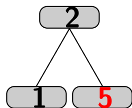
Simulação: put

Inserindo 5. Node com 3 chaves temporário

1 2 5

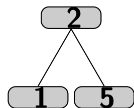
Simulação: put

Inseriu 5. Aumento a altura.



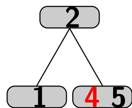
Simulação: put

Vai inserir 4



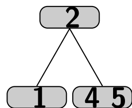
Simulação: put

Inseriu 4



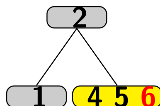
Simulação: put

Vai inserir 6



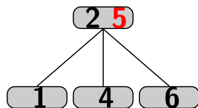
Simulação: put

Vai inserir 6. Cria um Node com 3 chaves temporário.



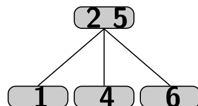
Simulação: put

Inserindo 6. Promoveu o 5.

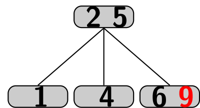


Simulação: put

Vai inserir 9

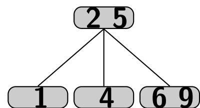


Simulação: put



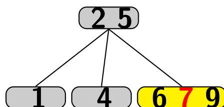
Simulação: put

Vai inserir 7

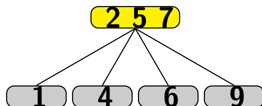


Simulação: put

Inserindo 7 em temporário. Vai promover chave central 7.

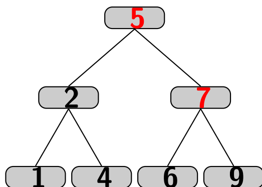


Inserindo 7. Dividiu temporário. Raiz fica temporária.



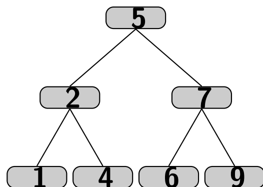
Simulação: put

Inserindo 7. Divide a raiz. Promoveu o central 5.

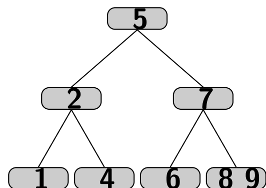


Simulação: put

Vai inserir 8

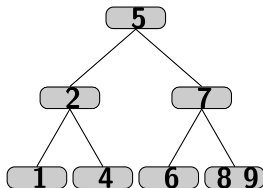


Simulação: put

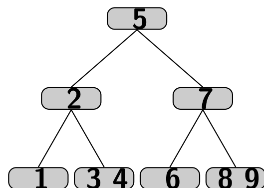


Simulação: put

Vai inserir 3



Simulação: put



Invariantes em uma árvore 2-3

- Novas chaves são sempre inseridas nas folhas
- Ordem simétrica é mantida
- Todo caminho da raiz para uma folha tem o mesmo tamanho
- Nodes tem 2 ou 3 filhos

Árvore 2-3: complexidades

Altura da árvore

Aumenta quando raiz fica com 3 chaves.

Operação de divisão de Node com 3 chaves tem custo constante.

- Pior caso: inserção $O(\log N)$
- Caso médio: $\log N$

Todas as operações são $O(\log N)$ – garantido.

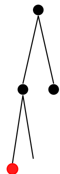
Árvore rubro-negra (ARN)

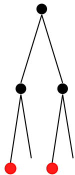
- A árvore rubro-negra (ARN) equivale à árvore 2-3
- ARN tem implementação melhor e mais simples

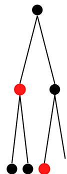
ARN: modificações

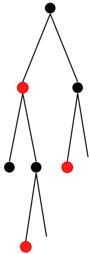
Modificações à árvore ABB original.

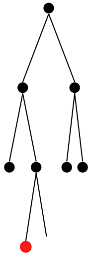
```
1  static final boolean RED = true;
2  static final boolean BLACK = false;
3
4  class Node {
5    Chave chave;
6    Valor valor;
7
8    Node esq, dir;
9    boolean cor; // cor da ligação ao pai
10 }
11
12 private boolean isRed(Node x) {
13   if (x == null) return false;
14   return x.cor == RED;
15 }
```

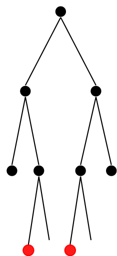


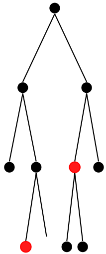


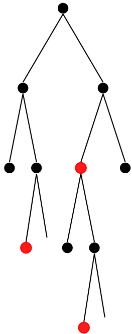


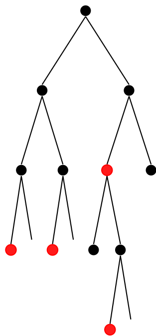


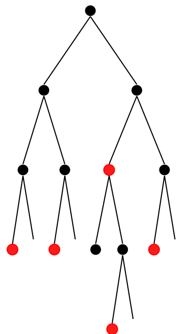


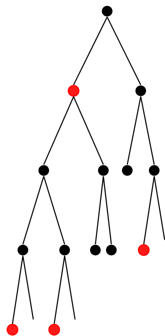


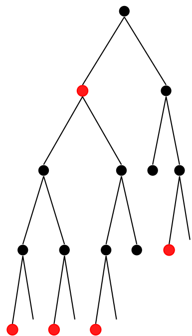


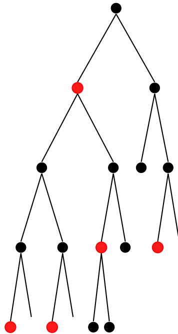


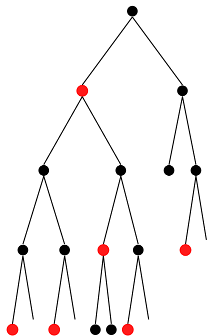


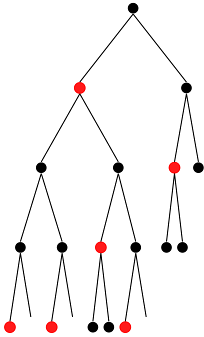


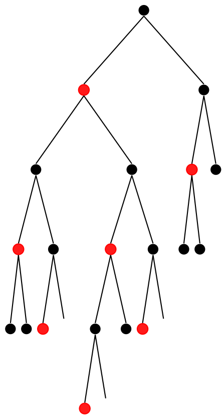


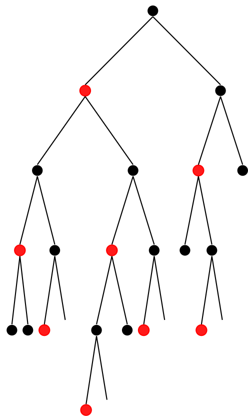


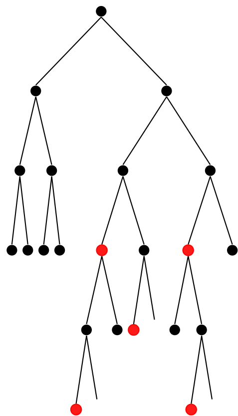


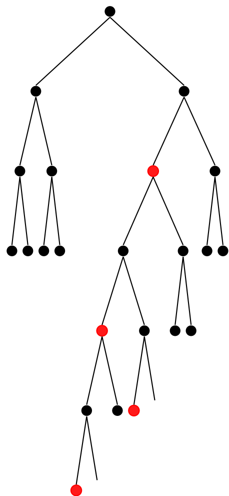


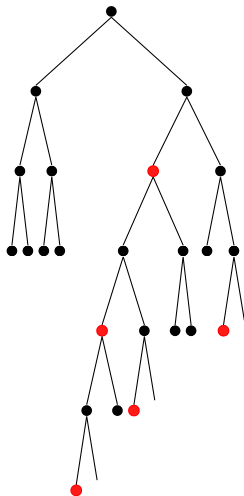


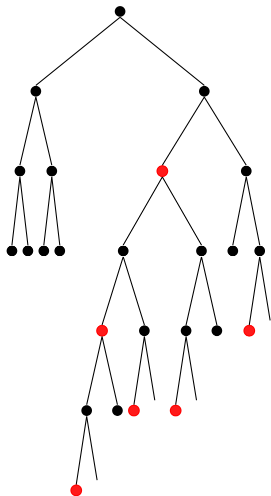


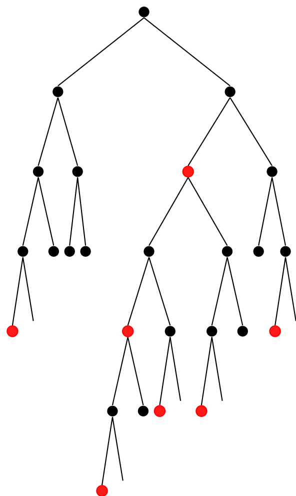


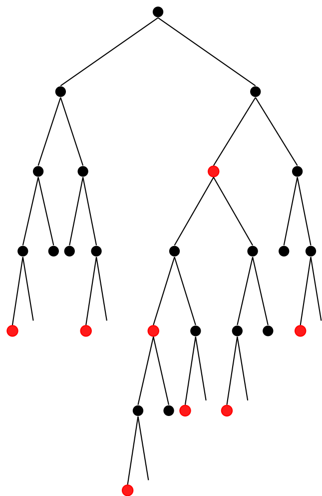


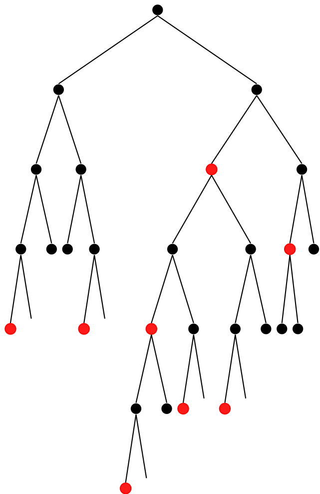


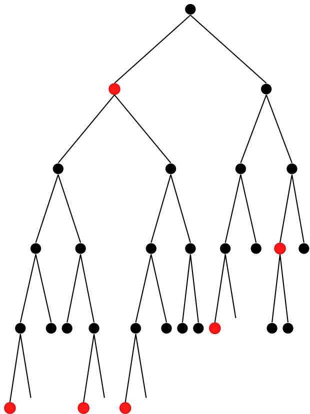


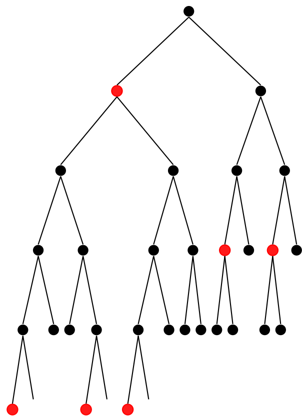


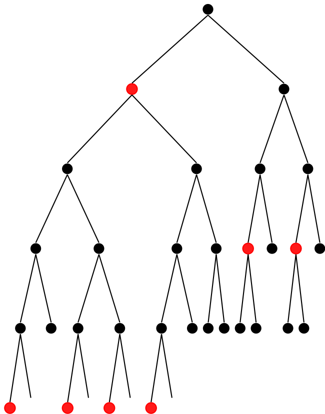


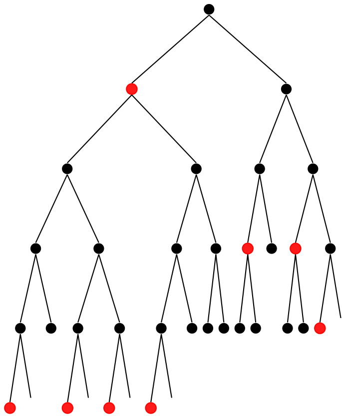


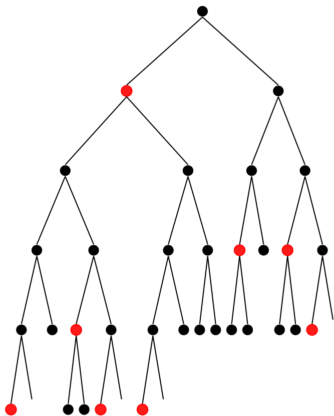


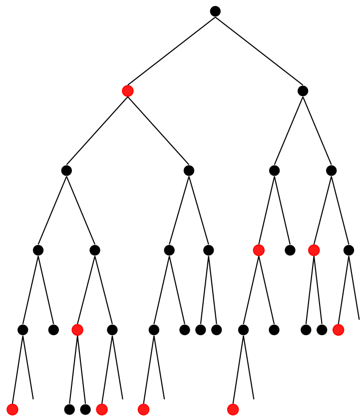


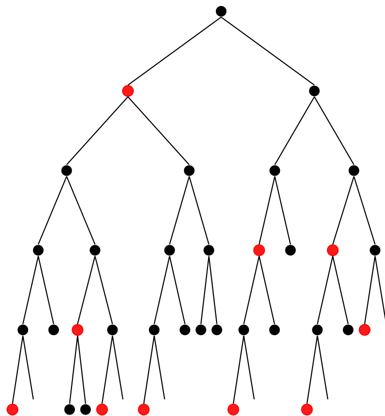


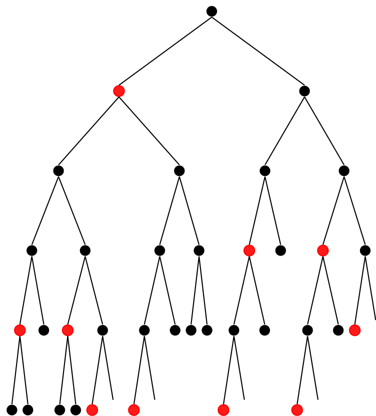


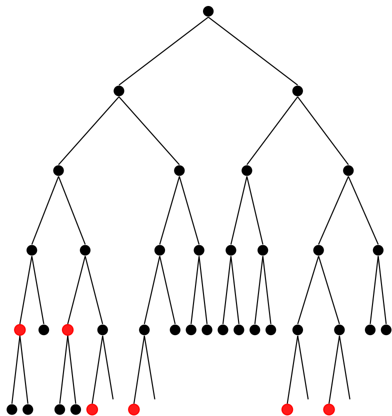


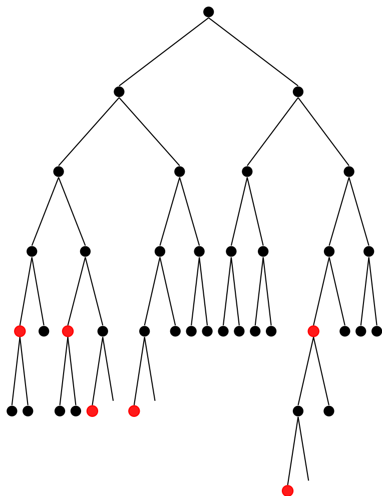


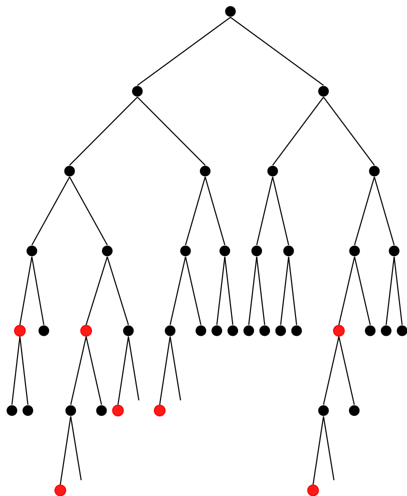


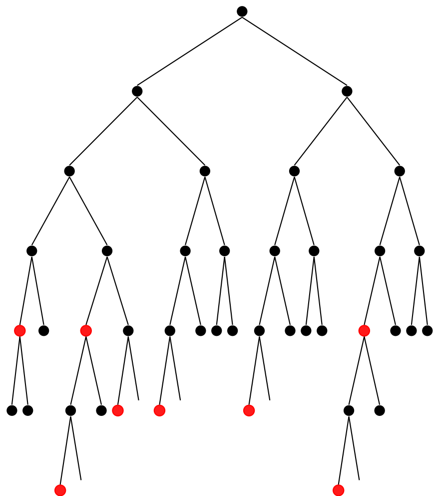


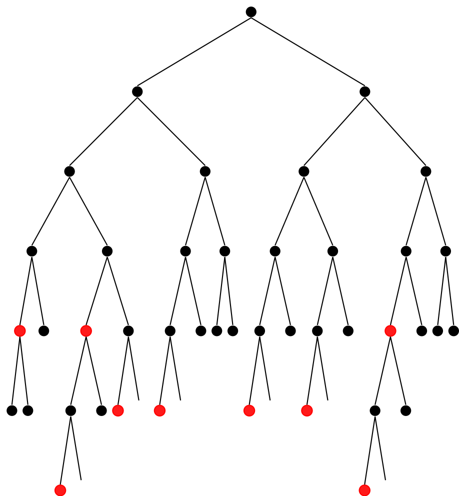


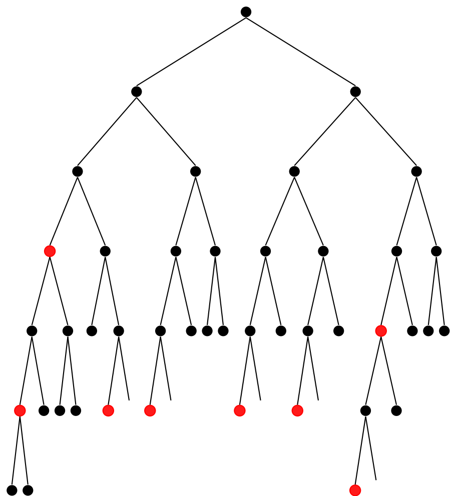


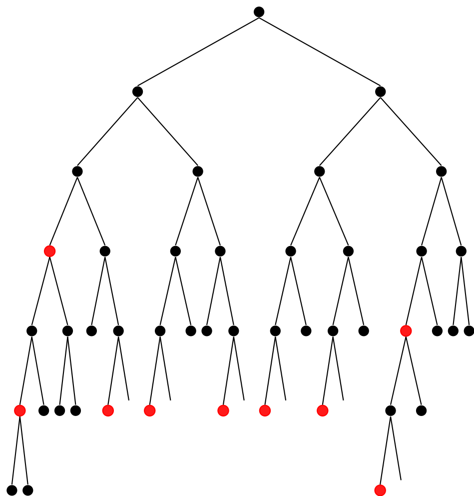


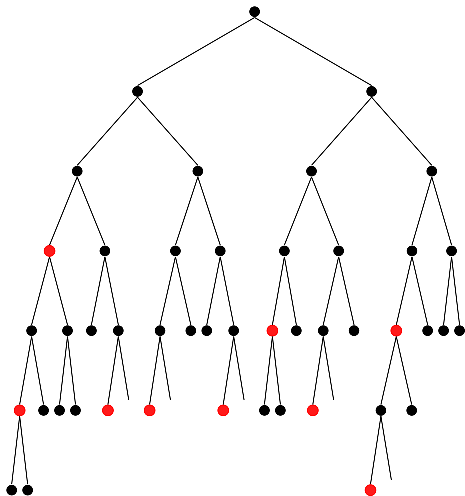


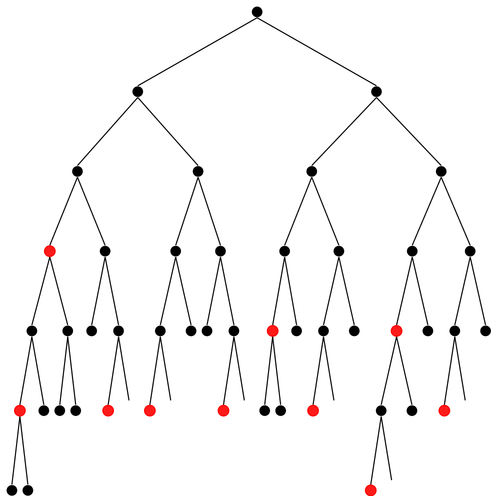


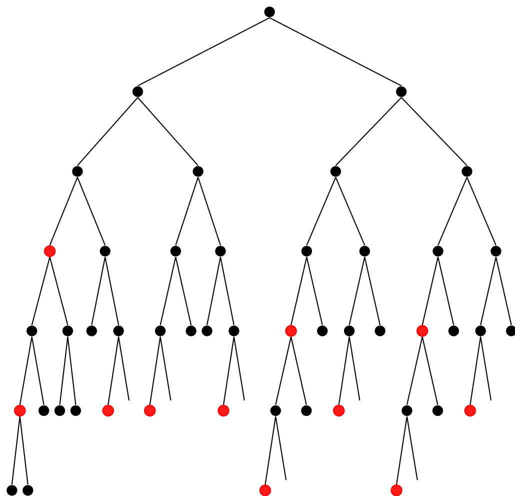


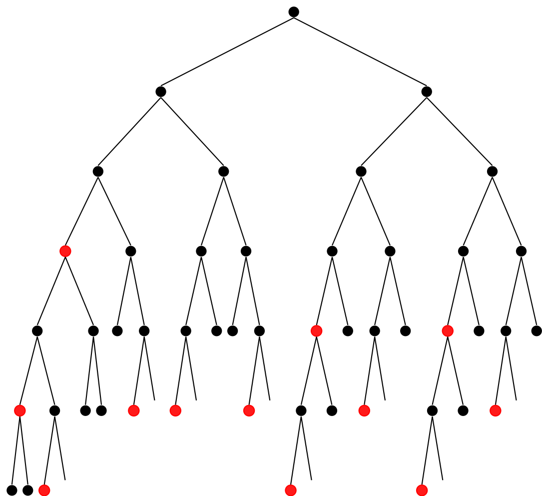


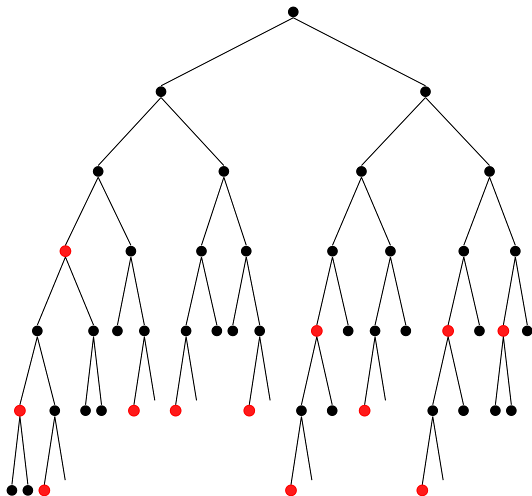


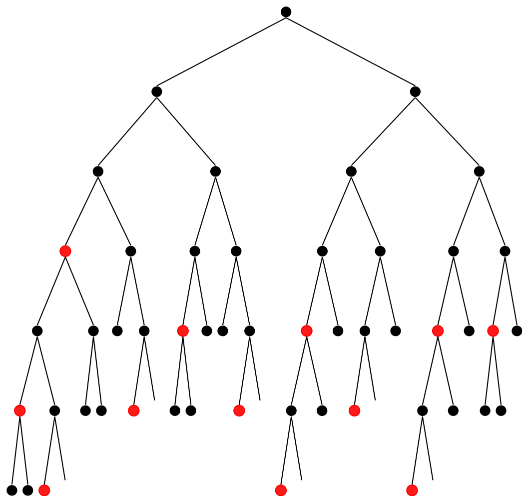


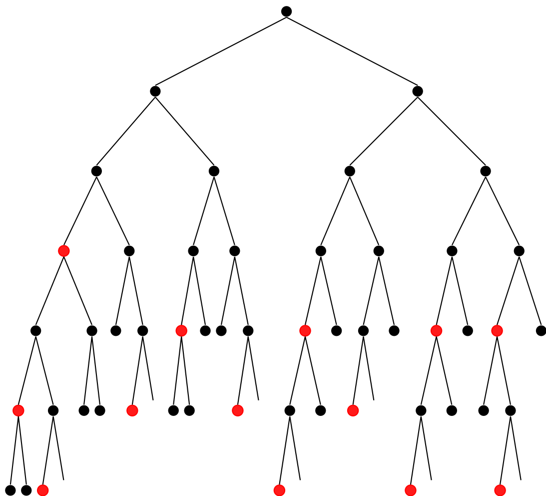


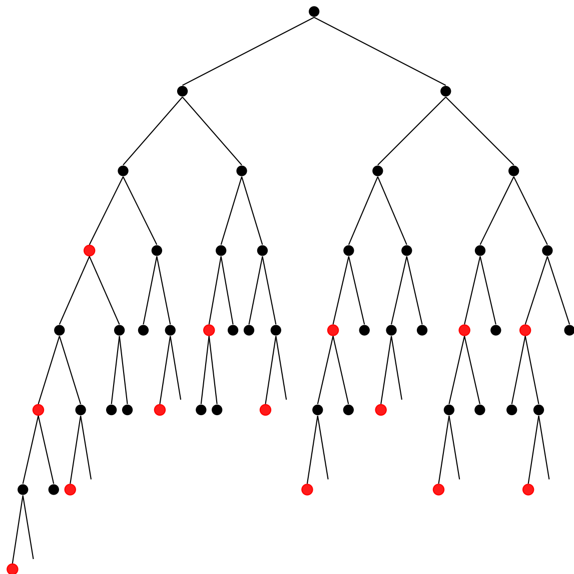


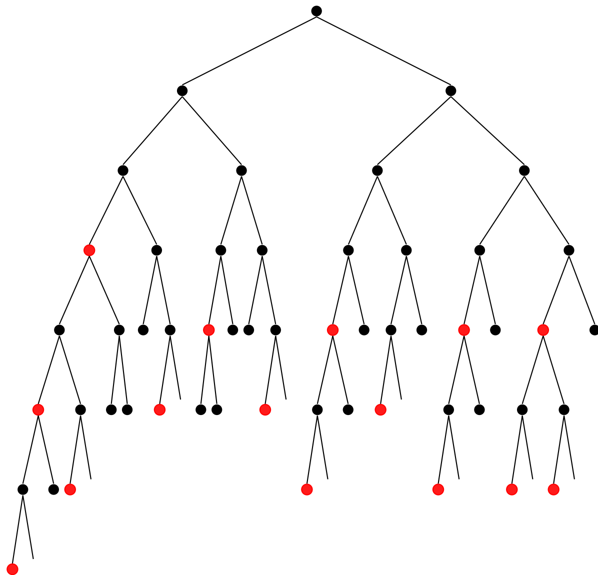


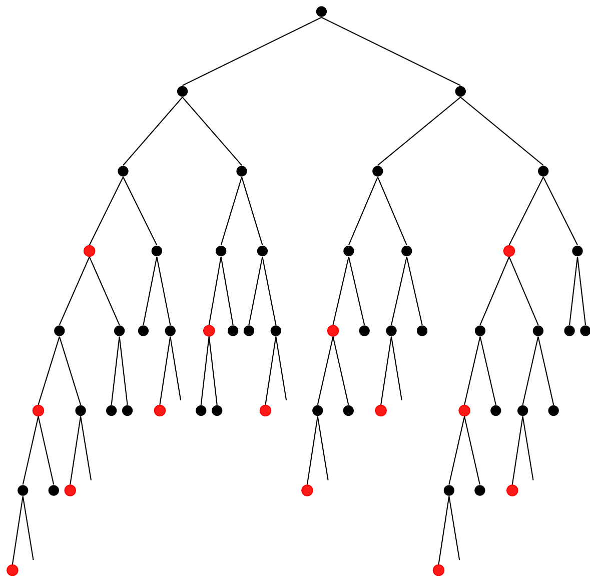


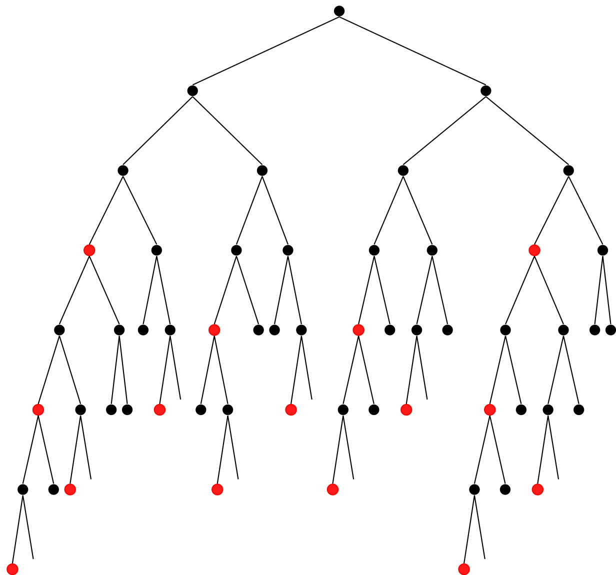


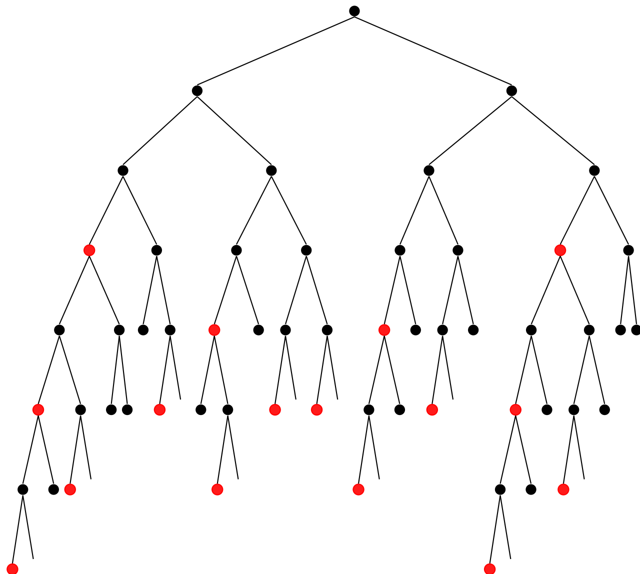


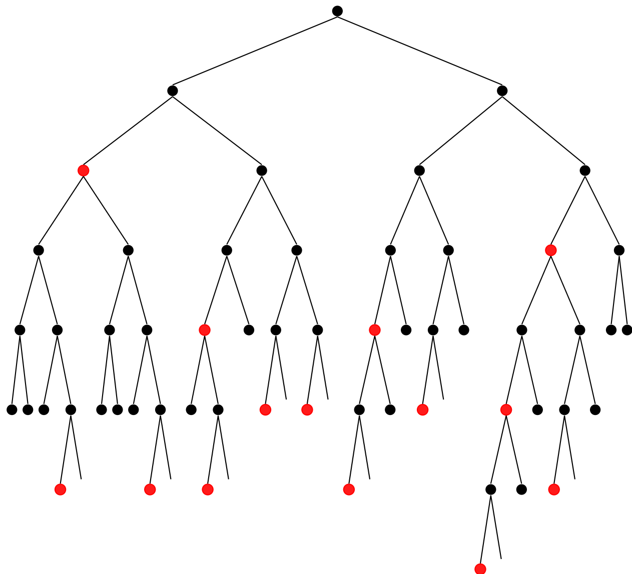


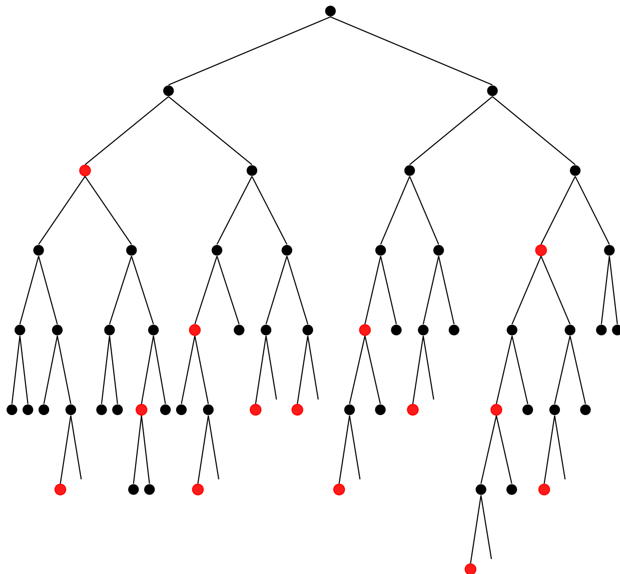


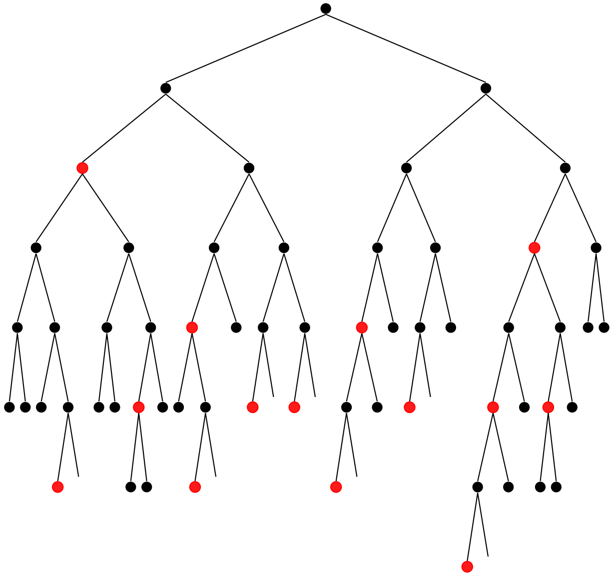


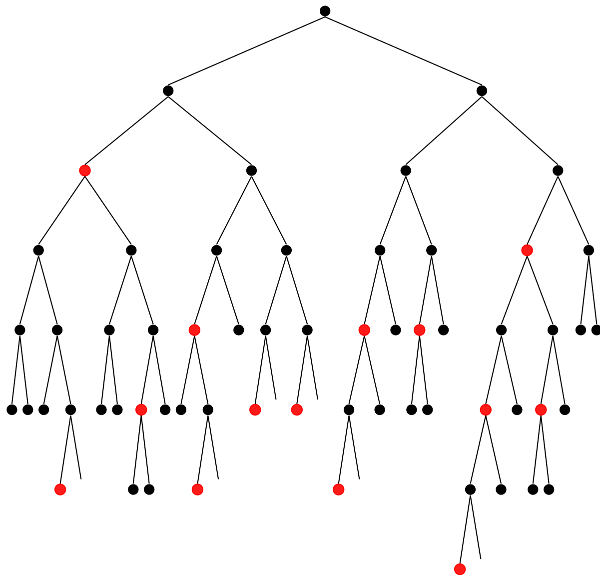


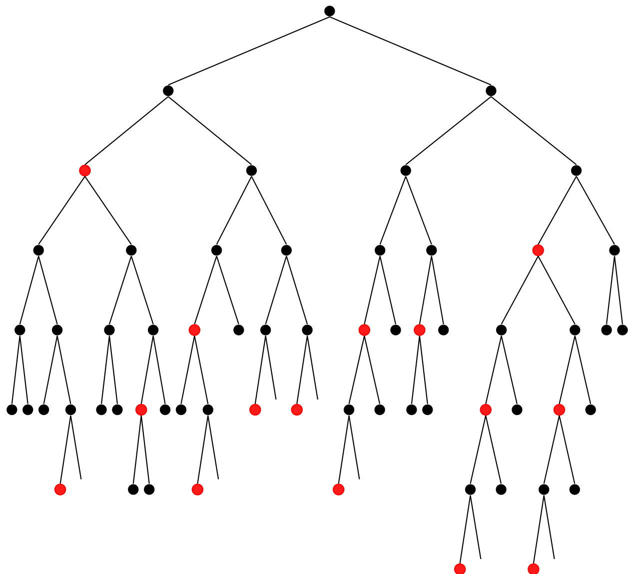


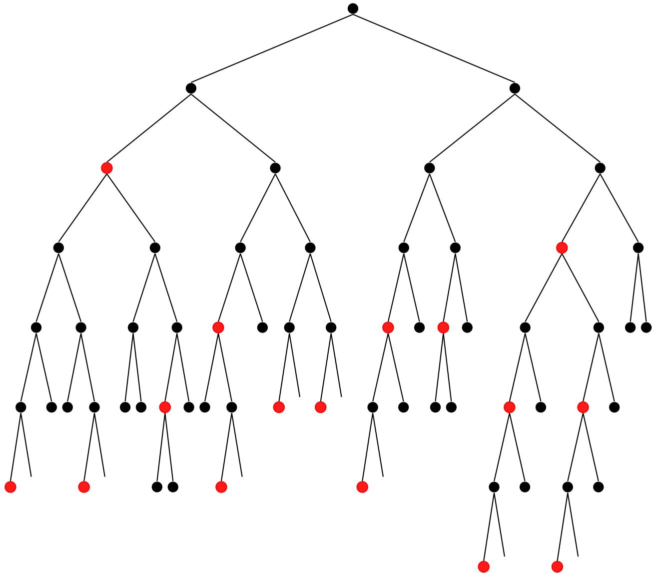


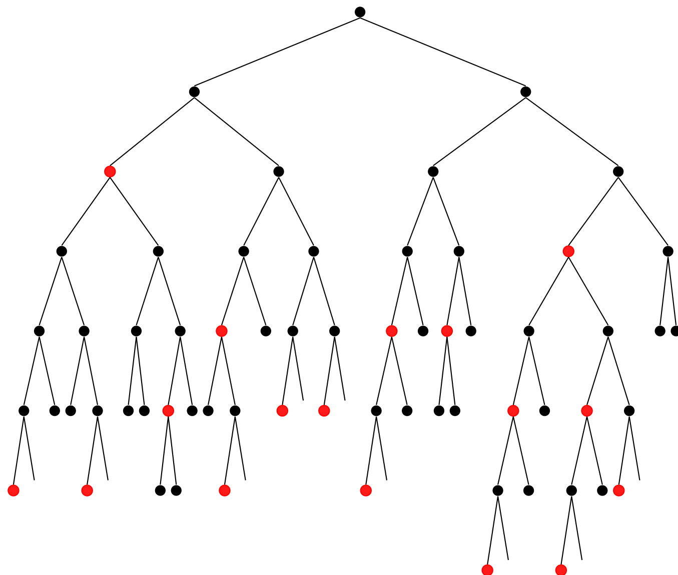


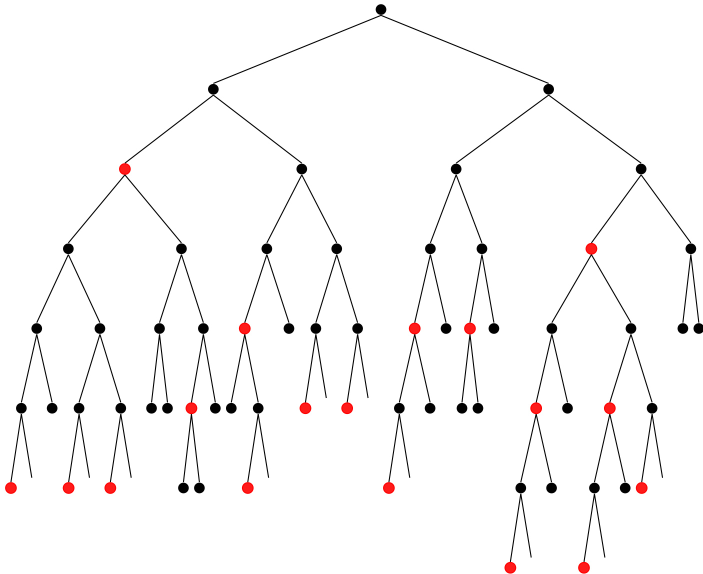


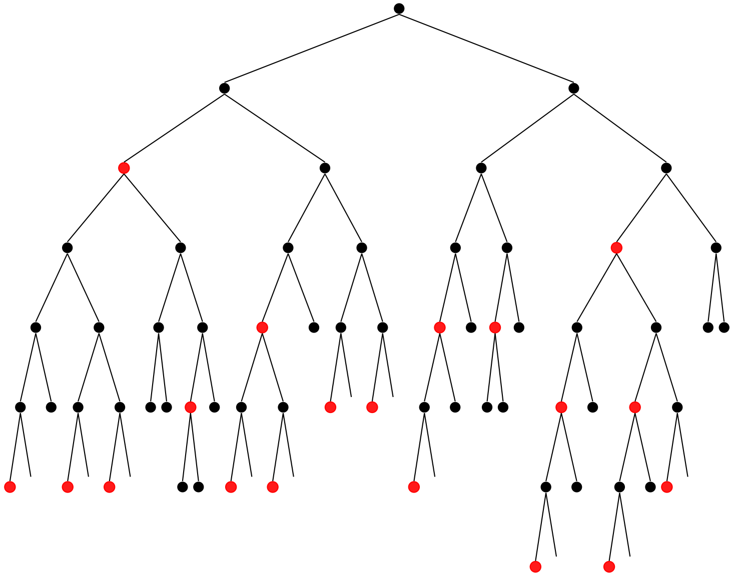


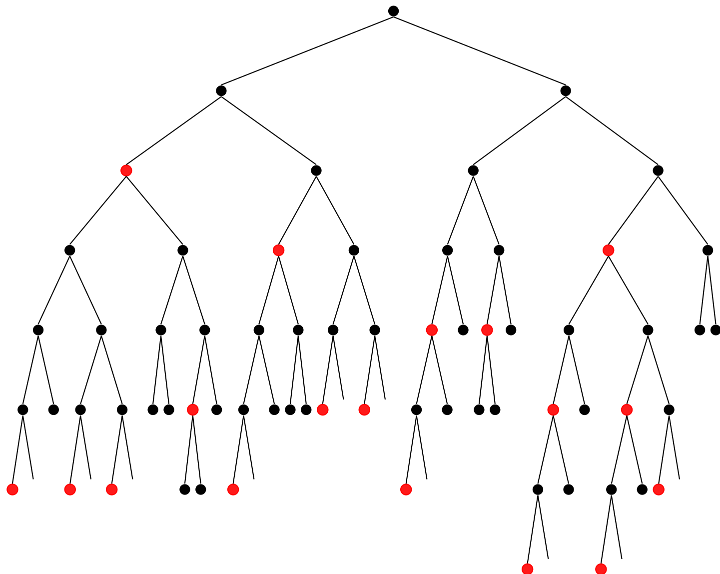


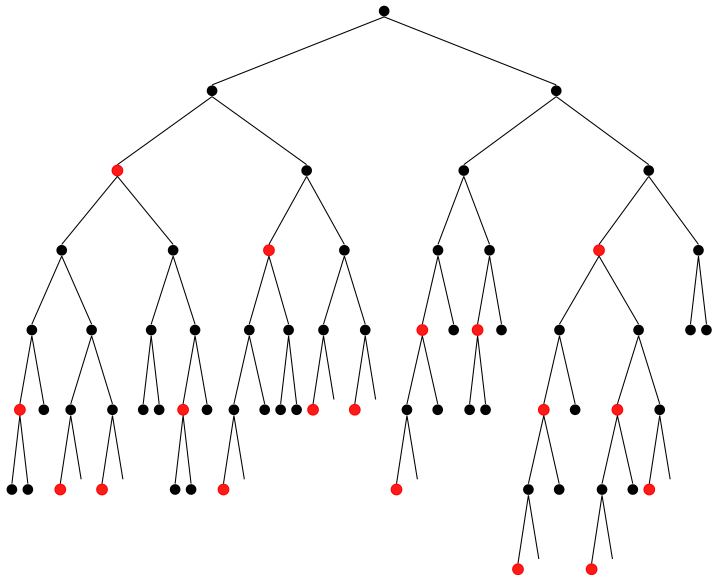


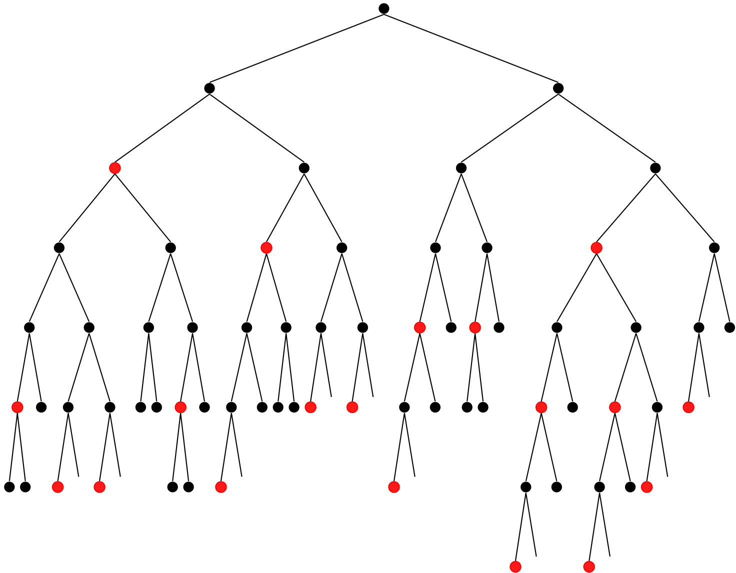


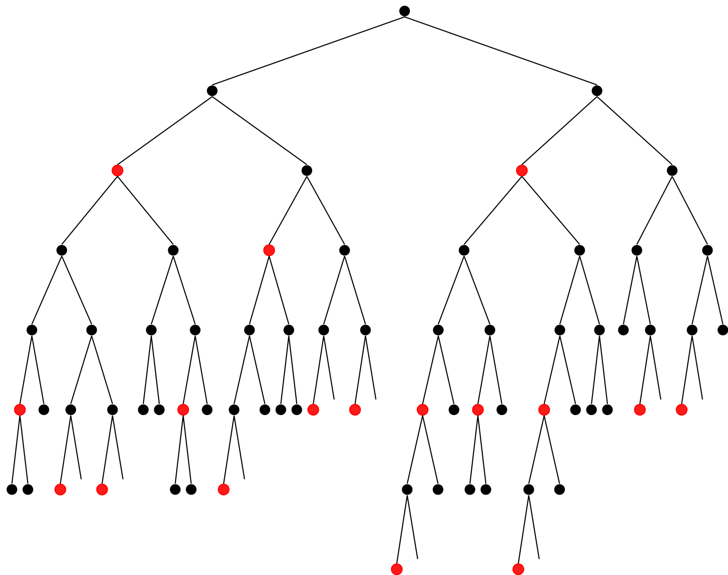


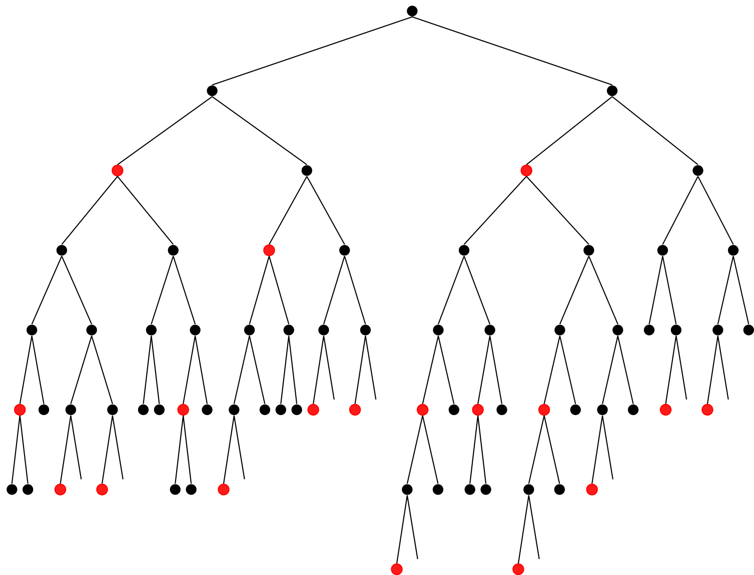


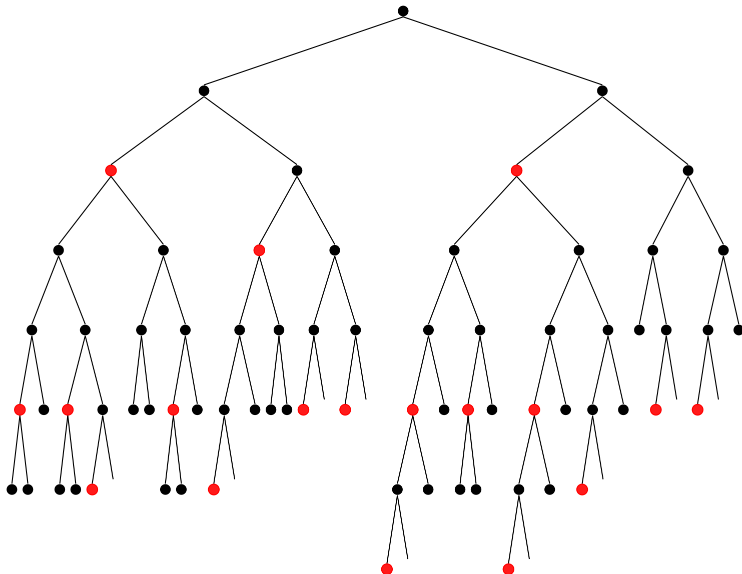


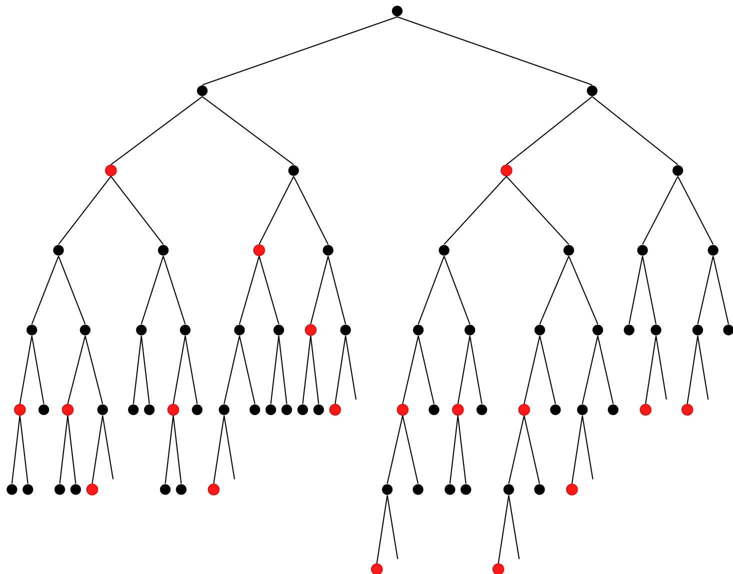


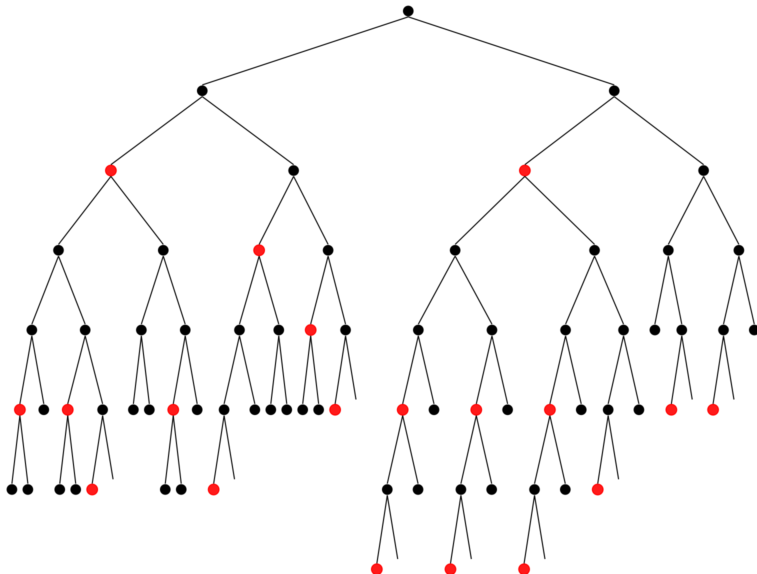


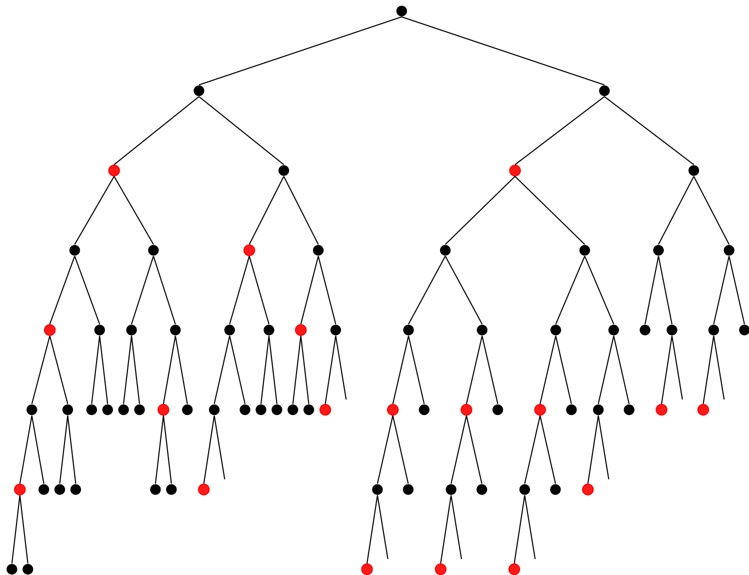


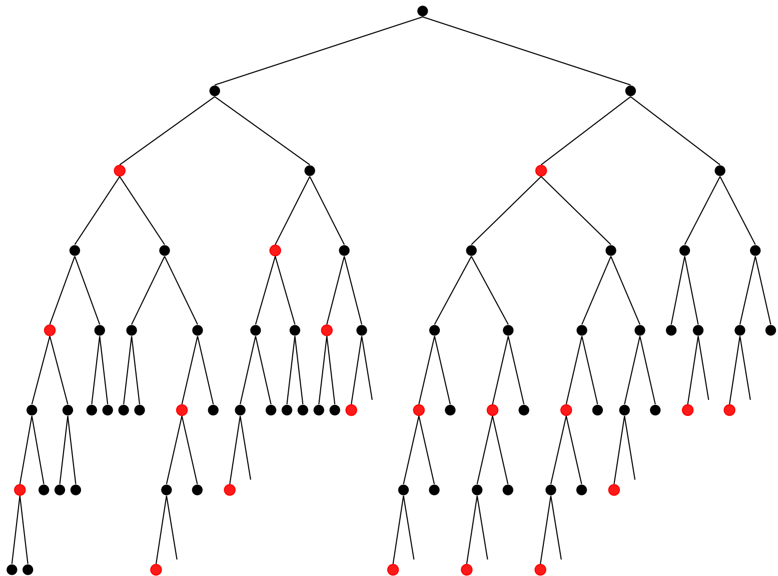


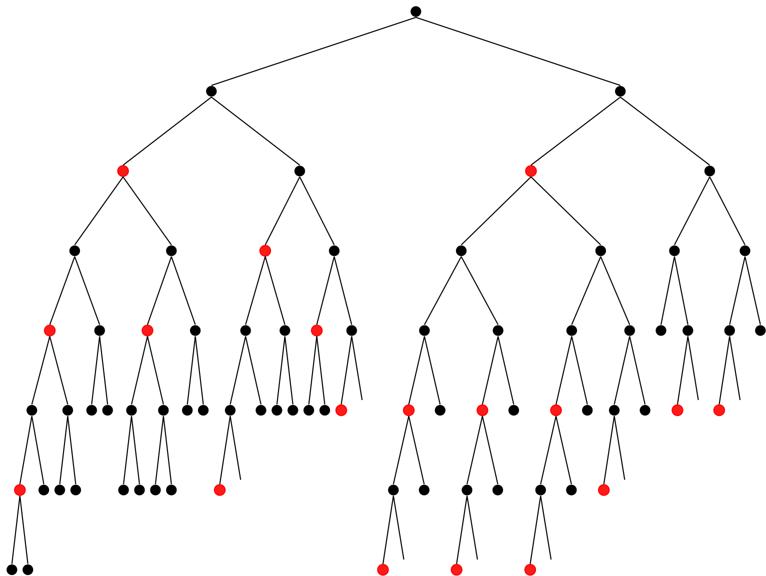


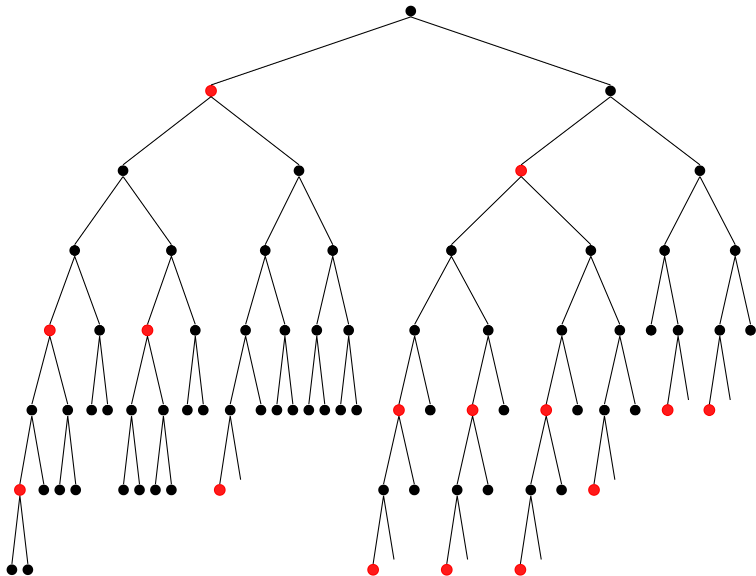


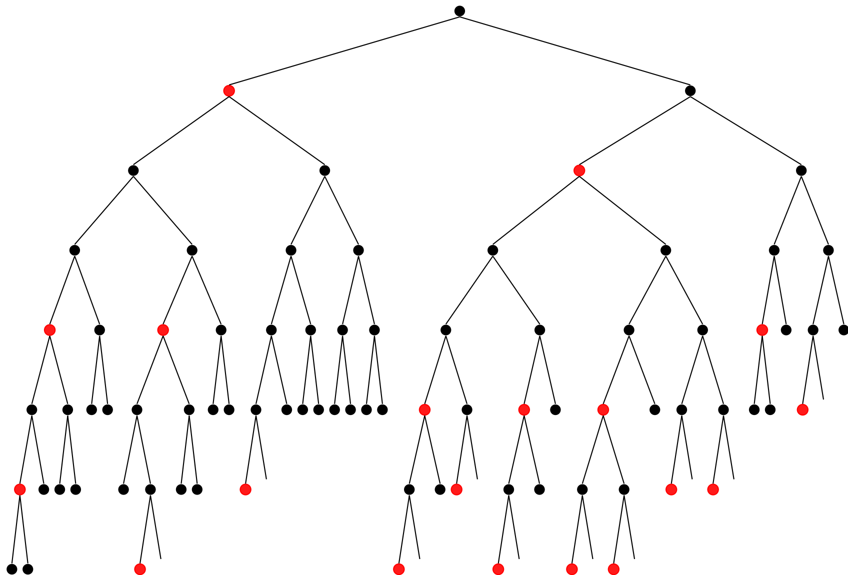


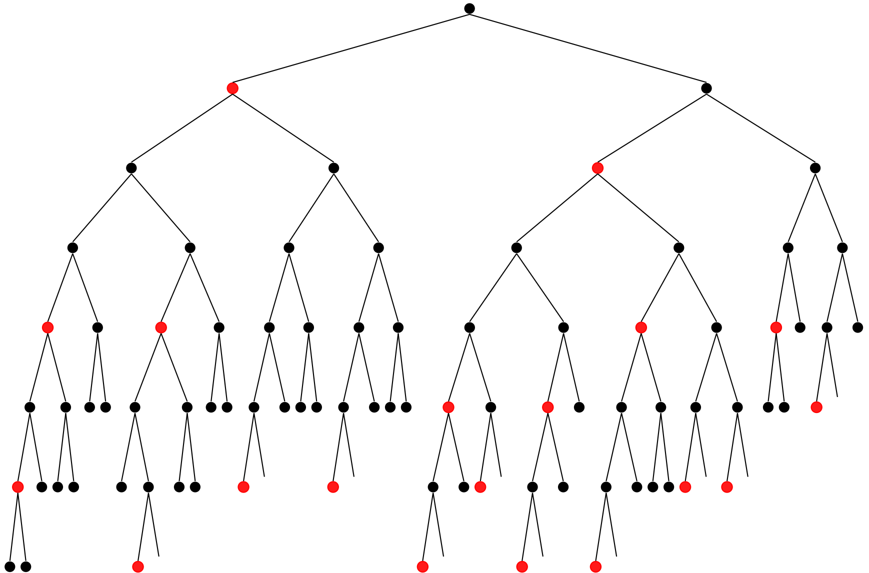


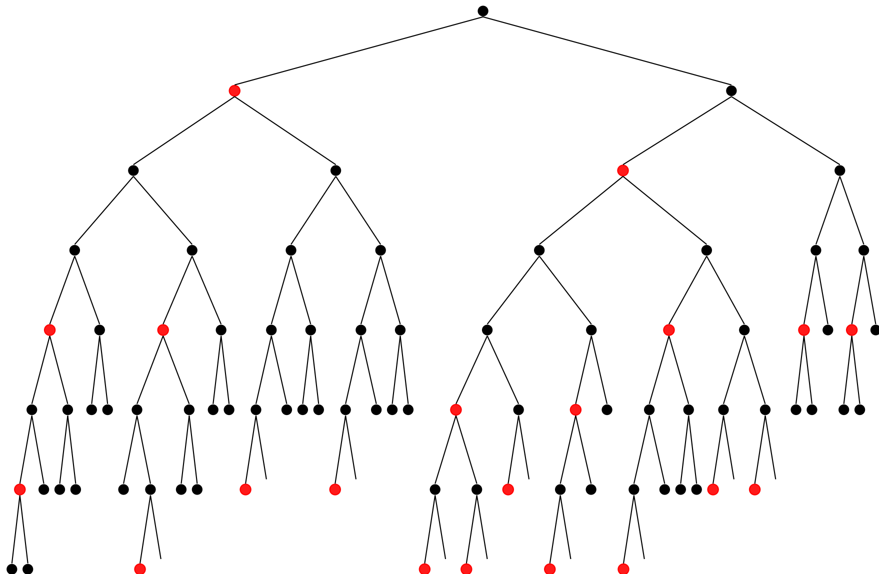


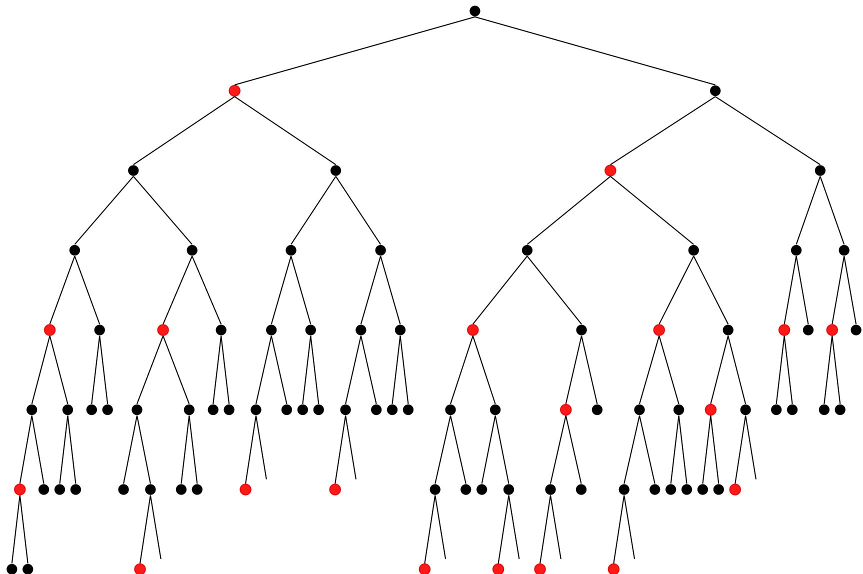


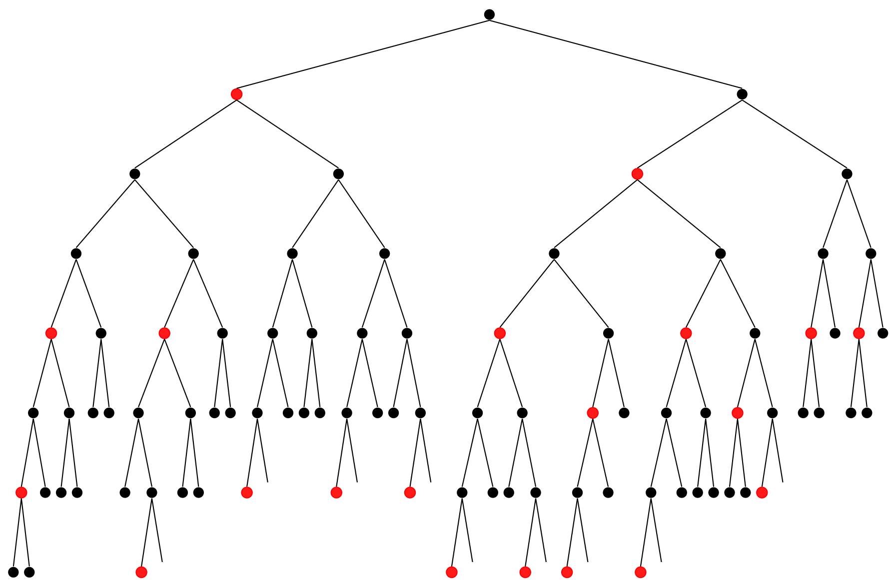


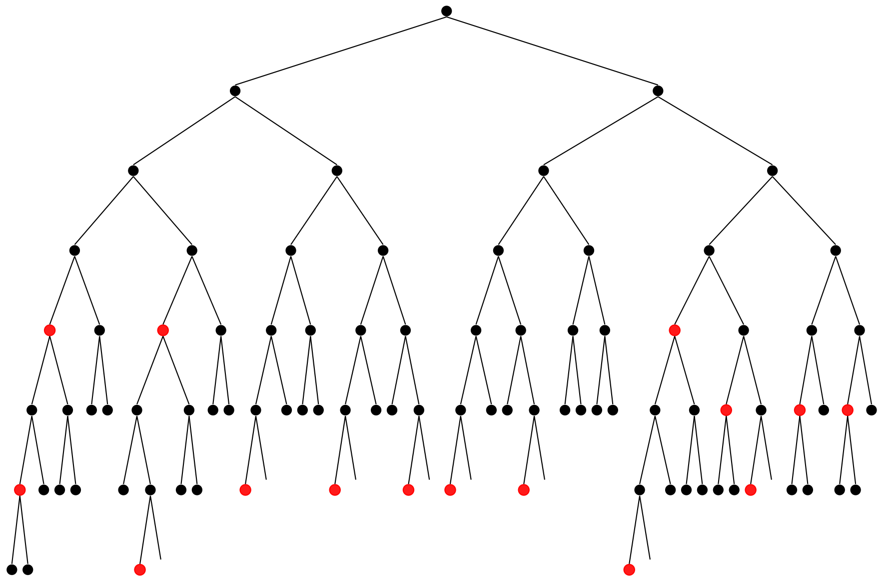




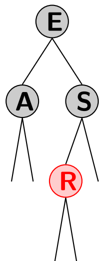




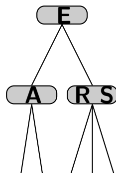


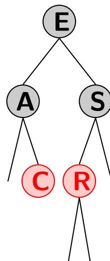


Inserir C em

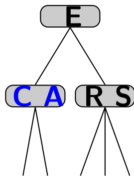


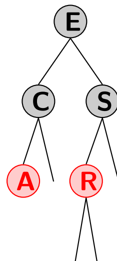
equivalente a



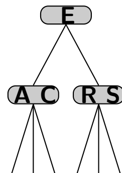


C A deve ser arrumado com uma rotação à esquerda em A





equivalente a



Rotação à esquerda

Orientar uma ligação vermelha pendendo à direita para pender à esquerda.

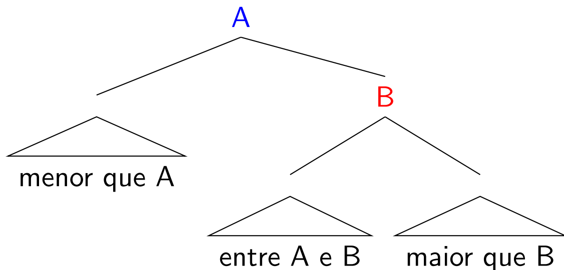
```
1 // Quando chamada, A.dir deve ser RED
2 Node rotacaoEsq(Node A) {
3   Node B = A.dir;
4   A.dir = B.esq;
5   B.esq = A;
6   B.cor = A.cor;
7   A.cor = RED;
8   return B;
9 }
```

Pai vira filho à esquerda do próprio filho.

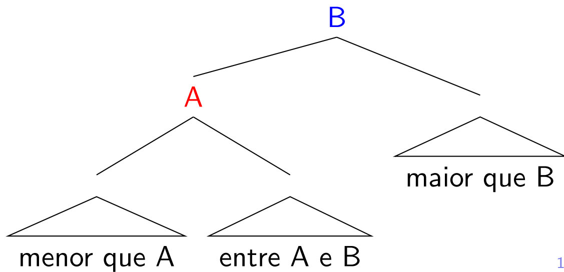
A.dir é RED.

```
1 Node rotacaoEsq(
      Node A) {
2  Node B = A.dir;
3  A.dir = B.esq;
4  B.esq = A;
5  B.cor = A.cor;
6  A.cor = RED;
7  return B;
8 }
```

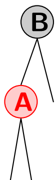
Cor azul pode ser RED ou BLACK.



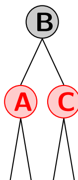
Rotação à esquerda em A:



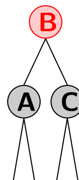
Inserir C em



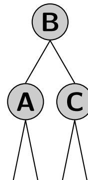
Inserção no `null`



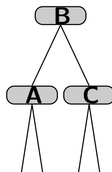
Temporário com 3 chaves



Inverte cores



Raiz sempre é BLACK



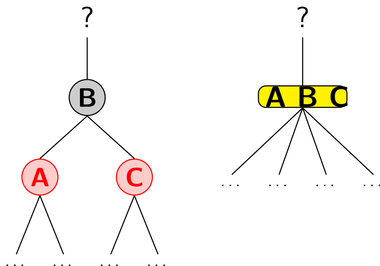
ARN: operações

Troca de cores

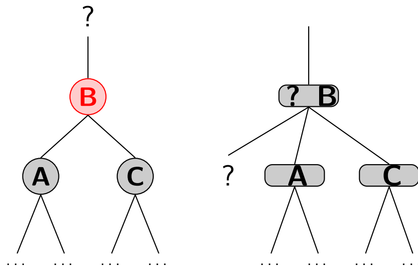
Recolorir para dividir um 4-node.

```
1 void trocaCores(Node h) {
2   h.cor = RED;
3   h.esq.cor = BLACK;
4   h.dir.cor = BLACK;
5 }
```

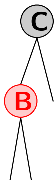
Situação representa Node com três chaves (um 4-node)



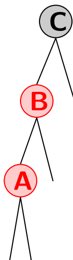
Promoveu chave central ao Node pai e criou 2 Node filhos.



Inserir A em

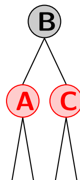


Inserção no `null`



Promover chave central. Ou seja, rotacionar à direita em B

A B C



Inverter cores. Vimos.

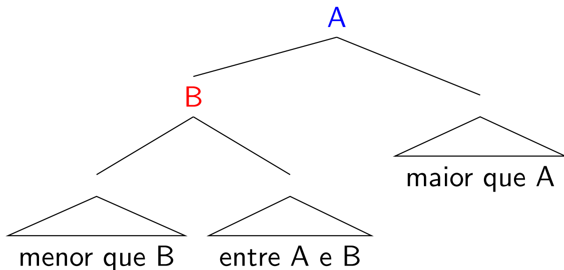
Rotação à direita

Orientar uma ligação vermelha pendendo à esquerda para pender à direita.

```
1 // Quando chamada, A.esq deve ser RED
2 Node rotacaoDir(Node A) {
3   Node B = A.esq;
4   A.esq = B.dir;
5   B.dir = A;
6   B.cor = A.cor;
7   A.cor = RED;
8   return B;
9 }
```

Pai vira filho à direita do próprio filho.

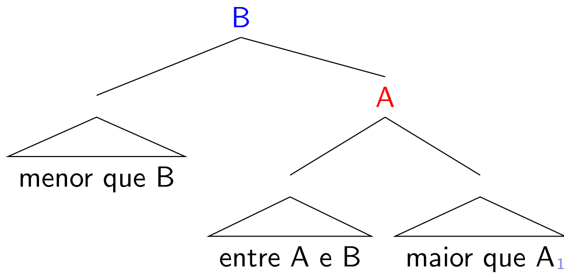
Cor azul pode ser RED ou BLACK.



A.esq é RED.

```
1 Node rotacaoDir(
      Node A){
2   Node B = A.esq;
3   A.esq = B.dir;
4   B.dir = A;
5   B.cor = A.cor;
6   A.cor = RED;
7   return B;
8 }
```

Rotação à direita em A:



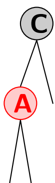
ARN: inserção

Estratégia

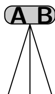
Manter correspondencia com árvores 2-3 com as operações da ARN

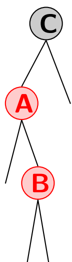
```
1  Node put(Node h, Chave c, Valor v) {
2    if (h == null) return new Node(c, v, RED);
3    int cmp = c.compareTo(h.chave);
4
5    if (cmp < 0) h.esq = put(h.esq, c, v);
6    else if (cmp >0) h.dir = put(h.dir, c, v);
7    else h.valor = v;
8
9    if (isRed(h.dir) && !isRed(h.esq))
10     h = rotacaoEsq(h);
11   if (isRed(h.esq) && isRed(h.esq.esq))
12     h = rotacaoDir(h);
13   if (isREd(h.esq) && isRed(h.dir))
14     trocaCores(h);
15
16   return h;
17 }
```

Inserir B em

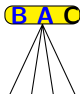


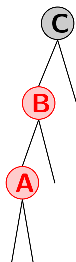
Insere no fim da recursão



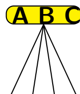


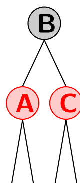
Volta na recursão, faz rotação à esquerda em A



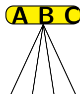


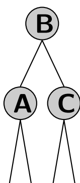
Volta na recursão, rotação à direita em C.





Inverter cores





Continua volta recursiva