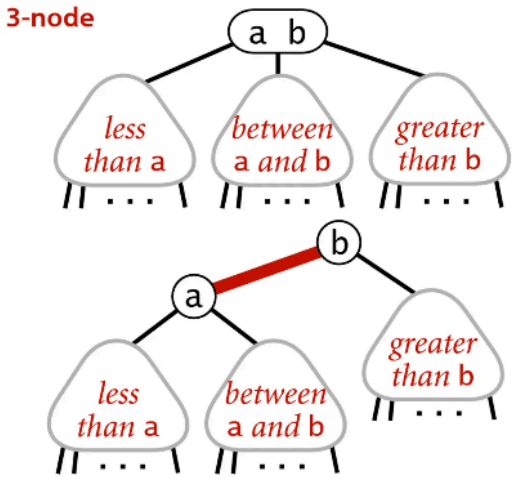
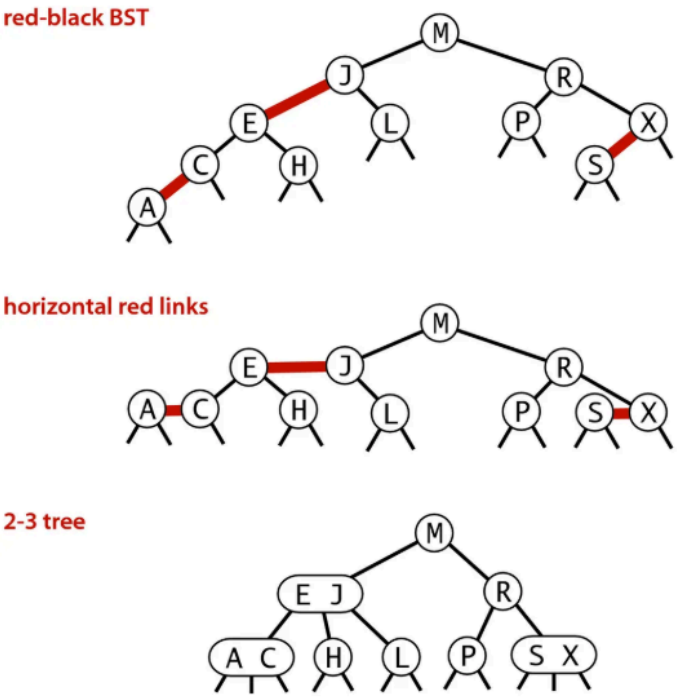
Aula 04 - Árvore Red-Black

- É uma maneira de implementar a **Árvore 2-3**
- Cada nó duplo da árvore 2-3 é representado por dois nós simples ligados por um link red.
- Nossas BSTs são esquerdistas (*left-leaning*)



Encoding a 3-node with two 2-nodes connected by a left-leaning red link

- Definição: A BST red-black é uma BST cujos os links são red e black e tem as seguintes propriedades
    - O link red são sempre para a esquerda
    - Nenhum nó incide em dois links red
    - Balanceamento negro perfeito: todo caminho da raiz até um link NULL tem o mesmo número de links negros.



- A profundidade black de um nó “x” é o número de links negros no caminho da raiz até “x”

A altura black de árvore é o máximo da profundidade negra de todos os nós

```
typedef struct STNode* link;
enum tipo{RED,BLACK};          // Define que tipo só pode
assumir RED ou BLACK
#define RED 0
#define BLACK 1

struct STNode {
 Item item;
 link l, r;
 int N;
 int color;
};

link h, z;

link NEW(Item item, link l, link r, int N){
     link x = malloc(sizeof(*x));
     x->item = item;                // Equivalente a x.item =
item
```

```
      x->l = l;
      x->r = r;
      x->N = N;
      x->color = RED;                    // Sempre tentamos
aglutinar os nós na 2-3
      return x;
}

void STinit(){
      h = (z = NEW(NULLITEM, 0, 0, 0));
      z->color = BLACK;
}
```

```
void STinit(){
      h = (z = NEW(NULLITEM, 0, 0, 0));
      z->color = BLACK;
}
```

Como fazer a Busca?

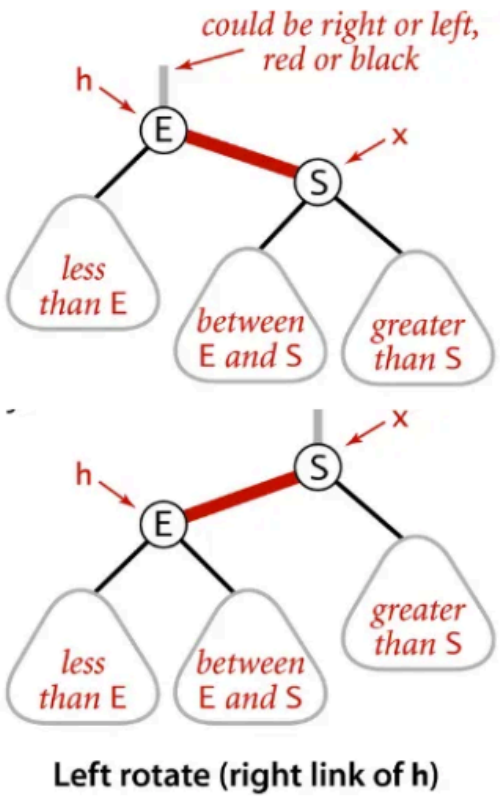
```
Item seachR(link r, Key k){
      if (r == z)
            return NULLITEM;
      Key t = Key(r->item);
      if (eq(k, t))
            return r->item;
      if(less(k,t))
            return searchR(r->l, k);
      return searchR(r->r, k);
}

// Wrapper
Item STsearch (Key k){
      return searchR(h, k);
}
```

Rotações

- A inserção é complicada. Depende de inserções

Rotação Esquerda (anti-horária)

O filho de h da direita sobe. Adota a raiz antiga como seu filho esquerdo e o filho direito se mantém.



Left rotate (right link of h)

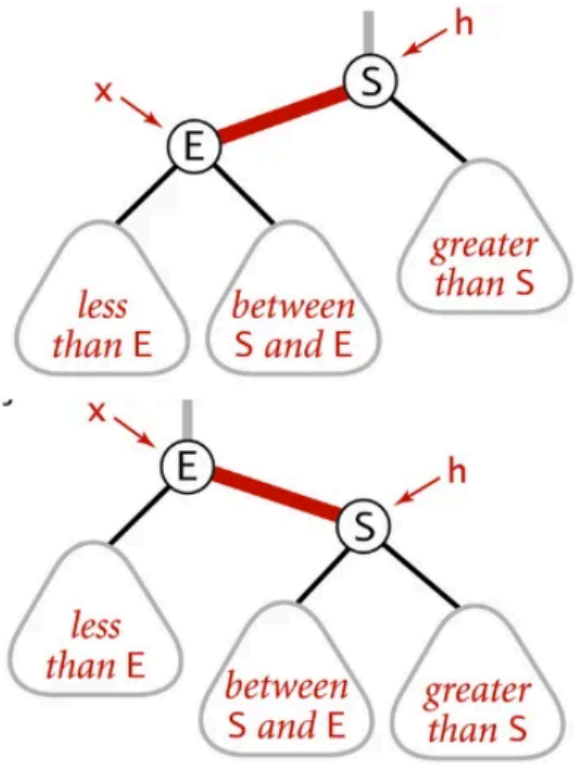
Como implementar?

```
link rotateLeft(link r){
      link x = r->r;
      r->r = x->l;
      x->l = r;
      x->color = r->color;
      r->color = RED;
      x->N = r->N
      r->N = 1 + r->l->N + r->r->N;
      return x;
}

Node rotateLeft(Node h) {
     Node x = h.right;
     h.right = x.left;
     x.left = h;
```

```
    x.color = h.color;
    h.color = RED;
    x.N = h.N;
    h.N = 1 + size(h.left) + size(h.right);
    return x;
}
```

Rotação direita (horária)

O filho esquerdo de h sobe e adota h como seu filho direito e mantém seu filho a esquerda



Right rotate (left link of h)

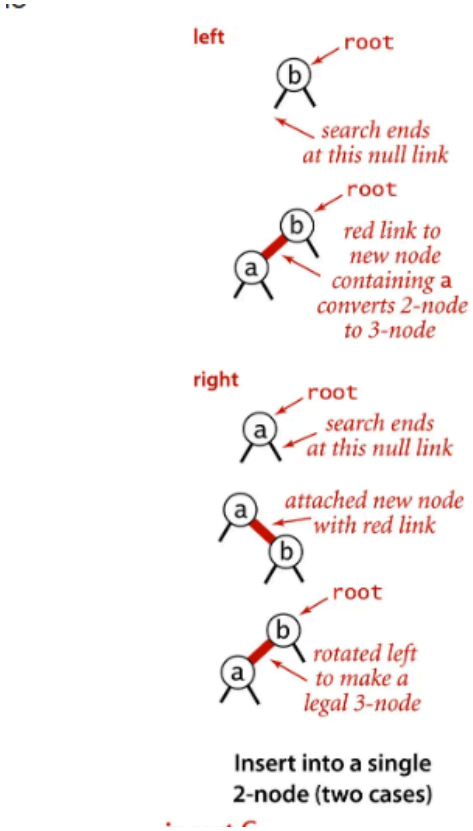
Como implementar?

```
link rotateRight(link r){
     link x = r->l;
     r->l = x->r;
     x->r = r;
     x->color = r->color;
     r->color = RED;
```

```
    x->N = r->N;
    r->N = 1 + r->l->N + r->r->N;
    return x;
}

Node rotateRight(Node h) {
  Node x = h.left;
  h.left = x.right;
  x.right = h;
  x.color = h.color;
  h.color = RED;
  x.N = h.N;
  h.N = 1 + size(h.left) + size(h.right);
  return x;
}
```

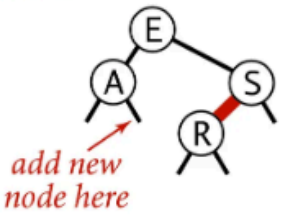
- As operações de rotação são locais
- Após as rotações a altura black se mantém, porém podemos ter um nó red no lado errado ou 2 nós red seguidos

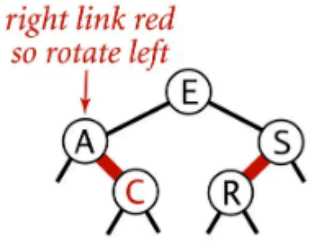
Inserção

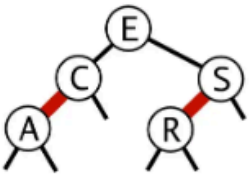


Insert into a single 2-node (two cases)

insert C





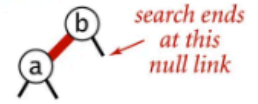


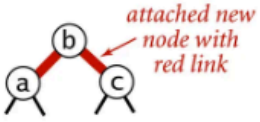
Insert into a 2-node
at the bottom

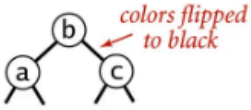
smaller between

Insert into a 2-node
at the bottom

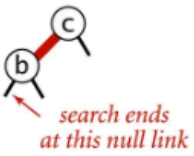
larger

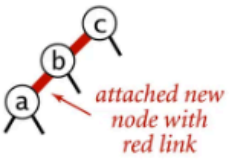


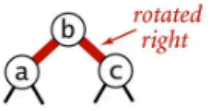


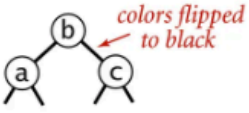


smaller

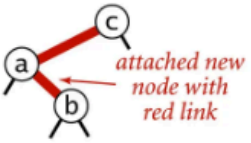


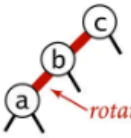


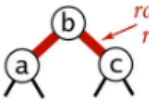


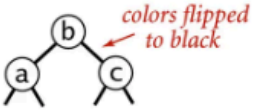


between





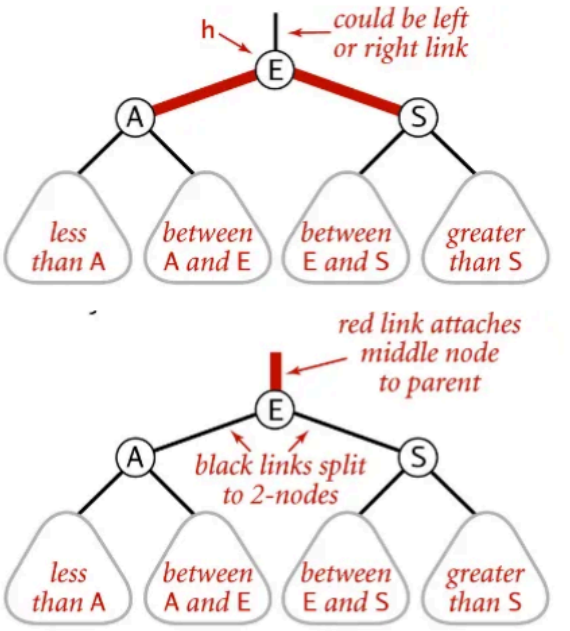




Insert into a single 3-node (three cases)

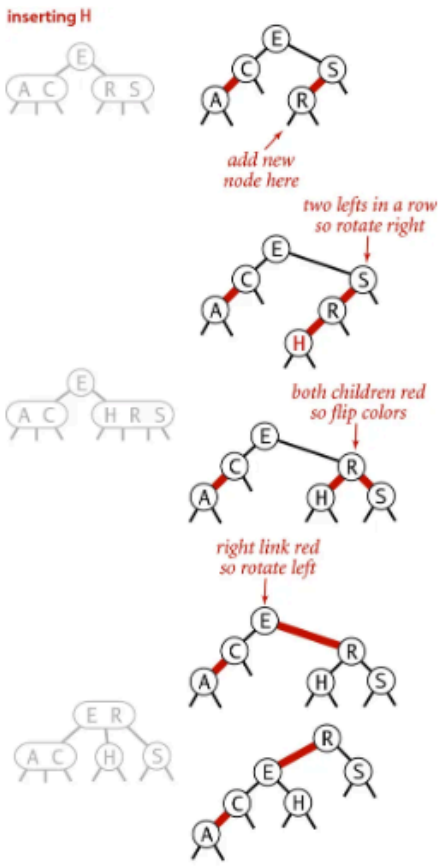
Insert into a single 3-node (three cases)

- Toda inserção de nós precisa de 0, 1 ou 2 rotações seguida de uma troca de cores
- No flip colors o pai fica com o link vermelho



Implementação do Flip Colors:

```
void flipColors(Node h) {
     h.color = RED;
     h.left.color = BLACK;
     h.right.color = BLACK;
}
```



Insert into a 3-node at the bottom

Como implementar a inserção:

```
link insertR(link r, Item item){
      if(r == z)
            return NEW(item,z, z, 1);
      Key k = Key(item), t = Key(r->item);
      if(less(k,t))
            r->l = insertR(r->l, item);
      else
            r->r = insertR(r->r, item);
      (r->N)++;

      if(isRed(r->r) && !isRed(r->l)) r = rotateL(r);
      if(isRed(r->l) && isRed(r->l->l)) r = rotateR(r);
      if(isRed(r->l) && isRed(r->r) flipColors(r);
      r->N = r->l->N + r->r->N + 1;
      return r;
}
// Wrapper
void STinsert(Item item){
      h = insertR(h, item);
}
```