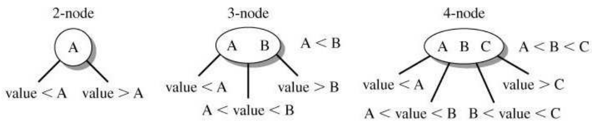
## 2-3-4 stromy a červeno čierne stromy

- pripájam len pracovný materiál, obsahujúci zväčša výňatky zverejnených dokumentov iných autorov, pretože momentálne nemám spracovanú prezentáciu k dispozícii.

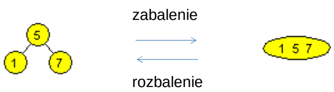
## 2-3-4 stromy

- 2-3-4 strom má 2-uzly, ktoré majú dvoch potomkov a jednu hodnotu, 3-uzly s tromi potomkami a dvomi hodnotami a 4-uzly so štyrmi potomkami a tromi hodnotami.




## 2-3-4 stromy

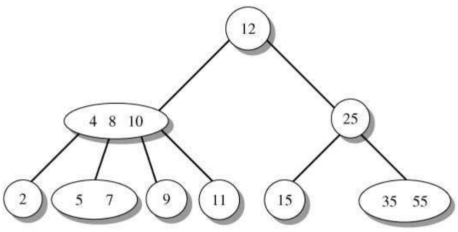
- Je to 2-3 strom, kde tri 2-uzly sú nahradené jedným 4-uzlom, čo zjednodušuje algoritmus vkladania a mazania hodnôt.




## 2-3-4 stromy searching

- Začni na koreni a porovnávaj hľadanú hodnotu s hodnotami v uzly. Ak nenastane zhoda, tak pokračuj v náležitom podstrome. Opakuj postup, až kým nenájdeš zhodu alebo nedosiahneš koniec podstromu.

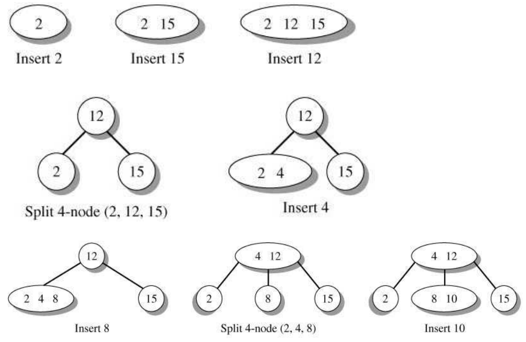
## 2-3-4 stromy searching



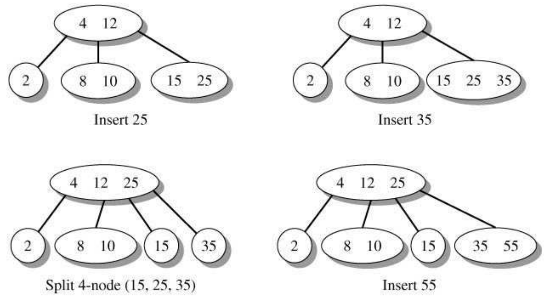

## 2-3-4 stromy insert

- Nájdi list, do ktorého sa bude hodnota vkladať.
- Počas hľadania, keď narazíš na 4-uzol, tak ho rozbaľ.
- Ak je list, do ktorého vkladáme 2-uzol alebo 3-uzol, tak vlož do lista.
- Ak je list 4-uzol, tak ho rozbaľ tak, že prostrednú hodnotu vlož do rodičovského uzla a vkladanú hodnotu vlož do príslušného lista. Miesto v rodičovskom uzle sa určite nájde, keďže sme pri ceste dole rozbalili všetky 4-uzly. Preto nemusíme rekurzívne postupovať hore do ďalších uzlov ako to bolo pri 2-3 stromoch.

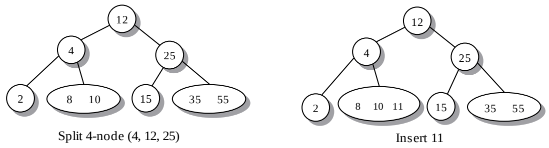
# 2-3-4 stromy insert




# 2-3-4 stromy insert




# 2-3-4 stromy insert

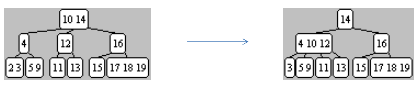



# 2-3-4 stromy delete

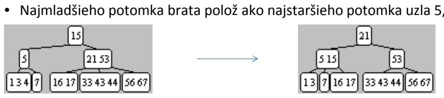
- Nájdi hodnotu, ktorá sa bude vymazávať a nahraď ju hodnotou inorder nasledovníka alebo predchodcu.
- Počas hľadania hodnoty a jeho nasledovníka zabaľuj 2-uzly do 3-uzlov alebo 4-uzlov. Takto zabezpečíme, že odoberaná hodnota bude v 3-uzle alebo 4-uzle.

# 2-3-4 stromy delete

- Prípady zbalenia:
  - Mažeme uzol 2, nachádzame sa na uzle 4, ktorého pravý brat je 2-uzol:
    - Spoj susedné hodnoty a deliacu hodnotu rodiča do jedného uzla (otec nemôže byť 2-uzol, iba aj je koreňom, vtedy sa nový uzol stáva kreňom)

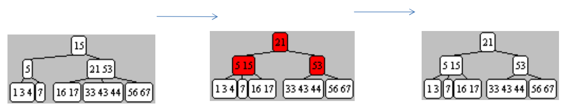


  - Mažeme uzol 4, sme na uzle 5, ktorého pravý brat je 3-uzol:
    - Deliacu hodnotu otca (15) vlož do uzla 5 a na jeho miesto vlož najmenšiu hodnotou brata.
    - Najmladšieho potomka brata polož ako najstaršieho potomka uzla 5,15



# 2-3-4 stromy delete

Delete 4



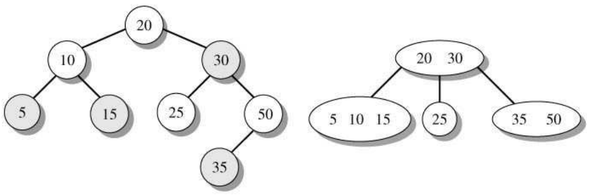
## 2-3-4 stromy zložitosť (max.)

- Vyhľadanie – int((log$_2$n)+1) = O(log n)
- Vkladanie = max. počet rozdeľovania uzlov * rozdeľovanie uzlov
  = int((log$_2$n)+1)*O(1)
  = O(log n)
- Vymazanie – O(log n)

## Red-Black Trees

- A red-black tree is a binary search tree in which each node has the color attribute BLACK or RED. It was designed as a representation of a 2-3-4 tree, using different color combinations to describe 3-nodes and 4-nodes.

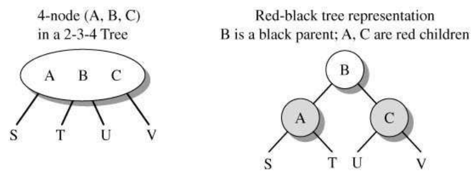
## Red-Black Trees (continued)




Red–black search tree and the equivalent 2-3-4 tree.

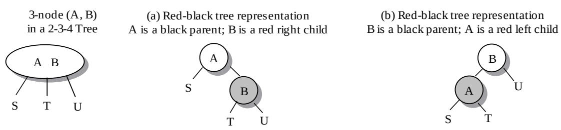
## Representing 2-3-4 Tree Nodes

- A 2-node is always black.
- A 4-node has the middle value as a black parent and the other values as red children.

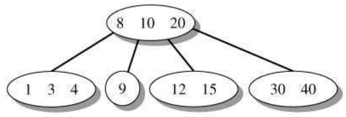



## Representing 2-3-4 Tree Nodes (concluded)

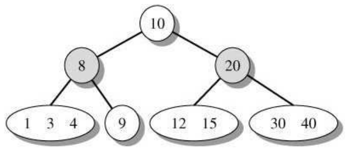
- Represent a 3-node with a BLACK parent and a smaller RED left child or with a BLACK parent and a larger RED right child.



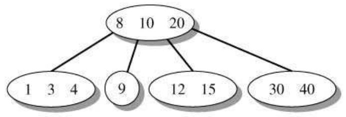

## Representing a 2-3-4 Tree as a Red-Black Tree



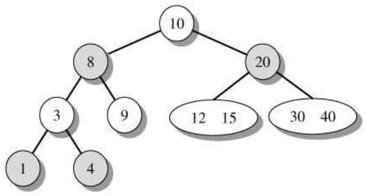

2-3-4 tree with 11 values and two levels.



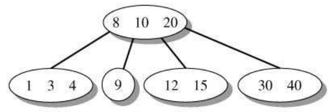
## Representing a 2-3-4 Tree as a Red-Black Tree (continued)



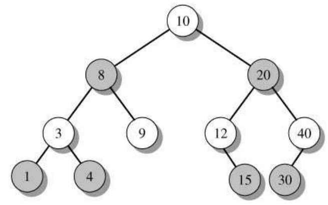
2-3-4 tree with 11 values and two levels.



## Representing a 2-3-4 Tree as a Red-Black Tree (concluded)



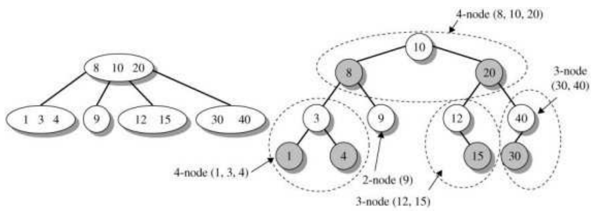
2-3-4 tree with 11 values and two levels.



## Properties of a Red-Black Tree

- These properties follow from the representation of a 2-3-4 tree as a red-black tree.
  - Root of red-black tree is always BLACK.
  - A RED parent never has a RED child. Thus in a red-black tree there are never two successive RED nodes.
  - Every path from the root to an empty subtree contains the same number of BLACK nodes. The number, called the *black height*, defines balance in a red-black tree.

## Properties of a Red-Black Tree (concluded)




The black-height of the red-black tree is 2. Each path from the root contains exactly two BLACK nodes.

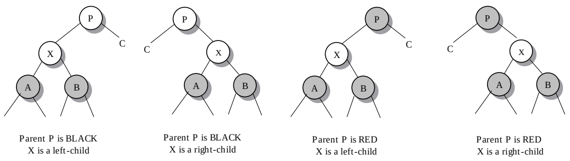
## Inserting a Node in a Red-Black Tree

- When scanning down a path to find the insertion location, split a 4-node (a BLACK parent with two RED children) by coloring the children BLACK and the parent RED. This is the red-black tree equivalent of splitting a 4-node in a 2-3-4 tree. Splitting 4-nodes may involve performing rotations and color changes.

## Inserting a Node in a Red-Black Tree (continued)

- Enter a new element into the tree as a RED leaf node.
- Inserting a RED node at the bottom of a tree may result in two successive RED nodes. When this occurs, use a rotation and recoloring to reorder the tree.
- Maintain the root as a BLACK node.

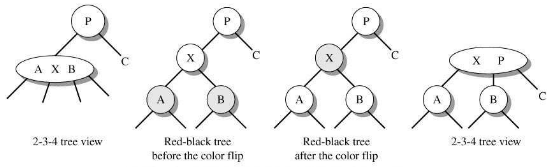
## Inserting a Node in a Red-Black Tree (continued)




Four Situations in the Splitting of a 4-Node

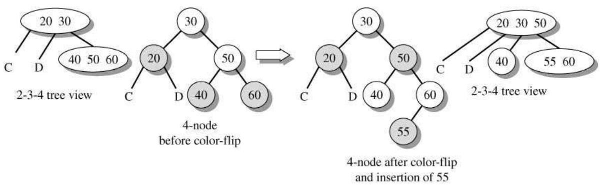
## Inserting a Node in a Red-Black Tree (continued)

- If the parent is BLACK, a color flip does not cause a rotation.

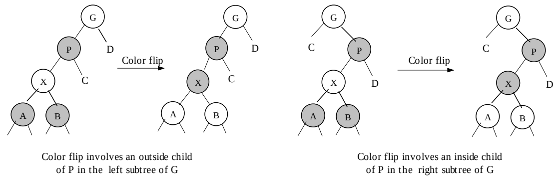



Splitting a 4-node that is the left child of a BLACK parent P.

## Inserting a Node in a Red-Black Tree (continued)




## Inserting a Node in a Red-Black Tree (continued)

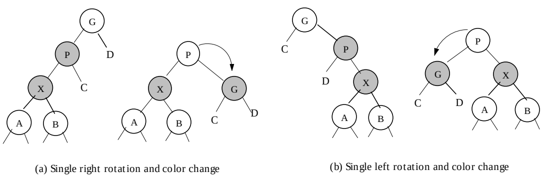



A color-flip of a 4-node with a RED parent leaves successive red nodes.

## Rebalancing with a Single Rotation

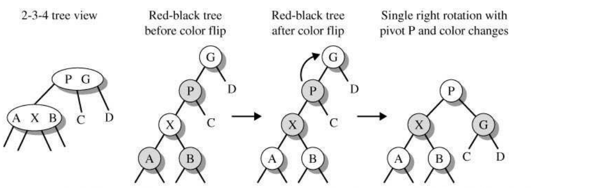
- When a 4-node is an outside child of its parent P and the color flip imbalances the tree, use a single rotation about node P. In the process, change the color for nodes P and G.

## Rebalancing with a Single Rotation (continued)




Single left and right rotations with color changes

# Rebalancing with a Single Rotation (concluded)

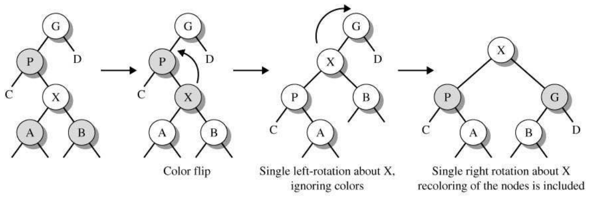



The before-and-after views of the 2-3-4 tree correspond to the splitting of the 4-node (A, X, B).

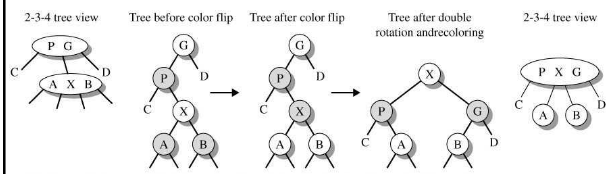
# Rebalancing with a Double Rotation

- A double rotation is used when the 4-node is an inside child of the parent and the color flip creates a color conflict. As with single rotations, double rotations are symmetric depending on whether the parent P is a left or a right child of G.

# Rebalancing with a Double Rotation (continued)



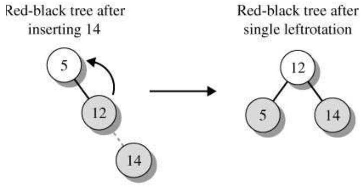

# Rebalancing with a Double Rotation (concluded)



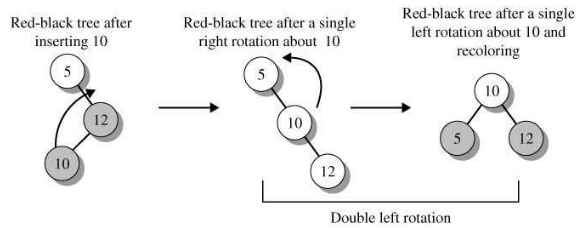

The figure displays a red-black tree before the splitting of a 4-node, after a color flip, and then after the double rotation. The before-and-after views of the 2-3-4 tree correspond to the splitting of the 4-node (A, X, B).

# Insertion at the Bottom of the Tree

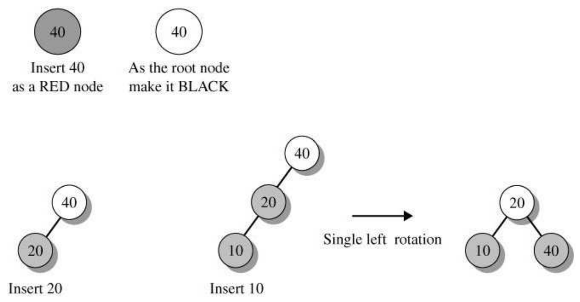
- Inserting a node into the tree with color RED may require a rotation.



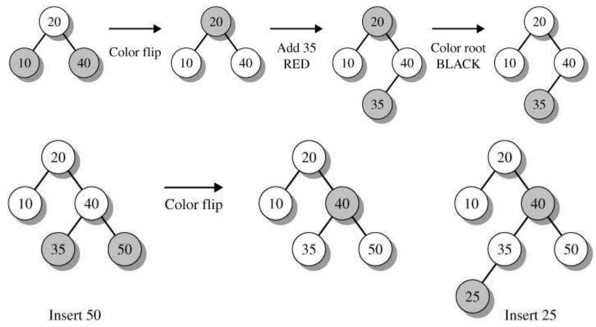

# Insertion at the Bottom of the Tree (concluded)



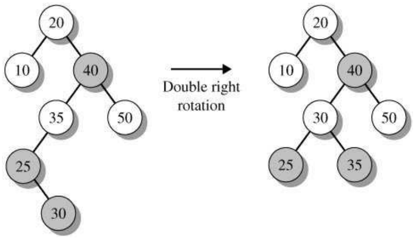
## Building a Red-Black Tree




## Building a Red-Black Tree (continued)



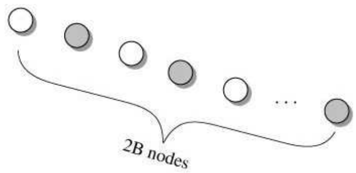

## Building a Red-Black Tree (concluded)




Insert 30

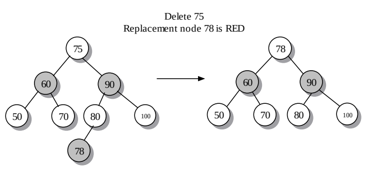
## Red-Black Tree Search Running Time

- The worst-case running time to search a red-black tree or insert an item is $O(\log_2 n)$.
  - The maximum length of a path in a red-black tree with black height B is 2*B-1.



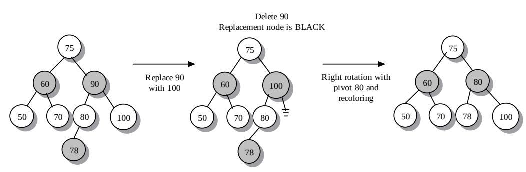

## Erasing a Node in a Red-Black Tree

- Erasing a node from an ordinary binary search tree is more difficult than inserting a node.
  - No further action necessary when replacement node is RED.




## Erasing a Node in a Red-Black Tree (concluded)

- Erasing a node from a red-black tree requires recoloring and rotations when the replacement node is BLACK.



## The RBTree Class

- RBTree is a generic class that implements the Collection interface and uses RBNode objects to create a red-black tree.

| left | nodeValue | parent | color | right |
|---|---|---|---|---|

RBNode

## The RBTree Class (continued)

| **class RBTree<T> implements Collection<T>** | **ds.util** |
|---|---|
| | **Constructor** |
| | RBTree()<br>Creates an empty red-black tree. |
| | **Methods** |
| String | **displayTree**(int maxCharacters)<br>Returns a string that gives a hierarchical view of the tree. An asterisk (*) marks red nodes. |
| void | **drawTree**(int maxCharacters)<br>Creates a single frame that gives a graphical display of the tree. Nodes are colored. |
| String | **drawTrees**(int maxCharacters)<br>Creates of the action of the function and any return value. |
| String | **toString**()<br>Returns a string that describes the elements in a comma-separated list enclosed in brackets. |

## The RBTree Class (concluded)

- The Web supplement contains the document "RBTree Class.pdf" which provides an expanded explanation of the RBTree class implementation. The document includes a discussion of the private section of the class and the algorithms for splitting a 4-node and performing a top-down insertion.