Haskell - A Search Tree Application

Functional Programming

Jens Egholm Pedersen and Anders Kalhauge

cphbusiness

Spring 2018

Outline

Search Trees

- Binary Search Tree
- Balanced Search Trees

Haskell IO

- Some basic examples
- Chapter 8 and 9

Assignment

Search Trees

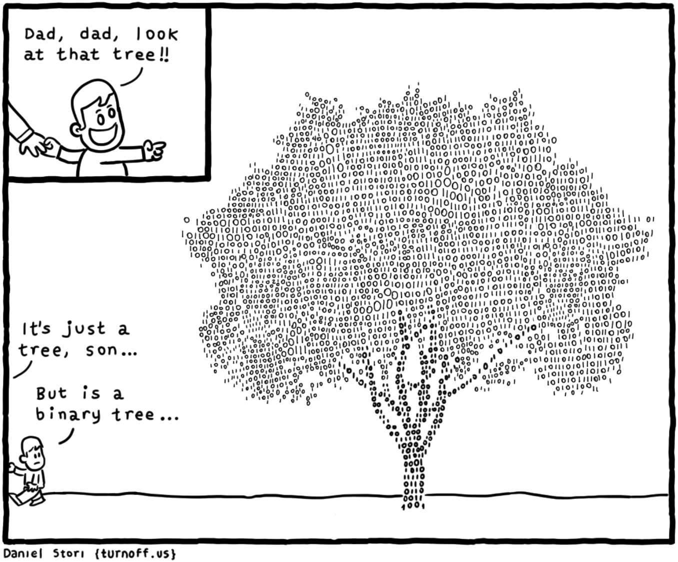
- Binary Search Tree
- Balanced Search Trees

Haskell IO

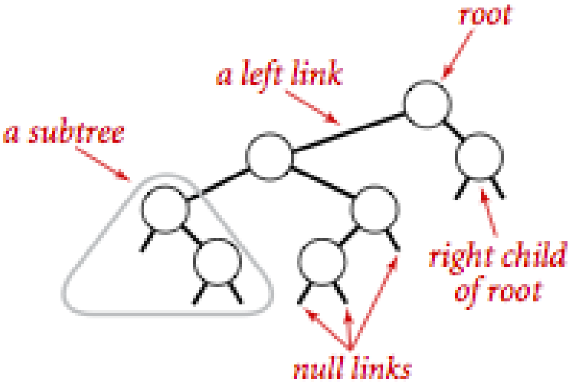
- Some basic examples
- Chapter 8 and 9

Assignment

Binary Trees

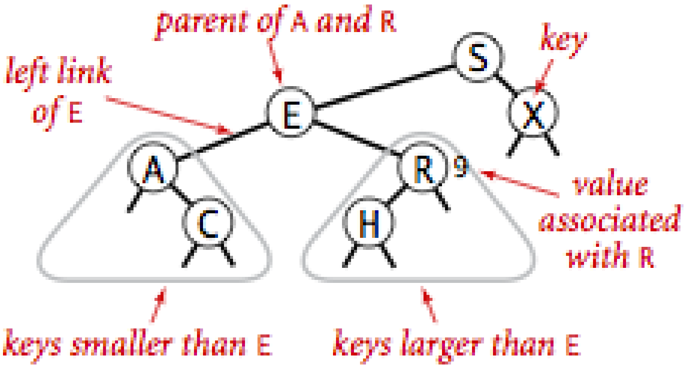


Binary Trees



Anatomy of a binary tree

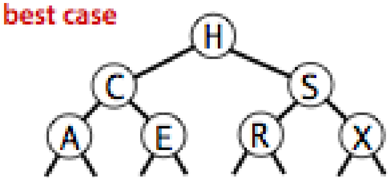
Binary Search Trees



Anatomy of a binary search tree

Binary Search Trees

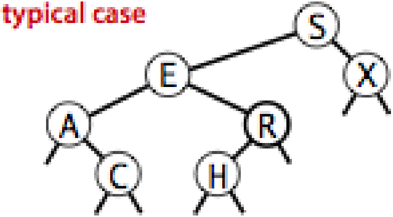
Analysis



$$O(\log n)$$

Binary Search Trees

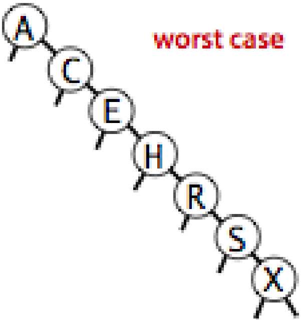
Analysis



$$O(\log n)$$

Binary Search Trees

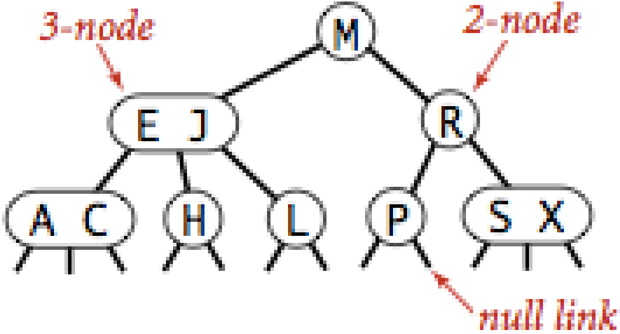
Analysis



$$O(n)$$

2-3 Search trees

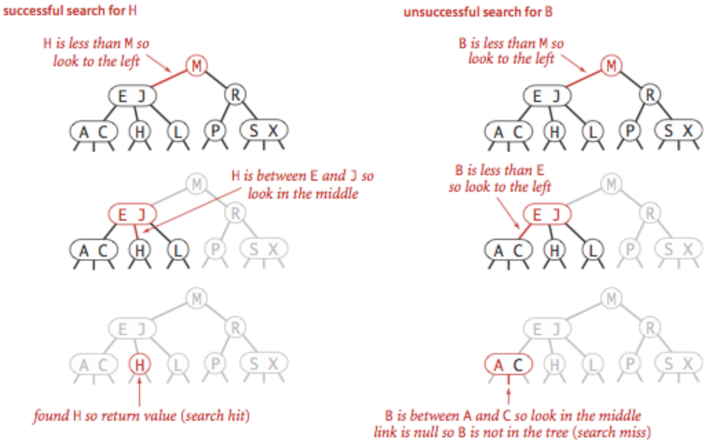
Anatomy



Anatomy of a 2-3 search tree

2-3 Search trees

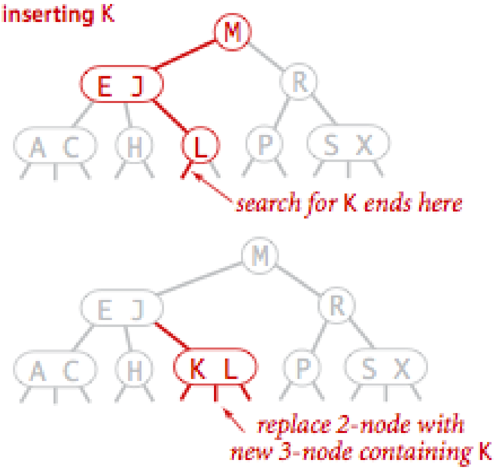
Searching



successful search for H
H is less than M so look to the left
M
E J
R
A C
H
L
P
S X
H is between E and J so look in the middle
found H so return value (search hit)
unsuccessful search for B
B is less than M so look to the left
B is less than E so look to the left
B is between A and C so look in the middle link is null so B is not in the tree (search miss)

Search hit (left) and search miss (right) in a 2-3 tree

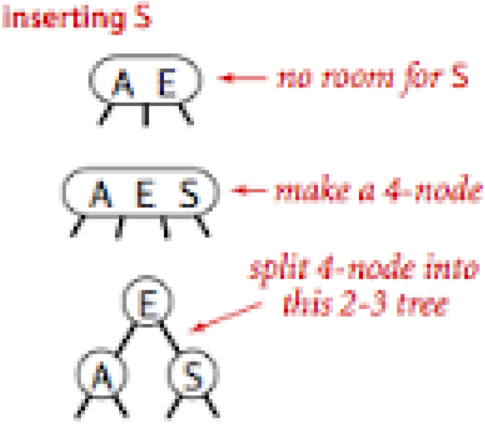
Inserting



Insert into a 2-node

2-3 Search trees

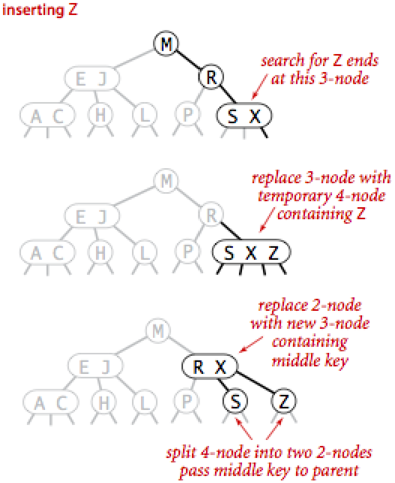
Inserting



Insert into a single 3-node

2-3 Search trees

Inserting



inserting Z

search for Z ends at this 3-node

replace 3-node with temporary 4-node containing Z

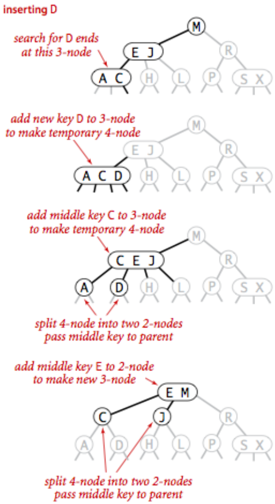
replace 2-node with new 3-node containing middle key

split 4-node into two 2-nodes pass middle key to parent

Insert into a 3-node whose parent is a 2-node

2-3 Search trees

Inserting


inserting D

search for D ends at this 3-node

add new key D to 3-node to make temporary 4-node

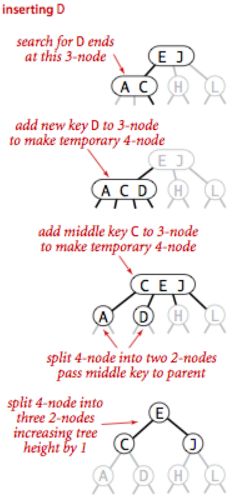
add middle key C to 3-node to make temporary 4-node

split 4-node into two 2-nodes pass middle key to parent

add middle key E to 2-node to make new 3-node

split 4-node into two 2-nodes pass middle key to parent

Insert into a 3-node whose parent is a 3-node


inserting D

search for D ends at this 3-node

add new key D to 3-node to make temporary 4-node

add middle key C to 3-node to make temporary 4-node

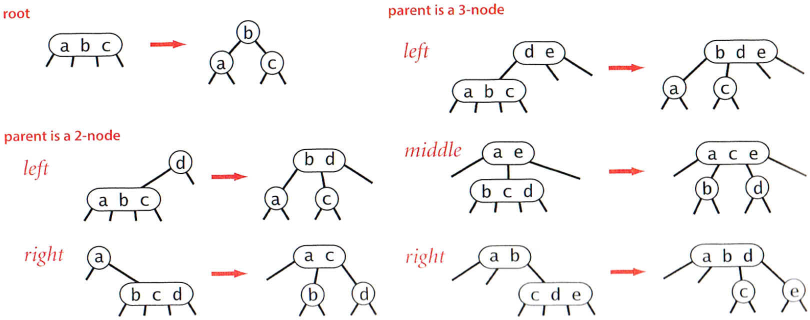
split 4-node into two 2-nodes pass middle key to parent

split 4-node into three 2-nodes increasing tree height by 1

Splitting the root

2-3 Search trees

Summing up



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

Search Trees

- Binary Search Tree
- Balanced Search Trees

Haskell IO

- Some basic examples
- Chapter 8 and 9

Assignment

Read and Write to console

```haskell
main :: IO ()
main = do
  putStrLn "Hi,␣what's␣your␣name?"
  name <- getLine
  putStrLn ("Hello␣"++name)
```

In GHCi

```
ghci> :t getLine
getLine :: IO String
ghci> :t putStrLn
putStrLn :: String -> IO ()
ghci>
```

Show and Read

```
main :: IO ()
main = do
  putStrLn "Hi,␣what's␣your␣age?"
  line <- getLine
  let age = (read line :: Int)
  putStrLn ((show age)++"␣years")
```

Reading lines from file

```haskell
import System.IO

main :: IO ()
main = do
  withFile "numbers.txt" ReadMode (\handle -> do
    contents <- hGetContents handle
    let numberLines = lines contents
    putStr (contents++(numberLines!!2)++"\n")
    )
```

Define function

```
fact :: Integer -> Integer
fact 0 = 1
fact n = n * fact (n - 1)

main :: IO ()
main = do
  putStrLn ("50!␣=␣"++(show (fact 50)))
```

```
50! = 30414093201713378043612608166064768844377641568
      960512000000000000
```

Mutable state

```haskell
import Data.IORef

data Tree d = Empty | Node d (Tree d) (Tree d)

insert :: Ord d => Tree d -> d -> Tree d
insert tree value = ...

main :: IO ()
main = do
  treeRef <- newIORef Empty
  writeIORef treeRef (Node 7 Empty Empty)
  tree <- readIORef treeRef
  putStrLn (show tree)
  writeIORef treeRef (insert tree 8)
```

More information on Haskell IO

Consult chapter 8 and 9 in Learn You a Haskell for Great Good!

Search Trees

- Binary Search Tree
- Balanced Search Trees

Haskell IO

- Some basic examples
- Chapter 8 and 9

Assignment

Assignment

Create a Haskell application that:

- □ Implements a balanced search tree[1] with functions for finding items, inserting items, and traversing the tree.
- □ Implements a console interface with:
  - □ Search item
  - □ Insert item
  - □ List all items in order
  - □ Load items from text file

Start with integer numbers as items, expand with any item deferring `Ord`

Can be done individually or in groups.

[1] or at least a binary search tree