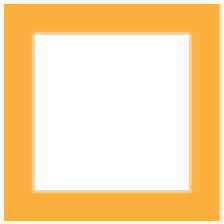
Searching Lists
Algorithms and Datastructures

Marjahan Begum and Anders Kalhauge



cphbusiness

Spring 2017

Outline

cphbusiness
COPENHAGEN BUSINESS ACADEMY

Symbol Tables continued

- Hashed
- Search Trees
- Balanced Search Trees

Symbol Tables continued

- Hashed
- Search Trees
- Balanced Search Trees

cphbusiness
COPENHAGEN BUSINESS ACADEMY

Problem

We want to access a symbol table with n keys, using a key $k \in K$ as was it an index to an array with n elements.

Perfect solution

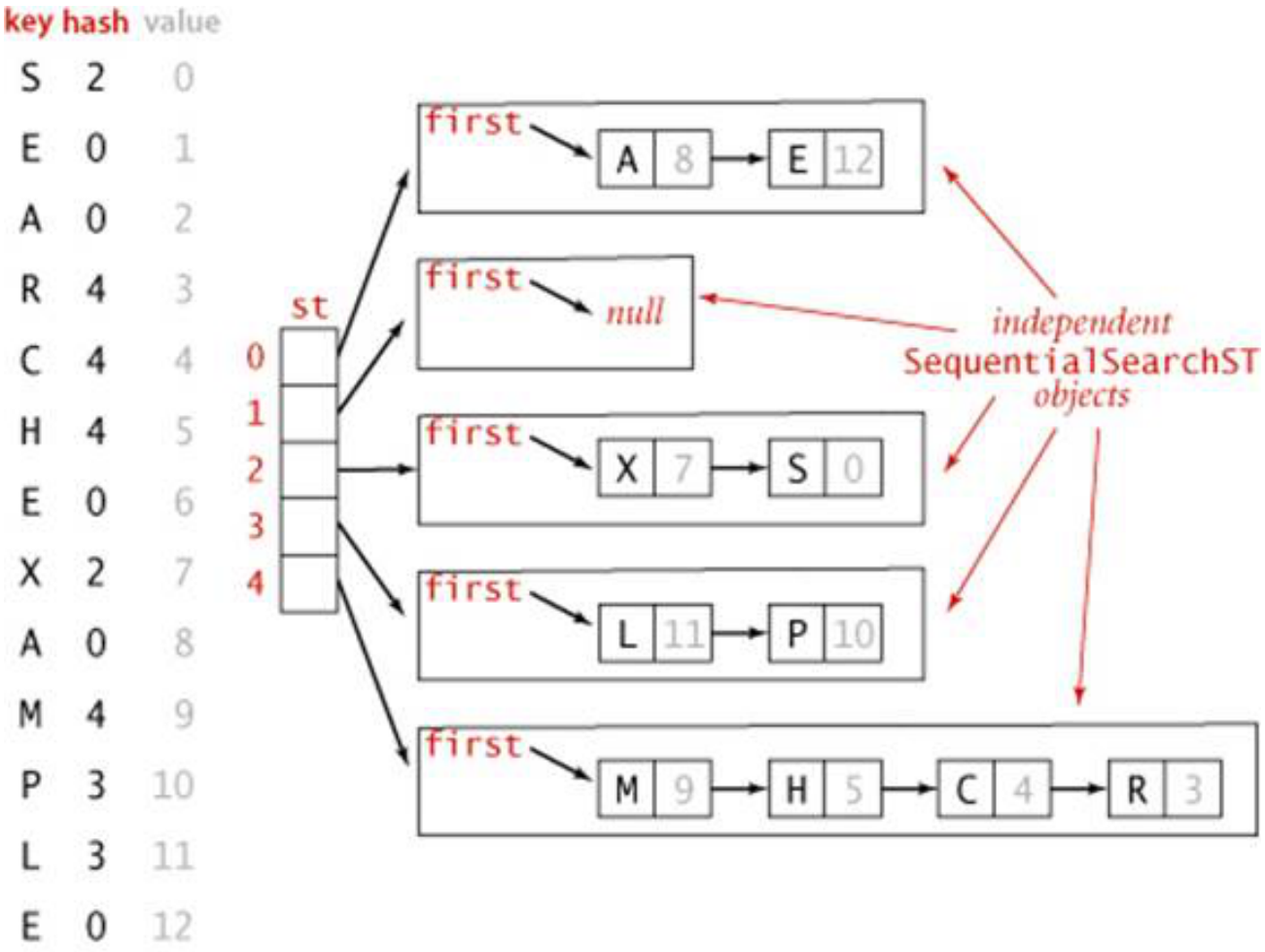
By **magic** we find a mapping from the keys K to $\{0, \ldots, n-1\}$.
We call it the hash function: $h : K \to \{0, \ldots, n-1\}$.

Realistic Solution

Find a hash function that maps the keys K uniformly over $\{0, \ldots, m-1\}$ where $m \approx n$.

Hashed Symbol Tables

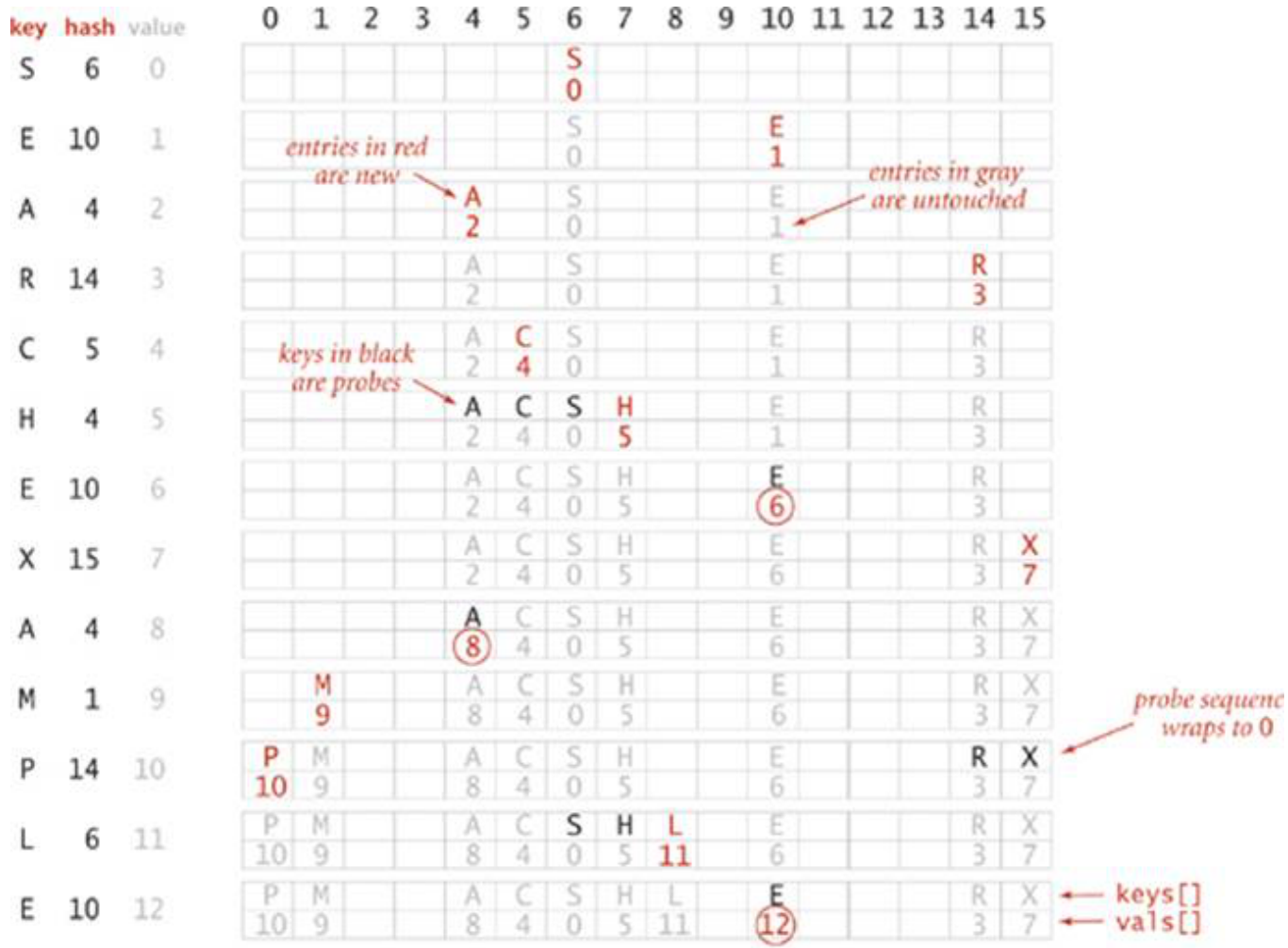
Chained hashing



Hashing with separate chaining for standard indexing client

Chained hashing

Linear probing



Trace of linear-probing ST implementation for standard indexing client

With:

```
@FunctionalInterface
public interface HashFunction {
  int function(String key);
  }
```

You can:

```
public static test(String[] keys, HashFunction hash) {
  for (String key : keys)
      System.out.println(hash.function(key));
  }
public static void main(String... args) {
  String[] words =
      FileUtility.toStringArray("sp.txt", "[^A-Za-z]");
  test(words, k -> k.length());
  }
```

Exercise 1 - Hash function

cphbusiness
COPENHAGEN BUSINESS ACADEMY

Experiment with various hash implementation of hash functions. All functions should return a number between 0 and 31.

The signature of the hash function should be:

```
@FunctionalInterface
public interface HashFunction {
  int function(String key);
  }
```

Create hash functions that uses:

- the first character code
- the last character code
- the sum of character codes
- `Strings hashCode()` method

You can use `n%32` to fit `n` into the range of $0...31$

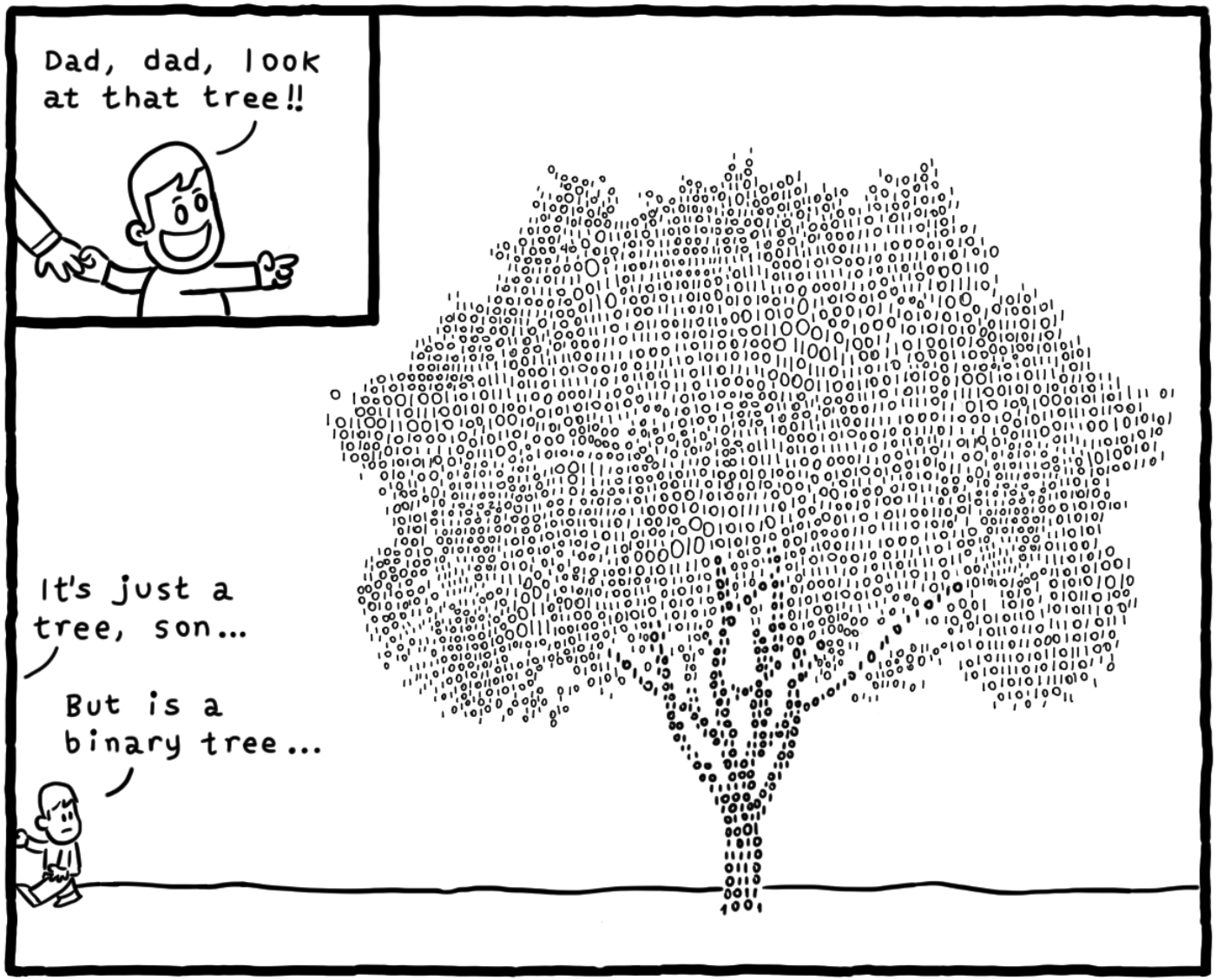
Make a histogram of each hash function

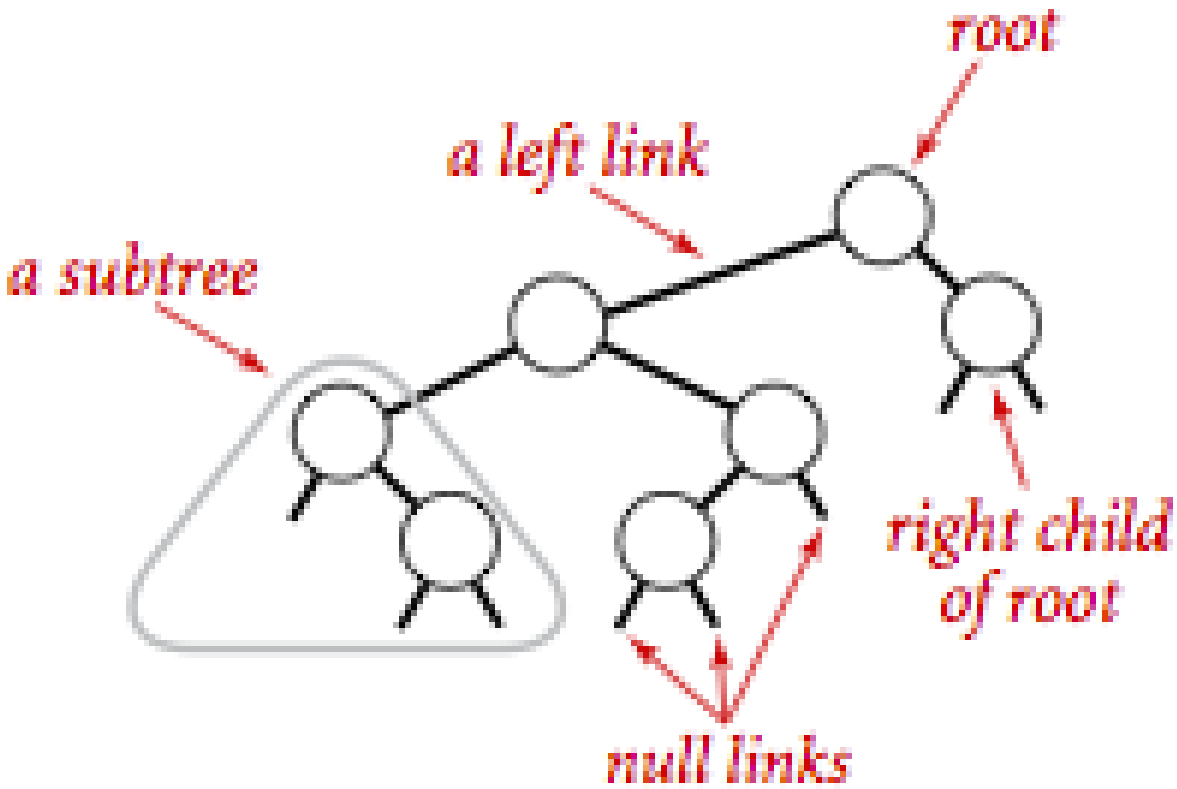
Other hashing aspects

cphbusiness
COPENHAGEN BUSINESS ACADEMY

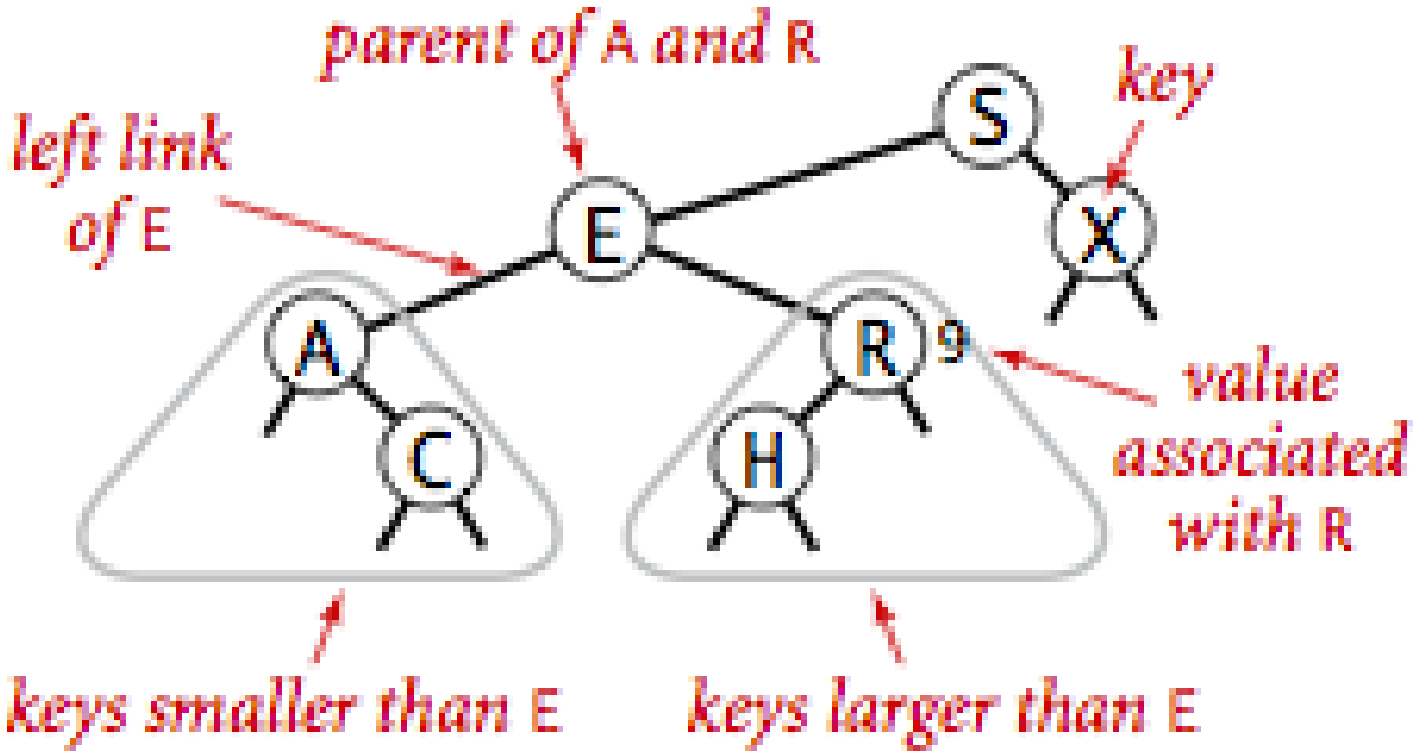
- Universal hashing
- Static tables
- Perfect hashing

Binary Trees





Anatomy of a binary tree



Anatomy of a binary search tree

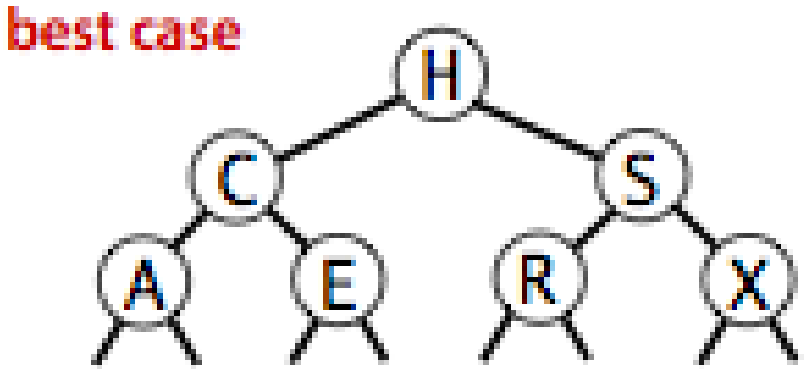
Exercise 3 - BST Structure

cphbusiness
COPENHAGEN BUSINESS ACADEMY

Implement the structure for a binary search tree

cphbusiness
COPENHAGEN BUSINESS ACADEMY

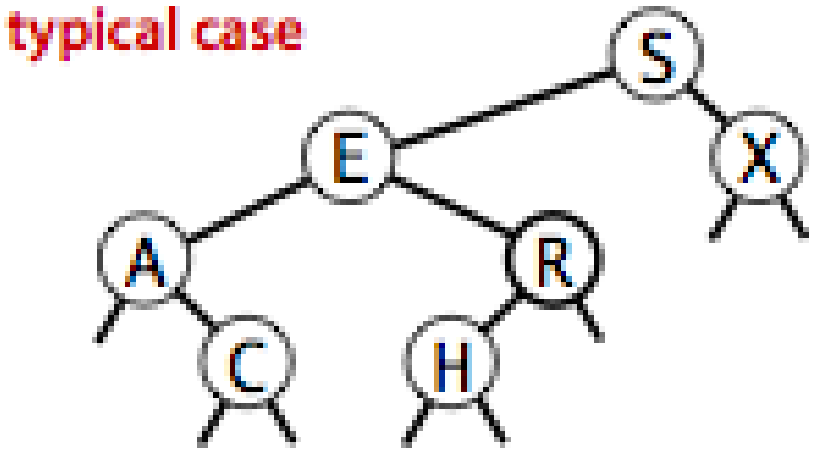
Analysis



$$O(\log n)$$

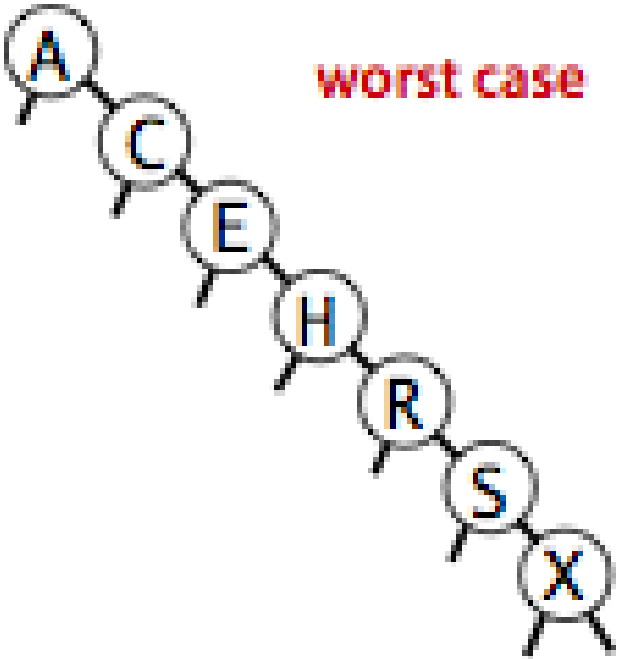
cphbusiness
COPENHAGEN BUSINESS ACADEMY

Analysis



$$O(\log n)$$

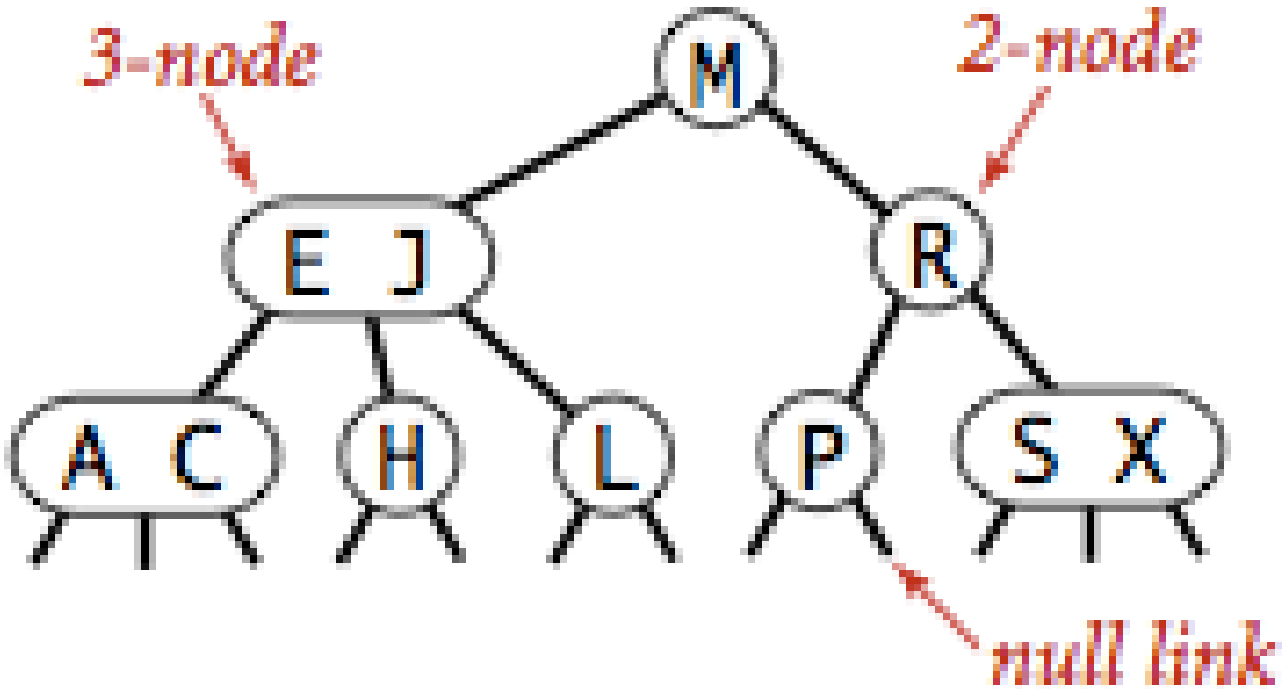
Analysis



$$O(n)$$

cphbusiness
COPENHAGEN BUSINESS ACADEMY

Anatomy



Anatomy of a 2-3 search tree

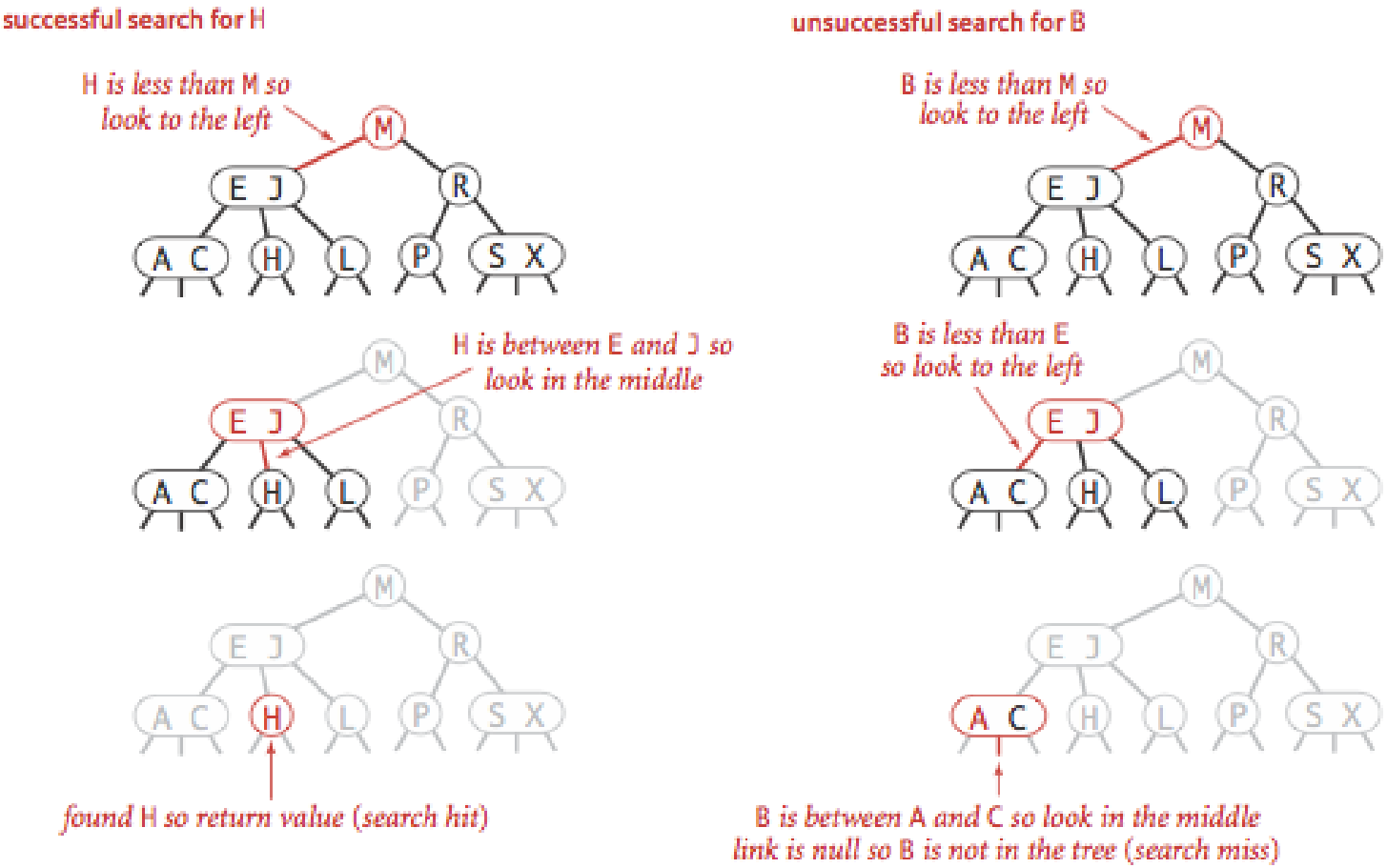
Exercise 3 - BST Structure

cphbusiness
COPENHAGEN BUSINESS ACADEMY

Implement the structure for a binary search tree

cphbusiness
COPENHAGEN BUSINESS ACADEMY

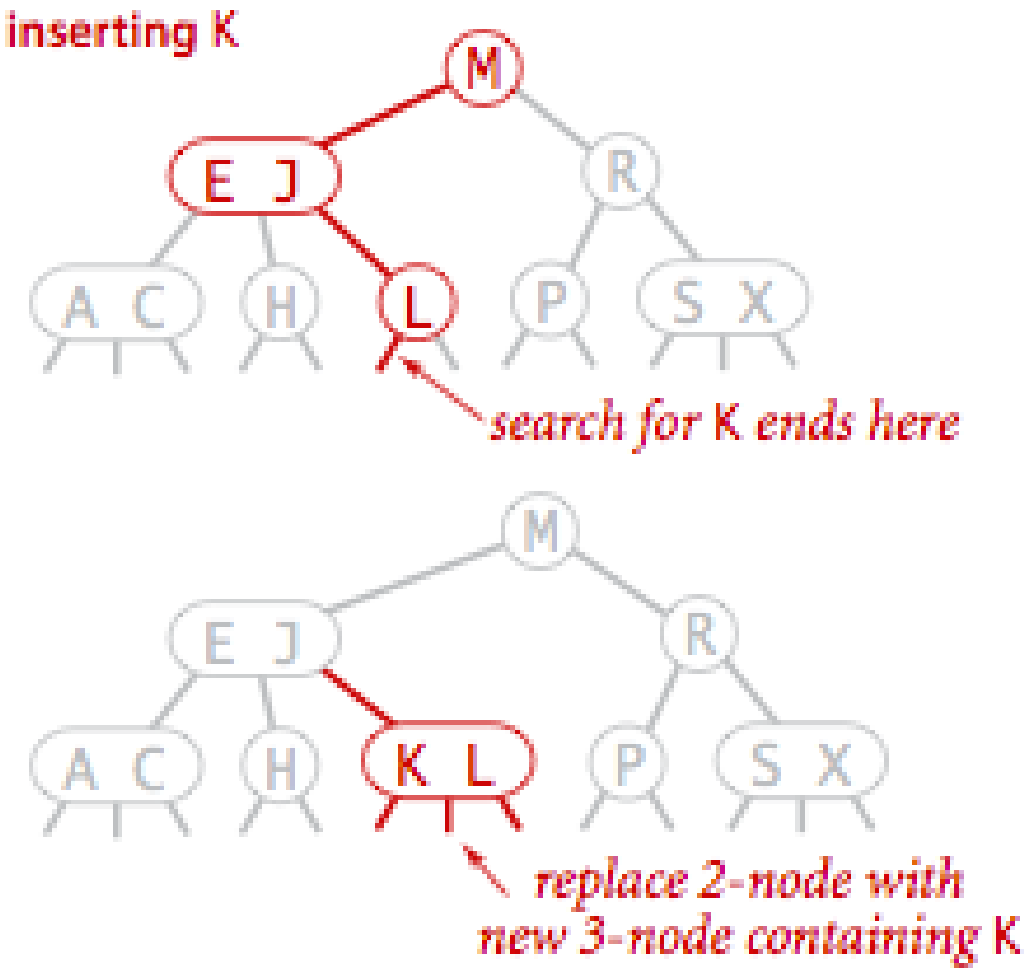
Searching



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

cphbusiness
COPENHAGEN BUSINESS ACADEMY

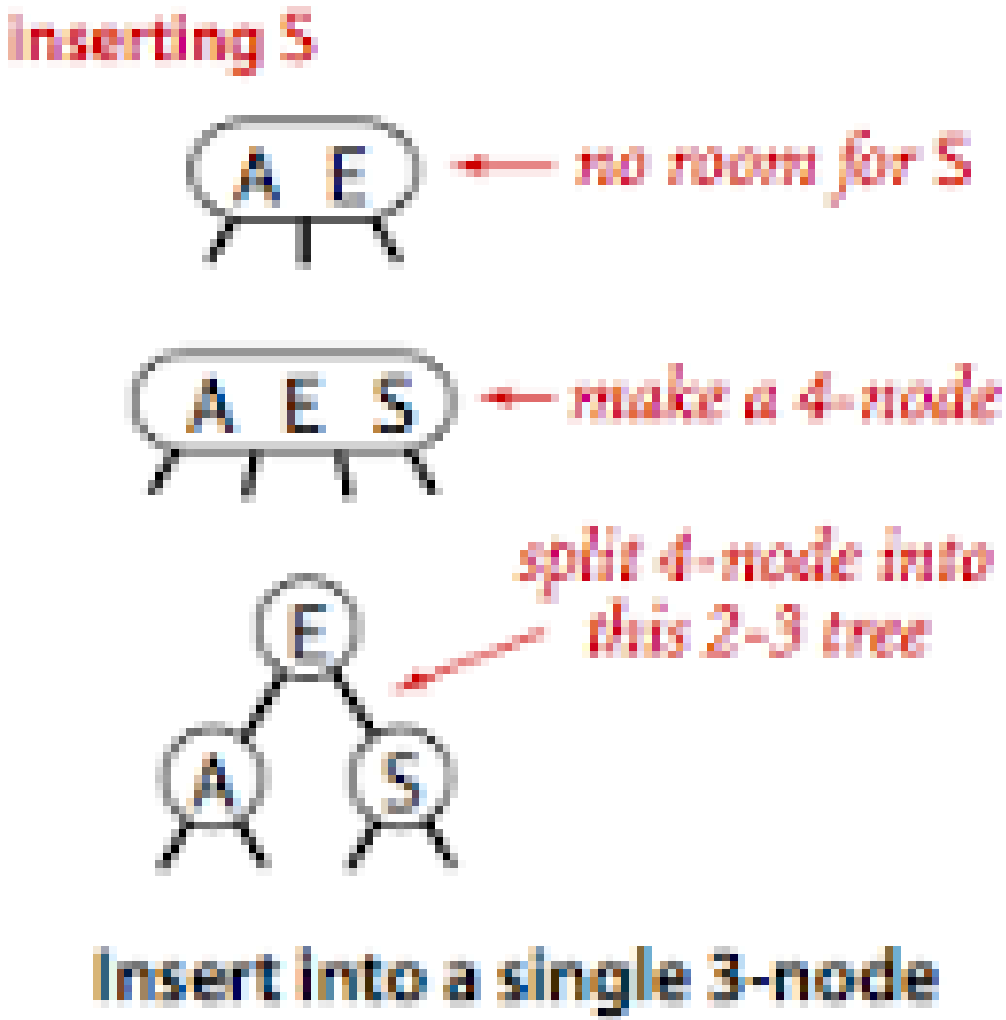
Inserting



Insert into a 2-node

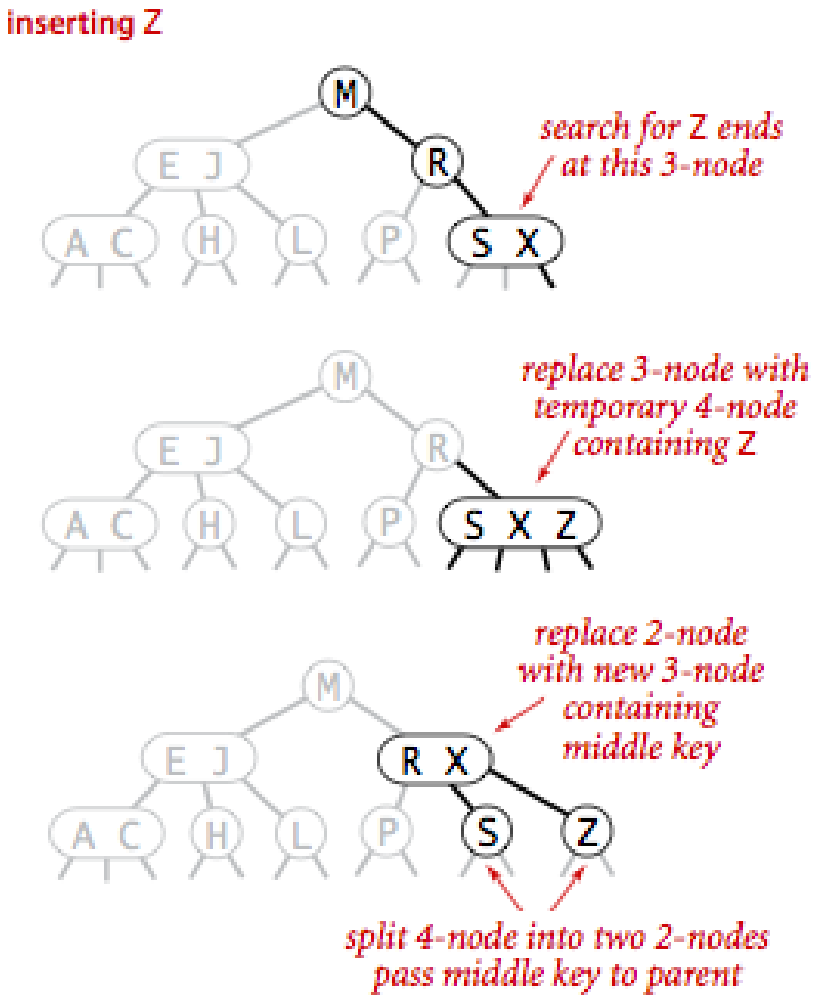
cphbusiness
COPENHAGEN BUSINESS ACADEMY

Inserting



cphbusiness
COPENHAGEN BUSINESS ACADEMY

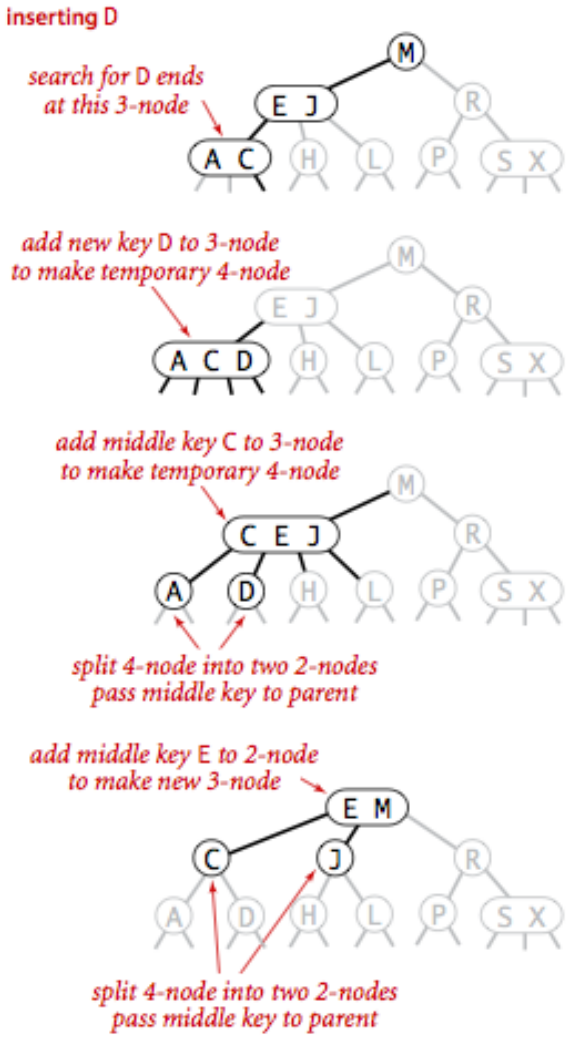
Inserting



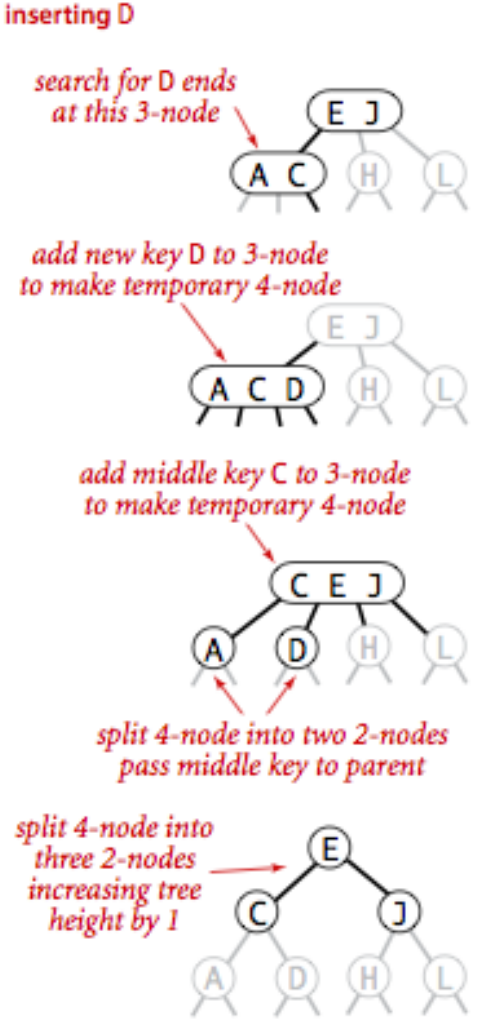
Insert into a 3-node whose parent is a 2-node

cphbusiness
COPENHAGEN BUSINESS ACADEMY

Inserting

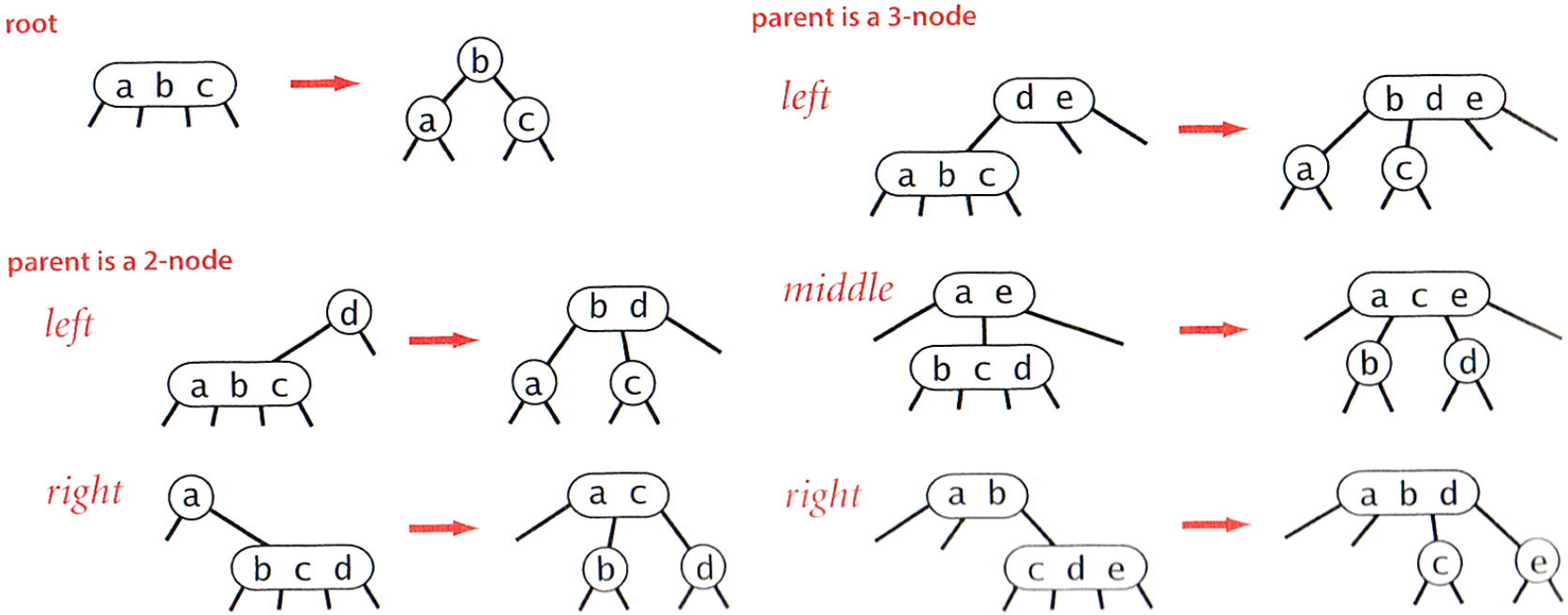


Insert into a 3-node whose parent is a 3-node



Splitting the root

Summing up



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