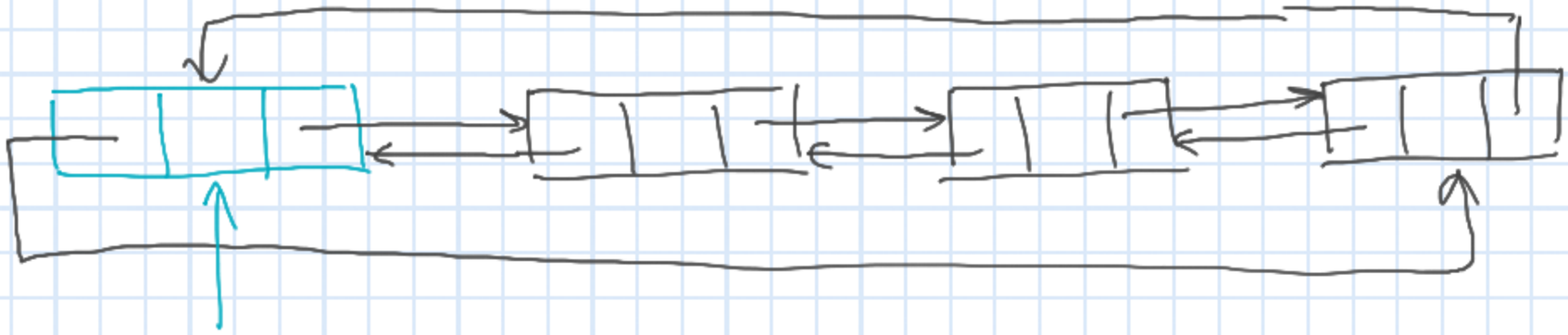
November 12, 2020

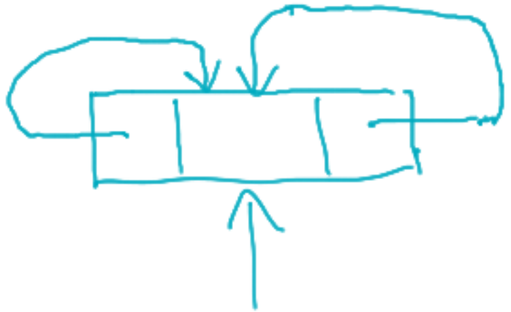
Double linked lists with a sentinel



sentinel

key tag is undefined
prev pointer points to the last element
next pointer points to the first element

Empty list



Sentinel → next = Sentinel
Sentinel → prev = Sentinel

Search, Insert, Delete

```
Search ( Sentinel[L], k )
x := Sentinel[L] → next
while  x ≠ Sentinel[L]  AND  x → key ≠ k  do
       x := x → next
return x
```

Insert (Sentinel[L], x)

x will be the new first element

x -> next := Sentilel[L] -> next

Sentinel[L] -> next -> prev := x

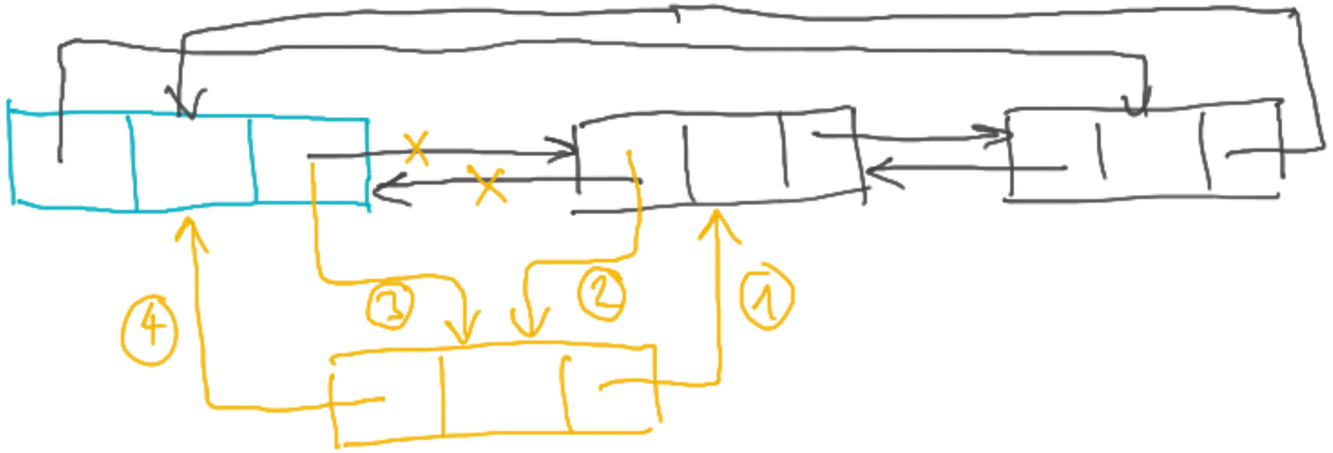
Sentinel[L] -> next := x

x -> prev := Sentinel[L]

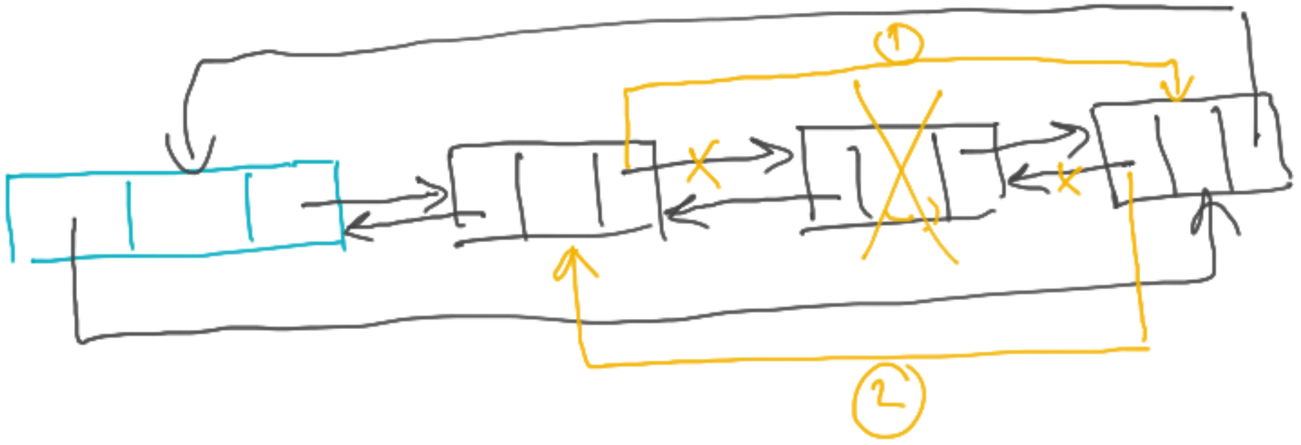
Delete (Sentinel[L], x)

x -> prev -> next := x -> next

x -> next -> prev := x -> prev



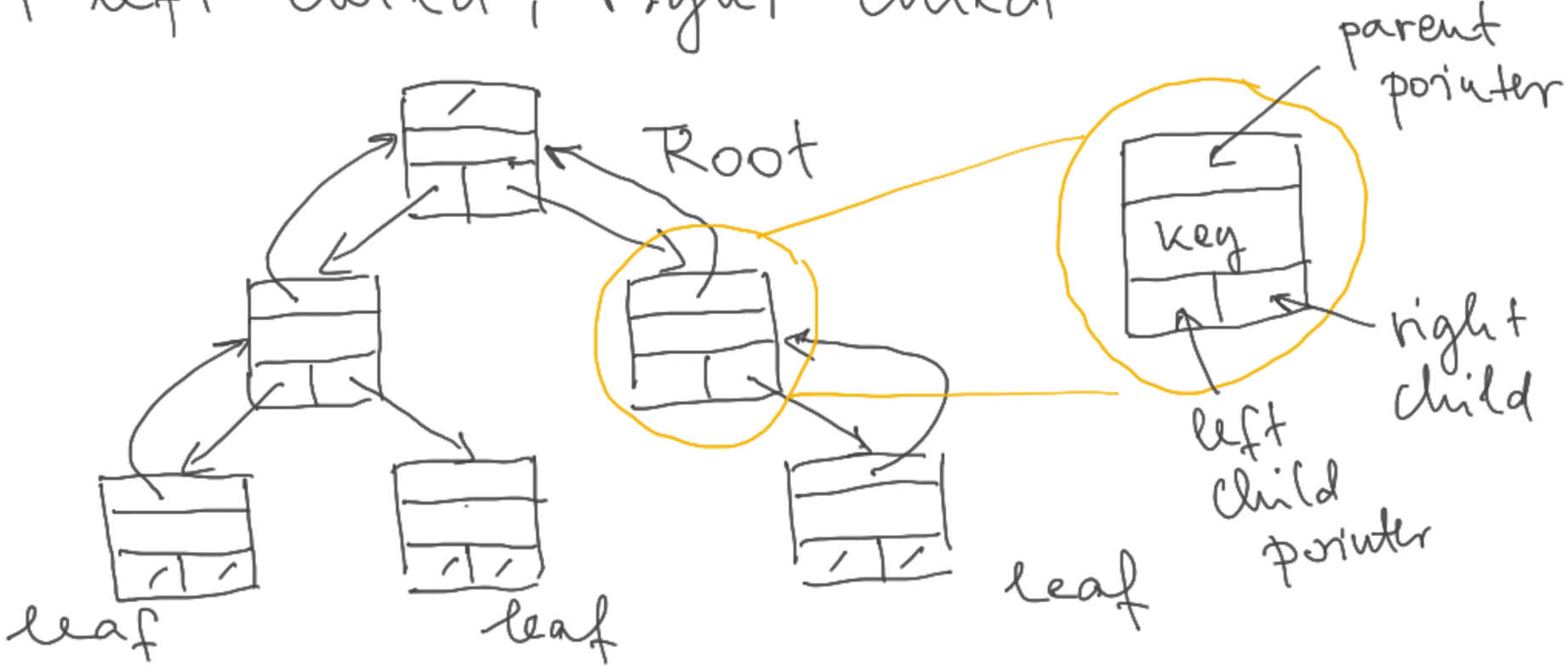
Insert



Delete

Binary search trees

Data structure where data are organized in a binary tree structure using pointers: parent, left child, right child



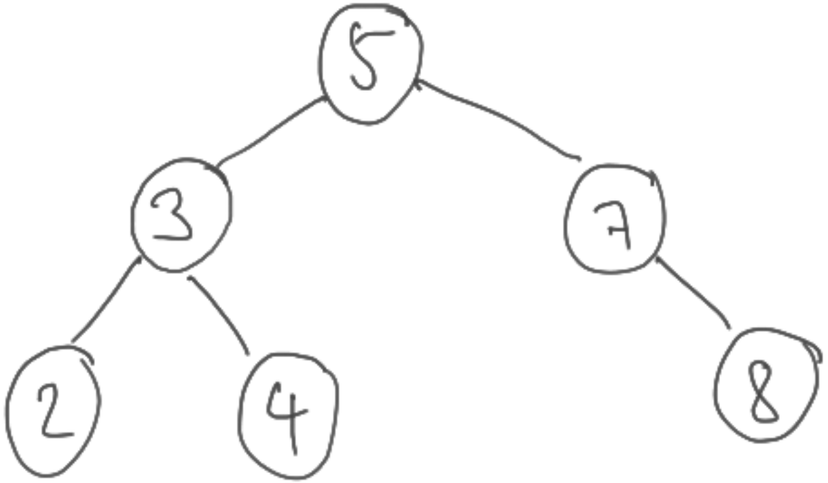
Binary search tree property

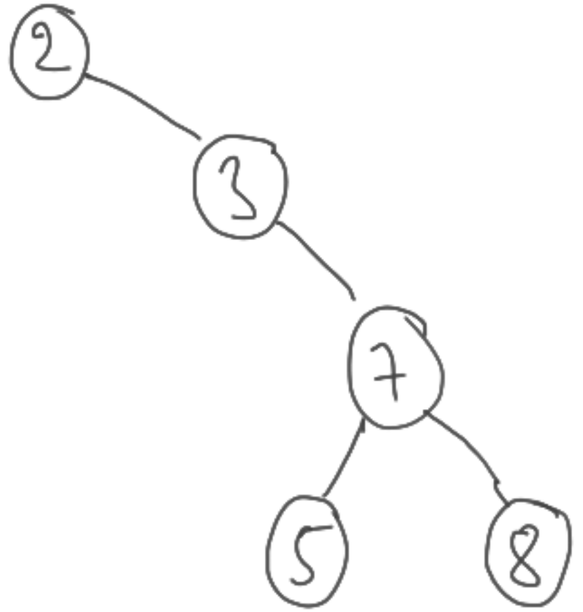
Let x be an arbitrary vertex of a binary serch tree. Now if y is a vertex of the left subtree of x and z is a vertex of the right subtree of x then

$$y \to key < x \to key \leq z \to key$$

(if duplication is not allowed we can write $<$ instead of $\leq$)

Examples





two binary search trees

Traversals

in binary (not necessarily) search trees

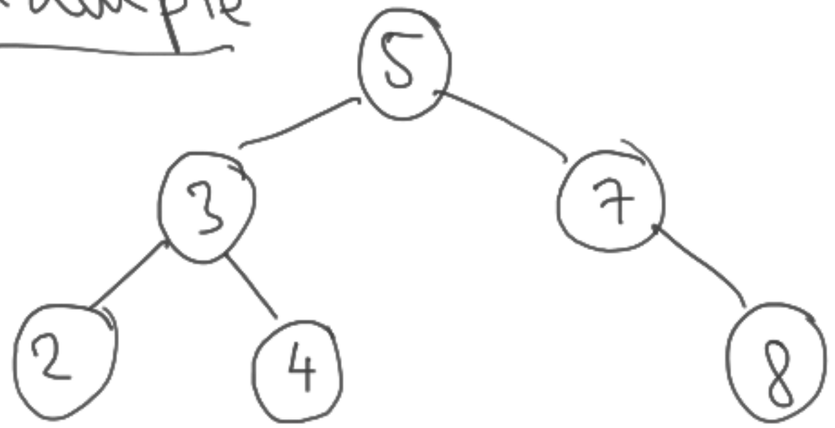
- preorder
- inorder
- postorder

```
Preorder (x)          ← root
if x ≠ nil then
    print x → key
    Preorder (x → left)
    Preorder (x → right)
Inorder (x)           ← root
if x ≠ nil then
    Inorder (x → left)
    print x → key
    Inorder (x → right)
```

```
Postorder(x)          <-- root
if x ≠ nil then
    Postorder(x->left)
    Postorder(x->right)
    Print x->key
```

For binary search trees the inorder traversal scan (list) the keys in increasing order

Example



Execute the inorder traversal for this BST

IT (5)

IT (3)

IT (2)

IT (2 → left = nil)

Print 2

IT (2 → right = nil)

print [3]
IT (④)
IT (④ → left = nil)
print [4]
IT (④ → right = nil)
print [5]
IT (⑦)
IT (⑦ → left = nil)
print [7]
IT (⑦ → right = ⑧)

```
i    IT (⑧ → left = nil)
     print [8]
     IT (⑧ → right = nil)
```

This is a linear time algorithm!

Operations for BST

Search(k), Min, Max, Prev(x), Next(x)

Prev(x), Next(x) according to the inorder traversal!

Insert(k), Delete(k)

Search

Idea:

Compare the key we want to find with the root's key

1. they are equal: ☺
2. the root's key is bigger:
   continue the search recursively in the left subtree

③ the root's key is smaller:
continue the search recursively
in the right subtree

Be careful:

→ successful

→ unsuccessful: in this case you try to go into an empty subtree

This is a linear recursion so we can implement it by a while loop instead of a recursive way

```
Search ( root[T], k )
x := root[T]
while  x ≠ nil and k ≠ x → key do
    if  k < x → key
        then   x := x → left
        else   x := x → right
return x
```

Successful search:

x is a pointer to the object containing k as key

Unseccessful search:

x = nil

Cost of search:

O(h) where h is the height of the tree (length of a longest root-leaf path)

Efficient :

$$h = O(\log n)$$