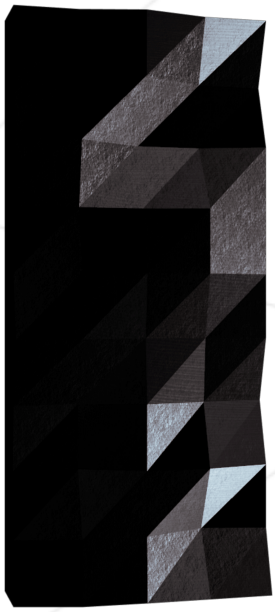

# Go Piscine

## Go 10

*Summary: THIS document is the subject for the Go 10 module of the Go Piscine @ 42Tokyo.*

# Contents


| | | |
|---|---|---|
| I | Instructions | 2 |
| II | Exercise 00 : btreeinsertdata | 3 |
| III | Exercise 01 : btreeapplyinorder | 5 |
| IV | Exercise 02 : btreeapplypreorder | 7 |
| V | Exercise 03 : btreesearchitem | 9 |
| VI | Exercise 04 : btreelevelcount | 11 |
| VII | Exercise 05 : btreeisbinary | 13 |
| VIII | Exercise 06 : btreeapplybylevel | 15 |
| IX | Exercise 07 : btreemax | 17 |
| X | Exercise 08 : btreemin | 19 |
| XI | Exercise 09 : btreetransplant | 21 |
| XII | Exercise 10 : btreedeletenode | 23 |

# Chapter I

# Instructions

- Only this page will serve as reference; do not trust rumors.
- Watch out! This document could potentially change up to an hour before submission.
- These exercises are carefully laid out by order of difficulty - from easiest to hardest. We `will not` take into account a successfully completed harder exercise if an easier one is not perfectly functional.
- Make sure you have the appropriate permissions on your files and directories.
- You have to follow the submission procedures for every exercise.
- Your exercises will be checked and graded by your fellow classmates.
- You cannot leave any additional file in your directory than those specified in the subject.
- Got a question? Ask your peer on the right. Otherwise, try your peer on the left.
- Your reference guide is called `Google / man / the Internet / ...`.
- Examine the examples thoroughly. They could very well call for details that are not explicitly mentioned in the subject...
- If no other explicit information is displayed, you must use the latest versions of Go.
- Your turn-in directory for each exercise should look something like this:

```
ex[XX]
|-- main.go
|-- vendor
   |-- ft
      |-- printrune.go
   |-- piscine
      |-- [excercisename].go
```

# Chapter II

# Exercise 00 : btreeinsertdata

|  | Exercise 00 |
|---|---|
| btreeinsertdata | |
| Turn-in directory : $ex00/$ | |
| Files to turn in : `*` | |
| Allowed packages : `fmt` | |
| Allowed builtin functions : `None` | |

Write a function that inserts new data in a `binary search tree` following the special properties of a `binary search trees`.

- Excepted function

```
type TreeNode struct {
        Left, Right, Parent *TreeNode
        Data                string
}

func BTreeInsertData(root *TreeNode, data string) *TreeNode {

}
```

- Usage

```
package main

import (
        "fmt"
        "piscine"
)

func main() {
        root := &piscine.TreeNode{Data: "4"}
        piscine.BTreeInsertData(root, "1")
        piscine.BTreeInsertData(root, "7")
        piscine.BTreeInsertData(root, "5")
        fmt.Println(root.Left.Data)
        fmt.Println(root.Data)
        fmt.Println(root.Right.Left.Data)
        fmt.Println(root.Right.Data)

}
```

- And its output:

```
$ go mod init ex00
$ go run .
1
4
5
7
$
```

# Chapter III

# Exercise 01 : btreeapplyinorder

|  | Exercise 01 |
|---|---|
| btreeapplyinorder | |
| Turn-in directory : $ex01/$ | |
| Files to turn in : `*` | |
| Allowed packages : `fmt` | |
| Allowed builtin functions : `None` | |

Write a function that applies a given function `f`, in order, to each element in the tree.

- Excepted function

```
func BTreeApplyInorder(root *TreeNode, f func(...interface{}) (int, error)) {

}
```

- Usage

```
package main

import (
        "fmt"
        "piscine"
)

func main() {
        root := &piscine.TreeNode{Data: "4"}
        piscine.BTreeInsertData(root, "1")
        piscine.BTreeInsertData(root, "7")
        piscine.BTreeInsertData(root, "5")
        piscine.BTreeApplyInorder(root, fmt.Println)

}
```

- And its output:

```
$ go mod init ex01
$ go run .
1
4
5
7
$
```

# Chapter IV

# Exercise 02 : btreeapplypreorder

|  | Exercise 02 |
|---|---|
| btreeapplypreorder | |
| Turn-in directory : $ex02/$ | |
| Files to turn in : `*` | |
| Allowed packages : `fmt` | |
| Allowed builtin functions : `None` | |

Write a function that applies a given function `f` to each element in the tree using a preorder walk.

- Excepted function

```
func BTreeApplyPreorder(root *TreeNode, f func(...interface{}) (int, error)) {

}
```

- Usage

```
package main

import (
        "fmt"
        "piscine"
)

func main() {
        root := &piscine.TreeNode{Data: "4"}
        piscine.BTreeInsertData(root, "1")
        piscine.BTreeInsertData(root, "7")
        piscine.BTreeInsertData(root, "5")
        piscine.BTreeApplyPreorder(root, fmt.Println)

}
```

- And its output:

```
$ go mod init ex02
$ go run .
4
1
7
5
$
```

# Chapter V

# Exercise 03 : btreesearchitem

|  | Exercise 03 |
|---|---|
| btreesearchitem | |
| Turn-in directory : $ex03/$ | |
| Files to turn in : `*` | |
| Allowed packages : `fmt` | |
| Allowed builtin functions : `None` | |

Write a function that returns the `TreeNode` with a `data` field equal to `elem` if it exists in the tree, otherwise return `nil`.

- Excepted function

```
func BTreeSearchItem(root *TreeNode, elem string) *TreeNode {

}
```

- Usage

```
package main

import (
	"fmt"
	"piscine"
)

func main() {
	root := &piscine.TreeNode{Data: "4"}
	piscine.BTreeInsertData(root, "1")
	piscine.BTreeInsertData(root, "7")
	piscine.BTreeInsertData(root, "5")
	selected := piscine.BTreeSearchItem(root, "7")
	fmt.Print("Item selected -> ")
	if selected != nil {
		fmt.Println(selected.Data)
	} else {
		fmt.Println("nil")
	}

	fmt.Print("Parent of selected item -> ")
	if selected.Parent != nil {
		fmt.Println(selected.Parent.Data)
	} else {
		fmt.Println("nil")
	}

	fmt.Print("Left child of selected item -> ")
	if selected.Left != nil {
		fmt.Println(selected.Left.Data)
	} else {
		fmt.Println("nil")
	}

	fmt.Print("Right child of selected item -> ")
	if selected.Right != nil {
		fmt.Println(selected.Right.Data)
	} else {
		fmt.Println("nil")
	}
}
```

- And its output:

```
$ go mod init ex03
$ go run .
Item selected -> 7
Parent of selected item -> 4
Left child of selected item -> 5
Right child of selected item -> nil
$
```

# Chapter VI

# Exercise 04 : btreelevelcount

|  | Exercise 04 |
|---|---|
| btreelevelcount | |
| Turn-in directory : $ex04/$ | |
| Files to turn in : `*` | |
| Allowed packages : `fmt` | |
| Allowed builtin functions : `None` | |

Write a function, `BTreeLevelCount`, that returns the number of levels of the binary tree (height of the tree)

- Excepted function

```
func BTreeLevelCount(root *TreeNode) int {

}
```

- Usage

```
package main

import (
        "fmt"

        "piscine"
)

func main() {
        root := &piscine.TreeNode{Data: "4"}
        piscine.BTreeInsertData(root, "1")
        piscine.BTreeInsertData(root, "7")
        piscine.BTreeInsertData(root, "5")
        fmt.Println(piscine.BTreeLevelCount(root))
}
```

- And its output:

```
$ go mod init ex04
$ go run .
3
$
```

# Chapter VII

# Exercise 05 : btreeisbinary

|  | Exercise 05 |
|---|---|
| btreeisbinary | |
| Turn-in directory : $ex05/$ | |
| Files to turn in : `*` | |
| Allowed packages : `fmt` | |
| Allowed builtin functions : `None` | |

Write a function, `BTreeIsBinary`, that returns `true` only if the tree given by `root` follows the binary search tree properties.

- Excepted function

```
func BTreeIsBinary(root *TreeNode) bool {

}
```

- Usage

```
package main

import (
        "fmt"
        "piscine"
)

func main() {
        root := &piscine.TreeNode{Data: "4"}
        piscine.BTreeInsertData(root, "1")
        piscine.BTreeInsertData(root, "7")
        piscine.BTreeInsertData(root, "5")
        fmt.Println(piscine.BTreeIsBinary(root))
}
```

- And its output:

```
$ go mod init ex05
$ go run .
true
$
```

# Chapter VIII

# Exercise 06 : btreeapplybylevel

|  | Exercise 06 |
|---|---|
| btreeapplybylevel | |
| Turn-in directory : $ex06/$ | |
| Files to turn in : `*` | |
| Allowed packages : `fmt` | |
| Allowed builtin functions : `None` | |

Write a function, `BTreeApplyByLevel`, that applies the function given by `f`, to each node of the tree given by `root`.

- Excepted function

```
func BTreeApplyByLevel(root *TreeNode, f func(...interface{}) (int, error))  {

}
```

- Usage

```
package main

import (
        "fmt"
        "piscine"
)

func main() {
        root := &piscine.TreeNode{Data: "4"}
        piscine.BTreeInsertData(root, "1")
        piscine.BTreeInsertData(root, "7")
        piscine.BTreeInsertData(root, "5")
        piscine.BTreeApplyByLevel(root, fmt.Println)
}
```

- And its output:

```
$ go mod init ex06
$ go run .
4
1
7
5
$
```

# Chapter IX

# Exercise 07 : btreemax

|  | Exercise 07 |
|---|---|
| btreemax | |
| Turn-in directory : $ex07/$ | |
| Files to turn in : `*` | |
| Allowed packages : `fmt` | |
| Allowed builtin functions : `None` | |

Write a function, `BTreeMax`, that returns the node with the maximum value in the tree given by `root`.

- Excepted function

```
func BTreeMax(root *TreeNode) *TreeNode {

}
```

- Usage

```
package main

import (
        "fmt"
        "piscine"
)

func main() {
        root := &piscine.TreeNode{Data: "4"}
        piscine.BTreeInsertData(root, "1")
        piscine.BTreeInsertData(root, "7")
        piscine.BTreeInsertData(root, "5")
        max := piscine.BTreeMax(root)
        fmt.Println(max.Data)
}
```

- And its output:

```
$ go mod init ex07
$ go run .
7
$
```

# Chapter X

# Exercise 08 : btreemin

|  | Exercise 08 |
|---|---|
| btreemin | |
| Turn-in directory : $ex08/$ | |
| Files to turn in : `*` | |
| Allowed packages : `fmt` | |
| Allowed builtin functions : `None` | |

Write a function, `BTreeMin`, that returns the node with the minimum value in the tree given by `root`.

- Excepted function

```
func BTreeMin(root *TreeNode) *TreeNode {

}
```

- Usage

```
package main

import (
        "fmt"
        "piscine"
)

func main() {
        root := &piscine.TreeNode{Data: "4"}
        piscine.BTreeInsertData(root, "1")
        piscine.BTreeInsertData(root, "7")
        piscine.BTreeInsertData(root, "5")
        max := piscine.BTreeMax(root)
        fmt.Println(max.Data)
}
```

- And its output:

```
$ go mod init ex08
$ go run .
1
$
```

# Chapter XI

# Exercise 09 : btreetransplant

|  | Exercise 09 |
|---|---|
| btreetransplant | |
| Turn-in directory : *ex*09/ | |
| Files to turn in : `*` | |
| Allowed packages : `fmt` | |
| Allowed builtin functions : `None` | |

In order to move subtrees around within the binary search tree, write a function, `BTreeTransplant`, which replaces the subtree started by `node` with the node `rplc` in the tree given by `root`.

- Excepted function

```
func BTreeTransplant(root, node, rplc *TreeNode) *TreeNode {

}
```

- Usage

```
package main

import (
        "fmt"
        "piscine"
)

func main() {
        root := &piscine.TreeNode{Data: "4"}
        piscine.BTreeInsertData(root, "1")
        piscine.BTreeInsertData(root, "7")
        piscine.BTreeInsertData(root, "5")
        node := piscine.BTreeSearchItem(root, "1")
        replacement := &piscine.TreeNode{Data: "3"}
        root = piscine.BTreeTransplant(root, node, replacement)
        piscine.BTreeApplyInorder(root, fmt.Println)
}
```

- And its output:

```
$ go mod init ex09
$ go run .
3
4
5
7
$
```

# Chapter XII

# Exercise 10 : btreedeletenode

|  | Exercise 10 |
|---|---|
| btreedeletenode | |
| Turn-in directory : $ex10/$ | |
| Files to turn in : `*` | |
| Allowed packages : `fmt` | |
| Allowed builtin functions : `None` | |

Write a function, `BTreeDeleteNode`, that deletes `node` from the tree given by `root`. The resulting tree should still follow the binary search tree rules.

- Excepted function

```
func BTreeDeleteNode(root, node *TreeNode) *TreeNode {

}
```

- Usage

```
package main

import (
        "fmt"
        "piscine"
)

func main() {
        root := &piscine.TreeNode{Data: "4"}
        piscine.BTreeInsertData(root, "1")
        piscine.BTreeInsertData(root, "7")
        piscine.BTreeInsertData(root, "5")
        node := piscine.BTreeSearchItem(root, "4")
        fmt.Println("Before delete:")
        piscine.BTreeApplyInorder(root, fmt.Println)
        root = piscine.BTreeDeleteNode(root, node)
        fmt.Println("After delete:")
        piscine.BTreeApplyInorder(root, fmt.Println)
}
```

- And its output:

```
$ go mod init ex10
$ go run .
Before delete:
1
4
5
7
After delete:
1
5
7
$
```