Update from 4
Administrator создал 1 месяц назад

2e099c3e

README.md 9,3 КБ

Day 05 - Go Boot camp

Santa is back in town

Contents

Chapter I

General rules

- Your programs should not quit unexpectedly (giving an error on a valid input). If this happens, your project will be considered non functional and will receive a 0 during the evaluation.
- We encourage you to create test programs for your project even though this work won't have to be submitted and won't be graded. It will give you a chance to easily test your work and your peers' work. You will find those tests especially useful during your defence. Indeed, during defence, you are free to use your tests and/or the tests of the peer you are evaluating.
- Submit your work to your assigned git repository. Only the work in the git repository will be graded.
- If your code is using external dependencies, it should use Go Modules for managing them

Chapter II

Rules of the day

- You should only turn in `*.go` files and (in case of external dependencies) `go.mod + go.sum`
- Your code for this task should be buildable with just `go build`

Chapter III

Intro

— I don't know - said Lily. - The only thing I read about this thing ancient dudes called "Christmas" is that you are supposed to have, like, a tree, something called "a garland", and, finally, a "heap of presents", whatever that means.

You move neuralink visor down to your neck.

— Come on, girl, it's just an urban legend! Why do you think a combination of such basic things should lead to something interesting?

She looked up to the ceiling, dreaming.

— There used to be this, like, old guy in red hoodie or something... Do you think he was one of the first rebellion hackers? You know, sharing quickhacks with everybody? So if script kiddies were thrilled about freedom and fighting the corpos, they could use their "presents" to breach enterprise firewalls?

— Yeah, seems legit. Urban legends of the underground tend to have this mystical aura, you know. Most likely it was just some bearded open source enthusiast. Crazy as people are nowadays, at least nobody says something like "he was riding an antigravity sleigh pulled by robo reindeers". It's more likely that he had a botnet of portable ELF binaries on enterprise servers collecting secret stuff to give it to people for free.

Lily leaned back on the couch and pulled up a bunch of holograms.

— Okay, so everyone knows how trees look like - a bunch of 3d graphs without cycles were floating above her head - Which of them do we need?

Chapter IV

Exercise 00: Toys on a Tree

After some time, you two put together a structure for a Binary tree node:

```
type TreeNode struct {
    HasToy bool
    Left *TreeNode
    Right *TreeNode
}
```

— Looks like you are supposed to... "hang toys" on trees? - Lily looked a bit confused. - Okay, anyway, let's hope just a boolean value will suffice. But they say it's also wrong to put more toys on one side, should it be uniform?

— Okay, I get it - you said. - Let's write a function `areToysBalanced` which will receive a pointer to a tree root as an argument. The point is to spit out a true/false boolean value depending on if left subtree has the same amount of toys as the right one. The value on the root itself can be ignored.

So, your function should return `true` for such trees (0/1 represent false/true, equal amount of 1's on both subtrees):

```
    0
   / \
  0   1
 / \
0   1
```

```
    1
   /  \
  1    0
 / \   / \
1   0 1   1
```

and `false` for such trees (non-equal amount of 1's on both subtrees):

```
  1
 / \
1   0
```

```
  0
 / \
1   0
 \   \
  1   1
```

Chapter V

Exercise 01: Decorating

— So, now about this "garland"... It is supposed to be "reeled up" on a tree.

Lily rotated hologram back and forth, trying to think of something. Then suddenly she lights up with enthusiasm.

— I get it! Let's do it like this... - she draws something that resembles a 3d snake on top of the tree.

So, now you have to write another function called `unrollGarland()`, which also receives a pointer to a root node. The idea is to go top down, layer by layer, going right on even horisontal layers and going left on every odd. The returned value of this function should be a slice of bools. So, for this tree:

```
    1
   /  \
  1    0
 / \   / \
1   0 1   1
```

The answer will be [true, true, false, true, true, false, true] (root is true, then on second level we go from left to right, and then on third from right to left, like a zig-zag).

Chapter VI

Exercise 02: Heap of Presents

— Perfect! I have no idea what those old dudes meant by "Christmas tree", but I think we've met the general requirements.

— So, about those "presents"...

— Presents, right! - Lily raises her elegant finger with a very long purple nail. It was specifically reinforced to fight enemies and (a lot more frequently) to unscrew various devices. - So, let's think of it as a pile. Every such "present" may look like this:

```
type Present struct {
    Value int
    Size int
}
```

— Hmm, what's "Value"?

— Well, some things you tend to value more than the others, right? So they should be comparable.

— Okay, and "Size" is about how long will it take me to download it, right?

— Exactly! So, the the coolest things should be on top.

You need to implement a PresentHeap data structure (using built-in library "container/heap" is recommended, but is not strictly required). Presents are compared by Value first (most valuable present goes on top of the heap). *Only* in case two presents have an equal Value, the smaller one is considered to be "cooler" than the other one (wins in comparison).

Apart from the structure itself, you should implement a function `getNCoolestPresents()`, that, given an unsorted slice of Presents and an integer `n`, will return a sorted slice (desc) of the "coolest" ones from the list. It should use the PresentHeap data structure inside and return an error if `n` is larger than the size of the slice or is negative.

So, if we represent each Present by a tuple of two numbers (Value, Size), then for this input:

```
(5, 1)
(4, 5)
(3, 1)
(5, 2)
```

the two "coolest" Presents would be [(5, 1), (5, 2)], because the first one has the smaller size of those two with Value = 5.

Chapter VII

Exercise 03: Knapsack

— Wait! - you said. - But how do I know that all these amazing presents won't eat up all the space on my hard drive?

Lily thought for a moment, but then proposed:

— For this case, let's only download the most valuable presents!

— But the Heap is using a different ordering and won't help us here...

— True, true. Anyway, there should be some argument to figure out how to get the most value out of the space you have, right?

...It's been a great winter night in CyberCity. Even though traditions changed a lot in last centuries, you two had a feeling you did everything right. Also, Lily didn't know yet about a cool new portable cyberdeck you've prepared as a gift to her this evening. And you had no idea what's in that small mysterious box on her table.

As a last task, you have to implement a classic dynamic programming algorithm, also known as "Knapsack Problem". Input is almost the same, as in the last task - you have a slice of Presents, each with Value and Size, but this time you also have a hard drive with a limited capacity. So, you have to pick only those presents, that fit into that capacity and maximize the resulting value.

Please write a function `grabPresents()`, that receives a slice of Present instances and a capacity of your hard drive. As an output, this function should give out another slice of Presents, which should have a maximum cumulative Value that you can get with such capacity.

Chapter VIII

Reading

Binary Tree Breadth-First Search Depth-First Search Recursion in Go Heap Heap implementation in Go Knapsack Problem Multi-Dimensional arrays and slices in Go