Go_Day05-1

Развилка из недоступного проекта

Обновление с 5
Администратор создал 5 месяцев назад

Имя	Последний коммит	Последнее обновление
образцы кода	Инициировать фиксацию	1 год назад
образцы данных	Инициировать фиксацию	1 год назад
наборы данных	Инициировать фиксацию	1 год назад
материалы	Обновление с 5	5 месяцев назад
разное	Инициировать фиксацию	1 год назад
источник	Инициировать фиксацию	1 год назад
.gitignore	Обновление с 3	1 год назад
ИЗМЕНЕНИЙ	Инициировать фиксацию	1 год назад
ЛИЦЕНЗИЯ	Инициировать фиксацию	1 год назад
README.md	Обновление с 4	7 месяцев назад

README.md

День 05. Учебный лагерь

Санта вернулся в город

Содержание

Глава I

Основные правила

- Ваши программы не должны неожиданно завершать работу (выдавая ошибку при правильном вводе). Если это произойдет, ваш проект будет считаться неработоспособным и получит 0 баллов при оценке.

- Мы рекомендуем вам создать тестовые программы для вашего проекта, даже если эту работу не нужно будет отправлять и она не будет оцениваться. Это даст вам возможность легко проверить свою работу и работу ваших коллег. Вы найдете эти тесты особенно полезными во время вашей защиты. Действительно, во время защиты вы можете использовать свои тесты и/или тесты коллеги, которого вы оцениваете.
- Отправьте свою работу в назначенный репозиторий git. Оцениваться будет только работа в репозитории git.
- Если ваш код использует внешние зависимости, он должен использовать модули Go для управления ими.

Глава II

Правила дня

- Вы должны только сдать `*.go` файлы и (в случае внешних зависимостей) `go.mod` + `go.sum`
- Ваш код для этой задачи должен быть собран с помощью всего лишь `go build`

Глава III

вступление

— Не знаю, — сказала Лили. - Единственное, что я читал об этой штуке, которую древние чуваки называли "Рождеством", это то, что у тебя должна быть, типа, елка, что-то под названием "гирлянда" и, наконец, "куча подарков", что бы это ни значило.

Вы опускаете нейролинковый визор на шею.

— Да ладно, девочка, это просто городская легенда! Как вы думаете, почему сочетание таких простых вещей должно привести к чему-то интересному?

Она смотрела в потолок, мечтая.

— Раньше был такой старик в красной толстовке или что-то в этом роде... Думаешь, он был одним из первых хакеров-бунтовщиков? Знаешь, делиться лайфхаками со всеми? Так что, если сценаристы были в восторге от свободы и борьбы с корпорациями, они могли бы использовать свои «подарки» для взлома корпоративных брандмауэров?

— 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.

Пожалуйста, напишите функцию `grabPresents()`, которая получает слайс экземпляров Present и емкость вашего жесткого диска. На выходе эта функция должна выдать еще один кусочек Подарков, который должен иметь максимальную совокупную Ценность, которую вы можете получить с такой емкостью.

Глава VIII

Чтение

Двоичное дерево Поиск в ширину Поиск в глубину Рекурсия в Go Heap Реализация кучи в Go Проблема рюкзака Многомерные массивы и срезы в Go