Софийски университет „Св. Климент Охридски“
Факултет по математика и информатика

Домашна работа 2

курс Функционално програмиране
за специалности Информатика и Компютърни науки (2-ри поток)
зимен семестър 2020/21 г.

Редакции

2020-11-15 Добавено пояснение за празните (whitespace) символи в представянето на дърво.

Изисквания

Освен на това, което е описано в документа “Схема за оценяване”, решенията ви трябва да отговарят и на дадените по-долу изисквания:

Решението си организирайте в два файла по следния начин:

- `tree.rkt`, който съдържа кода на решението;
- `tree-test.rkt`, който съдържа unit test-овете към решението.

Задачата трябва да се реши на език `racket/base`. За да можете да работите с потоци, във файла с решението включете `racket/stream`. Във файла с unit test-овете включете `rackunit` и `rackunit/gui`.

Описание

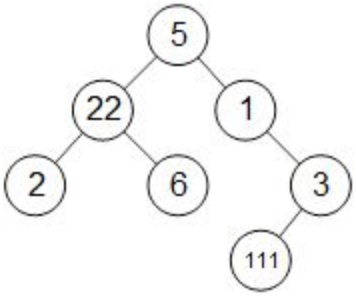
Двоично дърво ще представяме като символен низ по следния начин:

- Празното дърво се представя със символа звезда.
- Дърво с корен N и наследници L и R представяме като "{N L R}".

В програма на Scheme ще представяме дърво по следния начин:

- Празното дърво се представя като празния списък ().
- Дърво с корен N и наследници L и R представяме като списъка (N L R).

По-долу е даден пример за дърво и неговите представяния:



Представяне като символен низ:
```
"{5 {22 {2 * *} {6 * *}} {1 * {3 {111 * *} *}}}"
```

Представяне в Scheme:
```
(list
  (5 (22 (2 () ()) (6 () ())) (1 () (3 (111 () ()) ())))
)
```

Реализирайте дадените по-долу функции. Всички освен `string->tree` и `visualize` трябва да се покрият с подходящи unit test-ове.

`(tree? str)`
Проверява дали подаденият `ѝ` като аргумент символен низ е коректно представяне на дърво.

Редакция (2020-11-15): За определеност, допускаме, че потребителят може да постави произволен брой (вкл. нула) празни символи между отделните елементи на един възел. Например следните са валидни представяния на едно и също дърво:

```
{2 {4 * *} *}
{2{4**}*}
{2     {4     * *}              *}
```

`(string->tree str)`
Конструира и връща двоично дърво по подаденото му текстово представяне `str`. Функцията трябва да направи проверка за коректност на въведените данни (използвайте наготово `tree?`). В случай, че низът не е валидно представяне на двоично дърво, да се върне стойност `#f`.

`(balanced? tree)`
Проверява дали двоичното дърво `tree` е балансирано по височина. Връща `#t` или `#f`.

`(ordered? tree)`
Проверява дали двоичното дърво `tree` е двоично наредено дърво. Връща `#t` или `#f`.

`(tree->string tree)`
Преобразува двоичното дърво `tree` до неговото представяне като символен низ.

Редакция (2020-11-15): Когато преобразувате дърво до символен низ, трябва да слагате точно по един интервал между елементите на възлите и да няма разстояние между тях и фигурните скоби. Например `{2 {4 * *} *}` е валидна стойност, която tree->string може да върне, а следните НЕ СА:

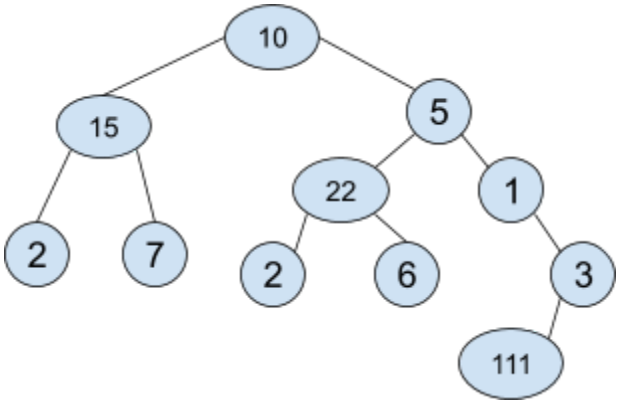
```
{ 2 { 4 * * } * }
{2{4**}*}
{2     {4     * *}              *}
```

`(tree->stream tree order)`
Преобразува двоичното дърво `tree` до поток от неговите елементи. Редът на обхождането се определя от параметъра `order`. Той може да бъде един от символите `'inorder`, `'postorder` или `'preorder`. Справка за редовете на обхождане можете да направите в тази статия на Wikipedia: https://en.wikipedia.org/wiki/Tree_traversal

Например:

- Поток с inorder ред на обхождане, за дървото от примера по-горе, би върнал следната последователност: 2, 22, 6, 5, 1, 111, 3
- Поток с inorder ред на обхождане, за дървото от примера към функцията `visualize` по-долу, би върнал следната последователност: 2, 15, 7, 10, 2, 22, 6, 5, 1, 111, 3

`(visualize tree)`
Визуализира двоичното дърво `tree`. При визуализацията, левият наследник на даден възел трябва да се покаже точно под него, а десният му -- вдясно от него. По-долу е показана визуализацията на дървото от нашия пример. Забележете, че дължината на дъгите, които излизат от всеки възел зависи съответно от височината на десния и широчината на левия му наследници:




```
10-----5------1---3
|      |           |
|      |           111
|      22--6
|      |
|      2
15--7
|
2
```