Некоторые задачи

Косарев Дмитрий a.k.a. Kakadu

матмех СПбГУ

30 октября 2018 г.

Задача за “Жизнь”

Реализуйте игру “Жизнь”, а именно как эволюционирует и моргает поле. В итоге должен получиться бесконечный список состояний, которые будет принимать поле. Разумеется, поле надо уметь распечатывать, чтобы было видно как фигурки моргают, летят и т.д.

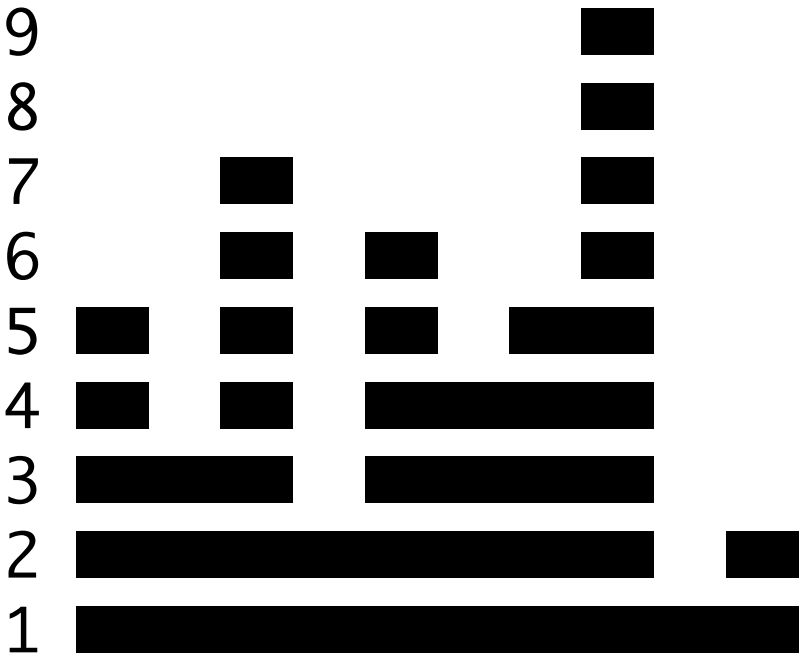
Вариации:

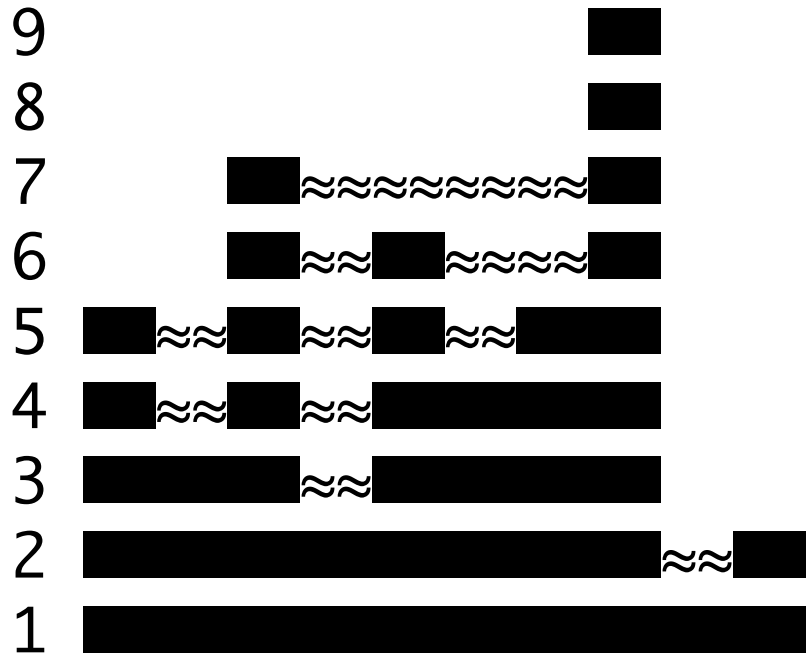
- 1мерный вариант – поле является лентой, неограниченной с двух сторон.
- 2мерный вариант – стандартный, как во всех книжках.

Замечание: можно размер поля прибить гвоздями, а можно расширять, когда что-нибудь приближается ко краю. Сделайте как-нибудь.

P.S. Примеры живущих популяций – тут

Посчитать сколько дождевой воды скопится в ямках





Это известная задача. Вопрос на засыпку – как она связана с предыдущими?

Random access list

N.B. Вспомните, что такое список, и какие там асимптотические характеристики добавления в голову и доступа к случайному элементу.

Скрестите деревья и списки, чтобы `cons` и `tail` работали быстро, а `!!` быстрее, чем со связными списками.

P.S. Когда я переводил книжку Окасаки, мне глава про эту штуку показалась наиболее интересной.

Печатаем дерево красиво

Как утилита `tree` из GNU

```
# tree /etc/ppp
/etc/ppp
├── ip-down.d
│   ├── 0000usepeerdns
│   └── 0dns-down
├── ip-up.d
│   ├── 0000usepeerdns
│   └── 0dns-up
├── ipv6-down.d
├── ipv6-up.d
├── options.pptp
└── resolv
```

Предыдущую задачу можно *упростить*, забив на псевдографические палочки и рисуя отступы пробелами.

Предыдущую задачу можно *усложнить*, сделав рисование не прибитым к конкретной рисуемой структуре данных (выше — это дерево).

А ещё можно мастерскѝ реализовать что-то сильно напоминающее типобезопасный `sprintf`. А потом ещё и `scanf`. Типобезопасность в том смысле, что если в формате сказано печатать число, то надо передать агумент, который будет типа `Int`, а не какую-нибудь строку.

Пример

APIшечка, которая должна получиться.

```
tp1 = sprintf (lit   "Hello world")
-- "Hello world"
ts1 = sscanf "Hello world"  (lit "Hello world") ()
-- Just ()
tp2 = sprintf (lit "Hello "  ^  lit "world"  ^ char)
              '!'
-- "Hello world!"
ts2 = sscanf "Hello World!"
             (lit "Hello " ^ lit "world" ^ char)
             id
-- Just '!'
```

“Ой, сложна“ 1/2

Опять расширим язык лямбд целыми числами, булевыми значениями и конструкцией if-then-else.

```
data Term =
    EInt Int
  | EBool Bool
  | EIfThenElse {cond :: Term,
                 thenBranch :: Term,
                 elseBranch :: Term }
  | ELessThen Term Term
  | ...
```

А теперь мы хотим, чтобы первый аргумент `IfThenElse` представлял собой всегда терм, который вычисляется в `Bool`, если там оказался какой-нибудь `EInt 199` – считать это ошибкой и падать (по аналогии с ошибками, когда мы пытались сложить не-числа).

“Ой, сложна“ 2/2

На прошлом слайде были слабо типизированные термы. Надо переписать тип `Term` так, чтобы плохие конструкции `IfThenElse` нельзя было сконструировать.

Это не так просто сделать. Но я планирую рассказать про это позже. Кому интересно, любое из этих понятий поможет решить задачу.

- Leibniz type equality (можно найти в ”Typing Dynamic Typing“)
- GADT