

FUNKCIJSKO PROGRAMIRANJE



Countdown problem: **Igra s brojevima**

Što je Countdown?

- Popularni TV kviz koji je počeo s emitiranjem na Britanskoj televiziji još 1982. godine.
- Zasnovan na originalnoj francuskoj verziji: "Des Chiffres et Des Lettres".
- Sadrži igru s brojevima koju zovemo **Countdown problem**.

Primjer

Koristeći brojeve

1 3 7 10 25 50

I aritmetičke operatore

+ - * ÷

konstruirajte izraz čija je vrijednost: 765

Pravila

- Svi brojevi, uključujući i međurezultate, moraju biti prirodni brojevi.
- Svaki od ponuđenih brojeva može se koristiti najviše jednom u konstrukciji izraza.
- Sva ostala pravila u originalnoj igri ćemo ignorirati radi jednostavnosti.



U slučaju našeg primjera, jedno od mogućih rješenja je:

$$(25-10) * (50+1) = 765$$

Primjedbe:

- Postoji 780 rješenja ovog primjera.
- Promijenimo li ciljani broj na **831**, igra neće imati rješenja

Evaluiranje izraza

Operatori:

```
data Op = Add | Sub | Mul | Div
```

Primjena operatora:

```
apply :: Op → Int → Int → Int  
apply Add x y = x + y  
apply Sub x y = x - y  
apply Mul x y = x * y  
apply Div x y = x `div` y
```

Potrebno je provjeriti daje li primjena operatora na dva prirodna broja opet prirodan broj:

```
valid :: Op → Int → Int → Bool
valid Add _ _ = True
valid Sub x y = x > y
valid Mul _ _ = True
valid Div x y = x `mod` y == 0
```

Izraz:

```
data Expr = Val Int | App Op Expr Expr
```


Računanje vrijednosti izraza pod uvjetom da su međurezultati prirodni brojevi:

```
eval :: Expr → [Int]
eval (Val n)      = [n | n > 0]
eval (App o l r) = [apply o x y | x ← eval l
                                   , y ← eval r
                                   , valid o x y]
```

Ili uspijeva evaluirati izraz i vraća jednočlanu listu, ili ne uspijeva i vraća praznu listu.

Formaliziranje problema

Vraća listu svih mogućih načina izbora nula ili više elemenata iz liste:

```
choices :: [a] → [[a]]
```

Primjerice:

```
> choices [1,2]  
[[], [1], [2], [1,2], [2,1]]
```

Vraća listu svih vrijednosti u izrazu:

```
values :: Expr → [Int]
values (Val n)      = [n]
values (App _ l r) = values l ++ values r
```

Ispitajte je li izraz rješenje za danu listu ponuđenih brojeva i ciljani broj:

```
solution :: Expr → [Int] → Int → Bool
solution e ns n = elem (values e) (choices ns)
                  && eval e == [n]
```

Brute Force rješenje

Vraća listu svih mogućih načina razdvajanja liste u dvije neprazne liste:

```
split :: [a] → [([a],[a])]
```

Primjerice:

```
> split [1,2,3,4]
```

```
[[([1],[2,3,4]),([1,2],[3,4]),([1,2,3],[4])]]
```

Vraća listu svih mogućih izraza čije su vrijednosti točno jednake onima u danoj listi brojeva:

```
exprs :: [Int] → [Expr]
exprs [] = []
exprs [n] = [Val n]
exprs ns = [e | (ls,rs) ← split ns
                , l      ← exprs ls
                , r      ← exprs rs
                , e      ← combine l r]
```

Ovo je ključna funkcija za
Countdown problem

Kombinirajte dva izraza koristeći sve moguće operatore:

```
combine :: Expr → Expr → [Expr]
combine l r =
    [App o l r | o ← [Add, Sub, Mul, Div]]
```

Računanje liste izraza koji rješavaju instancu **Countdown** problema:

```
solutions :: [Int] → Int → [Expr]
solutions ns n = [e | ns' ← choices ns
                      , e   ← exprs ns'
                      , eval e == [n]]
```

Koliko je brzo ovo rješenje?

Sustav: 2.8GHz Core 2 Duo, 4GB RAM

Compiler: GHC version 7.10.2

Primjer: `solutions [1,3,7,10,25,50] 765`

Jedno rješenje: 0.108 seconds

Sva rješenja: 12.224 seconds

Možemo li dati bolje rješenje?

- Veliki broj izraza koji se razmatraju nisu ispravni i neće se moći evaluirati.
- Primjerice, u našem primjeru od 33 milijuna mogućih izraza samo su 5 milijuna ispravni.
- Kombiniranje generiranja s evaluacijom izraza omogućit će ranije odbacivanje neispravnih izraza.

Spajanje dvije funkcije

Ispravni izrazi i njihove vrijednosti:

```
type Result = (Expr, Int)
```

Definiranje funkcije koja spaja generiranje i evaluaciju izraza:

```
results :: [Int] → [Result]  
results ns = [(e,n) | e ← exprs ns  
                    , n ← eval e]
```

Ovo je moguće postići s funkcijom:

```
results [] = []  
results [n] = [(Val n,n) | n > 0]  
results ns =  
    [res | (ls,rs) ← split ns  
           , lx    ← results ls  
           , ry    ← results rs  
           , res    ← combine' lx ry]
```

gdje je

```
combine' :: Result → Result → [Result]
```

Kombiniranje rezultata:

```
combine' (l,x) (r,y) =  
  [(App o l r, apply o x y)  
   | o ← [Add,Sub,Mul,Div]  
   , valid o x y]
```

Nova funkcija koja rješava **Countdown** problem:

```
solutions' :: [Int] → Int → [Expr]  
solutions' ns n =  
  [e | ns' ← choices ns  
    , (e,m) ← results ns'  
    , m == n]
```

Izmjerimo sada brzinu:

Primjer:

```
solutions' [1,3,7,10,25,50] 765
```

Jedno
rješenje:

0.014 seconds

Oko 10 puta
brže!!!

Sva rješenja: 1.312 seconds

Možemo li bolje?

- Mnogi izrazi su zapravo isti koristeći aritmetičke operacije za koje vrijede svojstva kao što su:

$$x * y = y * x$$

$$x * 1 = x$$

- Koristeći ova svojstva moguće je značajno smanjiti prostor pretraživanja i prostor rješenja.

Iskorištavanja uočenih svojstava

„Dodefiniranje” funkcija za operatore koje uzimaju u obzir svojstvo komutativnosti i jediničnog elementa:

```
valid :: Op → Int → Int → Bool
```

```
valid Add x y =  $x \leq y$ 
```

```
valid Sub x y =  $x > y$ 
```

```
valid Mul x y =  $x \leq y \ \&\& \ x \neq 1 \ \&\& \ y \neq 1$ 
```

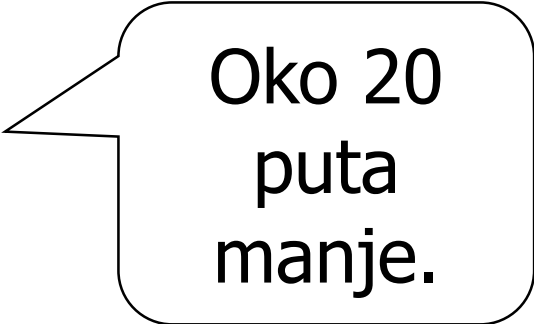
```
valid Div x y =  $x \text{ `mod` } y == 0 \ \&\& \ y \neq 1$ 
```

Izmjerimo sada brzinu...

Primjer:

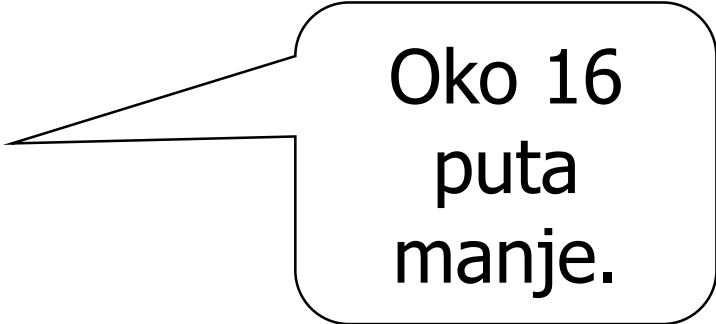
```
solutions'' [1,3,7,10,25,50] 765
```

Ispravnih: 250,000 izraza



Oko 20
puta
manje.

Rješenja: 49 izraza



Oko 16
puta
manje.

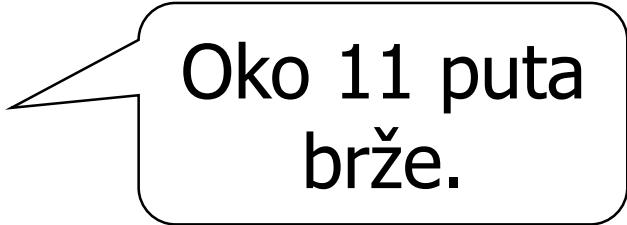
Jedno
rješenje:

0.007 sekundi



Oko dva
puta brže.

Sva rješenja: 0.119 sekundi



Oko 11 puta
brže.

Naš program vraća sva rješenja u djeliću sekunde
i brži je oko 100 puta od originalne verzije.