

Problema 1: Expressió postfixa 1

Escriviu una funció `eval1 :: String → Int` que avalui una expressió postfixa que es troba en una string. Els elements en l'expressió són valors (nombres naturals) i els operadors de suma, resta, producte i divisió. Els elements es separen per espais. Per exemple, l'avaluació de "15 1 2 + 24 * + 3 -" és 84.

La solució canònica per evaluar expressions postfixes és utilitzar una pila: Començant per una pila buida, es processen els elements de l'expressió d'esquerra a dreta. Si l'element és un valor, s'empila. Si l'element és un operador, es desempilen dos valors, s'operen d'acord amb l'operador i s'empila el resultat. Al final, la pila conté un sol element, que és el resultat de l'avaluació de l'expressió.

Podeu suposar que no hi ha mai errors a l'expressió ni divisions per zero.

Solucioneu el problema recursivament. La funció `words` us pot ser útil.

Problema 2: Expressió postfixa 2

Escriviu una funció `eval2 :: String → Int` que avalui una expressió postfixa com al Problema 1, però sense utilitzar recursivitat.

Problema 3: fsmap

Definiu una funció `fsmap :: a → [a → a] → a` que, donats un element x de tipus a i una llista fs de funcions de tipus $a → a$, fa que `fsmap x fs` retorni l'aplicació (d'esquerra a dreta) de totes les funcions de fs a x . Es valorà com de succinta és la vostra solució.

Problema 4: Dividir i vèncer

Escriviu una funció d'ordre superior que definixi l'esquema de dividir i vèncer i utilitzeu-la per fer l'algorisme de quicksort per a llistes d'enters.

La funció per dividir i vèncer ha de tenir aquesta interfície:

$$\text{divideNconquer} :: (a \rightarrow \text{Maybe } b) \rightarrow (a \rightarrow (a, a)) \rightarrow (a \rightarrow (a, a) \rightarrow (b, b) \rightarrow b) \rightarrow a \rightarrow b$$

on a és el tipus del problema, b és el tipus de la solució, i `divideNconquer base divide conquer x` utilitza:

- `base :: (a → Maybe b)` per obtenir la solució directa per a un problema si és trivial (quan és un `Just b`) o per indicar que no és trivial (quan és `Nothing`).
- `divide :: (a → (a, a))` per dividir un problema no trivial en un parell de subproblemes més petits.

- $conquer :: (a \rightarrow (a, a) \rightarrow (b, b) \rightarrow b)$ per, donat un problema no trivial, els seus subproblemes i les seves respectives subsolucions, obtenir la solució al problema original.
- $x :: a$ denota el problema a solucionar.

La funció pel quicksort ha de ser $quickSort :: [Int] \rightarrow [Int]$ i ha d'utilitzar *divideNconquer*.

Problema 5: Racionals

Definiu un tipus *Racional* per manipular nombres racionals positius amb operacions per:

- construir un racional a través d'un numerador i d'un denominador naturals,
- obtenir el numerador de la seva forma simplificada,
- obtenir el denominador de la seva forma simplificada.

A més, feu que *Racional* pertanyi a la classe **Eq** i a la classe **Show**, fent que els racionals es mostrin de la forma " x/y ".

Seguiu aquesta interfície:

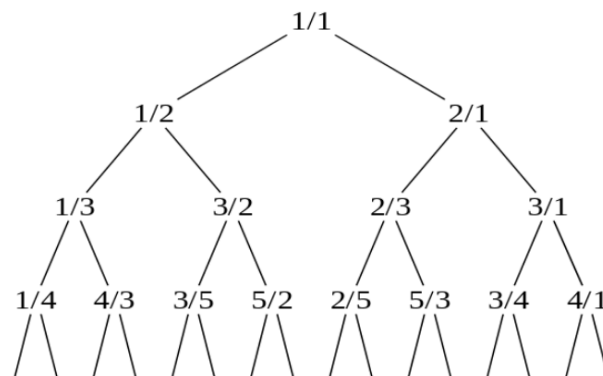
```
data Racional = ...
racional :: Integer -> Integer -> Racional
numerador :: Racional -> Integer
denominador :: Racional -> Integer
```

Si voleu, podeu utilitzar la funció estàndard **gcd** que retorna el màxim comú divisor de dos naturals.

Problema 6: Arbre de Calkin-Wilf

L'arbre de Calkin-Wilf és un arbre binari que representa tots els racionals positius. L'arbre té com arrel el racional $1/1$ i, qualsevol node a/b té dos fills $a/(a+b)$ i $(a+b)/b$.

Aquests són els primers nivells de l'arbre de Calkin-Wilf: [figura extreta de Wikipedia]



Escriviu una funció $racinals :: [Racional]$ que retorni la llista infinita de tots els nombres racionals positius segons l'ordre de l'arbre de Calkin-Wilf.

Per a fer-ho, utilitzeu el tipus *Racional* del problema anterior. També podeu aprofitar les definicions genèriques d'arbre infinit i del seu recorregut per nivells que es donen a continuació:

```

data Tree a = Node a (Tree a) (Tree a)
recXnivells :: Tree a → [a]
recXnivells t = recXnivells' [t]
  where recXnivells' ((Node x fe fd):ts) = x:recXnivells' (ts ++ [fe, fd])

```

Important

El problema té diferents apartats. Cada apartat val 2 punts sobre 10 (però el Jutge en suma 12). Heu de resoldre 5 dels 6 apartats (no els 6!).

Exemple d'entrada 1

```

eval1 "15 1 2 + 24 * + 3 -"
eval1 "66"
eval1 "6 2 -"
eval1 "7 2 /"
eval1 "3 4 + 2 /"

```

Exemple d'entrada 2

```

eval2 "15 1 2 + 24 * + 3 -"
eval2 "66"
eval2 "6 2 -"
eval2 "7 2 /"
eval2 "3 4 + 2 /"

```

Exemple d'entrada 3

```

fsmmap 3 [(+2), (*3), (+4)]
fsmmap "o" [(++"la"), (:)'h', (++"!")]
fsmmap False []

```

Exemple d'entrada 4

```

quickSort [5, 3, 2, 3, 4, 1]

```

Exemple d'entrada 5

```

numerator (racional 1 2)
denominator (racional 1 2)
numerator (racional 2 4)
denominator (racional 2 4)
racional 1 2
racional 2 4
racional 1 2 == racional 2 4
racional 1 2 == racional 1 3

```

Exemple d'entrada 6

```

take 10 racionals

```

Exemple de sortida 1

```

84
66
4
3
3

```

Exemple de sortida 2

```

84
66
4
3
3

```

Exemple de sortida 3

```

19
"hola!"
False

```

Exemple de sortida 4

```

[1, 2, 3, 3, 4, 5]

```

Exemple de sortida 5

```

1
2
1
2
1/2
1/2
True
False

```

Exemple de sortida 6

```

[1/1, 1/2, 2/1, 1/3, 3/2, 2/3, 3/1, 1/4, 4/3, 3/5]

```

Informació del problema

Autoria: Jordi Petit

Generació: 2026-02-03T17:06:08.833Z

© Jutge.org, 2006–2026.

<https://jutge.org>