TPFI 2023/24
Hw 3: Strutture Dati Infinite

assegnato: 23 aprile 2024, consegna 10 maggio 2024

1. Insomnia

Scrivere una funzione Haskell che genera la lista infinita di caratteri `insonnia = "1 sheep 2 sheep 3 sheep 4 sheep ..."`. Provare a scrivere un "one-liner", cioè un programma che semplicemente compone opportunamente funzioni. Può essere utile la funzione `show :: Show a => a => String` che trasforma un elemento di qualsiasi tipo che implementa la classe `Show` in una stringa, cioè una lista di caratteri.

2. Triangolo di Tartaglia

Definite in Haskell la lista infinita di liste finite `tartaglia`, tale che `tartaglia!!n` sia l'n-esima riga del triangolo di Tartaglia, e quindi `tartaglia!!n!!k` sia il coefficiente binomiale $\binom{n}{k}$.

3. Numeri Fortunati

Esercizio 3 (Numeri Fortunati) I *numeri fortunati*, introdotti da Stanislaw Ulam, sono definiti come segue:

1. Dalla sequenza dei numeri naturali (escluso lo zero) tolgo tutti i *secondi numeri*, cioè i pari.

2. il *secondo numero rimasto* è il 3 e quindi si tolgono tutti i terzi numeri tra i sopravvissuti (5, 11, 17, ...).

3. ora si considera il *terzo numero rimasto* cioè il 7 e rimuovo tutti i settimi numeri (il primo è il 19) e così via, fino a ottenere tutti i numeri sopravvissuti a tutte le operazioni di "filtraggio".

Scrivere una funzione Haskell che genera lo stream dei numeri fortunati.

Esercizi Difficili

I seguenti esercizi sono dati per stimolare alcuni eventuali studenti particolarmente interessati alla programmazione funzionale oppure a cimentarsi con problemi difficili e sono quindi da ritenersi *assolutamente* facoltativi.

1D. Iterazione, Ricorsione Primitiva, Church & Ackerman

Una funzione $f :: \mathsf{Nat} \to a$ è definita per ricorsione primitiva dalle funzioni $h :: \mathsf{Nat} \to a \to \mathsf{Nat}$ e $g :: a$ se rispetta le equazioni:

$$f\ (n+1) = h\ n\ (f\ n) \quad \text{e} \quad f\ 0 = g$$

1. Scrivere in Haskell il funzionale primRec che definisce la ricorsione primitiva;

2. Scrivere in Haskell il funzionale primRec′ che definisce la ricorsione primitiva, senza fare ricorsione sui naturali, ma usando l'iterazione e le coppie, cio usando il funzionale for visto a lezione (slide 12, Lezione **8**)

3. Dedurne che in λ–calcolo si può facilmente definire la ricorsione primitiva usando i numerali di Church (che sono essenzialmente iteratori).

4. Un tradizionale risultato di computabilità, stabilisce che la funzione di Ackermann (vedi slide 16, Lezione **12**) non sia definibile per ricorsione primitiva, ma ciò vale al prim'ordine. Dare una definizione della funzione di Ackermann usando il funzionale primRec.

2D. Le Partizioni dell'Infinito

Guardate attentamente la figura nella slide 2 della Lezione **14** (titolo della sotto-lezione **14**(**a**)). Si tratta di uno streams di stream (che chiameremo allPartitions nel seguito) e guardandolo attentamente potete vedere che i numeri in colore ciano sono le partizioni (vedi Homework **1**) del 5, questi uniti a quelli in blu, sono le partizioni del 6. Unendo i numeri verdi si ottengono quelle del 7 e con i numeri rossi otteniamo quelle dell'8.

1. Scrivere un one-liner Haskell partsFromAll tale che partsFromAll n allPartitions sia proprio la lista di liste che rappresenta le partizioni di n (in ordine ascendente, preferibilmente).

2. Scrivere un'equazione ricorsiva che genera allPartitions.

3. Sviluppare qualche idea per rappresentare altre strutture combinatorie in modo analogo, tipo: tutti i sottoinsiemi (finiti) dei Naturali, tutte le permutazioni (a dominio finito) dei Naturali o altre di vostro gradimento.

3D. Numeri di Ulam con programma "circolare"

Leggete la sezione **14**(**c**) delle slides di Lezione **14**. I numeri di Ulam potrebbero essere generati selezionandoli dallo stream di streams ottenibile con allSums ulams (vedi figura nell'ultima slide). Ci sono tuttavia degli strani problemi di "produttività".

1. Date una definizione circolare dei numeri di Ulam, usando allSums ulams (dove ulams è la lista che state generando, usando magari diags (vedi Lezione **14**(**a**)) per aggirare i sopracitati problemi di produttività.

2. La soluzione precedente riduce (computazionalmente) a quella generativa discussa nella lezione **14**(**c**). Ma si può fare "meglio", cioè evitando di dover fare a fette uno stream di streams e considerare le diagonali finite?

Io non lo so, e voi?

4D. Albero dei Razionali di Calkin-Wilf

Tradizionalmente, si dimostra che i numeri razionali sono *numerabili* fornendo una biiezione tra i numeri naturali e le coppie di numeri naturali: tuttavia questa biiezione contiene più volte gli stessi razionali (in realtà ogni razionale p/q appare infinite volte, in quanto tutte le coppie di naturali $\{(kp, kq) \mid k > 0\}$ rappresentano lo stesso numero razionale).

L'*albero di Calkin-Wilf* (vedi figura) fornisce un metodo per costruire una biiezione diretta tra numeri naturali e numeri razionali: i nodi dell'albero contengono tutte e sole le coppie di razionali ridotte ai minimi termini. Ci basta sapere che l'albero di Calkin-Wilf ha una semplice regola di costruzione: se (m, n) è il valore in un nodo, il figlio sinistro contiene la coppia $(m, m + n)$ e il figlio destro la coppia $(m + n, n)$. Viene richiesto di:

1. scrivere un'equazione ricorsiva che genera l'albero di Calkin-Wilf;

2. scrivere la funzione `takeNlevels::Int -> BinTree a -> BinTree a` che taglia un albero (eventualmente infinito) ai primi n livelli; stipulando che `takeNlevels 0 b` torni sempre l'albero vuoto `Empty` e `takeNlevels 1 (Node r lft rgt)` torni `Node r Empty Empty`, etc., l'albero in figura si ottiene con la chiamata `takeNlevels 4 calkinWilf`.

3. ★scrivere una funzione `visitaLivelli :: BinTree a -> [a]` che produce la lista dei valori contenuti nei nodi di un albero (finito) nella sequenza ottenuta da una visita per livelli.

Quindi, la chiamata `visitaLivelli (takeNlevels n calkinWilf)` produrrà i primi $2^n - 1$ razionali nell'ordine indotto da questa biiezione

Qualche spiegazione... Un altro modo di vedere la cosa, è che l'algoritmo di Euclide (in forma sottrattiva) per il Massimo Comun Divisore fa esattamente le stesse sequenze di chiamate ricorsive su tutte le coppie $\{(km, kn) \mid k, m, n > 0\}$:

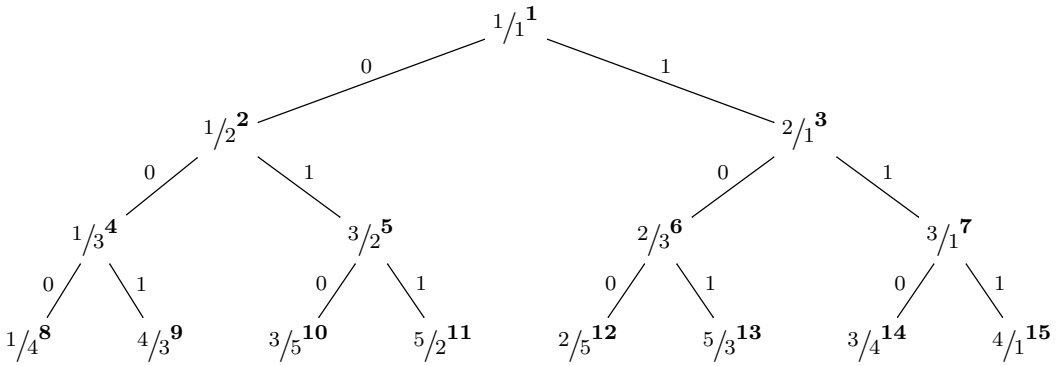

Figura 1: L'albero dei razionali di Calkin-Wilf. In apice e in neretto il naturale corrispondente alla frazione nella numerazione proposta.

i cammini (dal basso verso l'alto) che congiungono $^m/_n$ a $^1/_1$ sono le corrispondenti sequenze binarie che codificano le computazioni di $\mathsf{mcd}(m, n)$ con l'algoritmo di Euclide in versione sottrattiva. I naturali in neretto possono essere interpretati come la somma tra il naturale rappresentato dalla sequenza di 0 e 1 (letta dall'alto verso il basso) sommato all'ultimo numero del livello precedente più 1.