Programmazione Funzionale e Parallela

Corso di Laurea in Ingegneria Informatica e Automatica - A.A. 2019-2020

Esercitazione del 20 aprile 2020

Istruzioni per l'esercitazione:

- Aprite il form di consegna in un browser e loggatevi con le vostre credenziali `uniroma1`.
- Scaricate e decomprimete sulla scrivania il codice dell'esercitazione. Vi sarà una sotto-directory separata per ciascun esercizio di programmazione. Non modificate in alcun modo i programmi di test `E*Main.scala`.
- Rinominare la directory chiamandola `cognome.nome`. Sulle postazioni del laboratorio sarà `/home/studente/Desktop/cognome.nome/`.
- È possibile consultare appunti/libri e il materiale didattico online.
- Rispondete alle domande online sul modulo di consegna.
- **Finiti gli esercizi**, e non oltre le 23:59 :
  - **zippate la directory di lavoro** in `cognome.nome.zip` (`zip -r cognome.nome.zip cognome.nome/`).
- **Per consegnare**:
  - inserite nel form di consegna come autovalutazione il punteggio di ciascuno dei test forniti (inserite zero se l'esercizio non è stato svolto, non compila, o dà errore di esecuzione).
  - fate **upload** del file `cognome.nome.zip`.
- È possibile consultare la documentazione delle API di Scala, in particolare quelle sulle liste, e la dispensa Scala.
- **Se avete domande** accedete a Google Meet all'indirizzo meet.google.com/sph-eiax-fsv durante orario 14:00-16:00 stabilito per l'esercitazione accedendo con la vostra **mail istituzionale**. Troverete online il docente e il tutor del corso. In alternativa, scrivete via email.

Per maggiori informazioni fate riferimento al regolamento delle esercitazioni.

Esercizio 1 (creazione di matrici immutabili in Scala)

Scrivere nel file `E1.scala` un metodo currificato `def buildMatrix(rows:Int, cols:Int)(f:(Int,Int) => Double):Vector[Vector[Double]]` che restituisce una matrice di `Double` con `rows` righe e `cols` colonne dove `f(i,j)` descrive il contenuto della cella `(i,j)`.

Più precisamente, il metodo restituisce un `Vector[Vector[Double]] v` tale che per ogni `i` in `[0,rows-1]` e per ogni `j` in `[0,cols-1]` si ha `v(i)(j) == f(i,j)`.

Usare il main di prova nella directory di lavoro `E1`. Non modificare alcun file tranne `E1.scala`.

Esercizio 2 (numeri di Fibonacci - parallelismo in Scala)

Scrivere nel file `E2.scala` una versione parallela `fibPar` del seguente metodo ricorsivo `fib` definito nel file `Fib.scala` usando fork-join in Scala mediante il costrutto `par`. La soluzione deve assumere la presenza di 2 core nella CPU.

```
def fib(a:Int, b:Int)(n:Int):Long =
    if (n < 2) a
    else if (n == 2) b
    else fib(a,b)(n-1) + fib(a,b)(n-2)
```

Usare il main di prova nella directory di lavoro `E2`, mettendo sulla riga di comando tutti i file Scala forniti. Non modificare alcun file tranne `E2.scala`.

Esercizio 3 (sottolista non consecutiva)

Scrivere nel file `E3.scala` un metodo `def subList(l:List[T]):Boolean` applicabile su un oggetto `List[T] s` che restituisce `true` se e solo se tutti gli elementi di `l` appaiono anche in `s` nello stesso ordine, anche non consecutivamente.

Usare il main di prova nella directory di lavoro `E3`. Non modificare alcun file tranne `E3.scala`.

Esercizio 4 (data analytics con metodi sulle collezioni)

Scrivere nel file `E4.scala` un metodo `piuGiovane` che, dato un `Vector s` di oggetti `Studente(id, nome)` e un `Vector e` di oggetti `Eta(id, eta)`, restituisce un `Option[String]` che vale `None` se `s` è vuoto e `Some(x)`, dove `x` è il nome dello studente più giovane, altrimenti. Si assuma che ogni id contenuto in `s` sia anche contenuto in `e`. Gli id sono unici in ciascuna collezione e servono come chiave primaria in `s` e in `e`.

Usare il main di prova nella directory di lavoro `E4`. Non modificare alcun file tranne `E4.scala`.