Progetto Programmazione Logica e Funzionale

Alessio Muzi | mat. n°299329 | Informatica Applicata | A.S. 2021/2022

Sommario:

1 – Specifica del problema

Scrivere un programma Haskell e un programma Prolog che acquisiscono da tastiera una lista di numeri interi e poi stampano su schermo la lista ordinata usando gli algoritmi di ordinamento Insert Sort, Select Sort e Bubble Sort, misurando il tempo di esecuzione.

2 – Analisi del problema

Dati di ingresso del problema:
L'unico dato di ingresso del problema è rappresentato da una lista di numeri interi.

Dati di uscita del problema:
I dati di uscita del problema sono rappresentati da tre liste di numeri interi e i relativi tempi ottenuti misurando l'esecuzione dell'ordinamento delle liste.

Relazioni intercorrenti:
Dato un insieme, l'algoritmo di ordinamento ne posiziona gli elementi secondo una sequenza stabilita da una relazione d'ordine, in modo che ogni elemento sia minore (o maggiore) di quello che lo segue.

Una relazione d'ordine su un insieme è una relazione binaria tra gli elementi appartenenti all'insieme che gode delle seguenti proprietà:

- Riflessiva: sse $(a, a) \in R$ per ogni $a \in$ campo(R)
- Antisimmetrica: sse $(a_1, a_2) \in R$ implica $(a_2, a_1) \notin R$ per ogni $a_1, a_2 \in$ campo(R), $a_1 \neq a_2$.
- Transitiva: sse $(a_1, a_2) \in R$ e $(a_2, a_3) \in R$ implicano $(a_1, a_3) \in R$ per ogni $a_1, a_2, a_3 \in$ campo(R)

Una relazione d'ordine può essere parziale o totale, se possiede o meno anche la proprietà di dicotomia: $(a_1, a_2) \in R$ o $(a_2, a_1) \in R$ per ogni $a_1, a_2 \in$ campo(R). Generalmente, gli algoritmi di ordinamento sfruttano relazioni d'ordine totale.

Inoltre, gli algoritmi di ordinamento possono possedere due caratteristiche: la stabilità, se preserva l'ordine relativo dei dati con chiavi uguali, e l'operatività sul posto, se non utilizza strutture dati ausiliarie, risparmiando memoria.

Fondamentale è la definizione della complessità temporale, ovvero il tempo di esecuzione dall'algoritmo espresso in funzione della lunghezza dell'input nel caso ottimo, medio e pessimo. Le classi di complessità comunemente usate (ma non le uniche) sono: costante O(1), logaritmico O(log n), lineare O(n), quadratico $O(n^2)$ polinomiale $O(n^k)$ e esponenziale $O(2^{n^k})$.

Insert Sort: sul posto, stabile, ottimo: O(n), medio: $O(n^2)$, pessimo: $O(n^2)$;
Select Sort: sul posto, stabile, ottimo: $O(n^2)$, medio: $O(n^2)$, pessimo: $O(n^2)$;
Bubble Sort: sul posto, stabile, ottimo: $O(n^2)$, medio: $O(n^2)$, pessimo: $O(n^2)$.

3 – Progettazione dell’algoritmo

Scelte di progetto:

La lista di numeri interi si presta a essere rappresentata banalmente mediante una lista allocata dinamicamente in quanto non si conosce a priori la lunghezza dell’input, evitando ogni limitazione imposta rispetto alla specifica del problema. Poiché il dato d’ingresso è una lista, eventuali numeri duplicati sono accettabili. Inoltre, considerando che la stessa lista di numeri verrà ordinata tramite tre diversi algoritmi di ordinamento, una volta acquisito l’input genereremo tre duplicati.

Per calcolare il tempo di esecuzione, misureremo il tempo all’inizio e alla fine dello svolgimento, sottraendo il secondo valore al primo.

Nell’applicare gli algoritmi di ordinamento, la lista in input viene divisa in una sequenza ancora da ordinare e una sequenza già ordinata.

Passi dell’algoritmo:

I passi dell’algoritmo per risolvere il problema sono i seguenti:

- Acquisire la lista di numeri interi.
- Applicare l’algoritmo di ordinamento Insert Sort e stampare i risultati:
    - Misurare il tempo iniziale.
    - Ad ogni step il primo elemento della sequenza non ordinata viene rimosso da essa e inserito nella posizione corretta della sequenza ordinata.
    - Misurare il tempo finale.
- Applicare l’algoritmo di ordinamento Select Sort e stampare i risultati:
    - Misurare il tempo iniziale.
    - Ad ogni step l’elemento minimo della sequenza ancora da ordinare viene posizionato alla fine della sequenza ordinata.
    - Misurare il tempo finale.
- Applicare l’algoritmo di ordinamento Bubble Sort e stampare i risultati:
    - Misurare il tempo iniziale.
    - Ad ogni step sono eseguiti confronti e scambi sistematici tra elementi adiacenti dal fondo per selezionare l’elemento minimo.
    - Misurare il tempo finale.

4 – Implementazione dell’algoritmo

File sorgente `ordinamento.hs`

```
{- Programma Haskell che implementa gli algoritmi di ordinamento Insert Sort,
    Select Sort e Bubble Sort. -}

import Data.Time  -- necessario per usare getCurrentTime e diffUTCTime
import Data.List  -- necessario per usare delete

main :: IO ()
main = do putStrLn "Inserire la lista di numeri interi da ordinare racchiusa tra parentesi quadre:"
          input <- getLine
          let lista1 = read input :: [Int]
          let lista2 = read input :: [Int]
          let lista3 = read input :: [Int]
          putStrLn "\nLista ordinata con Insert Sort:"
          start1 <- getCurrentTime
          putStrLn $ show (insertSort lista1)
          stop1 <- getCurrentTime
          putStrLn "Tempo di esecuzione di Insert Sort:"
          putStrLn $ show (diffUTCTime stop1 start1)
          putStrLn "\nLista ordinata con Select Sort:"
          start2 <- getCurrentTime
          putStrLn $ show (selectSort lista2)
          stop2 <- getCurrentTime
          putStrLn "Tempo di esecuzione di Select Sort:"
          putStrLn $ show (diffUTCTime stop2 start2)
          putStrLn "\nLista ordinata con Bubble Sort:"
          start3 <- getCurrentTime
          putStrLn $ show (bubbleSort lista3)
          stop3 <- getCurrentTime
          putStrLn "Tempo di esecuzione di Bubble Sort:"
          putStrLn $ show (diffUTCTime stop3 start3)


{- La funzione insertSort implementa l'algoritmo di ordinamento Insert Sort.
   Il suo unico argomento è la lista da ordinare.
   Ad ogni step il primo elemento della lista viene rimosso dalla sequenza
   non ordinata e viene inserito nella posizione corretta nella sequenza ordinata. -}

insertSort :: (Ord a) => [a] -> [a]
insertSort []                    = []
insertSort (x : xs)              = insert $ insertSort xs
    where
      insert []                  = [x]
      insert (y : ys) | x < y    = x : y : ys
                      | otherwise = y : insert ys

{- La funzione selectSort implementa l'algoritmo di ordinamento Select Sort.
   Il suo unico argomento è la lista da ordinare.
   Ad ogni step l'elemento minimo viene posizionato alla fine della sequenza ordinata. -}

selectSort :: (Ord a) => [a] -> [a]
selectSort []      = []
selectSort xs      = min : selectSort (delete min xs)
      where
        min        = minimum xs
```

```haskell
{- La funzione bubbleSort implementa l'algoritmo di ordinamento Bubble Sort.
   Il suo unico argomento è la lista da ordinare.
   Ad ogni step vengono effettuati confronti e scambi a due a due
   fino a posizionare l'elemento minimo alla fine della sequenza ordinata. -}

bubbleSort :: (Ord a) => [a] -> [a]
bubbleSort []                                   = []
bubbleSort xs                                   = bubble id [] xs
     where
       bubble next xs (x : y : ys) | x <= y     = bubble next (x : xs) (y : ys)
                                   | otherwise  = bubble bubbleSort (y : xs) (x : ys)
       bubble next xs ys                        = next (reverse xs ++ ys)
```

File sorgente `ordinamento.pl`

```
        /* Programma Prolog che implementa gli algoritmi di ordinamento Insert Sort, Select Sort e
        Bubble Sort. */

main :- write('Inserire la lista di numeri interi da ordinare racchiusa tra parentesi quadre:'), nl,
        read(Input), interi(Input, Lista), copy_term(Lista, Lista1), copy_term(Lista, Lista2),
        copy_term(Lista, Lista3), nl,
        write('Lista ordinata con Insert Sort:'), nl,
        statistics(cpu_time, [T1, _]),
        insertSort(Lista1, A), write(A), nl,
        statistics(cpu_time, [T2, _]), Tempo1 is T2 - T1,
        write('Tempo di esecuzione di Insert Sort:'), nl, write('circa '), write(Tempo1), write(' ms'), nl, nl,
        write('Lista ordinata con Select Sort:'), nl,
        statistics(cpu_time, [T3, _]),
        selectSort(Lista2, B), write(B), nl,
        statistics(cpu_time, [T4, _]), Tempo2 is T4 - T3,
        write('Tempo di esecuzione di Select Sort:'), nl, write('circa '), write(Tempo2), write(' ms'), nl, nl,
        write('Lista ordinata con Bubble Sort:'), nl,
        statistics(cpu_time, [T5, _]),
        bubbleSort(Lista3, C), write(C), nl,
        statistics(cpu_time, [T6, _]), Tempo3 is T6 - T5,
        write('Tempo di esecuzione di Bubble Sort:'), nl, write('circa '), write(Tempo3), write(' ms'), nl.

        interi([], []).
        interi([A|Coda], [A|Coda1]) :- integer(A), interi(Coda, Coda1).

        /* Il predicato insertSort racchiude l'implementazione dell'algoritmo:
           - il primo argomento è la lista input;
           - il secondo argomento è la lista ordinata.

           Il predicato ordinaI esegue l'ordinamento:
           - il primo argomento è la lista input;
           - il secondo argomento è la sequenza ancora da ordinare;
           - il terzo argomento è la sequenza ordinata.

           Il predicato inserimento esegue l'inserimento dei numeri:
           - il primo argomento è l'elemento preso in considerazione;
           - il secondo argomento è la lista nella quale deve essere inserito;
           - il terzo argomento è la lista ottenuta con l'inserimento dell'elemento. */

      insertSort(Lista, Ordinata) :- ordinaI(Lista, [], Ordinata).

      ordinaI([], Resto, Resto).
      ordinaI([T|Coda], Resto, Ordinata) :- inserimento(T, Resto, Resto1), ordinaI(Coda, Resto1, Ordinata).

      inserimento(X, [], [X]).
      inserimento(X, [Y|Coda], [Y|Coda1]) :- X > Y, inserimento(X, Coda, Coda1).
      inserimento(X, [Y|Coda], [X,Y|Coda]) :- X =< Y.

       /* Il predicato selectSort implementa l'algoritmo:
          - il primo argomento è la lista da ordinare;
          - il secondo argomento è la lista ordinata.

          Il predicato min trova l'elemento minimo della lista:
          - il primo argomento è la lista presa in considerazione;
          - il secondo argomento è l'elemento minimo;
          - il terzo argomento è la lista senza l'elemento minimo. */

      selectSort([], []).
      selectSort(Lista, [Min|Coda]) :- min(Lista, Min, Coda1), selectSort(Coda1, Coda).
```

```
min([H], H, []).
min([H|Coda], Min, [H|Coda1]) :- min(Coda, Min, Coda1), H >= Min.
min([H|Coda], H, [Min|Coda1]) :- min(Coda, Min, Coda1), H < Min.

 /*  Il predicato bubbleSort racchiude l'implementazione dell'algoritmo:
    - il primo argomento è la lista da ordinare;
    - il secondo argomento è la lista ordinata.

    Il predicato ordinaB esegue l'ordinamento:
    - il primo argomento è la lista da ordinare;
    - il secondo argomento è la sequenza ancora da ordinare;
    - il secondo argomento è la sequenza ordinata.

    Il predicato bubble esegue il confronto e lo scambio dei numeri:
    - il primo argomento è l'elemento preso in considerazione;
    - il secondo argomento è la lista nella quale deve essere inserito;
    - il terzo argomento è la lista ottenuta con l'inserimento dell'elemento;
    - il quarto argomento è il numero massimo. */

bubbleSort(Lista, Ordinata) :- ordinaB(Lista, [], Ordinata).

ordinaB([], Resto, Resto).
ordinaB([T|Coda], Resto, Ordinata) :- bubble(T, Coda, Coda1, Max), ordinaB(Coda1, [Max|Resto], Ordinata).

bubble(X, [], [], X).
bubble(X, [Y|Coda], [Y|Coda1], Max) :- X > Y, bubble(X, Coda, Coda1, Max).
bubble(X, [Y|Coda], [X|Coda1], Max) :- X =< Y, bubble(Y, Coda, Coda1, Max).
```

5 – Testing del programma

Test Haskell 1
Lista input: `[]`

Lista ordinata (Insert Sort): `[]`
Lista ordinata (Select Sort): `[]`
Lista ordinata (Bubble Sort): `[]`

Tempo di esecuzione(Insert Sort): 0.000097 s
Tempo di esecuzione(Select Sort): 0.000067 s
Tempo di esecuzione(Bubble Sort): 0.000047 s

Test Haskell 2
Lista input: `[1]`

Lista ordinata (Insert Sort): `[1]`
Lista ordinata (Select Sort): `[1]`
Lista ordinata (Bubble Sort): `[1]`

Tempo di esecuzione(Insert Sort): 0.000118 s
Tempo di esecuzione(Select Sort): 0.000084 s
Tempo di esecuzione(Bubble Sort): 0.000071 s

Test Haskell 3
Lista input: `[3,2,1]`

Lista ordinata (Insert Sort): `[1,2,3]`
Lista ordinata (Select Sort): `[1,2,3]`
Lista ordinata (Bubble Sort): `[1,2,3]`

Tempo di esecuzione(Insert Sort): 0.000139 s
Tempo di esecuzione(Select Sort): 0.000096 s
Tempo di esecuzione(Bubble Sort): 0.000078 s

Test Haskell 4
Lista input: `[3,2,3,2,1,1]`

Lista ordinata (Insert Sort): `[1,1,2,2,3,3]`
Lista ordinata (Select Sort): `[1,1,2,2,3,3]`
Lista ordinata (Bubble Sort): `[1,1,2,2,3,3]`

Tempo di esecuzione(Insert Sort): 0.000206 s
Tempo di esecuzione(Select Sort): 0.000133 s
Tempo di esecuzione(Bubble Sort): 0.000124 s

Test Haskell 5
Lista input: `[10,9,8,7,6,5,4,3,2,1]`

Lista ordinata (Insert Sort): `[1,2,3,4,5,6,7,8,9,10]`
Lista ordinata (Select Sort): `[1,2,3,4,5,6,7,8,9,10]`
Lista ordinata (Bubble Sort): `[1,2,3,4,5,6,7,8,9,10]`

Tempo di esecuzione(Insert Sort): 0.000316 s
Tempo di esecuzione(Select Sort): 0.000244 s
Tempo di esecuzione(Bubble Sort): 0.000256 s

Test Haskell 6
Lista input: `[100,90,80,70,60,50,40,30,20,10,0]`

Lista ordinata (Insert Sort): `[0,10,20,30,40,50,60,70,80,90,100]`
Lista ordinata (Select Sort): `[0,10,20,30,40,50,60,70,80,90,100]`
Lista ordinata (Bubble Sort): `[0,10,20,30,40,50,60,70,80,90,100]`

Tempo di esecuzione(Insert Sort): 0.000316 s
Tempo di esecuzione(Select Sort): 0.000257 s
Tempo di esecuzione(Bubble Sort): 0.000260 s

Test Haskell 7
Lista input: `[20,20,22,22,18,18,6,7,7,7]`

Lista ordinata (Insert Sort): `[6,7,7,7,18,18,20,20,22,22]`
Lista ordinata (Select Sort): `[6,7,7,7,18,18,20,20,22,22]`
Lista ordinata (Bubble Sort): `[6,7,7,7,18,18,20,20,22,22]`

Tempo di esecuzione(Insert Sort): 0.000245 s
Tempo di esecuzione(Select Sort): 0.000209 s
Tempo di esecuzione(Bubble Sort): 0.000190 s

Test Haskell 8
Lista input: `[15,28,72,66,33,100,0,1,2,3]`

Lista ordinata (Insert Sort): `[0,1,2,3,15,28,33,66,72,100]`
Lista ordinata (Select Sort): `[0,1,2,3,15,28,33,66,72,100]`
Lista ordinata (Bubble Sort): `[0,1,2,3,15,28,33,66,72,100]`

Tempo di esecuzione(Insert Sort): 0.000319 s
Tempo di esecuzione(Select Sort): 0.000338 s
Tempo di esecuzione(Bubble Sort): 0.000308 s

Test Haskell 9
Lista input: `[3,2,1,6,54,9,8,7,10,11,67,43,567,1,0,5,0]`

Lista ordinata (Insert Sort): `[0,0,1,1,2,3,5,6,7,8,9,10,11,43,54,67,567]`
Lista ordinata (Select Sort): `[0,0,1,1,2,3,5,6,7,8,9,10,11,43,54,67,567]`
Lista ordinata (Bubble Sort): `[0,0,1,1,2,3,5,6,7,8,9,10,11,43,54,67,567]`

Tempo di esecuzione(Insert Sort): 0.000537 s
Tempo di esecuzione(Select Sort): 0.000398 s
Tempo di esecuzione(Bubble Sort): 0.000512 s

Test Haskell 10
Lista input: `[32,34,76,78,123,234,321,331,567,638,678,769,798]`

Lista ordinata (Insert Sort): `[32,34,76,78,123,234,567,638,,678,769,798]`
Lista ordinata (Select Sort): `[32,34,76,78,123,234,567,638,,678,769,798]`
Lista ordinata (Bubble Sort): `[32,34,76,78,123,234,567,638,,678,769,798]`

Tempo di esecuzione(Insert Sort): 0.000491 s
Tempo di esecuzione(Select Sort): 0.000475 s
Tempo di esecuzione(Bubble Sort): 0.000347 s

Test Prolog 1
Lista input: `[]`

Lista ordinata (Insert Sort): `[]`
Lista ordinata (Select Sort): `[]`
Lista ordinata (Bubble Sort): `[]`

Tempo di esecuzione(Insert Sort): circa 0 ms
Tempo di esecuzione(Select Sort): circa 0 ms
Tempo di esecuzione(Bubble Sort): circa 0 ms

Test Prolog 2
Lista input: `[1]`

Lista ordinata (Insert Sort): `[1]`
Lista ordinata (Select Sort): `[1]`
Lista ordinata (Bubble Sort): `[1]`

Tempo di esecuzione(Insert Sort): circa 0 ms
Tempo di esecuzione(Select Sort): circa 0 ms
Tempo di esecuzione(Bubble Sort): circa 0 ms

Test Prolog 3
Lista input: `[3,2,1]`

Lista ordinata (Insert Sort): `[1,2,3]`
Lista ordinata (Select Sort): `[1,2,3]`
Lista ordinata (Bubble Sort): `[1,2,3]`

Tempo di esecuzione(Insert Sort): circa 0 ms
Tempo di esecuzione(Select Sort): circa 0 ms
Tempo di esecuzione(Bubble Sort): circa 0 ms

Test Prolog 4
Lista input: `[3,2,3,2,1,1]`

Lista ordinata (Insert Sort): `[1,1,2,2,3,3]`
Lista ordinata (Select Sort): `[1,1,2,2,3,3]`
Lista ordinata (Bubble Sort): `[1,1,2,2,3,3]`

Tempo di esecuzione(Insert Sort): circa 0 ms
Tempo di esecuzione(Select Sort): circa 0 ms
Tempo di esecuzione(Bubble Sort): circa 1 ms

Test Prolog 5
Lista input: `[10,9,8,7,6,5,4,3,2,1]`

Lista ordinata (Insert Sort): `[1,2,3,4,5,6,7,8,9,10]`
Lista ordinata (Select Sort): `[1,2,3,4,5,6,7,8,9,10]`
Lista ordinata (Bubble Sort): `[1,2,3,4,5,6,7,8,9,10]`

Tempo di esecuzione(Insert Sort): circa 0 ms
Tempo di esecuzione(Select Sort): circa 1 ms
Tempo di esecuzione(Bubble Sort): circa 1 ms

Test Prolog 6
Lista input: `[100,90,80,70,60,50,40,30,20,10,0]`

Lista ordinata (Insert Sort): `[0,10,20,30,40,50,60,70,80,90,100]`
Lista ordinata (Select Sort): `[0,10,20,30,40,50,60,70,80,90,100]`
Lista ordinata (Bubble Sort): `[0,10,20,30,40,50,60,70,80,90,100]`

Tempo di esecuzione(Insert Sort): circa 1 ms
Tempo di esecuzione(Select Sort): circa 1 ms
Tempo di esecuzione(Bubble Sort): circa 1 ms

Test Prolog 7
Lista input: `[20,20,22,22,18,18,6,7,7,7]`

Lista ordinata (Insert Sort): `[6,7,7,7,18,18,20,20,22,22]`
Lista ordinata (Select Sort): `[6,7,7,7,18,18,20,20,22,22]`
Lista ordinata (Bubble Sort): `[6,7,7,7,18,18,20,20,22,22]`

Tempo di esecuzione(Insert Sort): circa 0 ms
Tempo di esecuzione(Select Sort): circa 2 ms
Tempo di esecuzione(Bubble Sort): circa 1 ms

Test Prolog 8
Lista input: `[15,28,72,66,33,100,0,1,2,3]`

Lista ordinata (Insert Sort): `[0,1,2,3,15,28,33,66,72,100]`
Lista ordinata (Select Sort): `[0,1,2,3,15,28,33,66,72,100]`
Lista ordinata (Bubble Sort): `[0,1,2,3,15,28,33,66,72,100]`

Tempo di esecuzione(Insert Sort): circa 0 ms
Tempo di esecuzione(Select Sort): circa 0 ms
Tempo di esecuzione(Bubble Sort): circa 0 ms

Test Prolog 9
Lista input: `[3,2,1,6,54,9,8,7,10,11,67,43,567,1,0,5,0]`

Lista ordinata (Insert Sort): `[0,0,1,1,2,3,5,6,7,8,9,10,11,43,54,67,567]`
Lista ordinata (Select Sort): `[0,0,1,1,2,3,5,6,7,8,9,10,11,43,54,67,567]`
Lista ordinata (Bubble Sort):`[0,0,1,1,2,3,5,6,7,8,9,10,11,43,54,67,567]`

Tempo di esecuzione(Insert Sort): circa 1 ms
Tempo di esecuzione(Select Sort): circa 5 ms
Tempo di esecuzione(Bubble Sort): circa 1 ms

Test Prolog 10
Lista input: `[32,34,76,78,123,234,321,331,567,638,678,769,798]`

Lista ordinata (Insert Sort): `[32,34,76,78,123,234,567,638,,678,769,798]`
Lista ordinata (Select Sort): `[32,34,76,78,123,234,567,638,,678,769,798]`
Lista ordinata (Bubble Sort):`[32,34,76,78,123,234,567,638,,678,769,798]`

Tempo di esecuzione(Insert Sort): circa 2 ms
Tempo di esecuzione(Select Sort): circa 6 ms
Tempo di esecuzione(Bubble Sort): circa 1 ms