Esercitazione 1
Ricorsione

Corso di Fondamenti di Informatica II

A.A. 2014/2015

6 Marzo 2015

Sommario

Scopo di questa esercitazione è risolvere problemi facendo uso di diversi tipi di ricorsione. Si ha ricorsione lineare quando vi è (al più) una chiamata ricorsiva all'interno della funzione, e non lineare nel caso in cui le chiamate ricorsive siano più di una. Un caso particolare della ricorsione non lineare, o multipla, è quello in cui le chiamate ricorsive sono due (ricorsione binaria).

1 Ricorsione lineare: inverti lista

Si vuole creare un metodo per invertire gli elementi di una lista di interi. Ad esempio, data la lista $L = [10, 20, 5, 6, 8]$ in ingresso, si vuole creare una nuova lista L' tale che $L' = [8, 6, 5, 20, 10]$. Inoltre, si vuole che il metodo richieda tempo $O(n)$ (dove n è il numero di elementi della lista) e utilizzi soltanto l'iteratore associato ad L. Il metodo *non deve modificare la lista di input* L.

Specifiche. Scrivere una classe Java `InvertiLista.java` (risp. un modulo C con intestazione `invert_list.h`) contenente il metodo

```
public static LinkedList<Integer>
        invertiLista(ListIterator<Integer> li)

linked_list * invert_list(linked_list_iterator * iter)
```

Si proceda a testare `InvertiLista` attraverso il driver `InvertiListaTest.java` (risp. utilizzando `main.c` passando come argomento `inverti`) fornito nella cartella dell'esercitazione.

2 Ricorsione binaria: stampa righello

Si vuole creare un metodo per stampare un righello, specificando la dimensione del righello (quante unità contiene) e la lunghezza delle unità. Si vuole inoltre, che il

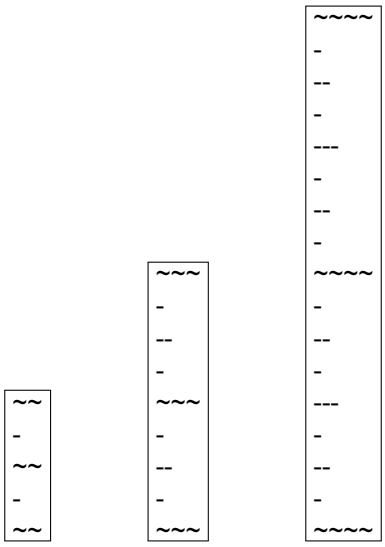

Figura 1: Tre righelli aventi tutti 2 unità, ma con tacche di dimensione massima rispettivamente 2, 3 e 4.

righello, oltre ad avere delle tacche in corrispondenza di ciascuna unità, abbia anche delle tacche intermedie secondo lo schema esemplificato nella figura seguente.

Specifiche. Scrivere una classe Java `Righello.java` (risp. un modulo C con intestazione `righello.h`) contenente il metodo statico

```
public static void righello(int lunghezza, int unita)

void righello(int lunghezza, int unita)
```

Si proceda a testare `Righello` attraverso il driver `RighelloTest.java` (risp. utilizzando `main.c` passando come argomento `righello`) fornito nella cartella dell'esercitazione.

3 Ricorsione multipla: risolvi labirinto

Dato un labirinto rappresentato da una matrice quadrata, si vuole trovare un cammino dalla cella di entrata alla cella di uscita. Nel labirinto ci sono celle *vuote* che possono essere percorse, e celle *piene* che non possono essere percorse. Il movimento può avvenire in quattro direzioni (sinistra, destra, alto, basso). Gli spostamenti in diagonale non sono ammessi.

Si rappresenti il labirinto con una matrice `m[][]` di dimensione $n \times n$. L'entrata corrisponde alla posizione $(0, 0)$ (angolo in alto a sinistra), e l'uscita alla posizione $(n-1, n-1)$ (angolo in basso a destra). Ciascun elemento `m[i][j]` rappresenta una cella vuota o piena (si introducano delle costanti simboliche `VUOTA` e `PIENA`, rispettivamente).

```
.....          ....#
###.#          ###.#
....#          ....#
#.##.          #.##.
.....          ...#.
```

Figura 2: Due labirinti 5×5. Il primo ammette soluzione, il secondo no.

Si intende scrivere una classe che determini se esiste un percorso dall'entrata all'uscita del labirinto e, nel caso che esista, lo visualizzi. La lunghezza del cammino trovato non è rilevante.

Suggerimento. Si utilizzi un algoritmo ricorsivo che via via *marca* – in una matrice ausiliaria `marcata[][]` di $n \times n$ `boolean` – le celle esplorate nel percorso parziale costruito. Si utilizzi il fatto che l'uscita è raggiungibile da una certa cella vuota (i, j) se e solo se essa è raggiungibile da qualche cella vuota adiacente ad (i, j).

Specifiche. Scrivere una classe Java `Labirinto` (risp. un modulo C con intestazione `labirinto.h`) con la seguente interfaccia. Si consiglia di implementare i metodi nell'ordine indicato.

```
public Labirinto(int n)

labirinto * labirinto_new(int n)
```

Crea un labirinto contenente $n \times n$ celle vuote.

```
public void setPiena(int r, int c)

void labirinto_setpiena(labirinto * lab, int r, int c)
```

Setta a `PIENA` la cella `m[r][c]` (dove $0 \leq r < n$, $0 \leq c < n$).

```
private boolean uscita(int r, int c)

int labirinto_uscita(labirinto * lab, int r, int c)
```

Restituisce `true` se la cella (r, c) corrisponde all'uscita del labirinto.

```
public boolean percorribile(int r, int c)

int labirinto_percorribile(labirinto * lab, int r, int c)
```

Restituisce `true` se la cella (r, c) è percorribile, dove "percorribile" significa che la cella è dentro ai bordi del labirinto e non è né `PIENA` né marcata.

```
private boolean uscitaRaggiungibileDa(int r, int c)

int labirinto_uscitaraggiungibileda(labirinto * lab, int r, int c)
```

Restituisce `true` se l'uscita del labirinto è raggiungibile dalla cella (r, c) utilizzando solo celle percorribili. *Questo metodo deve essere ricorsivo.*

```
public boolean risolvibile()

int labirinto_risolvibile(labirinto * lab)
```

Restituisce `true` se e solo se il labirinto ammette soluzione. In tal caso, al termine del metodo le celle che fanno parte della soluzione dovranno risultare marcate.

```
public String toString()

void labirinto_tostring(labirinto * lab, char * buffer)
```

Restituisce una stringa rappresentante il labirinto, incluso l'eventuale percorso dall'entrata all'uscita (si assuma che il metodo `risolvibile()` sia già stato invocato prima dell'invocazione di `toString()`). Si rappresenti una cella vuota (non marcata) col carattere `'.'`, una piena col carattere `'#'`, e una cella marcata col carattere `'+'`.

Si proceda a testare `Labirinto` attraverso il driver `LabirintoTest.java` (risp. utilizzando `main.c` passando come argomento `labirinto`) fornito nella cartella dell'esercitazione e i labirinti di esempio contenuti nei file di testo `lab1.in`, `lab2.in`. La riga di comando

```
java LabirintoTest < lab1.in

a.exe < lab1.in
```

invoca il risolutore sul primo esempio. La classe `LabirintoTest` permette anche di leggere il labirinto di input direttamente da console.

Riferimenti bibliografici

[1] M. T. Goodrich and R. Tamassia. *Strutture dati e algoritmi in Java.* Zanichelli, 2007.