Laboratorio di Programmazione a.a. 2016-17

Progetto "Codifica di Huffman" – Parte II

19 Maggio 2017

1. Coda con Priorità

Definisci una classe `NodeQueue` che possa sostituire `PriorityQueue<Node>` nel programma `Huffman` offrendo le stesse funzionalità della classe predefinita. Il protocollo deve quindi prevedere il costruttore e i metodi così specificati:

`public NodeQueue()`	*costruttore: creazione della coda vuota*
`public int size()`	*restituisce il numero di elementi contenuti nella coda*
`public Node poll()`	*restituisce e rimuove dalla coda l'elemento con "peso minore"*
`public void add( Node n )`	*aggiunge un nuovo elemento* `n` *alla coda*

Realizza la rappresentazione interna utilizzando strumenti base di Java, in particolare gli *array*, senza ricorrere all'importazione di package di supporto (oltre a `java.lang`, che viene importato automaticamente).

Verifica infine la correttezza della soluzione utilizzando la versione del programma `Huffman` collegata a questa parte del progetto nella sezione delle pagine del corso dedicata al Laboratorio.

2. Rappresentazione "permanente" di alberi di Huffman

Definisci una classe `HuffmanTree` che consenta di gestire compattamente le operazioni di salvataggio e ripristino relative ad alberi di Huffman. Più specificamente, il protocollo deve comprendere i seguenti costruttori e metodi:

`public HuffmanTree( Node n )`	*costruttore: creazione dell'albero di Huffman di radice* `n`
`public HuffmanTree( String src )`	*costruttore: creazione (ripristino) dell'albero di Huffman a partire dalla sua codifica testuale contenuta nel file di nome* `src`
`public Node root()`	*restituisce la radice dell'albero di Huffman rappresentato dall'oggetto*
`public void save( String dst )`	*salva l'albero di Huffman (in forma testuale) in un file di nome* `dst`

I file coinvolti (riferiti attraverso i termini `src` e `dst`) sono intesi a contenere esclusivamente la codifica di un albero di Huffman. In altri termini è come se questi file fossero la versione resa *permanente* dell'albero di Huffman in questione, separato da un eventuale contesto di utilizzo. In particolare, il file "sorgente" il cui nome è passato come argomento al secondo costruttore deve contenere solo la codifica testuale di un albero di Huffman; simmetricamente, il file "destinazione" argomento del metodo `save` viene creato dal metodo stesso e alla fine conterrà solo la codifica testuale dell'albero di Huffman rappresentato dall'oggetto di tipo `HuffmanTree`. In entrambi i casi, i file sono aperti e chiusi dal costruttore/metodo. Per quanto riguarda il resto del protocollo, l'argomento del primo costruttore è la radice di un albero di Huffman strutturalmente già costruito, per esempio secondo le modalità consuete; il metodo `root`, infine, restituisce la radice, e quindi l'intera struttura, dell'albero rappresentato dall'oggetto di tipo `HuffmanTree`. Ai fini della realizzazione della classe `HuffmanTree` puoi ovviamente utilizzare, invocandoli direttamente, i metodi statici definiti nel "modulo" `Huffman`.

Per concludere, verifica sperimentalmente le funzionalità della nuova classe applicando i seguenti frammenti di codice:

```
int[] freq = Huffman.charHistogram( "Huffman.java" );
HuffmanTree tree1 = new HuffmanTree( Huffman.huffmanTree(freq) );
tree1.save( "H.txt" );
```

```
HuffmanTree tree2 = new HuffmanTree( "H.txt" );
Huffman.flattenTree( tree2.root() );
```

nonché confrontando i risultati ottenuti con il contenuto dell'intestazione del documento compresso secondo le modalità discusse a lezione.