# RELAZIONE PROGETTO BIGINT

Classe Astratta AbstractBigInt

La classe implementa l’interfaccia BigInt.

* **String toString() :** Attraverso un “for each” inserisco in uno StringBuilder i singoli Integer che compongono il BigInt.
* **boolean equals( Object x ) :** Classico equals, ritorna true in caso this sia uguale a x per aliasing. Se i due BigInt hanno lunghezza diversi non saranno uguali, quindi ritorna false. Se non rientra in questi casi, grazie all’iteratore scorre gli Integer che compongono i due BigInt se trova due Integer corrispondenti diversi tra loro, ritorna false. Infine, nel caso in cui gli Integer siano tutti uguali, il metodo ritorna true.
* **int hashCode() :** Usando la tecnica dello shuffling, per ogni Integer di BigInt aggiungo ad h il suo valore moltiplicato per un numero primo e hashCode() proprio delle stringhe dato dal metodo String.valueOf().

Interfaccia BigInt

L’interfaccia estende Comparable<BigInt> per poter implementare su di essa il compareTo() tra BigInt e Iterable<Integer> per poter iterare sul BigInt.

* **String value() :** Il metodo ritona il valore del BigInt sottoforma di stringa. Con un “for each” (usabile grazie all’iteratore) scorre gli Integer del BigInt e inserisce il loro valore sottoforma di stringa nello StringBuilder, precedentemente istanziato. Ritorna il toString dello StringBuilder.
* **Int lenght() :** Il metodo ritorna il numero di cifre del BigInt. Attraverso un “for each” scorre gli Integer del BigInt incrementando un contatore. Ritorna il valore del contatore.
* **BigInt factory(int x):** implementato nella classe concreata BigIntLL.
* **BigInt incr():** Il metodo ritorna un BigInt incrementato di 1. Viene creato un BigInt con il factory() passandogli come parametro 1. Esso viene addizionato al BigInt this tramite add(factory(1) e il BigInt risultante viene ritornato come risultato del metodo.
* **BigInt decr():** Il metodo ritorna un BigInt decrementato di 1. Se il BigInt sul quale si invoca il metodo (this) ha valore pari a 0 viene sollevata una IllegalArgumentException in quanto la classe non prevede che i BigInt siano negativi. È sviluppato analogamente al metodo incr(), l’unica differenza è che il BigInt creato con il factory(1) viene sottratto al BigInt sul quale è invocato il metodo (this). Il risultato della sottrazione viene ritornato.
* **BigInt add( BigInt a ) :** implementato nella classe concreta BigIntLL.
* **BigInt sub( BigInt s ) :** implementato nella classe concreta BigIntLL.
* **BigInt mul( BigInt m ) :** implementato nella classe concreta BigIntLL.
* **BigInt div( BigInt d ) :** Il metodo ritorna il quoziente della divisione intera tra this e d. Se Il BigInt sul quale è invocato il metodo è più piccolo del BigInt passato come parametro (d) viene sollevata una IllegalArgumentException. Viene inizializzato un BigInt “risultato” copiando this, per non apportare modifiche al BigInt sul quale viene invocato il metodo. Il metodo funziona attraverso un while che sottrae dal BigInt (this) il BigInt passato come parametro(d) fin quando il risultato di tale sottrazione è maggiore o uguale a (d). Per ogni sottrazione effettuata viene incrementato un BigInt contatore, il quale sarà il risultato della divisione intera.
* **BigInt rem( BigInt d ) :** Il metodo ritorna il resto della divisione intera tra this e d. Anche in questo caso se Il BigInt sul quale è invocato il metodo è più piccolo del BigInt passato come parametro (d) viene sollevata una IllegalArgumentException. Viene salvato il BigInt risultante dalla divisione tra this e d. Esso viene successivamente moltiplicato per d. Tale prodotto infine viene sottratto dal this, e il risultato di questa sottrazione sarà il resto della divisione intera.
* **BigInt pow( int exponent ) :** Il metodo calcola la potenza this^exponent. Il metodo sfrutta la ripetizione di moltiplicazioni su una copia di this fin quando il contatore(inizializzato a 0) risulta minore del exponent.
* **int compareTo( BigInt o ) :** Il metodo confronta due BigInt. Se la lunghezza del this è maggiore di quella di o ritorno subito 1. Se la lunghezza tra i due BigInt è la medesima, grazie all’iteratore che parte per entrambi dall’ultima posizione (a sinistra) prendo i singoli Integer che compongono i BigInt e appena trovo un Integer di this > Integer di o ritorna 1, se < ritorna -1 e se sono uguali si continua a scorrere i BigInt con gli iteratori. Nel caso in cui gli Integer dei BigInt risultassero tutti uguali il metodo ritorna 0.
* **ListIterator<Integer> iterator(int i) :** implementato nella classe concreta BigIntLL.

Classe Concreta BigIntLL

La classe estende la classe astratta AbstractBigInt, descritta in precedenza, quindi è necessario implementare i metodi lasciati incompiuti nell’interfaccia.

È dichiarata privata una LinkedList di Integer “lista” che sarà appunto il “contenitore” del BigInt.

* **BigIntLL( String s ) :** Costruttore usato passando come parametro una stringa. In esso viene controllato che la stringa sia effettivamente un numero attraverso una regex. Successivamente si controlla che il numero corrispondente alla stringa non sia minore di 0, in quanto la classe prevede che il BigInt non possa essere negativo. Infine inserisco nella lista i singoli Integer(caratteri) della stringa.
* **BigIntLL( int x ) :** Costruttore usato passando come parametro un int x. Se il numero è minore di 0 si solleva una IllegalArgumentExption. Successivamente trasformando l’int in String inserisco i singoli Integer nella lista, come nel costruttore precedente.
* **BigIntLL factory( int x ) :** Metodo necessario per il funzionamento dei metodi implementati di default nell’interfaccia. Ritorna un nuovo BigIntLL chiamando il costruttore per interi.
* **int length() :** Viene fatto un Override (per motivi di costo) di questo metodo in quanto già implementato nell’interfaccia. In questo caso basta solo richiamare il metodo size() delle LinkedList sulla lista. In questo modo si evita di iterare su di essa.
* **String value() :** Viene fatto un Override (per motivi di costo) di questo metodo in quanto già implementato nell’interfaccia. In questo caso basta semplicemente richiamare il toString del BigInt (definito nella classe astratta).
* **BigInt add( BigInt a) :** Il metodo esegue l’addizione tra il BigInt sul quale viene invocato (this) e quello passato come argomento(d). Vengono copiati i valori dei singoli Integer come caratteri su due StringBuilder. Successivamente il metodo si divide in base se i due SB abbiano lunghezza uguale o se il prima abbia lunghezza maggiore del secondo. In caso il primo abbia lunghezza inferiore del secondo, inverto i due SB (sfruttando la proprietà commutativa dell’addizione), così rientrando nel caso precedente. L’algoritmo nei due casi è molto simile, viene eseguita l’addizione dei singoli Integer (Integer1+Integer2)+riporto.In caso il risultato di queste addizioni venga >10 si pone a 1 la variabile riporto. L’unica differenza sta nel fatto che in caso la lunghezza di SB1 sia maggiore di SB2, viene salvata la porzione in più di SB1 e successivamente ad aver svolto le varie singole addizione si aggiunge al risultato la porzione in eccesso, incrementata di 1 se l’ultimo riporto era pari a 1.
* **BigInt sub( BigInt s ) :** Il metodo esegue la sottrazione tra il BigInt sul quale viene invocato (this) e quello passato come argomento(d). Viene sollevata una IllegalArgumentException se this sia minore di (d) in quanto la classe non prevede che i BigInt possano essere negativi. Il metodo è molto simile alla add con la differenza che nelle varie singole sottrazioni se il risultato risulta <0 viene aggiunto +10 e l’Integer successivo di this decrementato di 1. Alla fine se rimangono degli zeri a sinistra del numero risultante, quindi inutili, essi vengono eliminati.
* **ListIterator<Integer> iterator() :** Ritorna il ListIterator di LinkedList invocato sulla lista.
* **ListIterator<Integer> iterator(int i) :** Ritorna il ListIterator di LinkedList invocato sulla lista e aperto in posizione (i).
* **BigInt mul( BigInt m ) :** Il metodo esegue la moltiplicazione tra il BigInt sul quale viene invocato il metodo (this) e il BigInt passato come parametro (m). Attraverso due cicli innestati eseguo la moltiplicazione tra il primo Integer di this e tutti gli Integer di (m). Alla fine di ogni secondo ciclo faccio la somma tra i vari risultati prodotti da ogni numero di this, mettendo uno zero a destra per ogni ciclo principale.

Andrea Dell’Osso, mat. n°209865.