Vogliamo realizzare un sistema generale di thread-pooling che ci permetta, in modo semplice, di effettuare scheduling di task, eventualmente con priorità.

I concetti che ci servono sono i seguenti:

Task. Un task è, intuitivamente, un oggetto che rappresenta un compito da svolgere; deve avere fondamentalmente un metodo:

- `void perform() throws InterruptedException` che esegue il task;

TaskPool. Un task pool è essenzialmente un insieme di task. Potete pensarlo come a una coda: nuovi task possono essere messi in coda, oppure possono essere prelevati dalla coda. Per permettere al task pool di scegliere in modo più sofisticato (cioè, non LIFO) quale task prelevare, quando si aggiunge un task si può specificare un oggetto che ne descrive la priorità.

- `void add(Task t)` aggiunge il task `t`;
- `void add(Task t,Object priority)` aggiunge il task `t` con priorità `priority`;
- `Task next()` restituisce il primo task da eseguire, e lo toglie dal pool;
- `boolean isEmpty()` guarda se il task pool è vuoto.

Executor. Un executor è essenzialmente un thread che ha accesso a un certo TaskPool. In ciascun istante un executor può essere ozioso oppure può essere in esecuzione su un certo task. Quando un executor è ozioso, aspetta semplicemente che qualche task si aggiunga al TaskPool; se ciò avviene, tenta di procurarsi un task (prelevandolo dal TaskPool) e lo esegue.

La realizzazione di un `TaskPool` LIFO è molto semplice. Realizzare invece un TaskPool non LIFO, ma basato su priorità, è molto più sofisticato. L'idea è di usare una coda di priorità realizzata mediante uno heap.

Uno *heap* (letteralmente: mucchio) è (almeno idealmente) un albero binario i cui nodi contengono dei dati, ciascuno caratterizzato da un campo *chiave*, che soddisfa le seguenti proprietà:

1. tutte le foglie dello heap hanno altezza (cioè distanza dalla radice) h o $h-1$ per un valore opportuno di h;
2. le foglie di altezza $h-1$ (se ce ne sono) stanno "a destra" delle foglie di altezza h;
3. per ogni nodo, la chiave del nodo è minore o uguale della chiave dei figli.

Conseguenza della definizione, naturalmente, è che la radice ha chiave minima.

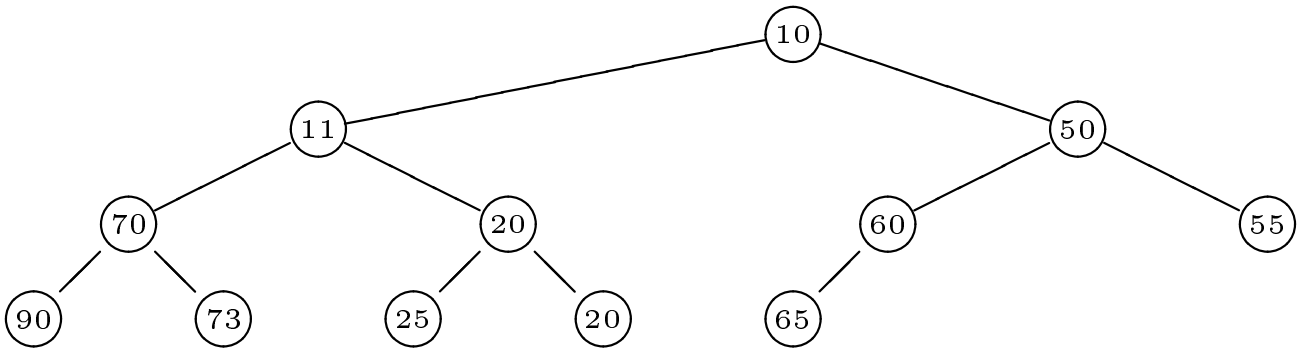

Figure 1: Esempio di heap

Di solito, per memorizzare uno heap, non si utilizza una struttura dinamica ma un semplice vettore. Più precisamente, uno heap con N nodi si memorizza in un vettore $a[0], \ldots, a[N-1]$ in cui il nodo $a[i]$ ha come figlio sinistro il nodo $a[2i+1]$ e come figlio destro il nodo $a[2i+2]$.

Indice	0	1	2	3	4	5	6	7	8	9	10	11
Contenuto	10	11	50	70	20	60	55	90	73	25	20	65

Figure 2: Rappresentazione dello heap di Fig. 1 mediante un vettore

Notate che nello heap l'elemento con chiave minima è *sempre* la radice. Supponete di voler togliere l'elemento di chiave minima: lo heap si può ristrutturare, mettendo l'ultimo elemento (l'ultima foglia) in cima e poi facendolo scendere, se serve, fino al punto in cui va fatto scendere per riottenere uno heap.

Viceversa, quando si aggiunge un elemento lo si può aggiungere in fondo, facendolo prima risalire e poi ridiscendere.