Laboratorio di algoritmi e strutture dati

Docente: Violetta Lonati

Heap e code di priorità*

1 Esercizio: implementazione di code di priorità con heap

Realizzate la struttura di dati *coda di priorità* dichiarando e implementando questa *iterfaccia*:

```
typedef struct pqueue *Pqueue;

/* crea una coda di priorità vuota che potrà contenere al massimo n Item */
Pqueue pqueue_new( int n );

/* distrugge la coda di priorità */
void pqueue_destroy( Pqueue );

/* restituisce la lunghezza della coda di priorità */
int pqueue_length( Pqueue );

/* inserisce l'Item nella coda di priorità */
void pqueue_insert( Pqueue, Item );

/* estrae dalla coda di priorità l'Item con chiave minima */
Item pqueue_extractmin( Pqueue );

/* restituisce l'Item con chiave minima nella coda di priorità */
Item pqueue_min( Pqueue );
```

Implementate queste funzioni usando la struttura dati *heap*.

Dovrete innanzitutto definire il tipo **struct** pqueue:

```
typedef Item *Heap;

struct pqueue {
  Heap h;
  int size, count;
};
```

Inoltre sarà utile definire alcune funzioni ausiliarie (ad esempio `heapify_up` o `heapify_down`).

Applicazione: ordinamento tramite coda di priorità

Usate le funzioni dichiarate in per realizzare la funzione

*Ultima modifica 2 dicembre 2020

```
pqueue_sort( Item a[], int l, int r );
```

che ordina la porzione compresa tra gli indici `l` e `r` dell'array *a*. Potete usare il seguente algoritmo:

```
crea una nuova coda di priorità Q
inserisci in Q un elemento di a alla volta
finchè Q non è vuota
  estrai il minimo m da Q
  stampa m
```

Complessità in tempo: caso pessimo

Considerate la complessità dell'algoritmo di ordinamento descritto sopra. Individuate degli esempi di heap in cui si realizza il "caso pessimo", ovvero in cui l'algoritmo esegue effettivamente un numero di confronti nell'ordine di grandezza di $O(nlogn)$.

Individuate esempi in cui il caso pessimo è dovuto alla fase di inserimento degli elementi in coda e esempi in cui è invece dovuto al ciclo con l'estrazione ripetuta del minimo.

Applicazione: heapsort

Completate l'esercizio con una funzione `heapsort` che ordina un array di interi usando uno *heap* e con la funzione `main` che testa il funzionamento di `heapsort`.

In questo caso non serve usare una coda di priorità, ma potete riutilizzare le definizioni di tipi e le funzioni ausiliarie per manipolare lo *heap* che avete scritto per i punti precedenti. Se volete, potete anche predisporre un file di interfaccia per rendere disponibili pubblicamente alcune di queste definizioni/funzioni.

Cosa cambia rispetto all'applicazione precedente?

In particolare, rispetto alla complessità, quali sono le situazioni più critiche con questo algoritmo? Cosa cambia rispetto all'algoritmo implementato prima?

[Facoltativo] Altre implementazioni

Provate ad implementare la coda di priorità in uno dei modi seguenti (meno efficienti!) e a confrontarne il funzionamento:

- array (o lista) + puntatore al minimo;
- lista ordinata;
- array ordinato.

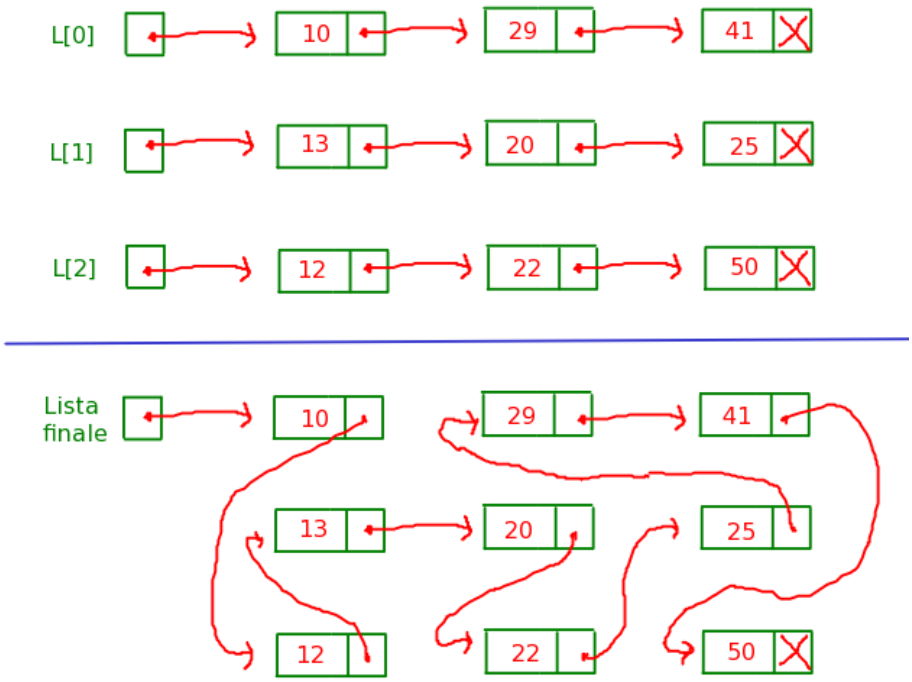
2 Esercizio: merging di k liste ordinate di n/k elementi

Usate la struttura dati *coda di priorità* (implementata efficientemente tramite heap) per risolvere il seguente problema in tempo $O(n \log k)$:

Input k liste ordinate $L_0, L_1, \ldots, L_{k-1}$, ciascuna con n/k elementi;

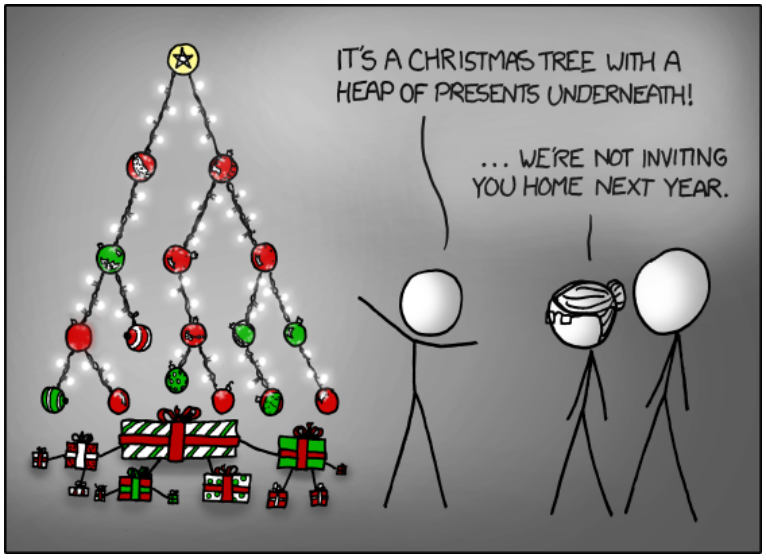
Output un'unica lista ordinata L contenente gli n elementi delle liste $L_0, L_1, \ldots, L_{k-1}$.

Suggerimento: non è necessario copiare gli elementi delle varie liste, basta modificare man mano i puntatori `next` in modo da costruire la nuova lista. Per mantenere l'ordinamento, è opportuno mantenere un puntatore alla coda delle lista finale, in modo da aggiungere i nuovi nodi in coda.



E visto che si avvicinano le feste...

Tratto dal mitico XKCD, https://xkcd.com/835/



Not only is that terrible in general, but you just KNOW Billy's going to open the root present first, and then everyone will have to wait while the heap is rebuilt.