Attività 12 – Coda a priorità e Heap

Per la risoluzione degli esercizi di questa lezione si richiede di implementare il proprio tipo di dato Coda a Priorità con le seguenti funzioni:

newPQ : *( ) → PRIORITYQUEUE*

emptyPQ : *(PRIORITYQUEUE) → BOOLEAN*

getMax : *(PRIORITYQUEUE) → ITEM*

deleteMax : *(PRIORITYQUEUE) → PRIORITYQUEUE*

insertPQ : *(PRIORITYQUEUE, ITEM) → PRIORITYQUEUE*

Dopodiché implementare le funzioni ai seguenti problemi che prevedono l'uso di Heap e di Code e Priorità:

1. **Min/Max**: dato un albero heap di nodi i cui item sono interi, realizzare due funzioni che restituiscano il valore del nodo massimo e del nodo minimo presenti nell'albero.

2. **Incrementa chiave**: dato un albero heap H i cui nodi sono item interi, un valore K e un valore newK (tale che K<newK), scrivere una funzione che modifichi l'albero H cercando il nodo il cui valore corrisponde a K e rimpiazzandolo con newK.
   Nota. il nodo incrementato potrebbe non trovarsi più in una posizione che rispetta la condizione dell'albero di essere uno heap.

3. **Heapify array:** dato un array di interi non ordinato, realizzare una funzione `int[] heapify(int [] a)` tale da restituire un array i cui elementi sono disposti in maniera tale da corrispondere ad uno heap.

4. **Stack con coda a priorità**: attraverso il tipo ADT Coda a Priorità, simulare una struttura dati di tipo stack che memorizza valori interi. La coda a priorità dovrà simulare opportunamente le funzioni per l'inserimento e l'estrazione di elementi nello stack (push e pop).

5. **Merge di code**: date due code a priorità Q1 e Q2, scrivere una funzione che crei una terza coda Q3 ottenuta per combinazione delle due code date in input
   `PQueue merge(PQueue q1, PQueue q2)`
   Le code originali non devono essere distrutte.

6. **Sequenza**: Data la seguente stringa *P R I O * R * * I * T * Y * * * Q U E * * * U * E*, leggere un carattere alla volta con la seguente regola: ogni volta che si incontra una lettera, questa viene inserita all'interno di una coda a priorità mentre ogni volta che si incontra un asterisco si effettua una cancellazione di un elemento dalla coda a priorità (la priorità è determinata dall'ordinamento alfabetico delle lettere, quindi la Z ha massima priorità di cancellazione sulla lettera A che invece ha la minima priorità). Scrivere una funzione che stampi gli elementi rimossi dalla coda secondo l'ordine di rimozione determinato dagli

asterischi.
Nota. la seguenza è solo di esempio, si può generalizzare a qualsiasi stringa composta da lettere e asterischi.

7. ***[avanzato]* E' uno Heap?**: Dato un albero binario BTree i cui item sono interi, scrivere una funzione che attraverso la restituzione di un valore booleano verifichi che esso sia un albero heap. `bool isHeap(BTree tree).`

8. ***[avanzato]* Priorità dei processi**: i sistemi operativi possono schedulare i processi in funzione della priorità ad essi assegnata. Implementare un tipo di dato *Processo* che rappresenta ogni processo con un ID univoco e un CPU-Burst di tipo intero (così come fatto per l'esercizio Round-Robin dell'esercitazione con le code). Dato un file contenente i dati dei processi da caricare in memoria <ID, CPU-Burst> crei una coda a priorità che consenta di schedulare i processi in ordine CPU-burst (*CPU-burst minori hanno precedenza maggiore*).

ESERCIZI TOTALMENTE OPZIONALI SULLE TABELLE HASH

1. **Conteggi**: Data un testo in input (ad esempio un articolo di giornale, una poesia ecc) catalogare le parole del testo per mezzo di una funzione hash che consenta di conteggiare il numero di parole con 2, 3, 4, 5…lettere. Come scegliere la funzione di hashing e quante dovrebbero essere le entry della tabella hash?

2. **Ricerca e modifica**: data una tabella hash H in input, una chiave K e una nuova chiave newK (tale che newK ≠ K), cercare nella tabella H se l'elemento di chiave K esiste e modificare la sua chiave con newK.
Per effetto di tale modifica, l'elemento potrebbe non trovarsi più nella entry opportuna.