Esercitazione di laboratorio n. 9

(Caricamento sul portale entro le 23.59 del 29/12/2017 di tutti gli esercizi)

Esercizio n. 1: Operazioni su alberi

Si scriva un programma in linguaggio C che esegua le seguenti operazioni su alberi binari di ricerca (BST), realizzati come ADT di I categoria:

- lettura/scrittura di un intero BST da/a file, in preorder
- conteggio del numero di nodi completi (cioè con 2 figli), compresi tra due livelli L_1 e L_2 dell'albero (livelli numerati a partire da 0)
- conteggio del numero di archi che separano due nodi dell'albero (si noti che i due nodi non sono necessariamente sullo stesso cammino radice-foglia)
- inversione dell'ordine (nel sottoalbero sinistro chiavi minori di quella della radice, nel sottoalbero destro chiavi maggiori), mediante una funzione `BSTmirror`.

Per svolgere questo esercizio, scrivere i file `BST.c`, `BST.h`, `Item.c` e `Item.h`, utilizzabili dal client `client.c` fornito.

Il tipo `Item` sia costituito da una stringa, che funge da chiave, e da un numero intero.

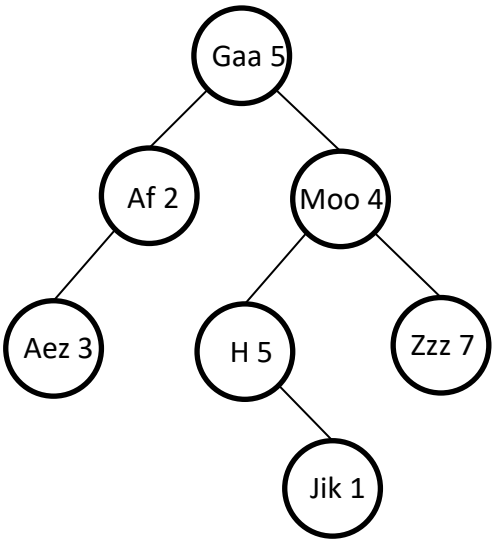
Il formato di file di ingresso accettato dalla libreria è composto da una n-pla, una per riga, che contiene per ogni nodo la chiave (stringa priva di spazi di al massimo 10 caratteri), l'intero, e una coppia di valori binari a indicare la presenza (1) o assenza (0) dei figli destro e sinistro, rispettivamente.

Si assuma che la chiave di un nodo sia unica e che il criterio di confronto tra chiavi sia quello della `strcmp`.

Per maggiore leggibilità, i due file di esempio sono caratterizzati da un'indentazione crescente all'aumentare della profondità di ogni nodo.

L'esempio proposto nel file di ingresso `alb1.txt` è riprodotto a seguire.

```
Gaa 5 1 1
        Af 2 1 0
                Aez 3 0 0
        Moo 4 1 1
                H 5 0 1
                        Jjk 1 0 0
                Zzz 7 0 0
```



Esercizio n. 2: Ranking

Si supponga di dover gestire la classifica di un torneo in cui valgono le seguenti regole:

- ogni partecipante entra nel torneo con 10 punti
- un partecipante può essere aggiunto al torneo in qualsiasi momento
- un partecipante può essere eliminato dal torneo in qualsiasi momento
- il torneo evolve facendo sfidare i due partecipanti col minor numero di punti:

- chi "vince" una sfida ottiene il 25% dei punti dell'avversario (approssimato all'intero superiore)
- chi "perde" una sfida perde il 25% dei propri punti (approssimato all'intero superiore)
- chi esaurisce i punti è eliminato dal torneo.

Si realizzi un programma C che, attraverso un'apposita interfaccia utente, permetta di gestire l'evoluzione della classifica del torneo, nelle condizioni descritte sopra, appoggiandosi ad una struttura dati di tipo coda a priorità (PQ). Il tipo `Item` contenuto nella coda sia progettato per memorizzare le informazioni relative a ogni partecipante ritenute opportune.

Le operazione permesse devono essere quelle di:

- stampa dello stato della classifica
- inserimento di un nuovo partecipante
- eliminazione di un partecipante
- evoluzione della classifica (una singola sfida, il cui esito è generato casualmente)
- caricamento di dati da file (in una coda inizialmente vuota)
- salvataggio dei dati su file.

In questo esercizio, il programma deve essere realizzato su tre moduli distinti:

- l'interfaccia utente (il `client`)
- un modulo `PQ` con le funzioni per la gestione della coda, realizzato come ADT di I categoria basato su heap
- un modulo `Item` con le funzioni per la gestione dei singoli dati, realizzato come ADT di I categoria (tipologia 4).

Nota:
Per la generazione di un numero casuale si consiglia di utilizzare la funzione `rand()`, che ritorna un numero intero casuale compreso tra 0 e `RAND_MAX`. Una risposta casuale `R` (in sostanza equiprobabile) tra 0 e 1 potrebbe essere generata con:

```
R = rand()  < RAND_MAX/2;
```

La funzione `rand()` genera in realtà un numero pseudo-casuale, in quanto, ripetendo il programma, si ripetono generano gli stessi numeri dell'esecuzione precedente (utilissimo, peraltro, per fare debug del programma).
Per evitare questo, occorre inizializzare il generatore di numeri casuali, ad ogni esecuzione, con un "seme" diverso dalla precedente esecuzione.
A tale scopo, includere la libreria `time.h` e aggiungere la seguente riga di codice in apertura del main:

```
srand((unsigned int)time(NULL));
```

Esercizio n. 3: Corpo libero
Il corpo libero è una specialità della ginnastica artistica/acrobatica in cui l'atleta deve eseguire una sequenza di elementi senza l'ausilio di attrezzi, ad eccezione della pedana di gara. Ogni elemento è caratterizzato da un corrispettivo punteggio atteso (da 0.1 a 3.0) e relativo grado di difficoltà (da 1 a 6). Gli elementi si dividono in tre categorie: non acrobatici, acrobatici avanti e acrobatici indietro.

L'elenco degli elementi tra cui è possibile scegliere è riportato in un file di testo (`elementi.txt`), il cui formato è il seguente:

- sulla prima riga è riportata una terna di interi `NA AA AI` a rappresentare il numero di esercizi per ognuna delle tre categorie presenti in elenco
- seguono tre sezioni, una per categoria, con i dettagli di ogni elemento disponibile in ragione di uno per riga, per un totale di `NA+AA+AI` righe. Ogni elemento è descritto da una terna `<nome> <grado> <punti>`.

Le capacità di un atleta sono descritte da quattro valori: `maxNA`, `maxAA`, `maxAI` e `maxSUM`. I primi tre rappresentano il massimo grado di difficoltà eseguibile per ogni categoria. L'ultimo valore è la massima difficoltà complessiva (come somma) di tutti gli esercizi portati in gara.
Ad esempio, la quaterna `3 4 2 20` rappresenterebbe la situazione in cui l'atleta sa presentare elementi non acrobatici fino al grado 3, elementi acrobatici avanti fino al grado 4, elementi acrobatici indietro fino al grado 2 e al massimo esercizi per una difficoltà complessiva pari a 20.

Per la composizione del programma di gara, si considerino le seguenti regole:

- ogni atleta deve presentare esattamente 8 elementi
- sono ammessi elementi ripetuti, ma non comportano punti
- un atleta non può portare in gara difficoltà al di fuori delle sue possibilità
- se l'atleta porta elementi di almeno due categorie diverse, riceve un bonus fisso di 2.5 punti (*esigenza di composizione*)
- se l'atleta termina il programma con un elemento acrobatico, avanti o indietro, di grado almeno 5, riceve un bonus fisso di 2.5 punti (*esigenza di uscita*)
- le due esigenze non possono essere soddisfatte dal medesimo elemento
- il programma di gara dell'atleta deve avere il punteggio atteso più alto possibile.

Scrivere un programma in C che letto il file di elementi e acquisiti i parametri dell'atleta, generi una possibile routine di gara che rispetti tutti i vincoli di cui sopra.

Si risolva il problema mediante:

- un algoritmo completo, quindi in grado di individuare un programma di gara ottimale
- uno o più algoritmi greedy, definendo opportune funzioni obiettivo.