



Esercitazione di laboratorio n. 8

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

Esercizio n. 1: Koala

(Rivisitazione del) Tema d'esame del 18/09/2017 – Traccia da 18 punti

Sugli alberi di una foresta di eucalipti vivono dei koala, appartenenti a famiglie diverse, che possono essere tra di loro nemiche. Le seguenti informazioni descrivono la situazione:

- i koala sono N e sono identificati con interi tra 0 e $N-1$. Per ogni koala è dato l'insieme degli alberi su cui può vivere. Ogni koala appartiene a una e una sola famiglia
- gli eucalipti sono T e sono identificati con interi tra 0 e $T-1$. Ogni eucalipto può ospitare un numero massimo m noto di koala: tale numero è lo stesso per tutti gli alberi. Nella foresta c'è certamente posto per tutti i koala ($mT \geq N$). È possibile che su un albero non viva nessun koala
- le famiglie sono S e sono identificate con interi tra 0 e $S-1$. Sono note le coppie di famiglie nemiche.

Le informazioni di cui sopra sono memorizzate in 3 file:

1. `hab.txt`: descrive l'habitat. La prima riga contiene il numero N di koala, seguono N blocchi in cui sulla prima riga di ciascuno è presente il numero E_k di alberi su cui ogni k -esimo koala può vivere seguito dall'elenco su righe separate degli identificativi degli alberi su cui può vivere
2. `fam.txt`: descrive le famiglie. La prima riga contiene il numero S di famiglie, seguito da S blocchi in cui sulla prima riga di ciascuno c'è il numero K_f di koala appartenenti alla f -esima famiglia seguito su righe separate dall'elenco di koala che vi appartengono
3. `nem.txt`: descrive le incompatibilità. Su righe separate compaiono gli identificativi delle coppie di famiglie incompatibili.

Si assuma che i file siano corretti.

Esempio:

hab.txt	Spiegazione	fam.txt	Spiegazione	nem.txt	Spiegazione
6	6 koala in tutto	3	3 famiglie in tutto	0 1	famiglie 0 e 1 nemiche
2	il koala 0 può vivere su 2 alberi	2	famiglia 0 con 2 membri	1 2	famiglie 1 e 2 nemiche
1	l'albero 1	1	il koala 1
4	l'albero 4	3	e il koala 3		
1	il koala 1 può vivere su 1 albero	1	famiglia 1 con 1 membro		
0	l'albero 0	0	il koala 0		
4	il koala 2 può vivere su 4 alberi		
...					

Si scriva un programma C che, ricevuto sulla riga di comando il valore di m (numero massimo di koala che possono vivere su un eucalipto), determini, se esiste, un'allocazione dei koala tra gli alberi nelle 3 seguenti configurazioni:

- ignorando i vincoli imposti dal file `hab.txt`, ossia supponendo che tutti i koala possano vivere su uno qualsiasi degli alberi, fino a un numero massimo m di koala su ogni albero, tenendo conto dei vincoli di incompatibilità tra famiglie
- considerando tutti i vincoli imposti dai file di input.



Suggerimento: come operazione preliminare si identifichi il tipo di problema in riferimento a quanto visto a lezione (modelli del Calcolo Combinatorio, powerset, partizioni).

Esercizio n. 2: Questioni di equilibrio

Si consideri una sequenza di N frecce disposte lungo una linea orizzontale, in cui ogni freccia può puntare a destra oppure a sinistra, come illustrato nella figura seguente.



Si assuma che N sia sempre un numero pari e che la freccia più a sinistra occupi la posizione 0, mentre la freccia più a destra la posizione $N-1$.

- Si definisce uno *scontro* quando due frecce consecutive puntano una verso l'altra. Nell'esempio precedente le frecce in posizione 0 e 1 e le frecce in posizione 4 e 5 determinano due scontri
- dato uno scontro, la sequenza più lunga di frecce orientate nello stesso verso della freccia a sinistra (destra) dello scontro, rappresentano la parte sinistra (destra) dello scontro
- uno scontro è in equilibrio se la parte destra e la parte sinistra dello scontro sono lunghe uguali. Nell'esempio di riferimento lo scontro (0,1) è equilibrato, mentre lo scontro (4,5) non lo è poiché la parte sinistra è lunga 3 mentre la parte destra è lunga 1
- una sequenza di frecce è equilibrata se tutti gli scontri al suo interno sono equilibrati.

Come formato di input, si assuma che il valore 0 indichi una freccia orientata verso destra, mentre il valore 1 indichi una freccia orientata verso sinistra. La sequenza di 0 e di 1 è sempre preceduta dal numero di frecce prese in considerazione. L'esempio precedente risulterebbe quindi codificato come segue: 6 0 1 0 0 0 1. In questo esempio le condizioni imposte dal problema sarebbero rispettate scambiando di verso alla freccia in posizione 4.

Si scriva un programma in C in grado di determinare un insieme minimo di frecce da *girare* così che ogni freccia partecipi a uno scontro e ogni scontro sia equilibrato, ossia per far sì che l'intera sequenza sia equilibrata.

Si risolva il problema proponendo due soluzioni alternative, quali:

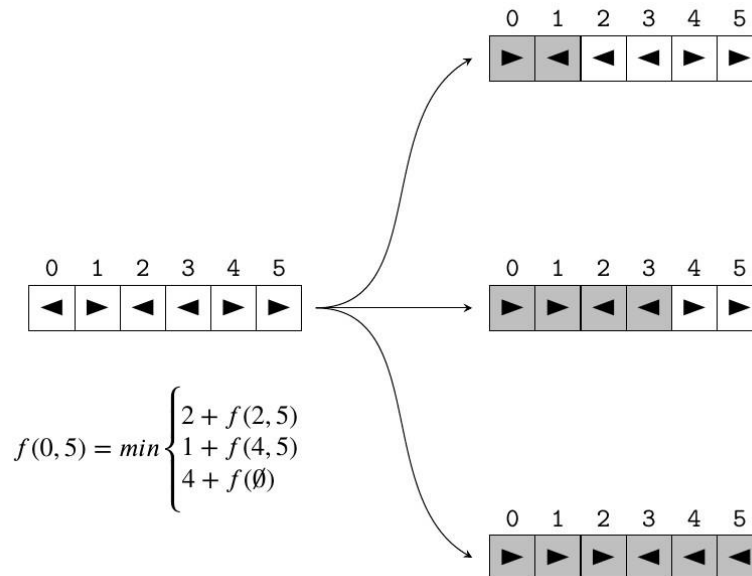
- una soluzione che implementi il modello combinatorio ritenuto più adatto, tra quelli visti a lezione
- una soluzione che sfrutti il paradigma della programmazione dinamica

Suggerimento

Per quanto riguarda la soluzione basata su programmazione dinamica, si tengano a mente le seguenti indicazioni:

- una sequenza vuota è equilibrata per definizione
- ogni sequenza equilibrata non vuota è composta da uno scontro equilibrato seguito da una sequenza equilibrata
- dato uno scontro non equilibrato, il minimo numero di scambi necessari per portarlo in equilibrio è calcolabile in $O(n)$

Come ulteriore suggerimento, si consideri la seguente rappresentazione grafica da cui derivare la scomposizione del problema originale in sottoproblemi. In tale rappresentazione, con la dicitura $f(a,b)$ indichiamo il sottoproblema relativo alle frecce dalla posizione a alla posizione b .



Esercizio n. 3: Atleti (piano di allenamenti multi-file)

Si consideri lo scenario introdotto nell'esercizio n.2 del laboratorio 7 (piano di allenamenti).
Si riorganizzi il codice scritto precedentemente suddividendolo in più moduli, quali:

- un modulo client contenente il main con l'interfaccia utente/menù
- un modulo per la gestione degli atleti
- un modulo per la gestione degli esercizi

Il modulo per gli atleti deve fornire le funzionalità di:

- acquisizione da file delle informazioni relative agli atleti, mantenendo la medesima struttura a lista richiesta nel laboratorio precedente
- inserimento/cancellazione di un atleta
- ricerca per codice di un atleta
- ricerca per cognome (anche parziale) di un atleta
- stampa dei dettagli di un atleta (incluso il piano di allenamento, se presente)
- acquisizione da file del piano di allenamento per un atleta (individuato mediante codice)
- modifica del piano di allenamento di un atleta
 - aggiunta/cancellazione di un esercizio
 - modifica del numero di set per un dato esercizio già presente nel piano
 - modifica del numero di ripetizioni per un dato esercizio già presente nel piano

Il modulo per gli esercizi deve fornire le funzionalità di:

- acquisizione da file delle informazioni relative agli esercizi, mantenendo la medesima struttura a vettore richiesta nel laboratorio precedente
- ricerca di un esercizio per nome
- stampa dei dettagli di un esercizio



Esercizio n. 4: Atleti (piano di allenamenti multi-file, con ADT)

A partire dal codice prodotto per l'esercizio n. 3, si adatti il codice così che sia il modulo `atleti` sia il modulo `esercizi` risultino ADT di prima classe.

Si tenga conto che esistono

- un tipo di dato per un esercizio e un tipo di dato per un vettore di esercizi
- un tipo di dato per un atleta e un tipo di dato per una lista di atleti
- un tipo di dato per una lista di (riferimenti a) esercizi

Si chiede di realizzare tutti i tipi di dato come ADT, utilizzando

- la versione “quasi ADT” (struct visibile) per i tipi esercizio (`es_t`) e atleta (`atl_t`)
- la versione “ADT di I classe” per le collezioni, cioè il vettore di esercizi (`esArray_t`), la lista di atleti (`atlList_t`) e le liste di riferimenti a esercizi (`esList_t`).

Per i riferimenti ad esercizi di evitare i puntatori, in quanto richiederebbero accesso a una struttura dati interna (all'ADT vettore di esercizi). Si utilizzino invece gli indici: ad un dato esercizio si fa quindi riferimento mediante un intero (progressivo a partire da 0) che ne identifica la posizione nel vettore di esercizi.

Pur potendo unificare l'ADT esercizio e il vettore esercizi in un unico modulo, come pure l'ADT atleta e l'ADT lista di atleti, si consiglia di realizzare un modulo per ognuno degli ADT. Si realizzeranno quindi 5 file `.c` (`atl.c`, `es.c`, `atlList.c`, `esArray.c`, `esList.c`) e 5 file `.h` (`atl.h`, `es.h`, `atlList.h`, `esArray.h`, `esList.h`) per i moduli, più un `.c` per il client (ad esempio `allenamenti.c`).

I moduli `atl` ed `es` realizzano tipi di dato composti per valore (sono quindi simili al tipo/ADT `Item` spesso usato in altri problemi): saranno ovviamente di dimensione ridotta.

I moduli per collezioni di dati (`atlList.c`, `esArray.c`, `esList.c`) dovranno fornire operazioni di creazione/distruzione, più eventuali operazioni di input o output, ricerca, modifica e/o cancellazione. Eventuali altre operazioni possono essere fornite qualora ritenute necessarie.

ATTENZIONE

Si ricorda che un ADT di I classe NASCONDE i dettagli interni di un dato: NON E' QUINDI POSSIBILE A UN CLIENT ACCEDERE A TALI DETTAGLI: non sarà possibile, quindi la modifica di un esercizio (numero di set e/o ripetizioni) di un dato atleta da parte di un client in modo diretto, cioè ottenendo il puntatore all'elemento in lista, per accedere ai campi da modificare. Il client dovrà quindi, ad esempio, chiamare una opportuna funzione (fornita dal modulo `atlList`) `modificaEsAtl`, avente come parametri il codice di un atleta, il nome dell'esercizio e i nuovi valori per set/ripetizioni. Tale funzione, all'interno, effettuerà una ricerca dell'atleta e ne modificherà l'esercizio selezionato NON in modo diretto, ma chiamando, ad esempio, una funzione (fornita dal modulo `esList`) `modificaEs`, che riceve come parametri il nome dell'esercizio e i valori da aggiornare. Quest'ultima funzione cerca l'esercizio in una lista e ne modifica i valori.

NOTA BENE: da svolgere dopo aver completato l'esercizio n.3 e in seguito alle lezioni della settimana che inizia il 4/12 in cui si completeranno gli ADT.