

# INFORMATICA GENERALE

## Homework di Recupero 2016

docente: IVANO SALVO  
Sapienza Università di Roma

### Gruppo 1

**Esercizio 1.1** Scrivere un programma C che presi in input due interi positivi  $a$  ed  $b$  ( $a, b > 0$ ) calcoli il logaritmo intero di  $a$  in base  $b$ .

Più formalmente, il programma deve restituire in output un numero intero  $k$ , per cui sia verificata la disuguaglianza  $b^k \leq a < b^{k+1}$ .

ESEMPI: Presi in input 63 e 2, il programma deve stampare 5, in quanto  $2^5 = 32 \leq 63 < 64 = 2^6$ . Presi in input 81 e 3, il programma deve stampare 4, in quanto  $3^4 = 81 < 243 = 3^5$ .

**Esercizio 1.2** Un intero positivo  $n$  si dice *perfetto* se è uguale alla somma di tutti i suoi divisori propri. Ad esempio,  $28 = 1 + 2 + 4 + 7 + 14$  è perfetto.  $n$  si dice *abbondante* se la somma di tutti i suoi divisori propri è maggiore di  $n$ . Ad esempio  $24 < 1 + 2 + 3 + 4 + 6 + 8 + 12 = 36$  è abbondante.  $n$  si dice *difettivo* se la somma di tutti i suoi divisori propri è minore di  $n$ . Ad esempio  $27 > 1 + 3 + 9 = 13$ .

Scrivere una programma C che preso in input un numero positivo  $n$  stampi 0 se  $n$  è un numero perfetto,  $-1$  se è difettivo ed 1 se è abbondante.

**Esercizio 1.3** Dato un intero strettamente positivo  $n$ , definiamo il *successivo* di un naturale  $n > 1$  come segue (1 non ha successivi):

$$\begin{array}{ll} n/2 & \text{se } n \text{ è pari} \\ 3n + 1 & \text{se } n \text{ è dispari } > 1 \end{array}$$

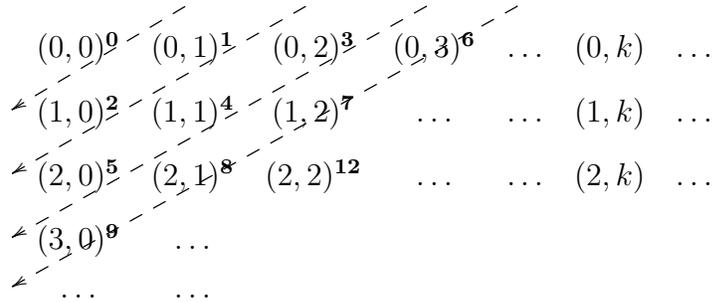


Figura 1: codifica *a coda di rondine* di Cantor dei “razionali”

Scrivere un programma che letto in input un numero intero positivo  $n$  produca in output il numero di passi necessari affinché la sequenza che comincia con  $n$  raggiunga il numero 1.

ESEMPIO: La sequenza che comincia con 3, prosegue con 10, 5, 16, 8, 4, 2, 1: quindi in questo caso il programma dovrebbe stampare 7.

**Esercizio 1.4** (COPPIE). Possiamo codificare le coppie di naturali come segue. Prima codifichiamo l’unica coppia di somma 0  $(0,0)$  con il numero 0, poi le coppie di somma 1, codificando con 1 la coppia  $(0,1)$  e con 2 la coppia  $(1,0)$ , poi tutte le coppie di somma 2, assegnando 3 a  $(0,2)$ , 4 a  $(1,1)$  e 5 a  $(2,0)$  e così via (vedi Figura 1). Il vostro programma dovrà leggere una coppia di interi e produrre in output la corrispondente codifica. Questo esercizio vi faccia riflettere sul fatto che l’insieme dei  $\mathbb{N}$  dei naturali è equipotente all’insieme  $\mathbb{Q}$  dei razionali [**Sugg:** riflettete sulla codifica delle coppie nella forma  $(0, k)$ ].

INPUT: Due numeri interi, da leggersi con due `scanf("%d", &m)`.

OUTPUT: Un numero intero che rappresenta relativa codifica.

## Gruppo 2

**Esercizio 2.1** (ENIGMISTICA: LUCCHETTO) Nel linguaggio enigmistico un *lucchetto* è un gioco caratterizzato dallo schema  $XY + YZ \rightarrow XZ$ .

Forse un informatico teorico direbbe piuttosto che prese tre parole (o sequenze)  $w_1, w_2, w_3$  su un certo alfabeto di simboli  $\Sigma$ ,  $w_3$  è un lucchetto di  $w_1$  e  $w_2$  se e solo se esistono tre parole  $x, y, z$  tale che  $w_1 = xy$ ,  $w_2 = yz$  e  $w_3 = xz$ .

Dovete scrivere un programma che legge due stringhe e produce in output il *lucchetto massimale* tra le due parole. Il programma deve quindi trovare la *più lunga* sequenza finale di  $w_1$  che è uguale alla sequenza iniziale di  $w_2$ , e produrre in output il lucchetto risultante seguendo le regole sopra descritte

(senza ovviamente preoccuparsi che il risultato abbia significato o meno nella lingua italiana).

ESEMPI: Se le parole di ingresso sono uguali, il risultato è la sequenza vuota. Se non esiste nessun suffisso della prima comune a un prefisso della seconda, allora il risultato è semplicemente la concatenazione delle due parole. Ad esempio se l'input fosse: **rara** e **mente** il vostro programma dovrebbe scrivere in output **raramente**. Attenzione però che in alcuni casi il lucchetto non è unico, ma voi dovete restituire il massimale. Ad esempio se l'input fosse **tono** e **onore** dovete scrivere in output **tre** e non **tonnore**.

**Esercizio 2.2** (ANAGRAMMI) Scrivere un programma C che legge in input due stringhe  $s_1$  ed  $s_2$  e verifica se  $s_1$  è un anagramma di  $s_2$ . In caso affermativo, stampa 1, altrimenti stampa 0. Attenzione al fatto che eventuali caratteri ripetuti *devono avere lo stesso numero di occorrenze* in  $s_1$  ed  $s_2$ .

ESEMPI: La stringa *roma* è anagramma di *mora*, ma non di *morra*.

**Esercizio 2.3** (MASTERMIND) Scrivere un programma che legge in input 20 cifre (una alla volta con `scanf("%d", ...)`;) e considera le prime dieci come una combinazione segreta del gioco del master mind, e le seconde 10 come un tentativo di indovinare la combinazione segreta. Produrre in output 2 numeri (su due righe diverse), tali che il primo sia il numero delle cifre corrette al posto giusto, e il secondo sia il numero delle cifre corrette, ma al posto sbagliato.

ESEMPIO: Supponiamo venga letta la seguente sequenza di cifre: 1 2 3 4 5 5 4 3 2 1 1 6 7 2 1 5 5 3 1 4. Va interpretata come segue:

combinazione segreta:	1	2	3	4	5	5	4	3	2	1
tentativo:	1	6	7	2	1	5	5	3	1	4
risultato:	●			○	○	●	○	●		○

dove ● significa “numero giusto al posto giusto” e ○ significa “numero giusto al posto sbagliato”. Osservate che l'ultimo 1 del tentativo non produce nessun risultato perchè ho già “impegnato” entrambi gli 1 della combinazione segreta. In questo caso il vostro programma dovrebbe rispondere quindi:

3  
4

**Esercizio 2.4** (MEDIANA) Scrivere un programma che legge un intero  $n > 0$ , un intero  $k$  ( $0 \leq k < n$ ) e poi una sequenza di  $n$  numeri interi  $a_1 \dots a_n$  e

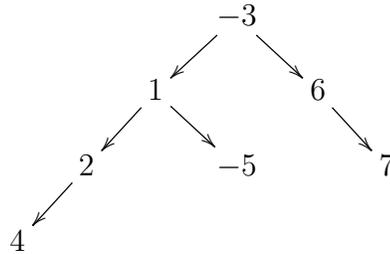


Figura 2: Albero binario nell'esempio

restituisce come output il numero della sequenza  $a_j$  tale che la cardinalità dell'insieme  $\{a_i \leq a_j \mid i = 1, \dots, n, i \neq j\}$  sia esattamente  $k$ .

ESEMPIO: Supponiamo venga letto il numero 8, poi il numero 3 e poi gli 8 numeri 2, 11, 3, 4, 23, 12, 100, 9.

Il programma deve rispondere 9, in quanto ci sono 3 numeri minori di lui (2, 3 e 4).

SFIDA (DIFFICILE): Scrivere un algoritmo lineare basato sul *divide et impera*, ispirato alla funzione *partiziona* usata nell'algoritmo di ordinamento *quickSort*.

## Gruppo 4

L'input degli esercizi di questa sezione è costituito da un numero intero  $n$  che rappresenta il numero di nodi di un albero binario  $T$  di interi e poi da  $2n$  interi, di cui i primi  $n$  sono il risultato di una visita *preorder* di  $T$  e i secondi  $n$  sono gli stessi interi, nell'ordine ottenuto da una visita *inorder* di  $T$  (vedi Homework 3). Sotto l'ipotesi che tutti i nodi *contengano informazioni distinte*, è possibile ricostruire in modo univoco l'albero binario  $T$ .

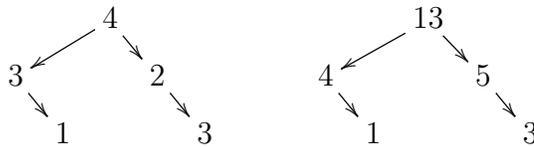
Infatti, la visita *preorder* ci permette di identificare in modo univoco la radice (che è il primo elemento della visita). Nella visita *inorder* la radice separa gli elementi del sottoalbero sinistro da quelli del sottoalbero destro. A questo punto, è possibile ricostruire il sottoalbero sinistro e destro riapplicando ricorsivamente lo stesso ragionamento, dopo aver opportunamente selezionato le sequenze di elementi che rappresentano le visite *preorder* e *inorder* del sottoalbero sinistro e del sottoalbero destro.

ESEMPIO: Supponiamo di leggere come input il numero 7 e 14 numeri nel seguente ordine: -3 1 2 4 -5 6 7 4 2 1 -5 -3 6 7. L'albero corrispondente è mostrato in Fig. 3. Infatti, la visita *preorder* ci dice che -3 è la radice. Dopodichè, andando a vedere la visita *inorder*, è possibile scoprire che 4 2

1 -5 è la visita in order del sottoalbero sinistro e 6 7 è la visita in order del sottoalbero destro. Dato che anche nella visita preorder la visita del sottoalbero sinistro precede la visita del sottoalbero destro, possiamo dedurre che 1 2 4 -5 sia la visita preorder del sottoalbero sinistro e 6 7 la visita preorder del sottoalbero destro. Ora è possibile riapplicare ricorsivamente il ragionamento considerando la coppia di visite 4 2 1 -5 e 1 2 4 -5 per ricostruire il sottoalbero sinistro e la coppia di visite 6 7 e 6 7 per il sottoalbero destro.

**Esercizio 4.1** Leggete un albero di interi come descritto sopra. In output, occorre produrre l'albero in cui l'etichetta di ciascun nodo stata sostituita con la somma delle etichette sottoalbero radicato in quel nodo. Produrre come output la sequenza di interi corrispondente a una visita *inorder* dell'albero.

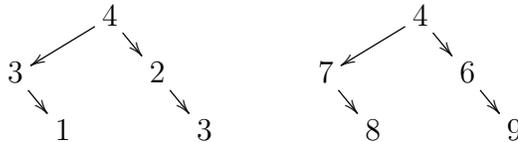
ESEMPIO: Dato in input l'albero a sinistra, bisogna costruire l'albero a destra e produrre in output la sua visita in order, senza andare a capo.



OUTPUT: 4 1 9 5 3.

**Esercizio 4.2** Leggete un albero di interi come descritto sopra. In output, occorre produrre l'albero in cui l'etichetta di ciascun nodo stata sostituita con la somma delle etichette nel cammino che lega il nodo alla radice. Produrre come output la sequenza di interi corrispondente a una visita *postorder* dell'albero.

ESEMPIO: Dato in input l'albero a sinistra, bisogna costruire l'albero a destra e produrre in output la sua visita postorder, senza andare a capo.



OUTPUT: 8 7 9 6 4.

**Esercizio 4.3** Diremo che un albero binario di interi  $A$  è *minore per tracce* di un albero di interi  $B$  (notazione  $A \preceq B$ ) se per ogni sequenza di etichette  $p$

in un cammino di  $A$ , esiste un cammino in  $B$  in cui incontro la stessa sequenza di etichette  $p$ .

Gli alberi  $A$  e  $B$  sono *equivalenti per tracce* se e solo se  $A \preceq B$  e  $B \preceq A$ .

Scrivere un programma che legge 2 alberi binari di interi e che ritorna 1 se gli alberi in ingresso  $A$  e  $B$  sono equivalenti per tracce e 0 altrimenti.

ESEMPI: Presi gli alberi in Fig. 3, abbiamo chiaramente che  $A \preceq C$  e  $A \preceq B$ . Osservate che la sequenza di etichette  $\langle 11, 23, 19 \rangle$  si trova nell'albero  $B$  nel cammino contenente le etichette  $\langle 11, 23, 19, 42 \rangle$ . Questo cammino è il controesempio che mostra che  $B \not\preceq A$  e  $B \not\preceq C$ . Infine anche  $C \preceq A$  e quindi  $A$  e  $C$  sono equivalenti per tracce.

SUGGERIMENTO: Organizzare il codice in funzioni che risolvono sottoproblemi più semplici. Scrivere una funzione `C` di prototipo: `int eqTrace(binTree A, binTree B)`; che a sua volta chiama una funzione che implementa  $\preceq$ .

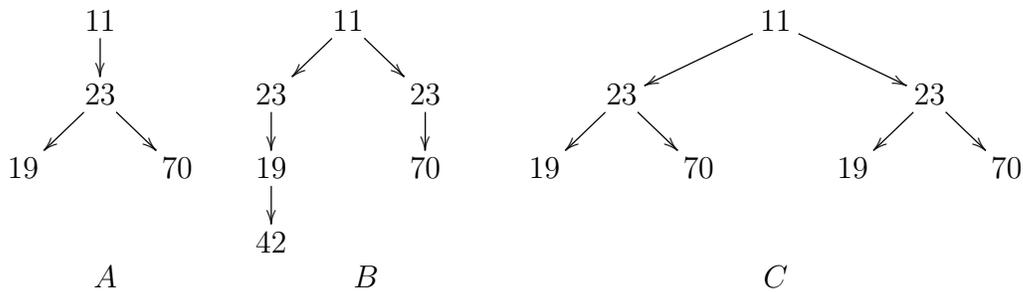


Figura 3: Alberi binari di esempio