

INFORMATICA GENERALE

Homework di Recupero 2018

docente: IVANO SALVO
Sapienza Università di Roma
email: `salvo@di.uniroma1.it`

Istruzioni per l'Homework di Recupero

Ogni studente deve totalizzare 5 punti per superare gli Homework. Chi non ha fatto nessun Homework, deve scegliere 5 esercizi (pescando in tutti e 3 i gruppi *almeno* un esercizio e *mai più* di due esercizi da ciascun gruppo).

Chi ha fatto gli Homework, ma mancano un certo numero di punti, dovrà consegnare tanti esercizi quanti sono i punti che gli mancano. Dovrebbe anche scegliere dal gruppo corrispondente agli Homework mancanti. Ad esempio, se uno studente fosse andato male nell'Homework **3** (Alberi), dovrebbe fare gli esercizi dal Gruppo **3** (Alberi).

Gli Homework vanno **consegnati al docente via mail**, in qualunque momento, ma *almeno un paio di giorni prima* di un eventuale appello orale. A differenza degli homework dati durante l'anno, occorre scrivere *programmi completi*, con tanto di main ed input/output.

Gruppo 1 (Naturali)

Esercizio 1.1 Scrivere un programma C che presi in input due interi positivi a ed b ($a, b > 0$) calcola 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):

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

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 (Vettori)

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 Σ , 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 $w_3 = xz$.

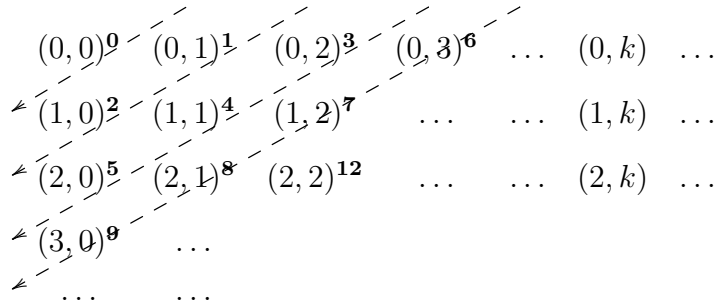


Figura 1: codifica *a coda di rondine* di Cantor dei “razionali”

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 Considerare il problema di trovare il sotto-vettore di somma massima in un vettore di numeri. Il problema è interessante solo se il vettore contiene sia numeri positivi che negativi (se contenesse solo positivi, il sotto-vettore di somma massima è il vettore stesso, mentre se contenesse solo negativi, sarebbe il vettore vuoto). Ad esempio, dato il vettore in Fig. 2, il sotto-vettore di somma massima è quello compreso tra gli indici 2 e 6 (compresi).

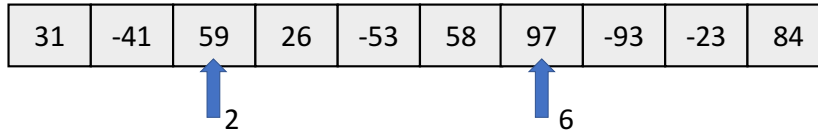


Figura 2: Esempio di vettore e relativo sotto-vettore di somma massima.

Scrivere una funzione `int svSommaMax(int a[], int n, int* b, int* e)` che dato il vettore a di interi di lunghezza n , restituisce la somma del sotto-vettore di somma massima e carica nelle variabili b ed e gli indici di dove comincia e finisce tale sotto-vettore (seguendo il galateo del C, nel nostro esempio, si dovrebbe tornare 187 e caricare rispettivamente 2 e 7 in b ed e).

Esercizio 2.4 [MATRICE A SPIRALE]. Scrivere un programma che legge un numero intero n e crea una matrice a spirale $n \times n$.

Si mette in alto a sinistra (in posizione $[0,0]$) il numero 1 e poi, andando in senso *orario* verso il centro della matrice, si dispongono gli altri numeri 2,3,4, ... fino a n^2 .

Ecco ad esempio le matrici a spirale di lato 3 e 4.

| | | | | | | |
|---|---|---|----|----|----|---|
| 1 | 2 | 3 | 1 | 2 | 3 | 4 |
| 8 | 9 | 4 | 12 | 13 | 14 | 5 |
| 7 | 6 | 5 | 11 | 16 | 15 | 6 |
| | | | 10 | 9 | 8 | 7 |

Input: un numero intero n , dimensione della matrice.

Output: la matrice a spirale $n \times n$.

Gruppo 3 (Alberi)

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

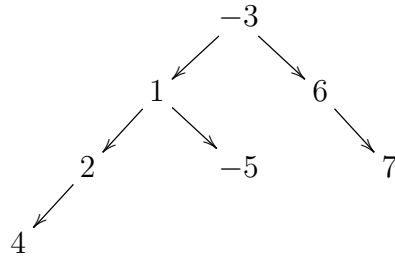


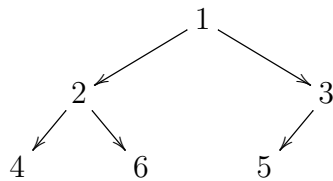
Figura 3: Albero binario nell'esempio

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. 4. 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 inorder del sottoalbero sinistro e $6\ 7$ è la visita inorder 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 Scrivere un programma che leggendo una sequenza di interi (chiusa da uno zero), memorizza gli interi della sequenza in un albero binario bilanciato. Procedere nel seguente modo: mettere il primo elemento nella radice dell'albero e poi ricorsivamente gli elementi di posto pari nella sequenza nel sottoalbero sinistro e quelli di posto dispari (eccetto il primo, memorizzato nella radice) nel sottoalbero destro.

ESEMPIO: Leggendo la sequenza $\langle 1, 2, 3, 4, 5, 6 \rangle$, deve essere prodotto il seguente albero:



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 sottoalbero radicato in quel nodo. Produrre come output la sequenza di interi corrispondente a una visita *inorder* dell'albero.

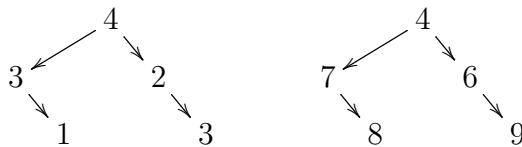
ESEMPIO: Dato in input l'albero a sinistra, bisogna costruire l'albero a destra e produrre in output la sua visita *inorder*, senza andare a capo.



OUTPUT: 4 1 9 5 3.

Esercizio 4.3 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 4.4 Diciamo che un albero binario di interi è *crescente* se percorrendo ogni cammino dell'albero si incontrano etichette ordinate in modo crescente.

Scrivere una funzione `C` di prototipo: `int prune(binTree B);` che ritorna 1 se l'albero B è crescente e 0 altrimenti. Quando B non è crescente, la funzione `prune` trasforma B nel suo massimo prefisso crescente.

OSSERVAZIONI: osservate che il puntatore alla radice di B non viene mai modificato dalla funzione `prune`, in quanto, alla peggio, la sola radice è sempre un prefisso crescente di B .

Possibilmente scrivete la funzione `prune` in modo che i sottoalberi cancellati vengano deallocati.

ESEMPI: Considerate gli alberi in Fig. 4 (prima riga). L'albero A è crescente e quindi la funzione `prune` ritorna 1 e lo lascia immutato. Viceversa, la funzione `prune` risponde 0 su B e C e li trasforma rispettivamente negli alberi B' e C' .

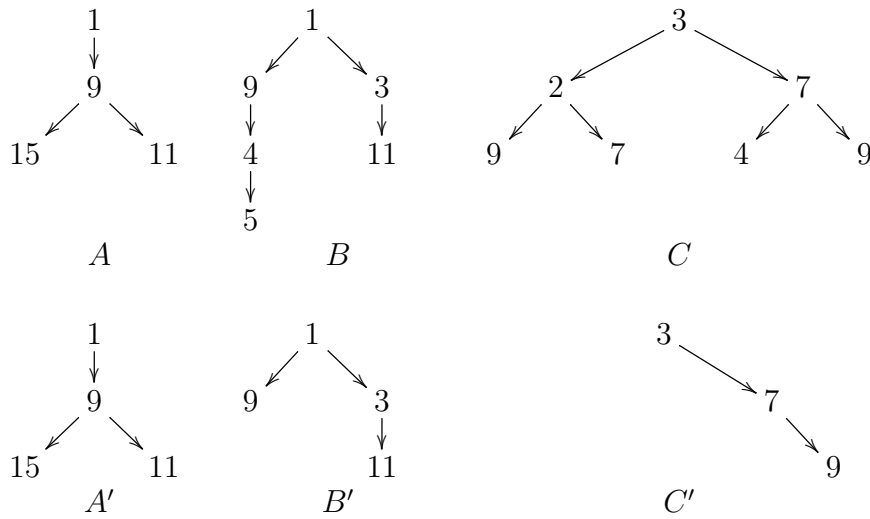


Figura 4: Alberi binari di esempio per l'esercizio 4.3