Commento al Laboratorio n.0

Esercizio n. 1: Sequenze numeriche in vettore

Si estendono le specifiche prevedendo la possibilità di utilizzare i dati di un vettore default oppure di leggerli da tastiera. Si presentano 3 soluzioni richiamabili dallo stesso `main` sulla base di un valore di selezione passato come argomento alla riga di comandi.

Soluzione 1: si interpreta il problema come `n` problemi di ricerca di sottovettori lunghezza nota e decrescente. Non appena si trova una soluzione con una certa lunghezza è certo che questa sia la lunghezza massima.
Si procede mediante un ciclo `for` alla ricerca di sottovettori di lunghezza decrescente (da un massimo di `n` a un minimo di 1) con l'accortezza di bloccare la ricerca di sottovettori di lunghezza inferiore a quella appena identificata. Questo si realizza mediante il flag `trovato`. Il ciclo `for` interno verifica, per ogni possibile indice di partenza di sottovettore, l'esistenza di un sottovettore di elementi non nulli della lunghezza corrente. Se la verifica ha esito positivo, si setta il flag trovato per impedire la ricerca di altri sottovettori che avranno certamente lunghezza minore, si stampa il sottovettore trovato e si procede dal prossimo indice di inizio immediatamente successivo al sottovettore stampato. Questo perché i sottovettori non possono avere intersezione tra di loro, essendo sempre separati tra loro da almeno un dato nullo.
L'algoritmo ha complessità quadratica nella dimensione del vettore.

Soluzioni 2 e 3: si interpreta il problema come problema di ottimizzazione, ancorché molto semplice. I problemi di ottimizzazione si risolvono enumerando tutte le possibili soluzioni, valutando per ciascuna di esse la funzione obiettivo (in questo caso la lunghezza del sottovettore di elementi non nulli) e tenendo traccia della soluzione correntemente migliore, che al termine sarà la soluzione ottima che verrà stampata. Si ricordi che nei problemi di ottimizzazione sono possibili più soluzioni, diverse tra di loro, ma che condividono lo stesso valore (minimo o massimo) della funzione obiettivo. In questo caso è richiesto che si stampino tutte. La ricerca del massimo è un problema di ottimizzazione molto semplice che si risolve in maniera simile alla verifica con quantificatori mediante un'iterazione sui dati in cui si gestisce un massimo provvisorio opportunamente inizializzato, aggiornato ogni volta che si trova una soluzione che lo migliora.

Soluzione 2: si procede in 2 passi:

1. identificazione della la lunghezza massima di sottovettori di elementi non nulli. Per fare questo mediante una scansione lineare si identificano i sottovettori di elementi non nulli, se ne calcola la lunghezza e si tiene traccia della lunghezza massima. A tale scopo si memorizza in `i0` l'indice di inizio del sottovettore, in `i` l'indice a cui termina e di conseguenza se ne calcola la lunghezza come `i - i0`. Si confronta la lunghezza con quella massima trovata fino al momento (valore di inizializzazione 0) e, se è il caso, si aggiorna il massimo.
2. stampa dei soli sottovettori a lunghezza massima: si procede come al passo precedente nell'identificazione dei sottovettori e dellaloro lunghezza, ma li si filtra, stampando solo quelli di lunghezza pari a quella massima trovata al passo 1.

L'algoritmo ha complessità lineare nella dimensione del vettore.

Soluzione 3: in un vettore ausiliario `massimi_i0[MAXN]` si accumulano gli indici di partenza dei sottovettori a lunghezza massima. Una prima fase identifica i sottovettori, ne calcola la lunghezza e la confronta con quella stimata massima. Se maggiore, non solo si aggiorna la stima, ma si registra a partire dall'indice 0 in `massimi_i0` l'indice di partenza. Altrimenti, se uguale alla stima, si registra

nella prossima posizione di `massimi_i0` l'indice di partenza. Si osservi che identificare una nuova lunghezza massima corrente comporta il "reset" del vettore `massimi_i0`, in quanto vi si ricomincia a scrivere dall'indice 0. Terminata questa fase si scorre `massimi_i0` e si stampano i vettori a lunghezza massima che iniziano dall'indice in esso contenuto.
L'algoritmo ha complessità lineare nella dimensione del vettore.

Esercizio n. 2: Manipolazione di stringhe
Si tratta di un problema di filtro dove si selezionano le sottostringhe di una certa lunghezza data `n` che contengono esattamente 2 vocali. In estensione a quanto richiesto dalle specifiche si calcola non solo in numero totale di sottostringhe con 2 vocali, ma anche il numero di sottostringhe con 2 vocali di una parola. La funzione `conta` identifica con un ciclo `for` esterno tutti gli indici di inizio di possibili sottostringhe di lunghezza `n`, con un ciclo `for` interno verifica ciascuna sottostringa. La verifica che un carattere sia una vocale è proposta con 4 strategie: usando un vettore, un `if`, una condizione logica o un `case`.
La complessità è O(lunghezza stringa * lunghezza sottostringa).

Esercizio n. 3: Rotazione di vettori

Si estendono le specifiche prevedendo la possibilità di utilizzare i dati di un vettore default oppure di leggerli da tastiera. Si presentano 4 soluzioni richiamabili dallo stesso `main` sulla base di un valore di selezione passato come argomento alla riga di comandi. Letto `P`, se `P==0` si termina il programma, mentre se `P<0` le funzioni di risoluzione `ruotaVettore` terminano immediatamente. Si osservi che, se `P>N`, effettuare `P` rotazioni è equivalente a effettuarne `P%N`.

Le 4 soluzioni si differenziano nella gestione della direzione in quanto:
- nelle soluzioni 1 e 2 si distinguono esplicitamente la rotazione a sinistra e la rotazione a destra, scrivendo per ciascuna codice specifico
- nella soluzione 3 la direzione serve a determinare alcune informazioni, quali indici di partenza, di sorgente e di destinazione per un'assegnazione, ma le istruzioni eseguite sono poi le stesse in entrambi i casi
- nella soluzione 4 si unifica il concetto di direzione, in quanto la rotazione di `P` posizioni in una direzione equivale alla rotazione di `N-P` posizioni nella direzione opposta.

Le 4 soluzioni si differenziano nell'uso di variabili temporanee di appoggio in quanto:
- nella soluzione 1 si usa una sola variabile, in quanto una rotazione di `P` posizioni è equivalente a `P` rotazioni di 1 posizione ciascuna. La complessità è O(P*N), la soluzione è *in loco* (cfr Cap03 *Gli ordinamenti*)
- nelle soluzioni 2, 3 e 4 si utilizza un vettore temporaneo dove si accumulano `P` celle del vettore originale, si spostano le celle nel vettore originale ed infine si ricopiano le celle del vettore temporaneo nella posizione opportuna. La complessità è O(N), la soluzione non è *in loco.*

Soluzione 1: nella variabile temporanea si salva `V[0]` per rotazioni a sinistra e `V[N-1]` per rotazioni a destra, poi si procede con P shift nella direzione opportuna. Infine si assegna la variabile temporanea a `V[N-1]` per rotazioni a sinistra e a `V[0]` per rotazioni a destra.

Soluzione 2: per una rotazione a sinistra si salvano le prime `P` celle di `V` in un vettore temporaneo, si procede con `P` shift a sinistra di `V` a partire dalla posizione `P` ed infine si ricopiano in `V` a partire dalla

posizione `N-P` i dati salvati nel vettore temporaneo. Dualmente si procede per la rotazione a destra salvando le `P` celle finali in un vettore temporaneo, procedendo con `P` shift a destra di `V` a partire dalla posizione `N-P-1` ed infine ricopiando in `V` a partire dalla posizione 0 i dati salvati nel vettore temporaneo.

Soluzione 3: a seconda della direzione si identificano: l'indice di partenza `t0` della porzione di vettore `V` che andrà salvato nel vettore temporaneo, l'indice di partenza `t1` nel vettore `V` dove sarà alla fine ricopiato il vettore temporaneo e l'indice della prima cella di destinazione degli shift. Si salva la porzione desiderata di `V` nel vettore temporaneo, si procede con lo spostamento dalla cella sorgente alla cella destinazione ed infine si ricopia il vettore temporaneo a partire dall'indice `t1`. Si osservi che l'indice della cella sorgente si determina semplicemente sommando o sottraendo `P` all'indice della cella destinazione a seconda della direzione.

Soluzione 4: se necessario, si trasforma la rotazione a destra nell'equivalente rotazione a sinistra e poi si procede seguendo la strategia della soluzione 2, ridotta alla sola parte che esegue rotazioni a sinistra.

Esercizio n. 4: Iterazione su matrici

L'esercizio consta di 2 problemi distinti:

- identificazione e stampa delle sottomatrici quadrate di dimensione `dim` x `dim`. Pur trattandosi concettualmente di un problema di filtro, non lo si realizza generando tutte le possibili matrici di qualunque dimensione e poi verificandone le dimensioni. Il problema permette infatti di considerare le sole sottomatrici. della dimensione specificata
- ottimizzazione (ricerca del massimo): identificazione e stampa di una sottomatrice quadrata di dimensione `dim` x `dim` la somma dei cui elementi sia massima.

La soluzione è strutturata:

- in una fase di lettura della matrice da file
- seguita un ciclo in cui:
  - si acquisisce la dimensione delle sottomatrici quadrate da identificare, con il vincolo che tale dimensione sia maggiore di 0 e non ecceda né il numero di righe né il numero di colonne della matrice in ingresso
  - si chiama una funzione `matDxD` che risolve i 2 sottoproblemi.

Funzione `matDxD`

Si veda il commento all'esercizio 1 a proposito di problemi di ottimizzazione. Si osservi che in questo caso non è richiesto che si stampino tutte le soluzioni, basta che se ne stampi una qualsiasi.

La funzione `matDxD` enumera le sottomatrici di dimensione `dim` x `dim` mediante 2 cicli `for` annidati che identificano tutte le celle di indice (`i`, `j`) dalle quali può iniziare una sottomatrice. All'interno essa chiama la funzione `matSommaStampa`, che stampa a video la sottomatrice e calcola la somma `sum` dei suoi elementi tramite altri 2 cicli `for` annidati. Per il controllo di ottimalità si tiene traccia della massima somma `maxsum` calcolata sino a quel momento. Se la somma `sum` calcolata nell'iterazione corrente migliora la stima `maxsum`, si aggiornano `maxsum` e le coordinate dell'elemento in alto a sinistra `max_i0` e `max_j0` della matrice correntemente stimata ottima. La soluzione ottima non viene memorizzata esplicitamente, ma vi si può fare accesso in stampa operando sulla matrice di ingresso conoscendo `dim`, `max_i0` e `max_j0`.

Si osservi che alla prima iterazione si deve sempre aggiornare sempre `maxsum` con `sum`, mentre successivamente l'aggiornamento è condizionato da `sum > maxsum`. Per fare ciò si possono seguire 2 strade:

- la condizione di aggiornamento è un OR logico tra quanto è vero alla prima iterazione (`max_i0 == -1 && max_j0 == -1`) e la condizione `sum > maxsum`. In fase di aggiornamento si assegnano valori diversi da -1 a `max_i0` e `max_j0` in modo che, essendo il primo termine dell'OR sempre falso, si valuti solo la condizione `sum > maxsum`.
- in alternativa si inizializzata a `maxsum` a -∞, che in C si approssima con `-INT_MAX`, il più grande intero negativo rappresentabile, ricordando di includere `limits.h`. Attenzione: il più grande intero negativo rappresentabile non deve essere un dato che la matrice può contenere.